RM0068
Reference manual

SPC564A70B4, SPC564A70L7
32-bit MCU family built on the embedded Power Architecture®

Introduction

The primary objective of this document is to define the functionality of the SPC564A70 microcontrollers for use by software and hardware developers. The SPC564A70 is built on Power Architecture® technology and integrates technologies that are important for today's lower-end applications.

www.st.com

Contents

List of tables

List of figures

Preface

Overview

The primary objective of this document is to define the functionality of the SPC564A70 family of microcontrollers for use by software and hardware developers. The SPC564A70 family is built on Power Architecture® technology and integrates technologies that are important for today’s lower-end applications.

As with any technical documentation, it is the reader’s responsibility to be sure he or she is using the most recent version of the documentation.

To locate any published errata or updates for this document, visit the ST Web site at www.st.com.

Audience

This manual is intended for system software and hardware developers and applications programmers who want to develop products with the SPC564A70 device. It is assumed that the reader understands operating systems, microprocessor system design, basic principles of software and hardware, and basic details of the Power Architecture.

Chapter organization and device-specific information

This document includes chapters that describe:

- The device as a whole
- The functionality of the individual modules on the device

In the latter, any device-specific information is presented in the section “Information Specific to This Device” at the beginning of the chapter.

References

In addition to this reference manual, the following documents provide additional information on the operation of the SPC564A70:

- IEEE-ISTO 5001™ - 2003 and 2010, The Nexus 5001™ Forum Standard for a Global Embedded Processor Debug Interface
- IEEE 1149.1-2001 standard - IEEE Standard Test Access Port and Boundary-Scan Architecture

1 Introduction

1.1 The SPC564A70 microcontroller family

The SPC564A70 is part of a family of microcontrollers that serves two main application areas:

- Mid-range engine management
- Automotive transmission control

The SPC564A70 contains features of ST's SPC563M family and many new features coupled with high performance 90 nm CMOS technology to provide substantial reduction of cost per feature and significant performance improvement.

The e200z4 host processor core of the SPC564A70 complies with the Power Architecture® technology. It is 100% user mode compatible (with floating point library) with the Power Architecture instruction set. This architecture provides enhancements for embedded applications. In addition to the Power Architecture instruction set, this core also has additional instruction support for digital signal processing (DSP).

The SPC564A70 has two levels of memory hierarchy consisting of 8 KB of instruction cache, backed by up to 128 KB on-chip SRAM and up to 2 MB of internal flash memory. The SPC564A70 includes a 'calibration bus' that is only accessible when using calibration tools.

On-chip modules include:

- Dual issue, 32-bit Power Architecture compliant e200z4 CPU core complex
- Memory protection unit (MPU)
- Interrupt controller (INTC)
- Frequency-modulated phase-locked loop (FMPLL)
- System integration unit (SIU)
- Boot assist module (BAM)
- 32-channel second generation enhanced time processor unit (eTPU2)
- 24-channels enhanced modular Input Output System (eMIOS)
- Enhanced queued analog-to-digital converter (eQADC)
- 3 deserial serial peripheral interface (DSPI) modules
- 3 enhanced serial communication interface (eSCI) modules
- 3 controller-area network (FlexCAN) modules
- Cyclic redundancy check (CRC) module
- System timers
- Nexus development interface (NDI) per IEEE-ISTO 5001™ - 2003 standard
- On-chip voltage regulator for regulating 5 V down to 3.3 V for internal functions and Nexus interface
- On-chip voltage regulator controller for regulating 5 V down to 1.2 V for core logic

1.2 SPC564A70 device comparison

Table 1 summarizes the SPC564A70 family of microcontrollers.

Table 1. SPC564A70 device feature summary

Feature		SPC564A70	SPC564A80
Process		90 nm	
Core		e200z4	
	SIMD	Yes	
	VLE	Yes	
	Cache	8 KB instruction	
	Non-Maskable Interrupt (NMI)	NMI and Critical Interrupt	
	MMU	24-entry	
	MPU	16-entry	
	Crossbar switch	4 × 4	5 × 4
	Core performance	0 – 150 MHz	
Windowing software watchdog		Yes	
Core Nexus		Class 3+	
SRAM		128 KB	192 KB
Flash		2 MB	4 MB
Flash fetch accelerator		4 × 128-bit	4 × 256-bit
External bus		None	16-bit (incl. 32-bit muxed)
Calibration bus		16-bit (incl. 32-bit muxed)	
DMA		64 channels	
DMA Nexus		None	
Serial		3	
	eSCI_A	Yes (MSC uplink)	
	eSCI_B	Yes (MSC uplink)	
	eSCI_C	Yes	
CAN		3	
	CAN_A	64 message buffers	
	CAN_B	64 message buffers	
	CAN_C	64 message buffers	
SPI		3	
	Micro Second Channel (MSC) bus downlink	Yes	
	DSPI_A	No	
	DSPI_B	Yes (with LVDS)	
	DSPI_C	Yes (with LVDS)	
	DSPI_D	Yes	
FlexRay		Yes	

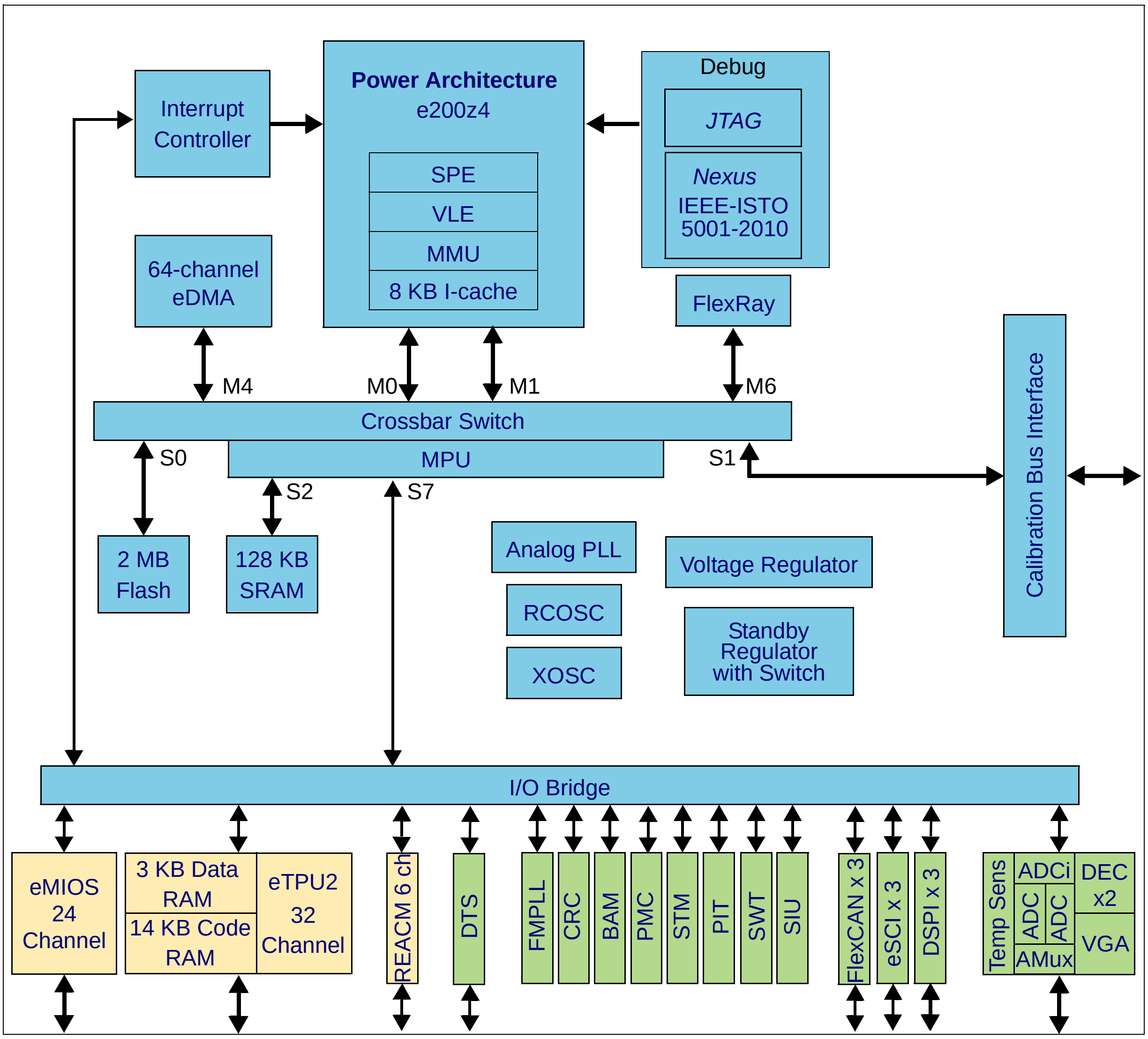
Table 1. SPC564A70 device feature summary (continued)

Feature		SPC564A70	SPC564A80
System timers		5 PIT channels 4 STM channels 1 Software Watchdog	
eMIOS		24 channels	
eTPU		32-channel eTPU2	
	Code memory	14 KB	
	Data memory	3 KB	
	Reaction module	6 channels	
Interrupt controller		485 channels(1)	
ADC		40 channels	
	ADC_0	Yes	
	ADC_1	Yes	
	Temperature sensor	Yes	
	Variable gain amplifier	Yes	
	Decimation filter	2	
	Sensor diagnostics	Yes	
CRC		Yes	
FMPLL		Yes	
VRC		Yes	
Supplies		5 V, 3.3 V(2)	
Low-power modes		Stop Mode Slow Mode	
Packages		LQFP176(3) PBGA324 496-pin CSP(4)	LQFP176(3) PBGA324 Known Good Die (KGD) 496-pin CSP(4)

1. 197 interrupt vectors are reserved.
2. 5 V single supply only for LQFP176
3. Pinout compatible with STMicroelectronics' SPC563M64 devices
4. For ST calibration tool only

1.3 Device Block Diagram

Figure 1 shows a top-level block diagram of the SPC564A70.



LEGEND

ADC	– Analog to Digital Converter
ADCi	– ADC interface
AMux	– Analog Multiplexer
BAM	– Boot Assist Module
CRC	– Cyclic Redundancy Check unit
DEC	– Decimation Filter
DTS	– Development Trigger Semaphore
DSPI	– Deserial/Serial Peripheral Interface
eDMA	– Enhanced Direct Memory Access
eMIOS	– Enhanced Modular Input Output System
eSCI	– Enhanced Serial Communications Interface
eTPU2	– Second gen. Enhanced Time Processing Unit
FlexCAN	– Controller Area Network (FlexCAN)
FMPLL	– Frequency-Modulated Phase-Locked Loop
JTAG	– IEEE 1149.1 Test Controller
MMU	– Memory Management Unit
MPU	– Memory Protection Unit
PMC	– Power Management Controller
PIT	– Periodic Interrupt Timer
RCOSC	– Low-speed RC Oscillator
REACM	– Reaction Module
SIU	– System Integration Unit
SPE	– Signal Processing Extension
SRAM	– Static RAM
STM	– System Timer Module
SWT	– Software Watchdog Timer
VGA	– Variable Gain Amplifier
VLE	– Variable Length (instruction) Encoding
XOSC	– XTAL Oscillator

Figure 1. SPC564A70 block diagram

1.4 Feature summary

- 150 MHz e200z4 Power Architecture® core
  - Variable length instruction encoding (VLE)
  - Superscalar architecture with 2 execution units
  - Up to 2 integer or floating point instructions per cycle
  - Up to 4 multiply and accumulate operations per cycle
- Memory organization
  - 2 MB on-chip flash memory with ECC and read-while-write (RWW)
  - 128 KB on-chip SRAM with standby functionality (32 KB) and ECC
  - 8 KB instruction cache (with line locking), configurable as 2- or 4-way
  - 14 + 3 KB eTPU code and data RAM
  - 4 × 4 crossbar switch (XBAR)
  - 24-entry MMU
- Fail Safe Protection
  - 16-entry Memory Protection Unit (MPU)
  - CRC unit with 3 submodules
  - Junction temperature sensor
- Interrupt
  - Configurable interrupt controller (INTC) with non-maskable interrupt (NMI)
  - 64-channel eDMA
- Serial channels
  - 3 eSCI modules
  - 3 DSPI modules (2 of which support downstream Micro Second Channel [MSC])
  - 3 FlexCAN modules with 64 message buffers each
  - 1 FlexRay module (V2.1) up to 10 Mbit/s w/dual or single channel, 128 message objects, ECC
- 1 eMIOS
  - 24 unified channels
- 1 eTPU2 (second generation eTPU)
  - 32 standard channels
  - 1 reaction module (6 channels with 3 outputs per channel)
- 2 enhanced queued analog-to-digital converters (eQADCs)
  - Forty 12-bit input channels (multiplexed on 2 ADCs); expandable to 56 channels with external multiplexers
  - 6 command queues
  - Trigger and DMA support
  - 688 ns minimum conversion time
- On-chip CAN/SCI Bootstrap loader with Boot Assist Module (BAM)
- Nexus: Class 3+ for core; Class 1 for eTPU
- JTAG (5-pin)
- Development Trigger Semaphore (DTS)
  - EVTO pin for communication with external tool

- Clock generation
  - On-chip 4–40 MHz main oscillator
  - On-chip FMPLL (frequency-modulated phase-locked loop)
- Up to 112 general purpose I/O lines
  - Individually programmable as input, output or special function
  - Programmable threshold (hysteresis)
- Power reduction modes: slow, stop, and standby
- Flexible supply scheme
  - 5 V single supply with external ballast
  - Multiple external supply: 5 V, 3.3 V, and 1.2 V

1.5 Feature Details

1.5.1 e200z4 core

SPC564A70 devices have a high performance e200z4 core processor:

- 32-bit Power Architecture technology programmer’s model
- Variable Length Encoding (VLE) enhancements
- Dual issue, 32-bit Power Architecture technology compliant CPU
- 8 KB, 2/4-way set associative instruction cache
- Thirty-two 64-bit general purpose registers (GPRs)
- Memory Management Unit (MMU) with 24-entry fully-associative translation look-aside buffer (TLB)
- Harvard Architecture: Separate instruction bus and load/store bus
- Vectored interrupt support
- Non-maskable interrupt input
- Critical Interrupt input
- New ‘Wait for Interrupt’ instruction, to be used with new low power modes
- Reservation instructions for implementing read-modify-write accesses
- Signal processing extension (SPE) APU
- Single Precision Floating point (scalar and vector)
- Nexus Class 3+ debug
- Process ID manipulation for the MMU using an external tool
- In-order execution and retirement
- Precise exception handling
- Branch processing unit
  - Dedicated branch address calculation adder
  - Branch target prefetching using 8-entry BTB
- Supports independent instruction and data accesses to different memory subsystems, such as SRAM and flash memory via independent Instruction and Data BIUs
- Load/store unit
  - 2-cycle load latency
  - Fully pipelined
  - Big and Little endian support
  - Misaligned access support
- Signal Processing Extension (SPE1.1) APU supporting SIMD fixed-point operations using the 64-bit General Purpose Register file
- Embedded Floating-Point (EFP2) APU supporting scalar and vector SIMD single-precision floating-point operations, using the 64-bit General Purpose Register file
- Power management
  - Low power design – extensive clock gating
  - Power saving modes: wait
  - Dynamic power management of execution units, cache and MMU

- Testability
  - Synthesizeable, MuxD scan design
  - ABIST/MBIST for arrays
  - Built-in Parallel Signature Unit
- Calibration support allowing an external tool to modify address mapping

1.5.2 Crossbar switch (XBAR)

The XBAR multiport crossbar switch supports simultaneous connections between four master ports and four slave ports. The crossbar supports a 32-bit address bus width and a 64-bit data bus width.

The crossbar allows three concurrent transactions to occur from the master ports to any slave port but each master must access a different slave. If a slave port is simultaneously requested by more than one master port, arbitration logic selects the higher priority master and grants it ownership of the slave port. All other masters requesting that slave port are stalled until the higher priority master completes its transactions. Requesting masters are treated with equal priority and are granted access to a slave port in round-robin fashion, based upon the ID of the last master to be granted access. The crossbar provides the following features:

- 4 master ports
  - CPU instruction bus
  - CPU data bus
  - eDMA
  - FlexRay
- 4 slave ports
  - Flash
  - Calibration bus interface
  - SRAM
  - Peripheral bridge
- 32-bit internal address, 64-bit internal data paths

1.5.3 Enhanced direct memory access (eDMA)

The enhanced direct memory access (eDMA) controller is a second-generation module capable of performing complex data movements via 64 programmable channels, with minimal intervention from the host processor. The hardware micro-architecture includes a DMA engine which performs source and destination address calculations, and the actual data movement operations, along with an SRAM-based memory containing the transfer control descriptors (TCD) for the channels. This implementation minimizes overall block size. The eDMA module provides the following features:

- All data movement via dual-address transfers: read from source, write to destination
- Programmable source and destination addresses, transfer size, plus support for enhanced addressing modes
- Transfer control descriptor organized to support two-deep, nested transfer operations
- An inner data transfer loop defined by a “minor” byte transfer count
- An outer data transfer loop defined by a “major” iteration count

- Channel activation via one of three methods:
  - Explicit software initiation
  - Initiation via a channel-to-channel linking mechanism for continuous transfers
  - Peripheral-paced hardware requests (one per channel)
- Support for fixed-priority and round-robin channel arbitration
- Channel completion reported via optional interrupt requests
- 1 interrupt per channel, optionally asserted at completion of major iteration count
- Error termination interrupts optionally enabled
- Support for scatter/gather DMA processing
- Ability to suspend channel transfers by a higher priority channel

1.5.4 Interrupt controller (INTC)

The INTC provides priority-based preemptive scheduling of interrupt requests, suitable for statically scheduled hard real-time systems.

For high priority interrupt requests, the time from the assertion of the interrupt request from the peripheral to when the processor is executing the interrupt service routine (ISR) has been minimized. The INTC provides a unique vector for each interrupt request source for quick determination of which ISR needs to be executed. It also provides an ample number of priorities so that lower priority ISRs do not delay the execution of higher priority ISRs. To allow the appropriate priorities for each source of interrupt request, the priority of each interrupt request is software configurable.

When multiple tasks share a resource, coherent accesses to that resource need to be supported. The INTC supports the priority ceiling protocol for coherent accesses. By providing a modifiable priority mask, the priority can be raised temporarily so that all tasks which share the resource cannot preempt each other.

The INTC provides the following features:

- 9-bit vector addresses
- Unique vector for each interrupt request source
- Hardware connection to processor or read from register
- Each interrupt source can assigned a specific priority by software
- Preemptive prioritized interrupt requests to processor
- ISR at a higher priority preempts executing ISRs or tasks at lower priorities
- Automatic pushing or popping of preempted priority to or from a LIFO
- Ability to modify the ISR or task priority to implement the priority ceiling protocol for accessing shared resources
- Low latency—3 clocks from receipt of interrupt request from peripheral to interrupt request to processor

This device also includes a non-maskable interrupt (NMI) pin that bypasses the INTC and multiplexing logic.

1.5.5 Memory protection unit (MPU)

The Memory Protection Unit (MPU) provides hardware access control for all memory references generated in a device. Using preprogrammed region descriptors, which define memory spaces and their associated access rights, the MPU concurrently monitors all system bus transactions and evaluates the appropriateness of each transfer. Memory references with sufficient access control rights are allowed to complete; references that are not mapped to any region descriptor or have insufficient rights are terminated with a protection error response.

The MPU has these major features:

- Support for 16 memory region descriptors, each 128 bits in size
  - Specification of start and end addresses provide granularity for region sizes from 32 bytes to 4 GB
  - MPU is invalid at reset, thus no access restrictions are enforced
  - 2 types of access control definitions: processor core bus master supports the traditional {read, write, execute} permissions with independent definitions for supervisor and user mode accesses; the remaining non-core bus masters (eDMA, FlexRay) support {read, write} attributes
  - Automatic hardware maintenance of the region descriptor valid bit removes issues associated with maintaining a coherent image of the descriptor
  - Alternate memory view of the access control word for each descriptor provides an efficient mechanism to dynamically alter the access rights of a descriptor only
  - For overlapping region descriptors, priority is given to permission granting over access denying as this approach provides more flexibility to system software
- Support for two XBAR slave port connections (SRAM and PBRIDGE)
  - For each connected XBAR slave port (SRAM and PBRIDGE), MPU hardware monitors every port access using the preprogrammed memory region descriptors
  - An access protection error is detected if a memory reference does not hit in any memory region or the reference is flagged as illegal in all memory regions where it does hit. In the event of an access error, the XBAR reference is terminated with an error response and the MPU inhibits the bus cycle being sent to the targeted slave device
  - 64-bit error registers, one for each XBAR slave port, capture the last faulting address, attributes, and detail information

1.5.6 Frequency-modulated phase-locked loop (FMPLL)

The FMPLL allows the user to generate high speed system clocks from a 4 MHz to 40 MHz crystal oscillator or external clock generator. Further, the FMPLL supports programmable frequency modulation of the system clock. The PLL multiplication factor, output clock divider ratio are all software configurable. The PLL has the following major features:

- Input clock frequency from 4 MHz to 40 MHz
- Reduced frequency divider (RFD) for reduced frequency operation without forcing the PLL to relock
- 3 modes of operation
  - Bypass mode with PLL off
  - Bypass mode with PLL running (default mode out of reset)
  - PLL normal mode
- Each of the 3 modes may be run with a crystal oscillator or an external clock reference

- Programmable frequency modulation
  - Modulation enabled/disabled through software
  - Triangle wave modulation up to 100 kHz modulation frequency
  - Programmable modulation depth (0% to 2% modulation depth)
  - Programmable modulation frequency dependent on reference frequency
- Lock detect circuitry reports when the PLL has achieved frequency lock and continuously monitors lock status to report loss of lock conditions
- Clock Quality Module
  - Detects the quality of the crystal clock and causes interrupt request or system reset if error is detected
  - Detects the quality of the PLL output clock; if error detected, causes system reset or switches system clock to crystal clock and causes interrupt request
- Programmable interrupt request or system reset on loss of lock
- Self-clocked mode (SCM) operation

1.5.7 System integration unit (SIU)

The SPC564A70 SIU controls MCU reset configuration, pad configuration, external interrupt, general purpose I/O (GPIO), internal peripheral multiplexing, and the system reset operation. The reset configuration block contains the external pin boot configuration logic. The pad configuration block controls the static electrical characteristics of I/O pins. The GPIO block provides uniform and discrete input/output control of the I/O pins of the MCU. The reset controller performs reset monitoring of internal and external reset sources, and drives the $\overline{\text{RSTOUT}}$ pin. Communication between the SIU and the e200z4 CPU core is via the crossbar switch. The SIU provides the following features:

- System configuration
  - MCU reset configuration via external pins
  - Pad configuration control for each pad
  - Pad configuration control for virtual I/O via DSPI serialization
- System reset monitoring and generation
  - Power-on reset support
  - Reset status register provides last reset source to software
  - Glitch detection on reset input
  - Software controlled reset assertion
- External interrupt
  - Rising or falling edge event detection
  - Programmable digital filter for glitch rejection
  - Critical Interrupt request
  - Non-Maskable Interrupt request
- GPIO
  - Centralized control of I/O and bus pins
  - Virtual GPIO via DSPI serialization (requires external deserialization device)
  - Dedicated input and output registers for setting each GPIO and Virtual GPIO pin

- Internal multiplexing
  - Allows serial and parallel chaining of DSPIs
  - Allows flexible selection of eQADC trigger inputs
  - Allows selection of interrupt requests between external pins and DSPI
  - From a set of eTPU output channels, allows selection of source signals for decimation filter integrators

1.5.8 Flash memory

The SPC564A70 provides 2 MB of programmable, non-volatile, flash memory. The non-volatile memory (NVM) can be used to store instructions or data, or both. The flash module includes a Fetch Accelerator that optimizes the performance of the flash array to match the CPU architecture. The flash module interfaces the system bus to a dedicated flash memory array controller. For CPU ‘loads’, DMA transfers and CPU instruction fetch, it supports a 64-bit data bus width at the system bus port, and 128-bit read data interfaces to flash memory. The module contains a prefetch controller which prefetches sequential lines of data from the flash array into the buffers. Prefetch buffer hits allow no-wait responses.

The flash memory provides the following features:

- Supports a 64-bit data bus for instruction fetch, CPU loads and DMA access. Byte, halfword, word and doubleword reads are supported. Only aligned word and doubleword writes are supported.
- Fetch Accelerator
  - Architected to optimize the performance of the flash
  - Configurable read buffering and line prefetch support
  - 4-entry 128-bit wide line read buffer
  - Prefetch controller
- Hardware and software configurable read and write access protections on a per-master basis
- Interface to the flash array controller pipelined with a depth of one, allowing overlapped accesses to proceed in parallel for pipelined flash array designs
- Configurable access timing usable in a wide range of system frequencies
- Multiple-mapping support and mapping-based block access timing (0–31 additional cycles) usable for emulation of other memory types
- Software programmable block program/erase restriction control
- Erase of selected block(s)
- Read page size of 128 bits (4 words)
- ECC with single-bit correction, double-bit detection
- Program page size of 128 bits (4 words) to accelerate programming
- ECC single-bit error corrections are visible to software
- Minimum program size is 2 consecutive 32-bit words, aligned on a 0-modulo-8 byte address, due to ECC
- Embedded hardware program and erase algorithm
- Erase suspend, program suspend and erase-suspended program
- Shadow information stored in non-volatile shadow block
- Independent program/erase of the shadow block

1.5.9 Static random access memory (SRAM)

The SRAM provides 128 KB of general purpose system SRAM. The first 32 KB block of the SRAM is powered by its own power supply pin only during standby operation.

The SRAM controller includes these features:

- 128 KB data RAM implemented as eight 16 KB (2048 × 78 bits) blocks
- Each 16 KB block has 2 rows repairable (RAMs with internal repair feature)
- Supports read/write accesses mapped to the SRAM memory from any master
- 32 KB block powered by separate supply for standby operation
- Byte, halfword, word and doubleword addressable
- ECC performs single bit correction, double bit detection

1.5.10 Boot assist module (BAM)

The BAM is a block of read-only memory that is programmed once by ST and is identical for all SPC564A70 MCUs. The BAM program is executed every time the MCU is powered on or reset in normal mode. The BAM supports different modes of booting. They are:

- Booting from internal flash memory
- Serial boot loading (boot code is downloaded into RAM via eSCI or the FlexCAN and then executed)

The BAM also reads the reset configuration half word (RCHW) from internal flash memory and configures the SPC564A70 hardware accordingly. The BAM provides the following features:

- Sets up MMU to cover all resources and mapping of all physical addresses to logical addresses with minimum address translation
- Sets up MMU to allow user boot code to execute as either Power Architecture technology code (default) or as VLE code
- Location and detection of user boot code
- Automatic switch to serial boot mode if internal flash is blank or invalid
- Supports user programmable 64-bit password protection for serial boot mode
- Supports serial bootloading via FlexCAN bus and eSCI using standard protocol
- Supports serial bootloading via FlexCAN bus and eSCI with auto baud rate sensing
- Supports serial bootloading of either Power Architecture technology code (default) or VLE code
- Supports booting from calibration bus interface
- Supports censorship protection for internal flash memory
- Provides an option to enable the core watchdog timer
- Provides an option to disable the system watchdog timer

1.5.11 Enhanced modular input/output system (eMIOS)

The eMIOS timer module provides the capability to generate or measure events in hardware.

The eMIOS module features include:

- Twenty-four 24-bit wide channels
- 3 channels' internal timebases sharable between channels
- 1 timebase from eTPU2 can be imported and used by the channels
- Global enable feature for all eMIOS and eTPU timebases
- Dedicated pin for each channel (not available on all package types)
- Each channel (0–23) supports the following functions:
  - General Purpose Input/Output (GPIO)
  - Single Action Input Capture (SAIC)
  - Single Action Output Compare (SAOC)
  - Output Pulse Width Modulation Buffered (OPWMB)
  - Input Period Measurement (IPM)
  - Input Pulse Width Measurement (IPWM)
  - Double Action Output Compare (DOAC)
  - Modulus Counter Buffered (MCB)
  - Output Pulse Width & Frequency Modulation Buffered (OPWFMB)
- Each channel has its own pin (not available on all package types)

1.5.12 Second generation enhanced time processing unit (eTPU2)

The eTPU2 is an enhanced co-processor designed for timing control. Operating in parallel with the host CPU, the eTPU2 processes instructions and real-time input events, performs output waveform generation, and accesses shared data without host intervention. Consequently, for each timer event, the host CPU setup and service times are minimized or eliminated. A powerful timer subsystem is formed by combining the eTPU2 with its own instruction and data RAM. High-level assembler/compiler and documentation allows customers to develop their own functions on the eTPU2.

SPC564A70 devices feature the second generation of the eTPU, called eTPU2. Enhancements of the eTPU2 over the standard eTPU include:

- The Timer Counter (TCR1), channel logic and digital filters (both channel and the external timer clock input [TCRCLK]) now have an option to run at full system clock speed or system clock / 2.
- Channels support unordered transitions: transition 2 can now be detected before transition 1. Related to this enhancement, the transition detection latches (TDL1 and TDL2) can now be independently negated by microcode.
- A new User Programmable Channel Mode has been added: the blocking, enabling, service request and capture characteristics of this channel mode can be programmed via microcode.
- Microinstructions now provide an option to issue Interrupt and Data Transfer requests selected by channel. They can also be requested simultaneously at the same instruction.
- Channel Flags 0 and 1 can now be tested for branching, in addition to selecting the entry point.
- Channel digital filters can be bypassed.

The eTPU2 includes these distinctive features:

- 32 channels; each channel associated with one input and one output signal
  - Enhanced input digital filters on the input pins for improved noise immunity
  - Identical, orthogonal channels: each channel can perform any time function. Each time function can be assigned to more than one channel at a given time, so each signal can have any functionality.
  - Each channel has an event mechanism which supports single and double action functionality in various combinations. It includes two 24-bit capture registers, two 24-bit match registers, 24-bit greater-equal and equal-only comparators.
  - Input and output signal states visible from the host
- 2 independent 24-bit time bases for channel synchronization:
  - First time base clocked by system clock with programmable prescale division from 2 to 512 (in steps of 2), or by output of second time base prescaler
  - Second time base counter can work as a continuous angle counter, enabling angle based applications to match angle instead of time
  - Both time bases can be exported to the eMIOS timer module
  - Both time bases visible from the host
- Event-triggered microengine:
  - Fixed-length instruction execution in two-system-clock microcycle
  - 14 KB of code memory (SCM)
  - 3 KB of parameter (data) RAM (SPRAM)
  - Parallel execution of data memory, ALU, channel control and flow control sub-instructions in selected combinations
  - 32-bit microengine registers and 24-bit wide ALU, with 1 microcycle addition and subtraction, absolute value, bitwise logical operations on 24-bit, 16-bit, or byte operands, single-bit manipulation, shift operations, sign extension and conditional execution
  - Additional 24-bit Multiply/MAC/Divide unit which supports all signed/unsigned Multiply/MAC combinations, and unsigned 24-bit divide. The MAC/Divide unit works in parallel with the regular microcode commands.
- Resource sharing features support channel use of common channel registers, memory and microengine time:
  - Hardware scheduler works as a “task management” unit, dispatching event service routines by predefined, host-configured priority
  - Automatic channel context switch when a “task switch” occurs, that is, one function thread ends and another begins to service a request from other channel: channel-specific registers, flags and parameter base address are automatically loaded for the next serviced channel
  - SPRAM shared between host CPU and eTPU2, supporting communication either between channels and host or inter-channel
  - Hardware implementation of 4 semaphores support coherent parameter sharing between both eTPU engines
  - Dual-parameter coherency hardware support allows atomic access to 2 parameters by host

- Test and development support features:
  - Nexus Class 1 debug, supporting single-step execution, arbitrary microinstruction execution, hardware breakpoints and watchpoints on several conditions
  - Software breakpoints
  - SCM continuous signature-check built-in self test MISC (multiple input signature calculator), runs concurrently with eTPU2 normal operation

1.5.13 Reaction module (REACM)

The REACM provides the ability to modulate output signals to manage closed loop control without CPU assistance. It works in conjunction with the eQADC and eTPU2 to increase system performance by removing the CPU from the current control loop.

The REACM has the following features:

- 6 reaction channels with peak and hold control blocks
- Each channel output is a bus of 3 signals, providing ability to control 3 inputs.
- Each channel can implement a peak and hold waveform, making it possible to implement up to six independent peak and hold control channels

Target applications include solenoid control for direct injection systems and valve control in automatic transmissions.

1.5.14 Enhanced queued analog-to-digital converter (eQADC)

The eQADC block provides accurate and fast conversions for a wide range of applications. The eQADC provides a parallel interface to two on-chip analog-to-digital converters (ADC), and a single master to single slave serial interface to an off-chip external device. Both on-chip ADCs have access to all the analog channels.

The eQADC prioritizes and transfers commands from six command conversion command ‘queues’ to the on-chip ADCs or to the external device. The block can also receive data from the on-chip ADCs or from an off-chip external device into the six result queues, in parallel, independently of the command queues. The six command queues are prioritized with Queue_0 having the highest priority and Queue_5 the lowest. Queue_0 also has the added ability to bypass all buffering and queuing and abort a currently running conversion on either ADC and start a Queue_0 conversion. This means that Queue_0 will always have a deterministic time from trigger to start of conversion, irrespective of what tasks the ADCs were performing when the trigger occurred. The eQADC supports software and external hardware triggers from other blocks to initiate transfers of commands from the queues to the on-chip ADCs or to the external device. It also monitors the fullness of command queues and result queues, and accordingly generates DMA or interrupt requests to control data movement between the queues and the system memory, which is external to the eQADC.

The ADCs also support features designed to allow the direct connection of high impedance acoustic sensors that might be used in a system for detecting engine knock. These features include differential inputs; integrated variable gain amplifiers for increasing the dynamic range; programmable pull-up and pull-down resistors for biasing and sensor diagnostics.

The eQADC also integrates a programmable decimation filter capable of taking in ADC conversion results at a high rate, passing them through a hardware low pass filter, then down-sampling the output of the filter and feeding the lower sample rate results to the result FIFOs. This allows the ADCs to sample the sensor at a rate high enough to avoid aliasing of out-of-band noise; while providing a reduced sample rate output to minimize the amount DSP processing bandwidth required to fully process the digitized waveform.

The eQADC provides the following features:

- Dual on-chip ADCs
  - 2 × 12-bit ADC resolution
  - Programmable resolution for increased conversion speed (12-bit, 10-bit, 8-bit)
    12-bit conversion time – 938 ns (1M sample/s)
    10-bit conversion time – 813 ns (1.2M sample/s)
    8-bit conversion time – 688 ns (1.4M sample/s)
  - Up to 10-bit accuracy at 500K sample/s and 8-bit accuracy at 1M sample/s
  - Differential conversions
  - Single-ended signal range from 0 to 5 V
  - Sample times of 2 (default), 8, 64, or 128 ADC clock cycles
  - Provides time stamp information when requested
  - Allows time stamp information relative to eTPU clock sources, such as an angle clock
  - Parallel interface to eQADC command FIFOs (CFIFOs) and result FIFOs (RFIFOs)
  - Supports both right-justified unsigned and signed formats for conversion results
- 40 single-ended input channels, expandable to 56 channels with external multiplexers (supports 4 external 8-to-1 muxes)
- 8 channels can be used as 4 pairs of differential analog input channels
- Differential channels include variable gain amplifier for improved dynamic range (×1, ×2, ×4)
- Differential channels include programmable pull-up and pull-down resistors for biasing and sensor diagnostics (200 kΩ, 100 kΩ, 5 kΩ)
- Additional internal channels for monitoring voltages (such as core voltage, I/O voltage, LVI voltages, etc.) inside the device
- An internal bandgap reference to allow absolute voltage measurements
- Silicon die temperature sensor
  - Provides temperature of silicon as an analog value
  - Read using an internal ADC analog channel
  - May be read with either ADC

- 2 decimation filters
  - Programmable decimation factor (1 to 16)
  - Selectable IIR or FIR filter
  - Up to 4th order IIR or 8th order FIR
  - Programmable coefficients
  - Saturated or non-saturated modes
  - Programmable Rounding (Convergent; Two's Complement; Truncated)
  - Prefill mode to precondition the filter before the sample window opens
  - Supports Multiple Cascading Decimation Filters to implement more complex filter designs
  - Optional Absolute Integrators on the output of Decimation Filters
- Full duplex synchronous serial interface (SSI) to an external device
  - Free-running clock for use by an external device
  - Supports a 26-bit message length
- Priority based queues
  - Supports 6 queues with fixed priority. When commands of distinct queues are bound for the same ADC, the higher priority queue is always served first
  - Queue_0 can bypass all prioritization, buffering and abort current conversions to start a Queue_0 conversion a deterministic time after the queue trigger
  - Supports software and hardware trigger modes to arm a particular queue
  - Generates interrupt when command coherency is not achieved
- External hardware triggers
  - Supports rising edge, falling edge, high level and low level triggers
  - Supports configurable digital filter

1.5.15 Deserial serial peripheral interface (DSPI)

The DSPI block provides a synchronous serial interface for communication between the SPC564A70 MCU and external devices. The DSPI supports pin count reduction through serialization and deserialization of eTPU and eMIOS channels and memory-mapped registers. The channels and register content are transmitted using a SPI-like protocol. This SPI-like protocol is completely configurable for baud rate, polarity and phase, frame length, chip select assertion, etc. Each bit in the frame may be configured to serialize either eTPU channels, eMIOS channels or GPIO signals. The DSPI can be configured to serialize data to an external device that implements the Microsecond Bus protocol. There are three identical DSPI blocks on the SPC564A70 MCU. The DSPI pins support 5 V logic levels or Low Voltage Differential Signalling (LVDS) to improve high speed operation.

DSPI module features include:

- Selectable LVDS pads working at 40 MHz for SOUT and SCK pins for DSPI_B and DSPI_C
- Support for downstream Micro Second Channel (MSC) with Timed Serial Bus (TSB) configuration on DSPI_B and DSPI_C
- 3 sources of serialized data: eTPU_A, eMIOS output channels, and memory-mapped register in the DSPI
- 4 destinations for deserialized data: eTPU_A and eMIOS input channels, SIU external Interrupt input request, memory-mapped register in the DSPI
- 32-bit DSI and TSB modes require 32 PCR registers, 32 GPO and GPI registers in the SIU to select either GPIO, eTPU or eMIOS bits for serialization
- The DSPI module can generate and check parity in a serial frame

1.5.16 Enhanced serial communications interface (eSCI)

Three eSCI modules provide asynchronous serial communications with peripheral devices and other MCUs, and include support to interface to Local Interconnect Network (LIN) slave devices. Each eSCI block provides the following features:

- Full-duplex operation
- Standard mark/space non-return-to-zero (NRZ) format
- 13-bit baud rate selection
- Programmable 8-bit or 9-bit data format
- Programmable 12-bit or 13-bit data format for Timed Serial Bus (TSB) configuration to support the Microsecond bus standard
- Automatic parity generation
- LIN support
  - Compatible with LIN slaves from revisions 1.x and 2.0 of the LIN standard
  - Autonomous transmission of entire frames
  - Configurable to support all revisions of the LIN standard
  - Automatic parity bit generation
  - Double stop bit after bit error
  - 10- or 13-bit break support
- Separately enabled transmitter and receiver
- Programmable transmitter output parity
- 2 receiver wake-up methods:
  - Idle line wake-up
  - Address mark wake-up
- Interrupt-driven operation with flags
- Receiver framing error detection
- Hardware parity checking
- 1/16 bit-time noise detection
- DMA support for both transmit and receive data
  - Global error bit stored with receive data in system RAM to allow post processing of errors

1.5.17 Controller area network (FlexCAN)

The SPC564A70 MCU includes three FlexCAN blocks. The FlexCAN module is a communication controller implementing the CAN protocol according to Bosch Specification version 2.0B. The CAN protocol was designed to be used primarily as a vehicle serial data bus, meeting the specific requirements of this field: real-time processing, reliable operation in the EMI environment of a vehicle, cost-effectiveness and required bandwidth. Each FlexCAN module contains 64 message buffers.

The FlexCAN modules provide the following features:

- Full Implementation of the CAN protocol specification, Version 2.0B
  - Standard data and remote frames
  - Extended data and remote frames
  - Zero to eight bytes data length
  - Programmable bit rate up to 1 Mbit/s
- Content-related addressing
- 64 message buffers of 0 to 8 bytes data length
- Individual Rx Mask Register per message buffer
- Each message buffer configurable as Rx or Tx, all supporting standard and extended messages
- Includes 1088 bytes of embedded memory for message buffer storage
- Includes 256-byte memory for storing individual Rx mask registers
- Full-featured Rx FIFO with storage capacity for 6 frames and internal pointer handling
- Powerful Rx FIFO ID filtering, capable of matching incoming IDs against 8 extended, 16 standard or 32 partial (8 bits) IDs, with individual masking capability
- Selectable backwards compatibility with previous FlexCAN versions
- Programmable clock source to the CAN Protocol Interface, either system clock or oscillator clock
- Listen only mode capability
- Programmable loop-back mode supporting self-test operation
- 3 programmable Mask Registers
- Programmable transmit-first scheme: lowest ID, lowest buffer number or highest priority
- Time Stamp based on 16-bit free-running timer
- Global network time, synchronized by a specific message
- Maskable interrupts
- Warning interrupts when the Rx and Tx Error Counters reach 96
- Independent of the transmission medium (an external transceiver is assumed)
- Multi-master concept
- High immunity to EMI
- Short latency time due to an arbitration scheme for high-priority messages
- Low power mode, with programmable wakeup on bus activity

1.5.18 FlexRay

The SPC564A70 includes one dual-channel FlexRay module that implements the FlexRay Communications System Protocol Specification, Version 2.1 Rev A. Features include:

- Single channel support
- FlexRay bus data rates of 10 Mbit/s, 8 Mbit/s, 5 Mbit/s, and 2.5 Mbit/s supported
- 128 message buffers, each configurable as:
  - Receive message buffer
  - Single-buffered transmit message buffer
  - Double-buffered transmit message buffer (combines two single-buffered message buffers)
- 2 independent receive FIFOs
  - 1 receive FIFO per channel
  - Up to 255 entries for each FIFO
- ECC support

1.5.19 System timers

The system timers include two distinct types of system timer:

- Periodic interrupts/triggers using the Periodic Interrupt Timer (PIT)
- Operating system task monitors using the System Timer Module (STM)

Periodic interrupt timer (PIT)

The PIT provides five independent timer channels, capable of producing periodic interrupts and periodic triggers. The PIT has no external input or output pins and is intended to provide system ‘tick’ signals to the operating system, as well as periodic triggers for eQADC queues. Of the five channels in the PIT, four are clocked by the system clock and one is clocked by the crystal clock. This one channel is also referred to as Real-Time Interrupt (RTI) and is used to wake up the device from low power stop mode.

The following features are implemented in the PIT:

- 5 independent timer channels
- Each channel includes 32-bit wide down counter with automatic reload
- 4 channels clocked from system clock
- 1 channel clocked from crystal clock (wake-up timer)
- Wake-up timer remains active when System STOP mode is entered; used to restart system clock after predefined time-out period
- Each channel optionally able to generate an interrupt request or a trigger event (to trigger eQADC queues) when timer reaches zero

System timer module (STM)

The STM is designed to implement the software task monitor as defined by AUTOSAR(a). It consists of a single 32-bit counter, clocked by the system clock, and four independent timer

---

a. AUTOSAR: AUTomotive Open System ARchitecture (see www.autosar.org) comparators. These comparators produce a CPU interrupt when the timer exceeds the programmed value.

The following features are implemented in the STM:

- One 32-bit up counter with 8-bit prescaler
- Four 32-bit compare channels
- Independent interrupt source for each channel
- Counter can be stopped in debug mode

1.5.20 Software watchdog timer (SWT)

The SWT is a second watchdog module to complement the standard Power Architecture watchdog integrated in the CPU core. The SWT is a 32-bit modulus counter, clocked by the system clock or the crystal clock, that can provide a system reset or interrupt request when the correct software key is not written within the required time window.

The following features are implemented:

- 32-bit modulus counter
- Clocked by system clock or crystal clock
- Optional programmable watchdog window mode
- Can optionally cause system reset or interrupt request on timeout
- Reset by writing a software key to memory mapped register
- Enabled out of reset
- Configuration is protected by a software key or a write-once register

1.5.21 Cyclic redundancy check (CRC) module

The CRC computing unit is dedicated to the computation of CRC off-loading the CPU. The CRC module features:

- Support for CRC-16-CCITT (*x*25 protocol):
  - $x^{16} + x^{12} + x^5 + 1$
- Support for CRC-32 (Ethernet protocol):
  - $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$
- Zero wait states for each write/read operations to the CRC_CFG and CRC_INP registers at the maximum frequency

1.5.22 Error correction status module (ECSM)

The ECSM provides a myriad of miscellaneous control functions regarding program-visible information about the platform configuration and revision levels, a reset status register, a software watchdog timer, wakeup control for exiting sleep modes, and information on platform memory errors reported by error-correcting codes and/or generic access error information for certain processor cores.

The Error Correction Status Module supports a number of miscellaneous control functions for the platform. The ECSM includes these features:

- Registers for capturing information on platform memory errors if error-correcting codes (ECC) are implemented
- For test purposes, optional registers to specify the generation of double-bit memory errors are enabled on the SPC564A70.

The sources of the ECC errors are:

- Flash memory
- SRAM
- Peripheral RAM (FlexRay, CAN, eTPU2 parameter RAM)

1.5.23 Peripheral bridge (PBRIDGE)

The PBRIDGE implements the following features:

- Duplicated periphery
- Master access privilege level per peripheral (per master: read access enable; write access enable)
- Write buffering for peripherals
- Checker applied on PBRIDGE output toward periphery
- Byte endianess swap capability

1.5.24 Calibration bus interface

The calibration bus interface controls data transfer across the crossbar switch to/from memories or peripherals attached to the calibration tool connector in the calibration address space. The calibration bus interface is only available in the calibration tool.

Features include:

- 3.3 V ± 10% I/O (3.0 V to 3.6 V)
- Memory controller supports various memory types
- 16-bit data bus, up to 22-bit address bus
- Pin muxing supports 32-bit muxed bus
- Selectable drive strength
- Configurable bus speed modes
- Bus monitor
- Configurable wait states

1.5.25 Power management controller (PMC)

The PMC contains circuitry to generate the internal 3.3 V supply and to control the regulation of 1.2 V supply with an external NPN ballast transistor. It also contains low voltage inhibit (LVI) and power-on reset (POR) circuits for the 1.2 V supply, the 3.3 V supply, the 3.3 V/5 V supply of the closest I/O segment (VDDEH1), and the 5 V supply of the regulators (VDDREG).

1.5.26 Nexus port controller (NPC)

The NPC block provides real-time Nexus Class3+ development support capabilities for the SPC564A70 Power Architecture technology-based MCU in compliance with the IEEE-ISTO 5001-2010 standard. MDO port widths of 4 pins and 12 pins are available in all packages.

1.5.27 JTAG controller (JTAGC)

The JTAG controller (JTAGC) block provides the means to test chip functionality and connectivity while remaining transparent to system logic when not in test mode. Testing is performed via a boundary scan technique, as defined in the IEEE 1149.1-2001 standard. All data input to and output from the JTAGC block is communicated in serial format. The JTAGC block is compliant with the IEEE 1149.1-2001 standard and supports the following features:

- IEEE 1149.1-2001 Test Access Port (TAP) interface 4 pins (TDI, TMS, TCK, and TDO)
- A 5-bit instruction register that supports the following IEEE 1149.1-2001 defined instructions:
  - BYPASS, IDCODE, EXTEST, SAMPLE, SAMPLE/PRELOAD, HIGHZ, CLAMP
- A 5-bit instruction register that supports the additional following public instructions:
  - ACCESS_AUX_TAP_NPC
  - ACCESS_AUX_TAP_ONCE
  - ACCESS_AUX_TAP_eTPU
  - ACCESS_CENSOR
- 3 test data registers to support JTAG Boundary Scan mode
  - Bypass register
  - Boundary scan register
  - Device identification register
- A TAP controller state machine that controls the operation of the data registers, instruction register and associated circuitry
- Censorship Inhibit Register
  - 64-bit Censorship password register
  - If the external tool writes a 64-bit password that matches the Serial Boot password stored in the internal flash shadow row, Censorship is disabled until the next system reset.

1.5.28 Development trigger semaphore (DTS)

SPC564A70 devices include a system development feature, the Development Trigger Semaphore (DTS) module, that enables user software to signal to an external tool—by driving a persistent (affected only by reset or an external tool) signal on an external device pin—that data is available. The DTS includes a register of semaphores (32-bits) and an identification register.

There are a variety of ways this module can be used, including as a component of an external real-time data acquisition system.

2 Memory map

2.1 Introduction

Table 2 shows the memory map for the SPC564A70. All addresses on the SPC564A70, including those that are reserved, are identified in the table. The addresses represent the physical addresses assigned to each IP block.

2.2 Memory Map

Table 2. SPC564A70 Memory Map

Start address	End address	Allocated size (bytes)	Used size (bytes)	Region name
0x0000_0000	0x001F_FFFF	2 MB	2 MB	Flash memory
0x0020_0000	0x00FF_BFFF	—	—	Reserved
0x00FF_C000	0x00FF_FFFF	16 KB	16 KB	Flash memory Shadow Block
0x0100_0000	0x1FFF_FFFF	496 MB	—	Emulation Remapping of flash memory
0x2000_0000	0x2FFF_FFFF	256 MB	—	Reserved
0x3000_0000	0x3FFF_FFFF	256 MB	—	Calibration Bus
0x4000_0000	0x4000_7FFF	32 KB	32 KB	SRAM Array, Standby Powered
0x4000_8000	0x4001_FFFF	96 KB	96 KB	SRAM Array
0x4002_0000	0xBFFF_FFFF	2048 MB – 128 KB	—	Reserved
Bridge A Peripherals				
0xC000_0000	0xC3EF_FFFF	63 MB	—	Reserved
0xC3F0_0000	0xC3F0_3FFF	16 KB	—	Peripheral Bridge A
0xC3F0_4000	0xC3F7_FFFF	—	—	Reserved
0xC3F8_0000	0xC3F8_3FFF	16 KB	—	PLL
0xC3F8_4000	0xC3F8_7FFF	16 KB	—	Calibration Bus Interface Configuration
0xC3F8_8000	0xC3F8_BFFF	16 KB	—	Flash Configuration
0xC3F8_C000	0xC3F8_FFFF	—	—	Reserved
0xC3F9_0000	0xC3F9_3FFF	16 KB	—	SIU
0xC3F9_4000	0xC3F9_BFFF	—	—	Reserved
0xC3F9_C000	0xC3F9_FFFF	16 KB	—	DTS
0xC3FA_0000	0xC3FA_3FFF	16 KB	—	eMIOS
0xC3FA_4000	0xC3FB_BFFF	—	—	Reserved
0xC3FB_C000	0xC3FB_FFFF	16 KB	—	PMC
0xC3FC_0000	0xC3FC_3FFF	16 KB	—	eTPU Registers
0xC3FC_4000	0xC3FC_6FFF	—	—	Reserved

Table 2. SPC564A70 Memory Map (continued)

Start address	End address	Allocated size (bytes)	Used size (bytes)	Region name
0xC3FC_7000	0xC3FC_7FFF	4 KB	—	Reaction Module
0xC3FC_8000	0xC3FC_BFFF	16 KB	3 KB	eTPU Parameter RAM
0xC3FC_C000	0xC3FC_FFFF	16 KB	3 KB	eTPU Parameter RAM Mirror
0xC3FD_0000	0xC3FD_37FF	14 KB	14 KB	eTPU Code RAM
0xC3FD_4400	0xC3FE_FFFF	—	—	Reserved
0xC3FF_0000	0xC3FF_3FFF	16 KB	—	PIT/RTI
0xC3FF_4000	0xFFE6_7FFF	—	—	Reserved
0xFFE6_8000	0xFFE6_BFFF	16 KB	—	Cyclic Redundancy Check (CRC) Unit
0xFFE6_C000	0xFFEF_FFFF	—	—	Reserved
0xFFF0_0000	0xFFF0_3FFF	16 KB	—	Peripheral Bridge B Registers
0xFFF0_4000	0xFFF0_7FFF	16 KB	—	Crossbar switch (XBAR)
0xFFF0_8000	0xFFF0_FFFF	16 KB	—	Reserved
0xFFF1_0000	0xFFF1_3FFF	16 KB	—	MPU
0xFFF1_4000	0xFFF3_7FFF	144 KB	—	Reserved
0xFFF3_8000	0xFFF3_BFFF	16 KB	—	SWT
0xFFF3_C000	0xFFF3_FFFF	16 KB	—	STM
0xFFF4_0000	0xFFF4_3FFF	16 KB	—	ECSM
0xFFF4_4000	0xFFF4_7FFF	16 KB	—	eDMA
0xFFF4_8000	0xFFF4_BFFF	16 KB	—	INTC
0xFFF4_C000	0xFFF7_FFFF	208 KB	—	Reserved
0xFFF8_0000	0xFFF8_3FFF	16 KB	—	eQADC
0xFFF8_4000	0xFFF8_7FFF	16 KB	—	Reserved
0xFFF8_8000	0xFFF8_BFFF	16 KB	—	Decimation Filter A
0xFFF8_C000	0xFFF8_FFFF	16 KB	—	Decimation Filter B
0xFFF9_0000	0xFFF9_3FFF	16 KB	—	Reserved
0xFFF9_4000	0xFFF9_7FFF	16 KB	—	DSPI_B
0xFFF9_8000	0xFFF9_BFFF	16 KB	—	DSPI_C
0xFFF9_C000	0xFFF9_FFFF	16 KB	—	DSPI_D
0xFFFA_0000	0xFFFA_FFFF	64 KB	—	Reserved
0xFFFB_0000	0xFFFB_3FFF	16 KB	—	eSCI_A
0xFFFB_4000	0xFFFB_7FFF	16 KB	—	eSCI_B
0xFFFB_8000	0xFFFB_BFFF	16 KB	—	eSCI_C
0xFFFB_C000	0xFFFB_FFFF	16 KB	—	Reserved
0xFFFC_0000	0xFFFC_3FFF	16 KB	—	FlexCAN_A

Table 2. SPC564A70 Memory Map (continued)

Start address	End address	Allocated size (bytes)	Used size (bytes)	Region name
0xFFFC_4000	0xFFFC_7FFF	16 KB	—	FlexCAN_B
0xFFFC_8000	0xFFFC_BFFF	16 KB	—	FlexCAN_C
0xFFFC_C000	0xFFFD_FFFF	80 KB	—	Reserved
0xFFFE_0000	0xFFFE_3FFF	16 KB	—	FlexRay
0xFFFE_4000	0xFFFE_BFFF	32 KB	—	Reserved
0xFFFE_C000	0xFFFE_FFFF	16 KB	—	System Information Module (Temperature sensor calibration parameters and unique device ID code)
0xFFFF_0000	0xFFFF_BFFF	48 KB	—	Reserved
0xFFFF_C000	0xFFFF_FFFF	16 KB	16 KB	Boot Assist Module

3 Signal Description

This chapter describes signals that connect off chip. It includes a table of signal properties and the detailed descriptions of signals.

3.1 Signal Properties

Table 3. SPC564A70 signal properties

Name(1)	Function(2)	P / A / G(3)	PCR PA field (4)	PCR (5)	I/O type	Voltage(6) / Pad type(7)	Status(8)		Package pin No.		
							During reset	After reset	176	208(9)	324
GPIO											
FR_A_TX GPIO[12]	FlexRay transmit data channel A GPIO	A1 G	010 000	12	O I/O	VDDE-EH / Medium	— / Up	— / Up	—	—	P1
FR_A_TX_EN GPIO[13]	FlexRay ch. A tx data enable GPIO	A1 G	010 000	13	O I/O	VDDE-EH / Medium	— / Up	— / Up	—	—	P2
FR_A_RX GPIO[14]	FlexRay receive data ch. A GPIO	A1 G	010 000	14	I I/O	VDDE-EH / Medium	— / Up	— / Up	—	—	R1
FR_B_TX GPIO[15]	FlexRay transmit data ch. B GPIO	A1 G	010 000	15	O I/O	VDDE-EH / Medium	— / Up	— / Up	—	—	R2
FR_B_TX_EN GPIO[16]	FlexRay tx data enable for ch. B GPIO	A1 G	010 000	16	O I/O	VDDE-EH / Medium	— / Up	— / Up	—	—	T1
FR_B_RX GPIO[17]	FlexRay receive data channel B GPIO	A1 G	010 000	17	I I/O	VDDE-EH / Medium	— / Up	— / Up	—	—	T2
GPIO[206] ETRIG0	GPIO / eQADC Trigger Input	G	00	206	I/O(10)	VDDEH7 / Slow(11)	— / Up	— / Up	143	R4	AA7
GPIO[207] ETRIG1	GPIO / eQADC Trigger Input	G	00	207	I/O(10)	VDDEH7 / Slow	— / Up	— / Up	144	P5	Y9
GPIO[219]	GPIO	G	000	219 (12)	I/O	VDDEH7 / MultV	— / Up	— / Up	122	T6	—
Reset / Configuration											
$\overline{\text{RESET}}$	External Reset Input	P	—	—	I	VDDEH6 / Slow	$\overline{\text{RESET}}$ / Up	$\overline{\text{RESET}}$ / Up	97	L16	R22
$\overline{\text{RSTOUT}}$	External Reset Output	P	01	230	O	VDDEH6 / Slow	$\overline{\text{RSTOUT}}$ / Down	$\overline{\text{RSTOUT}}$ / Down	102	K15	P21

Table 3. SPC564A70 signal properties (continued)

Name[1]	Function[2]	P / A / G[3]	PCR PA field[4]	PCR[5]	I/O type	Voltage[6] / Pad type[7]	Status[8]		Package pin No.		
							During reset	After reset	176	208[9]	324
PLLREF IRQ[4] ETRIG2 GPIO[208]	FMPLL Mode Selection External Interrupt Request eQADC Trigger Input GPIO	P A1 A2 G	001 010 100 000	208	I I I I/O	VDDEH6 / Slow	— / Up	PLLREF / Up	83	M14	V21
PLLCFG1[13] IRQ[5] DSPI_D_SOUT GPIO[209]	— External interrupt request DSPI D data output GPIO	— A1 A2 G	— 010 100 000	209	— I O I/O	VDDEH6 / Medium	— / Up	— / Up	—	—	U20
RSTCFG GPIO[210]	RSTCFG GPIO	P G	01 00	210	I I/O	VDDEH6 / Slow	— / Down	—	—	—	P22
BOOTCFG[0] IRQ[2] GPIO[211]	Boot Config. Input External Interrupt Request GPIO	P A1 G	01 10 00	211	I I I/O	VDDEH6 / Slow	— / Down	BOOTCFG[0] / Down	—	—	U21
BOOTCFG[1] IRQ[3] ETRIG3 GPIO[212]	Boot Config. Input External Interrupt Request eQADC Trigger Input GPIO	P A1 A2 G	001 010 100 000	212	I I I I/O	VDDEH6 / Slow	— / Down	BOOTCFG[1] / Down	85	M15	T20
WKPCFG $\overline{\text{NMI}}$ DSPI_B_SOUT GPIO[213]	Weak Pull Config. Input Non-Maskable Interrupt DSPI B data output GPIO	P A1 A2 G	001 010 100 000	213	I I O I/O	VDDEH6 / Medium	— / Up	WKPCFG / Up	86	L15	R19
Calibration Bus											
CAL_CS0	Calibration chip select	P	01	336	O	VDDE12 / Fast		— / —	—	—	—
CAL_CS2 CAL_ADDR[10] CAL_WE[2]/BE[2]	Calibration chip select Calibration address bus Calibration write/byte enable	P A1 A2	001 010 100	338	O I/O O	VDDE12 / Fast		— / —	—	—	—
CAL_CS3 CAL_ADDR[11] CAL_$\overline{\text{WE}}$[3]/$\overline{\text{BE}}$[3]	Calibration chip select Calibration address bus Calibration write/byte enable	P A1 A2	001 010 100	339	O I/O O	VDDE12 / Fast		— / —	—	—	—
CAL_ADDR[12] CAL_$\overline{\text{WE}}$[2]/$\overline{\text{BE}}$[2]	Calibration address bus Calibration write/byte enable	P A1	01 10	340	I/O O	VDDE12 / Fast		— / —	—	—	—
CAL_ADDR[13] CAL_$\overline{\text{WE}}$[3]/$\overline{\text{BE}}$[3]	Calibration address bus Calibration write/byte enable	P A1	01 10	340	I/O O	VDDE12 / Fast		— / —	—	—	—

Table 3. SPC564A70 signal properties (continued)

Name[(1)]	Function[(2)]	P / A / G[(3)]	PCR PA field [(4)]	PCR [(5)]	I/O type	Voltage[(6)] / Pad type[(7)]	Status[(8)]		Package pin No.		
							During reset	After reset	176	208[(9)]	324
CAL_ADDR[14] CAL_DATA[31]	Calibration address bus Calibration data bus	P A1	01 10	340	I/O I/O	VDDE12 / Fast		— / —	—	—	—
CAL_ADDR[15] CAL_ALE	Calibration address bus Calibration address latch enable	P A1	01 10	340	I/O O	VDDE12 / Fast		— / —	—	—	—
CAL_ADDR[16] CAL_DATA[16]	Calibration address bus Calibration data bus	P A1	01 10	345	I/O I/O	VDDE12 / Fast		— / —	—	—	—
CAL_ADDR[17] CAL_DATA[17]	Calibration address bus Calibration data bus	P A1	01 10	345	I/O I/O	VDDE12 / Fast		— / —	—	—	—
CAL_ADDR[18] CAL_DATA[18]	Calibration address bus Calibration data bus	P A1	01 10	345	I/O I/O	VDDE12 / Fast		— / —	—	—	—
CAL_ADDR[19] CAL_DATA[19]	Calibration address bus Calibration data bus	P A1	01 10	345	I/O I/O	VDDE12 / Fast		— / —	—	—	—
CAL_ADDR[20] CAL_DATA[20]	Calibration address bus Calibration data bus	P A1	01 10	345	I/O I/O	VDDE12 / Fast		— / —	—	—	—
CAL_ADDR[21] CAL_DATA[21]	Calibration address bus Calibration data bus	P A1	01 10	345	I/O I/O	VDDE12 / Fast		— / —	—	—	—
CAL_ADDR[22] CAL_DATA[22]	Calibration address bus Calibration data bus	P A1	01 10	345	I/O I/O	VDDE12 / Fast		— / —	—	—	—
CAL_ADDR[23] CAL_DATA[23]	Calibration address bus Calibration data bus	P A1	01 10	345	I/O I/O	VDDE12 / Fast		— / —	—	—	—
CAL_ADDR[24] CAL_DATA[24]	Calibration address bus Calibration data bus	P A1	01 10	345	I/O I/O	VDDE12 / Fast		— / —	—	—	—
CAL_ADDR[25] CAL_DATA[25]	Calibration address bus Calibration data bus	P A1	01 10	345	I/O I/O	VDDE12 / Fast		— / —	—	—	—
CAL_ADDR[26] CAL_DATA[26]	Calibration address bus Calibration data bus	P A1	01 10	345	I/O I/O	VDDE12 / Fast		— / —	—	—	—
CAL_ADDR[27] CAL_DATA[27]	Calibration address bus Calibration data bus	P A1	01 10	345	I/O I/O	VDDE12 / Fast		— / —	—	—	—
CAL_ADDR[28] CAL_DATA[28]	Calibration address bus Calibration data bus	P A1	01 10	345	I/O I/O	VDDE12 / Fast		— / —	—	—	—
CAL_ADDR[29] CAL_DATA[29]	Calibration address bus Calibration data bus	P A1	01 10	345	I/O I/O	VDDE12 / Fast		— / —	—	—	—

Table 3. SPC564A70 signal properties (continued)

Name[1]	Function[2]	P / A / G[3]	PCR PA field [4]	PCR [5]	I/O type	Voltage[6] / Pad type[7]	Status[8]		Package pin No.		
							During reset	After reset	176	208[9]	324
CAL_ADDR[30] CAL_DATA[30]	Calibration address bus Calibration data bus	P A1	01 10	345	I/O I/O	VDDE12 / Fast		— / —	—	—	—
CAL_DATA[0]	Calibration data bus	P	01	341	I/O	VDDE12 / Fast	— / Up	— / Up	—	—	—
CAL_DATA[1]	Calibration data bus	P	01	341	I/O	VDDE12 / Fast	— / Up	— / Up	—	—	—
CAL_DATA[2]	Calibration data bus	P	01	341	I/O	VDDE12 / Fast	— / Up	— / Up	—	—	—
CAL_DATA[3]	Calibration data bus	P	01	341	I/O	VDDE12 / Fast	— / Up	— / Up	—	—	—
CAL_DATA[4]	Calibration data bus	P	01	341	I/O	VDDE12 / Fast	— / Up	— / Up	—	—	—
CAL_DATA[5]	Calibration data bus	P	01	341	I/O	VDDE12 / Fast	— / Up	— / Up	—	—	—
CAL_DATA[6]	Calibration data bus	P	01	341	I/O	VDDE12 / Fast	— / Up	— / Up	—	—	—
CAL_DATA[7]	Calibration data bus	P	01	341	I/O	VDDE12 / Fast	— / Up	— / Up	—	—	—
CAL_DATA[8]	Calibration data bus	P	01	341	I/O	VDDE12 / Fast	— / Up	— / Up	—	—	—
CAL_DATA[9]	Calibration data bus	P	01	341	I/O	VDDE12 / Fast	— / Up	— / Up	—	—	—
CAL_DATA[10]	Calibration data bus	P	01	341	I/O	VDDE12 / Fast	— / Up	— / Up	—	—	—
CAL_DATA[11]	Calibration data bus	P	01	341	I/O	VDDE12 / Fast	— / Up	— / Up	—	—	—
CAL_DATA[12]	Calibration data bus	P	01	341	I/O	VDDE12 / Fast	— / Up	— / Up	—	—	—
CAL_DATA[13]	Calibration data bus	P	01	341	I/O	VDDE12 / Fast	— / Up	— / Up	—	—	—
CAL_DATA[14]	Calibration data bus	P	01	341	I/O	VDDE12 / Fast	— / Up	— / Up	—	—	—

Table 3. SPC564A70 signal properties (continued)

Name[1]	Function[2]	P / A / G[3]	PCR PA field (4)	PCR (5)	I/O type	Voltage[6] / Pad type[7]	Status[8]		Package pin No.		
							During reset	After reset	176	208[9]	324
CAL_DATA[15]	Calibration data bus	P	01	341	I/O	VDDE12 / Fast	— / Up	— / Up	—	—	—
CAL_RD_WR	Calibration data bus	P	01	342	O	VDDE12 / Fast		— / —	—	—	—
CAL_WE[0]	Calibration write enable	P	01	342	O	VDDE12 / Fast		— / —	—	—	—
CAL_WE[1]	Calibration write enable	P	01	342	O	VDDE12 / Fast		— / —	—	—	—
CAL_OE	Calibration output enable	P	01	342	O	VDDE12 / Fast		— / —	—	—	—
CAL_TS CAL_ALE	Calibration transfer start Address Latch Enable	P A1	01 10	343	O O	VDDE12 / Fast		— / —	—	—	—
CAL_MDO[4]	Calibration Nexus Message Data Out	P	01	—	O	VDDE12 / Fast	—	CAL_MDO[4] / —	—	—	—
CAL_MDO[5]	Calibration Nexus Message Data Out	P	01	—	O	VDDE12 / Fast	—	CAL_MDO[5] / —	—	—	—
CAL_MDO[6]	Calibration Nexus Message Data Out	P	01	—	O	VDDE12 / Fast	—	CAL_MDO[6] / —	—	—	—
CAL_MDO[7]	Calibration Nexus Message Data Out	P	01	—	O	VDDE12 / Fast	—	CAL_MDO[7] / —	—	—	—
CAL_MDO[8]	Calibration Nexus Message Data Out	P	01	—	O	VDDE12 / Fast	—	CAL_MDO[8] / —	—	—	—
CAL_MDO[9]	Calibration Nexus Message Data Out	P	01	—	O	VDDE12 / Fast	—	CAL_MDO[9] / —	—	—	—
CAL_MDO[10]	Calibration Nexus Message Data Out	P	01	—	O	VDDE12 / Fast	—	CAL_MDO[10] / —	—	—	—
CAL_MDO[11]	Calibration Nexus Message Data Out	P	01	—	O	VDDE12 / Fast	—	CAL_MDO[11] / —	—	—	—
NEXUS											
$\overline{\text{EVTI}}$	Nexus event in	P	01	231	I	VDDEH7 / MultiV	— / Up	$\overline{\text{EVTI}}$ / Up	116	E15	F21

Table 3. SPC564A70 signal properties (continued)

Name(1)	Function(2)	P / A / G(3)	PCR PA field (4)	PCR (5)	I/O type	Voltage(6) / Pad type(7)	Status(8)		Package pin No.		
							During reset	After reset	176	208(9)	324
$\overline{\text{EVTO}}$	Nexus event out	P	01	227	O	VDDEH7 / MultiV	—	$\overline{\text{EVTO}}$ / —	120	D15	F22
MCKO	Nexus message clock out	P	—	219(12)	O	VRC33 / Fast	—	MCKO / —	14	F15	G20
MDO[0]	Nexus message data out	P	01	220	O	VRC33 / Fast	—	MDO[0] / —	17	A14	B20
MDO[1]	Nexus message data out	P	01	221	O	VRC33 / Fast	—	MDO[1] / —	18	B14	C19
MDO[2]	Nexus message data out	P	01	222	O	VRC33 / Fast	—	MDO[2] / —	19	A13	C18
MDO[3]	Nexus message data out	P	01	223	O	VRC33 / Fast	—	MDO[3] / —	20	B13	D18
MDO[4] ETPUA2_O GPIO[75]	Nexus message data out eTPU A channel (output only) GPIO	P A1 G	01 10 00	75	O O I/O	VDDEH7 / MultiV	—	— / —	126	P10	B19
MDO[5] ETPUA4_O GPIO[76]	Nexus message data out eTPU A channel (output only) GPIO	P A1 G	01 10 00	76	O O I/O	VDDEH7 / MultiV	—	— / —	129	T10	C17
MDO[6] ETPUA13_O GPIO[77]	Nexus message data out eTPU A channel (output only) GPIO	P A1 G	01 10 00	77	O O I/O	VDDEH7 / MultiV	—	— / —	135	T11	D17
MDO[7] ETPUA19_O GPIO[78]	Nexus message data out eTPU A channel (output only) GPIO	P A1 G	01 10 00	78	O O I/O	VDDEH7 / MultiV	—	— / —	136	N11	B18
MDO[8] ETPUA21_O GPIO[79]	Nexus message data out eTPU A channel (output only) GPIO	P A1 G	01 10 00	79	O O I/O	VDDEH7 / MultiV	—	— / —	137	P11	A19
MDO[9] ETPUA25_O PIO[80]	Nexus message data out eTPU A channel (output only) GPIO	P A1 G	01 10 00	80	O O I/O	VDDEH7 / MultiV	—	— / —	139	T7	B17
MDO[10] ETPUA27_O GPIO[81]	Nexus message data out eTPU A channel (output only) GPIO	P A1 G	01 10 00	81	O O I/O	VDDEH7 / MultiV	—	— / —	134	R10	A18

Table 3. SPC564A70 signal properties (continued)

Name[1]	Function[2]	P / A / G[3]	PCR PA field (4)	PCR (5)	I/O type	Voltage[6] / Pad type[7]	Status[8]		Package pin No.		
							During reset	After reset	176	208[9]	324
MDO[11] ETPUA29_O GPIO[82]	Nexus message data out eTPU A channel (output only) GPIO[82]	P A1 G	01 10 00	82	O O I/O	VDDEH7 / MultiV	—	— / —	124	P9	A17
MSEO[0]	Nexus message start/end out	P	01	224	O	VDDEH7 / MultiV	—	$\overline{\text{MSEO[0]}}$ / —	118	C15	G21
MSEO[1]	Nexus message start/end out	P	01	225	O	VDDEH7 / MultiV	—	$\overline{\text{MSEO[1]}}$ / —	117	E16	G22
RDY	Nexus ready output	P	01	226	O	VDDEH7 / MultiV	—	—	—	—	G19
JTAG											
TCK	JTAG test clock input	P	01	—	I	VDDEH7 / MultiV	TCK / Down	TCK / Down	128	C16	D21
TDI	JTAG test data input	P	01	232	I	VDDEH7 / MultiV	TDI / Up	TDI / Up	130	E14	D22
TDO	JTAG test data output	P	01	228	O	VDDEH7 / MultiV	TDO / Up	TDO / Up	123	F14	E21
TMS	JTAG test mode select input	P	01	—	I	VDDEH7 / MultiV	TMS / Up	TMS / Up	131	D14	E20
JCOMP	JTAG TAP controller enable	P	01	—	I	VDDEH7 / MultiV	JCOMP / Down	JCOMP / Down	121	F16	F20
FlexCAN											
CAN_A_TX SCI_A_TX GPIO[83]	FlexCAN A transmit eSCI A transmit GPIO	P A1 G	01 10 00	83	O O I/O	VDDEH6 / Slow	— / Up	— / Up	81	P12	Y17
CAN_A_RX SCI_A_RX GPIO[84]	FlexCAN A receive eSCI A receive GPIO	P A1 G	01 10 00	84	I I I/O	VDDEH6 / Slow	— / Up	— / Up	82	R12	AA18
CAN_B_TX DSPI_C_PCS[3] SCI_C_TX GPIO[85]	FlexCAN B transmit DSPI C peripheral chip select eSCI C transmit GPIO	P A1 A2 G	001 010 100 000	85	O O O I/O	VDDEH6 / Slow	— / Up	— / Up	88	T12	AB18

Table 3. SPC564A70 signal properties (continued)

Name(1)	Function(2)	P / A / G(3)	PCR PA field (4)	PCR (5)	I/O type	Voltage(6) / Pad type(7)	Status(8)		Package pin No.		
							During reset	After reset	176	208(9)	324
CAN_B_RX DSPI_C_PCS[4] SCI_C_RX GPIO[86]	FlexCAN B receive DSPI C peripheral chip select eSCI C receive GPIO	P A1 A2 G	001 010 100 000	86	I O I I/O	VDDEH6 / Slow	— / Up	— / Up	89	R13	AB19
CAN_C_TX DSPI_D_PCS[3] GPIO[87]	FlexCAN C transmit DSPI D peripheral chip select GPIO	P A1 G	01 10 00	87	O O I/O	VDDEH6 / Medium	— / Up	— / Up	101	K13	P19
CAN_C_RX DSPI_D_PCS[4] GPIO[88]	FlexCAN C receive DSPI D peripheral chip select GPIO	P A1 G	01 10 00	88	I O I/O	VDDEH6 / Slow	— / Up	— / Up	98	L14	R20
eSCI											
SCI_A_TX EMIOS13(14) GPIO[89]	eSCI A transmit eMIOS channel GPIO	P A1 G	01 10 00	89	O O I/O	VDDEH6 / Medium	— / Up	— / Up	100	J14	N20
SCI_A_RX EMIOS15[14] GPIO[90]	eSCI A receive eMIOS channel GPIO	P A1 G	01 10 00	90	I O I/O	VDDEH6 / Medium	— / Up	— / Up	99	K14	P20
SCI_B_TX DSPI_D_PCS[1] GPIO[91]	eSCI B transmit DSPI D peripheral chip select GPIO	P A1 G	01 10 00	91	O O I/O	VDDEH6 / Medium	— / Up	— / Up	87	L13	R21
SCI_B_RX DSPI_D_PCS[5] GPIO[92]	eSCI B receive DSPI D peripheral chip select GPIO	P A1 G	01 10 00	92	I O I/O	VDDEH6 / Medium	— / Up	— / Up	84	M13	T19
SCI_C_TX GPIO[244]	eSCI C transmit GPIO	P G	01 00	244	O I/O	VDDEH6 / Medium	— / Up	— / Up	—	—	W18
SCI_C_RX GPIO[245]	eSCI C receive GPIO	P G	01 00	245	I I/O	VDDEH6 / Medium	— / Up	— / Up	—	—	Y19
DSPI											
DSPI_A_SCK(15) DSPI_C_PCS[1] GPIO[93]	— DSPI C peripheral chip select GPIO	— A1 G	— 10 00	93	— O I/O	VDDEH7 / Medium	— / Up	— / Up	—	—	L22

Table 3. SPC564A70 signal properties (continued)

Name[1]	Function[2]	P / A / G[3]	PCR PA field[4]	PCR[5]	I/O type	Voltage[6] / Pad type[7]	Status[8]		Package pin No.		
							During reset	After reset	176	208[9]	324
DSPI_A_SIN[15] DSPI_C_PCS[2] GPIO[94]	— DSPI C peripheral chip select GPIO	— A1 G	— 10 00	94	— O I/O	VDDEH7 / Medium	— / Up	— / Up	—	—	L21
DSPI_A_SOUT[15] DSPI_C_PCS[5] GPIO[95]	— DSPI C peripheral chip select GPIO	— A1 G	— 10 00	95	— O I/O	VDDEH7 / Medium	— / Up	— / Up	—	—	L20
DSPI_A_PCS[0][15] DSPI_D_PCS[2] GPIO[96]	— DSPI C peripheral chip select GPIO	— A1 G	— 10 00	96	— O I/O	VDDEH7 / Medium	— / Up	— / Up	—	—	M20
DSPI_A_PCS[1][15] DSPI_B_PCS[2] GPIO[97]	— DSPI C peripheral chip select GPIO	— A1 G	— 10 00	97	— O I/O	VDDEH7 / Medium	— / Up	— / Up	—	—	M19
DSPI_A_PCS[2][15] DSPI_D_SCK GPIO[98]	— SPI clock pin for DSPI module GPIO	— A1 G	— 10 00	98	— I/O I/O	VDDEH7 / Medium	— / Up	— / Up	141	J15	M21
DSPI_A_PCS[3][15] DSPI_D_SIN GPIO[99]	— DSPI D data input GPIO	— A1 G	— 10 00	99	— I I/O	VDDEH7 / Medium	— / Up	— / Up	142	H13	K19
DSPI_A_PCS[4][15] DSPI_D_SOUT GPIO[100]	— DSPI D data output GPIO	— A1 G	— 10 00	100	— O I/O	VDDEH7 / Medium	— / Up	— / Up	—	—	N19
DSPI_A_PCS[5][15] DSPI_B_PCS[3] GPIO[101]	— DSPI B peripheral chip select GPIO	— A1 G	— 10 00	101	— O I/O	VDDEH7 / Medium	— / Up	— / Up	—	—	N21
DSPI_B_SCK DSPI_C_PCS[1] GPIO[102]	SPI clock pin for DSPI module DSPI B peripheral chip select GPIO	P A1 G	01 10 00	102	I/O O I/O	VDDEH6 / Medium	— / Up	— / Up	106	J16	K21
DSPI_B_SIN DSPI_C_PCS[2] GPIO[103]	DSPI B data input DSPI C peripheral chip select GPIO	P A1 G	01 10 00	103	I O I/O	VDDEH6 / Medium	— / Up	— / Up	112	G15	H22
DSPI_B_SOUT DSPI_C_PCS[5] GPIO[104]	DSPI B data output DSPI C peripheral chip select GPIO	P A1 G	01 10 00	104	O O I/O	VDDEH6 / Medium	— / Up	— / Up	113	G13	J19

Table 3. SPC564A70 signal properties (continued)

Name(1)	Function(2)	P / A / G(3)	PCR PA field (4)	PCR (5)	I/O type	Voltage(6) / Pad type(7)	Status(8)		Package pin No.		
							During reset	**After reset**	**176**	**208(9)**	**324**
DSPI_B_PCS[0] DSPI_D_PCS[2] GPIO[105]	DSPI B peripheral chip select DSPI D peripheral chip select GPIO	P A1 G	01 10 00	105	I/O O I/O	VDDEH6 / Medium	— / Up	— / Up	111	G16	J21
DSPI_B_PCS[1] DSPI_D_PCS[0] GPIO[106]	DSPI B peripheral chip select DSPI D peripheral chip select GPIO	P A1 G	01 10 00	106	O I/O I/O	VDDEH6 / Medium	— / Up	— / Up	109	H16	J22
DSPI_B_PCS[2] DSPI_C_SOUT GPIO[107]	DSPI B peripheral chip select DSPI C data output GPIO	P A1 G	01 10 00	107	O O I/O	VDDEH6 / Medium	— / Up	— / Up	107	H15	K22
DSPI_B_PCS[3] DSPI_C_SIN GPIO[108]	DSPI B peripheral chip select DSPI C data input GPIO	P A1 G	01 10 00	108	O I I/O	VDDEH6 / Medium	— / Up	— / Up	114	G14	J20
DSPI_B_PCS[4] DSPI_C_SCK GPIO[109]	DSPI B peripheral chip select SPI clock pin for DSPI module GPIO	P A1 G	01 10 00	109	O I/O I/O	VDDEH6 / Medium	— / Up	— / Up	105	H14	K20
DSPI_B_PCS[5] DSPI_C_PCS[0] GPIO[110]	DSPI B peripheral chip select DSPI C peripheral chip select GPIO	P A1 G	01 10 00	110	O I/O I/O	VDDEH6 / Medium	— / Up	— / Up	104	J13	L19
eQADC											
AN0 DAN0+	Single Ended Analog Input Positive Terminal Differential Input	P	—	—	I	VDDA / Analog Pull-up/down	I / —	AN[0] / —	172	B5	B8
AN1 DAN0–	Single Ended Analog Input Negative Terminal Differential Input	P	—	—	I	VDDA / Analog Pull-up/down	I / —	AN[1] / —	171	A6	A8
AN2 DAN1+	Single Ended Analog Input Positive Terminal Differential Input	P	—	—	I	VDDA / Analog Pull-up/down	I / —	AN[2] / —	170	D6	D10
AN3 DAN1–	Single Ended Analog Input Negative Terminal Differential Input	P	—	—	I	VDDA / Analog Pull-up/down	I / —	AN[3] / —	169	C7	C9
AN4 DAN2+	Single Ended Analog Input Positive Terminal Differential Input	P	—	—	I	VDDA / Analog Pull-up/down	I / —	AN[4] / —	168	B6	B9

Table 3. SPC564A70 signal properties (continued)

Name[1]	Function[2]	P / A / G[3]	PCR PA field [4]	PCR [5]	I/O type	Voltage[6] / Pad type[7]	Status[8]		Package pin No.		
							During reset	After reset	176	208[9]	324
AN5 DAN2–	Single Ended Analog Input Negative Terminal Differential Input	P	—	—	I	VDDA / Analog Pull-up/down	I / —	AN[5] / —	167	A7	A9
AN6 DAN3+	Single Ended Analog Input Positive Terminal Differential Input	P	—	—	I	VDDA / Analog Pull-up/down	I / —	AN[6] / —	166	D7	D11
AN7 DAN3–	Single Ended Analog Input Negative Terminal Differential Input	P	—	—	I	VDDA / Analog Pull-up/down	I / —	AN[7] / —	165	C8	C10
AN8 ANW	Single-ended Analog Input Multiplexed Analog Input	P	01	—	I	VDDA / Analog	I / —	AN[8] / —	9	B3	D6
AN9 ANX	Single-ended Analog Input External Multiplexed Analog Input	P	01	—	I	VDDA / Analog	I / —	AN[9] / —	5	A2	D7
AN10 ANY	Single-ended Analog Input Multiplexed Analog Input	P	01	—	I	VDDA / Analog	I / —	AN[10] / —	—	—	D8
AN11 ANZ	Single-ended Analog Input Multiplexed Analog Input	P	01	—	I	VDDA / Analog	I / —	AN[11] / —	4	A3	A5
AN12 - SDS MA0 ETPUA19_O $\overline{\text{SDS}}$	Single-ended Analog Input MUX Address 0 eTPU A channel (output only) eQADC Serial Data Select	P A1 A2 G	001 010 100 000	215	I O O I/O	VDDEH7 / Medium	I / —	AN[12] / —	148	A12	A16
AN13 - SDO MA1 ETPUA21_O SDO	Single-ended Analog Input MUX Address 1 eTPU A channel (output only) eQADC Serial Data Out	P A1 A2 G	001 010 100 000	216	I O O O	VDDEH7 / Medium	I / —	AN[13] / —	147	B12	B16
AN14 - SDI MA2 ETPUA27_O SDI	Single-ended Analog Input MUX Address 2 eTPU A channel (output only) eQADC Serial Data In	P A1 A2 G	001 010 100 000	217	I O O I	VDDEH7 / Medium	I / —	AN[14] / —	146	C12	C16
AN15 - FCK FCK ETPUA29_O	Single-ended Analog Input eQADC Free Running Clock eTPU A channel (output only)	P A1 A2	001 010 100	218	I O O	VDDEH7 / Medium	I / —	AN[15] / —	145	C13	D16
AN16	Single-ended Analog Input	P	—	—	I	VDDA / Analog	I / —	AN[16] / —	3	C6	B7

Table 3. SPC564A70 signal properties (continued)

Name[1]	Function[2]	P / A / G[3]	PCR PA field[4]	PCR[5]	I/O type	Voltage[6] / Pad type[7]	Status[8]		Package pin No.		
							During reset	After reset	176	208[9]	324
AN17	Single-ended Analog Input	P	—	—	I	VDDA / Analog	I / —	AN[17] / —	2	C4	C6
AN18	Single-ended Analog Input	P	—	—	I	VDDA / Analog	I / —	AN[18] / —	1	D5	D9
AN19	Single-ended Analog Input	P	—	—	I	VDDA / Analog	I / —	AN[19] / —	—	—	B6
AN20	Single-ended Analog Input	P	—	—	I	VDDA / Analog	I / —	AN[20] / —	—	—	C7
AN21	Single-ended Analog Input	P	—	—	I	VDDA / Analog	I / —	AN[21] / —	173	B4	C8
AN22	Single-ended Analog Input	P	—	—	I	VDDA / Analog	I / —	AN[22] / —	161	B8	C11
AN23	Single-ended Analog Input	P	—	—	I	VDDA / Analog	I / —	AN[23] / —	160	C9	B11
AN24	Single-ended Analog Input	P	—	—	I	VDDA / Analog	I / —	AN[24] / —	159	D8	D12
AN25	Single-ended Analog Input	P	—	—	I	VDDA / Analog	I / —	AN[25] / —	158	B9	C12
AN26	Single-ended Analog Input	P	—	—	I	VDDA / Analog	I / —	AN[26] / —	—	—	B12
AN27	Single-ended Analog Input	P	—	—	I	VDDA / Analog	I / —	AN[27] / —	157	A10	A12
AN28	Single-ended Analog Input	P	—	—	I	VDDA / Analog	I / —	AN[28] / —	156	B10	A13
AN29	Single-ended Analog Input	P	—	—	I	VDDA / Analog	I / —	AN[29] / —	—	—	D13
AN30	Single-ended Analog Input	P	—	—	I	VDDA / Analog	I / —	AN[30] / —	155	D9	C13
AN31	Single-ended Analog Input	P	—	—	I	VDDA / Analog	I / —	AN[31] / —	154	D10	B13
AN32	Single-ended Analog Input	P	—	—	I	VDDA / Analog	I / —	AN[32] / —	153	C10	B14

Table 3. SPC564A70 signal properties (continued)

Name[1]	Function[2]	P / A / G[3]	PCR PA field [4]	PCR [5]	I/O type	Voltage[6] / Pad type[7]	Status[8]		Package pin No.		
							During reset	After reset	176	208[9]	324
AN33	Single-ended Analog Input	P	—	—	I	VDDA / Analog	I / —	AN[33] / —	152	C11	C14
AN34	Single-ended Analog Input	P	—	—	I	VDDA / Analog	I / —	AN[34] / —	151	C5	D14
AN35	Single-ended Analog Input	P	—	—	I	VDDA / Analog	I / —	AN[35] / —	150	D11	A14
AN36	Single-ended Analog Input	P	—	—	I	VDDA / Analog	I / —	AN[36] / —	174	F4	B4
AN37	Single-ended Analog Input	P	—	—	I	VDDA / Analog	I / —	AN[37] / —	175	E3	A4
AN38	Single-ended Analog Input	P	—	—	I	VDDA / Analog	I / —	AN[38] / —	—	—	C5
AN39	Single-ended Analog Input	P	—	—	I	VDDA / Analog	I / —	AN[39] / —	8	D2	B5
VRH	Voltage Reference High	P	—	—	I	VDDA / —	I / —	—	163	A8	A10
VRL	Voltage Reference Low	P	—	—	I	VDDA / —	I / —	—	162	A9	A11
REFBYBC	Reference Bypass Capacitor Input	P	—	—	I	VDDA / Analog	I / —	—	164	B7	B10
eTPU2											
TCRCLKA IRQ[7] GPIO[113]	eTPU A TCR clock External interrupt request GPIO	P A1 G	01 10 00	113	I I I/O	VDDEH4 / Slow	— / Up	— / Up	—	L4	M2
ETPUA0 ETPUA12_O ETPUA19_O GPIO[114]	eTPU A channel eTPU A channel (output only) eTPU A channel (output only) GPIO	P A1 A2 G	001 010 100 000	114	I/O O O I/O	VDDEH4 / Slow	— / WKPCFG	— / WKPCFG	61	N3	L3
ETPUA1 ETPUA13_O GPIO[115]	eTPU A channel eTPU A channel (output only) GPIO	P A1 G	01 10 00	115	I/O O I/O	VDDEH4 / Slow	— / WKPCFG	— / WKPCFG	60	M3	L4

Table 3. SPC564A70 signal properties (continued)

Name(1)	Function(2)	P / A / G(3)	PCR PA field (4)	PCR (5)	I/O type	Voltage(6) / Pad type(7)	Status(8)		Package pin No.		
							During reset	After reset	176	208(9)	324
ETPUA2 ETPUA14_O GPIO[116]	eTPU A channel eTPU A channel (output only) GPIO	P A1 G	01 10 00	116	I/O O I/O	VDDEH4 / Slow	— / WKPCFG	— / WKPCFG	59	P2	K3
ETPUA3 ETPUA15_O GPIO[117]	eTPU A channel eTPU A channel (output only) GPIO	P A1 G	01 10 00	117	I/O O I/O	VDDEH4 / Slow	— / WKPCFG	GPIO / WKPCFG	58	P1	L2
ETPUA4 ETPUA16_O — FR_B_TX GPIO[118]	eTPU A channel eTPU A channel (output only) — FlexRay transmit data channel B GPIO	P A1 A2 A3 G	0001 0010 — 1000 0000	118	I/O O — O I/O	VDDEH4 / Slow	— / WKPCFG	— / WKPCFG	56	N2	L1
ETPUA5 ETPUA17_O DSPI_B_SCK_LVDS– FR_B_TX_EN GPIO[119]	eTPU A channel eTPU A channel (output only) LVDS negative DSPI clock FlexRay tx data enable for ch. B GPIO	P A1 A2 A3 G	0001 0010 0100 1000 0000	119	I/O O O O I/O	VDDEH4 / Slow + LVDS	— / WKPCFG	— / WKPCFG	54	M4	K4
ETPUA6 ETPUA18_O DSPI_B_SCK_LVDS+ FR_B_RX GPIO[120]	eTPU A channel eTPU A channel (output only) LVDS positive DSPI clock FlexRay receive data channel B GPIO	P A1 A2 A3 G	0001 0010 0100 1000 0000	120	I/O O O I I/O	VDDEH4 / Medium + LVDS	— / WKPCFG	— / WKPCFG	53	L3	J3
ETPUA7 ETPUA19_O DSPI_B_SOUT_LVDS– ETPUA6_O GPIO[121]	eTPU A channel eTPU A channel (output only) LVDS negative DSPI data out eTPU A channel (output only) GPIO	P A1 A2 A3 G	0001 0010 0100 1000 0000	121	I/O O O O I/O	VDDEH4 / Slow + LVDS	— / WKPCFG	— / WKPCFG	52	K3	K2
ETPUA8 ETPUA20_O DSPI_B_SOUT_LVDS+ GPIO[122]	eTPU A channel eTPU A channel (output only) LVDS positive DSPI data out GPIO	P A1 A2 G	001 010 100 000	122	I/O O O I/O	VDDEH4 / Slow + LVDS	— / WKPCFG	— / WKPCFG	51	N1	K1
ETPUA9 ETPUA21_O RCH1_B GPIO[123]	eTPU A channel eTPU A channel (output only) Reaction channel 1B GPIO	P A1 A2 G	001 010 100 000	123	I/O O O I/O	VDDEH4 / Slow	— / WKPCFG	— / WKPCFG	50	M2	J4

Table 3. SPC564A70 signal properties (continued)

Name[1]	Function[2]	P / A / G[3]	PCR PA field [4]	PCR [5]	I/O type	Voltage[6] / Pad type[7]	Status[8]		Package pin No.		
							During reset	After reset	176	208[9]	324
ETPUA10 ETPUA22_O RCH1_C GPIO[124]	eTPU A channel eTPU A channel (output only) Reaction channel 1C GPIO	P A1 A2 G	001 010 100 000	124	I/O O O I/O	VDDEH1 / Slow	— / WKPCFG	— / WKPCFG	49	M1	H3
ETPUA11 ETPUA23_O RCH4_B GPIO[125]	eTPU A channel eTPU A channel (output only) Reaction channel 4B GPIO	P A1 A2 G	001 010 100 000	125	I/O O O I/O	VDDEH1 / Slow	— / WKPCFG	— / WKPCFG	48	L2	J2
ETPUA12 DSPI_B_PCS[1] RCH4_C GPIO[126]	eTPU A channel DSPI B peripheral chip select Reaction channel 4C GPIO	P A1 A2 G	001 010 100 000	126	I/O O O I/O	VDDEH1 / Medium	— / WKPCFG	— / WKPCFG	47	L1	J1
ETPUA13 DSPI_B_PCS[3] GPIO[127]	eTPU A channel DSPI B peripheral chip select GPIO	P A1 G	01 10 00	127	I/O O I/O	VDDEH1 / Medium	— / WKPCFG	— / WKPCFG	46	J4	G4
ETPUA14 DSPI_B_PCS[4] ETPUA9_O RCH0_A GPIO[128]	eTPU A channel DSPI B peripheral chip select eTPU A channel (output only) Reaction channel 0A GPIO	P A1 A2 A3 G	0001 0010 0100 1000 0000	128	I/O O O O I/O	VDDEH1 / Medium	— / WKPCFG	— / WKPCFG	42	J3	G3
ETPUA15 DSPI_B_PCS[5] RCH1_A GPIO[129]	eTPU A channel DSPI B peripheral chip select Reaction channel 1A GPIO	P A1 A2 G	001 010 100 000	129	I/O O O I/O	VDDEH1 / Medium	— / WKPCFG	— / WKPCFG	40	K2	H2
ETPUA16 DSPI_D_PCS[1] RCH2_A GPIO[130]	eTPU A channel DSPI D peripheral chip select Reaction channel 2A GPIO	P A1 A2 G	001 010 100 000	130	I/O O O I/O	VDDEH1 / Slow	— / WKPCFG	— / WKPCFG	39	K1	H1
ETPUA17 DSPI_D_PCS[2] RCH3_A GPIO[131]	eTPU A channel DSPI D peripheral chip select Reaction channel 3A GPIO	P A1 A2 G	001 010 100 000	131	I/O O O I/O	VDDEH1 / Slow	— / WKPCFG	— / WKPCFG	38	H3	F3
ETPUA18 DSPI_D_PCS[3] RCH4_A GPIO[132]	eTPU A channel DSPI D peripheral chip select Reaction channel 4A GPIO	P A1 A2 G	001 010 100 000	132	I/O O O I/O	VDDEH1 / Slow	— / WKPCFG	— / WKPCFG	37	H4	F4

Table 3. SPC564A70 signal properties (continued)

Name[1]	Function[2]	P / A / G[3]	PCR PA field[4]	PCR[5]	I/O type	Voltage[6] / Pad type[7]	Status[8]		Package pin No.		
							During reset	After reset	176	208[9]	324
ETPUA19 DSPI_D_PCS[4] RCH5_A GPIO[133]	eTPU A channel DSPI D peripheral chip select Reaction channel 5A GPIO	P A1 A2 G	001 010 100 000	133	I/O O O I/O	VDDEH1 / Slow	— / WKPCFG	— / WKPCFG	36	J2	G2
ETPUA20 IRQ[8] RCH0_B FR_A_TX GPIO[134]	eTPU A channel External interrupt request Reaction channel 0B FlexRay transmit data channel A GPIO	P A1 A2 A3 G	0001 0010 0100 1000 0000	134	I/O I O O I/O	VDDEH1 / Slow	— / WKPCFG	— / WKPCFG	35	J1	G1
ETPUA21 IRQ[9] RCH0_C FR_A_RX GPIO[135]	eTPU A channel External interrupt request Reaction channel 0C FlexRay receive channel A GPIO	P A1 A2 A3 G	0001 0010 0100 1000 0000	135	I/O I O I I/O	VDDEH1 / Slow	— / WKPCFG	— / WKPCFG	34	G4	E4
ETPUA22 IRQ[10] ETPUA17_O GPIO[136]	eTPU A channel External interrupt request eTPU A channel (output only) GPIO	P A1 A2 G	001 010 100 000	136	I/O I O I/O	VDDEH1 / Slow	— / WKPCFG	— / WKPCFG	32	H2	F2
ETPUA23 IRQ[11] ETPUA21_O FR_A_TX_EN GPIO[137]	eTPU A channel External interrupt request eTPU A channel (output only) FlexRay ch. A transmit enable GPIO	P A1 A2 A3 G	0001 0010 0100 1000 0000	137	I/O I O O I/O	VDDEH1 / Slow	— / WKPCFG	— / WKPCFG	30	H1	F1
ETPUA24 IRQ[12] DSPI_C_SCK_LVDS– GPIO[138]	eTPU A channel External interrupt request LVDS negative DSPI clock GPIO	P A1 A2 G	001 010 100 000	138	I/O I O I/O	VDDEH1 / Slow + LVDS	— / WKPCFG	— / WKPCFG	28	G1	E1
ETPUA25 IRQ[13] DSPI_C_SCK_LVDS+ GPIO[139]	eTPU A channel External interrupt request LVDS positive DSPI clock GPIO	P A1 A2 G	001 010 100 000	139	I/O I O I/O	VDDEH1 / Medium + LVDS	— / WKPCFG	— / WKPCFG	27	G3	E3
ETPUA26 IRQ[14] DSPI_C_SOUT_LVDS– GPIO[140]	eTPU A channel External interrupt request LVDS negative DSPI data out GPIO	P A1 A2 G	001 010 100 000	140	I/O I O I/O	VDDEH1 / Slow + LVDS	— / WKPCFG	— / WKPCFG	26	F3	D3

Table 3. SPC564A70 signal properties (continued)

Name[1]	Function[2]	P / A / G[3]	PCR PA field [4]	PCR [5]	I/O type	Voltage[6] / Pad type[7]	Status[8]		Package pin No.		
							During reset	After reset	176	208[9]	324
ETPUA27 IRQ[15] DSPI_C_SOUT_LVDS+ DSPI_B_SOUT GPIO[141]	eTPU A channel External interrupt request LVDS positive DSPI data out DSPI B data output GPIO	P A1 A2 A3 G	0001 0010 0100 1000 0000	141	I/O I O O I/O	VDDEH1 / Slow + LVDS	— / WKPCFG	— / WKPCFG	25	G2	E2
ETPUA28 DSPI_C_PCS[1] RCH5_B GPIO[142]	eTPU A channel DSPI C peripheral chip select Reaction channel 5B GPIO	P A1 A2 G	001 010 100 000	142	I/O O O I/O	VDDEH1 / Medium	— / WKPCFG	— / WKPCFG	24	F1	D1
ETPUA29 DSPI_C_PCS[2] RCH5_C GPIO[143]	eTPU A channel DSPI C peripheral chip select Reaction channel 5C GPIO	P A1 A2 G	001 010 100 000	143	I/O O O I/O	VDDEH1 / Medium	— / WKPCFG	— / WKPCFG	23	F2	D2
ETPUA30 DSPI_C_PCS[3] ETPUA11_O GPIO[144]	eTPU A channel DSPI C peripheral chip select eTPU A channel (output only) GPIO	P A1 A2 G	001 010 100 000	144	I/O O O I/O	VDDEH1 / Medium	— / WKPCFG	— / WKPCFG	22	E1	C1
ETPUA31 DSPI_C_PCS[4] ETPUA13_O GPIO[145]	eTPU A channel DSPI C peripheral chip select eTPU A channel (output only) GPIO	P A1 A2 G	001 010 100 000	145	I/O O O I/O	VDDEH1 / Medium	— / WKPCFG	— / WKPCFG	21	E2	C2
eMIOS											
EMIOS0 ETPUA0_O ETPUA25_O GPIO[179]	eMIOS channel eTPU A channel (output only) eTPU A channel (output only) GPIO	P A1 A2 G	001 010 100 000	179	I/O O O I/O	VDDEH4 / Slow	— / Up	— / Up	63	T4	AB10
EMIOS1 ETPUA1_O GPIO[180]	eMIOS channel eTPU A channel (output only) GPIO	P A1 G	01 10 00	180	I/O O I/O	VDDEH4 / Slow	— / Up	— / Up	64	T5	AB11
EMIOS2 ETPUA2_O RCH2_B GPIO[181]	eMIOS channel eTPU A channel (output only) Reaction channel 2B GPIO	P A1 A2 G	001 010 100 000	181	I/O O O I/O	VDDEH4 / Slow	— / Up	— / Up	65	N7	W12

Table 3. SPC564A70 signal properties (continued)

Name[1]	Function[2]	P / A / G[3]	PCR PA field (4)	PCR (5)	I/O type	Voltage[6] / Pad type[7]	Status[8]		Package pin No.		
							During reset	After reset	176	208[9]	324
EMIOS3 ETPUA3_O GPIO[182]	eMIOS channel eTPU A channel (output only) GPIO	P A1 G	01 10 00	182	I/O O I/O	VDDEH4 / Slow	— / WKPCFG	— / WKPCFG	66	R6	AA11
EMIOS4 ETPUA4_O RCH2_C GPIO[183]	eMIOS channel eTPU A channel (output only) Reaction channel 2C GPIO	P A1 A2 G	001 010 100 000	183	I/O O O I/O	VDDEH4 / Slow	— / WKPCFG	— / WKPCFG	67	R5	AB12
EMIOS5 ETPUA5_O GPIO[184]	eMIOS channel eTPU A channel (output only) GPIO	P A1 G	01 10 00	184	I/O O I/O	VDDEH4 / Slow	— / WKPCFG	— / WKPCFG	—	—	AA12
EMIOS6 ETPUA6_O GPIO[185]	eMIOS channel eTPU A channel (output only) GPIO	P A1 G	01 10 00	185	I/O O I/O	VDDEH4 / Slow	— / Down	— / Down	68	P7	Y12
EMIOS7 ETPUA7_O GPIO[186]	eMIOS channel eTPU A channel (output only) GPIO	P A1 G	01 10 00	186	I/O O I/O	VDDEH4 / Slow	— / Down	— / Down	69	—	AB13
EMIOS8 ETPUA8_O SCI_B_TX GPIO[187]	eMIOS channel eTPU A channel (output only) eSCI B transmit GPIO	P A1 A2 G	001 010 100 000	187	I/O O O I/O	VDDEH4 / Slow	— / Up	— / Up	70	P8	W13
EMIOS9 ETPUA9_O SCI_B_RX GPIO[188]	eMIOS channel eTPU A channel (output only) eSCI B receive GPIO	P A1 A2 G	001 010 100 000	188	I/O O I I/O	VDDEH4 / Slow	— / Up	— / Up	71	R7	AA13
EMIOS10 DSPI_D_PCS[3] RCH3_B GPIO[189]	eMIOS channel DSPI D peripheral chip select Reaction channel 3B GPIO	P A1 A2 G	001 010 100 000	189	I/O O O I/O	VDDEH4 / Medium	— / WKPCFG	— / WKPCFG	73	N8	Y13
EMIOS11 DSPI_D_PCS[4] RCH3_C GPIO[190]	eMIOS channel DSPI D peripheral chip select Reaction channel 3C GPIO	P A1 A2 G	001 010 100 000	190	I/O O O I/O	VDDEH4 / Medium	— / WKPCFG	— / WKPCFG	75	R8	AB14

Table 3. SPC564A70 signal properties (continued)

Name[1]	Function[2]	P / A / G[3]	PCR PA field[4]	PCR[5]	I/O type	Voltage[6] / Pad type[7]	Status[8]		Package pin No.		
							During reset	After reset	176	208[9]	324
EMIOS12 DSPI_C_SOUT ETPUA27_O GPIO[191]	eMIOS channel DSPI C data output eTPU A channel (output only) GPIO	P A1 A2 G	001 010 100 000	191	I/O O O I/O	VDDEH4 / Medium	— / WKPCFG	— / WKPCFG	76	N10	W15
EMIOS13 DSPI_D_SOUT GPIO[192]	eMIOS channel DSPI D data output GPIO	P A1 G	01 10 00	192	I/O O I/O	VDDEH4 / Medium	— / WKPCFG	— / WKPCFG	77	T8	AA14
EMIOS14 IRQ[0] ETPUA29_O GPIO[193]	eMIOS channel External interrupt request eTPU A channel (output only) GPIO	P A1 A2 G	001 010 100 000	193	I/O I O I/O	VDDEH4 / Slow	— / Down	— / Down	78	R9	AB15
EMIOS15 IRQ[1] GPIO[194]	eMIOS channel External interrupt request GPIO	P A1 G	01 10 00	194	I/O I I/O	VDDEH4 / Slow	— / Down	— / Down	79	T9	Y14
EMIOS16 GPIO[195]	eMIOS channel GPIO	P G	01 00	195	I/O I/O	VDDEH4 / Slow	— / Up	— / Up	—	—	AA15
EMIOS17 GPIO[196]	eMIOS channel GPIO	P G	01 00	196	I/O I/O	VDDEH4 / Slow	— / Up	— / Up	—	—	Y15
EMIOS18 GPIO[197]	eMIOS channel GPIO	P G	01 00	197	I/O I/O	VDDEH4 / Slow	— / Up	— / Up	—	—	AB16
EMIOS19 GPIO[198]	eMIOS channel GPIO	P G	01 00	198	I/O I/O	VDDEH4 / Slow	— / WKPCFG	— / WKPCFG	—	—	AA16
EMIOS20 GPIO[199]	eMIOS channel GPIO	P G	01 00	199	I/O I/O	VDDEH4 / Slow	— / WKPCFG	— / WKPCFG	—	—	AB17
EMIOS21 GPIO[200]	eMIOS channel GPIO	P G	01 00	200	I/O I/O	VDDEH4 / Slow	— / WKPCFG	— / WKPCFG	—	—	W16
EMIOS22 GPIO[201]	eMIOS channel GPIO	P G	01 00	201	I/O I/O	VDDEH4 / Slow	— / Down	— / Down	—	—	Y16
EMIOS23 GPIO[202]	eMIOS channel GPIO	P G	01 00	202	I/O I/O	VDDEH4 / Slow	— / Down	— / Down	80	R11	AA17
EMIOS14[14] GPIO[203]	eMIOS channel GPIO	P G	01 00	203	O I/O	VDDEH7 / Slow	— / Down	— / Down	—	—	H20

Table 3. SPC564A70 signal properties (continued)

Name[1]	Function[2]	P / A / G[3]	PCR PA field[4]	PCR[5]	I/O type	Voltage[6] / Pad type[7]	Status[8]		Package pin No.		
							During reset	After reset	176	208[9]	324
EMIOS15[14] GPIO[204]	eMIOS channel GPIO	P G	01 00	204	O I/O	VDDEH7 / Slow	— / Down	— / Down	—	—	H21
Clock Synthesizer											
XTAL	Crystal oscillator output	P	01	—	O	VDDEH6 / Analog	—	—	93	P16	V22
EXTAL	Crystal oscillator input	P	01	—	I	VDDEH6 / Analog	—	—	92	N16	U22
CLKOUT	System clock output	P	01	229	O	VDDE12 / Fast	—	CLKOUT	—	—	AA20
ENGCLK	Engineering clock output	P	01	214	O	VDDE12 / Fast	—	ENGCLK	—	T14	AB21
Power / Ground											
VDDREG	Voltage regulator supply	—		—	I	5 V	I / —	VDDREG	10	K16	M22
VRCCTL	Voltage regulator control output	—		—	O	—	O / —	VRCCTL	11	N14	V20
VRC33[16]	Internal regulator output	—		—	O	3.3 V	I/O / —	VRC33	13	A15, D1, N6, N12	A21, B1, P4, W7, Y22
	Input for external 3.3 V supply	—		—	I	3.3 V					
VDDA	eQADC high reference voltage	—		—	I	5 V	I / —	VDDA	6	A4, B11	A6, C15
VSSA	eQADC ground/low reference voltage	—		—	I	—	I / —	VSSA	7	A5, A11	A7, A15, B15
VDDPLL	FMPLL supply voltage	—		—	I	1.2 V	I / —	VDDPLL	91	R16	W22
VSTBY	Power supply for standby RAM	—		—	I	0.9 V – 6 V	I / —	VSTBY	12	C1	A3
VDD	Core supply for input or decoupling	—		—	I	1.2 V	I / —	VDD	33, 45, 62, 103, 132, 149, 176	B1, B16, C2, D3, E4, N5, P4, P13, R3, R14, T2, T15	A2, A20, B3, C4, C22, D5, V19, W5, W20, Y4, Y21, AA3, AA22, AB2
VDDE12	External supply input for calibration bus interfaces	—		—	I	3.0 V – 3.6 V	I / —	VDDE12	—	—	—

Table 3. SPC564A70 signal properties (continued)

Name(1)	Function(2)	P / A / G(3)	PCR PA field (4)	PCR (5)	I/O type	Voltage(6) / Pad type(7)	Status(8)		Package pin No.		
							During reset	After reset	176	208(9)	324
VDDE5	External supply input for ENGCLK and CLKOUT	—		—	I	3.0 V – 3.6 V	I / —	VDDE5	—	T13	W17, Y18, AA19, AB20
VDDE-EH	External supply for EBI interfaces	—		—	I	3.0 V – 5.0 V	I / —	VDDE-EH	—	—	R3, W2
VDDEH1A(17)	I/O supply input	—		—	I	3.3 V – 5.0 V	I / —	VDDEH1A(17)	31	—	—
VDDEH1B(17)	I/O supply input	—		—	I	3.3 V – 5.0 V	I / —	VDDEH1B(17)	41	—	—
VDDEH1AB(17)	I/O supply input	—		—	I	3.3 V – 5.0 V	I / —	VDDEH1AB(17)	—	K4	H4
VDDEH4(18)	I/O supply input	—		—	I	3.3 V – 5.0 V	I / —	VDDEH4(18)	—	—	—
VDDEH4A(18)	I/O supply input	—		—	I	3.3 V – 5.0 V	I / —	VDDEH4A(18)	55	—	—
VDDEH4B(18)	I/O supply input	—		—	I	3.3 V – 5.0 V	I / —	VDDEH4B(18)	74	—	—
VDDEH4AB(18)	I/O supply input	—		—	I	3.3 V – 5.0 V	I / —	VDDEH4AB(18)	—	N9	W14
VDDEH6(19)	I/O supply input	—		—	I	3.3 V – 5.0 V	I / —	VDDEH6(19)	—	—	—
VDDEH6A(19)	I/O supply input	—		—	I	3.3 V – 5.0 V	I / —	VDDEH6A(19)	95	—	—
VDDEH6B(19)	I/O supply input	—		—	I	3.3 V – 5.0 V	I / —	VDDEH6B(19)	110	—	—
VDDEH6AB(19)	I/O supply input	—		—	I	3.3 V – 5.0 V	I / —	VDDEH6AB(19)	—	F13	H19, U19
VDDEH7(20)	I/O supply input	—		—	I	3.3 V – 5.0 V	I / —	VDDEH7	—	D12	D15
VDDEH7A(20)	I/O supply input	—		—	I	3.3 V – 5.0 V	I / —	VDDEH7A	125	—	—

Table 3. SPC564A70 signal properties (continued)

Name[1]	Function[2]	P / A / G[3]	PCR PA field[4]	PCR[5]	I/O type	Voltage[6] / Pad type[7]	Status[8]		Package pin No.		
							During reset	After reset	176	208[9]	324
VDDEH7B[20]	I/O supply input	—		—	I	3.3 V – 5.0 V	I / —	VDDEH7B	138	—	—
VSS	Ground	—		—	I	—	I / —	VSS	15, 29, 43, 57, 72, 90, 94, 96, 108, 115, 127, 133, 140	A1, A16, B2, B15, C3, C14, D4, D13, G7, G8, G9, G10, H7, H8, H9, H10, J7, J8, J9, J10, K7, K8, K9, K10, M16, N4, N13, P3, P14, R2, R15, T1, T16	A1, A22, B2, B21, C3, C20, D4, D19, J9, J10, J11, J12, J13, K9, K10, K11, K12, K13, K14, L9, L10, L11, L12, L13, L14, M11, M12, M13, M14, N9, N10, N12, N13, N14, P9, P10, P12, P13, P14, T21, T22, W4, W19, Y3, Y20, AA2, AA21, AB1, AB22

1. The suffix “_O” identifies an output-only eTPU channel
2. For each pin in the table, each line in the Function column is a separate function of the pin. For all I/O pins the selection of primary pin function or secondary function or GPIO is done in the SIU except where explicitly noted. See the Signal details table for a description of each signal.
3. The P/A/G column indicates the position a signal occupies in the muxing order for a pin—Primary, Alternate 1, Alternate 2, Alternate 3, or GPIO. Signals are selected by setting the PA field value in the appropriate PCR register in the SIU module. The PA field values are as follows: P - 0b0001, A1 - 0b0010, A2 - 0b0100, A3 - 0b1000, or G - 0b0000. Depending on the register, the PA field size can vary in length. For PA fields having fewer than four bits, remove the appropriate number of leading zeroes from these values.
4. The Pad Configuration Register (PCR) PA field is used by software to select pin function.
5. Values in the PCR column refer to registers in the System Integration Unit (SIU). The actual register name is “SIU_PCR” suffixed by the PCR number. For example, PCR[190] refers to the SIU register named SIU_PCR190.
6. The VDDE and VDDEH supply inputs are broken into segments. Each segment of slow I/O pins (VDDEH) may have a separate supply in the 3.3 V to 5.0 V range (–10%/+5%). Each segment of fast I/O (VDDE) may have a separate supply in the 1.8 V to 3.3 V range (+/– 10%).
7. See *Table 4* for details on pad types.

8. The Status During Reset pin is sampled after the internal POR is negated. Prior to exiting POR, the signal has a high impedance. Terminology is O (output), I (input), Up (weak pull up enabled), Down (weak pull down enabled), Low (output driven low), High (output driven high). A dash for the function in this column denotes that both the input and output buffer are turned off. The signal name to the left or right of the slash indicates the pin is enabled.
9. LBGA208 is available upon specific request. Please contact your ST sales office for details.
10. When used as ETRIG, this pin must be configured as an input. For GPIO it can be configured either as an input or output.
11. Maximum frequency is 50 kHz
12. PCR219 controls two different pins: MCKO and GPIO[219]. Please refer to Pad Configuration Register 219 section in SIU chapter of device reference manual for details.
13. On LQFP176 and LBGA208 packages, this pin is tied low internally.
14. Output only
15. This signal name is used to support legacy naming.
16. Do not use VRC33 to drive external circuits.
17. VDDEH1A, VDDEH1B and VDDEH1AB are shorted together in all production packages. The separation of the signal names is present to support legacy naming, however they should be considered as the same signal in this document.
18. VDDEH4, VDDEH4A, VDDEH4B and VDDEH4AB are shorted together in all production packages. The separation of the signal names is present to support legacy naming, however they should be considered as the same signal in this document.
19. VDDEH6, VDDEH6A, VDDEH6B and VDDEH6AB are shorted together in all production packages. The separation of the signal names is present to support legacy naming, however they should be considered as the same signal in this document.
20. VDDEH7, VDDEH7A and VDDE7B are shorted together in all production packages. The separation of the signal names is present to support legacy naming, however they should be considered as the same signal in this document.

Table 4. Pad types

Pad Type	Name	I/O Voltage Range
Slow	pad_ssr_hv	3.0V - 5.5 V
Medium	pad_msr_hv	3.0 V - 5.5 V
Fast	pad_fc	3.0 V - 3.6 V
MultiV(1),(2)	pad_multv_hv	3.0 V - 5.5 V (high swing mode) 3.0 V - 3.6 V (low swing mode)
Analog	pad_ae_hv	0.0 - 5.5 V
LVDS	pad_lo_lv	—

1. Multivoltage pads are automatically configured in low swing mode when a JTAG or Nexus function is selected, otherwise they are high swing.
2. VDDEH7 supply cannot be below 4.5 V when in low-swing mode.

3.2 Signal Details

Table 5. Signal details

Signal	Module or function	Description
CLKOUT	Clock Generation	SPC564A70 clock output for the calibration bus interface
ENGCLK	Clock Generation	Clock for external ASIC devices
EXTAL	Clock Generation	Input pin for an external crystal oscillator or an external clock source based on the value driven on the PLLREF pin at reset
PLLREF	Clock Generation Reset/Configuration	PLLREF is used to select whether the oscillator operates in xtal mode or external reference mode from reset. PLLREF = 0 selects external reference mode. On the PBGA324 package, PLLREF is bonded to the ball used for PLLCFG[0] for compatibility with previous devices. For the 176-pin QFP and 208-ball BGA packages: 0: External reference clock is selected 1: XTAL oscillator mode is selected For the 324-ball BGA package: If RSTCFG is 0: 0: External reference clock is selected 1: XTAL oscillator mode is selected If RSTCFG is 1, XTAL oscillator mode is selected.
XTAL	Clock Generation	Crystal oscillator input
DSPI_B_SCK_LVDS– DSPI_B_SCK_LVDS+	DSPI	LVDS pair used for DSPI_B TSB mode transmission
DSPI_B_SOUT_LVDS– DSPI_B_SOUT_LVDS+	DSPI	LVDS pair used for DSPI_B TSB mode transmission

Table 5. Signal details (continued)

Signal	Module or function	Description
DSPI_C_SCK_LVDS– DSPI_C_SCK_LVDS+	DSPI	LVDS pair used for DSPI_C TSB mode transmission
DSPI_C_SOUT_LVDS– DSPI_C_SOUT_LVDS+	DSPI	LVDS pair used for DSPI_C TSB mode transmission
DSPI_B_PCS[0] DSPI_C_PCS[0] DSPI_D_PCS[0]	DSPI_B – DSPI_D	Peripheral chip select when device is in master mode—slave select when used in slave mode
DSPI_B_PCS[1:5] DSPI_C_PCS[1:5] DSPI_D_PCS[1:5]	DSPI_B – DSPI_D	Peripheral chip select when device is in master mode—not used in slave mode
DSPI_B_SCK DSPI_C_SCK DSPI_D_SCK	DSPI_B – DSPI_D	DSPI clock—output when device is in master mode; input when in slave mode
DSPI_B_SIN DSPI_C_SIN DSPI_D_SIN	DSPI_B – DSPI_D	DSPI data in
DSPI_B_SOUT DSPI_C_SOUT DSPI_D_SOUT	DSPI_B – DSPI_D	DSPI data out
eMIOS[0:23]	eMIOS	eMIOS I/O channels
AN[0:39]	eQADC	Single-ended analog inputs for analog-to-digital converter
AN[0:7]/DAN+	eQADC	Differential analog input pair for analog-to-digital converter with pull-up/pull-down functionality
AN[0:7]/DAN–	eQADC	Differential analog input pair for analog-to-digital converter with pull-up/pull-down functionality
FCK	eQADC	eQADC free running clock for eQADC SSI
MA[0:2]	eQADC	These three control bits are output to enable the selection for an external Analog Mux for expansion channels.
REFBYPC	eQADC	Bypass capacitor input
SDI	eQADC	Serial data in
SDO	eQADC	Serial data out
SDS	eQADC	Serial data select
VRH	eQADC	Voltage reference high input
VRL	eQADC	Voltage reference low input
SCI_A_RX SCI_B_RX SCI_C_RX	eSCI_A – eSCI_C	eSCI receive
SCI_A_TX SCI_B_TX SCI_C_TX	eSCI_A – eSCI_C	eSCI transmit

Table 5. Signal details (continued)

Signal	Module or function	Description
ETPU_A[0:31]	eTPU	eTPU I/O channel
RCH0_[A:C] RCH1_[A:C] RCH2_[A:C] RCH3_[A:C] RCH4_[A:C] RCH5_[A:C]	eTPU2 Reaction Module	eTPU2 reaction channels. Used to control external actuators, e.g., solenoid control for direct injection systems and valve control in automatic transmissions
TCRCLKA	eTPU2	Input clock for TCR time base
CAN_A_TX CAN_B_TX CAN_C_TX	FlexCAN_A – FlexCAN_C	FlexCAN transmit
CAN_A_RX CAN_B_RX CAN_C_RX	FlexCAN_A – FlexCAN_C	FlexCAN receive
FR_A_RX FR_B_RX	FlexRay	FlexRay receive (Channels A, B)
FR_A_TX_EN FR_B_TX_EN	FlexRay	FlexRay transmit enable (Channels A, B)
FR_A_TX FR_B_TX	FlexRay	FlexRay transmit (Channels A, B)
JCOMP	JTAG	Enables the JTAG TAP controller
TCK	JTAG	Clock input for the on-chip test logic
TDI	JTAG	Serial test instruction and data input for the on-chip test logic
TDO	JTAG	Serial test data output for the on-chip test logic
TMS	JTAG	Controls test mode operations for the on-chip test logic
$\overline{\text{EVTI}}$	Nexus	$\overline{\text{EVTI}}$ is an input that is read on the negation of $\overline{\text{RESET}}$ to enable or disable the Nexus Debug port. After reset, the $\overline{\text{EVTI}}$ pin is used to initiate program synchronization messages or generate a breakpoint.
$\overline{\text{EVTO}}$	Nexus	Output that provides timing to a development tool for a single watchpoint or breakpoint occurrence
MCKO	Nexus	MCKO is a free running clock output to the development tools which is used for timing of the MDO and $\overline{\text{MSEO}}$ signals.
MDO[0:11]	Nexus	Trace message output to development tools. This pin also indicates the status of the crystal oscillator clock following a power-on reset, when MDO[0] is driven high until the crystal oscillator clock achieves stability and is then negated.
$\overline{\text{MSEO}}$[0:1]	Nexus	Output pin—Indicates the start or end of the variable length message on the MDO pins

Table 5. Signal details (continued)

Signal	Module or function	Description
RDY	Nexus	Nexus Ready Output (RDY)—Indicates to the development tools that data is ready to be read from or written to the Nexus read/write access registers.
BOOTCFG[0:1]	SIU – Configuration	Two BOOTCFG signals are implemented in SPC564A70 MCUs. The BAM program uses the BOOTCFG0 bit to determine where to read the reset configuration word, and whether to initiate a FlexCAN or eSCI boot. The BOOTCFG1 pin is sampled during the assertion of the RSTOUT signal, and the value is used to update the RSR and the BAM boot mode. See reference manual section “Reset Configuration Half Word (RCHW)” for details on the RCHW. The table “Boot Modes” in reference manual section “BAM Program Operation” defines the boot modes specified by the BOOTCFG1 pin. The following values are for BOOTCFG[0:1}: 00: Boot from internal flash memory 01: FlexCAN/eSCI boot 10: Boot from external memory using calibration bus 11: Reserved Note: For the 176-pin QFP and 208-ball BGA packages BOOTCFG[0] is always 0 since the EBI interface is not available.
WKPCFG	SIU – Configuration	The WKPCFG pin is applied at the assertion of the internal reset signal (assertion of RSTOUT), and is sampled four clock cycles before the negation of the RSTOUT pin. The value is used to configure whether the eTPU and eMIOS pins are connected to internal weak pull up or weak pull down devices after reset. The value latched on the WKPCFG pin at reset is stored in the Reset Status Register (RSR), and is updated for all reset sources except the Debug Port Reset and Software External Reset. 0:Weak pulldown applied to eTPU and eMIOS pins at reset 1:Weak pullup applied to eTPU and eMIOS pins at reset
ETRIG[2:3]	SIU – eQADC Triggers	External signal eTRIGx triggers eQADC CFIFOx
GPIO[206] ETRIG0 (Input)	SIU – eQADC Triggers	External signal eTRIGx triggers eQADC CFIFOx
GPIO[207] ETRIG1 (Input)	SIU – eQADC Triggers	External signal eTRIGx triggers eQADC CFIFOx

Table 5. Signal details (continued)

Signal	Module or function	Description
IRQ[0:5] IRQ[7:15]	SIU – External Interrupts	The IRQ[0:15] pins connect to the SIU IRQ inputs. IMUX Select Register 1 is used to select the IRQ[0:15] pins as inputs to the IRQs. See reference manual section “External IRQ Input Select Register (SIU_EIISR)” for more information.
NMI	SIU – External Interrupts	Non-Maskable Interrupt
GPIO[12:17] GPIO[75:110] GPIO[113:145] GPIO[179:204] GPIO[206:213] GPIO[219] GPIO[244:245]	SIU – GPIO	Configurable general purpose I/O pins. Each GPIO input and output is separately controlled by an 8-bit input (GPDI) or output (GPDO) register. Additionally, each GPIO pin is configured using a dedicated SIU_PCR register. The GPIO pins are generally multiplexed with other I/O pin functions. See the following reference manual sections for more information: – “Pad Configuration Registers (SIU_PCR)” – “GPIO Pin Data Output Registers (SIU_GPDO0_3 – SIU_GPDO412_413)” – “GPIO Pin Data Input Registers (SIU_GPDI0_3 – SIU_GPDI_232)”
$\overline{\text{RESET}}$	SIU – Reset	The $\overline{\text{RESET}}$ pin is an active low input. The $\overline{\text{RESET}}$ pin is asserted by an external device during a power-on or external reset. The internal reset signal asserts only if the $\overline{\text{RESET}}$ pin asserts for 10 clock cycles. Assertion of the $\overline{\text{RESET}}$ pin while the device is in reset causes the reset cycle to start over. The $\overline{\text{RESET}}$ pin has a glitch detector which detects spikes greater than two clock cycles in duration that fall below the switch point of the input buffer logic of the VDDEH input pins. The switch point lies between the maximum VIL and minimum VIH specifications for the VDDEH input pins.
RSTCFG	SIU – Reset	Used to enable or disable the PLLREF and the BOOTCFG[0:1] configuration signals. 0:Get configuration information from BOOTCFG[0:1] and PLLREF 1:Use default configuration of booting from internal flash with crystal clock source For the 176-pin QFP and 208-ball BGA packages RSTCFG is always 0, so PLLREF and BOOTCFG signals are used.

Table 5. Signal details (continued)

Signal	Module or function	Description
$\overline{\text{RSTOUT}}$	SIU – Reset	The $\overline{\text{RSTOUT}}$ pin is an active low output that uses a push/pull configuration. The $\overline{\text{RSTOUT}}$ pin is driven to the low state by the MCU for all internal and external reset sources. There is a delay between initiation of the reset and the assertion of the $\overline{\text{RSTOUT}}$ pin. See reference manual section “RSTOUT” for details.

Table 6. Power/ground segmentation

Power segment	Voltage	I/O pins powered by segment
VDDE5	3.0 V – 3.6 V	DATA[0:15], CLKOUT, ENGCLK
VDDE12	3.0 V – 3.6 V	CAL_CS0, CAL_CS2, CAL_CS3, CAL_ADDR[12:30], CAL_DATA[0:15], CAL_RD_WR, CAL_WE0, CAL_WE1, CAL_OE, CAL_TS
VDDE-EH	3.0 V – 5.5 V	FR_A_TX, FR_A_TX_EN, FR_A_RX, FR_B_TX, FR_B_TX_EN, FR_B_RX
VDDEH1	3.3 V – 5.5 V	ETPUA[10:31]
VDDEH4	3.3 V – 5.5 V	EMIOS[0:23], TCRCLKA, ETPUA[0:9]
VDDEH6	3.3 V – 5.5 V	$\overline{\text{RESET}}$, $\overline{\text{RSTOUT}}$, PLLREF, PLLCFG1, RSTCFG, BOOTCFG0, BOOTCFG1, WKPCFG, CAN_A_TX, CAN_A_RX, CAN_B_TX, CAN_B_RX, CAN_C_TX, CAN_C_RX, SCI_A_TX, SCI_A_RX, SCI_B_TX, SCI_B_RX, SCI_C_TX, SCI_C_RX, DSPI_B_SCK, DSPI_B_SIN, DSPI_B_SOUT, DSPI_B_PCS[0:5], EXTAL, XTAL
VDDEH7	3.3 V – 5.5 V	EMIOS14, EMIOS15, GPIO[98:99], GPIO[203:204], GPIO[206], GPIO[207], GPIO[219], EVTI, EVTO, MDO[4:11], MSEO0, MSEO1, RDY, TCK, TDI, TDO, TMS, JCOMP, DSPI_A_SCK, DSPI_A_SIN, DSPI_A_SOUT, DSPI_A_PCS[0:1], DSPI_A_PCS[4:5], AN12-SDS, AN13-SDO, AN14-SDI, AN15-FCK
VDDA	5.0 V	AN[0:11], AN[16:39], VRH, VRL, REFBYBC
VRC33	3.3 V	MCKO, MDO[0:3]
Other power segments		
VDDREG	5.0 V	—
VRCCTL	—	—
VDDPLL	1.2 V	—
VSTBY	0.9 V – 6.0 V	—
VSS	—	—

4 Resets

Note: Throughout this text the phrase “reset configuration pins” is used to refer to WKPCFG, BOOTCFG, and PLLREF pins.

Not all packages have BOOTCFG[0]. In this case, BOOTCFG[0] is sampled as 0b0.

4.1 Reset sources

This device supports the following system reset sources:

- Power-on Reset
- External Reset
- Loss of Lock Reset
- Loss of Clock Reset
- Watchdog Timer/Debug Reset
- JTAG Reset
- Software System Reset
- Software External Reset (resets external resources but not the device)

All reset sources are processed by the reset controller, which monitors the reset input sources, and upon detection of a reset event, resets internal logic and controls the assertion of the $\overline{\text{RSTOUT}}$ pin. The Software External Reset only causes the $\overline{\text{RSTOUT}}$ pin to be asserted for a number of clock cycles determined by the PLL mode (refer to *Section 4.3.2, $\overline{\text{RSTOUT}}$*), and does not reset the device.

For all reset sources, the BOOTCFG[0:1] and PLLREF signals are used to determine the boot mode and configuration of the FMPLL, respectively. *Table 7* shows the options for BOOTCFG[0:1] and *Table 8* for PLLCFG[0:2]. Refer to *Chapter 16: Frequency-modulated phase locked loop (FMPLL)*, for information on the FMPLL during reset.

Table 7. BOOTCFG options

BOOTCFG[0]	BOOTCFG[1]	Meaning
0	0	Boot from internal flash memory
0	1	FlexCAN / eSCI boot
1	0	Boot from external memory (no arbitration)(1)
1	1	Reserved

1. This mode is only available in packages that have an EBI.

Table 8. PLLREF options

PLLREF	Clock mode
0	Normal mode with external reference
1	Normal mode with crystal reference

The Reset Status Register (SIU_RSR) gives the source, or sources, of the last reset and indicates whether a glitch has occurred on the RESET pin. The SIU_RSR is updated for all reset sources except JTAG reset.

All reset sources initiate execution of the Boot Assist Module (BAM) program with the exception of the Software External Reset.

The Reset Configuration Half Word (RCHW) determines the MCU configuration after reset. The RCHW is stored in internal flash, or a default configuration is used. During reset, the RCHW is read from internal flash memory. The BOOTCFG[0:1][b] pins are defined in *Chapter 15: System Integration Unit (SIU).* The BAM program reads the value of the BOOTCFG[0:1] pins from field SIU_RSR[BOOTCFG], then reads the RCHW from the specified location, and then uses the RCHW value to determine and execute the specified boot procedure. Note: the reset controller latches the value on the BOOTCFG input to the SIU four clock cycles prior to the negation of RSTOUT.

4.2 Reset vector

The reset vector for this device is 0xFFFF_FFFC. This is a fixed location in the BAM. The BAM program executes after every internal reset. The BAM program determines where to branch after its execution completes based on the value on the BOOTCFG[0:1] pins. See the BAM chapter’s functional description for details on the BAM program operation and branch location to application software.

4.3 Reset pins

4.3.1 RESET

The RESET pin is an active low input. The RESET pin must be asserted by an external device during a power-on or whenever an external reset is required. The internal reset signal asserts only if the RESET pin asserts for 10 clock cycles. Assertion of the RESET pin while the reset state machine is already processing a reset causes the reset cycle to start over. The RESET pin has a glitch detector which detects spikes greater than two clocks in duration that fall below the switch point of the input buffer logic of the VDDEH input pins. The switch point lies between the maximum VIL and minimum VIH specifications for the VDDEH input pins. *Figure 2* and *Figure 3* show logic flows of the reset state machine on assertion of RESET.

4.3.2 RSTOUT

The RSTOUT pin is an active low output that uses a push/pull configuration. The RSTOUT pin is driven to the low state by the MCU for all internal and external reset sources.

Depending on the PLL configuration, External Reference or Crystal Mode, the RSTOUT pin is asserted after a delay defined in *Table 9*, plus four cycles for sampling of the configuration pins.

---

[b] BOOTCFG[0] is not available on all packages.

The $\overline{\text{RSTOUT}}$ pin can also be asserted by a write to the SER bit of the System Reset Control Register (SIU_SRCR). Asserting SIU_SRCR[SER], the $\overline{\text{RSTOUT}}$ duration will follow the value specified in *Table 9*.

Table 9. Timing for reset sources

Reset source	Description	Number of clocks	
		Crystal reference	External reference
POR	Power On Reset	2400	16000
ER	External Reset ($\overline{\text{RESET}}$ pin)	2900	16500
LLR	Loss of Lock Reset	3400	17000
WTR	Watchdog Timer (core) or Debug Reset	3900	17500
SWTR	System Software Watchdog Reset	4900	18500
LCR	Loss of Clock Reset	5400	19000
SSR	SIU Software External Reset	5900	19500
SER	SIU Software System Reset	6400	20000

4.4 FMPLL lock gating signal

The FMPLL Loss of Lock reset request is connected to both a reset request and a reset gating signal in the SIU. The FMPLL asserts the Loss of Lock reset request until the PLL has achieved lock.

4.5 Reset source descriptions

For the following reset source descriptions refer to the reset flow diagrams in *Figure 2* and *Figure 3*. *Figure 2* shows the reset flow for assertion of the $\overline{\text{RESET}}$ pin. *Figure 3* shows the internal processing of reset for all reset sources.

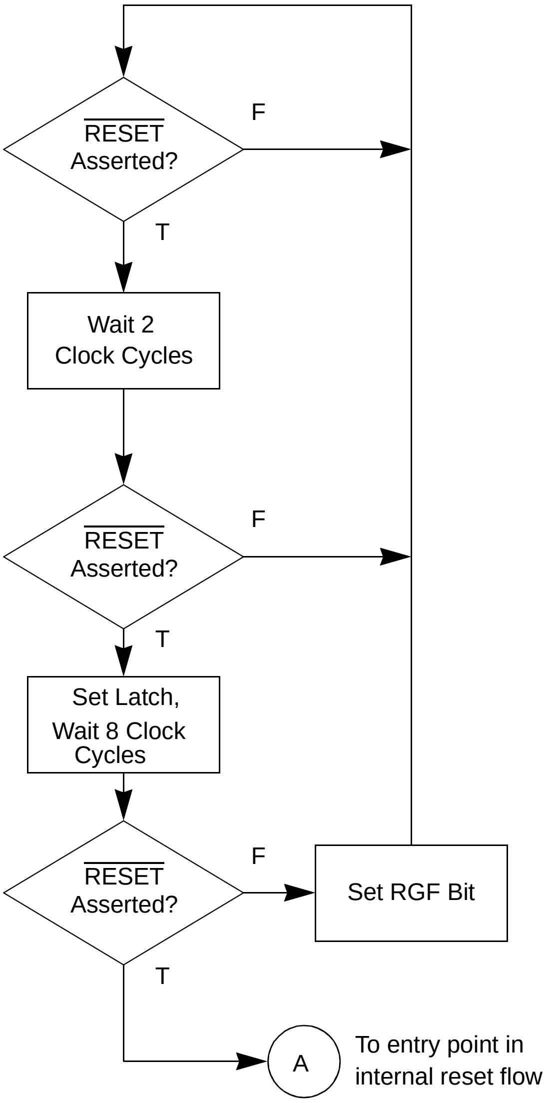
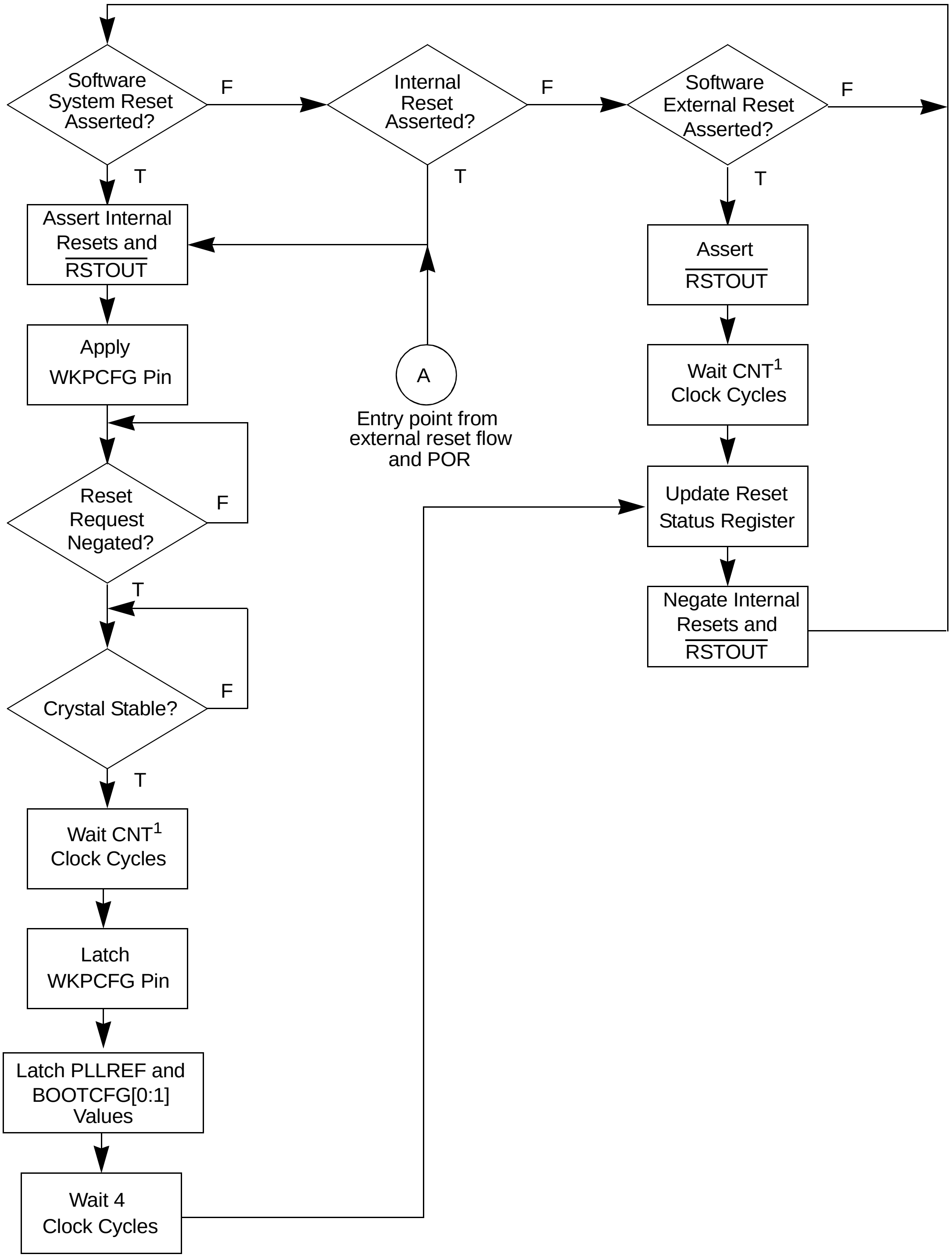


Figure 2. External reset flow diagram



NOTES:

1. The clock count CNT depends on the reset source and type of clock reference. Please refer to *Table 9*.

Figure 3. **Internal reset flow diagram**

4.5.1 Power-on reset (POR)

The internal power-on reset signal is asserted when either the supply voltages, nominally 3.3 V or 1.2 V or the RESET supply (VDDEH6a) fall below defined values. See the device data sheet for the threshold specifications of these voltages. The output signals from the power-on reset circuits are active low signals. All power-on reset output signals are combined into one POR signal at the 1.2 V level and input to the reset controller. Although assertion of the power-on reset signal causes reset, the RESET pin must be asserted during a power-on reset to guarantee proper operation of the MCU.

The PLLREF pin determines the source of reference clock, either crystal or external, at the negation of RSTOUT. During the assertion of RSTOUT, the system clock will switch to the input specified by the PLLREF pin. The value on the PLLREF pin must be kept constant during reset to avoid transients in the system clock. See *Section 16.2.3, Modes of operation*, for more details.

The signal on the WKPCFG pin determines whether weak pull up or pull down devices are enabled after reset on the eTPU and eMIOS pins. The WKPCFG pin is applied on the assertion of the internal reset signal (assertion of RSTOUT). See *Section 4.7.3, Reset weak pull up/down configuration* for more information.

Once a power-on-reset is triggered, if the clock reference is the crystal (PLLREF = 1), then the clock to the whole chip, including the reset state machine, is kept frozen until the Clock Quality Monitor detects that the crystal oscillator has already stabilized. If the clock reference is external (PLLREF = 0) the clock is released to the system immediately. When the clock is stable and released to the chip, the reset controller counts a predetermined number of clock cycles (refer to *Section 4.3.2, RSTOUT*) before negating the RSTOUT pin. The WKPCFG and BOOTCFG[0:1] pins are sampled four clock cycles before the negation of RSTOUT, and the associated bits/fields are updated in the SIU_RSR. In addition, SIU_RSR[PORS] and SIU_RSR[ERS] are set, and all other reset status bits are cleared in the SIU_RSR.

4.5.2 External reset

When the reset controller detects assertion of the RESET pin, the internal reset signal and RSTOUT pin are asserted. The values on the WKPCFG pin and PLLCFG pins are applied at the assertion of the internal reset signal (assertion of RSTOUT). Once the RESET pin is negated and the FMPLL Loss of Lock reset request signal is negated, the reset controller waits for a predetermined number of clock cycles (refer to *Section 4.3.2, RSTOUT*). Once the clock count finishes, the reset configuration pins are latched. The reset controller then waits four clock cycles before negating RSTOUT, and the associated bits/fields are updated in the SIU_RSR. In addition, SIU_RSR[ERS] is set, and all other reset status bits in the SIU_RSR are cleared.

4.5.3 Loss of lock

A Loss of Lock Reset occurs when the FMPLL loses lock and the Loss of Lock Reset Enable (LOLRE) bit in the FMPLL Synthesizer Control Register (SYNCR) is set. The internal reset signal and RSTOUT pin are asserted. The value on the WKPCFG pin is applied at the assertion of the internal reset signal (assertion of RSTOUT), as is the PLLREF value. Once the FMPLL Loss of Lock reset request signal is negated, the reset controller waits for a predetermined number of clock cycles (refer to *Section 4.3.2, RSTOUT*). Once the clock count finishes, the WKPCFG and BOOTCFG[0:1] pins are sampled. The reset controller then waits four clock cycles before negating RSTOUT, and the associated bits/fields are updated in the SIU_RSR. In addition, SIU_RSR[LLRS] is set, and all other reset status bits in the SIU_RSR are cleared. Refer to *Section 16.5.3, Lock detection* for more information on loss of lock.

4.5.4 Loss of clock

A Loss of Clock Reset occurs when the Clock Quality Monitor Module (CQM) detects a failure in either the reference signal or FMPLL output, and the Loss of Clock Reset Enable (LOCRE) bit in the SYNCR is set. The internal reset signal and RSTOUT pin are asserted. The value on the WKPCFG pin is applied at the assertion of the internal reset signal (assertion of RSTOUT), as is the PLLREF value. Once the Loss of Clock reset request signals is negated, the reset controller waits for a predetermined number of clock cycles (refer to *Section 4.3.2, RSTOUT*). Once the clock count finishes, the WKPCFG and BOOTCFG[0:1] pins are sampled. The reset controller then waits four clock cycles before negating RSTOUT, and the associated bits/fields are updated in the SIU_RSR. In addition, SIU_RSR[LCRS] is set, and all other reset status bits in the SIU_RSR are cleared. Refer to *Section 16.5.3, Lock detection,* for more information on loss of clock.

The CQM module, when enabled, can generate either a system reset or an interrupt signal (refer to *Section 16.5.4, Loss-of-clock detection* for details).

4.5.5 Core watchdog timer/debug reset

There are two watchdog timer resets: A core watchdog and a platform watchdog.

A Core Watchdog Timer Reset occurs when the e200z4 core watchdog timer is enabled (the e200z4 core watchdog is counting core clocks, which is different than the peripheral/platform clocks), and a time-out occurs with the Enable Next Watchdog Timer (EWT) and Watchdog Timer Interrupt Status (WIS) bits set in the Timer Status Register, and with the Watchdog Reset Control (WRC) field in the Timer Control Register configured for a reset. SIU_RSR[WDRS] is also set when a debug reset command is issued from a debug tool. To determine whether SIU_RSR[WDRS] was set due to a Watchdog Timer or Debug Reset, see the WRS field in the e200z4 core Timer Status Register.

The effect of a Watchdog Timer or Debug Reset request is the same for the reset controller. The internal reset signal and RSTOUT pin are asserted. The value on the WKPCFG pin is applied at the assertion of the internal reset signal (assertion of RSTOUT), as is the PLLREF value. Once the Watchdog Timer/Debug reset request is negated and the FMPLL Loss of Lock reset request signal is negated, the reset controller waits for a predetermined number of clock cycles (refer to *Section 4.3.2, RSTOUT*). Once the clock count finishes the reset configuration pins are sampled. The reset controller then waits four clock cycles before negating RSTOUT, and the associated bits/fields are updated in the SIU_RSR. In addition, SIU_RSR[WDRS] is set, and all other reset status bits in the SIU_RSR are cleared.

Refer to the *e200z4 Power Architecture® Core Reference Manual* for descriptions of the Timer Status Register and Timer Control Register, as for more information on the core watchdog timer and debug operation.Refer to *Chapter 19: Software Watchdog Timer (SWT)*, for more information on the platform watchdog.

4.5.6 JTAG reset

A system reset occurs when JTAG is enabled and either the EXTEST, CLAMP, or HIGHZ instructions are executed by the JTAG controller. The internal reset signal is asserted. The state of the RSTOUT pin is determined by the JTAG instruction. The value on the WKPCFG pin is applied at the assertion of the internal reset signal, as is the PLLREF value. After the JTAG reset request is negated, the reset controller waits for a predetermined number of clock cycles (refer to *Section 4.3.2, RSTOUT*). Once the clock count finishes the WKPCFG and BOOTCFG[0:1] pins are sampled, and the associated bits/fields are updated in the SIU_RSR. The reset status bits in the SIU_RSR are unaffected. Refer to *Chapter 33: JTAG Controller (JTAGC),* for more information.

4.5.7 Software system reset

A Software System Reset is caused by a write to field SIU_SRCR[SSR]; see *Section 15.6.5, System Reset Control Register (SIU_SRCR).* A write of ‘1’ to SIU_SRCR[SSR] causes an internal reset of the MCU. The internal reset signal and RSTOUT pin are asserted. The value on the WKPCFG pin is applied at the assertion of the internal reset signal (assertion of RSTOUT), as is the PLLREF value. SIU_SRCR[SSR] is automatically cleared and the reset controller waits for a predetermined number of clock cycles (refer to *Section 4.3.2, RSTOUT*). Once the clock count finishes the WKPCFG and BOOTCFG[0:1] pins are sampled. The reset controller then waits four clock cycles before negating RSTOUT, and the associated bits/fields are updated in the SIU_RSR. In addition, SIU_RSR[SSRS] is set, and all other reset status bits in the SIU_RSR are cleared.

4.5.8 Software external reset

A write of ‘1’ to field SIU_SRCR[SER] causes the external RSTOUT pin to be asserted for a predetermined number of clock cycles (refer to *Section 4.3.2, RSTOUT*). SIU_SRCR[SER] automatically clears after the clock counting expires. A Software External Reset does not cause a reset of the MCU, the BAM program is not executed, the PLLREF, BOOTCFG, and WKPCFG pins are not sampled. Field SIU_RSR[SERF] is set, but no other status bits are affected. SIU_RSR[SERF] is not automatically cleared and remains set until cleared by software or another reset besides the Software External Reset occurs.

For a Software External Reset, the e200z4 core will continue to execute instructions, timers that are enabled will continue to operate, and interrupt requests will continue to be processed. The application must ensure that devices connected to RSTOUT are not accessed during a Software External Reset, and it must determine how to manage MCU resources when using the Software External Reset.

4.6 Reset registers in the SIU

The System Integration Unit (SIU) on this device includes two registers, SIU_RSR and SIU_SRCR, that affect the reset behavior of this device. See *Chapter 15: System Integration Unit (SIU),* for descriptions of these registers.

4.7 Reset configuration

4.7.1 Reset configuration half word (RCHW)

RCHW overview

The Reset Configuration Half Word (RCHW) defines boot options and must be programmed in a choice of predefined locations in internal flash. The word at the word address boundary after the RCHW must be programmed with the user application's starting address. The BAM passes control to the user application at this starting address.

On every reset except the Software External Reset (SER), in internal or external boot modes, the BAM attempts to read the RCHW from internal or external memory respectively. The locations for the RCHW are given in *Table 10*. For internal boot, the predefined locations are searched in the order given in the table. If a valid RCHW is not found in internal boot mode or in external boot mode, the BAM initiates the serial boot mode. Note that in serial boot mode, a user defined start address must still be supplied as part of the download protocol. Refer to the BAM Chapter for complete details.

Table 10. RCHW location

Boot mode	Address
External	0x0000_0000
Internal	0x0000_0000 0x0000_4000 0x0001_0000 0x0001_C000 0x0002_0000 0x0003_0000

RCHW structure

When booting from the external flash device, the RCHW must reside in the first 16 bits of memory.

Figure 4. Reset Configuration Half Word

BOOT_BLOCK_ADDRESS + 0x0000_0000

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
				SWT	WTE	PS0	VLE	0	1	0	1	1	0	1	0
								Boot Identifier = 0x5A							

Table 11. Reset Configuration Half Word (RCHW) field descriptions

Field	Description
0–3	Reserved These bit values are ignored when the halfword is read. Program to 0 for future compatibility.
SWT	Software watchdog timer enable This bit determines if the software watchdog timer is enabled after passing control to the user application code. 0 Disable software watchdog timer 1 Enable software watchdog timer after reset. The timeout period is 261,600 system clocks.
WTE	MCU core watchdog timer enable This bit determines if the core software watchdog timer is enabled.after passing control to the user application code. 0 Disable core software watchdog timer 1 Enable core watchdog timer after reset. The timeout period is $2.5*2^{17}$ system clocks.

Table 11. Reset Configuration Half Word (RCHW) field descriptions

Field	Description
PS0	Port size Defines the width of the data bus connected to the memory on D_CS0. After system reset, the BAM changes D_CS0 to a 16-bit port to fetch the RCHW from either 16- or 32-bit external memories. Then the BAM reconfigures the EBI as a 16- or 32-bit port, depending on this bit. 0 32-bit D_CS0 port size 1 16-bit D_CS0 port size Used in development bus boot modes only (not available on all packages). Do not clear this bit if the device only has a 16-bit data bus.
VLE	VLE Code Indicator This bit configures the MMU entries 1–3 coded as Power Architecture embedded category instructions or as VLE instructions. 0 User code executes as classic Book E code 1 User code executes as VLE code
BOOTID	Boot identifier This field serves two functions: – Indicates which block in flash memory contains the boot program – Indicates if the flash memory is programmed (BOOTID=0x5A) or invalid

When enabled by RCHW[SWT, WTE] bits, the watchdog timeout periods are as shown in *Table 12*.

Note the following:

- The SWT clock source is directly from the crystal oscillator. The core WD is clocked by the PLL.
- The core WD timeouts reported here correspond to the PLL settings after reset. Core WD timeouts will change as soon as the PLL is programmed with different multipliers.

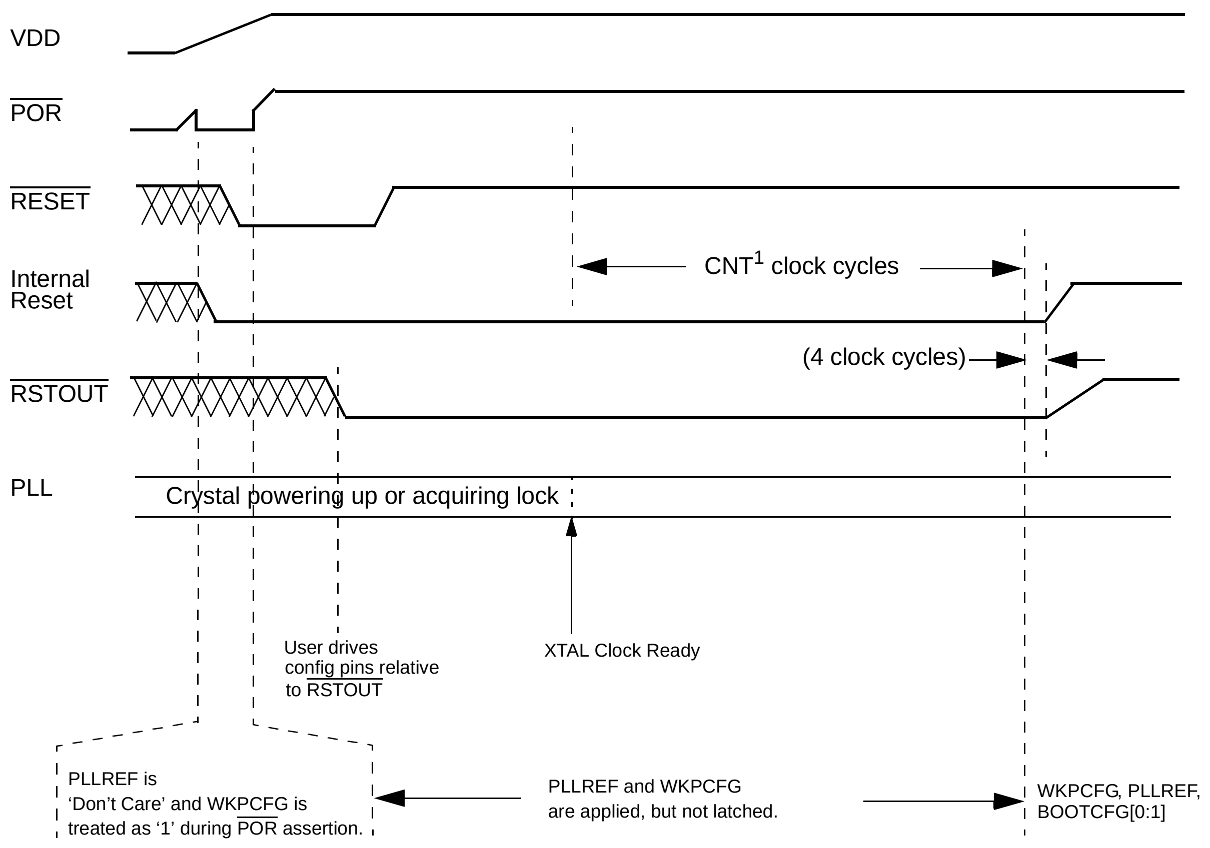
Table 12. Watchdog timeout periods

Crystal frequency (MHz)	Core WD timeout[1] (ms)	SWT timeout[2] (ms)
8	40.1	32.7
12	27.3	21.8
16	20.5	16.35
20	16.4	13.08
40	8.2	6.54

1. 327,680 system clocks
2. 261,600 system clocks

4.7.2 Reset configuration timing

The timing diagram in *Figure 5* shows the sampling of the BOOTCFG[0:1], WKPCFG, and PLLREF pin for a power-on reset. The timing diagram is also valid for internal/external resets assuming that VDD and VRC33 are within valid operating ranges. The value of the PLLREF pin is latched at the negation of the $\overline{\text{RSTOUT}}$ pin. The value of the WKPCFG signal is applied at the assertion of the internal reset signal (assertion of $\overline{\text{RSTOUT}}$). The values of the WKPCFG and BOOTCFG[0:1] pins are latched four clock cycles before the negation of the RSTOUT pin and stored in the SIU_RSR.



NOTE:
1. The clock count CNT depends on the reset source and type of clock reference. Please refer to *Table 9*.

Figure 5. Reset configuration timing

4.7.3 Reset weak pull up/down configuration

The signal on the WKPCFG pin determines whether specified eTPU and eMIOS pins are connected to weak pull up or weak pull down devices at reset (see the Signal Description chapter for the eTPU and eMIOS pins that are affected by WKPCFG). For all reset sources except the Software External Reset, the WKPCFG pin is applied at the assertion of the internal reset signal (assertion of RSTOUT). If the WKPCFG signal is logic high at this time, pull up devices will be enabled on the eTPU and eMIOS pins. If the WKPCFG signal is logic low at the assertion of the internal reset signal, pull down devices will be enabled on those pins. The value on WKPCFG must be held constant during reset to avoid oscillations on the eTPU and eMIOS pins caused by switching pull up/down states. The final value of WKPCFG is latched four clock cycles before the negation of RSTOUT. After reset, software may modify the weak pull up/down selection for all I/O pins through the PCR registers in the SIU.

5 Operating Modes and Clocking

5.1 Overview

This section gives a brief overview of the operating modes of this device.

5.2 Modes of operation

5.2.1 Normal mode

Normal Mode is the functional mode of this device.

5.2.2 Debug mode

Debug Mode provides access to powerful debugging and development features of this device. The debug and development features are distributed between Nexus blocks in the e200z4 core, eDMA and the eTPU, and some of the peripheral modules. The Nexus debug and development features are described in *Chapter 34: Nexus Port Controller (NPC)*.

The peripheral blocks that implement Debug Mode are:

- DSPI B, DSPI C, DSPI D
- FlexCAN A, FlexCAN B, FlexCAN C
- eMIOS
- eQADC
- eTPU (referenced as Halt State in *Chapter 22: Enhanced Time Processing Unit (eTPU2)*)

See the “Modes of Operation” section of the individual module for a description of how the Debug Mode affects the behavior of the module.

5.2.3 Low power modes

This device can be configured such that the clock to some or all of the modules can be stopped to reduce the power consumption. A tiered approach towards clock gating is implemented. In the first tier (Module Disable mode) some modules can be configured to stop the clock to the non-memory mapped registers within the module. In the second tier (Module Halt mode) the clock to each of the modules, including the CPU, can be completely stopped.

Module disable mode

Module Disable Mode is a low-power mode supported by some of the modules on this device, in which the clock to the non-memory mapped registers within the module is gated-off. *Table 14* lists the modules that support Module Disable Mode. The register and bit in each module that must be written to enter or exit this mode are also listed. See the “Modes of Operation” section of the individual module for a description of how the Module Disable Mode affects the behavior of the module.

Module halt mode

Module Halt mode is a low power mode in which the clock to all registers within each module can be completely halted. The control of the clock gating is centralized in the SIU_HLT register, which has one control bit for each module to be halted. The CPU itself can also be halted.

Standby mode

In this mode, the power is removed from all functions except the standby RAM. Standby mode is entered by removing all power supplies except the one on the VSTBY pin. The device is recovered from the standby mode when powered again; see *Chapter 4: Resets* for more information.

5.3 Clock architecture

The following sections detail the SPC564A70 clocking architecture.

5.3.1 Overview

This section describes different sources for the system clocks. The SPC564A70 clocking architecture consists of the following:

- On-chip MHz oscillator: Range (4–40 MHz)
- Relaxation oscillator (RCOSC): 16 MHz
- Phase-locked loop (PLL): VCO range (256–512 MHz)
- PLLREF top level pin to control PLL reference
- Clock quality monitor
- System Clock Divider (SYSDIV) used to further reduce the system clock frequency
- Register to control system clock source and programming of PLL parameter
- Clock gating for individual modules controlled by either SIU_HLT register or module’s MDIS register bit

5.3.2 Block diagram

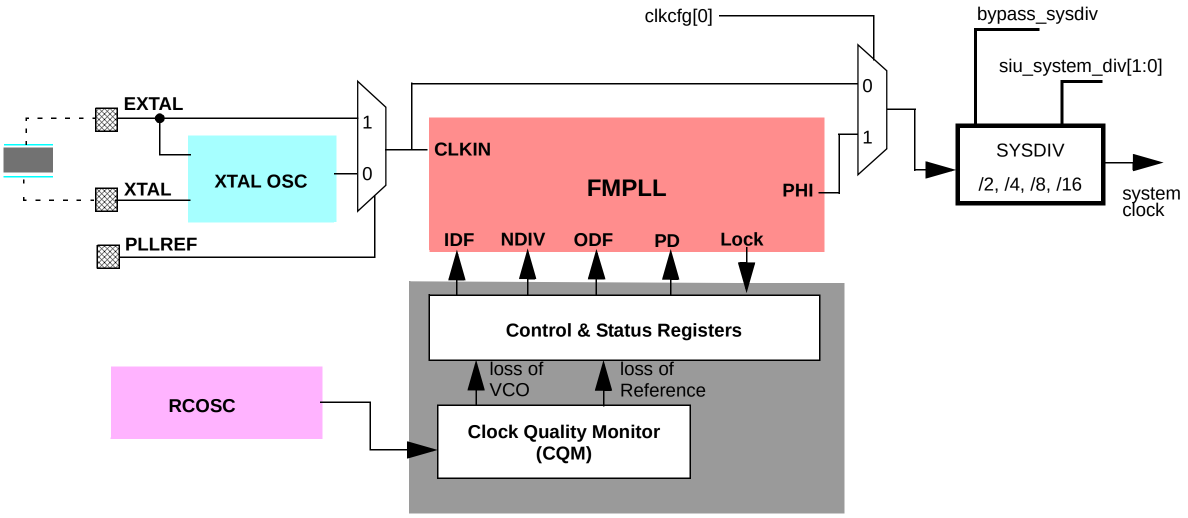


Figure 6. System clock diagram

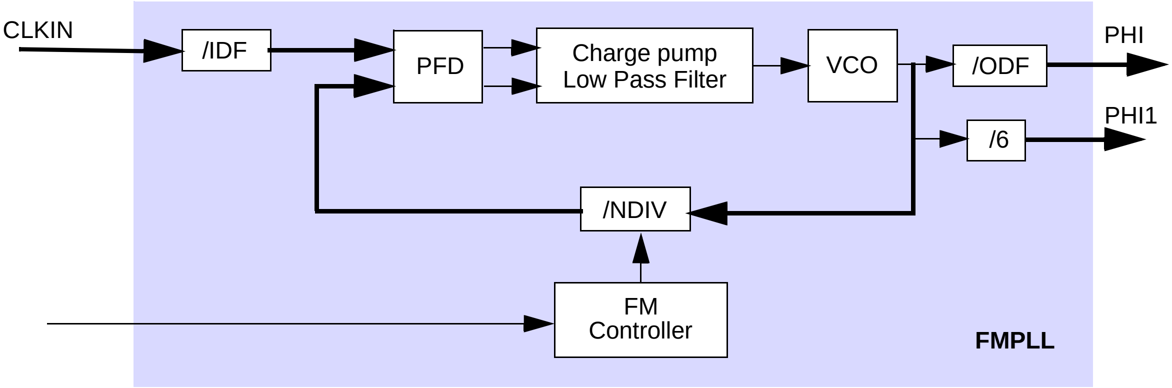


Figure 7. FMPLL

5.3.3 System clock sources

The on-chip MHz oscillator, PLL and the top level pin are the possible sources of system clock. The system clock can be generated using any of the following options:

- PLL disabled
  - MHz crystal oscillator with crystal as the reference
  - MHz crystal oscillator bypassed
- PLL enabled
  - MHz crystal oscillator (with crystal as the reference) output used as PLL reference frequency
  - MHz crystal oscillator (bypassed) output used as PLL reference frequency

Upon Reset, the system clock source is the oscillator clock with either crystal as reference or bypassed based on the PLLREF pin value driven during system reset.

Please note the following:

1. RCOSC is never used as a source of system clock.
2. PHI1 output from PLL is never used as a source of system clock. It is used as one of the clock sources for the FlexRay module.
3. See the FMPLL chapter for details on FMPLL operation.

Support for 150 MHz system clock generation

The oscillator and PLL support generation of a 150 MHz system clock while using the 40 MHz crystal required for FlexRay operation. A possible PLL configuration is shown below:

- Input clock (crystal frequency): 40 MHz
- EPREDIV/IDF divider = /8 (1–15 range supported)
- EMFD/NDIV loop divider = 60 (32–96 supported)
- VCO clock out = 300 MHz (256–512 MHz range supported)
- ERFD/ODF output divider = /2 (/2, /4, /8, /16 supported)
- SYSDIV divider = /1 (/1, /2, /4, /8, /16 supported)
- System clock = 150 MHz

Support for 100 MHz system clock generation

The oscillator and PLL support generation of a 100 MHz system clock while using the 40 MHz crystal required fro FlexRay operation. A possible PLL configuration is shown below:

- Input clock (crystal frequency): 40 MHz
- EPREDIV/IDF divider = /8 (1–15 range supported)
- EMFD/NDIV loop divider = 80 (32–96 supported)
- VCO clock out = 400 MHz (256–512 MHz range supported)
- ERFD/ODF output divider = /4 (/2, /4, /8, /16 supported)
- SYSDIV divider = /1 (/1, /2, /4, /8, /16 supported)
- System clock = 100 MHz

Support for FlexRay operation

The SPC564A70 MCU supports generation of the clock signals needed for the operation of the FlexRay module. Two options are supported for the generation of the FlexRay clock:

- If the PLL is used with Frequency Modulation enabled, a 40 MHz crystal or external clock source must be used to supply the FlexRay clock.
- If the PLL is configured to generate a 120 MHz system clock without Frequency Modulation, then the FlexRay module can be clocked from the system clock, allowing the use of other crystal frequencies. In this mode of operation, the VCO frequency would be 480 MHz (256–512 MHz VCO range supported) with the /4 output divider to achieve 120 MHz system clock. The VCO/6 (80 MHz) output from PLL(PHI1) would be selected as the clock source for FlexRay by configuration of the MCF[CLKSEL] control bit on the FlexRay module.

Support for CAN interface operation

The FlexCAN modules have two distinct software controlled clock domains. One of the clock domains is always derived from the system clock. This clock domain includes the message buffer logic.

The source for the second clock domain can be either the system clock or a direct feed from the crystal oscillator pin. The logic in the second clock domain controls the CAN interface pins. Field FlexCAN_CR[CLKSRC] selects between the system clock and the on-chip MHz oscillator clock as the clock source for the second domain. Selecting the oscillator as the clock source ensures very low jitter on the CAN bus.

Software can gate both clocks by writing to FlexCAN_MCR[MDIS] or by writing to the SIU_HLT register.

5.3.4 FMPLL modes of operation

Upon reset, the FMPLL operational mode is bypass with PLL running, and the source of the reference clock, either the crystal oscillator or external clock, is determined by the state of the CLKCFG[2] bit of the FMPLL_ESYNCR1 register. The reset state of this bit comes from an external signal to the module connected to a package pin called PLLREF. After reset, a different operational mode can be selected by writing to FMPLL_ESYNCR1[CLKCFG]. The available modes are specified in *Table 13*.

Table 13. Clock Mode Selection

CLKCFG[0] ($\overline{\text{Bypass}}$)	CLKCFG[1][1] (PLL enable)	CLKCFG[2][2] (Clock source)	Clock mode
0	0	0	Bypass mode with external reference and PLL off
0	0	1	Bypass mode with crystal reference and PLL off
0	1	0	Bypass mode with external reference and PLL running
0	1	1	Bypass mode with crystal reference and PLL running
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Normal mode with external reference
1	1	1	Normal mode with crystal reference

1. CLKCFG[1] is not writable to zero while CLKCFG[0]=1.
2. The reset state of this bit is determined by the logical state applied to the **PLLREF** pin.

The reset state of the FMPLL is enabled with the pre-divider set such that it inhibits the clock to the PLL Phase detector, making the VCO run within its free-running frequency range of 25 MHz to 125 MHz, unconnected from the system clock (since bypass is the default mode at reset). If using crystal reference, after power-on reset the Clock Quality Monitor (CQM) inhibits the system clock and keep system reset asserted while the crystal oscillator has not stabilized. The PLLREF pin must be kept stable during the whole period while system reset is asserted.

Bypass mode with crystal reference

In the bypass mode with crystal reference, the FMPLL is completely bypassed and the system clock is driven from the crystal oscillator. The user must supply a crystal that is within the appropriate frequency range, the crystal manufacturer recommended external support circuitry, and short signal route from the MCU to the crystal.

In bypass mode the PLL itself may or may not be running, depending on the state of the CLKCFG[1] bit of the FMPLL_ESYNCR1 register, but the PLL output is not connected to the system clock. Consequently, frequency modulation is not available. The pre-divider is also bypassed, but the system clock divider (SYSDIV) can be used to reduce the system clock frequency. The system clock divider can be programmed by writing to SIU_ SYSDIV[SYSDIV].

Bypass mode with crystal reference is the default mode at reset if the PLLREF pin is driven high. After reset, this mode can be entered by programming FMPLL_ESYNCR1[CLKCFG] as shown in *Table 13*.

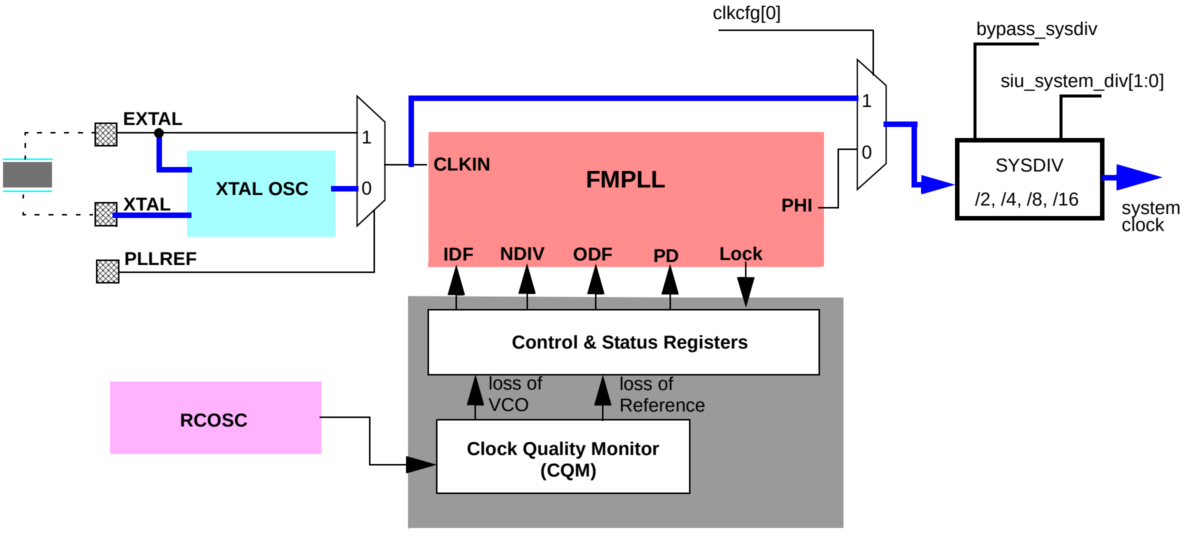


Figure 8. Bypass mode with crystal reference

Bypass mode with external reference

The bypass mode with external reference functions the same as bypass mode with crystal reference, except that the system clock is driven by an external clock generator connected to the EXTAL pin, rather than a crystal oscillator. The input frequency range is the same and frequency modulation is not available.

Bypass mode with external reference is the default mode at reset if the PLLREF pin is driven low. After reset, this mode can be entered by programming FMPLL_ESYNCR1[CLKCFG] as shown in *Table 13*.

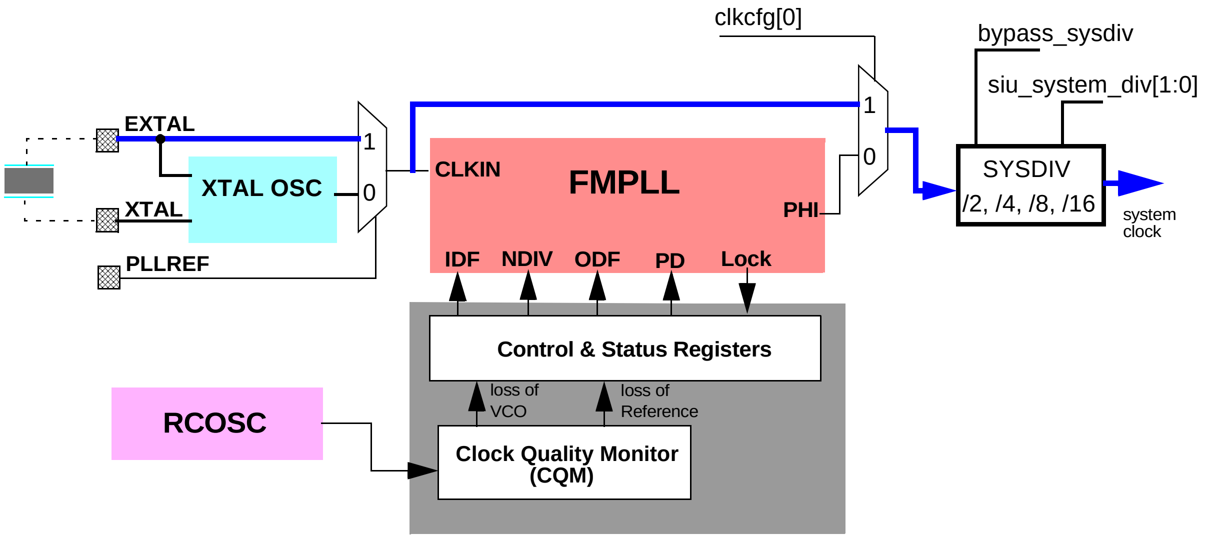


Figure 9. Bypass mode with external reference

Normal mode with crystal reference

In the normal mode with crystal reference, the FMPLL receives an input clock frequency from the crystal oscillator and the pre-divider, and multiplies the frequency to create the FMPLL output clock. The user must supply a crystal that is within the appropriate frequency range, the crystal manufacturer recommended external support circuitry, and short signal route from the MCU to the crystal.

In normal mode with crystal reference, the FMPLL can generate a frequency-modulated clock or a non-modulated clock (locked on a single frequency). The modulation rate, modulation depth, output divider (RFD) and whether the FMPLL is modulating or not can be programmed by writing to the FMPLL registers. The system clock divider (SYSDIV) can also be used to further reduce the system clock frequency in addition to the FMPLL output divider. The system clock divider can be programmed by writing to SIU_SYSDIV[SYSDIV]. See *Section 15.6.31, System Clock Register (SIU_SYSDIV)* for details.

Note: The PLL output frequency (before the system clock divider) must not exceed the maximum device operating frequency. Therefore, when operating at the maximum operating frequency, the only division factor allowed in the system clock divider is divide-by-1.

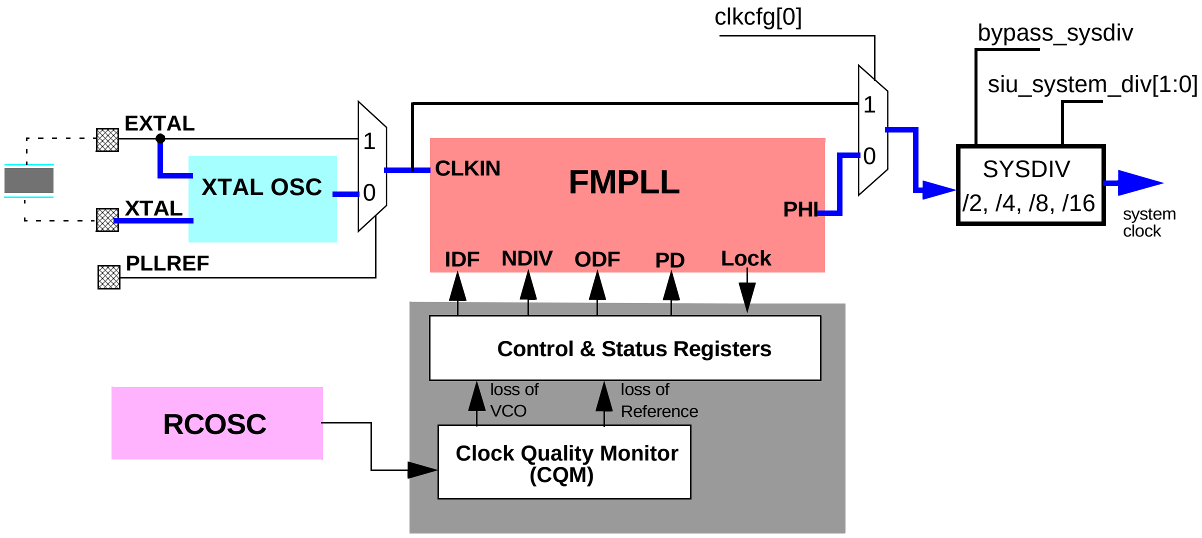


Figure 10. Normal Mode with Crystal Reference

Normal mode with external reference

The normal mode with external reference functions the same as normal mode with crystal reference, except that the input clock reference to the FMPLL is driven by an external clock generator connected to the EXTAL pin, rather than a crystal oscillator. The input frequency range is the same and frequency modulation is available.

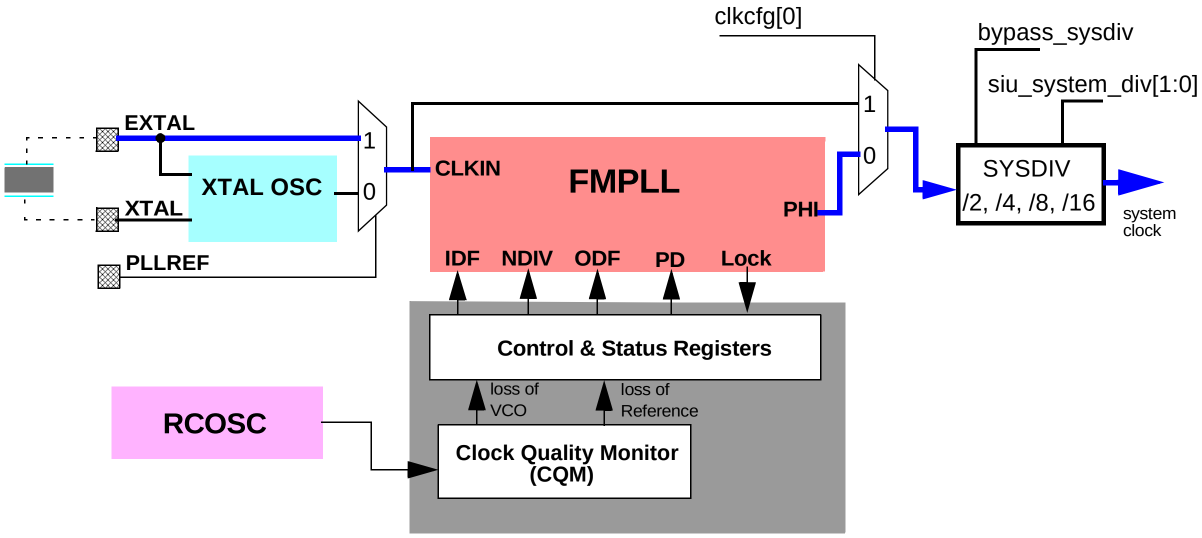


Figure 11. Normal Mode with External Reference

Software controlled power management

Software controlled power management and clock gating is supported on a peripheral by peripheral basis, using a three tiered approach. The first tier is a clock gating feature implemented in some of the IP modules, which allows software to disable the non-memory-mapped portions of the modules by setting the module disable (MDIS(c)) bits in the registers within the modules. The second tier is provided by the SIU_HLT register, which can be used to halt the clock of both memory-mapped and non-memory-mapped portions of each module. The third tier is provided by the WAIT instruction of the Power Architecture instruction set, which controls the clock gating of the CPU itself. *Figure 12* illustrates how the MDIS and halt bits affect the clocks to the modules.

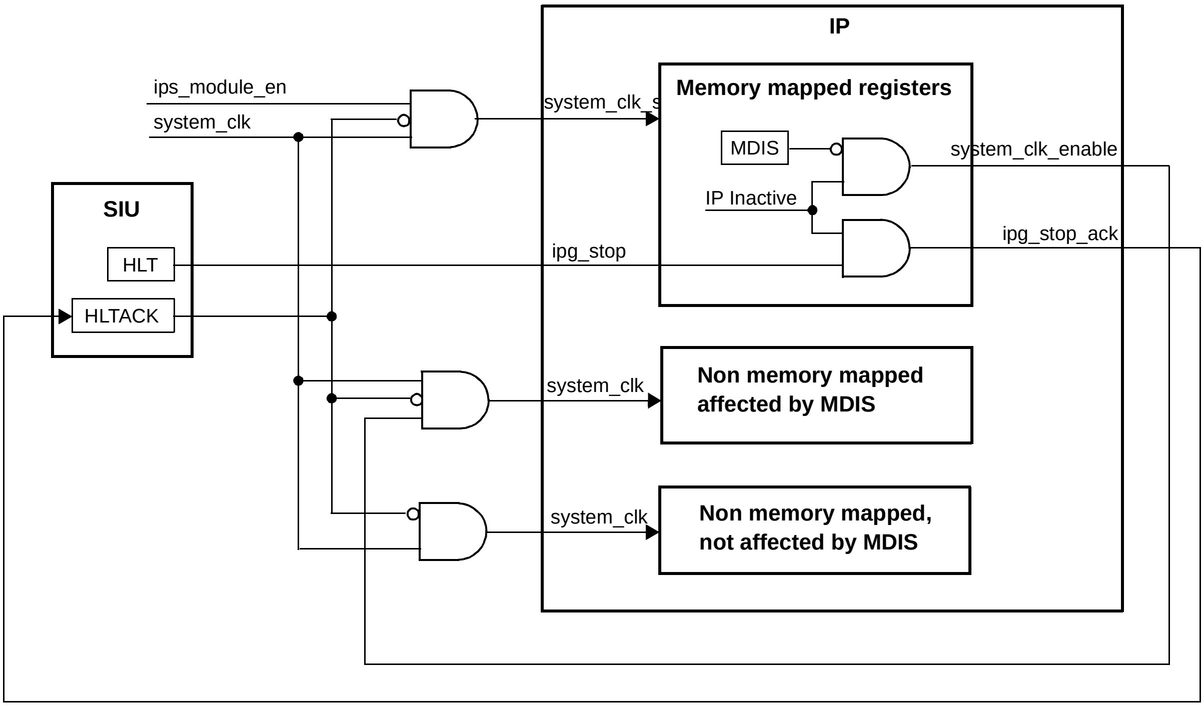


Figure 12. MDIS and halt clock gating

MDIS clock gating

The MDIS bit disables the clock to some or all of the non-memory-mapped registers of the module. The memory-mapped portion of the modules are clocked by the system clock when they are accessed.

When the NPC module is disabled by MDIS, the MCKO clock is disabled. Furthermore, the NPC can be configured to disable the MCKO clock when there are no messages pending.

The flash memory array can be disabled by writing to a bit in the flash memory map.

The modules that implement the MDIS function are listed in *Table 14*, along with the registers and bits that disable each module. The software controlled clocks are enabled when the CPU comes out of reset.

---

c. For compatibility with legacy devices, the default value of MDIS bit is zero.

Table 14. MDIS support[1]

Block name	Register name	Bit name
DSPI_B	DSPI_B_MCR	MDIS
DSPI_C	DSPI_C_MCR	MDIS
DSPI_D	DSPI_D_MCR	MDIS
eTPU	ETPUECR_1	MDIS
FlexCAN A	FLEXCAN_A_MCR	MDIS
FlexCAN B	FLEXCAN_B_MCR	MDIS
FlexCAN C	FLEXCAN_C_MCR	MDIS
eMIOS	EMIOS_MCR	MDIS
eSCI_A	eSCI_A_SCICR3	MDIS
eSCI_B	eSCI_B_SCICR3	MDIS
eSCI_C	eSCI_C_SCICR3	MDIS
Decimation Filter	DECFILTERMCR	MDIS
Red Line Module	TBD	MDIS
NPC	NPC_PCR	MCKO_EN, MCKO_GT
Flash Memory Array	FLASH_MCR	STOP

1. The MDIS bit default reset value is zero.

Halt clock gating

Software controlled clock gating can be done via the centralized halt mechanism. The SIU_HLT register bits corresponding to individual modules are configured to determine which modules are clock gated.

The SIU_HLT register bits are used to drive a stop request signal to the modules. After the module completes a clean shut down, the module asserts a stop acknowledge handshake signal that is used to gate the clock to the module (see *Figure 12*). The stop acknowledge signals are also captured in the SIU_HLTACK read-only register bits.

The halted modules recover when the corresponding SIU_HLT register bit is cleared by software. Once the bit is cleared, logic will re-enable the clocks to the modules and then negate the stop request signal after the required timing has been met.

CPU clock gating

The SIU_HLT register has a bit to halt the clock to the CPU, but in order to prevent accidental CPU halting, a stop request is only activated if the CPU is in wait state due to the execution of the WAIT instruction.

Note: To gate the CPU clock you need to first program the SIU_HLT register bit assigned for CPU and then execute the CPU WAIT instruction.

The CPU recovers from the halted state when one of the following events happens:

- A valid pending interrupt is detected by the core
- A request to enter debug mode is made by setting the DR bit in the OnCE control register (OCR)
- The processor is in a debug session
- A request to enable the CPU clock input has been made by setting the WKUP bit in the OCR

When one of these events is detected, the CPU asserts an asynchronous output signal that re-enables the clock to the CPU so that it can exit the stopped state. Typically, the wake-up interrupt request will come from one of three sources: periodic interval timer (PIT) interrupt, external pin interrupt or CAN wake-up interrupt.

When the clock to the CPU is gated, the clocks to the platform, the system RAM and the flash memory are also gated. The platform logic includes the cross-bar, peripheral bridge, DMA and flash memory controller. Note that the interrupt controller (INTC) and the SIU are not clock gated to allow interrupts to be used to recover the CPU halt state.

Clock dividers

The MCU provides five clock dividers:

- System Clock Divider (SYSDIV)
- Calibration Bus Clock Divider (CLKOUT-DIV)
- Nexus Message Clock Divider (MCKO-DIV)
- Engineering Clock Divider (ENGDIV)
- FlexCAN clock divider (CAN2:1)

System Clock Divider (SYSDIV)

The system clock divider is placed right at the output of the system clock mux (selection between FMPLL and the crystal clock) and before the clock is used by any other circuits, including the other clock dividers. It affects the clock in both normal mode and bypass mode. The system clock divider can be programmed to divide by 1, 2, 4, 8, or 16 depending on the values of fields BYPASS and SYSCLKDIV in the SIU_SYSDIV register.

SIU_SYSDIV[BYPASS] determines whether or not the system clock divider is bypassed. The SIU_SYSDIV[BYPASS] reset value ‘1’ causes the system clock divider to be bypassed and results in a divide-by-1 reset configuration of the system clock divider. Only if the SIU_SYSDIV[BYPASS] value is ‘0’ can field SIU_SYSDIV[SYSCLKDIV] be programmed to divide by 2, 4, 8, or 16.

External Bus Clock (CLKOUT)

The calibration bus clock (CLKOUT) divider can be programmed to divide the system clock by one, two or four based on the settings of the EBDF field in the SIU external clock control register (SIU_ECCR). The reset value of SIU_ECCR[EBDF] selects a CLKOUT frequency of one half of the system clock frequency. The calibration bus supports gating of the CLKOUT signal when there are no external bus accesses in progress.

The hold time for external bus pins can be changed by writing to SIU_ECCR[EBTS] (external bus tap select bit).

Note: *The CLKOUT pin is only available in the 324-pin package.*

Nexus Message Clock (MCKO)

The Nexus message clock (MCKO) divider can be programmed to divide the system clock by two, four or eight based on the MCKO_DIV field in the port configuration register (PCR) in the Nexus port controller (NPC). The reset value of NPC_PCR[MCKO_DIV] selects an MCKO clock frequency one half of the system clock frequency. The MCKO divider is configured by writing to the NPC through the JTAG port.

Engineering Clock Divider (ENGDIV)

The engineering clock divider (ENGDIV) can be programmed to divide system clock. This clock is mainly used to clock some ASIC devices integrated on the board. There is no timing relation of this clock with respect to any other clock in the design.

Refer to *Section 15.6.26: External Clock Control Register (SIU_ECCR)* for ENGDIV register bit programming.

FlexCAN Clock Divider (CAN2:1)

The FlexCAN module has the ability to run from the system clock. It is possible, at the input to the FlexCAN module, to perform a divide-by-1 or a divide-by-2 division of the system clock.

This FlexCAN system clock divider can be programmed by configuring the FlexCAN2:1 mode bit (field SIU_SYSDIV[CAN_SRC]). The reset value is to divide-by-1.

The FlexCAN module does not support a divide-by-1 of the system clock above a certain frequency, defined in the devicedatasheet. When running at maximum system frequency this setting will have to be adjusted from its default value.

6 Device Performance Optimization

6.1 Introduction

The SPC564A70 contains several features that can influence the overall level of performance provided by the device.

Some of these features may be initialized upon negation of reset either by a software program called the Boot Assist Module (BAM), by a hardware state machine or by appropriate default register settings. Although the device exits the reset state into a functional state it does not necessarily have the default optimum performance settings for any given application.

This chapter provides guidance for users to fully optimize their application to achieve the highest possible performance from the SPC564A70. It provides a description of the areas that should be focused on when optimizing an application for performance by describing the features and recommending settings to be applied. It focuses on hardware configurations although certain aspects of the application software such as compiler settings and optimizations will be discussed.

6.2 Features

The SPC564A70 has the following hardware features that can be configured to impact the overall performance of the device:

- Branch Prediction
  - Branch Target Buffer
  - Branch Prediction Control
- Frequency-modulated PLL
- Flash Bus Interface Unit
  - Flash access wait state and address pipelining control
  - Flash instruction prefetching
  - Flash data prefetching
- Crossbar switch
- System Cache
  - Instruction Cache
- Memory Management Unit

Further application level features can impact the application performance:

- Hardware Single Precision Floating point
- Signal Processing Extension (SPE-APU)
- Variable Length Encoding (VLE)
- Compiler optimizations

Further factors that impact the overall application performance are the use of the intelligent peripherals:

- Use of DMA rather than CPU to transfer data efficiently
- Use of DMA service requests rather than CPU interrupts to avoid software polling
- Off-loading tasks from the CPU to the eTPU2 or eDMA
- Careful allocation of cache usage for code and data ranges, particularly when using with external memories.

Different items in this list will have different performance impacts in a real system. Features like the system cache, the FMPLL and the flash access times tend to provide the most significant performance impacts in terms of hardware settings.

The subsequent sections in this chapter describe how to configure and use these features.

6.3 Configuring hardware features

6.3.1 Branch target buffer (BTB)

Description

To resolve branch instructions and improve the accuracy of branch predictions the e200z4 core implements a dynamic branch prediction mechanism using a branch target buffer (BTB), a fully associative address cache of branch target addresses. Its purpose is to accelerate the execution of software loops with some potential change of flow within the loop body. In addition, the BTB on the e200z4 has a subroutine call stack that speeds up indirect branches.

Recommended configuration

By default, this BTB is disabled following negation of reset. It is controlled by the Branch Unit Control and Status Register (BUCSR). The BTB’s contents should be flushed and invalidated by writing BUCSR[BBFI] = 1, and it may be enabled by subsequently writing BUCSR[BPEN] = 1.

Additional control is available in BUCSR[BPRED] and BUCSR[BALLOC] to control whether forward or backward branches (or both) are candidates for entry into the BTB, and thus for branch prediction. By default the BUCSR[BPRED] and BUCSR[BALLOC] fields are set to 0b00, which enables forward and backward branch prediction. It is recommended to not disable branch prediction although for extremely fine tuning of a given application the optimum setting of BUCSR[BPRED] and BUCSR[BALLOC] should be assessed.

Figure 13. Branch Unit Control and Status Register (BUCSR)

0	BBFI	0	BALLOC	0	BPRED	BPEN
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21	22	23 24 25	26 27	28	29 30	31

SPR - 1013; Read/Write; Reset - 0x0

Table 15. BUCSR field descriptions

Field	Description
BBFI	Branch target buffer flash invalidate When set, BBFI flash clears the valid bit of all BTB entries; clearing occurs regardless of the value of the enable bit (BPEN). **Note**: BBFI is always read as 0.
BALLOC	Branch Target Buffer Allocation Control 00: Branch Target Buffer allocation for all branches is enabled. 01: Branch Target Buffer allocation is disabled for backward branches. 10: Branch Target Buffer allocation is disabled for forward branches. 11: Branch Target Buffer allocation is disabled for both branch directions. This field controls BTB allocation for branch acceleration when BPEN = 1. Note that BTB hits are not affected by the settings of this field. Note that for branches with AA = ‘1’, the MSB of the displacement field is still used to indicate forward/backward, even though the branch is absolute.
BPRED	Branch Prediction Control (Static) 00: Branch predicted taken on BTB miss for all branches. 01: Branch predicted taken on BTB miss only for forward branches. 10: Branch predicted taken on BTB miss only for backward branches. 11: Branch predicted not taken on BTB miss for both branch directions. This field controls operation of static prediction mechanism on a BTB miss. Unless disabled, fetching of the predicted target location will be performed for branch acceleration. BPRED operates independently of BPEN, and with a BPEN setting of 0, will be used to perform static prediction of all unresolved branches. Note that BTB hits are not affected by the settings of this field. Note that for certain applications, setting BPRED to a non-default value may result in improved performance.
BPEN	Branch target buffer (BTB) enable 0: BTB prediction disabled. No hits are generated from the BTB and no new entries are allocated. Entries are not automatically invalidated when BPEN is cleared; BBFI controls entry invalidation. 1: BTB prediction enabled (enables BTB to predict branches).

Further details of the BUCSR can be found in the e200z4 Power Architecture® Core Reference Manual.

6.3.2 Frequency-modulated PLL

Description

The frequency-modulated phase-locked loop (FMPLL) allows the user to generate high speed system clocks from a crystal oscillator or external clock generator. Further, the FMPLL supports programmable frequency modulation of the system clock. This module is typically configured early in the initialization code to ensure satisfactory performance levels are achieved.

Recommended configuration

The default operating frequency of the SPC564A70 device is 2 to 3 times the crystal reference frequency depending on the state of the PLL configuration pins as reset negates. Typically, the system frequency is increased shortly after reset negates to provide acceptable performance. *Chapter 16: Frequency-modulated phase locked loop (FMPLL)*, provides details on how the frequency-modulated phase-locked loop should be initialized in an application. The maximum frequency of operation for this device is specified in the device data sheet.

System performance cannot be linearly extrapolated with system frequency, as is often the expectation. It is due to the insertion of additional Flash wait states as system frequency increases that system performance does not scale linearly. Take care to ensure that the correct internal and/or external Flash configuration is chosen for the selected system frequency. The specific flash access times to be applied are detailed in *Section , Bus Interface Unit Configuration Register (BIUCR).*

6.3.3 Flash bus interface unit

Description

The Flash Bus Interface Unit (FBIU) interfaces the system bus to the Flash memory array controller. The FBIU contains prefetch buffers and a prefetch controller which, if enabled, speculatively prefetches sequential lines of data from the Flash array into the buffer. Prefetch buffer hits allow zero-wait state responses.

The Flash Bus Interface Configuration Registers (BIUCRx) control access to the internal Flash array. Its settings define the number of cycles required to access the array, access times, and how the prefetch buffering scheme operates.

Following negation of reset and execution of the BAM, the instruction and data prefetching is disabled, and the number of cycles required to access the internal Flash array is set to its maximum value of fifteen additional wait states.

Recommended configuration

As the operating frequency of the device is set by configuring the FMPLL (see *Section 6.3.2, Frequency-modulated PLL*), the number of cycles required to access the internal array should be configured accordingly. Note that the Flash BIUCRx registers cannot be altered by code executing from the Flash array. Code for configuring the Flash should be executed from a separate memory array i.e copied to and executed from system RAM.

Section , Bus Interface Unit Configuration Register (BIUCR), documents the register fields used to configure flash wait state settings. The “*Platform flash controller electrical characteristics”* section of the device data sheet contains the specific values for the flash wait state settings for a given operating frequency. This also provides recommendations for the prefetch buffer settings. Note that the BIUCRx settings may vary between revisions of the SPC564A70.

6.3.4 Crossbar switch

Description

The multi-port crossbar switch (XBAR) supports simultaneous connections between master ports and slave ports. The XBAR allows for concurrent transactions to occur from any master port to any slave port. If a slave port is simultaneously requested by more than one master port, arbitration logic selects the higher priority master and grants it ownership of the slave port. All other masters requesting that slave port are stalled until the higher priority master completes its transactions. By default, requesting masters are granted access based on a fixed priority. A round-robin priority mode also is available.

The main goal of the XBAR is to increase overall system performance by allowing multiple masters to communicate concurrently with multiple slaves. In order to maximize data throughput it is essential to keep arbitration delays to a minimum. The configuration of the crossbar can have implications for the performance of a system and particular care should be taken when assigning master priorities in a fixed priority application. Further, by correctly parking saves on relevant masters the initial access times to the slaves can be minimized by negating any initial arbitration penalties.

Recommended configuration

The specific settings for a given situation are application dependent and thus should be assessed by the user. However, some general guidelines are available.

Optimal XBAR settings are application dependent, but in e200z4/7 (Harvard configuration) based devices assigning the CPU data bus to have highest priority and parking the slave port associated with system RAM on this master generally provides the best overall performance.

To reconfigure the XBAR as described on the SPC564A70, write the following registers:

1. XBAR_SGPCR2 = 0x0000_0001. This parks slave 2 (internal SRAM) on master port 1 (CPU data bus).
2. Write XBAR_MPR0 = 0x5432_0001. This sets slave port 0 (Flash) to give the master port 1 (CPU data bus) highest priority.

On the e200z4 based devices it may also be beneficial to assign the eDMA to have highest priority for the Flash slave port depending upon the application.

More details of the XBAR register configuration can be found in *Section 9.2, XBAR registers*.

6.3.5 Cache

Description

The SPC564A70 provides an 8 KB Instruction, 2-way or 4-way set-associative, Harvard cache design with a 32-byte line size. The cache is disabled by default when reset is negated.

The cache improves system performance by providing low-latency instructions to the e200z4 instruction pipelines, which decouples processor performance from system memory performance. There are several stages to enabling the cache. Not only does the cache itself have to be invalidated then enabled, but memory regions upon which it can operate must be configured in the MMU to permit cache access.

Recommended configuration

The exact usage of cache is application dependent but some general guidelines for using cache to improve performance in a typical application are listed below:

- Enable instruction cache for all internal and external memories that code is being executed from.
- Consider locking critical performance routines in cache.

The process of enabling the instruction cache involves first invalidating the cache (by setting L1CSR1[ICINV]) then when invalidation is completed (L1CSR1[ICINV, ICABT] = 0) enabling the cache (by setting L1CSR1[ICE]).

The L1CSR1 special purpose register is detailed below.For further details of cache configuration registers, refer to the e200z4 Power Architecture® Core Reference Manual.

Figure 14. L1 Cache Control and Status Register 1 (L1CSR1)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0															ICECE	ICEI	0	ICEDT		0	ICUL	ICLO	ICLFC	ICLOA	ICEA		ICORG		ICABT	ICINV	ICE

SPR - 1011; Read/Write; Reset - 0x0

Table 16. L1CSR1 field descriptions

Field	Description
ICECE	Instruction Cache Error Checking Enable
ICEI	Instruction Cache Error Injection Enable
ICEDT	Instruction Cache Error Detection Type
ICUL	Instruction Cache Unable to Lock
ICLO	Instruction Cache Lock Overflow
ICLFC	Instruction Cache Lock Bits Flash Clear
ICLOA	Instruction Cache Lock Overflow Allocate
ICEA	Instruction Cache Error Action
ICORG	Cache Organization 0 The cache is organized as 64 sets and 2 ways 1 The cache is organized as 32 sets and 4 ways
ICABT	Instruction Cache Operation Aborted Indicates a Cache Invalidate or a Cache Lock Bits Flash Clear operation was aborted prior to completion. This bit is set by hardware on an aborted condition, and will remain set until cleared by software writing 0 to this bit location.
ICINV	Instruction Cache Invalidate 0: No cache invalidate 1: Cache invalidation operation When written to a ‘1’, a cache invalidation operation is initiated by hardware. Once complete, this bit is reset to ‘0’. Writing a ‘1’ while an invalidation operation is in progress will result in an undefined operation. Writing a ‘0’ to this bit while an invalidation operation is in progress will be ignored. Cache invalidation operations require approximately 36 cycles to complete. Invalidation occurs regardless of the enable (ICE) value. During cache invalidations, the parity check bits are written with a value dependent on the ICEDT selection. ICEDT should be written with the desired value for subsequent cache operation when ICINV is set to ‘1’ for proper operation of the cache.
ICE	Instruction Cache Enable 0: Cache is disabled 1: Cache is enabled When disabled, cache lookups are not performed for instruction accesses. Other L1CSR0 cache control operations are still available.

Note that configuration of the cache has to be performed in conjunction with configuration of the Memory Management unit. Refer to section *6.3.6, Memory management unit (MMU)*.

6.3.6 Memory management unit (MMU)

Description

The Memory Management Unit is a 32-bit Power Architecture compliant implementation which provides functionality that includes address translation and application of access attributes to memory ranges defined in Translation Lookaside Buffer entries. Although the MMU does not directly impact performance, it is within the MMU that memory regions are configured to permit the use of system cache to improve performance and Variable Length Encoding (VLE) to enhance code density. Thus it is essential that the MMU is correctly configured to ensure optimal application performance is achieved.

Recommended configuration

The core uses MMU Assist Registers (MASx) which are special purpose registers to facilitate reading, writing and searching the Translation Lookaside Buffer (TLB) entries. These MAS registers are software managed by **tlbre**, **tlbwe**, **tlbsx**, **tlbsync**, and **tlbivax** instructions. Refer to the core reference manual for full details of the MMU and its configurations.

There are several MMU Assist Register registers (MAS0–3) that require configuring. Details of these are provided in the e200z4 Power Architecture® Core Reference Manual. Specifically, the MAS2 register contains the fields to control whether a specified memory region described by the valid TLB Entry is cache inhibited or whether VLE encoding is valid.

Figure 15. MMU Assist Register 2 (MAS2)

<table>
<tr><td colspan="22">EPN</td><td colspan="4">0</td><td>VLE</td><td>W</td><td>I</td><td>M</td><td>G</td><td>E</td></tr>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td><td>25</td><td>26</td><td>27</td><td>28</td><td>29</td><td>30</td><td>31</td></tr>
</table>

SPR - 626; Read/Write

Table 17. MAS2 field descriptions

Field	Description
EPN	Effective page number [0:21]
VLE	VLE 0: This page is a standard BookE page 1: This page is a VLE page
W	Write-through required
I	Cache inhibited 0: This page is considered cacheable 1: This page is considered cache-inhibited
M	Memory coherence required
G	Guarded
E	Endianness

Refer to the e200z4 Power Architecture® Core Reference Manual for further details of MMU configuration registers.

6.4 Application software

6.4.1 Compiler optimizations

The most significant opportunity for influencing the performance of a given application is by compiler and linker optimizations. Optimizing is a trade off between code size and performance. Typically higher performance of the application comes at the expense of larger code size. Compilers use a host of features, such as loop unrolling, function inlining, and application profile feedback to make the desired trade-offs between enhanced performance and minimized code size.

The data in *Figure 16* shows the effects of compiler optimization on a simple application. In this case, the Dhrystone benchmark was run under three conditions:

- Optimized for small code size
- Optimized for high performance
- A trade-off between code size and performance

Although this is an extreme example, it highlights how significant the role of the compiler and linker is in determining the overall performance of an application.

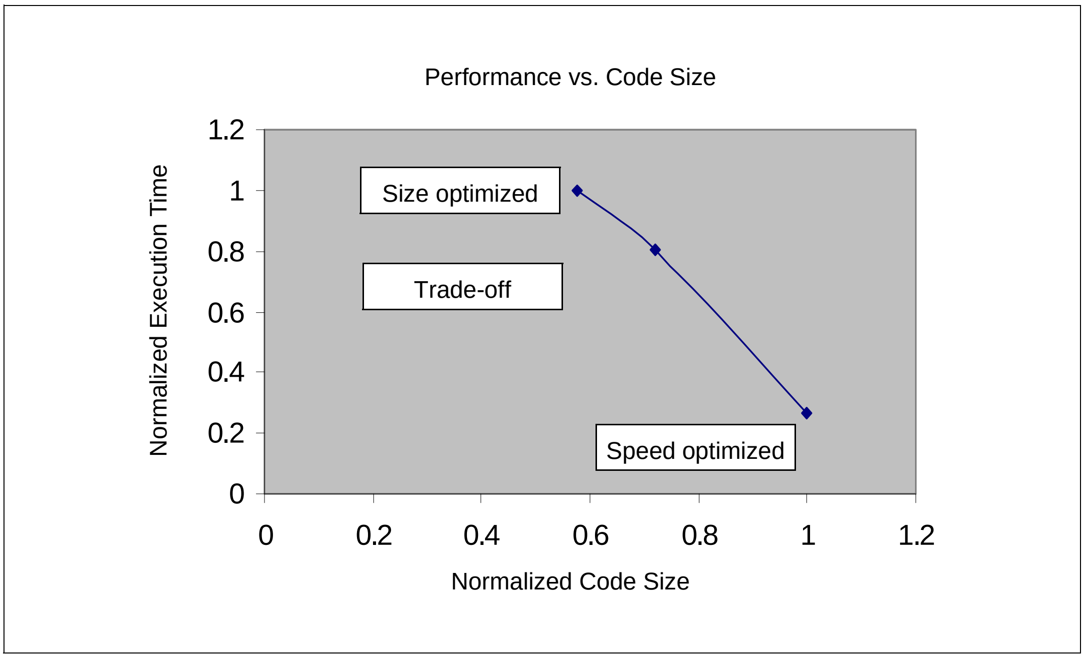


Figure 16. Influence of compiler settings on application performance and code size

Note: Data measured using Dhrystone version 2.1 run on a Power Architecture based powertrain device that uses a standard commercial compiler.

The compiler optimizations do not necessarily have to be applied to the entire application. Analysis of an application can identify time critical functions that may subsequently be targeted for performance optimization, without incurring the impact of optimizing the entire application.

There are several other aspects of the compiler and linker that should be considered. In particular, the use of Small Data Areas (SDAs, sometimes referred to as Special Data Areas) can make a significant performance improvement. Refer to compiler documentation for usage guidelines on Small Data Areas.

6.4.2 Signal processing extension

To further optimize time critical functions, the Signal Processing Extension Auxiliary Processing Unit (SPE-APU) may be used. The SPE-APU provides a set of Single Instruction Multiple Data (SIMD) instructions. These SIMD instructions typically involve performing the same operation on multiple data elements stored within a single 64-bit register. Through the implementation of SIMD instructions, including vector multiply and accumulate (MAC) instructions, the SPE APU provides Digital Signal Processing (DSP) functionality. This can be used to accelerate signal processing routines, such as Finite Impulse Response (FIR), Infinite Impulse Response (IIR) and Discrete Fourier Transforms (DFT). A more general benefit of the SPE instruction set is the ability to load/store 64-bits of data in single instruction. Thus highly load/store intensive functions make good candidates for SPE optimization.

Figure 17. Machine State Register (MSR)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0					UCLE	SPE	0						WE	CE	0	EE	PR	FP	ME	0		DE	0			IS	DS	0		RI	0

Read / Write; Reset - 0x0

Table 18. MSR field descriptions

Field	Description
UCLE	User Cache Lock Enable
SPE	SPE Available 0: Execution of SPE APU vector instructions is disabled; SPE Unavailable exception taken instead, and SPE bit is set in ESR. 1: Execution of SPE APU vector instructions is enabled.
WE	Wait State (Power management) enable
CE	Critical Interrupt Enable
EE	External Interrupt Enable
PR	Problem State
FP	Floating-Point Available
ME	Machine Check Enable
DE	Debug Interrupt Enable
IS	Instruction Address Space
DS	Data Address Space
RI	Recoverable Interrupt

6.4.3 Hardware single precision floating point

The SPE-APU also supports 32-bit IEEE®-754 single-precision floating-point formats, and supports vector and scalar single-precision floating-point operations. Most compiler vendors include libraries that can emulate floating point functionality. However, by specifying the correct compiler options, the single precision floating point instructions may be used.

To enable use of hardware floating point the MSR[SPE] field must be set. Refer to *Section 6.4.2, Signal processing extension* for register details.

6.4.4 Variable length encoding

In addition to the base Power Architecture instruction set support, the e200z4 core also implements the VLE (variable-length encoding) APU, providing improved code density. The VLE-APU can be viewed as a supplement to the existing Power Architecture instruction set that can be conditionally applied to a portion of, or an entire application for which improved code density is desired.

Using it is straightforward:

1. Select the appropriate compiler target and option to generate VLE code.
2. Configure the Memory Management Unit (MMU) to specify VLE attributes for the relevant MMU pages. Refer to the register description in *Section 6.3.6, Memory management unit (MMU)*.

VLE-enabled cores run both Power Architecture and VLE instruction encodings on a page by page basis, with pages defined by the MMU. The reduction is code size is typically between 25% and 30%.

6.5 Peripherals and general application guidelines

Optimizing the device configuration and compiler setup is only one part of optimizing an entire application. Correct use of the peripherals can also dramatically improve overall system performance. In particular, use of the interrupt controller, the enhanced Direct Memory Access (eDMA), and intelligent peripherals such as the Enhanced Timer Processing unit (eTPU2), can off-load significant work from the CPU.

For example, eDMA may be used to shift data to avoid unnecessary CPU loading. Most peripheral modules can generate eDMA requests to trigger data transfers. An example of a typical application is to use the eDMA to pass conversion commands to the analog to digital converter (ADC) ,while maintaining circular buffers results of the ADC in the system RAM, with no core intervention.

Section 6.6, Performance optimization checklist provides several system level examples of how to optimize an application.

6.6 Performance optimization checklist

Table 19. Performance optimization checklist—Part 1. Hardware configuration

Description	Register(s)	Details
Branch Target Buffer	Flush with BUCSR[BBFI] Enable with BUCSR[BPEN]	Flush and enable to improve accuracy of branch predictions.
Branch Prediction	BUCSR[BPRED] BUCSR[BALLOC]	Consider fine tuning of BTB operation for specific applications.
System Frequency	FMPLL_ESYNCR1 FMPLL_ESYNCR2	Select desired frequency taking into account the performance impact of additional wait states.
Flash Wait States	BIUCR[APC, WW, RWSC]	Refer to Flash chapter *Section , Bus Interface Unit Configuration Register (BIUCR)*, for BIUCR settings for FMPLL frequency ranges.
Flash Prefetching	BIUCR[DPFEN, IPFEN, PFLIM, BFEN]	Enable prefetching for instructions. Prefetching for data should be assessed for the specific application.
Flash Prefetch Algorithm	BIUCR2[LBCFG]	Allocate buffers to data and/or instructions. Fine tune for specific applications.
Crossbar Switch	Park slave SRAM on master port with XBAR_SGPCR2. Set Flash slave port to highest priority with XBAR_MPR0.	For e200z7 based devices reconfigure to optimize for Harvard architecture.
Cache	Invalidate Icache with L1CSR1[ICINV] Enable Icache with L1CSR1[ICE]	Invalidate and the enable the cache for instructions.
Memory Management Unit	TLB_MAS2[VLE, I]	Configure relevant pages for cache and VLE by setting MMU TLB attributes.

Table 20. Performance optimization checklist—Part 2. Software configuration

Description	Registers	Details
Compiler optimization	—	Use the features of the compiler to select the optimum trade off between code size and performance improvements.
Hardware Single Precision Floating Point	Enable with MSR[SPE]	Set compiler switches to specify using hardware single precision floating point as opposed to software emulation.
Signal Processing Extensions	Enable with MSR[SPE]	Take advantage of the SPE-APU to encode time critical functions using SPE assembly code.
Variable Length Encoding	Enabled with TLB_MAS2[VLE]	Set compiler switches and configure the MMU to take advantage of the VLE-APU.

Table 21. Performance optimization checklist—Part 3. Peripherals and general application guidelines

Peripherals and general application guidelines
Use eDMA rather than the core to move data where possible. Most peripherals can generate eDMA requests to shift data. – Use eDMA to control movement of commands and results from ADC and to maintain circular buffers in system memory. – Create circular buffers so that ADC results can be stored in RAM with no core overhead.
Shift loading from the CPU to the eTPU2 whenever possible. – The eTPU2 can provide effective CPU off-loading for time and angle based operations. – The eTPU2 can trigger the ADC directly with no need for CPU interruption.
Avoid software polling and allow peripherals to trigger interrupts or request eDMA servicing. – Use hardware instead of software vectored interrupts to reduce latency. – Trigger eDMA requests rather than interrupting the CPU to move data/results.
Configure the external memory interface. – Enable bursting on the external bus. – Reduce external bus wait states from default maximum settings. – Place time critical functions in internal memory. – Small, but frequently executed routines can be considered as candidates to be locked in cache.

7 e200z4 Core

This chapter contains an overview of the e200z4 processor core integrated in SPC564A70 devices.For detailed information see the publication e200z4 Power Architecture core reference manual.

Note: There are two differences between the processor core in SPC564A70 devices and the e200z4 documented in the core reference manual. SPC564A70 devices feature a e200z448n3 core with 8 KB of instruction cache (vs. 4 KB) and 24 MMU entries (vs. 16).

7.1 Overview

The microcontroller’s cost-efficient e200z4 host processor core is built on the Power Architecture technology and designed specifically for embedded applications.

The e200z4 is a dual-issue, 32-bit Power Architecture compliant design with 64-bit general purpose registers (GPRs). Power Architecture floating-point instructions are not supported by this core in hardware, but are trapped and may be emulated by software.

An Embedded Floating-point (EFPU) APU is provided to support real-time single-precision embedded numerics operations using the general-purpose registers.

A Signal Processing Extension (SPE) APU is provided to support real-time SIMD fixed point and single-precision, embedded numerics operations using the general-purpose registers. All arithmetic instructions that execute in the core operate on data in the general purpose registers (GPRs). The GPRs have been extended to 64-bits in order to support vector instructions defined by the SPE APU. These instructions operate on a vector pair of 16-bit or 32-bit data types, and deliver vector and scalar results.

In addition to the base Power Architecture instruction set support, the e200z4 core also implements the VLE (variable-length encoding) technology, providing improved code density.

The e200z4 processor integrates a pair of integer execution units, a branch control unit, instruction fetch unit and load/store unit, and a multi-ported register file capable of sustaining six read and three write operations per clock. Most integer instructions execute in a single clock cycle. Branch target prefetching is performed by the branch unit to allow single-cycle branches in many cases.

The e200z4 contains an 8 KB Instruction Cache as well as a Memory Management Unit. A Nexus Class 3 module is also integrated.

7.2 Features

Features of the e200z4 core include:

- Dual issue, 32-bit Power Architecture compliant CPU
- Implements the VLE APU for reduced code footprint
- In-order execution and retirement
- Precise exception handling
- Branch processing unit
  - Dedicated branch address calculation adder
  - Branch target prefetching using 8-entry BTB
- Supports independent instruction and data accesses to different memory subsystems, such as SRAM and Flash memory via independent Instruction and Data BIUs
- Load/store unit
  - 2 cycle load latency
  - Fully pipelined
  - Big and Little endian support
  - Misaligned access support
- 64-bit General Purpose Register file
- 64-bit Instruction bus, 64-bit Data bus
- Memory Management Unit (MMU) with 24-entry fully-associative TLB and multiple page size support
- 8 KB, 2-way or 4-way Set Associative Instruction Cache
- Signal Processing Extension (SPE1.1) APU supporting SIMD fixed-point operations using the 64-bit General Purpose Register file.
- Embedded Floating-Point (EFP2) APU supporting scalar and vector SIMD single-precision floating-point operations, using the 64-bit General Purpose Register file.
- Nexus Class 3 real-time Development Unit
- Power management
  - Power saving mode: WAIT
- Process ID manipulation for the MMU using an external tool

7.3 Microarchitecture summary

The e200z4 utilizes a five-stage pipeline for instruction execution.

These stages are:

- Instruction Fetch (stage 1)
- Instruction Decode/Register file Read/Effective Address Calculation (stage 2)
- Execute 0/Memory Access 0 (stage 3)
- Execute 1/Memory Access 1 (stage 4)
- Register Writeback (stage 5)

The stages operate in an overlapped fashion, allowing single clock instruction execution for most instructions.

The integer execution unit consists of a 32-bit Arithmetic Unit (AU), a Logic Unit (LU), a 32-bit Barrel shifter (Shifter), a Mask-Insertion Unit (MIU), a Condition Register manipulation Unit (CRU), a Count-Leading-Zeros unit (CLZ), a 32x32 Hardware Multiplier array, and result feed-forward hardware. Integer EU1 also supports hardware division.

Most arithmetic and logical operations are executed in a single cycle with the exception of multiply, which is implemented with a 2-cycle pipelined hardware array, and the divide instructions. A Count-Leading-Zeros unit operates in a single clock cycle.

The Instruction Unit contains a PC incrementer and dedicated Branch Address adders to minimize delays during change of flow operations. Sequential prefetching is performed to ensure a supply of instructions into the execution pipeline. Branch target prefetching using the BTB is performed to accelerate taken branches. Prefetched instructions are placed into an 8-entry instruction buffer, with each entry capable of holding a single 32-bit instruction or a pair of 16-bit instructions.

Branch target addresses are calculated in parallel with branch instruction decode. Conditional branches, which are not taken execute in a single clock. Branches with successful BTB target prefetching have an effective execution time of one clock if correctly predicted.

Memory load and store operations are provided for byte, halfword, word (32-bit), and doubleword data with automatic zero or sign extension of byte and halfword load data as well as optional byte reversal of data. These instructions can be pipelined to allow effective single cycle throughput. Load and store multiple word instructions allow low overhead context save and restore operations. The load/store unit contains a dedicated effective address adder to allow effective address generation to be optimized. There is a single load-to-use bubble for load instructions.

The Condition Register unit supports the condition register (CR) and condition register operations defined by the Power Architecture technology. The condition register consists of eight 4-bit fields that reflect the results of certain operations, such as move, integer and floating-point compare, arithmetic, and logical instructions, and provides a mechanism for testing and branching.

Vectored and autovectored interrupts are supported by the CPU. Vectored interrupt support is provided to allow multiple interrupt sources to have unique interrupt handlers invoked with no software overhead.

The SPE APU supports vector instructions operating on 16 and 32-bit fixed-point data types, as well as 32-bit IEEE-754 single-precision floating-point formats, and supports single-precision floating-point operations in a pipelined fashion. The 64-bit general purpose register file is used for source and destination operands, and there is a unified storage model for single-precision floating-point data types of 32-bits and the normal integer type. Low latency fixed-point and floating-point add, subtract, multiply, multiply-add, multiply-sub, divide, compare, and conversion operations are provided, and most operations can be pipelined.

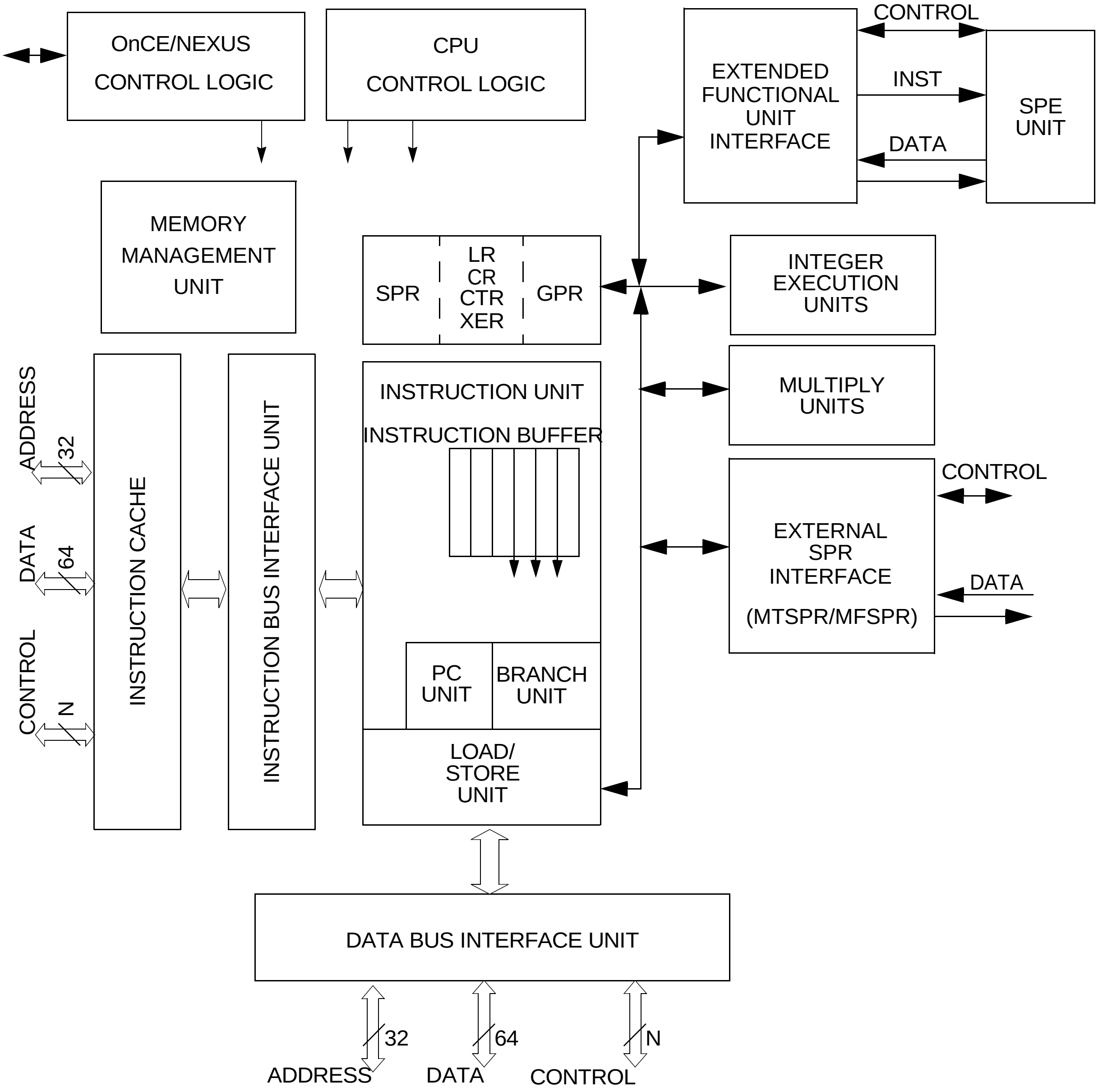


Figure 18. e200z4 block diagram

7.3.1 Instruction unit features

The features of e200z4 instruction unit are:

- 64-bit path to cache supports fetching of two 32-bit instruction per clock
- Instruction buffer holds up to eight 32-bit instructions
- Dedicated PC incrementer supporting instruction prefetches
- Branch unit with dedicated branch address adder, and branch lookahead logic (BTB) supporting single cycle execution of successfully predicted branches

7.3.2 Integer unit features

The e200z4 integer units support single cycle execution of most integer instructions:

- 32-bit AU for arithmetic and comparison operations
- 32-bit LU for logical operations
- 32-bit priority encoder for count leading zero's function
- 32-bit single cycle barrel shifter for static shifts and rotates
- 32-bit mask unit for data masking and insertion
- Divider logic for signed and unsigned divide in <=14 clocks with minimized execution timing (EU1 only)
- Pipelined 32x32 hardware multiplier array supports 32x32->32 multiply with 2 clock latency, 1 clock throughput

7.3.3 Load/Store unit features

The e200z4 load/store unit supports load, store, and the load multiple / store multiple instructions:

- 32-bit effective address adder for data memory address calculations
- Pipelined operation supports throughput of one load or store operation per cycle
- Dedicated 64-bit interface to memory supports saving and restoring of up to two registers per cycle for load multiple and store multiple word instructions

7.3.4 Cache features

The features of the e200z4 Cache are as follows:

- 8 KB, 2- or 4-way configurable set-associative Instruction Cache
- Linefill Buffer
- 32-bit address bus plus attributes and control
- Supports cache line locking
- Supports Way allocation
- Supports Tag and Data Parity
- Supports Tag and Data Double Error Detection
- Correction/Auto-invalidation capability

7.3.5 MMU features

The features of the MMU are as follows:

- Virtual Memory support
- 32-bit Virtual and Physical Addresses
- 8-bit Process Identifier
- 24-entry fully-associative TLB
- Per-entry multiple page size support from 1 Kbyte to 4 Gbyte
- Entry Flush Protection
- Process ID manipulation for the MMU using an external tool

7.3.6 e200z4 system bus features

The features of the e200z4 System Bus interface are as follows:

- Independent Instruction and Data buses
- 32-bit address bus, 64-bit data bus, plus attributes and control
- Data interface provides separate unidirectional 64-bit read and write data buses

7.3.7 Nexus 3 features

The Nexus 3 module is compliant with Class 3 of the IEEE-ISTO 5001™ - 2003 standard, with certain additional Class 4 features available. The following features are implemented:

- Program Trace via Branch Trace Messaging (BTM). Branch trace messaging displays program flow discontinuities (direct and indirect branches, exceptions, etc.), allowing the development tool to interpolate what transpires between the discontinuities. Thus, static code may be traced.
- Data Trace via Data Write Messaging (DWM) and Data Read Messaging (DRM). This provides the capability for the development tool to trace reads and/or writes to selected internal memory resources.
- Ownership Trace via Ownership Trace Messaging (OTM). OTM facilitates ownership trace by providing visibility of which process ID or operating system task is activated. An Ownership Trace Message is transmitted when a new process/task is activated, allowing the development tool to trace ownership flow.
- Run-time access to the processor memory map via the JTAG port. This allows for enhanced download/upload capabilities.
- Watchpoint Messaging via the auxiliary interface.
- Watchpoint Trigger enable of Program and/or Data Trace Messaging.
- Data Acquisition Messaging (DQM) allows code to be instrumented to export customized information to the Nexus Auxiliary Output Port.
- Auxiliary interface for higher data input/output.
- Registers for Program Trace, Data Trace, Ownership Trace, Data Acquisition, and Watchpoint Trigger control.
- All features controllable and configurable via the JTAG port.

8 Enhanced Direct Memory Access Controller (eDMA)

8.1 Introduction

This device includes an enhanced direct memory access controller (eDMA) block. The eDMA is a second-generation platform block capable of performing complex data movements through 64 programmable channels with minimal intervention from the host processor. The hardware microarchitecture includes a DMA engine that performs source and destination address calculations, and the actual data movement operations, along with an SRAM-based memory containing the transfer control descriptors (TCD) for the channels.

8.1.1 Block diagram

Figure 19 shows a simplified block diagram of the eDMA.

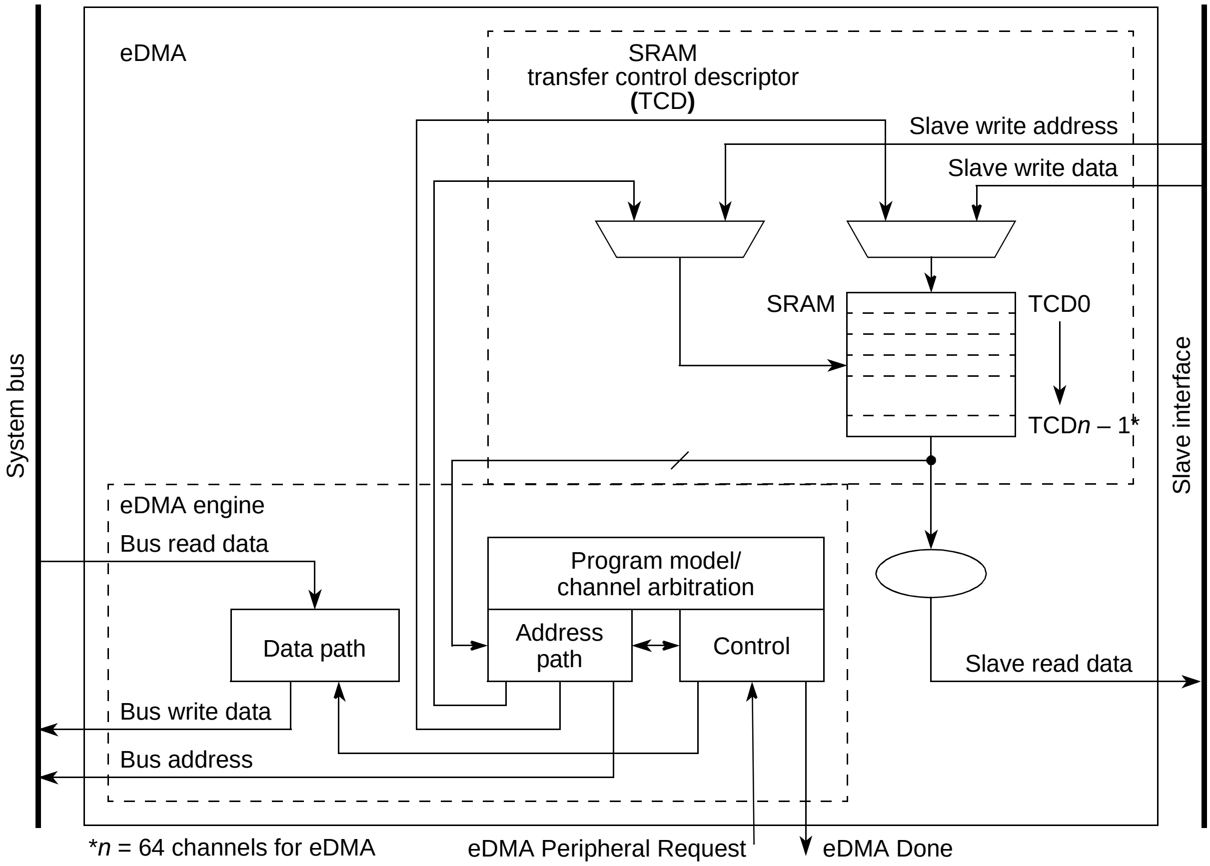


Figure 19. eDMA block diagram

8.1.2 Features

The eDMA has these major features:

- All data movement via dual-address transfers: read from source, write to destination
  - Programmable source, destination addresses, transfer size, and support for enhanced addressing modes
- Both 32- and 64-channel implementation performs complex data transfers with minimal intervention from a host processor
  - 32 bytes of data registers, used as temporary storage to support burst transfers (refer to SSIZE bit)
  - Connections to the crossbar switch for bus mastering the data movement
- Transfer control descriptor organized to support two-deep, nested transfer operations
  - An inner data transfer loop defined by a minor byte transfer count
  - An outer data transfer loop defined by a major iteration count
- Channel activation via 1 of 3 methods:
  - Explicit software initiation
  - Initiation via a channel-to-channel linking mechanism for continuous transfers
  - Peripheral-paced hardware requests (one per channel)

    All three methods require one activation per execution of the minor loop
- Support for fixed-priority and round-robin channel arbitration
- Support for complex data structures
- Support to cancel transfers via software
- Channel completion reported via optional interrupt requests
  - 1 interrupt per channel, optionally asserted at completion of major iteration count
  - Error terminations are optionally enabled per channel and logically summed together to form a single error interrupt (32-channel eDMA) or two error interrupts (64-channel eDMA).
- Support for scatter-gather DMA processing
- Support for complex data structures
- Any channel can be programmed to be suspended by a higher priority channel's activation, before completion of a minor loop.

8.1.3 Modes of operation

There are two main operating modes of eDMA: normal mode and debug mode. These modes are briefly described in this section.

Normal mode

In normal mode, the eDMA is used to transfer data between a source and a destination. The source and destination can be a memory block or an I/O block capable of operation with the eDMA.

Debug mode

In debug mode, the eDMA does not accept new transfer requests when its debug input signal is asserted. If the signal is asserted during transfer of a block of data described by a minor loop in the current active channel's TCD, the eDMA continues operation until completion of the minor loop.

8.2 External signal description

The eDMA has no external signals.

8.3 Memory map and registers

This section provides a detailed description of all eDMA registers.

8.3.1 Module memory map

The eDMA memory map is shown in *Table 22*. The address of each register is given as an offset to the eDMA base address. Registers are listed in address order, identified by complete name and mnemonic, and list the type of accesses allowed. *Table 23* shows a graphical representation of the same memory map. In register names, an "*x*" is used to indicate A or B, depending on which eDMA's register you are using. If a register only exists in one of the eDMAs, the register description will state that.

The eDMA's programming model is partitioned into two regions: the first region defines a number of registers providing control functions; however, the second region corresponds to the local transfer control descriptor memory.

Some registers are implemented as two 32-bit registers, and include H and L suffixes, signaling the high and low portions of the control function.

Table 22. eDMA memory map

Offset from EDMA_BASE	Register	Location	Size
EDMA_BASE (0xFFF4_4000)	EDMA_CR—eDMA control register	*on page 8-170*	32
EDMA_BASE + 0x0004	EDMA_ESR—eDMA error status register	*on page 8-173*	32
EDMA_BASE + 0x0008	EDMA_ERQRH—eDMA enable request high register (channels 63–32)	*on page 8-176*	32
EDMA_BASE + 0x000C	EDMA_ERQRL—eDMA enable request low register (channels 31–00)	*on page 8-176*	32
EDMA_BASE + 0x0010	EDMA_EEIRLH—eDMA enable error Interpol register (channels 63–32)	*on page 8-177*	32
EDMA_BASE + 0x0014	EDMA_EEIRL—eDMA enable error interrupt register (channels 31–00)	*on page 8-177*	32
EDMA_BASE + 0x0018	EDMA_SERQR—eDMA set enable request register	*on page 8-178*	8
EDMA_BASE + 0x0019	EDMA_CERQR—eDMA clear enable request register	*on page 8-179*	8
EDMA_BASE + 0x001A	EDMA_SEEIR—eDMA set enable error interrupt register	*on page 8-180*	8
EDMA_BASE + 0x001B	EDMA_CEEIR—eDMA clear enable error interrupt register	*on page 8-180*	8
EDMA_BASE + 0x001C	EDMA_CIRQR—eDMA clear interrupt request register	*on page 8-181*	8

Table 22. eDMA memory map (continued)

Offset from EDMA_BASE	Register	Location	Size
EDMA_BASE + 0x001D	EDMA_CER—eDMA clear error register	*on page 8-181*	8
EDMA_BASE + 0x001E	EDMA_SSBR—eDMA set start bit register	*on page 8-182*	8
EDMA_BASE + 0x001F	EDMA_CDSBR—eDMA clear done status bit register	*on page 8-183*	8
EDMA_BASE + 0x0020	EDMA_IRQRH—eDMA interrupt request register (channels 63–32)	*on page 8-183*	32
EDMA_BASE + 0x0024	EDMA_IRQRL—eDMA interrupt request register (channels 31–00)	*on page 8-183*	32
EDMA_BASE + 0x0028	EDMA_ERH—eDMA error register (channels 63–32)	*on page 8-184*	32
EDMA_BASE + 0x002C	EDMA_ERL—eDMA error register (channels 31–00)	*on page 8-184*	32
EDMA_BASE + 0x0030	EDMA_HRSH—eDMA hardware request status register (channels 63–32)	*on page 8-186*	32
EDMA_BASE + 0x0034	EDMA_HRSL—eDMA hardware request status register (channels 31–00)	*on page 8-186*	32
EDMA_BASE + 0x0038– EDMA_BASE + 0x00FF	Reserved		
EDMA_BASE + 0x0100	EDMA_CPR0—eDMA channel 0 priority register	*on page 8-187*	8
EDMA_BASE + 0x0101	EDMA_CPR1—eDMA channel 1 priority register	*on page 8-187*	8
EDMA_BASE + 0x0102	EDMA_CPR2—eDMA channel 2 priority register	*on page 8-187*	8
EDMA_BASE + 0x0103	EDMA_CPR3—eDMA channel 3 priority register	*on page 8-187*	8
EDMA_BASE + 0x0104	EDMA_CPR4—eDMA channel 4 priority register	*on page 8-187*	8
EDMA_BASE + 0x0105	EDMA_CPR5—eDMA channel 5 priority register	*on page 8-187*	8
EDMA_BASE + 0x0106	EDMA_CPR6—eDMA channel 6 priority register	*on page 8-187*	8
EDMA_BASE + 0x0107	EDMA_CPR7—eDMA channel 7 priority register	*on page 8-187*	8
EDMA_BASE + 0x0108	EDMA_CPR8—eDMA channel 8 priority register	*on page 8-187*	8
EDMA_BASE + 0x0109	EDMA_CPR9—eDMA channel 9 priority register	*on page 8-187*	8
EDMA_BASE + 0x010A	EDMA_CPR10—eDMA channel 10 priority register	*on page 8-187*	8
EDMA_BASE + 0x010B	EDMA_CPR11—eDMA channel 11 priority register	*on page 8-187*	8
EDMA_BASE + 0x010C	EDMA_CPR12—eDMA channel 12 priority register	*on page 8-187*	8
EDMA_BASE + 0x010D	EDMA_CPR13—eDMA channel 13 priority register	*on page 8-187*	8
EDMA_BASE + 0x010E	EDMA_CPR14—eDMA channel 14 priority register	*on page 8-187*	8
EDMA_BASE + 0x010F	EDMA_CPR15—eDMA channel 15 priority register	*on page 8-187*	8
EDMA_BASE + 0x0110	EDMA_CPR16—eDMA channel 16 priority register	*on page 8-187*	8
EDMA_BASE + 0x0111	EDMA_CPR17—eDMA channel 17 priority register	*on page 8-187*	8
EDMA_BASE + 0x0112	EDMA_CPR18—eDMA channel 18 priority register	*on page 8-187*	8
EDMA_BASE + 0x0113	EDMA_CPR19—eDMA channel 19 priority register	*on page 8-187*	8

Table 22. eDMA memory map (continued)

Offset from EDMA_BASE	Register	Location	Size
EDMA_BASE + 0x0114	EDMA_CPR20—eDMA channel 20 priority register	*on page 8-187*	8
EDMA_BASE + 0x0115	EDMA_CPR21—eDMA channel 21 priority register	*on page 8-187*	8
EDMA_BASE + 0x0116	EDMA_CPR22—eDMA channel 22 priority register	*on page 8-187*	8
EDMA_BASE + 0x0117	EDMA_CPR23—eDMA channel 23 priority register	*on page 8-187*	8
EDMA_BASE + 0x0118	EDMA_CPR24—eDMA channel 24 priority register	*on page 8-187*	8
EDMA_BASE + 0x0119	EDMA_CPR25—eDMA channel 25 priority register	*on page 8-187*	8
EDMA_BASE + 0x011A	EDMA_CPR26—eDMA channel 26 priority register	*on page 8-187*	8
EDMA_BASE + 0x011B	EDMA_CPR27—eDMA channel 27 priority register	*on page 8-187*	8
EDMA_BASE + 0x011C	EDMA_CPR28—eDMA channel 28 priority register	*on page 8-187*	8
EDMA_BASE + 0x011D	EDMA_CPR29—eDMA channel 29 priority register	*on page 8-187*	8
EDMA_BASE + 0x011E	EDMA_CPR30—eDMA channel 30 priority register	*on page 8-187*	8
EDMA_BASE + 0x011F	EDMA_CPR31—eDMA channel 31 priority register	*on page 8-187*	8
EDMA_BASE + 0x0120	EDMA_CPR32—eDMA channel 32 priority register	*on page 8-187*	8
EDMA_BASE + 0x0121	EDMA_CPR33—eDMA channel 33 priority register	*on page 8-187*	8
EDMA_BASE + 0x0122	EDMA_CPR34—eDMA channel 34 priority register	*on page 8-187*	8
EDMA_BASE + 0x0123	EDMA_CPR35—eDMA channel 35 priority register	*on page 8-187*	8
EDMA_BASE + 0x0124	EDMA_CPR36—eDMA channel 36 priority register	*on page 8-187*	8
EDMA_BASE + 0x0125	EDMA_CPR37—eDMA channel 37 priority register	*on page 8-187*	8
EDMA_BASE + 0x0126	EDMA_CPR38—eDMA channel 38 priority register	*on page 8-187*	8
EDMA_BASE + 0x0127	EDMA_CPR39—eDMA channel 39 priority register	*on page 8-187*	8
EDMA_BASE + 0x0128	EDMA_CPR40—eDMA channel 40 priority register	*on page 8-187*	8
EDMA_BASE + 0x0129	EDMA_CPR41—eDMA channel 41 priority register	*on page 8-187*	8
EDMA_BASE + 0x012A	EDMA_CPR42—eDMA channel 42 priority register	*on page 8-187*	8
EDMA_BASE + 0x012B	EDMA_CPR43—eDMA channel 43 priority register	*on page 8-187*	8
EDMA_BASE + 0x012C	EDMA_CPR44—eDMA channel 44 priority register	*on page 8-187*	8
EDMA_BASE + 0x012D	EDMA_CPR45—eDMA channel 45 priority register	*on page 8-187*	8
EDMA_BASE + 0x012E	EDMA_CPR46—eDMA channel 46 priority register	*on page 8-187*	8
EDMA_BASE + 0x012F	EDMA_CPR47—eDMA channel 47 priority register	*on page 8-187*	8
EDMA_BASE + 0x0130	EDMA_CPR48—eDMA channel 48 priority register	*on page 8-187*	8
EDMA_BASE + 0x0131	EDMA_CPR49—eDMA channel 49 priority register	*on page 8-187*	8
EDMA_BASE + 0x0132	EDMA_CPR50—eDMA channel 50 priority register	*on page 8-187*	8
EDMA_BASE + 0x0133	EDMA_CPR51—eDMA channel 51 priority register	*on page 8-187*	8
EDMA_BASE + 0x0134	EDMA_CPR52—eDMA channel 52 priority register	*on page 8-187*	8
EDMA_BASE + 0x0135	EDMA_CPR53—eDMA channel 53 priority register	*on page 8-187*	8

Table 22. eDMA memory map (continued)

Offset from EDMA_BASE	Register	Location	Size
EDMA_BASE + 0x0136	EDMA_CPR54—eDMA channel 54 priority register	*on page 8-187*	8
EDMA_BASE + 0x0137	EDMA_CPR55—eDMA channel 55 priority register	*on page 8-187*	8
EDMA_BASE + 0x0138	EDMA_CPR56—eDMA channel 56 priority register	*on page 8-187*	8
EDMA_BASE + 0x0139	EDMA_CPR57—eDMA channel 57 priority register	*on page 8-187*	8
EDMA_BASE + 0x013A	EDMA_CPR58—eDMA channel 58 priority register	*on page 8-187*	8
EDMA_BASE + 0x013B	EDMA_CPR59—eDMA channel 59 priority register	*on page 8-187*	8
EDMA_BASE + 0x013C	EDMA_CPR60—eDMA channel 60 priority register	*on page 8-187*	8
EDMA_BASE + 0x013D	EDMA_CPR61—eDMA channel 61 priority register	*on page 8-187*	8
EDMA_BASE + 0x013E	EDMA_CPR62—eDMA channel 62 priority register	*on page 8-187*	8
EDMA_BASE + 0x013F	EDMA_CPR63—eDMA channel 63 priority register	*on page 8-187*	8
EDMA_BASE + 0x0140– EDMA_BASE + 0x0FFF	Reserved		
EDMA_BASE + 0x1000	EDMA_TCD00—eDMA transfer control descriptor 00	*on page 8-188*	256
EDMA_BASE + 0x1020	EDMA_TCD01—eDMA transfer control descriptor 01	*on page 8-188*	256
EDMA_BASE + 0x1040	EDMA_TCD02—eDMA transfer control descriptor 02	*on page 8-188*	256
EDMA_BASE + 0x1060	EDMA_TCD03—eDMA transfer control descriptor 03	*on page 8-188*	256
EDMA_BASE + 0x1080	EDMA_TCD04—eDMA transfer control descriptor 04	*on page 8-188*	256
EDMA_BASE + 0x10A0	EDMA_TCD05—eDMA transfer control descriptor 05	*on page 8-188*	256
EDMA_BASE + 0x10C0	EDMA_TCD06—eDMA transfer control descriptor 06	*on page 8-188*	256
EDMA_BASE + 0x10E0	EDMA_TCD07—eDMA transfer control descriptor 07	*on page 8-188*	256
EDMA_BASE + 0x1100	EDMA_TCD08—eDMA transfer control descriptor 08	*on page 8-188*	256
EDMA_BASE + 0x1120	EDMA_TCD09—eDMA transfer control descriptor 09	*on page 8-188*	256
EDMA_BASE + 0x1140	EDMA_TCD10—eDMA transfer control descriptor 10	*on page 8-188*	256
EDMA_BASE + 0x1160	EDMA_TCD11—eDMA transfer control descriptor 11	*on page 8-188*	256
EDMA_BASE + 0x1180	EDMA_TCD12—eDMA transfer control descriptor 12	*on page 8-188*	256
EDMA_BASE + 0x11A0	EDMA_TCD13—eDMA transfer control descriptor 13	*on page 8-188*	256
EDMA_BASE + 0x11C0	EDMA_TCD14—eDMA transfer control descriptor 14	*on page 8-188*	256
EDMA_BASE + 0x11E0	EDMA_TCD15—eDMA transfer control descriptor 15	*on page 8-188*	256
EDMA_BASE + 0x1200	EDMA_TCD16—eDMA transfer control descriptor 16	*on page 8-188*	256
EDMA_BASE + 0x1220	EDMA_TCD17—eDMA transfer control descriptor 17	*on page 8-188*	256
EDMA_BASE + 0x1240	EDMA_TCD18—eDMA transfer control descriptor 18	*on page 8-188*	256
EDMA_BASE + 0x1260	EDMA_TCD19—eDMA transfer control descriptor 19	*on page 8-188*	256
EDMA_BASE + 0x1280	EDMA_TCD20—eDMA transfer control descriptor 20	*on page 8-188*	256
EDMA_BASE + 0x12A0	EDMA_TCD21—eDMA transfer control descriptor 21	*on page 8-188*	256

Table 22. eDMA memory map (continued)

Offset from EDMA_BASE	Register	Location	Size
EDMA_BASE + 0x12C0	EDMA_TCD22—eDMA transfer control descriptor 22	*on page 8-188*	256
EDMA_BASE + 0x12E0	EDMA_TCD23—eDMA transfer control descriptor 23	*on page 8-188*	256
EDMA_BASE + 0x1300	EDMA_TCD24—eDMA transfer control descriptor 24	*on page 8-188*	256
EDMA_BASE + 0x1320	EDMA_TCD25—eDMA transfer control descriptor 25	*on page 8-188*	256
EDMA_BASE + 0x1340	EDMA_TCD26—eDMA transfer control descriptor 26	*on page 8-188*	256
EDMA_BASE + 0x1360	EDMA_TCD27—eDMA transfer control descriptor 27	*on page 8-188*	256
EDMA_BASE + 0x1380	EDMA_TCD28—eDMA transfer control descriptor 28	*on page 8-188*	256
EDMA_BASE + 0x13A0	EDMA_TCD29—eDMA transfer control descriptor 29	*on page 8-188*	256
EDMA_BASE + 0x13C0	EDMA_TCD30—eDMA transfer control descriptor 30	*on page 8-188*	256
EDMA_BASE + 0x13E0	EDMA_TCD31—eDMA transfer control descriptor 31	*on page 8-188*	256
EDMA_BASE + 0x1400	EDMA_TCD32—eDMA transfer control descriptor 32	*on page 8-188*	256
EDMA_BASE + 0x1420	EDMA_TCD33—eDMA transfer control descriptor 33	*on page 8-188*	256
EDMA_BASE + 0x1440	EDMA_TCD34—eDMA transfer control descriptor 34	*on page 8-188*	256
EDMA_BASE + 0x1460	EDMA_TCD35—eDMA transfer control descriptor 35	*on page 8-188*	256
EDMA_BASE + 0x1480	EDMA_TCD36—eDMA transfer control descriptor 36	*on page 8-188*	256
EDMA_BASE + 0x14A0	EDMA_TCD37—eDMA transfer control descriptor 37	*on page 8-188*	256
EDMA_BASE + 0x14C0	EDMA_TCD38—eDMA transfer control descriptor 38	*on page 8-188*	256
EDMA_BASE + 0x14E0	EDMA_TCD39—eDMA transfer control descriptor 39	*on page 8-188*	256
EDMA_BASE + 0x1500	EDMA_TCD40—eDMA transfer control descriptor 40	*on page 8-188*	256
EDMA_BASE + 0x1520	EDMA_TCD41—eDMA transfer control descriptor 41	*on page 8-188*	256
EDMA_BASE + 0x1540	EDMA_TCD42—eDMA transfer control descriptor 42	*on page 8-188*	256
EDMA_BASE + 0x1560	EDMA_TCD43—eDMA transfer control descriptor 43	*on page 8-188*	256
EDMA_BASE + 0x1580	EDMA_TCD44—eDMA transfer control descriptor 44	*on page 8-188*	256
EDMA_BASE + 0x15A0	EDMA_TCD45—eDMA transfer control descriptor 45	*on page 8-188*	256
EDMA_BASE + 0x15C0	EDMA_TCD46—eDMA transfer control descriptor 46	*on page 8-188*	256
EDMA_BASE + 0x15E0	EDMA_TCD47—eDMA transfer control descriptor 47	*on page 8-188*	256
EDMA_BASE + 0x1600	EDMA_TCD48—eDMA transfer control descriptor 48	*on page 8-188*	256
EDMA_BASE + 0x1620	EDMA_TCD49—eDMA transfer control descriptor 49	*on page 8-188*	256
EDMA_BASE + 0x1640	EDMA_TCD50—eDMA transfer control descriptor 50	*on page 8-188*	256
EDMA_BASE + 0x1660	EDMA_TCD51—eDMA transfer control descriptor 51	*on page 8-188*	256
EDMA_BASE + 0x1680	EDMA_TCD52—eDMA transfer control descriptor 52	*on page 8-188*	256
EDMA_BASE + 0x16A0	EDMA_TCD53—eDMA transfer control descriptor 53	*on page 8-188*	256
EDMA_BASE + 0x16C0	EDMA_TCD54—eDMA transfer control descriptor 54	*on page 8-188*	256
EDMA_BASE + 0x16E0	EDMA_TCD55—eDMA transfer control descriptor 55	*on page 8-188*	256

Table 22. eDMA memory map (continued)

Offset from EDMA_BASE	Register	Location	Size
EDMA_BASE + 0x1700	EDMA_TCD56—eDMA transfer control descriptor 56	*on page 8-188*	256
EDMA_BASE + 0x1720	EDMA_TCD57—eDMA transfer control descriptor 57	*on page 8-188*	256
EDMA_BASE + 0x1740	EDMA_TCD58—eDMA transfer control descriptor 58	*on page 8-188*	256
EDMA_BASE + 0x1760	EDMA_TCD59—eDMA transfer control descriptor 59	*on page 8-188*	256
EDMA_BASE + 0x1780	EDMA_TCD60—eDMA transfer control descriptor 60	*on page 8-188*	256
EDMA_BASE + 0x17A0	EDMA_TCD61—eDMA transfer control descriptor 61	*on page 8-188*	256
EDMA_BASE + 0x17C0	EDMA_TCD62—eDMA transfer control descriptor 62	*on page 8-188*	256
EDMA_BASE + 0x17E0	EDMA_TCD63—eDMA transfer control descriptor 63	*on page 8-188*	256

Table 23. eDMA 32-bit memory map—graphical view

<table>
<tr><th>Address</th><th colspan="4">Register</th></tr>
<tr><td>0xFFF4_4000</td><td colspan="4">eDMA Control Register (EDMA_CR)</td></tr>
<tr><td>0xFFF4_4004</td><td colspan="4">eDMA Error Status (EDMA_ESR)</td></tr>
<tr><td>0xFFF4_4008</td><td colspan="4">eDMA enable request high register (EDMA_ERQRH)</td></tr>
<tr><td>0xFFF4_400C</td><td colspan="2">eDMA Enable Request (EDMA_ERQRL, channels 31–16)</td><td colspan="2">eDMA Enable Request (EDMA_ERQRL, channels 15–00)</td></tr>
<tr><td>0xFFF4_4010</td><td colspan="2">eDMA Enable Error Interrupt High (EDMA_EEIRH, channels 63–48)</td><td colspan="2">eDMA Enable Error Interrupt High (EDMA_EEIRH, Channels 47–32)</td></tr>
<tr><td>0xFFF4_4014</td><td colspan="2">eDMA Enable Error Interrupt Low (EDMA_EEIRL, channels 31–16)</td><td colspan="2">eDMA Enable Error Interrupt Low (EDMA_EEIRL, Channels 15–00)</td></tr>
<tr><td>0xFFF4_4018</td><td>eDMA Set Enable Request (EDMA_SERQR)</td><td>eDMA Clear Enable Request (EDMA_CERQR)</td><td>eDMA Set Enable Error Interrupt (EDMA_SEEIR)</td><td>eDMA Clear Enable Error Interrupt (EDMA_CEEIR)</td></tr>
<tr><td>0xFFF4_401C</td><td>eDMA Clear Interrupt Request (EDMA_CIRQR)</td><td>eDMA Clear Error (EDMA_CER)</td><td>eDMA Set Start Bit, Activate Channel (EDMA_SSBR)</td><td>eDMA Clear Done Status Bit (EDMA_CDSBR)</td></tr>
<tr><td>0xFFF4_4020</td><td colspan="2">eDMA Interrupt Request High (EDMA_IRQRH channels 63–48)</td><td colspan="2">eDMA Interrupt Request High (EDMA_IRQRH, Channels 47–32)</td></tr>
<tr><td>0xFFF4_4024</td><td colspan="2">eDMA Interrupt Request Low (EDMA_IRQRL, channels 31–16)</td><td colspan="2">eDMA Interrupt Request Low (EDMA_IRQRL, Channels 15–00)</td></tr>
<tr><td>0xFFF4_4028</td><td colspan="2">eDMA Error High (EDMA_ERL, channels 63–48)</td><td colspan="2">eDMA Error High (EDMA_ERL, Channels 47–32)</td></tr>
<tr><td>0xFFF4_402C</td><td colspan="2">eDMA Error Low (EDMA_ERL, channels 31–16)</td><td colspan="2">eDMA Error Low (EDMA_ERL, Channels 15–00)</td></tr>
<tr><td>0xFFF4_4030</td><td colspan="2">eDMA Hardware Request Status High (EDMA_HRSL, channels 63–48)</td><td colspan="2">eDMA Hardware Request Status High (EDMA_HRSL, Channels 47–32)</td></tr>
</table>

Table 23. eDMA 32-bit memory map—graphical view (continued)

Address	Register			
0xFFF4_4034	eDMA Hardware Request Status Low (EDMA_HRSL, channels 31–16)		eDMA Hardware Request Status Low (EDMA_HRSL, Channels 15–00)	
0xFFF4_4038 – 0xFFF4_40FC	Reserved			
0xFFF4_4100	eDMA Channel 0 Priority (EDMA_CPR0)	eDMA Channel 1 Priority (EDMA_CPR1)	eDMA Channel 2 Priority (EDMA_CPR2)	eDMA Channel 3 Priority (EDMA_CPR3)
0xFFF4_4104	eDMA Channel 4 Priority (EDMA_CPR4)	eDMA Channel 5 Priority (EDMA_CPR5)	eDMA Channel 6 Priority (EDMA_CPR6)	eDMA Channel 7 Priority EDMA_CPR7)
0xFFF4_4108	eDMA Channel 8 Priority (EDMA_CPR8)	eDMA Channel 9 Priority (EDMA_CPR9)	eDMA Channel 10 Priority (EDMA_CPR10)	eDMA Channel 11 Priority (EDMA_CPR11)
0xFFF4_410C	eDMA Channel 12 Priority (EDMA_CPR12)	eDMA Channel 13 Priority (EDMA_CPR13)	eDMA Channel 14 Priority (EDMA_CPR14)	eDMA Channel 15 Priority (EDMA_CPR15)
0xFFF4_4110	eDMA Channel 16 Priority (EDMA_CPR16)	eDMA Channel 17 Priority (EDMA_CPR17)	eDMA Channel 18 Priority (EDMA_CPR18)	eDMA Channel 19 Priority (EDMA_CPR19)
0xFFF4_4114	eDMA Channel 20 Priority (EDMA_CPR16)	eDMA Channel 21 Priority (EDMA_CPR17)	eDMA Channel 22 Priority (EDMA_CPR18)	eDMA Channel 23 Priority (EDMA_CPR19)
0xFFF4_4118	eDMA Channel 24 Priority (EDMA_CPR16)	eDMA Channel 25 Priority (EDMA_CPR17)	eDMA Channel 26 Priority (EDMA_CPR18)	eDMA Channel 27 Priority (EDMA_CPR19)
0xFFF4_411C	eDMA Channel 28 Priority (EDMA_CPR16)	eDMA Channel 29 Priority (EDMA_CPR17)	eDMA Channel 30 Priority (EDMA_CPR18)	eDMA Channel 31 Priority (EDMA_CPR19)
0xFFF4_4100	eDMA Channel 32 Priority (EDMA_CPR32)	eDMA Channel 33 Priority (EDMA_CPR33)	eDMA Channel 34 Priority (EDMA_CPR34)	eDMA Channel 35 Priority (EDMA_CPR35)
0xFFF4_4104	eDMA Channel 36 Priority (EDMA_CPR36)	eDMA Channel 37 Priority (EDMA_CPR37)	eDMA Channel 38 Priority (EDMA_CPR38)	eDMA Channel 39 Priority EDMA_CPR39)
0xFFF4_4108	eDMA Channel 40 Priority (EDMA_CPR40)	eDMA Channel 41 Priority (EDMA_CPR41)	eDMA Channel 42 Priority (EDMA_CPR42)	eDMA Channel 43 Priority (EDMA_CPR43)
0xFFF4_410C	eDMA Channel 44 Priority (EDMA_CPR44)	eDMA Channel 45 Priority (EDMA_CPR45)	eDMA Channel 46 Priority (EDMA_CPR46)	eDMA Channel 47 Priority (EDMA_CPR47)
0xFFF4_4110	eDMA Channel 48 Priority (EDMA_CPR48)	eDMA Channel 49 Priority (EDMA_CPR49)	eDMA Channel 50 Priority (EDMA_CPR50)	eDMA Channel 51 Priority (EDMA_CPR51)

Table 23. eDMA 32-bit memory map—graphical view (continued)

Address	Register			
0xFFF4_4114	eDMA Channel 52 Priority (EDMA_CPR52)	eDMA Channel 53 Priority (EDMA_CPR53)	eDMA Channel 54 Priority (EDMA_CPR54)	eDMA Channel 55 Priority (EDMA_CPR55)
0xFFF4_4118	eDMA Channel 56 Priority (EDMA_CPR56)	eDMA Channel 57 Priority (EDMA_CPR57)	eDMA Channel 58 Priority (EDMA_CPR58)	eDMA Channel 59 Priority (EDMA_CPR59)
0xFFF4_411C	eDMA Channel 60 Priority (EDMA_CPR60)	eDMA Channel 61 Priority (EDMA_CPR61)	eDMA Channel 62 Priority (EDMA_CPR62)	eDMA Channel 63 Priority (EDMA_CPR63)
0xFFF4_5000 – 0xFFF4_51FC	EDMA_TCD00–EDMA_TCD15			
0xFFF4_5200 – 0xFFF4_53FC	EDMA_TCD16–EDMA_TCD31			
0xFFF4_5400 – 0xFFF4_55FC	EDMA_TCD32–EDMA_TCD47			
0xFFF4_5600 – 0xFFF4_57FC	EDMA_TCD48–EDMA_TCD63			
0xFFF4_5800	Reserved			

8.3.2 Register descriptions

This section lists the eDMA registers in address order and describes the registers and their bitfields.

Reading reserved bits in a register returns the value of zero. Writes to reserved bits in a register are ignored. Reading or writing to a reserved memory location generates a bus error.

Many of the control registers have a bit width that matches the number of channels implemented in the module:

- 64 bits for eDMA (made up of two 32-bit registers: high and low—for example, EDMA_ERQRH has upper 32 channels of eDMA)

eDMA Control Register (EDMA_CR)

The 32-bit EDMA_CR defines the basic operating configuration of the eDMA.

The eDMA arbitrates channel service requests in four (eDMA) groups (0, 1, 2, 3) of 16 channels each:

- Group 0 contains channels 0–15
- Group 1 contains channels 16–31
- Group 2 contains channels 32–47 (eDMA only)
- Group 3 contains channels 48–63 (eDMA only)

Arbitration within a group can be configured to use a fixed priority or a round robin. In fixed-priority arbitration, the highest priority channel requesting service is selected to execute. The priorities are assigned by the channel priority registers. See *Section , eDMA Channel n*

Priority Registers (EDMA_CPRn). In round-robin arbitration mode, the channel priorities are ignored and the channels within each group are cycled through, from channel 15 down to channel 0,without regard to priority.

The group priorities operate in a similar fashion. In group fixed-priority arbitration mode, channel service requests in the highest priority group are executed first where priority level 3 (eDMA) is the highest and priority level 0 is the lowest. The group priorities are assigned in the GRP*n*PRI fields of the eDMA control register (EDMA_CR). All group priorities must have unique values prior to any channel service requests occur, otherwise a configuration error is reported. In group round-robin mode, the group priorities are ignored and the groups are cycled through, from group 3 (eDMA) down to group 0, without regard to priority.

Minor loop offsets are address offset values added to the final source address (SADDR) or destination address (DADDR) upon minor loop completion. When minor loop offsets are enabled, the minor loop offset (MLOFF) is added to the final source address (SADDR) or to the final destination address (DADDR) or to both addresses prior to the addresses being written back into the TCD. If the major loop is complete, the minor loop offset is ignored and the major loop address offsets (SLAST and DLAST_SGA) are used to compute the next EDMA_TCD[SADDR] and EDMA_TCD[DADDR] values.

When minor loop mapping is enabled (EDMA_CR[EMLM] = 1), TCD*n* word2 is redefined. A portion of TCD*n* word2 is used to specify multiple fields: a source enable bit (SMLOE) to specify that the minor loop offset should be applied to the source address (SADDR) upon minor loop completion, a destination enable bit (DMLOE) to specify the minor loop offset should be applied to the destination address (DADDR) upon minor loop completion, and the sign extended minor loop offset value (MLOFF). The same offset value (MLOFF) is used for both source and destination minor loop offsets.

When either of the minor loop offsets is enabled (SMLOE is set or DMLOE is set), the NBYTES field is reduced to 10 bits. When both minor loop offsets are disabled (SMLOE is cleared and DMLOE is cleared), the NBYTES field becomes a 30-bit vector.

When minor loop mapping is disabled (EDMA_CR[EMLM] = 0), all 32 bits of TCD*n* word2 are assigned to the NBYTES field. See *Section , Transfer control descriptor (TCD)* for more details.

Figure 20. eDMA Control Register (EDMA_CR)

Offset: EDMA_BASE + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	CX	ECX
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	GRP3PRI		GRP2PRI		GRP1PRI		GRP0PRI		EMLM	CLM	HALT	HOE	ERGA	ERCA	EDBG	0
W																
Reset	1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0

Table 24. EDMA_CR field descriptions

Field	Description
CX	Cancel Transfer 0 Normal operation 1 Cancel the remaining data transfer. Stop the executing channel and force the minor loop to be finished. The cancel takes effect after the last write of the current read/write sequence. The CX bit clears itself after the cancel has been honored. This cancel retires the channel normally as if the minor loop was completed.
ECX	Error cancel transfer 0 Normal operation 1 Cancel the remaining data transfer in the same fashion as the CX cancel transfer. Stop the executing channel and force the minor loop to be finished. The cancel takes effect after the last write of the current read/write sequence. The ECX bit clears itself after the cancel has been honored. In addition to cancelling the transfer, the ECX treats the cancel as an error condition; thus updating the EDMA_ESR register and generating an optional error interrupt. See *Section , eDMA Error Status Register (EDMA_ESR).*
GRP3PRI	Channel group 3 priority Group 3 priority level when fixed priority group arbitration is enabled.
GRP2PRI	Channel group 2 priority Group 2 priority level when fixed priority group arbitration is enabled.
GRP1PRI	Channel group 1 priority Group 1 priority level when fixed priority group arbitration is enabled.
GRP0PRI	Channel group 0 priority Group 0 priority level when fixed priority group arbitration is enabled.
EMLM	Enable minor loop mapping 0 Minor loop mapping disabled. TCD Word 2 is defined as a 32-bit nbytes field. 1 Minor loop mapping enabled. When set, TCD*n* Word 2 is redefined to include individual enable fields, an offset field and the NBYTES field. The individual enable fields allow the minor loop offset to be applied to the source address, the destination address, or both. The NBYTES field is reduced when either offset is enabled.
CLM	Continuous link mode 0 A minor loop channel link made to itself goes through channel arbitration before being activated again. 1 A minor loop channel link made to itself does not go through channel arbitration before being activated again. Upon minor loop completion, the channel is active again if that channel has a minor loop channel link enabled and the link channel is itself. This effectively applies the minor loop offsets and restarts the next minor loop.
HALT	Halt DMA operations 0 Normal operation 1 Stall the start of any new channels. Executing channels are allowed to complete. Channel execution resumes when the HALT bit is cleared.
HOE	Halt on error 0 Normal operation 1 Any error causes the HALT bit to be set. Subsequently, all service requests are ignored until the HALT bit is cleared.

Table 24. EDMA_CR field descriptions (continued)

Field	Description
ERGA	Enable round-robin group arbitration 0 Fixed-priority arbitration is used for selection among the groups. 1 Round-robin arbitration is used for selection among the groups.
ERCA	Enable Round-Robin Channel Arbitration 0 Fixed-priority arbitration is used for channel selection within each group. 1 Round-robin arbitration is used for channel selection within each group.
EDBG	Enable Debug 0 The assertion of the system debug control input is ignored. 1 The assertion of the system debug control input causes the eDMA to stall the start of a new channel. Executing channels are allowed to complete. Channel execution resumes when either the system debug control input is negated or the EDBG bit is cleared.

eDMA Error Status Register (EDMA_ESR)

The EDMA_ESR provides information about the last recorded channel error. Channel errors can be caused by a configuration error (an illegal setting in the transfer control descriptor or an illegal priority register setting in fixed-arbitration mode) or an error termination to a bus master read or write cycle.

A configuration error is caused when the starting source or destination address, source or destination offsets, minor loop byte count, and the transfer size represent an inconsistent state. The addresses and offsets must be aligned on 0-modulo-transfer_size boundaries, and the minor loop byte count must be a multiple of the source and destination transfer sizes. All source reads and destination writes must be configured to the natural boundary of the programmed transfer size respectively.

In fixed-arbitration mode, a configuration error is generated when any two channel priority levels are equal and any channel is activated. The ERRCHN field is undefined for this type of error. All channel priority levels must be unique before any service requests are made.

If a scatter-gather operation is enabled on channel completion, a configuration error is reported if the scatter-gather address (DLAST_SGA) is not aligned on a 32-byte boundary. If minor loop channel linking is enabled on channel completion, a configuration error is reported when the link is attempted if bit EDMA_TCD[CITER.E_LINK] is not equal to bit EDMA_TCD[BITER.E_LINK]. All configuration error conditions except scatter-gather and minor loop link error are reported as the channel is activated and assert an error interrupt request if enabled. When properly enabled, a scatter-gather configuration error is reported when the scatter-gather operation begins at major loop completion. A minor loop channel link configuration error is reported when the link operation is serviced at minor loop completion.

If a system bus read or write is terminated with an error, the data transfer is immediately stopped and the appropriate bus error flag is set. In this case, the state of the channel's transfer control descriptor is updated by the DMA engine with the current source address, destination address, and minor loop byte count at the point of the fault. If a bus error occurs on the last read prior to beginning the write sequence, the write is executed using the data captured during the bus error. If a bus error occurs on the last write prior to switching to the next read sequence, the read sequence is executed before the channel is terminated due to the destination bus error.

A transfer may be cancelled by software via the bit EDMA_CR[CX]. When a cancel transfer request is recognized, the eDMA engine stops processing the channel. The current read-write sequence is allowed to finish. If the cancel occurs on the last read-write sequence of a major or minor loop, the cancel request is discarded and the channel retires normally.

The error cancel transfer is the same as a cancel transfer except the DMAES register is updated with the cancelled channel number and error cancel bit is set. The TCD of a cancelled channel has the source address and destination address of the last transfer saved in the TCD. It is the responsibility of the user to initialize the TCD again should the channel need to be restarted because the aforementioned fields have been modified by the eDMA engine and no longer represent the original parameters. When a transfer is cancelled via the error cancel transfer mechanism (setting EDMA_CR[ECX]), the channel number is loaded into field EDMA_ESR[ERRCHN] and the bits EDMA_ESR[ECX] and EDMA_ESR[VLD] are set. In addition, an error interrupt may be generated if enabled. Refer to *Section , eDMA Error Registers (EDMA_ERH, EDMA_ERL)*.

The occurrence of any type of error causes the DMA engine to stop the active channel and the appropriate channel bit in the eDMA error register to be asserted. At the same time, the details of the error condition are loaded into the EDMA_ESR. The major loop complete indicators, setting the transfer control descriptor DONE flag and the possible assertion of an interrupt request, are not affected when an error is detected. After the error status has been updated, the DMA engine continues to operate by servicing the next appropriate channel. A channel that experiences an error condition is not automatically disabled. If a channel is terminated by an error and then issues another service request before the error is fixed, that channel will execute and terminate with the same error condition.

Figure 21. eDMA Error Status Register (EDMA_ESR)

Offset: EDMA_BASE + 0x0004 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VLD	0	0	0	0	0	0	0	0	0	0	0	0	0	0	ECX
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	GPE	CPE	ERRCHN						SAE	SOE	DAE	DOE	NCE	SGE	SBE	DBE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 25. EDMA_ESR field descriptions

Field	Description
VLD	Valid Bit Logical OR of all EDMA_ERL status bits. 0 No EDMA_ER bits are set. 1 At least one EDMA_ER bit is set indicating a valid error exists that has not been cleared.
ECX	Transfer canceled 0 No canceled transfers 1 The last recorded entry was a canceled transfer via the error cancel transfer input.

Table 25. EDMA_ESR field descriptions (continued)

Field	Description
GPE	Group-priority error 0 No group-priority error 1 The last recorded error was a configuration error among the group priorities indicating not all group priorities are unique.
CPE	Channel-Priority Error 0 No channel-priority error 1 The last recorded error was a configuration error in the channel priorities within a group, indicating not all channel priorities within a group are unique.
ERRCHN	Error Channel Number or Canceled Channel Number Channel number of the last recorded error (excluding GPE and CPE errors) or last recorded transfer that was error cancelled. Do not rely on the number in the ERRCHN field group for channel-priority errors. Group- and Channel-priority errors must be resolved by inspection. The application code must interrogate the priority registers to find groups or channels with duplicate priority level.
SAE	Source Address Error 0 No source address configuration error 1 The last recorded error was a configuration error detected in field EDMA_TCD[SADDR], indicating EDMA_TCD[SADDR] is inconsistent with EDMA_TCD[SSIZE].
SOE	Source Offset Error 0 No source offset configuration error 1 The last recorded error was a configuration error detected in field EDMA_TCD[SOFF], indicating EDMA_TCD[SOFF] is inconsistent with EDMA_TCD[SSIZE].
DAE	Destination Address Error 0 No destination address configuration error 1 The last recorded error was a configuration error detected in field EDMA_TCD[DADDR], indicating EDMA_TCD[DADDR] is inconsistent with EDMA_TCD[DSIZE].
DOE	Destination Offset Error 0 No destination offset configuration error 1 The last recorded error was a configuration error detected in field EDMA_TCD[DOFF], indicating EDMA_TCD[DOFF] is inconsistent with EDMA_TCD[DSIZE].
NCE	NBYTES/CITER Configuration Error 0 No NBYTES/CITER configuration error 1 The last recorded error was a configuration error detected in fields EDMA_TCD[NBYTES] or EDMA_TCD[CITER], indicating the following conditions exist: – EDMA_TCD[NBYTES] is not a multiple of EDMA_TCD[SSIZE] and EDMA_TCD[DSIZE], or – EDMA_TCD[CITER] is equal to zero, or – EDMA_TCD[CITER.E_LINK] is not equal to EDMA_TCD[BITER.E_LINK].
SGE	Scatter-Gather Configuration Error 0 No scatter-gather configuration error 1 The last recorded error was a configuration error detected in field EDMA_TCD[DLAST_SGA], indicating EDMA_TCD[DLAST_SGA] is not on a 32-byte boundary. This field is checked at the beginning of a scatter-gather operation after major loop completion if EDMA_TCD[E_SG] is enabled.

Table 25. EDMA_ESR field descriptions (continued)

Field	Description
SBE	Source Bus Error 0 No source bus error 1 The last recorded error was a bus error on a source read.
DBE	Destination Bus Error 0 No destination bus error 1 The last recorded error was a bus error on a destination write.

eDMA Enable Request Registers (EDMA_ERQRH, EDMA_ERQRL)

The EDMA_ERQRH and EDMA_ERQRL provide a bitmap for the 32 (or 64 for eDMA) channels to enable the request signal for each channel. EDMA_ERQRH supports channels 63–32, while EDMA_ERQRL covers channels 31–0.

The state of any given channel enable is directly affected by writes to these registers; the state is also affected by writes to the EDMA_SERQR and EDMA_CERQR. The EDMA_CERQR and EDMA_SERQR are provided so that the request enable for a single channel can be modified without performing a read-modify-write sequence to the EDMA_ERQRH and EDMA_ERQRL.

Both the eDMA request input signal and this enable request flag must be asserted before a channel's hardware service request is accepted. The state of the eDMA enable request flag does not effect a channel service request made through software or a linked channel request.

Figure 22. eDMA Enable Request High Register (EDMA_ERQRH)

Address: EDMA_BASE + 0x0008 Access: User R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	ERQ63	ERQ62	ERQ61	ERQ60	ERQ59	ERQ58	ERQ57	ERQ56	ERQ55	ERQ54	ERQ53	ERQ52	ERQ51	ERQ50	ERQ49	ERQ48
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	ERQ47	ERQ46	ERQ45	ERQ44	ERQ43	ERQ42	ERQ41	ERQ40	ERQ39	ERQ38	ERQ37	ERQ36	ERQ35	ERQ34	ERQ33	ERQ32
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 23. eDMA Enable Request Register (EDMA_ERQRL)

Offset: EDMA_BASE + 0x000C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	ERQ31	ERQ30	ERQ29	ERQ28	ERQ27	ERQ26	ERQ25	ERQ24	ERQ23	ERQ22	ERQ21	ERQ20	ERQ19	ERQ18	ERQ17	ERQ16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	19	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	ERQ15	ERQ14	ERQ13	ERQ12	ERQ11	ERQ10	ERQ09	ERQ08	ERQ07	ERQ06	ERQ05	ERQ04	ERQ03	ERQ02	ERQ01	ERQ00
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 26. EDMA_ERQRL field descriptions

Field	Description
ERQ*n*	Enable eDMA Hardware Service Request *n* 0 The eDMA request signal for channel *n* is disabled. 1 The eDMA request signal for channel *n* is enabled.

As a given channel completes processing its major iteration count, there is a flag in the transfer control descriptor that may affect the ending state of the EDMA_ERQR bit for that channel. If bit EDMA_TCD[D_REQ] is set, then the corresponding EDMA_ERQR bit is cleared after the major loop is complete, disabling the eDMA hardware request. Otherwise if the D_REQ bit is cleared, the state of the EDMA_ERQR bit is unaffected.

eDMA Enable Error Interrupt Registers (EDMA_EEIRH, EDMA_EEIRL)

The EDMA_EEIRH and EDMA_EEIRL provide a bitmap for the 32 (or 64 for eDMA) channels to enable the error interrupt signal for each channel. EDMA_EEIRH supports channels 63–32, while EDMA_EEIRL covers channels 31–0.

The state of any given channel’s error interrupt enable is directly affected by writes to these registers; it is also affected by writes to the EDMA_SEEIR and EDMA_CEEIR. The EDMA_SEEIR and EDMA_CEEIR are provided so that the error interrupt enable for a single channel can be modified without the performing a read-modify-write sequence to the EDMA_EEIRH and EDMA_EEIRL.

Both the eDMA error indicator and this error interrupt enable flag must be asserted before an error interrupt request for a given channel is asserted.

Figure 24. eDMA Enable Error Interrupt High Register (EDMA_EEIRH)

Address: EDMA_BASE + 0x0010 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	EEI63	EEI62	EEI61	EEI60	EEI59	EEI58	EEI57	EEI56	EEI55	EEI54	EEI53	EEI52	EEI51	EEI50	EEI49	EEI48
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	EEI47	EEI46	EEI45	EEI44	EEI43	EEI42	EEI41	EEI40	EEI39	EEI38	EEI37	EEI36	EEI35	EEI34	EEI33	EEI32
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 25. eDMA Enable Error Interrupt Low Register (EDMA_EEIRL)

Address: EDMA_BASE + 0x0014 Access: User R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	EEI31	EEI30	EEI29	EEI28	EEI27	EEI26	EEI25	EEI24	EEI23	EEI22	EEI21	EEI20	EEI19	EEI18	EEI17	EEI16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	EEI15	EEI14	EEI13	EEI12	EEI11	EEI10	EEI09	EEI08	EEI07	EEI06	EEI05	EEI04	EEI03	EEI02	EEI01	EEI00
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 27. EDMA_EEIRL field descriptions

Field	Description
EEI*n*	Enable Error Interrupt *n* 0 The error signal for channel *n* does not generate an error interrupt. 1 The assertion of the error signal for channel *n* generate an error interrupt request.

eDMA Set Enable Request Register (EDMA_SERQR)

The EDMA_SERQR provides a simple memory-mapped mechanism to set a given bit in the EDMA_ERQRH or EDMA_ERQRL to enable the eDMA request for a given channel. The data value on a register write causes the corresponding bit in the EDMA_ERQRH or EDMA_ERQRL to be set. Setting bit 1 (SERQ[0]) provides a global set function, forcing the entire contents of EDMA_ERQRH and EDMA_ERQRL to be asserted. Reads of this register return all zeroes.

If bit 0 is set, the SERQ command is ignored. This allows multiple byte registers to be written as a 32-bit word. Reads of this register return all zeroes.

Figure 26. eDMA Set Enable Request Register (EDMA_SERQR)

Offset: EDMA_BASE + 0x0018 Access: User write-only

	0	1	2	3	4	5	6	7
R								
W	NOP	SERQ[0:6]						
Reset	0	0	0	0	0	0	0	0

Table 28. EDMA_SERQR field descriptions

Field	Descriptions
0 NOP	No operation 0 Normal operation. 1 No operation, ignore bits 1–7.
1–7 SERQ[0:6]	Set Enable Request 0–32 (64 for eDMA) Set corresponding bit in EDMA_ERQRH or EDMA_ERQRL. 64–127 Set all bits in EDMA_ERQRH and EDMA_ERQRL.

eDMA Clear Enable Request Register (EDMA_CERQR)

The EDMA_CERQR provides a simple memory-mapped mechanism to clear a given bit in the EDMA_ERQRH or EDMA_ERQRL to disable the DMA request for a given channel. The data value on a register write causes the corresponding bit in the EDMA_ERQRH or EDMA_ERQRL to be cleared. Setting bit 1 (CERQ[0]) provides a global clear function, forcing the entire contents of EDMA_ERQRH and EDMA_ERQRL to be zeroed, disabling all eDMA request inputs. Reads of this register return all zeroes.

If bit 0 is set, the CERQ command is ignored. This allows multiple byte registers to be written as a 32-bit word. Reads of this register return all zeroes.

Figure 27. eDMA Clear Enable Request Register (EDMA_CERQR)

Offset: EDMA_BASE + 0x0019 Access: User write-only

	0	1	2	3	4	5	6	7
R								
W	NOP	CERQ[0:6]						
Reset	0	0	0	0	0	0	0	0

Table 29. EDMA_CERQR field descriptions

Field	Description
0 NOP	No operation 0 Normal operation 1 No operation, ignore bits 1–7.
1–7 CERQ[0:6]	Clear Enable Request 0–32 (64 for eDMA) Clear corresponding bit in EDMA_ERQRH or EDMA_ERQRL. 64–127 Clear all bits in EDMA_ERQRH and EDMA_ERQRL.

eDMA Set Enable Error Interrupt Register (EDMA_SEEIR)

The EDMA_SEEIR provides a memory-mapped mechanism to set a given bit in the EDMA_EEIRH or EDMA_EEIRL to enable the error interrupt for a given channel. The data value on a register write causes the corresponding bit in the EDMA_EEIRH or EDMA_EEIRL to be set. Setting bit 1 (SEEI[0]) provides a global set function, forcing the entire contents of EDMA_EEIRH or EDMA_EEIRL to be asserted. Reads of this register return all zeroes.

If bit 0 is set, the SEEI command is ignored. This allows multiple byte registers to be written as a 32-bit word. Reads of this register return all zeroes.

Figure 28. eDMA Set Enable Error Interrupt Register (EDMA_SEEIR)

Offset: EDMA_BASE + 0x001A Access: User write-only

	0	1	2	3	4	5	6	7
R								
W	NOP	SEEI[0:6]						
Reset	0	0	0	0	0	0	0	0

Table 30. EDMA_SEEIR field descriptions

Field	Description
0 NOP	No operation 0 Normal operation 1 No operation, ignore bits 1–7.
1–7 SEEI[0:6]	Set Enable Error Interrupt 0–32 (64 for eDMA) Set corresponding bit in EDMA_EIRRH or EDMA_EIRRL. 64–127 Set all bits in EDMA_EIRRH or EDMA_EEIRL.

eDMA Clear Enable Error Interrupt Register (EDMA_CEEIR)

The EDMA_CEEIR provides a memory-mapped mechanism to clear a given bit in the EDMA_EEIRH or EDMA_EEIRL to disable the error interrupt for a given channel. The data value on a register write causes the corresponding bit in the EDMA_EEIRH or EDMA_EEIRL to be cleared. Setting bit 1 (CEEI[0]) provides a global clear function, forcing the entire contents of the EDMA_EEIRH or EDMA_EEIRL to be zeroed, disabling error interrupts for all channels. Reads of this register return all zeroes.

If bit 0 is set, the CEEI command is ignored. This allows multiple byte registers to be written as a 32-bit word. Reads of this register return all zeroes.

Figure 29. eDMA Clear Enable Error Interrupt Register (EDMA_CEEIR)

Offset: EDMA_BASE + 0x001B Access: User write-only

	0	1	2	3	4	5	6	7
R								
W	NOP	CEEI[0:6]						
Reset	0	0	0	0	0	0	0	0

Table 31. EDMA_CEEIR field descriptions

Field	Description
NOP	No operation 0 Normal operation 1 No operation, ignore bits 1-7.
CEEI[0:6]	Clear Enable Error Interrupt 0–32 (64 for eDMA) Clear corresponding bit in EDMA_EEIRH or EDMA_EEIRL. 64–127 Clear all bits in EDMA_EEIRH or EDMA_EEIRL.

eDMA Clear Interrupt Request Register (EDMA_CIRQR)

The EDMA_CIRQR provides a memory-mapped mechanism to clear a given bit in the EDMA_IRQRH or EDMA_IRQRL to disable the interrupt request for a given channel. The given value on a register write causes the corresponding bit in the EDMA_IRQRH or EDMA_IRQRL to be cleared. Setting bit 1 (CINT[0]) provides a global clear function, forcing the entire contents of the EDMA_IRQRH or EDMA_IRQRL to be zeroed, disabling all eDMA interrupt requests. Reads of this register return all zeroes.

If bit 0 is set, the CINT command is ignored. This allows multiple byte registers to be written as a 32-bit word. Reads of this register return all zeroes.

Figure 30. eDMA Clear Interrupt Request (EDMA_CIRQR)

Offset: EDMA_BASE + 0X001C

Access: User write-only

	0	1	2	3	4	5	6	7
R								
W	NOP	CINT[0:6]						
Reset	0	0	0	0	0	0	0	0

Table 32. EDMA_CIRQR field descriptions

Field	Description
NOP	No operation 0 Normal operation 1 No operation, ignore bits 1–7.
CINT[0:6]	Clear Interrupt Request 0–32 (64 for eDMA) Clear corresponding bit in EDMA_IRQRH or EDMA_IRQRL. 64–127 Clear all bits in EDMA_IRQRH or EDMA_IRQRL.

eDMA Clear Error Register (EDMA_CER)

The EDMA_CER provides a memory-mapped mechanism to clear a given bit in the EDMA_ERH or EDMA_ERL to disable the error condition flag for a given channel. The given value on a register write causes the corresponding bit in the EDMA_ERH or EDMA_ERL to be cleared. Setting bit 1 (CERR[0]) provides a global clear function, forcing the entire contents of the EDMA_ERH or EDMA_ERL to be zeroed, clearing all channel error indicators. Reads of this register return all zeroes.

If bit 0 is set, the CERR command is ignored. This allows multiple byte registers to be written as a 32-bit word. Reads of this register return all zeroes.

Figure 31. eDMA Clear Error Register (EDMA_CER)

Offset: EDMA_BASE + 0x001D Access: User write-only

	0	1	2	3	4	5	6	7
R								
W	NOP	CERR[0:6]						
Reset	0	0	0	0	0	0	0	0

Table 33. EDMA_CER field descriptions

Field	Description
NOP	No operation 0 Normal operation 1 No operation, ignore bits 1–7.
CERR[0:6]	Clear Error Indicator 0–32 (64 for eDMA) Clear corresponding bit in EDMA_ERH or EDMA_ERL. 64–127 Clear all bits in EDMA_ERH or EDMA_ERL.

eDMA Set START Bit Register (EDMA_SSBR)

The EDMA_SSBR provides a memory-mapped mechanism to set the START bit in the TCD of the given channel. The data value on a register write causes the START bit in the corresponding transfer control descriptor to be set. Setting bit 1 (SSB[0]) provides a global set function, forcing all START bits to be set. Reads of this register return all zeroes.

If bit 0 is set, the SSB command is ignored. This allows multiple byte registers to be written as a 32-bit word. Reads of this register return all zeroes.

Figure 32. eDMA Set START Bit Register (EDMA_SSBR)

Offset: EDMA_BASE + 0x001E Access: User write-only

	0	1	2	3	4	5	6	7
R								
W	NOP	SSB[0:6]						
Reset	0	0	0	0	0	0	0	0

Table 34. EDMA_SSBR field descriptions

Field	Description
NOP	No operation 0 Normal operation 1 No operation, ignore bits 1–7.
SSB[0:6]	Set START Bit (channel service request) 0–32 (64 for eDMA) Set the corresponding channel’s TCD START bit. 64–127 Set all TCD START bits.

eDMA Clear DONE Status Bit Register (EDMA_CDSBR)

The EDMA_CDSBR provides a memory-mapped mechanism to clear the DONE bit in the TCD of the given channel. The data value on a register write causes the DONE bit in the corresponding transfer control descriptor to be cleared. Setting bit 1 (CDSB[0]) provides a global clear function, forcing all DONE bits to be cleared.

If bit 0 is set, the CDSB command is ignored. This allows multiple byte registers to be written as a 32-bit word. Reads of this register return all zeroes.

Figure 33. eDMA Clear DONE Status Bit Register (EDMA_CDSBR)

Offset: EDMA_BASE + 0x001F

Access: User write-only

	0	1	2	3	4	5	6	7
R								
W	NOP	CDSB[0:6]						
Reset	0	0	0	0	0	0	0	0

Table 35. EDMA_CDSBR field descriptions

Field	Description
NOP	No operation 0 Normal operation 1 No operation, ignore bits 1–7.
CDSB[0:6]	Clear DONE Status Bit 0–32 (64 for eDMA) Clear the corresponding channel's DONE bit. 64–127 Clear all TCD DONE bits.

eDMA Interrupt Request Registers (EDMA_IRQRH, EDMA_IRQRL)

The EDMA_IRQRH and EDMA_IRQRL provides a bitmap for the 32 channels signaling the presence of an interrupt request for each channel. EDMA_IRQRH maps to channels 63–32 and EDMA_IRQRL maps to channels 31–0.

The DMA engine signals the occurrence of a programmed interrupt on the completion of a data transfer as defined in the transfer control descriptor by setting the appropriate bit in this register. The outputs of this register are directly routed to the interrupt controller (INTC). During the execution of the interrupt service routine associated with any given channel, software must clear the appropriate bit, negating the interrupt request. Typically, a write to the EDMA_CIRQR in the interrupt service routine is used for this purpose.

The state of any given channel's interrupt request is directly affected by writes to this register; it is also affected by writes to the EDMA_CIRQR. On writes to the EDMA_IRQRH or EDMA_IRQRL, a 1 in any bit position clears the corresponding channel's interrupt request. A 0 in any bit position has no effect on the corresponding channel's current interrupt status. The EDMA_CIRQR is provided so the interrupt request for a single channel can be cleared without performing a read-modify-write sequence to the EDMA_IRQRH and EDMA_IRQRL.

Figure 34. eDMA Interrupt Request High Register (EDMA_IRQRH)

Address: EDMA_BASE + 0x0020 Access: User R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	INT63	INT62	INT61	INT60	INT59	INT58	INT57	INT56	INT55	INT54	INT53	INT52	INT51	INT50	INT49	INT48
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	INT47	INT46	INT45	INT44	INT43	INT42	INT41	INT40	INT39	INT38	INT37	INT36	INT35	INT34	INT33	INT32
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 35. eDMA Interrupt Request Register (EDMA_IRQRL)

Address: EDMA_BASE + 0x0024 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	INT31	INT30	INT29	INT28	INT27	INT26	INT25	INT24	INT23	INT22	INT21	INT20	INT19	INT18	INT17	INT16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	19	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	INT15	INT14	INT13	INT12	INT11	INT10	INT09	INT08	INT07	INT06	INT05	INT04	INT03	INT02	INT01	INT00
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 36. EDMA_IRQRL field descriptions

Field	Description
0–31 INT*n*	eDMA Interrupt Request *n* 0 The interrupt request for channel *n* is cleared. 1 The interrupt request for channel *n* is active.

eDMA Error Registers (EDMA_ERH, EDMA_ERL)

Register EDMA_ERH and EDMA_ERL provide a bitmap for the 32 channels signaling the presence of an error for each channel. EDMA_ERH supports channels 63–32 (for eDMA) and EDMA_ERL maps to channels 31-0.

The DMA engine signals the occurrence of a error condition by setting the appropriate bit in this register. The outputs of this register are enabled by the contents of the EDMA_EEIR, then logically summed across 32 (64 for eDMA) channels to form an error interrupt request, which is then routed to the interrupt controller. During the execution of the interrupt service routine associated with any eDMA errors, it is software’s responsibility to clear the appropriate bit, negating the error interrupt request. Typically, a write to the EDMA_CER in the interrupt service routine is used for this purpose. The normal eDMA channel completion indicators, setting the transfer control descriptor DONE flag and the possible assertion of an interrupt request, are not affected when an error is detected.

The contents of this register can also be polled and a non-zero value indicates the presence of a channel error, regardless of the state of the EDMA_EEIR. Bit EDMA_ESR[VLD] is a logical OR of all bits in this register and it provides a single bit indication of any errors. The state of any given channel’s error indicators is affected by writes to this register; it is also affected by writes to the EDMA_CER. On writes to EDMA_ERH or EDMA_ERL, a ‘1’ in any bit position clears the corresponding channel’s error status. A ‘0’ in any bit position has no effect on the corresponding channel’s current error status. The EDMA_CER is provided so the error indicator for a single channel can be cleared.

Figure 36. eDMA Error High Register (EDMA_ERH)

Address: EDMA_BASE + 0x0028 Access: User R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	ERR63	ERR62	ERR61	ERR60	ERR59	ERR58	ERR57	ERR56	ERR55	ERR54	ERR53	ERR52	ERR51	ERR50	ERR49	ERR48
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	ERR47	ERR46	ERR45	ERR44	ERR43	ERR42	ERR41	ERR40	ERR39	ERR38	ERR37	ERR36	ERR35	ERR34	ERR33	ERR32
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 37. eDMA Error Register (EDMA_ERL)

Address: EDMA_BASE + 0x002C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	ERR31	ERR30	ERR29	ERR28	ERR27	ERR26	ERR25	ERR24	ERR23	ERR22	ERR21	ERR20	ERR19	ERR18	ERR17	ERR16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	ERR15	ERR14	ERR13	ERR12	ERR11	ERR10	ERR09	ERR08	ERR07	ERR06	ERR05	ERR04	ERR03	ERR02	ERR01	ERR00
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 37. EDMA_ERL field descriptions

Field	Description
0–31 ERR*n*	eDMA Error *n* 0 An error in channel *n* has not occurred. 1 An error in channel *n* has occurred.

DMA Hardware Request Status Registers (EDMA_HRSH, EDMA_HRSL)

Registers EDMA_HRSH and EDMA_HRSL provide a bitmap for the implemented channels (32 or 64) to show the current hardware request status for each channel. EDMA_HRSH maps to channels 64–32 and EDMA_HRSL maps to channels 31-00.

Address: EDMA_BASE + 0x0030 Access: User R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	HRS63	HRS62	HRS61	HRS60	HRS59	HRS58	HRS57	HRS56	HRS55	HRS54	HRS53	HRS52	HRS51	HRS50	HRS49	HRS48
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	HRS47	HRS46	HRS45	HRS44	HRS43	HRS42	HRS41	HRS40	HRS39	HRS38	HRS37	HRS36	HRS35	HRS34	HRS33	HRS32
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 38. EDMA Hardware Request Status Register High (EDMA_HRSH)

Address: EDMA_BASE + 0x0034 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	HRS31	HRS30	HRS29	HRS28	HRS27	HRS26	HRS25	HRS24	HRS23	HRS22	HRS21	HRS20	HRS19	HRS18	HRS17	HRS16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	HRS15	HRS14	HRS13	HRS12	HRS11	HRS10	HRS09	HRS08	HRS07	HRS06	HRS05	HRS04	HRS03	HRS02	HRS01	HRS00
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 39. EDMA Hardware Request Status Register Low (EDMA_HRSL)

Table 38. EDMA_HRSL field descriptions

Field	Description
0–31 HRS*n*	DMA Hardware Request Status 0 A hardware service request for channel *n* is not present. 1 A hardware service request for channel *n* is present. The hardware request status reflects the state of the request as seen by the arbitration logic. Therefore, this status is affected by bit EDMA_ERQRL[ERQ*n*].

eDMA Channel *n* Priority Registers (EDMA_CPR*n*)

When the fixed-priority channel arbitration mode is enabled (EDMA_CR[ERCA] = 0), the contents of these registers define the unique priorities associated with each channel. The channel priorities are evaluated by numeric value; that is, 0 is the lowest priority, 1 is the next higher priority, then 2, 3, etc. If software modifies channel priority values, then the software must ensure that the channel priorities contain unique values. Otherwise, a configuration error is reported. The range of the priority value is limited to the values of 0 through 15. When read, the GRPPRI bits of the EDMA_CPR*n* register reflect the current priority level of the group of channels in which the corresponding channel resides. GRPPRI bits are not affected by writes to the EDMA_CPR*n* registers. The group priority is assigned in the EDMA_CR. See *Figure 20* and *Table 24* for the EDMA_CR definition.

Channel pre-emption is enabled on a per-channel basis by setting the ECP bit in the EDMA_CPR*n* register. Channel pre-emption allows the executing channel's data transfers to be temporarily suspended in favor of starting a higher priority channel. After the pre-empting channel has completed all its minor loop data transfers, the pre-empted channel is restored and resumes execution. After the restored channel completes one read/write sequence, it is again eligible for pre-emption. If any higher priority channel requests service, the restored channel is suspended and the higher priority channel is serviced. Nested pre-emption (attempting to pre-empt a pre-empting channel) is not supported. After a pre-empting channel begins execution, it cannot be pre-empted. Pre-emption is available only when fixed arbitration is selected for both group and channel arbitration modes.

A channel's ability to pre-empt another channel can be disabled by setting EDMA_CPR[DPA]. When a channel's pre-empt ability is disabled, that channel cannot suspend a lower priority channel's data transfer; regardless of the lower priority channel's ECP setting. This allows for a pool of low priority, large data moving channels to be defined. These low priority channels can be configured to not pre-empt each other, thus preventing a low priority channel from consuming the pre-empt slot normally available a true, high priority channel.

Figure 40. eDMA Channel *n* Priority Register (EDMA_CPR*n*)

Address: EDMA_BASE + 0x0100 + *n*

Access: User read/write

	0	1	2	3	4	5	6	7
R	ECP	DPA	GRPPRI		CHPRI			
W								
Reset	0	0	0	0	—[1]			

1. The reset value for the channel priority field, CHPRI[0–3], is equal to the corresponding channel number for each priority register; that is, EDMA_CPRI0[CHPRI] = 0b0000 and EDMA_CPR15[CHPRI] = 0b1111.

Table 39. EDMA_CPRn field descriptions

Field	Description
ECP	Enable Channel Pre-emption 0 Channel *n* cannot be suspended by a higher priority channel's service request. 1 Channel *n* can be temporarily suspended by the service request of a higher priority channel.
DPA	Disable pre-empt ability 0 Channel n can suspend a lower priority channel. 1 Channel n cannot suspend any channel, regardless of channel priority.

Table 39. EDMA_CPRn field descriptions (continued)

Field	Description
GRPPRI[0:1]	Channel *n* current group priority Group priority assigned to this channel group when fixed-priority arbitration is enabled. These two bits are read-only; writes are ignored. The reset value for the group priority fields, is equal to the corresponding channel number for each priority register; that is, EDMA_CPR31[GRPPRI] = 0b01.
CHPRI[0:3]	Channel *n* Arbitration Priority Channel priority when fixed-priority arbitration is enabled. The reset value for the channel priority fields CHPRI[0–3], is equal to the corresponding channel number for each priority register; that is, EDMA_CPR31[CHPRI] = 0b1111.

Transfer control descriptor (TCD)

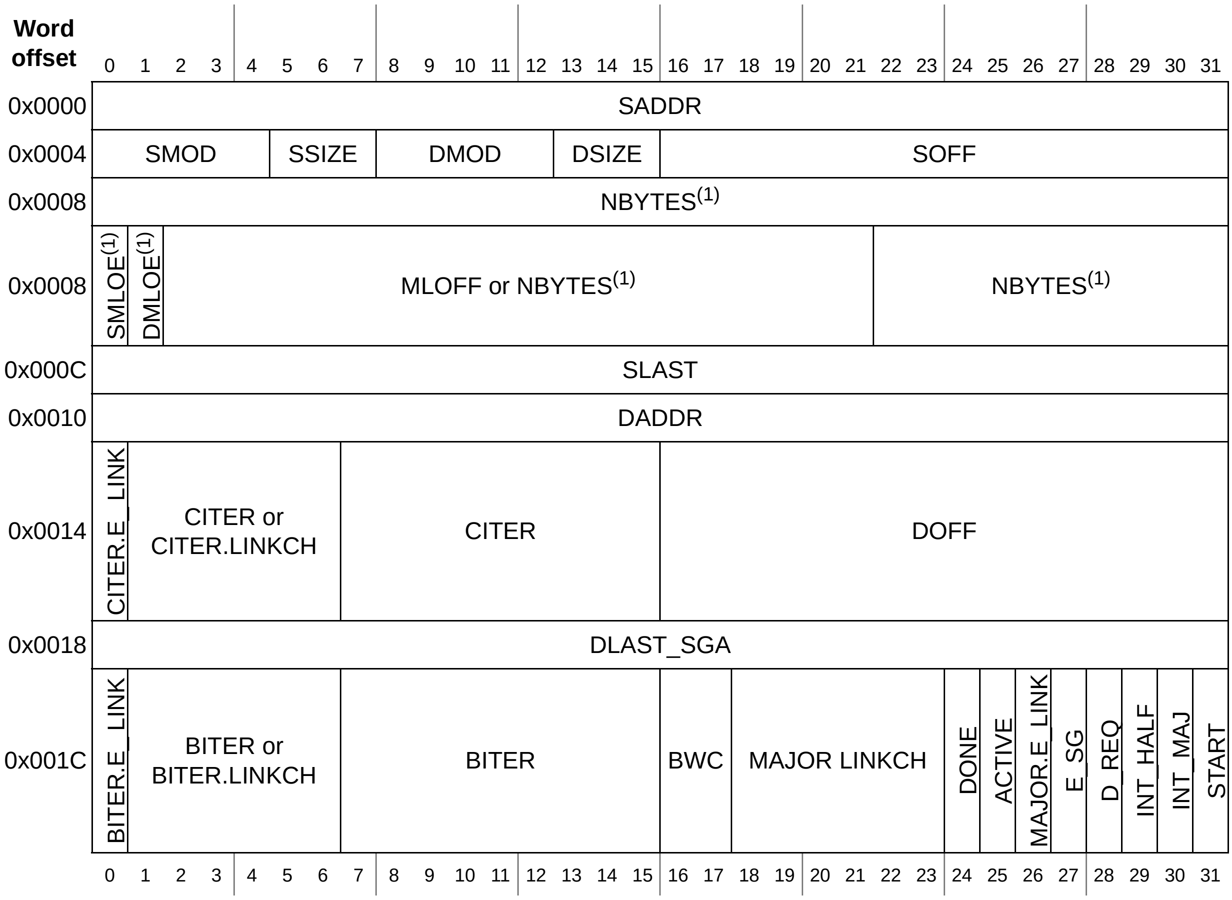
Each channel requires a 32-byte transfer control descriptor for defining the desired data movement operation. The channel descriptors are stored in the local memory in sequential order: channel 0, channel 1,... channel 31 (63 for eDMA). The definitions of the TCD are presented as eight 32-bit values. *Table 40* is a field list of the basic TCD structure.

Table 40. TCD*n* 32-bit memory structure

eDMA offset	TCD*n* field	
0x1000+(32 x *n*)+0x0000	Source address (saddr)	
0x1000+(32 x *n*)+0x0004	Transfer attributes	Signed source address offset (soff)
0x1000+(32 x *n*)+0x0008	Inner minor byte count (nbytes)	
0x1000+(32 x *n*)+0x000C	Last source address adjustment (slast)	
0x1000+(32 x *n*)+0x0010	Destination address (daddr)	
0x1000+(32 x *n*)+0x0014	Current major iteration count (citer)	Signed destination address offset (doff)
0x1000 (32 x *n*) 0x0018	Last destination address adjustment / scatter-gather address (dlast_sga)	
0x1000+(32 x *n*)+0x001c	Beginning major iteration count (biter)	Channel control/status

Figure 41 and *Table 41* define the fields of the TCD*n* structure.

Figure 41. TCD structure



1. The fields implemented in Word 2 depend on whether EDMA_CR(EMLM) is set to ‘0’ or ‘1’. Refer to *Table 24*.

Note: *The TCD structures for the eDMA channels shown in Figure 41 are implemented in internal SRAM. These structures are not initialized at reset; therefore, all channel TCD parameters must be initialized by the application code before activating that channel.*

Table 41. TCDn field descriptions

Bits / Word offset [n:n]	Name	Description
0–31 / 0x0 [0:31]	SADDR [0:31]	Source address Memory address pointing to the source data. Word 0x0, bits 0–31.
32–36 / 0x4 [0:4]	SMOD [0:4]	Source address modulo 0 Source address modulo feature is disabled. non-0 This value defines a specific address range that is specified to be the value after SADDR + SOFF calculation is performed or the original register value. The setting of this field provides the ability to easily implement a circular data queue. For data queues requiring power-of-2 size bytes, the queue should start at a 0-modulo-size address and the SMOD field should be set to the appropriate value for the queue, freezing the desired number of upper address bits. The value programmed into this field specifies the number of lower address bits that are allowed to change. For this circular queue application, the SOFF is typically set to the transfer size to implement post-increment addressing with the SMOD function constraining the addresses to a 0-modulo-size range.
37–39 / 0x4 [5:7]	SSIZE [0:2]	Source data transfer size 000 8-bit 001 16-bit 010 32-bit 011 64-bit 100 Reserved 101 32-byte (64-bit, 4 beat, WRAP4 burst) 110 Reserved 111 Reserved The attempted specification of a reserved encoding causes a configuration error.
40–44 / 0x4 [8:12]	DMOD [0:4]	Destination address modulo See the SMOD[0:5] definition.
45–47 / 0x4 [13:15]	DSIZE [0:2]	Destination data transfer size See the SSIZE[0:2] definition.
48–63 / 0x4 [16:31]	SOFF [0:15]	Source address signed offset Sign-extended offset applied to the current source address to form the next-state value as each source read is completed.
64 0x8 [0]	SMLOE 0	Source minor loop offset enable This flag selects whether the minor loop offset is applied to the source address upon minor loop completion. 0 The minor loop offset is not applied to the saddr. 1 The minor loop offset is applied to the saddr.

Table 41. TCDn field descriptions (continued)

Bits / Word offset [n:n]	Name	Description
65 0x8 [1]	DMLOE 1	Destination minor loop offset enable This flag selects whether the minor loop offset is applied to the destination address upon minor loop completion. 0 The minor loop offset is not applied to the daddr. 1 The minor loop offset is applied to the daddr.
66–85 0x8 [2-21]	MLOFF or NBYTES [0:19]	Inner “minor” byte transfer count or Minor loop offset If both SMLOE and DMLOE are cleared, this field is part of the byte transfer count. If either SMLOE or DMLOE are set, this field represents a sign-extended offset applied to the source or destination address to form the next-state value after the minor loop is completed.
86–95 / 0x8 [22:31]	NBYTES	Inner “minor” byte transfer count Number of bytes to be transferred in each service request of the channel. As a channel is activated, the contents of the appropriate TCD is loaded into the eDMA engine, and the appropriate reads and writes performed until the complete byte transfer count has been transferred. This is an indivisible operation and cannot be stalled or halted. Once the minor count is exhausted, the current values of the SADDR and DADDR are written back into the local memory, the major iteration count is decremented and restored to the local memory. If the major iteration count is completed, additional processing is performed. The NBYTES value of 0x0000_0000 is interpreted as 0x1_0000_0000, thus specifying a 4 GB transfer.
96–127 / 0xC [0:31]	SLAST [0:31]	Last source address adjustment Adjustment value added to the source address at the completion of the outer major iteration count. This value can be applied to “restore” the source address to the initial value, or adjust the address to reference the next data structure.
128–159 / 0x10 [0:31]	DADDR [0:31]	Destination address Memory address pointing to the destination data.
160 / 0x14 [0]	CITER.E_LINK	Enable channel-to-channel linking on minor loop completion As the channel completes the inner minor loop, this flag enables the linking to another channel, defined by CITER.LINKCH[0:5]. The link target channel initiates a channel service request via an internal mechanism that sets the bit EDMA_TCD[START] of the specified channel. If channel linking is disabled, the CITER value is extended to 15 bits in place of a link channel number. If the major loop is exhausted, this link mechanism is suppressed in favor of the MAJOR.E_LINK channel linking. 0 The channel-to-channel linking is disabled. 1 The channel-to-channel linking is enabled. This bit must be equal to the BITER.E_LINK bit. Otherwise, a configuration error is reported.

Table 41. TCDn field descriptions (continued)

Bits / Word offset [n:n]	Name	Description
161–166 / 0x14 [1:6]	CITER [0:5] or CITER.LINKCH [0:5]	Current major iteration count or link channel number If channel-to-channel linking is disabled (EDMA_TCD[CITER.E_LINK] = 0), then – No channel-to-channel linking (or chaining) is performed after the inner minor loop is exhausted. TCD bits [161:175] are used to form a 15-bit CITER field. Otherwise, – After the minor loop is exhausted, the DMA engine initiates a channel service request at the channel defined by CITER.LINKCH[0:5] by setting that channel's EDMA_TCD[START] bit.
167–175 / 0x14 [7:15]	CITER [6:14]	Current major iteration count This 9 or 15-bit count represents the current major loop count for the channel. It is decremented each time the minor loop is completed and updated in the transfer control descriptor memory. After the major iteration count is exhausted, the channel performs a number of operations (for example, final source and destination address calculations), optionally generating an interrupt to signal channel completion before reloading the CITER field from the beginning iteration count (BITER) field. When the CITER field is initially loaded by software, it must be set to the same value as that contained in the BITER field. If the channel is configured to execute a single service request, the initial values of BITER and CITER should be 0x0001.
176–191 / 0x14 [16:31]	DOFF [0:15]	Destination address signed Offset Sign-extended offset applied to the current destination address to form the next-state value as each destination write is completed.
192–223 / 0x18 [0:31]	DLAST_SGA [0:31]	Last destination address adjustment or the memory address for the next transfer control descriptor to be loaded into this channel (scatter-gather). If scatter-gather processing for the channel is disabled (EDMA_TCD[E_SG] = 0) then – Adjustment value added to the destination address at the completion of the outer major iteration count. This value can be applied to restore the destination address to the initial value, or adjust the address to reference the next data structure. Otherwise, – This address points to the beginning of a 0-modulo-32 byte region containing the next transfer control descriptor to be loaded into this channel. This channel reload is performed as the major iteration count completes. The scatter-gather address must be 0-modulo-32 byte, otherwise a configuration error is reported.

Table 41. TCDn field descriptions (continued)

Bits / Word offset [n:n]	Name	Description
224 / 0x1C [0]	BITER.E_LINK	Enables channel-to-channel linking on minor loop complete As the channel completes the inner minor loop, this flag enables the linking to another channel, defined by BITER.LINKCH[0:5]. The link target channel initiates a channel service request via an internal mechanism that sets the bit EDMA_TCD[START] of the specified channel. If channel linking is disabled, the BITER value is extended to 15 bits in place of a link channel number. If the major loop is exhausted, this link mechanism is suppressed in favor of the MAJOR.E_LINK channel linking. 0 The channel-to-channel linking is disabled. 1 The channel-to-channel linking is enabled. When the TCD is first loaded by software, this field must be set equal to the corresponding CITER field. Otherwise, a configuration error is reported. As the major iteration count is exhausted, the contents of this field is reloaded into the CITER field.
225–230 / 0x1C [1:6]	BITER [0:5] or BITER.LINKCH[0:5]	Starting major iteration count or link channel number If channel-to-channel linking is disabled (EDMA_TCD[BITER.E_LINK] = 0), then – No channel-to-channel linking (or chaining) is performed after the inner minor loop is exhausted. TCD bits [225:239] are used to form a 15-bit BITER field. Otherwise, – After the minor loop is exhausted, the DMA engine initiates a channel service request at the channel, defined by BITER.LINKCH[0:5], by setting that channel’s EDMA_TCD[START] bit. When the TCD is first loaded by software, this field must be set equal to the corresponding CITER field. Otherwise, a configuration error is reported. As the major iteration count is exhausted, the contents of this field is reloaded into the CITER field.
231–239 / 0x1C [7:15]	BITER [6:14]	Starting major iteration count As the transfer control descriptor is first loaded by software, this field must be equal to the value in the CITER field. As the major iteration count is exhausted, the contents of this field are reloaded into the CITER field. If the channel is configured to execute a single service request, the initial values of BITER and CITER should be 0x0001.
240–241 / 0x1C [16:17]	BWC [0:1]	Bandwidth control This two-bit field provides a mechanism to effectively throttle the amount of bus bandwidth consumed by the eDMA. In general, as the eDMA processes the inner minor loop, it continuously generates read/write sequences until the minor count is exhausted. This field forces the eDMA to stall after the completion of each read/write access to control the bus request bandwidth seen by the system bus crossbar switch (XBAR). 00 No DMA engine stalls 01 Reserved 10 DMA engine stalls for 4 cycles after each r/w 11 DMA engine stalls for 8 cycles after each r/w

Table 41. TCDn field descriptions (continued)

Bits / Word offset [n:n]	Name	Description
242–247 / 0x1C [18:23]	MAJOR.LINKCH [0:5]	Link channel number If channel-to-channel linking on major loop complete is disabled (EDMA_TCD[MAJOR.E_LINK] = 0) then, – No channel-to-channel linking (or chaining) is performed after the outer major loop counter is exhausted. Otherwise – After the major loop counter is exhausted, the DMA engine initiates a channel service request at the channel defined by MAJOR.LINKCH[0:5] by setting that channel's EDMA_TCD[START] bit.
248 / 0x1C [24]	DONE	Channel done This flag indicates the eDMA has completed the outer major loop. It is set by the DMA engine as the CITER count reaches zero; it is cleared by software or hardware when the channel is activated (when the DMA engine has begun processing the channel, not when the first data transfer occurs). This bit must be cleared to write the MAJOR.E_LINK or E_SG bits.
249 / 0x1C [25]	ACTIVE	Channel active This flag signals the channel is currently in execution. It is set when channel service begins, and is cleared by the DMA engine as the inner minor loop completes or if any error condition is detected.
250 / 0x1C [26]	MAJOR.E_LINK	Enable channel-to-channel linking on major loop completion As the channel completes the outer major loop, this flag enables the linking to another channel, defined by MAJOR.LINKCH[0:5]. The link target channel initiates a channel service request via an internal mechanism that sets bit EDMA_TCD[START] of the specified channel. To support the dynamic linking coherency model, this field is forced to zero when written to while the bit EDMA_TCD[DONE] is set. 0 The channel-to-channel linking is disabled. 1 The channel-to-channel linking is enabled.
251 / 0x1C [27]	E_SG	Enable scatter-gather processing As the channel completes the outer major loop, this flag enables scatter-gather processing in the current channel. If enabled, the DMA engine uses DLAST_SGA as a memory pointer to a 0-modulo-32 address containing a 32-byte data structure which is loaded as the transfer control descriptor into the local memory. To support the dynamic scatter-gather coherency model, this field is forced to zero when written to while the bit EDMA_TCD[DONE] is set. 0 The current channel's TCD is normal format. 1 The current channel's TCD specifies a scatter gather format. The DLAST_SGA field provides a memory pointer to the next TCD to be loaded into this channel after the outer major loop completes its execution.

Table 41. TCDn field descriptions (continued)

Bits / Word offset [n:n]	Name	Description
252 / 0x1C [28]	D_REQ	Disable hardware request If this flag is set, the eDMA hardware automatically clears the corresponding EDMA_ERQH or EDMA_ERQL bit when the current major iteration count reaches zero. 0 The channel's EDMA_ERQH or EDMA_ERQL bit is not affected. 1 The channel's EDMA_ERQH or EDMA_ERQL bit is cleared when the outer major loop is complete.
253 / 0x1C [29]	INT_HALF	Enable an interrupt when major counter is half complete If this flag is set, the channel generates an interrupt request by setting the appropriate bit in the EDMA_ERQH or EDMA_ERQL when the current major iteration count reaches the halfway point. Specifically, the comparison performed by the eDMA engine is (CITER == (BITER >> 1)). This halfway point interrupt request is provided to support double-buffered (aka ping-pong) schemes, or other types of data movement where the processor needs an early indication of the transfer's progress. CITER = BITER = 1 with INT_HALF enabled will generate an interrupt as it satisfies the equation (CITER == (BITER >> 1)) after a single activation. 0 The half-point interrupt is disabled. 1 The half-point interrupt is enabled.
254 / 0x1C [30]	INT_MAJ	Enable an interrupt when major iteration count completes If this flag is set, the channel generates an interrupt request by setting the appropriate bit in the EDMA_ERQH or EDMA_ERQL when the current major iteration count reaches zero. 0 The end-of-major loop interrupt is disabled. 1 The end-of-major loop interrupt is enabled.
255 / 0x1C [31]	START	Channel start If this flag is set the channel is requesting service. The eDMA hardware automatically clears this flag after the channel begins execution. 0 The channel is not explicitly started. 1 The channel is explicitly started via a software initiated service request.

8.4 Functional description

This section provides an overview of the microarchitecture and functional operation of the eDMA block.

The eDMA module is partitioned into two major modules: the DMA engine and the transfer control descriptor local memory. The DMA engine is further partitioned into four submodules, which are detailed below.

- DMA engine
  - Address path: This module implements registered versions of two channel transfer control descriptors: channel x and channel y, and is responsible for all the master bus address calculations. All the implemented channels provide the same functionality. This hardware structure allows the data transfers associated with one channel to be pre-empted after the completion of a read/write sequence if a higher

priority channel service request is asserted while the first channel is active. After a channel is activated, it runs until the minor loop is completed unless pre-empted by a higher priority channel. This capability provides a mechanism (optionally enabled by EDMA_CPR*n*[ECP]) where a large data move operation can be pre-empted to minimize the time another channel is blocked from execution.

  - When another channel is activated, the contents of its transfer control descriptor is read from the local memory and loaded into the registers of the other address path channel{x,y}. After the inner minor loop completes execution, the address path hardware writes the new values for the TCD*n*.{SADDR, DADDR, CITER} back into the local memory. If the major iteration count is exhausted, additional processing is performed, including the final address pointer updates, reloading the TCDn.CITER field, and a possible fetch of the next TCD*n* from memory as part of a scatter-gather operation.
  - Data path: This module implements the actual bus master read/write datapath. It includes 32 bytes of register storage (matching the maximum transfer size) and the necessary mux logic to support any required data alignment. The system read data bus is the primary input, and the system write data bus is the primary output.
  - The address and data path modules directly support the two-stage pipelined system bus. The address path module represents the 1st stage of the bus pipeline (the address phase), while the data path module implements the second stage of the pipeline (the data phase).
  - Program model/channel arbitration: This module implements the first section of eDMA's programming model and also the channel arbitration logic. The programming model registers are connected to the slave bus (not shown). The eDMA peripheral request inputs and eDMA interrupt request outputs are also connected to this module (via the control logic).
  - Control: This module provides all the control functions for the DMA engine. For data transfers where the source and destination sizes are equal, the DMA engine performs a series of source read, destination write operations until the number of bytes specified in the inner minor loop byte count has been moved.

    A minor loop interaction is defined as the number of bytes to transfer (*n*bytes) divided by the transfer size. Transfer size is defined as:

    ```
    if (SSIZE < DSIZE)
    transfer size = destination transfer size (# of bytes)
    else
    transfer size = source transfer size (# of bytes)
    ```

    Minor loop TCD variables are SOFF, SMOD, DOFF, DMOD, NBYTES, SADDR, DADDR, BWC, ACTIVE, AND START. Major loop TCD variables are DLAST, SLAST, CITER, BITER, DONE, D_REQ, INT_MAJ, MAJOR_LNKCH, and INT_HALF.

    For descriptors where the sizes are not equal, multiple access of the smaller size data are required for each reference of the larger size. For example, if the source size references 16-bit data and the destination is 32-bit data, two reads are performed, then one 32-bit write.

- TCD local memory
  - Memory controller: This logic implements the required dual-ported controller, handling accesses from both the DMA engine as well as references from the slave bus. As noted earlier, in the event of simultaneous accesses, the DMA engine is

given priority and the slave transaction is stalled. The hooks to a BIST controller for the local TCD memory are included in this module.

- Memory array: The TCD is implemented using a single-ported, synchronous compiled RAM memory array.

8.4.1 eDMA basic data flow

The eDMA transfers data based on a two-deep, nested flow. The basic flow of a data transfer can be partitioned into three segments. As shown in *Figure 42*, the first segment involves the channel service request. In the diagram, this example uses the assertion of the eDMA peripheral request signal to request service for channel *n*. Channel service request via software and the TCDn.START bit follows the same basic flow as an eDMA peripheral request. The eDMA peripheral request input signal is registered internally and then routed to through the DMA engine, first through the control module, then into the program model/channel arbitration module. In the next cycle, the channel arbitration is performed using the fixed-priority or round-robin algorithm. After the arbitration is complete, the activated channel number is sent through the address path and converted into the required address to access the TCD local memory. Next, the TCD memory is accessed and the required descriptor read from the local memory and loaded into the DMA engine address path channel{x,y} registers. The TCD memory is organized 64-bits in width to minimize the time needed to fetch the activated channel’s descriptor and load it into the eDMA engine address path channel{x,y} registers.

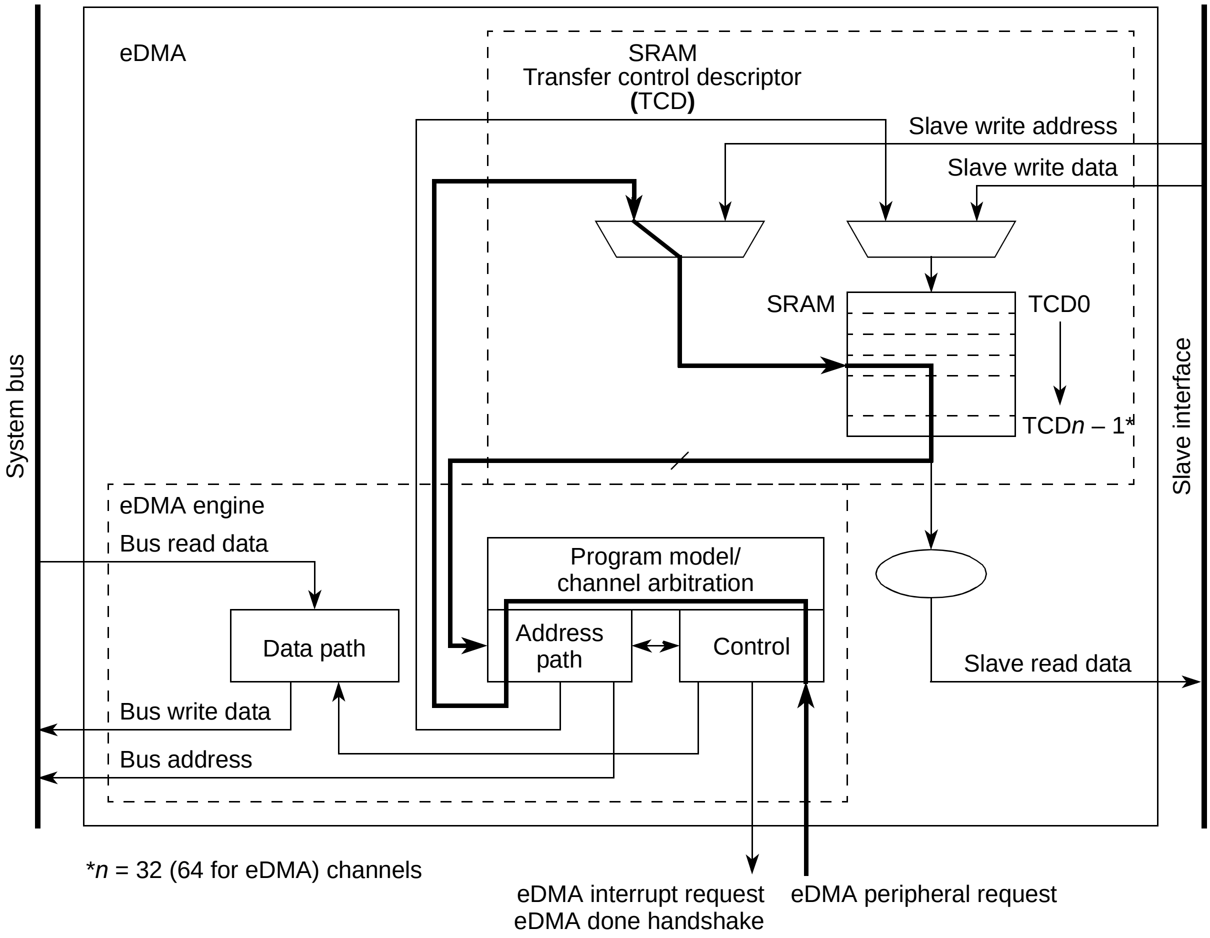


Figure 42. eDMA operation, Part 1

In the second part of the basic data flow as shown in *Figure 43*, the modules associated with the data transfer (address path, data path, and control) sequence through the required source reads and destination writes to perform the actual data movement. The source reads are initiated and the fetched data is temporarily stored in the data path module until it is gated onto the system bus during the destination write. This source read/destination write processing continues until the inner minor byte count has been transferred. The eDMA done handshake signal is asserted at the end of the minor byte count transfer.

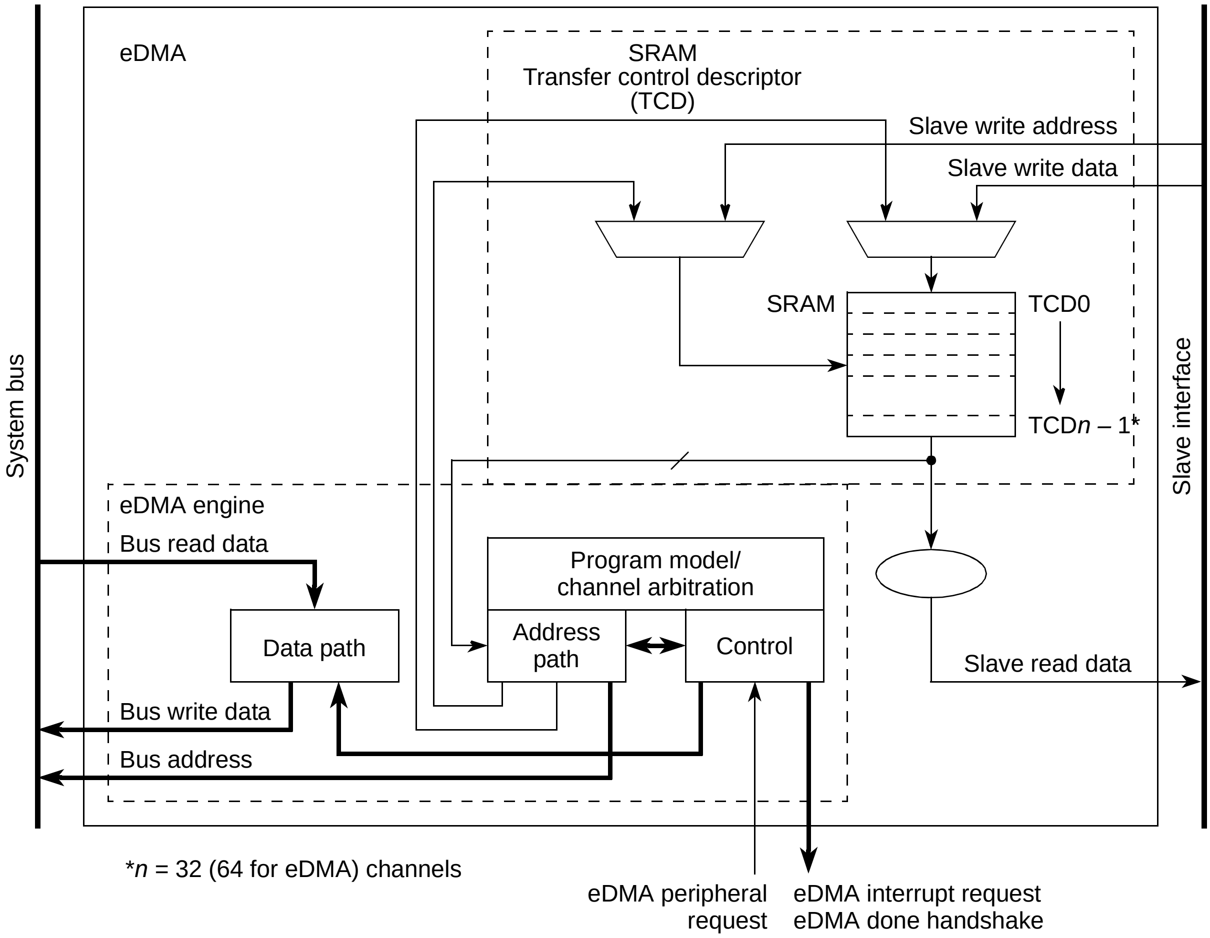


Figure 43. eDMA operation, Part 2

After the inner minor byte count has been moved, the final phase of the basic data flow is performed. In this segment, the address path logic performs the required updates to certain fields in the channel’s TCD; for example, SADDR, DADDR, CITER. If the outer major iteration count is exhausted, then there are additional operations performed. These include the final address adjustments and reloading of the BITER field into the CITER. Additionally, assertion of an optional interrupt request occurs at this time, as does a possible fetch of a new TCD from memory using the scatter-gather address pointer included in the descriptor. The updates to the TCD memory and the assertion of an interrupt request are shown in *Figure 44*.

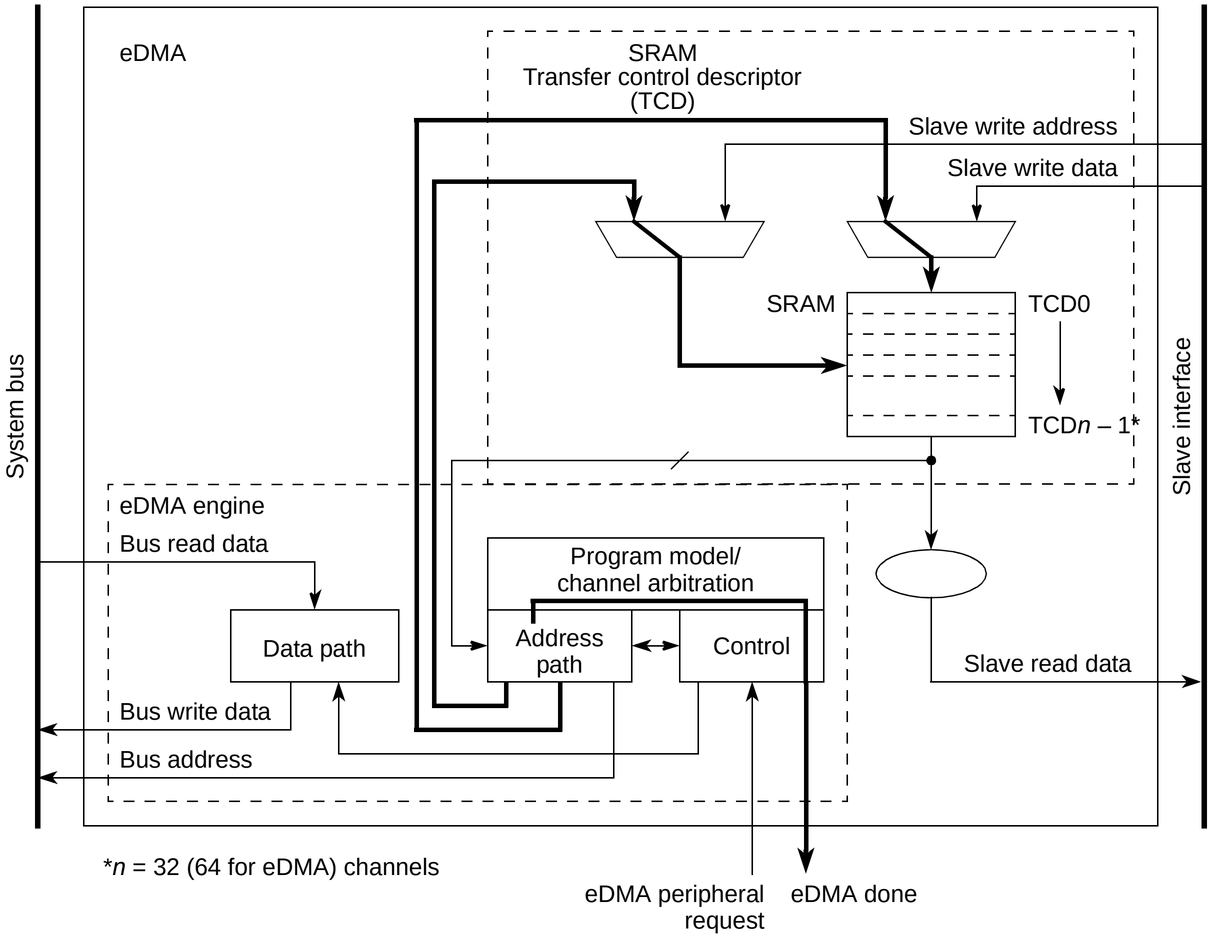


Figure 44. eDMA operation, Part 3

8.5 Initialization / Application information

8.5.1 eDMA initialization

A typical initialization of the eDMA has the following sequence:

1. Write the EDMA_CR if a configuration other than the default is desired.
2. Write the channel priority levels into the EDMA_CPR*n* registers if a configuration other than the default is desired.
3. Enable error interrupts in the EDMA_EEIRL and/or EDMA_EEIRH registers if desired.
4. Write the 32-byte TCD for each channel that may request service.
5. Enable any hardware service requests via the EDMA_ERQRH and/or EDMA_ERQRL registers.
6. Request channel service by software (setting bit EDMA_TCD[START]) or by hardware (slave device asserting its DMA peripheral request signal).

After any channel requests service, a channel is selected for execution based on the arbitration and priority levels written into the programmer's model. The DMA engine reads the entire TCD, including the primary transfer control parameter shown in *Table 42*, for the selected channel into its internal address path module. As the TCD is being read, the first transfer is initiated on the system bus unless a configuration error is detected. Transfers from the source (as defined by the source address, EDMA_TCD[SADDR]) to the destination (as defined by the destination address, EDMA_TCD.DADDR) continue until the specified number of bytes (EDMA_TCD[NBYTES]) have been transferred. When the transfer is complete, the DMA engine's local EDMA_TCD[SADDR], EDMA_TCD.DADDR, and EDMA_TCD.CITER are written back to the main TCD memory and any minor loop channel linking is performed, if enabled. If the major loop is exhausted, further post processing is executed; for example, interrupts, major loop channel linking, and scatter-gather operations, if enabled.

Table 42. TCD primary control and status fields

TCD field name	Description
START	Control bit to start channel when using a software initiated DMA service (Automatically cleared by hardware)
ACTIVE	Status bit indicating the channel is currently in execution
DONE	Status bit indicating major loop completion (cleared by software when using a software initiated DMA service)
D_REQ	Control bit to disable DMA request at end of major loop completion when using a hardware-initiated DMA service
BWC	Control bits for throttling bandwidth control of a channel
E_SG	Control bit to enable scatter-gather feature
INT_HALF	Control bit to enable interrupt when major loop is half complete
INT_MAJ	Control bit to enable interrupt when major loop completes

Figure 45 shows how each DMA request initiates one minor loop transfer (iteration) without CPU intervention. DMA arbitration can occur after each minor loop, and one level of minor loop DMA pre-emption is allowed. The number of minor loops in a major loop is specified by the beginning iteration count (biter).

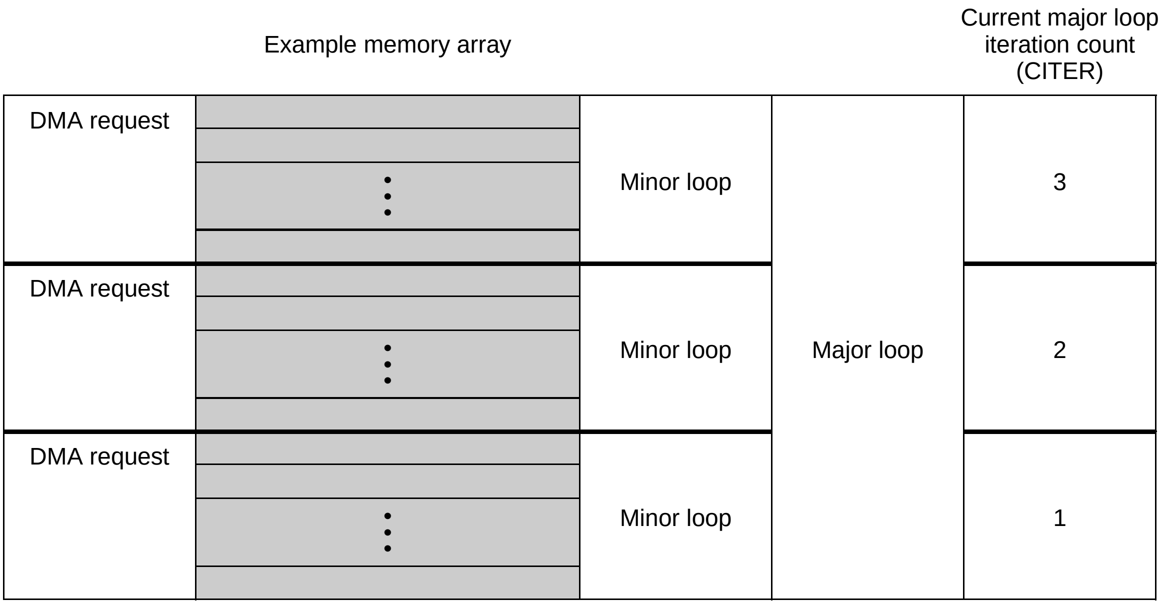


Figure 45. Example of multiple loop iterations

Figure 46 lists the memory array terms and how the TCD settings interrelate.

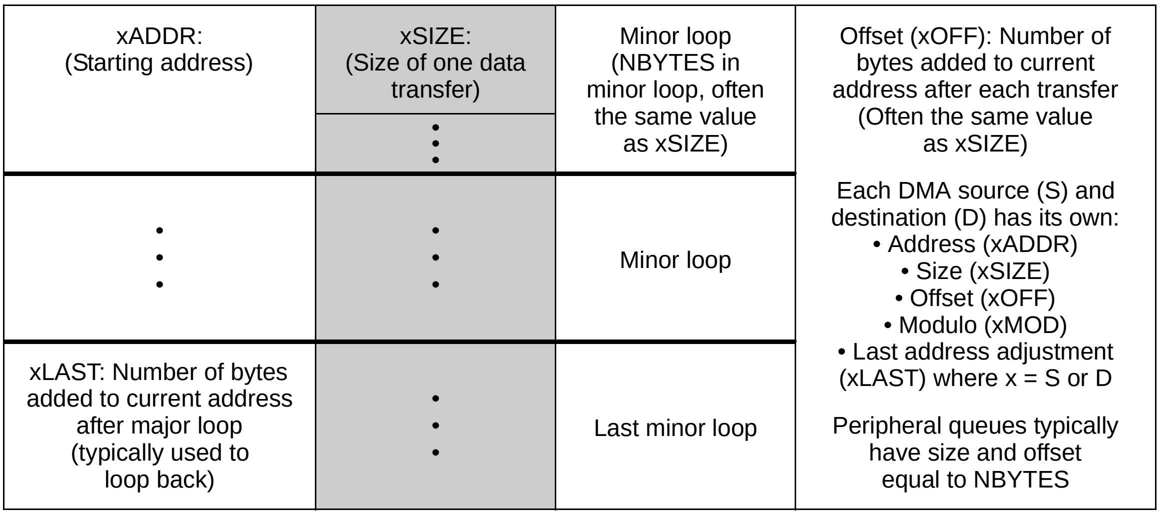


Figure 46. Memory array terms

8.5.2 DMA programming errors

The DMA performs various tests on the transfer control descriptor to verify consistency in the descriptor data. Most programming errors are reported on a per-channel basis with the exception of two errors: group-priority error and channel-priority error, or EDMA_ESR[GPE] and EDMA_ESR[CPE], respectively.

For all error types other than group- or channel-priority errors, the channel number causing the error is recorded in the EDMA_ESR. If the error source is not removed before the next activation of the problem channel, the error is detected and recorded again.

Channel-priority errors are identified within a group after that group has been selected as the active group. For example, all of the channel priorities in group 1 are unique, but some of the channel priorities in group 0 are the same:

1. The DMA is configured for fixed-group and fixed-channel arbitration modes.
2. Group 1 is the highest priority and all channels are unique in that group.
3. Group 0 is the next highest priority and has two channels with the same priority level.
4. If group 1 has any service requests, those requests are executed.
5. After all of group 1 requests have completed, group 0 becomes the next active group.
6. If group 0 has a service request, then an undefined channel in group 0 is selected and a channel-priority error will occur.
7. This repeats until the all of group 0 requests have been removed or a higher priority group 1 request comes in.

In this sequence, for item 2, the DMA acknowledge lines assert only if the selected channel is requesting service via the DMA peripheral request signal. If interrupts are enabled for all channels, the user receives an error interrupt, but the channel number for the EDMA_ER and the error interrupt request line are undetermined because they reflect the undefined channel. A group-priority error is global and any request in any group causes a group-priority error.

If priority levels are not unique, the highest (channel/group) priority that has an active request is selected, but the lowest numbered (channel/group) with that priority is selected by arbitration and executed by the DMA engine. The hardware service request handshake signals, error interrupts, and error reporting are associated with the selected channel.

8.5.3 DMA request assignments

The assignments between the DMA requests from the modules to the channels of the eDMA are shown in *Table 43*. The source column is written in C language syntax. The syntax is module_instance.register[bit].

Table 43. DMA request summary for eDMA

DMA request	Channel	Source	Description
eQADC_FISR0_CFFF0	0	EQADC.FISR0[CFFF0]	eQADC Command FIFO 0 Fill Flag
eQADC_FISR0_RFDF0	1	EQADC.FISR0[RFDF0]	eQADC Receive FIFO 0 Drain Flag
eQADC_FISR1_CFFF1	2	EQADC.FISR1[CFFF1]	eQADC Command FIFO 1 Fill Flag
eQADC_FISR1_RFDF1	3	EQADC.FISR1[RFDF1]	eQADC Receive FIFO 1 Drain Flag
eQADC_FISR2_CFFF2	4	EQADC.FISR2[CFFF2]	eQADC Command FIFO 2 Fill Flag
eQADC_FISR2_RFDF2	5	EQADC.FISR2[RFDF2]	eQADC Receive FIFO 2 Drain Flag
eQADC_FISR3_CFFF3	6	EQADC.FISR3[CFFF3]	eQADC Command FIFO 3 Fill Flag
eQADC_FISR3_RFDF3	7	EQADC.FISR3[RFDF3]	eQADC Receive FIFO 3 Drain Flag

Table 43. DMA request summary for eDMA (continued)

DMA request	Channel	Source	Description
eQADC_FISR4_CFFF4	8	EQADC.FISR4[CFFF4]	eQADC Command FIFO 4 Fill Flag
eQADC_FISR4_RFDF4	9	EQADC.FISR4[RFDF4]	eQADC Receive FIFO 4 Drain Flag
eQADC_FISR5_CFFF5	10	EQADC.FISR5[CFFF5]	eQADC Command FIFO 5 Fill Flag
eQADC_FISR5_RFDF5	11	EQADC.FISR5[RFDF5]	eQADC Receive FIFO 5 Drain Flag
DSPIB_SR_TFFF	12	DSPIB.SR[TFFF]	DSPIB Transmit FIFO Fill Flag
DSPIB_SR_RFDF	13	DSPIB.SR[RFDF]	DSPIB Receive FIFO Drain Flag
DSPIC_SR_TFFF	14	DSPIC.SR[TFFF]	DSPIC Transmit FIFO Fill Flag
DSPIC_SR_RFDF	15	DSPIC.SR[RFDF]	DSPIC Receive FIFO Drain Flag
DSPID_SR_TFFF	16	DSPID.SR[TFFF]	DSPID Transmit FIFO Fill Flag
DSPID_SR_RFDF	17	DSPID.SR[RFDF]	DSPID Receive FIFO Drain Flag
eSCIA_COMBTX	18	ESCIA.SR[TDRE] || ESCIA.SR[TC] || ESCIA.SR[TXRDY]	eSCIA combined DMA request of the Transmit Data Register Empty, Transmit Complete, and LIN Transmit Data Ready DMA requests
eSCIA_COMBRX	19	ESCIA.SR[RDRF] || ESCIA.SR[RXRDY]	eSCIA combined DMA request of the Receive Data Register Full and LIN Receive Data Ready DMA requests
eMIOS_GFR_F0	20	EMIOS.GFR[F0]	eMIOS channel 0 Flag
eMIOS_GFR_F1	21	EMIOS.GFR[F1]	eMIOS channel 1 Flag
eMIOS_GFR_F2	22	EMIOS.GFR[F2]	eMIOS channel 2 Flag
eMIOS_GFR_F3	23	EMIOS.GFR[F3]	eMIOS channel 3 Flag
eMIOS_GFR_F4	24	EMIOS.GFR[F4]	eMIOS channel 4 Flag
eMIOS_GFR_F8	25	EMIOS.GFR[F8]	eMIOS channel 8 Flag
eMIOS_GFR_F9	26	EMIOS.GFR[F9]	eMIOS channel 9 Flag
eTPU_CDTRSR_A_DTRS0	27	ETPU.CDTRSR_A[DTRS0]	eTPUA Channel 0 Data Transfer Request Status
eTPU_CDTRSR_A_DTRS1	28	ETPU.CDTRSR_A[DTRS1]	eTPUA Channel 1 Data Transfer Request Status
eTPU_CDTRSR_A_DTRS2	29	ETPU.CDTRSR_A[DTRS2]	eTPUA Channel 2 Data Transfer Request Status
eTPU_CDTRSR_A_DTRS14	30	ETPU.CDTRSR_A[DTRS14]	eTPUA Channel 14 Data Transfer Request Status
eTPU_CDTRSR_A_DTRS15	31	ETPU.CDTRSR_A[DTRS15]	eTPUA Channel 15 Data Transfer Request Status
No Request	32	—	
No Request	33	—	

Table 43. DMA request summary for eDMA (continued)

DMA request	Channel	Source	Description
eSCIB_COMBTX	34	ESCIB.SR[TDRE] || ESCIB.SR[TC] || ESCIB.SR[TXRDY]	eSCIB combined DMA request of the Transmit Data Register Empty, Transmit Complete, and LIN Transmit Data Ready DMA requests
eSCIB_COMBRX	35	ESCIB.SR[RDRF] || ESCIB.SR[RXRDY]	eSCIB combined DMA request of the Receive Data Register Full and LIN Receive Data Ready DMA requests
eMIOS_GFR_F6	36	EMIOS.GFR[F6]	eMIOS channel 6 Flag
eMIOS_GFR_F7	37	EMIOS.GFR[F7]	eMIOS channel 7 Flag
eMIOS_GFR_F10	38	EMIOS.GFR[F10]	eMIOS channel 10 Flag
eMIOS_GFR_F11	39	EMIOS.GFR[F11]	eMIOS channel 11 Flag
eMIOS_GFR_F16	40	EMIOS.GFR[F16]	eMIOS channel 16 Flag
eMIOS_GFR_F17	41	EMIOS.GFR[F17]	eMIOS channel 17 Flag
eMIOS_GFR_F18	42	EMIOS.GFR[F18]	eMIOS channel 18 Flag
eMIOS_GFR_F19	43	EMIOS.GFR[F19]	eMIOS channel 19 Flag
eTPU_CDTRSR_A_DTRS12	44	ETPU.CDTRSR_A[DTRS12]	eTPUA Channel 12 Data Transfer Request Status
eTPU_CDTRSR_A_DTRS13	45	ETPU.CDTRSR_A[DTRS13]	eTPUA Channel 13 Data Transfer Request Status
eTPU_CDTRSR_A_DTRS28	46	ETPU.CDTRSR_A[DTRS28]	eTPUA Channel 28 Data Transfer Request Status
eTPU_CDTRSR_A_DTRS29	47	ETPU.CDTRSR_A[DTRS29]	eTPUA Channel 29 Data Transfer Request Status
SIU_EISR_EIF0	48	SIU.SIU_EISR[EIF0]	SIU External Interrupt Flag 0
SIU_EISR_EIF1	49	SIU.SIU_EISR[EIF1]	SIU External Interrupt Flag 1
SIU_EISR_EIF2	50	SIU.SIU_EISR[EIF2]	SIU External Interrupt Flag 2
SIU_EISR_EIF3	51	SIU.SIU_EISR[EIF3]	SIU External Interrupt Flag 3
DECFIL_FILL_BUF_A	52	DECFIL_A.DECFILTER_IB[INPBUF]	Decimation Filter A Fill Buffer
DECFIL_DRAIN_BUF_A	53	DECFIL_A.DECFILTER_OB[OUTBUF]	Decimation Filter A Drain Buffer
DECFIL_FILL_BUF_B	54	DECFIL_B.DECFILTER_IB[INPBUF]	Decimation Filter B Fill Buffer
DECFIL_DRAIN_BUF_B	55	DECFIL_B.DECFILTER_OB[OUTBUF]	Decimation Filter B Drain Buffer
eSCIC_COMBTX	56	ESCIC.SR[TDRE] || ESCIC.SR[TC] || ESCIC.SR[TXRDY]	eSCIC combined DMA request of the Transmit Data Register Empty, Transmit Complete, and LIN Transmit Data Ready DMA requests

Table 43. DMA request summary for eDMA (continued)

DMA request	Channel	Source	Description
eSCIC_COMBRX	57	ESCIB.SR[RDRF] || ESCIB.SR[RXRDY]	eSCIC combined DMA request of the Receive Data Register Full and LIN Receive Data Ready DMA requests
No Request	58-63	—	

8.5.4 DMA arbitration mode considerations

Fixed-group arbitration, fixed-channel arbitration

In this mode, the channel service request from the highest priority channel in the highest priority group is selected to execute. If the eDMA is programmed so the channels within one group use fixed priorities, and that group is assigned the highest fixed priority of all groups, it is possible for that group to take all the bandwidth of the eDMA controller. That is, no other groups can be serviced if there is always at least one DMA request pending on a channel in the highest priority group when the controller arbitrates the next DMA request. The advantage of this scenario is that latency can be small for channels that need to be serviced quickly. pre-emption is available in this scenario only.

Round-robin group arbitration, fixed-channel arbitration

When one or more DMA requests arrive from one or more groups, the channel with the highest priority from a specific group is serviced first. Groups are serviced starting with the highest group number with a service request and rotating through to the lowest group number containing a service request.

After the channel request is serviced, the group round robin algorithm selects the highest pending request from the next group in the round-robin sequence. Servicing continues round robin, always servicing the highest priority channel in the next group in the sequence, or skipping a group if it has no pending requests.

If a channel requests service at a rate that equals or exceeds the round robin service rate, then that channel is always serviced before lower priority channels in the same group, and the lower priority channels are never serviced. The advantage of this scenario is that no one group can consume all the eDMA bandwidth. The highest priority channel selection latency is potentially greater than fixed/fixed arbitration. Excessive request rates on high-priority channels can prevent the servicing of lower priority channels in the same group.

Round-robin group arbitration, round-robin channel arbitration

Groups are serviced as described in *Section , Round-robin group arbitration, fixed-channel arbitration* but this time channels are serviced in channel number order. One channel only is serviced from each requesting group for each round robin pass through the groups.

Within each group, channels are serviced starting with the highest channel number and rotating through to the lowest channel number without regard to channel priority levels.

Because channels are serviced in round-robin manner, any channel that generates DMA requests faster than a combination of the group round-robin service rate and the channel service rate for its group does not prevent the servicing of other channels in its group. Any DMA requests that are not serviced are simply lost, but at least one channel gets serviced.

This scenario ensures that all channels are guaranteed service at some point, regardless of the request rates. However, the potential latency could be high. All channels are treated equally. Priority levels are not used in round-robin/round-robin mode.

Fixed-group arbitration, round-robin channel arbitration

The highest priority group with a request is serviced. Lower priority groups are serviced if no pending requests exist in the higher priority groups.

Within each group, channels are serviced starting with the highest channel number and rotating through to the lowest channel number without regard to the channel priority levels assigned within the group.

This scenario could cause the same bandwidth consumption problem as indicated in *Section , Fixed-group arbitration, fixed-channel arbitration*but all the channels in the highest priority group get serviced. Service latency is short on the highest priority group, but could potentially get longer and longer as the group priority decreases.

8.5.5 DMA transfer

Single request

To perform a simple transfer of *n* bytes of data with one activation, set the major loop to ‘1’ (EDMA_TCD[CITER] = EDMA_TCD[BITER] = 1). The data transfer begins after the channel service request is acknowledged and the channel is selected to execute. After the transfer is complete, bit EDMA_TCD[DONE] is set and an interrupt is generated if properly enabled.

For example, the following TCD entry is configured to transfer 16 bytes of data. The eDMA is programmed for one iteration of the major loop transferring 16 bytes per iteration. The source memory has a byte wide memory port located at 0x1000. The destination memory has a word wide port located at 0x2000. The address offsets are programmed in increments to match the size of the transfer; one byte for the source and four bytes for the destination. The final source and destination addresses are adjusted to return to their beginning values.

EDMA_TCD[CITER] = EDMA_TCD[BITER] = 1

EDMA_TCD[NBYTES] = 16

EDMA_TCD[SADDR] = 0x1000

EDMA_TCD[SOFF] = 1

EDMA_TCD[SSIZE] = 0

EDMA_TCD[SLAST] = –16

EDMA_TCD[DADDR] = 0x2000

EDMA_TCD[DOFF] = 4

EDMA_TCD[DSIZE] = 2

EDMA_TCD[DLAST_SGA] = –16

EDMA_TCD[INT_MAJ] = 1

EDMA_TCD[START] = 1 (Must be written last after all other fields have been initialized)

All other TCD fields = 0

This would generate the following sequence of events:

1. Slave write to the EDMA_TCD[START] bit requests channel service.
2. The channel is selected by arbitration for servicing.
3. eDMA engine writes: EDMA_TCD[DONE] = 0, EDMA_TCD[START] = 0, EDMA_TCD[ACTIVE] = 1.
4. eDMA engine reads: channel TCD data from local memory to internal register file.
5. The source to destination transfers are executed as follows:
   - a) read_byte(0x1000), read_byte(0x1001), read_byte(0x1002), read_byte(0x1003)
   - b) write_word(0x2000) Æ first iteration of the minor loop
   - c) read_byte(0x1004), read_byte(0x1005), read_byte(0x1006), read_byte(0x1007)
   - d) write_word(0x2004) Æ second iteration of the minor loop
   - e) read_byte(0x1008), read_byte(0x1009), read_byte(0x100A), read_byte(0x100B)
   - f) write_word(0x2008) Æ third iteration of the minor loop
   - g) read_byte(0x100C), read_byte(0x100D), read_byte(0x100E), read_byte(0x100F)
   - h) write_word(0x200C) Æ last iteration of the minor loop Æ major loop complete
6. eDMA engine writes: EDMA_TCD[SADDR] = 0x1000, EDMA_TCD[DADDR] = 0x2000, EDMA_TCD[CITER] = 1 (EDMA_TCD[BITER]).
7. eDMA engine writes: EDMA_TCD[ACTIVE] = 0, EDMA_TCD[DONE] = 1, EDMA_IRQR*n* = 1.
8. The channel retires.

The eDMA goes idle or services the next channel.

Multiple requests

The next example is the same as previous, excepting transferring 32 bytes via two hardware requests. The only fields that change are the major loop iteration count and the final address offsets. The eDMA is programmed for two iterations of the major loop transferring 16 bytes per iteration. After the channel’s hardware requests are enabled in the EDMA_ERQR, channel service requests are initiated by the slave device (ERQR should be set after TCD). Note that EDMA_TCD[START] = 0.

EDMA_TCD[CITER = EDMA_TCD[BITER] = 2
EDMA_TCD[NBYTES] = 16
EDMA_TCD[SADDR] = 0x1000
EDMA_TCD[SOFF] = 1
EDMA_TCD[SSIZE] = 0
EDMA_TCD[SLAST] = –32
EDMA_TCD[DADDR] = 0x2000
EDMA_TCD[DOFF] = 4
EDMA_TCD[DSIZE] = 2
EDMA_TCD[DLAST_SGA] = –32
EDMA_TCD[INT_MAJ] = 1
EDMA_TCD[START] = 0 (Must be written last after all other fields have been initialized)
All other TCD fields = 0

This generates the following sequence of events:

1. First hardware (eDMA peripheral request) request for channel service.
2. The channel is selected by arbitration for servicing.
3. eDMA engine writes: EDMA_TCD[DONE] = 0, EDMA_TCD[START] = 0, EDMA_TCD[ACTIVE] = 1.
4. eDMA engine reads: channel TCD data from local memory to internal register file.
5. The source to destination transfers are executed as follows:
   a) read_byte(0x1000), read_byte(0x1001), read_byte(0x1002), read_byte(0x1003)
   b) write_word(0x2000) Æ first iteration of the minor loop
   c) read_byte(0x1004), read_byte(0x1005), read_byte(0x1006), read_byte(0x1007)
   d) write_word(0x2004) Æ second iteration of the minor loop
   e) read_byte(0x1008), read_byte(0x1009), read_byte(0x100A), read_byte(0x100B)
   f) write_word(0x2008) Æ third iteration of the minor loop
   g) read_byte(0x100C), read_byte(0x100D), read_byte(0x100E), read_byte(0x100F)
   h) write_word(0x200C) Æ last iteration of the minor loop
6. eDMA engine writes: EDMA_TCD[SADDR] = 0x1010, EDMA_TCD[DADDR] = 0x2010, EDMA_TCD[CITER] = 1.
7. eDMA engine writes: EDMA_TCD[ACTIVE] = 0.
8. The channel retires Æ one iteration of the major loop.

The eDMA goes idle or services the next channel.

9. Second hardware (eDMA peripheral request) requests channel service.
10. The channel is selected by arbitration for servicing.
11. eDMA engine writes: EDMA_TCD[DONE] = 0, EDMA_TCD[START] = 0, EDMA_TCD[ACTIVE] = 1.
12. eDMA engine reads: channel TCD data from local memory to internal register file.
13. The source to destination transfers are executed as follows:
    a) read_byte(0x1010), read_byte(0x1011), read_byte(0x1012), read_byte(0x1013)
    b) write_word(0x2010) Æ first iteration of the minor loop
    c) read_byte(0x1014), read_byte(0x1015), read_byte(0x1016), read_byte(0x1017)
    d) write_word(0x2014) Æ second iteration of the minor loop
    e) read_byte(0x1018), read_byte(0x1019), read_byte(0x101A), read_byte(0x101B)
    f) write_word(0x2018) Æ third iteration of the minor loop
    g) read_byte(0x101C), read_byte(0x101D), read_byte(0x101E), read_byte(0x101F)
    h) write_word(0x201C) Æ last iteration of the minor loop Æ major loop complete
14. eDMA engine writes: EDMA_TCD[SADDR] = 0x1000, EDMA_TCD[DADDR] = 0x2000, EDMA_TCD[CITER] = 2 (EDMA_TCD[BITER]).
15. eDMA engine writes: EDMA_TCD[ACTIVE] = 0, EDMA_TCD[DONE] = 1, EDMA_IRQR*n* = 1.
16. The channel retires Æ major loop complete.

The eDMA goes idle or services the next channel.

Modulo feature

The modulo feature of the eDMA provides the ability to implement a circular data queue in which the size of the queue is a power of two. MOD is a 5-bit bitfield for both the source and destination in the TCD and specifies which lower address bits are allowed to increment from their original value after the address + offset calculation. All upper address bits remain the same as in the original value. A setting of 0 for this field disables the modulo feature.

Table 44 shows how the transfer addresses are specified based on the setting of the MOD field. Here a circular buffer is created where the address wraps to the original value while the 28 upper address bits (0x1234567x) retain their original value. In this example the source address is set to 0x12345670, the offset is set to 4 bytes and the MOD field is set to 4, allowing for a 2^4 byte (16-byte) size queue.

Table 44. Modulo feature example

Transfer number	Address
1	0x12345670
2	0x12345674
3	0x12345678
4	0x1234567C
5	0x12345670
6	0x12345674

8.5.6 TCD status

Minor loop complete

There are two methods to test for minor loop completion when using software initiated service requests. The first method is to read the field EDMA_TCD[CITER] and test for a change. Another method may be extracted from the sequence below. The second method is to test the bit EDMA_TCD[START] and the bit EDMA_TCD[ACTIVE]. The minor loop complete condition is indicated by both bits reading zero after EDMA_TCD[START] was written to a ‘1’. Polling the EDMA_TCD[ACTIVE] bit may be inconclusive because the active status may be missed if the channel execution is short in duration.

The TCD status bits execute the following sequence for a software activated channel:

1. EDMA_TCD[START] = 1, EDMA_TCD[ACTIVE] = 0, EDMA_TCD[DONE] = 0 (channel service request via software).
2. EDMA_TCD[START] = 0, EDMA_TCD[ACTIVE] = 1, EDMA_TCD[DONE] = 0 (channel is executing).
3. EDMA_TCD[START] = 0, EDMA_TCD[ACTIVE] = 0, EDMA_TCD[DONE] = 0 (channel has completed the minor loop and is idle), or
4. EDMA_TCD[START] = 0, EDMA_TCD[ACTIVE] = 0, EDMA_TCD[DONE] = 1 (channel has completed the major loop and is idle).

The best method to test for minor loop completion when using hardware initiated service requests is to read field EDMA_TCD[CITER] and test for a change. The hardware request and acknowledge handshakes signals are not visible in the programmer’s model.

The TCD status bits execute the following sequence for a hardware activated channel:

1. eDMA peripheral request asserts (channel service request via hardware).
2. EDMA_TCD[START] = 0, EDMA_TCD[ACTIVE] = 1, EDMA_TCD[DONE] = 0 (channel is executing).
3. EDMA_TCD[START] = 0, EDMA_TCD[ACTIVE] = 0, EDMA_TCD[DONE] = 0 (channel has completed the minor loop and is idle), or
4. EDMA_TCD[START] = 0, EDMA_TCD[ACTIVE] = 0, EDMA_TCD[DONE] = 1 (channel has completed the major loop and is idle).

For both activation types, the major loop complete status is explicitly indicated via bit EDMA_TCD[DONE].

Bit EDMA_TCD[START] is cleared automatically when the channel begins execution, regardless of how the channel was activated.

Active channel TCD reads

The eDMA will read back the true EDMA_TCD[SADDR], EDMA_TCD[DADDR], and EDMA_TCD[NBYTES] values if read while a channel is executing. The true values of the SADDR, DADDR, and NBYTES are the values the eDMA engine is currently using in its internal register file and not the values in the TCD local memory for that channel. The addresses (SADDR and DADDR) and NBYTES (decrements to zero as the transfer progresses) can give an indication of the progress of the transfer. All other values are read back from the TCD local memory.

Pre-emption status

Pre-emption is available only when fixed arbitration is selected for both group- and channel-arbitration modes. A pre-emptable situation is one in which a pre-empt-enabled channel is running and a higher priority request becomes active. When the eDMA engine is not operating in fixed group, fixed-channel arbitration mode, the determination of the relative priority of the actively running and the outstanding requests become undefined. Channel and group priorities are treated as equal (or more exactly, constantly rotating) when round-robin arbitration mode is selected.

Bit EDMA_TCD[ACTIVE] for the pre-empted channel remains asserted throughout the pre-emption. The pre-empted channel is temporarily suspended while the pre-empting channel executes one iteration of the major loop. Two EDMA_TCD[ACTIVE] bits set at the same time in the overall TCD map indicates a higher priority channel is actively pre-empting a lower priority channel.

8.5.7 Channel linking

Channel linking (or chaining) is a mechanism in which one channel sets bit EDMA_TCD[START] of another channel (or itself), thus initiating a service request for that channel. This operation is automatically performed by the eDMA engine at the conclusion of the major or minor loop when properly enabled.

The minor loop channel linking occurs at the completion of the minor loop (or one iteration of the major loop). Field EDMA_TCD[CITER.E_LINK] is used to determine whether a minor loop link is requested. When enabled, the channel link is made after each iteration of the minor loop except for the last. When the major loop is exhausted, only the major loop channel link fields are used to determine if a channel link should be made. For example, with the initial fields of:

EDMA_TCD[CITER.E_LINK] = 1
EDMA_TCD[CITER.LINKCH] = 0xC
EDMA_TCD[CITER] value = 0x4
EDMA_TCD[MAJOR.E_LINK] = 1
EDMA_TCD[MAJOR.LINKCH] = 0x7

will execute as:

1. Minor loop done Æ set channel 12 EDMA_TCD[START] bit
2. Minor loop done Æ set channel 12 EDMA_TCD[START] bit
3. Minor loop done Æ set channel 12 EDMA_TCD[START] bit
4. Minor loop done, major loop done Æ set channel 7 EDMA_TCD[START] bit

When minor loop linking is enabled (EDMA_TCD[CITER.E_LINK] = 1), field EDMA_TCD[CITER] uses a 9-bit vector to form the current iteration count.

When minor loop linking is disabled (EDMA_TCD[CITER.E_LINK] = 0), field EDMA_TCD[CITER] uses a 15-bit vector to form the current iteration count. The bits associated with field EDMA_TCD[CITER.LINKCH] are concatenated onto the CITER value to increase the range of the CITER.

Note: *After configuration, bit EDMA_TCD[CITER.E_LINK] and bit EDMA_TCD[BITER.E_LINK] must be equal or a configuration error is reported. The CITER and BITER vector widths must be equal to calculate the major loop, halfway done interrupt point.*

Table 45 summarizes how a DMA channel can link to another DMA channel, that is, use another channel's TCD, at the end of a loop.

Table 45. Channel linking parameters

Desired link behavior	TCD control field name	Description
Link at end of minor loop	citer.e_link	Enable channel-to-channel linking on minor loop completion (current iteration).
	citer.linkch	Link channel number when linking at end of minor loop (current iteration).
Link at end of major loop	major.e_link	Enable channel-to-channel linking on major loop completion.
	major.linkch	Link channel number when linking at end of major loop.

8.5.8 Dynamic programming

Dynamic channel linking

Dynamic channel linking is the process of setting the TCD.major.e_link bit during channel execution. This bit is read from the TCD local memory at the end of channel execution, thus allowing the user to enable the feature during channel execution.

Because the user is allowed to change the configuration during execution, a coherency model is needed. Consider the scenario where the user attempts to execute a dynamic channel link by enabling the TCD.major.e_link bit at the same time the eDMA engine is retiring the channel. The TCD.major.e_link would be set in the programmer's model, but it would be unclear whether the actual link was made before the channel retired.

The coherency model in *Table 46* is recommended when executing a dynamic channel link request.

Table 46. Coherency model for a dynamic channel link request

Step	Action
1	Write 1b to the TCD.major.e_link bit.
2	Read back the TCD.major.e_link bit.
3	Test the TCD.major.e_link request status: – If TCD.major.e_link = 1b, the dynamic link attempt was successful. – If TCD.major.e_link = 0b, the attempted dynamic link did not succeed (the channel was already retiring).

For this request, the TCD local memory controller forces the TCD.major.e_link bit to zero on any writes to a channel's TCD.word7 after that channel's TCD.done bit is set, indicating the major loop is complete.

Note: The user must clear the TCD.done bit before writing the TCD.major.e_link bit. The TCD.done bit is cleared automatically by the eDMA engine after a channel begins execution.

Dynamic scatter/gather

Dynamic scatter/gather is the process of setting the TCD.e_sg bit during channel execution. This bit is read from the TCD local memory at the end of channel execution, thus allowing the user to enable the feature during channel execution.

Because the user is allowed to change the configuration during execution, a coherency model is needed. Consider the scenario where the user attempts to execute a dynamic scatter/gather operation by enabling the TCD.e_sg bit at the same time the eDMA engine is retiring the channel. The TCD.e_sg would be set in the programmer's model, but it would be unclear whether the actual scatter/gather request was honored before the channel retired.

Two methods for this coherency model are shown in the following subsections. Method 1 has the advantage of reading the major.linkch field and the e_sg bit with a single read. For both dynamic channel linking and scatter/gather requests, the TCD local memory controller forces the TCD.major.e_link and TCD.e_sg bits to zero on any writes to a channel's TCD.word7 if that channel's TCD.done bit is set indicating the major loop is complete.

Note: The user must clear the TCD.done bit before writing the TCD.major.e_link or TCD.e_sg bits. The TCD.done bit is cleared automatically by the eDMA engine after a channel begins execution.

Method 1 (channel not using major loop channel linking)

For a channel not using major loop channel linking, the coherency model in *Table 47* may be used for a dynamic scatter/gather request.

When the TCD.major.e_link bit is zero, the TCD.major.linkch field is not used by the eDMA. In this case, the TCD.major.linkch bits may be used for other purposes. This method uses the TCD.major.linkch field as a TCD indentification (ID).

Table 47. Coherency model for method 1

Step	Action
1	When the descriptors are built, write a unique TCD ID in the TCD.major.linkch field for each TCD associated with a channel using dynamic scatter/gather.
2	Write 1b to theTCD.d_req bit. Should a dynamic scatter/gather attempt fail, setting the d_req bit will prevent a future hardware activation of this channel. This stops the channel from executing with a destination address (daddr) that was calculated using a scatter/gather address (written in the next step) instead of a dlast final offest value.
3	Write theTCD.dlast_sga field with the scatter/gather address.
4	Write 1b to the TCD.e_sg bit.
5	Read back the 16 bit TCD control/status field.
6	Test the TCD.e_sg request status and TCD.major.linkch value: – If e_sg = 1b, the dynamic link attempt was successful. – If e_sg = 0b and the major.linkch (ID) did not change, the attempted dynamic link did not succeed (the channel was already retiring). – If e_sg = 0b and the major.linkch (ID) changed, the dynamic link attempt was successful (the new TCD’s e_sg value cleared the e_sg bit).

Method 2 (channel using major loop linking)

For a channel using major loop channel linking, the coherency model in *Table 48* may be used for a dynamic scatter/gather request. This method uses the TCD.dlast_sga field as a TCD indentification (ID).

Table 48. Coherency model for method 2

Step	Action
1	Write 1b to theTCD.d_req bit. Should a dynamic scatter/gather attempt fail, setting the d_req bit will prevent a future hardware activation of this channel. This stops the channel from executing with a destination address (daddr) that was calculated using a scatter/gather address (written in the next step) instead of a dlast final offest value.
2	Write theTCD.dlast_sga field with the scatter/gather address.
3	Write 1b to the TCD.e_sg bit.
4	Read back the TCD.e_sg bit.
5	Test the TCD.e_sg request status: – If e_sg = 1b, the dynamic link attempt was successful. – If e_sg = 0b, read the 32 bit TCD dlast_sga field. – If e_sg = 0b and the dlast_sga did not change, the attempted dynamic link did not succeed (the channel was already retiring). – If e_sg = 0b and the dlast_sga changed, the dynamic link attempt was successful (the new TCD’s e_sg value cleared the e_sg bit).

9 Multi-Layer AHB Crossbar Switch (XBAR)

9.1 Introduction

9.1.1 Overview

This section provides an overview of the multi-layer AHB crossbar switch (XBAR). The purpose of the XBAR is to concurrently support simultaneous connections between master ports and slave ports. The XBAR supports a 32-bit address bus width. Only a single data bus width is supported throughout the design, thus, all master and slave ports have the same data bus width.

The XBAR has four master ports and four slave ports. *Figure 47* shows a block diagram of the XBAR.

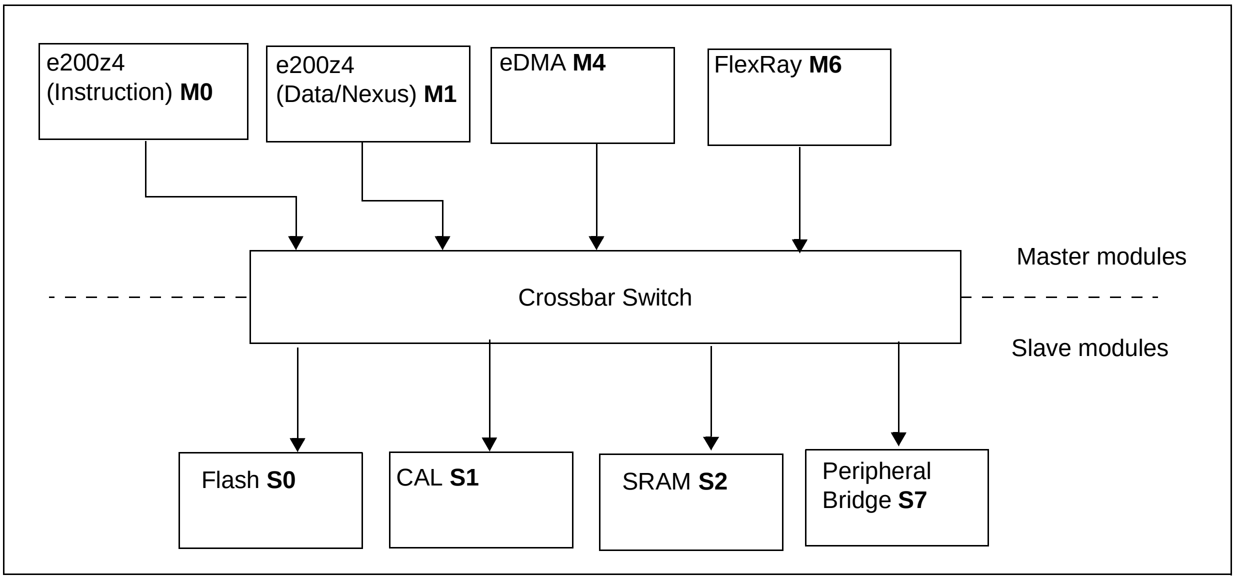


Figure 47. XBAR device-specific block diagram

The port mappings are shown in *Table 49*.

Table 49. Master/Slave mappings

Module	Port		Physical master ID
	Type	Logical number	
e200z4 core instruction	Master	M0	0
e200z4 core Load/Store	Master	M1	0
e200z4 core Nexus	Master	M1	8
eDMA	Master	M4	4
FlexRay Interface	Master	M6	6
Flash Memory	Slave	S0	—

Table 49. Master/Slave mappings

Module	Port		Physical master ID
	Type	Logical number	
Calibration Bus(1)	Slave	S1	—
SRAM	Slave	S2	—
Peripheral Bridge	Slave	S7	—

1. The calibration bus is only available on the calibration package.

9.1.2 Features

The XBAR has the ability to gain control of all the slave ports and prevent any masters from making accesses to the slave ports. This feature is useful for turning off the clocks to the system and ensuring that no bus activity will be interrupted.

The XBAR can put each slave port into a low power park mode so that the slave port will not dissipate any power transitioning address, control or data signals when not being actively accessed by a master port.

Each slave port can also support multiple master priority schemes—the user can dynamically change master priority levels on a slave port by slave port basis.

The XBAR allows concurrent transactions to occur from any master port to any slave port. It is possible for all master ports and slave ports to be in use at the same time as a result of independent master requests. If a slave port is simultaneously requested by more than one master port, arbitration logic will select the higher priority master and grant it ownership of the slave port. All other masters requesting that slave port will stalled until the higher priority master completes its transactions.

The XBAR has a 32-bit internal address bus and a 64-bit internal data bus.

9.1.3 Limitations

The XBAR routes bus transactions initiated on the master ports to the appropriate slave ports. There is no provision included to route transactions initiated on the slave ports to other slave ports or to master ports. Simply put, the slave ports do not support the bus request/bus grant protocol; the XBAR assumes it is the sole master of each slave port.

9.1.4 General operation

When a master makes an access to the XBAR the access will be immediately taken by the XBAR. If the targeted slave port of the access is available then the access will be immediately presented on the slave port. It is possible to make single clock (zero wait state) accesses through the XBAR. If the targeted slave port of the access is busy or parked on a different master port the requesting master will simply see wait states inserted until the targeted slave port can service the master’s request. The latency in servicing the request depends on each master’s priority level and the responding peripheral’s access time.

Since the XBAR appears to be just another slave to the master device, the master device will have no knowledge of whether or not it actually owns the slave port it is targeting. While the master does not have control of the slave port it is targeting it will simply be wait stated.

A master is given control of the targeted slave port only after a previous access to a different slave port has completed, regardless of its priority on the newly targeted slave port. This prevents deadlock from occurring when a master has an outstanding request to one slave port that has a long response time, has a pending access to a different slave port, and a lower priority master is also making a request to the same slave port as the pending access of the higher priority master.

Once the master has control of the slave port it is targeting, the master remains in control of that slave port until it gives up the slave port by running an IDLE cycle or by leaving that slave port for its next access. The master could also lose control of the slave port if another higher priority master makes a request to the slave port; however, if the master is running a locked or fixed length burst transfer it retains control of the slave port until that transfer is completed.

The XBAR will terminate all master IDLE transfers (as opposed to allowing the termination to come from one of the slave busses). Additionally, when no master is requesting access to a slave port the XBAR will drive IDLE transfers onto the slave bus, even though a default master may be granted access to the slave port.

When a slave bus is being IDLEd by the XBAR it can park the slave port on the master port indicated by the PARK bits in the SGPCR (Slave General Purpose Control Register). This can be done in an attempt to save the initial clock of arbitration delay that would otherwise be seen if the master had to arbitrate to gain control of the slave port. The slave port can also be put into low power park mode in attempt to save power.

9.2 XBAR registers

This section provides information on XBAR registers.

9.2.1 Register summary

There are two registers that reside in each slave port of the XBAR. These registers are IP bus compliant registers. Read and write transfers both require two IP bus clock cycles. Read and writen operations can be performed on these registers only in supervisor mode. Additionally, these registers can only be read from or written to by 32-bit accesses.

The registers are fully decoded and an error response is returned if an unimplemented location is accessed within the XBAR.

The slave registers also feature a bit, which when written with a 1, will prevent the registers from being written to again. The registers will still be readable, but future write attempts will have no effect on the registers and will be terminated with an error response.

The memory map for the XBAR program-visible registers is shown in *Table 50*.

Table 50. XBAR Register Configuration Summary

Address	Register	Location
XBAR_Base (0xFFF0_4000)	MPR0 — Master Priority Register for Slave port 0	*on page 9-219*
XBAR_Base + 0x004 – XBAR_Base + 0x00F	Reserved	—
XBAR_Base + 0x010	SGPCR0 — General Purpose Control Register for Slave port 0	*on page 9-221*

Table 50. XBAR Register Configuration Summary (continued)

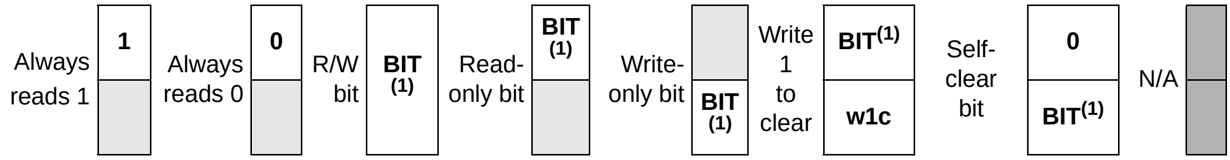
Address	Register	Location
XBAR_Base + 0x014 – XBAR_Base + 0x0FF	Reserved	—
XBAR_Base + 0x100	MPR1 — Master Priority Register for Slave port 1	*on page 9-219*
XBAR_Base + 0x104 – XBAR_Base + 0x10F	Reserved	—
XBAR_Base + 0x110	SGPCR1 — General Purpose Control Register for Slave port 1	*on page 9-221*
XBAR_Base + 0x114 – XBAR_Base + 0X1FF	Reserved	—
XBAR_Base + 0x200	MPR2 — Master Priority Register for Slave port 2	*on page 9-219*
XBAR_Base + 0x204 – XBAR_Base + 0x20F	Reserved	—
XBAR_Base + 0x210	SGPCR2 — General Purpose Control Register for Slave port 2	*on page 9-221*
XBAR_Base + 0x214 – XBAR_Base + 0X6FF	Reserved	—
XBAR_Base + 0x700	MPR7 — Master Priority Register for Slave port 7	*on page 9-219*
XBAR_Base + 0x704 – XBAR_Base + 0X70F	Reserved	—
XBAR_Base + 0x710	SGPCR7 — General Purpose Control Register for Slave port 7	*on page 9-221*
XBAR_Base + 0x714 – XBAR_Base + 0XF03	Reserved	—

9.2.2 XBAR register descriptions

The following paragraphs provide detailed descriptions of the various XBAR registers.

Refer to *Figure 48* for the various bit configurations that appear in the register maps.

Figure 48. Key to Register Fields



1. “BIT” refers to a field name in the register. Some fields span multiple bits.

Master Priority Register (XBAR_MPRn)

The Master Priority Register (MPR) resides in each slave port and sets the priority of each master port on a per slave port basis, e.g., MPR0 sets priority for each master port for slave port 0.

Figure 49. Master Priority Register (XBAR_MPRn)

MPR0: Address: XBAR_Base (0xFFF0_4000) + 0x0000
MPR1: Address: XBAR_Base (0xFFF0_4000) + 0x0100
MPR2: Address: XBAR_Base (0xFFF0_4000) + 0x0200
MPR7: Address: XBAR_Base (0xFFF0_4000) + 0x0700

Access: Supervisor

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	MSTR6			0	0	0	0	0	MSTR4		
W	▨	▨	▨	▨	▨				▨	▨	▨	▨	▨			
Reset	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	MSTR1			0	MSTR0		
W	▨	▨	▨	▨	▨	▨	▨	▨	▨				▨			
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

▨ = not implemented

Table 51. XBAR Master Priority Register Field Descriptions

Field	Description
0	**Reserved** This bit is reserved for future expansion. It is read as zero and should be written with zero for upward compatibility.
1:3	**Reserved**
4	**Reserved** This bit is reserved for future expansion. It is read as zero and should be written with zero for upward compatibility.
5:7 MSTR6	**Master 6 Priority** These bits set the arbitration priority for master port 6 (FlexRay) on the associated slave port. These bits are initialized by hardware reset. The reset value is 110. 000: This master has the highest priority when accessing the slave port. ... 111: This master has the lowest priority when accessing the slave port.
8	**Reserved** This bit is reserved for future expansion. It is read as zero and should be written with zero for upward compatibility.

Table 51. XBAR Master Priority Register Field Descriptions (continued)

Field	Description
9:11	**Reserved** These bits are reserved for future expansion. They are read as zero and should be written with zero for upward compatibility.
12	**Reserved** This bit is reserved for future expansion. It is read as zero and should be written with zero for upward compatibility.
13:15 MSTR4	**Master 4 Priority** These bits set the arbitration priority for master port 4 (eDMA) on the associated slave port. These bits are initialized by hardware reset. The reset value is 100. 000: This master has the highest priority when accessing the slave port. ... 111: This master has the lowest priority when accessing the slave port.
16	**Reserved** This bit is reserved for future expansion. It is read as zero and should be written with zero for upward compatibility.
17:19	**Reserved** These bits are reserved for future expansion. They are read as zero and should be written with zero for upward compatibility.
20	**Reserved** This bit is reserved for future expansion. It is read as zero and should be written with zero for upward compatibility.
21:23	**Reserved** These bits are reserved for future expansion. They are read as zero and should be written with zero for upward compatibility.
24	**Reserved** This bit is reserved for future expansion. It is read as zero and should be written with zero for upward compatibility.
25:27 MSTR1	**Master 1 Priority** These bits set the arbitration priority for master port 1 (e200z4 core load/store bus and e200z4 core Nexus) on the associated slave port. These bits are initialized by hardware reset. The reset value is 001. 000: This master has the highest priority when accessing the slave port. ... 111: This master has the lowest priority when accessing the slave port.
28	**Reserved** This bit is reserved for future expansion. It is read as zero and should be written with zero for upward compatibility.

Table 51. XBAR Master Priority Register Field Descriptions (continued)

Field	Description
29:31 MSTR0	**Master 0 Priority** These bits set the arbitration priority for master port 0 (e200z4 core instruction bus) on the associated slave port. These bits are initialized by hardware reset. The reset value is 000 000: This master has the highest priority when accessing the slave port. ... 111: This master has the lowest priority when accessing the slave port.

The Master Priority Register can only be accessed in supervisor mode with 32-bit accesses. Once the RO (Read Only) bit has been set in the slave General Purpose Control Register the Master Priority Register can only be read from, attempts to write to it will have no effect on the MPR and result in an error response.

Note: No two available master ports may be programmed with the same priority level. Attempts to program two or more available masters with the same priority level will result in an error response and the MPR will not be updated.

Slave General Purpose Control Register (XBAR_SGPCRn)

The Slave General Purpose Control Register (SGPCR) controls several features of each slave port.

The Read Only (RO) bit will prevent any registers associated with this slave port from being written to once set. This bit may be written with 0 as many times as the user desires, but once it is written to a 1 only a reset condition will allow it to be written again.

The PCTL bits determine how the slave port will park when no master is actively making a request. The available options are to park on the master defined by the PARK bits, park on the last master to use the slave port, or go into a low power park mode which will force all the outputs of the slave port to inactive states when no master is requesting an access. The low power park feature can result in an overall power savings if a the slave port is not saturated; however, it will force an extra clock of latency whenever any master tries to access it when it is not in use because it will not be parked on any master.

The PARK bits determine which master the slave will park on when no master is making an active request. Please use caution to only select master ports that are actually present in the design. If the user programs the PARK bits to a master not present in the current design implementation undefined behavior will result.

Note: The SGPCR can only be accessed in supervisor mode with 32-bit accesses. Once the RO (Read Only) bit has been set in the SGPCR the SGPCR can only be read, attempts to write to it will have no effect on the SGPCR and result in an error response.

Figure 50. Slave General Purpose Control Register (XBAR_SGPCRn)

SGPCR0: Address: XBAR_Base + 0X0010 + 0x0000 (0xFFF0_4010)
SGPCR1: Address: XBAR_Base + 0X0010 + 0x0100 (0xFFF0_4110)
SGPCR2: Address: XBAR_Base + 0X0010 + 0x0200 (0xFFF0_4210)
SGPCR7: Address: XBAR_Base + 0X0010 + 0x0700 (0xFFF0_4710)

Access: Supervisor

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	RO	0	0	0	0	0	0	0	0	HPE6	0	HPE4	0	0	HPE1	HPE0
W		▨	▨	▨	▨	▨	▨	▨	▨		▨		▨	▨		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	ARB		0	0	PCTL		0	PARK		
W	▨	▨	▨	▨	▨	▨			▨	▨			▨			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = not implemented

Table 52. XBAR Slave General Purpose Control Register Field Descriptions

Field	Description
0 RO	**Read Only** This bit is used to force all of a slave port's registers to be read only. Once written to 1 it can only be cleared by hardware reset. This bit is initialized by hardware reset. The reset value is 0. 0: All this slave port's registers can be written. 1: All this slave port's registers are read only and cannot be written (attempted writes have no effect and result in an error response).
1:7	**Reserved** These bits are reserved for future expansion. They read as zero and should be written with zero for upward compatibility.
8:15 HPEx	**High Priority Enable** These bits are used to enable the mX_high_priority inputs for the respective master. These bits are initialized by hardware reset. The reset value is 0. 0: The mX_high_priority input is disabled on this slave port 1: The mX_high_priority input is enabled on this slave port.
16:21	**Reserved** These bits are reserved for future expansion. They are read as zero and should be written with zero for upward compatibility.

Table 52. XBAR Slave General Purpose Control Register Field Descriptions (continued)

Field	Description
22:23 ARB	**Arbitration Mode** These bits are used to select the arbitration policy for the slave port. These bits are initialized by hardware reset. The reset value is 00. 00: Fixed Priority 01: Round Robin (rotating) Priority 10: Reserved 11: Reserved
24:25	**Reserved** These bits are reserved for future expansion. They are read as zero and should be written with zero for upward compatibility.
26:27 PCTL	**Parking Control** These bits determine the parking control used by this slave port. These bits are initialized by hardware reset. The reset value is 00. 00: When no master is making a request the arbiter will park the slave port on the master port defined by the PARK bit field. 01: When no master is making a request the arbiter will park the slave port on the last master to be in control of the slave port. 10: When no master is making a request the arbiter will park the slave port on no master and will drive all outputs to a constant safe state. 11: Reserved
28	**Reserved** This bit is reserved for future expansion. It is read as zero and should be written with zero for upward compatibility.
29:31 PARK	**PARK** These bits are used to determine which master port this slave port parks on when no masters are actively making requests and the PCTL bits are set to 00. These bits are initialized by hardware reset. The reset value is 000. 000: Park on Master Port 0 (e200z448n3 core instruction) 001: Park on Master Port 1 (e200z448n3 core Load/Store) 010: Reserved 011: Reserved 100: Park on Master Port 4 (eDMA) 101: Reserved 110: Park on Master Port 6 (FlexRay) 111: Park on Master Port 7 (EBI)

9.2.3 Coherency

Since the content of the registers has a real time effect on the operation of the XBAR it is important for the user to understand that any register modifications take effect as soon as the register is written. The values of the registers do not track with slave port related AHB accesses but instead track only with IP bus accesses.

9.3 Function

This section describes in more detail the functionality of the XBAR.

9.3.1 Arbitration

The XBAR supports two arbitration schemes: a simple fixed-priority comparison algorithm and a simple round-robin fairness algorithm. The arbitration scheme is independently programmable for each slave port.

Fixed priority operation

When operating in fixed-priority mode, each master is assigned a unique priority level in the MPR (Master Priority Register). If two masters both request access to a slave port the master with the highest priority in the selected priority register will gain control over the slave port.

Any time a master makes a request to a slave port the slave port checks to see if the new requesting master’s priority level is higher than that of the master that currently has control over the slave port (unless the slave port is in a parked state). The slave port does an arbitration check at every clock edge to ensure that the proper master (if any) has control of the slave port.

If the new requesting master’s priority level is higher than that of the master that currently has control of the slave port the new requesting master will be granted control over the slave port at the next clock edge. The exception to this rule is if the master that currently has control over the slave port is running a fixed length burst transfer or a locked transfer. In this case the new requesting master will have to wait until the end of the burst transfer or locked transfer before it will be granted control of the slave port. If the master is running an undefined length burst transfer the new requesting master must wait until an arbitration point for the undefined length burst transfer before it will be granted control of the slave port. Arbitration points for an undefined length burst are defined in the MGPCR for each master.

If the new requesting master’s priority level is lower than that of the master that currently has control of the slave port the new requesting master will be forced to wait until the master that currently has control of the slave port either runs an IDLE cycle or runs a non IDLE cycle to a location other than the current slave port.

Round-Robin priority operation

When operating in round-robin mode, each master is assigned a relative priority based on the master number.This relative priority is compared to the ID of the last master to perform a transfer on the slave bus. The highest priority requesting master will become owner of the slave bus as the next transfer boundary (accounting for locked and fixed-length burst transfers). Priority is based on how far ahead the ID of the requesting master is to the ID of the last master (ID is defined by master port number).

Once granted access to a slave port, a master may perform as many transfers as desired to that port until another master makes a request to the same slave port. The next master in line will be granted access to the slave port if the current master has no pending access request.

As an example of arbitration in round-robin mode, assume the XBAR is implemented with master ports 0, 1, 4 and 5. If the last master of the slave port was master 1, and master 0, 4 and 5 make simultaneous requests, they will be serviced in the order 4, 5 and then 0.

Parking may still be used in a round-robin mode, but will not affect the round-robin pointer unless the parked master actually performs a transfer. Handoff occurs to the next master in line after one cycle of arbitration. If the slave port is put into low power park mode the round-robin pointer is reset to point at master port 0, giving it the highest priority.

Parking

If no master is currently requesting the slave port, the slave port is parked. The slave port parks in one of three places, indicated by the value of the PCTL field in the XBAR_SGPCR.

- If park-on-specific master mode is selected, the slave port parks on the master designated by the PARK field. When the master accesses the slave port again, a one clock arbitration penalty is incurred only for an access request made by another master port to the slave port. No other arbitration penalties are incurred. All other masters pay a one clock penalty.
- If park-on-last (POL) mode is selected, then the slave port parks on the last master to access it, passing that master’s signals through to the slave bus. When the master accesses the slave port again, no other arbitration penalties are incurred except that a one clock arbitration penalty is incurred for each access request to the slave port made by another master port. All other masters pay a one clock penalty.
- If the low-power-park (LPP) mode is selected, then the slave port enters low-power park mode. It is not under control by any master and does not transmit any master signals to the slave bus. All slave bus activity halts because all slave bus signals are not toggling. This saves power if the slave port is not used for some time. However, when a master does make a request to a slave port parked in low-power-park, a one clock arbitration delay is incurred to get ownership of the slave port.

9.3.2 Priority assignment

Each master port needs to be assigned a unique 3-bit priority level. If an attempt is made to program multiple master ports with the same priority level within a register (MPR) the XBAR will respond with an error and the registers will not be updated.

10 Peripheral Bridge (PBRIDGE)

The Peripheral Bridge (PBRIDGE) provides an interface between the system crossbar switch bus and the lower-bandwidth peripheral bus.

10.1 PBRIDGE features

The PBRIDGE:

- Is only meant for slave peripherals
- Supports 32-bit peripherals (byte, halfword, and word reads and write are supported to each)
- Supports a pair of accesses for 64-bit fetches

10.2 PBRIDGE modes of operation

The PBRIDGE has only one operating mode.

10.3 PBRIDGE block diagram

The PBRIDGE is the interface between the system bus interface and on-chip peripherals as shown in *Figure 51*.

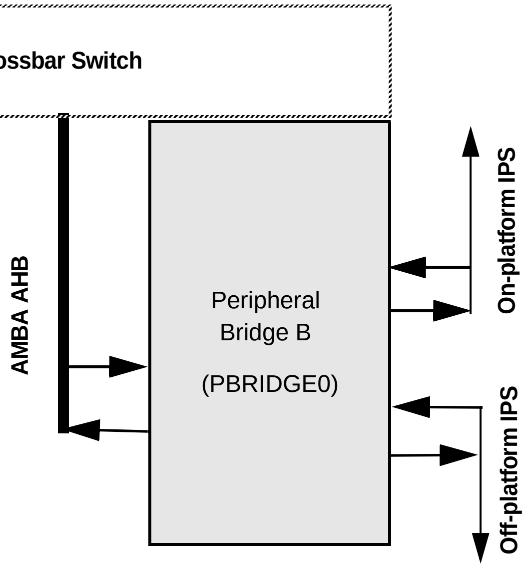


Figure 51. PBRIDGE interface

10.4 PBRIDGE signal description

The PBRIDGE has no external signals.

10.5 PBRIDGE functional description

The PBRIDGE functions as a protocol translator. Support is provided for generating a pair of 32-bit slave bus instruction accesses (not data accesses) when targeted by a 64-bit system bus access.

Accesses which fall within the address space of the PBRIDGE are decoded to provide individual module selects for peripheral devices.

10.5.1 Read cycles

Two clock read accesses are possible with the PBRIDGE when the requested access size is 32-bits or smaller, and is not misaligned across a 32-bit boundary. If the requested instruction access size is 64-bits, then a minimum of three clocks are required to complete the access. Misaligned read accesses are not supported. 64-bit data reads (not instruction) are not supported.

10.5.2 Write cycles

Three clock write accesses are possible with the PBRIDGE when the requested access size is 32-bits or smaller, and is not misaligned across a 32-bit boundary. Misaligned writes that do not cross a 32-bit boundary are supported. 64-bit data writes (not instruction) are not supported.

10.6 Memory map and register description

10.6.1 Memory map

Each register in the PBRIDGE module has a size of 32 bits. The registers are listed in *Table 53* . The memory map organization is shown in *Table 54*. The organizational hierarchy is as follows:

- The module has multiple registers with the same register name (MPCR, PACR, OPACR), each at a different address offset.
- Each register has multiple similarly-named fields, each with a different number.
- Each field has subfields as defined elsewhere in this section.

Accesses to registers or register fields marked as reserved will return zeros on reads, and will be ignored on writes.

Table 53. PBRIDGE registers

Offset from PBRIDGE_BASE (0xFFF0_0000)	Register	Location
0x0000–0x0007[(1)]	Master Privilege Control Registers (MPCR)	*on page 10-228*
0x0008–0x001F	Reserved	

Table 53. PBRIDGE registers (continued)

Offset from PBRIDGE_BASE (0xFFF0_0000)	Register	Location
0x0020–0x003F(1)	Peripheral Access Control Registers (PACR)	*on page 10-229*
0x0040–0x006F(1)	Off-Platform Peripheral Access Control Registers (OPACR)	*on page 10-230*
0x0070–0x3FFF	Reserved	

1. This memory range contains reserved areas. See *Table 54*.

Table 54. PBRIDGE memory map

Address offset	Register name	Bit numbers							
		0–3	4–7	8–11	12–15	16–19	20–23	24–27	28–31
0x0000	MPCR	MPCR0	Reserved			MPCR4	Reserved	MPCR6	MPCR7
0x0004		MPCR8	Reserved						
0x0020	PACR	Reserved	PACR1	Reserved		PACR4	Reserved		
0x0024		Reserved						PACR14	PACR15
0x0028		PACR16	PACR17	PACR18	Reserved				
0x002C		Reserved							
0x0040	OPACR	OPACR0	Reserved	OPACR2	OPACR3	Reserved	OPACR5	OPACR6	OPACR7
0x0044		Reserved				OPACR12	OPACR13	OPACR14	Reserved
0x0048		OPACR16	OPACR17	OPACR18	Reserved				
0x004C		OPACR24	Reserved		OPACR27	Reserved			OPACR31
0x0050		Reserved							
0x0054		Reserved							
0x0058		Reserved							
0x005C		Reserved		OPACR58	Reserved				
0x0060		OPACR64	OPACR65	OPACR66	OPACR67	OPACR68	Reserved		OPACR71
0x0064		OPACR72	Reserved						OPACR79
0x0068		OPACR80	OPACR81	OPACR82	OPACR83	OPACR84	Reserved		
0x006C		Reserved				OPACR92	Reserved		

10.6.2 Register descriptions

Master privilege control registers (MPCR)

Each MPCR register contains one or more 4-bit fields, called MPCR*n*, as shown in *Table 54*. Each of these fields defines the access privilege level associated with bus master *n* in the platform as well as specifies whether write accesses from this master are bufferable. The registers provide one field per bus master. See the “Logical master IDs” section in the XBAR chapter for a list of master numbers and names.

Each MPCR*n* field has the structure described in *Figure 52* and *Table 55*.

	0	1	2	3
R	0	MTR	MTW	MPL
W				
Reset	0	1	1	1

Figure 52. MPCR*n* field structure

Table 55. MPCR*n* field structure descriptions

Subfield	Description
MTR	Master Trusted for Reads This bit determines whether the master is trusted for read accesses. 0 This master is not trusted for read accesses. 1 This master is trusted for read accesses.
MTW	Master Trusted for Writes This bit determines whether the master is trusted for write accesses. 0 This master is not trusted for write accesses. 1 This master is trusted for write accesses.
MPL	Master Privilege Level This bit determines how the privilege level of the master is determined. 0 Accesses from this master are forced to user-mode. 1 Accesses from this master are not forced to user-mode.

Table 56. MPCR register fields

Register	Master name	Reset value
MPCR 0	z4 core (instruction + load/store)	0b0111, meaning MTR = 1 MTW = 1 MPL = 1
MPCR 4	DMA	
MPCR 6	FlexRay	
MPCR 8	z4 core Nexus	

Peripheral access control registers (PACR)

Each PACR register contains one or more 4-bit fields, called PACR*n*, as shown in *Table 54*. Each of these fields defines the access levels supported by the associated module. The lists of modules and their corresponding numbers are shown in *Table 58*.

Each PACR*n* field has the structure described in *Figure 53* and *Table 57*.

	0	1	2	3
R	0	SP	WP	TP
W				
Reset	0	1	0	0

Figure 53. PACR*n* field structure

Table 57. PACR*n* field structure descriptions

Subfield	Description
SP	Supervisor Protect This bit determines whether the peripheral requires supervisor privilege level for access. 0 This peripheral does not require supervisor privilege level for accesses. 1 This peripheral requires supervisor privilege level for accesses. The MPCRx[MPL] control bit for the master must be set. If not, the access is terminated with an error response and no peripheral access is initiated on the IPS bus.
WP	Write Protect This bit determines whether the peripheral allows write accesses. 0 This peripheral allows write accesses. 1 This peripheral is write protected. If a write access is attempted, the access is terminated with an error response and no peripheral access is initiated on the IPS bus.
TP	Trusted Protect This bit determines whether the peripheral allows accesses from an untrusted master. 0 Accesses from an untrusted master are allowed. 1 Accesses from an untrusted master are not allowed. If an access is attempted by an untrusted master, the access is terminated with an error response and no peripheral access is initiated on the IPS bus.

Table 58. Peripheral Access Control Register (PACR) fields

Register	Peripheral	Reset value
PACR 1	Crossbar	0100b, meaning SP = 1 WP = 0 TP = 0
PACR 4	MPU	
PACR 14	SWT	
PACR 15	STM	
PACR 16	ECSM	
PACR 17	DMA	
PACR 18	Interrupt controller	

Off-platform peripheral access control registers (OPACR)

The OPACR defines the access levels supported by the associated module. Each OPACR has a format identical to the PACR described in *Section , Peripheral access control registers (PACR)*.

The lists of off-platform peripheral registers and their corresponding modules are listed in *Table 59*.

Table 59. Off-platform Peripheral Access Control Register (OPACR) fields

Field	Peripheral	Reset value
OPACR 0	eQADC	0100b, meaning SP = 1 WP = 0 TP = 0
OPACR 2	Decimation filter A	
OPACR 3	Decimation filter B	
OPACR 5	DSPI B	
OPACR 6	DSPI C	
OPACR 7	DSPI D	
OPACR 12	eSCI A	
OPACR 13	eSCI B	
OPACR 14	eSCI C	
OPACR 16	FlexCAN A	
OPACR 17	FlexCAN B	
OPACR 18	FlexCAN C	
OPACR 24	FlexRay	
OPACR 27	System Information Module	
OPACR 31	BAM	
OPACR 58	CRC	
OPACR 64	FM PLL	0100b, meaning SP = 1 WP = 0 TP = 0
OPACR 66	Flash module A	
OPACR 68	SIU	
OPACR 71	DTS	
OPACR 72	eMIOS	
OPACR 79	PMC	
OPACR 80	eTPU2	
OPACR 81	Reaction module	
OPACR 82	eTPU parameter RAM	
OPACR 83	eTPU parameter RAM mirror	
OPACR 84	eTPU code RAM	
OPACR 92	PIT	

11 Flash memory

11.1 Introduction

This section presents information about the following components on this device:

- The flash memory blocks
- The platform flash memory controller

The primary function of the flash memory module is to serve as electrically programmable and erasable non-volatile memory. The NVM memory can be used for instruction and data storage. The block is a non-volatile solid-state silicon memory device consisting of blocks of single-transistor storage elements, an electrical means for selectively adding (programming) and removing (erasing) charge from these elements, and a means of selectively sensing (reading) the charge stored in these elements. The flash is addressable by word (32 bits) and page (128 bits).

The flash array block comprises two functional units: the flash core (FC) and the memory interface (MI).

The FC is composed of arrayed non-volatile storage elements, sense amplifiers, row selects, column selects, charge pumps, and redundancy logic. The arrayed storage elements in the FC are subdivided into physically separate units referred to as blocks.

The MI contains the registers and logic which control the operation of the FC. The MI is also the interface to the platform flash bus interface unit (PFBIU).

The flash array's core has three address spaces: low-address space, mid-address space, and high-address space (see *Figure 54*).

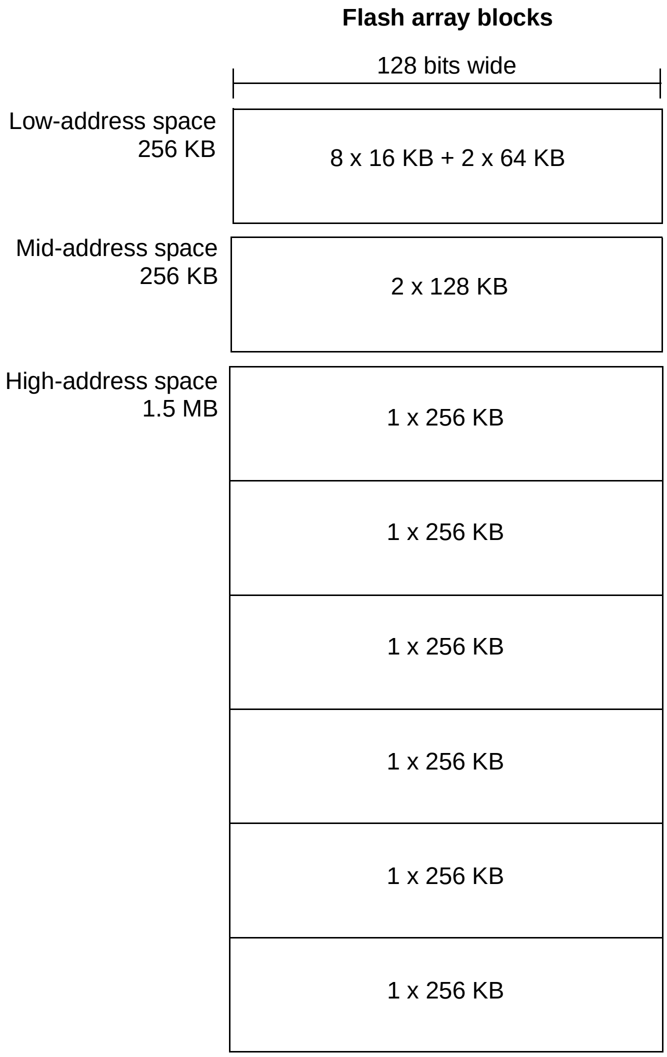
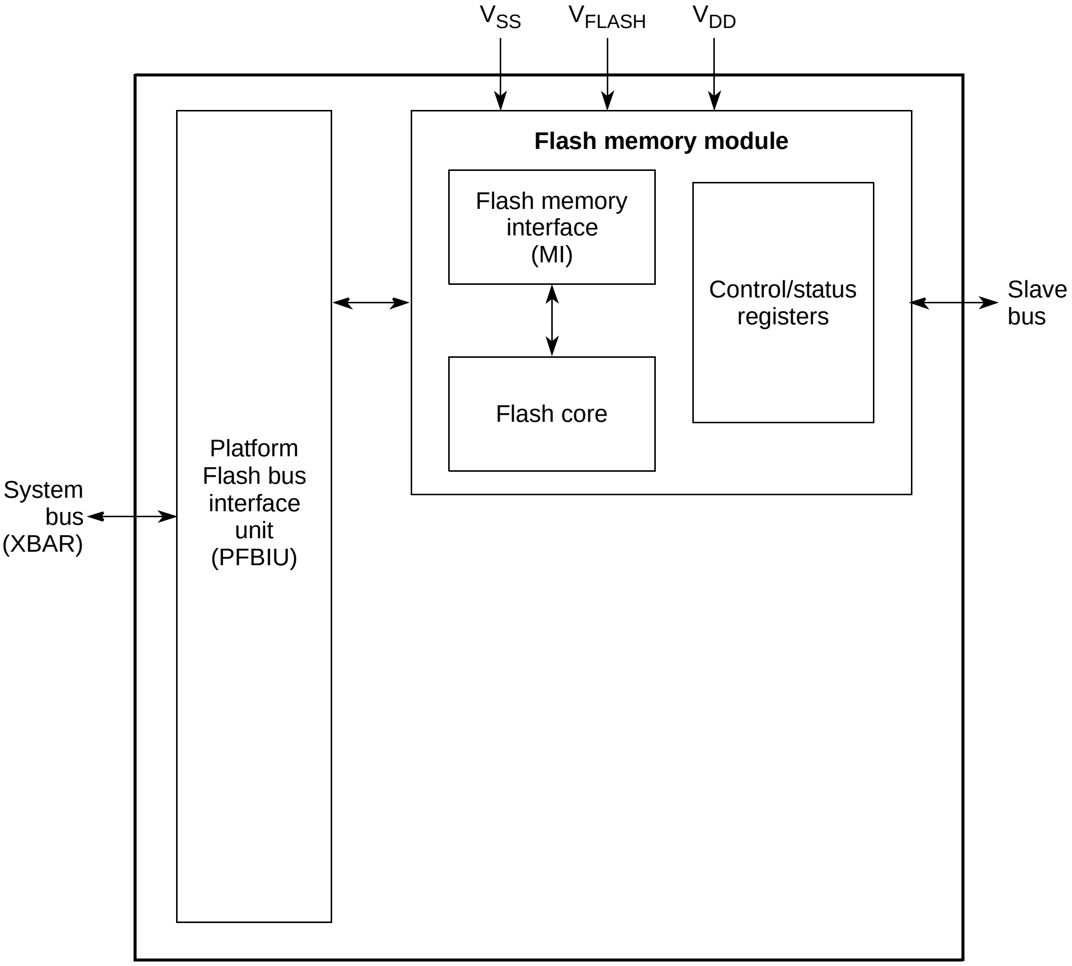


Figure 54. Flash segmentation

11.1.1 Block diagram

Figure 55 shows a block diagram of the flash memory module. The FBIU is addressed through the system bus while the flash control and status registers are addressed through the slave (peripheral) bus.



V_{FLASH} is the only externally visible power supply that is necessary for the programming and erasing of the flash array (see *Section 11.2, External signal description*).

Figure 55. Flash system block diagram

11.1.2 Features

The flash memory module has these major features:

- Support for a 64-bit data bus for instruction fetch
- Support for a 32-bit data bus for CPU loads and DMA access. Byte, halfword, word and doubleword reads are supported. Only aligned word and doubleword writes are supported.
- Configurable read buffering and line prefetch support. Device flash has 1 set of 4 line read buffers for the 128-bit wide low, medium and high address space.
- Hardware and software configurable(d) read and write access protections on a per-master basis
- Interface to the flash array controller is pipelined with a depth of 1, allowing overlapped accesses to proceed in parallel for interleaved or pipelined flash array designs.
- Configurable access timing allowing use in a wide range of system frequencies.
- Multiple-mapping support and mapping-based block access timing (0–31 additional cycles) allowing use for emulation of other memory types
- Software programmable block program/erase restriction control for low, mid and high address spaces
- Erase of selected block(s)
- Read page size of 128 bits
- ECC with single-bit correction, double-bit detection
- Minimum program size is 2 consecutive 32 bit words, aligned on a 0-modulo-8 byte address, due to ECC.
- Embedded hardware program and erase algorithm
- Read-while-write with multiple partitions
- Erase suspend, program suspend and erase-suspended program
- Automotive flash which meets automotive endurance and reliability requirements
- Shadow information stored in non-volatile shadow block
- Independent program/erase of the shadow block

11.1.3 Modes of operation

Flash User mode

User mode is the default operating mode of the flash module. In this mode, it is possible to read and write, program and erase the flash module.

---

d. Software executing from flash must not write to registers that control flash behavior, e.g., wait state settings or prefetch enable/disable. Doing so can cause data corruption. On SPC564A70 devices these registers include BIUCR, BIUAPR, and BIUCR2.Further, flash configuration registers should be written only with 32-bit write operations to avoid any issues associated with register “incoherency” caused by bit fields spanning smaller size (8- and 16-bit) boundaries.

11.2 External signal description

V_{FLASH} is the only externally visible power supply that is necessary for programming and erasing the flash array. The other flash supplies are tied to the appropriate supply pads in the package.

11.3 Memory map and registers

This section provides a detailed description of all flash memory registers.

11.3.1 Module memory map

The flash memory map is shown in *Table 60*. The addresses are given as an offset to the flash memory base address.

There are no program-visible registers that physically reside in the flash. The flash controller contains the registers to control and configure the flash (see *Table 62*). Reference these registers only with 32-bit accesses.

Table 60. Flash memory map

Offset from FLASH_BASE (0x0000_0000)	Use	Block	Partition	Data width (bits)	Block size (Kbytes)
0x0000_0000	Low-address space	L0	1	128	16
0x0000_4000		L1		128	16
0x0000_8000		L2		128	16
0x0000_C000		L3		128	16
0x0001_0000		L4	2	128	16
0x0001_4000		L5		128	16
0x0001_8000		L6		128	16
0x0001_C000		L7		128	16
0x0002_0000		L8	3	128	64
0x0003_0000		L9		128	64
0x0004_0000	Mid-address space	M0	4	128	128
0x0006_0000		M1		128	128
0x0008_0000	High-address space	H0	5	128	256
0x000C_0000		H1		128	256
0x0010_0000		H2	6	128	256
0x0014_0000		H3		128	256
0x0018_0000		H4	7	128	256
0x001C_0000		H5		128	256
0x0020_0000	Reserved				

Table 60. Flash memory map (continued)

Offset from FLASH_BASE (0x0000_0000)	Use	Block	Partition	Data width (bits)	Block size (Kbytes)
0x00FF_C000	Shadow row (Flash A)	S1	All(1)	128	16
0x0100_0000	Reserved				

1. For read-while-write operations, the shadow row behaves as if it is in all partitions.

Table 61. Flash Shadow block mapping

Offset from FLASH_BASE (0x0000_0000)	Use
0x00FF_C000 – 0x00FF_FDD7	General use
0x00FF_FDD8	Serial passcode (0xFEED_FACE_CAFE_BEEF)
0x00FF_FDE0	Censorship control word (0x55AA_55AA)
0x00FF_FDE4	General use
0x00FF_FDE8	LMLR reset configuration (0x00FF_FFFF)
0x00FF_FDEC	General use
0x00FF_FDF0	HLR reset configuration (0x0FFF_FFFF)
0x00FF_FDF4	General use
0x00FF_FDF8	SLMLR reset configuration (0x000F_FFFF)
0x00FF_FDFD	General use
0x00FF_FE10	NVUSR0
0x00FF_FE14 – 0x00FF_FFFF	General use

Table 62. Flash configuration register memory map

Offset from FLASH_*x*_ REGS_BASE(1)	Register	Location
0x0000	MCR—Module configuration register	*on page 11-238*
0x0004	LMLR—Low-/mid-address space block lock register	*on page 11-243*
0x0008	HLR—High-address space block lock register	*on page 11-244*
0x000C	SLMLR—Secondary low/mid-address space block lock register	*on page 11-245*
0x0010	LMSR—Low-/mid-address space block select register	*on page 11-246*
0x0014	HSR—High-address space block select register	*on page 11-247*

Table 62. Flash configuration register memory map (continued)

Offset from FLASH_*x*_ REGS_BASE[1]	Register	Location
0x0018	AR—Address register	*on page 11-248*
0x001C	BIUCR—Bus interface unit configuration register	*on page 11-249*
0x0020	BIUAPR—Bus interface unit access protection register	*on page 11-252*
0x0024	BIUCR2—Bus interface unit configuration register 2	*on page 11-253*
0x0028 – 0x0038	Reserved	
0x003C	FLASH_x_UT0—User Test 0 Register	*on page 11-253*
0x0040	FLASH_x_UT1—User Test 1 Register	*on page 11-255*
0x0044	FLASH_x_UT2—User Test 2 Register	*on page 11-256*
0x0048	UMISR0—User Multiple Input Signature Register 0	*on page 11-257*
0x004C	UMISR1—User Multiple Input Signature Register 1	*on page 11-257*
0x0050	UMISR2—User Multiple Input Signature Register 2	*on page 11-257*
0x0054	UMISR3—User Multiple Input Signature Register 3	*on page 11-257*
0x0058	UMISR4—User Multiple Input Signature Register 4	*on page 11-257*
0x005C – 0x3FFF	Reserved	

1. FLASH_A_REGS_BASE = 0xC3F8_8000

11.3.2 Register descriptions

This section lists the flash memory registers in address order and describes the registers and their bitfields.

Module Configuration Register (MCR)

Figure 56. Module Configuration Register (MCR)

Offset 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	SIZE			0	LAS			0	0	0	MAS
W																
Reset	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	EER	RWE	SBC	0	PEAS	DONE	PEG	0	0	0	0	PGM	PSUS	ERS	ESUS	EHV
W	w1c	w1c	w1c													
Reset	0	0	0	0	0	0/1	1	0	0	0	0	0	0	0	0	0

Table 63. MCR field description

Field	Description
SIZE[2:0]	Array Space Size The value of the SIZE field is dependent upon the size of the flash module. SIZE is read only. 000: 128 KB (Only LAS option for this size is LAS = 2, and consists of two 16 KB and two 48 KB blocks, 128 KB of LAS available, and no MAS or HAS available) 001: 256 KB (Only LAS option for this size is LAS = 1, and LAS = 2, no MAS or HAS available) 010: 512 KB (Any LAS or MAS option is available, no HAS available) 011: 1.0 MB (256 KB of LAS, 256 KB of MAS, and 512 KB of HAS) 100: 1.5 MB (256 KB of LAS, 256 KB of MAS, and 1 MB of HAS) 101: 2.0 MB (256 KB of LAS, 256 KB of MAS, and 1.5 MB of HAS) 110: Reserved 111: Reserved
LAS[2:0]	Low-Address Space The value of the LAS field corresponds to the configuration of the Low-Address Space. LAS is read only. 000: One 256 KB block 001: Two 128 KB blocks 010: Four 16 KB, four 48 KB blocks 011: Reserved 100: Eight 16 KB, two 64 KB blocks 101: Reserved 110: Two 16 KB, two 48 KB, two 64 KB blocks 111: Reserved
MAS	Mid-Address Space The value of the MAS field corresponds to the configuration of the Mid-Address Space. MAS is read only. 0: Two 128 KB blocks 1: One 256 KB block (Only available if LAS = 0)

Table 63. MCR field description (continued)

Field	Description
EER	ECC Event Error EER provides information on previous reads. If a double bit detection occurred, the EER bit is set to a 1. This bit must then be cleared, or a reset must occur before this bit returns to a 0 state. This bit may not be set by the user. In the event of a single bit detection and correction, this bit is not be set. If EER is not set, or remains 0, this indicates that all previous reads (from the last reset, or clearing of EER) are correct. Since this bit is an error flag, it must be cleared to a 0 by writing a 1 to the register location. A write of 0 has no effect. 0: Reads are occurring normally. 1: An ECC Error occurred during a previous read.
RWE	Read While Write Event Error RWE provides information on previous RWW reads. If a Read While Write error occurs, this bit is set to 1. This bit must then be cleared, or a reset must occur before this bit returns to a 0 state. This bit may not be written to a 1 by the user. If RWE is not set, or remains 0, this indicates that all previous RWW reads (from the last reset, or clearing of RWE) are correct. Since this bit is an error flag, it must be cleared to a 0 by writing a 1 to the register location. A write of 0 has no effect. 0: Reads are occurring normally. 1: A Read While Write Error occurred during a previous read.
SBC	Single Bit Correction SBC provides information on previous reads provided the UT0[SPCE] is set. If a single bit correction occurred, the SBC bit is set to a 1. This bit must then be cleared, or a reset must occur before this bit returns to a 0 state. If SBC is not set, or remains 0, this indicates that all previous reads (from the last reset, or clearing of SBC) did not require a correction. Since this bit is an error flag, it must be cleared to a 0 by writing a 1 to the register location. A write of 0 has no effect. 0: Reads are occurring without corrections. 1: A Single Bit Correction occurred during a previous read.
PEAS	Program/Erase Access Space PEAS is used to indicate which space is valid for program and erase operations, either main array space or shadow space. PEAS = 0 indicates that the main address space is active for all FC program and erase operations. PEAS = 1 indicates the shadow address space is active for program/erase. The value in PEAS is captured and held when the shadow block is enabled with the first interlock write done for program or erase operations. The value of PEAS is retained between sampling events (that is, subsequent first interlock writes). The value in PEAS may be changed during erase-suspended program, and reverts back to its' original state once the erase-suspended program is completed. PEAS is read only. 0: Shadow address space is disabled for program/erase and main address space enabled. 1: Shadow address space is enabled for program/erase and main address space disabled.
DONE	State Machine Status DONE indicates if the flash module is performing a high voltage operation. DONE is set to a 1 on termination of the flash module reset. DONE is read only. DONE is cleared within a 0 to 1 transition of EHV which initiates a high voltage operation. DONE is cleared of resuming a suspended operation. DONE is set to a 1 at the end of program and erase high voltage sequences. DONE is set to a 1 within a 1 to 0 transition of EHV which aborts a high voltage operation. 0: Flash is executing a high voltage operation. 1: Flash is not executing a high voltage operation.

Table 63. MCR field description (continued)

Field	Description
PEG	Program/Erase Good The PEG bit indicates the completion status of the last flash program or erase sequence for which high voltage operations were initiated. The value of PEG is updated automatically during the program and erase high voltage operations. Aborting a program/erase high voltage operation causes PEG to be cleared, indicating the sequence failed. PEG is set to a 1 when the module is reset. PEG is read only. The value of PEG is valid only when PGM = 1 and/or ERS = 1 and after DONE transitions from 0 to 1 due to an abort or the completion of a program/erase operation. PEG is valid until PGM/ERS makes a 1 to 0 transition or EHV makes a 0 to 1 transition. The value in PEG is not valid after a 0 to 1 transition of DONE caused by PSUS or ESUS being set to logic 1. If PGM and ERS are both 1 when DONE makes a qualifying 0 to 1 transition the value of PEG indicates the completion status of the PGM sequence. This happens in an erase-suspended program operation. 0: Program or erase operation failed 1: Program or erase operation successful If program or erases are attempted on blocks that are locked, the response from flash is PEG = 1, indicating that the operation was successful, and the contents of the block are properly protected from the program or erase operation.
PGM	Program PGM is used to set up flash for a program operation. A 0 to 1 transition of PGM initiates a program sequence. A 1 to 0 transition of PGM ends the program sequence. PGM can be set only under one of the following conditions: – User mode read (ERS is low and UTE is low) – Erase suspend (ERS and ESUS are 1) with EHV low PGM can be cleared by the user only when PSUS and EHV are low and DONE is high. PGM is cleared on reset. 0: Flash is not executing a program sequence. 1: Flash is executing a program sequence. In an erase-suspended program, programming Flash locations in blocks which were being operated on in the erase may corrupt FC data. This should be avoided due to reliability implications.
PSUS	Program Suspend PSUS is used to indicate the flash module is in program suspend or in the process of entering a suspend state. The module is in program suspend when PSUS = 1 and DONE = 1. PSUS can be set high only when PGM and EHV are high. A 0 to 1 transition of PSUS starts the sequence which sets DONE and places the flash module in program suspend. The module enters suspend within this transition. PSUS can be cleared only when DONE and EHV are high. A 1 to 0 transition of PSUS with EHV = 1 starts the sequence which clears DONE and returns the flash module to program. The module cannot exit program suspend and clear DONE while EHV is low. PSUS is cleared on reset. 0: Program sequence is not suspended. 1: Program sequence is suspended.
ERS	Erase ERS is used to set up flash for an erase operation. A 0 to 1 transition of ERS initiates an erase sequence. A 1 to 0 transition of ERS ends the erase sequence. ERS can only be set only in user mode read (PGM is low and UTE is low). ERS can be cleared by the user only when ESUS and EHV are low and DONE is high. ERS is cleared on reset. 0: Flash is not executing an erase sequence. 1: Flash is executing an erase sequence.

Table 63. MCR field description (continued)

Field	Description
ESUS	Erase Suspend ESUS is used to indicate that the flash module is in erase suspend or in the process of entering a suspend state. The module is in erase suspend when ESUS = 1 and DONE = 1. ESUS can be set high only when ERS and EHV are high and PGM is low. A 0 to 1 transition of ESUS starts the sequence which sets DONE and places the flash in erase suspend. The flash module enters suspend within this transition. ESUS can be cleared only when DONE and EHV are high and PGM is low. A 1 to 0 transition of ESUS with EHV = 1 starts the sequence which clears DONE and returns the module to erase. The flash module cannot exit erase suspend and clear DONE while EHV is low. ESUS is cleared on reset. 0: Erase sequence is not suspended. 1: Erase sequence is suspended.
EHV	Enable High Voltage The EHV bit enables the flash module for a high voltage program/erase operation. EHV is cleared on reset. EHV must be set after an interlock write to start a program/erase sequence. EHV may be set, initiating a program/erase, after an interlock under one of the following conditions: – Erase (ERS = 1, ESUS = 0) – Program (ERS = 0, ESUS = 0, PGM = 1, PSUS = 0) – Erase-suspended program (ERS = 1, ESUS = 1, PGM = 1, PSUS = 0) If a program operation is to be initiated while an erase is suspended the user must clear EHV while in erase suspend before setting PGM. In normal operation, a 1 to 0 transition of EHV with DONE high, PSUS and ESUS low terminates the current program/erase high voltage operation. When an operation is aborted, there is a 1 to 0 transition of EHV with DONE low and the suspend bit for the current program/erase sequence low. An abort causes the value of PEG to be cleared, indicating a failed program/erase; address locations being operated on by the aborted operation contain indeterminate data after an abort. A suspended operation cannot be aborted. EHV may be written during suspend. EHV must be high for the flash module to exit suspend. EHV may not be written after a suspend bit is set high and before DONE transitions high. EHV may not be set low after the current suspend bit is set low and before DONE transitions low. 0: Flash is not enabled to perform a high voltage operation. 1: Flash is enabled to perform a high voltage operation. Aborting a high voltage operation leaves FC addresses in an indeterminate data state. This may be recovered by executing an erase on the affected blocks.

MCR simultaneous register writes

A number of MCR bits are protected against write when another bit, or set of bits, is in a specific state. These write locks are covered on a bit by bit basis in the preceding section. The write locks detailed in the previous section do not consider the effects of trying to write two or more bits simultaneously. The effects of writing bits simultaneously which put the module in an illegal state are detailed here.

The flash module does not allow the user to write bits simultaneously which put the device into an illegal state. This is implemented through a priority mechanism among the bits. The bit changing priorities are detailed in *Table 64*.

Table 64. MCR bit set/clear priority levels

Priority level	MCR bit(s)
1	ERS
2	PGM
3	EHV
4	ESUS, PSUS

If the user attempts to write two or more MCR bits simultaneously then only the bit with the lowest priority level is written. Setting two bits with the same priority level is prevented by existing write locks or do not put the flash in an illegal state.

For example, setting ERS and PGM simultaneously results in only ERS being set. Attempting to clear EHV while setting PSUS results in EHV being cleared, while PSUS is unaffected.

Low/Mid-Address Space Block Lock Register (LMLR)

The Low/Mid-Address Space Block Lock Register (LMLR) provides a means to protect blocks from being modified. These bits, along with the SLLOCK bits in the SLMLR, determine if the block is locked from program or erase. An “OR” of LMLR and SLMLR determines the final lock status.

Note: *A reset value of 1* in Figure 57 indicates that the reset value of these registers is determined by Flash values in the shadow block. An erased shadow block causes the reset value to be 1.*

Figure 57. Low/Mid-Address Space Block Lock Register (LMLR)

Offset 0x0004 Access: User read/write

R	LME	0	0	0	0	0	0	0	0	0	0	SLOCK	0	0	MLOCK	
W		▨	▨	▨	▨	▨	▨	▨	▨	▨	▨		▨	▨		
Reset	0	0	0	0	0	0	0	0	0	0	0	X	0	0	X	X

R	0	0	0	0	0	0	LLOCK									
W	▨	▨	▨	▨	▨	▨										
Reset	0	0	0	0	0	0	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*

▨ = Unimplemented or Reserved

Table 65. LMLR field descriptions

Field	Description
LME	Low/Mid-Address Lock Enable The LME bit is used to enable the Lock fields (SLOCK, MLOCK and LLOCK) to be set or cleared by register writes. LME is a status bit only, and may not be written or cleared, and the reset value is 0. The method to set LME is to write a password, and if the password matches, LME is set to reflect the status of enabled, and is enabled until a reset operation occurs. For LME, the password 0xA1A1_1111 must be written to the LMLR. 0: Low/Mid-Address Locks are disabled, and can not be modified. 1: Low/Mid-Address Locks are enabled to be written.
SLOCK	Shadow Lock This SLOCK bit is used to lock the shadow block from programs and erases. 1: Shadow block is locked for program and erase. 0: Shadow block is available to receive program and erase pulses. SLOCK is not writable once an interlock write is completed until MCR[DONE] is set at the completion of the requested operation. Likewise, SLOCK is not writable if a high voltage operation is suspended. SLOCK is also not writeable during UTest operations, when AIE is high. Upon reset, information from the shadow block is loaded into SLOCK. The SLOCK bit may be written as a register. Reset causes the bits to go back to their shadow block value. The default value of the SLOCK bit (assuming erased shadow location) is locked. SLOCK is not writable unless LME is high.
MLOCK[1:0]	Mid-Address Space Block Lock A value of 1 in a bit of the lock register signifies that the corresponding block is locked for program and erase. A value of 0 in the lock register signifies that the corresponding block is available to receive program and erase pulses. The block numbering for Mid-Address Space starts with MLOCK[0] and continues until all blocks are accounted. The lock register is not writable once an interlock write is completed until MCR[DONE] is set at the completion of the requested operation. Likewise, the lock register is not writable if a high voltage operation is suspended. MLOCK is also not writeable during UTest operations, when AIE is high. Upon reset, information from the shadow block is loaded into the block registers. The LOCK bits may be written as a register. Reset causes the bits to go back to their shadow block value. The default value of the LOCK bits (assuming erased shadow location) is locked. In the event that blocks are not present (due to configuration or total memory size), the LOCK bits default to be locked, and are not writable. The reset value is always 1 (independent of the shadow block), and register writes have no effect. MLOCK is not writable unless LME is high.
LLOCK[9:0]	Low-Address Space Block Lock A value of 1 in a bit of the lock register signifies that the corresponding block is locked for program and erase. A value of 0 in the lock register signifies that the corresponding block is available to receive program and erase pulses. The block numbering for Low-Address Space starts with LLOCK[0] and continues until all blocks are accounted. For more details on LLOCK, please see MLOCK field description. LLOCK is not writable unless LME is high.

High-Address Space Block Lock Register (HLR)

The High-Address Space Block Lock Register (HLR) provides a means to protect blocks from being modified.

Note: *A reset value of 1* in Figure 58 indicates that the reset value of these registers is determined by Flash values in the shadow block. An erased shadow block causes the reset value to be 1.*

Figure 58. High-Address Space Block Lock Register (HLR)

Offset 0x0008 Access: User read/write

R	HBE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

R	0	0	0	0	0	0	0	0	0	0	HBLOCK					
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	1*	1*	1*	1*	1*	1*

▨ = Unimplemented or Reserved

Table 66. HLR field descriptions

Field	Description
HBE	High-Address Lock Enable This bit is used to enable the Lock registers (HBLOCK) to be set or cleared by register writes. This bit is a status bit only, and may not be written or cleared, and the reset value is 0. The method to set this bit is to provide a password, and if the password matches, the HBE bit is set to reflect the status of enabled, and is enabled until a reset operation occurs. For HBE, the password B2B2_2222h must be written to the HLR. 0: High-Address Locks are disabled, and can not be modified. 1: High-Address Locks are enabled to be written.
HBLOCK[5:0]	High-Address Space Block Lock HBLOCK has the same characteristics as LLOCK. Please see this description for more information. The block numbering for High-Address Space starts with HBLOCK[0] and continues until all blocks are accounted. HBLOCK is not writable unless HBE is high.

Secondary Low/Mid-Address Space Block Lock Register (SLMLR)

The Secondary Low/Mid-Address Space Block Lock Register (SLMLR) provides an alternative means to protect blocks from being modified. This has the effect of creating a “tiered” locking scheme to enable different flash users to provide different default locking on blocks. These bits, along with the LLOCK bits in the LMLR, determine if the block is locked from program or erase. An “OR” of LMLR and SLMLR determine the final lock status.

Note: *A reset value of 1* in Figure 59 indicates that the reset value of these registers is determined by Flash values in the shadow block. An erased shadow block causes the reset value to be 1.*

Figure 59. Secondary Low/Mid-Address Space Block Lock Register (SLMLR)

Offset 0x000C Access: User read/write

R	SLE	0	0	0	0	0	0	0	0	0	0	SSLOCK	0	0	SMLOCK	
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨		▨	▨		
Reset	0	0	0	0	0	0	0	0	0	0	0	1*	0	0	1*	1*

R	0	0	0	0	0	0	SLLOCK									
W	▨	▨	▨	▨	▨	▨										
Reset	0	0	0	0	0	0	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*

▨ = Unimplemented or Reserved

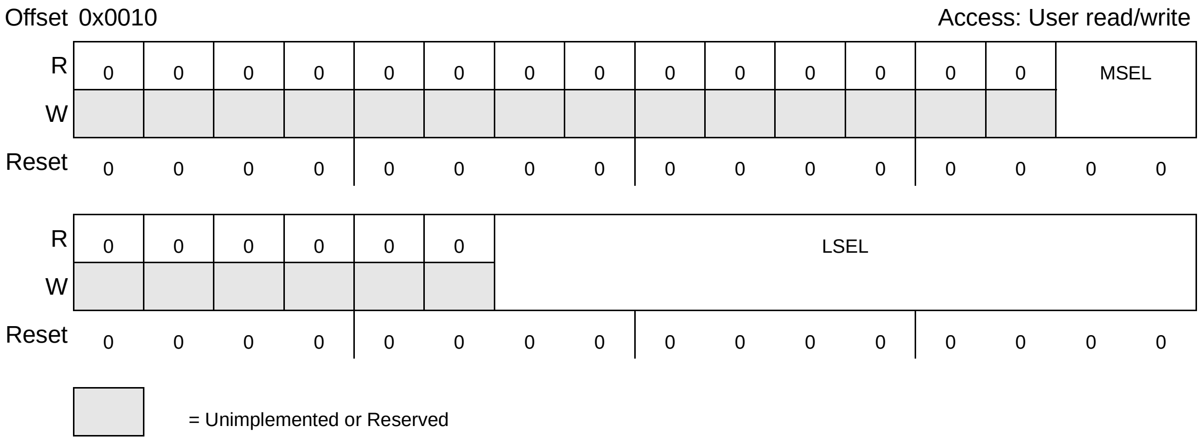
Table 67. SLMLR field descriptions

Field	Description
SLE	Secondary Low/Mid-Address Lock Enable The SLE bit is used to enable the Lock fields (SSLOCK, SMLOCK, and SLLOCK) to be set or cleared by register writes. SLE is a status bit only, and may not be written or cleared, and the reset value is 0. The method to set SLE is to provide a password, and if the password matches, SLE is set to reflect the status of enabled, and is enabled until a reset operation occurs. For SLE, the password 0xC3C3_3333 must be written to the SLMLR. 0: Secondary Low/Mid-Address Locks are disabled, and can not be modified. 1: Secondary Low/Mid-Address Locks are enabled to be written.
SSLOCK	Secondary Shadow Lock The SSLOCK bit is an alternative method that may be used to lock the shadow block from programs and erases. SSLOCK has the same description as SLOCK. SSLOCK is not writable unless SLE is high.
SMLOCK[1:0]	Secondary Mid-Address Block Lock The SMLOCK field is an alternative method that may be used to lock the Mid-Address Space blocks from programs and erases. SMLOCK has the same description as MLOCK. SMLOCK is not writable unless SLE is high.
SLLOCK[9:0]	Secondary Low-Address Block Lock The SLLOCK field is an alternative method that may be used to lock the Low-Address Space blocks from programs and erases. SLLOCK has the same description as LLOCK. SLLOCK is not writable unless SLE is high.

Low/Mid-Address Space Block Select Register (LMSR)

The Low/Mid-Address Space Block Select Register (LMSR) provides a means to select blocks to be operated on during erase.

Figure 60. Low/Mid-Address Space Block Select Register (LMSR)



Offset 0x0010 Access: User read/write

R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	MSEL	
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

R	0	0	0	0	0	0	LSEL									
W	▨	▨	▨	▨	▨	▨										
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Table 68. LMSR field descriptions

Field	Description
MSEL[1:0]	Mid-Address Space Block Select A value of 1 in the select register signifies that the block is selected for erase. A value of 0 in the select register signifies that the block is not selected. The reset value for the select registers is 0, or unselected. The blocks must be selected (or unselected) before doing an erase interlock write as part of the erase sequence. The select register is not writable once an interlock write is completed until MCR[DONE] is set at the completion of the requested operation, or if a high voltage operation is suspended. MSEL is also not writable during UTest operations, when AIE is high. In the event that blocks are not present (due to configuration or total memory size), the corresponding select bits default to unselected, and are not writable. The reset value is always 0, and register writes have no effect.
LSEL[9:0]	Low-Address Space Block Select A value of 1 in the select register signifies that the block is selected for erase. A value of 0 in the select register signifies that the block is not selected. The reset value for the select registers is 0, or unselected. The blocks must be selected (or unselected) before doing an erase interlock write as part of the erase sequence. The select register is not writable once an interlock write is completed until MCR[DONE] is set at the completion of the requested operation, or if a high voltage operation is suspended. LSEL is also not writable during UTest operations, when AIE is high. In the event that blocks are not present (due to configuration or total memory size), the corresponding select bits default to unselected, and are not writable. The reset value is always 0, and register writes have no effect.

High-Address Space Block Select Register (HSR)

The High-Address Space Block Select Register (HSR) provides a means to select blocks to be operated on during erase.

Figure 61. High-Address Space Block Select Register (HSR)

Offset 0x0014 Access: User read/write

R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

R	0	0	0	0	0	0	0	0	0	0	HSEL					
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Table 69. HSR field descriptions

Field	Description
HSEL[26:31]	High-Address Space Block Select High-Address Block Select has the same characteristics as LSEL.

Address Register (AR)

The Address register (AR) provides the first failing address in the event module failures (ECC or PGM/Erase state machine)

Figure 62. Address Register (AR)

Offset 0x0018 Access: User read/write

R	SAD	0	0	0	0	0	0	0	0	0	0	0	0	0	ADDR[14-13]	
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

R	ADDR[12-0]													0	0	0
W	▨													▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Table 70. AR field descriptions

Field	Description
SAD	Shadow Address The SAD bit qualifies the address captured during an ECC Event Error, Single Bit Correction, or State Machine operation. The SAD register is not writable. 0: Address Captured is from Main Array Space. 1: Address Captured is from Shadow Array Space.
ADDR[14:0]	Address The ADDR field provides the first failing address in the event of ECC event error (MCR[EER] set), single bit correction (MCR[SBC] set), as well as providing the address of a failure that may have occurred in a state machine operation (MCR[PEG] cleared). ECC event errors take priority over single bit corrections, which take priority over state machine errors. This is especially valuable in the event of a RWW operation, where the read senses an ECC error or single bit correction, and the state machine fails simultaneously. This address is always a Double Word address that selects 64 bits. The ADDR field is writable, and can be used in the UTEST ECC Logic Check. If the ECC logic check is enabled (UT0[EIE] = 1) then the AR will not update for ECC event error, single bit correction or state machine errors. If MCR[EER] or MCR[SBC] are set, the AR is locked from writing. MCR[PEG] does not affect the writability of the ADDR field.

Bus Interface Unit Configuration Register (BIUCR)

The Bus Interface Unit Configuration Register (BIUCR) is used to specify operation of the dual-flash controller.

Figure 63. Bus Interface Unit Configuration Register (BIUCR)

Offset: FLASH_REGS_BASE + 0x001C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	M6PFE	0	M4PFE	0	0	M1PFE	M0PFE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	APC			WWSC		RWSC			0	DPFEN	0	IFPFEN	0	PFLIM		BFEN
W																
Reset	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0

Table 71. BIUCR field descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>M*n*PFE</td><td>Master *n* prefetch enable
These bits are used to control whether prefetching may be triggered based on the master ID of a requesting master. These bits are cleared by hardware reset.
0: No prefetching may be triggered by this master
1: Prefetching may be triggered by this master
These bits refer to the master ID, not the master port number, as shown in the following:
<table>
<tr><th>Master ID</th><th>Module</th></tr>
<tr><td>0</td><td>z4 Core Instruction</td></tr>
<tr><td>1</td><td>z4 Core Load/Store</td></tr>
<tr><td>4</td><td>eDMA</td></tr>
<tr><td>6</td><td>FlexRay</td></tr>
</table></td></tr>
<tr><td>APC</td><td>Address Pipelining Control
This field is used to control the number of cycles between pipelined access requests.
It must be set to a value corresponding to the operating frequency of the PFLASH(1). Higher operating frequencies require non-zero settings for this field for proper flash operation. This field is set to 0b111 by hardware reset.
000: Accesses may be pipelined back-to-back
001: Access requests require one additional hold cycle
010: Access requests require two additional hold cycles
...
110: Access requests require six additional hold cycles
111: No address pipelining
The settings for APC and RWSC should be the same.</td></tr>
<tr><td>WWSC</td><td>Write Wait State Control
This field is used to control the number of wait-states to be added to the best-case flash array access time for writes. The best-case flash array access time for writes is two cycles. This field must be set to a value corresponding to the operating frequency of the PFLASH(1). Higher operating frequencies require non-zero settings for this field for proper flash operation. This field is set to 0b11 by hardware reset.
00: No additional wait-states are added
01: One additional wait-state is added
10: Two additional wait-states are added
11: Three additional wait-states are added</td></tr>
</table>

Table 71. BIUCR field descriptions (continued)

Field	Description
RWSC	Read Wait State Control This field is used to control the number of wait-states to be added to the best-case flash array access time for reads. The best-case flash array access time for reads is one cycle. This field must be set to a value corresponding to the operating frequency of the PFLASH and the actual read access time of the PFLASH(1). Higher operating frequencies require non-zero settings for this field for proper flash operation. This field is set to 0b111 by hardware reset. 000: No additional wait-states are added 001: One additional wait-state is added ... 111: Seven additional wait-states are added The settings for APC and RWSC should be the same.
DPFEN	Data Prefetch Enable This field enables or disables prefetching initiated by a data read access. This field is cleared by hardware reset. 0: No prefetching is triggered by a data read access 1: Prefetching may be triggered by any data read access
IPFEN	Instruction Prefetch Enable This bit enables or disables prefetching initiated by an instruction read access. This field is cleared by hardware reset. 0: No prefetching is triggered by an instruction read access 1: Prefetching may be triggered by any instruction read access
PFLIM	PFLASH Prefetch Limit This field controls the prefetch algorithm used by the PFLASH prefetch controller. This field defines a limit on the maximum number of sequential prefetches which will be attempted between buffer misses. In all situations when enabled, only a single prefetch is initiated on each buffer miss or hit. This field is cleared by hardware reset. 00: No prefetching or buffering is performed. 01: The referenced line is prefetched on a buffer miss, that is, prefetch on miss. 1x: The referenced line is prefetched on a buffer miss, or the next sequential line is prefetched on a buffer hit (if not already present), that is, prefetch on miss or hit.
BFEN	PFLASH Line Read Buffers Enable This bit enables or disables line read buffer hits. It is also used to invalidate the buffers. This bit is cleared by hardware reset. 0: The line read buffers are disabled from satisfying read requests, and all buffer valid bits are cleared. 1: The line read buffers are enabled to satisfy read requests on hits. Buffer valid bits may be set when the buffers are successfully filled.

1. Valid settings are specified in the device datasheet.

Bus Interface Unit Access Protection Register (BIUAPR)

Figure 64. Bus Interface Unit Access Protection Register (BIUAPR)

Offset: FLASH_REGS_BASE + 0x0020 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	M6AP		0	0	M4AP		0	0	0	0	M1AP		M0AP	
W																
Reset	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

Table 72. BIUAPR field descriptions

Field	Description
M*n*AP	Master *n* Access Protection These fields are used to control whether read and write accesses to the flash are allowed based on the master ID of a requesting master. 00: No accesses may be performed by this master 01: Only read accesses may be performed by this master 10: Only write accesses may be performed by this master 11: Both read and write accesses may be performed by this master These bits refer to the master ID, not the master port number, as shown in the following:

Master ID	Module
0	z4 Core Instruction
1	z4 Core Load/Store
4	eDMA
6	FlexRay

Bus Interface Unit Configuration Register 2 (BIUCR2)

Figure 65. Bus Interface Unit Configuration Register 2 (BIUCR2)

Offset: FLASH_REGS_BASE + 0x0024 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	LBCFG		1	1	1	1	1	1	1	1	1	1	1	1	1	1
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Table 73. BIUCR2 field descriptions

Field	Description
LBCFG	Line Buffer Configuration This field controls the configuration of all the line buffers in the PFLASH controller. The buffers can be organized as a “pool” of available resources, or with a fixed partition between instruction and data buffers. In all cases, when a buffer miss occurs, it is allocated to the least-recently-used buffer within the group and the just-fetched entry then marked as most-recently-used. If the flash access is for the next-sequential line, the buffer is not marked as most-recently-used until the given address produces a buffer hit. This field is initialized by hardware reset to the value contained in address 0x7e00 of the shadow block of the flash array. An erased or unprogrammed flash sets this field to 0b11. This field controls the configuration of both the 4 x 128 and 4 x 256 line buffers. 00: All four buffers are available for any flash access, that is, there is no partitioning of the buffers based on the access type. 01: Reserved 10: The buffers are partitioned into two groups with buffers 0 and 1 allocated for instruction fetches and buffers 2 and 3 for data accesses. 11: The buffers are partitioned into two groups with buffers 0,1,2 allocated for instruction fetches and buffer 3 for data accesses.

User Test 0 (UT0) Register

The User Test 0 (UT0) Register provides a means to control UTest. The UTest mode gives the users of the flash module the ability to perform test features on the flash. This register is only writable when the flash is put into UTest mode by writing a passcode.

Figure 66. User Test 0 (UT0) Register

Offset: FLASH_REGS_BASE + 0x003C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	UTE	SCBE	0	0	0	0	0	0	DSI							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	EA	0	MRE	MRV	EIE	AIS	AIE	AID
W																
Reset	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1

Table 74. UT0 field descriptions

Field	Description
UTE	UTest Enable This status bit gives indication when UTest is enabled. All bits in UT0, UT1, UT2, UMISR0, UMISR1, UMISR2, UMISR3, and UMISR4 are locked when this bit is 0. This bit is not writable to a 1, but may be cleared. The reset value is 0. The method to set this bit is to provide a password, and if the password matches, the UTE bit is set to reflect the status of enabled, and is enabled until it is cleared by a register write. The UTE password will only be accepted if MCR[PGM] = 0 and MCR [ERS] = 0 (program and erase are not being requested). UTE can only be cleared if UT0[AID] = 1, UT0[AIE] and UT0[EIE] = 0. While clearing UTE, writes to set AIE or set EIE will be ignored. For UTE, the password 0xF9F9_9999 must be written to the UT0 register.
SCBE	Single Bit Correction Enable SBC enables Single Bit Correction results to be observed in MCR[SBC]. Also is used as an enable for interrupt signals created by the c90fl module. ECC corrections that occur when SBCE is cleared will not be logged. 0: Single Bit Corrections observation is disabled. 1: Single Bit Correction observation is enabled.
DSI	Data Syndrome Input These bits enable checks of ECC logic by allowing check bits to be input into the ECC logic and then read out by doing array reads or array integrity checks. The DSI[7:0] correspond to the 8 ECC check bits on a double word.
EA	ECC Algorithm. EA is a status bit that provides information about the ECC algorithm used within the Flash. Either a modified Hamming code is used, or a modified Hsiao code is used. 0: Default ECC Algorithm, modified Hamming algorithm. 1: Optional/Alternative ECC Algorithm, modified Hsiao algorithm.
MRE	Margin Read Enable MRE combined with MRV enables Factory Margin Reads to be done. Margin reads are only active during Array Integrity Checks. Normal user reads are not affected by MRE. MRE is not writable if AID is low. 0: Margin reads are not enabled. 1: Margin reads are enabled during Array Integrity Checks.

Table 74. UT0 field descriptions (continued)

Field	Description
MRV	Margin Read Value MRV selects the margin level that is being checked. Margin can be checked to an erased level (MRV = 1) or to a programmed level (MRV = 0). In order for this value to be valid, MRE must also be set. MRV is not writable if AID is low. 0: Zero's margin reads are requested. 1: One's margin reads are requested.
EIE	ECC Data Input Enable EIE enables the input registers (DSI and DAI) to be the source of data for the array. This is useful in the ECC logic check. If this bit is set, data read through a BIU read request will be from the DSI and DAI registers when an address match is achieved to the AR. EIE is not simultaneously writable to a 1 as UTI is being cleared to a 0. 0: Data read is from the flash array. 1: Data read is from the DSI and DAI registers.
AIS	Array Integrity Sequence AIS determines the address sequence to be used during array integrity checks. The default sequence (AIS = 0) is meant to replicate sequences normal "user" code follows, and thoroughly checks the read propagation paths. This sequence is proprietary. The alternative sequence (AIS = 1) is just logically sequential. It should be noted that the time to run a sequential sequence is significantly shorter than the time to run the proprietary sequence. If MRE is set, AIS has no effect. 0: Array integrity sequence is proprietary sequence. 1: Array integrity sequence is sequential.
AIE	Array Integrity Enable AIE set to one starts the array integrity check done on all selected and unlocked blocks. The address sequence selected is determined by bit AIS, and the MISR (UMISR0 through UMISR4) can be checked after the operation is complete, to determine if a correct signature is obtained. Once an Array Integrity operation is requested (AIE = 1), it may be terminated by clearing AIE if the operation has finished (AID = 1) or aborted by clearing AIE if the operation is ongoing (AID = 0). AIE is not simultaneously writable to a 1 as UTI is being cleared to a 0. 0: Array integrity checks are not enabled. 1: Array integrity checks are enabled.
AID	Array Integrity Done AID is cleared upon an Array integrity check being enabled (to signify the operation is ongoing). Once completed, AID is set to indicate that the array integrity check is complete. At this time the MISR (UMISR registers) can be checked. AID can not be written, and is status only. 0: Array integrity check is ongoing. 1: Array integrity check is done.

User Test 1 (UT1) Register

The User Test 1 (UT1) Register provides added controllability to UTest.

Figure 67. User Test 1 (UT1) Register

Offset: FLASH_REGS_BASE + 0x0040 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	DAI															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	DAI															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 75. UT1 field descriptions

Field	Description
DAI [31:0]	Data Array Input These bits enable checks of ECC logic by allowing data bits to be input into the ECC logic and then read out by doing array reads or array integrity checks. The DAI[31:0] correspond to the 32 Array bits representing Word 0 of the double word selected in the AR.

User Test 2 (UT2) Register

Figure 68. User Test 2 (UT2) Register

Offset: FLASH_REGS_BASE + 0x0044 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	DAI															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	DAI															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 76. UT2 field descriptions

Field	Description
DAI [63:32]	Data Array Input These bits enable checks of ECC logic by allowing data bits to be input into the ECC logic and then read out by doing array reads or array integrity checks. The DAI[63:32] correspond to the 32 Array bits representing Word 1of the double word selected in the AR.

User Multiple Input Signature Register [0:4] (UMISR*n*)

The User Multiple Input Signature Registers (UMISR*n*) provide a means to evaluate array integrity.

Figure 69. User Multiple Input Signature Register 0 (UMISR0)

Offset: FLASH_REGS_BASE + 0x0048 Access: User read/write

R	MS[031-016]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

R	MS[015-000]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 70. User Multiple Input Signature Register 1 (UMISR1)

Offset: FLASH_REGS_BASE + 0x004C Access: User read/write

R	MS[063-048]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

R	MS[047-032]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 71. User Multiple Input Signature Register 2 (UMISR2)

Offset: FLASH_REGS_BASE + 0x0050 Access: User read/write

R	MS[095-080]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

R	MS[079-064]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 72. User Multiple Input Signature Register 3 (UMISR3)

Offset: FLASH_REGS_BASE + 0x0054 Access: User read/write

R W	MS[127-112]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

R W	MS[111-096]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 73. User Multiple Input Signature Register 4 (UMISR4)

Offset: FLASH_REGS_BASE + 0x0058 Access: User read/write

R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

R W	MS[143-128]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Table 77. UMISR*n* field descriptions

Field	Description
MS	Multiple Input Signature Register bits The MS bitfields accumulate a signature from an array integrity event. The MISR captures all data fields, as well as ECC fields, and the read transfer error signal. The MISR can be seeded to any value by writing the UMISR registers. The UMISR provides a means to calculate an MISR during Array Integrity operations. The MISR can be represented by the following polynomial: $x^{145} + x^6 + x^5 + x^1 + 1$ The MISR is calculated by taking the previous MISR value and then “exclusive ORing” the new data. In addition the most significant bit (in this case it is MISR[144]), is then “exclusive ORed” into input of MISR[6], MISR[5], MISR[1], and MISR[0]. The result of the “exclusive OR” is shifted left on each read. The MISR is used in Array Integrity operations. If during address sequencing, reads extend into an invalid address location (i.e., greater than the maximum address for a given array size) or locked/unselected blocks, reads are still executed to the array, but the results from the array read are not deterministic. In this instance, the signature is not recalculated and the previous value is retained. After running the user-test-mode margin read (also referenced as factory margin read) sequence on the C90fl flash module, the MISR registers cannot be written such that the next user-test-mode margin read sequence cannot seed the MISRs as desired. This will cause the generated MISRs to be unexpected for the next user margin read sequences, in case customers want to run the user margin read more than once. To be able to write the MISR registers: 1) Assert reset after each user margin read sequence so that MISRs can be written again. 2) Do a dummy program to a locked block after user margin read.

11.4 Functional description

11.4.1 Flash User Mode

In user mode the flash module can be read and written (register writes and interlock writes), programmed or erased. The following subsections define all actions that can be performed in user mode.

11.4.2 Flash Read and Write

The default state of the flash module is read. The main and shadow address space can be read only in the read state. The module configuration register (MCR) is always available for read. The flash module enters the read state on reset. The flash module is in the read state under three sets of conditions:

- The read state is active when PGM = 1 or ERS = 1 in the MCR and high-voltage operation is ongoing (read while write).

Note: Reads done to the partition(s) being operated on (either erased or programmed) will result in an error and the RWE bit in the MCR will be set.

- The read state is active when PGM = 1 and PSUS = 1 in the MCR (program suspend).
- The read state is active when ERS = 1 and ESUS = 1 and PGM = 0 in the MCR (erase suspend).

In Flash memory module, FC reads return 128 bits (1 page). MCR reads return 32 bits of data.

Note: FC reads are done through the BIU. In many cases the BIU will do page buffering to allow sequential reads to be done with higher performance. This can create a data coherency issue that must be handled with software. Data coherency can be an issue after a program, erase, or shadow row operations.

In flash user mode, registers can be written. Array can be written to do interlock writes.

Array reads attempted to invalid locations will result in indeterminate data. Invalid locations occur when addressing is done to blocks that do not exist in non 2^n array sizes.

Interlock writes attempted to invalid locations (due to blocks that do not exist in non 2^n array sizes), will result in an interlock occurring, but attempts to program or erase these blocks will not occur since they are forced to be locked. Erase occurs to selected and unlocked blocks even if the interlock write is to an invalid location.

11.4.3 Read While Write (RWW)

The flash core is divided into partitions. Partitions always comprise two or more blocks. Partitions are used to determine read-while-write (RWW) groupings. While a write (program or erase) is being done within a given partition, a read can be simultaneously executed to any other partition. Partitions are listed in *Table 60*. Each partition in high address space comprises two 256 KB blocks. The shadow block has unique RWW restrictions described in *Section 11.4.7, Flash shadow block*.

The FC is also divided into blocks to implement independent erase or program protection. The shadow block exists outside the normal address space and is programmed, erased, and read independently of the other blocks. The shadow block is included to support systems that require NVM for security or system initialization information.

A software mechanism is provided to independently lock or unlock each block in high-, mid-, and low-address space against program and erase. Two hardware locks are also provided to enable/disable the FC for program/erase. See *Section , Software Locking* for more information.

11.4.4 UTest Mode

UTest mode is a mode that customers can put the flash module in to do specific tests to check the integrity of the Flash module.

Array Integrity Self Check

Array Integrity is checked using a pre-defined address sequence (based on UT0[AIS]), and this operation is executed on selected blocks. The data to be read is customer specific, thus a customer can provide user code into the flash and the correct MISR value is calculated. The customer is free to provide any random or non-random code, and a valid MISR signature is calculated. Once the operations is completed, the results of the reads can be checking by reading the MISR value, to determine if an incorrect read, or ECC detection was noted. Array integrity is controlled by the system clock (IPG), and it is required that the Read Wait States and Address Pipelined control registers in the BIU be set to match the user defined frequency being used.

Note: While Array Integrity is being executed, flash memory array accesses through the BIU should not be requested.

The Array Integrity Check consists of the following sequence of events:

1. Enable UTest mode.
2. Select the block, or blocks to receive array integrity check by writing ones to the appropriate registers in LMS or HBS registers.

Note: Locked Blocks can be tested with Array Integrity if selected in LMS and HBS.

Note: It is not possible to do UTest operations on the shadow block.

3. If desired, Set the UT0[AIS] bit to 1 for sequential addressing only.

Note: For normal integrity checks of the flash memory, sequential addressing is recommended.

If it is required to more fully check the read path (in a diagnostic mode), it is recommend that AIS be left at 0, to use the address sequence that checks the read path more fully, and examine read transitions. This sequence takes more time.

4. Seed the MISR UMISR0 through UMISR4 with desired values.
5. Set the UT0[AIE] bit.
   a) If desired, the Array Integrity operation may be aborted prior to UT0[AID] going high. This may be done by clearing the UT0[AIE] bit and then continuing to the next step. It should be noted that in the event of an aborted array integrity check the MISR registers will contain a signature for the portion of the operation that was completed prior to the abort, and will not be deterministic. Prior to doing another array integrity operation, the UMISR0, UMISR1, UMISR2, and UMISR3 registers may need to be initialized to the desired seed value by doing register writes.
6. Wait until the UT0[AID] bit goes high.
7. Read values in the MISR registers (UMISR0 through UMISR4) to ensure correct signature.
8. Write a logic 0 to the UT0[AIE] bit.

Factory Margin Read

Factory Margin Read must be done following “Initial Factory Conditions”. One Factory Margin Read is allowed per erase.

Factory Margin Read may be done to selected and unlocked blocks by combining UT0[MRE] and UT0[MRV] with the Array Integrity check. If UT0[MRE] is set, UT0[AIS] has no affect, and the reads will be done sequentially.

The data to be read is customer specific, thus a customer can provide user code into the flash and the correct MISR value is calculated. The customer is free to provide any random or non-random code, and a valid MISR signature is calculated. Once the operations is completed, the results of the reads can be checking by reading the MISR value. Factory Margin Read is a self timed event, and is independent of system clocks, or wait states selected. Margin ECC corrections or detections are not done during the Factory Margin Read test:

1. Enable UTest mode.
2. Select the block, or blocks to be receive margin read check by writing ones to the appropriate registers in LMS or HBS/EHS registers. Make sure that selected blocks are also unlocked.

Note: It is not possible to do UTest operations on the shadow block.

Note: It is possible to do User Mode array reads during the Factory Margin Read test, if desired, but the partition rules for Read While Write used during program and erase are in effect during Factory Margin Reads.

3. Set the UT0[MRE] bit.
4. Set the UT0[MRV] bit to desired value depending on it is desired to do One’s Margin or Zero’s Margin.
5. Seed the MISR UMISR0 thru UMISR4 with desired values.
6. Set the UT0[AIE] bit.
   a) If desired, the Margin Read operation may be aborted prior to UT0[AID] going high. This may be done by clearing the UT0[AIE] bit and then continuing to the next step. It should be noted that in the event of an aborted Margin Read check the MISR registers will contain a signature for the portion of the operation that was completed prior to the abort, and will not be deterministic.
7. Wait until the UT0[AID] bit goes high.
8. Read values in the MISR registers (UMISR0 through UMISR4) to ensure correct signature.
9. Write a logic 0 to the UT0[AIE] bit.

Note: If it is desired to do two or more margin reads, and it is desired to re-seed the MISR, a reset must be done between operations. If the subsequent margin reads can be done with the previously calculated MISR value, then a reset is not required.

ECC Logic Check

ECC logic can be checked by providing data to be read in the UT0[DSI], UT1[DAI] and/or UT2[DAI] registers. Then array reads can be done, ensuring expected results. The ECC Logic Check consists of the following sequence of events:

1. Enable UTest mode.
2. Write UT0[EIE] to 1.
3. Write UT0[DSI], UT1[DAI] and/or UT2[DAI] bits to provide data and check bit values to be read. Single or Double bit detections/corrections can be simulated by properly choosing Data and Check Bit combinations.
4. Write double word address to receive the data inputted in step 3 into the ADR register.
5. Reads can now be done through the BIU in a Read Request type fashion. In the event of a BIU read requested from an address that matches the address in the ADR register, expected data, and corrections or detections should be observed based on data written into the UT0[DSI], UT1[DAI] and/or UT2[DAI] registers. MCR[EER] and MCR[SBCSBC] can be checked to evaluate the status of reads done.

Note: In the event of an ECC error or Single Bit Correction, during the ECC Logic Check (UTO[EIE] high), the ADR register will not be loaded, and the address tagged to receive the UT0[DSI], UT1[DAI] and/or UT2[DAI] values will be persevered.

6. Once completed, clear the UT0[EIE] bit to 0.

11.4.5 Flash Programming

Programming changes the value stored in an array bit from logic 1 to logic 0 only. Programming cannot change a stored logic 0 to a logic 1. Addresses in locked/disabled blocks cannot be programmed. The user can program the values in any or all of four words within a page in a single program sequence. Word addresses are selected using bits 3:2 of the page-bound word.

Whenever a program operation occurs, ECC bits are programmed. ECC is handled on a 64-bit boundary. Thus, if only one word in any given 64-bit ECC segment is programmed, the adjoining word (in that segment) should not be programmed because ECC calculation has already completed for that 64-bit segment. Attempts to program the adjoining word will probably result in an operation failure. It is recommended that all programming operations be from 64 bits to 128 bits, and be 64-bit aligned. The programming operation should completely fill selected ECC segments within the page.

Warning: In rare cases “over programming” of a 64 bit ECC segment may be done (EEPROM emulation).

The program operation consists of the following sequence of events:

1. Change the value in the MCR[PGM] bit from a 0 to a 1.

Note: Ensure the block that contains the address to be programmed is unlocked. See Section , Low/Mid-Address Space Block Lock Register (LMLR), Section , High-Address Space Block

Lock Register (HLR) and Section , Secondary Low/Mid-Address Space Block Lock Register (SLMLR) for more information.

2. Write the first address to be programmed in the flash module with the program data. This write is referred to as a program data interlock write. An interlock write may be either be an aligned word or doubleword.
3. If more than one word or doubleword is to be programmed, write each additional address in the page with data to be programmed. This is referred to as a program data write. All unwritten data words default to 0xFFFF_FFFF.
4. Write a logic 1 to the MCR[EHV] bit to start the internal program sequence or skip to step 9 to terminate.
5. Wait until the MCR[DONE] bit goes high.
6. Confirm MCR[PEG] = 1.
7. Write a logic 0 to the MCR[EHV] bit.
8. If more addresses are to be programmed, return to step 2.
9. Write a logic 0 to the MCR[PGM] bit to terminate the program sequence.

The program sequence is presented graphically in *Figure 74*. The program suspend operation detailed in *Figure 74* is discussed in *Section , Flash Program Suspend/Resume*.

The first write after a program is initiated determines the page address to be programmed. Program may be initiated with the 0 to 1 transition of the MCR[PGM] bit or by clearing the MCR[EHV] bit at the end of a previous program. This first write is referred to as an interlock write. If the program is not an erase-suspended program, the interlock write determines if the shadow or normal array space will be programmed and causes MCR[PEAS] to be set/cleared.

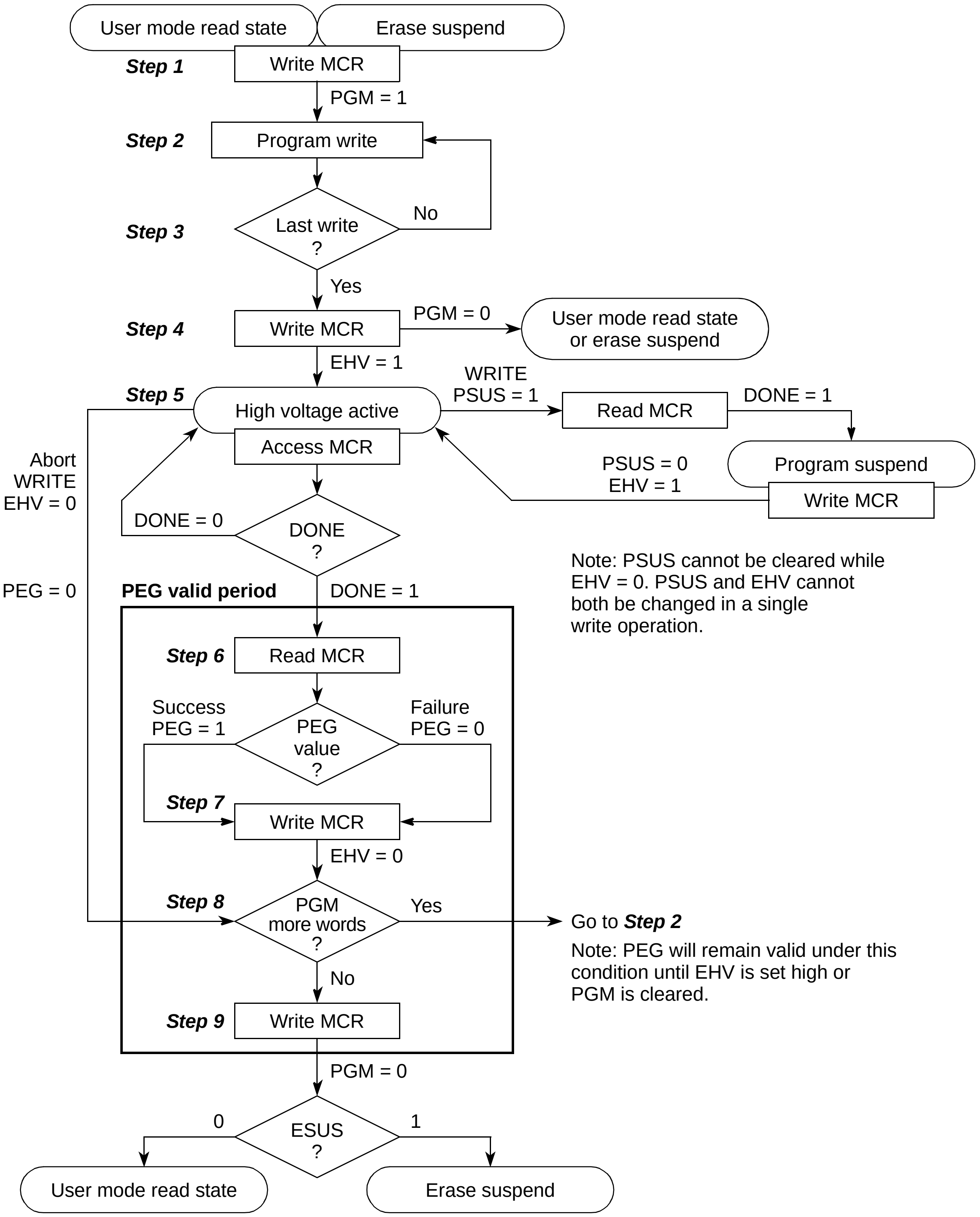
In the case of an erase-suspended program, the value in MCR[PEAS], is retained from the erase.

An interlock write must be performed before setting MCR[EHV]. The user may terminate a program sequence by clearing MCR[PGM] prior to setting MCR[EHV].

If multiple writes are done to the same location the data for the last write is used in programming.

While MCR[DONE] is low, MCR[EHV] is high, and MCR[PSUS] is low, the user may clear MCR[EHV], resulting in a program abort. A program abort forces the module to step 8 of the program sequence. An aborted program will result in MCR[PEG] being set low, indicating a failed operation. The data space being operated on before the abort will contain indeterminate data. The user may not abort a program sequence while in program suspend.

Caution: Aborting a program operation will leave the flash core addresses being programmed in an indeterminate data state. This may be recovered by executing an erase on the affected blocks.



Note: PSUS cannot be cleared while EHV = 0. PSUS and EHV cannot both be changed in a single write operation.

Go to **Step 2**

Note: PEG will remain valid under this condition until EHV is set high or PGM is cleared.

Figure 74. Program Sequence

Software Locking

A software mechanism is provided to independently lock/unlock each high-, mid-, and low-address space against program and erase.

Software locking is done through the LMLR (low/mid-address space block lock register), SLMLR (secondary low/mid-address space block lock register), or HLR (high-address space block lock register). These can be written through register writes and read through register reads.

When the program/erase operations are enabled through hardware, software locks are enforced through doing register writes.

Flash Program Suspend/Resume

The program sequence may be suspended to allow read access to the flash core. It is not possible to erase or program during a program suspend. Interlock writes should not be attempted during program suspend.

A program suspend can be initiated by changing the value of the MCR[PSUS] bit from a 0 to a 1. MCR[PSUS] can be set high at any time when MCR[PGM] and MCR[EHV] are high. A 0 to 1 transition of MCR[PSUS] causes the flash module to start the sequence to enter program suspend, which is a read state. The module is not suspended until MCR[DONE] = 1. At this time flash core reads may be attempted. After it is suspended, the flash core may be read only. Reads to the blocks being programmed/erased return indeterminate data.

The program sequence is resumed by writing a logic 0 to MCR[PSUS]. MCR[EHV] must be set to a 1 before clearing MCR[PSUS] to resume operation. When the operation resumes, the flash module continues the program sequence from one of a set of predefined points. This may extend the time required for the program operation.

11.4.6 Flash Erase

Erase changes the value stored in all bits of the selected block(s) to logic 1. An erase sequence operates on any combination of blocks in the Low, Mid or High Address Space, or the shadow block. The erase sequence is fully automated within the flash. The user only needs to select the blocks to be erased and initiate the erase sequence. Locked/disabled blocks cannot be erased. If multiple blocks are selected for erase during an erase sequence, the blocks are erased sequentially starting with the lowest numbered block and terminating with the highest. The erase sequence consists of the following sequence of events:

The erase sequence consists of the following sequence of events:

1. Change the value in the MCR[ERS] bit from 0 to a 1.
2. Select the block, or blocks, to be erased by writing 1s to the appropriate bits in LMSR or HSR. If the shadow row is to be erased, this step may be skipped, and LMSR and HSR are ignored. For shadow row erase, see *Section 11.4.7, Flash shadow block* for more information.

Note: Lock and select are independent. If a block is selected and locked, no erase will occur. See Section , Low/Mid-Address Space Block Lock Register (LMLR), Section , High-Address

Space Block Lock Register (HLR) and Section , Secondary Low/Mid-Address Space Block Lock Register (SLMLR) for more information.

3. Write to any address in flash. This is referred to as an erase interlock write. The interlock write causes the values of SOC specific shadow enable to be captured and causing MCR[PEAS] to be set/cleared.
4. Write a logic 1 to the MCR[EHV] bit to start an internal erase sequence or skip to step 9 to terminate.
5. Wait until the MCR[DONE] bit goes high.
6. Confirm MCR[PEG] = 1.
7. Write a logic 0 to the MCR[EHV] bit.
8. If more blocks are to be erased, return to step 2.
9. Write a logic 0 to the MCR[ERS] bit to terminate the erase.

The erase sequence is presented graphically in *Figure 75*. The erase suspend operation detailed in *Figure 75* is discussed in *Section , Flash erase suspend/resume*.

After setting MCR[ERS], one write, referred to as an interlock write, must be performed before MCR[EHV] can be set to a 1. This interlock causes the values of SOC specific shadow enable to be captured. Data words written during erase sequence interlock writes are ignored. The user may terminate the erase sequence by clearing MCR[ERS] before setting MCR[EHV].

An erase operation may be aborted by clearing MCR[EHV] assuming MCR[DONE] is low, MCR[EHV] is high, and MCR[ESUS] is low. An erase abort forces the module to step 8 of the erase sequence. An aborted erase results in MCR[PEG] being set low, indicating a failed operation. The blocks being operated on before the abort contain indeterminate data. The user may not abort an erase sequence while in erase suspend.

Warning: Aborting an erase operation will leave the flash core blocks being erased in an indeterminate data state. This may be recovered by executing an erase on the affected blocks.

Flash erase suspend/resume

The erase sequence may be suspended to allow read access to the FC. The erase sequence may also be suspended to program (erase-suspended program) the FC. A program started during erase suspend can in turn be suspended. Only one erase suspend and one program suspend are allowed at a time during an operation. It is not possible to erase during an erase suspend, or program during a program suspend. During suspend, all reads to FC locations targeted for program and blocks targeted for erase return indeterminate data. Programming locations in blocks targeted for erase during erase-suspended program may result in corrupted data. Read While Write may also be used to read the array during an erase sequence providing the read is to a partition not selected for erase.

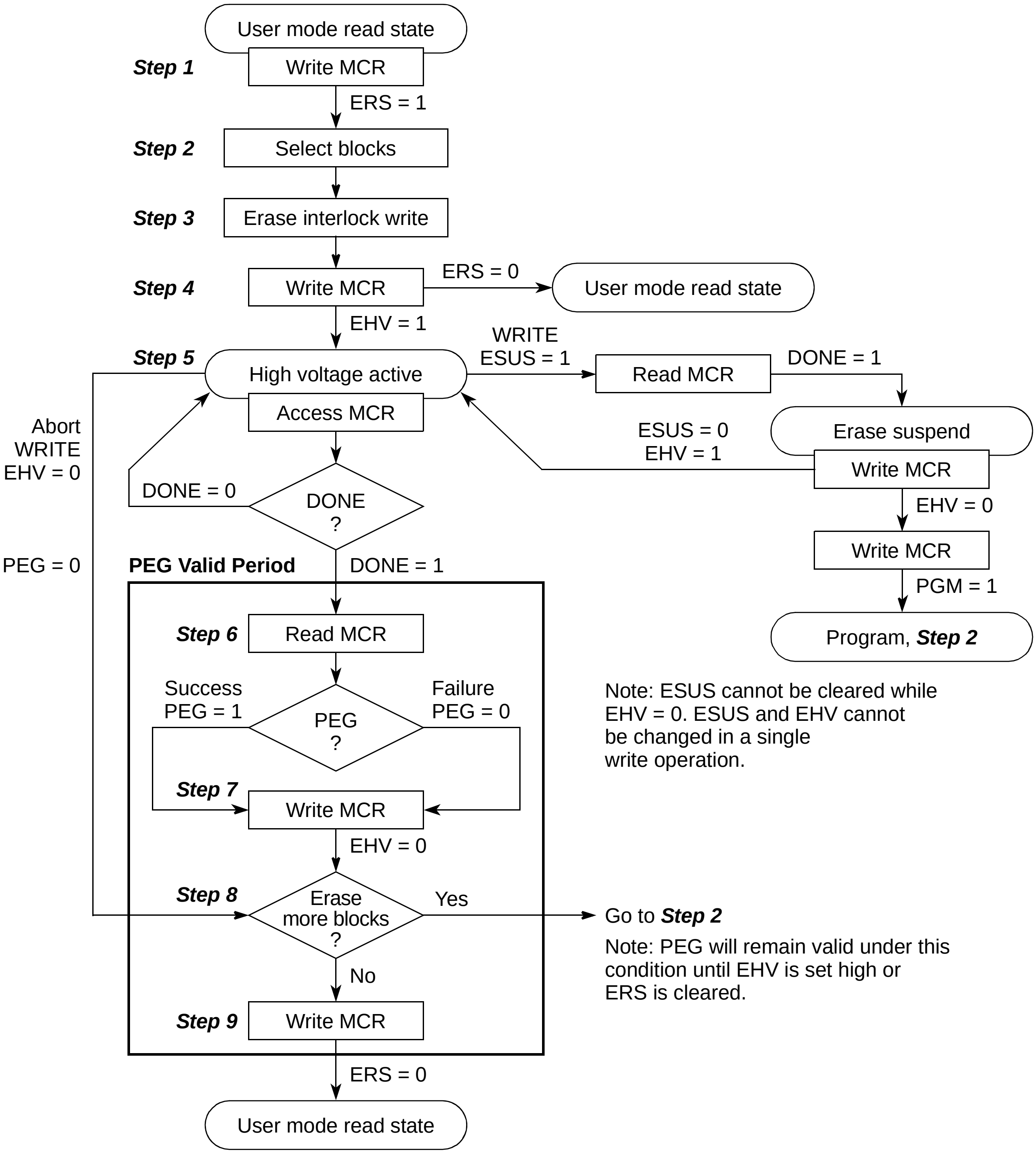
An erase suspend can be initiated by changing the value of the MCR[ESUS] bit from a 0 to a 1.

MCR[ESUS] can be set to a 1 at any time when MCR[ERS] and MCR[EHV] are high and MCR[PGM] is low. A 0 to 1 transition of MCR[ESUS] causes the module to start the sequence which places it in erase suspend. The user must wait until MCR[DONE] = 1 before the module is suspended and further actions are attempted. MCR[DONE] goes high no more than T_{esus} after MCR[ESUS] is set to a 1. Once suspended, the array may be read or a program sequence may be initiated (erase-suspended program). Before initiating a program sequence the user must first clear MCR[EHV]. If a program sequence is initiated the values of SOC specific shadow enable is recaptured. Once the erase-suspended program is completed, the value of PEAS is returned to its “erase” value. FC reads while MCR[ESUS] = 1 from the blocks being erased return indeterminate data.

The erase sequence is resumed by writing a logic 0 to MCR[ESUS]. MCR[EHV] must be set to a 1 and MCR[PGM] must be cleared (in the event of an erase suspended program) before MCR[ESUS] can be cleared to resume the operation. The module continues the erase sequence from one of a set of predefined points. This may extend the time required for the erase operation.

Warning: Repeated suspends at a high frequency may result in the operation timing out, and the flash module will respond by completing the operation with a fail code (MCR[PEG] = 0), or the operation not able to finish (MCR[DONE] = 1 during Erase operation). The minimum time between erase suspends to ensure this does not occur is T_{esrt}.

Warning: In an erase-suspended program, programming flash locations in blocks which were being operated on in the erase may corrupt flash core data.



Note: ESUS cannot be cleared while EHV = 0. ESUS and EHV cannot be changed in a single write operation.

Note: PEG will remain valid under this condition until EHV is set high or ERS is cleared.

Figure 75. Erase sequence

11.4.7 Flash shadow block

The flash shadow block is a memory-mapped block in the flash memory map. Program and erase of the shadow block are enabled when MCR[PEAS] = 1 only. After the user has begun an erase operation on the shadow block, the operation cannot be suspended to program the main address space and vice-versa. The user must terminate the shadow erase operation to program or erase the main address space.

Note: If an erase of user space is requested, and a suspend is done with attempts to erase suspend program shadow space, this attempted program will be directed to user space as dictated by the state of MCR[PEAS]. Likewise an attempted erase suspended program of user space, while the shadow space is being erased, will be directed to shadow space as dictated by the state of MCR[PEAS].

The shadow block cannot use the RWW feature. After an operation is started in the shadow block, a read cannot be done to the shadow block, or any other block. Likewise, after an operation is started in a block in low-/mid-/high-address space, a read cannot be done in the shadow block.

The shadow block contains information about how the lock registers are reset. The first and second words can be used for reset configuration words. All other words can be used for user-defined functions or other configuration words.

The shadow block may be locked/unlocked against program or erase by using the LMLR or SLMLR discussed in *Section 11.3.2, Register descriptions*.

Programming the shadow row has similar restrictions to programming the array in terms of how ECC is calculated. See *Section 11.4.5, Flash Programming* for more information. Only one program is allowed per 64-bit ECC segment between erases. Erase of the shadow row is done similarly as an array erase. See *Section 11.4.6, Flash Erase* for more information.

11.4.8 Flash reset

A reset is the highest priority operation for the flash and terminates all other operations.

The flash uses reset to initialize register and status bits to their default reset values. If the flash is executing a program or erase operation and a reset is issued, the operation will be aborted and the flash will disable the high voltage logic without damage to the high-voltage circuits. Reset aborts all operations and forces the flash into user mode ready to receive accesses.

After reset is negated, register accesses can be performed, although it should be noted that registers that require updating from shadow information, or other inputs, cannot read updated values until flash exits reset.

11.4.9 DMA requests

The flash has no DMA requests.

11.4.10 Interrupt requests

The flash has no interrupt requests.

12 General-Purpose Static RAM (SRAM)

12.1 Introduction

The SPC564A70 includes 128 Kbytes of general-purpose SRAM. The first 32 Kbytes of SRAM is powered by its own power supply pin during standby operation.

12.2 Features

The SRAM controller includes these features:

- Supports read/write accesses mapped to the SRAM memory from any master
- 32-Kbyte block powered by separate supply for standby operation
- Byte, halfword, word and doubleword addressable
- 7-bit ECC

12.3 Modes of operation

12.3.1 Normal (Functional) mode

Allows reads and writes of the SRAM memory arrays.

12.3.2 Standby mode

Preserves contents of the standby portion of the memory when the 1.2 V (V_{DD}) power drops below the level of the standby power supply voltage. There are two possible supplies for standby: 1.0 V directly from the VSTBY pin and 2 – 5 volts (also on the VSTBY pin), which enables a standby regulator.

VSTBY pad needs an external RC to slower the ramp to atleast 100 us to prevent a false ESD trigger.

Updates to the standby portion of the SRAM are inhibited during system reset or during Standby Mode.

12.4 Block diagram

The SRAM block diagram is shown in *Figure 76*.

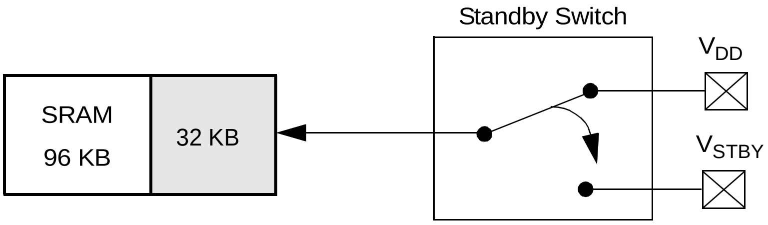


Figure 76. SRAM Block Diagram

12.5 External signal description

The external signal for SRAM is the V_{STBY} RAM power supply. If the standby feature of the SRAM is not used, tie the V_{STBY} pin to V_{SS}.

12.6 Register memory map

The SRAM occupies 128 Kbytes of memory starting at the base address as shown in *Table 78*.

Table 78. SRAM memory map

Address	Register Name	Register Description	Size
Base (0x4000_0000)	—	SRAM powered by V_{STBY}	32 KB
Base + 0x8000	—	96-KB RAM	96 KB

The internal SRAM has no registers. Registers for the SRAM ECC are located in the ECSM. See *Chapter 17: Error Correction Status Module (ECSM)*.

Note: The ECSM module contains the register MUDCR that enables SRAM to be configured with an additional wait state. This is required when the CPU is configured to operate at its maximum frequency. See Section 17.4.3, Miscellaneous User-Defined Control Register (ECSM_MUDCR), for details.

12.7 Functional description

ECC checks are performed during the read portion of an SRAM ECC read/write (R/W) operation, and ECC calculations are performed during the write portion of a read/write (R/W) operation. Because the ECC bits can contain random data after the device is powered on, you must initialize the SRAM by executing 32-bit write instructions to the entire SRAM. For more information, see *Section 12.9, Initialization and application information*.

12.8 SRAm ecc mechanism

The SRAM ECC detects the following conditions and produces the following results:

- Detects and corrects all 1-bit errors
- Detects and flags all 2-bit errors as non-correctable errors

SRAM does not detect all errors greater than 2 bits. Internal SRAM writes are done on byte boundaries:

- 1 byte (0:7 bits)
- 2 bytes (0:15 bits)
- 4 bytes or 1 word (0:31 bits)

If the entire 32 data bits are written to SRAM, no read operation is performed and the ECC is calculated across the 32 bits of data. The 7-bit ECC is appended to the data segment and written to SRAM. If the write operation is less than the entire 32-bit data width (1- or 2-byte segment), the following occurs:

1. The ECC mechanism checks the entire 32 bits of data for errors, detecting and either correcting or flagging errors.
2. The write data bytes (1- or 2-byte segment) are merged with the corrected 32 bits on the data bus.
3. The ECC is then calculated on the resulting 32 bits formed in the previous step.
4. The 7-bit ECC result is appended to the 32 bits from the data, and the 39-bit value is then written to SRAM.

12.8.1 Access timing

The system bus is a two stage pipelined bus, which makes the timing of any access dependent on the access during the previous clock. *Table 79* shows the wait states for accesses, current is the access being measured, previous is the RAM access during the previous clock.

Table 79. Number of wait states required for RAM operation

Current operation	Previous operation	Number of wait states
Read	Idle	0(1) / 1(2)
	Read	0(1)/ 1(2)
	32 or 64-bit write	0(1) / 1(2)
	8 or 16-bit write	1(1) / 2(2)
32 or 64-bit write	Idle	0
	Read	0
	32 or 64-bit write	0
	8 or 16-bit write	1
8 or 16-bit write	Idle	0
	Read	0
	32 or 64-bit write	0
	8 or 16-bit write	1

1. Applies if additional SRAM read wait state in ECSM_MUDCR is disabled
2. Applies if additional SRAM read wait state in ECSM_MUDCR is enabled

12.8.2 Reset effects on SRAM accesses

If a reset event asserts during a read or write operation to SRAM, the completion of that access depends on the cycle at which the reset occurs. Data read from or written to SRAM before the reset event occurred is retained, and no other address locations are accessed or changed.

If the system SRAM is cached, cache lines can retain indeterminate data that is not written to memory unless the region is set for write-through mode.

Note: Standby memory can contain the previous data values if a reset occurs while cache is running in copy back mode.

12.9 Initialization and application information

To use the SRAM, the ECC must check all bits that require initialization after power on. Use either a 32-bit or 64-bit cache-inhibited write to each SRAM location to initialize the SRAM array as part of the application initialization code. All writes must specify an even number of registers performed on 32-bit or 64-bit word-aligned boundaries respectively. If the write is not the entire 32 bits (8 or 16 bits), a read/modify/write operation is generated that checks the ECC value upon the read. See *Section 12.8, SRAm ecc mechanism*.

*Note: You **must** initialize SRAM, even if the application does not use ECC reporting.*

12.9.1 Example code

To initialize SRAM correctly, use a store multiple word (**stmw)** instruction to implement 64-bit writes to all SRAM locations. The **stmw** instruction concatenates two 32-bit registers to implement a single 64-bit write. To ensure the writes are 64 bits, specify an even number of registers and write on 64-bit word-aligned boundaries.

The following example code illustrates the use of the **stmw** instruction to initialize the SRAM ECC bits.

```
init_RAM:
lis   r11,0x4000 # base address of the SRAM, 64-bit word aligned
ori   r11,r11,0  # not needed for this address but could be for others
li    r12,1536   # loop counter to get all of SRAM;
                 # 192*1024/4 bytes/32 GPRs =1536
mtctr r12
init_ram_loop:
stmw r0,0(r11)   # write all 32 GPRs to SRAM
addi r11,r11,128 # inc the ram ptr; 32 GPRs * 4 bytes = 128
bdnz init_ram_loop # loop for 192K of SRAM
blr              # done
```

13 Memory Protection Unit (MPU)

13.1 Introduction

The memory protection unit (MPU) provides hardware access control for all memory references generated in a device. Using preprogrammed region descriptors that define memory spaces and their associated access rights, the MPU concurrently monitors all system bus transactions and evaluates the appropriateness of each transfer. Memory references with sufficient access control rights are allowed to complete, but references that are not mapped to any region descriptor or have insufficient rights are terminated with a protection error response.

The MPU implements a set of program-visible region descriptors that monitor all system bus addresses. The result is a hardware structure with a two-dimensional connection matrix, where the region descriptors represent one dimension and the individual system bus addresses and attributes are the second dimension.

13.1.1 Features

The MPU has these major features:

- Support for 16 memory region descriptors, each 128 bits in size
  - Specification of start and end addresses provide granularity for region sizes from 32 bytes to 4 GB
  - MPU is invalid at reset, thus no access restrictions are enforced
  - 2 types of access control definitions: processor core bus master supports the traditional {read, write, execute} permissions with independent definitions for supervisor and user mode accesses; the remaining non-core bus masters (eDMA and FlexRay) support {read, write} attributes
  - Automatic hardware maintenance of the region descriptor valid bit removes issues associated with maintaining a coherent image of the descriptor
  - Alternate memory view of the access control word for each descriptor provides an efficient mechanism to dynamically alter the access rights of a descriptor only
  - For overlapping region descriptors, priority is given to permission granting over access denying as this approach provides more flexibility to system software
- Support for two XBAR slave port connections (SRAM and PBRIDGE)
  - For each connected XBAR slave port (SRAM and PBRIDGE), MPU hardware monitors every port access using the preprogrammed memory region descriptors
  - An access protection error is detected if a memory reference does not hit in any memory region or the reference is flagged as illegal in all memory regions where it does hit. In the event of an access error, the XBAR reference is terminated with an error response and the MPU inhibits the bus cycle being sent to the targeted slave device
  - 64-bit error registers, one for each XBAR slave port, capture the last faulting address, attributes, and detail information

13.1.2 Modes of operation

The MPU does not support any special modes of operation.

13.2 MPU-to-XBAR slave port mapping

In some of the register field descriptions, references are made to “slave ports”. This is not referring to the slave ports of the XBAR—it refers instead to the slave ports of the MPU. The mapping is as follows:

Table 80. MPU-to-XBAR slave port mapping

MPU Slave Port	XBAR Slave Port	Description
0	2	Device SRAM
1	7	Device Peripheral Bridge (PBRIDGE)

13.3 Signal description

The MPU does not include any external signals.

13.4 Memory map and registers

This section provides a detailed description of all MPU registers.

13.4.1 Module memory map

The MPU memory map is shown in *Table 81*. The address of each register is given as an offset to the MPU base address. Registers are listed in address order, identified by complete name and mnemonic, and list the type of accesses allowed.

The MPU registers can be referenced using 32-bit (word) accesses only. Attempted references using different access sizes, to undefined (reserved) addresses, or with a non-supported access type (for example, a write to a read-only register or a read of a write-only register) generate an error termination.

Table 81. MPU Memory Map

Offset from MPU_BASE (0xFFF1_0000)	Register	Location
0x0000	MPU_CESR — MPU control/error status register	*on page 13-279*
0x0004–0x000F	Reserved	
0x0010	MPU_EAR0 — MPU error address register, slave port 0	*on page 13-280*
0x0014	MPU_EDR0 — MPU error detail register, slave port 0	*on page 13-281*
0x0018	MPU_EAR1 — MPU error address register, slave port 1	*on page 13-280*
0x001C	MPU_EDR1 — MPU error detail register, slave port 1	*on page 13-281*
0x0020	Reserved	

Table 81. MPU Memory Map (continued)

Offset from MPU_BASE (0xFFF1_0000)	Register	Location
0x0024	Reserved	
0x0028–0x03FF	Reserved	
0x0400	MPU_RGD0 — MPU region descriptor 0	*on page 13-282*
0x0410	MPU_RGD1 — MPU region descriptor 1	*on page 13-282*
0x0420	MPU_RGD2 — MPU region descriptor 2	*on page 13-282*
0x0430	MPU_RGD3 — MPU region descriptor 3	*on page 13-282*
0x0440	MPU_RGD4 — MPU region descriptor 4	*on page 13-282*
0x0450	MPU_RGD5 — MPU region descriptor 5	*on page 13-282*
0x0460	MPU_RGD6 — MPU region descriptor 6	*on page 13-282*
0x0470	MPU_RGD7 — MPU region descriptor 7	*on page 13-282*
0x0480	MPU_RGD8 — MPU region descriptor 8	*on page 13-282*
0x0490	MPU_RGD9 — MPU region descriptor 9	*on page 13-282*
0x04A0	MPU_RGD10 — MPU region descriptor 10	*on page 13-282*
0x04B0	MPU_RGD11 — MPU region descriptor 11	*on page 13-282*
0x04C0	MPU_RGD12 — MPU region descriptor 12	*on page 13-282*
0x04D0	MPU_RGD13 — MPU region descriptor 13	*on page 13-282*
0x04E0	MPU_RGD14 — MPU region descriptor 14	*on page 13-282*
0x04F0	MPU_RGD15 — MPU region descriptor 15	*on page 13-282*
0x00500–0x07FF	Reserved	
0x0800	MPU_RGDAAC0 — MPU RGD alternate access control 0	*on page 13-287*
0x0804	MPU_RGDAAC1 — MPU RGD alternate access control 1	*on page 13-287*

Table 81. MPU Memory Map (continued)

Offset from MPU_BASE (0xFFF1_0000)	Register	Location
0x0808	MPU_RGDAAC2 — MPU RGD alternate access control 2	*on page 13-287*
0x080C	MPU_RGDAAC3 — MPU RGD alternate access control 3	*on page 13-287*
0x0810	MPU_RGDAAC4 — MPU RGD alternate access control 4	*on page 13-287*
0x0814	MPU_RGDAAC5 — MPU RGD alternate access control 5	*on page 13-287*
0x0818	MPU_RGDAAC6 — MPU RGD alternate access control 6	*on page 13-287*
0x081C	MPU_RGDAAC7 — MPU RGD alternate access control 7	*on page 13-287*
0x0820	MPU_RGDAAC8 — MPU RGD alternate access control 8	*on page 13-287*
0x0824	MPU_RGDAAC9 — MPU RGD alternate access control 9	*on page 13-287*
0x0828	MPU_RGDAAC10 — MPU RGD alternate access control 10	*on page 13-287*
0x082C	MPU_RGDAAC11 — MPU RGD alternate access control 11	*on page 13-287*
0x0830	MPU_RGDAAC12 — MPU RGD alternate access control 12	*on page 13-287*
0x0834	MPU_RGDAAC13 — MPU RGD alternate access control 13	*on page 13-287*
0x0838	MPU_RGDAAC14 — MPU RGD alternate access control 14	*on page 13-287*
0x083C	MPU_RGDAAC15 — MPU RGD alternate access control 15	*on page 13-287*
0x0840–0x3FFF	Reserved	

13.4.2 Register descriptions

This section lists the MPU registers in address order and describes the registers and their bitfields.

Note: *The programming model can only be referenced using 32-bit (word) accesses. Attempted references using different access sizes, to undefined (reserved) addresses, or with a non-supported access type (for example, a write to a read-only register or a read of a write-only register) generate a bus error termination.*

MPU Control/Error Status Register (MPU_CESR)

The MPU_CESR provides one byte of error status and three bytes of configuration information. A global MPU enable/disable bit is also included in this register.

Figure 77. MPU Control/Error Status Register (MPU_CESR)

Address: MPU_BASE (0xFFF1_0000) + 0x0000 Access: Supervisor

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SPERR[0:7](1)												HRL			
W									▨	▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	NSP				NRGD											VLD
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	
Reset	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0

▨ = not implemented

1. Each SPERR bit can be cleared by writing a one to the bit location.

Table 82. MPU_CESR field descriptions

Field	Description
0–7 SPERR	Slave Port *n*(1) Error, where the slave port number matches the bit number Each bit in this read-only field represents a flag maintained by the MPU for signaling the presence of a captured error contained in the MPU_EAR*n* and MPU_EDR*n* registers. The individual bit is set when the hardware detects an error and records the faulting address and attributes. It is cleared when the corresponding bit is written to a logical one. If another error is captured at the exact same cycle as a write of a logical one, this flag remains set. A find-first-one instruction (or equivalent) can be used to detect the presence of a captured error. 0 The corresponding MPU_EAR*n*/MPU_EDR*n* registers do not contain an unread captured error 1 The corresponding MPU_EAR*n*/MPU_EDR*n* registers do contain an unread captured error Bit 0 indicates an SRAM access protection error and bit 1 a peripheral bridge protection error.
12–15 HRL	Hardware Revision Level This 4-bit read-only field specifies the MPU's hardware and definition revision level. It can be read by software to determine the functional definition of the module. This field reads as 0 on SPC564A70.
16–19 NSP	Number of Slave Ports This 4-bit read-only field specifies the number of slave ports [1–8] connected to the MPU. This field reads as 0b0010 on the SPC564A70 at reset, indicating two slaves.

Table 82. MPU_CESR field descriptions (continued)

Field	Description
20–23 NRGD	Number of Region Descriptors This 4-bit read-only field specifies the number of region descriptors implemented in the MPU. The defined encodings include: 0000 8 region descriptors 0010 16 region descriptors This field reads as 0b0010 on the SPC564A70
31 VLD	Valid This bit provides a global enable/disable for the MPU. 0 The MPU is disabled 1 The MPU is enabled While the MPU is disabled, all accesses from all bus masters are allowed.

1. See *Table 80* in *Section 13.2, MPU-to-XBAR slave port mapping*, for MPU slave port details.

MPU Error Address Register, Slave Port 0 to 1 (MPU_EAR*n*)

When the MPU detects an access error on slave port *n*(e), the 32-bit reference address is captured in this read-only register and the corresponding bit in the MPU_CESR[SPERR] field is set. Additional information about the faulting access is captured in the corresponding MPU_EDR*n* register at the same time.

Figure 78. MPU Error Address Register, Slave Port *n* (MPU_EAR*n*)

Address: MPU_BASE (0xFFF1_0000) + 0x0010 (MPU_EAR0)
MPU_BASE (0xFFF1_0000) + 0x0018 (MPU_EAR1)
Access: User read only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	EADDR[31:16]															
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	EADDR[16:0]															
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

▨ = not implemented

---

e. See *Table 80* in *Section 13.2, MPU-to-XBAR slave port mapping*, for MPU slave port details.

Table 83. MPU_EAR*n* field descriptions

Field	Description
0–31 EADDR	Error Address This read-only field is the reference address from slave port *n* that generated the access error.

MPU Error Detail Register, Slave Port 0 to 1 (MPU_EDR*n*)

When the MPU detects an access error on slave port *n*[f], 32 bits of error detail are captured in this read-only register and the corresponding bit in the MPU_CESR[SPERR] field is set. Information on the faulting address is captured in the corresponding MPU_EAR*n* register at the same time. A read of the MPU_EDR*n* register clears the corresponding bit in the MPU_CESR[SPERR] field.

Figure 79. MPU Error Detail Register, Slave Port *n* (MPU_EDR*n*)

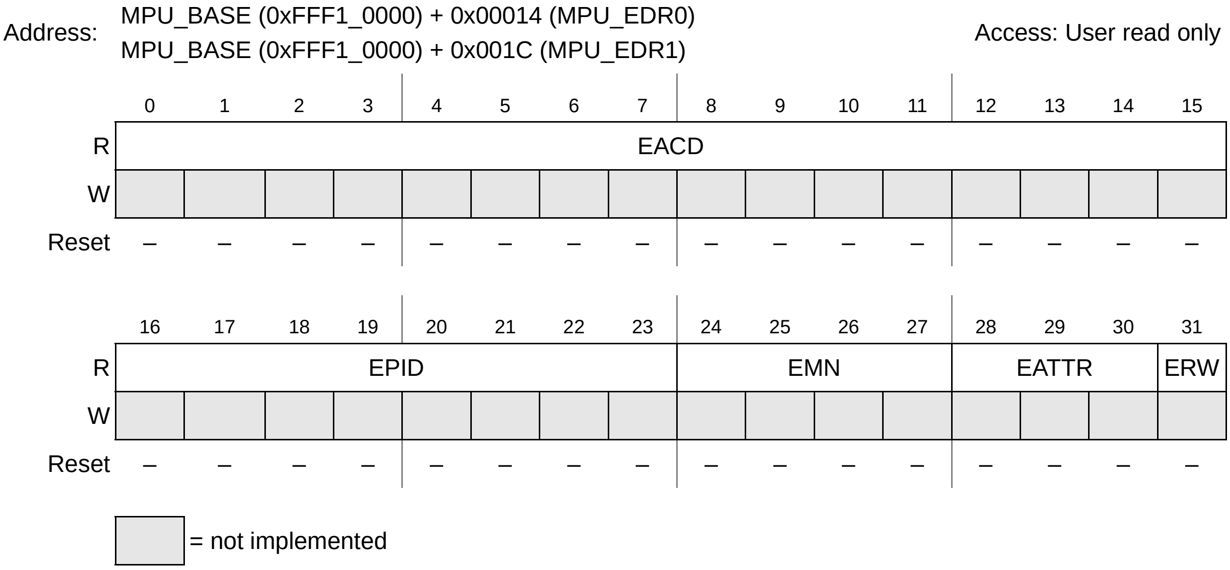


Table 84. MPU_EDR*n* field descriptions

Field	Description
0–15 EACD	Error Access Control Detail This 16-bit read-only field implements one bit per region descriptor and is an indication of the region descriptor hit logically-ANDed with the access error indication. The MPU performs a reference-by-reference evaluation to determine the presence/absence of an access error. When an error is detected, the hit-qualified access control vector is captured in this field. If the MPU_EDR*n* register contains a captured error and the EACD field is all zeroes, this signals an access that did not hit in any region descriptor. All non-zero EACD values signal references that hit in a region descriptor(s), but failed due to a protection error as defined by the specific set bits.

---

[f] See *Table 80* in *Section 13.2, MPU-to-XBAR slave port mapping*, for MPU slave port details.

Table 84. MPU_EDR*n* field descriptions (continued)

Field	Description
16–23 EPID	Error Process Identification This 8-bit read-only field records the process identifier of the faulting reference. The process identifier is typically driven by processor cores only; for other bus masters, this field is cleared.
24–27 EMN	Error Master Number This 4-bit read-only field records the logical master number of the faulting reference. This field is used to determine the bus master that generated the access error.
28–30 EATTR	Error Attributes This 3-bit read-only field records attribute information about the faulting reference. The supported encodings are defined as: 000 User mode, instruction access 001 User mode, data access 010 Supervisor mode, instruction access 011 Supervisor mode, data access All other encodings are reserved. For non-core bus masters, the access attribute information is typically wired to supervisor, data (0b011).
31 ERW	Error Read/Write This 1-bit read-only field signals the access type (read, write) of the faulting reference. 0 Read 1 Write

MPU Region Descriptor *n* (MPU_RGD*n*)

Each 128-bit (16-byte) region descriptor specifies a given memory space and the access attributes associated with that space. The descriptor definition is fundamental to the operation of the MPU.

The region descriptors are organized sequentially in the MPU’s programming model and each of the four 32-bit words are detailed in the subsequent sections.

MPU Region Descriptor *n*, Word 0 (MPU_RGD*n*.Word0)

The first word of the MPU region descriptor defines the 0-modulo-32 byte start address of the memory region. Writes to this word clear the region descriptor’s valid bit.

Figure 80. MPU Region Descriptor *n*, Word 0 Register (MPU_RGD*n*.Word0)

Address: MPU_BASE (0xFFF1_0000) + (16**n*) + 0x0 (MPU_RGD*n*.Word0) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SRTADDR[26:11]															
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SRTADDR[10:0]															
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	–	–	–	–	–	–	–	–	–	–	–	0	0	0	0	0

▨ = not implemented

Table 85. MPU_RGD*n* Word 0 field description

Field	Description
0–26 SRTADDR	Start Address This field defines the most significant bits of the 0-modulo-32 byte start address of the memory region.

MPU Region Descriptor *n*, Word 1 (MPU_RGD*n*.Word1)

The second word of the MPU region descriptor defines the 31-modulo-32 byte end address of the memory region. Writes to this word clear the region descriptor's valid bit.

Figure 81. MPU Region Descriptor *n*, Word 1 Register (MPU_RGD*n*.Word1)

Address: MPU_BASE (0xFFF1_0000) + (16**n*) + 0x4 (MPU_RGD*n*.Word1) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	ENDADDR[26:11]															
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	ENDADDR[10:0]															
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	–	–	–	–	–	–	–	–	–	–	–	1	1	1	1	1

▨ = not implemented

Table 86. MPU_RGD*n* Word 1 field description

Field	Description
0–26 ENDADDR	End Address This field defines the most significant bits of the 31-modulo-32 byte end address of the memory region. There are no hardware checks to verify that ENDADDR ≥ SRTADDR; the software must properly load these region descriptor fields.

MPU Region Descriptor *n*, Word 2 (MPU_RGD*n*.Word2)

The third word of the MPU region descriptor defines the access control rights of the memory region. The access control privileges are dependent on two broad classifications of bus masters. Bus masters 0–3 are typically reserved for processor cores. The corresponding access control is a 6-bit field defining separate privilege rights for user and supervisor mode accesses as well as the optional inclusion of a process identification field within the definition. Bus masters 4–7 are typically reserved for data movement engines and their capabilities are limited to separate read and write permissions. For these fields, the bus master number refers to the physical master ID defined in *Table 49* in *Chapter 9: Multi-Layer AHB Crossbar Switch (XBAR)*.

Note: For the processor privilege rights, there are three flags associated with this function: {read, write, execute}. In this context, these flags follow the traditional definition:

Read (r) permission refers to the ability to access the referenced memory address using an operand (data) fetch.

Write (w) permission refers to the ability to update the referenced memory address using a store (data) instruction.

Execute (x) permission refers to the ability to read the referenced memory address using an instruction fetch.

Writes to this word clear the region descriptor’s valid bit. Because it is also expected that system software may adjust only the access controls within a region descriptor (MPU_RGD*n*.Word2) as different tasks execute, an alternate programming view of this 32-bit entity is provided. If only the access controls are being updated, this operation should be performed by writing to MPU_RGDAAC*n* (alternate access control *n*) as stores to these locations do not affect the descriptor’s valid bit.

The MPU operates on the following masters:

- M0: e200z4 core
- M4: eDMA
- M6: FlexRay

Figure 82. MPU Region Descriptor *n*, Word 2 Register (MPU_RGD*n*.Word2)

Address: MPU_BASE (0xFFF1_0000) + 0x400 + (16**n*) + 0x8 (MPU_RGD*n*.Word2) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	M7RE	M7WE	M6RE	M6WE			M4RE	M4WE								
W					▨	▨			▨	▨	▨	▨	▨	▨	▨	▨
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R											M0PE	M0SM		M0UM		
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨				r	w	x
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

▨ = not implemented

Refer to *Table 49*, in the XBAR chapter, to see the Master ID assignments.

Table 87. MPU_RGD*n* Word 2 field description

Field	Description
6 M7RE	Bus Master ID 7 (EBI) Read Enable If set, this flag allows bus master ID 7 to perform read operations. If cleared, any attempted read by bus master ID 4 terminates with an access error and the read is not performed. Bus Master 7 (EBI) is available for Factory Test only.
7 M7WE	Bus Master ID 7 (EBI) Write Enable If set, this flag allows bus master ID 7 to perform write operations. If cleared, any attempted write by bus master ID 7 terminates with an access error and the write is not performed. Bus Master 7 (EBI) is available for Factory Test only.
6 M6RE	Bus Master ID 6 (FlexRay) Read Enable If set, this flag allows bus master ID 6 to perform read operations. If cleared, any attempted read by bus master ID 6 terminates with an access error and the read is not performed.
7 M6WE	Bus Master ID 6 (FlexRay) Write Enable If set, this flag allows bus master ID 6 to perform write operations. If cleared, any attempted write by bus master ID 6 terminates with an access error and the write is not performed.
6 M4RE	Bus Master ID 4 (eDMA) Read Enable If set, this flag allows bus master ID 4 to perform read operations. If cleared, any attempted read by bus master ID 4 terminates with an access error and the read is not performed.
7 M4WE	Bus Master ID 4 (eDMA) Write Enable If set, this flag allows bus master ID 4 to perform write operations. If cleared, any attempted write by bus master ID 4 terminates with an access error and the write is not performed.
bits 8–25	Reserved

Table 87. MPU_RGD*n* Word 2 field description (continued)

Field	Description
26 M0PE	Bus Master ID 0 (Core) Process Identifier Enable. If set, this flag specifies that the process identifier and mask defined in MPU_RGD*n*.Word3 are to be included in the region hit evaluation. If cleared, the region hit evaluation does not include the process identifier.
27–28 M0SM	Bus Master ID 0 (Core) Supervisor Mode Access Control This 2-bit field defines the access controls for bus master ID 0 when operating in supervisor mode. The M0SM field is defined as: 00 r, w, x = read, write and execute allowed 01 r, –, x = read and execute allowed, but no write 10 r, w, – = read and write allowed, but no execute 11 Same access controls as that defined by M0UM for user mode
29–31 M0UM	Bus Master ID 0 (Core) User Mode Access Control This 3-bit field defines the access controls for bus master ID 0 when operating in user mode. The M0UM field consists of three independent bits, enabling read, write, and execute permissions: {r, w, x}. If set, the bit allows the given access type to occur; if cleared, an attempted access of that mode may be terminated with an access error (if not allowed by any other descriptor) and the access not performed.

MPU Region Descriptor *n*, Word 3 (MPU_RGD*n*.Word3)

The fourth word of the MPU region descriptor contains the optional process identifier and mask, plus the region descriptor's valid bit.

Because the region descriptor is a 128-bit entity, there are potential coherency issues as this structure is being updated because multiple writes are required to update the entire descriptor. Accordingly, the MPU hardware assists in the operation of the descriptor valid bit to prevent incoherent region descriptors from generating spurious access errors. In particular, it is expected that a complete update of a region descriptor is typically done with sequential writes to MPU_RGD*n*.Word0, then MPU_RGD*n*.Word1, ... and MPU_RGD*n*.Word3. The MPU hardware automatically clears the valid bit on any writes to words {0,1,2} of the descriptor. Writes to this word set/clear the valid bit in a normal manner.

Because it is also expected that system software may adjust the access controls within a region descriptor (MPU_RGDn.Word2) only as different tasks execute, an alternate programming view of this 32-bit entity is provided. If only the access controls are being updated, this operation must be performed by writing to MPU_RGDAAC*n* (alternate access control *n*) as stores to these locations do not affect the descriptor's valid bit.

Figure 83. MPU Region Descriptor *n*, Word 3 Register (MPU_RGD*n*.Word3)

Address: MPU_BASE (0xFFF1_0000) + 0x400 + (16**n*) + 0xc (MPU_RGD*n*.Word3) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	PID								PIDMASK							
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	VLD
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = not implemented

Table 88. MPU_RGD*n* Word 3 field description

Field	Description
0–7 PID	Process Identifier This 8-bit field specifies that the optional process identifier is to be included in the determination of whether the current access hits in the region descriptor. This field is combined with the PIDMASK and included in the region hit determination if MPU_RGD*n*.Word2[MxPE] is set.
8–15 PIDMASK	Process Identifier Mask This 8-bit field provides a masking capability so that multiple process identifiers can be included as part of the region hit determination. If a bit in the PIDMASK is set, the corresponding bit of the PID is ignored in the comparison. This field is combined with the PID and included in the region hit determination if MPU_RGD*n*.Word2[MxPE] is set. For more information on the handling of the PID and PIDMASK, see *Section , Access Evaluation—Hit Determination*.
31 VLD	Valid This bit signals the region descriptor is valid. Any write to MPU_RGD*n*.Word{0,1,2} clears this bit, but a write to MPU_RGD*n*.Word3 sets or clears this bit depending on bit 31 of the write operand. 0 Region descriptor is invalid 1 Region descriptor is valid

MPU Region Descriptor Alternate Access Control *n* (MPU_RGDAAC*n*)

As noted in *Section , MPU Region Descriptor n, Word 2 (MPU_RGDn.Word2)*, it is expected that because system software may adjust the access controls within a region descriptor (MPU_RGD*n*.Word2) only as different tasks execute, an alternate programming view of this 32-bit entity is desired. If only the access controls are being updated, this operation should be performed by writing to MPU_RGDAAC*n* (alternate access control *n*) as stores to these locations do not affect the descriptor's valid bit.

The memory address therefore provides an alternate location for updating MPU_RGD*n*.Word2.

Figure 84. MPU RGD Alternate Access Control *n* (MPU_RGDAAC*n*)

Address: MPU_BASE (0xFFF1_0000) + 0x800 + (4**n*) (MPU_RGDAAC*n*) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	M7RE	M7WE	M6RE	M6WE	0	0	M4RE	M4WE	0	0	0	0	0	0	0	0
W					▨	▨			▨	▨	▨	▨	▨	▨	▨	▨
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	M0PE	M0SM		M0UM		
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨				r	w	x
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

▨ = not implemented

Because the MPU_RGDAAC*n* register is another memory mapping for MPU_RGD*n*.Word2, the field definitions shown in *Table 89* are identical to those presented in *Table 87*.

Table 89. MPU_RGDAAC*n* field descriptions

Field	Description
6 M7RE	Bus Master ID 7 (EBI) Read Enable If set, this flag allows bus master ID 7 to perform read operations. If cleared, any attempted read by bus master ID 4 terminates with an access error and the read is not performed. Bus Master 7 (EBI) is available for Factory Test only.
7 M7WE	Bus Master ID 7 (EBI) Write Enable If set, this flag allows bus master ID 7 to perform write operations. If cleared, any attempted write by bus master ID 7 terminates with an access error and the write is not performed. Bus Master 7 (EBI) is available for Factory Test only.
6 M6RE	Bus Master ID 6 (FlexRay) Read Enable If set, this flag allows bus master ID 6 to perform read operations. If cleared, any attempted read by bus master ID 6 terminates with an access error and the read is not performed.
7 M6WE	Bus Master ID 6 (FlexRay) Write Enable If set, this flag allows bus master ID 6 to perform write operations. If cleared, any attempted write by bus master ID 6 terminates with an access error and the write is not performed.
bits 4–5	Reserved These bits must never be set.
6 M4RE	Bus Master ID 4 Read Enable If set, this flag allows bus master ID 4 to perform read operations. If cleared, any attempted read by bus master ID 4 terminates with an access error and the read is not performed.

Table 89. MPU_RGDAAC*n* field descriptions (continued)

Field	Description
7 M4WE	Bus Master 4 Write Enable If set, this flag allows bus master 4 to perform write operations. If cleared, any attempted write by bus master 4 terminates with an access error and the write is not performed.
bits 8–25	Reserved
26 M0PE	Bus Master 0 Process Identifier Enable If set, this flag specifies that the process identifier and mask (defined in MPU_RGDn.Word3) are to be included in the region hit evaluation. If cleared, then the region hit evaluation does not include the process identifier.
27–28 M0SM	Bus Master 0 Supervisor Mode Access Control This 2-bit field defines the access controls for bus master 0 when operating in supervisor mode. The M0SM field is defined as: 00 r, w, x = read, write and execute allowed 01 r, –, x = read and execute allowed, but no write 10 r, w, – = read and write allowed, but no execute 11 Same access controls as that defined by M0UM for user mode
29–31 M0UM	Bus Master 0 User Mode Access Control This 3-bit field defines the access controls for bus master 0 when operating in user mode. The M0UM field consists of three independent bits, enabling read, write, and execute permissions: {r, w, x}. If set, the bit allows the given access type to occur; if cleared, an attempted access of that mode may be terminated with an access error (if not allowed by any other descriptor) and the access not performed.

13.5 Functional Description

In this section, the functional operation of the MPU is detailed. In particular, subsequent sections discuss the operation of the access evaluation macro as well as the handling of error-terminated XBAR bus cycles.

13.5.1 Access Evaluation

As discussed, the basic operation of the MPU is performed in the access evaluation macro, a hardware structure replicated in the two-dimensional connection matrix. The access evaluation macro inputs the XBAR system bus address and the contents of a region descriptor (RGD*n*) and performs two major functions: region hit determination and detection of an access protection violation.

Access Evaluation—Hit Determination

To determine if the current XBAR reference hits in the given region, two magnitude comparators are used with the region’s start and end addresses. There are no hardware checks to verify that the region end address is greater than or equal to the region start address. The software must properly load appropriate values into these fields of the region descriptor.

In addition to the comparison of the XBAR reference address versus the region descriptor’s start and end addresses, the optional process identifier is examined against the region descriptor’s PID and PIDMASK fields. For XBAR bus masters that do not output a process identifier, the MPU forces the PID term to be asserted.

Access Evaluation—Privilege Violation Determination

While the access evaluation macro is making the region hit determination, the logic is also evaluating if the current access is allowed by the permissions defined in the region descriptor. Using the XBAR supervisor/user mode signals, a set of permissions is generated from the appropriate fields in the region descriptor. The protection violation logic evaluates the access against the effective permissions.

The access evaluation macro then uses the hit and permission signals to determine if the current access is allowed and the MPU_EDR*n* (error detail register) is updated in the event of an error.

13.5.2 XBAR Error Terminations

For each XBAR slave port being monitored, the MPU tests any access for permission violations as above. If a violation occurs, the MPU terminates the bus cycle and reports a protection error for three conditions:

1. If the access does not hit in any region descriptor, a protection error is reported.
2. If the access hits in a single region descriptor and that region signals a protection violation, a protection error is reported.
3. If the access hits in multiple (overlapping) regions and all regions signal protection violations, then a protection error is reported.

The third condition reflects that priority is given to permission granting over access denying for overlapping regions as this approach provides more flexibility to system software in region descriptor assignments. For an example of the use of overlapping region descriptors, see *Section 13.7, Application Information*.

When the MPU causes a termination error to occur, the effect on the system depends on the bus master requesting the access. If the error was caused by a core access, a machine check is taken. If the error was caused by an eDMA access, an eDMA source or destination error occurs in the eDMA controller, which can be enabled to provide an interrupt request through the INTC. If the error was caused by a FlexRay access, a controller host interface (CHI) illegal system memory access error occurs in the FlexRay controller, which can be enabled to provide an interrupt request to the INTC.

13.6 Initialization Information

The reset state of MPU_CESR[VLD] disables the entire module. While the MPU is disabled, all accesses from all bus masters are allowed. This state also minimizes the power dissipation of the MPU. The power dissipation of each access evaluation macro is minimized when the associated region descriptor is marked as invalid or when MPU_CESR[VLD] = 0.

Typically the appropriate number of region descriptors (MPU_RGD*n*) are loaded at system startup, including the setting of the MPU_RGD*n*.Word3[VLD] bits, before MPU_CESR[VLD] is set, enabling the module. This approach allows all the loaded region descriptors to be enabled simultaneously. Once the MPU is enabled, if a memory reference does not hit in any region descriptor, the attempted access is terminated with an error.

13.7 Application Information

In an application's system, interfacing with the MPU can generally be classified into the following activities:

1. Creation of a new memory region requires loading the appropriate region descriptor into an available register location. When a new descriptor is loaded into a RGD*n*, it would typically be performed using four 32-bit word writes. As discussed in *Section , MPU Region Descriptor n, Word 3 (MPU_RGDn.Word3)*, the hardware assists in the maintenance of the valid bit, so if this approach is followed, there are no coherency issues associated with the multi-cycle descriptor writes. Deletion/removal of an existing memory region is performed by clearing MPU_RGD*n*.Word3[VLD].
2. If only the access rights for an existing region descriptor need to change, a 32-bit write to the alternate version of the access control word (MPU_RGDAAC*n*) would typically be performed. Writes to the region descriptor using this alternate access control location do not affect the valid bit, so there are, by definition, no coherency issues involved with the update. The access rights associated with the memory region switch instantaneously to the new value as the IPS write completes.
3. If the region's start and end addresses are to be changed, this would typically be performed by writing a minimum of three words of the region descriptor: MPU_RGD*n*.Word{0,1,3}, where the writes to Word0 and Word1 redefine the start and end addresses respectively and the write to Word3 re-enables the region descriptor valid bit. In many situations, all four words of the region descriptor would be rewritten.
4. Typically, references to the MPU's programming model would be restricted to supervisor mode accesses from a specific processor(s), so a region descriptor would be specifically allocated for this purpose with attempted accesses from other masters or while in user mode terminated with an error.
5. When the MPU detects an access error, the current XBAR bus cycle is terminated with an error response and information on the faulting reference captured in the MPU_EAR*n* and MPU_EDR*n* registers. The error-terminated XBAR bus cycle typically initiates some type of error response in the originating bus master. For example, a processor core may respond with a bus error exception, while a data movement bus master may respond with an error interrupt. In any event, the processor can retrieve the captured error address and detail information simply be reading the MPU_E{A,D}R*n* registers. Information on which error registers contain captured fault data is signaled by MPU_CESR[SPERR].
6. Finally, consider the use of overlapping region descriptors. Application of overlapping regions can reduce the number of descriptors required for a given set of access controls. In the overlapping memory space, the protection rights of the corresponding region descriptors are logically summed together (the boolean OR operator). In the following example of a dual-core system, there are four bus masters: the two processors (CP0, CP1) and two DMA engines (eDMA, a traditional data movement engine transferring data between RAM and peripherals, and FlexRay, a second engine transferring data to/from the RAM only). Consider the region descriptor assignments shown in *Table 90*:

Table 90. Overlapping region descriptor example

Region description	RGD*n*	CP0	CP1	eDMA	FlexRay	Memory map space
CP0 Code	0	r w x	r – –	– –	– –	Flash
CP1 Code	1	r – –	r w x	– –	– –	
CP0 Data & Stack	2	r w –	– – –	– –	– –	RAM
CP0 –> CP1 Shared Data	3	r – –	r – –	– –	– –	
CP1 –> CP0 Shared Data						
CP0 Data & Stack	4	– – –	r w –	– –	—	
Shared DMA Data	5	r w –	r w –	r w	r w	
MPU	6	r w –	r w –	– –	– –	IPS
Peripherals	7	r w –	r w –	r w	– –	

In this example, there are eight descriptors used to span nine regions in the three main spaces of the system memory map (flash, RAM, and IPS peripheral space). Each region indicates the specific permissions for each of the four bus masters and this definition provides an appropriate set of shared, private and executable memory spaces.

Of particular interest are the two overlapping spaces: region descriptors 2 and 3, and 3 and 4.

The space defined by RGD2 with no overlap is a private data and stack area that provides read/write access to CP0 only. The overlapping space between RGD2 and RGD3 defines a shared data space for passing data from CP0 to CP1 and the access controls are defined by the logical OR of the two region descriptors. Thus, CP0 has (r w – | r – –) = (r w –) permissions, while CP1 has (– – – | r – –) = (r – –) permission in this space. Both DMA engines are excluded from this shared processor data region. The overlapping spaces between RGD3 and RGD4 defines another shared data space, this one for passing data from CP1 to CP0. For this overlapping space, CP0 has (r – – | – – –) = (r – –) permission, while CP1 has (r w – | r – –) = (r w –) permission. The non-overlapped space of RGD4 defines a private data and stack area for CP1 only.

The space defined by RGD5 is a shared data region, accessible by all four bus masters. Finally, the slave peripheral space mapped onto the peripheral bus is partitioned into two regions: one (RGD6) containing the MPU's programming model accessible only to the two processor cores, and the remaining peripheral region (RGD7) accessible to both processors and the traditional eDMA master.

This example is intended to show one possible application of the capabilities of the memory protection unit in a typical system.

Warning: Program code occupies the end of the MPU region (#0) in which core instruction accesses are allowed. The address region immediately afterwards is protected by the MPU (region #1) from instruction fetches by the core (or any PID=1 access).

If the last instruction in the MPU region #0 space is a branch which the core takes while the core attempts to fetch instructions via instruction cache line fill from the MPU region #1 the MPU asserts a bus error (a PID=1 executable access into a region which only allows read/write accesses from PID=2). The core immediately takes the exception as a 'machine check'.

In this case, modify the 'machine check' exception handler to expect this behavior.

14 Interrupt Controller (INTC)

14.1 Information specific to this device

This section presents device-specific parameterization and customization information not specifically referenced in the remainder of this chapter.

14.1.1 Device-specific features

- 279 peripheral interrupts
- 197 reserved interrupts
- 8 software interrupts

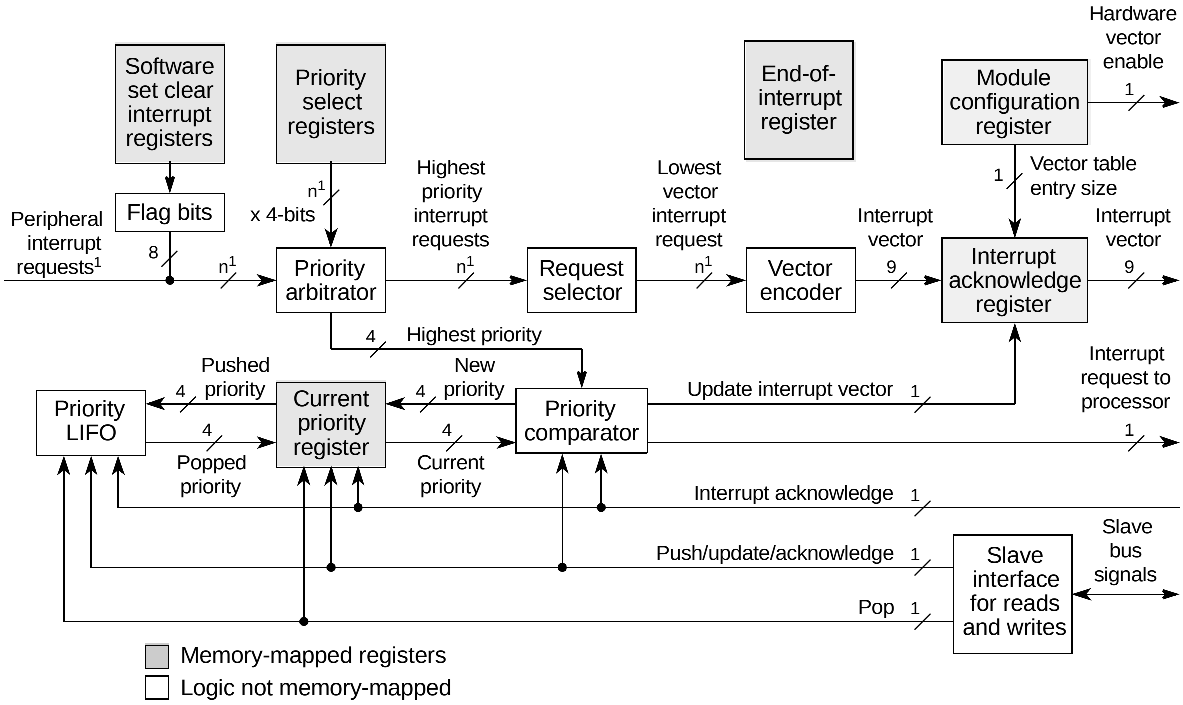
14.2 Introduction

This chapter describes the interrupt controller (INTC), which schedules interrupt requests (IRQs) from software and internal peripherals to the e200z4 core. The INTC provides interrupt prioritization and preemption, interrupt masking, interrupt priority elevation, and protocol support.

Interrupts implemented by the MCU are defined in the *e200z4 Power Architecture® Core Reference Manual*

14.2.1 Block diagram

Figure 85 shows the details of the interrupt controller.



[1] Although N (largest addressable IRQ vector number) = 485, this does not indicate the total number of interrupts available on this device. The total number of available interrupts on this device is 486: 279 peripheral IRQs, 8 software-configurable IRQs, and 199 reserved.

Figure 85. INTC Block Diagram

14.2.2 Overview

Interrupt functionality for the device is handled between the e200z4 core and the interrupt controller. The CPU core has 19 exception sources, each of which can interrupt the core. One exception source is from the interrupt controller (INTC). The INTC provides priority-based scheduling of interrupt requests and supports programmable preemption. This scheduling scheme is suitable for statically scheduled hard real-time systems. The INTC is optimized for a large number of interrupt requests.

Table 91 displays the interrupt sources and the number of interrupts available for each module; *Figure 86* shows a general diagram of INTC software vector mode.

Table 91. Interrupt sources available

Interrupt Source (IRQs)	Number of Interrupts Available
Software	8
Watchdog	0
SRAM error correction	1
Flash error correction	1

Table 91. Interrupt sources available (continued)

Interrupt Source (IRQs)	Number of Interrupts Available
eDMA	66
FMPLL	2
External IRQ input pins (SIU)	6
eMIOS	24
eTPU engine A	33
eQADC	31
DSPI	15
eSCI	3
FlexCAN	63
FlexRay	8
STM	5
Decimation Filter	3
System (PIT, RTI, PMC, etc)	6

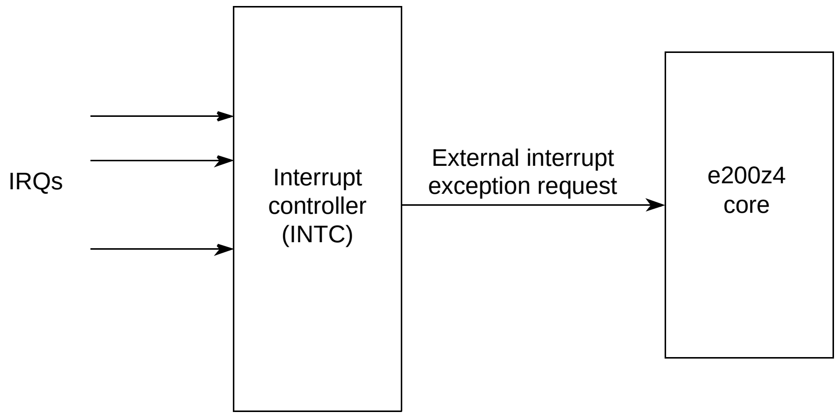


Figure 86. INTC software vector mode

Two modes are available to determine the vector for the interrupt request source:

- Software vector mode
- Hardware vector mode

In software vector mode, as shown in *Figure 86*, the e200z4 branches to a common interrupt exception handler whose location is determined by an address derived from special purpose registers IVPR and IVOR4. The interrupt exception handler reads the INTC_IACKR to determine the vector of the interrupt request source. Typical program flow for software vector mode is shown in *Figure 87*.

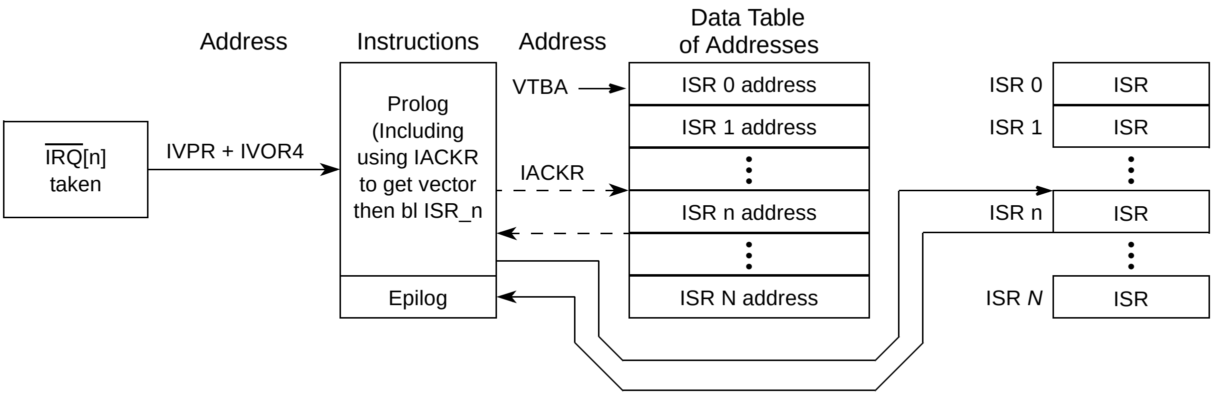


Figure 87. Program Flow–Software Vector Mode

N is the largest vector number (485) with the greatest hexadecimal address (IVPR + 0x1D90) that is available in the interrupt memory map for this device. As memory blocks throughout the total memory map are used for other purposes, the maximum vector number does not indicate the total number of available interrupt sources for this device.

The total number of entries in the interrupt memory map on this device is 486: 279 peripheral IRQs, 8 software configurable IRQs, and 199 reserved.

In hardware vector mode, the core branches to an interrupt exception handler unique for each interrupt request source. Typical program flow for hardware vector mode is shown in *Figure 88*.

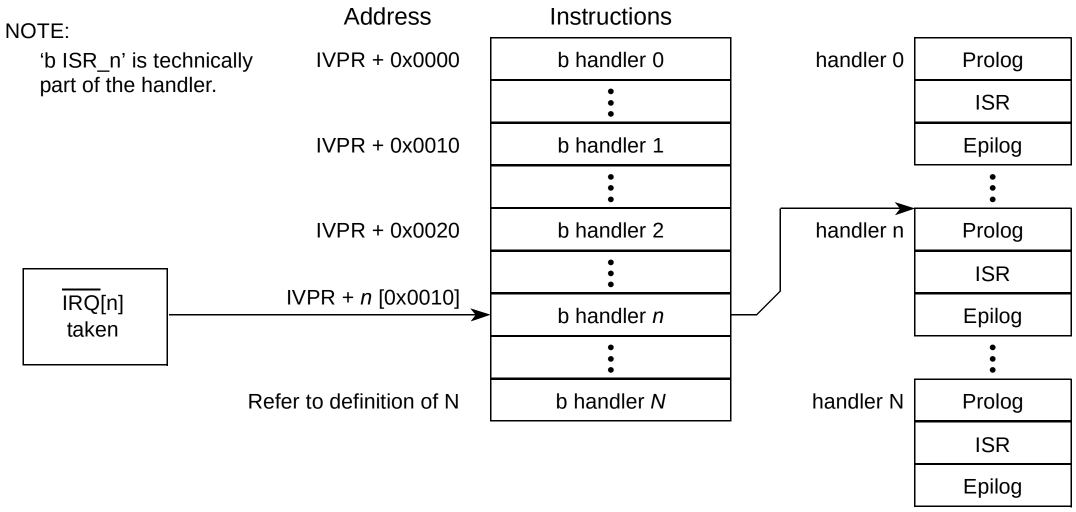


Figure 88. Program Flow–Hardware Vector Mode

The INTC supports a hardware vector mode that reduces the time between assertion of an interrupt and execution of the service routine. It also provides 16 priorities so that lower priority ISRs do not delay the execution of higher priority ISRs. The priority assigned to each interrupt source is programmable in the range 0 to 15, with 0 being the lowest and 15 being the highest priority.

When multiple tasks share a resource, coherent accesses to that resource need to be supported. The INTC supports the Priority Ceiling Protocol (PCP) for coherent accesses. By providing a modifiable priority mask, the priority level can be raised temporarily so that no task can preempt another task that shares the same resource.

Multiple processors can assert interrupt requests to each other through software configurable interrupt requests, i.e., by using application software to assert an interrupt request. These same software configurable interrupt requests also can be used to break the work involved in servicing an interrupt request into a high priority portion and a low priority portion. The high priority portion is initiated by a peripheral interrupt request, but the ISR can assert a software configurable interrupt request to finish the servicing in a low priority ISR.

14.2.3 Features

Features include the following:

- Total number of interrupt vectors is 486, of which:
  - 279 are peripheral interrupt vectors
  - 8 are software configurable sources
  - 199 are reserved sources
- 9-bit unique vector for each interrupt request source in hardware vector mode.
- Each interrupt source can be programmed to one of 16 priorities.
- Preemption.
  - Preemptive prioritized interrupt requests to processor.
  - ISR at a higher priority preempts ISRs or tasks at lower priorities.
  - Automatic pushing or popping of preempted priority to or from a LIFO.
  - Ability to modify the ISR or task priority. Modifying the priority can be used to implement the PCP for accessing shared resources.
- Low latency–three clocks from receipt of interrupt request from peripheral to interrupt request to processor.

14.2.4 Modes of operation

The interrupt controller has two handshaking modes with the processor: software vector mode and hardware vector mode. The state of the hardware vector enable bit, INTC_MCR[HVEN], determines which mode is used.

In debug mode, the interrupt controller operation is identical to its normal operation of software vector mode or hardware vector mode.

Software vector mode

In the software vector mode, there is a common interrupt exception handler address that is calculated by hardware as shown in *Figure 89*. The upper half of the interrupt vector prefix register (IVPR) is added to the offset contained in the external input interrupt vector offset register (IVOR4).

Note: *Since bits IVOR4[28:31] are not part of the offset value, the vector offset must be located on a quad-word (16-byte) aligned location in memory.*

In the software vector mode, the interrupt exception handler software must read the INTC interrupt acknowledge register (INTC_IACKR) to obtain the vector number and base address of the handler associated with the corresponding peripheral or software interrupt request. The INTC_IACKR register contains a 21-bit or 20-bit address for a vector table base address (VTBA). The address is then used to branch to the corresponding routine for that peripheral or software interrupt source.

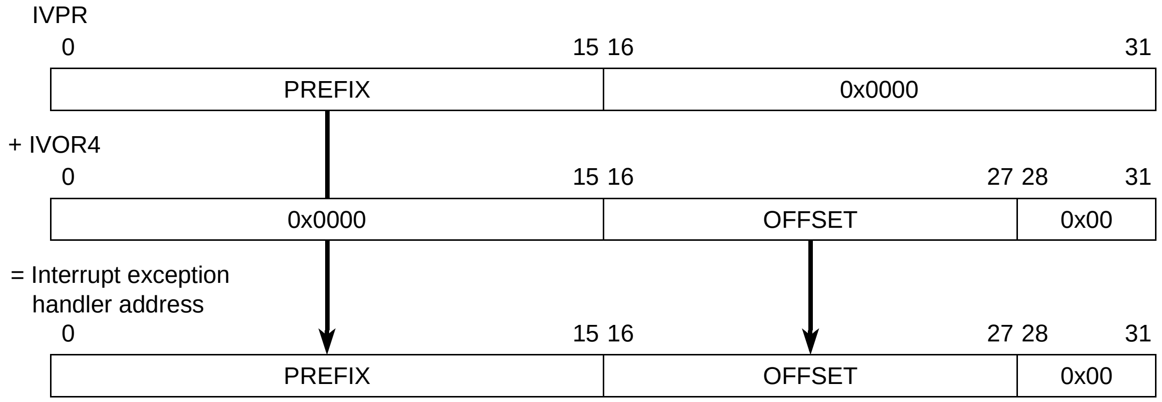


Figure 89. Software Vector Mode: Interrupt Exception Handler Address Calculation

Reading the INTC_IACKR acknowledges the INTC’s interrupt request and negates the interrupt request to the processor. The interrupt request to the processor does not clear if a higher priority interrupt request arrives. Even in this case, INTVEC does not update to the higher priority request until the lower priority interrupt request is acknowledged by reading the INTC_IACKR. The reading also pushes the PRI value in the INTC current priority register (INTC_CPR) to the LIFO and updates PRI in the INTC_CPR with the priority of the interrupt request. The INTC_CPR masks any peripheral or software configurable interrupt request at the same or lower priority of the current value of the PRI field in INTC_CPR from generating an interrupt request to the processor.

The interrupt exception handler must write to the end-of-interrupt register (INTC_EOIR) to complete the operation (assuming the source of the interrupt has been cleared). Writing to the INTC_EOIR ends the servicing of the interrupt request. The INTC’s LIFO is popped into the INTC_CPR's PRI field by writing to the INTC_EOIR, and the size of a write does not affect the operation of the write. Those values and sizes written to this register neither update the INTC_EOIR contents nor affect whether the LIFO pops. For possible future compatibility, write four bytes of all 0s to the INTC_EOIR. The timing relationship between popping the LIFO and disabling recognition of external input has no restriction. The writes can happen in either order.

However, disabling recognition of the external input before popping the LIFO eases the calculation of the maximum stack depth at the cost of postponing the servicing of the next interrupt request.

Hardware vector mode

In hardware vector mode, the interrupt exception handler address is specific to the peripheral or software configurable interrupt source rather than being common to all of them. No IVOR is used. The interrupt exception handler address is calculated by hardware as shown in *Figure 90*. The upper half of the interrupt vector prefix register (IVPR) is added to an offset, which corresponds to the peripheral or software interrupt source that caused the interrupt request. The offset matches the value in the Interrupt Vector field, INTC_IACKR[INTVEC]. Each interrupt exception handler address is aligned on a four-word

(16-byte) boundary. IVOR4 is not used in this mode, and software does not need to read INTC_IACKR to get the interrupt vector number.

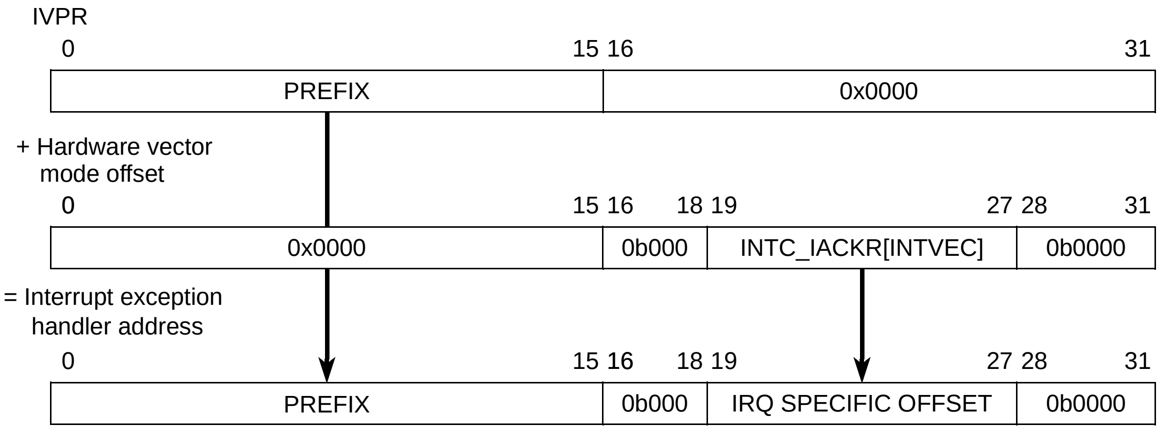


Figure 90. Hardware Vector Mode: Interrupt Exception Handler Address Calculation

The processor negates INTC's interrupt request when automatically acknowledging the interrupt request. However, the interrupt request to the processor do not negate if a higher priority interrupt request arrives. Even in this case, the interrupt vector number does not update to the higher priority request until the lower priority request is acknowledged by the processor.

The assertion of the interrupt acknowledge signal pushes the PRI value in the INTC_CPR onto the LIFO and updates PRI in the INTC_CPR with the new priority.

14.3 External signal description

The INTC does not have any direct external MCU signals. However, there are 15 external pins that can be configured in the SIU as external interrupt request input pins. When configured for an external interrupt request function, an interrupt on that pin sets an external interrupt flag. These flags cause one of five peripheral interrupt requests to the interrupt controller.

For more information on external interrupts, the pins used, and how to configure them:

- Refer to the Signals chapter for a list and number of the external interrupt pins.
- Refer to the SIU chapter for more information on how to configure these pins.

14.4 Memory map and register definition

Table 92 is the INTC memory map.

Table 92. INTC Memory Map

Address	Register Name	Register Description	Bits
Base (0xFFF4_8000)	INTC_MCR	INTC module configuration register	32
Base + 0x0004	—	Reserved	—

Table 92. INTC Memory Map (continued)

Address	Register Name	Register Description	Bits
Base + 0x0008	INTC_CPR	INTC current priority register	32
Base + 0x000C	—	Reserved	—
Base + 0x0010	INTC_IACKR	INTC interrupt acknowledge register [(1)]	32
Base + 0x0014	—	Reserved	—
Base + 0x0018	INTC_EOIR	INTC end-of-interrupt register	32
Base + 0x001C	—	Reserved	—
Base + 0x0020	INTC_SSCIR0	INTC software set/clear interrupt register 0	8
Base + 0x0021	INTC_SSCIR1	INTC software set/clear interrupt register 1	8
Base + 0x0022	INTC_SSCIR2	INTC software set/clear interrupt register 2	8
Base + 0x0023	INTC_SSCIR3	INTC software set/clear interrupt register 3	8
Base + 0x0024	INTC_SSCIR4	INTC software set/clear interrupt register 4	8
Base + 0x0025	INTC_SSCIR5	INTC software set/clear interrupt register 5	8
Base + 0x0026	INTC_SSCIR6	INTC software set/clear interrupt register 6	8
Base + 0x0027	INTC_SSCIR7	INTC software set/clear interrupt register 7	8
Base + (0x0028–0x003F)	—	Reserved	—
Base + (0x0040–0x01A7)	INTC_PSR*n*	INTC priority select registers [(2)] 0–485	8

1. When the HVEN bit in the INTC_MCR is asserted, a read of the INTC_IACKR has no side effects.
2. The PRI fields are “Reserved” for peripheral interrupt requests whose vectors are labeled as Reserved in *Table 97*.

14.4.1 Register descriptions

Except INTC_SSCI*n* and INTC_PSR*n* registers, all registers are 32-bits wide. Any combination of accessing the 4 bytes of a register with a single access is supported, provided that the access does not cross the register boundary. These supported accesses include types and sizes of 8 bits, aligned 16 bits, and aligned 32 bits.

Although INTC_SSCI*n* and INTC_PSR*n* are 8-bits wide, they can be accessed with a single 16-bit or 32-bit access, provided that the access does not cross a 32-bit boundary.

In the software vector mode, the side effects of a read of the INTC interrupt acknowledge register (INTC_IACKR) are the same regardless of the size of the read. In either software or hardware vector mode, the size of a write to the INTC end-of-interrupt register (INTC_EOIR) does not affect the operation of the write.

INTC Module Configuration Register (INTC_MCR)

The INTC_MCR is used to configure options of the INTC.

Address: Base + 0x0000 (INTC_MCR) Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	VTES	0	0	0	0	HVEN
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨		▨	▨	▨	▨	
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 91. INTC Module Configuration Register (INTC_MCR)

Table 93. INTC_MCR Field Descriptions

Field	Description
0–25	Reserved, must be cleared.
26 VTES	Vector table entry size. Controls the number of ‘0’s to the right of INTVEC in *Section , INTC Interrupt Acknowledge Register (INTC_IACKR)*. If the contents of INTC_IACKR are used as an address of an entry in a vector table as in software vector mode, then the number of rightmost ‘0’s determines the size of each vector table entry. VTES impacts software vector mode operation but also affects the INTC_IACKR[INTVEC] position in both hardware vector mode and software vector mode. 0 4 bytes (Normal expected use) 1 8 bytes
27–30	Reserved, must be cleared.
31 HVEN	Hardware vector enable. Controls whether the INTC is in hardware vector mode or software vector mode. Refer to *Section 14.2.4, Modes of operation*, for the details of the handshaking with the processor in each mode. 0 Software vector mode 1 Hardware vector mode

INTC Current Priority Register (INTC_CPR)

The INTC_CPR masks any peripheral or software configurable interrupt request set at the same or lower priority as the current value of the INTC_CPR[PRI] field from generating an interrupt request to the processor. When the INTC interrupt acknowledge register (INTC_IACKR) is read in the software vector mode or the interrupt acknowledge signal from the processor is asserted in the hardware vector mode, the value of PRI is pushed onto the LIFO, and PRI is updated with the priority of the preempting interrupt request. When the INTC end-of-interrupt register (INTC_EOIR) is written, the LIFO is popped into the INTC_CPR’s PRI field.

The masking priority can be raised or lowered by writing to the PRI field, supporting the PCP. Refer to *Section 14.6.5, Priority ceiling protocol*.

Note: On some Power Architecture MCUs, a store to raise the PRI field which closely precedes an access to a shared resource can result in a non-coherent access to that resource unless an ***mbar*** *or* ***msync*** *followed by an* ***isync*** *sequence of instructions is executed between the* *accesses. An **mbar** or **msync** instruction is also necessary after accessing the resource but before lowering the PRI field. Refer to Section , Ensuring coherency.*

Address: Base + 0x0008 (INTC_CPR) Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	PRI			
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨				
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

▨ = Unimplemented or Reserved

Figure 92. INTC Current Priority Register (INTC_CPR)

Table 94. INTC_CPR Field Descriptions

Field	Description
0–27	Reserved, must be cleared.
28–31 PRI	Priority. PRI is the priority of the currently executing ISR according to the field values defined below. 1111 Priority 15 (highest) 1110 Priority 14 ... 0001 Priority 1 0000 Priority 0 (lowest)

INTC Interrupt Acknowledge Register (INTC_IACKR)

The INTC_IACKR provides a value that can be used to load the address of an ISR from a vector table. The vector table can be composed of addresses of the ISRs specific to their respective interrupt vectors.

In software vector mode, reading the INTC_IACKR acknowledges the INTC's interrupt request. Refer to *Section , Software vector mode*, for a detailed description of the effect on the interrupt request to the processor. The reading also pushes the PRI value in the INTC current priority register (INTC_CPR) onto the LIFO and updates PRI in the INTC_CPR with the priority of the interrupt request. The side effect from the reads in software vector mode, that is, the effect on the interrupt request to the processor, the current priority, and the LIFO, are the same regardless of the size of the read

Reading the INTC_IACKR does not have side effects in hardware vector mode.

Note: The INTC_IACKR must not be read speculatively while in software vector mode. Therefore, for future compatibility, the TLB entry covering the INTC_IACKR must be configured to be guarded.

In software vector mode, the INTC_IACKR must be read before setting MSR[EE]. No synchronization instruction is needed after reading the INTC_IACKR and before setting MSR[EE].

However, the time for the processor to recognize the assertion or negation of the external input to it is not defined by the book E architecture and can be greater than 0. Therefore, insert instructions between the reading of the INTC_IACKR and the setting of MSR[EE] that consumes at least two processor clock cycles. This length of time allows the interrupt request negation to propagate through the processor before MSR[EE] is set.

Address: Base + 0x0010 (INTC_IACKR) Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	VTBA (most significant 16 bits)															
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	VTBA (least significant 5 bits)					INTVEC(1)									0	0
W						▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

1. When the VTES bit in the INTC Module Configuration Register (INTC_MCR) is asserted, INTVEC is shifted to the left one bit. Bit 29 is read as a ‘0’. VTBA is narrowed to 20 bits in width.

Figure 93. INTC Interrupt Acknowledge Register (INTC_IACKR)— INTC_MCR[VTES] = 0

Address: Base + 0x0010 (INTC_IACKR) Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	VTBA (most significant 16 bits)															
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	VTBA (least significant 4 bits)				INTVEC(1)									0	0	0
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

1. When the VTES bit in the INTC Module Configuration Register (INTC_MCR) is asserted, INTVEC is shifted to the left one bit. Bit 29 is read as a ‘0’. VTBA1 is narrowed to 20 bits in width.

Figure 94. INTC Interrupt Acknowledge Register (INTC_IACKR)— INTC_MCR[VTES] = 1

Table 95. INTC_IACKR Field Descriptions

Field	Description
0–20 *or* 0–19 VTBA	Vector table base address. Can be the base address of a vector table of addresses of ISRs. The VTBA only uses the left-most 20 bits when the VTES bit in INTC_MCR is asserted.
21–29 *or* 20–28 INTVEC	Interrupt vector. Vector of peripheral or software-configurable interrupt requests that caused the interrupt request to the processor. When the interrupt request to the processor asserts, the INTVEC is updated, whether the INTC is in software or hardware vector mode. If INTC_MCR[VTES] = 1, then the INTVEC field is shifted left one position to bits 20–28. VTBA is then shortened by one bit to bits 0–19.
30–31 or 29–31	Reserved, must be cleared.

INTC End-of-Interrupt Register (INTC_EOIR)

Writing to the INTC_EOIR signals the end of the servicing of the interrupt request. When the INTC_EOIR is written, the priority last pushed on the LIFO is popped into INTC_CPR. The values and size of data written to the INTC_EOIR are ignored. The values and sizes written to this register neither update the INTC_EOIR contents nor affect whether the LIFO pops. For possible future compatibility, write four bytes of all 0’s to the INTC_EOIR.

Reading the INTC_EOIR has no effect on the LIFO.

Address: Base + 0x0018 (INTC_EOIR) Access: W/O

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 95. INTC End-of-Interrupt Register (INTC_EOIR)

INTC Software Set/Clear Interrupt Registers INTC_SSCIR0_3 — INTC_SSCIR4_7)

The INTC_SSCIR*n* supports the setting or clearing of software configurable interrupt requests. These registers contain eight independent sets of bits to set and clear a corresponding flag bit by software. With the exception of being set by software, this flag bit behaves the same as a flag bit set within a peripheral. This flag bit generates an interrupt request within the INTC just like a peripheral interrupt request. Writing a 1 to SET*n* leaves SET*n* unchanged at 0 but sets CLR*n*. Writing a 0 to SET*n* has no effect. CLR*n* is the flag bit. Writing a 1 to CLR*n* clears it. Writing a 0 to CLR*n* has no effect. If a 1 is written to a pair SET*n* and CLR*n* bits at the same time, CLR*n* is asserted, regardless of whether CLR*n* was asserted before the write.

Although INTC_SSCI*n* is 8 bits wide, it can be accessed with a single 16-bit or 32-bit access, provided that the access does not cross a 32-bit boundary.

Address: Base + 0x0020 + *n* (INTC_SSCIR*n*); *n* = 0–7 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	CLR0	0	0	0	0	0	0	0	CLR1
W	▨	▨	▨	▨	▨	▨	SET0		▨	▨	▨	▨	▨	▨	SET1	
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	CLR2	0	0	0	0	0	0	0	CLR3
W	▨	▨	▨	▨	▨	▨	SET2		▨	▨	▨	▨	▨	▨	SET3	
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 96. INTC Software Set/Clear Interrupt Register (INTC_SSCIR*n*)

Table 96. INTC_SSCIR*n* Field Descriptions

Field	Description
0–5	Reserved, must be cleared.
6 SET*n*	Set flag bits. Writing a 1 sets the corresponding CLR*n* bit. Writing a 0 has no effect. Each SET*n* is always read as a 0.
7 CLR*n*	Clear flag bits. CLR*n* is the flag bit. Writing a 1 to CLR*n* clears it provided that a 1 is not written simultaneously to its corresponding SET*n* bit. Writing a 0 to CLR*n* has no effect. 0 Interrupt request not pending within INTC. 1 Interrupt request pending within INTC.

INTC Priority Select Registers (INTC_PSR0–485)

The INTC_PSR*n* allows you to select a priority for each interrupt request source (peripheral IRQs or software configurable IRQs). Each priority select register (INTC_PSR*n)* is assigned to the IRQ source vector with the same number. For example, the software configurable IRQs 0–7 are assigned vectors 0–7, and their priorities are configured in INTC_PSR0–INTC_PSR7, respectively. The peripheral interrupt requests are assigned vectors 8–485 and their priorities are configured in priority select registers INTC_PSR8 through INTC_PSR485, respectively.

Although INTC_PSR*n* is 8-bits wide, you can use a single 16-bit or 32-bit access, provided that it does not cross a 32-bit boundary.

Note: Do not modify the PRIn field in INTC_PSRn when the IRQ is asserted.

Address: Base + 0x0010 (INTC_IACKR) Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PRI0				0	0	0	0	PRI1			
W	▨	▨	▨	▨					▨	▨	▨	▨				
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	PRI2				0	0	0	0	PRI3			
W	▨	▨	▨	▨					▨	▨	▨	▨				
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 97. INTC Priority Select Register 0–3 (INTC_PSR0_3)

Address: Base + 0x0040 + *n* (INTC_PSR*n*); *n* = 0–485 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PRI482				0	0	0	0	PRI483			
W	▨	▨	▨	▨					▨	▨	▨	▨				
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	PRI484				0	0	0	0	PRI485			
W	▨	▨	▨	▨					▨	▨	▨	▨				
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 98. INTC Priority Select Register 482–485 (INTC_PSR482_485)

Table 97. INTC_PSR*n* Field Descriptions

Field	Description
0–3	Reserved, must be cleared.
4–7 PRI*n*	Priority select. Selects the priority for corresponding interrupt request. 1111 Priority 15 (highest) 1110 Priority 14 ... 0001 Priority 1 0000 Priority 0 (lowest)

14.5 Functional description

14.5.1 Interrupt request sources

The INTC has two types of interrupt requests, peripheral and software configurable. The assignments between the interrupt requests from the modules to the vectors for input to the e200z4 are shown in *Table 98*. The Hardware Vector Mode Offset column lists the IRQ-specific offsets when using hardware vector mode. The Source column shows the C language syntax for the register bit label: module_register[bit]. Interrupt requests from the same module location are ORed together. The individual interrupt priorities are selected in INTC_PSR*n*, where the priority select register is assigned according to the vector number.

Table 98. Interrupt Request Sources

Hardware Vector Mode Offset	Vector Number (1)	Source(2)	Description
Software			
0x0000	0	INTC_SSCIR0[CLR0]	INTC software settable Clear flag 0
0x0010	1	INTC_SSCIR1[CLR1]	INTC software settable Clear flag 1
0x0020	2	INTC_SSCIR2[CLR2]	INTC software settable Clear flag 2
0x0030	3	INTC_SSCIR3[CLR3]	INTC software settable Clear flag 3
0x0040	4	INTC_SSCIR4[CLR4]	INTC software settable Clear flag 4
0x0050	5	INTC_SSCIR5[CLR5]	INTC software settable Clear flag 5
0x0060	6	INTC_SSCIR6[CLR6]	INTC software settable Clear flag 6
0x0070	7	INTC_SSCIR7[CLR7]	INTC software settable Clear flag 7
ECC			
0x0080	8	Reserved	—
0x0090	9	ECSM_ESR[RNCE] ECSM_ESR[FNCE]	ECSM combined interrupt requests: Internal SRAM Non-Correctable Error and Flash Non-Correctable Error
eDMAC			
0x00A0	10	EDMA_ERL[ERR31:ERR0]	eDMA channel Error flags 31–0
0x00B0	11	EDMA_IRQRL[INT00]	eDMA channel Interrupt 0
0x00C0	12	EDMA_IRQRL[INT01]	eDMA channel Interrupt 1
0x00D0	13	EDMA_IRQRL[INT02]	eDMA channel Interrupt 2
0x00E0	14	EDMA_IRQRL[INT03]	eDMA channel Interrupt 3
0x00F0	15	EDMA_IRQRL[INT04]	eDMA channel Interrupt 4
0x0100	16	EDMA_IRQRL[INT05]	eDMA channel Interrupt 5
0x0110	17	EDMA_IRQRL[INT06]	eDMA channel Interrupt 6
0x0120	18	EDMA_IRQRL[INT07]	eDMA channel Interrupt 7
0x0130	19	EDMA_IRQRL[INT08]	eDMA channel Interrupt 8
0x0140	20	EDMA_IRQRL[INT09]	eDMA channel Interrupt 9
0x0150	21	EDMA_IRQRL[INT10]	eDMA channel Interrupt 10
0x0160	22	EDMA_IRQRL[INT11]	eDMA channel Interrupt 11
0x0170	23	EDMA_IRQRL[INT12]	eDMA channel Interrupt 12
0x0180	24	EDMA_IRQRL[INT13]	eDMA channel Interrupt 13
0x0190	25	EDMA_IRQRL[INT14]	eDMA channel Interrupt 14

Table 98. Interrupt Request Sources (continued)

Hardware Vector Mode Offset	Vector Number (1)	Source(2)	Description
0x01A0	26	EDMA_IRQRL[INT15]	eDMA channel Interrupt 15
0x01B0	27	EDMA_IRQRL[INT16]	eDMA channel Interrupt 16
0x01C0	28	EDMA_IRQRL[INT17]	eDMA channel Interrupt 17
0x01D0	29	EDMA_IRQRL[INT18]	eDMA channel Interrupt 18
0x01E0	30	EDMA_IRQRL[INT19]	eDMA channel Interrupt 19
0x01F0	31	EDMA_IRQRL[INT20]	eDMA channel Interrupt 20
0x0200	32	EDMA_IRQRL[INT21]	eDMA channel Interrupt 21
0x0210	33	EDMA_IRQRL[INT22]	eDMA channel Interrupt 22
0x0220	34	EDMA_IRQRL[INT23]	eDMA channel Interrupt 23
0x0230	35	EDMA_IRQRL[INT24]	eDMA channel Interrupt 24
0x0240	36	EDMA_IRQRL[INT25]	eDMA channel Interrupt 25
0x0250	37	EDMA_IRQRL[INT26]	eDMA channel Interrupt 26
0x0260	38	EDMA_IRQRL[INT27]	eDMA channel Interrupt 27
0x0270	39	EDMA_IRQRL[INT28]	eDMA channel Interrupt 28
0x0280	40	EDMA_IRQRL[INT29]	eDMA channel Interrupt 29
0x0290	41	EDMA_IRQRL[INT30]	eDMA channel Interrupt 30
0x02A0	42	EDMA_IRQRL[INT31]	eDMA channel Interrupt 31
PLL			
0x02B0	43	FMPLL_SYNSR[LOCF]	FMPLL Loss of Clock Flag
0x02C0	44	FMPLL_SYNSR[LOLF]	FMPLL Loss of Lock Flag
SIU			
0x02D0	45	SIU_OSR[OVF15:OVF0]	SIU combined overrun interrupt requests of the external interrupt Overrun Flags
0x02E0	46	SIU_EIISR[EIF0]	SIU External Interrupt Flag 0
0x02F0	47	SIU_EIISR[EIF1]	SIU External Interrupt Flag 1
0x0300	48	SIU_EIISR[EIF2]	SIU External Interrupt Flag 2
0x0310	49	SIU_EIISR[EIF3]	SIU External Interrupt Flag 3
0x0320	50	SIU_EIISR[EIF15:EIF4]	SIU External Interrupt Flags 15–4
eMIOS			
0x0330	51	EMIOS_GFR[F0]	eMIOS channel 0 Flag
0x0340	52	EMIOS_GFR[F1]	eMIOS channel 1 Flag
0x0350	53	EMIOS_GFR[F2]	eMIOS channel 2 Flag
0x0360	54	EMIOS_GFR[F3]	eMIOS channel 3 Flag

Table 98. Interrupt Request Sources (continued)

Hardware Vector Mode Offset	Vector Number (1)	Source(2)	Description
0x0370	55	EMIOS_GFR[F4]	eMIOS channel 4 Flag
0x0380	56	EMIOS_GFR[F5]	eMIOS channel 5 Flag
0x0390	57	EMIOS_GFR[F6]	eMIOS channel 6 Flag
0x03A0	58	EMIOS_GFR[F7]	eMIOS channel 7 Flag
0x03B0	59	EMIOS_GFR[F8]	eMIOS channel 8 Flag
0x03C0	60	EMIOS_GFR[F9]	eMIOS channel 9 Flag
0x03D0	61	EMIOS_GFR[F10]	eMIOS channel 10 Flag
0x03E0	62	EMIOS_GFR[F11]	eMIOS channel 11 Flag
0x03F0	63	EMIOS_GFR[F12]	eMIOS channel 12 Flag
0x0400	64	EMIOS_GFR[F13]	eMIOS channel 13 Flag
0x0410	65	EMIOS_GFR[F14]	eMIOS channel 14 Flag
0x0420	66	EMIOS_GFR[F15]	eMIOS channel 15 Flag
eTPU_A			
0x0430	67	ETPU_MCR[MGEA] ETPU_MCR[MGEB] ETPU_MCR[ILFA] ETPU_MCR[ILFB] ETPU_MCR[SCMMISF]	eTPU Global Exception
0x0440	68	ETPU_CISR_A[CIS0]	eTPU Engine A Channel 0 Interrupt Status
0x0450	69	ETPU_CISR_A[CIS1]	eTPU Engine A Channel 1 Interrupt Status
0x0460	70	ETPU_CISR_A[CIS2]	eTPU Engine A Channel 2 Interrupt Status
0x0470	71	ETPU_CISR_A[CIS3]	eTPU Engine A Channel 3 Interrupt Status
0x0480	72	ETPU_CISR_A[CIS4]	eTPU Engine A Channel 4 Interrupt Status
0x0490	73	ETPU_CISR_A[CIS5]	eTPU Engine A Channel 5 Interrupt Status
0x04A0	74	ETPU_CISR_A[CIS6]	eTPU Engine A Channel 6 Interrupt Status
0x04B0	75	ETPU_CISR_A[CIS7]	eTPU Engine A Channel 7 Interrupt Status
0x04C0	76	ETPU_CISR_A[CIS8]	eTPU Engine A Channel 8 Interrupt Status
0x04D0	77	ETPU_CISR_A[CIS9]	eTPU Engine A Channel 9 Interrupt Status
0x04E0	78	ETPU_CISR_A[CIS10]	eTPU Engine A Channel 10 Interrupt Status
0x04F0	79	ETPU_CISR_A[CIS11]	eTPU Engine A Channel 11 Interrupt Status
0x0500	80	ETPU_CISR_A[CIS12]	eTPU Engine A Channel 12 Interrupt Status
0x0510	81	ETPU_CISR_A[CIS13]	eTPU Engine A Channel 13 Interrupt Status
0x0520	82	ETPU_CISR_A[CIS14]	eTPU Engine A Channel 14 Interrupt Status
0x0530	83	ETPU_CISR_A[CIS15]	eTPU Engine A Channel 15 Interrupt Status

Table 98. Interrupt Request Sources (continued)

Hardware Vector Mode Offset	Vector Number (1)	Source(2)	Description
0x0540	84	ETPU_CISR_A[CIS16]	eTPU Engine A Channel 16 Interrupt Status
0x0550	85	ETPU_CISR_A[CIS17]	eTPU Engine A Channel 17 Interrupt Status
0x0560	86	ETPU_CISR_A[CIS18]	eTPU Engine A Channel 18 Interrupt Status
0x0570	87	ETPU_CISR_A[CIS19]	eTPU Engine A Channel 19 Interrupt Status
0x0580	88	ETPU_CISR_A[CIS20]	eTPU Engine A Channel 20 Interrupt Status
0x0590	89	ETPU_CISR_A[CIS21]	eTPU Engine A Channel 21 Interrupt Status
0x05A0	90	ETPU_CISR_A[CIS22]	eTPU Engine A Channel 22 Interrupt Status
0x05B0	91	ETPU_CISR_A[CIS23]	eTPU Engine A Channel 23 Interrupt Status
0x05C0	92	ETPU_CISR_A[CIS24]	eTPU Engine A Channel 24 Interrupt Status
0x05D0	93	ETPU_CISR_A[CIS25]	eTPU Engine A Channel 25 Interrupt Status
0x05E0	94	ETPU_CISR_A[CIS26]	eTPU Engine A Channel 26 Interrupt Status
0x05F0	95	ETPU_CISR_A[CIS27]	eTPU Engine A Channel 27 Interrupt Status
0x0600	96	ETPU_CISR_A[CIS28]	eTPU Engine A Channel 28 Interrupt Status
0x0610	97	ETPU_CISR_A[CIS29]	eTPU Engine A Channel 29 Interrupt Status
0x0620	98	ETPU_CISR_A[CIS30]	eTPU Engine A Channel 30 Interrupt Status
0x0630	99	ETPU_CISR_A[CIS31]	eTPU Engine A Channel 31 Interrupt Status
eQADC			
0x0640	100	EQADC_FISR*x*[TORF] EQADC_FISR*x*[RFOF] EQADC_FISR*x*[CFUF]	eQADC combined overrun interrupt request s from all of the FIFOs: Trigger Overrun, Receive FIFO Overflow, and command FIFO Underflow
0x0650	101	EQADC_FISR0[NCF]	eQADC command FIFO 0 Non-Coherency Flag
0x0660	102	EQADC_FISR0[PF]	eQADC command FIFO 0 Pause Flag
0x0670	103	EQADC_FISR0[EOQF]	eQADC command FIFO 0 command queue End of Queue Flag
0x0680	104	EQADC_FISR0[CFFF]	eQADC Command FIFO 0 Fill Flag
0x0690	105	EQADC_FISR0[RFDF]	eQADC Receive FIFO 0 Drain Flag
0x06A0	106	EQADC_FISR1[NCF]	eQADC command FIFO 1 Non-Coherency Flag
0x06B0	107	EQADC_FISR1[PF]	eQADC command FIFO 1 Pause Flag
0x06C0	108	EQADC_FISR1[EOQF]	eQADC command FIFO 1 command queue End of Queue Flag
0x06D0	109	EQADC_FISR1[CFFF]	eQADC Command FIFO 1 Fill Flag
0x06E0	110	EQADC_FISR1[RFDF]	eQADC Receive FIFO 1 Drain Flag
0x06F0	111	EQADC_FISR2[NCF]	eQADC command FIFO 2 Non-Coherency Flag

Table 98. Interrupt Request Sources (continued)

Hardware Vector Mode Offset	Vector Number [(1)]	Source[(2)]	Description
0x0700	112	EQADC_FISR2[PF]	eQADC command FIFO 2 Pause Flag
0x0710	113	EQADC_FISR2[EOQF]	eQADC command FIFO 2 command queue End of Queue Flag
0x0720	114	EQADC_FISR2[CFFF]	eQADC Command FIFO 2 Fill Flag
0x0730	115	EQADC_FISR2[RFDF]	eQADC Receive FIFO 2 Drain Flag
0x0740	116	EQADC_FISR3[NCF]	eQADC command FIFO 3 Non-Coherency Flag
0x0750	117	EQADC_FISR3[PF]	eQADC command FIFO 3 Pause Flag
0x0760	118	EQADC_FISR3[EOQF]	eQADC command FIFO 3 command queue End of Queue Flag
0x0770	119	EQADC_FISR3[CFFF]	eQADC Command FIFO 3 Fill Flag
0x0780	120	EQADC_FISR3[RFDF]	eQADC Receive FIFO 3 Drain Flag
0x0790	121	EQADC_FISR4[NCF]	eQADC command FIFO 4 Non-Coherency Flag
0x07A0	122	EQADC_FISR4[PF]	eQADC command FIFO 4 Pause Flag
0x07B0	123	EQADC_FISR4[EOQF]	eQADC command FIFO 4 command queue End of Queue Flag
0x07C0	124	EQADC_FISR4[CFFF]	eQADC Command FIFO 4 Fill Flag
0x07D0	125	EQADC_FISR4[RFDF]	eQADC Receive FIFO 4 Drain Flag
0x07E0	126	EQADC_FISR5[NCF]	eQADC command FIFO 5 Non-Coherency Flag
0x07F0	127	EQADC_FISR5[PF]	eQADC command FIFO 5 Pause Flag
0x0800	128	EQADC_FISR5[EOQF]	eQADC command FIFO 5 command queue End of Queue Flag
0x0810	129	EQADC_FISR5[CFFF]	eQADC Command FIFO 5 Fill Flag
0x0820	130	EQADC_FISR5[RFDF]	eQADC Receive FIFO 5 Drain Flag
DSPI			
0x0830	131	DSPI_BSR[TFUF] DSPI_BSR[RFOF] DSPI_BSR[SPEF] DSPI_BSR[DPEF]	DSPI_B combined overrun interrupt requests: Transmit FIFO Underflow and Receive FIFO Overflow, SPI and DSI parity error
0x0840	132	DSPI_BSR[EOQF]	DSPI_B transmit FIFO End of Queue Flag
0x0850	133	DSPI_BSR[TFFF]	DSPI_B Transmit FIFO Fill Flag
0x0860	134	DSPI_BSR[TCF]	DSPI_B Transfer Complete Flag
0x0870	135	DSPI_BSR[RFDF] DSPI_BSR[DDIF]	DSPI_B Receive FIFO Drain Flag DSPI_B DSI Data Interrupt

Table 98. Interrupt Request Sources (continued)

Hardware Vector Mode Offset	Vector Number (1)	Source(2)	Description
0x0880	136	DSPI_CSR[TFUF] DSPI_CSR[RFOF] DSPI_CSR[SPEF] DSPI_CSR[DPEF]	DSPI_C combined overrun interrupt requests: Transmit FIFO Underflow and Receive FIFO Overflow , SPI and DSI parity error
0x0890	137	DSPI_CSR[EOQF]	DSPI_C transmit FIFO End of Queue Flag
0x08A0	138	DSPI_CSR[TFFF]	DSPI_C Transmit FIFO Fill Flag
0x08B0	139	DSPI_CSR[TCF]	DSPI_C Transfer Complete Flag
0x08C0	140	DSPI_CSR[RFDF] DSPI_CSR[DDIF]	DSPI_C Receive FIFO Drain Flag
0x08D0	141	DSPI_DSR[TFUF] DSPI_DSR[RFOF] DSPI_DSR[SPEF] DSPI_DSR[DPEF]	DSPI_D combined overrun interrupt requests: Transmit FIFO Underflow and Receive FIFO Overflow, SPI and DSI parity error
0x08E0	142	DSPI_DSR[EOQF]	DSPI_D transmit FIFO End of Queue Flag
0x08F0	143	DSPI_DSR[TFFF]	DSPI_D Transmit FIFO Fill Flag
0x0900	144	DSPI_DSR[TCF]	DSPI_D Transfer Complete Flag
0x0910	145	DSPI_DSR[RFDF] DSPI_DSR[DDIF]	DSPI_D Receive FIFO Drain Flag
eSCI			
0x0920	146	ESCIA_SR[TDRE] ESCIA_SR[TC] ESCIA_SR[RDRF] ESCIA_SR[IDLE] ESCIA_SR[OR] ESCIA_SR[NF] ESCIA_SR[FE] ESCIA_SR[PF] ESCIA_SR[BERR] ESCIA_SR[RXRDY] ESCIA_SR[TXRDY] ESCIA_SR[LWAKE] ESCIA_SR[STO] ESCIA_SR[PBERR] ESCIA_SR[CERR] ESCIA_SR[CKERR]	– Transmit Data Register Empty, Transmit Complete, Receive Data Register Full, Idle line, Overrun, Noise Flag, Framing Error Flag, and Parity Error Flag interrupt requests, SCI Status Register 2 Bit Error interrupt request, LIN Status Register 1 Receive Data Ready, Transmit Data Ready, Received LIN Wakeup Signal, Slave TimeOut, Physical Bus Error, CRC Error, Checksum Error, Frame Complete interrupts requests, and LIN Status Register 2 Receive Register Overflow – Combined Interrupt Requests of ESCI Module A
0x0930	147	GIFER[LRNE] GIFER[DRNE]	FlexRay LRAM non corrected error FlexRay DRAM non corrected error

Table 98. Interrupt Request Sources (continued)

Hardware Vector Mode Offset	Vector Number (1)	Source(2)	Description
0x0940	148	GIFER[LRCE] GIFER[DRCE]	FlexRay LRAM corrected error FlexRay DRAM corrected error
0x0950	149	ESCIB_SR[TDRE] ESCIB_SR[TC] ESCIB_SR[RDRF] ESCIB_SR[IDLE] ESCIB_SR[OR] ESCIB_SR[NF] ESCIB_SR[FE] ESCIB_SR[PF] ESCIB_SR[BERR] ESCIB_SR[RXRDY] ESCIB_SR[TXRDY] ESCIB_SR[LWAKE] ESCIB_SR[STO] ESCIB_SR[PBERR] ESCIB_SR[CERR] ESCIB_SR[CKERR]	Combined Interrupt Requests of ESCI Module B: Transmit Data Register Empty, Transmit Complete, Receive Data Register Full, Idle line, Overrun, Noise Flag, Framing Error Flag, and Parity Error Flag interrupt requests, SCI Status Register 2 Bit Error interrupt request, LIN Status Register 1 Receive Data Ready, Transmit Data Ready, Received LIN Wakeup Signal, Slave TimeOut, Physical Bus Error, CRC Error, Checksum Error, Frame Complete interrupts requests, and LIN Status Register 2 Receive Register Overflow
0x0960	150	Reserved	Reserved
0x0970	151	Reserved	Reserved
FlexCAN_A and FlexCAN_C			
0x0980	152	CANA_ESR[BOFF_INT]	FLEXCAN_A Bus off Interrupt
0x0990	153	CANA_ESR[ERR_INT]	FLEXCAN_A Error Interrupt
0x09A0	154	CAN_A.IPI_INT_WAKEIN	FLEXCAN_A wake up Interrupt
0x09B0	155	CANA_IFRL[BUF0]	FLEXCAN_A Buffer 0 Interrupt
0x09C0	156	CANA_IFRL[BUF1]	FLEXCAN_A Buffer 1 Interrupt
0x09D0	157	CANA_IFRL[BUF2]	FLEXCAN_A Buffer 2 Interrupt
0x09E0	158	CANA_IFRL[BUF3]	FLEXCAN_A Buffer 3 Interrupt
0x09F0	159	CANA_IFRL[BUF4]	FLEXCAN_A Buffer 4 Interrupt
0x0A00	160	CANA_IFRL[BUF5]	FLEXCAN_A Buffer 5 Interrupt
0x0A10	161	CANA_IFRL[BUF6]	FLEXCAN_A Buffer 6 Interrupt
0x0A20	162	CANA_IFRL[BUF7]	FLEXCAN_A Buffer 7 Interrupt
0x0A30	163	CANA_IFRL[BUF8]	FLEXCAN_A Buffer 8 Interrupt
0x0A40	164	CANA_IFRL[BUF9]	FLEXCAN_A Buffer 9 Interrupt
0x0A50	165	CANA_IFRL[BUF10]	FLEXCAN_A Buffer 10 Interrupt
0x0A60	166	CANA_IFRL[BUF11]	FLEXCAN_A Buffer 11 Interrupt

Table 98. Interrupt Request Sources (continued)

Hardware Vector Mode Offset	Vector Number (1)	Source(2)	Description
0x0A70	167	CANA_IFRL[BUF12]	FLEXCAN_A Buffer 12 Interrupt
0x0A80	168	CANA_IFRL[BUF13]	FLEXCAN_A Buffer 13 Interrupt
0x0A90	169	CANA_IFRL[BUF14]	FLEXCAN_A Buffer 14 Interrupt
0x0AA0	170	CANA_IFRL[BUF15]	FLEXCAN_A Buffer 15 Interrupt
0x0AB0	171	CANA_IFRL[BUF31I:BUF16]	FLEXCAN_A Buffers 31–16 Interrupts
0x0AC0	172	CANA_IFRH[BUF63I:BUF32]	FLEXCAN_A Buffers 63–32 Interrupts
0x0AD0	173	CANC_ESR[BOFF_INT]	FLEXCAN_C Bus off Interrupt
0x0AE0	174	CANC_ESR[ERR_INT]	FLEXCAN_C Error Interrupt
0x0AF0	175	CAN_C.IPI_INT_WAKEIN	FLEXCAN_C wake up Interrupt
0x0B00	176	CANC_IFRL[BUF0]	FLEXCAN_C Buffer 0 Interrupt
0x0B10	177	CANC_IFRL[BUF1]	FLEXCAN_C Buffer 1 Interrupt
0x0B20	178	CANC_IFRL[BUF2]	FLEXCAN_C Buffer 2 Interrupt
0x0B30	179	CANC_IFRL[BUF3]	FLEXCAN_C Buffer 3 Interrupt
0x0B40	180	CANC_IFRL[BUF4]	FLEXCAN_C Buffer 4 Interrupt
0x0B50	181	CANC_IFRL[BUF5]	FLEXCAN_C Buffer 5 Interrupt
0x0B60	182	CANC_IFRL[BUF6]	FLEXCAN_C Buffer 6 Interrupt
0x0B70	183	CANC_IFRL[BUF7]	FLEXCAN_C Buffer 7 Interrupt
0x0B80	184	CANC_IFRL[BUF8]	FLEXCAN_C Buffer 8 Interrupt
0x0B90	185	CANC_IFRL[BUF9]	FLEXCAN_C Buffer 9 Interrupt
0x0BA0	186	CANC_IFRL[BUF10]	FLEXCAN_C Buffer 10 Interrupt
0x0BB0	187	CANC_IFRL[BUF11]	FLEXCAN_C Buffer 11 Interrupt
0x0BC0	188	CANC_IFRL[BUF12]	FLEXCAN_C Buffer 12 Interrupt
0x0BD0	189	CANC_IFRL[BUF13]	FLEXCAN_C Buffer 13 Interrupt
0x0BE0	190	CANC_IFRL[BUF14]	FLEXCAN_C Buffer 14 Interrupt
0x0BF0	191	CANC_IFRL[BUF15]	FLEXCAN_C Buffer 15 Interrupt
0x0C00	192	CANC_IFRL[BUF31:BUF16]	FLEXCAN_C Buffers 31–16 Interrupts
0x0C10	193	CANC_IFRH[BUF63:BUF32]	FLEXCAN_C Buffers 63–32 Interrupts
0x0C20	194—196	Reserved	Reserved
0x0C50	197	DECFIL_A_In	Decimation A input (Fill)
0x0C60	198	DECFIL_A_Out	Decimation A output (Drain)
0x0C70	199	DECFIL_A_Err	Decimation A Error
0x0C80	200	STM0	STM[0]

Table 98. Interrupt Request Sources (continued)

Hardware Vector Mode Offset	Vector Number (1)	Source(2)	Description
0x0C90	201	STM1_or_STM2_or_STM3	STM[1:3]
eMIOS			
0x0CA0	202	EMIOS_GFR[F16]	eMIOS channel 16 Flag
0x0CB0	203	EMIOS_GFR[F17]	eMIOS channel 17 Flag
0x0CC0	204	EMIOS_GFR[F18]	eMIOS channel 18 Flag
0x0CD0	205	EMIOS_GFR[F19]	eMIOS channel 19 Flag
0x0CE0	206	EMIOS_GFR[F20]	eMIOS channel 20 Flag
0x0CF0	207	EMIOS_GFR[F21]	eMIOS channel 21 Flag
0x0D00	208	EMIOS_GFR[F22]	eMIOS channel 22 Flag
0x0D10	209	EMIOS_GFR[F23]	eMIOS channel 23 Flag
eDMA			
0x0D20	210	EDMA_ERRH[ERR63:ERR32]	eDMA channel Error flags 63–32
0x0D30	211	EDMA_IRQRH[INT32]	eDMA channel Interrupt 32
0x0D40	212	EDMA_IRQRH[INT33]	eDMA channel Interrupt 33
0x0D50	213	EDMA_IRQRH[INT34]	eDMA channel Interrupt 34
0x0D60	214	EDMA_IRQRH[INT35]	eDMA channel Interrupt 35
0x0D70	215	EDMA_IRQRH[INT36]	eDMA channel Interrupt 36
0x0D80	216	EDMA_IRQRH[INT37]	eDMA channel Interrupt 37
0x0D90	217	EDMA_IRQRH[INT38]	eDMA channel Interrupt 38
0x0DA0	218	EDMA_IRQRH[INT39]	eDMA channel Interrupt 39
0x0DB0	219	EDMA_IRQRH[INT40]	eDMA channel Interrupt 40
0x0DC0	220	EDMA_IRQRH[INT41]	eDMA channel Interrupt 41
0x0DD0	221	EDMA_IRQRH[INT42]	eDMA channel Interrupt 42
0x0DE0	222	EDMA_IRQRH[INT43]	eDMA channel Interrupt 43
0x0DF0	223	EDMA_IRQRH[INT44]	eDMA channel Interrupt 44
0x0E00	224	EDMA_IRQRH[INT45]	eDMA channel Interrupt 45
0x0E10	225	EDMA_IRQRH[INT46]	eDMA channel Interrupt 46
0x0E20	226	EDMA_IRQRH[INT47]	eDMA channel Interrupt 47
0x0E30	227	EDMA_IRQRH[INT48]	eDMA channel Interrupt 48
0x0E40	228	EDMA_IRQRH[INT49]	eDMA channel Interrupt 49
0x0E50	229	EDMA_IRQRH[INT50]	eDMA channel Interrupt 50
0x0E60	230	EDMA_IRQRH[INT51]	eDMA channel Interrupt 51
0x0E70	231	EDMA_IRQRH[INT52]	eDMA channel Interrupt 52

Table 98. Interrupt Request Sources (continued)

Hardware Vector Mode Offset	Vector Number (1)	Source(2)	Description
0x0E80	232	EDMA_IRQRH[INT53]	eDMA channel Interrupt 53
0x0E90	233	EDMA_IRQRH[INT54]	eDMA channel Interrupt 54
0x0EA0	234	EDMA_IRQRH[INT55]	eDMA channel Interrupt 55
0x0EB0	235	EDMA_IRQRH[INT56]	eDMA channel Interrupt 56
0x0EC0	236	EDMA_IRQRH[INT57]	eDMA channel Interrupt 57
0x0ED0	237	EDMA_IRQRH[INT58]	eDMA channel Interrupt 58
0x0EE0	238	EDMA_IRQRH[INT59]	eDMA channel Interrupt 59
0x0EF0	239	EDMA_IRQRH[INT60]	eDMA channel Interrupt 60
0x0F00	240	EDMA_IRQRH[INT61]	eDMA channel Interrupt 61
0x0F10	241	EDMA_IRQRH[INT62]	eDMA channel Interrupt 62
0x0F20	242	EDMA_IRQRH[INT63]	eDMA channel Interrupt 63
0x0F30	243–279	Reserved	Reserved
FlexCAN_B			
0x1180	280	CANB_ESR[BOFF_INT]	FLEXCAN_B Bus off Interrupt
0x1190	281	CANB_ESR[ERR_INT]	FLEXCAN_B Error Interrupt
0x11A0	282	CANB.IPI_INT_WAKEIN	FLEXCAN_B wake up Interrupt
0x11B0	283	CANB_IFRL[BUF0]	FLEXCAN_B Buffer 0 Interrupt
0x11C0	284	CANB_IFRL[BUF1]	FLEXCAN_B Buffer 1 Interrupt
0x11D0	285	CANB_IFRL[BUF2]	FLEXCAN_B Buffer 2 Interrupt
0x11E0	286	CANB_IFRL[BUF3]	FLEXCAN_B Buffer 3 Interrupt
0x11F0	287	CANB_IFRL[BUF4]	FLEXCAN_B Buffer 4 Interrupt
0x1200	288	CANB_IFRL[BUF5]	FLEXCAN_B Buffer 5 Interrupt
0x1210	289	CANB_IFRL[BUF6]	FLEXCAN_B Buffer 6 Interrupt
0x1220	290	CANB_IFRL[BUF7]	FLEXCAN_B Buffer 7 Interrupt
0x1230	291	CANB_IFRL[BUF8]	FLEXCAN_B Buffer 8 Interrupt
0x1240	292	CANB_IFRL[BUF9]	FLEXCAN_B Buffer 9 Interrupt
0x1250	293	CANB_IFRL[BUF10]	FLEXCAN_B Buffer 10 Interrupt
0x1260	294	CANB_IFRL[BUF11]	FLEXCAN_B Buffer 11 Interrupt
0x1270	295	CANB_IFRL[BUF12]	FLEXCAN_B Buffer 12 Interrupt
0x1280	296	CANB_IFRL[BUF13]	FLEXCAN_B Buffer 13 Interrupt
0x1290	297	CANB_IFRL[BUF14]	FLEXCAN_B Buffer 14 Interrupt
0x12A0	298	CANB_IFRL[BUF15]	FLEXCAN_B Buffer 15 Interrupt
0x12B0	299	CANB_IFRL[BUF31:BUF16]	FLEXCAN_B Buffers 31–16 Interrupts

Table 98. Interrupt Request Sources (continued)

Hardware Vector Mode Offset	Vector Number (1)	Source[2]	Description
0x12C0	300	CANB_IFRH[BUF63:BUF32]	FLEXCAN_B Buffers 63–32 Interrupts
0x12D0	301	PIT0	PIT[0]
0x12E0	302	PIT1	PIT[1]
0x12F0	303	PIT2	PIT[2]
0x1300	304	PIT3	PIT[3]
0x1310	305	RTI	RTI
0x1320	306	PMC	PMC
0x1330	307	ECSM_ESR[R1BC] ECSM_ESR[F1BC]	Flash and SRAM single-bit ECC error correction
0x1340	308–349	Reserved	Reserved
Flexray			
0x15E0	350	GIFER[MIF]	FlexRay MIF
0x15F0	351	GIFER[PRIF]	FlexRay PRIF
0x1600	352	GIFER[CHIF]	FlexRay CHIF
0x1610	353	GIFER[WUP_IF]	FlexRay WUP_IF
0x1620	354	GIFER[FBNE_F]	FlexRay FBNE_F
0x1630	355	GIFER[FANE_F]	FlexRay FANE_F
0x1640	356	GIFER[RBIF]	FlexRay RBIF
0x1650	357	GIFER[TBIF]	FlexRay TBIF
0x1660	358	STM1	STM[1]
0x1670	359	STM2	STM[2]
0x1680	360	STM3	STM[3]
0x1690	361–365	REACM_GE REACM[0] REACM[1] REACM[2] REACM[3]	Reaction Channel Global Error Reaction Channel 0–3 Interrupt
0x16E0	366	DECFIL_B_In	Decimation B input (Fill)
0x16F0	367	DECFIL_B_Out	Decimation B output (Drain)
0x1700	368	DECFIL_B_Err	Decimation B Error
0x1710	369—472	Reserved	Reserved
Reaction Module			

Table 98. Interrupt Request Sources (continued)

Hardware Vector Mode Offset	Vector Number[1]	Source[2]	Description
0x1690	361–365	REACM_GE REACM[0] REACM[1] REACM[2] REACM[3]	Reaction Channel Global Error Reaction Channel 0–3Interrupt
0x1D90	473	ESCIC_SR[TDRE] ESCIC_SR[TC] ESCIC_SR[RDRF] ESCIC_SR[IDLE] ESCIC_SR[OR] ESCIC_SR[NF] ESCIC_SR[FE] ESCIC_SR[PF] ESCIC_SR[BERR] ESCIC_SR[RXRDY] ESCIC_SR[TXRDY] ESCIC_SR[LWAKE] ESCIC_SR[STO] ESCIC_SR[PBERR] ESCIC_SR[CERR] ESCIC_SR[CKERR]	Combined Interrupt Requests of ESCI Module C: Transmit Data Register Empty, Transmit Complete, Receive Data Register Full, Idle line, Overrun, Noise Flag, Framing Error Flag, and Parity Error Flag interrupt requests, SCI Status Register 2 Bit Error interrupt request, LIN Status Register 1 Receive Data Ready, Transmit Data Ready, Received LIN Wakeup Signal, Slave TimeOut, Physical Bus Error, CRC Error, Checksum Error, Frame Complete interrupts requests, and LIN Status Register 2 Receive Register Overflow
0x1DA0	474—483	Reserved	Reserved
0x1E40	484—485	REACM[4] REACM[5]	Reaction Channel 4–5 Interrupts

1. The maximum vector number (485) is used to identify the location of the last available interrupt vector in memory for this device. Because blocks of memory throughout the total memory map are used for other purposes, the maximum vector number does not indicate the total number of available interrupt sources for this device.
2. Interrupt requests from the same module location are ORed together.

The peripheral or software configurable interrupt request asserts when the PRI*n* value in the interrupt priority select register (INTC_PSRn) is greater than the PRI*n* value in interrupt current priority register (INTC_CPR).

If an asserted peripheral or software configurable interrupt request negates before the processor acknowledges its request, the interrupt request can reassert and remain asserted. If this occurs, the processor uses the INTC_PSR*n* value to locate the IRQ vector, and updates the PRI*n* value in the INTC_CPR with the PRI*n* value in INTC_PSR*n*.

Clearing the peripheral interrupt request enable bit for the peripheral initiating the request, or setting the IRQ mask bit has the same consequences as clearing its flag bit. Setting its enable bit or clearing its mask bit while its flag bit is asserted has the same effect on the INTC as an interrupt event setting the flag bit.

Peripheral interrupt requests

An interrupt event in a peripheral's hardware sets a flag bit, which resides in that peripheral. The interrupt request from the peripheral is driven by that flag bit.

The time from when the peripheral starts to drive its peripheral interrupt request to the INTC to the time that the INTC starts to drive the interrupt request to the processor is three clocks.

Software configurable interrupt requests

The software set/clear interrupt registers (INTC_SSCIR*x*_*x*) support the setting or clearing of software-configurable interrupt requests. These registers contain eight independent sets of bits to set and clear a corresponding flag bit by software. With the exception of being set by software, this flag bit behaves the same as a flag bit set within a peripheral. This flag bit generates an interrupt request within the INTC just like a peripheral interrupt request.

An interrupt request is triggered by software writing a 1 to the SET*n* bit in INTC software set/clear interrupt registers (INTC_SSCIR0–INTC_SSCIR7). This write sets a CLR*n* flag bit that generates an interrupt request. The interrupt request is cleared by writing a 1 to the CLR*n* bit. Specific behavior includes the following:

- Writing a 1 to SET*n* leaves SET*n* unchanged at 0 but sets the flag bit (CLR*n* bit).
- Writing a 0 to SET*n* has no effect.
- Writing a 1 to CLR*n* clears the flag (CLR*n*) bit.
- Writing a 0 to CLR*n* has no effect.
- If a 1 is written to a pair of SET*n* and CLR*n* bits at the same time, the flag (CLR*n*) is set, regardless of whether CLR*n* was asserted before the write.

The time from the write to the SET*n* bit to the time that the INTC starts to drive the interrupt request to the processor is four clocks.

Unique vector for each interrupt request source

Each peripheral and software configurable interrupt request is assigned a hardwired unique 9-bit vector. Software configurable interrupts 0–7 are assigned vectors 0–7, respectively. The peripheral interrupt requests are assigned vectors 8 to as high as needed to cover all of the peripheral interrupt requests.

14.5.2 Priority management

The asserted interrupt requests are compared to each other based on their PRI*n* values in INTC priority select registers (INTC_PSR0–INTC_PSR485). The result of the comparison also is compared to PRI in INTC current priority register (INTC_CPR). The results of those comparisons are used to manage the priority of the ISR being executed by the processor. The LIFO also assists in managing the priority.

Current priority and preemption

The priority arbitrator, selector, encoder, and comparator submodules shown in *Figure 85* are used to compare the priority of the asserted interrupt requests to the current priority. If the priority of any asserted peripheral or software configurable interrupt request is higher than the current priority, then the interrupt request to the processor is asserted. Also, a unique vector for the preempting peripheral or software configurable interrupt request is generated for INTC interrupt acknowledge register (INTC_IACKR), and if in hardware vector mode, for the interrupt vector provided to the processor.

Priority arbitrator submodule

The priority arbitrator submodule compares all the priorities of all of the asserted interrupt requests, both peripheral and software configurable. The output of the priority arbitrator submodule is the highest of those priorities. Also, any interrupt requests which have this highest priority are output as asserted interrupt requests to the request selector submodule.

Request selector submodule

If only one interrupt request from the priority arbitrator submodule is asserted, then it is passed as asserted to the vector encoder submodule. If multiple interrupt requests from the priority arbitrator submodule are asserted, then only the one with the lowest vector is passed as asserted to the vector encoder submodule. The lower vector is chosen regardless of the time order of the assertions of the peripheral or software configurable interrupt requests.

Vector encoder submodule

The vector encoder submodule generates the unique 9-bit vector for the asserted interrupt request from the request selector submodule.

Priority comparator submodule

The priority comparator submodule compares the highest priority output from the priority arbitrator submodule with PRI in INTC_CPR. If the priority comparator submodule detects that this highest priority is higher than the current priority, then it asserts the interrupt request to the processor. This interrupt request to the processor asserts whether this highest priority is raised above the value of PRI in INTC_CPR or the PRI value in INTC_CPR is lowered below this highest priority. This highest priority then becomes the new priority which is written to PRI in INTC_CPR when the interrupt request to the processor is acknowledged. Interrupt requests whose PRI*n* in INTC_PSR*n* are zero does not cause a preemption because their PRI*n* is not higher than PRI in INTC_CPR.

LIFO

The LIFO stores the preempted PRI values from the INTC_CPR. Therefore, because these priorities are stacked within the INTC, if interrupts need to be enabled during the ISR, at the beginning of the interrupt exception handler the PRI value in the INTC_CPR does not need to be loaded from the INTC_CPR and stored onto the context stack. Likewise at the end of the interrupt exception handler, the priority does not need to be loaded from the context stack and stored into the INTC_CPR.

The PRI value in the INTC_CPR is pushed onto the LIFO when the INTC_IACKR is read in software vector mode or the interrupt acknowledge signal from the processor is asserted in hardware vector mode. The priority is popped into PRI in the INTC_CPR whenever the INTC_EOIR is written.

Although the INTC supports 16 priorities, an ISR executing with PRI in the INTC_CPR equal to 15 is not preempted. Therefore, the LIFO supports the stacking of 15 priorities. However, the LIFO is only 14 entries deep. An entry for a priority of 0 is not needed because of how pushing onto a full LIFO and popping an empty LIFO are treated. If the LIFO is pushed 15 or more times than it is popped, the priorities first pushed are overwritten. A priority of 0 is an overwritten priority. However, the LIFO pop zeros if it is popped more times than it is pushed. Therefore, although a priority of 0 was overwritten, it is regenerated with the popping of an empty LIFO.

The LIFO is not memory mapped.

14.5.3 Details on handshaking with processor

Software vector mode handshaking

Acknowledging interrupt request to processor

A timing diagram of the interrupt request and acknowledge handshaking in software vector mode, along with the handshaking near the end of the interrupt exception handler, is shown in *Figure 99*. The INTC examines the peripheral and software configurable interrupt requests. When it finds an asserted peripheral or software configurable interrupt request with a higher priority than PRI in INTC current priority register (INTC_CPR), it asserts the interrupt request to the processor. The INTVEC field in INTC interrupt acknowledge register (INTC_IACKR) is updated with the preempting interrupt request's vector when the interrupt request to the processor is asserted. The INTVEC field retains that value until the next time the interrupt request to the processor is asserted. The rest of the handshaking is described in *Section , Software vector mode*.

End-of-interrupt exception handler

Before the interrupt exception handling completes, INTC end-of-interrupt register (INTC_EOIR) must be written. When it is written, the LIFO is popped so that the preempted priority is restored into PRI of the INTC_CPR. Before it is written, the peripheral or software configurable flag bit must be cleared so that the peripheral or software configurable interrupt request is negated.

Note: To ensure proper operation across all Power Architecture MCUs, execute an MBAR or MSYNC instruction between the access to clear the flag bit and the write to the INTC_EOIR.

When returning from the preemption, the INTC does not search for the peripheral or software configurable interrupt request whose ISR was preempted. Depending on how much the ISR progressed, that interrupt request can no longer be asserted. When PRI in INTC_CPR is lowered to the priority of the preempted ISR, the interrupt request for the preempted ISR or any other asserted peripheral or software configurable interrupt request at or below that priority does not cause a preemption. Instead, after the restoration of the preempted context, the processor returns to the instruction address that it was to next execute before it was preempted. This next instruction is part of the preempted ISR or the interrupt exception handler's prolog or epilog.

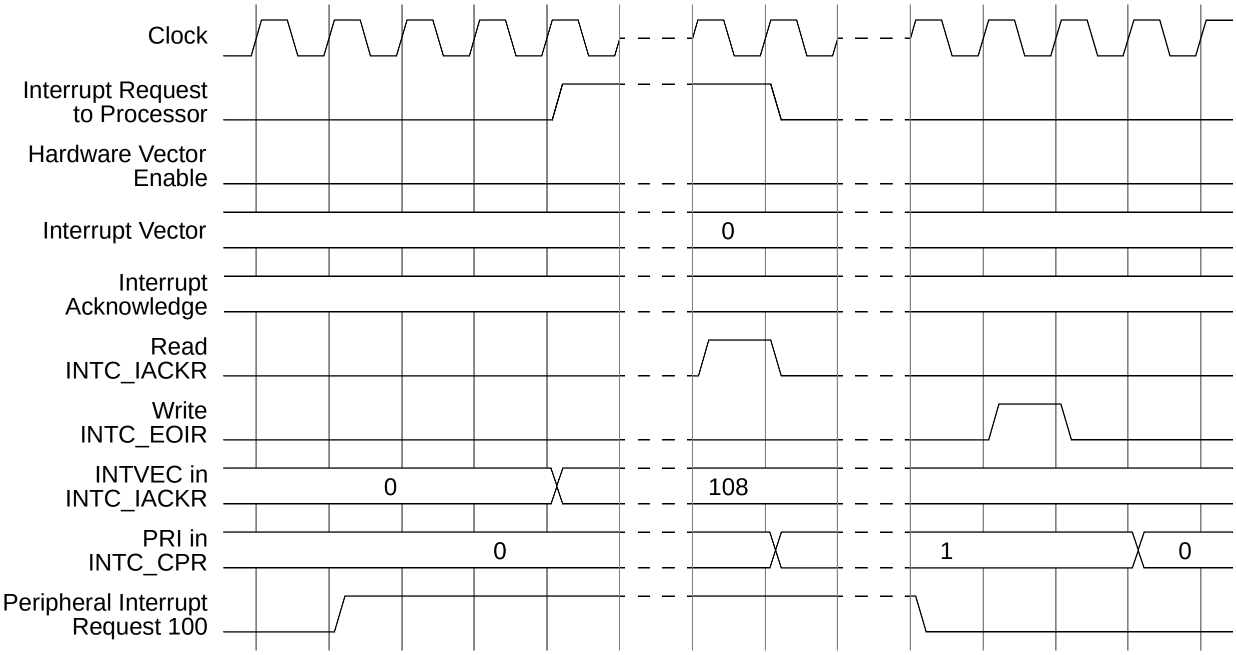


Figure 99. Software Vector Mode Handshaking Timing Diagram

Hardware vector mode handshaking

A timing diagram of the interrupt request and acknowledge handshaking in hardware vector mode, along with the handshaking near the end of the interrupt exception handler, is shown in *Figure 100*. As in software vector mode, the INTC examines the peripheral and software configurable interrupt requests, and when it finds an asserted one with a higher priority than PRI in INTC_CPR, it asserts the interrupt request to the processor. The INTVEC field in the INTC_IACKR is updated with the preempting peripheral or software configurable interrupt request's vector when the interrupt request to the processor is asserted. The INTVEC field retains that value until the next time the interrupt request to the processor is asserted. In addition, the value of the interrupt vector to the processor matches the value of the INTVEC field in the INTC_IACKR. The rest of the handshaking is described in *Section , Hardware vector mode*.

The handshaking near the end of the interrupt exception handler, that is the writing to the INTC_EOIR, is the same as in software vector mode. Refer to *Section , End-of-interrupt exception handler*.

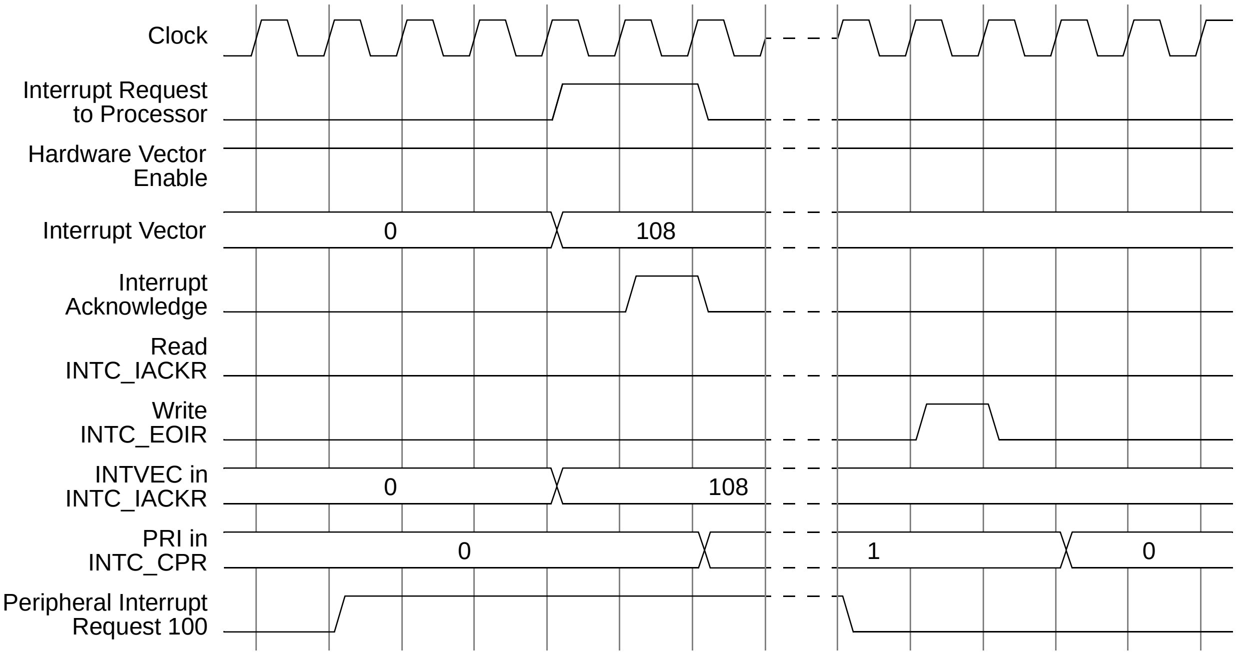


Figure 100. Hardware Vector Mode Handshaking Timing Diagram

14.6 Initialization and application information

14.6.1 Initialization flow

After exiting reset, all of the PRI*n* fields in INTC priority select registers (INTC_PSR0–INTC_PSR485) is zero, and PRI in INTC current priority register (INTC_CPR) is 15. These reset values prevent the INTC from asserting the interrupt request to the processor. The enable or mask bits in the peripherals are reset such that the peripheral interrupt requests are negated.

An initialization sequence that allows the peripheral and software configurable interrupt requests to generate an interrupt request to the processor is:

```
interrupt_request_initialization:
configure VTES and HVEN in INTC_MCR
configure VTBA in INTC_IACKR
raise the PRIn fields in INTC_PSRn
set the enable bits or clear the mask bits for the peripheral interrupt
requests
lower PRI in INTC_CPR to zero
enable processor recognition of interrupts
```

14.6.2 Interrupt exception handler

These example interrupt exception handlers use Power Architecture assembly code.

Software vector mode

```
interrupt_exception_handler:
code to create stack frame, save working register, and save SRR0 and SRR1
```

```
lis   r3,INTC_IACKR@ha# form adjusted upper half of INTC_IACKR address
lwz   r3,INTC_IACKR@l(r3)# load INTC_IACKR, which clears request to processor
lwz   r3,0x0(r3)      # load address of ISR from vector table
wrteei1               # enable processor recognition of interrupts

code to save rest of context required by e500 EABI

mtlr  r3              # move the INTC_IACKR address into the link register
blrl                  # branch to ISR; link register updated with epilog
                      # address

epilog:
code to restore most of context required by e500 EABI

# Popping the LIFO after the restoration of most of the context and the
disabling of processor
# recognition of interrupts eases the calculation of the maximum stack depth
at the cost of
# postponing the servicing of the next interrupt request.
mbar                  # ensure store to clear flag bit has completed
lis   r3,INTC_EOIR@ha # form adjusted upper half of INTC_EOIR address
li    r4,0x0          # form 0 to write to INTC_EOIR
wrteei0               # disable processor recognition of interrupts
stw   r4,INTC_EOIR@l(r3) # store to INTC_EOIR, informing INTC to lower
priority

code to restore SRR0 and SRR1, restore working registers, and delete stack
frame

rfi

vector_table_base_address:
address of ISR for interrupt with vector 0
address of ISR for interrupt with vector 1
      .
      .
      .
address of ISR for interrupt with vector 510
address of ISR for interrupt with vector 511


ISRx:
code to service the interrupt event
code to clear flag bit which drives interrupt request to INTC

blr                   # return to epilog
```

Hardware vector mode

This interrupt exception handler is useful with processor and system bus implementations that support a hardware vector. In this example, each `interrupt_exception_handlerx` has space for only four instructions, and therefore a branch to `interrupt_exception_handler_continuedx` is needed.

```
interrupt_exception_handlerx:
b     interrupt_exception_handler_continuedx# 4 instructions available,
branch to continue
```

```
interrupt_exception_handler_continuedx:
code to create stack frame, save working register, and save SRR0 and SRR1

wrteei1              # enable processor recognition of interrupts

code to save rest of context required by e500 EABI

bl   ISRx            # branch to ISR for interrupt with vector x


epilog:
code to restore most of context required by e500 EABI

# Popping the LIFO after the restoration of most of the context and the
disabling of processor
# recognition of interrupts eases the calculation of the maximum stack depth
at the cost of
# postponing the servicing of the next interrupt request.
mbar                 # ensure store to clear flag bit has completed
lis  r3,INTC_EOIR@ha # form adjusted upper half of INTC_EOIR address
li   r4,0x0          # form 0 to write to INTC_EOIR
wrteei0              # disable processor recognition of interrupts
stw  r4,INTC_EOIR@l(r3)# store to INTC_EOIR, informing INTC to lower
priority

code to restore SRR0 and SRR1, restore working registers, and delete stack
frame

rfi


ISRx:
code to service the interrupt event
code to clear flag bit which drives interrupt request to INTC

blr                  # branch to epilog
```

14.6.3 ISR, RTOS, and task hierarchy

The RTOS and all of the tasks under its control typically execute with PRI in INTC current priority register (INTC_CPR) having a value of 0. The RTOS execute the tasks according to whatever priority scheme that it has, but that priority scheme is independent and has a lower priority of execution than the priority scheme of the INTC. In other words, the ISRs execute above INTC_CPR priority 0 and outside the control of the RTOS, the RTOS executes at INTC_CPR priority 0, and while the tasks execute at different priorities under the control of the RTOS, they also execute at INTC_CPR priority 0.

If a task shares a resource with an ISR and the PCP is being used to manage that shared resource, then the task’s priority can be elevated in the INTC_CPR while the shared resource is being accessed.

An ISR whose PRI*n* in INTC priority select registers (INTC_PSR0–INTC_PSR485) has a value of 0 does not cause an interrupt request to the processor, even if its peripheral or software configurable interrupt request is asserted. For a peripheral interrupt request, not setting its enable bit or disabling the mask bit causes it to remain negated, which consequently also does not cause an interrupt request to the processor. Since the ISRs are outside the control of the RTOS, this ISR does not run unless called by another ISR or the interrupt exception handler, perhaps after executing another ISR.

14.6.4 Order of execution

An ISR with a higher priority can preempt an ISR with a lower priority, regardless of the unique vectors associated with each of their peripheral or software configurable interrupt requests. However, if multiple peripheral or software configurable interrupt requests are asserted, more than one has the highest priority, and that priority is high enough to cause preemption, the INTC selects the one with the lowest unique vector regardless of the order in time that they asserted. However, the ability to meet deadlines with this scheduling scheme is no less than if the ISRs execute in the time order that their peripheral or software configurable interrupt requests asserted.

The example in *Table 99* shows the order of execution of both ISRs with different priorities, and with the same priority.

Table 99. Order of ISR Execution Example

Step	Step Description	Code Executing At End of Step						PRI in INTC_CPR at End of Step
		RTOS	ISR108[1]	ISR208	ISR308	ISR408	Interrupt Exception Handler	
1	RTOS at priority 0 is executing.	X						0
2	Peripheral interrupt request 100 at priority 1 asserts. Interrupt taken.		X					1
3	Peripheral interrupt request 400 at priority 4 asserts. Interrupt taken.					X		4
4	Peripheral interrupt request 300 at priority 3 asserts.					X		4
5	Peripheral interrupt request 200 at priority 3 asserts.					X		4
6	ISR408 completes. Interrupt exception handler writes to INTC_EOIR.						X	1
7	Interrupt taken. ISR208 starts to execute, even though peripheral interrupt request 300 asserted first.			X				3
8	ISR208 completes. Interrupt exception handler writes to INTC_EOIR.						X	1
9	Interrupt taken. ISR308 starts to execute.				X			3
10	ISR308 completes. Interrupt exception handler writes to INTC_EOIR.						X	1

Table 99. Order of ISR Execution Example (continued)

Step	Step Description	Code Executing At End of Step						PRI in INTC_CPR at End of Step
		RTOS	ISR108[1]	ISR208	ISR308	ISR408	Interrupt Exception Handler	
11	ISR108 completes. Interrupt exception handler writes to INTC_EOIR.						X	0
12	RTOS continues execution.	X						0

1. ISR108 executes for peripheral interrupt request 100 because the first eight ISRs are for software configurable interrupt requests.

14.6.5 Priority ceiling protocol

Elevating priority

The PRI field in INTC current priority register (INTC_CPR) is elevated in the OSEK PCP to the ceiling of all of the priorities of the ISRs that share a resource. This protocol therefore allows coherent accesses of the ISRs to that shared resource.

For example, ISR1 has a priority of 1, ISR2 has a priority of 2, and ISR3 has a priority of 3. They all share the same resource. Before ISR1 or ISR2 can access that resource, they must raise the PRI value in INTC_CPR to 3, the ceiling of all of the ISR priorities. After they release the resource, the PRI value in INTC_CPR can be lowered. If they do not raise their priority, then ISR2 can preempt ISR1, and ISR3 can preempt ISR1 or ISR2, possibly corrupting the shared resource. Another possible failure mechanism is deadlock. If the higher priority ISR needs the lower priority ISR to release the resource before it can continue, but the lower priority ISR cannot release the resource until the higher priority ISR completes and execution returns to the lower priority ISR.

Using the PCP instead of disabling processor recognition of all interrupts eliminates the time when accessing a shared resource that all higher priority interrupts are blocked. For example, while ISR3 cannot preempt ISR1 while it is accessing the shared resource, all of the ISRs with a priority higher than 3 can preempt ISR1.

Ensuring coherency

Non-coherent accesses to a shared resource can occur. As an example, ISR1 and ISR2 both share a resource. ISR1 has a lower priority, therefore it executes and then writes the new PRI value in the current priority register (INTC_CPR). The next instruction writes a value to a shared coherent data block.

If INTC asserts the ISR2 interrupt request to the processor just before or at the same time as the first ISR1 write, it is possible for both the ISR1 and ISR2 writes to execute while the processor responds to the INTC request, discards the transactions, and flushes the processing pipeline. However, ISR2 cannot access the data block coherently because the data block is now corrupted.

OSEK uses the GetResource and ReleaseResource system services to manage access to a shared resource. To prevent corrupting a coherent data block, use the following code to modify the PRI in INTC_CPR. Interrupts must be enabled before executing the following GetResource code sequence:

```
GetResource:
raise PRI
mbar
isync

ReleaseResource:
mbar
lower PRI
```

14.6.6 Selecting priorities according to request rates and deadlines

The selection of the priorities for the ISRs can be made using Rate Monotonic Scheduling (RMS) or a superset of it, Deadline Monotonic Scheduling (DMS). In RMS, the ISRs that have higher request rates have higher priorities. In DMS, if the deadline is before the next time the ISR is requested, then the ISR is assigned a priority according to the time from the request for the ISR to the deadline, not from the time of the request for the ISR to the next request for it.

For example, ISR1 executes every 100 µs, ISR2 executes every 200 µs, and ISR3 executes every 300 µs. ISR1 has a higher priority than ISR2, which has a higher priority than ISR3. However, if ISR3 has a deadline of 150 µs, then it has a higher priority than ISR2.

The INTC has 16 priorities, which can be considerably less than the number of ISRs. In this case, group the ISRs with other ISRs that have similar deadlines. For example, when a priority is allocated for every time, the request rate doubles ISRs with request rates around 1 ms share a priority; ISRs with request rates around 500 µs share a priority; ISRs with request rates around 250 µs share a priority, etc. With this approach, a range of ISR request rates of 2^{16} can be covered, regardless of the number of ISRs.

Reducing the number of priorities reduces the processor's ability to meet its deadlines. However, it also allows easier management of ISRs with similar deadlines that share a resource. They do not need to use the PCP to access the shared resource.

14.6.7 Software configurable interrupt requests

The software configurable interrupt requests can be used in two ways. They can be used to schedule a lower priority portion of an ISR and for processors to interrupt other processors in a multiple processor system.

Scheduling a lower priority portion of an ISR

A portion of an ISR needs to be executed at the PRI*n* value in INTC priority select registers (INTC_PSR0–INTC_PSR485), which becomes the PRI value in INTC current priority register (INTC_CPR) with the interrupt acknowledgement. The ISR, however, can have a portion of it that does not need to be executed at this higher priority. Therefore, executing this later portion that does not need to be executed at this higher priority can prevent the execution of ISRs, which do not have a higher priority than the earlier portion of the ISR but do have a higher priority than what the later portion of the ISR needs. This preemptive scheduling inefficiency reduces the processor's ability to meet its deadlines.

One option is for the ISR to complete the earlier higher priority portion, but then schedule through the RTOS a task to execute the later lower priority portion. However, some RTOSs can require a large amount of time for an ISR to schedule a task. Therefore, a second option for the ISR is, after completing the higher priority portion, to set a SET*n* bit in INTC software set/clear interrupt registers (INTC_SSCIR0–INTC_SSCIR7). Writing a 1 to SET*n* causes a software configurable interrupt request. This software configurable interrupt request, which usually has a lower PRI*n* value in the INTC_PSR*n*, does not cause preemptive scheduling inefficiencies.

After generating a software configurable interrupt request, the higher priority ISR completes. The lower priority ISR is scheduled according to its priority. Execution of the higher priority ISR is not resumed after the completion of the lower priority ISR.

Scheduling an ISR on another processor

Since the SET*n* bits in the INTC_SSCIR*n* are memory mapped, processors in multiple processor systems can schedule ISRs on the other processors. One application is that one processor simply wants to command another processor to perform a piece of work, and the initiating processor does not need to use the results of that work. If the initiating processor is concerned that processor executing the software configurable ISR has not completed the work before asking it to again execute that ISR, it can check if the corresponding CLR*n* bit in INTC_SSCIR*n* is asserted before again writing a 1 to the SET*n* bit.

Another application is the sharing of a block of data. For example, a first processor has completed accessing a block of data and wants a second processor to then access it. Furthermore, after the second processor has completed accessing the block of data, the first processor again wants to access it. The accesses to the block of data must be done coherently. The procedure is that the first processor writes a 1 to a SET*n* bit on the second processor. The second processor, after accessing the block of data, clears the corresponding CLR*n* bit and then writes 1 to a SET*n* bit on the first processor, informing it that it now can access the block of data.

14.6.8 Lowering priority within an ISR

In implementations without the software-configurable interrupt requests in the INTC software set/clear interrupt registers (INTC_SSCIR0–INTC_SSCIR7), a way — besides scheduling a task through an RTOS — to prevent preemptive scheduling inefficiencies with an ISR whose work spans multiple priorities (as described in *Section , Scheduling a lower priority portion of an ISR*) is to lower the current priority. However, the INTC has a LIFO whose depth is determined by the number of priorities.

Note: Lowering the PRI value in INTC current priority register (INTC_CPR) within an ISR to below the ISR's corresponding PRI value in INTC priority select registers (INTC_PSR0–INTC_PSR485) allows more preemptions than the depth of the LIFO can support.

Therefore, through its use of the LIFO the INTC does not support lowering the current priority within an ISR as a way to avoid preemptive scheduling inefficiencies.

14.6.9 Negating an interrupt request outside of its ISR

Negating an interrupt request as a side effect of an ISR

Some peripherals have flag bits which can be cleared as a side effect of servicing a peripheral interrupt request. For example, reading a specific register can clear the flag bits, and consequently their corresponding interrupt requests, too. This clearing as a side effect of servicing a peripheral interrupt request can cause the negation of other peripheral interrupt requests besides the peripheral interrupt request whose ISR presently is executing. This negating of a peripheral interrupt request outside of its ISR can be a desired effect.

Negating multiple interrupt requests in one ISR

An ISR can clear other flag bits besides its own flag bit. One reason that an ISR clears multiple flag bits is because it serviced those other flag bits, and therefore the ISRs for these other flag bits do not need to be executed.

Proper setting of interrupt request priority

Whether an interrupt request negates outside of its own ISR due to the side effect of an ISR execution or the intentional clearing a flag bit, the priorities of the peripheral or software configurable interrupt requests for these other flag bits must be selected properly. Their PRI*n* values in INTC priority select registers (INTC_PSR0–INTC_PSR485) must be selected to be at or lower than the priority of the ISR that cleared their flag bits. Otherwise, those flag bits still can cause the interrupt request to the processor to assert. Furthermore, the clearing of these other flag bits also has the same timing relationship to the writing to INTC end-of-interrupt register (INTC_EOIR) as the clearing of the flag bit that caused the present ISR to be executed. Refer to *Section , End-of-interrupt exception handler*, for more information.

A flag bit whose enable bit or mask bit is negating its peripheral interrupt request can be cleared at any time, regardless of the peripheral interrupt request's PRI*n* value in INTC_PSR*n*.

14.6.10 Examining LIFO contents

Normally you do not need to know the contents of the LIFO, or even how deep the LIFO is nested. Although the LIFO contents are not memory mapped, you can read the contents by popping the LIFO and reading the PRI field in the INTC current priority register (INTC_CPR). Disabling processor recognition of interrupts while examining the LIFO contents provides a coherent view of the preempted priorities.

The code sequence is:

```
pop_lifo:
store to INTC_EOIR
load INTC_CPR, examine PRI, and store onto stack
if PRI is not zero or value when interrupts were enabled, branch to
pop_lifo
```

When you are finished examining the LIFO contents, you can restore it in software vector mode using the following code sequence. In hardware vector mode, reading the INTC_IACKR does not push the INTC_CPR[PRI] onto the LIFO, therefore the LIFO contents cannot be restored in hardware vector mode.

```
push_lifo:
load stacked PRI value and store to INTC_CPR
load INTC_IACKR
if stacked PRI values are not depleted, branch to push_lifo
```

Note: *Reading the INTC_IACKR acknowledges the interrupt request to the processor and updates the INTC_CPR[PRI] with the priority of the preempting interrupt request. If the processor recognition of interrupts is disabled during the LIFO restoration, interrupt requests to the processor can go undetected. However, since the peripheral or software configurable interrupt requests are not cleared, the peripheral interrupt request to the processor re-asserts when INTC_CPR[PRI] is lower than the priorities of those peripheral or software configurable interrupt requests.*

15 System Integration Unit (SIU)

15.1 Overview

The System integration unit (SIU) controls this device’s reset configuration, pad configuration, external interrupt, general purpose I/O (GPIO), internal peripheral multiplexing, and the system reset operation. The reset configuration block contains the external pin boot configuration logic. The pad configuration block controls the static electrical characteristics of I/O pins. The GPIO block provides uniform and discrete input/output control of the MCU I/O pins. The reset controller performs reset monitoring of internal and external reset sources, and drives the $\overline{\text{RSTOUT}}$ pin. The SIU is accessed by the core through the peripheral bus.

15.2 Features

- System configuration
  - MCU reset configuration via external pins
  - Pad configuration control
- System reset monitoring and generation
  - Power-on reset support
  - Reset Status Register provides last reset source to software
  - Glitch detection on reset input
  - Software controlled reset assertion
- External interrupt
  - 15 interrupt requests
  - 1 Non-Maskable/Critical Interrupt request (NMI)
  - Rising or falling edge event detection
  - Programmable digital filter for glitch rejection
- GPIO
  - GPIO function on 163 I/O pins
  - Dedicated input and output registers for each GPIO pin
- Internal multiplexing
  - Allows serial and parallel chaining of DSPIs
  - Allows flexible selection of eQADC trigger inputs
  - Allows selection of interrupt requests between external pins and DSPI
  - Allows selection of some eTPU inputs from external eTPU pins or deserialized output from the DSPI module
  - Allows selection of serialized data source for the DSPI

15.3 Modes of operation

15.3.1 Normal mode

In normal mode, the SIU provides the register interface and logic that controls system configuration, the reset controller, and GPIO.

15.3.2 Debug mode

SIU operation in debug mode is identical to operation in normal mode.

15.4 Block diagram

Figure 101 is a block diagram of the SIU. The signals shown are external pins to the device. The SIU registers are accessed through the crossbar switch. Note that the Power-on Reset Detection block, Pad Interface/Pad Ring block, and Peripheral I/O Channels are external to the SIU.

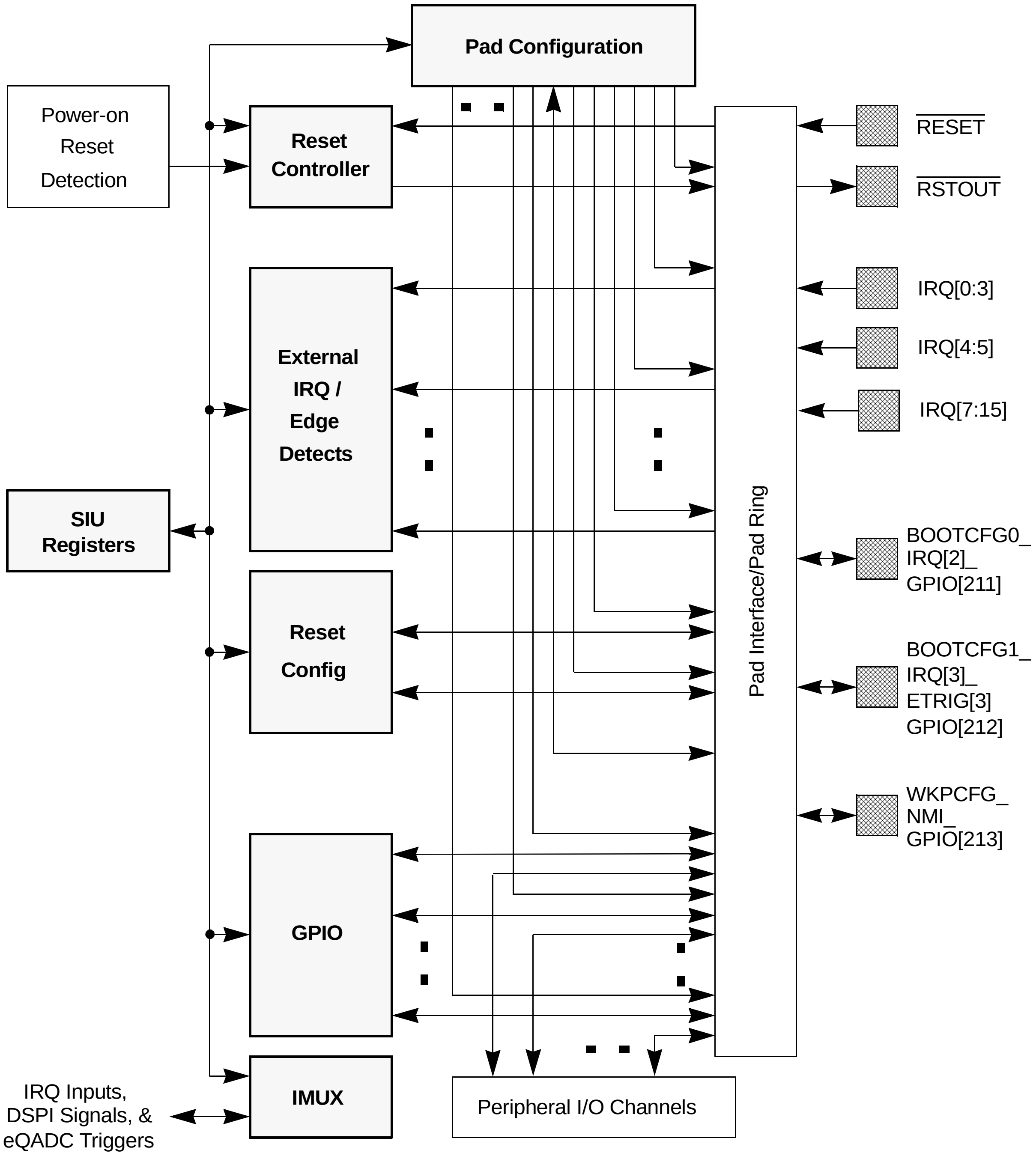


Figure 101. SIU block diagram

15.5 Signal description

Table 100 lists the external pins used by the SIU.

Table 100. SIU signal properties

Name	I/O type	Pad type	Function	Pull up/down(1)
RESETS				
$\overline{\text{RESET}}$	Input	—	Reset Input	Up
$\overline{\text{RSTOUT}}$	Output	Slow	Reset Output	Up
SYSTEM CONFIGURATION				
BOOTCFG0	Input	Slow	Boot Configuration Input	Down
BOOTCFG1	Input	Slow	Boot Configuration Input	Down
WKPCFG_ NMI_ GPIO[213]	Input Input I/O	Slow	Weak Pull Configuration Pin / Non-Maskable Interrupt / General Purpose I/O	Up — Up/Down
GPIO CONFIGURATION				
GPIO[0:245]	I/O	Slow	General Purpose I/O	Up/Down
EXTERNAL INTERRUPT				
IRQ[0:5,7:15]	Input	Slow	External Interrupt Request Input	—(2)

1. Internal weak pull up/down. The reset weak pull up/down state is given by the pull up/down state for the primary pin function. For example, the reset weak pull up/down state of the BOOTCFG1 pin is weak pull down enabled.
2. See *Table 3* in *Section 3.1, Signal Properties* for more information.

15.6 Memory map and register descriptions

15.6.1 Memory map

Table 101 is the address map for the SIU registers.

Table 101. SIU address map

Address	Use	Bits per register	Location
SIU_BASE	MCU ID Register 2 (SIU_MIDR2)	32	*on page 15-337*
SIU_BASE+0x4	MCU ID Register (SIU_MIDR)	32	*on page 15-339*
SIU_BASE+0x8	Reserved		
SIU_BASE+0xC	Reset Status Register (SIU_RSR)	32	*on page 15-340*
SIU_BASE+0x10	System Reset Control Register (SIU_SRCR)	32	*on page 15-342*
SIU_BASE+0x14	SIU External Interrupt Status Register (SIU_EISR)	32	*on page 15-343*
SIU_BASE+0x18	DMA/Interrupt Request Enable Register (SIU_DIRER)	32	*on page 15-344*

Table 101. SIU address map (continued)

Address	Use	Bits per register	Location
SIU_BASE+0x1C	DMA/Interrupt Request Select Register (SIU_DIRSR)	32	on page 15-345
SIU_BASE+0x20	Overrun Status Register (SIU_OSR)	32	on page 15-346
SIU_BASE+0x24	Overrun Request Enable Register (SIU_ORER)	32	on page 15-347
SIU_BASE+0x28	External IRQ Rising-Edge Event Enable Register (SIU_IREER)	32	on page 15-347
SIU_BASE+0x2C	External IRQ Falling-Edge Event Enable Register (SIU_IFEER)	32	on page 15-348
SIU_BASE+0x30	External IRQ Digital Filter Register (SIU_IDFR)	32	on page 15-349
SIU_BASE+0x34	IRQ Filtered Input Register (SIU_IFIR)	32	on page 15-350
SIU_BASE+0x38 – SIU_BASE+0x3F	Reserved		
SIU_BASE+0x40 – SIU_BASE+0x37B	Pad Configuration Register 0 (SIU_PCR0) – Pad Configuration Register 413 (SIU_PCR413)(1)	16	on page 15-351
SIU_BASE+0x37C – SIU_BASE+0x5FF	Reserved		
SIU_BASE+0x600 – SIU_BASE+0x79D	GPIO Pin Data Output Register 0 – 3 (SIU_GPDO0_3) – GPIO Pin Data Output Register 412 – 413 (SIU_GPDO412_413)[1]	8	on page 15-440
SIU_BASE+0x79E – SIU_BASE+0x7FF	Reserved		
SIU_BASE+0x800 – SIU_BASE+0x8E9	GPIO Pin Data Input Register 0 – 3 (SIU_GPDI0_3) – GPIO Pin Data Input Register 232 – 233 (SIU_GPDI232_233)(1)	8	on page 15-442
SIU_BASE+0x8EA – SIU_BASE+0x8FF	Reserved		
SIU_BASE+0x900	eQADC trigger IMUX Select Register (ETISR)(2)	32	on page 15-443
SIU_BASE+0x904	External Interrupt IMUX Select Register (EIISR)(3)	32	on page 15-445
SIU_BASE+0x908	DSPI IMUX Select Register (DISR)(4)	32	on page 15-447
SIU_BASE+0x90C	IMUX Select Register 3 (SIU_ISEL3)	32	on page 15-449
SIU_BASE+0x910 – SIU_BASE+0x91F	Reserved		
SIU_BASE+0x920	IMUX Select Register 8 (SIU_ISEL8)	32	on page 15-456

Table 101. SIU address map (continued)

Address	Use	Bits per register	Location
SIU_BASE+0x924	IMUX Select Register 9 (SIU_ISEL9)	32	*on page 15-457*
SIU_BASE+0x928	IMUX Select Register 10 (SIU_ISEL10)	32	*on page 15-459*
SIU_BASE+0x92C – SIU_BASE+0x97F	Reserved		
SIU_BASE+0x980	Chip Configuration Register (SIU_CCR)	32	*on page 15-460*
SIU_BASE+0x984	External Clock Control Register (SIU_ECCR)	32	*on page 15-461*
SIU_BASE+0x988	Compare A High Register (SIU_CARH)	32	*on page 15-463*
SIU_BASE+0x98C	Compare A Low Register (SIU_CARL)	32	*on page 15-463*
SIU_BASE+0x990	Compare B High Register (SIU_CBRH)	32	*on page 15-464*
SIU_BASE+0x994	Compare B Low Register (SIU_CBRL)	32	*on page 15-464*
SIU_BASE+0x998	Reserved		
SIU_BASE+0x9A0	System Clock Register (SIU_SYSDIV)	32	*on page 15-465*
SIU_BASE+0x9A4	Halt Register (SIU_HLT)	32	*on page 15-466*
SIU_BASE+0x9A8	Halt Acknowledge Register (SIU_HLTACK)	32	*on page 15-469*
SIU_BASE+0x9AC – SIU_BASE+0x9B3	Reserved		
SIU_BASE+0x9B4	Core MMU PID Control Register (SIU_EMPCR0)	32	*on page 15-471*
SIU_BASE+0x9B8 – SIU_BASE+0x9FF	Reserved		

1. Gaps exist in this memory space where I/O pins are not available in the specified package.
2. The ETISR is sometimes referred to as ISEL0
3. The EIISR is sometimes referred to as ISEL1
4. The DISR is sometimes referred to as ISEL2

15.6.2 MCU ID Register 2 (SIU_MIDR2)

The MCU ID Register 2 contains additional configuration information about the device.

SIU_BASE + 0x0

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Function	S_F	FLASH_SIZE_1				FLASH_SIZE_2				TEMP_ RANGE		Res.	MAX_ FREQ		Res.	SUP PLY
	S_F (1)	0	1	1	1	0	0	0	0	1	1	0	1	1	0	0
		7				0										

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Function	PART_NUMBER (ASCII Character)								Res.			EE	Res.			FR
	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	1
	A = 0x41															

1. S_F set with metal option

Figure 102. MCU ID Register 2 (SIU_MIDR2)

Table 102. SIU_MIDR2 field description

Bit	Name	Description
0	S_F	Identifies the manufacturer 1: ST
1–4	FLASH_SIZE_1	Define major Flash memory size (see *Table 103* for details)
5–8	FLASH_SIZE_2	Define Flash memory size, small granularity (see *Table 104* for details)
19–10	TEMP_RANGE	Define maximum operating range
11	—	Reserved for future enhancements
12–13	MAX_FREQ	Define maximum device speed 00b: 150 MHz 01b: 120 MHz 10b: 80 MHz
14	—	Reserved for future enhancements
15	SUPPLY	Defines if the part is 5 V or 3 V 1: 3 V 0: 5 V
16–23	PART_NUMBER	Contain the ASCII representation of the character that indicates the product family.
24–26	—	Reserved for future enhancements
27	EE	Indicates if Data Flash is present 1: Data Flash present 0: Data Flash not present
28–30	—	Reserved for future enhancements
31	FR	Indicates if Data FlexRay is present 1: FlexRay present 0: FlexRay not present

Table 103. Flash memory size

FLASH_SIZE_1 field	Size
0h	16 KB
1h	32 KB
2h	64 KB
3h	128 KB
4h	256 KB
5h	512 KB
6h	1024 KB
7h	2048 KB
	...
n	2^{4+n} KB

Table 104. Flash memory size detailed[1]

FLASH_SIZE_2 field	Size
0h	0x(Flash size 1)/8
1h	1x(Flash size 1)/8
2h	2x(Flash size 1)/8
...	...
n	nx(Flash size 1)/8

1. Total flash memory size = (flash size 1) + (flash size 2)

15.6.3 MCU ID Register (SIU_MIDR)

The MCU ID Register contains the part number and the package ID of the device.

SIU_BASE + 0x4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PARTNUM [0–15] (4 Digits)															
W																
Reset	0	1	0	1	0	1	1	0	0	1	0	0	0	0	0	0
	5				6				4				0			

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CSP	PKG [0–4]					Reserved		MASKNUM [0–3] - Major				MASKNUM [0–3] - Minor			
W																
Reset	—[1]	—[2]	—	—	—	—	0	0	0	0	0	0	0	0	0	0

1. Values corresponding to device packaging; see *Table 105*.
2. Values for these bits vary according to the package. See *Table 105* for details.

Figure 103. MCU ID Register (SIU_MIDR)

Table 105. SIU_MIDR field description

Bit	Name	Description
0–15	PARTNUM [0–15]	0x5640
16	CSP	CSP configuration: 1:Calibration tool package 0: Standard QFP package or BGA208 package
17–21	PKG [0–4]	Indicate the package the die is mounted in 10001: 176-pin QFP 10000: 208-ball BGA 10100: 324-ball BGA
22–23	—	Reserved
24–27	Major MASKNUM [0–3]	MCU major mask number; the current value applies to revision 0 and will be updated for each complete resynthesis
28–31	Minor MASKNUM [0–3]	MCU minor mask number; the current value applies to revision 0 and will be updated for each mask revision

15.6.4 Reset Status Register (SIU_RSR)

The Reset Status Register (SIU_RSR) reflects the most recent source, or sources, of reset. This register contains one bit for each reset source, except JTAG reset. A bit set to logic one indicates the type of reset that occurred. Simultaneous reset requests cause more than one bit to set at the same time. Once set, the reset source bits in the SIU_RSR remain set until another reset occurs. A Software External Reset causes the SERF bit to be set, but no previously set bits in the SIU_RSR will be cleared.

The unidirectional mode of reset operation is implemented, all registers named Mode 1 are implemented.

SIU_BASE + 0xC

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PORS	ERS	LLRS	LCRS	WDRS	0	SWTRS	0	0	0	0	0	0	0	SSRS	SERF
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset[1]	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	WKPCFG	0	0	0	0	0	0	0	0	0	0	0	ABR	BOOTCFG [0–1]		RGF
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨		▨
Reset[1]	U[2]	0	0	0	0	0	0	0	0	0	0	0	U[3]	0	0	0

▨ = Unimplemented or Reserved

1. The reset values for this register are defined for power-on reset only.
2. The reset value of this bit is determined by the value latched on the associated pin at the negation of the last reset.
3. The reset value of this bit is determined by the inverse of the value latched on the associated EVTO pin.

Figure 104. Reset Status Register (SIU_RSR)

Table 106. SIU_RSR field description

Bits	Name	Description
0	PORS	Power-On Reset Status 1: A Power-On Reset has occurred. 0: No Power-On Reset has occurred.
1	ERS	External Reset Status 1: An External Reset has occurred. 0: No External Reset has occurred.
2	LLRS	Loss of Lock Reset Status 1: A Loss of Lock Reset has occurred. 0: No Loss of Lock Reset has occurred.
3	LCRS	Loss of Clock Reset Status 1: A Loss of Clock Reset has occurred due to a loss of the reference or failure of the FMPLL. 0: No Loss of Clock Reset has occurred.
4	WDRS	Watchdog Timer/Debug Reset Status 1: A Watchdog Timer or Debug Reset has occurred. 0: No Watchdog Timer or Debug Reset has occurred.
5	—	Reserved
6	SWTRS	Software Watchdog Timer Reset Status 1: An enabled SWT Reset has occurred. 0: No enabled SWT Reset has occurred.

Table 106. SIU_RSR field description (continued)

Bits	Name	Description
7–13	—	Reserved
14	SSRS	Software System Reset Status 1: A Software System Reset has occurred. 0: No Software System Reset has occurred.
15	SERF	Software External Reset Flag 1: A Software External Reset has occurred. 0: No Software External Reset has occurred.
16	WKPCFG	Weak Pull Configuration Pin Status 1: WKPCFG pin latched during the last reset was logical one and weak pull up is the default setting. 0: WKPCFG pin latched during the last reset was logical zero and weak pull down is the default setting.
17–27	—	Reserved
28	ABR	Auto Baud Rate 1: Auto Baud Rate Enabled. 0: Auto Baud Rate Disabled.
29–30	BOOTCFG[0:1]	Reset Configuration Pin Status The BOOTCFG field holds the value of the BOOTCFG[1] pin that was latched on the last negation of the RSTOUT pin. The BOOTCFG field is used by the BAM program to determine the location of the Reset Configuration Word. See *Table 10* in *Section , RCHW overview* for a translation of the Reset Configuration Half Word location from the BOOTCFG field value. 0b00: Boot from internal flash memory (default) 0b01: FlexCAN / eSCI boot 0b10: Boot from external memory (no arbitration) 0b11: Reserved
31	RGF	RESET Glitch Flag This bit is set by the MCU when the RESET pin is asserted for more than 2 clock cycles, but less than the minimum RESET assertion time of 10 consecutive clock cycles to cause a reset. This bit is cleared by the reset controller for a valid assertion of the RESET pin or a power-on reset or a write of one to the bit. 1: A glitch was detected on the RESET pin. 0: No glitch was detected on the RESET pin.

15.6.5 System Reset Control Register (SIU_SRCR)

The System Reset Control Register (SIU_SRCR) allows software to generate either a Software System Reset or Software External Reset. The Software System Reset causes an internal reset sequence, while the Software External Reset only causes the external RSTOUT pin to be asserted for the predetermined number of clock cycles (refer to *Section 4.3.2, RSTOUT*). When written to one, the SER bit automatically clears after the clock count expires. If the value of the SER bit is one and a zero is written to the bit, the bit is cleared and the RSTOUT pin is negated regardless if the clock count has expired.

SIU_BASE + 0xE

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SSR	SER	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W			▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	Res.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W		▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	1(1)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

1. This bit in the SPC564A70 MCU has no effect as checkstop reset is not supported.

Figure 105. System Reset Control Register (SIU_SRCR)

Table 107. SIU_SRCR field description

Field	Description
SSR	Software System Reset Writing a one to this bit causes an internal reset and assertion of the $\overline{\text{RSTOUT}}$ pin. The bit is automatically cleared by all reset sources except the Software External Reset. 1: Generate a Software System Reset. 0: Do not generate a Software System Reset.
SER	Software External Reset Writing a one to this bit causes a Software External Reset. The $\overline{\text{RSTOUT}}$ pin is asserted for the predetermined number of clock cycles (refer to *Section 4.3.2, RSTOUT*), but the MCU is not reset. The bit is automatically cleared when the Software External Reset completes. 1: Generate a Software External Reset. 0: Do not generate a Software External Reset.

15.6.6 External Interrupt Status Register (SIU_EISR)

The External Interrupt Status Register is used to record edge triggered events on the IRQ0 – IRQ15 inputs to the SIU. It also records the critical interrupts NMI and SWT.

SIU_BASE + 0x14

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	NMI	0	0	0	0	0	0	0	SWT	0	0	0	0	0	0	0
W		▨	▨	▨	▨	▨	▨	▨		▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	EIF15	EIF14	EIF13	EIF12	EIF11	EIF10	EIF9	EIF8	EIF7	EIF6	EIF5	EIF4	EIF3	EIF2	EIF1	EIF0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 106. External IRQ Status Register (SIU_EISR)

Table 108. SIU_EISR field description

Field	Description
NMI	Non-Maskable Interrupt Flag This bit is set when a NMI interrupt occurs on the NMI input pin. 1: An NMI event has occurred on the NMI input 0: No NMI event has occurred on the NMI input
SWT	Software Watch Dog Timer Interrupt Flag, from platform This bit is set when a SWT interrupt occurs on the platform. 1: An SWT event has occurred 0: No SWT event has occurred
EIF*x*	External Interrupt Request Flag *x* This bit is set when an edge triggered event occurs on the corresponding IRQ*x* input. 1: An edge triggered event has occurred on the corresponding IRQ*x* input 0: No edge triggered event has occurred on the corresponding IRQ*x* input

15.6.7 DMA/Interrupt Request Enable Register (SIU_DIRER)

The DMA/Interrupt Request Enable Register allows the assertion of a DMA or interrupt request if the corresponding flag bit is set in *Section 15.6.6, External Interrupt Status Register (SIU_EISR)*. The External Interrupt Request Enable bits enable the interrupt or DMA request. There is only one interrupt request from the SIU to the interrupt controller. The EIRE bits allow selection of which External Interrupt Request Flag bits cause assertion of the one interrupt request signal.

SIU_BASE + 0x18

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	NMI_SEL(1)	0	0	0	0	0	0	0	NMI_SEL0(1)	0	0	0	0	0	0	0
W		▨	▨	▨	▨	▨	▨	▨		▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	EIRE15	EIRE14	EIRE13	EIRE12	EIRE11	EIRE10	EIRE9	EIRE8	EIRE7	0	EIRE5	EIRE4	EIRE3	EIRE2	EIRE1	EIRE0
W										▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

1. This bit is cleared only by a reset.

Figure 107. DMA/Interrupt Request Enable Register (SIU_DIRER)

Table 109. SIU_DIRER field description

Field	Description
NMI_SEL	Non-Maskable Interrupt / Critical Interrupt Selection x The SIU generates two specific sources of interrupt to the core. One of them is defined as the critical interrupt (IVOR0 core exception) and the other is defined as the non-maskable interrupt (NMI) (IVOR1 core exception). The NMI_SEL bit selects which exception will be generated by the external NMI pin. This bit is cleared only by a reset. 1: Critical interrupt (IVOR0) is enabled 0: NMI (IVOR1) is enabled
NMI_SEL0	Non-Maskable Interrupt / Critical Interrupt Selection x The SIU generates two specific sources of interrupt to the core. One of them is defined as the critical interrupt (IVOR0 core exception) and the other is defined as the non-maskable interrupt (NMI) (IVOR1 core exception). The NMI_SEL0 bit selects which exception will be generated by the SWT interrupt. This bit is cleared only by a reset. 1: Critical interrupt (IVOR0) is enabled 0: NMI (IVOR1) is enabled
EIRE*x*	External DMA/Interrupt Request Enable *x* This bit enables the assertion of a DMA or the interrupt request from the SIU to the interrupt controller when an edge triggered event occurs on the IRQ*x* inputs. 1: External interrupt request is enabled 0: External interrupt request is disabled

15.6.8 DMA/Interrupt Request Select Register (SIU_DIRSR)

The DMA/Interrupt Request Select Register allows selection between a DMA or interrupt request for events on the IRQ[0:3] inputs.

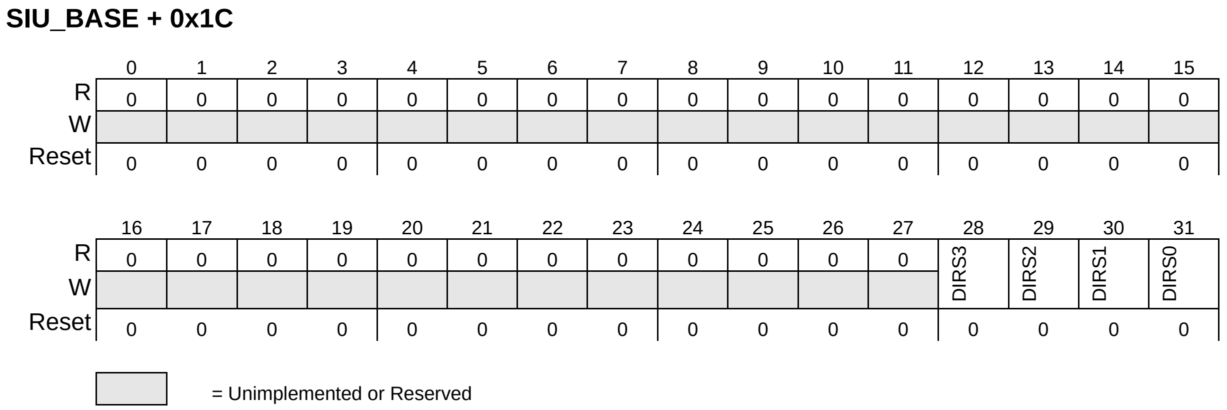


Figure 108. DMA/Interrupt Request Select Register (SIU_DIRSR)

Table 110. SIU_DIRSR field description

Field	Description
DIRS*x*	DMA/Interrupt Request Select *x* This bit selects between a DMA or interrupt request when an edge triggered event occurs on the corresponding IRQ*x* input. 1: DMA request is selected (on this device these DMA connections do not exist, causing the interrupt to be inhibit) 0: Interrupt request is selected

15.6.9 Overrun Status Register (SIU_OSR)

The Overrun Status Register contains flag bits that record an overrun.

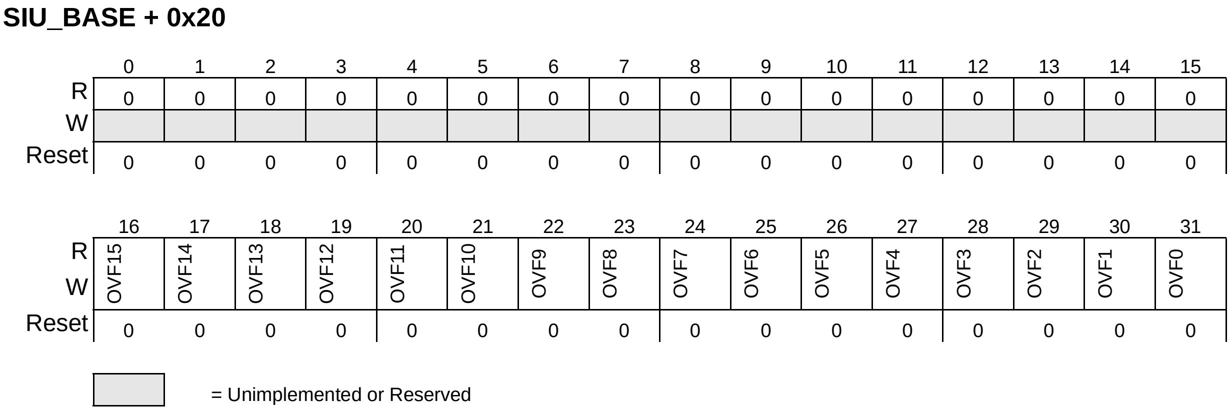


Figure 109. Overrun Status Register (SIU_OSR)

Table 111. SIU_OSR field description

Field	Description
OVF*x*	Overrun Flag *x* This bit is set when an overrun occurs on the corresponding IRQ*x* input. 1: An overrun has occurred on the corresponding IRQ*x* input 0: No overrun has occurred on the corresponding IRQ*x* input

15.6.10 Overrun Request Enable Register (SIU_ORER)

The Overrun Request Enable Register contains bits to enable an overrun if the corresponding flag bit is set in the SIU_OSR. If any Overrun Request Enable bit and the corresponding flag bit is set, the single combined overrun request from the SIU to the interrupt controller is asserted.

SIU_BASE + 0x24

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	ORE15	ORE14	ORE13	ORE12	ORE11	ORE10	ORE9	ORE8	ORE7	ORE6	ORE5	ORE4	ORE3	ORE2	ORE1	ORE0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 110. Overrun Request Enable Register (SIU_ORER)

Table 112. SIU_ORER field description

Field	Description
ORE*x*	Overrun Request Enable *x* This bit enables the corresponding overrun request when an overrun occurs on the corresponding IRQ*x* input. 1: Overrun request is enabled 0: Overrun request is disabled

15.6.11 IRQ Rising-Edge Event Enable Register (SIU_IREER)

The IRQ Rising-Edge Event Enable Register allows rising edge triggered events to be enabled on the corresponding IRQ*x* inputs. Rising and falling edge events can be enabled by setting the corresponding bits in both the SIU_IREER and SIU_IFEER.

SIU_BASE + 0x28

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	NMIRE[(1)]	0	0	0	0	0	0	0	NMIRE0[(1)]	0	0	0	0	0	0	0
W		▨	▨	▨	▨	▨	▨	▨		▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	IREE15	IREE14	IREE13	IREE12	IREE11	IREE10	IREE9	IREE8	IREE7	IREE6	IREE5	IREE4	IREE3	IREE2	IREE1	IREE0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

1. This bit is cleared only by a reset.

Figure 111. IRQ Rising-Edge Event Enable Register (SIU_IREER)

Table 113. SIU_IREER field description

Field	Description
NMIRE	NMI Rising-Edge Event Enable *(NMI Input)* This write once bit enables rising-edge triggered events on the NMI input. This bit is cleared only by a reset. 1: Rising edge event is enabled 0: Rising edge event is disabled
NMIRE0	NMI Rising-Edge Event Enable *(SWT)* This write once bit enables rising-edge triggered events by SWT. This bit is cleared only by a reset. 1: Rising edge event is enabled 0: Rising edge event is disabled
IREE*x*	IRQ Rising-Edge Event Enable *x* This bit enables rising-edge triggered events on the corresponding IRQ*x* input. 1: Rising edge event is enabled 0: Rising edge event is disabled

15.6.12 External IRQ Falling-Edge Event Enable Register (SIU_IFEER)

The External IRQ Falling-Edge Event Enable Register allows falling edge triggered events to be enabled on the corresponding IRQ*x* inputs. Rising and falling edge events can be enabled by setting the corresponding bits in both the SIU_IREER and SIU_IFEER.

SIU_BASE + 0x2C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	NMIFE[1]	0	0	0	0	0	0	0	NMIFE0[1]	0	0	0	0	0	0	0
W		▨	▨	▨	▨	▨	▨	▨		▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	IFEE15	IFEE14	IFEE13	IFEE12	IFEE11	IFEE10	IFEE9	IFEE8	IFEE7	IFEE6	IFEE5	IFEE4	IFEE3	IFEE2	IFEE1	IFEE0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

1. This bit is cleared only by a reset.

Figure 112. External IRQ Falling-Edge Event Enable Register (SIU_IFEER)

Table 114. SIU_IFEER field description

Field	Description
NMIFE	NMI Falling-Edge Event Enable *(NMI Input)* This write once bit enables falling-edge triggered events on the NMI input. This bit is cleared only by a reset. 1: Falling edge event is enabled 0: Falling edge event is disabled
NMIFE0	NMI Falling-Edge Event Enable *(SWT)* This write once bit enables falling-edge triggered events by SWT. This bit is cleared only by a reset. 1: Falling edge event is enabled 0: Falling edge event is disabled
IFEE*x*	IRQ Falling-Edge Event Enable *x* This bit enables falling-edge triggered events on the corresponding IRQ*x* input. 1: Falling edge event is enabled 0: Falling edge event is disabled

15.6.13 External IRQ Digital Filter Register (SIU_IDFR)

The External IRQ Digital Filter Register specifies the amount of digital filtering on the IRQ0 – IRQ15 inputs. The Digital Filter Length field specifies the number of system clocks that define the period of the digital filter.

SIU_BASE + 0x30

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	DFL[0–3]			
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨				
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 113. IRQ Digital Filter Register (SIU_IDFR)

Table 115. SIU_IDFR field description

Field	Description
DFL[0–3]	Digital Filter Length This field defines the digital filter period on the IRQ*x* inputs according to *Equation 1*: **Equation 1** $\text{Filter Period} = (\text{SystemClockPeriod} \times 2^{\text{DFL}}) + 1(\text{SystemClockPeriod})$ For a 100 MHz system clock, this gives a range of 20 ns to 328 µs. The minimum time of two clocks accounts for synchronization of the IRQ input pins with the system clock. Using the same calculation, for a 150 MHz system clock, this gives a range of 13.3 ns to 218 µs.

15.6.14 IRQ Filtered Input Register (SIU_IFIR)

The SIU_IFIR is a read-only register used to capture the filtered values of the IRQ0–31 pins. This feature is enabled with a parameter at the top level of the module.

The MSB positions of the register correspond to NMI pins and the number of NMI pins are defined by a parameter.

The LSB positions of the register corresponds to the IRQ pins and the number of IRQ pins is defined by a parameter.

SIU_BASE + 0x2C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	NMIFE[1]	0	0	0	0	0	0	0	NMIFE0[1]	0	0	0	0	0	0	0
W		▨	▨	▨	▨	▨	▨	▨		▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	IFEE15	IFEE14	IFEE13	IFEE12	IFEE11	IFEE10	IFEE9	IFEE8	IFEE7	IFEE6	IFEE5	IFEE4	IFEE3	IFEE2	IFEE1	IFEE0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

1. This bit is cleared only by a reset.

Figure 114. IRQ Filtered Input Register (SIU_IFIR)

15.6.15 Pad Configuration Registers (SIU_PCR)

The Pad Configuration Registers (PCRs) select function, I/O direction and some electrical characteristics for configurable device pins. Not all device pins are configurable.

PCRs are 16-bit registers but may be read or written as 32-bit values aligned on 32-bit address boundaries. They are based on a common set of fields, but only the pertinent fields appear in each register.

The information in the following sections pertains to the bits and fields that are active for a given pin or group of pins, and the reset state of the register. The reset state given for each PCR is the state prior to execution of the BAM program. The BAM program may change certain PCRs based on the reset configuration. See *20: Boot Assist Module (BAM)* for more details.

The device is available in the packages listed in *Chapter 1: Introduction*. Some of the I/O functions controlled by the SIU PCRs are not available in the smaller packages. The port enable logic for these PCRs is the same for PCRs that control I/O functions that are available in all packages. For the smaller packages where some of the I/O functions are not available, the pad drivers are disabled in the pad interface logic. The user should take care not to select the unavailable functions via the PA field. See *Section 3.1, Signal Properties*, for a definition of which I/O functions are available in each package.

Table 116 lists and describes the fields contained in the PCRs. Not all fields appear in each PCR but each field has an identical function in each register where it resides.

Table 116. SIU_PCR field description

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>—</td><td>Reserved fields are indicated by shading in the register maps.</td></tr>
<tr><td>PA</td><td>Pin assignment
Selects the function of a multiplexed pad.
<table>
<tr><th>PA value(1)</th><th colspan="2">Pin function</th></tr>
<tr><td>0b0001</td><td>P</td><td>Primary function</td></tr>
<tr><td>0b0010</td><td>A1</td><td>Alternate function 1</td></tr>
<tr><td>0b0100</td><td>A2</td><td>Alternate function 2</td></tr>
<tr><td>0b1000</td><td>A3</td><td>Alternate function 3</td></tr>
<tr><td>0b0000</td><td>G</td><td>GPIO</td></tr>
</table>
1. Depending on the register, the PA field size can vary in length. For PA fields having fewer than four bits, remove the appropriate number of leading zeroes from these values.</td></tr>
<tr><td>OBE(1),(2)</td><td>Output buffer enable
Enables the pad as an output and drives the output buffer enable signal.

0 Disable output buffer for the pad.
1 Enable output buffer for the pad is enabled.</td></tr>
<tr><td>IBE(1),(2)</td><td>Input buffer enable
Enables the pad as an input and drives the input buffer enable signal.

0 Disable input buffer for the pad.
1 Enable input buffer for the pad is enabled.

For all PCRs where GPIO function is available on the pin, if the pin is configured as an output and the IBE bit is set, the actual value of the pin will be reflected in the corresponding GPDI*x*_*x* register. Negating the IBE bit when the pin is configured as an output will reduce noise and power consumption.</td></tr>
<tr><td>DSC(3)</td><td>Drive strength control
Controls the pad drive strength. Drive strength control pertains to pins with the fast I/O pad type.

00 10 pF drive strength
01 20 pF drive strength
10 30 pF drive strength
11 50 pF drive strength</td></tr>
<tr><td>ODE(3)</td><td>Open drain output enable
Controls output driver configuration for the pads. Either open drain or push/pull driver configurations can be selected. This feature applies to output pins only.

0 Disable open drain for the pad (push/pull driver enabled).
1 Enable open drain for the pad.</td></tr>
</table>

Table 116. SIU_PCR field description (continued)

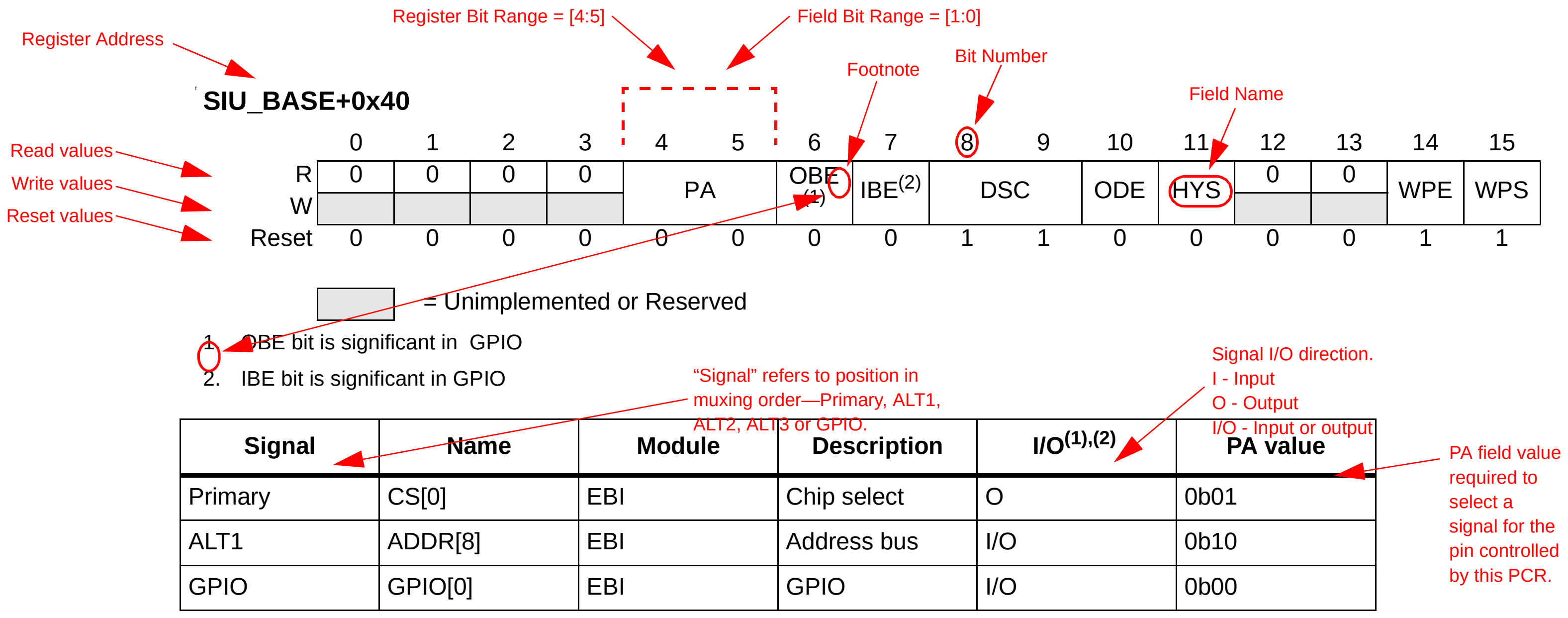
Field	Description
HYS[4]	Input hysteresis Controls whether hysteresis is enabled for the pad. 0 Disable hysteresis for the pad. 1 Enable hysteresis for the pad.
SRC[3]	Slew rate control Controls slew rate for the pad. Slew rate control pertains to pins with slow or medium I/O pad types, and the output signals are driven according to the value of this field. Actual slew rate depends on the pad type and load. Refer to the electrical specifications for this information. 00 Minimum slew rate 01 Medium slew rate 10 Invalid value 11 Maximum slew rate
WPE[5]	Weak pullup/down enable Controls whether the weak pullup/down devices are enabled/disabled for the pad. Pullup/down devices are enabled by default. 0 Disable weak pull device for the pad. 1 Enable weak pull device for the pad.
WPS[5]	Weak pullup/down select Controls whether weak pullup or weak pulldown devices are used for the pad when weak pullup/down devices are enabled. The WKPCFG pin determines whether pullup or pulldown devices are enabled during reset. The WPS bit determines whether weak pullup or pulldown devices are used after reset, or for pads in which the WKPCFG pin does not determine the reset weak pullup/down state. 0 Pulldown is enabled for the pad. 1 Pullup is enabled for the pad.

1. In cases where an I/O function is either input-only or output-only the IBE and OBE bits do not need to be set to enable pin I/O.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.
3. If a pin is configured as an input, the ODE, SRC, and DSC bits do not apply.
4. If a pin is configured as an output, the HYS bit does not apply.
5. When a pin is configured as an output, the weak internal pull up/down is disabled regardless of the WPE or WPS settings in the PCR.

The following sections describe PCR functions using maps that show the fields contained in each register. Refer to *Table 116* for the details of each field.

Figure 115 shows a sample PCR map. Please note the following:

- The register bit numbering order follows the Power Architecture standard of the most significant bit being bit 0. Field bit ranges are the opposite—the least significant bit is referred to as bit 0.
- Bit 0 is an example of a reserved field. It is read-only and always returns a value of 0.



1. OBE bit is significant in GPIO
2. IBE bit is significant in GPIO

Signal	Name	Module	Description	I/O[(1),(2)]	PA value
Primary	CS[0]	EBI	Chip select	O	0b01
ALT1	ADDR[8]	EBI	Address bus	I/O	0b10
GPIO	GPIO[0]	EBI	GPIO	I/O	0b00

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Figure 115. Sample PCR map

Pad Configuration Register 12 (SIU_PCR12)

SIU_BASE+0x58

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA			OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨						▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

▨ = Unimplemented or Reserved

1. When configured as GPO, the OBE bit should be set to one.
2. When configured as FR_A_TX or GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. When configured as GPI, the IBE bit should be set to one.

Figure 116. Pad Configuration Register (SIU_PCR12)

Table 117. SIU_PCR12 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
ALT1	FR_A_TX	EBI	FlexRay transmit	O	0b010
GPIO	GPIO[12]	SIU	GPIO	I/O	0b000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 13 (SIU_PCR13)

SIU_BASE+0x5A

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA			OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨						▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

▨ = Unimplemented or Reserved

1. When configured as GPO, the OBE bit should be set to one.
2. When configured as FR_A_TX_EN or GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. When configured as GPI, the IBE bit should be set to one.

Figure 117. Pad Configuration Register (SIU_PCR13)

Table 118. SIU_PCR13 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
ALT1	FR_A_TX_EN	EBI	FlexRay transmit enable	O	0b010
GPIO	GPIO[13]	SIU	GPIO	I/O	0b000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 14 (SIU_PCR14)

SIU_BASE+0x5C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA			OBE[1]	IBE[2]	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨						▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

▨ = Unimplemented or Reserved

1. When configured as FR_A_RX the OBE bit has no effect. When configured as GPO, the OBE bit should be set to one.
2. When configured as GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. When configured as GPI, the IBE bit should be set to one.

Figure 118. Pad Configuration Register (SIU_PCR14)

Table 119. SIU_PCR14 PA values

Signal	Name	Module	Description	I/O[1],[2]	PA value
ALT1	FR_A_RX	FlexRay	FlexRay receive	I	0b010
GPIO	GPIO[14]	SIU	GPIO	I/O	0b000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 15 (SIU_PCR15)

SIU_BASE+0x5E

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA			OBE[1]	IBE[2]	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨						▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

▨ = Unimplemented or Reserved

1. When configured as GPO, the OBE bit should be set to one.
2. When configured as FR_B_TX or GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. When configured as GPI, the IBE bit should be set to one.

Figure 119. Pad Configuration Register (SIU_PCR15)

Table 120. SIU_PCR15 PA values

Signal	Name	Module	Description	I/O[1],[2]	PA value
ALT1	FR_B_TX	FlexRay	FlexRay transmit	O	0b010
GPIO	GPIO[15]	SIU	GPIO	I/O	0b000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 16 (SIU_PCR16)

SIU_BASE+0x60

<table>
<tr><th></th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th></tr>
<tr><td>R</td><td>0</td><td>0</td><td>0</td><td colspan="3" rowspan="2">PA</td><td rowspan="2">OBE
(1)</td><td rowspan="2">IBE⁽²⁾</td><td>0</td><td>0</td><td rowspan="2">ODE</td><td rowspan="2">HYS</td><td colspan="2" rowspan="2">SRC</td><td rowspan="2">WPE</td><td rowspan="2">WPS</td></tr>
<tr><td>W</td><td>▨</td><td>▨</td><td>▨</td><td>▨</td><td>▨</td></tr>
<tr><td>Reset</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td></tr>
</table>

▨ = Unimplemented or Reserved

1. When configured as GPO, the OBE bit should be set to one.
2. When configured as FR_B_TX_EN or GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. When configured as GPI, the IBE bit should be set to one.

Figure 120. Pad Configuration Register (SIU_PCR16)

Table 121. SIU_PCR16 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
ALT1	FR_B_TX_EN	FlexRay	FlexRay transmit enable	O	0b010
GPIO	GPIO[16]	SIU	GPIO	I/O	0b000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 17 (SIU_PCR17)

SIU_BASE+0x62

<table>
<tr><th></th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th></tr>
<tr><td>R</td><td>0</td><td>0</td><td>0</td><td colspan="3" rowspan="2">PA</td><td rowspan="2">OBE
(1)</td><td rowspan="2">IBE⁽²⁾</td><td>0</td><td>0</td><td rowspan="2">ODE</td><td rowspan="2">HYS</td><td colspan="2" rowspan="2">SRC</td><td rowspan="2">WPE</td><td rowspan="2">WPS</td></tr>
<tr><td>W</td><td>▨</td><td>▨</td><td>▨</td><td>▨</td><td>▨</td></tr>
<tr><td>Reset</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td></tr>
</table>

▨ = Unimplemented or Reserved

1. When configured as FR_B_RX the OBE bit has no effect. When configured as GPO, the OBE bit should be set to one.
2. When configured as GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. When configured as GPI, the IBE bit should be set to one.

Figure 121. Pad Configuration Register (SIU_PCR17)

Table 122. SIU_PCR17 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
ALT1	FR_B_RX	FlexRay	FlexRay receive	I	0b010
GPIO	GPIO[17]	SIU	GPIO	I/O	0b000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Registers 75–82 (SIU_PCR75–SIU_PCR82)

The SIU_PCR75–SIU_PCR82 registers control the pin function, direction, and static electrical attributes of the MDO[4:11]_GPIO[75:82] pins. GPIO is the default function at reset for these pins.

Note: The full port mode (FPM) and NEXCFG bits in the Nexus port controller (NPC) port configuration register control whether these pins function as MDO[4:11] or GPIO[75:82]. When the FPM and NEXCFG bits are set, the NPC enables the MDO port enable, and disables GPIO. When the FPM or NEXCFG bit is cleared, the NPC disables the MDO port enable, and enables GPIO.

Pad Configuration Register 75 (SIU_PCR75)

SIU_BASE+0xD6

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE (1)	IBE (1)	0	0	ODE (2)	HYS (3)	SRC		WPE (4)	WPS
W	▨	▨	▨	▨					▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

1. This bit applies only to GPIO operation.
2. The ODE bit should be set to zero for MDO operation.
3. The HYS bit has no effect on MDO operation.
4. The WPE bit should be set to zero for MDO operation.

Figure 122. Pad Configuration Register (SIU_PCR75)

Table 123. SIU_PCR75 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	MDO[4]	Nexus	Message data out	O	0b001
ALT1	ETPU_A[2]	eTPU	eTPU channel	O	0b010
GPIO	GPIO[75]	SIU	GPIO	I/O	0b000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 76 (SIU_PCR76)

SIU_BASE+0xD8

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE (1)	IBE (1)	0	0	ODE (2)	HYS (3)	SRC		WPE (4)	WPS
W	▨	▨	▨	▨					▨	▨						
Reset	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	1

▨ = Unimplemented or Reserved

1. This bit applies only to GPIO operation.

2. The ODE bit should be set to zero for MDO operation.
3. The HYS bit has no effect on MDO operation.
4. The WPE bit should be set to zero for MDO operation.

Figure 123. Pad Configuration Register (SIU_PCR76)

Table 124. SIU_PCR76 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	MDO[5]	Nexus	Message data out	O	0b001
ALT1	ETPU_A[4]	eTPU	eTPU channel	O	0b010
GPIO	GPIO[76]	SIU	GPIO	I/O	0b000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 77 (SIU_PCR77)

SIU_BASE+0xDA

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE(1)	IBE(1)	0	0	ODE(2)	HYS(3)	SRC		WPE(4)	WPS
W	▨	▨	▨	▨					▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

1. This bit applies only to GPIO operation.
2. The ODE bit should be set to zero for MDO operation.
3. The HYS bit has no effect on MDO operation.
4. The WPE bit should be set to zero for MDO operation.

Figure 124. Pad Configuration Register (SIU_PCR77)

Table 125. SIU_PCR77 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	MDO[6]	Nexus	Message data out	O	0b001
ALT1	ETPU_A[13]	eTPU	eTPU channel	O	0b010
GPIO	GPIO[77]	SIU	GPIO	I/O	0b000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 78 (SIU_PCR78)

SIU_BASE+0xDC

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE [1]	IBE [1]	0	0	ODE [2]	HYS [3]	SRC		WPE [4]	WPS
W	▨	▨	▨	▨					▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

1. This bit applies only to GPIO operation.
2. The ODE bit should be set to zero for MDO operation.
3. The HYS bit has no effect on MDO operation.
4. The WPE bit should be set to zero for MDO operation.

Figure 125. Pad Configuration Register (SIU_PCR78)

Table 126. SIU_PCR78 PA values

Signal	Name	Module	Description	I/O[1],[2]	PA value
Primary	MDO[7]	Nexus	Message data out	O	0b001
ALT1	ETPU_A[19]	eTPU	eTPU channel	O	0b010
GPIO	GPIO[78]	SIU	GPIO	I/O	0b000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 79 (SIU_PCR79)

SIU_BASE+0xDE

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE [1]	IBE [1]	0	0	ODE [2]	HYS [3]	SRC		WPE [4]	WPS
W	▨	▨	▨	▨					▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

1. This bit applies only to GPIO operation.
2. The ODE bit should be set to zero for MDO operation.
3. The HYS bit has no effect on MDO operation.
4. The WPE bit should be set to zero for MDO operation.

Figure 126. Pad Configuration Register (SIU_PCR79)

Table 127. SIU_PCR79 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	MDO[8]	Nexus	Message data out	O	0b001
ALT1	ETPU_A[21]	eTPU	eTPU channel	O	0b010
GPIO	GPIO[79]	SIU	GPIO	I/O	0b000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 80 (SIU_PCR80)

SIU_BASE+0xE0

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE(1)	IBE(1)	0	0	ODE(2)	HYS(3)	SRC		WPE(4)	WPS
W	▨	▨	▨	▨					▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

1. This bit applies only to GPIO operation.
2. The ODE bit should be set to zero for MDO operation.
3. The HYS bit has no effect on MDO operation.
4. The WPE bit should be set to zero for MDO operation.

Figure 127. Pad Configuration Register (SIU_PCR80)

Table 128. SIU_PCR80 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	MDO[9]	Nexus	Message data out	O	0b001
ALT1	ETPU_A[25]	eTPU	eTPU channel	O	0b010
GPIO	GPIO[80]	SIU	GPIO	I/O	0b000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 81 (SIU_PCR81)

SIU_BASE+0xE2

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE[1]	IBE[1]	0	0	ODE[2]	HYS[3]	SRC		WPE[4]	WPS
W	▨	▨	▨	▨					▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

1. This bit applies only to GPIO operation.
2. The ODE bit should be set to zero for MDO operation.
3. The HYS bit has no effect on MDO operation.
4. The WPE bit should be set to zero for MDO operation.

Figure 128. Pad Configuration Register (SIU_PCR81)

Table 129. SIU_PCR81 PA values

Signal	Name	Module	Description	I/O[1],[2]	PA value
Primary	MDO[10]	Nexus	Message data out	O	0b001
ALT1	ETPU_A[27]	eTPU	eTPU channel	O	0b010
GPIO	GPIO[81]	SIU	GPIO	I/O	0b000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 82 (SIU_PCR82)

SIU_BASE+0xE4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE[1]	IBE[1]	0	0	ODE[2]	HYS[3]	SRC		WPE[4]	WPS
W	▨	▨	▨	▨					▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

1. This bit applies only to GPIO operation.
2. The ODE bit should be set to zero for MDO operation.
3. The HYS bit has no effect on MDO operation.
4. The WPE bit should be set to zero for MDO operation.

Figure 129. Pad Configuration Register (SIU_PCR82)

Table 130. SIU_PCR82 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	MDO[11]	Nexus	Message data out	O	0b001
ALT1	ETPU_A[29]	eTPU	eTPU channel	O	0b010
GPIO	GPIO[82]	SIU	GPIO	I/O	0b000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 83 (SIU_PCR83)

SIU_BASE+0xE6

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨	▨					▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

▨ = Unimplemented or Reserved

1. When configured as GPO, the OBE bit should be set to one.
2. When configured as CAN_A_TX or GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. When configured as SCI_A_TX both OBE and IBE are set to one automatically. Setting the IBE bit to zero reduces power consumption. When configured as GPI, the IBE bit should be set to one.

Figure 130. Pad Configuration Register (SIU_PCR83)

Table 131. SIU_PCR83 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	CAN_A_TX	FlexCAN	FlexCAN transmit	O	0b01
ALT1	SCI_A_TX	eSCI	eSCI transmit	O	0b10
GPIO	GPIO[83]	SIU	GPIO	I/O	0b00

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 84 (SIU_PCR84)

SIU_BASE+0xE8

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE[1]	IBE[2]	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨	▨					▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

▨ = Unimplemented or Reserved

1. When configured as CAN_A_RX or SCI_A_RX, the OBE bit has no effect. When configured as GPO, the OBE bit should be set to one.
2. When configured as GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. When configured as GPI, the IBE bit should be set to one.

Figure 131. Pad Configuration Register (SIU_PCR84)

Table 132. SIU_PCR84 PA values

Signal	Name	Module	Description	I/O[(1),(2)]	PA value
Primary	CAN_A_RX	FlexCAN	FlexCAN receive	I	0b01
ALT1	SCI_A_RX	eSCI	eSCI receive	I	0b10
GPIO	GPIO[84]	SIU	GPIO	I/O	0b00

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 85 (SIU_PCR85)

SIU_BASE+0xEA

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA			OBE[1]	IBE[2]	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨						▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

▨ = Unimplemented or Reserved

1. When configured as GPO, the OBE bit should be set to one.
2. When configured as CAN_B_TX, DSPI_C_PCS[3], SCI_C_TX or GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. When configured as GPI, the IBE bit should be set to one.

Figure 132. Pad Configuration Register (SIU_PCR85)

Table 133. SIU_PCR85 PA values

Signal	Name	Module	Description	I/O[(1),(2)]	PA value
Primary	CAN_B_TX	FlexCAN	FlexCAN transmit	O	0b001
ALT1	DSPI_C_PCS[3]	DSPI	Chip select	O	0b010
ALT2	SCI_C_TX	eSCI	eSCI transmit	O	0b100
GPIO	GPIO[85]	SIU	GPIO	I/O	0b000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.

2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 86 (SIU_PCR86)

SIU_BASE+0xEC

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA			OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨						▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

▨ = Unimplemented or Reserved

1. When configured as CAN_B_RX or SCI_C_RX, the OBE bit has no effect. When configured as GPO, the OBE bit should be set to one.
2. When configured as DSPI_C_PCS[4] or GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. When configured as GPI, the IBE bit should be set to one.

Figure 133. Pad Configuration Register (SIU_PCR86)

Table 134. SIU_PCR86 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	CAN_B_RX	FlexCAN	FlexCAN receive	I	0b001
ALT1	DSPI_C_PCS[4]	DSPI	Chip select	O	0b010
ALT2	SCI_C_RX	eSCI	eSCI receive	I	0b100
GPIO	GPIO[86]	SIU	GPIO	I/O	0b000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 87 (SIU_PCR87)

SIU_BASE+0xEE

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨	▨					▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

▨ = Unimplemented or Reserved

1. When configured as GPO, the OBE bit should be set to one.
2. When configured as CAN_C_TX, DSPI_D_PCS[3] or GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. When configured as GPI, the IBE bit should be set to one.

Figure 134. Pad Configuration Register (SIU_PCR87)

Table 135. SIU_PCR87 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	CAN_C_TX	FlexCAN	FlexCAN transmit	O	0b01
ALT1	DSPI_D_PCS[3]	DSPI	Chip select	O	0b10
GPIO	GPIO[87]	SIU	GPIO	I/O	0b00

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 88 (SIU_PCR88)

SIU_BASE+0xF0

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS
W	▒	▒	▒	▒					▒	▒						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

▒ = Unimplemented or Reserved

1. When configured as CAN_C_RX, the OBE bit has no effect. When configured as GPO, the OBE bit should be set to one.
2. When configured as CAN_C_RX, DSPI_D_PCS[4] or GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. When configured as GPI, the IBE bit should be set to one.

Figure 135. Pad Configuration Register (SIU_PCR88)

Table 136. SIU_PCR88 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	CAN_C_RX	FlexCAN	FlexCAN receive	I	0b01
ALT1	DSPI_D_PCS[4]	DSPI	Chip select	O	0b10
GPIO	GPIO[88]	SIU	GPIO	I/O	0b00

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 89 (SIU_PCR89)

SIU_BASE+0xF2

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨	▨					▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

▨ = Unimplemented or Reserved

1. When configured as GPO, the OBE bit should be set to one.
2. When configured as SCI_A_TX, EMIOS[13] or GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. For SCI loop back operation the IBE bit must be set to one. Setting the IBE bit to zero reduces power consumption. When configured as GPI, the IBE bit should be set to one.

Figure 136. Pad Configuration Register (SIU_PCR89)

Table 137. SIU_PCR89 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	SCI_A_TX	eSCI	eSCI transmit	O	0b01
ALT1	EMIOS[13]	eMIOS	eMIOS channel	O	0b10
GPIO	GPIO[89]	SIU	GPIO	I/O	0b00

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 90 (SIU_PCR90)

SIU_BASE+0xF4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨	▨					▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

▨ = Unimplemented or Reserved

1. When configured as SCI_A_RX the OBE bit has no effect. When configured as GPO, the OBE bit should be set to one.
2. When configured as EMIOS[15] or GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. For SCI loop back operation the IBE bit must be set to one. Setting the IBE bit to zero reduces power consumption. When configured as GPI, the IBE bit should be set to one.

Figure 137. Pad Configuration Register (SIU_PCR90)

Table 138. SIU_PCR90 PA values

Signal	Name	Module	Description	I/O[(1),(2)]	PA value
Primary	SCI_A_RX	eSCI	eSCI receive	I	0b01
ALT1	EMIOS[15]	eMIOS	eMIOS channel	O	0b10
GPIO	GPIO[90]	SIU	GPIO	I/O	0b00

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 91 (SIU_PCR91)

SIU_BASE+0xF6

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE (1)	IBE (2)	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨	▨					▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

▨ = Unimplemented or Reserved

1. When configured as GPO, the OBE bit should be set to one.
2. When configured as SCI_B_TX, DSPI_D_PCS[1] or GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. For SCI loop back operation the IBE bit must be set to one. Setting the IBE bit to zero reduces power consumption. When configured as GPI, the IBE bit should be set to one.

Figure 138. Pad Configuration Register (SIU_PCR91)

Table 139. SIU_PCR91 PA values

Signal	Name	Module	Description	I/O[(1),(2)]	PA value
Primary	SCI_B_TX	eSCI	eSCI transmit	O	0b01
ALT1	DSPI_D_PCS[1]	DSPI	Chip select	O	0b10
GPIO	GPIO[91]	SIU	GPIO	I/O	0b00

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 92 (SIU_PCR92)

SIU_BASE+0xF8

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE (1)	IBE (2)	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨	▨					▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

▨ = Unimplemented or Reserved

1. When configured as SCI_B_RX, the OBE bit has no effect. When configured as GPO, the OBE bit should be set to one.

2. When configured as DSPI_D_PCS[5] or GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. For SCI loop back operation the IBE bit must be set to one. Setting the IBE bit to zero reduces power consumption. When configured as GPI, the IBE bit should be set to one.

Figure 139. Pad Configuration Register (SIU_PCR92)

Table 140. SIU_PCR92 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	SCI_B_RX	eSCI	eSCI receive	I	0b01
ALT1	DSPI_D_PCS[5]	DSPI	Chip select	O	0b10
GPIO	GPIO[92]	SIU	GPIO	I/O	0b00

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 93 (SIU_PCR93)

SIU_BASE+0xFA

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA(1)		OBE (2)	IBE(3)	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨	▨					▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

▨ = Unimplemented or Reserved

1. The SCKA function is not available on the SPC564A70. Do not select 0b01 or 0b11 for the PA field.
2. When configured as GPO, the OBE bit should be set to one.
3. When configured as DSPI_C_PCS[1] or GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. When configured as GPI, the IBE bit should be set to one.

Figure 140. Pad Configuration Register (SIU_PCR93)

Table 141. SIU_PCR93 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
SCKA(3)					
ALT1	DSPI_C_PCS[1]	DSPI	Chip select	O	0b10
GPIO	GPIO[93]	SIU	GPIO	I/O	0b00

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.
3. This signal name is used to support legacy naming.

Pad Configuration Register 94 (SIU_PCR94)

SIU_BASE+0xFC

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[1]		OBE[2]	IBE[3]	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨	▨					▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

▨ = Unimplemented or Reserved

1. The SINA function is not available on the SPC564A70. Do not select 0b01 or 0b11 for the PA field.
2. When configured as GPO, the OBE bit should be set to one.
3. When configured as DSPI_C_PCS[2] or GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. When configured as GPI, the IBE bit should be set to one.

Figure 141. Pad Configuration Register (SIU_PCR94)

Table 142. SIU_PCR94 PA values

Signal	Name	Module	Description	I/O[1],[2]	PA value
SINA[3]					
ALT1	DSPI_C_PCS[2]	DSPI	Chip select	O	0b10
GPIO	GPIO[94]	SIU	GPIO	I/O	0b00

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.
3. This signal name is used to support legacy naming.

Pad Configuration Register 95 (SIU_PCR95)

SIU_BASE+0xFE

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[1]		OBE[2]	IBE[3]	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨	▨					▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

▨ = Unimplemented or Reserved

1. The SOUTA function is not available on the SPC564A70. Do not select 0b01 or 0b11 for the PA field.
2. When configured as GPO, the OBE bit should be set to one.
3. When configured as DSPI_C_PCS[5] or GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. When configured as GPI, the IBE bit should be set to one.

Figure 142. Pad Configuration Register (SIU_PCR95)

Table 143. SIU_PCR95 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
SOUTA(3)					
ALT1	DSPI_C_PCS[5]	DSPI	Chip select	O	0b10
GPIO	GPIO[95]	SIU	GPIO	I/O	0b00

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.
3. This signal name is used to support legacy naming.

Pad Configuration Register 96 (SIU_PCR96)

SIU_BASE+0x100

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA(1)		OBE(2)	IBE(3)	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨	▨					▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

▨ = Unimplemented or Reserved

1. The PCSA[0] function is not available on the SPC564A70. Do not select 0b01 or 0b11 for the PA field.
2. When configured as GPO, the OBE bit should be set to one.
3. When configured as DSPI_D_PCS[2] or GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. When configured as GPI, the IBE bit should be set to one.

Figure 143. Pad Configuration Register (SIU_PCR96)

Table 144. SIU_PCR96 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
PCSA0(3)					
ALT1	DSPI_D_PCS[2]	DSPI	Chip select	O	0b10
GPIO	GPIO[96]	SIU	GPIO	I/O	0b00

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.
3. This signal name is used to support legacy naming.

Pad Configuration Register 97 (SIU_PCR97)

SIU_BASE+0x102

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA(1)		OBE(2)	IBE(3)	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨	▨					▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

▨ = Unimplemented or Reserved

1. The PCSA[1] function is not available on the SPC564A70 MCU. Do not select 0b01 or 0b11 for the PA field.
2. When configured as GPO, the OBE bit should be set to one.
3. When configured as DSPI_B_PCS[2] or GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. When configured as GPI, the IBE bit should be set to one.

Figure 144. Pad Configuration Register (SIU_PCR97)

Table 145. SIU_PCR97 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
PCSA1(3)					
ALT1	DSPI_B_PCS[2]	DSPI	Chip select	O	0b10
GPIO	GPIO[97]	SIU	GPIO	I/O	0b00

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.
3. This signal name is used to support legacy naming.

Pad Configuration Register 98 (SIU_PCR98)

SIU_BASE+0x104

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA(1)		OBE(2)	IBE(3)	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨	▨					▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

▨ = Unimplemented or Reserved

1. The PCSA[2] function is not available on the SPC564A70. Do not select 0b01 or 0b11 for the PA field.
2. When configured as DSPI_D_SCK, the OBE bit should be set to one for master operation, and set to zero for slave operation. When configured as GPO, the OBE bit should be set to one.
3. When configured as DSPI_D_SCK in slave operation, the IBE bit should be set to one. When configured as DSPI_D_SCK in master operation or GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. When configured as GPI, the IBE bit should be set to one.

Figure 145. Pad Configuration Register (SIU_PCR98)

Table 146. SIU_PCR98 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
ALT1	DSPI_D_SCK	DSPI	Clock	I/O	0b10
GPIO	GPIO[98]	SIU	GPIO	I/O	0b00

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 99 (SIU_PCR99)

SIU_BASE+0x106

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA(1)		OBE(2)	IBE(3)	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨	▨					▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

▨ = Unimplemented or Reserved

1. The PCSA[3] function is not available on the SPC564A70. Do not select 0b01 or 0b11 for the PA field.
2. When configured as GPO, the OBE bit should be set to one.
3. When configured as GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. When configured as GPI, the IBE bit should be set to one.

Figure 146. Pad Configuration Register (SIU_PCR99)

Table 147. SIU_PCR99 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
ALT1	DSPI_D_SIN	DSPI	Input	I	0b10
GPIO	GPIO[99]	SIU	GPIO	I/O	0b00

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 100 (SIU_PCR100)

SIU_BASE+0x108

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA(1)		OBE(2)	IBE(3)	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨	▨					▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

▨ = Unimplemented or Reserved

1. The PCSA[4] function is not available on the SPC564A70. Do not select 0b01 or 0b11 for the PA field.
2. When configured as DSPI_D_SOUT, the OBE bit has no effect. When configured as GPO, the OBE bit should be set to one.

3. When configured as DSPI_D_SOUT or GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. When configured as GPI, the IBE bit should be set to one.

Figure 147. Pad Configuration Register (SIU_PCR100)

Table 148. SIU_PCR100 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
PCSA4(3)					
ALT1	DSPI_D_SOUT	DSPI	Output	O	0b10
GPIO	GPIO[100]	SIU	GPIO	I/O	0b00

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.
3. This signal name is used to support legacy naming.

Pad Configuration Register 101 (SIU_PCR101)

SIU_BASE+0x10A

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA(1)		OBE (2)	IBE(3)	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨	▨					▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

▨ = Unimplemented or Reserved

1. The PCSA[5] function is not available on the SPC564A70. Do not select 0b01 or 0b11 for the PA field.
2. When configured as GPO, the OBE bit should be set to one.
3. When configured as DSPI_B_PCS[3] or GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. When configured as GPI, the IBE bit should be set to one.

Figure 148. Pad Configuration Register (SIU_PCR101)

Table 149. SIU_PCR101 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
PCSA5(3)					
ALT1	DSPI_B_PCS[3]	DSPI	Chip select	O	0b10
GPIO	GPIO[101]	SIU	GPIO	I/O	0b00

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.
3. This signal name is used to support legacy naming.

Pad Configuration Register 102 (SIU_PCR102)

SIU_BASE+0x10C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE[1]	IBE[2]	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨	▨					▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

▨ = Unimplemented or Reserved

1. When configured as DSPI_B_SCK, the OBE bit should be set to one for master operation, and set to zero for slave operation. When configured as GPO, the OBE bit should be set to one.
2. When configured as DSPI_B_SCK in slave operation the IBE bit should be set to one. When configured as DSPI_B_SCK in master operation or GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. When configured as GPI, the IBE bit should be set to one.

Figure 149. Pad Configuration Register (SIU_PCR102)

Table 150. SIU_PCR102 PA values

Signal	Name	Module	Description	I/O[1],[2]	PA value
Primary	DSPI_B_SCK	DSPI	Clock	I/O	0b01
ALT1	DSPI_C_PCS[1]	DSPI	Chip select	O	0b10
GPIO	GPIO[102]	SIU	GPIO	I/O	0b00

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 103 (SIU_PCR103)

SIU_BASE+0x10E

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE[1]	IBE[2]	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨	▨					▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

▨ = Unimplemented or Reserved

1. When configured as DSPI_B_SIN, the OBE bit should be set to zero. When configured as PCS, the OBE bit should be set to one.
2. When configured as DSPI_B_SIN or DSPI_C_PCS[2], the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. When configured as GPI, the IBE bit should be set to one.

Figure 150. Pad Configuration Register (SIU_PCR103)

Table 151. SIU_PCR103 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	DSPI_B_SIN	DSPI	Input	I	0b01
ALT1	DSPI_C_PCS[2]	DSPI	Chip select	O	0b10
GPIO	GPIO[103]	SIU	GPIO	I/O	0b00

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 104 (SIU_PCR104)

SIU_BASE+0x110

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨	▨					▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

▨ = Unimplemented or Reserved

1. When configured as GPO, the OBE bit should be set to one.
2. When configured as DSPI_B_SOUT or DSPI_C_PCS[5] or GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. When configured as GPI, the IBE bit should be set to one.

Figure 151. Pad Configuration Register (SIU_PCR104)

Table 152. SIU_PCR104 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	DSPI_B_SOUT	DSPI	Output	O	0b01
ALT1	DSPI_C_PCS[5]	DSPI	Chip select	O	0b10
GPIO	GPIO[104]	SIU	GPIO	I/O	0b00

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 105 (SIU_PCR105)

SIU_BASE+0x112

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨	▨					▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

▨ = Unimplemented or Reserved

1. When configured as GPO, the OBE bit should be set to one.

2. When configured as DSPI_B_PCS[0], DSPI_D_PCS[2] or GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. When configured as GPI, the IBE bit should be set to one.

Figure 152. Pad Configuration Register (SIU_PCR105)

Table 153. SIU_PCR105 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	DSPI_B_PCS[0]	DSPI	Chip select	I/O	0b01
ALT1	DSPI_D_PCS[2]	DSPI	Chip select	O	0b10
GPIO	GPIO[105]	SIU	GPIO	I/O	0b00

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 106 (SIU_PCR106)

SIU_BASE+0x114

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨	▨					▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

▨ = Unimplemented or Reserved

1. When configured as DSPI_D_PCS[0], the OBE bit should be set to one for master operation, and set to zero for slave operation. When configured as GPO, the OBE bit should be set to one.
2. When configured as DSPI_D_PCS[0] in slave operation, the IBE bit should be set to one. When configured as PCS in master operation or GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. When configured as GPI, the IBE bit should be set to one.

Figure 153. Pad Configuration Register (SIU_PCR106)

Table 154. SIU_PCR106 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	DSPI_B_PCS[1]	DSPI	Chip select	O	0b01
ALT1	DSPI_D_PCS[0]	DSPI	Chip select	I/O	0b10
GPIO	GPIO[106]	SIU	GPIO	I/O	0b00

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 107 (SIU_PCR107)

SIU_BASE+0x116

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨	▨					▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

▨ = Unimplemented or Reserved

1. When configured as GPO, the OBE bit should be set to one.
2. When configured as DSPI_B_PCS[2], DSPI_C_SOUT or GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. When configured as GPI, the IBE bit should be set to one.

Figure 154. Pad Configuration Register (SIU_PCR107)

Table 155. SIU_PCR107 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	DSPI_B_PCS[2]	DSPI	Chip select	O	0b01
ALT1	DSPI_C_SOUT	DSPI	Output	O	0b10
GPIO	GPIO[107]	SIU	GPIO	I/O	0b00

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 108 (SIU_PCR108)

SIU_BASE+0x118

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨	▨					▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

▨ = Unimplemented or Reserved

1. When configured as DSPI_C_SIN, the OBE bit has no effect. When configured as GPO, the OBE bit should be set to one.
2. When configured as DSPI_B_PCS[3] or GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. When configured as GPI, the IBE bit should be set to one.

Figure 155. Pad Configuration Register (SIU_PCR108)

Table 156. SIU_PCR108 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	DSPI_B_PCS[3]	DSPI	Chip select	O	0b01
ALT1	DSPI_C_SIN	DSPI	Input	I	0b10
GPIO	GPIO[108]	SIU	GPIO	I/O	0b00

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 109 (SIU_PCR109)

SIU_BASE+0x11A

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE (1)	IBE (2)	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨	▨					▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

▨ = Unimplemented or Reserved

1. When configured as DSPI_C_SCK, the OBE bit should be set to one for master operation, and set to zero for slave operation. When configured as GPO, the OBE bit should be set to one.
2. When configured as DSPI_C_SCK in slave operation, the IBE bit should be set to one. When configured as DSPI_B_PCS[4] or DSPI_C_SCK in master operation or GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. When configured as GPI, the IBE bit should be set to one.

Figure 156. Pad Configuration Register (SIU_PCR109)

Table 157. SIU_PCR109 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	DSPI_B_PCS[4]	DSPI	Chip select	O	0b01
ALT1	DSPI_C_SCK	DSPI	Clock	I/O	0b10
GPIO	GPIO[109]	SIU	GPIO	I/O	0b00

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 110 (SIU_PCR110)

SIU_BASE+0x11C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨	▨					▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

▨ = Unimplemented or Reserved

1. When configured as DSPI_C_PCS[0], the OBE bit should be set to one for master operation, and set to zero for slave operation. When configured as GPO, the OBE bit should be set to one.
2. When configured as DSPI_C_PCS[0] in slave operation, the IBE bit should be set to one. When configured as PCS in master operation or GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. When configured as GPI, the IBE bit should be set to one.

Figure 157. Pad Configuration Register (SIU_PCR110)

Table 158. SIU_PCR110 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	DSPI_B_PCS[5]	DSPI	Chip select	O	0b01
ALT1	DSPI_C_PCS[0]	DSPI	Chip select	I/O	0b10
GPIO	GPIO[110]	SIU	GPIO	I/O	0b00

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 113 (SIU_PCR113)

SIU_BASE+0x122

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨	▨					▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

▨ = Unimplemented or Reserved

1. When configured as TCRCLKA or IRQ, the OBE bit has no effect. When configured as GPO, the OBE bit should be set to one.
2. When configured as TCRCLKA or IRQ or GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. When configured as GPI, the IBE bit should be set to one.

Figure 158. Pad Configuration Register (SIU_PCR113)

Table 159. SIU_PCR113 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	TCRCLKA	eTPU	TCR time base input clock	I	0b01
ALT1	IRQ[7]	SIU	External interrupt	I	0b10
GPIO	GPIO[113]	SIU	GPIO	I/O	0b00

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.

2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 114–125 (SIU_PCR114–SIU_PCR125)

The SIU_PCR114 – SIU_PCR125 registers control the pin function, direction, and static electrical attributes of the ETPUA0 – ETPUA11 pins, which host the ETPU_A[0:11], ETPU_A[12:23] and GPIO[114:125] signals.

Note: Only the output channels of the ETPU_A[12:23] signals are connected to pins. Both the input and output channels of the ETPU_A[0:11] signals are connected to pins.

Pad Configuration Register 114 (SIU_PCR114)

SIU_BASE+0x124

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA			OBE[1]	IBE[2]	0	0	ODE	HYS	SRC		WPE	WPS[3]
W	▨	▨	▨						▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	WKP

▨ = Unimplemented or Reserved

1. The OBE bit must be set to one for both ETPU_A[0] and GPIO[114] when configured as outputs.
2. The IBE bit must be set to one for both ETPU_A[0] and GPIO[114] when configured as inputs. When configured as ETPU_A[12] or ETPU_A[19] or when ETPU_A[0] or GPIO[114] are configured as outputs, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register.
3. The weak pull up/down selection at reset for the ETPU_A[0] pin is determined by the WKPCFG pin.

Figure 159. Pad Configuration Register (SIU_PCR114)

Table 160. SIU_PCR114 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	ETPU_A[0]	eTPU	eTPU channel	I/O	0b001
ALT1	ETPU_A[12]	eTPU	eTPU channel	O	0b010
ALT2	ETPU_A[19]	eTPU	eTPU channel	O	0b100
GPIO	GPIO[114]	SIU	GPIO	I/O	0b000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 115 (SIU_PCR115)

SIU_BASE+0x126

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA			OBE[1]	IBE[2]	0	0	ODE	HYS	SRC		WPE	WPS[3]
W	▨	▨	▨						▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	WKP

▨ = Unimplemented or Reserved

1. The OBE bit must be set to one for both ETPU_A[1] and GPIO[115] when configured as outputs.
2. The IBE bit must be set to one for both ETPU_A[1] and GPIO[113] when configured as inputs. When configured as ETPU_A[13] or when ETPU_A[1] or GPIO[115] are configured as outputs, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register.
3. The weak pull up/down selection at reset for the ETPU_A[1] pin is determined by the WKPCFG pin.

Figure 160. Pad Configuration Register (SIU_PCR115)

Table 161. SIU_PCR115 PA values

Signal	Name	Module	Description	I/O[(1),(2)]	PA value
Primary	ETPU_A[1]	eTPU	eTPU channel	I/O	0b01
ALT1	ETPU_A[13]	eTPU	eTPU channel	O	0b10
GPIO	GPIO[115]	SIU	GPIO	I/O	0b00

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 116 (SIU_PCR116)

SIU_BASE+0x128

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE[1]	IBE[2]	0	0	ODE	HYS	SRC		WPE	WPS[3]
W	▨	▨	▨	▨					▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	WKP

▨ = Unimplemented or Reserved

1. The OBE bit must be set to one for both ETPU_A[2] and GPIO[116] when configured as outputs.
2. The IBE bit must be set to one for both ETPU_A[2] and GPIO[116] when configured as inputs. When configured as ETPU_A[14] or when ETPU_A[2] or GPIO[116] are configured as outputs, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register.
3. The weak pull up/down selection at reset for the ETPU_A[2] pin is determined by the WKPCFG pin.

Figure 161. Pad Configuration Register (SIU_PCR116)

Table 162. SIU_PCR116 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	ETPU_A[2]	eTPU	eTPU channel	I/O	0b01
ALT1	ETPU_A[14]	eTPU	eTPU channel	O	0b10
GPIO	GPIO[116]	SIU	GPIO	I/O	0b00

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 117 (SIU_PCR117)

SIU_BASE+0x12A

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS(3)
W	▨	▨	▨	▨					▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	WKP

▨ = Unimplemented or Reserved

1. The OBE bit must be set to one for both ETPU_A[3] and GPIO[117] when configured as outputs.
2. The IBE bit must be set to one for both ETPU_A[3] and GPIO[117] when configured as inputs. When configured as ETPU_A[15] or when ETPU_A[3] or GPIO[117] are configured as outputs, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register.
3. The weak pull up/down selection at reset for the ETPU_A[3] pin is determined by the WKPCFG pin.

Figure 162. Pad Configuration Register (SIU_PCR117)

Table 163. SIU_PCR117 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	ETPU_A[3]	eTPU	eTPU channel	I/O	0b01
ALT1	ETPU_A[15]	eTPU	eTPU channel	O	0b10
GPIO	GPIO[117]	SIU	GPIO	I/O	0b00

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 118 (SIU_PCR118)

SIU_BASE+0x12C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	PA				OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS(3)
W	▨	▨							▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	WKP

▨ = Unimplemented or Reserved

1. The OBE bit must be set to one for both ETPU_A[4] and GPIO[118] when configured as outputs.

2. The IBE bit must be set to one for both ETPU_A[4] and GPIO[118] when configured as inputs. When configured as ETPU_A[16] or FR_B_TX or when ETPU_A[4] or GPIO[118] are configured as outputs, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register.
3. The weak pull up/down selection at reset for the ETPU_A[4] pin is determined by the WKPCFG pin.

Figure 163. Pad Configuration Register (SIU_PCR118)

Table 164. SIU_PCR118 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	ETPU_A[4]	eTPU	eTPU channel	I/O	0b0001
ALT1	ETPU_A[16]	eTPU	eTPU channel	O	0b0010
ALT3	FR_B_TX	FlexRay	FlexRay transmit	O	0b1000
GPIO	GPIO[118]	SIU	GPIO	I/O	0b0000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 119 (SIU_PCR119)

SIU_BASE+0x12E

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	PA				OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS(3)
W	▨	▨							▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	WKP

▨ = Unimplemented or Reserved

1. The OBE bit must be set to one for both ETPU_A[5] and GPIO[119] when configured as outputs.
2. The IBE bit must be set to one for both ETPU_A[5] and GPIO[119] when configured as inputs. When configured as ETPU_A[17] or when ETPU_A[5] or GPIO[119] are configured as outputs, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register.
3. The weak pull up/down selection at reset for the ETPU_A[5] pin is determined by the WKPCFG pin.

Figure 164. Pad Configuration Register (SIU_PCR119)

Table 165. SIU_PCR119 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	ETPU_A[5]	eTPU	eTPU channel	I/O	0b0001
ALT1	ETPU_A[17]	eTPU	eTPU channel	O	0b0010
ALT2	DSPI_B_SCK_LVDS–	DSPI	LVDS– clock	O	0b0100
ALT3	FR_B_TX_EN	FlexRay	FlexRay transmit enable	O	0b1000
GPIO	GPIO[119]	SIU	GPIO	I/O	0b0000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 120 (SIU_PCR120)

SIU_BASE+0x130

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	PA				OBE[1]	IBE[2]	0	0	ODE	HYS	SRC		WPE	WPS[3]
W	▨	▨							▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	WKP

▨ = Unimplemented or Reserved

1. The OBE bit must be set to one for both ETPU_A[6] and GPIO[120] when configured as outputs.
2. The IBE bit must be set to one for both ETPU_A[6] and GPIO[120] when configured as inputs. When configured as ETPU_A[18] or when ETPU_A[6] or GPIO[119] are configured as outputs, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register.
3. The weak pull up/down selection at reset for the ETPU_A[6] pin is determined by the WKPCFG pin.

Figure 165. Pad Configuration Register (SIU_PCR120)

Table 166. SIU_PCR120 PA values

Signal	Name	Module	Description	I/O[1],[2]	PA value
Primary	ETPU_A[6]	eTPU	eTPU channel	I/O	0b0001
ALT1	ETPU_A[18]	eTPU	eTPU channel	O	0b0010
ALT2	DSPI_B_SCK_LVDS+	DSPI	LVDS+ clock	O	0b0100
ALT3	FR_B_RX	FlexRay	FlexRay receive	I	0b1000
GPIO	GPIO[120]	SIU	GPIO	I/O	0b0000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 121 (SIU_PCR121)

SIU_BASE+0x132

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	PA				OBE[1]	IBE[2]	0	0	ODE	HYS	SRC		WPE	WPS[3]
W	▨	▨							▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	WKP

▨ = Unimplemented or Reserved

1. The OBE bit must be set to one for both ETPU_A[7] and GPIO[121] when configured as outputs.
2. The IBE bit must be set to one for both ETPU_A[7] and GPIO[121] when configured as inputs. When configured as ETPU_A[19] or when ETPU_A[7] or GPIO[119] are configured as outputs, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register.
3. The weak pull up/down selection at reset for the ETPU_A[7] pin is determined by the WKPCFG pin.

Figure 166. Pad Configuration Register (SIU_PCR121)

Table 167. SIU_PCR121 PA values

Signal	Name	Module	Description	I/O[(1),(2)]	PA value
Primary	ETPU_A[7]	eTPU	eTPU channel	I/O	0b0001
ALT1	ETPU_A[19]	eTPU	eTPU channel	O	0b0010
ALT2	DSPI_B_SOUT_LVDS–	DSPI	LVDS– output	O	0b0100
ALT3	ETPU_A[6]	eTPU	eTPU channel	O	0b1000
GPIO	GPIO[121]	SIU	GPIO	I/O	0b0000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 122 (SIU_PCR122)

SIU_BASE+0x134

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA			OBE[(1)]	IBE[(2)]	0	0	ODE	HYS	SRC		WPE	WPS[(3)]
W	▨	▨	▨						▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	WKP

▨ = Unimplemented or Reserved

1. The OBE bit must be set to one for both ETPU_A[8] and GPIO[122] when configured as outputs.
2. The IBE bit must be set to one for both ETPU_A[8] and GPIO[122] when configured as inputs. When configured as ETPU_A[20] or when ETPU_A[8] or GPIO[122] are configured as outputs, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register.
3. The weak pull up/down selection at reset for the ETPU_A[8] pin is determined by the WKPCFG pin.

Figure 167. Pad Configuration Register (SIU_PCR122)

Table 168. SIU_PCR122 PA values

Signal	Name	Module	Description	I/O[(1),(2)]	PA value
Primary	ETPU_A[8]	eTPU	eTPU channel	I/O	0b001
ALT1	ETPU_A[20]	eTPU	eTPU channel	O	0b010
ALT2	DSPI_B_SOUT_LVDS+	DSPI	LVDS+ output	O	0b100
GPIO	GPIO[122]	SIU	GPIO	I/O	0b000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 123 (SIU_PCR123)

SIU_BASE+0x136

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA			OBE (1)	IBE (2)	0	0	ODE	HYS	SRC		WPE	WPS (3)
W	▨	▨	▨						▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	WKP

▨ = Unimplemented or Reserved

1. The OBE bit must be set to one for both ETPU_A[9] and GPIO[123] when configured as outputs.
2. The IBE bit must be set to one for both ETPU_A[9] and GPIO[123] when configured as inputs. When configured as ETPU_A[21] or when ETPU_A[9] or GPIO[123] are configured as outputs, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register.
3. The weak pull up/down selection at reset for the ETPU_A[9] pin is determined by the WKPCFG pin.

Figure 168. Pad Configuration Register (SIU_PCR123)

Table 169. SIU_PCR123 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	ETPU_A[9]	eTPU	eTPU channel	I/O	0b001
ALT1	ETPU_A[21]	eTPU	eTPU channel	O	0b010
ALT2	RCH1_B	Reaction	Reaction channel	O	0b100
GPIO	GPIO[123]	SIU	GPIO	I/O	0b000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 124 (SIU_PCR124)

SIU_BASE+0x138

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA			OBE (1)	IBE (2)	0	0	ODE	HYS	SRC		WPE	WPS (3)
W	▨	▨	▨						▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	WKP

▨ = Unimplemented or Reserved

1. The OBE bit must be set to one for both ETPU_A[10] and GPIO[124] when configured as outputs.
2. The IBE bit must be set to one for both ETPU_A[10] and GPIO[124] when configured as inputs. When configured as ETPU_A[22] or when ETPU_A[10] or GPIO[124] are configured as outputs, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register.
3. The weak pull up/down selection at reset for the ETPU_A[10] pin is determined by the WKPCFG pin.

Figure 169. Pad Configuration Register (SIU_PCR124)

Table 170. SIU_PCR124 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	ETPU_A[10]	eTPU	eTPU channel	I/O	0b001
ALT1	ETPU_A[22]	eTPU	eTPU channel	O	0b010
ALT2	RCH1_C	Reaction	Reaction channel	O	0b100
GPIO	GPIO[124]	SIU	GPIO	I/O	0b000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 125 (SIU_PCR125)

SIU_BASE+0x13A

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA			OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS(3)
W	▨	▨	▨						▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	WKP

▨ = Unimplemented or Reserved

1. The OBE bit must be set to one for both ETPU_A[11] and GPIO[125] when configured as outputs.
2. The IBE bit must be set to one for both ETPU_A[11] and GPIO[125] when configured as inputs. When configured as ETPU_A[23] or when ETPU_A[11] or GPIO[125] are configured as outputs, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register.
3. The weak pull up/down selection at reset for the ETPU_A[11] pin is determined by the WKPCFG pin.

Figure 170. Pad Configuration Register (SIU_PCR125)

Table 171. SIU_PCR125 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	ETPU_A[11]	eTPU	eTPU channel	I/O	0b001
ALT1	ETPU_A[23]	eTPU	eTPU channel	O	0b010
ALT2	RCH4_B	Reaction	Reaction channel	O	0b100
GPIO	GPIO[125]	SIU	GPIO	I/O	0b000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 126 (SIU_PCR126)

SIU_BASE+0x13C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA			OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS(3)
W	▨	▨	▨						▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	WKP

▨ = Unimplemented or Reserved

1. When configured as PCS, the OBE bit has no effect. The OBE bit must be set to one for both ETPUA and GPIO when configured as outputs.
2. The IBE bit must be set to one for both ETPUA and GPIO when configured as inputs. When configured as PCS, or ETPUA or GPO outputs, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register.
3. The weak pull up/down selection at reset for the ETPU_A[12] pin is determined by the WKPCFG pin.

Figure 171. Pad Configuration Register (SIU_PCR126)

Table 172. SIU_PCR126 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	ETPU_A[12]	eTPU	eTPU channel	I/O	0b001
ALT1	DSPI_B_PCS[1]	DSPI	Chip select	O	0b010
ALT2	RCH4_C	Reaction	Reaction channel	O	0b100
GPIO	GPIO[126]	SIU	GPIO	I/O	0b000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 127 (SIU_PCR127)

SIU_BASE+0x13E

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS(3)
W	▨	▨	▨	▨					▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	WKP

▨ = Unimplemented or Reserved

1. The OBE bit must be set to one for both ETPUA and GPIO when configured as outputs.
2. The IBE bit must be set to one for both ETPUA and GPIO when configured as inputs. When configured as PCS, or ETPUA or GPO outputs, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register.
3. The weak pull up/down selection at reset for the ETPU_A[13] pin is determined by the WKPCFG pin.

Figure 172. Pad Configuration Register (SIU_PCR127)

Table 173. SIU_PCR127 PA values

Signal	Name	Module	Description	I/O[(1)],[(2)]	PA value
Primary	ETPU_A[13]	eTPU	eTPU channel	I/O	0b01
ALT1	DSPI_B_PCS[3]	DSPI	Chip select	O	0b10
GPIO	GPIO[127]	SIU	GPIO	I/O	0b00

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 128 (SIU_PCR128)

SIU_BASE+0x140

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	PA				OBE (1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS (3)
W	▨	▨							▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	WKP

▨ = Unimplemented or Reserved

1. The OBE bit must be set to one for both ETPUA and GPIO when configured as outputs.
2. The IBE bit must be set to one for both ETPUA and GPIO when configured as inputs. When configured as PCS, or ETPUA or GPO outputs, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register.
3. The weak pull up/down selection at reset for the ETPU_A[14] pin is determined by the WKPCFG pin.

Figure 173. Pad Configuration Register (SIU_PCR128)

Table 174. SIU_PCR128 PA values

Signal	Name	Module	Description	I/O[(1)],[(2)]	PA value
Primary	ETPU_A[14]	eTPU	eTPU channel	I/O	0b0001
ALT1	DSPI_B_PCS[4]	DSPI	Chip select	O	0b0010
ALT2	ETPU_A[9]	eTPU	eTPU channel	O	0b0100
ALT3	RCH0_A	Reaction	Reaction channel	O	0b1000
GPIO	GPIO[128]	SIU	GPIO	I/O	0b0000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 129 (SIU_PCR129)

SIU_BASE+0x142

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0		PA		OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS(3)
W	▨	▨	▨						▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	WKP

▨ = Unimplemented or Reserved

1. The OBE bit must be set to one for both ETPUA and GPIO when configured as outputs.
2. The IBE bit must be set to one for both ETPUA and GPIO when configured as inputs. When configured as PCS, or ETPUA or GPO outputs, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register.
3. The weak pull up/down selection at reset for the ETPU_A[15] pin is determined by the WKPCFG pin.

Figure 174. Pad Configuration Register (SIU_PCR129)

Table 175. SIU_PCR129 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	ETPU_A[15]	eTPU	eTPU channel	I/O	0b001
ALT1	DSPI_B_PCS[5]	DSPI	Chip select	O	0b010
ALT2	RCH1_A	Reaction	Reaction channel	O	0b100
GPIO	GPIO[129]	SIU	GPIO	I/O	0b000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 130 (SIU_PCR130)

SIU_BASE+0x144

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0		PA		OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS(3)
W	▨	▨	▨						▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	WKP

▨ = Unimplemented or Reserved

1. The OBE bit must be set to one for both ETPUA and GPIO when configured as outputs.
2. The IBE bit must be set to one for both ETPUA and GPIO when configured as inputs. When configured as PCS, or ETPUA or GPO outputs, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register.
3. The weak pull up/down selection at reset for the ETPU_A[16] pin is determined by the WKPCFG pin.

Figure 175. Pad Configuration Register (SIU_PCR130)

Table 176. SIU_PCR130 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	ETPU_A[16]	eTPU	eTPU channel	I/O	0b001
ALT1	DSPI_D_PCS[1]	DSPI	Chip select	O	0b010

Table 176. SIU_PCR130 PA values

Signal	Name	Module	Description	I/O[(1),(2)]	PA value
ALT2	RCH2_A	Reaction	Reaction channel	O	0b100
GPIO	GPIO[130]	SIU	GPIO	I/O	0b000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 131 (SIU_PCR131)

SIU_BASE+0x146

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA			OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS(3)
W	▨	▨	▨						▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	WKP

▨ = Unimplemented or Reserved

1. The OBE bit must be set to one for both ETPUA and GPIO when configured as outputs.
2. The IBE bit must be set to one for both ETPUA and GPIO when configured as inputs. When configured as PCS, or ETPUA or GPO outputs, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register.
3. The weak pull up/down selection at reset for the ETPU_A[17] pin is determined by the WKPCFG pin.

Figure 176. Pad Configuration Register (SIU_PCR131)

Table 177. SIU_PCR131 PA values

Signal	Name	Module	Description	I/O[(1),(2)]	PA value
Primary	ETPU_A[17]	eTPU	eTPU channel	I/O	0b001
ALT1	DSPI_D_PCS[2]	DSPI	Chip select	O	0b010
ALT2	RCH3_A	Reaction	Reaction channel	O	0b100
GPIO	GPIO[131]	SIU	GPIO	I/O	0b000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 132 (SIU_PCR132)

SIU_BASE+0x148

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA			OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS(3)
W	▨	▨	▨						▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	WKP

▨ = Unimplemented or Reserved

1. The OBE bit must be set to one for both ETPUA and GPIO when configured as outputs.

2. The IBE bit must be set to one for both ETPUA and GPIO when configured as inputs. When configured as PCS, or ETPUA or GPO outputs, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register.
3. The weak pull up/down selection at reset for the ETPU_A[18] pin is determined by the WKPCFG pin.

Figure 177. Pad Configuration Register (SIU_PCR132)

Table 178. SIU_PCR132 PA values

Signal	Name	Module	Description	I/O[(1),(2)]	PA value
Primary	ETPU_A[18]	eTPU	eTPU channel	I/O	0b001
ALT1	DSPI_D_PCS[3]	DSPI	Chip select	O	0b010
ALT2	RCH4_A	Reaction	Reaction channel	O	0b100
GPIO	GPIO[132]	SIU	GPIO	I/O	0b000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 133 (SIU_PCR133)

SIU_BASE+0x14A

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA			OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS(3)
W	▨	▨	▨						▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	WKP

▨ = Unimplemented or Reserved

1. The OBE bit must be set to one for both ETPUA and GPIO when configured as outputs.
2. The IBE bit must be set to one for both ETPUA and GPIO when configured as inputs. When configured as PCS, or ETPUA or GPO outputs, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register.
3. The weak pull up/down selection at reset for the ETPU_A[19] pin is determined by the WKPCFG pin.

Figure 178. Pad Configuration Register (SIU_PCR133)

Table 179. SIU_PCR133 PA values

Signal	Name	Module	Description	I/O[(1),(2)]	PA value
Primary	ETPU_A[19]	eTPU	eTPU channel	I/O	0b001
ALT1	DSPI_D_PCS[4]	DSPI	Chip select	O	0b010
ALT2	RCH5_A	Reaction	Reaction channel	O	0b100
GPIO	GPIO[133]	SIU	GPIO	I/O	0b000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 134 (SIU_PCR134)

SIU_BASE+0x14C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	PA				OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS(3)
W	▨	▨							▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	WKP

▨ = Unimplemented or Reserved

1. When configured as IRQ, the OBE bit has no effect. The OBE bit must be set to one for both ETPU_A[20] and GPIO[134] when configured as outputs.
2. When configured as FR_A_TX, IRQ or GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. The IBE bit must be set to one for both ETPU_A[20] and GPIO[134] when configured as inputs.
3. The weak pull up/down selection at reset for the ETPU_A[20] pin is determined by the WKPCFG pin.

Figure 179. Pad Configuration Register (SIU_PCR134)

Table 180. SIU_PCR134 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	ETPU_A[20]	eTPU	eTPU channel	I/O	0b0001
ALT1	IRQ[8]	SIU	External interrupt	I	0b0010
ALT2	RCH0_B	Reaction	Reaction channel	O	0b0100
ALT3	FR_A_TX	FlexRay	FlexRay transmit	O	0b1000
GPIO	GPIO[134]	SIU	GPIO	I/O	0b0000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 135 (SIU_PCR135)

SIU_BASE+0x14E

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	PA				OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS(3)
W	▨	▨							▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	WKP

▨ = Unimplemented or Reserved

1. When configured as IRQ, the OBE bit has no effect. The OBE bit must be set to one for both ETPU_A[21] and GPIO[135] when configured as outputs.
2. When configured as FR_A_RX, IRQ or GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. The IBE bit must be set to one for both ETPU_A[21] and GPIO[135] when configured as inputs.
3. The weak pull up/down selection at reset for the ETPU_A[21] pin is determined by the WKPCFG pin.

Figure 180. Pad Configuration Register (SIU_PCR135)

Table 181. SIU_PCR135 PA values

Signal	Name	Module	Description	I/O[(1),(2)]	PA value
Primary	ETPU_A[21]	eTPU	eTPU channel	I/O	0b0001
ALT1	IRQ[9]	SIU	External interrupt	I	0b0010
ALT2	RCH0_C	Reaction	Reaction channel	O	0b0100
ALT3	FR_A_RX	FlexRay	FlexRay receive	I	0b1000
GPIO	GPIO[135]	SIU	GPIO	I/O	0b0000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 136 (SIU_PCR136)

SIU_BASE+0x150

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA			OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS(3)
W	▨	▨	▨						▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	WKP

▨ = Unimplemented or Reserved

1. When configured as IRQ, the OBE bit has no effect. The OBE bit must be set to one for both ETPU_A[22] and GPIO[136] when configured as outputs.
2. When configured as ETPU_A[17], IRQ or GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. The IBE bit must be set to one for both ETPU_A[22] and GPIO[136] when configured as inputs.
3. The weak pull up/down selection at reset for the ETPU_A[22] pin is determined by the WKPCFG pin.

Figure 181. Pad Configuration Register (SIU_PCR136)

Table 182. SIU_PCR136 PA values

Signal	Name	Module	Description	I/O[(1),(2)]	PA value
Primary	ETPU_A[22]	eTPU	eTPU channel	I/O	0b001
ALT1	IRQ[10]	SIU	External interrupt	I	0b010
ALT2	ETPU_A[17]	eTPU	eTPU channel	O	0b100
GPIO	GPIO[136]	SIU	GPIO	I/O	0b000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 137 (SIU_PCR137)

SIU_BASE+0x152

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	PA				OBE (1)	IBE (2)	0	0	ODE	HYS	SRC		WPE	WPS (3)
W	▨	▨							▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	WKP

▨ = Unimplemented or Reserved

1. When configured as IRQ, the OBE bit has no effect. The OBE bit must be set to one for both ETPU_A[23] and GPIO[137] when configured as outputs.
2. When configured as ETPU_A[21], FR_A_TX_EN, IRQ or GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. The IBE bit must be set to one for both ETPU_A[23] and GPIO[137] when configured as inputs.
3. The weak pull up/down selection at reset for the ETPU_A[23] pin is determined by the WKPCFG pin.

Figure 182. Pad Configuration Register (SIU_PCR137)

Table 183. SIU_PCR137 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	ETPU_A[23]	eTPU	eTPU channel	I/O	0b0001
ALT1	IRQ[11]	SIU	External interrupt	I	0b0010
ALT2	ETPU_A[21]	eTPU	eTPU channel	O	0b0100
ALT3	FR_A_TX_EN	FlexRay	FlexRay transmit enable	O	0b1000
GPIO	GPIO[137]	SIU	GPIO	I/O	0b0000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 138 (SIU_PCR138)

SIU_BASE+0x154

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA			OBE (1)	IBE (2)	0	0	ODE	HYS	SRC		WPE	WPS (3)
W	▨	▨	▨						▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	WKP

▨ = Unimplemented or Reserved

1. When configured as IRQ, the OBE bit has no effect. The OBE bit must be set to one for both ETPU_A[24] and GPIO[138] when configured as outputs.
2. When configured as DSPI_C_SCK_LVDS–, IRQ or GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. The IBE bit must be set to one for both ETPU_A[24] and GPIO[138] when configured as inputs.
3. The weak pull up/down selection at reset for the ETPU_A[24] pin is determined by the WKPCFG pin.

Figure 183. Pad Configuration Register (SIU_PCR138)

Table 184. SIU_PCR138 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	ETPU_A[24](3)	eTPU	eTPU channel	I/O	0b001
ALT1	IRQ[12]	SIU	External interrupt	I	0b010
ALT2	DSPI_C_SCK_LVDS–	DSPI	LVDS– clock	O	0b100
GPIO	GPIO[138]	SIU	GPIO	I/O	0b000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.
3. The eTPU function controlled by this register has an additional dependency on the SIU_ISEL8 register settings. Please see *Section 15.6.22, IMUX Select Register 8 (SIU_ISEL8)*, for more detail.

Pad Configuration Register 139 (SIU_PCR139)

SIU_BASE+0x156

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA			OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS(3)
W	▨	▨	▨						▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	WKP

▨ = Unimplemented or Reserved

1. When configured as IRQ, the OBE bit has no effect. The OBE bit must be set to one for both ETPU_A[25] and GPIO[139] when configured as outputs.
2. When configured as IRQ, DSPI_C_SCK_LVDS+ or GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. The IBE bit must be set to one for both ETPU_A[25] and GPIO[139] when configured as inputs.
3. The weak pull up/down selection at reset for the ETPU_A[25] pin is determined by the WKPCFG pin.

Figure 184. Pad Configuration Register (SIU_PCR139)

Table 185. SIU_PCR139 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	ETPU_A[25](3)	eTPU	eTPU channel	I/O	0b001
ALT1	IRQ[13]	SIU	External interrupt	I	0b010
ALT2	DSPI_C_SCK_LVDS+	DSPI	LVDS+ clock	O	0b100
GPIO	GPIO[139]	SIU	GPIO	I/O	0b000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.
3. The eTPU function controlled by this register has an additional dependency on the SIU_ISEL8 register settings. Please see *Section 15.6.22, IMUX Select Register 8 (SIU_ISEL8)*, for more detail.

Pad Configuration Register 140 (SIU_PCR140)

SIU_BASE+0x158

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA			OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS(3)
W	▨	▨	▨						▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	WKP

▨ = Unimplemented or Reserved

1. When configured as IRQ, the OBE bit has no effect. The OBE bit must be set to one for both ETPU_A[26] and GPIO[140] when configured as outputs.
2. When configured as IRQ, DSPI_C_SOUT_LVDS– or GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. The IBE bit must be set to one for both ETPU_A[26] and GPIO[140] when configured as inputs.
3. The weak pull up/down selection at reset for the ETPU_A[26] pin is determined by the WKPCFG pin.

Figure 185. Pad Configuration Register (SIU_PCR140)

Table 186. SIU_PCR140 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	ETPU_A[26](3)	eTPU	eTPU channel	I/O	0b001
ALT1	IRQ[14]	SIU	External interrupt	I	0b010
ALT2	DSPI_C_SOUT_LVDS–	DSPI	LVDS– output	O	0b100
GPIO	GPIO[140]	SIU	GPIO	I/O	0b000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.
3. The eTPU function controlled by this register has an additional dependency on the SIU_ISEL8 register settings. Please see *Section 15.6.22, IMUX Select Register 8 (SIU_ISEL8)*, for more detail.

Pad Configuration Register 141 (SIU_PCR141)

SIU_BASE+0x15A

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	PA				OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS(3)
W	▨	▨							▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	WKP

▨ = Unimplemented or Reserved

1. When configured as IRQ, the OBE bit has no effect. The OBE bit must be set to one for both ETPU_A[27] and GPIO[141] when configured as outputs.
2. When configured as IRQ, DSPI_C_SOUT_LVDS+, SOUTB or GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. The IBE bit must be set to one for both ETPU_A[27] and GPIO[141] when configured as inputs.
3. The weak pull up/down selection at reset for the ETPU_A[27] pin is determined by the WKPCFG pin.

Figure 186. Pad Configuration Register (SIU_PCR141)

Table 187. SIU_PCR141 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	ETPU_A[27](3)	eTPU	eTPU channel	I/O	0b0001
ALT1	IRQ[15]	SIU	External interrupt	I	0b0010
ALT2	DSPI_C_SOUT_LVDS+	DSPI	LVDS+ output	O	0b0100
ALT3	DSPI_B_SOUT	DSPI	Output	O	0b1000
GPIO	GPIO[141]	SIU	GPIO	I/O	0b0000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.
3. The eTPU function controlled by this register has an additional dependency on the SIU_ISEL8 register settings. Please see *Section 15.6.22, IMUX Select Register 8 (SIU_ISEL8)*, for more detail.

Pad Configuration Register 142 (SIU_PCR142)

SIU_BASE+0x15C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA			OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS(3)
W	▨	▨	▨						▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	WKP

▨ = Unimplemented or Reserved

1. When configured as GPO, the OBE bit should be set to one.
2. When configured as PCS or GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. The IBE bit must be set to one for GPIO when configured as input.
3. The weak pull up/down selection at reset for the ETPU_A[28] pin is determined by the WKPCFG pin.

Figure 187. Pad Configuration Register (SIU_PCR142)

Table 188. SIU_PCR142 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	ETPU_A[28](3)	eTPU	eTPU channel	I/O	0b001
ALT1	DSPI_C_PCS[1]	DSPI	Chip select	O	0b010
ALT2	RCH5_B	Reaction	Reaction channel	O	0b100
GPIO	GPIO[142]	SIU	GPIO	I/O	0b000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.
3. The eTPU function controlled by this register has an additional dependency on the SIU_ISEL8 register settings. Please see *Section 15.6.22, IMUX Select Register 8 (SIU_ISEL8)*, for more detail.

Pad Configuration Register 143 (SIU_PCR143)

SIU_BASE+0x15E

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA			OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS(3)
W	▨	▨	▨						▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	WKP

▨ = Unimplemented or Reserved

1. When configured as GPO, the OBE bit should be set to one.
2. When configured as PCS or GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. The IBE bit must be set to one for GPIO when configured as input.
3. The weak pull up/down selection at reset for the ETPU_A[29] pin is determined by the WKPCFG pin.

Figure 188. Pad Configuration Register (SIU_PCR143)

Table 189. SIU_PCR143 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	ETPU_A[29](3)	eTPU	eTPU channel	I/O	0b001
ALT1	DSPI_C_PCS[2]	DSPI	Chip select	O	0b010
ALT2	RCH5_C	Reaction	Reaction channel	O	0b100
GPIO	GPIO[143]	SIU	GPIO	I/O	0b000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.
3. The eTPU function controlled by this register has an additional dependency on the SIU_ISEL8 register settings. Please see *Section 15.6.22, IMUX Select Register 8 (SIU_ISEL8)*, for more detail.

Pad Configuration Register 144 (SIU_PCR144)

SIU_BASE+0x160

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA			OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS(3)
W	▨	▨	▨						▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	WKP

▨ = Unimplemented or Reserved

1. When configured as ETPUA output or GPO, the OBE bit should be set to one.
2. When configured as ETPUA output, PCS, or GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. The IBE bit must be set to one for ETPUA or GPIO when configured as input.
3. The weak pull up/down selection at reset for the ETPU_A[30] pin is determined by the WKPCFG pin

Figure 189. Pad Configuration Register (SIU_PCR144)

Table 190. SIU_PCR144 PA values

Signal	Name	Module	Description	I/O[1],[2]	PA value
Primary	ETPU_A[30]	eTPU	eTPU channel	I/O	0b001
ALT1	DSPI_C_PCS[3]	DSPI	Chip select	O	0b010
ALT2	ETPU_A[11]	eTPU	eTPU channel	O	0b100
GPIO	GPIO[144]	SIU	GPIO	I/O	0b000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 145 (SIU_PCR145)

SIU_BASE+0x162

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA			OBE[1]	IBE[2]	0	0	ODE	HYS	SRC		WPE	WPS[3]
W	▨	▨	▨						▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	WKP

▨ = Unimplemented or Reserved

1. When configured as ETPUA output or GPO, the OBE bit should be set to one.
2. When configured as ETPUA output, DSPI_C_PCS[4], ETPU_A[13] or GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. The IBE bit must be set to one for ETPUA or GPIO when configured as input.
3. The weak pull up/down selection at reset for the ETPU_A[31] pin is determined by the WKPCFG pin.

Figure 190. Pad Configuration Register (SIU_PCR145)

Table 191. SIU_PCR145 PA values

Signal	Name	Module	Description	I/O[1],[2]	PA value
Primary	ETPU_A[31]	eTPU	eTPU channel	I/O	0b001
ALT1	DSPI_C_PCS[4]	DSPI	Chip select	O	0b010
ALT2	ETPU_A[13]	eTPU	eTPU channel	O	0b100
GPIO	GPIO[145]	SIU	GPIO	I/O	0b000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 179 (SIU_PCR179)

SIU_BASE+0x1A6

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA			OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨						▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

▨ = Unimplemented or Reserved

1. The OBE bit must be set to one for both EMIOS[0] and GPIO[179] when configured as outputs.
2. When configured as ETPU, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. The IBE bit must be set to one for both EMIOS[0] and GPIO[179] when configured as inputs.

Figure 191. Pad Configuration Register (SIU_PCR179)

Table 192. SIU_PCR179 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	EMIOS[0]	eMIOS	eMIOS channel	I/O	0b001
ALT1	ETPU_A[0]	eTPU	eTPU channel	O	0b010
ALT2	ETPU_A[25]	eTPU	eTPU channel	O	0b100
GPIO	GPIO[179]	SIU	GPIO	I/O	0b000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 180 (SIU_PCR180)

SIU_BASE+0x1A8

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨	▨					▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

▨ = Unimplemented or Reserved

1. The OBE bit must be set to one for both EMIOS[1] and GPIO[180] when configured as outputs.
2. When configured as ETPU, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. The IBE bit must be set to one for both EMIOS[1] and GPIO[180] when configured as inputs.

Figure 192. Pad Configuration Register (SIU_PCR180)

Table 193. SIU_PCR180 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	EMIOS[1]	eMIOS	eMIOS channel	I/O	0b01

Table 193. SIU_PCR180 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
ALT1	ETPU_A[1]	eTPU	eTPU channel	O	0b10
GPIO	GPIO[180]	SIU	GPIO	I/O	0b00

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 181 (SIU_PCR181)

SIU_BASE+0x1AA

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA			OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨						▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

▨ = Unimplemented or Reserved

1. The OBE bit must be set to one for both EMIOS[2] and GPIO[181] when configured as outputs.
2. When configured as ETPU, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. The IBE bit must be set to one for both EMIOS[2] and GPIO[181] when configured as inputs.

Figure 193. Pad Configuration Register (SIU_PCR181)

Table 194. SIU_PCR181 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	EMIOS[2]	eMIOS	eMIOS channel	I/O	0b001
ALT1	ETPU_A[2]	eTPU	eTPU channel	O	0b010
ALT2	RCH2_B	Reaction	Reaction channel	O	0b100
GPIO	GPIO[181]	SIU	GPIO	I/O	0b000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 182 (SIU_PCR182)

SIU_BASE+0x1AC

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS(3)
W	▨	▨	▨	▨					▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	WKP

▨ = Unimplemented or Reserved

1. The OBE bit must be set to one for both EMIOS[3] and GPIO[182] when configured as outputs.

2. When configured as ETPU, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. The IBE bit must be set to one for both EMIOS[3] and GPIO[182] when configured as inputs.
3. The weak pull up/down selection at reset for the EMIOS[3] pin is determined by the WKPCFG pin.

Figure 194. Pad Configuration Register (SIU_PCR182)

Table 195. SIU_PCR182 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	EMIOS[3]	eMIOS	eMIOS channel	I/O	0b01
ALT1	ETPU_A[3]	eTPU	eTPU channel	O	0b10
GPIO	GPIO[182]	SIU	GPIO	I/O	0b00

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 183 (SIU_PCR183)

SIU_BASE+0x1AE

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA			OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS(3)
W	▨	▨	▨						▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	WKP

▨ = Unimplemented or Reserved

1. The OBE bit must be set to one for both EMIOS[4] and GPIO[183] when configured as outputs.
2. When configured as ETPU, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. The IBE bit must be set to one for both EMIOS[4] and GPIO[183] when configured as inputs.
3. The weak pull up/down selection at reset for the EMIOS[4] pin is determined by the WKPCFG pin.

Figure 195. Pad Configuration Register (SIU_PCR183)

Table 196. SIU_PCR183 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	EMIOS[4]	eMIOS	eMIOS channel	I/O	0b001
ALT1	ETPU_A[4]	eTPU	eTPU channel	O	0b010
ALT2	RCH2_C	Reaction	Reaction channel	O	0b100
GPIO	GPIO[183]	SIU	GPIO	I/O	0b000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 184 (SIU_PCR184)

SIU_BASE+0x1B0

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS(3)
W	▨	▨	▨	▨					▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	WKP

▨ = Unimplemented or Reserved

1. The OBE bit must be set to one for both EMIOS[5] and GPIO[184] when configured as outputs.
2. When configured as ETPU, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. The IBE bit must be set to one for both EMIOS[5] and GPIO[184] when configured as inputs.
3. The weak pull up/down selection at reset for the EMIOS[5] pin is determined by the WKPCFG pin.

Figure 196. Pad Configuration Register (SIU_PCR184)

Table 197. SIU_PCR184 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	EMIOS[5]	eMIOS	eMIOS channel	I/O	0b01
ALT1	ETPU_A[5]	eTPU	eTPU channel	O	0b10
GPIO	GPIO[184]	SIU	GPIO	I/O	0b00

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 185 (SIU_PCR185)

SIU_BASE+0x1B2

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨	▨					▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

▨ = Unimplemented or Reserved

1. The OBE bit must be set to one for both EMIOS[6] and GPIO[185] when configured as outputs.
2. When configured as ETPU, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. The IBE bit must be set to one for both EMIOS[6] and GPIO[185] when configured as inputs.

Figure 197. Pad Configuration Register (SIU_PCR185)

Table 198. SIU_PCR185 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	EMIOS[6]	eMIOS	eMIOS channel	I/O	0b01
ALT1	ETPU_A[6]	eTPU	eTPU channel	O	0b10
GPIO	GPIO[185]	SIU	GPIO	I/O	0b00

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 186 (SIU_PCR186)

SIU_BASE+0x1B4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨	▨					▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

▨ = Unimplemented or Reserved

1. The OBE bit must be set to one for both EMIOS[7] and GPIO[186] when configured as outputs.
2. When configured as ETPU, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. The IBE bit must be set to one for both EMIOS[7] and GPIO[186] when configured as inputs.

Figure 198. Pad Configuration Register (SIU_PCR186)

Table 199. SIU_PCR186 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	EMIOS[7]	eMIOS	eMIOS channel	I/O	0b01
ALT1	ETPU_A[7]	eTPU	eTPU channel	O	0b10
GPIO	GPIO[186]	SIU	GPIO	I/O	0b00

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 187 (SIU_PCR187)

SIU_BASE+0x1B6

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA			OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨						▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

▨ = Unimplemented or Reserved

1. The OBE bit must be set to one for both EMIOS[8] and GPIO[187] when configured as outputs.

2. When configured as ETPU or SCI_B_TX, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. The IBE bit must be set to one for both EMIOS[8] and GPIO[187] when configured as inputs.

Figure 199. Pad Configuration Register (SIU_PCR187)

Table 200. SIU_PCR187 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	EMIOS[8]	eMIOS	eMIOS channel	I/O	0b001
ALT1	ETPU_A[8]	eTPU	eTPU channel	O	0b010
ALT2	SCI_B_TX	eSCI	eSCI transmit	O	0b100
GPIO	GPIO[187]	SIU	GPIO	I/O	0b000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 188 (SIU_PCR188)

SIU_BASE+0x1B8

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA			OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨						▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

▨ = Unimplemented or Reserved

1. The OBE bit must be set to one for both EMIOS[9] and GPIO[188] when configured as outputs.
2. When configured as ETPU, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. The IBE bit must be set to one for both EMIOS[9] and GPIO[188] when configured as inputs.

Figure 200. Pad Configuration Register (SIU_PCR188)

Table 201. SIU_PCR188 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	EMIOS[9]	eMIOS	eMIOS channel	I/O	0b001
ALT1	ETPU_A[9]	eTPU	eTPU channel	O	0b010
ALT2	SCI_B_RX	eSCI	eSCI receive	I	0b100
GPIO	GPIO[188]	SIU	GPIO	I/O	0b000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 189 (SIU_PCR189)

SIU_BASE+0x1BA

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA			OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS(3)
W	▨	▨	▨						▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	WKP

▨ = Unimplemented or Reserved

1. The OBE bit must be set to one for both EMIOS[10] and GPIO[189] when configured as outputs.
2. When configured as GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. The IBE bit must be set to one for both EMIOS[10] and GPIO[189] when configured as inputs.
3. The weak pull up/down selection at reset for the EMIOS[10] pin is determined by the WKPCFG pin.

Figure 201. Pad Configuration Register (SIU_PCR189)

Table 202. SIU_PCR189 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	EMIOS[10]	eMIOS	eMIOS channel	I/O	0b001
ALT1	DSPI_D_PCS[3]	DSPI	Chip select	O	0b010
ALT2	RCH3_B	Reaction	Reaction channel	O	0b0100
GPIO	GPIO[189]	SIU	GPIO	I/O	0b000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 190 (SIU_PCR190)

SIU_BASE+0x1BC

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA			OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS(3)
W	▨	▨	▨						▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	WKP

▨ = Unimplemented or Reserved

1. The OBE bit must be set to one for both EMIOS[11] and GPIO[190] when configured as outputs.
2. When configured as GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. The IBE bit must be set to one for both EMIOS[11] and GPIO[190] when configured as inputs.
3. The weak pull up/down selection at reset for the EMIOS[11] pin is determined by the WKPCFG pin.

Figure 202. Pad Configuration Register (SIU_PCR190)

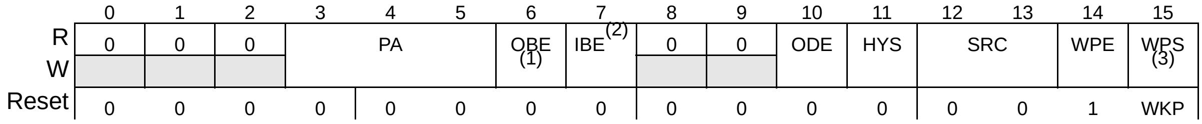
Table 203. SIU_PCR190 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	EMIOS[11]	eMIOS	eMIOS channel	I/O	0b001
ALT1	DSPI_D_PCS[4]	DSPI	Chip select	O	0b010
ALT2	RCH3_C	Reaction	Reaction channel	O	0b0100
GPIO	GPIO[190]	SIU	GPIO	I/O	0b000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 191 (SIU_PCR191)

SIU_BASE+0x1BE



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA			OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS(3)
W	▨	▨	▨						▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	WKP

▨ = Unimplemented or Reserved

1. The OBE bit must be set to one for GPIO[191] when configured as an output.
2. When configured as ETPU_A[27] or GPO the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. The IBE bit must be set to one for GPIO[191] when configured as an input.
3. The weak pull up/down selection at reset for the EMIOS[12] pin is determined by the WKPCFG pin.

Figure 203. Pad Configuration Register (SIU_PCR191)

Table 204. SIU_PCR191 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	EMIOS[12]	eMIOS	eMIOS channel	I/O	0b001
ALT1	DSPI_C_SOUT	DSPI	Output	O	0b010
ALT2	ETPU_A[27]	eTPU	eTPU channel	O	0b100
GPIO	GPIO[191]	SIU	GPIO	I/O	0b000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 192 (SIU_PCR192)

SIU_BASE+0x1C0

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS(3)
W	▨	▨	▨	▨					▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	WKP

▨ = Unimplemented or Reserved

1. The OBE bit must be set to one for GPIO[192] when configured as an output.
2. When configured as GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. The IBE bit must be set to one for GPIO[192] when configured as an input.
3. The weak pull up/down selection at reset for the EMIOS[13] pin is determined by the WKPCFG pin.

Figure 204. Pad Configuration Register (SIU_PCR192)

Table 205. SIU_PCR192 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	EMIOS[13]	eMIOS	eMIOS channel	I/O	0b01
ALT1	DSPI_D_SOUT	DSPI	Output	O	0b10
GPIO	GPIO[192]	SIU	GPIO	I/O	0b00

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 193 (SIU_PCR193)

SIU_BASE+0x1C2

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA			OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨						▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

▨ = Unimplemented or Reserved

1. The OBE bit must be set to one for GPIO[193] when configured as outputs.
2. When configured as IRQ, ETPU or GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. The IBE bit must be set to one for GPIO[193] when configured as inputs.

Figure 205. Pad Configuration Register (SIU_PCR193)

Table 206. SIU_PCR193 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	EMIOS[14]	eMIOS	eMIOS channel	I/O	0b001
ALT1	IRQ[0]	SIU	External interrupt	I	0b010
ALT2	ETPU_A[29]	eTPU	eTPU channel	O	0b100
GPIO	GPIO[193]	SIU	GPIO	I/O	0b000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 194 (SIU_PCR194)

SIU_BASE+0x1C4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨	▨					▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

▨ = Unimplemented or Reserved

1. The OBE bit must be set to one for GPIO[194] when configured as outputs.
2. When configured as IRQ, ETPU or GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. The IBE bit must be set to one for GPIO[194] when configured as inputs.

Figure 206. Pad Configuration Register (SIU_PCR194)

Table 207. SIU_PCR194 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	EMIOS[15]	eMIOS	eMIOS channel	I/O	0b01
ALT1	IRQ[1]	SIU	External interrupt	I	0b10
GPIO	GPIO[194]	SIU	GPIO	I/O	0b00

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 195 (SIU_PCR195)

SIU_BASE+0x1C6

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	PA	OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨	▨	▨				▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

▨ = Unimplemented or Reserved

1. The OBE bit must be set to one for both EMIOS[16] and GPIO[195] when configured as outputs.

2. When configured as GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. The IBE bit must be set to one for both EMIOS[16] and GPIO[195] when configured as inputs.

Figure 207. Pad Configuration Register (SIU_PCR195)

Table 208. SIU_PCR195 PA values

Signal	Name	Module	Description	I/O[1],[2]	PA value
Primary	EMIOS[16]	eMIOS	eMIOS channel	I/O	0b1
GPIO	GPIO[195]	SIU	GPIO	I/O	0b0

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 196 (SIU_PCR196)

SIU_BASE+0x1C8

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	PA	OBE[1]	IBE[2]	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨	▨	▨				▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

▨ = Unimplemented or Reserved

1. The OBE bit must be set to one for both EMIOS[17] and GPIO[196] when configured as outputs.
2. When configured as GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. The IBE bit must be set to one for both EMIOS[17] and GPIO[196] when configured as inputs.

Figure 208. Pad Configuration Register (SIU_PCR196)

Table 209. SIU_PCR196 PA values

Signal	Name	Module	Description	I/O[1],[2]	PA value
Primary	EMIOS[17]	eMIOS	eMIOS channel	I/O	0b1
GPIO	GPIO[196]	SIU	GPIO	I/O	0b0

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 197 (SIU_PCR197)

SIU_BASE+0x1CA

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	PA	OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨	▨	▨				▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

▨ = Unimplemented or Reserved

1. The OBE bit must be set to one for both EMIOS[18] and GPIO[197] when configured as outputs.
2. When configured as GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. The IBE bit must be set to one for both EMIOS[18] and GPIO[197] when configured as inputs.

Figure 209. Pad Configuration Register (SIU_PCR197)

Table 210. SIU_PCR197 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	EMIOS[18]	eMIOS	eMIOS channel	I/O	0b1
GPIO	GPIO[197]	SIU	GPIO	I/O	0b0

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 198 (SIU_PCR198)

SIU_BASE+0x1CC

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	PA	OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨	▨	▨				▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	WKP

▨ = Unimplemented or Reserved

1. The OBE bit must be set to one for both EMIOS[19] and GPIO[198] when configured as outputs.
2. When configured as GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. The IBE bit must be set to one for both EMIOS[19] and GPIO[198] when configured as inputs.

Figure 210. Pad Configuration Register (SIU_PCR198)

Table 211. SIU_PCR198 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	EMIOS[19]	eMIOS	eMIOS channel	I/O	0b1
GPIO	GPIO[198]	SIU	GPIO	I/O	0b0

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 199 (SIU_PCR199)

SIU_BASE+0x1CE

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	PA	OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS(3)
W	▨	▨	▨	▨	▨				▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	WKP

▨ = Unimplemented or Reserved

1. The OBE bit must be set to one for both EMIOS[20] and GPIO[199] when configured as outputs.
2. When configured as GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. The IBE bit must be set to one for both EMIOS[20] and GPIO[199] when configured as inputs.
3. The weak pull up/down selection at reset for the EMIOS[20] pin is determined by the WKPCFG pin.

Figure 211. Pad Configuration Register (SIU_PCR199)

Table 212. SIU_PCR199 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	EMIOS[20]	eMIOS	eMIOS channel	I/O	0b1
GPIO	GPIO[199]	SIU	GPIO	I/O	0b0

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 200 (SIU_PCR200)

SIU_BASE+0x1D0

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	PA	OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS(3)
W	▨	▨	▨	▨	▨				▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	WKP

▨ = Unimplemented or Reserved

1. The OBE bit must be set to one for both EMIOS[21] and GPIO[200] when configured as outputs.
2. When configured as GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. The IBE bit must be set to one for both EMIOS[21] and GPIO[200] when configured as inputs.
3. The weak pull up/down selection at reset for the EMIOS[21] pin is determined by the WKPCFG pin.

Figure 212. Pad Configuration Register (SIU_PCR200)

Table 213. SIU_PCR200 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	EMIOS[21]	eMIOS	eMIOS channel	I/O	0b1
GPIO	GPIO[200]	SIU	GPIO	I/O	0b0

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.

2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 201 (SIU_PCR201)

SIU_BASE+0x1D2

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	PA	OBE (1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨	▨	▨				▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

▨ = Unimplemented or Reserved

1. The OBE bit must be set to one for both EMIOS[22] and GPIO[201] when configured as outputs.
2. When configured as GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. The IBE bit must be set to one for both EMIOS[22] and GPIO[201] when configured as inputs.

Figure 213. Pad Configuration Register (SIU_PCR201)

Table 214. SIU_PCR201 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	EMIOS[22]	eMIOS	eMIOS channel	I/O	0b1
GPIO	GPIO[201]	SIU	GPIO	I/O	0b0

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 202 (SIU_PCR202)

SIU_BASE+0x1D4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	PA	OBE (1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨	▨	▨				▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

▨ = Unimplemented or Reserved

1. The OBE bit must be set to one for both EMIOS[23] and GPIO[202] when configured as outputs.
2. When configured as GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. The IBE bit must be set to one for both EMIOS[23] and GPIO[202] when configured as inputs.

Figure 214. Pad Configuration Register (SIU_PCR202)

Table 215. SIU_PCR202 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	EMIOS[23]	eMIOS	eMIOS channel	I/O	0b1
GPIO	GPIO[202]	SIU	GPIO	I/O	0b0

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 203 (SIU_PCR203)

SIU_BASE+0x1D6

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	PA(1)	OBE(2)	IBE(3)	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨	▨	▨				▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

▨ = Unimplemented or Reserved

1. The PA bit should be set to one for EMIOS and cleared to zero when used as GPIO.
2. When configured as GPO, the OBE bit should be set to one.
3. When configured as EMIOS or GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. When configured as GPI, the IBE bit should be set to one.

Figure 215. Pad Configuration Register (SIU_PCR203)

Table 216. SIU_PCR203 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	EMIOS[14]	eMIOS	eMIOS channel	O	0b1
GPIO	GPIO[203]	SIU	GPIO	I/O	0b0

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 204 (SIU_PCR204)

SIU_BASE+0x1D8

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	PA(1)	OBE (2)	IBE(3)	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨	▨	▨				▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

▨ = Unimplemented or Reserved

1. The PA bit should be set to one for EMIOS and cleared to zero when used as GPIO.
2. When configured as GPO, the OBE bit should be set to one.
3. When configured as EMIOS or GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. When configured as GPI, the IBE bit should be set to one.

Figure 216. Pad Configuration Register (SIU_PCR204)

Table 217. SIU_PCR204 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	EMIOS[15]	eMIOS	eMIOS channel	O	0b1
GPIO	GPIO[204]	SIU	GPIO	I/O	0b0

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 206 (SIU_PCR206)

SIU_BASE+0x1DC

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	PA	OBE (1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨	▨	▨				▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

▨ = Unimplemented or Reserved

1. When configured as GPO, the OBE bit should be set to one.
2. When configured as GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. When configured as GPI, the IBE bit should be set to one. Setting the IBE bit to zero reduces power consumption.

Figure 217. Pad Configuration Register (SIU_PCR206)

Table 218. SIU_PCR206 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
GPIO	GPIO[206] ETRIG0	SIU	GPIO	I/O	0b0

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 207 (SIU_PCR207)

SIU_BASE+0x1DE

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	PA	OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨	▨	▨				▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

▨ = Unimplemented or Reserved

1. When configured as GPO, the OBE bit should be set to one.
2. When configured as GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. When configured as GPI, the IBE bit should be set to one. Setting the IBE bit to zero reduces power consumption.

Figure 218. Pad Configuration Register (SIU_PCR207)

Table 219. SIU_PCR207 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
GPIO	GPIO[207] ETRIG1	SIU	GPIO	I/O	0b0

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 208 (SIU_PCR208)

SIU_BASE+0x1E0

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA			OBE(1)	IBE(2)	0	0	ODE	HYS(3)	SRC		WPE	WPS
W	▨	▨	▨						▨	▨						
Reset	0	0	0	0	0	1	0	1	0	0	0	1	0	0	1	1

▨ = Unimplemented or Reserved

1. When configured as IRQ, the OBE bit has no effect. When configured as GPO, the OBE bit should be set to one.
2. When configured as IRQ or GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. When configured as GPI, the IBE bit should be set to one.
3. When configured as IRQ, the HYS bit should be set to one.

Figure 219. Pad Configuration Register (SIU_PCR208)

Table 220. SIU_PCR208 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	PLLREF	Config	Selects FMPLL mode after reset	I	0b001
ALT1	IRQ[4]	SIU	External interrupt	I	0b010
ALT2	ETRIG2	SIU	Triggers eQADC CFIFO2	I	0b100
GPIO	GPIO[208]	SIU	GPIO	I/O	0b000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.

2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 209 (SIU_PCR209)

SIU_BASE+0x1E2

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA			OBE[1]	IBE[2]	0	0	ODE	HYS[3]	SRC		WPE	WPS
W	▨	▨	▨						▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

▨ = Unimplemented or Reserved

1. When configured as IRQ, the OBE bit has no effect. When configured as GPO, the OBE bit should be set to one.
2. When configured as IRQ or GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. When configured as GPI, the IBE bit should be set to one.
3. When configured as IRQ, the HYS bit should be set to one.

Figure 220. Pad Configuration Register (SIU_PCR209)

Table 221. SIU_PCR209 PA values

Signal	Name	Module	Description	I/O[1],[2]	PA value
PLLCFG1					
ALT1	IRQ[5]	SIU	External interrupt	I	0b010
ALT2	DSPI_D_SOUT	DSPI	Output	O	0b100
GPIO	GPIO[209]	SIU	GPIO	I/O	0b000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 210 (SIU_PCR210)

SIU_BASE+0x1E4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	PA[1]	OBE[2]	IBE[3]	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨	▨	▨				▨	▨						
Reset	0	0	0	0	0	1	0	1	0	0	0	1	0	0	1	0

▨ = Unimplemented or Reserved

1. $\overline{\text{RSTCFG}}$ function is only applicable during reset. The PA bit must be set to zero for GPIO operation
2. When configured as GPO, the OBE bit should be set to one.
3. When configured as GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. When configured as GPI, the IBE bit should be set to one.

Figure 221. Pad Configuration Register (SIU_PCR210)

Table 222. SIU_PCR210 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	RSTCFG	Reset/Config	Enable/disable PLLREF and BOOTCFG	I	0b1
GPIO	GPIO[210]	SIU	GPIO	I/O	0b0

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 211 (SIU_PCR211)

SIU_BASE+0x1E6

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA(1)		OBE (2)	IBE(3)	0	0	ODE	HYS (4)	SRC		WPE	WPS
W	▨	▨	▨	▨					▨	▨						
Reset	0	0	0	0	0	1	0	1	0	0	0	1	0	0	1	0

▨ = Unimplemented or Reserved

1. The BOOTCFG function applies only during reset when the $\overline{\text{RSTCFG}}$ pin is asserted during reset.
2. When configured as IRQ, the OBE bit has no effect. When configured as GPO, the OBE bit should be set to one.
3. When configured as IRQ or GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. When configured as GPI, the IBE bit should be set to one.
4. When configured as IRQ, the HYS bit should be set to one.

Figure 222. Pad Configuration Register (SIU_PCR211)

Table 223. SIU_PCR211 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	BOOTCFG[0]	Reset/Config	Selects boot mode	I	0b01
ALT1	IRQ[2]	SIU	External interrupt	I	0b10
GPIO	GPIO[211]	SIU	GPIO	I/O	0b00

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 212 (SIU_PCR212)

SIU_BASE+0x1E8

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA(1)			OBE (2)	IBE(3)	0	0	ODE	HYS (4)	SRC		WPE	WPS
W	▨	▨	▨						▨	▨						
Reset	0	0	0	0	0	1	0	1	0	0	0	1	0	0	1	0

▨ = Unimplemented or Reserved

1. The BOOTCFG function applies only during reset when the $\overline{\text{RSTCFG}}$ pin is asserted during reset.
2. When configured as IRQ, the OBE bit has no effect. When configured as GPO, the OBE bit should be set to one.

3. When configured as IRQ or GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. When configured as GPI, the IBE bit should be set to one.
4. When configured as IRQ, the HYS bit should be set to one.

Figure 223. Pad Configuration Register (SIU_PCR212)

Table 224. SIU_PCR212 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	BOOTCFG[1]	Reset/Config	Selects boot mode	I	0b001
ALT1	IRQ[3]	SIU	External interrupt	I	0b010
ALT2	ETRIG3	SIU	Triggers eQADC CFIFO3	I	0b100
GPIO	GPIO[212]	SIU	GPIO	I/O	0b000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 213 (SIU_PCR213)

SIU_BASE+0x1EA

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA(1)			OBE(2)	IBE(3)	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨						▨	▨						
Reset	0	0	0	0	0	1	0	1	0	0	0	1	0	0	1	1

▨ = Unimplemented or Reserved

1. WKPCFG function is only applicable during reset.
2. When configured as GPO, the OBE bit should be set to one.
3. When configured as GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. When configured as GPI, the IBE bit should be set to one.

Figure 224. Pad Configuration Register (SIU_PCR213)

Table 225. SIU_PCR213 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	WKPCFG	Reset/Config	Connects eTPU and eMIOS pins to internal weak pull-up or weak pull-down devices after reset	I	0b001
ALT1	$\overline{\text{NMI}}$	Reset/Config	Non-Maskable interrupt	I	0b010
ALT2	DSPI_B_SOUT	DSPI	Output	O	0b100
GPIO	GPIO[213]	SIU	GPIO	I/O	0b000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 214 (SIU_PCR214)

The SIU_PCR214 register controls the enabling/disabling and drive strength of the ENGCLK pin. The ENGCLK pin is enabled and disabled by setting and clearing the OBE bit. The ENGCLK pin is enabled during reset.

SIU_BASE+0x1EC

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	PA	OBE(1)	0	DSC		ODE	HYS	0	0	WPE	WPS
W	▨	▨	▨	▨	▨			▨					▨	▨		
Reset	0	0	0	0	0	1	1	0	1	1	0	0	0	0	0	0

▨ = Unimplemented or Reserved

1. ENGCLK is enabled/disabled by setting/clearing this bit.

Figure 225. Pad Configuration Register (SIU_PCR214)

Table 226. SIU_PCR214 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	ENGCLK	Clock Generation	Engineering clock output	O	0b001

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 215 (SIU_PCR215)

SIU_BASE+0x1EE

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	PA(1)				0	0	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨					▨	▨	▨	▨						
Reset	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

1. Input and output buffers are enabled/disabled based on PA selection. Both input and output buffer disabled for AN[12] function. Output buffer only enabled for MA[0], ETPU and SDS functions.

Figure 226. Pad Configuration Register (SIU_PCR215)

Table 227. SIU_PCR215 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	AN[12]	eQADC	Analog input	I	0b0001
ALT1	MA[0]	eQADC	Mux address	O	0b0010
ALT2	ETPU_A[19]	eTPU	eTPU channel	O	0b0100
GPIO	$\overline{\text{SDS}}$	eQADC	Serial data select	O	0b0000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.

2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 216 (SIU_PCR216)

SIU_BASE+0x1F0

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	PA[1]				0	0	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨					▨	▨	▨	▨						
Reset	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

1. Input and output buffers are enabled/disabled based on PA selection. Both input and output buffer disabled for AN[13] function. Output buffer only enabled for MA[1], ETPU and SDO functions.

Figure 227. Pad Configuration Register (SIU_PCR216)

Table 228. SIU_PCR216 PA values

Signal	Name	Module	Description	I/O[1],[2]	PA value
Primary	AN[13]	eQADC	Analog input	I	0b0001
ALT1	MA[1]	eQADC	Mux address	O	0b0010
ALT2	ETPU_A[21]	eTPU	eTPU channel	O	0b0100
GPIO	SDO	eQADC	Serial data output	O	0b0000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 217 (SIU_PCR217)

SIU_BASE+0x1F2

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	PA[1]				0	0	0	0	ODE	HYS	SRC		WPE[2]	WPS[3]
W	▨	▨					▨	▨	▨	▨						
Reset	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

1. Input and output buffers are enabled/disabled based on PA selection. Both input and output buffer disabled for AN[14] function. Output buffer only enabled for MA[2] and ETPU function and input buffer only enabled for SDI functions.
2. The WPE bit should be set to zero when configured as an analog input or MA[2], and set to one when configured as SDI.
3. The WPS bit should be set to one when configured as SDI.

Figure 228. Pad Configuration Register (SIU_PCR217)

Table 229. SIU_PCR217 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	AN[14]	eQADC	Analog input	I	0b0001
ALT1	MA[2]	eQADC	Mux address	O	0b0010
ALT2	ETPU_A[27]	eTPU	eTPU channel	O	0b0100
GPIO	SDI	eQADC	Serial data input	I	0b0000

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 218 (SIU_PCR218)

SIU_BASE+0x1F4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA(1)			0	0	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨				▨	▨	▨	▨						
Reset	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

1. Input and output buffers are enabled/disabled based on PA selection. Both input and output buffer disabled for AN[15] function. Output buffer only enabled for FCK and ETPU functions.

Figure 229. Pad Configuration Register (SIU_PCR218)

Table 230. SIU_PCR218 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	AN[15]	eQADC	Analog input	I	0b001
ALT1	FCK	eQADC	Free-running clock	O	0b010
ALT2	ETPU_A[29]	eTPU	eTPU channel	O	0b100

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 219 (SIU_PCR219)

Note: *The SIU_PCR219 register is unusual in that it controls pads for two separate device pins: GPIO[219] and MCKO. Please carefully note the pin(s) affected by the bits in this register.*

SIU_BASE+0x1F6

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA[2:0]			OBE(1)	IBE(2)	DSC[1:0]		ODE	HYS	SRC[1:0]		WPE	WPS
W	▨	▨	▨													
Reset	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1

▨ = Unimplemented or Reserved

1. When configured as GPO, the OBE bit should be set to 1.
2. When configured as GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. When configured as GPI, the IBE bit should be set to one.

Figure 230. Pad Configuration Register (SIU_PCR219)

Table 231. SIU_PCR219 field descriptions

Field	Description
0–2	Reserved
3–5 PA[2:0]	Pin assignment Selects the function of a multiplexed pad. This field should always be set to 0b000.
6 OBE(1)	Output buffer enable Enables the pad as an output and drives the output buffer enable signal. 0 Disable output buffer for the pad. 1 Enable output buffer for the pad is enabled. This field affects only the GPIO[219] pin.
7 IBE(1)	Input buffer enable Enables the pad as an input and drives the input buffer enable signal. 0 Disable input buffer for the pad. 1 Enable input buffer for the pad is enabled. For all PCRs where GPIO function is available on the pin, if the pin is configured as an output and the IBE bit is set, the actual value of the pin will be reflected in the corresponding GPDI*x_x* register. Negating the IBE bit when the pin is configured as an output will reduce noise and power consumption. This field affects only the GPIO[219] pin.

Table 231. SIU_PCR219 field descriptions (continued)

Field	Description
8–9 DSC[1:0][2]	Drive strength control Controls the pad drive strength. Drive strength control pertains to pins with the fast I/O pad type. 00 10 pF drive strength 01 20 pF drive strength 10 30 pF drive strength 11 50 pF drive strength This field affects only the MCKO pin.
10 ODE[3]	Open drain output enable Controls output driver configuration for the pads. Either open drain or push/pull driver configurations can be selected. This feature applies to output pins only. 0 Disable open drain for the pad (push/pull driver enabled). 1 Enable open drain for the pad. This field affects both the GPIO[219] and MCKO pins.
11 HYS[3]	Input hysteresis Controls whether hysteresis is enabled for the pad. 0 Disable hysteresis for the pad. 1 Enable hysteresis for the pad. This field affects both the GPIO[219] and MCKO pins.
12–13 SRC[1:0][3]	Slew rate control Controls slew rate for the pad. Slew rate control pertains to pins with slow or medium I/O pad types, and the output signals are driven according to the value of this field. Actual slew rate depends on the pad type and load. Refer to the electrical specifications for this information. 00 Minimum slew rate 01 Medium slew rate 10 Invalid value 11 Maximum slew rate This field affects only the GPIO[219] pin.
14 WPE[4]	Weak pullup/down enable Controls whether the weak pullup/down devices are enabled/disabled for the pad. Pullup/down devices are enabled by default. 0 Disable weak pull device for the pad. 1 Enable weak pull device for the pad. This field affects both the GPIO[219] and MCKO pins.

Table 231. SIU_PCR219 field descriptions (continued)

Field	Description
15 WPS[4]	Weak pullup/down select Controls whether weak pullup or weak pulldown devices are used for the pad when weak pullup/down devices are enabled. The WKPCFG pin determines whether pullup or pulldown devices are enabled during reset. The WPS bit determines whether weak pullup or pulldown devices are used after reset, or for pads in which the WKPCFG pin does not determine the reset weak pullup/down state. 0 Pulldown is enabled for the pad. 1 Pullup is enabled for the pad. This field affects both the GPIO[219] and MCKO pins.

1. In cases where an I/O function is either input-only or output-only the IBE and OBE bits do not need to be set to enable pin I/O.
2. If a pin is configured as an input, the ODE, SRC, and DSC bits do not apply.
3. If a pin is configured as an output, the HYS bit does not apply.
4. When a pin is configured as an output, the weak internal pull up/down is disabled regardless of the WPE or WPS settings in the PCR.

Pad Configuration Register 220 (SIU_PCR220)

SIU_BASE+0x1F8

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	DSC		ODE	HYS	0		WPE	WPS
W	▨	▨	▨	▨	▨	▨	▨	▨					▨	▨		
Reset	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 231. Pad Configuration Register (SIU_PCR220)

Table 232. SIU_PCR220 PA values

Signal	Name	Module	Description	I/O	PA value
Primary	MDO0	Nexus	Nexus message data out	O	—

Pad Configuration Register 221 (SIU_PCR221)

SIU_BASE+0x1FA

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	DSC		ODE	HYS	0		WPE	WPS
W	▨	▨	▨	▨	▨	▨	▨	▨					▨	▨		
Reset	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0

▨ = Unimplemented or Reserved

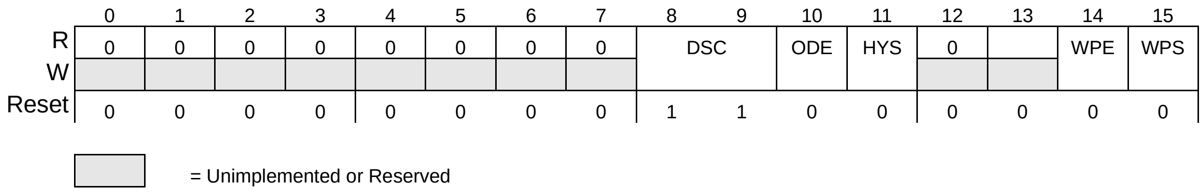
Figure 232. Pad Configuration Register (SIU_PCR221)

Table 233. SIU_PCR221 PA values

Signal	Name	Module	Description	I/O	PA value
Primary	MDO1	Nexus	Nexus message data out	O	—

Pad Configuration Register 222 (SIU_PCR222)

SIU_BASE+0x1FC



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	DSC		ODE	HYS	0		WPE	WPS
W	▨	▨	▨	▨	▨	▨	▨	▨					▨	▨		
Reset	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0

▨ = Unimplemented or Reserved

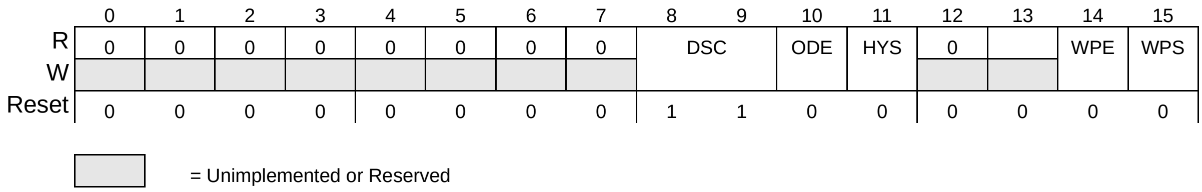
Figure 233. Pad Configuration Register (SIU_PCR222)

Table 234. SIU_PCR222 PA values

Signal	Name	Module	Description	I/O	PA value
Primary	MDO2	Nexus	Nexus message data out	O	—

Pad Configuration Register 223 (SIU_PCR223)

SIU_BASE+0x1FE



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	DSC		ODE	HYS	0		WPE	WPS
W	▨	▨	▨	▨	▨	▨	▨	▨					▨	▨		
Reset	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 234. Pad Configuration Register (SIU_PCR223)

Table 235. SIU_PCR223 PA values

Signal	Name	Module	Description	I/O	PA value
Primary	MDO3	Nexus	Nexus message data out	O	—

Pad Configuration Register 224 (SIU_PCR224)

SIU_BASE+0x200

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0		ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 235. Pad Configuration Register (SIU_PCR224)

Table 236. SIU_PCR224 PA values

Signal	Name	Module	Description	I/O	PA value
Primary	MSEO0	Nexus	Nexus message start/end out	O	—

Pad Configuration Register 225 (SIU_PCR225)

SIU_BASE+0x202

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0		ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 236. Pad Configuration Register (SIU_PCR225)

Table 237. SIU_PCR225 PA values

Signal	Name	Module	Description	I/O	PA value
Primary	MSEO1	Nexus	Nexus message start/end out	O	—

Pad Configuration Register 226 (SIU_PCR226)

SIU_BASE+0x204

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 237. Pad Configuration Register (SIU_PCR226)

Table 238. SIU_PCR226 PA values

Signal	Name	Module	Description	I/O	PA value
Primary	RDY	Nexus	Read/write ready	O	—

Pad Configuration Register 227 (SIU_PCR227)

SIU_BASE+0x206

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	PA	OBE	IBE	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨	▨	▨				▨	▨						
Reset	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	1

▨ = Unimplemented or Reserved

Figure 238. Pad Configuration Register (SIU_PCR227)

Table 239. SIU_PCR227 PA values

Signal	Name	Module	Description	I/O	PA value
Primary	$\overline{\text{EVTO}}$	Nexus	Nexus event out	O	—

Pad Configuration Register 228 (SIU_PCR228)

SIU_BASE+0x208

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	SRC		0	0
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨			▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 239. Pad Configuration Register (SIU_PCR228)

Table 240. SIU_PCR228 PA values

Signal	Name	Module	Description	I/O	PA value
Primary	TDO	JTAG	Nexus event out	O	01

Pad Configuration Register 229 (SIU_PCR229)

The SIU_PCR229 register controls the enabling/disabling and drive strength of the CLKOUT pin. The CLKOUT pin is enabled and disabled by setting and clearing the OBE bit. The CLKOUT pin is enabled during reset.

SIU_BASE+0x20A

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	PA	OBE (1)	0	DSC		ODE	HYS	0	0	WPE	WPS
W	▨	▨	▨	▨	▨			▨					▨	▨		
Reset	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0

▨ = Unimplemented or Reserved

1. CLKOUT pin is enabled and disabled by setting and clearing the OBE bit.

Figure 240. Pad Configuration Register (SIU_PCR229)

Table 241. SIU_PCR229 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	CLKOUT	Clock Generation	System clock output	O	0b001

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 230 (SIU_PCR230)

SIU_BASE+0x20C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0		ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0

▨ = Unimplemented or Reserved

Figure 241. Pad Configuration Register (SIU_PCR230)

Table 242. SIU_PCR230 PA values

Signal	Name	Module	Description	I/O	PA value
Primary	$\overline{\text{RSTOUT}}$	Reset	External Reset Output	O	0b001

Pad Configuration Register 231 (SIU_PCR231)

SIU_BASE+0x20E

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨						
Reset	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1

▨ = Unimplemented or Reserved

Figure 242. Pad Configuration Register (SIU_PCR231)

Table 243. SIU_PCR231 PA values

Signal	Name	Module	Description	I/O	PA value
Primary	$\overline{\text{EVTI}}$	Nexus	Nexus event in	I	—

Pad Configuration Register 232 (SIU_PCR232)

SIU_BASE+0x210

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	SRC		0	0
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨			▨	▨
Reset	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 243. Pad Configuration Register (SIU_PCR232)

Table 244. SIU_PCR232 PA values

Signal	Name	Module	Description	I/O	PA value
Primary	TDI	JTAG	Nexus event in	I	01

Pad Configuration Register 244 (SIU_PCR244)

SIU_BASE+0x228

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	PA	OBE (1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨	▨	▨				▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

▨ = Unimplemented or Reserved

1. When configured as GPO, the OBE bit should be set to one.
2. When configured as GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. When configured as GPI, the IBE bit should be set to one.

Figure 244. Pad Configuration Register (SIU_PCR244)

Table 245. SIU_PCR244 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	SCI_C_TX	eSCI	eSCI C transmit	O	0b01
GPIO	GPIO[244]	SIU	GPIO	I/O	0b00

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 245 (SIU_PCR245)

SIU_BASE+0x22A

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	PA	OBE(1)	IBE(2)	0	0	ODE	HYS	SRC		WPE	WPS
W	▨	▨	▨	▨	▨				▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

▨ = Unimplemented or Reserved

1. When configured as GPO, the OBE bit should be set to one.
2. When configured as GPO, the IBE bit may be set to one to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to zero reduces power consumption. When configured as GPI, the IBE bit should be set to one.

Figure 245. Pad Configuration Register (SIU_PCR245)

Table 246. SIU_PCR245 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	SCI_C_RX	eSCI	eSCI C receive	I	0b01
GPIO	GPIO[245]	SIU	GPIO	I/O	0b00

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 336 (SIU_PCR336)

SIU_BASE+0x2E0

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	DSC		ODE	HYS	0	0	WPE	WPS
W	▨	▨	▨	▨	▨	▨	▨	▨					▨	▨		
Reset	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 246. Pad Configuration Register (SIU_PCR336)

Table 247. SIU_PCR336 PA values

Signal	Name	Module	Description	I/O	PA value
Primary	CAL_CS0	Calibration bus	Calibration chip select	O	—

Pad Configuration Register 338 (SIU_PCR338)

SIU_BASE+0x2E4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA			0	0	DSC		ODE	HYS	0	0	WPE	WPS
W	▨	▨	▨				▨	▨					▨	▨		
Reset	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 247. Pad Configuration Register (SIU_PCR338)

Table 248. SIU_PCR338 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	CAL_CS2	Calibration bus	Calibration chip select	O	001
ALT1	CAL_ADDR[10]	Calibration bus	Calibration address bus	I/O	010
ALT2	CAL_WE[2]/BE[2]	Calibration bus	Calibration write/byte enable	O	100

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 339 (SIU_PCR339)

SIU_BASE+0x2E6

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA			0	0	DSC		ODE	HYS	0	0	WPE	WPS
W	▨	▨	▨				▨	▨					▨	▨		
Reset	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 248. Pad Configuration Register (SIU_PCR339)

Table 249. SIU_PCR339 PA values

Signal	Name	Module	Description	I/O(1),(2)	PA value
Primary	CAL_CS3	Calibration bus	Calibration chip select	O	001
ALT1	CAL_ADDR[11]	Calibration bus	Calibration address bus	I/O	010
ALT2	CAL_WE[3]/BE[3]	Calibration bus	Calibration write/byte enable	O	100

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 340 (SIU_PCR340)

Note: Unlike other pad configuration registers (PCRs), which control the function and electrical characteristics of one pin, this register controls the function and electrical characteristics for a group of pins on the Calibration bus.

SIU_BASE+0x2E8

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		0	0	DSC		ODE	HYS	0	0	WPE	WPS
W	▨	▨	▨	▨			▨	▨					▨	▨		
Reset	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 249. Pad Configuration Register (SIU_PCR340)

Table 250. SIU_PCR340 PA values

Signal	Name	Module	Description	I/O[1],[2]	PA value
Primary	CAL_ADDR[12:15]	Calibration bus	Calibration address bus	I/O	001
ALT1	CAL_WE[2]/BE[2]		Calibration write/byte enable	O	010
	CAL_WE3]/BE[3]		Calibration write/byte enable	O	
	CAL_DATA[31]		Calibration data bus	I/O	
	CAL_ALE		Calibration address latch enable	O	

1. In cases where an I/O function can be either an input or an output, I/O direction is specified using the IBE and OBE bits. Set IBE = 1 for input or OBE = 1 for output.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.

Pad Configuration Register 341 (SIU_PCR341)

Note: Unlike other pad configuration registers (PCRs), which control the function and electrical characteristics of one pin, this register controls the electrical characteristics for a group of pins on the Calibration bus.

SIU_BASE+0x2EA

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	DSC		ODE	HYS	0	0	WPE	WPS
W	▨	▨	▨	▨	▨	▨	▨	▨					▨	▨		
Reset	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1

▨ = Unimplemented or Reserved

Figure 250. Pad Configuration Register (SIU_PCR341)

Table 251. SIU_PCR341PA values

Signal	Name	Module	Description	I/O	PA value
Primary	CAL_DATA[0:15]	Calibration bus	Calibration data bus	I/O	—

Pad Configuration Register 342 (SIU_PCR342)

Note: Unlike other pad configuration registers (PCRs), which control the function and electrical characteristics of one pin, this register controls the function and electrical characteristics for a group of pins on the Calibration bus.

SIU_BASE+0x2EC

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	DSC		ODE	HYS	0	0	WPE	WPS
W	▨	▨	▨	▨	▨	▨	▨	▨					▨	▨		
Reset	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 251. Pad Configuration Register (SIU_PCR342)

Table 252. SIU_PCR342 PA values

Signal	Name	Module	Description	I/O	PA value
Primary	CAL_RD_WR	Calibration bus	Calibration data bus	I/O	—
	CAL_WE[0]/BE[0]		Calibration write/byte enable		
	CAL_WE[1]/BE[1]		Calibration write/byte enable		
	CAL_OE		Calibration output enable		

Pad Configuration Register 343 (SIU_PCR343)

SIU_BASE+0x2EE

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		0	0	DSC		ODE	HYS	0	0	WPE	WPS
W	▨	▨	▨	▨			▨	▨					▨	▨		
Reset	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 252. Pad Configuration Register (SIU_PCR343)

Table 253. SIU_PCR343 PA values

Signal	Name	Module	Description	I/O	PA value
Primary	CAL_TS	Calibration bus	Calibration transfer start	O	01
ALT1	CAL_ALE	Calibration bus	Address Latch Enable	O	10

Pad Configuration Register 345 (SIU_PCR345)

Note: Unlike other pad configuration registers (PCRs), which control the function and electrical characteristics of one pin, this register controls the function and electrical characteristics for a group of pins on the Calibration bus.

SIU_BASE+0x2F2

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		0	0	DSC		ODE	HYS	0	0	WPE	WPS
W	▨	▨	▨	▨			▨	▨					▨	▨		
Reset	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 253. Pad Configuration Register (SIU_PCR345)

Table 254. SIU_PCR345 PA values

Signal	Name	Module	Description	I/O	PA value
Primary	CAL_ADDR[16:30]	Calibration bus	Calibration address bus	I/O	01
ALT1	CAL_DATA[16:30]	Calibration bus	Calibration data bus	I/O	10

Pad Configuration Register 350 – 381 (SIU_PCR350 – SIU_PCR381)

The SIU_PCR350 – SIU_PCR381 registers control the muxing of the signals to the DSPI. PA field values are shown in *Table 255*.

SIU_BASE+0x2Fc – SIU_BASE+0x33a (32)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		0	0	0	0	0	0	0	0	0	0
W	▨	▨	▨	▨			▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 254. Pad Configuration Register 350 – 381 (SIU_PCR350 – SIU_PCR381)

Table 255. SIU_PCR350 – SIU_PCR381 DSPI muxing

Register	Address offset from SIU_BASE	DSPI serialize signal name	PA value			
			0b001	0b100	0b010	0b000
SIU_PCR350	0x2FC	DSPI_B -DSI0	ETPU_A_23	—	EMIOS_11	GPIO350
SIU_PCR351	0x2FE	DSPI_B -DSI1	ETPU_A_22	—	EMIOS_10	GPIO351
SIU_PCR352	0x300	DSPI_B -DSI2	ETPU_A_21	—	EMIOS_9	GPIO352
SIU_PCR353	0x302	DSPI_B -DSI3	ETPU_A_20	—	EMIOS_8	GPIO353
SIU_PCR354	0x304	DSPI_B -DSI4	ETPU_A_19	—	EMIOS_6	GPIO354
SIU_PCR355	0x306	DSPI_B -DSI5	ETPU_A_18	—	EMIOS_5	GPIO355
SIU_PCR356	0x308	DSPI_B -DSI6	ETPU_A_17	—	EMIOS_4	GPIO356
SIU_PCR357	0x30A	DSPI_B -DSI7	ETPU_A_16	—	EMIOS_3	GPIO357
SIU_PCR358	0x30C	DSPI_B -DSI8	ETPU_A_29	—	EMIOS_2	GPIO358
SIU_PCR359	0x30E	DSPI_B -DSI9	ETPU_A_28	—	EMIOS_1	GPIO359

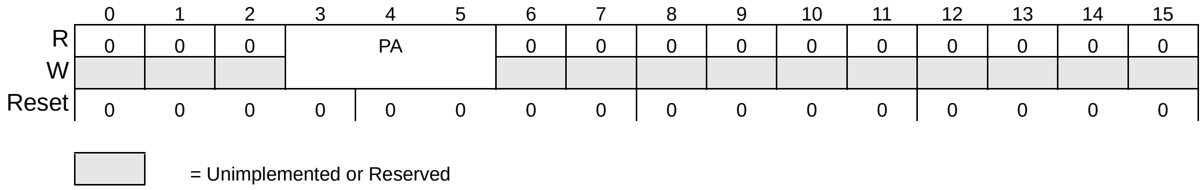
Table 255. SIU_PCR350 – SIU_PCR381 DSPI muxing (continued)

Register	Address offset from SIU_BASE	DSPI serialize signal name	PA value			
			0b001	0b100	0b010	0b000
SIU_PCR360	0x310	DSPI_B -DSI10	ETPU_A_27	—	EMIOS_0	GPIO360
SIU_PCR361	0x312	DSPI_B -DSI11	ETPU_A_26	—	EMIOS_23	GPIO361
SIU_PCR362	0x314	DSPI_B -DSI12	ETPU_A_25	—	EMIOS_15	GPIO362
SIU_PCR363	0x316	DSPI_B -DSI13	ETPU_A_24	—	EMIOS_14	GPIO363
SIU_PCR364	0x318	DSPI_B -DSI14	ETPU_A_31	—	EMIOS_13	GPIO364
SIU_PCR365	0x31A	DSPI_B -DSI15	ETPU_A_30	—	EMIOS_12	GPIO365
SIU_PCR366	0x31C	DSPI_B -DSI16	ETPU_A_12	—	EMIOS_23	GPIO366
SIU_PCR367	0x31E	DSPI_B -DSI17	ETPU_A_13	—	EMIOS_15	GPIO367
SIU_PCR368	0x320	DSPI_B -DSI18	ETPU_A_14	—	EMIOS_14	GPIO368
SIU_PCR369	0x322	DSPI_B -DSI19	ETPU_A_15	—	EMIOS_13	GPIO369
SIU_PCR370	0x324	DSPI_B -DSI20	ETPU_A_0	—	EMIOS_12	GPIO370
SIU_PCR371	0x326	DSPI_B -DSI21	ETPU_A_1	—	EMIOS_11	GPIO371
SIU_PCR372	0x328	DSPI_B -DSI22	ETPU_A_2	—	EMIOS_10	GPIO372
SIU_PCR373	0x32A	DSPI_B -DSI23	ETPU_A_3	—	EMIOS_9	GPIO373
SIU_PCR374	0x32C	DSPI_B -DSI24	ETPU_A_4	—	EMIOS_8	GPIO374
SIU_PCR375	0x32E	DSPI_B -DSI25	ETPU_A_5	—	EMIOS_6	GPIO375
SIU_PCR376	0x330	DSPI_B -DSI26	ETPU_A_6	—	EMIOS_5	GPIO376
SIU_PCR377	0x332	DSPI_B -DSI27	ETPU_A_7	—	EMIOS_4	GPIO377
SIU_PCR378	0x334	DSPI_B -DSI28	ETPU_A_8	—	EMIOS_3	GPIO378
SIU_PCR379	0x336	DSPI_B -DSI29	ETPU_A_9	—	EMIOS_2	GPIO379
SIU_PCR380	0x338	DSPI_B -DSI30	ETPU_A_10	—	EMIOS_1	GPIO380
SIU_PCR381	0x33A	DSPI_B -DSI31	ETPU_A_11	—	EMIOS_0	GPIO381

Pad Configuration Register 382 – 389 (SIU_PCR382 – SIU_PCR389)

The SIU_PCR382 – SIU_PCR389 registers control the muxing of the signals to the DSPI. PA field values are shown in *Table 256*.

SIU_BASE+0x33C – SIU_BASE+0x34A (8)



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0		PA		0	0	0	0	0	0	0	0	0	0
W	▨	▨	▨				▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 255. Pad Configuration Register 382 – 389 (SIU_PCR382 – SIU_PCR389)

Table 256. SIU_PCR382 – SIU_PCR389 DSPI muxing

Register	Address offset from SIU_BASE	DSPI serialize signal name	PA value			
			0b001	0b100	0b010	0b000
SIU_PCR382	0x33C	DSPI_C -DSI0	ETPU_A_12	EMIOS_7	EMIOS_12	GPIO382
SIU_PCR 383	0x33E	DSPI_C -DSI1	ETPU_A_13	EMIOS_16	EMIOS_13	GPIO383
SIU_PCR 384	0x340	DSPI_C -DSI2	ETPU_A_14	EMIOS_17	EMIOS_14	GPIO384
SIU_PCR 385	0x342	DSPI_C -DSI3	ETPU_A_15	EMIOS_18	EMIOS_15	GPIO385
SIU_PCR 386	0x344	DSPI_C -DSI4	ETPU_A_0	EMIOS_19	EMIOS_23	GPIO386
SIU_PCR 387	0x346	DSPI_C -DSI5	ETPU_A_1	EMIOS_20	EMIOS_0	GPIO387
SIU_PCR 388	0x348	DSPI_C -DSI6	ETPU_A_2	EMIOS_21	EMIOS_1	GPIO388
SIU_PCR 389	0x34A	DSPI_C -DSI7	ETPU_A_3	EMIOS_22	EMIOS_2	GPIO389

Pad Configuration Register 390 – 413 (SIU_PCR390 – SIU_PCR413)

The SIU_PCR390 – SIU_PCR413 registers control the muxing of the signals to the DSPI. PA field values are shown in *Table 257*.

SIU_BASE+0x34C – SIU_BASE+0x37A (24)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		0	0	0	0	0	0	0	0	0	0
W	▨	▨	▨	▨			▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 256. Pad Configuration Register 390 – 413 (SIU_PCR390 – SIU_PCR413)

Table 257. SIU_PCR390 – SIU_PCR413 DSPI muxing

Register	Address offset from SIU_BASE	DSPI serialize signal name	PA value			
			0b001	0b100	0b010	0b000
SIU_PCR 390	0x34C	DSPI_C -DSI8	ETPU_A_4	—	EMIOS_3	GPIO390
SIU_PCR 391	0x34E	DSPI_C -DSI9	ETPU_A_5	—	EMIOS_4	GPIO391
SIU_PCR 392	0x350	DSPI_C -DSI10	ETPU_A_6	—	EMIOS_5	GPIO392
SIU_PCR 393	0x352	DSPI_C -DSI11	ETPU_A_7	—	EMIOS_6	GPIO393
SIU_PCR 394	0x354	DSPI_C -DSI12	ETPU_A_8	—	EMIOS_8	GPIO394
SIU_PCR 395	0x356	DSPI_C -DSI13	ETPU_A_9	—	EMIOS_9	GPIO395
SIU_PCR 396	0x358	DSPI_C -DSI14	ETPU_A_10	—	EMIOS_10	GPIO396

Table 257. SIU_PCR390 – SIU_PCR413 DSPI muxing (continued)

Register	Address offset from SIU_BASE	DSPI serialize signal name	PA value			
			0b001	0b100	0b010	0b000
SIU_PCR 397	0x35A	DSPI_C - DSI15	ETPU_A_11	—	EMIOS_11	GPIO397
SIU_PCR 398	0x35C	DSPI_C - DSI16	ETPU_A_23	—	EMIOS_0	GPIO398
SIU_PCR 399	0x35E	DSPI_C - DSI17	ETPU_A_22	—	EMIOS_1	GPIO399
SIU_PCR 400	0x360	DSPI_C - DSI18	ETPU_A_21	—	EMIOS_2	GPIO400
SIU_PCR 401	0x362	DSPI_C - DSI19	ETPU_A_20	—	EMIOS_3	GPIO401
SIU_PCR 402	0x364	DSPI_C - DSI20	ETPU_A_19	—	EMIOS_4	GPIO402
SIU_PCR 403	0x366	DSPI_C - DSI21	ETPU_A_18	—	EMIOS_5	GPIO403
SIU_PCR 404	0x368	DSPI_C - DSI22	ETPU_A_17	—	EMIOS_6	GPIO404
SIU_PCR 405	0x36A	DSPI_C - DSI23	ETPU_A_16	—	EMIOS_8	GPIO405
SIU_PCR 406	0x36C	DSPI_C - DSI24	ETPU_A_29	—	EMIOS_9	GPIO406
SIU_PCR 407	0x36E	DSPI_C - DSI25	ETPU_A_28	—	EMIOS_10	GPIO407
SIU_PCR 408	0x370	DSPI_C - DSI26	ETPU_A_27	—	EMIOS_11	GPIO408
SIU_PCR 409	0x372	DSPI_C - DSI27	ETPU_A_26	—	EMIOS_12	GPIO409
SIU_PCR 410	0x374	DSPI_C - DSI28	ETPU_A_25	—	EMIOS_13	GPIO410
SIU_PCR 411	0x376	DSPI_C - DSI29	ETPU_A_24	—	EMIOS_14	GPIO411
SIU_PCR 412	0x378	DSPI_C - DSI30	ETPU_A_31	—	EMIOS_15	GPIO412
SIU_PCR 413	0x37A	DSPI_C - DSI31	ETPU_A_30	—	EMIOS_23	GPIO413

15.6.16 GPIO Pin Data Output Registers (SIU_GPDO0_3 – SIU_GPDO412_413)

The SIU_GPDO*x_x* registers are written to by software to drive data out on the external GPIO pin. Each byte of a register drives a single external GPIO pin, which allows the state of the pin to be controlled independently from other GPIO pins. Writes to the SIU_GPDO*x_x* registers have no effect on pin states if the pins are configured as inputs by the associated Pad Configuration Registers. The SIU_GPDO*x_x* register values are automatically driven to the GPIO pins without software update if the direction of the GPIO pins is changed from input to output.

Writes to the SIU_GPDO*x*_*x* registers have no effect on the state of the corresponding pins when the pins are configured for their primary function by the corresponding PCR.

The definition of the SIU_GPDO*x*_*x* registers is given in *Figure 257* and *Figure 258*. Each of the PDO bits corresponds to the pin with the same GPIO pin number. For example, PDO0 is the pin data output bit for the $\overline{\text{CS}}$[0]_ADDR[8]_GPIO[0] pin, and PDO213 is the pin data output bit for the WKPCFG_GPIO[213] pin. Gaps exist in this memory space where the pin is not available in the package.

SIU_BASE + 0x600

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	PDO 0	0	0	0	0	0	0	0	PDO 1
W	▨	▨	▨	▨	▨	▨	▨		▨	▨	▨	▨	▨	▨	▨	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	PDO 2	0	0	0	0	0	0	0	PDO 3
W	▨	▨	▨	▨	▨	▨	▨		▨	▨	▨	▨	▨	▨	▨	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 257. GPIO Pin Data Out Register 0 – 3 (SIU_GPDO0 – SIU_GPDO3)

SSIU_BASE + 0x79D

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	PDO 412	0	0	0	0	0	0	0	PDO 413
W	▨	▨	▨	▨	▨	▨	▨		▨	▨	▨	▨	▨	▨	▨	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 258. GPIO Pin Data Out Register 412 – 413 (SIU_GPDO410 – SIU_GPDO413)

Table 258. SIU_GPDOx_x field description

Field	Description
PDOx	Pin Data Out. This bit stores the data to be driven out on the external GPIO pin associated with the register. 0 VOL is driven on the external GPIO pin when the pin is configured as an output. 1 VOH is driven on the external GPIO pin when the pin is configured as an output.

15.6.17 GPIO Pin Data Input Registers (SIU_GPDI0_3 – SIU_GPDI_232)

The SIU_GPDI*x_x* registers are read-only registers that allow software to read the input state of an external GPIO pin. Each byte of a register represents the input state of a single external GPIO pin. If the GPIO pin is configured as an output, and the input buffer enable (IBE) bit is set in the associated Pad Configuration Register, the SIU_GPDI*x_x* register reflects the actual state of the output pin.

The definition of the SIU_GPDI*x_x* registers is given in *Figure 259* and *Figure 260*. Each of the GPDI bits corresponds to the pin with the same GPIO pin number. For example, GPDI0 is the pin data input bit for the $\overline{\text{CS}}$[0]_GPIO[0] pin, and PDI213 is the pin data input bit for the WKPCFG_GPIO[213] pin. Gaps exist in this memory space where the pin is not available in the package.

SIU_BASE + 0x800

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	PDI 0	0	0	0	0	0	0	0	PDI 1
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	U	0	0	0	0	0	0	0	U

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	PDI 2	0	0	0	0	0	0	0	PDI 3
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	U	0	0	0	0	0	0	0	U

▨ = Unimplemented or Reserved

Figure 259. GPIO Pin Data In Register 0 – 3 (SIU_GPDI0 – SIU_GPDI3)

SIU_BASE + 0x8EA

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	PDI 230	0	0	0	0	0	0	0	PDI 231
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	U	0	0	0	0	0	0	0	U

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	PDI 232	0	0	0	0	0	0	0	0
W	▨	▨	▨	▨	▨	▨	▨		▨	▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	U	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 260. GPIO Pin Data In Register 230 – 232 (SIU_GPDI230 – SIU_GPDI232)

Table 259. SIU_GPDIx_x field description

Field	Description
PDIx	Pin Data In. This bit stores the input state on the external GPIO pin associated with the register. 0 Signal on pin is less than or equal to VIL. 1 Signal on pin is greater than or equal to VIH.

15.6.18 eQADC Trigger Input Select Register (SIU_ETISR[g])

The eQADC Trigger Input Select Register (SIU_ETISR) selects the source for the eQADC trigger inputs.

The fields in this register operate in conjunction with the corresponding fields in the SIU_ISEL3 register. Each field in the SIU_ETISR offers direct selection of three signals to be used as a trigger to a eQADC CFIFO queue. The TSEL5 field is used to select the trigger source for eQADC CFIFO5, the TSEL4 field selects the trigger source for eQADC CFIFO4, and so on. Additionally, each SIU_ETISR field offers selection among a group of signals using the corresponding field in the SIU_ISEL3 register.

Figure 261 illustrates the available trigger selections for eQADC CFIFO5.

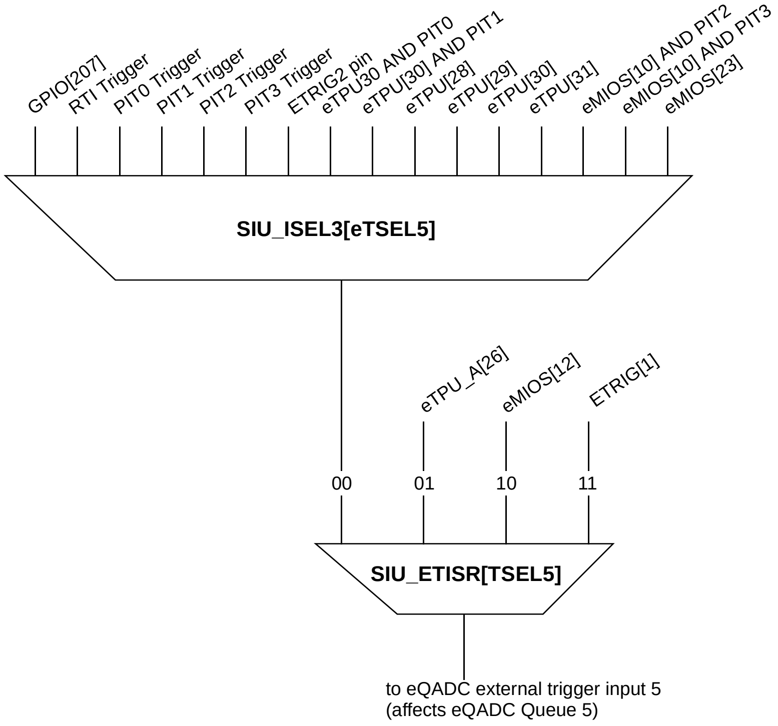


Figure 261. Trigger selection for eQADC CFIFO queue 5

---

[g] The ETISR is sometimes referred to as ISEL0

As shown in the above figure, the TSEL5 field can be used to select the eTPU_A[26], eMIOS[12] or GPIO[207] signal or, by assigning a value of 0b00 to the TSEL5 field, a variety of other signals can be selected using the eTSEL5 field of the SIU_ISEL3 register.

SIU_BASE+0x900

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TSEL5		TSEL4		TSEL3		TSEL2		TSEL1		TSEL0		0	0	0	0
W													▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 262. eQADC Trigger Input Select Register (SIU_ETISR)

Table 260. SIU_ETISR field description

Field	Description
0:1 TSEL5	eQADC Trigger Input Select 5. The eQADC trigger 5 input is as follows: 00 eTSEL5 (described in SIU_ISEL3) 01 eTPU_A[26] channel 10 eMIOS[12] channel 11 ETRIG[1] pin
2:3 TSEL4	eQADC Trigger Input Select 4. The eQADC trigger 4 input is as follows: 00 eTSEL4 (described in SIU_ISEL3) 01 eTPU_A[27] channel 10 eMIOS[13] channel 11 ETRIG[0] pin
4:5 TSEL3	eQADC Trigger Input Select 3. The eQADC trigger 3 input is as follows: 00 eTSEL3 (described in SIU_ISEL3) 01 eTPU_A[28] channel 10 eMIOS[14] channel 11 ETRIG[1] pin
6:7 TSEL2	eQADC Trigger Input Select 2. The eQADC trigger 2 input is as follows: 00 eTSEL2 (described in SIU_ISEL3) 01 eTPU_A[29] channel 10 eMIOS[15] channel 11 ETRIG[0] pin
8:9 TSEL1	eQADC Trigger Input Select 1. The eQADC trigger 1 input is as follows: 00 eTSEL1 (described in SIU_ISEL3) 01 eTPU_A[31] channel 10 eMIOS[11] channel 11 ETRIG[1] pin

Table 260. SIU_ETISR field description (continued)

Field	Description
10:11 TSEL0	eQADC Trigger Input Select 0. The eQADC trigger 0 input is as follows: 00 eTSEL0 (described in SIU_ISEL3) 01 eTPU_A[30] channel 10 eMIOS[10] channel 11 ETRIG[0] pin
12:31	Reserved

15.6.19 External IRQ Input Select Register (SIU_EIISR[h])

The EIISR selects the source for the external interrupt/DMA inputs.

SIU_BASE+0x904

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	ESEL15		ESEL14		ESEL13		ESEL12		ESEL11		ESEL10		ESEL9		ESEL8	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	ESEL7		ESEL6		ESEL5		ESEL4		ESEL3		ESEL2		ESEL1		ESEL0	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 263. External IRQ Input Select Register (SIU_EIISR)

Table 261. SIU_EIISR field description

Field	Description
0:1 ESEL15	External IRQ Input Select 15. IRQ[15] input is specified by ESEL15 as follows: 00 IRQ[15] pin 01 DSPI_B[15] deserialized output 10 DSPI_C[0] deserialized output 11 DSPI_D[1] deserialized output
2:3 ESEL14	External IRQ Input Select 14. IRQ[14] input is specified by ESEL14 as follows: 00 IRQ[14] pin 01 DSPI_B[14] deserialized output 10 DSPI_C[15] deserialized output 11 DSPI_D[0] deserialized output

---

[h] The EIISR is sometimes referred to as ISEL1

Table 261. SIU_EIISR field description (continued)

Field	Description
4:5 ESEL13	External IRQ Input Select 13. IRQ[13] input is specified by ESEL13 as follows: 00 IRQ[13] pin 01 DSPI_B[13] deserialized output 10 DSPI_C[14] deserialized output 11 DSPI_D[15] deserialized output
6:7 ESEL12	External IRQ Input Select 12. IRQ[12] input is specified by ESEL12 as follows: 00 IRQ[12] pin 01 DSPI_B[12] deserialized output 10 DSPI_C[13] deserialized output 11 DSPI_D[14] deserialized output
8:9 ESEL11	External IRQ Input Select 11. IRQ[11] input is specified by ESEL11 as follows: 00 IRQ[11] pin 01 DSPI_B[11] deserialized output 10 DSPI_C[12] deserialized output 11 DSPI_D[13] deserialized output
10:11 ESEL10	External IRQ Input Select 10. IRQ[10] input is specified by ESEL10 as follows: 00 IRQ[10] pin 01 DSPI_B[10] deserialized output 10 DSPI_C[11] deserialized output 11 DSPI_D[12] deserialized output
12:13 ESEL9	External IRQ Input Select 9. IRQ[9] input is specified by ESEL9 as follows: 00 IRQ[9] pin 01 DSPI_B[9] deserialized output 10 DSPI_C[10] deserialized output 11 DSPI_D[11] deserialized output
14:15 ESEL8	External IRQ Input Select 8. IRQ[8] input is specified by ESEL8 as follows: 00 IRQ[8] pin 01 DSPI_B[8] deserialized output 10 DSPI_C[9] deserialized output 11 DSPI_D[10] deserialized output
16:17 ESEL7	External IRQ Input Select 7. IRQ[7] input is specified by ESEL7 as follows: 00 IRQ[7] pin 01 DSPI_B[7] deserialized output 10 DSPI_C[8] deserialized output 11 DSPI_D[9] deserialized output
18:19 ESEL6	External IRQ Input Select 6. IRQ[6] is multiplexed on the TCRCLK_B pin, which is not available in any of the SPC564A70 packages. IRQ[6] input is specified by ESEL6 as follows: 00 IRQ[6] pin 01 DSPI_B[6] deserialized output 10 DSPI_C[7] deserialized output 11 DSPI_D[8] deserialized output
20:21 ESEL5	External IRQ Input Select 5. IRQ[5] input is specified by ESEL5 as follows: 00 IRQ[5] pin 01 DSPI_B[5] deserialized output 10 DSPI_C[6] deserialized output 11 DSPI_D[7] deserialized output

Table 261. SIU_EIISR field description (continued)

Field	Description
22:23 ESEL4	External IRQ Input Select 4. IRQ[4] input is specified by ESEL4 as follows: 00 IRQ[4] pin 01 DSPI_B[4] deserialized output 10 DSPI_C[5] deserialized output 11 DSPI_D[6] deserialized output
24:25 ESEL3	External IRQ Input Select 3. IRQ[3] input is specified by ESEL3 as follows: 00 IRQ[3] pin 01 DSPI_B[3] deserialized output 10 DSPI_C[4] deserialized output 11 DSPI_D[5] deserialized output
26:27 ESEL2	External IRQ Input Select 2. IRQ[2] input is specified by ESEL2 as follows: 00 IRQ[2] pin 01 DSPI_B[2] deserialized output 10 DSPI_C[3] deserialized output 11 DSPI_D[4] deserialized output
28:29 ESEL1	External IRQ Input Select 1. IRQ[1] input is specified by ESEL1 as follows: 00 IRQ[1] pin 01 DSPI_B[1] deserialized output 10 DSPI_C[2] deserialized output 11 eMIOS[15]
30:31 ESEL0	External IRQ Input Select 0. IRQ[0] input is specified by ESEL0 as follows: 00 IRQ[0] pin 01 DSPI_B[0] deserialized output 10 DSPI_C[1] deserialized output 11 eMIOS[14]

15.6.20 DSPI Input Select Register (SIU_DISR[i])

The DISR specifies the source of each DSPI data input, slave select, clock input, and trigger input to allow serial and parallel chaining of the DSPI blocks.

---

[i] The DISR is sometimes referred to as ISEL2

SIU_BASE+0x908

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	SINSELB		SSSELB		SCKSELB		TRIGSELB	
W	▨	▨	▨	▨	▨	▨	▨	▨								
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SINSELC		SSSELC		SCKSELC		TRIGSELC		SINSELD		SSSELD		SCKSELD		TRIGSELD	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 264. DSPI Input Select Register (SIU_DISR)

Table 262. SIU_DISR field description

Field	Description
0:7	Reserved
8:9 SINSELB	DSPI_B Data Input Select. The source of the data input of DSPI_B is specified by SINSELB as follows: 00 DSPI_B_SIN pin 01 Reserved 10 DSPI_C_SOUT 11 DSPI_D_SOUT
10:11 SSSELB	DSPI_B Slave Select Input Select. The source of the slave select input of DSPI_B is specified by SSSELB as follows: 00 DSPI_B_PCS[0] pin 01 Reserved 10 DSPI_C_PCS[0] (Master) 11 DSPI_D_PCS[0] (Master)
12:13 SCKSELB	DSPI_B Clock Input Select. The source of the clock input of DSPI_B when in slave mode is specified by SCKSELB as follows: 00 DSPI_B_SCK pin 01 Reserved 10 DSPI_C_SCK (Master) 11 DSPI_D_SCK (Master)
14:15 TRIGSELB	DSPI_B Trigger Input Select. The source of the trigger input of DSPI_B for master or slave mode is specified by TRIGSELB as follows: 00 Reserved 01 Reserved 10 DSPI_C_PCS[4] 11 DSPI_D_PCS[4]
16:17 SINSELC	DSPI_C Data Input Select. The source of the data input of DSPI_C is specified by SINSELC as follows: 00 DSPI_C_SIN pin 01 Reserved 10 DSPI_B_SOUT 11 DSPI_D_SOUT

Table 262. SIU_DISR field description (continued)

Field	Description
18:19 SSSELC	DSPI_C Slave Select Input Select. The source of the slave select input of DSPI_C is specified by SSSELC as follows: 00 DSPI_C_PCS[0] pin 01 Reserved 10 DSPI_B_PCS[0] (Master) 11 DSPI_D_PCS[0] (Master)
20:21 SCKSELC	DSPI_C Clock Input Select. The source of the clock input of DSPI_C when in slave mode is specified by SCKSELC as follows: 00 DSPI_C_SCK pin 01 Reserved 10 DSPI_B_SCK (Master) 11 DSPI_D_SCK (Master)
22:23 TRIGSELC	DSPI_C Trigger Input Select. The source of the trigger input of DSPI_C for master or slave mode is specified by TRIGSELC as follows: 00 Reserved 01 Reserved 10 DSPI_B_PCS[4] 11 DSPI_D_PCS[4]
24:25 SINSELD	DSPI_D Data Input Select. The source of the data input of DSPI_D is specified by SINSELD as follows: 00 DSPI_D_SIN pin 01 Reserved 10 DSPI_B_SOUT 11 DSPI_C_SOUT
26:27 SSSELD	DSPI_D Slave Select Input Select. The source of the slave select input of DSPI_D is specified by SSSELD as follows: 00 DSPI_D_PCS[0] pin 01 Reserved 10 DSPI_B_PCS[0] (Master) 11 DSPI_C_PCS[0] (Master)
28:29 SCKSELD	DSPI_D Clock Input Select. The source of the clock input of DSPI_D when in slave mode is specified by SCKSELD as follows: 00 DSPI_D_SCK pin 01 Reserved 10 DSPI_B_SCK (Master) 11 DSPI_C_SCK (Master)
30:31 TRIGSELD	DSPI_D Trigger Input Select. The source of the trigger input of DSPI_D for master or slave mode is specified by TRIGSELD as follows: 00 Reserved 01 Reserved 10 DSPI_B_PCS[4] 11 DSPI_C_PCS[4]

15.6.21 IMUX Select Register 3 (SIU_ISEL3)

The SIU_ISEL 3 register selects the source for the external eQADC trigger inputs.

SIU_BASE+0x90C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	eTSEL5					eTSEL4					eTSEL3			
W	▨	▨														
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	eTSEL 3	eTSEL2					eTSEL1					eTSEL0				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 265. IMUX Select Register 3 (SIU_ISEL3)

For options 0b01000, 0b01001, 0b10100, 0b10101 for each queue, two trigger sources are logically ANDed together. The intention is that the PIT provides the regular cyclic trigger, while the eTPU or eMIOS channels are used to ‘gate’ that cyclic trigger. This way, the ADC can be commanded to make regular samples but only during a given time or angle window.

Table 263. eQADC queue0 enhanced trigger selection

eTSEL0					eQADC enhanced trigger input
0	0	0	0	0	GPIO206 (eTRIG0)
0	0	0	0	1	RTI Trigger
0	0	0	1	0	PIT0 Trigger
0	0	0	1	1	PIT1 Trigger
0	0	1	0	0	PIT2 Trigger
0	0	1	0	1	PIT3 Trigger
0	0	1	1	0	Reserved
0	0	1	1	1	BOOTCFG[1] (eTRIG3)
0	1	0	0	0	eTPU30 AND PIT0
0	1	0	0	1	eTPU30 AND PIT1
0	1	0	1	0	Reserved
0	1	0	1	1	Reserved
0	1	1	0	0	eTPU28
0	1	1	0	1	eTPU29
0	1	1	1	0	eTPU30
0	1	1	1	1	eTPU31
1	0	0	0	0	Reserved
1	0	0	0	1	Reserved
1	0	0	1	0	Reserved

Table 263. eQADC queue0 enhanced trigger selection (continued)

eTSEL0					eQADC enhanced trigger input
1	0	0	1	1	Reserved
1	0	1	0	0	eMIOS10 AND PIT2
1	0	1	0	1	eMIOS10 AND PIT3
1	0	1	1	0	Reserved
1	0	1	1	1	Reserved
1	1	0	0	0	Reserved
1	1	0	0	1	Reserved
1	1	0	1	0	Reserved
1	1	0	1	1	Reserved
1	1	1	0	0	Reserved
1	1	1	0	1	Reserved
1	1	1	1	0	Reserved
1	1	1	1	1	eMIOS23

Table 264. eQADC queue1 enhanced trigger selection

eTSEL1					eQADC enhanced trigger input
0	0	0	0	0	GPIO207 (eTRIG1)
0	0	0	0	1	RTI Trigger
0	0	0	1	0	PIT0 Trigger
0	0	0	1	1	PIT1 Trigger
0	0	1	0	0	PIT2 Trigger
0	0	1	0	1	PIT3 Trigger
0	0	1	1	0	Reserved
0	0	1	1	1	PLLREF (eTRIG2)
0	1	0	0	0	eTPU31 AND PIT0
0	1	0	0	1	eTPU31 AND PIT1
0	1	0	1	0	Reserved
0	1	0	1	1	Reserved
0	1	1	0	0	eTPU28
0	1	1	0	1	eTPU29
0	1	1	1	0	eTPU30
0	1	1	1	1	eTPU31
1	0	0	0	0	Reserved
1	0	0	0	1	Reserved
1	0	0	1	0	Reserved

Table 264. eQADC queue1 enhanced trigger selection (continued)

eTSEL1					eQADC enhanced trigger input
1	0	0	1	1	Reserved
1	0	1	0	0	eMIOS11 AND PIT2
1	0	1	0	1	eMIOS11 AND PIT3
1	0	1	1	0	Reserved
1	0	1	1	1	Reserved
1	1	0	0	0	Reserved
1	1	0	0	1	Reserved
1	1	0	1	0	Reserved
1	1	0	1	1	Reserved
1	1	1	0	0	Reserved
1	1	1	0	1	Reserved
1	1	1	1	0	Reserved
1	1	1	1	1	eMIOS23

Table 265. eQADC queue2 enhanced trigger selection

eTSEL2					eQADC enhanced trigger input
0	0	0	0	0	GPIO206 (eTRIG0)
0	0	0	0	1	RTI Trigger
0	0	0	1	0	PIT0 Trigger
0	0	0	1	1	PIT1 Trigger
0	0	1	0	0	PIT2 Trigger
0	0	1	0	1	PIT3 Trigger
0	0	1	1	0	Reserved
0	0	1	1	1	BOOTCFG[1] (eTRIG3)
0	1	0	0	0	eTPU30 AND PIT0
0	1	0	0	1	eTPU30 AND PIT1
0	1	0	1	0	Reserved
0	1	0	1	1	Reserved
0	1	1	0	0	eTPU28
0	1	1	0	1	eTPU29
0	1	1	1	0	eTPU30
0	1	1	1	1	eTPU31
1	0	0	0	0	Reserved
1	0	0	0	1	Reserved
1	0	0	1	0	Reserved

Table 265. eQADC queue2 enhanced trigger selection (continued)

eTSEL2					eQADC enhanced trigger input
1	0	0	1	1	Reserved
1	0	1	0	0	eMIOS10 AND PIT2
1	0	1	0	1	eMIOS10 AND PIT3
1	0	1	1	0	Reserved
1	0	1	1	1	Reserved
1	1	0	0	0	Reserved
1	1	0	0	1	Reserved
1	1	0	1	0	Reserved
1	1	0	1	1	Reserved
1	1	1	0	0	Reserved
1	1	1	0	1	Reserved
1	1	1	1	0	Reserved
1	1	1	1	1	eMIOS23

Table 266. eQADC queue3 enhanced trigger selection

eTSEL3					eQADC enhanced trigger input
0	0	0	0	0	GPIO207 (eTRIG1)
0	0	0	0	1	RTI Trigger
0	0	0	1	0	PIT0 Trigger
0	0	0	1	1	PIT1 Trigger
0	0	1	0	0	PIT2 Trigger
0	0	1	0	1	PIT3 Trigger
0	0	1	1	0	Reserved
0	0	1	1	1	PLLREF (eTRIG2)
0	1	0	0	0	eTPU30 AND PIT0
0	1	0	0	1	eTPU30 AND PIT1
0	1	0	1	0	Reserved
0	1	0	1	1	Reserved
0	1	1	0	0	eTPU28
0	1	1	0	1	eTPU29
0	1	1	1	0	eTPU30
0	1	1	1	1	eTPU31
1	0	0	0	0	Reserved
1	0	0	0	1	Reserved
1	0	0	1	0	Reserved

Table 266. eQADC queue3 enhanced trigger selection (continued)

eTSEL3					eQADC enhanced trigger input
1	0	0	1	1	Reserved
1	0	1	0	0	eMIOS10 AND PIT2
1	0	1	0	1	eMIOS10 AND PIT3
1	0	1	1	0	Reserved
1	0	1	1	1	Reserved
1	1	0	0	0	Reserved
1	1	0	0	1	Reserved
1	1	0	1	0	Reserved
1	1	0	1	1	Reserved
1	1	1	0	0	Reserved
1	1	1	0	1	Reserved
1	1	1	1	0	Reserved
1	1	1	1	1	eMIOS23

Table 267. eQADC queue4 enhanced trigger selection

eTSEL4					eQADC enhanced trigger input
0	0	0	0	0	GPIO206 (eTRIG0)
0	0	0	0	1	RTI Trigger
0	0	0	1	0	PIT0 Trigger
0	0	0	1	1	PIT1 Trigger
0	0	1	0	0	PIT2 Trigger
0	0	1	0	1	PIT3 Trigger
0	0	1	1	0	Reserved
0	0	1	1	1	BOOTCFG[1] (eTRIG3)
0	1	0	0	0	eTPU30 AND PIT0
0	1	0	0	1	eTPU30 AND PIT1
0	1	0	1	0	Reserved
0	1	0	1	1	Reserved
0	1	1	0	0	eTPU28
0	1	1	0	1	eTPU29
0	1	1	1	0	eTPU30
0	1	1	1	1	eTPU31
1	0	0	0	0	Reserved
1	0	0	0	1	Reserved

Table 267. eQADC queue4 enhanced trigger selection (continued)

eTSEL4					eQADC enhanced trigger input
1	0	0	1	0	Reserved
1	0	0	1	1	Reserved
1	0	1	0	0	eMIOS10 AND PIT2
1	0	1	0	1	eMIOS10 AND PIT3
1	0	1	1	0	Reserved
1	0	1	1	1	Reserved
1	1	0	0	0	Reserved
1	1	0	0	1	Reserved
1	1	0	1	0	Reserved
1	1	0	1	1	Reserved
1	1	1	0	0	Reserved
1	1	1	0	1	Reserved
1	1	1	1	0	Reserved
1	1	1	1	1	eMIOS23

Table 268. eQADC queue5 enhanced trigger selection

eTSEL5					eQADC enhanced trigger input
0	0	0	0	0	GPIO207 (eTRIG1)
0	0	0	0	1	RTI Trigger
0	0	0	1	0	PIT0 Trigger
0	0	0	1	1	PIT1 Trigger
0	0	1	0	0	PIT2 Trigger
0	0	1	0	1	PIT3 Trigger
0	0	1	1	0	Reserved
0	0	1	1	1	PLLREF (eTRIG2)
0	1	0	0	0	eTPU30 AND PIT0
0	1	0	0	1	eTPU30 AND PIT1
0	1	0	1	0	Reserved
0	1	0	1	1	Reserved
0	1	1	0	0	eTPU28
0	1	1	0	1	eTPU29
0	1	1	1	0	eTPU30
0	1	1	1	1	eTPU31
1	0	0	0	0	Reserved
1	0	0	0	1	Reserved

Table 268. eQADC queue5 enhanced trigger selection (continued)

eTSEL5					eQADC enhanced trigger input
1	0	0	1	0	Reserved
1	0	0	1	1	Reserved
1	0	1	0	0	eMIOS10 AND PIT2
1	0	1	0	1	eMIOS10 AND PIT3
1	0	1	1	0	Reserved
1	0	1	1	1	Reserved
1	1	0	0	0	Reserved
1	1	0	0	1	Reserved
1	1	0	1	0	Reserved
1	1	0	1	1	Reserved
1	1	1	0	0	Reserved
1	1	1	0	1	Reserved
1	1	1	1	0	Reserved
1	1	1	1	1	eMIOS23

15.6.22 IMUX Select Register 8 (SIU_ISEL8)

The SIU_ISEL8 Register is used to multiplex the eTPU[24:29] inputs.

These six eTPU channels can come from the output of the DSPI or the corresponding pad.

When SIU_ISEL8 is in its default state, the eTPU pins listed in *Figure 266* will not be connected to their respective output pin, irrespective of the SIU_PCR[PA] field.

SIU_BASE+0x920

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	ESEL5	0	0	0	ESEL4
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨		▨	▨	▨	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	ESEL3	0	0	0	ESEL2	0	0	0	ESEL1	0	0	0	ESEL0
W	▨	▨	▨		▨	▨	▨		▨	▨	▨		▨	▨	▨	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 266. IMUX Select Register 8 (SIU_ISEL8)

Table 269. SIU_ISEL8 field description

Field	Description
0:10	Reserved
11 ESEL5	eTPU input channel connected as follows: 0 DSPI_B[8] deserialized output 1 eTPU channel 29
12:14	Reserved
15 ESEL4	eTPU input channel connected as follows: 0 DSPI_B[9] deserialized output 1 eTPU channel 28
16:18	Reserved
19 ESEL3	eTPU input channel connected as follows: 0 DSPI_B[10] deserialized output 1 eTPU channel 27
20:22	Reserved
23 ESEL2	eTPU input channel connected as follows: 0 DSPI_B[11] deserialized output 1 eTPU channel 26
24:26	Reserved
27 ESEL1	eTPU input channel connected as follows: 0 DSPI_B[12] deserialized output 1 eTPU channel 25
28:30	Reserved
31 ESEL0	eTPU input channel connected as follows: 0 DSPI_B[13] deserialized output 1 eTPU channel 24

15.6.23 IMUX Select Register 9 (SIU_ISEL9)

The eQADC has a mode of operation called “Streaming”. This mode requires a second trigger for queue 0. The source for this trigger can come from eTPU, eMIOS or PIT channels. A mux select register is required to select the source of this new queue0 trigger.

SIU_BASE+0x924

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	eTSEL0A				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 267. IMUX Select Register 9 (SIU_ISEL9)

Table 270. eQADC advance trigger selection

eTSEL0A					eQADC enhanced trigger input
0	0	0	0	0	Reserved
0	0	0	0	1	RTI Trigger
0	0	0	1	0	PIT0 Trigger
0	0	0	1	1	PIT1 Trigger
0	0	1	0	0	PIT2 Trigger
0	0	1	0	1	PIT3 Trigger
0	0	1	1	0	Reserved
0	0	1	1	1	Reserved
0	1	0	0	0	eTPU30 AND PIT0
0	1	0	0	1	eTPU30 AND PIT1
0	1	0	1	0	Reserved
0	1	0	1	1	Reserved
0	1	1	0	0	eTPU28
0	1	1	0	1	eTPU29
0	1	1	1	0	eTPU30
0	1	1	1	1	eTPU31
1	0	0	0	0	Reserved
1	0	0	0	1	Reserved
1	0	0	1	0	Reserved
1	0	0	1	1	Reserved
1	0	1	0	0	eMIOS10 AND PIT2
1	0	1	0	1	eMIOS10 AND PIT3
1	0	1	1	0	Reserved

Table 270. eQADC advance trigger selection (continued)

eTSEL0A					eQADC enhanced trigger input
1	0	1	1	1	Reserved
1	1	0	0	0	Reserved
1	1	0	0	1	Reserved
1	1	0	1	0	Reserved
1	1	0	1	1	Reserved
1	1	1	0	0	Reserved
1	1	1	0	1	Reserved
1	1	1	1	0	Reserved
1	1	1	1	1	eMIOS23

15.6.24 IMUX Select Register 10 (SIU_ISEL10)

The IMUX Select Register 10 (SIU_ISEL10) register contains bit fields that specify which eTPU output is connected to the decimation filter Integrator halt signal (HSELx) and Integrator reset signal (ZSELx).For more details refer to *Section 25.3.3, Integrator halt signal* and *Section 25.3.4, Integrator reset signal*.

SIU_BASE+0x928

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	HSELB				ZSELB				HSELA				ZSELA			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 268. IMUX Select Register 10 (SIU_ISEL10 or SIU_DECFIL1)

Table 271. Decimation filter control source selection

Field	Code				Source
ZSELA	0	0	0	1	eTPU[22]
	0	0	1	0	eTPU[23]
	0	0	1	1	eTPU[24]
	0	1	0	0	eTPU[25]
	Others				Unused

Table 271. Decimation filter control source selection

Field	Code				Source
HSELA	0	0	0	1	eTPU[22]
	0	0	1	0	eTPU[23]
	0	0	1	1	eTPU[24]
	0	1	0	0	eTPU[25]
	Others				Unused
ZSELB	0	0	0	1	eTPU[22]
	0	0	1	0	eTPU[23]
	0	0	1	1	eTPU[24]
	0	1	0	0	eTPU[25]
	Others				Unused
HSELB	0	0	0	1	eTPU[22]
	0	0	1	0	eTPU[23]
	0	0	1	1	eTPU[24]
	0	1	0	0	eTPU[25]
	Others				Unused

15.6.25 Chip Configuration Register (SIU_CCR)

SIU_BASE + 0x980

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	MATCH	DISNEX
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	U(1)	U(2)

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	CRSE	0
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨		▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

1. During reset the comparison is performed and result is uncertain
2. The value after reset is uncertain

Figure 269. Chip Configuration Register (SIU_CCR)

Table 272. SIU_CCR field description

Field	Description
0:13	Reserved
14 MATCH	Compare Register Match. The MATCH bit is a read only bit that holds the value of the match input signal to the SIU. The match bit is asserted if the password in shadow flash memory and the contents of the SIU_CBRH/SIU_CBRL registers are equal. 0 Match input signal is negated 1 Match input signal is asserted
15 DISNEX	Disable Nexus. The DISNEX bit is a read only bit that holds the value of the Nexus disable input signal to the SIU. When system reset negates, the value in this bit depends on the censorship control word and the boot configuration bits. 0 Nexus disable input signal is negated 1 Nexus disable input signal is asserted
16:29	Reserved
30 CRSE	Calibration Reflection Suppression Enable. The CRSE bit enables the suppression of reflections from the EBI's calibration bus onto the non-calibration bus. The EBI drives some outputs to both the calibration and non-calibration busses. When CRSE is asserted, the values driven onto the calibration bus pins will not be reflected onto the non-calibration bus pins. When CRSE is negated, the values driven onto the calibration bus pins will be reflected onto the non-calibration bus pins. CRSE only enables reflection suppression for non-calibration bus pins which do not have a negated state to which the pins return at the end of the access. CRSE does not enable reflection suppression for the non-calibration bus pins which have a negated state to which the pins return at the end of an access. Those reflections always are suppressed. Furthermore, the suppression of reflections from the non-calibration bus onto the calibration bus is not enabled by CRSE. Those reflections are also always suppressed. 0 Calibration reflection suppression is disabled 1 Calibration reflection suppression is enabled
31	Reserved

15.6.26 External Clock Control Register (SIU_ECCR)

The SIU_ECCR controls the timing relationship between the system clock and the external clock CLKOUT. All bits and fields in the SIU_ECCR are read/write and are reset by the global signals asynchronous reset signal.

SIU_BASE+0x984

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	ENGDIV								ENGSSE	0	0	0	EBTS	0	EBDF	
W										▨	▨	▨		▨		
Reset	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1

▨ = Unimplemented or Reserved

Figure 270. External Clock Control Register (SIU_ECCR)

Table 273. SIU_ECCR field description

Field	Description
0:15	Reserved
16:23 ENGDIV	Engineering Clock Division Factor. The ENGDIV field specifies the frequency ratio between the system clock and the ENGCLK. The ENGCLK frequency is divided from the system clock frequency according to the following equation. **Equation 2** $\text{ENGCLK} = \frac{\text{SystemClockorCrystalOscillator}}{\text{ENGDIVx2}}$ ENGDIV = 0 is reserved and results in the ENGCLK frequency being equal to the System Clock Frequency.
24 ENGSSE	Engineering clock (ENGCLK) source select. 0 ENGCLK source is system clock. 1 ENGCLK source is crystal oscillator clock.
25:27	Reserved
28 EBTS	External Bus Tap Select. The EBTS bit changes the phase relationship between the system clock and CLKOUT. Changing the phase relationship so that CLKOUT is advanced in relation to system clock increases the output hold time of the external bus signals to a non-zero value. It also increases the output delay times, increases the input hold times to non-zero values, and decreases the input setup times. The EBTS bit must not be modified while an external bus transaction is in progress.

Table 273. SIU_ECCR field description (continued)

Field	Description
29	Reserved
30:31 EBDF	External Bus Division Factor. The EBDF field specifies the frequency ratio between the system clock and CLKOUT. The EBDF field must not be changed during an external bus access or while an access is pending. 00 External bus division factor = 1 01 External bus division factor = 2 10 Reserved 11 External bus division factor = 4 The reset value of the EBDF field is divide-by-2. After reset, if EBDF is changed to divided-by-1, no glitches occurs on the CLKOUT signal, but if EBDF is changed back to divide-by-2 or divide-by-4, there is no guarantee that the switch will be glitchless.

15.6.27 Compare A High Register (SIU_CARH)

The SIU_CARH register holds the 32-bit value that is compared against the value in the SIU_CBRH register. The CMPAH field is read/write and is reset by the IP Green-Line synchronous reset signal.

SIU_BASE + 0x988

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CMPAH															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CMPAH															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 271. Compare A High Register (SIU_CARH)

15.6.28 Compare A Low Register (SIU_CARL)

The SIU_CARL register holds the 32-bit value that is compared against the value in the SIU_CBRL register. The CMPAL field is read/write and is reset by the IP Green-Line synchronous reset signal.

SIU_BASE + 0x98C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CMPAL															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CMPAL															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 272. Compare A Low Register (SIU_CARL)

15.6.29 Compare B High Register (SIU_CBRH)

The SIU_CBRH register holds the 32-bit value that is compared against the value in the SIU_CARH register. The CMPBH field is read/write and is reset by the IP Green-Line synchronous reset signal.

SIU_BASE + 0x990

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CMPBH															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CMPBH															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 273. Compare B High Register (SIU_CBRH)

15.6.30 Compare B Low Register (SIU_CBRL)

The SIU_CBRL register holds the 32-bit value that is compared against the value in the SIU_CARL register. The CMPBL field is read/write and is reset by the IP Green-Line synchronous reset signal.

SIU_BASE + 0x994

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CMPBL															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CMPBL															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 274. Compare B Low Register (SIU_CBRL)

15.6.31 System Clock Register (SIU_SYSDIV)

The fields in the SIU_SYSDIV register are read/write and are reset by the IP Green-Line synchronous reset signal.

SIU_BASE + 0x9A0

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	CAN_SRC
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	BYPASS	SYSCLKDIV		0	0
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨				▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

▨ = Unimplemented or Reserved

Figure 275. System Clock Register (SIU_SYSDIV)

Table 274. SIU_SYSDIV field description

Field	Description
0:14	Reserved
15 CAN_SRC	FlexCAN 2:1 mode 1 When CAN_CTRL[CLK_SRC] = 1, FlexCAN runs at half the system frequency 0 When CAN_CTRL[CLK_SRC] = 1, FlexCAN runs at the system frequency See *Section , Support for CAN interface operation*.
16:26	Reserved

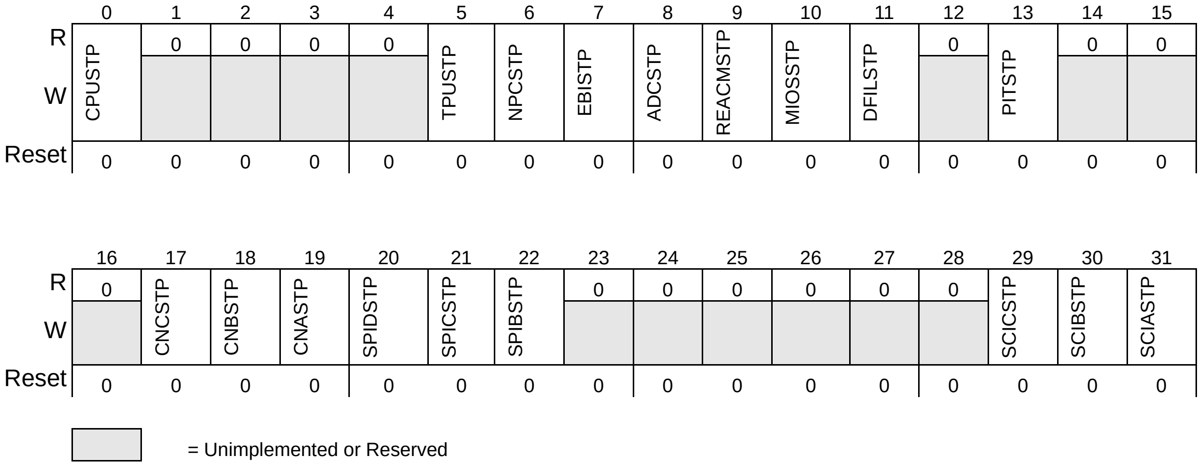
Table 274. SIU_SYSDIV field description (continued)

Field	Description
27 BYPASS	Bypass bit 1 System clock divider is bypassed 0 System clock divider is not bypassed
28:29 SYSCLKDIV	System Clock Divide The SYSCLKDIV bits select the divider value for the system clock (ipg_clk). Note that the SYSCLKDIV divider is required in addition to the RFD to allow the other sources for the system clock (16 MHz IRC and OSC) to be divided down to slowest frequencies to improve power. The output of the clock divider is nominally a 50% duty cycle. 00 Divide by 2 01 Divide by 4 10 Divide by 8 11 Divide by 16 Note: The above four divider values can be selected only if SIU_SYSDIV[BYPASS] value = 0. If SIU_SYSDIV[BYPASS] = 1, the system clock divider is bypassed and “divide by 1” is selected.
30:31	Reserved

15.6.32 Halt Register (SIU_HLT)

The SIU_HLT register is used to put various modules into stop mode to save power. Each bit will drive a separate stop request signal to a different module. When the module acknowledges the stop request, the clock to that module is halted. In order to remove the module from stop mode, the corresponding bit in the SIU_HLT register must be cleared. In the case of the CPU, stop mode in entered when the corresponding bit in SIU_HLT is set and a WAIT instruction is executed. The CPU exits stop mode upon reception of any interrupt request.

SIU_BASE + 0x9A4



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CPUSTP	0	0	0	0	TPUSTP	NPCSTP	EBISTP	ADCSTP	REACMSTP	MIOSSTP	DFILSTP	0	PITSTP	0	0
W		▨	▨	▨	▨								▨		▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	CNCSTP	CNBSTP	CNASTP	SPIDSTP	SPICSTP	SPIBSTP	0	0	0	0	0	0	SCICSTP	SCIBSTP	SCIASTP
W	▨							▨	▨	▨	▨	▨	▨			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 276. Halt Register (SIU_HLT)

Table 275. SIU_HLT field description

Field	Description
0 CPUSTP	CPU stop request When asserted, a stop request is sent to the following modules: CPU, cross-bar, peripheral bridge, system RAM, STM, and DMA. 1: Stop mode request 0: Normal operation
1:2	Reserved
3	Reserved (unimplemented)
4	Reserved
5 TPUSTP	eTPU stop request When asserted, a stop request is sent to the eTPU module and the eTPU Nexus module. 1: Stop mode request 0: Normal operation
6 NPCSTP	Nexus stop request When asserted, a stop request is sent to the Nexus Controller. 1: Stop mode request 0: Normal operation
7 EBISTP	EBI stop request When asserted, a stop request is sent to the external bus controller which handles the calibration interface. 1: Stop mode request 0: Normal operation
8 ADCSTP	eQADC stop request When asserted, a stop request is sent to the eQADC module. 1: Stop mode request 0: Normal operation
9 REACMSTP	Reaction module stop request When asserted, a stop request is sent to the Reaction module. 1: Stop mode request 0: Normal operation
10 MIOSSTP	eMIOS stop request When asserted, a stop request is sent to the eMIOS module. 1: Stop mode request 0: Normal operation
11 DFILSTP	Decimation filter stop request When asserted, a stop request is sent to the decimation filter module. 1: Stop mode request 0: Normal operation
12	Reserved
13 PITSTP	PIT stop request When asserted, a stop request is sent to the periodic interrupt timer module. 1: Stop mode request 0: Normal operation

Table 275. SIU_HLT field description (continued)

Field	Description
14:16	Reserved
17 CNCSTP	FlexCAN C stop request When asserted, a stop request is sent to the FlexCAN C module. 1: Stop mode request 0: Normal operation
18 CNBSTP	FlexCAN B stop request When asserted, a stop request is sent to the FlexCAN B module. 1: Stop mode request 0: Normal operation
19 CNASTP	FlexCAN A stop request When asserted, a stop request is sent to the FlexCAN A module. 1: Stop mode request 0: Normal operation
20 SPIDSTP	DSPI D stop request When asserted, a stop request is sent to the DSPI C. 1: Stop mode request 0: Normal operation
21 SPICSTP	DSPI C stop request When asserted, a stop request is sent to the DSPI C. 1: Stop mode request 0: Normal operation
22 SPIBSTP	DSPI B stop request When asserted, a stop request is sent to the DSPI B. 1: Stop mode request 0: Normal operation
23:28	Reserved
29 SCICSTP	eSCI C stop request When asserted, a stop request is sent to the eSCI C module. 1: Stop mode request 0: Normal operation
30 SCIBSTP	eSCI B stop request When asserted, a stop request is sent to the eSCI B module. 1: Stop mode request 0: Normal operation
31 SCIASTP	eSCI A stop request When asserted, a stop request is sent to the eSCI A module. 1: Stop mode request 0: Normal operation

15.6.33 Halt Acknowledge Register (SIU_HLTACK)

The bits in the SIU_HLTACK register indicate that the module requested to halt via the SIU_HLT register has completed the halt process and has entered a halted state with the module clocks disabled. This register is read-only.

SIU_BASE + 0x9A8

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CPUACK	0	0	0	NSETIACK	TPUACK	NPCACK	EBIACK	ADCACK	REACMACK	MIOSACK	DFILACK	0	PITACK	0	0
W	▨				▨	▨	▨	▨	▨	▨	▨	▨		▨		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	CNCACK	CNBACK	CNAACK	SPIDACK	SPICACK	SPIBACK	0	0	0	0	0	0	SCICACK	SCIBACK	SCIAACK
W		▨	▨	▨	▨	▨	▨							▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 277. Halt Acknowledge Register (SIU_HLTACK)

Table 276. SIU_HLTACK field description

Field	Description
0 CPUACK	CPU stop acknowledge When asserted, indicates that a stop acknowledge was received from the following modules: CPU, cross-bar, peripheral bridge, system RAM, Flash, STM, DMA. 1: Stop mode request 0: Normal operation
1:3	Reserved
4 NSETIACK	eTPU Nexus module (NSETI) stop acknowledge When asserted, indicates that a stop acknowledge was received from the NSETI module. 1: Stop mode request 0: Normal operation
5 TPUACK	eTPU stop acknowledge When asserted, indicates that a stop acknowledge was received from the eTPU module. 1: Stop mode request 0: Normal operation

Table 276. SIU_HLTACK field description (continued)

Field	Description
6 NPCACK	Nexus stop acknowledge When asserted, indicates that a stop acknowledge was received from the Nexus Controller. 1: Stop mode request 0: Normal operation
7 EBIACK	EBI stop acknowledge When asserted, indicates that a stop acknowledge was received from the external bus controller which handles the calibration interface. 1: Stop mode request 0: Normal operation
8 ADCACK	eQADC stop acknowledge When asserted, indicates that a stop acknowledge was received from the eQADC module. 1: Stop mode request 0: Normal operation
9 REACMACK	Reaction module (REACM) stop acknowledge When asserted, indicates that a stop acknowledge was received from the Reaction module. 1: Stop mode request 0: Normal operation
10 MIOSACK	eMIOS stop acknowledge When asserted, indicates that a stop acknowledge was received from the eMIOS module. 1: Stop mode request 0: Normal operation
11 DFILACK	Decimation filter stop acknowledge When asserted, indicates that a stop acknowledge was received from the decimation filter module. 1: Stop mode request 0: Normal operation
12	Reserved
13 PITACK	PIT stop acknowledge When asserted, indicates that a stop acknowledge was received from the periodic interrupt timer module. 1: Stop mode request 0: Normal operation
14:16	Reserved
17 CNCACK	FlexCAN C stop acknowledge When asserted, indicates that a stop acknowledge was received from the FlexCAN C module. 1: Stop mode request 0: Normal operation
18	FlexCAN B stop acknowledge When asserted, indicates that a stop acknowledge was received from the FlexCAN B module. 1: Stop mode request 0: Normal operation

Table 276. SIU_HLTACK field description (continued)

Field	Description
19 CNAACK	FlexCAN A stop acknowledge When asserted, indicates that a stop acknowledge was received from the FlexCAN A module. 1: Stop mode request 0: Normal operation
20	DSPI D stop acknowledge When asserted, indicates that a stop acknowledge was received from the DSPI D. 1: Stop mode request 0: Normal operation
21 SPICACK	DSPI C stop acknowledge When asserted, indicates that a stop acknowledge was received from the DSPI C. 1: Stop mode request 0: Normal operation
22 SPIBACK	DSPI B stop acknowledge When asserted, indicates that a stop acknowledge was received from the DSPI B. 1: Stop mode request 0: Normal operation
23:28	Reserved
29	eSCI C stop acknowledge When asserted, indicates that a stop acknowledge was received from the eSCI C module. 1: Stop mode request 0: Normal operation
30 SCIBACK	eSCI B stop acknowledge When asserted, indicates that a stop acknowledge was received from the eSCI B module. 1: Stop mode request 0: Normal operation
31 SCIAACK	eSCI A stop acknowledge When asserted, indicates that a stop acknowledge was received from the eSCI A module. 1: Stop mode request 0: Normal operation

15.6.34 Core MMU PID Control Register (SIU_EMPCR0)

The SIU_EMPCR0 register is part of a mechanism that provides the capability of real-time remapping of MMU entries by software or an external tool. This capability is intended for use in calibration activities requiring real-time switching between calibration data tables.

The SIU_EMPCR0 register works in conjunction with the Nexus module to enable an external calibration tool to modify the logical-to-physical address mapping of the calibration bus by replacing bits 6:7 of the MMU’s PID (process id) register with values specified in SIU_EMPCR0 register fields. This remapping does not interrupt normal application code execution.

In addition, the register provides a synchronization mechanism that enables the mapping to change when a specified instruction address is reached or a specified load/store address is accessed. Synchronization is implemented using Watchpoint Event 2 output in the processor core.

The mechanism is detailed in *Figure 278* and *Table 277*.

SIU_BASE + 0x9B4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	EXT_PID_EN	EXT_PID_SYNC0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W			▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	EXT_PID6	EXT_PID7
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 278. Core MMU PID Control Register (SIU_EMPCR0)

Table 277. SIU_EMPCR0 field description

Field	Description
EXT_PID_EN 0	External PID Selection Enable 0: The contents of this register are not used to select the alternate MMU mapping. 1: The contents of this register are used to select the alternate MMU mapping defined by EXT_PID6 and EXT_PID7.
EXT_PID_SYNC0 1	External PID Synchronization 0 0: The Nexus Watchpoint Event 2 does not transfer the EXT_PID6 and EXT_PID7 values to the MMU. 1: The Nexus Watchpoint Event 2 transfers the EXT_PID6 and EXT_PID7 values to the MMU.
2:29	Reserved
EXT_PID6 30	External PID bit 6 0: When the PID remapping is enabled (EXT_PID_EN = 1), the processor MMU’s PID register bit 6 is logic 0. 1: When the PID remapping is enabled (EXT_PID_EN = 1), the processor MMU’s PID register bit 6 is logic 1.
EXT_PID7 31	External PID bit 7 0: When the PID remapping is enabled (EXT_PID_EN = 1), the processor MMU’s PID register bit 7 is logic 0. 1: When the PID remapping is enabled (EXT_PID_EN = 1), the processor MMU’s PID register bit 7 is logic 1.

15.7 Functional description

The following sections provide an overview of the SIU operation features.

15.7.1 System configuration

Boot configuration

Two BOOTCFG signals are implemented in SPC564A70 MCUs.

The BAM program uses the BOOTCFG0 bit to determine where to read the reset configuration word, and whether to initiate a FlexCAN or eSCI boot. See *Section 4.7.1, Reset configuration half word (RCHW)*, for details on the RCHW.

Table 326 in *Section 20.5.2, BAM program operation*, defines the boot modes specified by the BOOTCFG0 and BOOTCFG1 pins. During the assertion of RSTOUT, the BOOTCFG0 and BOOTCFG1 pins are used to update the RSR and the BAM boot mode.

This device has a second serial boot mode to support a new serial boot with CAN and SCI baudrate auto-detection.

For additional details on the BAM program operation see *Chapter 20, Boot Assist Module (BAM)*.

Pad configuration

The Pad Configuration Registers (PCR) in the SIU allow software control of the static electrical characteristics of external pins. The multiplexed function of a pin, selection of pull up or pull down devices, the slew rate of I/O signals, open drain mode for output pins, hysteresis on input pins, and the drive strength for bus signals can be specified through the PCRs.

15.7.2 Reset control

The reset controller logic is located in the SIU. See *Chapter 4, Resets* for details on reset operation.

15.7.3 External interrupt request input (IRQ)

The fifteen external interrupt request inputs available on this device ($\overline{\text{IRQ}}$[0:5,7:15]) connect to the SIU IRQ inputs. The External IRQ Input Select Register (EIISR) specifies the $\overline{\text{IRQ}}$[0:5,7:15] signals that are input to the SIU IRQs.

Note: IRQ[6] can be only generated by the deserialized output of the DSPI module—not the external pins.

External interrupt requests are triggered by rising- and/or falling-edge events that are enabled by setting a bit in:

- IRQ rising-edge event enable register (SIU_IREER)
- IRQ falling-edge event enable register (SIU_IFEER)

If the bit is set in both registers, both rising- and falling-edge events trigger an interrupt request. Each IRQ has a counter that tracks the number of system clock cycles that occur between the rising- and falling-edge events. An IRQ counter exists for each IRQ rising- or falling-edge event enable bit.

The digital filter length field in the IRQ digital filter register (SIU_IDFR) specifies the minimum number of system clocks that the IRQ signal must hold a logic value to qualify the edge-triggered event as a valid state change. When the number of system clocks in the IRQ counter equals the value in the digital filter length field, the IRQ state latches and the IRQ counter is cleared.

If the previous filtered state of the IRQ does not match the current state, and the rising- or falling-edge event is enabled, the IRQ flag bit is set to 1. For example, the IRQ flag bit is set if a rising-edge event occurs under the following conditions:

- Previous filtered IRQ state was a logic 0
- Current latched IRQ state is a logic 1
- Rising-edge event is enabled for the IRQ

When the counter for an IRQ is not enabled, the state of the IRQ is held in the current and previous state latches. The IRQ counter operates independently of the IRQ or overrun flag bit. Clearing the IRQ flag or overrun flag bits does not clear or reload the counter.

Refer to the following sections for more information:

- *Section 15.6.6, External Interrupt Status Register (SIU_EISR)*
- *Section 15.6.11, IRQ Rising-Edge Event Enable Register (SIU_IREER)*
- *Section 15.6.12, External IRQ Falling-Edge Event Enable Register (SIU_IFEER)*
- *Section 15.6.13, External IRQ Digital Filter Register (SIU_IDFR)*

External interrupts

The IRQ signals map to 15 independent interrupt requests output from the SIU. The IRQ flag bit is set when a rising-edge and/or falling-edge event occurs for the IRQ. An external IRQ signal is asserted when all of the following occur:

- Enable bit is set in the IRQ rising- and/or falling-edge event registers (SIU_IREER, SIU_IFEER)
- IRQ flag bit is set in the external interrupt status register (SIU_EISR)
- Enable bit is cleared in the DMA/Interrupt request enable register (SIU_DIRER)
- Select bit is cleared in the DMA/Interrupt select register (SIU_DIRSR)

The NMI and SWT Interrupts can each generate an NMI Exception or Critical Interrupt Exception as an input to the core. This selection is controlled by the NMI_SEL8 and NMI_SEL0 (SIU_DIRER) signals respectively. When WKPCFG_NMI_GPIO213 is enabled as NMI, the pin will override the PCR configuration after reset. The SIU_DIRER selects between critical and non-maskable interrupt use, the SIU_EISR reports the status of NMI, and the SIU_IFEER selects edge sensitivity of the NMI input.

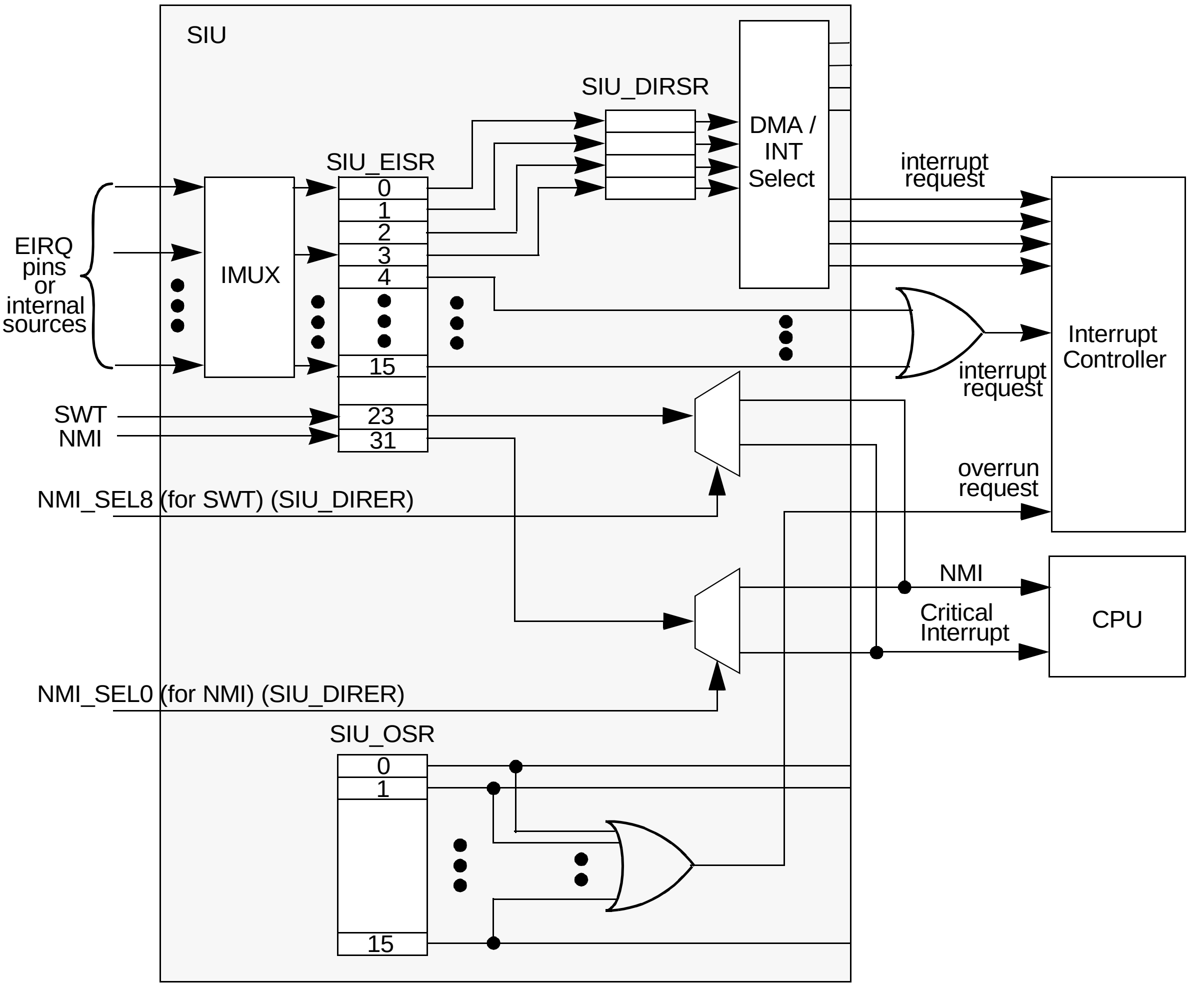


Figure 279. SIU DMA/Interrupt request diagram

Refer to the following sections for more information:

- *Section 15.6.7, DMA/Interrupt Request Enable Register (SIU_DIRER)*
- *Section 15.6.8, DMA/Interrupt Request Select Register (SIU_DIRSR)*

15.7.4 GPIO operation

The SIU provides all GPIO functionality for this device. Each device pin that has GPIO functionality has an associated Pin Configuration Register in the SIU where the GPIO function is selected for the pin. In addition, each device pin with GPIO functionality has an input data register (SIU_GPDI*x*_*x*) and an output data register (SIU_GPDO*x*_*x*).

15.7.5 Internal multiplexing

The IMUX Select Registers (SIU_ISEL*x*) provide selection of the eQADC external trigger inputs sources, the SIU external interrupts, some of the eTPU inputs and the DSPI signals that are used in the serial and parallel chaining of DSPI blocks.

eQADC external trigger input multiplexing

The six eQADC external trigger inputs can be connected to either an external pin, an eTPU channel, an eMIOS channel or a PIT or RTI trigger. The input source for each eQADC external trigger is individually specified in the ETISR, SIU_ISEL3 and SIU_ISEL9 registers. One of these inputs is in turn specified in the IMUX Select Register 3 (SIU_ISEL3). An example of the multiplexing of an eQADC external trigger input is given in *Figure 280*. As shown in the figure, the Trigger[0] input of the eQADC can be connected to either the ETRIG[0] pin, the eTPU_A[30] channel or the eMIOS[10] channel or the output of the mux, IMUX3. The inputs of IMUX3 can be some of the eTPU/eMIOS channels or PIT/RTI triggers or External triggers.

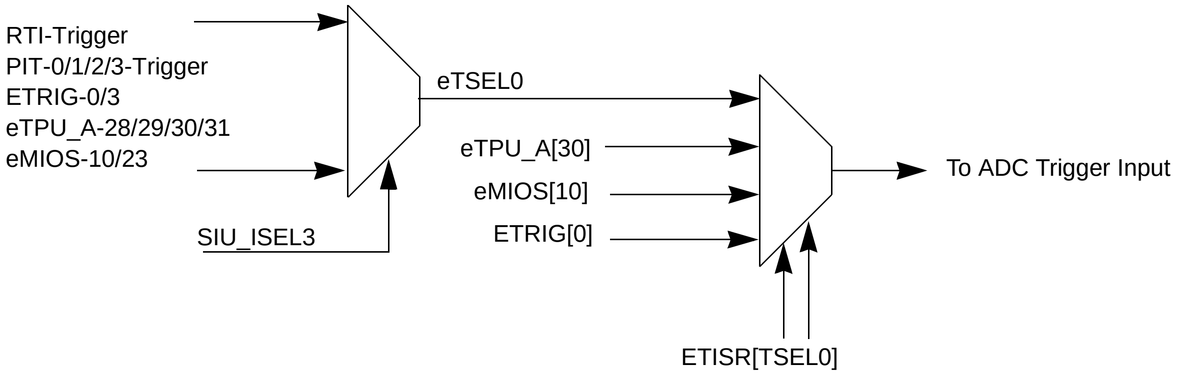


Figure 280. eQADC trigger input multiplexing example

The remaining ADC trigger inputs are multiplexed in the same manner. Note that if an ETRIG input is connected to an eTPU or eMIOS channel, the external pin used by that channel can be used by the alternate function on that pin.

Along with the six eQADC external trigger inputs, there is one additional trigger input to ADC. The source of that trigger is specified in IMUX Select Register 9 (SIU_ISEL9). It is similar to SIU_ISEL3, shown in the above figure.

SIU external interrupt input multiplexing

The fifteen SIU external interrupt inputs can be connected to either external pins or to deserialized output signals from a DSPI block. The input source for each SIU external interrupt is individually specified in the External IRQ Input Select Register(j) (EIISR). An example of the multiplexing of an SIU external interrupt input is given in *Figure 281*. As shown in the figure, the IRQ[0] input of the SIU can be connected to either the eMIOS[14]_IRQ[0]_eTPU_A[29]_GPIO[193] pin, the DSPI_B[0] deserialized output signal, the DSPI_C[1] deserialized output signal or eMIOS[14] channel. The remaining IRQ inputs are multiplexed in the same manner. Only IRQ[0] and [1] have an eMIOS channel as input. Other IRQ input source is one of the DSPI_D[15:4,1:0] output signal instead (as specified in the External IRQ Input Select Register (EIISR)). The inputs to the IRQ from each DSPI block are offset by one so that if more than one DSPI block is connected to the same external device type, a separate interrupt can be generated for each device. This also

---

j. The EIISR is sometimes referred to as ISEL1.

applies to DSPI blocks connected to external devices of different type that have status bits in the same bit location of the deserialized information.

See *Section 15.6.19, External IRQ Input Select Register (SIU_EIISR)*, for more information.

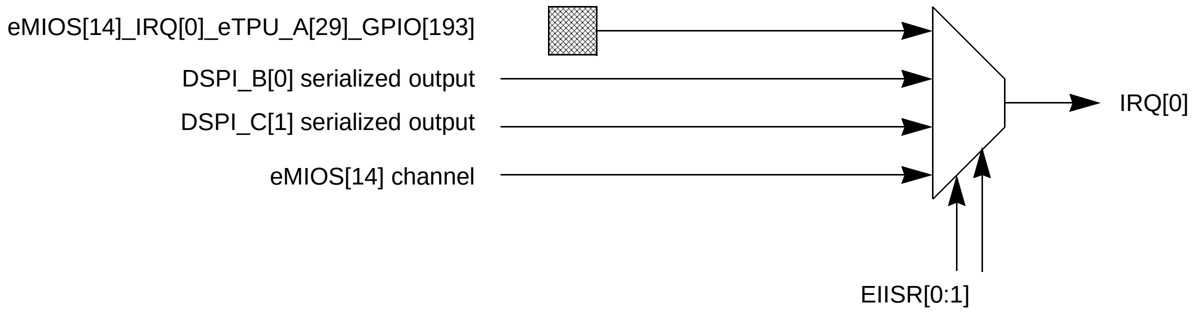


Figure 281. SIU external interrupt input multiplexing example

Multiplexed inputs for DSPI multiple transfer operation

To support multiple DSPIs transfer operations, an input multiplexor is required for the SIN, SS, SCK IN, and trigger signals of each DSPI. These DSPI input sources can be a pin or respectively the SOUT, PCS[0], SCK OUT, or PCSS of any other DSPI. They are individually specified in the DSPI Input Select Register (DISR).

See *Section , Multiple transfer operation (MTO)*, for more information on Multiple Transfer Operation.

Multiplexed inputs for eTPU[29:24]

The eTPU channel input pins, eTPU[29:24], are multiplexed with DSPI_B[8:13] deserialized output signals and then given as input to the eTPU block. These are individually specified in the IMUX Select Register 8 (SIU_ISEL8).

When SIU_ISEL8 is in its default state, the eTPU[29:24] will not be connected to their respective output pin, irrespective of the SIU_PCR[PA] field. The SIU_ISEL8 register must be modified if these signals are to be used as external inputs or outputs.

16 Frequency-modulated phase locked loop (FMPLL)

16.1 Information specific to this device

This section presents device-specific parameterization and customization information not specifically referenced in the remainder of this chapter.

16.1.1 Device-specific features

- On-chip oscillator for external crystal: Range (4–40 MHz)
- Internal RC oscillator (RCOSC): 16 MHz
- Phase-locked loop (PLL): VCO Range (256–512 MHz)
- PLLREF top level pin to control PLL reference
- Clock Quality monitor
- System Clock Divider (SYSDIV) used to further reduce the system clock frequency
- Register to control system clock source and programming of PLL parameter
- Clock gating for individual modules controlled by either SIU_HLT or module’s MDIS register bit (Refer to *Table 14 (MDIS support) 5, Operating Modes and Clocking*, to see which modules implement the MDIS bit.)

16.1.2 Device-specific parameters

Table 278 shows the reset values for several register fields on this device.

Table 278. Register field reset values

Parameter name	Value	Description
FMPLL_SYNCR[PREDIV]	0b111	inhibit the clock to the phase detector
FMPLL_SYNCR[MFD]	0b00100	divide-by-8
FMPLL_SYNCR[RFD]	0b010	divide-by-4
FMPLL_ESYNCR1[EMODE]	0b0	allowing legacy mode to be used
FMPLL_ESYNCR1[EPREDIV]	0b1111	inhibit the clock to the phase detector
FMPLL_ESYNCR1[EMFD]	0b0100000	divide-by-32
FMPLL_ESYNCR2[ERFD]	0b11	divide-by-16

16.2 Introduction

This chapter describes the features and functions of the FMPLL module.

16.2.1 Overview

The frequency modulated phase locked loop (FMPLL) allows the user to generate high speed system clocks from a crystal oscillator or from an external clock generator. Furthermore, the FMPLL supports programmable frequency modulation of the system clock. The FMPLL multiplication factor, reference clock predivider factor, output clock divider ratio, modulation depth and multiplication rate are all controllable through programmable registers.

Figure 282 shows the block diagram of the FMPLL.

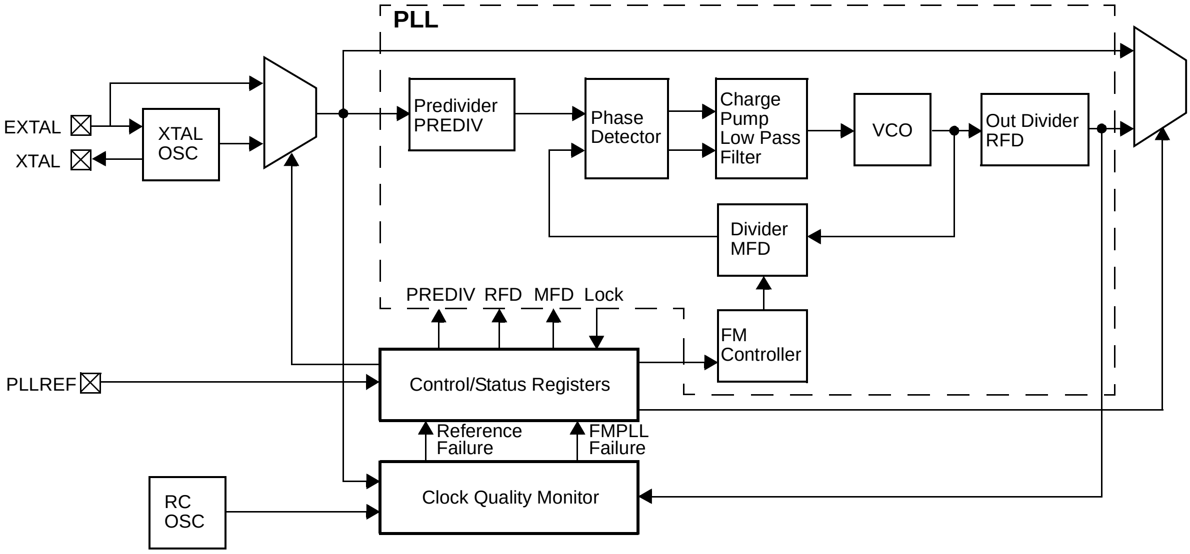


Figure 282. FMPLL block diagram

16.2.2 Features

The FMPLL has the following features:

- Reference clock predivider for finer frequency synthesis resolution
- Reduced frequency divider for reducing the FMPLL output clock frequency without forcing the FMPLL to relock
- Input clock frequency range from 4 MHz to 20 or 40 MHz(k) before the predivider, and from 4 MHz to 16 MHz after the predivider
- Voltage controlled oscillator (VCO) range from 256 MHz to 512 MHz
- VCO free-running frequency range from 25 MHz to 125 MHz
- 4 bypass modes: crystal or external reference with PLL on or off
- 2 normal modes: crystal or external reference
- Programmable frequency modulation
  - Triangle wave modulation
  - Register programmable modulation frequency and depth
- Lock detect circuitry reports when the FMPLL has achieved frequency lock and continuously monitors lock status to report loss of lock conditions
  - User-selectable ability to generate an interrupt request upon loss of lock
  - User-selectable ability to generate a system reset upon loss of lock
- Clock quality monitor (CQM) module provides loss-of-clock detection for the FMPLL reference and output clocks
  - User-selectable ability to generate an interrupt request upon loss of clock
  - User-selectable ability to generate a system reset upon loss of clock
  - Backup clock (reference clock or FMPLL free-running) can be applied to the system in case of loss of clock

16.2.3 Modes of operation

Upon reset, the operational mode is bypass with PLL running, and the source of the reference clock, either the crystal oscillator or external clock, is determined by the reset value of the CLKCFG[2] bit of the FMPLL_ESYNCR1. The reset state of this bit comes from an external signal to the module connected to a package pin called PLLREF. After reset, a different operational mode can be selected by writing to FMPLL_ESYNCR1[CLKCFG]. The available modes are specified in *Table 279*.

Table 279. Clock mode selection

CLKCFG[0]	CLKCFG[1](1)	CLKCFG[2](2)	Clock mode
0	0	0	Bypass mode with external reference and PLL off
0	0	1	Bypass mode with crystal reference and PLL off
0	1	0	Bypass mode with external reference and PLL running
0	1	1	Bypass mode with crystal reference and PLL running
1	0	0	Reserved

---

k. See *Section 16.1, Information specific to this device*, for information on crystal frequencies supported.

Table 279. Clock mode selection (continued)

CLKCFG[0]	CLKCFG[1][1]	CLKCFG[2][2]	Clock mode
1	0	1	Reserved
1	1	0	Normal mode with external reference
1	1	1	Normal mode with crystal reference

1. CLKCFG[1] is not writable to zero while CLKCFG[0] = 1.
2. The reset state of this bit is determined by the logical state applied to the PLLREF pin.

At reset the FMPLL is enabled, but the reset value of the predivider may be set by the SoC integration to inhibit the clock to the PLL, making the VCO run within its free-running frequency range of 25 MHz to 125 MHz, unconnected from the system clock (since bypass is the default mode at reset). If using crystal reference, after power-on reset the Clock Quality Monitor (CQM) will inhibit the system clock and keep system reset asserted while the crystal oscillator has not stabilized. The PLLREF input must be kept stable during the whole period while system reset is asserted.

Bypass mode with crystal reference

In the bypass mode with crystal reference, the FMPLL is completely bypassed and the system clock is driven from the crystal oscillator. The user must supply a crystal that is within the appropriate frequency range, the crystal manufacturer recommended external support circuitry, and short signal route from the MCU to the crystal.

In bypass mode the PLL itself may or may not be running, depending on the state of the CLKCFG[1] bit of FMPLL_ESYNCR1, but the PLL output is not connected to the system clock. Consequently, frequency modulation is not available. The predivider is also bypassed.

Bypass mode with crystal reference is the default mode at reset if the PLLREF input is driven high. After reset, this mode can be entered by programming FMPLL_ESYNCR1[CLKCFG] as shown in *Table 279*.

Bypass mode with external reference

The bypass mode with external reference functions the same as bypass mode with crystal reference, except that the system clock is driven by an external clock generator connected to the EXTAL pin, rather than a crystal oscillator. The input frequency range is the same and frequency modulation is not available.

Bypass mode with external reference is the default mode at reset if the PLLREF input is driven low. After reset, this mode can be entered by programming FMPLL_ESYNCR1[PLLCFG] as shown in *Table 279*.

Normal mode with crystal reference

In the normal mode with crystal reference, the FMPLL receives an input clock frequency from the crystal oscillator and the predivider, and multiplies the frequency to create the FMPLL output clock. The user must supply a crystal that is within the appropriate frequency range, the crystal manufacturer recommended external support circuitry, and short signal route from the MCU to the crystal.

In normal mode with crystal reference, the FMPLL can generate a frequency modulated clock or a non-modulated clock (locked on a single frequency). The modulation rate, modulation depth, output divider (RFD) and whether the FMPLL is modulating or not can be programmed by writing to the FMPLL registers.

Normal mode with external reference

The normal mode with external reference functions the same as normal mode with crystal reference, except that the input clock reference to the FMPLL is driven by an external clock generator connected to the EXTAL pin, rather than a crystal oscillator. The input frequency range is the same and frequency modulation is available.

16.3 External signal description

Table 280 lists external signals used by the FMPLL during normal operation.

Table 280. Signal properties

Name	Function	I/O	Pull
PLLREF	Configures the FMPLL clock reference at reset	I/O	Up
XTAL	Output drive for external crystal	O	—
EXTAL_EXTCLK	Crystal/external clock input	I/O	—
VDDPLL	Analog power supply (1.2V +/– 10%)	Power	—
VSSPLL(1)	Analog ground	Ground	—

1. This signal is internally bonded to VSS.

16.3.1 Detailed signal descriptions

Table 281 describes the external signals used by the FMPLL.

Table 281. FMPLL detailed signal descriptions

<table>
<tr><th>Signal</th><th>I/O</th><th colspan="2">Description</th></tr>
<tr><td rowspan="3">PLLREF</td><td rowspan="3">I/O</td><td colspan="2">PLL reference—Determines the reset state of the CLKCFG[2] bit in FMPLL_ESYNCR1. The PLLREF pin must be kept stable during system reset. After reset, this pin has no effect on the PLL configuration, therefore it can be assigned to another function such as GPIO.</td></tr>
<tr><td>State meaning</td><td>Asserted—Indicates that the reference clock comes from the crystal oscillator.
Negated—Indicates that the reference clock comes from the external clock generator.</td></tr>
<tr><td>Timing</td><td>Assertion or negation—Must be done at the beginning of the reset cycle and then kept stable for the whole reset duration.</td></tr>
<tr><td>XTAL</td><td>O</td><td colspan="2">Crystal oscillator—Output for an external crystal oscillator.</td></tr>
<tr><td>EXTAL_EXTCLK</td><td>I/O</td><td colspan="2">Crystal oscillator/external clock input—This pin is the input for an external crystal oscillator or an external clock source. The function of this pin is determined by the CLKCFG[2] bit in FMPLL_ESYNCR1, whose reset value is determined by the PLLREF pin.</td></tr>
<tr><td>VDDPLL / VSSPLL</td><td>—</td><td colspan="2">PLL power supply—These are the 1.2V supply and ground for the FMPLL.</td></tr>
</table>

16.4 Memory map and register definition

This section provides the memory map and detailed descriptions of all registers of the FMPLL.

16.4.1 Memory map

Table 282 shows the memory map. Addresses are given as offsets of the module base address.

Table 282. FMPLL memory map

Offset	Register	Location
0x0000	Synthesizer Control Register (SYNCR)	*on page 16-483*
0x0004	Synthesizer Status Register (SYNSR)	*on page 16-486*
0x0008	Enhanced Synthesizer Control Register 1 (ESYNCR1)	*on page 16-488*
0x000C	Enhanced Synthesizer Control Register 2 (ESYNCR2)	*on page 16-490*
0x0010	Reserved	—
0x0014	Reserved	—
0x0018	Synthesizer FM Modulation Register (SYNFMMR)	*on page 16-491*

16.4.2 Register descriptions

This section contains the register descriptions in ascending address order. Two different programming models are selectable through FMPLL_ESYNCR1[EMODE]:

- Legacy model—The FMPLL is controlled by the Synthesizer Control Register (SYNCR). In this model, the FMPLL operating mode changes automatically to normal mode when the register is written in the first time. There is no way to switch back to bypass mode once the operating mode has switched to normal.
- Enhanced model—The PLL is controlled by the Enhanced Synthesizer Control Registers 1–2 (ESYNCR1/ESYNCR2). In this model, it is possible to change the FMPLL operating mode back and forth between bypass and normal modes by programming FMPLL_ESYNCR1[CLKCFG].

The reset value of FMPLL_ESYNCR1[EMODE] is determined by the SoC integration. This bit is write once. After it is set to ‘1’, further write attempts to this bit will have no effect.

Synthesizer Control Register (SYNCR)

This register is provided for backwards compatibility with previous devices. New applications should use ESYNCR1/ESYNCR2 instead of SYNCR.

Figure 283. Synthesizer Control Register (SYNCR)

Offset 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	PREDIV			MFD					0	RFD			LOC EN	LOL RE	LOC RE
W																
Reset	0	–(1)	–(1)	–(1)	–(1)	–(1)	–(1)	–(1)	–(1)	0	–(1)	–(1)	–(1)	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	LOL IRQ	LOC IRQ	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1. Reset value is determined by the SoC integration.

Table 283. SYNCR field descriptions

Field	Description
0	Reserved, should be cleared.
1–3 PREDIV	Predivider This 3-bit field controls the value of the divider on the input clock. The output of the predivider circuit generates the reference clock to the FMPLL analog loop. The value 111 causes the input clock to be inhibited. 000 Divide by 1 001 Divide by 2 010 Divide by 3 011 Divide by 4 100 Divide by 5 101 Divide by 6 110 Divide by 7 111 Clock inhibit
4–8 MFD	Multiplication factor divider This 5-bit field controls the value of the divider in the FMPLL feedback loop. The value specified by the MFD bits establishes the multiplication factor applied to the reference frequency. 000xx Invalid 00100 Divide by 8 00101 Divide by 9 00110 Divide by 10 ... 10011 Divide by 23 10100 Divide by 24 10101 Invalid 1011x Invalid 11xxx Invalid
9	Reserved, should be cleared.

Table 283. SYNCR field descriptions (continued)

Field	Description
10–12 RFD	Reduced frequency divider This 3-bit field controls a divider at the output of the FMPLL. The value specified by the RFD bits establishes the division factor applied to the FMPLL frequency. 000 Divide by 1 001 Divide by 2 010 Divide by 4 011 Invalid 1xx Invalid
13 LOCEN	Loss-of-clock enable The LOCEN bit determines if the loss-of-clock function is operational. This bit only has effect in normal mode. In bypass mode, the loss-of-clock function is always enabled, regardless of the state of the LOCEN bit. Furthermore, the LOCEN bit has no effect on the loss-of-lock detection circuitry. 0 Loss of clock disabled 1 Loss of clock enabled
14 LOLRE	Loss-of-lock reset enable The LOLRE bit determines whether system reset is asserted or not upon a loss-of-lock indication. When operating in normal mode, the FMPLL must be locked before setting the LOLRE bit, otherwise reset is immediately asserted. Note that once reset is asserted, the operating mode is switched to bypass mode, and once in bypass, a loss-of-lock condition does not generate reset, regardless of the value of the LOLRE bit. See *Section 16.5.3, Lock detection*. 0 Ignore loss-of-lock. Reset not asserted. 1 Assert reset on loss-of-lock when operating in normal mode.
15 LOCRE	Loss-of-clock reset enable The LOCRE bit determines whether system reset is asserted or not upon a loss-of-clock condition when LOCEN = 1. LOCRE has no effect when LOCEN = 0. If the LOCF bit in the SYNSR indicates a loss-of-clock condition, setting the LOCRE bit causes immediate reset. In bypass mode with crystal reference, reset will occur if the reference clock fails, even if LOCRE = 0 or even if LOCEN = 0. The LOCRE bit has no effect in bypass mode with external reference. In this mode, the reference clock is not monitored at all. See *Section , Loss-of-clock reset*. 0 Ignore loss-of-clock. Reset not asserted. 1 Assert reset on loss-of-clock.
16	Reserved, should be cleared.
17 LOLIRQ	Loss-of-lock interrupt request The LOLIRQ bit enables a loss-of-lock interrupt request when the LOLF flag is set. If either LOLF or LOLIRQ is negated, the interrupt request is negated. When operating in normal mode, the FMPLL must be locked before setting the LOLIRQ bit, otherwise an interrupt is immediately asserted. The interrupt request only happens in normal mode, therefore the LOLIRQ bit has no effect in bypass mode. See *Section 16.5.3, Lock detection*. 0 Ignore loss-of-lock. Interrupt not requested. 1 Enable interrupt request upon loss-of-lock.

Table 283. SYNCR field descriptions (continued)

Field	Description
18 LOCIRQ	Loss-of-clock interrupt request The LOCIRQ bit enables a loss-of-clock interrupt request when the LOCF flag is set. If either LOCF or LOCIRQ is negated, the interrupt request is negated. If loss-of-clock is detected while in bypass mode, a system reset is generated. Therefore, LOCIRQ has no effect in bypass mode. See *Section , Loss-of-clock interrupt request*. 0 Ignore loss-of-clock. Interrupt not requested. 1 Enable interrupt request upon loss-of-clock.
19–31	Reserved, should be cleared.

Synthesizer Status Register (SYNSR)

Figure 284. Synthesizer Status Register (SYNSR)

Offset 0x0004 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	LOLF	LOC	MODE	PLL SEL	PLL REF	LOCKS	LOCK	LOCF	0	0
W							w1c							w1c		
Reset	0	0	0	0	0	0	0	0	0	0	–(1)	0	0	0	0	0

1. Reset value is determined by the state of the PLLREF pin.

Table 284. SYNSR field descriptions

Field	Description
0–21	Reserved, should be cleared.
22 LOLF	Loss-of-lock flag This bit provides the interrupt request flag for the loss-of-lock. To clear the flag, software must write a 1 to the bit. Writing 0 has no effect. This flag bit is sticky in the sense that if lock is reacquired, the bit will remain set until cleared by either writing 1 or asserting reset. It will not be asserted when lock is lost due to system reset, write to the FMPLL_SYNCR in legacy mode which modifies the PREDIV or MFD fields, or write to FMPLL_ESYNCR1 in enhanced mode which modifies the EMODE, EPREDIV, EMFD or CLKCFG[1:0] fields. Furthermore, it is not asserted if the loss-of-lock condition was detected while the FMPLL is in bypass mode. Nevertheless, going from normal to bypass will not automatically clear the flag if it was asserted while the FMPLL was in normal mode. See *Section 16.5.3, Lock detection*. 0 No loss of lock detected. Interrupt service not requested. 1 Loss of lock detected. Interrupt service requested.

Table 284. SYNSR field descriptions (continued)

Field	Description
23 LOC	Loss-of-clock This bit is an indication of whether a loss-of-clock condition is present. If LOC = 0, the system clocks are operating normally. If LOC = 1, the system clocks have failed due to a reference or VCO failure. If a loss-of-clock condition occurs which sets this bit and the clocks later return to normal, this bit will be cleared. A loss-of-clock condition can only be detected if LOCEN = 1. Furthermore, the LOC bit is not asserted when the FMPLL is in bypass mode (because, in bypass, the VCO clock is not monitored and a loss-of-clock on the reference clock causes reset). See *Section 16.5.4, Loss-of-clock detection*. 0 No loss-of-clock detected. Clocks are operating normally. 1 Loss-of-clock detected. Clocks are not operating normally.
24 MODE	Mode of operation This bit indicates whether the FMPLL is working in bypass mode or normal mode. The reset value indicates bypass mode. In legacy mode (FMPLL_ESYNCR1[EMODE] negated), the MODE bit will change to normal mode at the first time the FMPLL_SYNCR is written. In enhanced mode (FMPLL_ESYNCR1[EMODE] asserted), the MODE bit reflects the value of the CLKCFG[0] bit of the FMPLL_ESYNCR1. 0 Bypass mode 1 Normal mode
25 PLLSEL	Mode select Dual controller mode is not supported, therefore in legacy mode this bit resets to ‘0’ (bypass), but changes to ‘1’ (normal mode) at the first time the FMPLL_SYNCR is written. In enhanced mode, the MODE bit reflects the value of the CLKCFG[1] bit of the FMPLL_ESYNCR1. 0 Legacy mode: bypass or dual controller; enhanced mode: PLL off 1 Legacy mode: normal; enhanced mode: PLL on
26 PLLREF	FMPLL reference source This bit indicates whether the FMPLL reference is from a crystal oscillator or from an external clock generator. The reset value is determined by the state of the PLLREF pin. In legacy mode, the reset value captured from the PLLREF pin cannot be changed anymore after reset. In enhanced mode, the PLLREF bit reflects the value of the CLKCFG[2] bit of the FMPLL_ESYNCR1. 0 External clock reference 1 Crystal oscillator reference
27 LOCKS	Sticky FMPLL lock status bit This bit is set by the lock detect circuitry when the FMPLLL acquires lock after one of the following: – A system reset – A write to the FMPLL_SYNCR in legacy mode which changes the PREDIV or MFD fields – A write to the FMPLL_ESYNCR1 in enhanced mode which changes the EMODE, EPREDIV, EMFD or CLKCFG[1:2] fields Whenever the FMPLL loses lock, LOCKS is cleared. LOCKS remains cleared even after the FMPLL relocks, until one of the three previously stated conditions occurs. Coming in bypass mode from system reset, LOCKS is asserted as soon as the FMPLL has locked, even if normal mode was not entered yet. If the FMPLL is locked, going from normal to bypass mode does not clear the LOCKS bit. 0 FMPLL has lost lock since last system reset or last write to PLL registers which affect the lock status. 1 FMPLL has not lost lock.

Table 284. SYNSR field descriptions (continued)

Field	Description
28 LOCK	FMPLL lock status bit Indicates whether the FMPLL has acquired lock. FMPLL lock occurs when the synthesized frequency matches to within approximately 4% of the programmed frequency. The FMPLL loses lock when a frequency deviation of greater than approximately 16% occurs. The flag is also immediately negated when the PREDIV or MFD fields of the SYNCR are changed in legacy mode, or when EMODE, EPREDIV, EMFD or CLKCFG[1:2] are changed in enhanced mode, and then asserted again when the PLL regains lock. If operating in bypass mode, the LOCK bit is still asserted or negated when the FMPLL acquires or loses lock. 0 FMPLL is unlocked. 1 FMPLL is locked.
29 LOCF	Loss-of-clock flag This bit provides the interrupt request flag for the loss-of-clock. To clear the flag, software must write a 1 to the bit. Writing 0 has no effect. This flag bit is sticky in the sense that if clocks return to normal, the bit will remain set until cleared by either writing 1 or asserting reset. The LOCF flag is not asserted while the FMPLL is in bypass mode. See *Section 16.5.4, Loss-of-clock detection*, for information on which operating modes and conditions can this flag be asserted. 0 No loss of clock detected. Interrupt service not requested. 1 Loss of clock detected. Interrupt service requested.
30–31	Reserved, should be cleared.

Enhanced Synthesizer Control Register 1 (ESYNCR1)

Offset 0x0008 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	EMODE	CLKCFG			0	0	0	0	0	0	0	0	EPREDIV			
W																
Reset	0	0	1	–[1]	0	0	0	0	0	0	0	0	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	EMFD						
W																
Reset	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0

---

[1] Reset value determined by the PLLREF pin.

Figure 285. Enhanced Synthesizer Control Register 1 (ESYNCR1)

Table 285. ESYNCR1 field descriptions

Field	Description
0 EMODE	Enhanced mode enable This bit determines whether the FMPLL will be controlled by SYNCR or ESYNCR1/ESYNCR2. At SoC integration, a signal tie will dictate the default state that the PLL operates. If the SoC integration ties the FMPLL to run in enhanced mode, the EMODE bit will reflect this by reading a logic 1. Additionally, software writes to this bit to revert to legacy mode will not be allowed. If the signal is tied to select legacy mode as the default state, the EMODE bit will reflect this by reading a logic 0. In this case, software writes to this bit to enable enhanced mode is allowed, but it is a write once operation. After written to ‘1’, further write attempts to this bit will have no effect. 0 Legacy mode. FMPLL controlled by SYNCR. 1 Enhanced mode. FMPLL controlled by ESYNCR1/ESYNCR2.
1–3 CLKCFG	Clock configuration This 3-bit field is used to change the operating mode of the FMPLL. Bit 2 is not writable to ‘0’ while bit 1 is ‘1’. The reset state of bit 3 is determined by the state of the PLLREF pin. 000 Bypass mode with external reference and PLL off 001 Bypass mode with crystal reference and PLL off 010 Bypass mode with external reference and PLL running 011 Bypass mode with crystal reference and PLL running 100 Reserved 101 Reserved 110 Normal mode with external reference 111 Normal mode with crystal reference
4–11	Reserved, should be cleared.
12–15 EPREDIV	Enhanced predivider This 4-bit field controls the value of the divider on the input clock. The output of the predivider circuit generates the reference clock to the PLL analog loop. The PREDIV value 1111 causes the input clock to be inhibited. 0000 Divide by 1 0001 Divide by 2 0010 Divide by 3 0011 Divide by 4 0100 Divide by 5 0101 Divide by 6 0110 Divide by 7 0111 Divide by 8 1000 Divide by 9 1001 Divide by 10 1010 Divide by 11 1011 Divide by 12 1100 Divide by 13 1101 Divide by 14 1110 Divide by 15 1111 Clock inhibit

Table 285. ESYNCR1 field descriptions (continued)

Field	Description
16–24	Reserved, should be cleared.
25–31 EMFD	Enhanced multiplication factor divider This 7-bit field controls the value of the divider in the FMPLL feedback loop. The value specified by the EMFD bits establishes the multiplication factor applied to the reference frequency. The valid range of multiplication factors is 32 (010_0000) to 96 (110_0000). Values outside this range are invalid and will cause the FMPLL to produce unpredictable clock output. 00x_xxxx Invalid 010_0000 Divide by 32 010_0001 Divide by 33 ... 101_1111 Divide by 95 110_0000 Divide by 96 110_0001 Invalid ... 111_1111 Invalid

Enhanced Synthesizer Control Register 2 (ESYNCR2)

Figure 286. Enhanced Synthesizer Control Register 2 (ESYNCR2)

Offset 0x000C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	LOCEN	LOLRE	LOCRE	LOLIRQ	LOCIRQ	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	ERFD	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

Table 286. ESYNCR2 field descriptions

Field	Description
0–7	Reserved, should be cleared.
8 LOCEN	Loss-of-clock enable The LOCEN bit determines if the loss-of-clock function is operational. This bit only has effect in normal mode. In bypass mode, the loss-of-clock function is always enabled, regardless of the state of the LOCEN bit. Furthermore, the LOCEN bit has no effect on the loss-of-lock detection circuitry. 0 Loss of clock disabled 1 Loss of clock enabled

Table 286. ESYNCR2 field descriptions (continued)

Field	Description
9 LOLRE	Loss-of-lock reset enable The LOLRE bit determines whether system reset is asserted or not upon a loss-of-lock indication. When operating in normal mode, the FMPLL must be locked before setting the LOLRE bit, otherwise reset is immediately asserted. Note that once reset is asserted, the operating mode is switched to bypass mode, and once in bypass, a loss-of-lock condition does not generate reset, regardless of the value of the LOLRE bit. See *Section 16.5.3, Lock detection*. 0 Ignore loss-of-lock. Reset not asserted. 1 Assert reset on loss-of-lock when operating in normal mode.
10 LOCRE	Loss-of-clock reset enable The LOCRE bit determines whether system reset is asserted or not upon a loss-of-clock condition when LOCEN = 1. LOCRE has no effect when LOCEN = 0. If the LOCF bit in the SYNSR indicates a loss-of-clock condition, setting the LOCRE bit causes immediate reset. In bypass mode with crystal reference, reset will occur if the reference clock fails, even if LOCRE = 0 or even if LOCEN = 0. The LOCRE bit has no effect in bypass mode with external reference. In this mode, the reference clock is not monitored at all. See *Section , Loss-of-clock reset*. 0 Ignore loss-of-clock. Reset not asserted. 1 Assert reset on loss-of-clock.
11 LOLIRQ	Loss-of-lock interrupt request The LOLIRQ bit enables a loss-of-lock interrupt request when the LOLF flag is set. If either LOLF or LOLIRQ is negated, the interrupt request is negated. When operating in normal mode, the FMPLL must be locked before setting the LOLIRQ bit, otherwise an interrupt is immediately asserted. The interrupt request only happens in normal mode, therefore the LOLIRQ bit has no effect in bypass mode. See *Section 16.5.3, Lock detection*. 0 Ignore loss-of-lock. Interrupt not requested. 1 Enable interrupt request upon loss-of-lock.
12 LOCIRQ	Loss-of-clock interrupt request The LOCIRQ bit enables a loss-of-clock interrupt request when the LOCF flag is set. If either LOCF or LOCIRQ is negated, the interrupt request is negated. If loss-of-clock is detected while in bypass mode, a system reset is generated. Therefore, LOCIRQ has no effect in bypass mode. See *Section , Loss-of-clock interrupt request*. 0 Ignore loss-of-clock. Interrupt not requested. 1 Enable interrupt request upon loss-of-clock.
13–29	Reserved, should be cleared.
30–31 ERFD	Enhanced reduced frequency divider This 2-bit field controls a divider at the output of the FMPLL. The value specified by the ERFD bits establishes the division factor applied to the FMPLL frequency. 00 Divide by 2 01 Divide by 4 10 Divide by 8 11 Divide by 16

Synthesizer FM Modulation Register (SYNFMMR)

This register controls the frequency modulation (FM) features of the FMPLL. FM is not backwards compatible with previous devices. Therefore, this register must be used in enhanced mode. It can only be programmed when the FMPLL is locked. Writing to this register while the FMPLL is unlocked has no effect. Furthermore, when the PLL loses lock, frequency modulation is disabled and the FMPLL_SYNFMMR is reset.

Figure 287. Synthesizer FM Modulation Register (SYNFMMR)

Offset 0x0018 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	BSY	MOD EN	MOD SEL	MODPERIOD												
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	INCSTEP														
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 287. SYNFMMR field descriptions

Field	Description
0 BSY	Busy This bit is asserted soon after a write access to the FMPLL_SYNFMMR, and remains asserted while the FMPLL processes the new frequency modulation programming. The CPU must wait until this bit is negated before attempting another write access to this register. Any write attempt while the BSY flag is set will have no effect. 0 Write to the FMPLL_SYNFMMR is allowed. 1 The FMPLL is still busy processing the previous change on the FMPLL_SYNFMMR; write access to the register is not possible.
1 MODEN	Modulation enable This bit enables the frequency modulation. 0 Frequency modulation disabled 1 Frequency modulation enabled
2 MODSEL	Modulation selection This bit selects whether modulation will be centered around the nominal frequency or spread below the nominal frequency. 0 Modulation centered around nominal frequency. 1 Modulation spread below nominal frequency.
3–15 MODPERIOD	Modulation period This 13-bit field is the binary equivalent of the modperiod variable derived from the formula: $\text{modperiod} = \text{round}\left(\frac{f_{fbk}}{4 \times f_{mod}}\right)$ where f_{fbk} represents the frequency of the feedback divider, and f_{mod} represents the modulation frequency.

Table 287. SYNFMMR field descriptions (continued)

Field	Description
16	Reserved, should be cleared.
17–31 INCSTEP	Increment step This 14-bit field is the binary equivalent of the incstep variable derived from the formula: $\text{incstep} = \text{round}\left(\frac{(2^{15}-1)\times MD \times EMFD}{100\times 5\times \text{modperiod}}\right)$ where MD represents the peak modulation depth in percentage (+/–MD for centered modulation, –2 * MD for modulation below nominal frequency), and EMFD represents the nominal value of the feedback loop divider.

Note: The product of INCSTEP and MODPERIOD cannot be larger than $(2^{15} - 1)$.

16.5 Functional description

This section explains the FMPLL operation and configuration.

16.5.1 Input clock frequency

The FMPLL is designed to operate over an input clock frequency range as determined by the operating mode. The operating ranges for each mode are given in *Table 288*.

Table 288. Input clock frequency at the predivider input

Mode	Input frequency range
Bypass mode with crystal reference Normal mode with crystal reference	4 MHz – 20/40 MHz(1)
Bypass mode with external reference Normal mode with external reference	0 Hz – f_{sys}(2)

1. See *Section 16.1, Information specific to this device*, for information on crystal frequencies supported.
2. f_{sys} is the system frequency of the MCU. The predivider ratio has to be chosen such that the input to the PLL itself (after the predivider) does not exceed 16 MHz.

16.5.2 Clock configuration

In legacy mode, the relationship between the output frequency f_{sys} and input frequency f_{ref} is determined by the PREDIV, MFD and RFD values programmed in the FMPLL_SYNCR, according to the following equation:

Equation 3

$$f_{sys} = f_{ref} \times \frac{MFD + 4}{(PREDIV + 1) \times 2^{RFD}}$$

In legacy mode, the relationship between the VCO frequency f_{VCO} and the output frequency f_{sys} is determined by the value of the RFD value programmed in the SYNCR register, according to the following equation:

Equation 4

$$f_{VCO} = 4 \times f_{sys} \times 2^{RFD}$$

In enhanced mode, the relationship between input and output frequency is determined by the EPREDIV, EMFD and ERFD values programmed in the FMPLL_ESYNCR1 and FMPLL_ESYNCR2, according to the following equation:

Equation 5

$$f_{sys} = f_{ref} \times \frac{EMFD}{(EPREDIV + 1) \times 2^{(ERFD + 1)}}$$

When programming the FMPLL, be sure not to violate the maximum system clock frequency or max/min VCO frequency specification. In enhanced mode, the VCO frequency is calculated according to the following equation:

Equation 6

$$f_{VCO} = f_{ref} \times \frac{EMFD}{(EPREDIV + 1)}$$

Note: *Maximum system clock frequency is 150 MHz and max/min VCO frequency is 256 MHz to 512 MHz.*

Furthermore, the PREDIV or EPREDIV values must not be set to any value that causes the input frequency to the phase detector to go below 4 MHz.

The LOCK flag is immediately negated after any of the following events:

1. In legacy mode, the PREDIV or MFD fields of the FMPLL_SYNCR are changed
2. In enhanced mode, the EMODE, EPREDIV, EMFD of CLKCFG[1:2] fields of the FMPLL_ESYNCR1 are changed(l)

Upon any of these events an internal timer is initialized to count 64 cycles of the PLL input clock. During this period, the LOCK flag is held negated. After the timer expires, the LOCK

---

l. Note that changing only the CLKCFG[0] bit to move from bypass to normal or vice-versa, and keeping the values of the other FMPLL_ESYNCR1 fields unchanged, will not cause the PLL to lose lock or the lock flag to be cleared.

flag reflects the value coming from the PLL lock detection circuitry. To prevent an immediate reset, the LOLRE bit must be cleared before doing any of the above operations.

Changing RFD or ERFD does not affect the FMPLL, hence no relock delay is incurred. Resulting changes in clock frequency are synchronized to the next falling edge of the current system clock. However, RFD or ERFD should only be changed when the LOCK bit is set, to avoid exceeding the allowable system operating frequency.

Coming out of reset, the FMPLL will be enabled (on), but running in bypass mode. The recommended procedure to program the FMPLL and engage normal mode is:

1. Assert the EMODE bit and program the EPREDIV and EMFD fields of FMPLL_ESYNCR1 and the RFD field of FMPLL_ESYNCR2.
2. Poll FMPLL_SYNSR[LOCK] until it asserts.
3. If required, program the FMPLL_SYNFMMR with desired FM parameters, poll the BSY bit until it negates, then enable FM by asserting the MODEN bit.
4. Engage normal mode by writing to FMPLL_ESYNCR1[CLKCFG].

16.5.3 Lock detection

A pair of counters monitor the reference and feedback clocks to determine when the system has acquired frequency lock. Once the FMPLL has locked, the counters continue to monitor the reference and feedback clocks and will report if/when the FMPLL has lost lock. The FMPLL registers provide the flexibility to select whether to generate an interrupt, assert system reset or do nothing in the event that the FMPLL loses lock.

Loss-of-lock reset and interrupt are only generated when the FMPLL is operating in normal mode. The LOCF bit is not asserted by a loss-of-lock condition detected during bypass, although going to bypass mode from normal mode does not automatically clear the flag if it was asserted while the FMPLL was in normal mode.

16.5.4 Loss-of-clock detection

The FMPLL reference and output clocks may be continuously monitored by a module called Clock Quality Monitor (CQM), shown in *Figure 288*. The intent of the CQM is to assure that the system bus clock is created from good clock sources. Whether the clocks are monitored or not is determined by the clock operating mode and control bits in the FMPLL registers, as shown in *Table 289*.

In bypass mode with crystal reference, the reference clock is always monitored, regardless of the state of the LOCEN bit. In bypass mode with external reference, the reference clock is not monitored, regardless of the state of the LOCEN bit. This is done so that the whole device frequency range can be sourced from the external clock generator when using external reference mode. The FMPLL output may only monitored in normal mode, depending on the state of the LOCEN bit.

The clock quality monitor uses an internal 4 MHz RC oscillator as a reference time base to measure the frequency of the crystal oscillator and the FMPLL output. The frequency of these clocks are expected to be within the following frequency ranges:

- Reference clock must be within the crystal frequency range(m)
- PLL output must be above 1.5 MHz (minimum VCO free-running frequency divided by the maximum ERFD)

In the event either of the clocks fall outside the expected window, a loss of clock condition is reported. The FMPLL can be programmed to switch the system clock to a backup clock in the event of such a failure. Additionally, the user may select to have the system enter reset, assert an interrupt request, or do nothing if/when the FMPLL reports this condition.

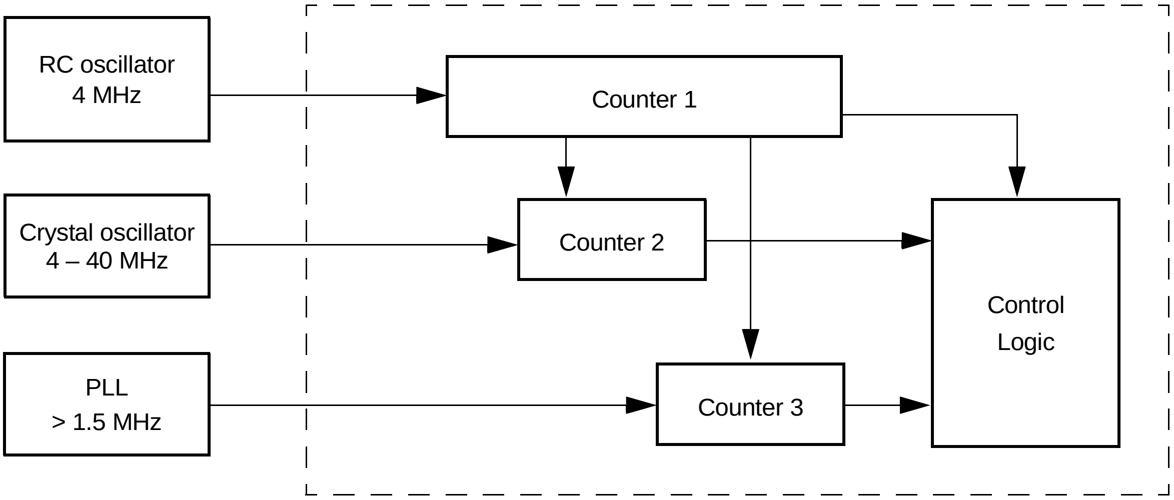


Figure 288. Clock quality monitor

Table 289. Loss-of-clock monitoring

Operating mode	LOCEN[1]	Reference clock monitored?	FMPLL output monitored?
Bypass mode with external reference and PLL off	—	No	No
Bypass mode with crystal reference and PLL off	—	Yes	No
Bypass mode with external reference and PLL running	—	No	No
Bypass mode with crystal reference and PLL running	—	Yes	No
Normal mode with external reference	0	No	No
	1	No	Yes
Normal mode with crystal reference	0	No	No
	1	Yes	Yes

1. LOCEN is the loss-of-clock enable bit in either FMPLL_SYNCR or FMPLL_ESYNCR2, depending on FMPLL_ESYNCR1[EMODE].

Alternate/Backup clock selection

If loss-of-clock detection is enabled by LOCEN, the FMPLL is operating in normal mode and the Clock Quality Monitor detects a failure at the FMPLL output clock, then a backup clock selection feature automatically connects the system clock to the reference clock input (either external or crystal reference). After this happens, the system clock remains connected to the reference clock until next system reset, even if the FMPLL regains itself and relocks. If,

---

m. See *Section 16.1, Information specific to this device*, for information on crystal frequencies supported.

however, the reference clock also fails, either simultaneously or after the FMPLL failure, the system clock is connected back to the FMPLL output.

If the reference fails in normal mode, then no backup clock selection occurs, and the FMPLL output continues to be the system clock. If the reference stops, the FMPLL will operate in free-running mode.

In bypass mode with crystal reference, a reference fail will force a reset. In bypass mode with external reference, no backup clock selection occurs if the reference fails.

Loss-of-clock reset

When a loss-of-clock condition is recognized, a system reset may be asserted depending on the clock operating mode and control bits in the FMPLL registers, as shown in *Table 290*. FMPLL_SYNSR[LOCF] and FMPLL_SYNSR[LOC] are cleared after reset, therefore, another means must be used externally to determine that a loss-of-clock condition occurred.

Table 290. Loss-of-clock reset

Operating mode	LOCEN (1)	LOCRE (2)	Reset	
			Reference failure	FMPLL failure
Bypass mode with external reference and PLL off	—	—	No	No
Bypass mode with crystal reference and PLL off	—	—	Yes	No
Bypass mode with external reference and PLL running	—	—	No	No
Bypass mode with crystal reference and PLL running	—	—	Yes	No
Normal mode with external reference	0	—	No	No
	1	0	No	No
	1	1	No	Yes
Normal mode with crystal reference	0	—	No	No
	1	0	No	No
	1	1	Yes	Yes

1. LOCEN is the loss-of-clock enable bit in either FMPLL_SYNCR or FMPLL_ESYNCR2, depending on FMPLL_ESYNCR1[EMODE].
2. LOCRE is the loss-of-clock reset enable bit in either FMPLL_SYNCR or FMPLL_ESYNCR2, depending on FMPLL_ESYNCR1[EMODE].

LOCEN and LOCRE have no effect in bypass mode. If the reference fails while the FMPLL is in bypass mode with crystal reference, a system reset is asserted regardless of the state of LOCEN and LOCRE. Since bypass is the FMPLL reset mode, the crystal oscillator must be present and functioning properly to exit reset when PLLREF = 1. When PLLREF = 0, the reference clock is not checked for loss-of-clock, so exit from reset can happen regardless the state of the reference clock. Exit from reset is not affected by the state of the FMPLL output because the FMPLL clock is not monitored in bypass mode.

Loss-of-clock interrupt request

When a loss-of-clock condition is recognized, an interrupt request may be asserted depending on the clock operating mode and control bits in the FMPLL registers, as shown in

Table 291.

LOCEN and LOCIRQ have no effect in bypass mode. If the reference fails in bypass mode with crystal reference, a system reset is asserted instead of an interrupt request. If the reference fails in bypass with external reference, no reset or interrupts are generated. Furthermore, no reset or interrupts are generated when lock is lost due to a write to the FMPLL_SYNCR in legacy mode which modifies the PREDIV or MFD fields, or a write to FMPLL_ESYNCR1 in enhanced mode which modifies the EMODE, EPREDIV, EMFD or CLKCFG[1:0] fields.

Table 291. Loss-of-clock interrupt request

Operating mode	LOCEN (1)	LOCIRQ (2)	Interrupt request	
			Reference failure	FMPLL failure
Bypass mode with external reference and PLL off	—	—	—	—
Bypass mode with crystal reference and PLL off	—	—	No	—
Bypass mode with external reference and PLL running	—	—	—	—
Bypass mode with crystal reference and PLL running	—	—	No	—
Normal mode with external reference	0	—	—	No
	1	0	—	No
	1	1	—	Yes
Normal mode with crystal reference	0	—	No	No
	1	0	No	No
	1	1	Yes	Yes

1. LOCEN is the loss-of-clock enable bit in either FMPLL_SYNCR or FMPLL_ESYNCR2, depending on the FMPLL_ESYNCR1[EMODE].
2. LOCIRQ is the loss-of-clock interrupt enable bit in either FMPLL_SYNCR or FMPLL_ESYNCR2, depending on the FMPLL_ESYNCR1[EMODE].

16.5.5 Frequency modulation

Frequency modulation uses a triangular profile as shown in *Figure 289*. The modulation frequency and depth are set using the MODPERIOD and INCSTEP fields of the FMPLL_SYNFMMR.

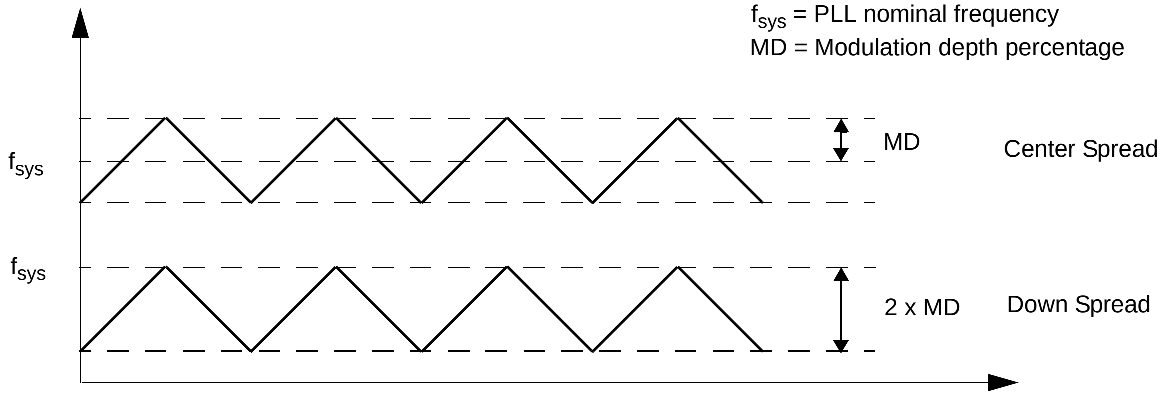


Figure 289. Triangular frequency modulation

The following equations define how to calculate MODPERIOD and INCSTEP based on the frequency of the feedback divider (f_{fbk}), the modulation frequency (f_{mod}) and the modulation depth percentage (MD):

Equation 7

$$\mathrm{MODPERIOD} = \mathrm{round}\left(\frac{f_{fbk}}{4 \times f_{mod}}\right)$$

Equation 8

$$\mathrm{INCSTEP} = \mathrm{round}\left(\frac{(2^{15} - 1) \times \mathrm{MD} \times \mathrm{EMFD}}{100 \times 5 \times \mathrm{MODPERIOD}}\right)$$

MODPERIOD and INCSTEP are subject to the following restriction:

Equation 9

$$(\mathrm{MODPERIOD} \times \mathrm{INCSTEP}) < 2^{15}$$

Because of the above rounding operations, the effective modulation depth applied to the FMPLL is given by the following formula:

Equation 10

$$\mathrm{INCSTEP} = \mathrm{round}\left(\frac{\mathrm{MODPERIOD} \times \mathrm{INCSTEP} \times 100 \times 5}{(2^{15} - 1) \times \mathrm{EMFD}}\right)$$

As an example, suppose the following configuration:

- Input frequency: 4 MHz
- Load divider (EMFD): 64
- Input divider: 1
- VCO frequency: 4 MHz × 64 = 256 MHz
- PLL output frequency: 256 MHz / ERFD
- Center spread (MODSEL = 0)
- Modulation frequency: 24 kHz
- Modulation depth: +/– 2.0 % (4% peak-to-peak)
- MODPERIOD = Round $[(4 \times 10^6)/(4 \times 24 \times 10^3)]$ = Round [41.66] = 42
- INCSTEP = Round $[((2^{15} - 1) \times 2 \times 64) / (100 \times 5 \times 42)]$ = Round [199.722] = 200
- MODPERIOD × INCSTEP = 42 × 200 = 8400 (which is less than 2^{15})
- MD (quantized) = $((42 \times 200 \times 100 \times 5) / ((2^{15} - 1) \times 64))$ = 2.00278 %

In this example, the modulation depth error is 0.00278%.

The FM parameters can only be changed, and FM can only be enabled, when the PLL is locked. Writing to the FMPLL_SYNFMMR while the PLL is unlocked has no effect. Furthermore, when the PLL loses lock, the FM parameters are reset and the modulation is disabled until the PLL relocks and the FMPLL_SYNFMMR is programmed again.

After programming the FM parameters, it takes some time until these parameters get propagated to the PLL analog circuitry. During this time, the BSY bit gets asserted. The modulation must only be enabled when the FM parameters have already propagated to the analog circuitry. Therefore, the sequence for programming FM is:

1. Poll FMPLL_SYNSR[LOCK] until it asserts.
2. Program the MODSEL, MODPERIOD and INCSTEP fields of the FMPLL_SYNFMMR.
3. Poll FMPLL_SYNFMMR[BSY] until it negates.
4. Assert FMPLL_SYNFMMR[MODEN].

17 Error Correction Status Module (ECSM)

17.1 Overview

The Error Correction Status Module (ECSM) provides control functions regarding information on memory errors reported by error-correcting codes. The ECSM is mapped into the IPS space and supports a number of miscellaneous control functions for the platform.

17.2 Features

- Program-visible information on the platform configuration and revision
- Optional address map for device’s crossbar switch (XBAR)
- Miscellaneous Reset Status Register (ECSM_MRSR)
- Registers for capturing information on memory errors if error-correcting codes (ECC) are implemented

17.3 Module memory map

The Error Correction Status Module does not include any logic that provides access control. Rather, this function is supported using the standard access control logic provided by the IPS controller.

Table 292 is a 32-bit view of the ECSM’s memory map.

Table 292. ECSM 32-bit memory map

<table>
<tr><th>ECSM Offset</th><th colspan="3">Register</th></tr>
<tr><td>0x00</td><td>Reserved</td><td colspan="2">Reserved</td></tr>
<tr><td>0x04</td><td>Reserved</td><td colspan="2">Reserved</td></tr>
<tr><td>0x08</td><td colspan="3">Reserved</td></tr>
<tr><td>0x0C</td><td colspan="2">Reserved</td><td>Miscellaneous Reset Status Register (ECSM_MRSR)</td></tr>
<tr><td>0x10</td><td colspan="2">Reserved</td><td>Miscellaneous Wakeup Control Register (ECSM_MWCR)</td></tr>
<tr><td>0x14</td><td colspan="3">Reserved</td></tr>
<tr><td>0x18</td><td colspan="3">Reserved</td></tr>
<tr><td>0x1C</td><td colspan="3">Reserved</td></tr>
<tr><td>0x20</td><td colspan="3">Reserved</td></tr>
<tr><td>0x24</td><td colspan="3">Miscellaneous User-Defined Control Register (ECSM_MUDCR)</td></tr>
<tr><td>0x28</td><td colspan="3">Reserved</td></tr>
<tr><td>0x2C – 0x3C</td><td colspan="3">Reserved</td></tr>
</table>

Table 292. ECSM 32-bit memory map (continued)

ECSM Offset	Register			
0x40	Reserved			ECC Configuration Register (ECSM_ECR)
0x44	Reserved			ECC Status Register (ECSM_ESR)
0x48	Reserved		ECC Error Generation Register (ECSM_EEGR)	
0x4C	Reserved			
0x50	Flash ECC Address Register (ECSM_FEAR)			
0x54	Reserved		Flash ECC Master Number Register (ECSM_FEMR)	Flash ECC Attributes (ECSM_FEAT) Register
0x58	Flash ECC Data Register High (ECSM_FEDRH)			
0x5C	Flash ECC Data Register Low (ECSM_FEDRL)			
0x60	RAM ECC Address Register (ECSM_REAR)			
0x64	Reserved	RAM ECC Syndrome Register (ECSM_PRESR)	RAM ECC Master Number Register (ECSM_REMR)	RAM ECC Attributes Register (ECSM_REAT)
0x68	RAM ECC Data Register High (ECSM_REDRH)			
0x6C	RAM ECC Data Register Low (ECSM_REDRL)			
0x70 – 0x7C	Reserved			

17.4 Register descriptions

Attempted accesses to reserved addresses result in an error termination, while attempted writes to read-only registers are ignored and do not terminate with an error.

Note: Unless noted otherwise, ***writes to the programming model must match the size of the register****, e.g., an n-bit register only supports n-bit writes, etc. Attempted writes of a different size than the register width produce an error termination of the bus cycle and no change to the targeted register.*

17.4.1 Miscellaneous Reset Status Register (ECSM_MRSR)

The ECSM_MRSR contains a bit for each of the reset sources to the device. An asserted bit indicates the last type of reset that occurred. Only one bit is set at any time in the ECSM_MRSR, reflecting the cause of the most recent reset as signalled by device reset input signals. The ECSM_MRSR can only be read from the IPS programming model. Any attempted write is ignored.

Figure 290. Miscellaneous Reset Status Register (ECSM_MRSR)

Register address: ECSM Base + 0x000F (0xFFF4_000F)

	0	1	2	3	4	5	6	7
R	POR	OFPLR	0	0	0	0	0	0
W	▨	▨	▨	▨	▨	▨	▨	▨
Reset	1	0	0	0	0	0	0	0

▨ = Unimplemented

Table 293. ECSM_MRSR field description

Name	Description
0 POR	**Power-On Reset** 1 = Last recorded event was caused by a power-on reset (based on a device input signal)
1 OFPLR	**Device Input Reset** 1 = Last recorded event was a reset caused by a device input reset.

17.4.2 Miscellaneous Wakeup Control Register (ECSM_MWCR)

Implementation of low-power sleep modes and exit from these modes via an interrupt require communication between the ECSM, the interrupt controller (INTC) and external logic typically associated with phase-locked loop clock generation circuitry. The Miscellaneous Wakeup Control Register (ECSM_MWCR) provides an 8-bit register controlling entry into these types of low-power modes as well as definition of the interrupt level needed to exit the mode.

The following sequence of operations is generally needed to enable this functionality. Note that the exact details are likely to be system-specific.

1. The processor core loads the appropriate data value into the ECSM_MWCR, setting the ENBWCR bit and the desired interrupt priority level.
2. At the appropriate time, the processor ceases execution. The exact mechanism varies by processor core. In some cases, a processor-is-stopped status is signaled to the ECSM and external logic. This assertion, if properly enabled by ECSM_MWCR[ENBWCR], causes the ECSM output signal “enter_low_power_mode” to be set. This, in turn, causes the selected external, low-power mode, to be entered, and the appropriate clock signals disabled. In most implementations, there are multiple low-power modes, where the exact clocks to be disabled vary across the different modes.
3. After entering the low-power mode, the interrupt controller enables a special combinational logic path which evaluates all unmasked interrupt requests. The device remains in this mode until an event which generates an unmasked interrupt request with a priority level greater than the value programmed in the ECSM_MWCR[PRILVL] occurs.

4. Once the appropriately-high interrupt request level arrives, the interrupt controller signals its presence, and the ECSM responds by asserting an “exit_low_power_mode” signal.
5. The external logic senses the assertion of the “exit” signal, and re-enables the appropriate clock signals.
6. With the processor core clocks enabled, the core handles the pending interrupt request.

Figure 291. Miscellaneous Wakeup Control Register (ECSM_MWCR)

Register address: ECSM Base + 0x13 (0xFFF4_0013)

	0	1	2	3	4	5	6	7
R	ENBWCR	0	0	0	PRILVL[0:3]			
W		░	░	░				
Reset	0	0	0	0	0	0	0	0

░ = Unimplemented

Table 294. ECSM_MWCR field description

Name	Description
0 ENBWCR	**Enable WCR** 0 = MWCR is disabled 1 = MWCR is enabled
4–7 PRILVL[0:3]	**Interrupt Priority Level** The interrupt priority level is a core-specific definition. It specifies the interrupt priority level needed to exit the low-power mode. Specifically, an unmasked interrupt request of a priority level greater than the PRILVL value is required to exit the mode. Certain interrupt controller implementations include logic associated with this priority level that restricts the data value contained in this field to a [0, maximum - 1] range. See the specific interrupt controller module for details.

17.4.3 Miscellaneous User-Defined Control Register (ECSM_MUDCR)

The ECSM_MUDCR is used to specify the number of additional wait states required for the device SRAM. Please see the device datasheet for details on the cut-off frequency for the addition of 1 wait state.

Figure 292. Miscellaneous User-Defined Control Register (ECSM_MUDCR)

Register address: ECSM Base + 0x0024 (0xFFF4_0024)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	SWSC	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	▨		▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented

Table 295. ECSM_MUDCR field description

Name	Description
0	Reserved
1 SWSC	**SRAM Wait State Control** 0 = No additional SRAM wait states 1 = 1 additional SRAM wait state
2–31	Reserved

17.4.4 ECC registers

For platform designs including error-correcting code (ECC) implementations to improve the quality and reliability of memories, there are a number of program-visible registers for the sole purpose of reporting and logging of memory failures. These registers include:

- ECC Configuration Register (ECSM_ECR)
- ECC Status Register (ECSM_ESR)
- ECC Error Generation Register (ECSM_EEGR)
- Flash ECC Address Register (ECSM_FEAR)
- Flash ECC Master Number Register (ECSM_FEMR)
- Flash ECC Attributes Register (ECSM_FEAT)
- Flash ECC Data Register (ECSM_FEDR)
- RAM ECC Address Register (ECSM_REAR)
- RAM ECC Syndrome Register (ECSM_PRESR)
- RAM ECC Master Number Register (ECSM_REMR)
- RAM ECC Attributes Register (ECSM_REAT)
- RAM ECC Data Register (ECSM_REDR)

The details on the ECC registers are provided in the subsequent sections.

The 32-bit ECC organization essentially provides two completely independent error checking mechanisms for the total 64-bit RAM width. The ECC logic provides a 1-of-3 error response vector for each 32 bits of memory: no error, single-bit correctable error, multi-bit non-correctable error. *Table 296* defines the association between the reported ECC result and the RAM bank chip selects.

Table 296. AHB Response and ECC Reporting for Even and Odd ECC

RAM Valid Even	RAM Valid Odd	ECC Even	ECC Odd	Reported ECC	RAM Bus Response Even	RAM Bus Response Odd	AHB HRESP
0	0	x	x	No access, No_error	xxxx	xxxx	okay
1	0	none	x	No_error	data	xxxx	okay
1	0	single	x	Even_single	corrected	xxxx	okay
1	0	multi	x	Even_multi	non-corrected	xxxx	err
0	1	x	none	No_error	xxxx	data	okay
0	1	x	single	Odd_single	xxxx	corrected	okay
0	1	x	multi	Odd_multi	xxxx	non-corrected	err
1	1	none	none	No_error	data	data	okay
1	1	single	none	Even_single	corrected	data	okay
1	1	multi	none	Even_multi	non-corrected	data	err
1	1	none	single	Odd_single	data	corrected	okay
1	1	single	single	Even_single	corrected	corrected	okay
1	1	multi	single	Even_multi	non-corrected	corrected	err
1	1	none	multi	Odd_multi	data	non-corrected	err
1	1	single	multi	Odd_multi	corrected	non-corrected	err
1	1	multi	multi	Even_multi	non-corrected	non-corrected	err

As shown in *Table 296*, accesses of only a single memory bank report the ECC from that bank directly. For accesses involving both banks, the “most severe” ECC response is reported with the even bank taking priority if the responses are equivalent. This approach also provides improved correction capabilities compared to the 64-bit ECC implementation.

ECC Configuration Register (ECSM_ECR)

The ECC Configuration Register is an 8-bit control register for specifying which types of memory errors are reported. In all systems with ECC, the occurrence of a non-correctable error causes the current access to be terminated with an error condition. In many cases, this error termination is reported directly by the initiating bus master. However, there are certain situations where the occurrence of this type of non-correctable error is **not** reported by the master. Examples include speculative instruction fetches which are discarded due to a change-of-flow operation, and buffered operand writes. The ECC reporting logic in the ECSM provides an optional error interrupt mechanism to signal all non-correctable memory errors. In addition to the interrupt generation, the ECSM captures specific information (memory address, attributes and data, bus master number, etc.) which can be useful for subsequent failure analysis.

Figure 293. ECC Configuration Register (ECSM_ECR)

Register address: ECSM Base + 0x0043 (0xFFF4_0043)

	0	1	2	3	4	5	6	7
R	0	0	ER1BR	EF1BR	0	0	ERNCR	EFNCR
W	▨	▨			▨	▨		
Reset	0	0	0	0	0	0	0	0

▨ = Unimplemented

Table 297. ECSM_ECR field description

Name	Description
2 ER1BR	**Enable RAM 1-bit Reporting** 0 = Reporting of single-bit platform RAM corrections is disabled. 1 = Reporting of single-bit platform RAM corrections is enabled. The occurrence of a single-bit RAM correction generates an ECSM ECC interrupt request as signalled by the assertion of ECSM_ESR[R1BC]. The address, attributes and data are also captured in the ECSM_REAR, ECSM_PRESR, ECSM_REMR, ECSM_REAT and ECSM_REDR registers.
3 EF1BR	**Enable Flash 1-bit Reporting** 0 = Reporting of single-bit platform flash corrections is disabled. 1 = Reporting of single-bit platform flash corrections is enabled. The occurrence of a single-bit flash correction generates an ECSM ECC interrupt request as signalled by the assertion of ECSM_ESR[F1BC]. The address, attributes and data are also captured in the ECSM_FEAR, ECSM_FEMR, ECSM_FEAT and ECSM_FEDR registers.
6 ERNCR	**Enable RAM Non-Correctable Reporting** 0 = Reporting of non-correctable platform RAM errors is disabled. 1 = Reporting of non-correctable platform RAM errors is enabled. The occurrence of a non-correctable multi-bit RAM error generates an ECSM ECC interrupt request as signalled by the assertion of ECSM_ESR[RNCE]. The faulting address, attributes and data are also captured in the ECSM_REAR, ECSM_PRESR, ECSM_REMR, ECSM_REAT and ECSM_REDR registers.
7 EFNCR	**Enable Flash Non-Correctable Reporting** 0 = Reporting of non-correctable platform flash errors is disabled. 1 = Reporting of non-correctable platform flash errors is enabled. The occurrence of a non-correctable multi-bit flash error generates an ECSM ECC interrupt request as signalled by the assertion of ECSM_ESR[FNCE]. The faulting address, attributes and data are also captured in the ECSM_FEAR, ECSM_FEMR, ECSM_FEAT and ECSM_FEDR registers.

ECC Status Register (ECSM_ESR)

The ECC Status Register is an 8-bit control register for signaling which types of properly-enabled ECC events have been detected. The ECSM_ESR signals the last, properly-enabled memory event to be detected. An ECC interrupt request is asserted if any flag bit is asserted and its corresponding enable bit is asserted.

ECC interrupt generation is separated into single-bit error detection/correction, uncorrectable error detection and the combination of the two as defined by the following boolean equations:

```
ECSM_ECC1BIT_IRQ
= ECSM_ECR[ER1BR] & ECSM_ESR[R1BC] // platform ram, 1-bit
correction
| ECSM_ECR[EF1BR] & ECSM_ESR[F1BC] // platform flash, 1-bit
correction

ECSM_ECCRNCR_IRQ
= ECSM_ECR[ERNCR] & ECSM_ESR[RNCE] // platform ram, noncorrectable
error

ECSM_ECCFNCR_IRQ
= ECSM_ECR[EFNCR] & ECSM_ESR[FNCE] // platform flash,
noncorrectable error

ECSM_ECC2BIT_IRQ
= ECSM_ECCRNCR_IRQ // platform ram, noncorrectable error
| ECSM_ECCFNCR_IRQ // platform flash, noncorrectable error

ECSM_ECC_IRQ
= ECSM_ECC1BIT_IRQ // 1-bit correction
| ECSM_ECC2BIT_IRQ // noncorrectable error
```

where the combination of a properly-enabled category in the ECSM_ECR and the detection of the corresponding condition in the ECSM_ESR produces the interrupt request.

The ECSM allows a maximum of one bit of the ECSM_ESR to be asserted at any given time. This preserves the association between the ECSM_ESR and the corresponding address and attribute registers, which are loaded on each occurrence of a properly-enabled ECC event. If there is a pending ECC interrupt and another properly-enabled ECC event occurs, the ECSM hardware automatically handles the ECSM_ESR reporting, clearing the previous data and loading the new state and thus guaranteeing that only a single flag is asserted.

To maintain the coherent software view of the reported event, the following sequence in the ECSM error interrupt service routine is suggested:

1. Read the ECSM_ESR and save it.
2. Read and save all the address and attribute reporting registers.
3. Re-read the ECSM_ESR and verify the current contents matches the original contents. If the two values are different, go back to step 1 and repeat.
4. When the values are identical, write a ‘1’ to the asserted ESR flag to negate the interrupt request.

Figure 294. ECC Status Register (ECSM_ESR)

Register address: ECSM Base + 0x0047 (0xFFF4_0047)

	0	1	2	3	4	5	6	7
R	0	0	R1BC	F1BC	0	0	RNCE	FNCE
W	▨	▨			▨	▨		
Reset	0	0	0	0	0	0	0	0

▨ = Unimplemented

Table 298. ECSM_ESR field description

Name	Description
2 R1BC	**Platform RAM 1-bit Correction** 0 = No reportable single-bit platform RAM correction has been detected. 1 = A reportable single-bit platform RAM correction has been detected. This bit can only be set if ECSM_ECR[ER1BR] is asserted. The occurrence of a properly-enabled single-bit RAM correction generates an ECSM ECC interrupt request. The address, attributes and data are also captured in the ECSM_REAR, ECSM_PRESR, ECSM_REMR, ECSM_REAT and ECSM_REDR registers. To clear this interrupt flag, write a ‘1’ to this bit. Writing a ‘0’ has no effect.
3 F1BC	**Platform Flash 1-bit Correction** 0 = No reportable single-bit platform flash correction has been detected. 1 = A reportable single-bit platform flash correction has been detected. This bit can only be set if ECSM_ECR[EF1BR] is asserted. The occurrence of a properly-enabled single-bit flash correction generates an ECSM ECC interrupt request. The address, attributes and data are also captured in the ECSM_FEAR, ECSM_FEMR, ECSM_FEAT and ECSM_FEDR registers. To clear this interrupt flag, write a ‘1’ to this bit. Writing a ‘0’ has no effect.
6 RNCE	**Platform RAM Non-Correctable Error** 0 = No reportable non-correctable platform RAM error has been detected. 1 = A reportable non-correctable platform RAM error has been detected. The occurrence of a properly-enabled non-correctable RAM error generates an ECSM ECC interrupt request. The faulting address, attributes and data are also captured in the ECSM_REAR, ECSM_PRESR, ECSM_REMR, ECSM_REAT and ECSM_REDR registers. To clear this interrupt flag, write a ‘1’ to this bit. Writing a ‘0’ has no effect.
7 FNCE	**Platform Flash Non-Correctable Error** 0 = No reportable non-correctable platform flash error has been detected. 1 = A reportable non-correctable platform flash error has been detected. The occurrence of a properly-enabled non-correctable flash error generates an ECSM ECC interrupt request. The faulting address, attributes and data are also captured in the ECSM_FEAR, ECSM_FEMR, ECSM_FEAT and ECSM_FEDR registers. To clear this interrupt flag, write a ‘1’ to this bit. Writing a ‘0’ has no effect.

In the event that multiple status flags are signaled simultaneously, the ECSM records the event with the R1BC as highest priority, then F1BC, then RNCE, and finally FNCE.

ECC Error Generation Register (ECSM_EEGR)

The ECC Error Generation Register is a 16-bit control register used to force the generation of single- and double-bit data inversions in the memories with ECC, most notably the RAM. This capability is provided for two purposes:

- It provides a software-controlled mechanism for “injecting” errors into the memories during data writes to verify the integrity of the ECC logic.
- It provides a mechanism to allow testing of the software service routines associated with memory error logging.

It should be noted that while the ECSM_EEGR is associated with the RAM, similar capabilities exist for the flash, that is, the ability to program the non-volatile memory with single- or double-bit errors is supported for the same two reasons previously identified.

For both types of memories (RAM and flash), the intent is to generate errors during data write cycles, such that subsequent reads of the corrupted address locations generate ECC events, either single-bit corrections or double-bit non-correctable errors that are terminated with an error response.

Figure 295. ECC Error Generation Register (ECSM_EEGR)

Register address: ECSM Base + 0x004A (0xFFF4_004A)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FRCAP	0	FRC1BI(1)	FR11BI(1)	0	0	FRCNCI	FR1NCI	0	ERRBIT[6:0]						
W		▨			▨	▨			▨							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented

1. This field is writable only in test mode in cut 1.0 devices.

Table 299. ECSM_EEGR field description

Name	Description
0 FRCAP	**Force RAM Error Injection Access Protection** 0 = All Platform masters are able to generate RAM ECC errors via the ECSM_EEGR. 1 = Only the Platform master with ID=0 (usually the core) can generate RAM ECC errors via the ECSM_EEGR. The assertion of this bit ensures that RAM data inversions can only occur from the master module with the master ID of 0. Since this is usually the core, this protects the RAM from errant or multiple simultaneous attempted data inversions from other master modules and, in the case of a multi-core system, ensures that only one core can issue a RAM data inversion. The reset value of the bit is 0 and as a result, RAM data inversions can be requested from any master module. It is the responsibility of the software to ensure the proper setting of this bit.

Table 299. ECSM_EEGR field description (continued)

Name	Description
2 FRC1BI(1)	**Force RAM Continuous 1-bit Data Inversions** 0 = No RAM continuous 1-bit data inversions are generated. 1 = 1-bit data inversions in the RAM are continuously generated. The assertion of this bit forces the RAM controller to create 1-bit data inversions, as defined by the bit position specified in ERRBIT[6:0], continuously on every write operation. The normal ECC generation takes place in the RAM controller, but then the polarity of the bit position defined by ERRBIT is inverted to introduce a 1-bit ECC event in the RAM. After this bit has been enabled to generate another continuous 1-bit data inversion, it must be cleared before being set again to correctly re-enable the error generation logic.
3 FR11BI(1)	**Force RAM One 1-bit Data Inversion** 0 = No RAM single 1-bit data inversion is generated. 1 = One 1-bit data inversion in the RAM is generated. The assertion of this bit forces the RAM controller to create one 1-bit data inversion, as defined by the bit position specified in ERRBIT[6:0], on the first write operation after this bit is set. The normal ECC generation takes place in the RAM controller, but then the polarity of the bit position defined by ERRBIT is inverted to introduce a 1-bit ECC event in the RAM. After this bit has been enabled to generate a single 1-bit data inversion, it must be cleared before being set again to properly re-enable the error generation logic.
6 FRCNCI	**Force RAM Continuous Non-Correctable Data Inversions** 0 = No RAM continuous 2-bit data inversions are generated. 1 = 2-bit data inversions in the RAM are continuously generated. The assertion of this bit forces the RAM controller to create 2-bit data inversions, as defined by the bit position specified in ERRBIT[6:0] and the overall odd parity bit, continuously on every write operation. After this bit has been enabled to generate another continuous non-correctable data inversion, it must be cleared before being set again to properly re-enable the error generation logic. The normal ECC generation takes place in the RAM controller, but then the polarity of the bit position defined by ERRBIT and the overall odd parity bit are inverted to introduce a 2-bit ECC error in the RAM.

Table 299. ECSM_EEGR field description (continued)

Name	Description
7 FR1NCI	**Force RAM One Non-Correctable Data Inversions** 0 = No RAM single 2-bit data inversions are generated. 1 = One 2-bit data inversion in the RAM is generated. The assertion of this bit forces the RAM controller to create one 2-bit data inversion, as defined by the bit position specified in ERRBIT[6:0] and the overall odd parity bit, on the first write operation after this bit is set. The normal ECC generation takes place in the RAM controller, but then the polarity of the bit position defined by ERRBIT and the overall odd parity bit are inverted to introduce a 2-bit ECC error in the RAM. After this bit has been enabled to generate a single 2-bit error, it must be cleared before being set again to properly re-enable the error generation logic.

Table 299. ECSM_EEGR field description (continued)

Name	Description
ERRBIT [6:0]	The vector defines the bit position which is complemented to create the data inversion on the write operation. For the creation of 2-bit data inversions, the bit specified by this field plus the odd parity bit of the ECC code are inverted. The RAM controller follows a vector bit ordering scheme where LSB=0. Errors in the ECC syndrome bits can be generated by setting this field to a value greater than the RAM width. For example, consider a 64-bit RAM implementation and ECC organized on a 32-bit boundary. The 32-bit ECC approach requires 7 code bits for each 32-bit word. For RAM data width of 64 bits, the actual SRAM is 2 × (32 bits data + 7 bits for ECC) = 78 bits which is organized as two 39-bit memory banks, “even” bank and “odd” bank. The following association between the ERRBIT field and the corrupted memory bit is defined: if ERRBIT = 0, then RAM[0] of the odd bank is inverted if ERRBIT = 1, then RAM[1] of the odd bank is inverted ... if ERRBIT = 31, then RAM[31] of the odd bank is inverted if ERRBIT = 32, then RAM[0] of the even bank is inverted if ERRBIT = 33, then RAM[1] of the even bank is inverted ... if ERRBIT = 63, then RAM[31] of the even bank is inverted if ERRBIT = 64, then ECC Parity[0] of the odd bank is inverted if ERRBIT = 65, then ECC Parity[1] of the odd bank is inverted ... if ERRBIT = 70, then ECC Parity[6] of the odd bank is inverted if ERRBIT = 71, then ECC Parity[0] of the even bank is inverted if ERRBIT = 72, then ECC Parity[1] of the even bank is inverted ... if ERRBIT = 77, then ECC Parity[6] of the even bank is inverted For ERRBIT values between 78 and 95, no bit position is inverted. To accommodate address bus inversions, the ERRBIT values start at 96 as defined: if ERRBIT = 96, then ADDR[0] is inverted if ERRBIT = 97, then ADDR[1] is inverted ... if ERRBIT = 114, then ADDR[18] is inverted if ERRBIT = 115, then ADDR[19] is inverted For ERRBIT values greater than 115, the address bus inversion has no effect as only the lower 20 bits are used by the platform RAM controller.

1. This field is writable only in test mode in cut 1.0 devices.

If an attempt to force a non-correctable inversion (by asserting ECSM_EEGR[FRCNCI] or ECSM_EEGR[FRC1NCI]) and ECSM_EEGR[ERRBIT] equals 64, then no data inversion will be generated.

The only allowable values for the four control bit enables {FR11BI, FRC1BI, FRCNCI, FR1NCI} are {0,0,0,0}, {1,0,0,0}, {0,1,0,0}, {0,0,1,0} and {0,0,0,1}. All other values result in unpredictable operations.

Flash ECC Address Register (ECSM_FEAR)

The ECSM_FEAR is a 32-bit register for capturing the address of the last, properly-enabled ECC event in the platform flash memory. Depending on the state of the ECC Configuration Register, an ECC event in the flash causes the address, attributes and data associated with the access to be loaded into the ECSM_FEAR, ECSM_FEMR, ECSM_FEAT and ECSM_FEDRs, and the appropriate flag (F1BC or FNCE) in the ECC Status Register to be asserted.

This register can only be read from the IPS programming model; any attempted write is ignored.

Figure 296. Flash ECC Address Register (ECSM_FEAR)

Register address: ECSM Base + 0x0050 (0xFFF4_0054)

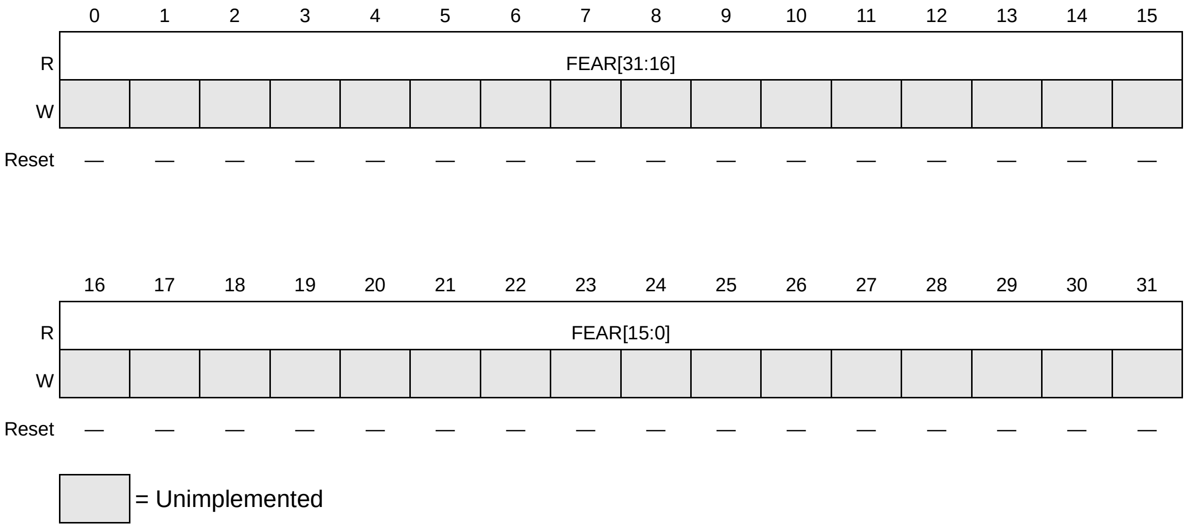


Table 300. ECSM_FEAR field description

Name	Description
0–31 FEAR[31:0]	**Flash ECC Address Register** This 32-bit register contains the faulting access address of the last, properly-enabled flash ECC event.

Flash ECC Master Number Register (ECSM_FEMR)

The ECSM_FEMR is a 4-bit register for capturing the XBAR bus master number of the last, properly-enabled ECC event in the flash memory. Depending on the state of the ECC Configuration Register, an ECC event in the flash causes the address, attributes and data associated with the access to be loaded into the ECSM_FEAR, ECSM_FEMR, ECSM_FEAT and ECSM_FEDR registers, and the appropriate flag (FNCE) in the ECC Status Register to be asserted.

This register can only be read from the IPS programming model; any attempted write is ignored.

Figure 297. Flash ECC Master Number Register (ECSM_FEMR)

Register address: ECSM Base + 0x0056 (0xFFF4_0056)

	0	1	2	3	4	5	6	7
R	0	0	0	0	FEMR[3:0]			
W	▨	▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	—	—	—	—

▨ = Unimplemented

Table 301. ECSM_FEMR field description

Name	Description
4–7 FEMR[3:0]	**Flash ECC Master Number Register** This 4-bit register contains the XBAR bus master number of the faulting access of the last, properly-enabled flash ECC event.

Flash ECC Attributes (ECSM_FEAT) Register

The ECSM_FEAT register is an 8-bit register for capturing the XBAR bus master attributes of the last, properly-enabled ECC event in the flash memory. Depending on the state of the ECC Configuration Register, an ECC event in the flash causes the address, attributes and data associated with the access to be loaded into the ECSM_FEAR, ECSM_FEMR, ECSM_FEAT and ECSM_FEDR registers, and the appropriate flag (FNCE) in the ECC Status Register to be asserted.

This register can only be read from the IPS programming model; any attempted write is ignored.

Figure 298. Flash ECC Attributes (ECSM_FEAT) Register

Register address: ECSM Base + 0x0057 (0xFFF4_0057)

	0	1	2	3	4	5	6	7
R	WRITE	SIZE[2:0]			PROT0	PROT1	PROT2	PROT3
W	▨	▨	▨	▨	▨	▨	▨	▨
Reset	—	—	—	—	—	—	—	—

▨ = Unimplemented

Table 302. ECSM_FEAT field description

Name	Description	
0 WRITE	**AMBA-AHB HWRITE** 0 = AMBA-AHB read access 1 = AMBA-AHB write access	
1–3 SIZE[2:0]	**AMBA-AHB HSIZE** 0b000 = 8-bit AMBA-AHB access 0b001 = 16-bit AMBA-AHB access 0b010 = 32-bit AMBA-AHB access 0b011 = Reserved 0b1xx = Reserved	
4–7 PROTn	**AMBA-AHB HPROT**	
	PROT3: Cacheable	0 = Non-cacheable,1 = Cacheable
	PROT2: Bufferable	0 = Non-bufferable,1 = Bufferable
	PROT1: Mode	0 = User mode, 1 = Supervisor mode
	PROT0: Type	0 = I-Fetch, 1 = Data

Flash ECC Data Register (ECSM_FEDRH, ECSM_FEDRL)

The ECSM_FEDR is a 64-bit register for capturing the data associated with the last, properly-enabled ECC event in the flash memory. Depending on the state of the ECC Configuration Register, an ECC event in the flash causes the address, attributes and data associated with the access to be loaded into the ECSM_FEAR, ECSM_FEMR, ECSM_FEAT, and ECSM_FEDR registers, and the appropriate flag (FNCE) in the ECC Status Register to be asserted.

The data captured on a multi-bit non-correctable ECC error is undefined.

This register can only be read from the IPS programming model; any attempted write is ignored.

Figure 299. Flash ECC Data Register (ECSM_FEDRH, ECSM_FEDRL)

Register address: ECSM Base + 0x58, +0x5C ((0xFFF4_0058, 0xFFF4_005C)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FEDH[31:16]															
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	FEDH[15:0]															
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
R	FEDL[31:16]															
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
R	FEDL[15:0]															
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

▨ = Unimplemented

Table 303. ECSM_FEDR field description

Name	Description
0–63 FEDH[31:0] FEDL[31:0]	**Flash ECC Data Register** This 64-bit register contains the data associated with the faulting access of the last, properly-enabled flash ECC event. The register contains the data value taken directly from the data bus.

RAM ECC Address Register (ECSM_REAR)

The ECSM_REAR is a 32-bit register for capturing the address of the last, properly-enabled ECC event in the RAM memory. Depending on the state of the ECC Configuration Register, an ECC event in the RAM causes the address, attributes and data associated with the access to be loaded into the ECSM_REAR, ECSM_PRESR, ECSM_REMR, ECSM_REAT and ECSM_REDR registers, and the appropriate flag (RNCE) in the ECC Status Register to be asserted.

This register can only be read from the IPS programming model; any attempted write is ignored.

Figure 300. RAM ECC Address Register (ECSM_REAR)

Register address: ECSM Base + 0x0060 (0xFFF4_0060)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	REAR[31:16]															
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	REAR[15:0]															
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

▨ = Unimplemented

Table 304. ECSM_REAR field description

Name	Description
0–31 REAR[31:0]	**RAM ECC Address Register** This 32-bit register contains the faulting access address of the last, properly-enabled RAM ECC event.

RAM ECC Syndrome Register (ECSM_PRESR)

The ECSM_PRESR is an 8-bit register for capturing the error syndrome of the last, properly-enabled ECC event in the RAM memory. Depending on the state of the ECC Configuration Register, an ECC event in the RAM causes the address, attributes and data associated with the access to be loaded into the ECSM_REAR, ECSM_PRESR, ECSM_REMR, ECSM_REAT and ECSM_REDR registers, and the appropriate flag (RNCE) in the ECC Status Register to be asserted.

The ECSM_PRESR can only be read from the IPS programming model; any attempted write is ignored.

Figure 301. RAM ECC Syndrome Register (ECSM_PRESR)

Register address: ECSM Base + 0x0065 (0xFFF4_0065)

	0	1	2	3	4	5	6	7
R	PRESR[7:0]							
W	▨	▨	▨	▨	▨	▨	▨	▨
Reset	—	—	—	—	—	—	—	—

▨ = Unimplemented

Table 305. ECSM_PRESR field description

Name	Description
0–7 PRESR[7:0]	**RAM ECC Syndrome Register** This 8-bit syndrome field includes optimized syndrome encoding for the entire 39-bit (32-bit data + 7-bit ECC) code word of each bank for single-bit errors. Syndrome values for non-correctable errors are not defined. For correctable single-bit errors, the mapping shown in *Table 306* associates the 8 bits of the syndrome with the data bit in error.

Note: Table 306 associates the 8 bits of the ECC syndrome with the exact data bit in error for single-bit correctable code words. This table follows the bit vectoring notation where the LSB = 0. The syndrome value of 0x00 implies no error condition but this value is not readable when the ECSM_PRESR is read for the no error case.

Table 306. RAM syndrome mapping for single-bit correctable errors

PRESR[7:0]	Data bit in error
0x01	ECC ODD[0]
0x02	ECC ODD[1]
0x04	ECC ODD[2]
0x07	DATA ODD BANK[31]
0x08	ECC ODD[3]
0x10	ECC ODD[4]
0x20	ECC ODD[5]
0x40	ECC ODD[6]
0x43	DATA ODD BANK[0]
0x45	DATA ODD BANK[1]
0x46	DATA ODD BANK[2]
0x49	DATA ODD BANK[3]
0x4A	DATA ODD BANK[4]
0x4C	DATA ODD BANK[5]
0x4F	DATA ODD BANK[21]
0x51	DATA ODD BANK[6]
0x52	DATA ODD BANK[7]
0x54	DATA ODD BANK[8]
0x57	DATA ODD BANK[22]
0x58	DATA ODD BANK[9]
0x5B	DATA ODD BANK[23]
0x5D	DATA ODD BANK[24]
0x5E	DATA ODD BANK[25]

Table 306. RAM syndrome mapping for single-bit correctable errors (continued)

PRESR[7:0]	Data bit in error
0x61	DATA ODD BANK[10]
0x62	DATA ODD BANK[11]
0x64	DATA ODD BANK[12]
0x67	DATA ODD BANK[26]
0x68	DATA ODD BANK[13]
0x6B	DATA ODD BANK[27]
0x6D	DATA ODD BANK[28]
0x6E	DATA ODD BANK[29]
0x70	DATA ODD BANK[14]
0x73	DATA ODD BANK[15]
0x75	DATA ODD BANK[16]
0x76	DATA ODD BANK[17]
0x79	DATA ODD BANK[18]
0x7A	DATA ODD BANK[19]
0x7C	DATA ODD BANK[20]
0x7F	DATA ODD BANK[30]
0x81	ECC EVEN[0]
0x82	ECC EVEN[1]
0x84	ECC EVEN[2]
0x87	DATA EVEN BANK[31]
0x88	ECC EVEN[3]
0x90	ECC EVEN[4]
0xA0	ECC EVEN[5]
0xC0	ECC EVEN[6]
0xC3	DATA EVEN BANK[0]
0xC5	DATA EVEN BANK[1]
0xC6	DATA EVEN BANK[2]
0xC9	DATA EVEN BANK[3]
0xCA	DATA EVEN BANK[4]
0xCC	DATA EVEN BANK[5]
0xCF	DATA EVEN BANK[21]
0xD1	DATA EVEN BANK[6]
0xD2	DATA EVEN BANK[7]
0xD4	DATA EVEN BANK[8]
0xD7	DATA EVEN BANK[22]

Table 306. RAM syndrome mapping for single-bit correctable errors (continued)

PRESR[7:0]	Data bit in error
0xD8	DATA EVEN BANK[9]
0xDB	DATA EVEN BANK[23]
0xDD	DATA EVEN BANK[24]
0xDE	DATA EVEN BANK[25]
0xE1	DATA EVEN BANK[10]
0xE2	DATA EVEN BANK[11]
0xE4	DATA EVEN BANK[12]
0xE7	DATA EVEN BANK[26]
0xE8	DATA EVEN BANK[13]
0xEB	DATA EVEN BANK[27]
0xED	DATA EVEN BANK[28]
0xEE	DATA EVEN BANK[29]
0xF0	DATA EVEN BANK[14]
0xF3	DATA EVEN BANK[15]
0xF5	DATA EVEN BANK[16]
0xF6	DATA EVEN BANK[17]
0xF9	DATA EVEN BANK[18]
0xFA	DATA EVEN BANK[19]
0xFC	DATA EVEN BANK[20]
0xFF	DATA EVEN BANK[30]

RAM ECC Master Number Register (ECSM_REMR)

The ECSM_REMR is a 4-bit register for capturing the XBAR bus master number of the last, properly-enabled ECC event in the RAM memory. Depending on the state of the ECC Configuration Register, an ECC event in the RAM causes the address, attributes and data associated with the access to be loaded into the ECSM_REAR, ECSM_PRESR, ECSM_REMR, ECSM_REAT and ECSM_REDR registers, and the appropriate flag (RNCE) in the ECC Status Register to be asserted.

This register can only be read from the IPS programming model; any attempted write is ignored.

Figure 302. RAM ECC Master Number Register (ECSM_REMR)

Register address: ECSM Base + 0x0066 (0xFFF4_0066)

	0	1	2	3	4	5	6	7
R	0	0	0	0	REMR[3:0]			
W	▨	▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	—	—	—	—

▨ = Unimplemented

Table 307. ECSM_REMR field description

Name	Description
4–7 REMR[3:0]	**RAM ECC Master Number Register** This 4-bit register contains the XBAR bus master number of the faulting access of the last, correctly-enabled RAM ECC event.

RAM ECC Attributes Register (ECSM_REAT)

The ECSM_REAT register is an 8-bit register for capturing the XBAR bus master attributes of the last, properly-enabled ECC event in the RAM memory. Depending on the state of the ECC Configuration Register, an ECC event in the RAM causes the address, attributes and data associated with the access to be loaded into the ECSM_REAR, ECSM_PRESR, ECSM_REMR, ECSM_REAT and ECSM_REDR registers, and the appropriate flag (RNCE) in the ECC Status Register to be asserted. The ECSM_REAT register is read-only.

Figure 303. RAM ECC Attributes (ECSM_REAT) Register

Register address: ECSM Base + 0x0067 (0xFFF4_0067)

	0	1	2	3	4	5	6	7
R	WRITE	SIZE[2:0]			PROT0	PROT1	PROT2	PROT3
W	▨	▨	▨	▨	▨	▨	▨	▨
Reset	—	—	—	—	—	—	—	—

▨ = Unimplemented

Table 308. ECSM_REAT field description

Name	Description
0 WRITE	**AMBA-AHB HWRITE** 0 = AMBA-AHB read access 1 = AMBA-AHB write access
1–3 SIZE[2:0]	**AMBA-AHB HSIZE** 0b000 = 8-bit AMBA-AHB access 0b001 = 16-bit AMBA-AHB access 0b010 = 32-bit AMBA-AHB access 0b011 = 64-bit AMBA-AHB access 0b1xx = Reserved
4–7 PROTn	**AMBA-AHB HPROT** PROT3: Cacheable 0 = Non-cacheable, 1 = Cacheable PROT2: Bufferable 0 = Non-bufferable, 1 = Bufferable PROT1: Mode 0 = User mode, 1 = Supervisor mode PROT0: Type 0 = I-Fetch, 1 = Data

RAM ECC Data Register (ECSM_REDRH, ECSM_REDRL)

The ECSM_REDR is a 64-bit register for capturing the data associated with the last, properly-enabled ECC event in the RAM memory. Depending on the state of the ECC Configuration Register, an ECC event in the RAM causes the address, attributes and data associated with the access to be loaded into the ECSM_REAR, ECSM_PRESR, ECSM_REMR, ECSM_REAT and ECSM_REDR registers, and the appropriate flag (RNCE) in the ECC Status Register to be asserted.

The data captured on a multi-bit non-correctable ECC error is undefined.

Since the RAM controller calculates ECC on a 32-bit boundary, only the 32-bit piece of data containing the error is recorded in the lower 32-bit word. The upper 32 bits will read back all zeroes as defined.

This register can only be read from the IPS programming model; any attempted write is ignored.

Figure 304. RAM ECC Data Register (ECSM_REDR)

Register address: ECSM Base + 0x68, +0x6C (0xFFF4_0068, (0xFFF4_006C)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	REDH[31:16]															
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	REDH[15:0]															
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
R	REDL[31:16]															
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
R	REDL[15:0]															
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

▨ = Unimplemented

Table 309. ECSM_REDR field description

Name	Description
0–63 REDH[31:0] REDL[31:0]	**RAM ECC Data Register** This 64-bit register contains the data associated with the faulting access of the last, properly-enabled RAM ECC event. The register contains the data value taken directly from the data bus.

18 System Timer Module (STM)

18.1 Information Specific to This Device

This section presents device-specific parameterization and customization information not specifically referenced in the remainder of this chapter.

18.1.1 Device-Specific Features

- One 32-bit up counter with 8-bit prescaler
- Four 32-bit compare channels
- Independent interrupt source for each channel
- Counter can be stopped in debug mode

18.2 Introduction

18.2.1 Overview

The System Timer Module (STM) is a 32-bit timer designed to support commonly required system and application software timing functions. The STM includes a 32-bit up counter and four 32-bit compare channels with a separate interrupt source for each channel. The counter is driven by the system clock divided by an 8-bit prescale value (1 to 256).

18.2.2 Modes of operation

The STM supports two device modes of operation: normal and debug. When the STM is enabled in normal mode, its counter runs continuously. In debug mode, operation of the counter is controlled by the FRZ bit in the STM_CR register. If the FRZ bit is set, the counter is stopped in debug mode, otherwise it continues to run.

18.3 External signal description

The STM does not have any external interface signals.

18.4 Memory map and register definition

The STM programming model has fourteen 32-bit registers. The STM registers can only be accessed using 32-bit (word) accesses. Attempted references using a different size or to a reserved address generates a bus error termination.

18.4.1 Memory map

The STM memory map is shown in *Table 310*.

Table 310. STM memory map

Address offset	Register description	Size (bits)	Access	Location
0x0000	STM Control Register(STM_CR)	32	R/W	*on page 18-527*
0x0004	STM Count Register(STM_CNT)	32	R/W	*on page 18-527*
0x0008	Reserved	—	—	—
0x000C	Reserved	—	—	—
0x0010	STM Channel 0 Control Register(STM_CCR0)	32	R/W	*on page 18-528*
0x0014	STM Channel 0 Interrupt Register(STM_CIR0)	32	R/W	*on page 18-528*
0x0018	STM Channel 0 Compare Register(STM_CMP0)	32	R/W	*on page 18-529*
0x001C	Reserved	—	—	—
0x0020	STM Channel 1 Control Register(STM_CCR1)	32	R/W	*on page 18-528*
0x0024	STM Channel 1 Interrupt Register(STM_CIR1)	32	R/W	*on page 18-528*
0x0028	STM Channel 1 Compare Register(STM_CMP1)	32	R/W	*on page 18-529*
0x002C	Reserved	—	—	—
0x0030	STM Channel 2 Control Register(STM_CCR2)	32	R/W	*on page 18-528*
0x0034	STM Channel 2 Interrupt Register(STM_CIR2)	32	R/W	*on page 18-528*
0x0038	STM Channel 2 Compare Register(STM_CMP2)	32	R/W	*on page 18-529*
0x003C	Reserved	—	—	—
0x0040	STM Channel 3 Control Register(STM_CCR3)	32	R/W	*on page 18-528*
0x0044	STM Channel 3 Interrupt Register(STM_CIR3)	32	R/W	*on page 18-528*
0x0048	STM Channel 3 Compare Register(STM_CMP3)	32	R/W	*on page 18-529*
0x004C – 0x3FFF	Reserved	—	—	—

18.4.2 Register descriptions

The following sections detail the individual registers within the STM programming model.

STM Control Register (STM_CR)

The STM Control Register (STM_CR) includes the prescale value, freeze control and timer enable bits.

Figure 305. STM Control Register (STM_CR)

Offset 0x000 Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CPS								0	0	0	0	0	0	FRZ	TEN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 311. STM_CR field description

Field	Description
CPS	Counter Prescaler Selects the clock divide value for the prescaler (1 – 256) 0x00 = Divide system clock by 1 0x01 = Divide system clock by 2 ... 0xFF = Divide system clock by 256
FRZ	Freeze Allows the timer counter to be stopped when the device enters debug mode 0 = STM counter continues to run in debug mode. 1 = STM counter is stopped in debug mode.
TEN	Timer Counter Enabled 0 = Counter is disabled 1 = Counter is enabled

STM Count Register (STM_CNT)

The STM Count Register (STM_CNT) holds the timer count value.

Figure 306. STM Count Register (STM_CNT)

Offset 0x004 Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CNT																															
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 312. STM_CNT field description

Field	Description
CNT	Timer count value used as the time base for all channels When enabled, the counter increments at the rate of the system clock divided by the prescale value.

STM Channel n Control Register (STM_CCRn)

The STM Channel n Control Register (STM_CCRn) has the enable bit for channel n of the timer.

Figure 307. STM Channel Control Register (STM_CCRn)

Offset 0x10+0x10*n Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	CEN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 313. STM_CCRn field description

Field	Description
CEN	Channel Enable 0 = The channel is disabled. 1 = The channel is enabled.

STM Channel n Interrupt Register (STM_CIRn)

The STM Channel n Interrupt Register (STM_CIRn) has the interrupt flag for channel n of the timer.

Figure 308. STM Channel Interrupt Register (STM_CIRn)

Offset 0x14+0x10*n Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	CIF
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 314. STM_CIRn field description

Field	Description
CIF	Channel Interrupt Flag The flag and interrupt are cleared by writing a ‘1’ to this bit. Writing a ‘0’ has no effect. 0 = No interrupt request 1 = Interrupt request due to a match on the channel

STM Channel Compare Register (STM_CMPn)

The STM channel compare register (STM_CMPn) holds the compare value for channel n.

Figure 309. STM Channel Compare Register (STM_CMPn)

Offset 0x18+0x10*n Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CMP																															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 315. STM_CMPn Register field description

Field	Description
CMP	Compare value for channel n If the STM_CCRn[CEN] bit is set and the STM_CMPn register matches the STM_CNT register, a channel interrupt request is generated and the STM_CIRn[CIF] bit is set.

18.5 Functional Description

The System Timer Module (STM) is a 32-bit timer designed to support commonly required system and application software timing functions. The STM includes a 32-bit up counter and four 32-bit compare channels with a separate interrupt source for each channel.

The STM has one 32-bit up counter (STM_CNT) that is used as the time base for all channels. When enabled, the counter increments at the system clock frequency divided by a prescale value. The STM_CR[CPS] field sets the divider to any value in the range from 1 to 256. The counter is enabled with the STM_CR[TEN] bit. When enabled in normal mode the counter continuously increments. When enabled in debug mode the counter operation is controlled by the STM_CR[FRZ] bit. When the STM_CR[FRZ] bit is set, the counter is stopped in debug mode, otherwise it continues to run in debug mode. The counter rolls over at 0xFFFF_FFFF to 0x0000_0000 with no restrictions at this boundary.

The STM has four identical compare channels. Each channel includes a channel control register (STM_CCRn), a channel interrupt register (STM_CIRn) and a channel compare register (STM_CMPn). The channel is enabled by setting the STM_CCRn[CEN] bit. When enabled, the channel will set the STM_CIR[CIF] bit and generate an interrupt request when the channel compare register matches the timer counter. The interrupt request is cleared by writing a ‘1’ to the STM_CIRn[CIF] bit. A write of ‘0’ to the STM_CIRn[CIF] bit has no effect.

19 Software Watchdog Timer (SWT)

19.1 Introduction

19.1.1 Overview

The Software Watchdog Timer (SWT) is a peripheral module that can prevent system lockup in situations such as software getting trapped in a loop or if a bus transaction fails to terminate. When enabled, the SWT requires periodic execution of a watchdog servicing operation. The servicing operation resets the timer to a specified time-out period. If this servicing action does not occur before the timer expires the SWT generates an interrupt or hardware reset. The SWT can be configured to generate a reset or interrupt on an initial time-out, a reset is always generated on a second consecutive time-out.

19.1.2 Features

The SWT has the following features:

- 32-bit time-out register to set the time-out period
- Programmable selection of system or oscillator clock for timer operation
- Programmable selection of window mode or regular servicing
- Programmable selection of reset or interrupt on an initial time-out
- Programmable selection of fixed or keyed servicing
- Master access protection
- Hard and soft configuration lock bits

19.1.3 Modes of operation

The SWT supports three device modes of operation: normal, debug and stop. When the SWT is enabled in normal mode, its counter runs continuously. In debug mode, operation of the counter is controlled by the FRZ bit in the SWT_MCR. If the FRZ bit is set, the counter is stopped in debug mode, otherwise it continues to run. In stop mode, operation of the counter is controlled by the STP bit in the SWT_MCR. If the STP bit is set, the counter is stopped in stop mode; otherwise, it continues to run.

19.2 External signal description

The SWT module does not have any external interface signals.

19.3 Memory map and register definition

The SWT programming model has seven 32-bit registers. The programming model can only be accessed using 32-bit (word) accesses. References using a different size are invalid. Other types of invalid accesses include: writes to read-only registers, incorrect values written to the service register when enabled, accesses to reserved addresses and accesses by masters without permission. If the RIA bit in the SWT_MCR is set then the SWT generates a system reset on an invalid access otherwise a bus error is generated. If either the HLK or SLK bits in the SWT_MCR are set then the SWT_MCR, SWT_TO, SWT_WN, SWT_SK registers are read-only.

19.3.1 Memory map

The SWT memory map is shown in *Table 316*. The reset values of SWT_MCR, SWT_TO and SWT_WN are device specific. These values are determined by SWT inputs.

Table 316. SWT memory map

Offset from SWT Base address (0xFFF3_8000)	Register name	Register description	Size (bits)	Access	Location
0x0000	SWT_MCR	SWT Module Control Register	32	R/W	*on page 19-532*
0x0004	SWT_IR	SWT Interrupt Register	32	R/W	*on page 19-534*
0x0008	SWT_TO	SWT Time-out Register	32	R/W	*on page 19-535*
0x000C	SWT_WN	SWT Window Register	32	R/W	*on page 19-535*
0x0010	SWT_SR	SWT Service Register	32	R/W	*on page 19-535*
0x0014	SWT_CO	SWT Counter Output Register	32	R	*on page 19-536*
0x0018	SWT_SK	SWT Service Key Register	32	R/W	*on page 19-536*
0x001C – 0x3FFF	Reserved		—	—	—

19.3.2 Register descriptions

The following sections detail the individual registers within the SWT programming model.

SWT Module Control Register (SWT_MCR)

The SWT_MCR contains fields for configuring and controlling the SWT. The reset value of this register is device specific. Some devices can be configured to automatically clear the SWT_MCR[WEN] bit during the boot process. This register is read-only if either the SWT_MCR[HLK] or SWT_MCR[SLK] bits are set.

Figure 310. SWT Module Control Register (SWT_MCR)

Offset 0x0000 Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MAP 0	MAP 1	MAP 2	MAP 3	MAP 4	MAP 5	MAP 6	MAP 7	0	0	0	0	0	0	0	0
W																
Reset[1]	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	KEY	RIA	WND	ITR	HLK	SLK	CSL	STP	FRZ	WEN
W																
Reset	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1	0

Table 317. SWT_MCR field description

Field	Description
MAPn	Master Access Protection for Master n The platform bus master assignments are device specific. 0 = Access for the master is not enabled 1 = Access for the master is enabled
KEY	Keyed Service Mode 0 = Fixed Service Sequence, the fixed sequence 0xA602, 0xB480 is used to service the watchdog 1 = Keyed Service Mode, two pseudorandom key values are used to service the watchdog
RIA	Reset on Invalid Access 0 = Invalid access to the SWT generates a bus error 1 = Invalid access to the SWT causes a system reset if WEN = 1
WND	Window Mode 0 = Regular mode, service sequence can be done at any time 1 = Windowed mode, the service sequence is only valid when the down counter is less than the value in the SWT_WN register.
ITR	Interrupt Then Reset 0 = Generate a reset on a time-out 1 = Generate an interrupt on an initial time-out, reset on a second consecutive time-out
HLK	Hard Lock This bit is only cleared at reset. 0 = SWT_MCR, SWT_TO, SWT_WN and SWT_SK are read/write registers if SLK = 0 1 = SWT_MCR, SWT_TO, SWT_WN and SWT_SK are read-only registers
SLK	Soft Lock This bit is cleared by writing the unlock sequence to the service register. 0 = SWT_MCR, SWT_TO SWT_WN and SWT_SK are read/write registers if HLK = 0 1 = SWT_MCR, SWT_TO, SWT_WN and SWT_SK are read-only registers

Table 317. SWT_MCR field description (continued)

Field	Description
CSL	Clock Selection Selects the clock that drives the internal timer 0 = System clock 1 = Oscillator clock
STP	Stop Mode Control Allows the watchdog timer to be stopped when the device enters stop mode 0 = SWT counter continues to run in stop mode 1 = SWT counter is stopped in stop mode
FRZ	Debug Mode Control Allows the watchdog timer to be stopped when the device enters debug mode 0 = SWT counter continues to run in debug mode 1 = SWT counter is stopped in debug mode
WEN	Watchdog Enabled 0 = SWT is disabled 1 = SWT is enabled

SWT Interrupt Register (SWT_IR)

The SWT_IR contains the time-out interrupt flag.

Figure 311. SWT Interrupt Register (SWT_IR)

Offset 0x0004 Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	TIF
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 318. SWT_IR field description

Field	Description
TIF	Time-out Interrupt Flag The flag and interrupt are cleared by writing a ‘1’ to this bit. Writing a ‘0’ has no effect. 0 = No interrupt request 1 = Interrupt request due to an initial time-out

SWT Time-Out Register (SWT_TO)

The SWT Time-Out (SWT_TO) register contains the 32-bit time-out period. The reset value for this register is device specific. This register is read-only if either the SWT_MCR[HLK] or SWT_MCR[SLK] bits are set.

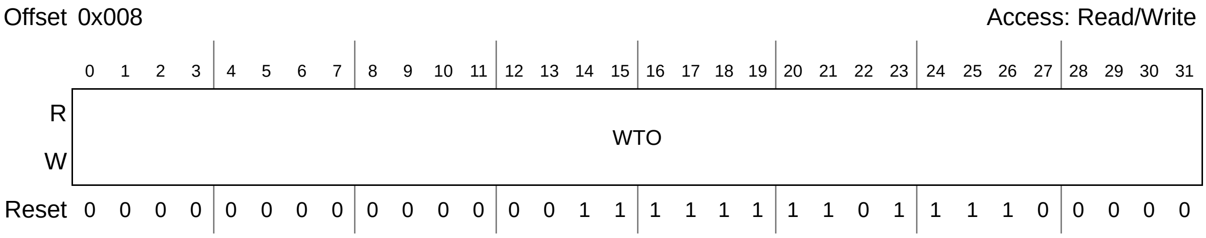


Figure 312. SWT Time-Out Register (SWT_TO)

Table 319. SWT_TO Register field description

Field	Description
WTO	Watchdog time-out period in clock cycles An internal 32-bit down counter is loaded with this value or 0x100 which ever is greater when the service sequence is written or when the SWT is enabled.

SWT Window Register (SWT_WN)

The SWT Window (SWT_WN) register contains the 32-bit window start value. This register is cleared on reset. This register is read-only if either the SWT_MCR[HLK] or SWT_MCR[SLK] bits are set.

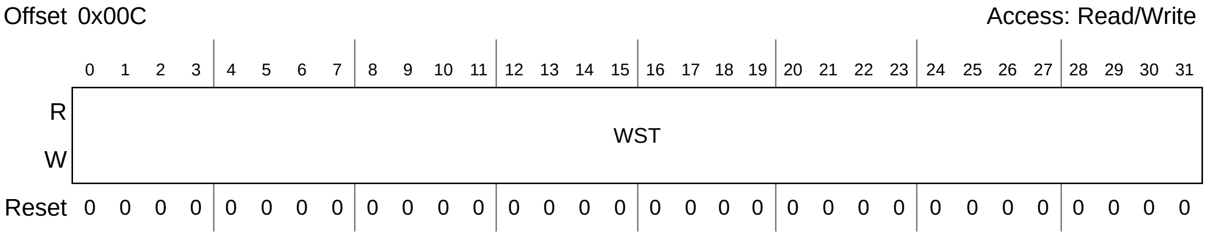


Figure 313. SWT Window Register (SWT_WN)

Table 320. SWT_WN Register field description

Field	Description
WST	Window start value When window mode is enabled, the service sequence can only be written when the internal down counter is less than this value.

SWT Service Register (SWT_SR)

The SWT Time-Out (SWT_SR) service register is the target for service operation writes used to reset the watchdog timer.

Offset 0x010 Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																	WSC															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 314. SWT Service Register (SWT_SR)

Table 321. SWT_SR field description

Field	Description
WSC	Watchdog Service Code This field is used to service the watchdog and to clear the soft lock bit (SWT_MCR[SLK]). If the SWT_MCR[KEY] bit is set, two pseudorandom key values are written to service the watchdog, see *Section 19.4, Functional description*, for details. Otherwise, the sequence 0xA602 followed by 0xB480 is written to the WSC field. To clear the soft lock bit (SWT_MCR[SLK]), the value 0xC520 followed by 0xD928 is written to the WSC field.

SWT Counter Output Register (SWT_CO)

The SWT Counter Output (SWT_CO) register is a read-only register that shows the value of the internal down counter when the SWT is disabled.

Offset 0x014 Access: Read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CNT																															
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	1	1	1	0	1	0	0	0	1

Figure 315. SWT Counter Output Register (SWT_CO)

Table 322. SWT_CO Register field description

Field	Description
CNT	Watchdog Count When the watchdog is disabled (SWT_MCR[WEN] = 0) this field shows the value of the internal down counter. When the watchdog is enabled the value of this field is 0x0000_0000. Values in this field can lag behind the internal counter value for up to six system plus eight counter clock cycles. Therefore, the value read from this field immediately after disabling the watchdog may be higher than the actual value of the internal counter.

SWT Service Key Register (SWT_SK)

The SWT Service Key (SWT_SK) register holds the previous (or initial) service key value. This register is read-only if either the SWT_MCR[HLK] or SWT_MCR[SLK] bits are set.

Offset 0x018 Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																	SK															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 316. SWT Service Register (SWT_SK)

Table 323. SWT_SK field description

Field	Description
SK	Service Key This field is the previous (or initial) service key value used in keyed service mode. If SWT_MCR[KEY] is set, the next key value to be written to the SWT_SR is (17*SK+3) mod 2^{16}.

19.4 Functional description

The SWT is a 32-bit timer designed to enable the system to recover in situations such as software getting trapped in a loop or if a bus transaction fails to terminate. It includes a control register (SWT_MCR), an interrupt register (SWT_IR), a time-out register (SWT_TO), a window register (SWT_WN), a service register (SWT_SR), a counter output register (SWT_CO) and a service key register (SWT_SK).

The SWT_MCR includes bits to enable the timer, set configuration options and lock configuration of the module. The watchdog is enabled by setting the SWT_MCR[WEN] bit. The reset value of the SWT_MCR[WEN] bit is dependent upon the SWT field in the Reset Configuration Half Word (see *Section 20.5.3, Reset configuration half word (RCHW)*). If the reset value of this bit is 1, the watchdog starts operation automatically after reset is released.

The SWT_TO register holds the watchdog time-out period in clock cycles unless the value is less than 0x100 in which case the time-out period is set to 0x100. This time-out period is loaded into an internal 32-bit down counter when the SWT is enabled and each time a valid service operation is performed. The SWT_MCR[CSL] bit selects which clock (system or oscillator) is used to drive the down counter.

The configuration of the SWT can be locked through use of either a soft lock or a hard lock. In either case, when locked the SWT_MCR, SWT_TO, SWT_WN and SWT_SK registers are read-only. The hard lock is enabled by setting the SWT_MCR[HLK] bit which can only be cleared by a reset. The soft lock is enabled by setting the SWT_MCR[SLK] bit and is cleared by writing the unlock sequence to the service register. The unlock sequence is a write of 0xC520 followed by a write of 0xD928 to the SWT_SR[WSC] field. There is no timing requirement between the two writes. The unlock sequence logic ignores service sequence writes and recognizes the 0xC520, 0xD928 sequence regardless of previous writes. The unlock sequence can be written at any time and does not require the SWT_MCR[WEN] bit to be set.

When enabled, the SWT requires periodic execution of a servicing operation which consists of writing two values to the SWT_SR. Writing the proper sequence of values loads the internal down counter with the time-out period. There is no timing requirement between the two writes and the service sequence logic ignores unlock sequence writes. If the SWT_MCR[KEY] bit is zero, the fixed sequence 0xA602, 0xB480 is written to the SWT_SR[WSC] field to service the watchdog. If the SWT_MCR[KEY] bit is set, then two pseudorandom keys are written to the SWT_SR[WSC] field to service the watchdog. The key values are determined by the pseudorandom key generator defined in *Equation 11*. This algorithm will generate a sequence of 2^{16} different key values before repeating. The state of the key generator is held in the SWT_SK register. For example, if SWT_SK[SK] is 0x0100 then the service sequence keys are 0x1103, 0x2136. In this mode, each time a valid key is written to the SWT_SR register, the SWT_SK register is updated. So, after servicing the watchdog by writing 0x1103 and then 0x2136 to the SWT_SR[WSC] field, SWT_SK[SK] is 0x2136 and the next key sequence is 0x3499, 0x7E2C.

Equation 11

$$SK_{n+1} = (17 * SK_n + 3) \bmod 2^{16}$$

Accesses to SWT registers occur with no peripheral bus wait states. (The peripheral bus bridge may add one or more system wait states.) However, due to synchronization logic in the SWT design, recognition of the service sequence or configuration changes may require up to three system plus seven counter clock cycles.

If window mode is enabled (SWT_MCR[WND] bit is set), the service sequence must be performed in the last part of the time-out period defined by the window register. The window is open when the down counter is less than the value in the SWT_WN register. Outside of this window, service sequence writes are invalid accesses and generate a bus error or reset depending on the value of the SWT_MCR[RIA] bit. For example, if the SWT_TO register is set to 5000 and SWT_WN register is set to 1000 then the service sequence must be performed in the last 20% of the time-out period. There is a short lag in the time it takes for the window to open due to synchronization logic in the watchdog design. This delay could be up to three system plus four counter clock cycles.

The interrupt then reset bit (SWT_MCR[ITR]) controls the action taken when a time-out occurs. If the SWT_MCR[ITR] bit is not set, a reset is generated immediately on a time-out. If the SWT_MCR[ITR] bit is set, an initial time-out causes the SWT to generate an interrupt and load the down counter with the time-out period. If the service sequence is not written before the second consecutive time-out, the SWT generates a system reset. The interrupt is indicated by the time-out interrupt flag (SWT_IR[TIF]). The interrupt request is cleared by writing a ‘1’ to the SWT_IR[TIF] bit.

The SWT_CO register shows the value of the down counter when the watchdog is disabled. When the watchdog is enabled this register is cleared. The value shown in this register can lag behind the value in the internal counter for up to six system plus eight counter clock cycles.

The SWT_CO can be used during a software self test of the SWT. For example, the SWT can be enabled and not serviced for a fixed period of time less than the time-out value. Then the SWT can be disabled (SWT_MCR[WEN] cleared) and the value of the SWT_CO read to determine if the internal down counter is working properly.

20 Boot Assist Module (BAM)

20.1 Overview

The Boot Assist Module (BAM) is a 4 KB block of read-only memory (ROM) containing the boot program code for this device.

The BAM program supports three different boot modes:

- Boot from internal Flash
- Serial boot via SCI or CAN interface
- Serial boot via SCI or CAN interface with baud rate detection

The BAM program is executed by the core just after a device reset. Depending on the boot mode, the program initializes appropriate minimum device resources to start user code application.

20.2 Features

- Initial core MMU setup with minimum address translation for all internal device resources
- MMU configuration to boot user application, compiled as Power Architecture technology code or as VLE code
- Passes control to user application code in the internal flash memory
- Automatic switch to Serial Boot mode if internal flash is blank or invalid
- Serial boot by loading user program via CAN bus or eSCI to the internal SRAM
  - User programmable 64-bit password protection
  - Optional automatic detection of the host SCI or CAN speed
- Controls core Watchdog Timer or/and the Software Watchdog Timer (SWT)

20.3 Modes of operation

20.3.1 Normal mode

The BAM program is executed immediately following the negation of reset.

20.3.2 Debug mode

The BAM program is not executed when the device comes out of reset in OnCE debug mode. The user must provide the required device initialization using the development tool before accessing the device resources.

20.3.3 Internal boot mode

This mode of operation is intended for systems that boot from internal flash memory. The internal flash memory is used for all code and all boot configuration data.

20.3.4 Serial boot mode

This mode of operation is intended to load a user program into internal SRAM, using either the eSCI or CAN serial interface, then to execute that program. The program can then be used to control the download of data and erasing/programming of the internal or external flash memory.

20.3.5 Calibration bus boot mode

Calibration bus boot is not supported. External bus boot is supported instead.

20.4 Memory map

The BAM occupies 16 KB of memory space, 0xFFFF_C000 to 0xFFFF_FFFF. The actual code size of the BAM program is less than 4 KB and starts at 0xFFFF_F000, repeating itself down every 4 kilobytes in the BAM address space. The CPU starts the BAM program execution at its reset vector from address 0xFFFF_FFFC.

The BAM exits to the user code at 0xFFFF_FFF8 address. The last BAM executed instruction is a BLR. The link register is preloaded with the user application start address. When booting from internal or external flash, the start address is taken from next to valid RCHW 32-bit word.When the device boots serially the start address is set according the serial boot protocol.

Table 324 shows the BAM address map.

Table 324. BAM memory map

Address	Description
0xFFFF_C000 – 0xFFFF_EFFF	BAM program mirrored
0xFFFF_F000 – 0xFFFF_FFFF	BAM program
0xFFFF_FFFC	Device reset vector
0xFFFF_FFF8	BAM Last executed instruction

20.5 Functional description

20.5.1 BAM Program flow chart

The BAM program flow chart is shown in *Figure 317*.

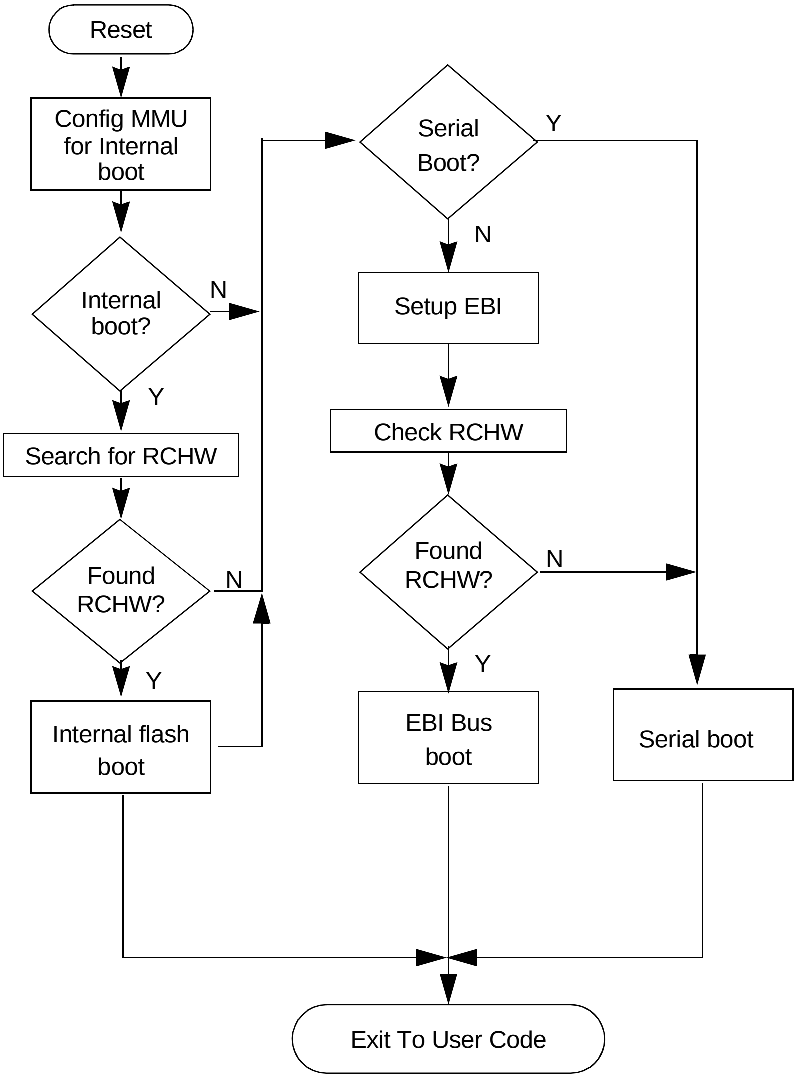


Figure 317. BAM program flow chart

20.5.2 BAM program operation

The BAM is accessed by the device core after the negation of $\overline{\text{RSTOUT}}$, before user code starts. First, the BAM program configures the core MMU to allow access to all device internal resources, according to *Table 325*. This MMU setup remains the same for internal Flash boot mode.

Table 325. MMU configuration for internal flash boot

TLB entry	Region	Logical base address	Physical base address	Size	Attributes
0	Peripheral Bridge B(1) and BAM	0xFFF0_0000	0xFFF0_0000	1 MB	Guarded Big endian Global PID
1	Internal Flash	0x0000_0000	0x0000_0000	16 MB	Not guarded Big endian Global PID
2	EBI	0x2000_0000	0x0000_0000	16 MB	Not guarded Big endian Global PID
3	Internal SRAM	0x4000_0000	0x4000_0000	256 KB	Not guarded Big endian Global PID
4	Peripheral Bridge A(1)	0xC3F0_0000	0xC3F0_0000	1 MB	Guarded Big endian Global PID

1. This device has only a single peripheral bridge, but to match the memory map of other devices the peripherals are mapped to appear as if they are on two different peripheral bridges.

The MMU regions are mapped with logical address the same as physical address

After the MMU configuration, the BAM program checks the BOOTCFG field of the reset status register (SIU_RSR) and the appropriate boot sequence is started as shown in *Table 326*.

Depending on the values stored in the censorship word and serial boot control word in the shadow row of the internal Flash memory, the internal Flash memory can be enabled or disabled, the Nexus port can be enabled or disabled, the password received in the serial boot mode is compared with the fixed public password or compared to a user programmable password in the internal Flash memory.

Table 326. Boot modes

BOOTCFG [0:1]	Censorship control 0x00FF_FDE0	Serial boot control 0x00FF_FDE2	Boot mode name	Internal Flash state	Nexus state	Serial password
00	!0x55AA(1)	Any value	Internal—Censored	Enabled	Disabled	Flash
	0x55AA		Internal—Public	Enabled	Enabled	Public
01	Any value	0x55AA	Serial—Flash password	Enabled	Disabled	Flash
		!0x55AA	Serial—Public password	Disabled	Enabled	Public
10	!0x55AA	Any value	External—No arbitration—Censored	Disabled	Enabled	Public
	0x55AA		External—No arbitration—Public	Enabled	Enabled	Public
11	Invalid value					

1. '!' = 'NOT,' as in!0x55AA, means all values except 0x55AA. Do not use 0x0000 or 0xFFFF for the value of the censorship control or serial boot control words.

The censorship word is a 32-bit word of data stored in the shadow row of internal flash memory. This memory location is read and interpreted by hardware as part of the boot process and is used in conjunction with the BOOTCFG pin to enable/disable the internal flash memory and the Nexus interface. The address of the Censorship word is 0x00FF_FDE0. The censorship word consists of two fields: censorship control and serial boot control. The censorship word is programmed during manufacturing to be 0x55AA_55AA. This results in a device that is not censored and uses a Flash-based password for serial boot mode.

Censorship Word @ 0x00FF_FDE0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	1	0	1	0	1	0	1	1	0	1	0	1	0	1	0
Censorship Control - showing an uncensored part (Factory Default)															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	1	0	1	1	0	1	0	1	0	1	0
Serial Boot Control - showing the use of the flash based password (Factory Default)															

Figure 318. Censorship word

The BAM program uses the state of bit SIU_CCR[DISNEX] to determine whether the serial password received in serial boot mode should be compared to a public password (fixed value of the 0xFEED_FACE_CAFE_BEEF) or needs to be compared to a Flash password - 64-bit data, stored in the shadow row of internal flash at address 0x00FF_FDD8. If the bit is set, the BAM uses the Flash serial password, if the bit is cleared, it uses the public password.

Flash Password @ 0x00FF_FDD8 - 0x00FF_FDDF

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
1	1	1	1	1	1	1	0	1	1	1	0	1	1	0	1
Serial Boot Password (0x00FF_FDD8) - 0xFEED (Factory Default)															

47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
1	1	1	1	1	0	1	0	1	1	0	0	1	1	1	0
Serial Boot Password (0x00FF_FDDA) - 0xFACE (Factory Default)															

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
1	1	0	0	1	0	1	0	1	1	1	1	1	1	1	0
Serial Boot Password (0x00FF_FDDC) - 0xCAFE (Factory Default)															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	1	1	1	0	1	1	1	1
Serial Boot Password (0x00FF_FDDE) - 0xBEEF (Factory Default)															

Figure 319. Serial boot flash password

A valid serial password must be always programmed, regardless the boot mode used. This provides capability to “rescue” the part using the serial boot mode, if the flash content becomes corrupted for whatever reason.

20.5.3 Reset configuration half word (RCHW)

The Reset Configuration Half Word defines boot options and has to be programmed by the user to predefined locations in the internal flash or at the beginning of the external flash device. The next 32-bit word after the RCHW has to be programmed with a starting address of the user application. The BAM program uses this location to fetch the address, where it passes control to.

Table 329 provides possible RCHW locations in the internal flash. When booting from the external flash device, the RCHW should reside in the very first 16-bit half word of the flash.

Figure 320 shows the fields of the RCHW.

BOOT_BLOCK_ADDRESS

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
				SWT	WTE	PS0	VLE	0	1	0	1	1	0	1	0
								Boot Identifier = 0x5A							

Figure 320. Reset configuration half word

Table 327. RCHW field description

Field	Description
bits 0–3	Reserved These bit values are ignored when the halfword is read. Write to 0 for future compatibility.
SWT	Watchdog timer enable This bit determines if the software watchdog timer is enabled after passing control to the user application code. 0 Disable software watchdog timer 1 Software watchdog timer maintains its default state out of reset, i.e. enabled. The timeout period is programmed to be 261600 system clocks.
WTE	Device core Watchdog timer enable This bit determines if the core software watchdog timer is enabled.after passing control to the user application code. 0 Disable core software watchdog timer 1 Software watchdog timer maintains its default state out of reset, i.e. enabled. The timeout period is programmed to be $2.5*2^{17}$ system clocks.
PS0	Port size Defines the width of the data bus connected to the memory on $\overline{\text{CS}}$0. After system reset, CS0 is changed to a 16-bit port by the BAM, which fetches the RCHW from either 16- or 32-bit external memories. Then the BAM reconfigures the EBI as a 16-bit bus or a 32-bit bus, according to the settings of this bit. 0 32-bit CS0 port size 1 16-bit CS0 port size Used in EBI boot mode only. Do not set the port to 32-bits if the device only has a 16-bit data bus.
VLE	VLE Code Indicator This bit is used to configure the MMU entries 1-3 coded as either Power Architecture instructions or as VLE instructions. 0 User code executes as Power Architecture code 1 User code executes as VLE code
BOOTID	Boot identifier This field serves two functions. First, it is used to indicate which block in flash memory contains the boot program. Second, it identifies whether the flash memory is programmed or invalid. The value of a valid boot identifier is 0x5A.

The watchdog timeout periods, when the watchdogs are controlled by RCHW, are shown in *Table 328*.

Table 328. Watchdog timeouts

Crystal frequency (MHz)	Core WD timeout(1) (ms)	SWT timeout(2) (ms)
8	40.1	32.7
12	27.3	21.8
16	20.5	16.35
20	16.4	13.08
40	8.2	6.54

1. 327,680 system clocks
2. 261,600 system clocks

Reset boot vector

The boot vector, shown in *Figure 321*, has to be programmed by the user to the user application code memory device (internal or external flash) to the next 32-bit word after the RCHW. The value from this location is used by the BAM program as a start address of the user application to switch to.

BOOT_BLOCK_ADDRESS + 0x0000_0004

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15

15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
A16	A17	A18	A19	A20	A21	A22	A23	A24	A25	A26	A27	A28	A29	A30	A31

Figure 321. Reset boot vector

20.5.4 Internal boot mode

When the BAM program detects internal Flash boot mode, it sets up a machine check exception handler because it will be accessing Flash memory locations that may be corrupted and cause a bus error. Then the BAM program tries to find a valid RCHW in six predefined locations. If a valid RCHW is not found, the BAM program proceeds to check of possibility of booting to the serial boot mode.

Finding reset configuration half word

The BAM searches the internal Flash memory for a valid reset configuration half word (RCHW). Possible RCHW locations are shown in *Table 329*.

Table 329. Possible RCHW locations in the internal flash

Block	Address
0	0x0000_0000
1	0x0000_4000
2	0x0000_8000
3	0x0000_C000
4	0x0001_0000
5	0x0001_4000
6	0x0001_8000
7	0x0001_C000
8	0x0002_0000
9	0x0003_0000

BOOT_BLOCK_ADDRESS is the address from *Table 329* where the BAM finds a valid RCHW. If the BAM program finds a valid RCHW, the core watchdog is enabled if the RCHW[WTE] bit is programmed high, the SWT is disabled if the RCHW[SWT] bit is programmed low, the BAM program fetches the reset vector from the address of the BOOT_BLOCK_ADDRESS + 0x4, and branches to the reset boot vector (shown in *Figure 321*). A user application should have a valid instruction at the reset boot vector address.

Enabling debug of a censored device

When a device is in a censored state, the debug port (JTAG/Nexus) is disabled and only JTAG BSDL commands can be used. Access to the Nexus/JTAG clients on a censored device requires inputting the proper password into the JTAG Censorship Control Register during reset.

Note: When the debug port is enabled on a censored device, it is enabled only until the next reset.

Figure 322 shows the logic that enables access to Nexus clients in a censored device using the JTAG port.

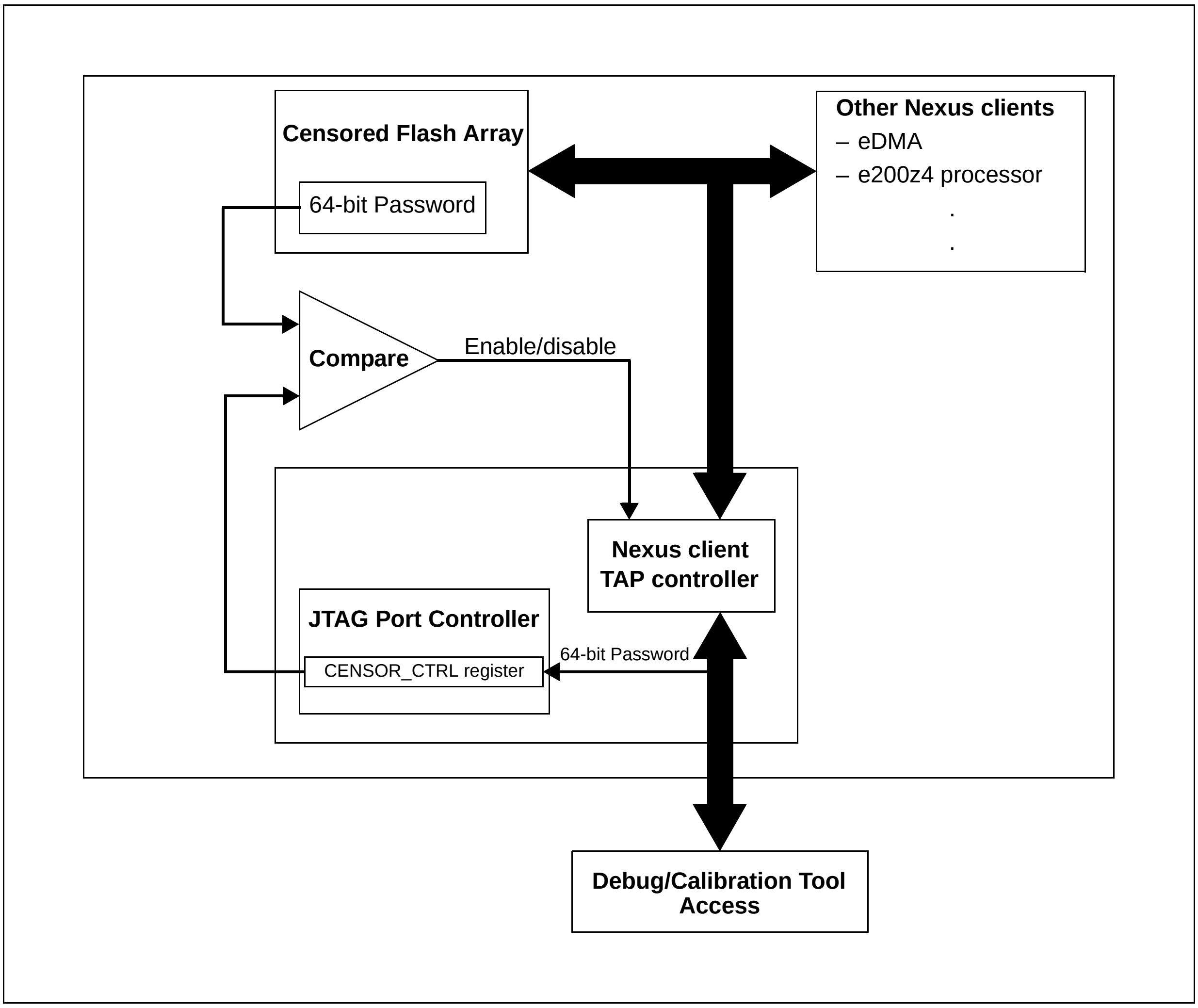


Figure 322. Enabling JTAG/Nexus port access on a censored device

The steps to enable the debug port on a censored device are as follows:

1. After the $\overline{\text{RSTOUT}}$ pin has is negated, hold the device in system reset state using a debugger or other tool.
2. While the device is being held in system reset state shift the 64-bit password into the CENSOR_CTRL register (see *Section , CENSOR_CTRL Register*) via the JTAG port using the JTAG ENABLE_CENSOR_CTRL instruction. The JTAG serial password is compared against the serial boot flash password from the flash shadow block.
3. If there is a match the Nexus client TAP controller enters normal operation mode and the flag SIU_CCR[DISNEX] is negated, indicating Nexus is enabled. Upon negation of reset the debug / calibration tool is able to access the device via NEXUS port and JTAG. If the JTAG serial password does not match the serial boot flash password or the serial boot flash password is an illegal password then the debug / calibration tool is not able to access the device.

   After the debug port is enabled, the tool can access the censored device and can erase and reprogram the shadow flash block in order to uncensor the device.

Note: If the shadow flash block is erased without reprogramming a new valid password before a reset it will contain an illegal password and the debug port will be inaccessible.

4. Subsequent resets will clear the JTAG censor password register and the Nexus client TAP controller will hold in reset again. Therefore, the tool must resend the JTAG serial password, as described above, in order to enable the Nexus client TAP controller again.

20.5.5 Serial boot mode

When the BAM program transitions to the Serial Boot mode, unused message buffers in CAN_A are used for the BAM program stack and variables and the SWT watchdog is reprogrammed with timeperiod greater than the default value.

The MMU setup depends on the way BAM enters the serial boot mode. The serial boot mode can run in either of two modes of operation:

- Standard serial boot mode using fixed baud rates derived from the crystal oscillator used
- Baud Rate Detection serial boot mode, which allows communication with adaptable speed, based on measured input signal

The Fixed Baud Rate mode or Baud Rate Detection mode are selected based on the state of the EVTO pin, recorded in the SIU_RSR[ABR] bit. If the bit is set, the Baud Rate Detection mode is selected if the bit is cleared, the Fixed Baud Rate is selected.

SIU_RSR[ABR] bit reflects the inverted state of the EVTO pin, thus to select Baud Rate Detection mode, the EVTO pin needs to be driven low.

When the Fixed Baud Rate mode is selected, the BAM program configures the SCI_A_RX pin to be the input of the eSCI_A module, CN_A_RX pin as an input, and CN_A_TX as an output of the CAN_A module.

When Baud Rate Detection Mode is selected, the BAM program configures SCI_A_RX and CN_A_RX pins as GPI inputs for polling their state by the CPU.

Table 330 shows the configuration summary for theSCI and CAN controllers pins.

Table 330. CAN/eSCI pins configuration for CAN/eSCI fixed baud rate boot modes

Pins	Reset function	Initial Serial Boot Mode		Serial Boot Mode after a valid CAN message received		Serial Boot Mode after a valid eSCI message received	
		Function	Pad configuration	Function	Pad configuration	Function	Pad configuration
CN_A_TX	GPIO	CN_A_TX	Push/Pull output, with medium slew rate	CN_A_TX	Push/Pull output, with medium slew rate	GPIO	—
CN_A_RX	GPIO	CN_A_RX	Input with pull-up and hysteresis	CN_A_RX	Input with pull-up and hysteresis	GPIO	—
SCI_A_TX	GPIO	GPIO	—	GPIO	—	SCI_A_TX	Push/Pull output, with medium slew rate
SCI_A_RX	GPIO	SCI_A_RX	Input with pull-up and hysteresis	GPIO	—	SCI_A_RX	Input with pull-up and hysteresis

The BAM configures the communication modules for reception with fixed baud rates as shown in the *Table 331* and waits for data reception.

Table 331. Serial boot mode – baud rate & watchdog summary

Crystal frequency (MHz)	System Clock frequency (MHz)	Desired eSCI Baud rate (baud)	Actual eSCI Baud rate (baud)	eSCI error (%)	CAN Baud rate (baud)	Core Watchdog(1) timeout period (s)	SWT timeout period during serial boot (s)
f_{xtal}	$f_{sys} = f_{xtal}$	f_{sys} / 833.33	f_{sys} / 832	—	f_{sys} / 40	$2.5 * 2^{27} / f_{sys}$	223696213 / f_{sys}
8	8	9600	9615.4	0.16	200K	42	27.96
12	12	14400	14423. 0	0.16	300K	28	18.64
16	16	19200	19230.8	0.16	400K	21	13.98
20	20	24000	24038.5	0.16	500K	16.8	11.18

1. The SWT is used as a watchdog during serial boot mode, but the core watchdog is enabled just before switching to the user application to provide compatibility with earlier parts.

If a message with 0x11 ID, containing 8 bytes, is received by the CAN controller first, the BAM program transitions to the Serial CAN Boot sub-mode, disabling eSCI, and reconfiguring the SCI_A_RX pin to its reset state.

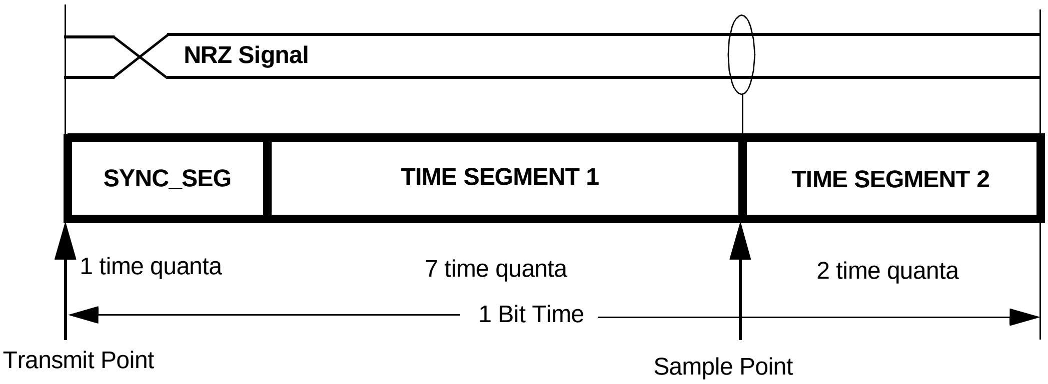
If a message from eSCI is received first, the BAM program transitions to the Serial SCI Boot submode, disables CAN_A module and configures its pins to their reset state.

Then the BAM program transitions to the serial download protocol execution.

CAN controller configuration in the fixed baud rate mode

The CAN controller is configured to operate at a baud rate equal to system frequency divided by 40, using the standard 11-bit identifier format detailed in CAN 2.0A specification. See *Table 331* for examples of baud rates. Only one message buffer 0 is used for all communications.

The bit timing is configured as shown in *Figure 323*.



Note: 1 Time quanta = 4 System clock periods

Figure 323. CAN bit timing

The BAM program ignores CAN errors and all received data is assumed to be good and is echoed out on the CN_A_TX signal. It is the responsibility of the host computer to compare the “echoes” with the sent data and restart the process if an error is detected.

SCI controller configuration in fixed baud rate mode

The eSCI is configured for 1 start bit, 8 data bits, no parity, 1 stop bit and to operate at a baud rate equal to system clock divided by 832. See *Table 331* for examples of baud rates.

The BAM program ignores the eSCI errors: All data received will be assumed to be good and will be echoed out on the TXD signal. It is the responsibility of the host computer to compare the echoes with the sent data and restart the process if an error is detected.

Serial boot mode download protocol

The download protocol follows four steps:

1. Host sends 64-bit password.
2. Host sends start address, size of download code in bytes, and VLE bit.
3. Host sends the application code data.
4. The device switches to the loaded code at the start address.

The communication is done in half-duplex manner, any transmission from host is followed by the device transmission. The host computer should not send data until it receives echo from the device. All multibyte data structures have to be sent most significant byte (MSB) first.

When the CAN is used for serial download, the data is packed into standard CAN messages in the following manner:

- A message with 0x11 ID and 8-byte length is used to send the password. The device transmits the same data, but the message ID is set to 0x1.
- A message with 0x12 ID and 8-byte length is used to send the start address, length, and the VLE mode bit. The device transmits back the same data, but with ID set to 0x2.
- Messages with 0x13 ID are used to send the downloaded data. The device transmits back received data with message ID of 0x3.

When the SCI is used for serial download, the data has to be sent on a byte-by-byte basis. the device transmits back the received data.

Download protocol execution

The BAM program executes the serial boot as follows:

1. Download 64-bit password.

   The received 8-byte password is checked for validity. For a password to be valid, none of its four 16-bit half words must equal 0x0000 or 0xFFFF.

   The BAM program then checks the censorship status of the device by checking the bit SIU_CCR[DISNEX]. If Nexus is disabled, the device is considered to be censored and the password is compared with a password stored in the shadow row in internal flash memory.

   If Nexus is enabled, the device is not considered to be censored and the password is compared to the fixed value = 0xFEED_FACE_CAFE_BEEF.

   If the password check fails, the device stops responding. To get the device out of that state, the RESET signal must be asserted.

   If the password check passes, the BAM transitions to the next step in the protocol.
2. Download start address, size of download, and VLE bit.

   The next 8 bytes received by the device are considered to contain a 32-bit start address, the VLE mode bit, and a 31-bit code length (see *Figure 324*).

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
START_ADDRESS[0:15]															

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
START_ADDRESS[16:31]															

32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
VLE	CODE_LENGTH[0:14]														

48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
CODE_LENGTH[15:30]															

Figure 324. Start address, VLE bit and download size in bytes

The start address defines where the received data will be stored and where the device will branch after the download is finished. The two least significant bits of the start address are ignored by the BAM program, such that the loaded code should be 32-bit word aligned.

The length defines how many data bytes to be loaded.

The VLE mode bit instructs the device to program MMU entries 1–3 with VLE attribute. If it is 1, the downloaded code must be compiled to VLE instructions, if it is 0 the code contains Power instructions.

3. Download data.

   Each byte of data received is stored in the device memory, starting at the address specified in the previous protocol step, and incrementing through memory until the number of bytes of data received and stored in memory matches the number specified in the previous protocol step.

   BAM program buffers incoming data, collecting up to eight bytes. The buffered data is written to the RAM with 64-bit writes to prevent ECC errors, which may happen if the device RAM is protected by 64-bit ECC code.

   Once the buffered data is written to the RAM the BAM program refreshes the SWT watchdog.

Note: Only system RAM supports 64-bit writes; therefore, attempting to download data to other RAM apart from system RAM will cause errors.

If the start address of the downloaded data is not on an 8-byte boundary, the BAM will write 0x0 to the memory locations from the proceeding 8-byte boundary to the start address (maximum 4 bytes). The BAM also writes 0x0 to all memory locations from the last byte of data downloaded to the following 8 byte boundary (maximum 7 bytes) and additional 8 zero bytes to prevent possible ECC errors may be caused by the CPU prefetching.

4. Switch to the loaded code.

   The BAM program waits for the last echo message transmission to complete, then the active communication controller is disabled. Its pins revert to GPIO inputs.

   To provide compatibility with older devices, the BAM writes the core time base registers (TBU and TBL) with 0x0 and enables the core watchdog to cause a reset after a time-out period of 2.5 x 2^{27} system clock cycles and disables SWT watchdog. See *Table 331* for examples of time out periods.

   The BAM code passes control to the loaded code at start address, which was received in step 2 of the protocol.

Note: The loaded code must periodically refresh the core watchdog timer or change the timeout period to a value that will not cause resets during normal operation.

Baud rate detection procedure

To improve baud rate detection accuracy the baud rate detection routine is copied to the beginning of the system RAM from the BAM ROM. Then the CPU branches to the RAM.

The device configures the CN_A_RX and SCI_A_RX pins as general purpose inputs and starts to poll them until one of them goes low.

If the CN_A_RX pin transitions first, the BAM program starts CAN baud rate detection routine, ignoring SCI_A_RX. After detecting the CAN baud rate, the BAM program transitions to the CAN download protocol routine described above.

If the SCI_A_RX pin transitions first, the SCI baud rate detection and download protocol routines are called, ignoring any further CAN pins activity.

SCI baud rate detection

The host has to send a zero byte to allow the device to detect the serial link baud rate. The host transmits 1 start bit, 8 zero data bits and 1 stop bit. The device does not echo it.

The device polls the SCI_A_RX pin for high to low transition and starts the core Time Base counter (TBU and TBL). Then the device polls for low to high transition on the pin and when it happens, the device turns off the TB counter. The TB content is used to calculate incoming signal baud rate. The SCI baud rate is equal to the TB content divided by 144 (measured over 9 bits with 16 system clocks per bit).

CAN baud rate detection

The host transmits a zero length message with zero 11-bit ID and the device measures time over 40 bits, polling CN_A_RX pin for high and low, according to the sent data. The device does not acknowledge this message.

The CAN baud rate depends on the number of quanta per bit and serial clock frequency, which is defined by a prescaler. The CAN baud rate detection routine selects these parameters to maximize number of quanta per bit and achieve minimum difference between measured value and duration of the 40 CAN bits, to be programmed with selected pair of the parameters.

The CAN controller can be programmed with 8 to 25 number of quanta per bit. The bit timing parameters, selected by the baud rate detection routine, are shown in *Table 332*. (See FlexCAN chapter for the parameters definition).

Table 332. CAN bit timing lookup table

Time quanta per bit	Time segment 1		Time segment 2	RJW
	PROPSEG	PSEG1	PSEG2	
8	1	3	3	2
9	2	3	3	2
10	3	3	3	2
11	4	3	3	2
12	3	4	4	2
13	4	4	4	3
14	5	4	4	3
15	6	4	4	4
16	7	4	4	3
17	8	4	4	3
18	7	5	5	4
19	8	5	5	4
20	7	6	6	4
21	8	6	6	4
22	7	6	6	4
23	8	6	6	4

Table 332. CAN bit timing lookup table (continued)

Time quanta per bit	Time segment 1		Time segment 2	RJW
	PROPSEG	PSEG1	PSEG2	
24	7	7	7	4
25	8	7	7	4

Maximum and minimum speeds of the serial communication modules are defined by the device system frequency and shown in *Table 333*.

Table 333. Maximum and minimum detectable baud rates

$f_{sys} = f_{xtal}$ [MHz]	Max baud rate for CAN ($f_{sys}/8$)(1) [bit/s]	Min CAN baud rate ($f_{sys}/25/256$) [bit/s]	Max baud rate for SCI ($f_{sys}/160$) [bit/s]	Min baud rate for SCI ($f_{sys}/16/2^{16}$) [bit/s]
8	1M	1250	50K	7.6
12	1M	1875	75K	11.5
16	1M	2500	100K	15.2
20	1M	3125	125K	19

1. Limited to 1 Mbit/s by CAN standard

21 Configurable Enhanced Modular IO Subsystem (eMIOS200)

21.1 Device-specific features

- Sixteen 24-bit wide channels
- 3 channels internal timebases can be shared between channels
- 1 timebase from the eTPU can be imported and used by the channels
- Global enable feature for all eMIOS200 and eTPU timebases
- Doze mode is not supported
- Each channel has its own pin (not available on all package types)

21.2 Introduction

The eMIOS200 module provides the capability to generate or measure timed events, for example generating PWM waveforms or measuring input pulse width. It is implemented with its own configuration of timer channels to suit the target applications needs, while maintaining full backwards compatibility with previous eMIOS implementations. The SPC564A70 has one eMIOS200 module that implements twenty-four 24-bit counters.

The overall architecture of the eMIOS200 resembles that of its predecessor, the MIOS. The MIOS timer block provided a framework where a set of sublocks with different timer functions were assembled to attend the specific needs of a device. The SPC564A70 eMIOS200 builds on this concept by using a modified Unified Channel module that provides a superset of the functionality of individual MIOS channels, while providing a consistent user interface. This allows more flexibility as each channel can be programmed for different functions in different applications of the device. In addition, the eMIOS200 architecture allows the use of dedicated channels that perform specific functions not included in MIOS inheritance.

Note: The SPC564A70 eMIOS200 uses a modified version of the Unified Channel block that contains a reduced set of functions . See Section 21.2.3, Channel configurations, for details.

Figure 325 shows the block diagram of the SPC564A70 eMIOS200 module.

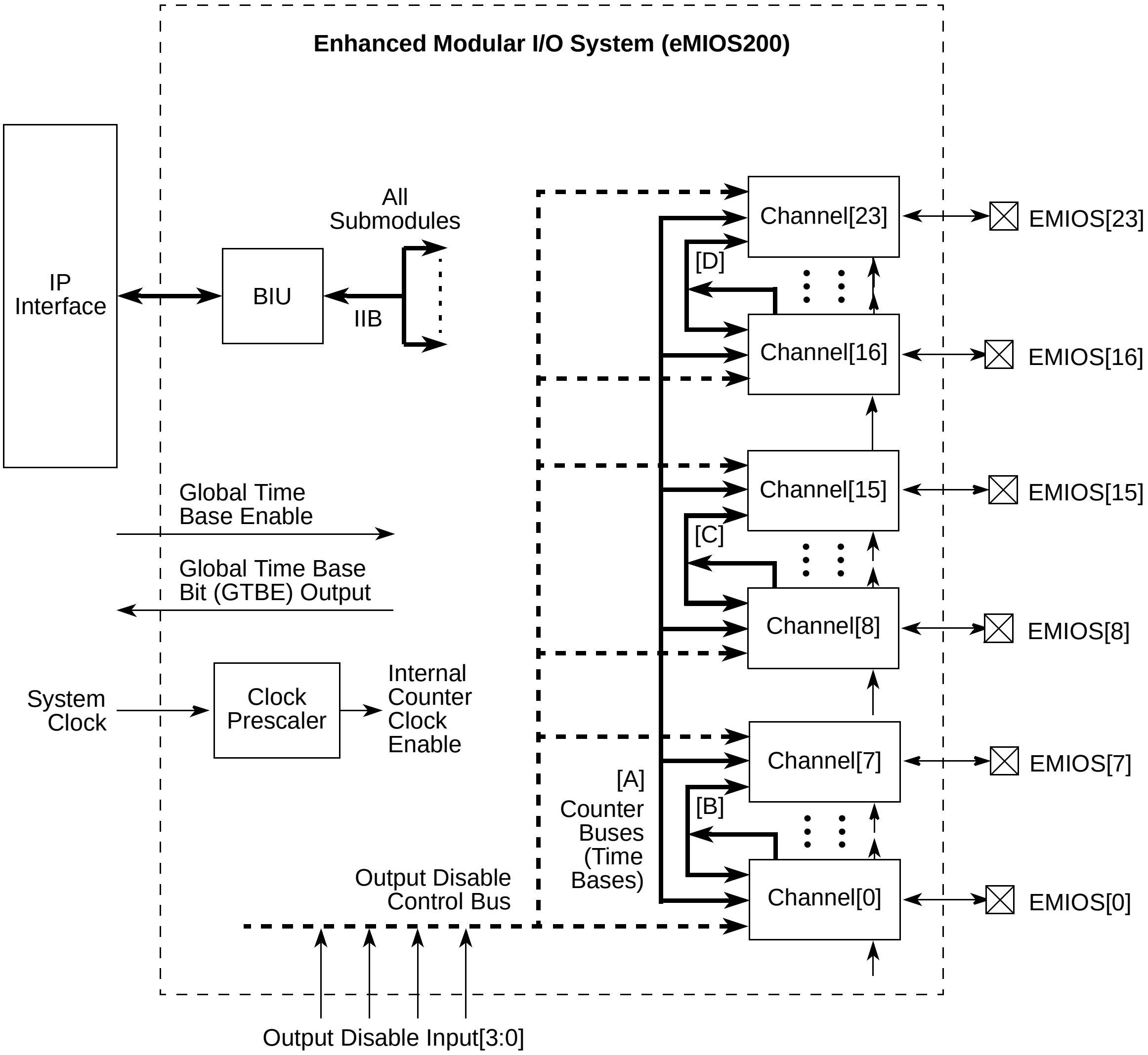


Figure 325. eMIOS200 block diagram

21.2.1 Features

The eMIOS timer module provides the capability to generate or measure events in hardware.

The eMIOS module features include:

- Twenty-four 24-bit wide channels
- 3 channels’ internal timebases can be shared between channels
- 1 timebase from eTPU2 can be imported and used by the channels
- Global enable feature for all eMIOS and eTPU timebases
- Dedicated pin for each channel (not available on all package types)

Each channel (0–23) supports the following functions:

- General-purpose input/output (GPIO)
- Single-action input capture (SAIC)
- Single-action output compare (SAOC)
- Output pulse-width modulation buffered (OPWMB)
- Input period measurement (IPM)
- Input pulse-width measurement (IPWM)
- Double-action output compare (DAOC)
- Modulus counter buffered (MCB)
- Output pulse width and frequency modulation buffered (OPWFMB)

21.2.2 Modes of operation

There are three main operating modes of eMIOS200: run mode, module disable mode, and debug mode.

- Run mode is the normal operation mode.
- Module disable mode is used for MCU power management.

  The clock to the non-memory-mapped logic in the eMIOS200 is stopped while in module disable mode. Module disable mode is entered when EMIOS_MCR[MDIS] = 1.
- Debug mode is individually programmed for each channel.

  When entering this mode, the unified channel registers' contents are frozen but remain available for read and write access through the IP interface.

21.2.3 Channel configurations

Table 334 shows all configurations available in the SPC564A70 eMIOS200. These modes are described in *Section , Channel modes of operation*.

Note: Not all configurations are available on all channels. If an unimplemented mode is selected (by writing a reserved value to MODE[0:6] in a channel's EMIOS_CCR[n]) the results are unpredictable. See Section , eMIOS200 Channel Control Register (EMIOS_CCR[n]), for more detail.

Table 334. All available SPC564A70 eMIOS channel configurations

Description	Name	Location
General Purpose Input / Output	GPIO	*on page 21-582*
Single Action Input Capture	SAIC	*on page 21-582*
Single Action Output Compare	SAOC	*on page 21-583*
Input Pulse Width Measurement	IPWM	*on page 21-585*
Input Period Measurement	IPM	*on page 21-586*
Double Action Output Compare	DAOC	*on page 21-588*
Modulus Counter Buffered (Up / Down)	MCB	*on page 21-589*
Output Pulse Width and Frequency Modulation Buffered	OPWFMB	*on page 21-592*
Output Pulse Width Modulation Buffered	OPWMB	*on page 21-597*

21.3 External signals description

Each channel has one external input and one external output signal. Depending on the chip integration, the input and output signals can be connected to two separate pins, or to a single bidirectional pin. See *Chapter 3 Signal Description* for details.

21.4 Memory map/register definition

21.4.1 Memory map

The overall address map organization is shown in *Table 335*.

Note: Whenever an access to either an absent register, an absent channel or a reserved address is performed, the eMIOS200 responds by asserting a Transfer Error signal from the slave bus (or STAC bus).

Table 335. SPC564A70 eMIOS memory map

Offset from EMIOS_BASE (0xC3FA_0000)	Register	Location
Global registers		
0x0000	EMIOS_MCR—Module Configuration Register	*on page 21-567*
0x0004	EMIOS_GFR—Global FLAG Register	*on page 21-569*
0x0008	EMIOS_OUDR—Output Update Disable Register	*on page 21-570*
0x000C	EMIOS_UCDIS—Channel Disable Register	*on page 21-571*
0x000C–0x001F	Reserved	
Channel 0 registers		
0x0020	EMIOS_CADR[0]—Channel A Data Register	*on page 21-571*
0x0024	EMIOS_CBDR[0]—Channel B Data Register	*on page 21-572*
0x0028	EMIOS_CCNTR[0]—Channel Counter Register	*on page 21-573*
0x002C	EMIOS_CCR[0]—Channel Control Register	*on page 21-573*
0x0030	EMIOS_CSR[0]—Channel Status Register	*on page 21-578*
0x0034	EMIOS_ALTA[0](1)—Alternate A Register	*on page 21-579*
0x0038–0x003F	Reserved	

Table 335. SPC564A70 eMIOS memory map (continued)

Offset from EMIOS_BASE (0xC3FA_0000)	Register	Location
Channel 1 registers		
0x0040	EMIOS_CADR[1]—A Register	*on page 21-571*
0x0044	EMIOS_CBDR[1]—B Register	*on page 21-572*
0x0048	EMIOS_CCNTR[1]—Counter Register	*on page 21-573*
0x004C	EMIOS_CCR[1]—Control Register	*on page 21-573*
0x0050	EMIOS_CSR[1]—Status Register	*on page 21-578*
0x0054	EMIOS_ALTA[1](1)—Alternate A Register	*on page 21-579*
0x0058–0x005F	Reserved	
Channel 2 registers		
0x0060	EMIOS_CADR[2]—A Register	*on page 21-571*
0x0064	EMIOS_CBDR[2]—B Register	*on page 21-572*
0x0068	EMIOS_CCNTR[2]—Counter Register	*on page 21-573*
0x006C	EMIOS_CCR[2]—Control Register	*on page 21-573*
0x0070	EMIOS_CSR[2]—Status Register	*on page 21-578*
0x0074	EMIOS_ALTA[2](1)—Alternate A Register	*on page 21-579*
0x0078–0x007F	Reserved	
Channel 3 registers		
0x0080	EMIOS_CADR[3]—A Register	*on page 21-571*
0x0084	EMIOS_CBDR[3]—B Register	*on page 21-572*
0x0088	EMIOS_CCNTR[3]—Counter Register	*on page 21-573*
0x008C	EMIOS_CCR[3]—Control Register	*on page 21-573*
0x0090	EMIOS_CSR[3]—Status Register	*on page 21-578*

Table 335. SPC564A70 eMIOS memory map (continued)

Offset from EMIOS_BASE (0xC3FA_0000)	Register	Location
0x0094	EMIOS_ALTA[3]⁽¹⁾—Alternate A Register	*on page 21-579*
0x0098–0x009F	Reserved	
Channel 4 registers		
0x00A0	EMIOS_CADR[4]—A Register	*on page 21-571*
0x00A4	EMIOS_CBDR[4]—B Register	*on page 21-572*
0x00A8	EMIOS_CCNTR[4]—Counter Register	*on page 21-573*
0x00AC	EMIOS_CCR[4]—Control Register	*on page 21-573*
0x00B0	EMIOS_CSR[4]—Status Register	*on page 21-578*
0x00B4	EMIOS_ALTA[4]⁽¹⁾—Alternate A Register	*on page 21-579*
0x00B8–0x00BF	Reserved	
Channel 5 registers		
0x00C0	EMIOS_CADR[5]—A Register	*on page 21-571*
0x00C4	EMIOS_CBDR[5]—B Register	*on page 21-572*
0x00C8	EMIOS_CCNTR[5]—Counter Register	*on page 21-573*
0x00CC	EMIOS_CCR[5]—Control Register	*on page 21-573*
0x00D0	EMIOS_CSR[5]—Status Register	*on page 21-578*
0x00D4	EMIOS_ALTA[5]⁽¹⁾—Alternate A Register	*on page 21-579*
0x00D8–0x00DF	Reserved	
Channel 6 registers		
0x00E0	EMIOS_CADR[6]—A Register	*on page 21-571*
0x00E4	EMIOS_CBDR[6]—B Register	*on page 21-572*
0x00E8	EMIOS_CCNTR[6]—Counter Register	*on page 21-573*

Table 335. SPC564A70 eMIOS memory map (continued)

Offset from EMIOS_BASE (0xC3FA_0000)	Register	Location
0x00EC	EMIOS_CCR[6]—Control Register	*on page 21-573*
0x00F0	EMIOS_CSR[6]—Status Register	*on page 21-578*
0x00F4	EMIOS_ALTA[6](1)—Alternate A Register	*on page 21-579*
0x00F8–0x00FF	Reserved	
Channel 7 registers		
0x0100	EMIOS_CADR[7]—A Register	*on page 21-571*
0x0104	EMIOS_CBDR[7]—B Register	*on page 21-572*
0x0108	EMIOS_CCNTR[7]—Counter Register	*on page 21-573*
0x010C	EMIOS_CCR[7]—Control Register	*on page 21-573*
0x0110	EMIOS_CSR[7]—Status Register	*on page 21-578*
0x0114	EMIOS_ALTA[7][1]—Alternate A Register	*on page 21-579*
0x0118–0x011F	Reserved	
Channel 8 registers		
0x0120	EMIOS_CADR[8]—A Register	*on page 21-571*
0x0124	EMIOS_CBDR[8]—B Register	*on page 21-572*
0x0128	EMIOS_CCNTR[8]—Counter Register	*on page 21-573*
0x012C	EMIOS_CCR[8]—Control Register	*on page 21-573*
0x0130	EMIOS_CSR[8]—Status Register	*on page 21-578*
0x0134	EMIOS_ALTA[8](1)—Alternate A Register	*on page 21-579*
0x0138–0x013F	Reserved	
Channel 9 registers		
0x0140	EMIOS_CADR[9]—A Register	*on page 21-571*

Table 335. SPC564A70 eMIOS memory map (continued)

Offset from EMIOS_BASE (0xC3FA_0000)	Register	Location
0x0144	EMIOS_CBDR[9]—B Register	*on page 21-572*
0x0148	EMIOS_CCNTR[9]—Counter Register	*on page 21-573*
0x014C	EMIOS_CCR[9]—Control Register	*on page 21-573*
0x0150	EMIOS_CSR[9]—Status Register	*on page 21-578*
0x0154	EMIOS_ALTA[9](1)—Alternate A Register	*on page 21-579*
0x0158–0x015F	Reserved	
Channel 10 registers		
0x0160	EMIOS_CADR[10]—A Register	*on page 21-571*
0x0164	EMIOS_CBDR[10]—B Register	*on page 21-572*
0x0168	EMIOS_CCNTR[10]—Counter Register	*on page 21-573*
0x016C	EMIOS_CCR[10]—Control Register	*on page 21-573*
0x0170	EMIOS_CSR[10]—Status Register	*on page 21-578*
0x0174	EMIOS_ALTA[10](1)—Alternate A Register	*on page 21-579*
0x0178–0x017F	Reserved	
Channel 11 registers		
0x0180	EMIOS_CADR[11]—A Register	*on page 21-571*
0x0184	EMIOS_CBDR[11]—B Register	*on page 21-572*
0x0188	EMIOS_CCNTR[11]—Counter Register	*on page 21-573*
0x018C	EMIOS_CCR[11]—Control Register	*on page 21-573*
0x0190	EMIOS_CSR[11]—Status Register	*on page 21-578*
0x0194	EMIOS_ALTA[11](1)—Alternate A Register	*on page 21-579*
0x0198–0x019F	Reserved	

Table 335. SPC564A70 eMIOS memory map (continued)

Offset from EMIOS_BASE (0xC3FA_0000)	Register	Location
Channel 12 registers		
0x01A0	EMIOS_CADR[12]—A Register	*on page 21-571*
0x01A4	EMIOS_CBDR[12]—B Register	*on page 21-572*
0x01A8	EMIOS_CCNTR[12]—Counter Register	*on page 21-573*
0x01AC	EMIOS_CCR[12]—Control Register	*on page 21-573*
0x01B0	EMIOS_CSR[12]—Status Register	*on page 21-578*
0x01B4	EMIOS_ALTA[12]⁽¹⁾—Alternate A Register	*on page 21-579*
0x01B8–0x01BF	Reserved	
Channel 13 registers		
0x01C0	EMIOS_CADR[13]—A Register	*on page 21-571*
0x01C4	EMIOS_CBDR[13]—B Register	*on page 21-572*
0x01C8	EMIOS_CCNTR[13]—Counter Register	*on page 21-573*
0x01CC	EMIOS_CCR[13]—Control Register	*on page 21-573*
0x01D0	EMIOS_CSR[13]—Status Register	*on page 21-578*
0x01D4	EMIOS_ALTA[13]⁽¹⁾—Alternate A Register	*on page 21-579*
0x01D8–0x01DF	Reserved	
Channel 14 registers		
0x01E0	EMIOS_CADR[14]—A Register	*on page 21-571*
0x01E4	EMIOS_CBDR[14]—B Register	*on page 21-572*
0x01E8	EMIOS_CCNTR[14]—Counter Register	*on page 21-573*
0x01EC	EMIOS_CCR[14]—Control Register	*on page 21-573*
0x01F0	EMIOS_CSR[14]—Status Register	*on page 21-578*

Table 335. SPC564A70 eMIOS memory map (continued)

Offset from EMIOS_BASE (0xC3FA_0000)	Register	Location
0x01F4	EMIOS_ALTA[14][1]—Alternate A Register	*on page 21-579*
0x01F8–0x01FF	Reserved	
Channel 15 registers		
0x0200	EMIOS_CADR[15]—A Register	*on page 21-571*
0x0204	EMIOS_CBDR[15]—B Register	*on page 21-572*
0x0208	EMIOS_CCNTR[15]—Counter Register	*on page 21-573*
0x020C	EMIOS_CCR[15]—Control Register	*on page 21-573*
0x0210	EMIOS_CSR[15]—Status Register	*on page 21-578*
0x0214	EMIOS_ALTA[15][1]—Alternate A Register	*on page 21-579*
0x0218–0x021F	Reserved	
Channel 16 registers		
0x0220	EMIOS_CADR[16]—A Register	*on page 21-571*
0x0224	EMIOS_CBDR[16]—B Register	*on page 21-572*
0x0228	EMIOS_CCNTR[16]—Counter Register	*on page 21-573*
0x022C	EMIOS_CCR[16]—Control Register	*on page 21-573*
0x0230	EMIOS_CSR[16]—Status Register	*on page 21-578*
0x0234	EMIOS_ALTA[16][1]—Alternate A Register	*on page 21-579*
0x0238–0x023F	Reserved	
Channel 17 registers		
0x0240	EMIOS_CADR[17]—A Register	*on page 21-571*
0x0244	EMIOS_CBDR[17]—B Register	*on page 21-572*
0x0248	EMIOS_CCNTR[17]—Counter Register	*on page 21-573*

Table 335. SPC564A70 eMIOS memory map (continued)

Offset from EMIOS_BASE (0xC3FA_0000)	Register	Location
0x024C	EMIOS_CCR[17]—Control Register	*on page 21-573*
0x0250	EMIOS_CSR[17]—Status Register	*on page 21-578*
0x0254	EMIOS_ALTA[17](1)—Alternate A Register	*on page 21-579*
0x0258–0x025F	Reserved	
Channel 18 registers		
0x0260	EMIOS_CADR[18]—A Register	*on page 21-571*
0x0264	EMIOS_CBDR[18]—B Register	*on page 21-572*
0x0268	EMIOS_CCNTR[18]—Counter Register	*on page 21-573*
0x026C	EMIOS_CCR[18]—Control Register	*on page 21-573*
0x0270	EMIOS_CSR[18]—Status Register	*on page 21-578*
0x0274	EMIOS_ALTA[18](1)—Alternate A Register	*on page 21-579*
0x0278–0x027F	Reserved	
Channel 19 registers		
0x0280	EMIOS_CADR[19]—A Register	*on page 21-571*
0x0284	EMIOS_CBDR[19]—B Register	*on page 21-572*
0x0288	EMIOS_CCNTR[19]—Counter Register	*on page 21-573*
0x028C	EMIOS_CCR[19]—Control Register	*on page 21-573*
0x0290	EMIOS_CSR[19]—Status Register	*on page 21-578*
0x0294	EMIOS_ALTA[19](1)—Alternate A Register	*on page 21-579*
0x0298–0x029F	Reserved	
Channel 20 registers		
0x02A0	EMIOS_CADR[20]—A Register	*on page 21-571*

Table 335. SPC564A70 eMIOS memory map (continued)

Offset from EMIOS_BASE (0xC3FA_0000)	Register	Location
0x02A4	EMIOS_CBDR[20]—B Register	*on page 21-572*
0x02A8	EMIOS_CCNTR[20]—Counter Register	*on page 21-573*
0x02AC	EMIOS_CCR[20]—Control Register	*on page 21-573*
0x02B0	EMIOS_CSR[20]—Status Register	*on page 21-578*
0x02B4	EMIOS_ALTA[20](1)—Alternate A Register	*on page 21-579*
0x02B8–0x02BF	Reserved	
Channel 21 registers		
0x02C0	EMIOS_CADR[21]—A Register	*on page 21-571*
0x02C4	EMIOS_CBDR[21]—B Register	*on page 21-572*
0x02C8	EMIOS_CCNTR[21]—Counter Register	*on page 21-573*
0x02CC	EMIOS_CCR[21]—Control Register	*on page 21-573*
0x02D0	EMIOS_CSR[21]—Status Register	*on page 21-578*
0x02D4	EMIOS_ALTA[21](1)—Alternate A Register	*on page 21-579*
0x02D8–0x02DF	Reserved	
Channel 22 registers		
0x02E0	EMIOS_CADR[22]—A Register	*on page 21-571*
0x02E4	EMIOS_CBDR[22]—B Register	*on page 21-572*
0x02E8	EMIOS_CCNTR[22]—Counter Register	*on page 21-573*
0x02EC	EMIOS_CCR[22]—Control Register	*on page 21-573*
0x02F0	EMIOS_CSR[22]—Status Register	*on page 21-578*
0x02F4	EMIOS_ALTA[22](1)—Alternate A Register	*on page 21-579*
0x02F8–0x02FF	Reserved	

Table 335. SPC564A70 eMIOS memory map (continued)

Offset from EMIOS_BASE (0xC3FA_0000)	Register	Location
Channel 23 registers		
0x0300	EMIOS_CADR[23]—A Register	*on page 21-571*
0x0304	EMIOS_CBDR[23]—B Register	*on page 21-572*
0x0308	EMIOS_CCNTR[23]—Counter Register	*on page 21-573*
0x030C	EMIOS_CCR[23]—Control Register	*on page 21-573*
0x0310	EMIOS_CSR[23]—Status Register	*on page 21-578*
0x0314	EMIOS_ALTA[23](1)—Alternate A Register	*on page 21-579*
0x0318–0x3FFF	Reserved	

1. The alternate address register provides and alternate read-only address to access A2 channel register in GPIO modes. If EMIOS_CADR[*n*] is used with EMIOS_ALTA[*n*], both A1 and A2 registers can be accessed in these modes.

21.4.2 Global registers

All global control registers are 32-bit wide but some do not use the most significant 8 bits because the SPC564A70 has 24 channels and 24-bit counters.

eMIOS200 Module Configuration Register (EMIOS_MCR)

The EMIOS_MCR contains global control bits for the eMIOS200 module.

Figure 326. eMIOS200 Module Configuration Register (EMIOS_MCR)

Address: EMIOS_BASE (0xC3FA_0000) + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	MDIS	FRZ	GTBE	ETB	GPREN	0	0	0	0	0	0	SRV[0:3]			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	GPRE[0:7]								0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 336. EMIOS_MCR field description

Field	Description
MDIS	Module Disable Puts the eMIOS200 in low power mode. The MDIS bit is used to stop the clock to the module, except access to the EMIOS_MCR, EMIOS_OUDR and EMIOS_UCDIS registers. 0 Clock is running 1 Enter low power mode
FRZ	Freeze Enables the eMIOS200 to freeze the channel registers when Debug Mode is requested at the MCU level. Each channel should have the FREN bit set in its EMIOS_CCR[n] register order to enter the freeze state. While in Freeze state, the eMIOS200 continues to operate to allow the MCU access to the channel registers. The channel remains frozen until the FRZ bit is written to zero, the MCU exits Debug mode or the channel’s FREN bit is cleared. 0 Exit freeze state 1 Stops channels operation when in Debug mode and the FREN bit is set in the EMIOS_CCR[n] register
GTBE	Global Time Base Enable The GTBE bit is used to export a Global Time Base Enable from the module and provide a method to start time bases of several blocks simultaneously. 0 Global Time Base Enable Out signal negated 1 Global Time Base Enable Out signal asserted The Global Time Base Enable input controls the internal counters. When asserted, Internal counters are enabled. When negated, internal counters disabled.
ETB	External Time Base The ETB bit selects the time base source that drives counter bus[A]. 0 Counter bus[A] assigned to eMIOS Channel 1 STAC drives counter bus [A] If ETB is set to select STAC as the counter bus[A] source, the GTBE must be set to enable the STAC to counter bus[A]. See the STAC bus configuration register (ETPU_REDCR) section of the eTPU chapter for more information about the STAC.
GPREN	Global Prescaler Enable The GPREN bit enables the prescaler counter. 0 Prescaler disabled (no clock) and prescaler counter is cleared 1 Prescaler enabled

Table 336. EMIOS_MCR field description (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>SRV[0:3]</td><td>Server time slot
Selects the address of a specific STAC server to which the STAC client submodule is assigned. See Section 21.5.3, STAC client submodule.

0000 eTPU engine A, TCR1
0001 Reserved
0010 eTPU engine A, TCR2
0011 Reserved
0100–1111 Reserved</td></tr>
<tr><td>GPRE[0:7]</td><td>Global Prescaler
The GPRE bits select the clock divider value for the global prescaler.
<table>
<tr><th>GPRE</th><th>Divide ratio</th></tr>
<tr><td>0000_0000</td><td>1</td></tr>
<tr><td>0000_0001</td><td>2</td></tr>
<tr><td>0000_0010</td><td>3</td></tr>
<tr><td>0000_0011</td><td>4</td></tr>
<tr><td>.
.
.</td><td>.
.
.</td></tr>
<tr><td>1111_1110</td><td>255</td></tr>
<tr><td>1111_1111</td><td>256</td></tr>
</table>
</td></tr>
</table>

eMIOS200 Global Flag Register (EMIOS_GFR)

Figure 327. eMIOS200 Global Flag Register (EMIOS_GFR)

Address: EMIOS_BASE (0xC3FA_0000) + 0x0004 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	F23	F22	F21	F20	F19	F18	F17	F16
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	F15	F14	F13	F12	F11	F10	F9	F8	F7	F6	F5	F4	F3	F2	F1	F0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 337. EMIOS_GFR field description

Field	Description
F*n*	FLAG The EMIOS_GFR is a read-only register that groups the FLAG bits from all channels. These bits are mirrors of the FLAG bits of each channel register (EMIOS_CSR[*n*]). See *Section , eMIOS200 Channel Status Register (EMIOS_CSR[n])*, for more detail.

eMIOS200 Output Update Disable Register (EMIOS_OUDR)

Figure 328. eMIOS200 Output Update Disable Register (EMIOS_OUDR)

Address: EMIOS_BASE (0xC3FA_0000) + 0x0008 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	OU 23	OU 22	OU 21	OU 20	OU 19	OU 18	OU 17	OU 16
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	OU 15	OU 14	OU 13	OU 12	OU 11	OU 10	OU9	OU8	OU7	OU6	OU5	OU4	OU3	OU2	OU1	OU0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 338. EMIOS_OUDR field description

Field	Description
OU*n*	Channel [*n*] Output Update Disable When running a channel in MC, MCB, or an output mode, values are written to registers A2 and B2. OU*n* bits are used to disable transfers from registers A2 to A1 and B2 to B1. Each bit controls one channel. 0 Transfer enabled. Depending on the operation mode, transfer may occur immediately or in the next period. Unless stated otherwise, transfer occurs immediately. 1 Transfers disabled.

eMIOS200 Channel Disable Register (EMIOS_UCDIS)

Figure 329. eMIOS200 Channel Disable Register (EMIOS_UCDIS)

Address: EMIOS_BASE (0xC3FA_0000) + 0x000C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	CHDIS23	CHDIS22	CHDIS21	CHDIS20	CHDIS19	CHDIS18	CHDIS17	CHDIS16
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CHDIS15	CHDIS14	CHDIS13	CHDIS12	CHDIS11	CHDIS10	CHDIS9	CHDIS8	CHDIS7	CHDIS6	CHDIS5	CHDIS4	CHDIS3	CHDIS2	CHDIS1	CHDIS0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 339. EMIOS_UCDIS field description

Field	Description
CHDIS*n*	Disable Channel [n] The CHDIS[n] bit is used to disable a channel by stopping its respective clock. 0 Channel [n] enabled 1 Channel [n] disabled

21.4.3 Channel registers

All channel control registers are 32-bit wide but some do not use the most significant 8 bits because the SPC564A70 has 24 channels and 24-bit counters.

eMIOS200 Channel A Data Register (EMIOS_CADR[n])

Figure 330. eMIOS200 Channel A Data Register (EMIOS_CADR[*n*])

Offset: UC[*n*] base address + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	A[0:23]							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	A[0:23]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Depending on the mode of operation, internal registers A1 or A2, used for matches and captures, can be assigned to address EMIOS_CADR[*n*]. A1 and A2 are cleared by reset. *Table 340* summarizes the EMIOS_CADR[*n*] write and read accesses for all operation modes. For more information see *Section , Channel modes of operation*.

eMIOS200 Channel B Data Register (EMIOS_CBDR[n])

Figure 331. eMIOS200 Channel B Data Register (EMIOS_CBDR[*n*])

Offset: UC[*n*] base address + 0x0004 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	B[0:23]							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	B[0:23]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Depending on the mode of operation, internal registers B1 or B2 can be assigned to address EMIOS_CBDR[*n*]. Both B1 and B2 are cleared by reset. *Table 340* summarizes the EMIOS_CBDR write and read accesses for all operation modes. For more information see *Section , Channel modes of operation*.

Depending on the channel's configuration, it may or may not have the EMIOS_CBDR. This means that if at least one mode that requires the register is implemented, then the register is present. Otherwise, it is absent. SPC564A70 has register B (EMIOS_CBDR) in all channels.

Table 340. EMIOS_CADR[*n*], EMIOS_CBDR[*n*], and EMIOS_ALTA[*n*] values assignment

Operation mode	**Register access**					
	write	**read**	**write**	**read**	**alt write**	**alt read**
GPIO	A1, A2	A1	B1,B2	B1	A2	A2
SAIC(1)	—	A2	B2	B2	—	—
SAOC(1)	A2	A1	B2	B2	—	—
IPWM	—	A2	—	B1	—	—
IPM	—	A2	—	B1	—	—
DAOC	A2	A1	B2	B1	—	—
MCB(1)	A2	A1	B2	B2	—	—
OPWFMB	A2	A1	B2	B1	—	—
OPWMB	A2	A1	B2	B1	—	—

1. In this mode, the register EMIOS_CBDR[n] is not used but B2 can be accessed.

eMIOS200 Channel Counter Register (EMIOS_CCNTR[n])

Figure 332. eMIOS200 Channel Counter Register (EMIOS_CCNTR[*n*])

Offset: UC[*n*] base address + 0x0008 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	C[0:23]							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	C[0:23]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1. In GPIO mode or freeze action, this register is writable.

The EMIOS_CCNTR[*n*] contains the value of the internal counter for eMIOS channel *n*. When GPIO mode is selected or the channel is frozen the EMIOS_CCNTR[*n*] is read/write. For all other modes, the EMIOS_CCNTR[*n*] is a read-only register. When entering some operation modes, this register is automatically cleared (refer to *Section , Channel modes of operation*, for details).

Depending on its configuration, a channel may have an internal counter or not. If at least one mode that requires the counter is implemented, the counter is present, otherwise it is not.

eMIOS200 Channel Control Register (EMIOS_CCR[n])

Figure 333. eMIOS200 Channel Control Register (EMIOS_CCR[*n*])

Offset: UC[*n*] base address + 0x000C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FREN	ODIS	ODISSL		UCPRE		UC PREN	DMA	0	IF				FCK	FEN	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	FORC MA	FORC MB	0	BSL		ED SEL	ED POL	MODE						
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register contains bits reflecting the status of channel input/output signals, the overflow condition of the internal counter, and several read/write control bits for eMIOS channel *n*.

Table 341. EMIOS_CCR field description

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>FREN</td><td>Freeze Enable
The FREN bit, if set and validated by bit EMIOS_MCR[FRZ], freezes all registers' values when in debug mode, allowing the MCU to perform debug functions.

0 Normal operation
1 Freeze unified channel registers' values</td></tr>
<tr><td>ODIS</td><td>Output Disable
The ODIS bit allows disabling the output pin when running any of the output modes with the exception of GPIO mode.

0 The output pin operates normally.
1 The output pin is driven to the value in EDPOL for OPWFMB and OPWMB modes and to the complement of EDPOL for other modes, but the channel continues to operate normally, that is, it continues to produce FLAG and matches. When the selected output disable input signal is negated, the output pin operates normally.</td></tr>
<tr><td>ODISSL</td><td>Output Disable Select
The ODISSL bits select one of the four output disable input signals.
<table>
<tr><th>ODISSL</th><th>Input signal</th></tr>
<tr><td>00</td><td>Output disable input 0</td></tr>
<tr><td>01</td><td>Output disable input 1</td></tr>
<tr><td>10</td><td>Output disable input 2</td></tr>
<tr><td>11</td><td>Output disable input 3</td></tr>
</table></td></tr>
<tr><td>UCPRE</td><td>Prescaler
The UCPRE bits select the clock divider value for the internal prescaler of the channel controlled by this register.
<table>
<tr><th>UCPRE</th><th>Divide ratio</th></tr>
<tr><td>00</td><td>1</td></tr>
<tr><td>01</td><td>2</td></tr>
<tr><td>10</td><td>3</td></tr>
<tr><td>11</td><td>4</td></tr>
</table></td></tr>
<tr><td>UCPREN</td><td>Prescaler Enable
The UCPREN bit enables the prescaler counter.

0 Prescaler disabled (no clock) and prescaler counter is loaded with UCPRE value
1 Prescaler enabled</td></tr>
</table>

Table 341. EMIOS_CCR field description (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>DMA</td><td>Direct Memory Access
The DMA bit selects whether the FLAG generation (see <i>Section , eMIOS200 Channel Status Register (EMIOS_CSR[n])</i>) is used as an interrupt or as a DMA request.

0 FLAG/overrun assigned to interrupt request
1 FLAG/overrun assigned to DMA request</td></tr>
<tr><td>IF</td><td>Input Filter
The IF bits control the programmable input filter, selecting the minimum input pulse width that can pass through the filter. For output modes, these bits have no meaning.
<table>
<tr><th>IF(1)</th><th>Minimum input pulse width [FLT_CLK periods]</th></tr>
<tr><td>0000</td><td>Bypassed(2)</td></tr>
<tr><td>0001</td><td>02</td></tr>
<tr><td>0010</td><td>04</td></tr>
<tr><td>0100</td><td>08</td></tr>
<tr><td>1000</td><td>16</td></tr>
<tr><td>All others</td><td>Reserved</td></tr>
</table>
1. Filter latency is three clock edges.
2. The input signal is synchronized before arriving to the digital filter.</td></tr>
<tr><td>FCK</td><td>Filter Clock Select
The FCK bit selects the clock source for the programmable input filter.

0 Prescaled clock
1 Main clock</td></tr>
<tr><td>FEN</td><td>FLAG Enable
The FEN bit allows the unified channel FLAG bit to generate an interrupt signal or a DMA request signal (the type of signal to be generated is defined by the DMA bit).

0 Disable (FLAG does not generate an interrupt or DMA request)
1 Enable (FLAG generates an interrupt or DMA request)</td></tr>
<tr><td>FORCMA</td><td>Force Match A
For output modes, the FORCMA bit is equivalent to a successful comparison on comparator A (except that the FLAG bit is not set). This bit is cleared by reset and is always read as 0. This bit is valid for every output operation mode which uses comparator A, otherwise it has no effect.

0 Has no effect
1 Force a match at comparator A

For input modes, the FORCMA bit is not used and writing to it has no effect.</td></tr>
</table>

Table 341. EMIOS_CCR field description (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>FORCMB</td><td>Force Match B
For output modes, the FORCMB bit is equivalent to a successful comparison on comparator B (except that the FLAG bit is not set). This bit is cleared by reset and is always read as 0. This bit is valid for every output operation mode which uses comparator B, otherwise it has no effect.

0 Has no effect.
1 Force a match at comparator B.

For input modes, the FORCMB bit is not used and writing to it has no effect.</td></tr>
<tr><td>BSL</td><td>Bus Select
The BSL bits are used to select either one of the counter buses or the internal counter to be used by the unified channel.

<table>
<tr><th>BSL</th><th>Selected bus</th></tr>
<tr><td>00</td><td>All channels: Counter bus[A].

When BSL = 0, Channel 23 must be in MCB mode.</td></tr>
<tr><td>01</td><td>– Counter bus[B] is driven by Channel 0 and can supply time base to channels 0 to 7.
– Counter bus[C] is driven by Channel 8 and can supply time base to channels 8 to 15.
– Counter bus[D] is driven by channel 16 and can supply time base to channels 16 to 23.

When BSL = 1, Channels 0, 8 and 16 must be in MCB mode.</td></tr>
<tr><td>10</td><td>Reserved</td></tr>
<tr><td>11</td><td>All channels: internal counter</td></tr>
</table>
</td></tr>
<tr><td>EDSEL</td><td>Edge Selection
For input modes, the EDSEL bit selects if the internal counter is triggered by both edges of a pulse or by a single edge only as defined by the EDPOL bit. When not shown in the mode of operation description, this bit has no effect.

0 Single edge triggering defined by the EDPOL bit.
1 Both edges triggering.

For GPIO in mode, the EDSEL bit selects if a FLAG can be generated.
0 A FLAG is generated as defined by the EDPOL bit.
1 No FLAG is generated.

For SAOC mode, the EDSEL bit selects the behavior of the output flip-flop at each match.
0 The EDPOL value is transferred to the output flip-flop.
1 The output flip-flop is toggled.</td></tr>
</table>

Table 341. EMIOS_CCR field description (continued)

Field	Description
EDPOL	Edge Polarity For input modes, the EDPOL bit asserts which edge triggers either the internal counter or an input capture or a FLAG. When not shown in the mode of operation description, this bit has no effect. 0 Trigger on a falling edge. 1 Trigger on a rising edge. For output modes, the EDPOL bit is used to select the logic level on the output pin. 0 A match on comparator A clears the output flip-flop, while a match on comparator B sets it. 1 A match on comparator A sets the output flip-flop, while a match on comparator B clears it.
MODE	Mode Selection The MODE bits select the mode of operation of the unified channel, as shown in *Table 342*. Refer to *Table 334* for more information on the different modes. If a reserved value is written to MODE, the results are unpredictable.

Table 342. MODE values

MODE[0:6]	Mode of operation
0000000	General purpose Input/Output mode (input)
0000001	General purpose Input/Output mode (output)
0000010	Single Action Input Capture
0000011	Single Action Output Compare
0000100	Input Pulse Width Measurement
0000101	Input Period Measurement
0000110	Double Action Output compare (with FLAG set on B match)
0000111	Double Action Output compare (with FLAG set on both match)
0001000 through 1001111	Reserved(1)
101000b(2)	Modulus Counter Buffered (Up counter)
101001b(2)	Reserved(1)
10101bb(2)	Modulus Counter Buffered (Up/Down counter)
10110b0	Output Pulse Width and Frequency Modulation Buffered
10110b1 – 10111b1(2)	Reserved(1)
11000b0(2)	Output Pulse Width Modulation Buffered
1100001 through 1111111	Reserved(1)

1. If a reserved value is written to MODE, the results are unpredictable.
2. b = adjust parameters for the mode of operation. Refer to *Section , Channel modes of operation*, for details.

eMIOS200 Channel Status Register (EMIOS_CSR[n])

Figure 334. eMIOS200 Channel Status Register (EMIOS_CSR[*n*])

Address: UC[*n*] base address + 0x0010 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	OVR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	w1c															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	OVFL	0	0	0	0	0	0	0	0	0	0	0	0	UCIN	UCOUT	FLAG
W	w1c															w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 343. EMIOS_CSR[*n*] field description

Field	Description
OVR	Overrun The OVR bit indicates that FLAG generation occurred when the FLAG bit was already set. This bit can be cleared by clearing the FLAG bit or by software writing a ‘1’. 0 Overrun has not occurred. 1 Overrun has occurred.
OVFL	Overflow The OVFL bit indicates that an overflow has occurred in the internal counter. This bit must be cleared by software writing a ‘1’. 0 An overflow has not occurred. 1 An overflow has occurred.
UCIN	Unified Channel Input Pin The UCIN bit reflects the input pin state after being filtered and synchronized.
UCOUT	Unified Channel Output The UCOUT bit reflects the output pin state.
FLAG	FLAG The FLAG bit is set when an input capture or a match event in the comparators occurred. This bit must be cleared by software writing a ‘1’. 0 FLAG cleared. 1 FLAG set event has occurred. When the DMA bit is set, the FLAG bit can be cleared by the DMA controller.

eMIOS200 UC Alternate A Register (EMIOS_ALTA[n])

Figure 335. eMIOS200 UC Alternate A Register (EMIOS_ALTA[*n*])

Address: UC[*n*] base address + 0x0014 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	ALTA[0:23]							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	ALTA[0:23]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

The EMIOS_ALTA[*n*] register provides an alternate read-only address to access A2 channel registers in GPIO, PEC, WPTA, and OPWMT modes. If the EMIOS_CADR[*n*] is used with EMIOS_ALTA[*n*], both A1 and A2 registers can be accessed in these modes.

21.5 Functional description

The eMIOS200 provides independently operating channels that can be configured and accessed by a host MCU. The channels are reduced-function versions of Unified Channels. The four time bases can be shared by the channels through four counter buses and each channel can generate its own time base.

The eMIOS200 block is reset asynchronously. All registers are cleared on reset.

21.5.1 Unified channel (UC)

Figure 336 shows the eMIOS200 Unified Channel(n) block diagram. Each Unified Channel consists of:

- Counter bus selector, which selects the time base to be used by the channel for all timing functions
- A programmable clock prescaler
- Two double buffered data registers A and B that allow up to two input capture and/or output compare events to occur before software intervention is needed.
- Two comparators (equal only) A and B, which compares the selected counter bus with the value in the data registers
- Internal counter, which can be used as a local time base or to count input events
- Programmable input filter, which ensures that only valid pin transitions are received by channel
- Programmable input edge detector, which detects the rising, falling or either edges
- An output flip-flop, which holds the logic level to be applied to the output pin
- eMIOS200 Status and Control register
- An Output Disable Input selector, which selects the Output Disable Input signal that will be used as output disable

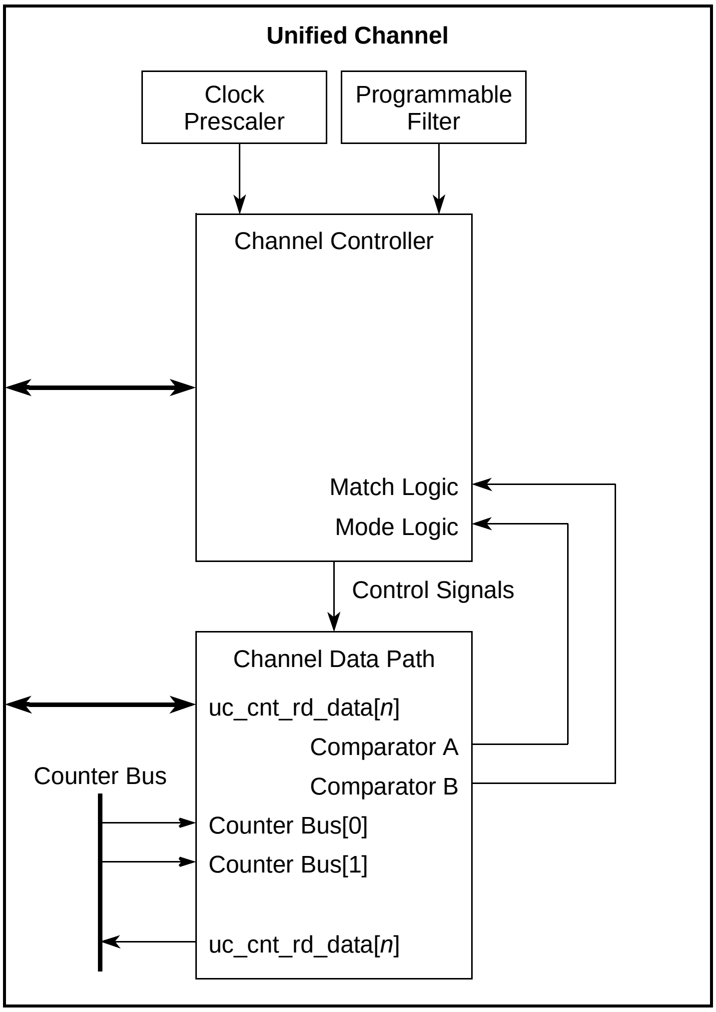


Figure 336. Unified Channel block diagram

Figure 337 shows both the Unified Channel Control and Datapath block diagram. The control block is responsible for the generation of signals to control the multiplexes in the Datapath sub-block. Each mode is implemented by a dedicated logic independent from others modes, thus allowing to optimize the logic by disabling the mode and therefore its associated logic. The unused gates are removed during the synthesis phase. Targeting the logic optimization, a set of registers is shared by the modes thus providing sequential events to be stored.

The Datapath block provides the channel A and B registers, the internal time base and comparators. Multiplexors select the input of comparators and data for the registers inputs, thus configuring the datapath in order to implement the channel modes. The outputs of A and B comparators are connected to the control block.

---

n. The eMIOS200 Unified Channel has a reduced set of functions when compared to legacy Unified Channel implementations.

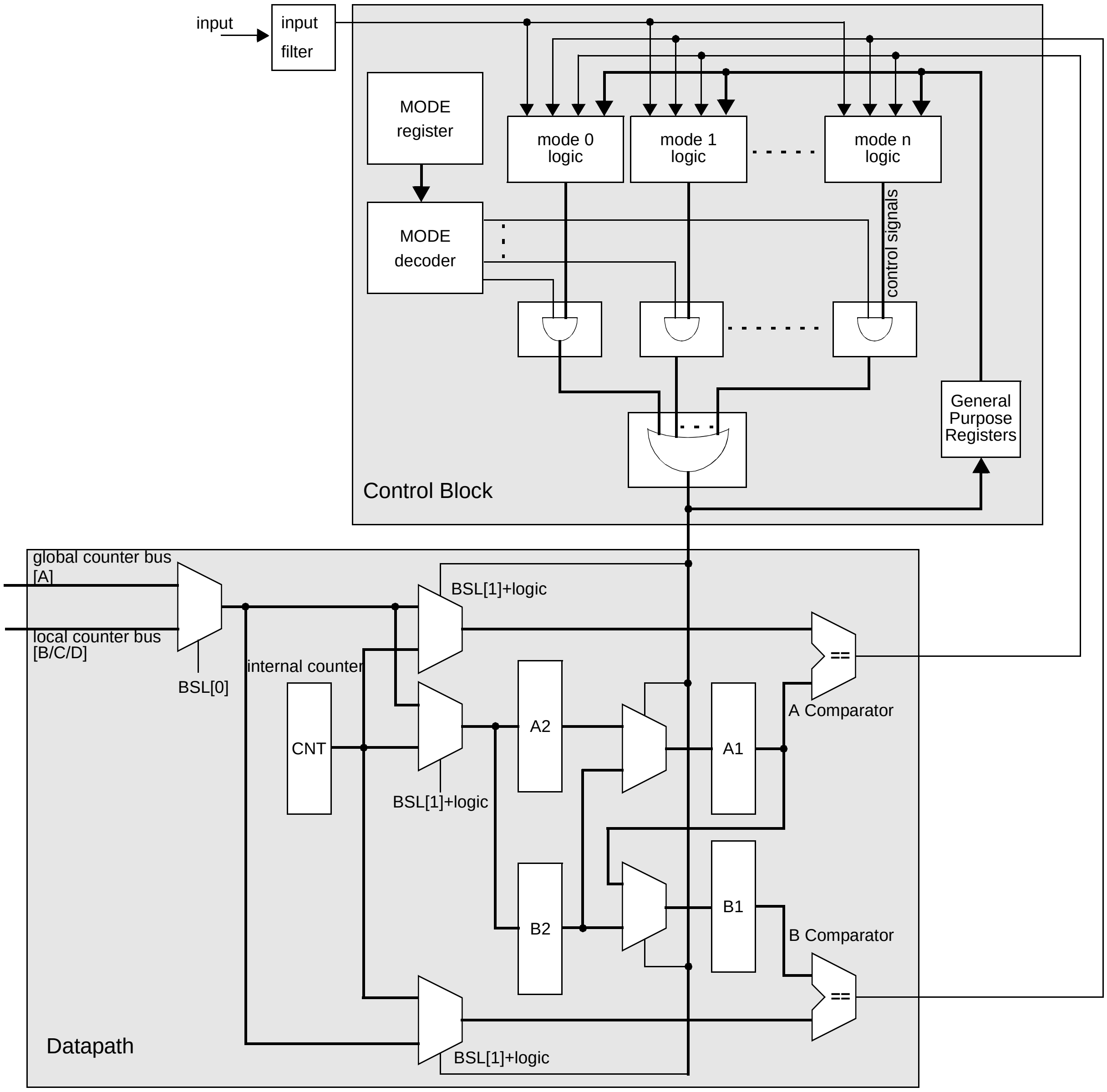


Figure 337. Unified Channel Control and Datapath block diagrams

Channel modes of operation

The mode of operation of *channel n* is determined by the mode select bits MODE[0:6] in the EMIOS_CCR[n] (see *Table 342* for details).

When entering an output mode (except for GPIO mode), the output flip-flop is set to disabled state according to ODIS bit in the EMIOS_CCR[n].

As the internal counter EMIOS_CCNTR[n] continues to run in all modes (except for GPIO mode), it is possible to use it as a time base if the resource is not used in the current mode.

To provide smooth waveform generation while allowing A and B registers to be asynchronously updated during UC operation, the double-buffered modes (MCB, OPWFMB and OPWMB) are provided. In these modes A and B registers are double buffered.

General purpose input/output mode (GPIO) mode

In GPIO mode, all input capture and output compare functions are disabled, the internal counter (EMIOS_CCNTR[n]) is cleared and disabled. All control bits remain accessible. In order to prepare the channel for a new operation mode, writing to registers EMIOS_CADR[n] or EMIOS_CBDR[n] stores the same value in registers A1/A2 or B1/B2, respectively. Writing to register EMIOS_ALTA[n] stores a value only in register A2.

The EMIOS_CCR[n]'s MODE[6] bit selects between input (MODE[6] = 0) and output (MODE[6] = 1) modes.

Note: It is required that when changing MODE[0:6], the application software goes to GPIO mode first in order to reset the channel's internal functions properly. Failure to do this could lead to invalid and unexpected output compare or input capture results or the FLAGs being set incorrectly.

In GPIO input mode (MODE[0:6] = 0000000), FLAG generation is determined according to the EMIOS_CCR[n]'s EDPOL and EDSEL bits and the input pin status can be determined by reading the EMIOS_CSR[n]'s UCIN bit.

In GPIO output mode (MODE[0:6] = 0000001), the channel is used as a single output port pin and the value of the EMIOS_CCR[n]'s EDPOL bit is permanently transferred to the output flip-flop.

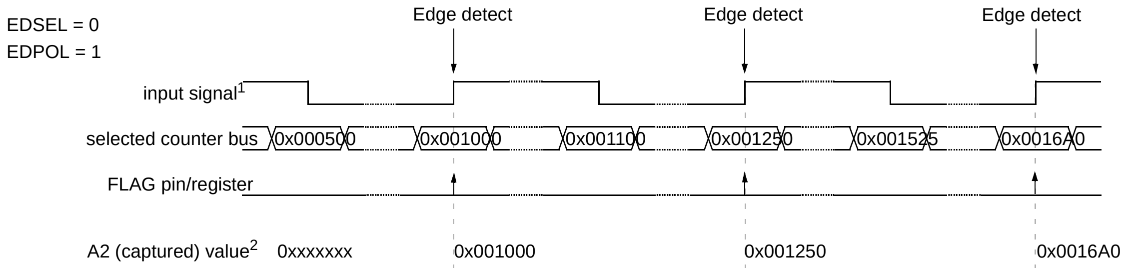
When changing the EMIOS_CCR[n]'s MODE bits, the application software must go to GPIO mode first to reset the channel's internal functions properly. Failure to do this could lead to invalid and unexpected output compare or input capture results or the FLAGs being set incorrectly.

Single action input capture (SAIC) mode

In SAIC mode (MODE[0:6] = 0000010), when a triggering event occurs on the input pin, the value on the selected time base is captured into register A2. The FLAG bit is set along with the capture event to indicate that an input capture has occurred. EMIOS_CADR[n] returns the value of register A2. As soon as the SAIC mode is entered exiting from GPIO mode the channel is ready to capture events. The events are captured as soon as they occur thus reading register A always returns the value of the latest captured event. Subsequent captures are enabled with no need of further reads from EMIOS_CADR[n]. The FLAG is set at any time a new event is captured.

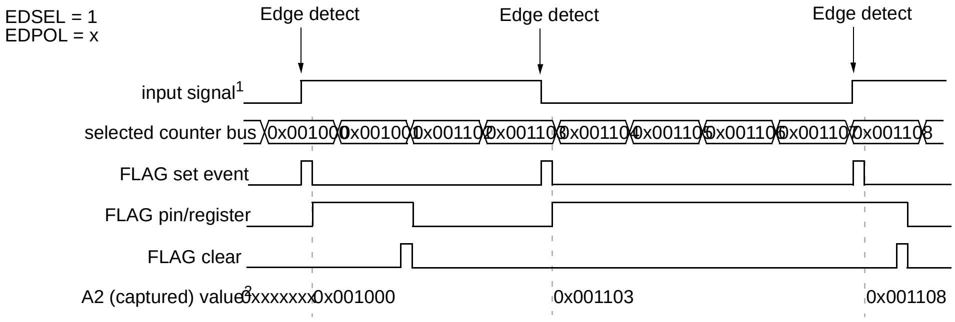
The input capture is triggered by a rising, falling or either edges in the input pin, as configured by EDPOL and EDSEL bits in EMIOS_CCR[n].

Figure 338 and *Figure 339* show how the Unified Channel can be used for input capture.



Notes: 1. After input filter
2. CADR[n] <= A2

Figure 338. Single Action Input Capture with rising edge triggering example



Notes: 1. After input filter
2. CADR[n] <= A2

Figure 339. Single Action Input Capture with both edges triggering example

Single action output compare (SAOC) mode

In SAOC mode (MODE[0:6] = 0000011) a match value is loaded in register A2 and then immediately transferred to register A1 to be compared with the selected time base. When a match occurs, the EDSEL bit selects whether the output flip-flop is toggled or the value in EDPOL is transferred to it. Along with the match the FLAG bit is set to indicate that the output compare match has occurred. Writing to EMIOS_CADR[n] stores the value in register A2 and reading to register EMIOS_CADR[n] returns the value of register A1.

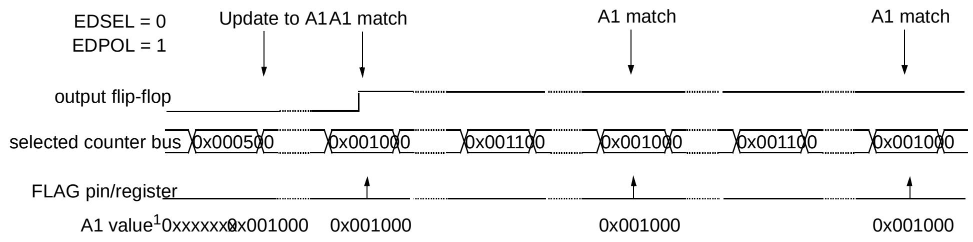
An output compare match can be simulated in software by setting the FORCMA bit in EMIOS_CCR[n]. In this case, the FLAG bit is not set.

When SAOC mode is entered exiting from GPIO mode the output flip-flop is set to the complement of the EDPOL bit in the EMIOS_CCR[n].

The counter bus can be either internal or external and is selected through BSL[0:1] bits.

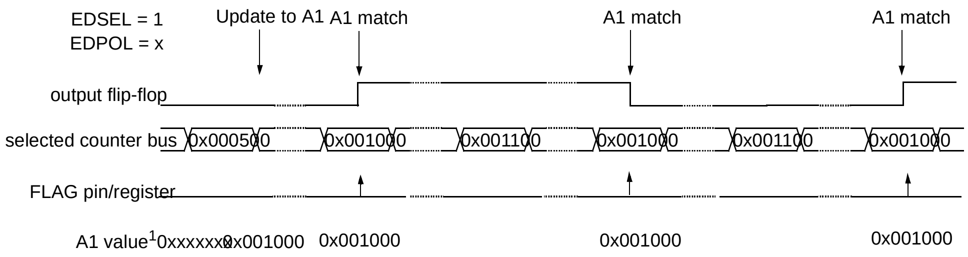
Figure 340 and *Figure 341* show how the channel can be used to perform a single output compare with EDPOL value being transferred to the output flip-flop and toggling the output flip-flop at each match, respectively. Note that once in SAOC mode the matches are enabled thus the desired match value on register A1 must be written before the mode is entered. A1 register can be updated at any time thus modifying the match value which will reflect in the output signal generated by the channel. Subsequent matches are enabled with no need of further writes to EMIOS_CADR[n]. The FLAG is set at the same time a match occurs (see *Figure 342*).

Note: The channel internal counter in SAOC mode is free-running. It starts counting as soon as the SAOC mode is entered.



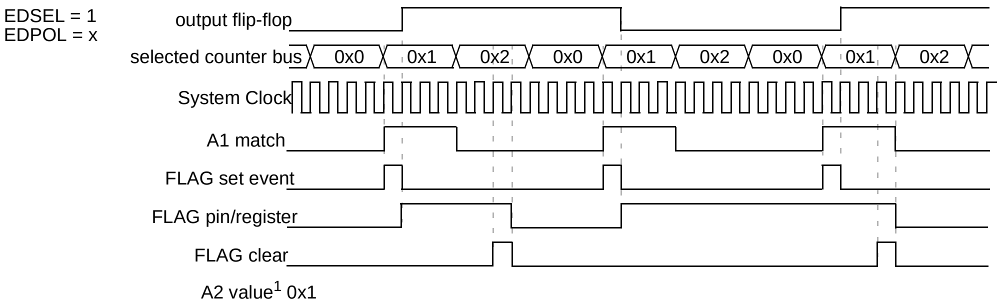
Notes: 1. CADR[n] = A2
A2 = A1 according to OU[n] bit

Figure 340. SAOC example with EDPOL value being transferred to the output flip-flop



Notes: 1. CADR[n] = A2
A2 = A1 according to OU[n] bit

Figure 341. SAOC example toggling the output flip-flop



Note: 1. CADR[n] <= A2

Figure 342. SAOC example with flag behavior

Input pulse width measurement (IPWM) mode

The IPWM mode (MODE[0:6] = 0000100) allows the measurement of the width of a positive or negative pulse by capturing the leading edge on register B1 and the trailing edge on register A2. Successive captures are done on consecutive edges of opposite polarity. The leading edge sensitivity (that is, pulse polarity) is selected by EDPOL bit in the EMIOS_CCR[n]. Registers EMIOS_CADR[n] and EMIOS_CBDR[n] return the values in registerS A2 and B1, respectively.

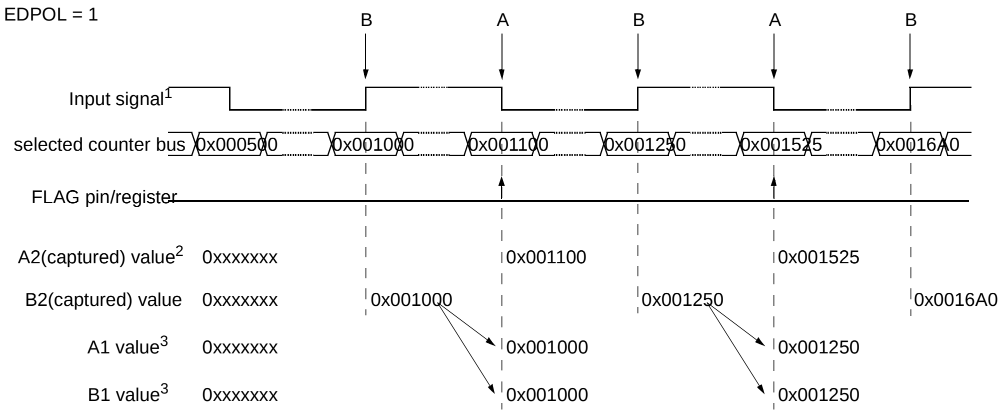
The capture function of register A2 remains disabled until the first leading edge triggers the first input capture on register B2. When this leading edge is detected, the count value of the selected time base is latched into register B2; the FLAG bit is not set. When the trailing edge is detected, the count value of the selected time base is latched into register A2 and, at the same time, the FLAG bit is set and the content of register B2 is transferred to register B1 and to register A1.

If subsequent input capture events occur while the corresponding FLAG bit is set, registers A2, B1 and A1 will be updated with the latest captured values and the FLAG will remain set. Registers EMIOS_CADR[n] and EMIOS_CBDR[n] return the value in registers A2 and B1, respectively.

In order to guarantee coherent access, reading EMIOS_CADR[n] forces B1 be updated with the content of register A1. At the same time transfers between B2 and B1 are disabled until the next read of EMIOS_CBDR[n]. Reading EMIOS_CBDR[n] forces B1 be updated with A1 register content and re-enables transfers from B2 to B1, to take effect at the next trailing edge capture. Transfers from B2 to A1 are not blocked at any time.

The input pulse width is calculated by subtracting the value in B1 from A2.

Figure 343 shows how the channel can be used for input pulse width measurement.



Notes: 1. After input filter
2. CADR[n] = A2
3. CBDR[n] = B1

Figure 343. Input Pulse Width Measurement example

Figure 344 shows the A1 and B1 updates when EMIOS_CADR[n] and EMIOS_CBDR[n] reads occur. Note that A1 register has always coherent data related to A2 register. Note also that when EMIOS_CADR[n] read is performed B1 register is loaded with A1 register content. This guarantees that the data in register B1 has always the coherent data related to the last EMIOS_CADR[n] read. The B1 register updates remain locked until EMIOS_CBDR[n] read occurs. If EMIOS_CADR[n] read is performed B1 is updated with A1 register content even if B1 update is locked by a previous EMIOS_CADR[n] read operation.

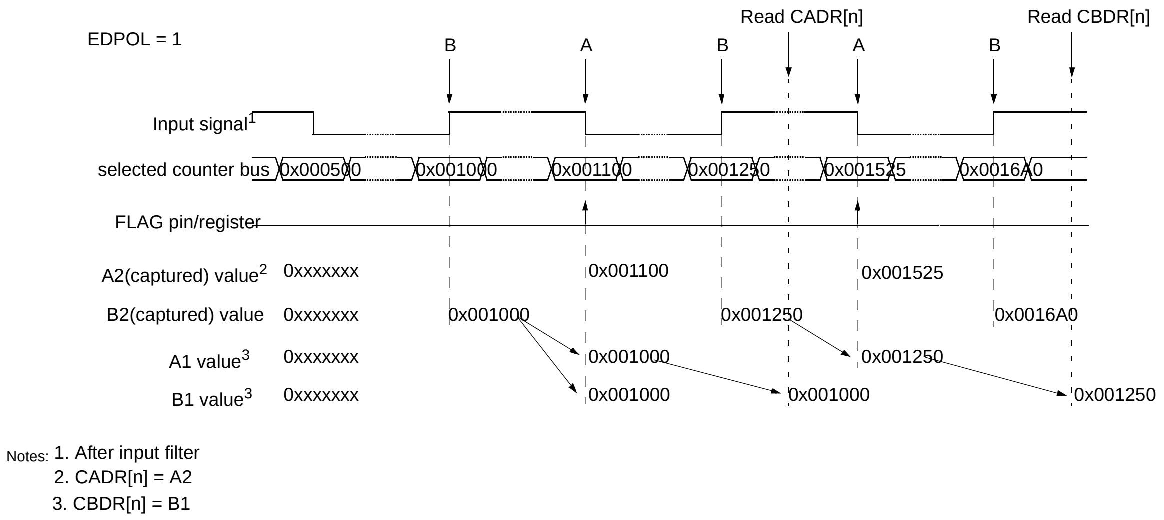


Figure 344. B1 and A1 updates at EMIOS_CADR[n] and EMIOS_CBDR[n] reads

Reading EMIOS_CADR[n] followed by EMIOS_CBDR[n] always provides coherent data. If no coherent data is required for any reason, the sequence of reads should be inverted, therefore EMIOS_CBDR[n] should be read prior to EMIOS_CADR[n]. Note that even in this case B1 register updates will be blocked after EMIOS_CADR[n] read, thus a second EMIOS_CBDR[n] is required in order to release B1 register updates.

Input period measurement (IPM) mode

The IPM mode (MODE[0:6] = 0000101) allows the measurement of the period of an input signal by capturing two consecutive rising edges or two consecutive falling edges. Successive input captures are done on consecutive edges of the same polarity. The edge polarity is defined by the EDPOL bit in the EMIOS_CCR[n].

When the first edge of selected polarity is detected, the selected time base is latched into the registers A2 and B2, and the data previously held in register B2 is transferred to register B1. On this first capture the FLAG line is not set, and the values in registers B1 is meaningless. On the second and subsequent captures, the FLAG line is set and data in register B2 is transferred to register B1.

When the second edge of the same polarity is detected, the counter bus value is latched into registers A2 and B2, the data previously held in register B2 is transferred to data register B1 and to register A1. The FLAG bit is set to indicate the start and end points of a complete period have been captured. This sequence of events is repeated for each subsequent capture. Registers EMIOS_CADR[n] and EMIOS_CBDR[n] return the values in register A2 and B1, respectively.

In order to allow coherent data, reading EMIOS_CADR[n] forces A1 content be transferred to B1 register and disables transfers between B2 and B1. These transfers are disabled until the next read of the EMIOS_CBDR[n]. Reading EMIOS_CBDR[n] forces A1 content to be transferred to B1 and re-enables transfers from B2 to B1, to take effect at the next edge capture.

The input pulse period is calculated by subtracting the value in B1 from A2.

Figure 345 shows how the channel can be used for input period measurement.

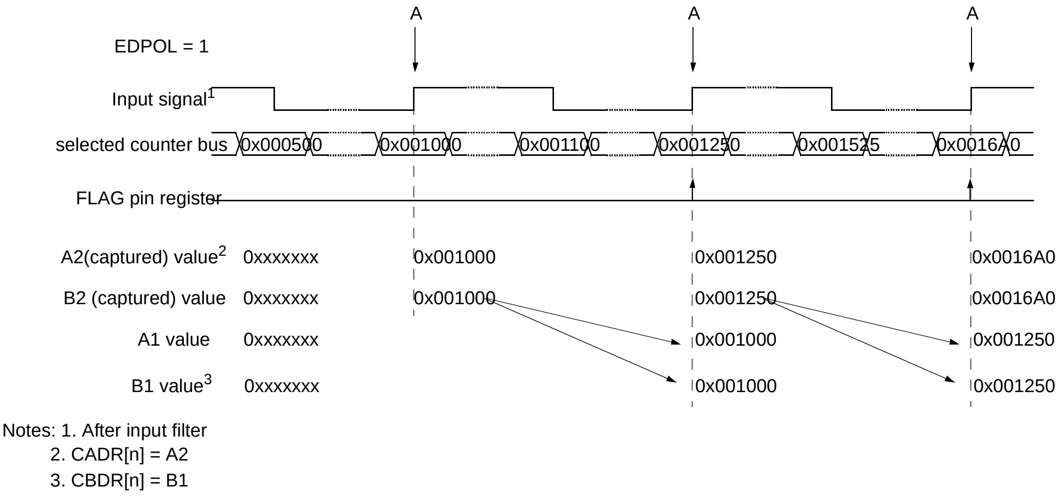


Figure 345. Input Period Measurement example

Figure 346 describes the A1 and B1 register updates when EMIOS_CADR[n] and EMIOS_CBDR[n] read operations are performed. When EMIOS_CADR[n] read occurs the content of A1 is transferred to B1 thus providing coherent data in A2 and B1 registers. Transfers from B2 to B1 are then blocked until EMIOS_CBDR[n] is read. After EMIOS_CBDR[n] is read, register A1 content is transferred to register B1 and the transfers from B2 to B1 are re-enabled to occur at the transfer edges, which is the leading edge in the *Figure 346* example.

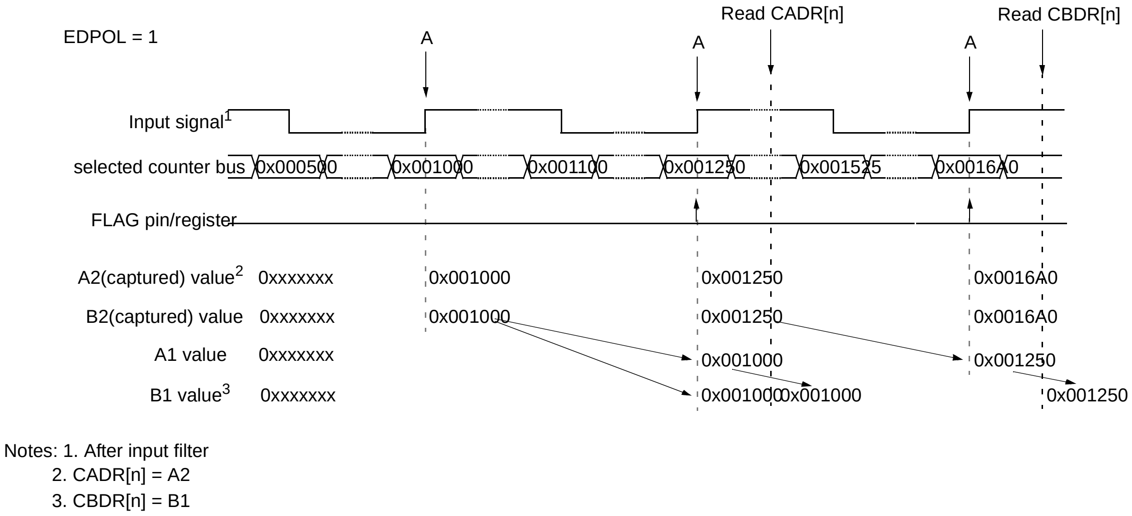


Figure 346. A1 and B1 updates at EMIOS_CADR[n] and EMIOS_CBDR[n] reads

Double action output compare (DAOC) mode

In the DAOC mode the leading and trailing edges of the variable pulse width output are generated by matches occurring on comparators A and B. There is no restriction concerning the order in which A and B matches occur.

When the DAOC mode is entered, exiting from GPIO mode both comparators are disabled and the output flip-flop is set to the complement of the EDPOL bit in the EMIOS_CCR[n].

Data written to A2 and B2 are transferred to A1 and B1, respectively, on the next system clock cycle if bit OU[n] of the EMIOS_OUDR is cleared (see *Figure 349*). The transfer is blocked if OU[n] bit is set. Comparator A is enabled only after the transfer to A1 register occurs and is disabled on the next A match. Comparator B is enabled only after the transfer to B1 register occurs and is disabled on the next B match. Comparators A and B are enabled and disabled independently.

The output flip-flop is set to the value of EDPOL when a match occurs on comparator A and to the complement of EDPOL when a match occurs on comparator B.

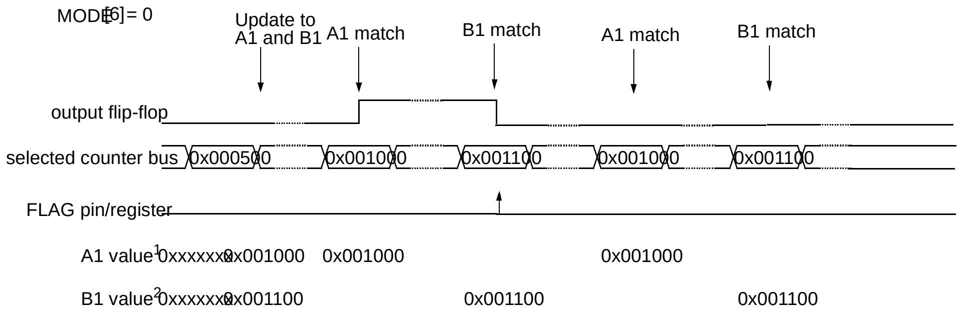
MODE[6] controls if the FLAG is set on both matches (MODE[0:6] = 0000111) or just on the B match (MODE[0:6] = 0000110). FLAG bit assertion depends on comparator enabling.

If subsequent enabled output compares occur on registers A1 and B1, pulses will continue to be generated, regardless of the state of the FLAG bit.

At any time, the FORCMA and FORCMB bits allow the software to force the output flip-flop to the level corresponding to a comparison event in comparator A or B, respectively. Note that the FLAG bit is not affected by these forced operations.

Note: If both registers (A1 and B1) are loaded with the same value, the B match prevails concerning the output pin state (output flip-flop is set to the complement of EDPOL), the FLAG bit is set and both comparators are disabled.

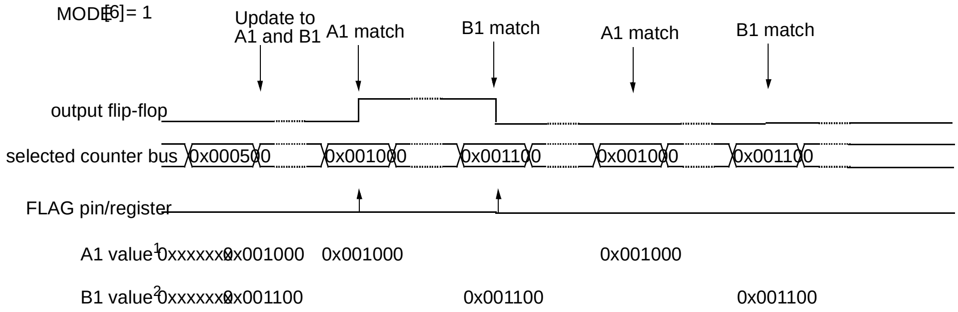
Figure 347 and *Figure 348* show how the channel can be used to generate a single output pulse with FLAG bit being set on the second match or on both matches, respectively.



Notes: 1. CADR[n] = A1 (when reading)
2. CBDR[n] = B1 (when reading)

A2 = A1according to OU[n] bit
B2 = B1according to OU[n] bit

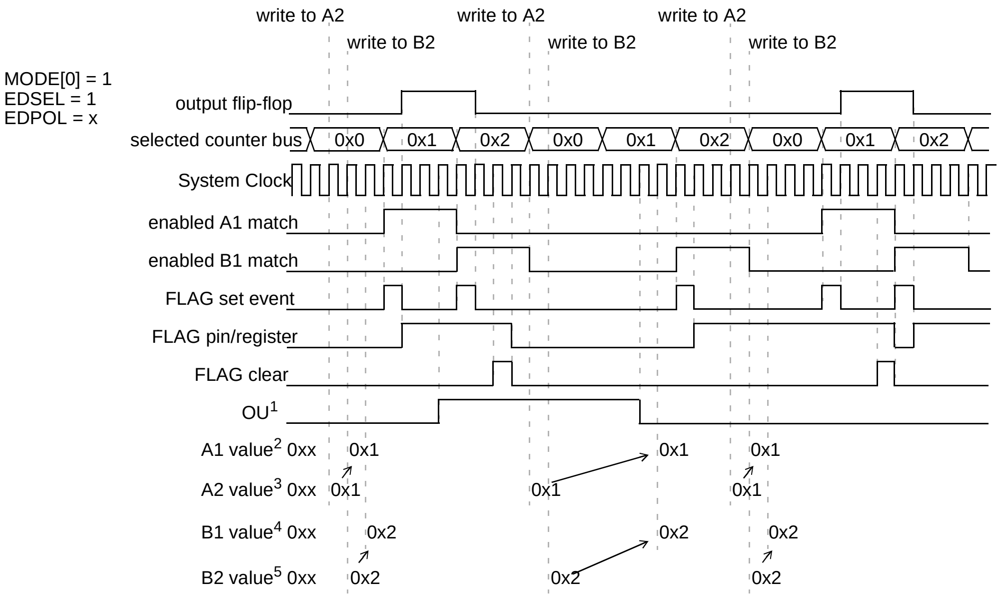
Figure 347. Double Action Output Compare with FLAG set on the second match



Notes:1. CADR[n] = A1 (when reading)
2. CBDR[n] = B1 (when reading)

A2 = A1according to OU[n] bit
B2 = B1according to OU[n] bit

Figure 348. Double Action Output Compare with FLAG set on both matches



Note: 1. OU[n] bit of the OUDR
2. CADR[n] = A1 (when reading)
3. CADR[n] = A2 (when writing)
4. CBDR[n] = B1 (when reading)
5. CBDR[n] = B2 (when writing)

Figure 349. DAOC with transfer disabling example

Modulus counter buffered (MCB) mode

The MCB mode provides a time base which can be shared with other channels through the internal counter buses. Register A1 is double buffered thus allowing smooth transitions between cycles when changing A2 register value on the fly. A1 register is updated at the cycle boundary, which is defined as when the internal counter transitions to 0x1.

The internal counter values operates within a range from 0x1 up to register A1 value. If when entering MCB mode exiting from GPIO mode the internal counter value is not within that range then the A match will not occur causing the channel internal counter to wrap at the maximum counter value which is 0xFF_FFFF for a 24-bit counter. After the counter wrap occurs it returns to 0x1 and resume normal MCB mode operation. Thus in order to avoid the counter wrap condition make sure its value is within the 0x1 to A1 register value range when the MCB mode is entered.

MODE[6] bit selects internal clock source if cleared or external if set. When external clock is selected the input channel pin is used as the channel clock source. The active edge of this clock is defined by EDPOL and EDSEL bits in the EMIOS_CCR[n].

When entering MCB mode, if the up counter is selected by MODE[4] = 0 (MODE[0:6] = 101000b), the internal counter starts counting from its current value to up direction until A1 match occurs. The internal counter is set to 0x1 when its value matches A1 value and a clock tick occurs (either prescaled clock or input pin event).

If the up/down counter is selected by setting MODE[4] = 1, the counter changes direction at A1 match and counts down until it reaches the value 0x1. After it has reached 0x1 it is set to count in up direction again. The B1 register is used to generate a match in order to set the internal counter in up-count direction if up/down mode is selected. Register B1 cannot be changed while this mode is selected.

Note that differently from the MC mode, the MCB mode counts between 0x1 and the A1 register value. Only values greater than 0x1 must be written at A1 register. Loading values other than those leads to unpredictable results. The counter cycle period is equal to A1 value in up counter mode. If in up/down counter mode, the period is defined by the expression: (2*A1)-2.

Figure 350 describes the counter cycle for several A1 values. Register A1 is loaded with the A2 register value at the cycle boundary. Thus any value written to the A2 register within cycle **n** will be updated to A1 at the next cycle boundary and therefore will be used on cycle **n+1**. The cycle boundary between cycle **n** and cycle **n+1** is defined as when the internal counter transitions from A1 value in cycle **n** to 0x1 in cycle **n+1**. Note that the FLAG is generated at the cycle boundary and has a synchronous operation, meaning that it is asserted one system clock cycle after the FLAG set event.

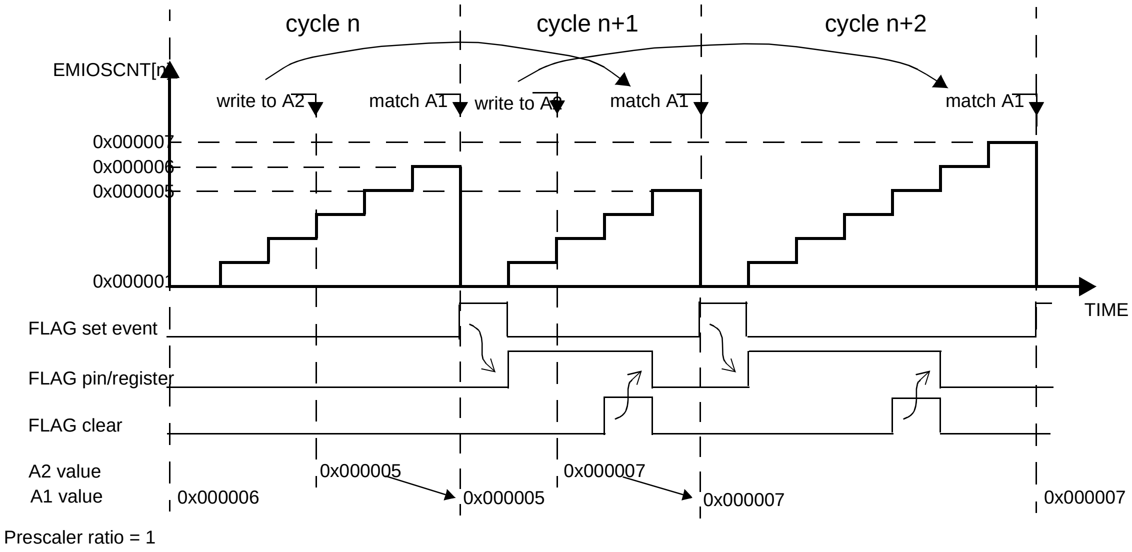


Figure 350. Modulus Counter Buffered (MCB) Up Count mode

Figure 351 describes the MCB in up/down counter mode (MODE[0:6] = 10101bb). The A1 register is updated at the cycle boundary. If A2 is written in cycle **n**, this new value will be used in cycle **n+1** for A1 match. Flags are generated only at A1 match start if MODE[5] is 0. If MODE[5] is set to 1 flags are also generated at the cycle boundary.

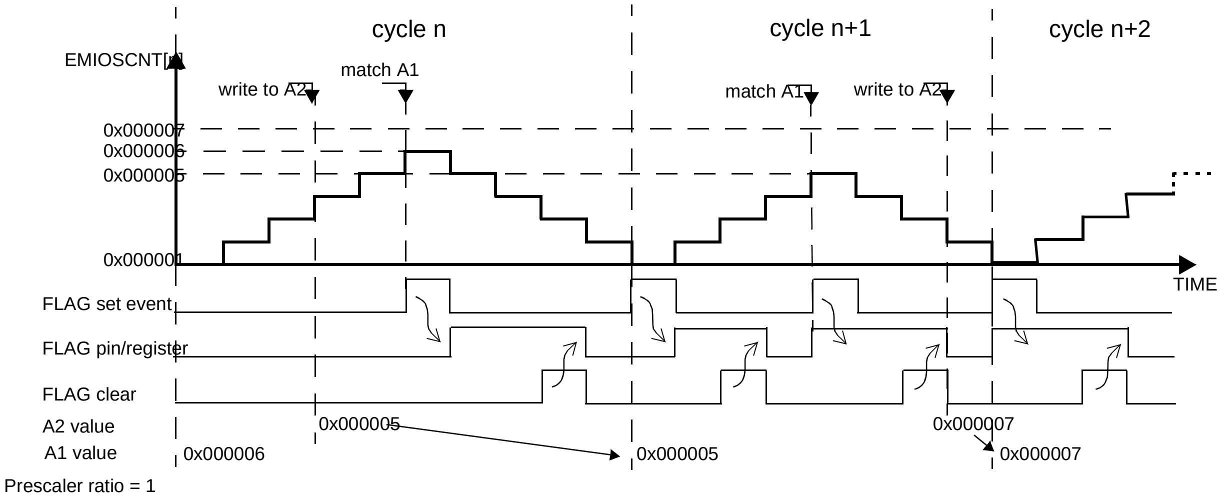


Figure 351. Modulus Counter Buffered (MCB) Up/Down Mode

Figure 352 describes in more detail the A1 register update process in up counter mode. The A1 load signal is generated at the last system clock period of a counter cycle. Thus, A1 is updated with A2 value at the same time that the counter (EMIOS_CCNTR[n]) is loaded with 0x1. The load signal pulse has the duration of one system clock period. If A2 is written within cycle **n** its value is available at A1 at the first clock of cycle **n+1** and the new value is used for match at cycle **n+1**. The update disable bits OU[n] of EMIOS_OUDR can be used to control the update of this register, thus allowing to delay the A1 register update for synchronization purposes.

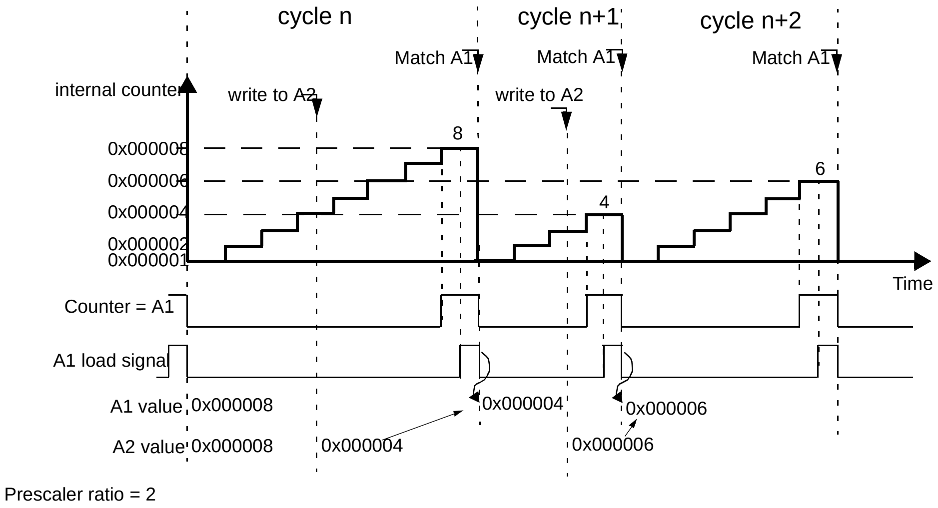


Figure 352. MCB Mode A1 Register Update in Up Counter Mode

Figure 353 describes the A1 register update in up/down counter mode. Note that A2 can be written at any time within cycle **n** in order to be used in cycle **n+1**. Thus A1 receives this new value at the next cycle boundary. Note that the update disable bits OU[n] of EMIOS_OUDR can be used to disable the update of register A1.

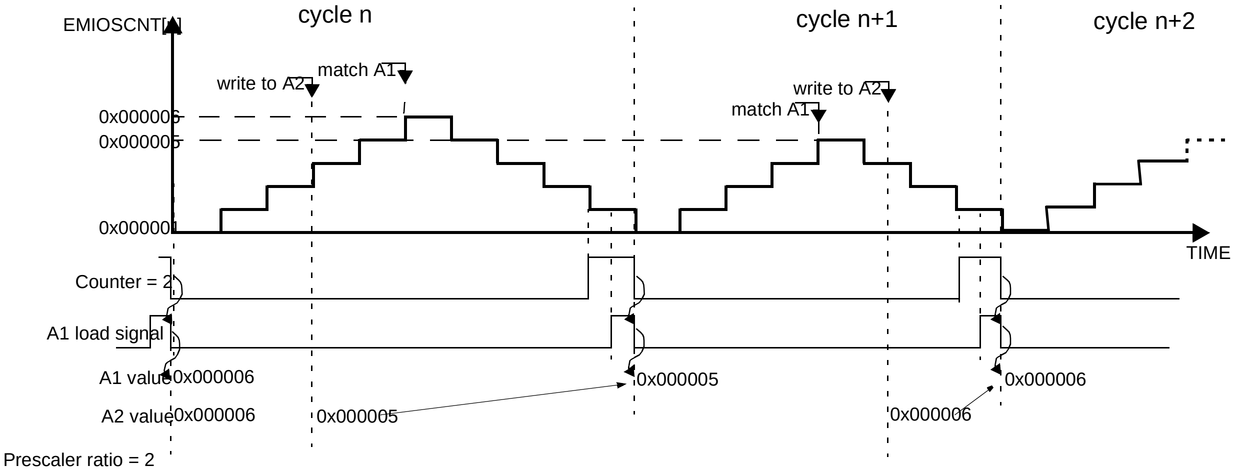


Figure 353. MCB Mode A1 Register Update in Up/Down Counter Mode

Output pulse width and frequency modulation buffered (OPWFMB) mode

This mode (MODE[0:6] = 10110b0) provides waveforms with variable duty cycle and frequency. The internal channel counter is automatically selected as the time base when this mode is selected. The A1 register indicates the duty cycle and B1 register the frequency. Both A1 and B1 registers are double buffered to allow smooth signal generation when changing the registers values on the fly. 0% and 100% duty cycles are supported.

At OPWFMB mode entry the output flip-flop is set to the value of the EDPOL bit in the EMIOS_CCR[n].

In order to provide smooth and consistent channel operation this mode differs substantially from the OPWFM mode. The main differences reside in the A1 and B1 registers update, on the delay from the A1 match to the output pin transition and on the range of the internal counter values which starts from 0x1 up to B1 register value.

If when entering OPWFMB mode exiting from GPIO mode the internal counter value is not within that range then the B match will not occur causing the channel internal counter to wrap at the maximum counter value which is 0xFF_FFFF for a 24-bit counter. After the counter wrap occurs it returns to 0x1 and resume normal OPWFMB mode operation. Thus in order to avoid the counter wrap condition make sure its value is within the 0x1 to B1 register value range when the OPWFMB mode is entered.

When a match on comparator A occurs the output register is set to the value of EDPOL. When a match on comparator B occurs the output register is set to the complement of EDPOL. B1 match also causes the internal counter to transition to 0x1, thus restarting the counter cycle.

Only values greater than 0x1 are allowed to be written to B1 register. Loading values other than those leads to unpredictable results.

Figure 354 describes the operation of the OPWFMB mode regarding output pin transitions and A1/B1 registers match events. Note that the output pin transition occurs when the A1 or B1 match signal is deasserted which is indicated by the A1 match negedge detection signal. If register A1 is set to 0x4 the output pin transitions four counter periods after the cycle had started, plus one system clock cycle. Note that in the example shown in *Figure 354* the internal counter prescaler has a ratio of two.

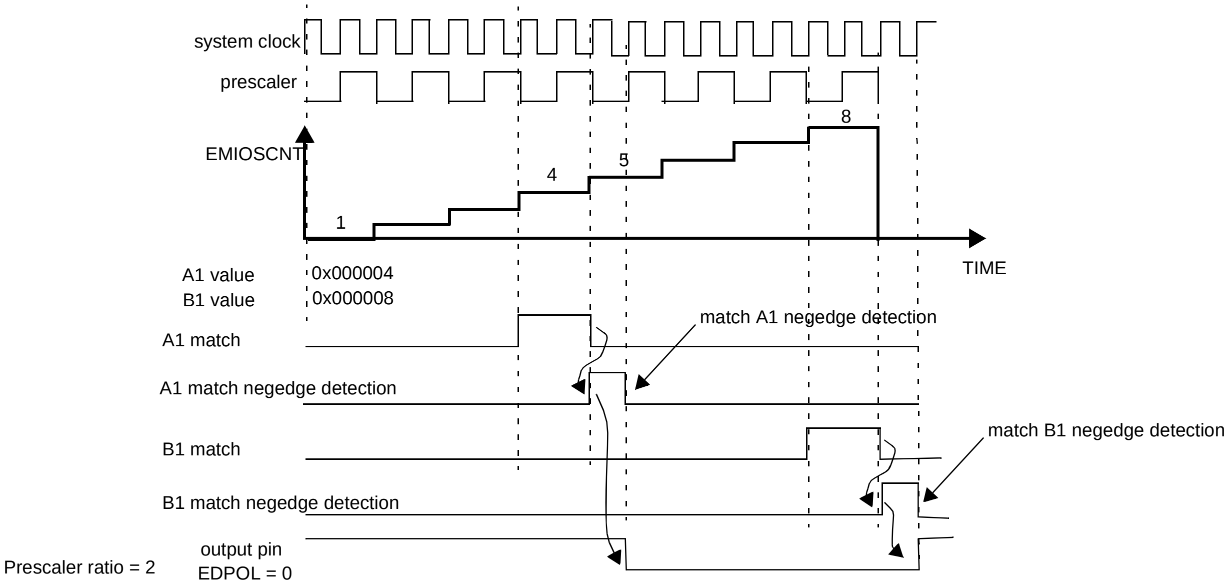


Figure 354. OPWFMB A1 and B1 match to Output Register Delay

Figure 355 describes the generated output signal if A1 is set to 0x0. Since the counter does not reach zero in this mode, the channel internal logic infers a match as if A1 = 0x1 with the difference that in this case, the posedge of the match signal is used to trigger the output pin transition instead of the negedge used when A1 = 0x1. Note that A1 posedge match signal from cycle **n+1** occurs at the same time as B1 negedge match signal from cycle **n**. This allows using the A1 posedge match to mask the B1 negedge match when they occur at the same time. The result is that no transition occurs on the output flip-flop and a 0% duty cycle is generated.

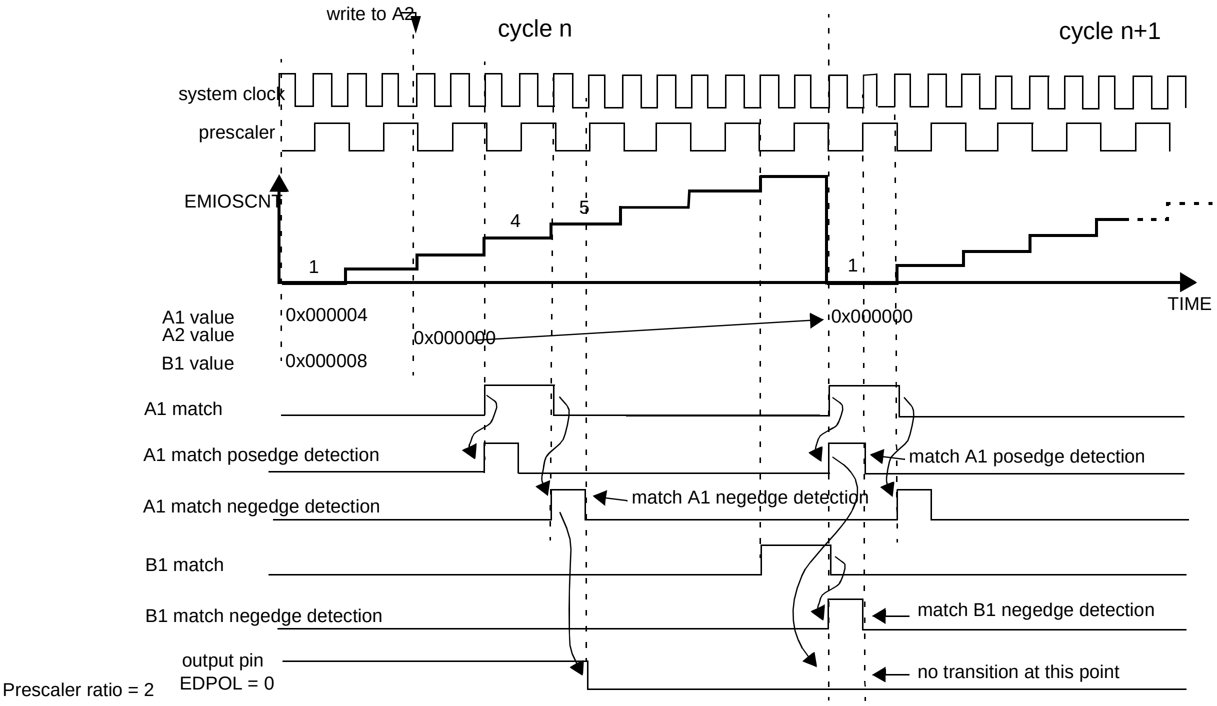


Figure 355. OPWFMB Mode with A1 = 0 (0% duty cycle)

Figure 356 describes the timing for the A1 and B1 registers load. The A1 and B1 load use the same signal which is generated at the last system clock period of a counter cycle. Thus, A1 and B1 are updated respectively with A2 and B2 values at the same time that the counter (EMIOS_CCNTR[n]) is loaded with 0x1. This event is defined as the cycle boundary. The load signal pulse has the duration of one system clock period. If A2 and B2 are written within cycle **n** their values are available at A1 and B1, respectively, at the first clock of cycle **n+1** and the new values are used for matches at cycle **n+1**. The update disable bits OU[n] of EMIOS_OUDR can be used to control the update of these registers, thus allowing to delay the A1 and B1 registers update for synchronization purposes.

In *Figure 356* it is assumed that both the channel and global prescalers are set to 0x1 (each divide ratio is two), meaning that the channel internal counter transitions at every four system clock cycles. FLAGs can be generated only on B1 matches when MODE[5] is cleared, or on both A1 and B1 matches when MODE[5] is set. Since B1 flag occurs at the cycle boundary, this flag can be used to indicate that A2 or B2 data written on cycle **n** were loaded to A1 or B1, respectively, thus generating matches in cycle **n+1**. Note that the FLAG has a synchronous operation, meaning that it is asserted one system clock cycle after the FLAG set event.

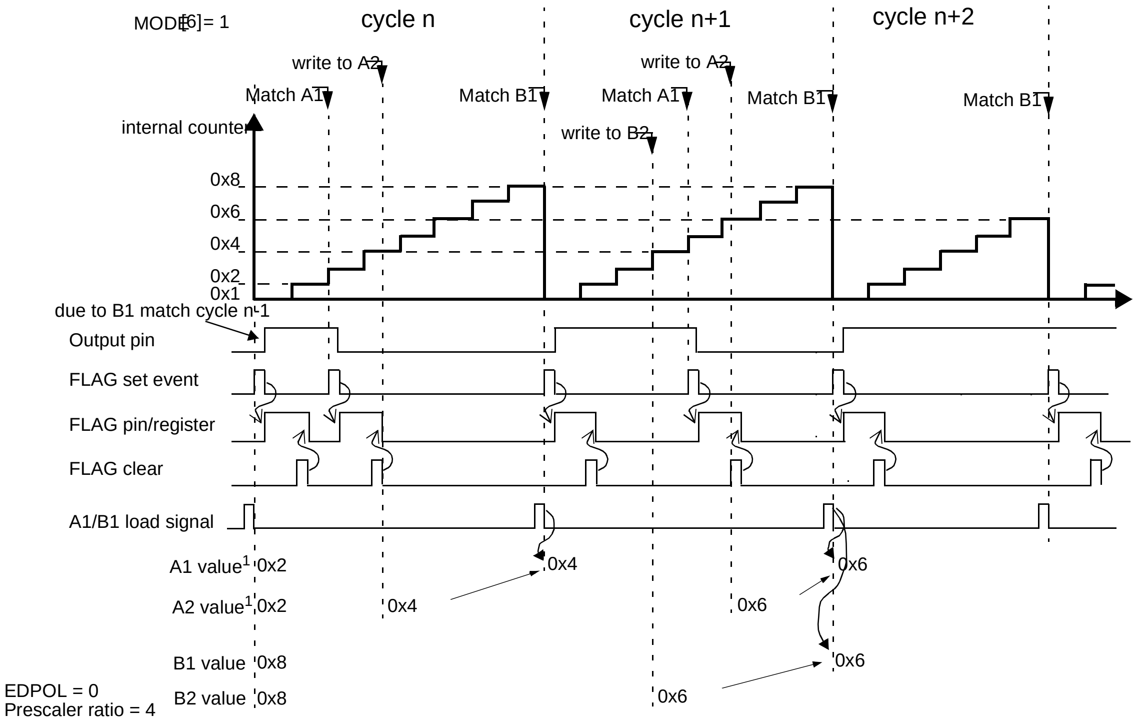


Figure 356. OPWFMB A1 and B1 registers update and flags

Figure 357 describes the operation of the Output Disable feature in OPWFMB mode. Differently from the OPWFM mode, the output disable forces the channel output flip-flop to EDPOL bit value. This functionality targets applications that use active high signals and a high to low transition at A1 match. In this case EDPOL should be set to 0. Note that both the channel and global prescalers are set to 0x0 (each divide ratio is one), meaning that the channel internal counter transitions at every system clock cycle.

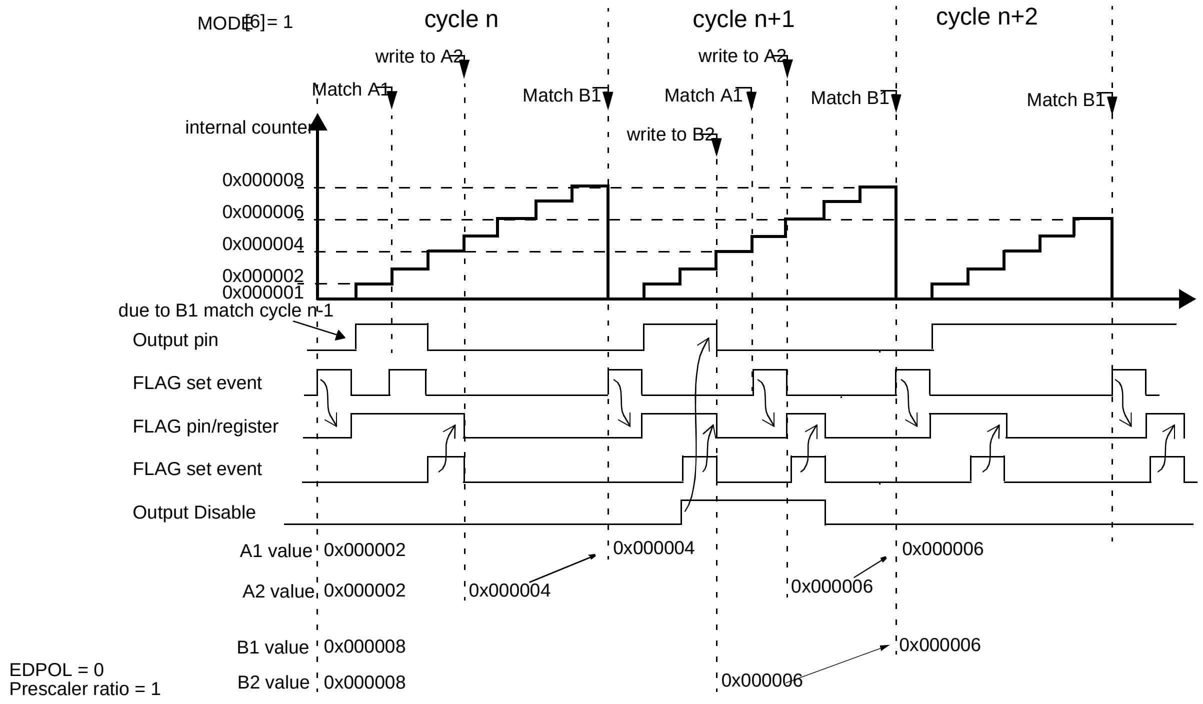


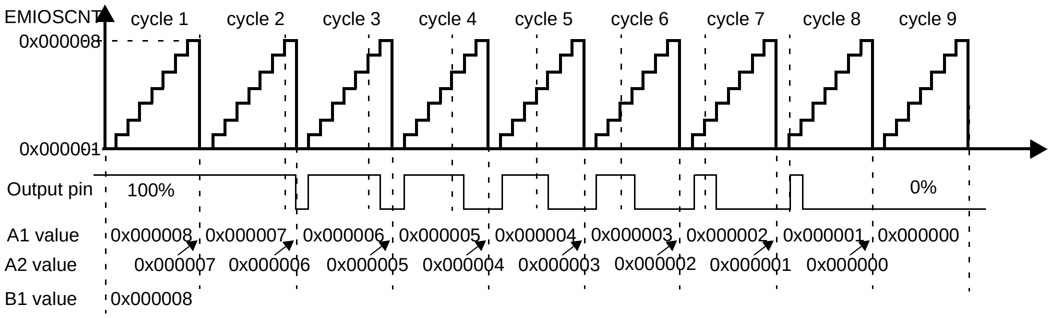
Figure 357. OPWFMB mode with active output disable

Note that the output disable has a synchronous operation, meaning that the assertion of the Output Disable input pin causes the channel output flip-flop to transition to EDPOL at the next system clock cycle. If the Output Disable input is deasserted the output pin transition at the following A1 or B1 match.

In *Figure 357* it is assumed that the Output Disable input is enabled and selected for the Channel. Please, refer to *Section , eMIOS200 Channel Control Register (EMIOS_CCR[n])*, for a detailed description about the ODIS and ODISSL bits, respectively enable and selection of the Output Disable inputs.

The FORCMA and FORCMB bits allow the software to force the output flip-flop to the level corresponding to a match on comparators A or B respectively. Similarly to a B1 match FORCMB sets the internal counter to 0x1. The FLAG bit is not set by the FORCMA or FORCMB bits being asserted.

Figure 358 describes the generation of 100% and 0% duty cycle signals. It is assumed EDPOL = 0 and the resultant prescaler value is 1. Initially A1 = 0x8 and B1 = 0x8. In this case, B1 match has precedence over A1 match, thus the output flip-flop is set to the complement of EDPOL bit. This cycle corresponds to a 100% duty cycle signal. The same output signal can be generated for any A1 value greater or equal to B1.



EDPOL = 0
Prescaler ratio = 1

Figure 358. OPWFMB mode from 100% to 0% duty cycle

A 0% duty cycle signal is generated if A1 = 0x0 as shown in *Figure 358* cycle 9. In this case B1 = 0x8 match from cycle 8 occurs at the same time as the A1 = 0x0 match from cycle 9. Please, refer to *Figure 355* for a description of the A1 and B1 match generation. In this case A1 match has precedence over B1 match and the output signal transitions to EDPOL.

Output pulse width modulation buffered (OPWMB) mode

OPWMB mode (MODE[0:6] = 11000b0) is used to generate pulses with programmable leading and trailing edge placement. An external counter driven in MCB Up mode must be selected from one of the counter buses. A1 register value defines the first edge and B1 the second edge. The output signal polarity is defined by the EDPOL bit. If EDPOL is zero, a negative edge occurs when A1 matches the selected counter bus and a positive edge occurs when B1 matches the selected counter bus.

The A1 and B1 registers are double buffered and updated from A2 and B2, respectively, at the cycle boundary. The load operation is similar to the OPWFMB mode. Please refer to *Figure 356* for more information about A1 and B1 registers update.

FLAG can be generated at B1 matches, when MODE[5] is cleared, or in both A1 and B1 matches, when MODE[5] is set. If subsequent matches occur on comparators A and B, the PWM pulses continue to be generated, regardless of the state of the FLAG bit.

FORCMA and FORCMB bits allow the software to force the output flip-flop to the level corresponding to a match on A1 or B1 respectively. FLAG bit is not set by the FORCMA and FORCMB operations.

At OPWMB mode entry the output flip-flop is set to the value of the EDPOL bit in the EMIOS_CCR[n].

Some rules applicable to the OPWMB mode are listed as follows:

- B1 matches have precedence over A1 matches if they occur at the same time within the same counter cycle
- A1 = 0 match from cycle **n** has precedence over B1 match from cycle **n-1**
- A1 matches are masked out if they occur after B1 match within the same cycle
- Any value written to A2 or B2 on cycle **n** is loaded to A1 and B1 registers at the following cycle boundary (assuming OU[n] bit of EMIOS_OUDR is not asserted). Thus the new values will be used for A1 and B1 matches in cycle **n+1**.

Figure 359 describes the operation of the OPWMB mode regarding A1 and B1 matches and the transition of the channel output pin. In this example EDPOL is set to zero.

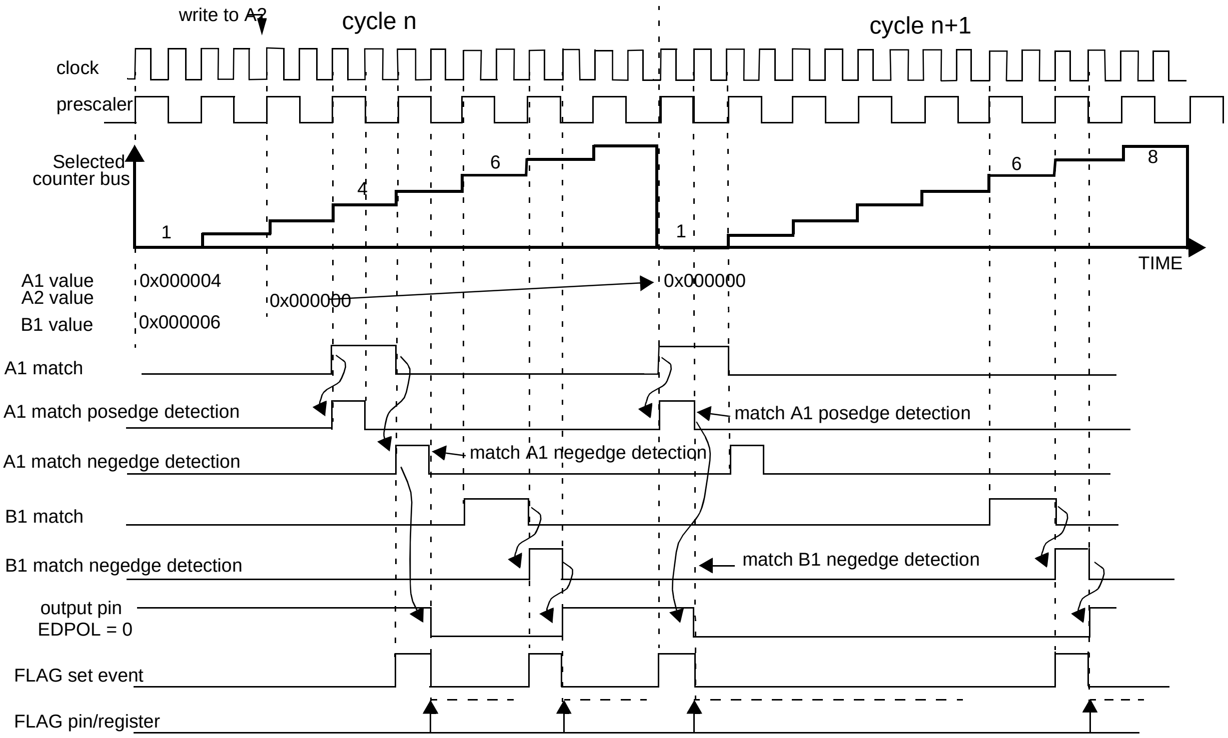


Figure 359. OPWMB mode matches and flags

Note that the output pin transitions are based on the negedges of the A1 and B1 match signals. *Figure 359* shows in cycle **n+1** the value of A1 register being set to zero. In this case the match posedge is used instead of the negedge to transition the output flip-flop.

Figure 360 describes the channel operation for 0% duty cycle. Note that the A1 match posedge signal occurs at the same time as the B1 = 0x8 negedge signal. In this case A1 match has precedence over B1 match, causing the output pin to remain at EDPOL bit value, thus generating a 0% duty cycle signal.

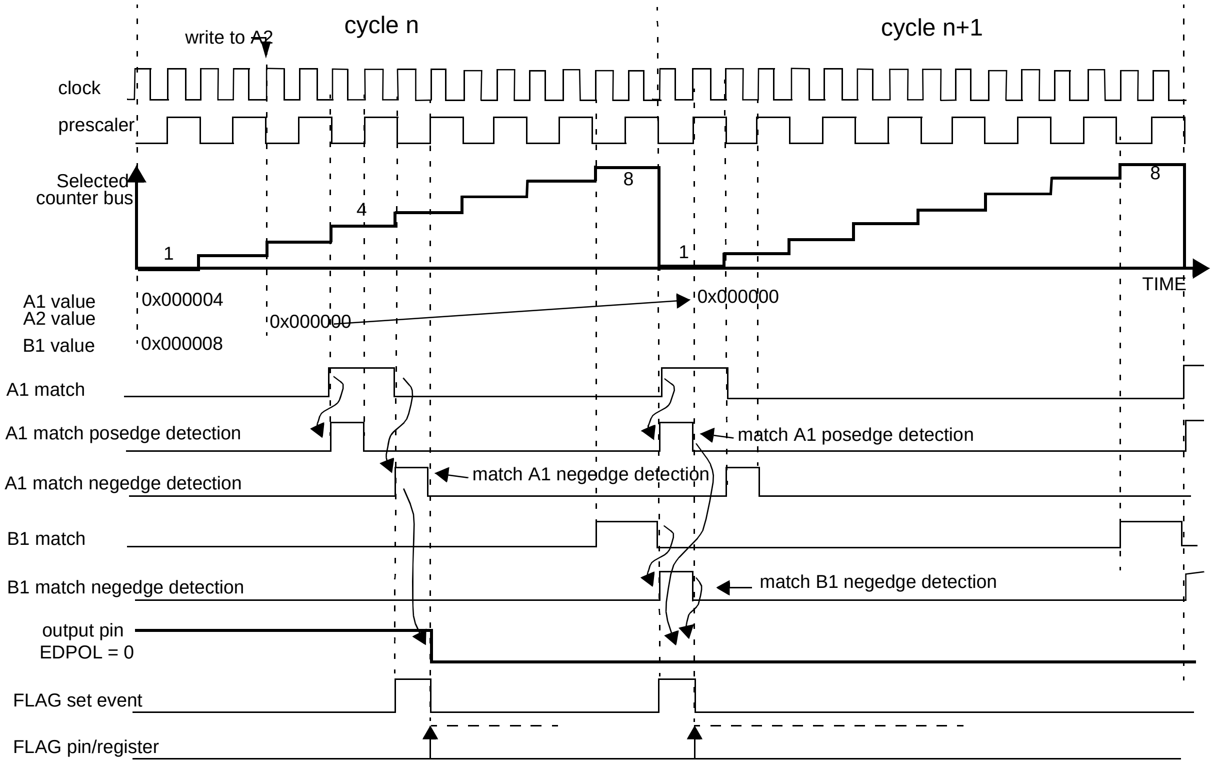


Figure 360. OPWMB mode with 0% duty cycle

Figure 361 describes the operation of the OPWMB mode with the Output Disable signal being asserted. The Output Disable forces a transition in the output pin to the EDPOL bit value. After deasserted, the output disable allows the output pin to transition at the following A1 or B1 match. Note that the Output Disable does not modify the Flag bit behavior. Note that there is one system clock delay between the assertion of the output disable signal and the transition of the output pin to EDPOL.

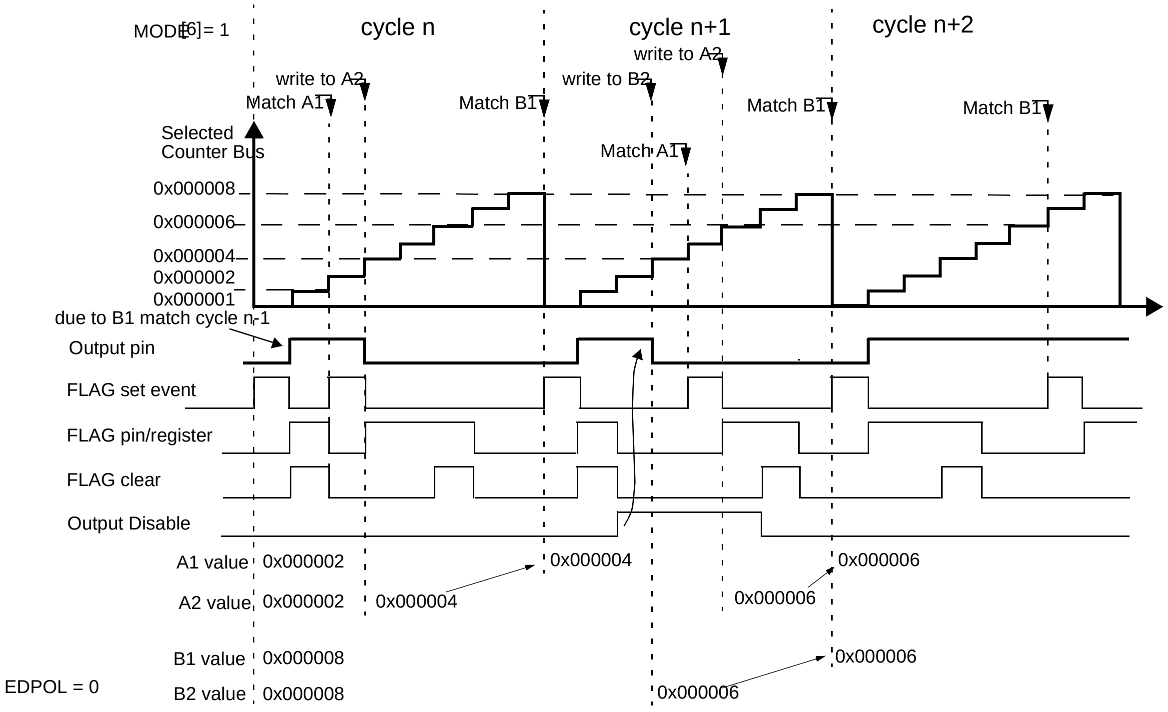


Figure 361. OPWMB mode with active output disable

Figure 362 shows a waveform changing from 100% to 0% duty cycle. EDPOL in this case is zero. In this example B1 is programmed to the same value as the period of the external selected time base.

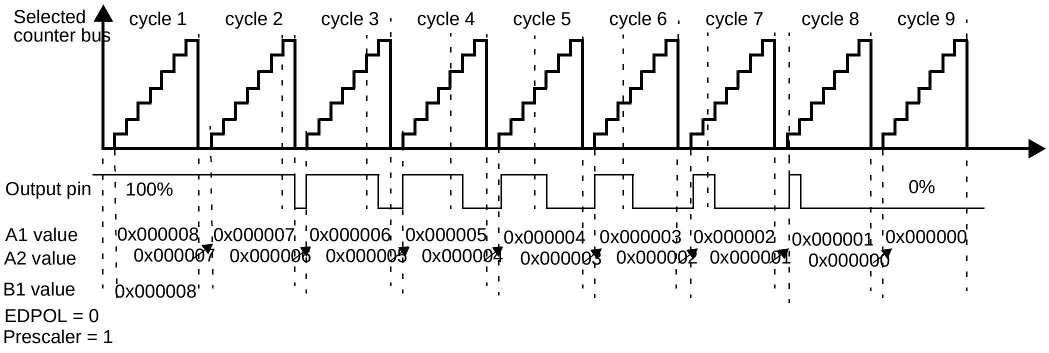


Figure 362. OPWMB mode from 100% to 0% duty cycle

In *Figure 362* if B1 is set to a value lower than 0x8 it is not possible to achieve 0% duty cycle by only changing A1 register value. Since B1 matches have precedence over A1 matches the output pin transitions to the opposite of EDPOL bit at B1 match. Note also that if B1 is set to 0x9, for instance, B1 match does not occur, thus a 0% duty cycle signal is generated.

Input programmable filter (IPF)

The IPF ensures that only valid input pin transitions are received by the channel edge detector. A block diagram of the IPF is shown in *Figure 363*.

The IPF is a 5-bit programmable up counter that is incremented by the selected clock source, according to bits IF[0:3] in EMIOS_CCR[n].

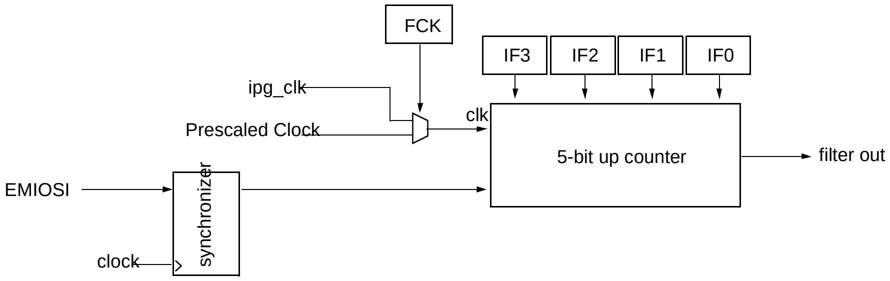


Figure 363. Input programmable filter submodule diagram

The input signal is synchronized by system clock. When a state change occurs in this signal, the 5-bit counter starts counting up. As long as the new state is stable on the pin, the counter remains incrementing. If a counter overflows occurs, the new pin value is validated. In this case, it is transmitted as a pulse edge to the edge detector. If the opposite edge appears on the pin before validation (overflow), the counter is reset. At the next pin transition, the counter starts counting again. Any pulse that is shorter than a full range of the masked counter is regarded as a glitch and it is not passed on to the edge detector. A timing diagram of the input filter is shown in *Figure 364*.

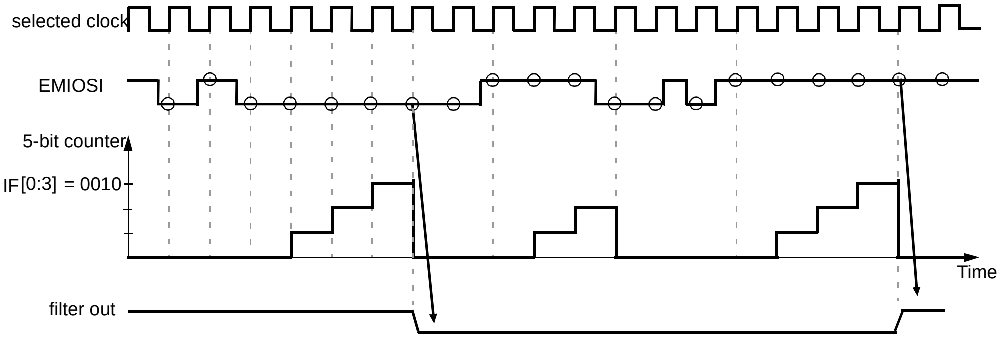


Figure 364. Input programmable filter example

The filter is not disabled during either freeze state or negated GTBE input.

Clock prescaler (CP)

The CP divides the GCP output signal to generate a clock enable for the internal counter of the Unified Channels. The GCP output signal is prescaled by the value defined in the UCPRE[0:1] bits in the EMIOS_CCR[n]. The prescaler is enabled by setting the UCPREN bit in the EMIOS_CCR[n] and can be stopped at any time by clearing this bit, thereby stopping the internal counter in the channel.

In order to ensure safe working and avoid glitches the following steps must be performed whenever any update in the prescaling rate is desired:

1. Write ‘0’ at both bit EMIOS_MCR[GPREN] and UCPREN bit in EMIOS_CCR[n], thus disabling prescalers;
2. Write the desired value for prescaling rate at UCPRE[0:1] bits in EMIOS_CCR[n];
3. Enable channel prescaler by writing ‘1’ at UCPREN bit in EMIOS_CCR[n];
4. Enable global prescaler by writing ‘1’ at bit EMIOS_MCR[GPREN].

The prescaler is not disabled during either freeze state or negated GTBE input.

Effect of freeze on the unified channel

When in debug mode, bit EMIOS_MCR[FRZ] and the FREN bit in the EMIOS_CCR[n] are both set, the internal counter and channel capture and compare functions are halted. The channel is frozen in its current state.

During freeze, all registers are accessible. When the channel is operating in an output mode, the force match functions remain available, allowing the software to force the output to the desired level.

Note that for input modes, any input events that may occur while the channel is frozen are ignored.

When exiting debug mode or freeze enable bit is cleared (bit EMIOS_MCR[FRZ] or FREN in the EMIOS_CCR[n]), the channel actions resume but may be inconsistent until the channel enters GPIO mode again.

21.5.2 IP bus interface unit (BIU)

The BIU provides the interface between the Internal Interface Bus (IIB) and the Peripheral Bus, allowing communication among all submodules and this IP interface.

The BIU allows 8, 16 and 32 bits access. They are performed over a 32-bit data bus in a single cycle clock.

Effect of freeze on the BIU

When bit EMIOS_MCR[FRZ] is set and the module is in debug mode, the operation of BIU is not affected.

21.5.3 STAC client submodule

The shared time and angle count (STAC) bus provides access to one external time base, imported from the STAC bus to the eMIOS unified channels. The eTPU module’s time bases and angle count can be exported and/or imported through the STAC client submodule interface. Time bases and/or angle information of the eTPU engine can be exported to the eMIOS module, which is only a STAC client. There are restrictions on engine export/import targets: an engine cannot export from or import to itself, nor can it import time base and/or angle count if in angle mode.

The device’s STAC server identification assignment is shown in *Table 344*. The time slot assignment is fixed, so only time bases running at system clock divided by four or slower can be integrally exported. The STAC client submodule runs with the system clock, and its time slot timing is synchronized with the eTPU timing on reset. The time slot sequence is 0-1-2-3.

Table 344. STAC client submodule server slot assignment

Time base	Server ID
TCR1	0
TCR2	2

Figure 365 provides a block diagram for the STAC client submodule.

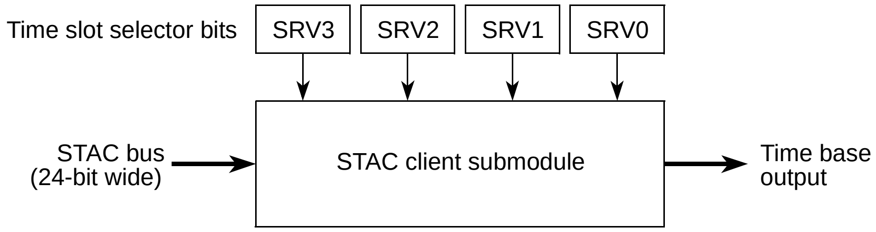
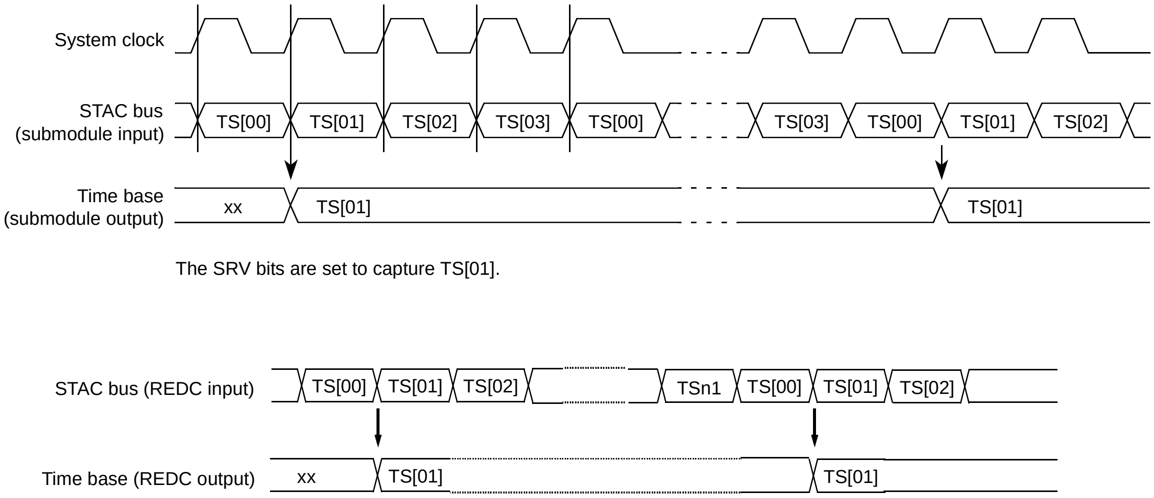


Figure 365. STAC client submodule block diagram

EMIOS_MCR[SRV] bits select the time slot of the STAC output bus. *Figure 366* shows a timing diagram for the STAC client submodule.



The SRV bits are set to capture TS[01].

NOTES:
1. Maximum of 16 time slots (TS*n*)
2. The SRV bits capture TS[01]

Figure 366. Timing diagram for STAC bus and STAC client submodule output

Every time the selected time slot changes, the STAC client submodule output is updated.

Effect of freeze on the STAC client submodule

When bit EMIOS_MCR[FRZ] is set and the module is in debug mode, the operation of the STAC client submodule is not affected; that is, there is no freeze function in this submodule.

21.5.4 Global clock prescaler submodule (GCP)

The GCP divides the system clock to generate a clock for the CPs of the channels. The main clock signal is prescaled by the value defined in the GPRE[0:7] bits in the EMIOS_MCR. The global prescaler is enabled by setting bit EMIOS_MCR[GPREN] and can be stopped at any time by clearing this bit, thereby stopping the internal counters in all the channels.

In order to ensure safe working and avoid glitches the following steps must be performed whenever any update in the prescaling rate is desired:

1. Write ‘0’ at bit EMIOS_MCR[GPREN], thus disabling global prescaler.
2. Write the desired value for prescaling rate at GPRE[0:7] bits in EMIOS_MCR.
3. Enable global prescaler by writing ‘1’ at bit EMIOS_MCR[GPREN].

The prescaler is not disabled during either freeze state or negated GTBE input.

Effect of freeze on the GCP

When bit EMIOS_MCR[FRZ] is set and the module is in debug mode, the operation of GCP submodule is not affected, that is, there is no freeze function in this submodule.

21.6 Initialization/Application information

On resetting the eMIOS200 the channels enter GPIO input mode.

21.6.1 Considerations

Before changing an operating mode, the UC must be programmed to GPIO mode and EMIOS_CADR[n] and EMIOS_CBDR[n] registers must be updated with the correct values for the next operating mode. Then the EMIOS_CCR[n] can be written with the new operating mode. If a channel is changed from one mode to another without performing this procedure, the first operation cycle of the selected time base can be random, that is, matches can occur in random time if the contents of EMIOS_CADR[n] or EMIOS_CBDR[n] were not updated with the correct value before the time base matches the previous contents of EMIOS_CADR[n] or EMIOS_CBDR[n].

When interrupts are enabled, the software must clear the FLAG bits before exiting the interrupt service routine.

21.6.2 Application information

Correlated output signals can be generated by all output operation modes. Bits OU[n] of the EMIOS_OUDR can be used to control the update of these output signals.

In order to guarantee that the internal counters of correlated channels are incremented in the same clock cycle, the internal prescalers must be set up before enabling the global prescaler. If the internal prescalers are set after enabling the global prescaler, the internal counters may increment in the same ratio but at a different clock cycle.

Channel/Modes initialization

The following basic steps summarize basic output mode startup, assuming the channels are initially in GPIO mode:

1. *[global]* Disable global prescaler.
2. *[timebase channel]* Disable channel prescaler.
3. *[timebase channel]* Write initial value at internal counter.
4. *[timebase channel]* Set A/B register.
5. *[timebase channel]* Set channel to MCB up mode.
6. *[timebase channel]* Set prescaler ratio.
7. *[timebase channel]* Enable channel prescaler.
8. *[output channel]* Disable channel prescaler.
9. *[output channel]* Set A/B register.
10. *[output channel]* Select timebase input through BSL[1:0] bits.

22 Enhanced Time Processing Unit (eTPU2)

22.1 Information specific to this device

This section presents device-specific parameterization and customization information not specifically referenced in the remainder of this chapter.

22.1.1 Device-specific features

- Single engine, 32 channel
- SCM size: 14 KB, no ECC support
- SDM size: 3 KB, no ECC support
- Nexus class 1 support
- Channels 24 to 29 input sources are selected via SIU IMUX (see *Section 15.6.22, IMUX Select Register 8 (SIU_ISEL8)*)
- Channel outputs can be serialized through via onboard DSPI
- Channel outputs 26 to 31 can be used to trigger the eQADC
- TCRCLK and Channel 0 are connected together internally on the 176-pin LQFP package

22.2 Introduction

eTPU is an intelligent, semi-autonomous co-processor designed for timing control. Operating in parallel with the Host CPU, the eTPU processes instructions, real-time input events, performs output waveform generation, and accesses shared data without Host intervention. Consequently, for each timer event, the Host CPU setup and service times are minimized or eliminated.

High-level assembler, compiler and documentation allows customers to develop their own functions on the eTPU.

eTPU is an enhanced version of the TPU module. Although there is no compatibility at microcode level, eTPU maintains several features of older TPU versions, making it easy to port older applications, at the same time adding several features listed in *Section , eTPU enhancements over TPU3*.

This document also includes the new features belonging to the version of the eTPU known as eTPU2. The new features are summarized in *Section , eTPU2 enhancements over eTPU*.

eTPU architecture aims at high resolution timing capabilities. From a system perspective, high resolution timing is limited by Host CPU overhead required for servicing timing tasks such as period measurement, pulse measurement, pulse width modulated waveform generation, etc. On the eTPU, high resolution timing is achieved by three main capabilities:

- Reduced latency: pin actions are immediate.
- Reduce or eliminate host interrupt service time.
- Double action channel capability reducing the channel request rate.

eTPU provides higher resolution than the Host CPU can achieve and creates no Host overhead for servicing timing tasks.

Latency is the interval from occurrence of an event to the start of event servicing. eTPU can service its own events without interrupting the Host. There are two types of timing events:

- Input pin transition
- Selected Time Base match, that is, a selected Time Base counter reached or exceeded a preprogrammed value

Service time is the time spent servicing an event. In general, in microcontrollers the service time is constrained because the instruction set is not optimized for time function synthesis. The eTPU instruction set is optimized, so that time functions can be implemented with much fewer instructions than the Host CPU. Instructions execute faster, service time is reduced and program memory compacted.

Instructions executed by the eTPU are connected directly to the eTPU timing hardware and allow parallelism of hardware related actions.

22.2.1 Overview

Figure 367 shows a top-level eTPU block diagram.

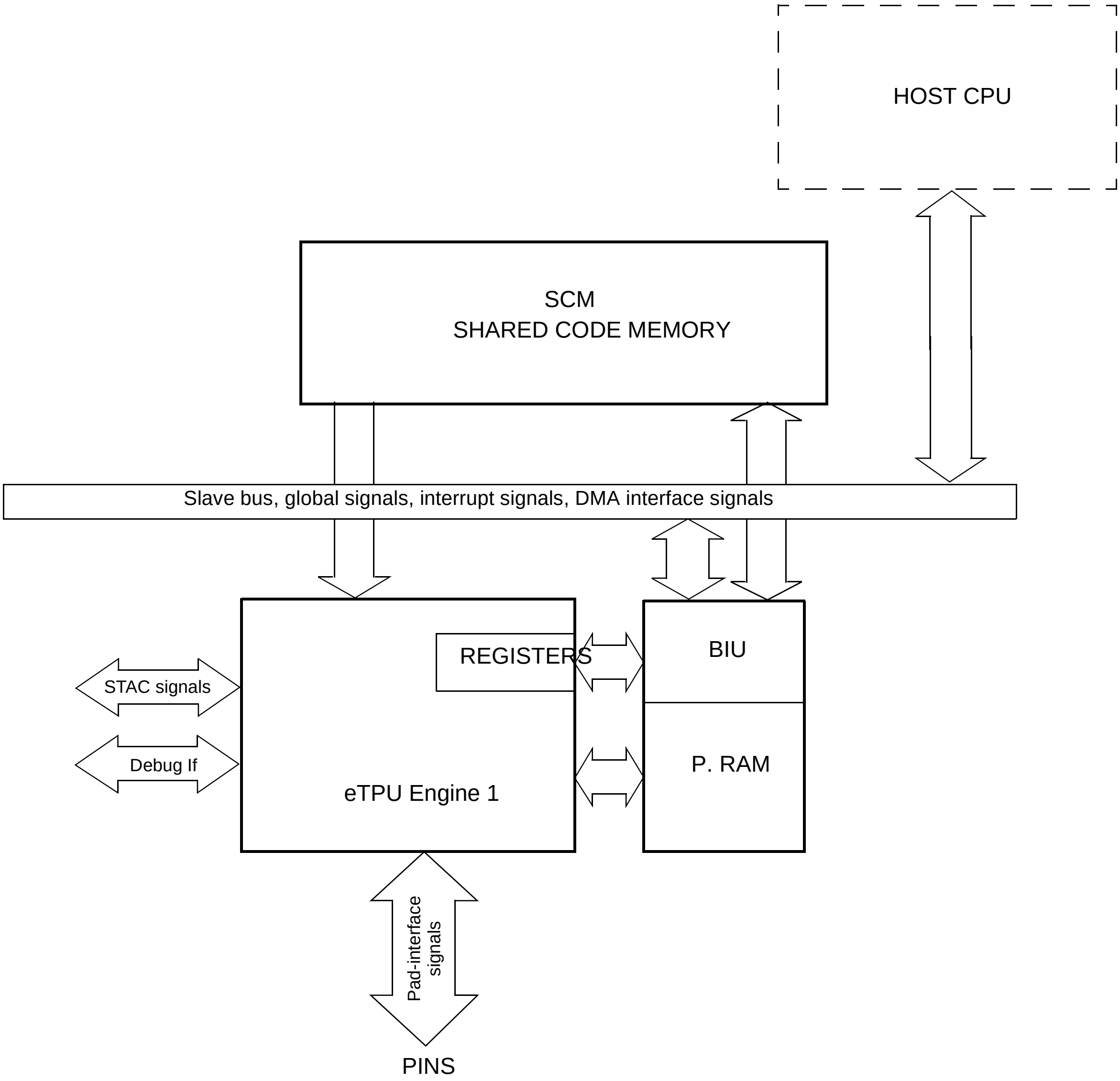


Figure 367. eTPU block diagram (single-engine)

The **eTPU engine** is responsible for processing input pin transitions and output pin waveform generation based on the **Time Bases**. The eTPU engine has its own microprocessor and dedicated hardware for processing signals on I/O pins and can also interface with external time bases through the STAC bus.

The eTPU engine CPU, hereafter called the **microengine**, fetches microinstructions from a **Shared Code Memory (SCM)**.

Shared Parameter RAM (**SPRAM)—**holds eTPU application parameters and work data. It is accessed by Host and the microengine.

Bus Interface Unit (BIU)—allows Host to access eTPU registers, SCM and SPRAM.

Each I/O signal pair is associated with a dedicated **Channel**, which provides hardware for input signal processing and output signal generation, in relationship with selected Time Bases.

The eTPU, as a microprocessed subsystem, works much like a typical real-time system: it runs microengine code from instruction memory (SCM) to handle specific events, accessing data memory (SPRAM) for parameters, work data and application state info; events may originate from I/O Channels (due to pin transitions and/or time base matches), Host CPU requests or inter-channel requests; events that call for local eTPU processing activate the microengine by issuing a **Service Request**. The Service Request microcode may set an interrupt to the Host CPU. I/O channel events cannot directly interrupt the Host CPU.

Each channel is associated with a **Function**, which defines its behavior: the Function is a software entity consisting, within the eTPU, of a set of microengine routines that attend to Service Requests. The Function routines are also responsible for Channel configuration. Function routines reside in SCM, which may contain several Functions. A Function may be assigned to several Channels, but a Channel can be associated with just one Function at a given moment. The association between Functions and Channels is defined by Host CPU, and is explained in detail in *Section 22.5.1, Functions and threads*.

eTPU hardware supplies resource sharing features that support concurrency:

- a hardware **Scheduler** dispatches the Service Request microengine routines based on a set of priorities defined by the Host CPU. Each Channel has its associated priority;
- a Service Request routine cannot be interrupted until it ends. This sequence of uninterrupted instruction execution is called a **Thread**.
- Channel-specific context (registers and flags) is automatically switched between the end of a Thread and the beginning of the next one.
- SPRAM arbitration, a dual-parameter coherency controller and semaphores can be used to ensure coherent access to eTPU data shared by both eTPU engines and Host CPU.

eTPU engine

The eTPU engine consists of two 24-bit time bases, 32 independent timer channels, a task scheduler, a microengine, and a Host interface and 32-bit Shared Parameter RAM (SPRAM). In dual-engine implementations of the eTPU, SPRAM is used for both eTPU engine's data storage and for passing information between the eTPU engines and the host CPU.

Figure 368 shows the block diagram for the eTPU engine.

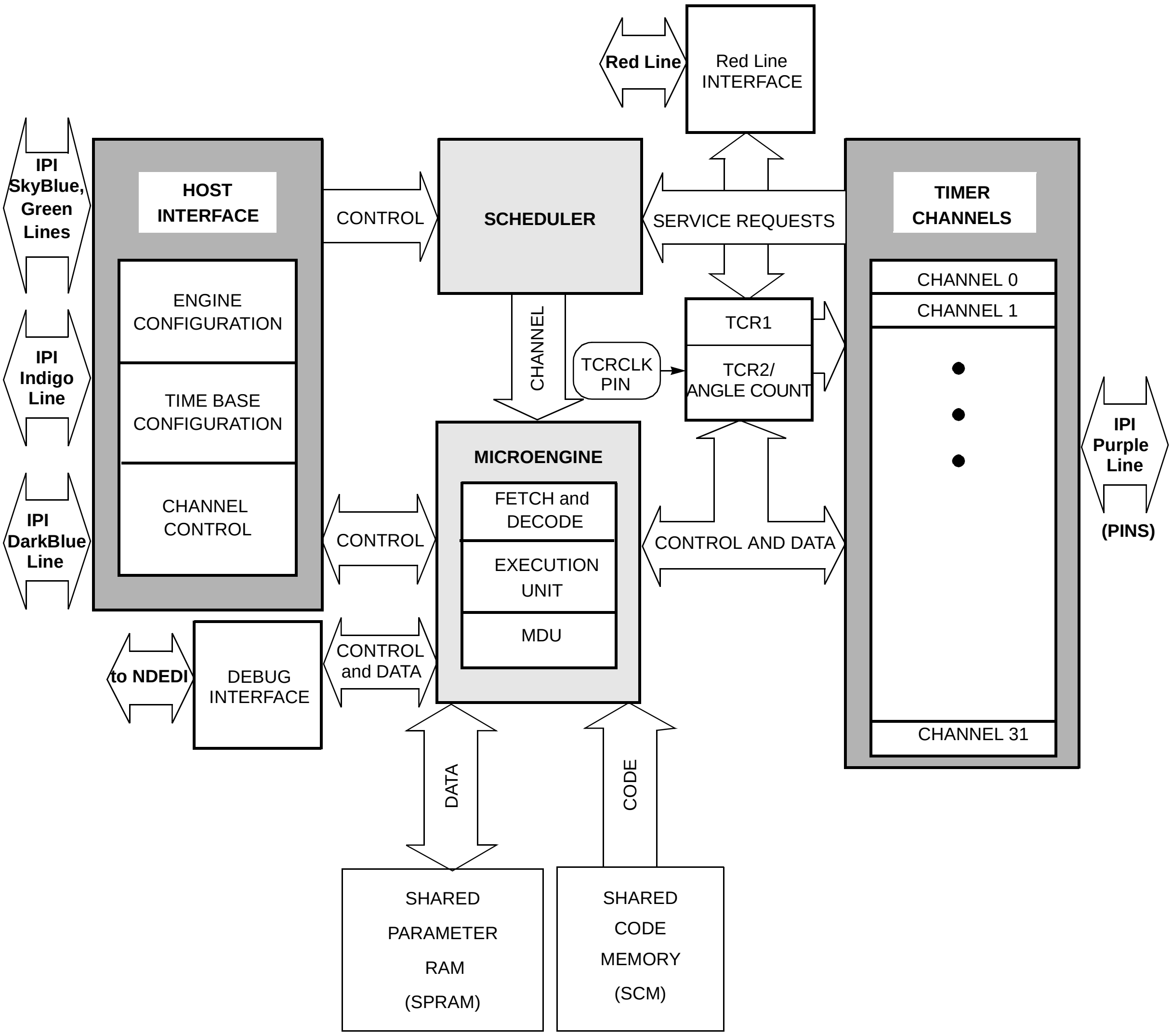


Figure 368. eTPU engine block diagram

eTPU engines 1 and 2 are sometimes called eTPU 1 and eTPU 2 throughout this document.

Time bases

Two 24-bit counters TCR1 and TCR2 provide reference time bases for all match and input capture events. Prescalers for both time bases are controlled by the Host CPU through bit fields in the eTPU engine configuration registers. The eTPU is able to export/import time to/from TCR1 or TCR2 in accordance to the Red Line bus specification.

The clock for each of TCR1 and TCR2 clock can be independently derived from the system clock or from an external input via the TCRCLK clock pin. In addition, the TCR2 timebase can be derived from special angle-clock hardware which enables implementing angle-based functions. This feature is added to support advanced angle based engine control applications.

For further details refer to *Section 22.5.6, Time Bases*.

eTPU timer channels

The eTPU engine has 32 independent channels, each corresponding to an Input/Output signal pair. The channels time resolution is 24 bits, and are all identical.

Each channel consists of logic which supports two events and output controls. The event logic contains two 24-bit capture registers, two 24-bit match registers, greater-equal and equal-only comparators. Supporting two events enables many combinations of double-action functions (for example the channel can handle two events with a single microcode service).

The channel configuration can be changed by the microengine on the fly. Each channel can perform double capture, double match and other capture-match combinations. Channel modes available can do ordered or unordered match. Some modes are also provided that can block one match by the occurrence of the other. Service request can be generated on one or both of the match events.

Input signal can be separated from output signal in each channel. They can, optionally, be combined in a single I/O pin driver. An output buffer enable signal, controlled by microcode, is provided for this case. Digital filters are provided for the input signals, with distinct filtering modes available.

Each channel can use any time base or angle counter for either match or capture operation. For example, a match on TCR1 can capture the value of TCR2. The channels can request service from the microengine due to recognized pin transitions (input events) or timebase matches.

The eTPU channels also support the basic single-action operations found on TPU3 functionality with the exception that time resolution is 24 bits.

Channel configuration combinations:

- Single input capture, no match (TPU3 functionality).
- Single input capture with single match timeout (TPU3 functionality).
- Single input capture with double match timeout with several double match submodes.
- Double input capture with single or double match timeout with several double match submodes.
- Single output match (TPU3 functionality).
- Double output match with several double match submodes.
- Input-dependent output generation.

The double match functionality has various combinations for generation of service request and determining pin actions. For more details refer to *Section 22.5.5, Enhanced Channels*.

In addition to the predefined channel configurations above, the user can also program its own channel configuration, defining how input captures, matches and service-requests are related.

Host interface

The Host interface allows the Host CPU to control the operation of the eTPU. The Host CPU must initialize the eTPU by writing to the appropriate Host interface registers to assign a Function and priority to each channel. In addition, the Host writes to the Host Service Request and channel configuration registers to further define Function operation for each initialized channel. Refer to *Section 22.5.2, Host interface* for a detailed description.

When the SCM is implemented by RAM, the Host must first initialize it with the proper microcode program prior to enabling any eTPU Function, and then enable eTPU access (which also disables Host access).

Shared parameter RAM (SPRAM)

The SPRAM works as data RAM which can be accessed by the Host CPU and up to two eTPU engines. This memory is used for information transfer between the Host CPU and the eTPU, as data storage for the eTPU microcode program or for communication between the two eTPU engines. SPRAM width is 32 bits, and is accessible by the Host as byte, 16-bit or 32-bit wide. eTPU can access it as full 32 bits, lower 24 bits or upper byte (8-bit).

The host can also access the SPRAM space mirrored in other area with Parameter Signal Extension (PSE). Parameter Signal Extension accesses differ from the usual host accesses to the original SPRAM area as follows:

- Writes are effective only to the lower 3 bytes of a word: the word’s most significant byte is kept unaltered in SPRAM.
- Reads return the lower 3 bytes of a word sign-extended to 32 bits, i.e.: the most significant bit of the word s 2nd most significant byte is copied in all 8 bits of the most significant read byte.

Each eTPU channel can be associated with a variable number of parameters located in the SPRAM, according to its selected Function. In addition, the SPRAM can be fully shared between two eTPU engines, enabling direct communication between them.

High flexibility of the SPRAM utilization is achieved as follows:

- Each channel has a programmable base address pointing to the address of its first parameter with two parameter granularity. This way the SPRAM can be partitioned according to the actual function needs.
- The microcode can access the first 128 parameters of the selected channel in channel relative access mode.
- Each engine can access all the SPRAM address space in indirect addressing mode. Blocks of data are easily transferred using stack operation.
- Absolute addressing mode can access the first 256 parameters (TPU3 functionality), implementing a shared pool of parameters holding global variables.

In the Host address space each parameter occupies four bytes. eTPU usage of the upper byte is achieved by having a 32-bit P register which can access the upper byte, the lower 24 bits or all the 32 bits. The microcode can switch between access sizes at any time.

Each Function may require a different number of parameters. During the eTPU initialization the Host has to program channel base addresses, allocating proper parameters for each channel according to its selected Function.

Scheduler

Out of reset, all channels are disabled. The Host CPU makes a channel active by assigning it one of three priorities: high, middle, or low. The Scheduler determines the order in which channels are serviced based on channel number and assigned priority. The priority mechanism, implemented in hardware, ensures that all requesting channels are serviced. For additional details refer to *Section 22.5.3, Scheduler*.

Microengine

eTPU microengine is a simple VLIW implementation that performs each instruction in a microcycle of two system clocks, while prefetching the next instruction through an instruction pipeline. Instruction execution time is constant unless it gets wait states from the SPRAM arbitration. Two eTPU engines share code memory without having any performance degradation by interleaving their accesses (the Shared Code Memory has one-clock access time).

Instruction width is 32 bits. The microengine instruction set provides basic arithmetic and logic operations, flow control (jumps and subroutine calls), SPRAM access, and Channel configuration and control. The instruction formats are defined in such a way that allow particular combinations of two or three of these operations with unconflicting resources to be executed in parallel in the same microcycle.

Microengine has also an independent Multiply/Divide/MAC unit that performs these complex operations in parallel with other microengine instructions.

Channel functionality is tightly integrated to the instruction set through Channel Control operations and conditional Branch operations, which support jumps/calls on Channel-specific conditions. This allows quick and terse Channel configuration and control code, contributing to reduced service time.

Detailed description can be found in *Section 22.5.8, Microengine*.

Single vs. dual eTPU engine system

An eTPU implementation can include one or two eTPU engines. The number is engines is specific to the microcontroller design and cannot be changed.

On devices with two eTPU engines, the eTPU parameter RAM (SPRAM), code memory (SCM) and Bus Interface Unit (BIU) are shared by both engines, enabling processor core-to-eTPU communication and eTPU engine-to-engine communication.

In dual-engine eTPUs the shared BIU includes coherency logic which supports dual-parameter (8 bytes) coherency in transfers between the processor core and eTPU, using a temporary parameter area within the SPRAM. More details on this can be found on *Section 22.5.4, Parameter sharing and coherency*.

22.2.2 Features

eTPU feature summary

The eTPU includes these distinctive features:

- Up to 32 channels per eTPU engine—each channel is associated with an I/O signal pair.
  - Enhanced input digital filters on the input pins for improved noise immunity. The eTPU digital filter can use 2 samples, 3 samples or work in continuous mode.
  - Identical, orthogonal channels, except for channel 0: each channel can perform any time function. Each time function can be assigned to more than one channel at a given time, so each signal can have any functionality. Channel 0 has the same capabilities of the others, but can also work with special Angle Counter logic (see below).

  - Link Service Request allows activation of a Channel function by request of another channel, even between eTPU engines.
  - Host Service Request allows activation of a Channel function by Host CPU request
  - Each channel has an event mechanism which supports single and double action functionality in various combinations. It includes two 24-bit capture registers, two 24-bit match registers, 24-bit greater-equal and equal-only comparators.
- 2 independent 24-bit time bases for channel synchronization:
  - First time base clocked by system clock with programmable prescaler division from 1 to 512 (in steps of 2), or by output of second time base prescaler.
  - First time base can also be clocked by external signal with programmable prescaler division of 1 to 256.
  - Second time base clocked by external signal with programmable prescaler division from 1 to 64.
  - Second time base external clock source can be replaced by system clock divided by 8.
  - Both time bases can be exported or imported via Shared Time and Counter) bus.
  - Second time base counter can work as an Angle counter, enabling angle based applications to match angle instead of time.
  - Second time base can also be used as a pulse accumulator gated by external signal.
- Event-Triggered VLIW processor (microengine):
  - 2 stage pipeline implementation (fetch and execution), with separate instruction memory - SCM - and data memory - SPRAM (Harvard architecture)
  - Fixed-length instruction execution in two system clock microcycle
  - Interleaved SCM access in dual eTPU engine avoids contention in time for instruction memory
  - SCM address space of up to 16K positions (64 Kbytes)
  - SPRAM with interleaved access in dual eTPU engine avoids contention for data memory
  - SPRAM address space of up to 8 Kbytes (both engines).
  - Instruction set with embedded Channel support, including specialized Channel control subinstructions and conditional branching on Channel-specific flags.
  - Channel-oriented addressing: channel-bound address mode with Host configured Channel Base Address allows channel data isolation, independent of microengine application code.
  - Channel-bound data address space of up to 128 32-bit parameters (512 bytes)
  - Global parameter address mode allows access to common Channel data of up to 256 32-bit parameters (1024 bytes)
  - Support for indirect and stacked data access schemes.
  - Parallel execution of: data access, ALU, Channel control and flow control subinstructions in selected combinations.
  - 32-bit microengine registers and 24-bit resolution ALU, with 1 microcycle addition and subtraction, absolute value, bitwise logical operations on 24-bit, 16-bit, or byte operands; single-bit manipulation, shift operations, sign extension and conditional execution.

  - Additional 24-bit Multiply/MAC/Divide unit which supports all signed/unsigned Multiply/MAC combinations, and unsigned 24-bit Divide. The MAC/Divide unit works in parallel with the regular microcode commands.
- Resource sharing features support channel sharing of channel registers, memory and microengine time:
  - Hardware Scheduler works as a “task management” unit, dispatching event service routines by predefined, Host-configured priority.
  - Automatic Channel context switch when a “task switch” occurs, i.e., one Function Thread ends and another begins to service a request from other Channel: Channel-specific registers, flags and parameter base address are automatically loaded for the next serviced channel.
  - Individual channel priority setting in 3 levels: high, middle and low.
  - Scheduler priority scheme allows calculation of worst-case latency for event servicing and ensures servicing all channels by preventing permanent blockage.
  - SPRAM shared between Host CPU and both eTPU engines, supporting communication either between Channels and Host or inter-channel.
  - Hardware implementation of 4 Semaphores supports resource sharing between both eTPU engines.
  - Hardware semaphores directly supported by the microengine instruction set.
  - Dual-parameter coherency hardware support allows atomic (to host) access to 2 parameters by microengine(s) in back-to-back accesses.
  - Coherent dual-parameter controller allows atomic (to microengines) accesses to 2 parameters by the host.
- Test and Development support features:
  - Nexus class 3 debug support (optional, associated with the eTPU-Nexus Block NDEDI).
  - Software breakpoints.
  - Debug interface supporting single-step execution, forced microinstruction execution, Hardware breakpoints and watchpoints on several conditions.
  - SCM (code memory) continuous signature-check built-in self test (MISC, or Multiple Input Signature Calculator), runs concurrently with eTPU normal operation.

eTPU enhancements over TPU3

- 32 orthogonal channels with enhanced functionality. Full support for double action with double match and double transition submode combinations.
- Input and Output features separated in channel logic and microinstructions, allowing input and output signals to be processed separately or combined.
- Increased time resolution and execution unit to 24 bits
- Increased linear code memory, shared by two eTPU engines, configurable up to 16K positions (64 Kbytes)
- Increased Parameter RAM address range (8 Kbytes each engine) and width (32 bits per parameter). The Parameter RAM can be dynamically allocated to support variable number of parameters for each channel. Each channel can have access to at least 256 parameters.

- The Parameter RAM is fully shared by two eTPU engines (SPRAM), supporting direct inter-engine communication with the help of hardware semaphores.
- Enhanced arithmetic operations, including add/subtract with carry, absolute value, multiple shift and rotate, conditional execution with variable operand widths
- Enhanced logic operations, including bitwise operations (and, or, xor) and bit manipulation, with conditional execution. Support for read-modify-write of any bit in the SPRAM.
- Hardware for Multiply/MAC/Divide, running in parallel to execution of other operations. The 24-bit divide result is available after 13 other unrelated instructions. Multiplication supports any data width of both operands (8, 16 or 24 bits), signed or unsigned. A 24x24 Multiply/MAC result is available after four other unrelated instructions. A 24x8 Multiply/MAC result is available after one other unrelated instruction.
- Supports export/import of time bases from other sources through the real time bus (STAC - Shared Time and Counter bus). This internal bus is used for sharing real time data between multiple peripherals.
- Contains angle clock hardware, supported by microcode, which can provide a 24-bit angle bus instead of time bus. This feature enables the eTPU to run angle based engine control applications.
- More interrupt types. Each eTPU channel can generate a data transfer request interrupt, in addition to regular interrupts, and one global exception interrupt. Data Transfer requests can be used either as interrupt sources or DMA requests. This feature takes advantage of DMA peripherals which offload the Host. Interrupt Overflow status is also provided.
- Improved visibility to the Host (pin states, time bases, serviced channel)
- An edge case of priority inversion on TPU3 Scheduler was resolved.
- Supports channel link requests between eTPU engines

eTPU2 enhancements over eTPU

- TCR1, channel logic and digital filters (both channel and TCRCLK) now have an option to run at divisions of full system clock speed, besides system clock / 2.
- Channels support unordered transitions: transition B can now be detected before transition A. Related to this enhancement, TDLA and TDLB can now be independently negated by microcode.
- Added a new User Programmable Channel Mode: the blocking, enabling, service request and capture characteristics of this channel mode can be programmed via microcode.
- Microinstructions now provide an option to issue Interrupt and Data Transfer requests selected by CHAN. They can also be requested simultaneously at the same instruction.
- Channel Flags 0 and 1 can now be tested for branching, besides selecting the entry point.
- Channel digital filters can be bypassed.
- Scheduler priority-passing mechanism can now be disabled.
- New Watchdog mechanism kills threads over a programmable timeout.
- New counter allows microengine load information collection for performance analysis
- Channels 1 and 2 (besides channel 0) can now be selected to control the EAC.
- Timebase prescalers are now reset when the GTBE input is negated, guaranteeing synchronization with eMIOS in all cases.
- New MISC flag indicates when an SCM signature calculation round is completed. This allows measuring of the average MISC scan period in a real application situation.
- New channel TCCEA flag allows continuous capture even after TDLA is set, making it fully compatible with TPU behavior.
- New branch condition PRSS tells the pin state at the time when a channel (match or transition) service request occurred.
- MRLEA/B can now be negated independently by microcode.
- New engine Relative address mode allows a function to access SDM address space common to one engine, but distinct between engines.
- Error Correction support for Code (SCM) and Data (SDM) memories (available on selected MCUs).
- All changes above are upward compatible with the classic eTPU, so that legacy object code (both Host and microcode) runs on eTPU+ and eTPU2 without modification.

22.2.3 Modes of operation

The eTPU2 is capable of working in the following modes:

- User Configuration Mode

  User has the ability to program the eTPU Cores with User Time Functions, having access to the Shared Code Memory (SCM).
- User Mode

  User does not access the eTPU Shared Code Memory:
  - Use of predefined eTPU Functions
  - No need for eTPU Core programming ability
- Debug Mode

  User debugs eTPU code, accessing special Trace/Debug features via Nexus interface:
  - Hardware breakpoint/watchpoint setting
  - Access to internal registers
  - Single-step execution
  - Forced instruction execution
  - Software breakpoint insertion and removal.
- Module Disable Mode

  eTPU engine clocks are stopped through a register write to ETPU_ECR bit MDIS, saving power. Input sampling stops. eTPU engines can be in Module Disable Mode independently. Module Disable Mode stops only the engine clock, so that the Shared BIU, and Global Channel registers can be accessed, and interrupts and DMAs can be cleared and enabled/disabled. An engine only enters Module Disable Mode when any currently running thread is finished (see *Section 22.5.1, Functions and threads*).
- Stop Mode

  Stop Mode is entered when eTPU answers device stop request assertion with stop acknowledge. The definition of which clocks are stopped is made at the MCU level, which defines whether or not registers can be accessed, interrupts and DMA requests cleared.

These modes are loosely selected: there is no unique register field or signals to choose between them. Some features of one mode can be used with features of other mode(s). More on this subject can be found on *Section , eTPU mode selection*, below.

Note: Throughout this document, an engine is said to be “stopped” if it is either in Module Disable mode or Stop mode.

eTPU mode selection

User and User Configuration are the production operating modes, and differ from each other only in access to SCM. User programmability is only possible with a RAM SCM.

On chips where the SCM is implemented as a RAM, it can either be accessed directly from IP-Bus for code loading, or for software breakpoint setting. On chips with a ROM SCM, an internal SCM Emulation RAM may be used, depending on the specific MCU implementation, to replace ROM SCM for test or debug purposes. SCM Emulation RAM is selected in an MCU-specific way. For more details, see *Section , SCM emulation*.

For more information on SCM access, Debug and Test features, refer to *Section 22.5.10, Test and Development Support*.

Debug Mode is characterized by the use of the debug interface features. Debug features may be implemented using the eTPU-NDEDI internal interface. Specifically, this interface may be used with Nexus implementation blocks to provide Nexus class 3 debug features. The use of eTPU-NDEDI interface and Nexus implementation is MCU-dependent.

Module Disable Mode is entered by setting ETPU_ECR bit MDIS. eTPU engines can be individually stopped going into Module Disable Mode (there is one ETPU_ECR for each engine). Each engine can leave Module Disable Mode by writing MDIS = 0 (which can only be done if VIS = 0).

Stop Mode is activated by IP-Bus (device stop request). In this case, the eTPU waits for both eTPU engines to enter in stop mode, and then asserts the stop acknowledge line. eTPU leaves Stop Mode when device stop request is negated, but only if VIS = 0. If device stop request is negated and VIS = 1, eTPU will leave Stop Mode as soon as VIS = 0.

Note: An engine can stay in Module Disable mode when it leaves Stop Mode if its bit MDIS = 1, even if the other leaves it.

22.3 External signal description

22.3.1 Overview

There are 69 external signals associated with each eTPU engine: 32 channel input signals, 32 channel output signals, 4 output disable inputs, and TCRCLK clock input, totaling 138 in a dual-engine system. These signals are described in *Table 345*.

Depending on the MCU integration, the input and output signals of a channel can be tied to one pin. In this case, the direction of each channel signal, either output or input, is determined by the activation of an output enable driver signal. eTPU provides one output buffer enable signal for each channel, controlled by microcode.

The TCRCLK signal is used to clock TCR1/2 counters or gate the TCR2 clock. In Angle Mode it is used as a tooth signal input. Refer to *Section 22.5.6, Time Bases*, and *Section 22.5.7, EAC – eTPU angle counter*, for proper use of this signal.

Table 345. eTPU signal properties

Name	Direction	Function	Reset state	Pull up
ipp_ind_etpuch_1(0) to ipp_ind_etpuch_1(31)	Input	eTPU engine 1 channel signals	—	MCU dependent
ipp_do_etpuch_1(0) to ipp_do_etpuch_1(31)	Output	eTPU engine 1 channel signals	0 / Hi-Z(1)	MCU dependent
ipp_ind_etpu_odis_1(0) to ipp_ind_etpu_odis_1(3)	Input	eTPU engine 1 output disable signals	—	MCU dependent
ipp_ind_tcrclk_1	Input	Clock/gate for eTPU engine 1 TCR counters; entry of the tooth signal in Angle Mode	—	MCU dependent
ipp_ind_etpuch_2(0) to ipp_ind_etpuch_2(31)	Input	eTPU engine 2 channel signals	—	MCU dependent
ipp_do_etpuch_2(0) to ipp_do_etpuch_2(31)	Output	eTPU engine 2 channel signals	0 / Hi-Z(1)	MCU dependent

Table 345. eTPU signal properties (continued)

Name	Direction	Function	Reset state	Pull up
ipp_ind_etpu_odis_2(0) to ipp_ind_etpu_odis_2(3)	Input	eTPU engine 2 output disable signals	—	MCU dependent
ipp_ind_tcrclk_2	Input	Clock/gate for eTPU engine 2 TCR counters; entry of the tooth signal in Angle Mode	—	MCU dependent

1. Value 0 refers to the reset value of the signal. Hi-Z refers to the state of the pads, if controlled by the eTPU output buffer Enable signals, i.e., eTPU output buffer Enable resets in negated state.

22.3.2 Detailed signal descriptions

ipp_do_etpuch_[1|2]([0 – 31]) — eTPU channel output signals

Each channel output signal is associated with a channel. The microcode may affect the logic level of an output signal(o) by implementing one of two actions:

- Specify the logic level output to the signal when there is a match or a transition.
- Immediately force a logic level.

The output signal may also be forced to a logic level, independently of the output value from the channel logic, by one of the four (each engine) output disable input signals ipp_ind_etpuodis (see *Section , ipp_ind_etpu_odis_[1|2]([0 – 3]) eTPU Channel Output Disable Signals*).

The output signal driver may be, depending on MCU integration, enabled by the output buffer enable internal signal that comes from eTPU. In this case, the output buffer can be controlled by microcode, through a specific microinstruction field. There is one independent output buffer Enable signal for each channel. For more information on output control from microcode, refer to *Section , Transition detection and pin action control*.

ipp_ind_etpuch_[1|2]([0 – 31]) — eTPU Channel Input Signals

Each channel input signal is associated with a channel. The microcode can directly control the effect of the transition edge. Each channel can be programmed to sense a transition when a rising and/or falling edge is detected. The channel logic can also process two transition events, and relate these events to each other and to other programmed timer events. The edge sensitivities of the two transition events are configured independently by microcode. For further information refer to *Section 22.5.5, Enhanced Channels*, and *Section , Transition detection and pin action control*.

Each channel input signal has an associated synchronizer made of two flip-flops sampling the signal on every other system clock(p), followed by a digital filter. This digital filter can work in three submodes, whose purpose is to filter out noise pulses that have width less then a programmed value of system clocks, preventing these transitions from being input to the transition detect logic. The synchronizer and digital filter are guaranteed to pass pulses that are greater than a programmed value. All channel input filters in one engine work on the

---

o. Note that the minimum pulse width is one microcycle (two system clocks), and slow 5V pads may not be able to transfer it on time. For generation of very short pulses the eTPU pads have to be programmed by the system integration for fast operation mode with the voltage levels defined for fast pad operation in the MCU technology.

p. Sampled on the T4 microcycle phase, see *Section 22.7.1, Microcycle and I/O timing*.

same mode and sampling clock. For more information on channel input filters, refer to *Section , Enhanced Digital Filter – EDF*. In one of the Angle Modes, the output of the digital filter of channel 0 is replaced by the output of TCRCLK digital filter (see *Section 22.5.7, EAC – eTPU angle counter*).

ipp_ind_tcrclk_etpu_[1|2] — Time Base Clock Signal — TCRCLK

TCRCLK is an input signal used to control the Time Bases TCR1 and TCR2. There is one independent TCRCLK input for each engine. For pulse accumulator operations TCRCLK can be used as a gate for a counter based on the system clock divided by eight. For Angle operations TCRCLK can be used to get the tooth transition indications in Angle Mode. Further details can be found on *Section 22.5.6, Time Bases*, and *Section 22.5.7, EAC – eTPU angle counter*.

Like the channel input signals, the TCRCLK signal has an associated synchronizer followed by a digital filter. This digital filter can work in two submodes, whose purpose is to filter out noise pulses that have width less then a programmed value of system clocks, preventing these transitions from being input to the transition detect logic. The synchronizer and digital filter are guaranteed to pass pulses that are greater than a programmed value. The clock and operation submode of the TCRCLK filter is configured independently of the other channel input filters, through the field ETPU_TBCR[TCRCF].

For more information on filter submodes, refer to *Section , TCRCLK digital filter*. In one of the Angle Modes, the output of the digital filter of channel 0 is replaced with the output of TCRCLK signal digital filter (see *Section 22.5.7, EAC – eTPU angle counter*).

ipp_ind_etpu_odis_[1|2]([0 – 3]) eTPU Channel Output Disable Signals

Each of these four input signals are used to force the outputs of a group of eight channels to an inactive level. When an ODIS input is active, all the channels in its group of eight that have their bits ODIS = 1 in ETPU_CxCR have their outputs forced to the opposite of the value specified in bit OPOL of the same register. Therefore, channels can be individually selected to be affected by the output disable signals, as well as their disabling forced polarity (see *Figure 403*).

The output disable channel groups are defined in *Table 346*.

Table 346. Output disable channel groups

Output disable signal[1]	Channels
ipp_ind_etpu_odis_[1|2](0)	0 to 7
ipp_ind_etpu_odis_[1|2](1)	8 to 15
ipp_ind_etpu_odis_[1|2](2)	16 to 23
ipp_ind_etpu_odis_[1|2](3)	24 to 31

[1] The ETPU2 output_disable signals ipp_ind_etpu_odis_1(0 to 3) are connected to the EMIOS channel _flags_ (channel 11 to 8) respectively.

In a dual-engine eTPU there are 8 output disable signals for the 64 channels.

22.4 Memory map/register definition

The guideline for the description of all bits and fields throughout *Section 22.4, Memory map/register definition*, is to provide only a brief explanation (without examples or method of use) of the features, since it will be used mainly as a reference for the reader that is studying *Section 22.5, Functional description*, where those features are explained in detail.

22.4.1 Memory map

The eTPU System simplified memory map is shown in *Table 347*. Each of the register areas shown may have their own reserved address areas.

Table 348 shows a detailed memory map. Offsets are relative to the eTPU base address, which is MCU-dependent.

The SCM used area is informed to the eTPU address decoding logic through the plug etpu_scmsize_plug[4:0]. SCM unused area is decoded and returns a fixed opcode, determined by the register ETPU_SCMOFFDATAR.

Table 347. High level memory map

Offset	Use
0x00 – 0x1F	System Configuration Registers
0x20 – 0x2F	eTPU 1 Time Base Registers
0x30 – 0x3F	RESERVED(1)
0x40 – 0x4F	RESERVED(1)
0x50 – 0x5F	RESERVED(1)
0x60 – 0x6F	eTPU 1 Extra Engine Registers
0x70 – 0x7F	RESERVED(1)
0x80 – 0xFF	RESERVED(1)
0x100 – 0x13F	RESERVED
0x140 – 0x1FF	RESERVED(1)
0x200 – 0x2FF	eTPU 1/2 Global Channel Registers
0x300 – 0x3FF	RESERVED(1)
0x400 – 0x7FF	eTPU 1 Channel Registers
0x800 – 0xBFF	eTPU 2 Channel Registers
0xC00 – 0x7FFF	RESERVED(1)
0x8000 – 0xBFFF(2)	SPRAM
0xC000 – 0xFFFF(2)	SPRAM PSE mirror (3)
0x10000 – 0x1FFFF(2)	SCM(4)

1. Reserved addresses must not be used. Access to these memory positions complete with 0-wait-states, but may cause unpredictable behavior.
2. Actual sizes of SCM and SPRAM are MCU-dependent.
3. Parameter Sign Extension access area, see *Section , Parameter access*
4. SCM access is available only when bit VIS = 1 on register ETPU_MCR, under certain conditions (see *Section , ETPU_MCR – eTPU Module Configuration Register*).

Table 348. Detailed memory map

Offset	Use	Location
0x00	ETPU_MCR – eTPU Module Configuration Register	*on page 22-627*
0x04	ETPU_CDCR – eTPU Coherent Dual-Parameter Controller Register	*on page 22-630*
0x08	RESERVED	
0x0C	ETPU_MISCCMPR – eTPU MISC Compare Register	*on page 22-633*
0x10	ETPU_SCMOFFDATAR – eTPU SCM Off-range Data Register(1)	*on page 22-633*
0x14	ETPU_ECR_1 – eTPU 1 Engine Configuration Register	*on page 22-634*
0x18	RESERVED	
0x1C	RESERVED	
0x20	ETPU_TBCR_1 – eTPU 1 Time Base Configuration Register	*on page 22-640*
0x24	ETPU_TB1R_A – eTPU Time Base 1 (TCR1) Visibility Register	*on page 22-644*
0x28	ETPU_TB2R_A – eTPU Time Base 2 (TCR2) Visibility Register	*on page 22-644*
0x2C	ETPU_REDCR_1 - eTPU 1 STAC Configuration Register	*on page 22-645*
0x30	RESERVED	
0x34	RESERVED	
0x38	RESERVED	
0x3C	RESERVED	
0x40	RESERVED	
0x44	RESERVED	
0x48	RESERVED	
0x4C	RESERVED	
0x50	RESERVED	
0x54	RESERVED	
0x58	RESERVED	
0x5C	RESERVED	
0x60	ETPU_WDTR_1 – eTPU 1 Watchdog Timer Register	*on page 22-648*
0x64	RESERVED	
0x68	ETPU_IDLE_1 – eTPU 1 Idle Counter Register	*on page 22-649*

Table 348. Detailed memory map (continued)

Offset	Use	Location
0x6C	RESERVED	
0x70	RESERVED	
0x74	RESERVED	
0x78	RESERVED	
0x7C	RESERVED	
0x80 – 0xFF	RESERVED	
0x100	RESERVED	
0x104	RESERVED	
0x108	RESERVED	
0x10C	RESERVED	
0x110	RESERVED	
0x114	RESERVED	
0x118	RESERVED	
0x11C	RESERVED	
0x120	RESERVED	
0x124	RESERVED	
0x128	RESERVED	
0x12C	RESERVED	
0x130	RESERVED	
0x134	RESERVED	
0x138	RESERVED	
0x13C – 0x1FF	RESERVED	
0x200	ETPU_CISR_1 – eTPU 1 Channel Interrupt Status Register	*on page 22-651*
0x204	RESERVED	
0x208	RESERVED	
0x20C	RESERVED	
0x210	ETPU_CDTRSR_1 – eTPU 1 Channel Data Transfer Request Status Register	*on page 22-652*
0x214	RESERVED	
0x218	RESERVED	
0x21C	RESERVED	
0x220	ETPU_CIOSR_1 – eTPU 1 Channel Interrupt Overflow Status Register	*on page 22-653*
0x224	RESERVED	
0x228	RESERVED	

Table 348. Detailed memory map (continued)

Offset	Use	Location
0x22C	RESERVED	
0x230	ETPU_CDTROSR_1 – eTPU 1 Channel Data Transfer Request Overflow Status Register	*on page 22-654*
0x234	RESERVED	
0x238	RESERVED	
0x23C	RESERVED	
0x240	ETPU_CIER_1 – eTPU 1 Channel Interrupt Enable Register	*on page 22-655*
0x244	RESERVED	
0x248	RESERVED	
0x24C	RESERVED	
0x250	ETPU_CDTRER_1 – eTPU 1 Channel Data Transfer Request Enable Register	*on page 22-656*
0x254	RESERVED	
0x258-0x27F	RESERVED	
0x280	ETPU_CPSSR_1 – eTPU 1 Channel Pending Service Status Register	*on page 22-657*
0x284	RESERVED	
0x288	RESERVED	
0x28C	RESERVED	
0x290	ETPU_CSSR_1 – eTPU 1 Channel Service Status Register	*on page 22-658*
0x294	RESERVED	
0x298	RESERVED	
0x29C	RESERVED	
0x300 – 0x3FF	RESERVED	
0x400	ETPU_C0CR_1 – eTPU 1 Channel 0 Configuration Register	*on page 22-661*
0x404	ETPU_C0SCR_1 – eTPU 1 Channel 0 Status and Control Register	*on page 22-663*
0x408	ETPU_C0HSRR_1 – eTPU 1 Channel 0 Host Service Request Register	*on page 22-666*
0x40C	RESERVED	
0x410	ETPU_C1CR_1 – eTPU 1 Channel 1 Configuration Register	*on page 22-661*
0x414	ETPU_C1SCR_1 – eTPU 1 Channel 1 Status and Control Register	*on page 22-663*
0x418	ETPU_C1HSRR_1 – eTPU 1 Channel 1 Host Service Request Register	*on page 22-666*

Table 348. Detailed memory map (continued)

Offset	Use	Location
0x41C	RESERVED	
	. .	
0x5F0	ETPU_C31CR_1 – eTPU 1 Channel 31 Configuration Register	*on page 22-661*
0x5F4	ETPU_C31SCR_1 – eTPU 1 Channel 31 Status and Control Register	*on page 22-663*
0x5F8	ETPU_C31HSRR_1 – eTPU 1 Channel 31 Host Service Request Register	*on page 22-666*
0x5FC – 0x7FF	RESERVED	
0x800	RESERVED	
0x804	RESERVED	
0x808	RESERVED	
0x80C	RESERVED	
0x810	RESERVED	
0x814	RESERVED	
0x818	RESERVED	
0x81C	RESERVED	
	. . .	
0x9F0	RESERVED	
0x9F4	RESERVED	
0x9F8	RESERVED	
0x9FC – 0x7FFF	RESERVED	
0x8000 – 0xBFFF(2)	Shared Parameter RAM – SPRAM	
0xC000 – 0xFFFF(2)	Shared Parameter RAM—SPRAM – PSE mirror(3)	
0x10000 – 1FFFF(4)	Shared Code Memory – SCM(5)	

1. This register is not implemented in some MCUs; see *Section , ETPU_SCMOFFDATAR – eTPU SCM Off-range Data Register*.
2. The actual SPRAM size is MCU-dependent.
3. Parameter Sign Extension access area, see *Section , Parameter access*
4. The actual SCM size is MCU-dependent. When the size not the maximum, the unused SCM address range returns the value of the register ETPU_SCMOFFDATAR.
5. SCM access is available only when bit VIS = 1 on register ETPU_MCR, under certain conditions (see *Section , ETPU_MCR – eTPU Module Configuration Register*). SCM can only be written in 32-bit accesses.

22.4.2 System configuration registers

ETPU_MCR – eTPU Module Configuration Register

This register is global to both eTPU engines, and resides in the Shared BIU. ETPU_MCR gathers global configuration and status in the eTPU system, including Global Exception. It is also used for configuring the SCM (Shared Code Memory) operation and test.

Figure 369. ETPU_MCR Register

Offset: eTPU_BASE + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	SDMERR	WDTOA	WDTOB	MGE 1	MGE 2	ILF1	ILF2	SCMERR	0	0	SCMSIZE				
W	GEC	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	SCMSIZE				

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	SCMMISC	SCMMISF	SCMMISEN	0	0	VIS	0	0	0	0	0	GTBE
W	▨	▨	▨	▨	SCMMISCC	▨		▨	▨		▨	▨	▨	▨	▨	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Table 349. ETPU_MCR field description

Field	Description
0	GEC—Global Exception Clear This write-only bit negates Global Exception request and clears Global Exception status bits MGE1, MGE2, ILF1, ILF2 and SCMMISF. 1: Negate Global Exception, clear status bits MGE1, MGE2, ILF1, ILF2 and SCMMISF 0: Keep Global Exception request and status bits MGE1, MGE2, ILF1, ILF2 and SCMMISF as is. GEC works the same way in Module Disable Mode.

Table 349. ETPU_MCR field description

Field	Description
1	SDMERR—SDM Read Error This flag indicates that an SDM read error occurred on a microengine read, generating a Global Exception. Errors from Host reads neither set this flag nor generate Global Exceptions. This bit is cleared by writing 1 to GEC. 1: Global Exception requested by SDM read error is pending. 0: No Global Exception pending because of SDM read error.
2	WDTOA—Watchdog Timeout Flag WDTOA indicates that a Watchdog Timeout occurred in eTPU engine A, generating a Global Exception. This bit is cleared by writing 1 to GEC. 1: Global Exception requested by Watchdog timeout 0: No Global Exception pending because of Watchdog timeout.
3	WDTOB—Watchdog Timeout Flag WDTOB indicates that a Watchdog Timeout occurred in eTPU engine B, generating a Global Exception. This bit is cleared by writing 1 to GEC. 1: Global Exception requested by Watchdog timeout 0: No Global Exception pending because of Watchdog timeout.
4	MGE1—Microcode Global Exception – Engine A This bit indicate that a Global Exception was asserted by microcode executed on eTPU engine A. The determination of the reason why the Global Exception was asserted is application dependent: it can be coded in an SPRAM status parameter, for instance. This bit is cleared by writing 1 to GEC. 1: Global Exception requested by microcode is pending 0: No microcode-requested Global Exception pending.
5	MGE2—Microcode Global Exception – Engine B This bit indicate that a Global Exception was asserted by microcode executed on eTPU engine B. The determination of the reason why the Global Exception was asserted is application dependent: it can be coded in an SPRAM status parameter, for instance. This bit is cleared by writing 1 to GEC. 1: Global Exception requested by microcode is pending 0: No microcode-requested Global Exception pending.
6	ILF1—Illegal Instruction Flag – eTPU A The ILF1 bit is set by the microengine to indicate that an illegal instruction was decoded in Engine A. This bit is cleared by host writing 1 to GEC. See *Section , Illegal Instructions*, for more details. 1: Illegal Instruction detected by eTPU A. 0: Illegal Instruction not detected.
7	ILF2—Illegal Instruction Flag – eTPU B The ILF2 bit is set by the microengine to indicate that an illegal instruction was decoded in Engine B. This bit is cleared by host writing 1 to GEC. See *Section , Illegal Instructions*, for more details. 1: Illegal Instruction detected by eTPU B. 0: Illegal Instruction not detected.

Table 349. ETPU_MCR field description

Field	Description
8	SCMERR—SCM Read Error This flag indicates that an SCM read error occurred on a microengine read, generating a Global Exception. Errors from Host reads neither set this flag nor generation Global Exceptions. This bit is cleared by writing 1 to GEC. 1: Global Exception requested by SCM read error is pending. 0: No Global Exception pending because of SCM read error.
9-10	Reserved
11-15	SCMSIZE[4:0]—SCM Size This read-only field holds the number of 2 Kbyte SCM Blocks minus 1. This bit is write protected when any of the engines are not halted or stopped(1). When VIS = 1, the ETPU_ECR MDIS bits are write protected, and only 32-bit aligned SCM writes are supported. The value written to SCM is unpredictable if other transfer sizes are used.
16-19	Reserved
20	SCMMISC, SCMMISCC—SCM MISC Complete, SCM MISC Complete Clear Flag SCMMISC indicates that MISC has completed the evaluation of the SCM signature since reset or the since the last time it was cleared. SCMMISC is cleared by writing 1 to SCMMISCC (at same bit position), and is not cleared when MISC is disabled (SCMMISEN = 0). SCMMISC asserts at the end of the SCM memory scan, either if the signature matches or not. 1: MISC completed at least one SCM signature calculation and compare since the last time SCMMISC was cleared. 0: MISC has not yet completed an SCM signature calculation and compare since the last time SCMMISC was cleared.writes are supported. The value written to SCM is unpredictable if other transfer sizes are used.
21	SCMMISF—SCM MISC Flag The SCMMISF bit is set by the SCM MISC (Multiple Input Signature Calculator) logic to indicate that the calculated signature does not match the expected value, at the end of a MISC iteration. See *Section 22.5.10, Test and Development Support*, for more details. 1: MISC has read entire SCM array and the expected signature in ETPU_MISCCMPR does not match the value calculated. 0: Signature mismatch not detected. This bit is cleared when Global Exception is cleared by writing 1 to GEC.

Table 349. ETPU_MCR field description

Field	Description
22	SCMMISEN—SCM MISC Enable The SCMMISEN bit is used for enabling/disabling the operation of the MISC logic. SCMMISEN is readable and writable at any time. The MISC logic will only operate when this bit is set to 1. When the bit is reset the MISC address counter is set to the initial SCM address. When enabled, the MISC will continuously cycle through the SCM addresses, reading each and calculating a CRC. In order to save power, the MISC can be disabled by clearing the SCMMISEN bit. See *Section 22.5.10, Test and Development Support*, for more details. 1: MISC operation enabled. 0: MISC operation disabled. The MISC logic is reset to its initial state. SCMMISEN resets automatically when MISC logic detects an error, i.e., when SCMMISF transitions from 0 to 1, disabling the MISC operation.
23-24	Reserved
25	VIS—SCM Visibility Bit VIS bit turns SCM visible to the IP-Bus and resets MISC state (but SCMMISEN keeps its value). 1: SCM is visible to the slave bus. MISC state is reset. 0: SCM is not visible to the slave bus. Accessing SCM address space issues a bus error, writes are protected and reads are meaningless. This bit is write protected when any of the engines are not halted or stopped(2). When VIS = 1, the ETPU_ECR MDIS bits are write protected, and only 32-bit aligned SCM writes are supported. The value written to SCM is unpredictable if other transfer sizes are used.
26-30	Reserved
31	GTBE–Global Time Base Enable GTBE enables time bases in both engines, allowing them to be started synchronously. 1: time bases in both engines are enabled to run. 0: time bases in both engines are disabled to run. Global Time Base Enable action may also depend on other blocks, as explained in *Section , GTBE – Global time base enable*. When GTBE is turned off with Angle Mode enabled, the EAC must be reinitialized before GTBE is turned on again. The EAC reinitialization procedure is described in *Section , Restarting angle logic*.

1. Engine is stopped in Module Disable or Stop Modes, but accesses to registers in Stop Mode is defined in the MCU level.
2. Engine is stopped in Module Disable or Stop Modes, but accesses to registers in Stop Mode is defined in the MCU level.

ETPU_CDCR – eTPU Coherent Dual-Parameter Controller Register

eTPU Shared Parameter RAM (SPRAM) can be accessed by the MCU’s processor core and the eTPU’s microengine(s) concurrently. In general, there is no guaranteed order by which a group of parameters is accessed, which may lead to a lack of internal consistency if two or more related parameters are read when only part of them is updated.

The eTPU provides mechanisms to guarantee parameter coherency, including the use of transfer service thread mechanism. and a mailbox (or “software semaphore”) mechanism.

A third mechanism, the Coherent Dual-parameter Controller (CDC), is also provided. It is used by the processor core to coherently transfer pairs of parameters between a parameter buffer located on SPRAM and locations on SPRAM where parameters are accessed directly by the channels. Coherency is guaranteed by SPRAM access arbitration. Although limited to two parameters only, it has low latency and wastes no microengine resources.

This register is used to configure and initiate CDC transfers between the parameter buffer area and the channel parameter area.

1. The host asserts the STS bit to start the data transfer.
2. CDC contends for the SPRAM and starts the transfer.
3. When the data transfer is complete, STS returns to 0. The host receives wait-states for writing STS = 1 while CDC contends for SPRAM and during the transfer.
4. The write access ends when CDC finishes the transfer. The host receives wait-states during the CDC transfer.

Note: If the host writes to the ETPU_CDCR with STS = 0 or does not write the STS bit, the CDC transfer does not occur.

CDC programming can be summarized as follows:

1. If it is a write transfer, i.e., from host to channel, write the two parameters into temporary area.
2. Write the ETPU_CDCR with STS = 1 and the remaining CDC programming parameters: parameter width (32 or 24 bits, field PWIDTH), transfer direction (read or write, field WR), temporary parameter area base address (field PBBASE), and the absolute addresses of the parameters to be transferred (concatenation of the fields CTBASE and PARAM0/1).
3. If it is a read transfer, i.e., from channel to host, read the two parameters from the temporary area into host memory/registers.

Offset: eTPU_Base + 0x004 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	STS	CTBASE					PBBASE									
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	PWIDTH	PARAM0							WR	PARAM1						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 370. ETPU_CDCR Register

Table 350. ETPU_CDCR field description

Field	Description
0	STS—Start Bit This bit is set by the host in order to start the data transfer between the parameter buffer pointed by PBBASE and the target addresses selected by the concatenation of fields CTBASE and PARAM0/1. The host receives wait-states until the data transfer is complete, when this bit is reset by coherency logic (see *Section , Coherent Dual-parameter Controller (CDC)*). Therefore, host always reads STS as 0. 1: (write) starts a coherent transfer. 0: (write) does not start a coherent transfer.
1-5	CTBASE[4:0]—Channel Transfer Base This field concatenates with fields PARAM0/PARAM1 to determine the absolute word offset (from the SPRAM base) of the parameters to be transferred: Parameter 0 word address = {CTBASE, PARAM0} + SPRAM base word address Parameter 1 word address = {CTBASE, PARAM1} + SPRAM base word address
6-15	PBBASE[9:0]—Parameter Buffer Base Address This field points to the base address of the parameter buffer location, with granularity of 2 parameters (8 bytes). The host (byte) address of the first parameter in the buffer is PBBASE*8 + SPRAM Base Byte Address. The microengine absolute (word) address of the first parameter in the buffer is PBBASE*2.
16	PWIDTH—Parameter Width Selection This bit selects the width of the parameters to be transferred between the PB and the target address. 1: Transfer 32-bit parameters. All 32 bits of the parameters are written in the destination address. 0: Transfer 24-bit parameters. The upper byte remains unchanged in the destination address.
17-23	PARAM0[6:0]—Channel Parameter number 0 This field, in concatenation with CTBASE[4:0], determines the word address offset (from the SPRAM base) of the parameters that are destination or source (defined by WR) of the coherent transfer. The word SPRAM address offset of the parameters are {CTBASE, PARAM0}.Note that PARAM0 and PARAM1 allow non-contiguous parameters to be transferred coherently. The parameter pointed by {CTBASE, PARAM0} is the first transferred.
24	WR—Read/Write selection This bit selects the direction of the coherent data transfer. 1: Write operation. Data transfer is from the PB to the selected parameter RAM address. 0: Read operation. Data transfer is from the selected parameter RAM address to the PB.
25-31	PARAM1[6:0]—Channel Parameter number 1 This field, in concatenation with CTBASE[4:0] determines the word address offset (from the SPRAM base) of the parameters that are destination or source (defined by WR) of the coherent transfer. The word SPRAM address offset of the parameters are {CTBASE, PARAM1}.Note that PARAM0 and PARAM1 allow non-contiguous parameters to be transferred coherently. The parameter pointed by {CTBASE, PARAM0} is the first transferred.

ETPU_MISCCMPR – eTPU MISC Compare Register

The eTPU includes a feature called the Multiple Input Signature Calculator (MISC) which comprises hardware that sequentially reads all Shared Code Memory (SCM) and calculates a 32-bit CRC signature. The ETPU_MISCCMPR stores the 32-bit expected value to be compared to the signature generated by the MISC.

The sequence is as follows:

1. The host loads the ETPU_MISCCMPR with the expected value to be found at the end of the MISC cycle
2. The host starts signature calculation by writing bit SCMMISEN = 1 in the ETPU_MCR. The MISC zeroes the signature accumulator and starts reading SCM data and calculating the signature.
3. After last SCM position is read, MISC compares the value in the signature accumulator against the value in the ETPU_MISCCMPR. If there is a mismatch, the MISC stops, issues a Global Exception and the SC MM I SF bit in the ETPU_MCR assumes value 1. If no mismatch is found, MISC repeats the procedure automatically.

Offset: eTPU_Base + 0x00C

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	ETPUMISCCMP[31:16]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	ETPUMISCCMP[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 371. ETPU_MISCCMPR Register

Table 351. ETPU_MISCCMPR field description

Field	Description
0-31	ETPUMISCCMP[31:0]—Expected Multiple Input Signature Register value See *Section , SCM Test – Multiple input signature calculator*.

ETPU_SCMOFFDATAR – eTPU SCM Off-range Data Register

When read accesses are made, either by the host or an eTPU2 microengine, to addresses above the limit corresponding to the SCMSIZE value in the ETPU_MCR, the value read comes from the ETPU_SCMOFFDATAR.

The host can program the register at initialization with an opcode value with operations that try to protect or recover the system from runaway code, for instance: terminate the thread, clear channel flags, disable match and transition service requests, issue an interrupt, or jump to an error recovery procedure. Writes to unimplemented addresses do not return an error and can write on unspecified mirror addresses, so they should be avoided.

The reset value is MCU dependent.

Offset: eTPU_Base + 0x010 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	ETPUSCMOFFDATA[31:16]															
Reset	etpu_scm_off_range_data_plug[31:16](1)															

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	ETPUSCMOFFDATA[15:0]															
Reset	etpu_scm_off_range_data_plug[15:0]															

▨ = Unimplemented or Reserved

1. The reset value depends on the MCU, and is usually 0xf3775ffb, an instruction that clears MRLEs, MRLs and TDLs, disables channel service requests, ends the thread and generates an illegal instruction Global Exception.

Figure 367. ETPU_SCMOFFDATAR Register

Table 352. ETPU_SCMOFFDATAR field description

Field	Description
0-31	ETPUSCMOFFDATA[31:0]—SCM Off-range read data value See *Section , SCM off-range data*.

ETPU_ECR – eTPU Engine Configuration Register

Each engine has its own ETPU_ECR. ETPU_ECR holds configuration and status fields that are programmed independently in each engine.

Offset: eTPU_A: eTPU_Base + 0x014; eTPU_B: eTPU_Base + 0x018 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FEND	MDIS	0	STF	0	0	0	0	HLTF	0	0	0	FCSS	FPSCK		
W			▨	▨	▨	▨	▨	▨	▨	▨	▨	▨				
Reset	0	0/1[1]	0	0	0	0	0	0	0(1[2])	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CDFC		0	ERBA					SPPDIS	0	0	ETB				
W			▨							▨	▨					
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

1. The MDIS reset value is MCU-dependent. Please consult the Reference Manual of the specific MCU.
2. Engine may go to Debug state (halted) soon after reset, depending on the NDEDI configuration.

Figure 367. ETPU_ECR Register

Table 353. ETPU_ECR field description

Field	Description
0	FEND—Force End FEND assertion terminates any current running thread as if an END instruction have been executed (see *Section , Ending current thread – END*). 1: Ends any ongoing thread. 0: Normal operation. This bit is self-negating when the thread ends. Writing FEND = 1 is ignored and FEND stays 0 when the microengine is in TST, halted, stopped, or idle (no thread executing). Only on rare occasions (e.g., during a long stall, see *Section , Microengine stall*) FEND can be read as 1, because it negates as soon as the end begins execution.

Table 353. ETPU_ECR field description

Field	Description
1	MDIS—Module Disable Bit When MDIS is set, the engine shuts down its internal clocks, going into Module Disable Mode. TCR1 and TCR2 cease to increment, and input sampling stops. The engine asserts the stop flag (STF) bit to indicate that it has stopped However, the BIU continues to run, and the Host can access all registers except for the channel registers (see list of channel registers on *Section 22.4.7, Channel configuration and control registers*). After MDIS is set, even before STF asserts, data read from the channel registers is not meaningful and writes are ineffective, issuing a Bus Error. When the MDIS bit is asserted while the microcode is executing, the eTPU will stop when the thread is complete. 1: Commands engine to stop its clocks. 0: eTPU engine runs. Stop completes on the next system clock after the stop condition is valid. The MDIS bit is write-protected when VIS = 1. The Timebase registers can still be read with MDIS = 1, but writes are ineffective and a Bus Error is issued. Global Channel Registers and SPRAM can be accessed normally. Once MDIS is switched from 1 to 0 or vice versa, it must not be written a different value until STF changes accordingly.
2	Reserved
3	STF—Stop Flag Bit The eTPU system is fully stopped after the eTPU engine asserts its stop flag (STF). In case of an IP-Bus stop, the eTPU acknowledges the stop only after any ongoing thread is complete and the eTPU engine has stopped. 1: Engine has stopped (after the local MDIS bit has been asserted, or after the IP-Bus stop line has been asserted). 0: Engine is operating. Summarizing engine stop conditions, which STF reflects: STF_1:= (after stop completed) MDIS_1 | device stop request STF_2:= (after stop completed) MDIS_2 | device stop request STF_1 and STF_2 mean STF bit from engine 1 and STF bit from engine 2 respectively.
4-7	Reserved

Table 353. ETPU_ECR field description

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>8</td><td>HLTF—Halt Mode Flag

If eTPU engine entered halt state, this flag is asserted. The flag remains asserted while the microengine is in halt state, even during a single-step or forced instruction execution. See <i>Section , Development support features</i>, for further details about entering Halt Mode.

1: eTPU engine is halted
0: eTPU engine is not halted.</td></tr>
<tr><td>9-11</td><td>Reserved</td></tr>
<tr><td>12</td><td>FCSS—Filter Clock Source Selection

Speeds up the filter clock source before the prescaler, allowing more input capture resolution at minimum prescaling.

1: use system clock as EDF clock source before prescaler
0: use system clock / 2 as EDF clock source before prescaler.

FCSS = 1 also makes the channel work on T2/T4 timing mode (see <i>Section , T2/T4 Channel Timing</i>).</td></tr>
<tr><td>13-15</td><td>FPSCK[2:0]—Filter Prescaler Clock Control

FPSCK controls the prescaling of the clocks used in digital filters for the channel input signals and TCRCLK input, as shown in <i>Table 354</i>. Filtering can be controlled independently by engine, but all input digital filters in the same engine have same clock prescaling. For more details see <i>Section , Filter Clock Prescaler</i>.

Table 354. Filter prescaler clock control
<table>
<tr><th>Filter control</th><th>Sample on system clock divided by:</th></tr>
<tr><td>000</td><td>2</td></tr>
<tr><td>001</td><td>4</td></tr>
<tr><td>010</td><td>8</td></tr>
<tr><td>011</td><td>16</td></tr>
<tr><td>100</td><td>32</td></tr>
<tr><td>101</td><td>64</td></tr>
<tr><td>110</td><td>128</td></tr>
<tr><td>111</td><td>256</td></tr>
</table>
A new value written to FPSCK only becomes effective when the filter prescaler finishes the current count.</td></tr>
</table>

Table 353. ETPU_ECR field description

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>16-17</td><td>CDFC[1:0]—Channel Digital Filter Control

These bits select a digital filtering mode for the channels when configured as inputs for improved noise immunity (refer to <i>Table 355</i>). The eTPU has three digital filtering modes for the channels which provide programmable trade-off between signal latency and noise immunity (see <i>Section , Enhanced Digital Filter – EDF</i>). Changing CDFC during eTPU normal input channel operation is not recommended since it changes the behavior of the transition detection logic while executing its operation.

Table 355. Channel digital filter control
<table>
<tr><th>CDFC</th><th>Selected digital filter</th></tr>
<tr><td>00</td><td>TPU2/3 Two Sample Mode: Using the filter clock which is the system clock divided by (2, 4, 8,.., 256) as a sampling clock (selected by FPSCK field in ETPU_ECR), comparing two consecutive samples which agree with each other sets the input signal state. This is the default reset state.</td></tr>
<tr><td>01</td><td>eTPU bypass mode: the input signal is taken unfiltered, also making the channels work on T2/T4 timing mode(1).</td></tr>
<tr><td>10</td><td>eTPU Three Sample Mode: Similar to the TPU2/3 two sample mode, but comparing three consecutive samples which agree with each other sets the input signal state.</td></tr>
<tr><td>11</td><td>eTPU Continuous Mode: Signal need to be stable for the whole filter clock period. This mode compares all the values at the rate of system clock (FCSS = 1) or system clock divided by two (FCSS = 0), between two consecutive filter clock pulses. Signal needs to be continuously stable for the entire period. If all the values agree with each other, input signal state is updated.</td></tr>
</table>
1. See <i>Section , T2/T4 Channel Timing</i></td></tr>
<tr><td>18</td><td>Reserved</td></tr>
<tr><td>19-23</td><td>ERBA—Engine Relative Base Address

This field value is concatenated with the AID instruction field in engine relative address mode to form the SPRAM address (see <i>Section , Engine relative addressing mode</i>).</td></tr>
<tr><td>24</td><td>SPPDIS—Schedule Priority Passing Disable

SPPDIS is used to disable the priority passing mechanism of the microengine scheduler (see <i>Section , Primary scheme – priority among channels on different levels</i>).

1: Scheduler priority passing mechanism disabled.
0: Scheduler priority passing mechanism enabled.

SPPDIS bit must not be changed while any channel is enabled.</td></tr>
<tr><td>25-26</td><td>Reserved</td></tr>
</table>

Table 353. ETPU_ECR field description

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>27-31</td><td>ETB[4:0]—Entry Table Base

The field determines the location of the microcode entry table for the eTPU functions in SCM (see <i>Section , Entry points</i>). <i>Table 356</i> shows the entry table base address options.

Table 356. Entry table base address options
<table>
<tr><th>ETB</th><th>Entry table base address for host address</th><th>Entry table base address for microcode address</th></tr>
<tr><td>00000</td><td>0x000</td><td>0x000</td></tr>
<tr><td>00001</td><td>0x800</td><td>0x200</td></tr>
<tr><td>00010</td><td>0x1000</td><td>0x400</td></tr>
<tr><td>.
.
.
.</td><td>.
.
.
.</td><td>.
.
.
.</td></tr>
<tr><td>11110</td><td>0xF000</td><td>0x3C00</td></tr>
<tr><td>11111</td><td>0xF800</td><td>0x3E00</td></tr>
</table>
</td></tr>
</table>

22.4.3 Time base registers

Each eTPU2 engine has two internally-generated time bases—Time Counter Registers—TCR1 and TCR2. They provide 24-bit time bases, shared by all 32 channels of their associated eTPU2 engine. The registers in the following sections control the configuration and visibility of the time bases. There is one of each of these registers for each eTPU engine.

Note: Writes to this register issue bus error and are ineffective when MDIS = 1. Reads are always allowed.

ETPU_TBCR – eTPU Time Base Configuration Register

This register configures several timebase options.

Offset: eTPU_A: eTPU_Base + 0x020 eTPU_B: eTPU_Base + 0x040 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TCR2CTL			TCRCF		AM		0	0	0	TCR2P					
W								▨	▨	▨						
Reset	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	TCR1CTL		TCR1CS	0	0	0	0	0	TCR1P							
W				▨	▨	▨	▨	▨								
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 372. ETPU_TBCR Register

Table 357. ETPU_TBCR field description

Field	Description
0-2	TCR2CTL[2:0]—TCR2 Clock/Gate Control These bits are part of the TCR2 clocking system (see *Section 22.5.6, Time Bases*). They determine the clock source for TCR2 before the prescaler. TCR2 can count on any detected edge of the TCRCLK signal or use it for gating system clock divided by 8. After reset - TCRCLK signal rising edge is selected. TCR2 can also be clocked by an internal peripheral timebase signal or system clock divided by 8. TCR2CTL also determines the TCRCLK edge selected for angle tooth detection in angle mode. See *Table 358*.

Table 358. TCR2 clock source

TCR2CTL	AM = 0	AM = 1
	TCR2 Clock before prescaler	Angle tooth detection
000	Gated DIV8 clock (system clock / 8). In this case, when the external TCRCLK signal is low, the DIV8 clock is blocked, preventing it from incrementing the TCR2 prescaler. When the external TCRCLK signal is high, TCR2 prescaler is incremented at the frequency of the system clock divided by 8.	do not use with AM = 1
001	Rise transition on TCRCLK signal increments the TCR2 prescaler.	rise edge
010	Fall transition on TCRCLK signal increments the TCR2 prescaler.	fall edge
011	Rise or Fall transition on TCRCLK signal increments the TCR2 prescaler.	both edges
100	DIV8 clock (system clock / 8)	do not use with AM = 1
101	Peripheral Timebase clock source	
110	do not use with AM = 0	no edge(1)
111	TCR2 frozen, except as STAC client	do not use with AM = 1

1. TCRCLK edges are not detected by the EAC logic, but they can still be detected by the channel 0 logic if AM = 01.

Table 357. ETPU_TBCR field description (continued)

Field	Description
3-4	TCRCF[1:0]—TCRCLK Signal Filter Control This field controls the TCRCLK digital filter (see *Section , TCRCLK digital filter*), determining whether the TCRCLK signal input (after a synchronizer) is filtered with the same filter clock as the channel input signals (see *Section , Enhanced Digital Filter – EDF*) or uses the system clock divided by 2, and also whether the TCRCLK digital filter works in integrator mode or two sample mode (see *Table 359*).
5-6	AM—Angle Mode Selection This field enables the Enhanced Angle Counter logic to generate angle information (see *Section 22.5.7, EAC – eTPU angle counter*), and also select the tooth signal input and the channel used to process it, as shown in *Table 360*. When EAC is not disabled by AM and neither TCR1 nor TCR2 are STAC Clients, the EAC (eTPU Angle Clock) hardware provides angle information to the channels using the TCR2 bus. When AM is reset (non-angle mode), the EAC operation is disabled, and its internal registers can be used as general purpose. For more information, see *Section 22.5.7, EAC – eTPU angle counter*.
7-9	Reserved

Table 359. TCRCLK filter clock/mode

TCRCF	Filter clock	Filter mode
00	system clock divided by 2	two sample
01	filter clock of the channels	two sample
10	system clock divided by 2	integrator
11	filter clock of the channels	integrator

Table 360. AM - angle mode selection

Value	TCR2 value	Tooth signal	Tooth processing channel
0 0	Timebase (EAC operation disabled)	not applicable	
0 1	Angle Ticks	TCRCLK input	0
1 0		channel 1 input	1
1 1		channel 2 input	2

If TCR1 or TCR2 is a STAC Bus Client (see *Section , STAC Interface*), the EAC operation is forbidden, and if AM is set the Angle Logic does not work properly.

Changing AM may cause spurious transition detections on the channel selected by AM, depending on the channel mode and state (see *Section , Transition Detection and Time Base Capture*). If AM must be changed with GTBE = 1, the recommended procedure is described in *Section , Restarting angle logic*.

Table 357. ETPU_TBCR field description (continued)

Field	Description
10-15	TCR2P[5:0]—Timer Count Register 2 Prescaler Control These bits are part of the TCR2 clocking system (see *Section 22.5.6, Time Bases*). TCR2 is clocked from the output of a prescaler. The prescaler divides its input by (TCR2P+1) allowing frequency divisions from 1 to 64. The prescaler input is the system clock divided by 8 (in gated or non-gated clock mode), or Internal Timebase input, or TCRCLK filtered input. This field has no effect on TCR2 in Angle Mode.
16-17	TCR1CTL—TCR1 Clock/Gate Control TCR1CTL is part of the TCR1 clocking system (see *Section 22.5.6, Time Bases*). It determines, together with TCR1CS, the clock source for TCR1. TCR1 can count on detected rising edge of the TCRCLK signal, a Peripheral Timebase source, system clock, or the system clock divided by 2 (see *Table 361*). After reset TCRCLK signal is selected
18	TCR1CS—TCR1 Clock Source TCR1CS provides the option to double the TCR1 incrementing speed, using system clock as its clock source instead of system clock / 2. 1: use system clock as TCR1 clock source before the prescaler; can only be set in specific combinations with TCR1CTL (see *Table 361*). 0: use system clock / 2 as TCR1 clock source before the prescaler, if that clock source is selected by TCR1CTL. TCR1CS = 1 also makes the channel work on T2/T4 timing mode (see *Section , T2/T4 Channel Timing*). The clock source of the EAC angle tick generator will still be an even division of system clock if TCR1CS = 1, obeying to the fields TCR1P as if TCR1CS = 0 (see *Section , Angle tick generator*). **Table 361. TCR1 clock source** (see table below)
19-23	Reserved

Table 361. TCR1 clock source

TCR1CTL	TCR1CS[1]	TCR1 Clock before prescaler
00	0	selects TCRCLK as clock source for the TCR1 prescaler[2]
01	0	selects Peripheral Timebase clock as source for the TCR1 prescaler
10	0	selects system clock divided by 2 as clock source for the TCR1 prescaler
10	1	selects system clock as clock source for the TCR1 prescaler
11	0	TCR1 frozen, except as a STAC client;

1. All other combinations of TCR1CTL and TCR1CS are reserved.
2. This selection must not be used in Angle Mode.

Table 357. ETPU_TBCR field description (continued)

Field	Description
24-31	TCR1P[7:0]—Timer Count Register 1 Prescaler Control TCR1 is clocked from the output of a prescaler. The input to the prescaler is the internal eTPU system clock divided by 2, system clock, or the output of TCRCLK filter, or Peripheral Timebase input. The prescaler divides this input by (TCR1P+1) allowing frequency divisions from 1 up to 256.

ETPU_TB1R – eTPU Time Base 1 (TCR1) Visibility Register

This register provides visibility of the TCR1 time base for host read access (see *Section 22.5.6, Time Bases*). This register is read-only. The value of the TCR1 time base shown can be driven by the TCR1 counter or imported from STAC bus, depending on the configuration set in ETPU_REDCR.

Offset: eTPU_A: eTPU_Base + 0x024; eTPU_B: eTPU_Base + 0x044 Access: User read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	TCR1[23:7]							
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	TCR1[8:0]															
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 373. ETPU_TB1R Register

Table 362. ETPU_TB1R field description

Field	Description
0-7	Reserved
8-31	TCR1[23:0]—TCR1 value TCR1 value used on matches and captures. See *Section 22.5.6, Time Bases*.

ETPU_TB2R – eTPU Time Base 2 (TCR2) Visibility Register

This register provides visibility of the TCR2 time base for host read access (see *Section 22.5.6, Time Bases*). This register is read-only. The value of the TCR2 time base shown can be driven by the TCR2 counter, the Angle Mode logic, or imported from STAC, depending on Angle Mode and STAC configurations set in registers ETPU_TBCR and ETPU_REDCR.

Offset: eTPU_A: eTPU_Base + 0x028; eTPU_B: eTPU_Base + 0x048 Access: User read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	TCR2[23:7]							
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	TCR2[8:0]															
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 374. ETPU_TB2R Register

Table 363. ETPU_TB2R field description

Field	Description
0-7	Reserved
8-31	TCR2[23:0]—TCR2 value TCR2 value used on matches and captures. See *Section 22.5.6, Time Bases*.

ETPU_REDCR – eTPU STAC Configuration Register

This register configures the eTPU STAC bus operation as a STAC Server/Client module (see *Section , STAC Interface*).

Offset: eTPU_A: eTPU_Base + 0x02C; eTPU_B: eTPU_Base + 0x04C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	REN 1	RSC 1	0	0	SERVER_ID1				0	0	0	0	SRV1			
W			▨	▨	▨	▨	▨	▨	▨	▨	▨	▨				
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	REN 2	RSC 2	0	0	SERVER_ID2				0	0	0	0	SRV2			
W			▨	▨	▨	▨	▨	▨	▨	▨	▨	▨				
Reset	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 375. ETPU_REDCR Register

Table 364. ETPU_REDCR field description

Field	Description
0	REN1—TCR1 Resource[1] Client/Server Operation Enable Bits This bit enables or disables Client/Server operation to eTPU STAC resources. REN1 enables TCR1 STAC bus operations. 1: Server/Client Operation for resource 1 is enabled. 0: Server/Client Operation for resource 1 is disabled.
1	RSC1—TCR1 Resource Server/Client Assignment Bits This bit selects the eTPU data resource assignment to be used as Servers or Clients. RSC1 selects the functionality of TCR1. For Server mode, external plugging determines the unique server address assigned to each TCR. For a Client mode, the SRV1 field determines the Server address to which the Client listens. 1: Resource Server operation. 0: Resource Client operation. When TCR1 is configured as a STAC Bus Client (REN2 = 1, RSC2 = 0) the eTPU Angle Clock hardware cannot be used. RSC1 must not be changed when the respective REN1 bit is asserted.
2-3	Reserved
4-7	SERVER_ID1—STAC Id 1 STAC Server Id (read-only plug values) used for TCR1 when STAC servers.

Table 364. ETPU_REDCR field description

Field	Description
12-15	SRV1—TCR1 Resource Server These bits select the address of the specific STAC Server to which the local TCR1 listens when configured as a STAC Client. SRV1 selects the STAC Server of TCR1.
16	REN2—TCR2 Resource[2] Client/Server Operation Enable Bits This bit enables or disables Client/Server operation to eTPU STAC resources. REN2 enables TCR2 STAC bus operations. 1: Server/Client Operation for resource 2 is enabled. 0: Server/Client Operation for resource 2 is disabled.
17	RSC2—TCR2 Resource Server/Client Assignment Bits This bit selects the eTPU data resource assignment to be used as Servers or Clients. RSC2 selects the functionality of TCR2. For Server mode, external plugging determines the unique server address assigned to each TCR. For a Client mode, the SRV2 field determines the Server address to which the Client listens. 1: Resource Server operation. 0: Resource Client operation. When TCR1 or TCR2 is configured as a STAC Bus Client (REN2 = 1, RSC2 = 0) the eTPU Angle Clock hardware cannot be used. RSC2 must not be changed when the respective REN1,2 bit is asserted.
18-19	Reserved
20-23	SERVER_ID2—STAC Id 2 STAC Server Id (read-only plug values) used for TCR2 when STAC servers.
24-27	Reserved
28-31	SRV2—TCR2 Resource Server These bits select the address of the specific STAC Server to which the local TCR2 listens when configured as a STAC Client. SRV2 selects the STAC Server of TCR2.

1. **resource** identifies any parameter that changes along the time and can be exported / imported from other device. In eTPU context, a resource can be TCR1 or TCR2 (either Time or Angle values).
2. **resource** identifies any parameter that changes along the time and can be exported / imported from other device. In eTPU context, a resource can be TCR1 or TCR2 (either Time or Angle values).

22.4.4 Engine related registers

This section gathers registers that are engine-related, other than ETPU_ECR (see *Section , ETPU_ECR – eTPU Engine Configuration Register*).

ETPU_WDTR – eTPU Watchdog Timer Register

This register configures the watchdog timer for the engine.

Offset: eTPU_A: eTPU_Base + 0x060; eTPU_B: eTPU_Base + 0x070 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	WDM		0	0	0	0	0	0	0	0	0	0	0	0	0	0
W			▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	WDCNT[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 376. ETPU_WDTR Register

Table 365. ETPU_WDTR field description

Field	Description
0-1	WDM—Watchdog Mode WDM selects the Watchdog operation mode, as shown below. For more information on the Watchdog operation, see *Section , Watchdog*. 00: disabled 01: reserved 10: thread length 11: busy length The watchdog must be disabled first before a new mode is configured.
2-15	Reserved

Table 365. ETPU_WDTR field description (continued)

Field	Description
16-31	WDCNT[15:0]—Watchdog Count This field indicates the maximum number of microcycles allowed for a thread (in thread length mode) or a sequence of threads (in busy length mode) before the current running thread is forced to end. For more information on Watchdog operation, see *Section , Watchdog*. The TST microcycles are also counted by the watchdog.

ETPU_IDLE – eTPU Idle Register

The Idle Counter Register (ETPU_IDLE) continuously counts microcycles in which the microengine is not busy with channel service. It can be used to measure the microengine utilization by rating the count measured during a period of time to the number of microcycles contained in the period. The Idle counter does not count microcycles when the engine is stopped, or is in TST or halt states.

Each eTPU2 engine has an associated ETPU_IDLE register.

Offset: eTPU_A: eTPU_Base + 0x068; eTPU_B: eTPU_Base + 0x078 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	IDLE_CNT[31:16]															
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	IDLE_CNT[15:0]															
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	ICLR
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 377. ETPU_IDLE Register

Table 366. ETPU_IDLE field description

Field	Description
0-31	IDLE_CNT[31:0]—Idle Count This is a freerunning count of the number of idle microcycles in the microengine. For more information on idle counter operation, see *Section , Idle Counter*.

Table 366. ETPU_IDLE field description (continued)

Field	Description
31	ICLR—Idle Clear This write-only bit is used to clear the idle count IDLE_CNT. 1: Clear the idle count IDLE_CNT 0: Do not clear idle count IDLE_CNT

22.4.5 Channel registers layout

The channel registers area is shown in *Figure 378* and detailed in the next sections for eTPU systems of 32 channels per engine. Reserved areas are placed to allow doubling the number of channels to 64 for each eTPU engine.

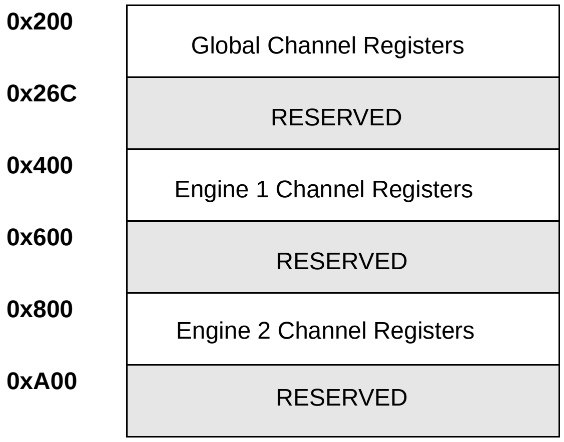


Figure 378. Channel registers area

22.4.6 Global channel registers

The registers in this section group, by type, the interrupt status and enable bits from all the channels. This organization eases management of all channels or groups of channels by a single interrupt handler routine. These bits, except the service and watchdog status, are mirrored in the individual channel registers, grouped by channel.

ETPU_CISR – eTPU Channel Interrupt Status Register

Host interrupt status (see *Section , Interrupts and data transfer requests*) from all channels are grouped in ETPU_CISR. Their bits are mirrored from the Channel Status/Control registers (see *Section 22.4.7, Channel configuration and control registers*) and Host must write 1 to clear a status bit.

Offset: eTPU_A: eTPU_Base + 0x200; eTPU_B: eTPU_Base + 0x204 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CIS31	CIS30	CIS29	CIS28	CIS27	CIS26	CIS25	CIS24	CIS23	CIS22	CIS21	CIS20	CIS19	CIS18	CIS17	CIS16
W	CIC31	CIC30	CIC29	CIC28	CIC27	CIC26	CIC25	CIC24	CIC23	CIC22	CIC21	CIC20	CIC19	CIC18	CIC17	CIC16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CIS15	CIS14	CIS13	CIS12	CIS11	CIS10	CIS9	CIS8	CIS7	CIS6	CIS5	CIS4	CIS3	CIS2	CIS1	CIS0
W	CIC15	CIC14	CIC13	CIC12	CIC11	CIC10	CIC9	CIC8	CIC7	CIC6	CIC5	CIC4	CIC3	CIC2	CIC1	CIC0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 379. ETPU_CISR Register

Table 367. ETPU_CISR field description

Field	Description
0-31	CISx—Channel x Interrupt Status 1: indicates that channel x has a pending interrupt to the Host CPU. 0: indicates that channel x has no pending interrupt to the Host CPU.
0-31	CICx—Channel x Interrupt Clear 1: clear interrupt status bit. 0: keep interrupt status bit unaltered. For details about interrupts see *Section , Channel interrupt and data transfer requests*.

ETPU_CDTRSR – eTPU Channel Data Transfer Request Status Register

Data Transfer request status (see *Section , Interrupts and data transfer requests*) from all channels are grouped in ETPU_CDTRSR. Their bits are mirrored from the Channel Status/Control registers (see *Section , ETPU_CxSCR – eTPU Channel x Status Control Register*).

Offset: eTPU_A: eTPU_Base + 0x210; eTPU_B: eTPU_Base + 0x214 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	DTRS31	DTRS30	DTRS29	DTRS28	DTRS27	DTRS26	DTRS25	DTRS24	DTRS23	DTRS22	DTRS21	DTRS20	DTRS19	DTRS18	DTRS17	DTRS16
W	DTRC31	DTRC30	DTRC29	DTRC28	DTRC27	DTRC26	DTRC25	DTRC24	DTRC23	DTRC22	DTRC21	DTRC20	DTRC19	DTRC18	DTRC17	DTRC16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	DTRS15	DTRS14	DTRS13	DTRS12	DTRS11	DTRS10	DTRS9	DTRS8	DTRS7	DTRS6	DTRS5	DTRS4	DTRS3	DTRS2	DTRS1	DTRS0
W	DTRC15	DTRC14	DTRC13	DTRC12	DTRC11	DTRC10	DTRC9	DTRC8	DTRC7	DTRC6	DTRC5	DTRC4	DTRC3	DTRC2	DTRC1	DTRC0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 380. ETPU_CDTRSR Register

Table 368. ETPU_CDTRSR field description

Field	Description
0-31	DTRSx—Channel x Data Transfer Request Status These bits mimic the corresponding ETPU DMA requests. DTRSx can be cleared by software (writing 1 to DTRCx) or by the assertion of corresponding DMA completion acknowledge line. 1: Indicates that channel x has a pending data transfer request. 0: Indicates that channel x has no pending data transfer request.
0-31	DTRCx—Channel x Data Transfer Request Clear 1: Clear status bit. 0: Keep status bit unaltered For details about interrupts see *Section , Channel interrupt and data transfer requests*.

ETPU_CIOSR – eTPU Channel Interrupt Overflow Status Register

Interrupt Overflow status (see *Section , Interrupts and data transfer requests*) from all channels is grouped in the ETPU_CIOSR. Their bits are mirrored from the Channel Status/Control registers (see *Section , ETPU_CxSCR – eTPU Channel x Status Control*

Register) and a write of ‘1’ clears a status bit.

Offset: eTPU_A: eTPU_Base + 0x220; eTPU_B: eTPU_Base + 0x224 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CIOS 31	CIOS 30	CIOS 29	CIOS 28	CIOS 27	CIOS 26	CIOS 25	CIOS 24	CIOS 23	CIOS 22	CIOS 21	CIOS 20	CIOS 19	CIOS 18	CIOS 17	CIOS 16
W	CIOC 31	CIOC 30	CIOC 29	CIOC 28	CIOC 27	CIOC 26	CIOC 25	CIOC 24	CIOC 23	CIOC 22	CIOC 21	CIOC 20	CIOC 19	CIOC 18	CIOC 17	CIOC 16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CIOS 15	CIOS 14	CIOS 13	CIOS 12	CIOS 11	CIOS 10	CIOS 9	CIOS 8	CIOS 7	CIOS 6	CIOS 5	CIOS 4	CIOS 3	CIOS 2	CIOS 1	CIOS 0
W	CIOC 15	CIOC 14	CIOC 13	CIOC 12	CIOC 11	CIOC 10	CIOC 9	CIOC 8	CIOC 7	CIOC 6	CIOC 5	CIOC 4	CIOC 3	CIOC 2	CIOC 1	CIOC 0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 381. ETPU_CIOSR Register

Table 369. ETPU_CIOSR field description

Field	Description
0-31	CIOSx—Channel x Interrupt Overflow Status 1: indicates that interrupt overflow occurred in the channel. 0: indicates that no interrupt overflow occurred in the channel.
0-31	CIOCx—Channel x Interrupt Overflow Clear 1: clear status bit. 0: keep status bit unaltered. For details about interrupt overflow, see *Section , Interrupt and data transfer request overflow*.

ETPU_CDTROSR – eTPU Channel Data Transfer Request Overflow Status Register

Data Transfer Request Overflow status (see *Section , Interrupts and data transfer requests*) from all channels is grouped in the ETPU_CDTROSR. Their bits are mirrored from the Channel Status/Control registers (see *Section , ETPU_CxSCR – eTPU Channel x Status Control Register*) and a write of ‘1’ clears a status bit.

Offset: eTPU_A: eTPU_Base + 0x230; eTPU_B: eTPU_Base + 0x234 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	DTROS31	DTROS30	DTROS29	DTROS28	DTROS27	DTROS26	DTROS25	DTROS24	DTROS23	DTROS22	DTROS21	DTROS20	DTROS19	DTROS18	DTROS17	DTROS16
W	DTROC31	DTROC30	DTROC29	DTROC28	DTROC27	DTROC26	DTROC25	DTROC24	DTROC23	DTROC22	DTROC21	DTROC20	DTROC19	DTROC18	DTROC17	DTROC16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	DTROS15	DTROS14	DTROS13	DTROS12	DTROS11	DTROS10	DTROS9	DTROS8	DTROS7	DTROS6	DTROS5	DTROS4	DTROS3	DTROS2	DTROS1	DTROS0
W	DTROC15	DTROC14	DTROC13	DTROC12	DTROC11	DTROC10	DTROC9	DTROC8	DTROC7	DTROC6	DTROC5	DTROC4	DTROC3	DTROC2	DTROC1	DTROC0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

░ = Unimplemented or Reserved

Figure 382. ETPU_CDTROSR Register

Table 370. ETPU_CDTROSR field description

Field	Description
0-31	DTROSx—Channel x Data Transfer Request Overflow Status 1: indicates that data transfer request overflow occurred in the channel. 0: indicates that no data transfer request overflow occurred in the channel.
0-31	DTROCx—Channel x Data Transfer Request Overflow Clear 1: clear status bit. 0: keep status bit unaltered. For details about data transfer request overflow, see *Section , Interrupt and data transfer request overflow*.

ETPU_CIER – eTPU Channel Interrupt Enable Register

Host interrupt enable (see *Section , Interrupts and data transfer requests*) from all channels are grouped in ETPU_CIER. Their bits are mirrored from the Channel Configuration registers (see *Section , ETPU_CxCR – eTPU Channel x Configuration Register*).

Offset: eTPU_A: eTPU_Base + 0x240; eTPU_B: eTPU_Base + 0x244 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CIE 31	CIE 30	CIE 29	CIE 28	CIE 27	CIE 26	CIE 25	CIE 24	CIE 23	CIE 22	CIE 21	CIE 20	CIE 19	CIE 18	CIE 17	CIE 16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CIE 15	CIE 14	CIE 13	CIE 12	CIE 11	CIE 10	CIE 9	CIE 8	CIE 7	CIE 6	CIE 5	CIE 4	CIE 3	CIE 2	CIE 1	CIE 0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 383. ETPU_CIER Register

Table 371. ETPU_CIER field description

Field	Description
0-31	CIEx—Channel x Interrupt Enable 1: interrupt enabled for channel x 0: interrupt disabled for channel x. For details about interrupts see *Section , Channel interrupt and data transfer requests*.

ETPU_CDTRER – eTPU Channel Data Transfer Request Enable Register

Data Transfer request enable (see *Section , Interrupts and data transfer requests*) from all channels are grouped in ETPU_CDTRER. These bits are mirrored from the Channel Configuration registers (see *Section , ETPU_CxCR – eTPU Channel x Configuration Register*).

Offset: eTPU_A: eTPU_Base + 0x250; eTPU_B: eTPU_Base + 0x254 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	DTRE 31	DTRE 30	DTRE 29	DTRE 28	DTRE 27	DTRE 26	DTRE 25	DTRE 24	DTRE 23	DTRE 22	DTRE 21	DTRE 20	DTRE 19	DTRE 18	DTRE 17	DTRE 16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	DTRE 15	DTRE 14	DTRE 13	DTRE 12	DTRE 11	DTRE 10	DTRE 9	DTRE 8	DTRE 7	DTRE 6	DTRE 5	DTRE 4	DTRE 3	DTRE 2	DTRE 1	DTRE 0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

░ = Unimplemented or Reserved

Figure 384. ETPU_CDTRER Register

Table 372. ETPU_CDTRER field description

Field	Description
0-31	DTREx—Channel x Data Transfer Request Enable 1: Data Transfer request enabled for channel x. 0: Data Transfer request disabled for channel x. For details about interrupts see *Section , Channel interrupt and data transfer requests*.

ETPU_CPSSR – eTPU Channel Pending Service Status Register

ETPU_CPSSR is a read-only register that holds the status of the pending Channel Service Requests (see *Section 22.5.1, Functions and threads*).

Offset: eTPU_A: eTPU_Base + 0x280; eTPU_B: eTPU_Base + 0x284 Access: User read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SR31	SR30	SR29	SR28	SR27	SR26	SR25	SR24	SR23	SR22	SR21	SR20	SR19	SR18	SR17	SR16
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SR15	SR14	SR13	SR12	SR11	SR10	SR9	SR8	SR7	SR6	SR5	SR4	SR3	SR2	SR1	SR0
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 385. ETPU_CPSSR Register

Table 373. ETPU_CPSSR field description

Field	Description
0-31	SRx—Pending Service Request x Indicates a pending Service Request for channel x. 1: pending Service Request for channel x 0: no Service Request pending for channel x Pending SR status is a logic OR of all service requests pending: if only HSR is active, SRx clears only at the end of the thread. SRx clear due to the other request sources is microcode dependent. The pending service status bit for a channel is 1 when a Service Request is pending, even if the Channel is disabled (CPRx = 0). There can be a delay of one clock between writing HSR > 0 in register ETPU_CxHSRR of a channel and its respective bit being asserted in ETPU_CPSSR.

ETPU_CSSR – eTPU Channel Service Status Register

ETPU_CSSR holds the current channel service status on whether it is being serviced or not (see *Section 22.5.1, Functions and threads*). Only one bit may be asserted in this register at a given time. When no channel is being serviced the register read value is 0x00000000. ETPU_CSSR is a read-only register. The register can be read during normal eTPU operation for monitoring the scheduler activity.

Note: *Channel Service Status does not always reflect decoding of the CHAN register, since the later can be changed by the service thread microcode.*

Offset: eTPU_A: eTPU_Base + 0x290; eTPU_B: eTPU_Base + 0x294 Access: User read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SS31	SS30	SS29	SS28	SS27	SS26	SS25	SS24	SS23	SS22	SS21	SS20	SS19	SS18	SS17	SS16
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SS15	SS14	SS13	SS12	SS11	SS10	SS9	SS8	SS7	SS6	SS5	SS4	SS3	SS2	SS1	SS0
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 386. ETPU_CSSR Register

Table 374. ETPU_CSSR field description

Field	Description
0-31	SSx—Service Status x Indicates that channel x is currently being serviced. It is updated at the 1st microcycle of a Time Slot Transition (see *Section , Time slot transition*), or when the microengine ends the thread. 1: channel x is currently being serviced 0: channel x is not currently being serviced

22.4.7 Channel configuration and control registers

Each channel has a group of three registers used to control, configure and check status of that channel as shown in *Table 375*. This organization eases individual channel management.

Note: 1.A bus error is issued on read or write accesses to these registers when ETPU_ECR bit MDIS = 1. Writes are ineffective on bus error.

2. The SIU_ISEL8 Register is used to multiplex the eTPU[24:29] inputs. When SIU_SEL8 is in its default state eTPU channels 24–29 will not be connected to their respective output pin, irrespective of the SIU_PCR[PA] field. See *Section 15.6.22, IMUX Select Register 8 (SIU_ISEL8)*.

Table 375. Channel registers structure

Channel offset	Register name
0x00	ETPU_CxCR – eTPU Channel Configuration Register
0x04	ETPU_CxSCR – eTPU Channel Status/Control Register
0x08	ETPU_CxHSRR – eTPU Channel Host Service Request Register
0x0C	RESERVED

One contiguous area is used to map all channel registers of each eTPU engine as shown in *Table 376*.

Table 376. Channel registers map

Offset	Registers structure
0x400	eTPU 1 Channel 0 Registers Structure
0x410	eTPU 1 Channel 1 Registers Structure
0x420	eTPU 1 Channel 2 Registers Structure
0x430	. .
0x5E0	eTPU 1 Channel 30 Registers Structure
0x5F0	eTPU 1 Channel 31 Registers Structure
0x600	RESERVED
0x800	eTPU 2 Channel 0 Registers Structure
0x810	eTPU 2 Channel 1 Registers Structure
0x820	. .
0x9E0	eTPU 2 Channel 30 Registers Structure
0x9F0	eTPU 2 Channel 31 Registers Structure
0xA00	RESERVED

There are 64 structures defined, one for each available channel in the eTPU System (32 for each engine). The base address for the structure presented can be calculated by using the following equation:

*Channel_Register_Base = ETPU_Engine_Channel_Base + (channel_number * 0x10)*

where:

ETPU_Engine_Channel_Base = ETPU_Base + 0x400 for Engine 1

ETPU_Engine_Channel_Base = ETPU_Base + 0x800 for Engine 2

ETPU_CxCR – eTPU Channel x Configuration Register

ETPU_CxCR gathers configurations set individually per channel.

Offset: Channel_Register_Base + 0x0 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CIE	DTRE	CPR		0	0	ETPD	ETCS	0	0	0	CFS				
W					▨	▨			▨	▨	▨					
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	ODIS	OPOL	0	0	0	CPBA										
W			▨	▨	▨											
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 387. ETPU_CxCR Register

Table 377. ETPU_CxCR field description

Field	Description
31	CIE—Channel Interrupt Enable (This bit is mirrored from ETPU_CIER – see *Section , ETPU_CIER – eTPU Channel Interrupt Enable Register*.) 1: Enable interrupt for this channel. 0: Disable interrupt for this channel. See *Section , Channel interrupt and data transfer requests*.

Table 377. ETPU_CxCR field description

Field	Description
30	DTRE—Channel Data Transfer Request Enable (This bit is mirrored from ETPU_CDTRER – see *Section , ETPU_CDTRER – eTPU Channel Data Transfer Request Enable Register*.) 1: Enable data transfer request for this channel. 0: Disable data transfer request for this channel. See *Section , Channel interrupt and data transfer requests*.
2-3	CPR[1:0]—Channel Priority This field defines the priority level for the channel, used by the Hardware Scheduler (see *Section 22.5.3, Scheduler*). 00: Disabled 01: Low 10: Middle 11: High
4-5	Reserved
6	ETPD—Entry Table Pin Direction This bit selects which channel signal, input or output, is used in the Entry Point selection. The ETPD value has to be compatible with the function chosen for the channel, selected in the field CFS. For details about Entry Table and condition encoding schemes, refer to *Section , Entry points*. 1: use PSTO for Entry Point selection. 0: use PSTI for Entry Point selection.
7	ETCS—Entry Table Condition Select This bit determines the channel condition encoding scheme that selects, according to channel conditions, the Entry Point to be taken in an Entry Table. ETCS value has to be compatible with the function chosen for the channel, selected in field CFS. Two condition encoding schemes are available. For details about Entry Table and condition encoding schemes, refer to *Section , Entry points*. 1: select Alternate Entry Table Condition encoding scheme. 0: select Standard Entry Table Condition encoding scheme. The fields ETCS, CFS and CPBA must only be changed while the channel is disabled (field CPR = 00).
8-10	Reserved

Table 377. ETPU_CxCR field description

Field	Description
11-15	CFS[4:0]—Channel Function Select This field defines the function to be performed by the channel (see *Section 22.5.1, Functions and threads*). The Function assigned to the channel has to be compatible with the channel condition encoding scheme, selected by field ETCS. The fields ETCS, CFS and CPBA must only be changed while the channel is disabled (field CPR = 00).
16	ODIS—Output Disable This bit enables the channel to have its output forced to the value opposite to OPOL when the output disable input signal corresponding to the channel group that it belongs is active. See *Section , ipp_ind_etpu_odis_[1|2]([0 – 3]) eTPU Channel Output Disable Signals* and *Figure 403*. 1: turns on the output disable feature for the channel 0: turns off the output disable feature for the channel.
17	OPOL—Output Polarity Determines the output signal polarity. The activation of the output disable signal forces, when enabled by the ODIS bit, the channel output signal to the opposite of this polarity (see *Figure 403*). 1: output active high (output disable drives output to low) 0: output active low (output disable drives output to high)
18-20	Reserved
21-31	CPBA[10:0]—Channel x Parameter Base Address The value of this field times 8 specifies the SPRAM parameter base host (byte) address for channel x (2-parameter granularity; see *Section , SPRAM organization*). As seen by the Host, the channel parameter base (byte) address is: without parameter sign extension: ETPU_Base + 0x8000 + CPBA*8 with parameter sign extension: ETPU_Base + 0xC000 + CPBA*8 The fields ETCS, CFS and CPBA must only be changed while the channel is disabled (field CPR = 00).

ETPU_CxSCR – eTPU Channel x Status Control Register

ETPU_CxSCR gathers the interrupt status bits of the channel, and also the Function Mode definition (read-write). Bits CIS, CIOS and DTRS for each channel can be also accessed from ETPU_CISR, ETPU_CIOSR and ETPU_CDTRSR registers respectively (see *Section 22.4.6, Global channel registers*). Host must write 1 to clear a status bit.

Offset: Channel_Register_Base + 0x4 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CIS	CIOS	0	0	0	0	0	0	DTRS	DTROS	0	0	0	0	0	0
W	CIC	CIOC	▨	▨	▨	▨	▨	▨	DTRC	DTROC	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	IPS	OPS	OBE	0	0	0	0	0	0	0	0	0	0	0	FM	
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨		
Reset	0/1[1]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

1. The IPS value after reset is MCU dependent

Figure 388. ETPU_CxSCR Register

Table 378. ETPU_CxSCR field description

Field	Description
31	CIS—Channel Interrupt Status 1: channel has a pending interrupt to the Host CPU. 0: channel has no pending interrupt to the Host CPU.
31	CIC—Channel Interrupt Clear 1: clear interrupt status bit. 0: keep interrupt status bit unaltered. These bits are mirrored in ETPU_CISR – see *Section , ETPU_CISR – eTPU Channel Interrupt Status Register*. See also *Section , Channel interrupt and data transfer requests*.
30	CIOS—Channel Interrupt Overflow Status 1: interrupt overflow asserted for this channel 0: interrupt overflow negated for this channel
30	CIOC—Channel Interrupt Overflow Clear 1: clear status bit. 0: keep status bit unaltered. These bits are mirrored in ETPU_CIOSR – see *Section , ETPU_CIOSR – eTPU Channel Interrupt Overflow Status Register*. See also *Section , Interrupt and data transfer request overflow*.

Table 378. ETPU_CxSCR field description

Field	Description
2-7	Reserved
8	DTRS—Data Transfer Request Status 1: Channel has a pending data transfer request. 0: Channel has no pending data transfer request.
8	DTRC—Data Transfer Request Clear 1: clear status bit. 0: keep status bit unaltered These bits are mirrored in ETPU_CISR – see *Section , ETPU_CDTRSR – eTPU Channel Data Transfer Request Status Register*. See also *Section , Channel interrupt and data transfer requests*.
9	DTROS—Data Transfer Request Overflow Status 1: data transfer request overflow asserted for this channel data transfer request overflow negated for this channel
9	DTROC—Data Transfer Request Overflow Clear 1: clear status bit. keep status bit unaltered. These bits are mirrored in ETPU_CDTROSR – see *Section , ETPU_CDTROSR – eTPU Channel Data Transfer Request Overflow Status Register*. See also *Section , Interrupt and data transfer request overflow*.
10-15	Reserved
16	IPS—Channel Input Pin State This bit shows the current value of the filtered channel input signal state
17	OPS—Channel Output Pin State This bit shows the current value driven in the channel output signal, including the effect of the external output disable feature (see *Section , ipp_ind_etpu_odis_[1|2]([0 – 3]) eTPU Channel Output Disable Signals*. If the channel input and output signals are connected to the same pad, OPS reflects the value driven to the pad (if OBE = 1). This is not necessarily the actual pad value, which drives the value in the bit IPS.
18	OBE—Output Buffer Enable This bit shows the state of the channel output buffer enable signal, controlled by microcode.
19-29	Reserved

Table 378. ETPU_CxSCR field description

Field	Description
30-31	FM[1:0]—Channel Function Mode[1] Each function uses this field for specific configuration. These bits can be tested by microengine code (see *Section , Conditional/Unconditional branch*).

1. These bits are equivalent to the TPU/TPU2/TPU3 Host Sequence (HSQ) bits.

ETPU_CxHSRR – eTPU Channel x Host Service Request Register

ETPU_CxHSRR is used by the Host to issue service requests to the channel.

Offset: Channel_Register_Base + 0x8 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	HSR		
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 389. ETPU_CxHSRR Register

Table 379. ETPU_CxHSRR field description

Field	Description
0-28	Reserved
29-31	HSR[2:0]—Host Service Request This field is used by the Host CPU to request service to the channel (see *Section , Host service requests*). HSR = 000: no Host Service Request pending HSR > 000: function-dependent Host Service Request pending. HSR value turns to 000 automatically at the end of microengine service for that channel, but only if the thread started due to an HSR. Host should write HSR > 0 only when HSR = 0. Writing HSR = 000 withdraws a pending request if scheduler did not begin to resolve the Entry Point yet, but it does not abort the service thread from that point on. For more details, see *Section , Entry points*, and *Section , Host service requests*.

22.5 Functional description

22.5.1 Functions and threads

eTPU processing is event-driven, in the sense that eTPU microcode only runs to service a request from an event. Service Requests may result from the occurrence of any of the following events:

- Host CPU writing a non-zero value to the channel HSR (Host Service Request) field in ETPU_CxHSR.
- occurrence of a time base match, an input signal transition, or a specific combination of them (depending on the Channel Mode currently configured).
- a Link Service Request.

A given event is always associated to only one Channel:

- There is one HSR register field for each Channel
- Each signal is associated with only one Channel, which has its own Match registers and independent mode configuration.
- Each Link Service Request can have only one Channel as a target.

Service Request processing is done by a set of microengine routines. A set of related routines that implement a specific channel application is called a **Function**. One or more Functions reside on SCM, limited only by the SCM space available, size of microcode Functions and the number of entry points available. Each engine can be controlled by up to 32 Functions at a time.

A Function can be assigned to several channels, but only one Function can be assigned to a given Channel at a time. This is defined by the Host through the Channel Configuration Registers (see *Section 22.4.7, Channel configuration and control registers*).

The term **Thread** will be used hereafter to refer to a service routine of a Function, or its execution. A Thread is constructed of a specific number of microinstructions, typically the code necessary to calculate the next phase of waveform to be input to, or output from, a given channel. Once a Thread begins, its execution cannot be interrupted. A Thread normally finishes when an END microinstruction is executed.

A given Thread is selected and called by the Scheduler depending on the following:

- the type of event that generated the service request.
- the Function assigned to the target channel.
- target channel pin state.
- the state of the channel logic.
- the priority assigned to the target channel, relative to the priorities of other channels with pending service requests

The mechanism to select a thread based on the channel Function and type of event is described in the *Section , Entry points*.

The priority mechanism that determines the order of Thread execution amongst pending service requests is described in *Section 22.5.3, Scheduler*.

Entry points

Entry table

Each Thread has its **Entry Point**, which contains the SCM address of its first instruction, besides other information. For a complete Entry Point description, see *Section , Entry point format*.

Once the Scheduler chooses a channel among pending Service Requests, the Entry Point is taken from an **Entry Table**, based on the Function assigned for the channel and other conditions. Entry Table layout is shown in *Figure 390*.

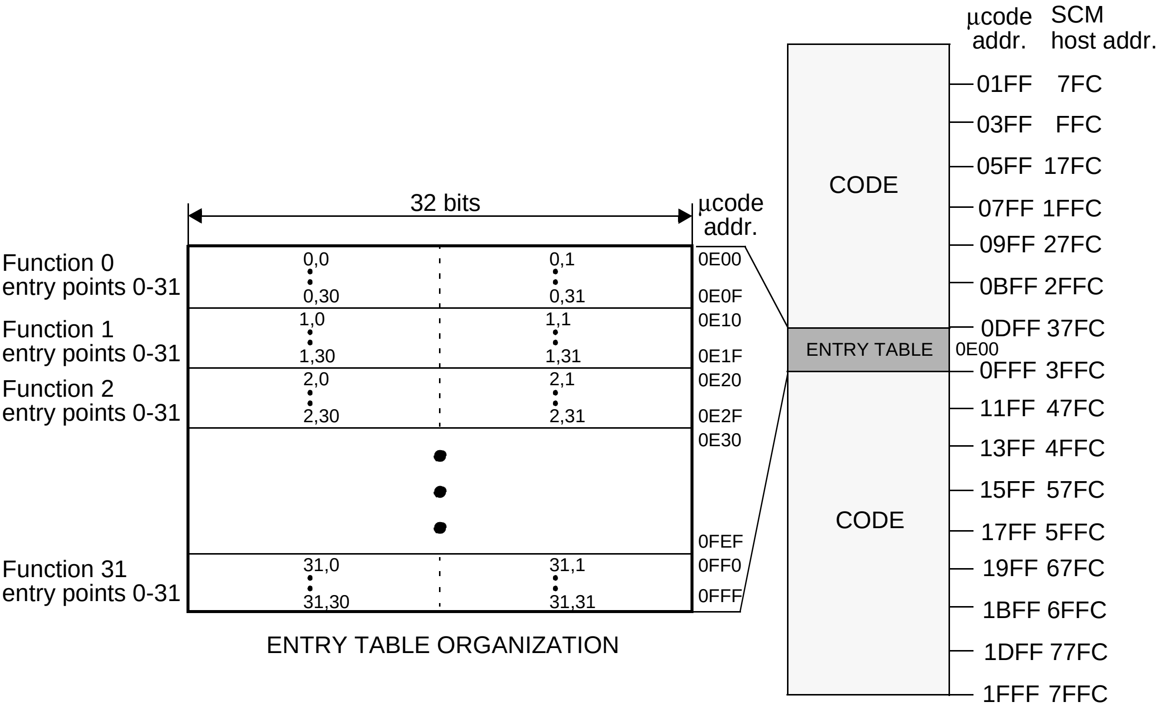


Figure 390. Entry Table

The Entry Table is organized by Functions. Each Function can have up to 32 Entry Points of 16 bits each, corresponding to 32 possible Threads per Function. Each Entry Point location in the table corresponds to a combination of events and channel states (see *Section , Entry point address generation*). A single Thread can be associated to more than one combination, having its Entry Point repeated in the table. Each 32-bit word in the Entry Table holds two Entry Points.

Note that the Entry Table can be placed in any SCM address multiple of the Entry Table size, determined by the field ETB[4:0] in the register ETPU_ECR. However, it is recommended to place the Entry Table at the start of the SCM to get continuous code memory and to ease the eventual migration of the code from larger parts down to smaller ones without rearranging the binary image, but this is not a restriction. Unused Entry Points may be used for microcode, so this organization extends the microcode continuous area to the unused area of the Entry Table. For this purpose, Function numbers should be selected from 0 up to 31. If, for example, only 8 functions are implemented, only the Entry Table locations for Functions 0 to 7 are used, and the Entry Table locations for functions 8 to 31 can be used as microinstruction memory (adding extra continuous 1536 bytes for microprogram usage).

One way of implementing different sets of Functions is having more than one Entry Table, and configuring the eTPU with the appropriate one for the application by changing field ETPU_ECR[ETB]. Note that the engines can use different Entry Tables, with or without the same set of Functions.

Entry point address generation

The Entry Point address within the Entry Table is determined by the Function assigned to the Channel, the state of the Channel, the type of event, and the condition encoding scheme. Together with the Entry Table base address, they form the Entry Point Address at the SCM, as shown in *Figure 391*.

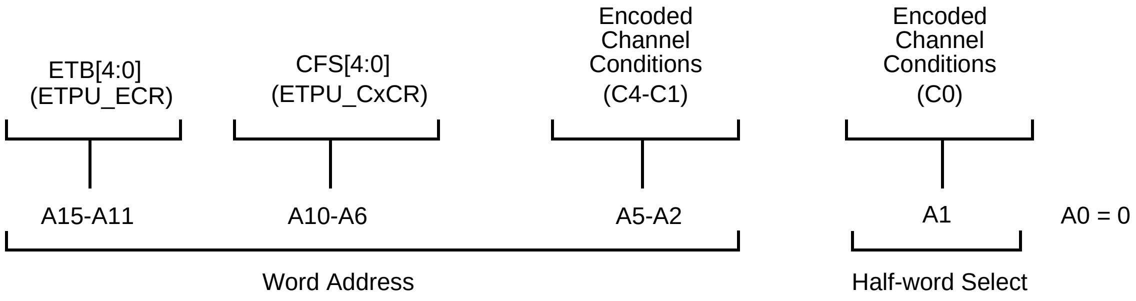


Figure 391. Entry Point Address (host address offset)

The type of event and channel state are coded in the Encoded Channel Conditions field C[4:0], according to one of two encoding schemes:

- Standard Entry Table Condition encoding scheme, shown in *Table 380*, which privileges Host Service Requests.
- Alternate Entry Table Condition encoding scheme, shown in *Table 381*, which focus on other events and state decoding.

The events that take part on condition encoding generate a Service Request, and have four origins:

1. Match Recognition (caused by greater/equal match, or equal-only, between the value TCR1/2 and the value stored in the channel match registers). eTPU channels support single and double match in various modes of match recognition; see *Section , Match Recognition*.
2. Transition Detect Service Request (channel input signal transition detection of a selected edge). The eTPU channels support single and double transition, which

together with the double match options provide various modes of transition detection; see *Section , Transition Detection and Time Base Capture*.

3. Channel Linking Service Request (microcode writing the channel number to the LINK register). Link service request allows one channel to activate another (see *Section , Channel Link*).
4. Host Service Request (Host writes a non-zero value to the HSR bits of the channel; see *Section , Host service requests*).

Note: Even if a Transition or Match Service Request is inhibited (by channel mode/state or SRI), the Transition Detection and Match Recognition are taken into account for condition encoding. That is, the MRLA/B and TDLA/B flags are used, not their respective Service Requests.

Columns Host Request Bits, Link Request, MatchA/TransB, and MatchB/TransA determine the type of event. A non-zero value in these columns represents the recognition of the event, while “x” indicates that its recognition is irrelevant. Values 1 and 0 mean that event was recognized or not, respectively. Note that Match and Transition events may occur and not be recognized, and in this case it assumes value 0 for the condition encoding. The recognition of such an occurred event depends on the channel mode assigned and other conditions, as described in *Section 22.5.5, Enhanced Channels*.

The **Host Service Request Bits** column refers to the value written by the Host CPU to the Host Service Request Register (ETPU_CxHSRR) of the Channel being serviced. Note that the bits on this row are coded (3-bit representation). If the value of HSR is not zero, then the Host actually requested service.

The **Link Request** column refers to the occurrence of a Channel Link request.

The **MatchA/TransB** column refers to the recognition of either a Match event specified by MatchA channel register or the detection of a channel input signal event specified by the IPACB configuration register (see *Section , Pin Control Registers*).

The **MatchB/TransA** column refers to the recognition of either a Match event specified by MatchB channel register or the detection of a channel input signal event specified by the IPACA configuration register (see *Section , Pin Control Registers*).

For the channel input signal, MatchA and MatchB provide double timeout conditions which depend on the channel mode programming (see *Section , Channel Modes*). If the channel is used for output only, there are no transition detections, so the MatchB/TransA column represents only Match B, and MatchA/TransB column the Match A. In this case Match A and Match B are separated to give better state resolution in double match output functions. For more information about channel requests refer to *Section 22.5.5, Enhanced Channels*.

Besides those events, the following channel state conditions help to determine the Entry Point:

1. Channel Flags 0 and 1: these are channel-internal flags (not in SPRAM) associated with a channel. Their values are set by microcode (see *Section , Channel flags operations*).
2. Input Pin state or Output Flip Flop: the state (0 or 1) of the channel input signal after the Enhanced Filter (see *Section , Enhanced Digital Filter – EDF*), or the state driven to the output signal. Which one (input or output) is used is selected by the ETPU_CxCR bit ETPD.

The two Entry Table Condition encoding schemes combine events and state conditions differently, as detailed in the following sections.

Standard condition encoding scheme

In this scheme, shown in *Table 380*, all seven HSR combinations are used and other event type columns are marked “x” when HSR is non-zero, indicating that Host Service Request has priority over any other type of event. However, when an HSR service thread is called (entry numbers 0 to 9), other events may also have been recognized, and it is microcode responsibility to check them.

When HSR is 0, i.e., Host did not issue a Service Request to the channel, the other event conditions, the input signal state and channel flags determine the Entry Point. Note that channel flag 1 does not influence the encoding in this scheme.

Table 380. Standard channel condition encoding scheme

No.	Encoded channel conditions [C4-C0]	Host service request bits	Link request	MatchA / TransB	Match.2 / TransA	In/Output pin state[1]	Channel flag1	Channel flag0
0	00000	001	x	x	x	0	x	0
1	00001	001	x	x	x	0	x	1
2	00010	001	x	x	x	1	x	0
3	00011	001	x	x	x	1	x	1
4	00100	010	x	x	x	x	x	x
5	00101	011	x	x	x	x	x	x
6	00110	100	x	x	x	x	x	x
7	00111	101	x	x	x	x	x	x
8	01000	110	x	x	x	x	x	x
9	01001	111	x	x	x	x	x	x
10	01010	000	1	1	1	x	x	0
11	01011	000	1	1	1	x	x	1
12	01100	000	0	0	1	0	x	0
13	01101	000	0	0	1	0	x	1
14	01110	000	0	0	1	1	x	0
15	01111	000	0	0	1	1	x	1
16	10000	000	0	1	0	0	x	0
17	10001	000	0	1	0	0	x	1
18	10010	000	0	1	0	1	x	0
19	10011	000	0	1	0	1	x	1

Table 380. Standard channel condition encoding scheme (continued)

No.	Encoded channel conditions [C4-C0]	Host service request bits	Link request	MatchA / TransB	Match.2 / TransA	In/Output pin state[1]	Channel flag1	Channel flag0
20	**10100**	**000**	**0**	1	1	**0**	**x**	**0**
21	**10101**	000	**0**	1	1	0	**x**	1
22	**10110**	000	**0**	1	1	1	**x**	0
23	**10111**	000	**0**	1	1	1	**x**	1
24	**11000**	000	1	0	0	**0**	**x**	**0**
25	**11001**	000	1	0	0	0	**x**	1
26	**11010**	000	1	0	0	1	**x**	0
27	**11011**	000	1	0	0	1	**x**	1
28	**11100**	000	1	0	1	**x**	**x**	**0**
29	**11101**	000	1	0	1	**x**	**x**	1
30	11110	000	1	1	**0**	**x**	**x**	0
31	11111	000	1	1	0	**x**	**x**	1

▨ Host Service Request

---

1. The ETPU_CxCR bit ETPD selects between input and output pin state.

Alternate condition encoding scheme

This scheme is shown in *Table 381*. Because the HSR bits cannot be tested by microcode, only three distinct Host Service Request can be used:

1. HSR = 010 or 011, which are coded into the same Entry Points (0 to 3)
2. HSR = 100,101 or 001, which are all coded into Entry Point 4
3. HSR = 110 or 111, which are both coded into Entry Point 5

The remaining Entry Points use both channel flags for better state decoding, making this scheme better suited for Functions which need more states and/or faster state decoding, without needing many HSRs.

Table 381. Standard channel condition encoding scheme

No.	Encoded channel conditions [C4-C0]	Host service request bits	Link request	MatchA / TransB	Match.2 / TransA	In/Output pin state(1)	Channel flag1	Channel flag0
0	00000	001	x	x	x	0	x	0
1	00001	001	x	x	x	0	x	1
2	00010	001	x	x	x	1	x	0
3	00011	001	x	x	x	1	x	1
4	00100	010	x	x	x	x	x	x
5	00101	011	x	x	x	x	x	x
6	00110	100	x	x	x	x	x	x
7	00111	101	x	x	x	x	x	x
8	01000	110	x	x	x	x	x	x
9	01001	111	x	x	x	x	x	x
10	01010	000	1	1	1	x	x	0
11	01011	000	1	1	1	x	x	1
12	01100	000	0	0	1	0	x	0
13	01101	000	0	0	1	0	x	1
14	01110	000	0	0	1	1	x	0
15	01111	000	0	0	1	1	x	1
16	10000	000	0	1	0	0	x	0
17	10001	000	0	1	0	0	x	1
18	10010	000	0	1	0	1	x	0
19	10011	000	0	1	0	1	x	1
20	10100	000	0	1	1	0	x	0
21	10101	000	0	1	1	0	x	1
22	10110	000	0	1	1	1	x	0
23	10111	000	0	1	1	1	x	1
24	11000	000	1	0	0	0	x	0
25	11001	000	1	0	0	0	x	1

Table 381. Standard channel condition encoding scheme (continued)

No.	Encoded channel conditions [C4-C0]	Host service request bits	Link request	MatchA / TransB	Match.2 / TransA	In/Output pin state[1]	Channel flag1	Channel flag0
26	**11010**	000	1	0	0	1	**x**	0
27	**11011**	000	1	0	0	1	**x**	1
28	**11100**	000	1	0	1	**x**	**x**	**0**
29	**11101**	000	1	0	1	**x**	**x**	1
30	11110	000	1	1	**0**	**x**	**x**	0
31	11111	000	1	1	0	**x**	**x**	1

▨ Host Service Request

1. The ETPU_CxCR bit ETPD selects between input and output pin state.

Entry point format

Entry Point information includes a Preload-Parameter selection field, a Match Enable field, and the first microcode address of the thread. The Entry Point format is illustrated in *Figure 392*.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PP	ME	MICROCODE ADDRESS													

Figure 392. Entry Point Format

PP—Preload Parameter

Field	Description
0-13 MICROCODE ADDRESS	Microcode Address This field specifies the microcode address on which the thread is to begin execution

Field	Description
14 ME	Match Enable ME specifies whether match event recognitions are enabled or disabled for the thread associated with the entry point during the thread execution. If they are disabled, a match recognition can only occur after channel service. For more details refer to *Section , Match Recognition*. Matches are disabled during the thread. Matches are enabled during the thread. The disabling of Match A/B recognition by MEF is dependent on IPACA/B configuration on the serviced channel (see *Section , Pin Control Registers*). If IPACA = 1xx, Match A is not disabled by ME = 0. Likewise, IPACB = 1xx overrides the effect of ME on Match B to “always on” If IPACA/B = 0xx, Match A/B is disabled for one microcycle during TST (see *Section , Time slot transition*) and is re-enabled when Entry Point is loaded, if ME = 1. Note that if the comparator is in equal-only mode and the time base reaches the value of the Match register during the time that recognition is disabled (beginning of TST, plus whole thread if ME = 0), the match recognition is lost. If the comparator is in greater-equal mode, the match event may be recognized after the disabling period if it satisfies the “greater-than” condition.
15 PP	Preload Parameter PP indicates which pair of channel parameters are loaded into registers P and DIOB from the SPRAM prior to the execution of a thread. Preloading occurs during the time-slot transition period (see *Section , Time slot transition*) Microengine register P is preloaded from parameter 0 and DIOB from parameter 1. Microengine register P is preloaded from parameter 2 and DIOB from parameter 3. The parameter numbers are offsets from the channel parameter base address. For more info, see *Section , Parameter access*.

Time slot transition

The Time Slot Transition period (also called TST for short) is the interval between the servicing of two channels, during which all Channel-specific context is loaded for the new serviced Channel. The primary tasks completed during this period include:

- Set MEF for the first microcycle plus eventual wait-states.
- Reset the MEF for one microcycle after the first microcycle plus wait-states.
- Update of the CHAN register with the number of the new channel to be serviced.
- Parallel update of ERTA and ERTB from CaptureA and CaptureB registers of the new serviced channel.
- Sampling of the branch conditions of the new channel to be serviced into the branch logic (this means flags TDLA/B, MRLA/B, LSR, FM[1], FM[0], and PSS). The branch conditions are coherent with the timebase capture values sampled into ERTA/B (if MRLA/B, TDLA/B are set at the same time of the sampling, either both old flag state and capture values are sampled, or both new values are sampled).
- Formation of the entry point address.
- Copy the ME bit in the Entry Point into MEF.
- Access to the entry point location and getting the first microinstruction address.
- Preload of two parameters from the SPRAM into P (32 bits) and DIOB (24 bits).
- Fetch the first instruction of the thread to be executed for the new channel.
- Preset the RAR value (see *Section , RAR – Report Address Register*).

The preload operation is 32-bit wide for P and 24-bit wide for DIOB. The P register is loaded with all the 32-bit parameter. The DIOB register is loaded with the lower 24-bits of the parameter. The microcode can switch at any time to access the lower 24-bits, upper byte, or all the 32-bits of any parameter in the SPRAM. Preload of P-DIOB pair of parameters is atomic with respect to Host and CDC accesses, and so are coherent with their dual-parameter coherent transfers. For more details see *Section 22.5.4, Parameter sharing and coherency*.

No instructions are executed at the engine where the time slot transition period occurs, but the other engine can execute normally. Match A/B is unconditionally disabled on the second TST microcycle, if IPACA/B = 0xx (respectively). During the rest of time slot transition, match recognition can be disabled or not, depending on IPACA/B field and ME. See *Section , Match Recognition*.

Time Slot Transition takes a minimum of 3 microcycles (6 system clocks), which may be extended due to SPRAM arbitration wait-states for the first preload access (see *Section , SPRAM Arbitration*). When no wait-states are received (*Figure 393*), DIOB is preloaded twice, one for each PP value, and the correct value remains in DIOB when the Entry Point is loaded. *Figure 394* and *Figure 395* show the timing for one and two wait-states, respectively.

Registers B, C, D and SR are not altered by TST and keep their values from the previous thread. The values of registers A, MACL and MACH are not guaranteed at the thread start.

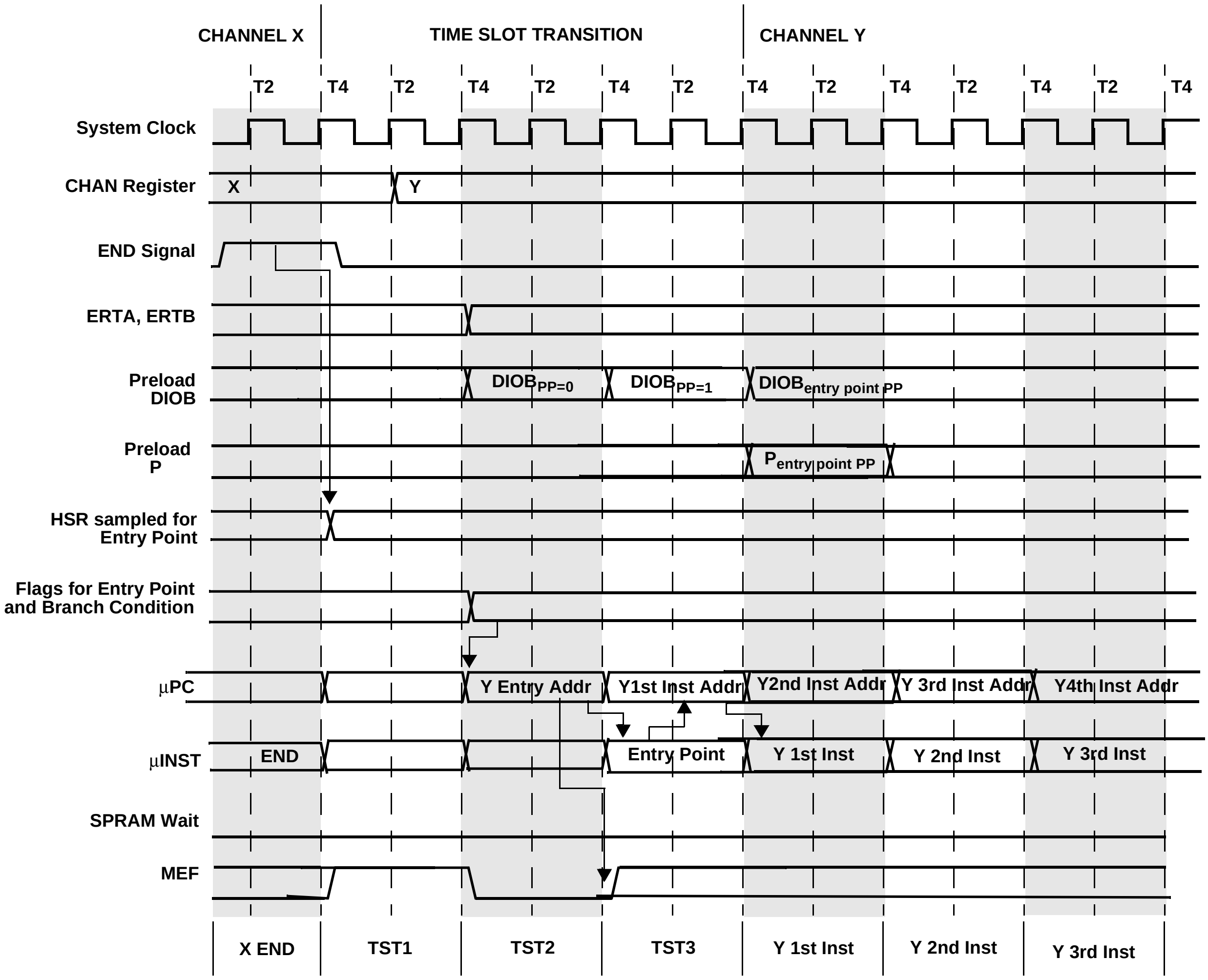


Figure 393. TST Timing – No Wait-states

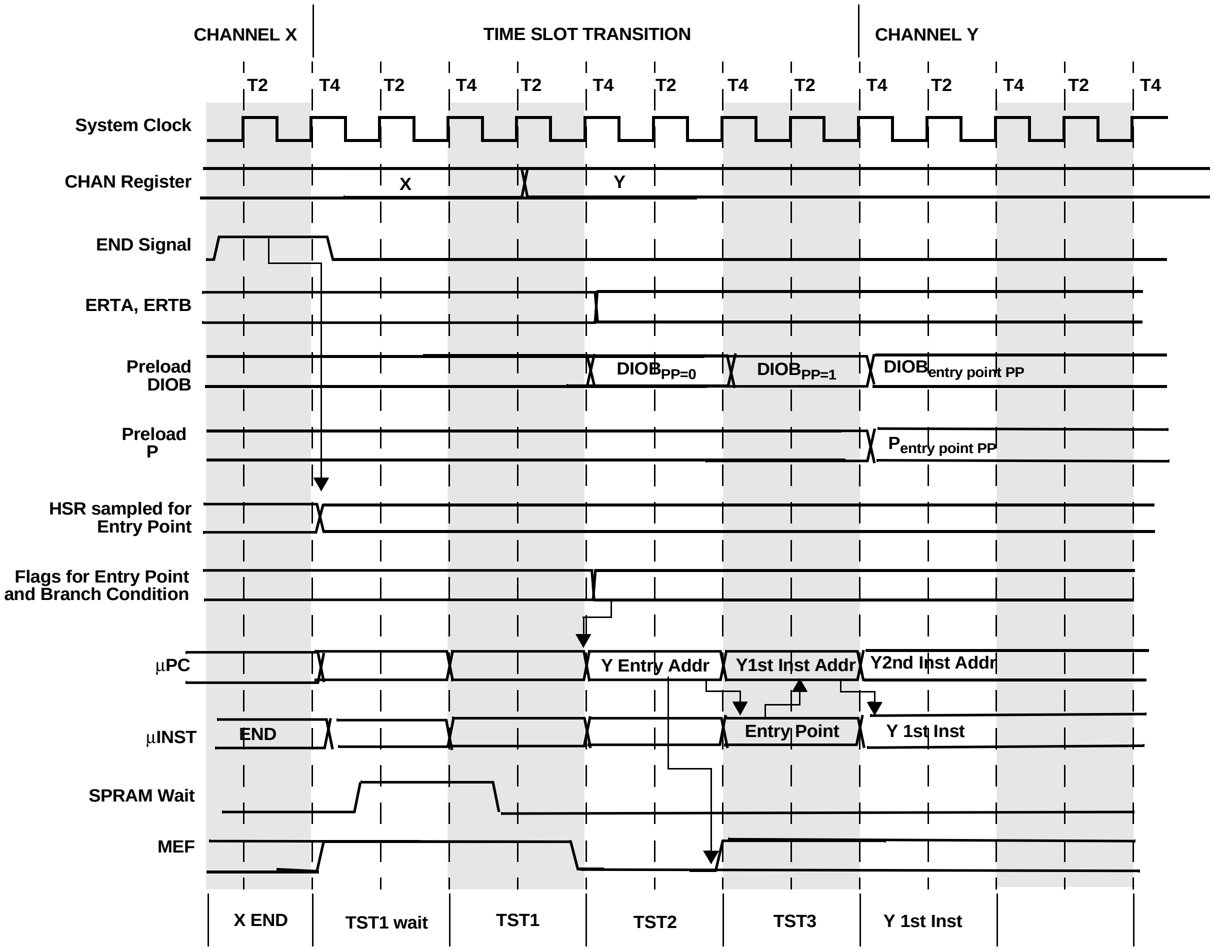


Figure 394. TST Timing – 1 Wait-State

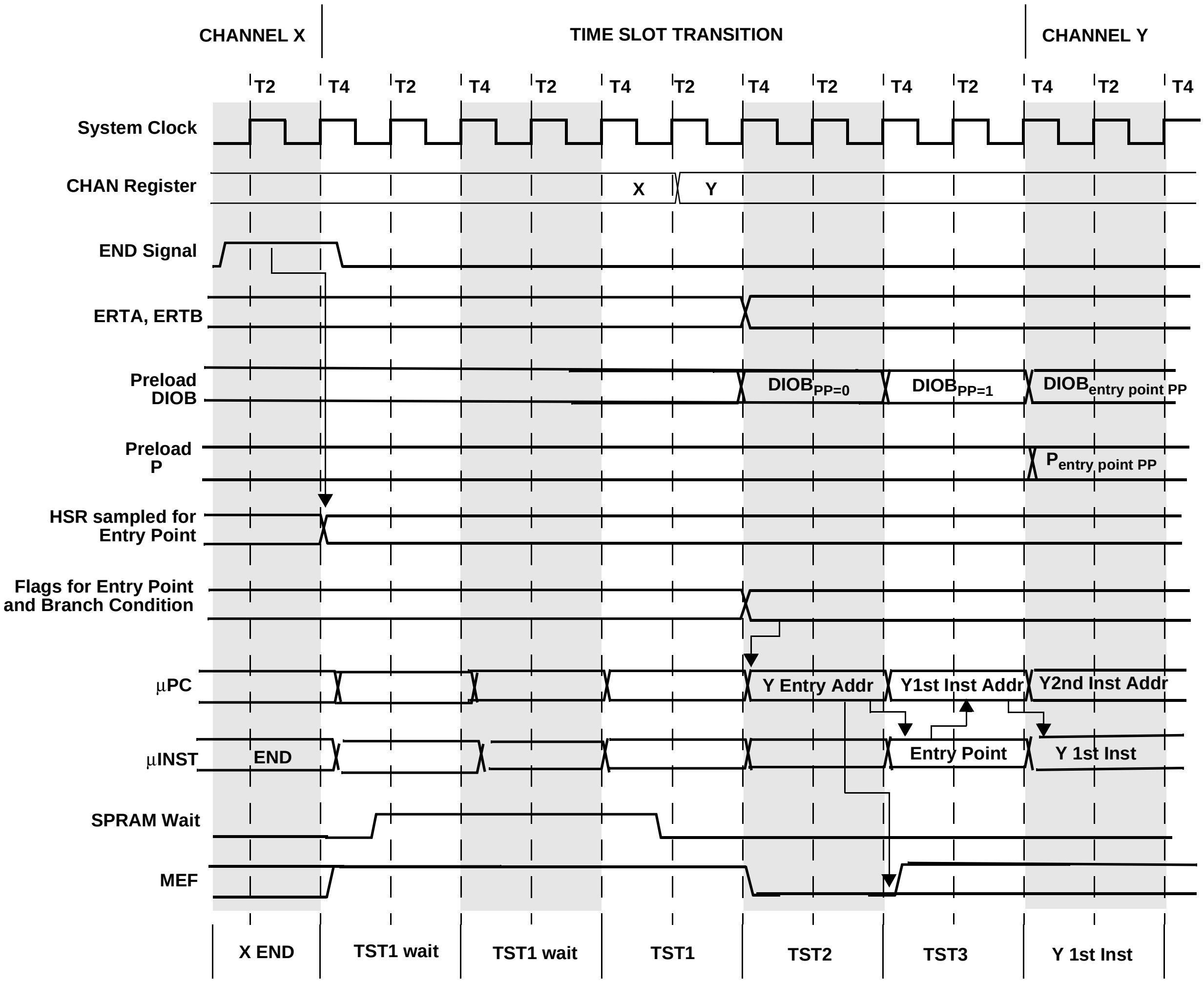


Figure 395. TST Timing – 2 Wait-states

For more information on Channel-specific registers and flags, refer to *Section 22.5.5, Enhanced Channels*. For more information on P, ERTA/B and DIOB registers refer to *Section , Registers*.

Thread ending

Threads can finish by either:

- An instruction with the END field active (see *Section , Ending current thread – END*).
- A forced END by host writing to the ETPU_ECR bit FEND (see *Section , ETPU_ECR – eTPU Engine Configuration Register*).
- A forced END caused by Watchdog timeout (see *Section , Watchdog*).

Watchdog

Each engine has a watchdog mechanism to prevent a thread or a sequence of threads from running too long, impacting the latency of the other channel services. The watchdog is configured through the register ETPU_WDTR (see *Section , ETPU_WDTR – eTPU Watchdog Timer Register*). When the watchdog is enabled, an internal counter increments on each microcycle when a thread is executing. If the count is greater than the value specified in the ETPU_WDTR field WDCNT and a thread is still executing, the watchdog:

1. Forces an END of the thread
2. Issues a Global Exception and sets the ETPU_MCR bit WDTO (see *Section , ETPU_MCR – eTPU Module Configuration Register*).

The watchdog can be configured in one of the following modes, defining how the internal watchdog count is reset:

- **Thread Length Mode**: the watchdog count is reset at the end of each thread.
- **Busy Length Mode**: the watchdog count is reset when the microengine goes idle. A sequence of threads, one right after another, keeps the count running. The counter is also reinitialized when a thread is forced to end, so that a new count begins if another TST initiates at the following microcycle.

The following applies to the watchdog mechanism:

- Microcycles during TST and SDM access wait-states (on TST or instruction execution) are counted.
- If the watchdog count equals WDCNT in the last microinstruction (with SDM wait-states or not) of a thread servicing a channel.
- If the watchdog count expires (gets greater than WDCNT) during the TST, the thread is forced end on its first instruction.
- The watchdog count does not wrap, so that a thread (in thread length mode) or a thread sequence (in busy length mode) that lasts for more than the maximum value of WDCNT does get a forced end.

Note: *Watchdog must not be enabled when the microengine enters halt mode.*

The counter does not run when the engine is stopped, and resets when the watchdog is disabled.

22.5.2 Host interface

System configuration

System Configuration Registers are described in *Section 22.4.2, System configuration registers*. Detailed explanation on the configured functionalities is found throughout *Section 22.5, Functional description*, and a specification for the initial configuration sequence is found on *Section 22.6.1, Configuration sequence*.

Interrupts and data transfer requests

Interrupt types and sources

Each one of the eTPU channels can be a source of two requests: **Channel Interrupt** request and **Data Transfer Request**. Channel Interrupts are targeted to a Host CPU. Data Transfer Requests may be targeted to a data transfer module (e.g., a DMA controller). Interrupt and Data Transfer registers are used by the Host to enable interrupts and data transfer requests, indicate their status and service them. Interrupt and Data Transfer requests have the same sets of registers and external signals, and are handled in the same way. They differ only by the fact that Data Transfer Requests are also cleared by the assertion of respective DMA completion acknowledge line. Data Transfer Requests can be used as another source for Host interrupts at MCU integration if not used with a DMA.

Note: Interrupt and Data Transfer requests can be cleared even when engines are in Module Disable Mode, through the Global Channel Registers, and also DMA completion for Data Transfer requests.

Channel Interrupts and Data Transfer Requests can only be issued by eTPU microcode, through one of the Channel Control instruction fields (see *Section , Channel interrupt and data transfer requests*).

Both Channel Interrupt and Data Transfer requests can be individually enabled for each channel.

eTPU Interrupt and Data Transfer Registers are mirrored in two organizations: grouped by Channel and grouped by type (interrupt status, interrupt enable, data transfer status, data transfer enable). This allows either “channel-oriented” or “bundled channel” Host interrupt service schemes, or a combination of them. For a detailed description, refer to *Section 22.4.5, Channel registers layout*, and *Section 22.4.6, Global channel registers*.

eTPU can also assert a **Global Exception** interrupt indicating a global illegal state. There are three possible sources for a Global Exception:

- Execution of an illegal instruction by the microengine (see *Section , Illegal Instructions*). This Global Exception source is flagged by the bits ILF1 and ILF2 in register ETPU_MCR.
- An SCM signature mismatch detected by the Multiple Input Signature Calculator (MISC). See *Section , SCM Test – Multiple input signature calculator*. This source is flagged by the bit SCMMISF in register ETPU_MCR.
- Microcode request, through microinstruction field CIRC (see *Section , Channel interrupt and data transfer requests*). This Global Exception source is flagged by bits MGE1(Engine 1) and MGE2(Engine 2) in register ETPU_MCR. The cause of this illegal state is application-dependent. The microcode may write an error code into the SPRAM to indicate the cause of the exception, for instance.
- An SDM or SCM non-correctable error due to a microengine access

Global Exceptions cannot be directly disabled within eTPU, except by disabling its sources (MISC and microcode), and it is cleared by writing 1 to the GEC bit in ETPU_MCR. Clearing Global Exception clears all Global Exception source status bits (ILF1, ILF2, SCMMISF, MGE1, MGE2). If GEC is written 1 at the same time any of the sources issues a Global Exception, both the interrupt and the status bit of that source remains asserted. The assertion of Global Exception by one of the sources above does not prevent the others from asserting it too, so any number of them, in any combination, can be flagged.

Note: There can be a race between the clear of a Global Exception and occurrence of a new set condition, such that the set happens just before the clear and cannot be sensed by the Host. Therefore, Global Exception cannot be used as a normal interrupt source: it should only be used for emergency procedures.

Interrupt and data transfer request overflow

If a Channel Interrupt was issued, its status bit is still set, and microcode issues another Channel Interrupt, the Interrupt Overflow status bit is set for that channel. Interrupt Overflow status can be checked by the Host in Channel Status register ETPU_CxSCR bit CIOS (*Section , ETPU_CxSCR – eTPU Channel x Status Control Register*), mirrored in register ETPU_CIOSR (*Section , ETPU_CIOSR – eTPU Channel Interrupt Overflow Status Register*). Interrupt Overflow status is not cleared automatically when Interrupt Status is cleared. The same mechanism and respective registers (ETPU_CDTROSR) are available for Data Transfer Requests.

If interrupt is set and cleared at the same time, set prevails and overflow is not altered (keeps the same state as it was before, asserted or not).

Global Exception has no overflow status.

Parameter access

Parameter access widths

From the Host side the SPRAM address space is mapped in bytes, and each 32-bit parameter occupies 4 contiguous, aligned bytes. The Host can read/write the SPRAM by 8-, 16-, or 32-bit accesses in aligned addresses.

In 32-bit access, Host can access all 32 bits or only the lower 24 bits with an automatic sign extension (see *Section , Parameter sign extension area*).

Parameter addresses and endianness

To access parameter number *xxx*, eTPU Microengine(s) would select address xxx. The Host would add *(xxx*4)* to the SPRAM base address to access the same parameter. For example, parameter 0x101 is seen by the Host in *(SPRAM base address +0x404)*. An example of SPRAM memory map is shown in *Figure 396*. The Host can access the SPRAM with a 32-bit-wide bus cycle to a four-byte aligned address, 16-bit-wide bus cycle to a two-byte aligned address, or 8-bit wide bus cycle to any byte address.

The address of the 24-bit parameters and the most significant byte depends on the endianness of the MCU. For more details, see the *Section 22.6.6, Endianness*.

Parameter concurrency

Host accesses to parameters may occur in parallel with eTPU Microengine accesses. Readings taken from a group of parameters while they are being simultaneously updated may lack coherency. eTPU provides mechanisms to ensure parameter coherency in accesses from both Host side and Microengine side, including the use of a coherent dual-parameter transfer mechanism, described in detail on *Section 22.5.4, Parameter sharing and coherency*.

Parameter sign extension area

The SPRAM address space to the Host is mirrored in a Parameter Sign Extension (PSE) area (see *Section 22.4.1, Memory map*). Accesses from the Host to the PSE area differ from accesses to the standard SPRAM address space as follows:

- **Writes**: the most significant byte of the parameters is not written, and the SPRAM retains the old byte value, regardless of the Host access size.
- **Reads**: the most significant bit of the 24-bit parameter (that is, the msbit of the second most significant 32-bit parameter byte) is repeated in the 8 most significant bits of the read value on all 32-bit reads and most significant 16- and 8-bit reads.

The same parameters written in the standard SPRAM address space are read from the PSE area with the same offsets, and vice-versa. See *Table 472* for a reference of the address offsets in big and little endian machines.

This feature reliefs the Host from extending the signal of 24-bit eTPU parameters before calculations, and from read-modify-write accesses to modify 24-bit parameters at the SPRAM.

SPRAM organization

The SPRAM internal partition for channel allocation is dynamic and programmed in the Channel Registers (see *Section , ETPU_CxCR – eTPU Channel x Configuration Register*).

The Host application is responsible for allocating a different parameter base address to each channel during the initial eTPU configuration, and to allocate enough parameters for the selected function, with no unintentional overlapping between parameters of different functions.

Besides channel parameters, global areas may have to be allocated for parameters that are shared by more than one channel, in one or both engines. Also, temporary parameter areas should be reserved to be used by the coherent parameter transfer mechanisms described in *Section 22.5.4, Parameter sharing and coherency*, if necessary.

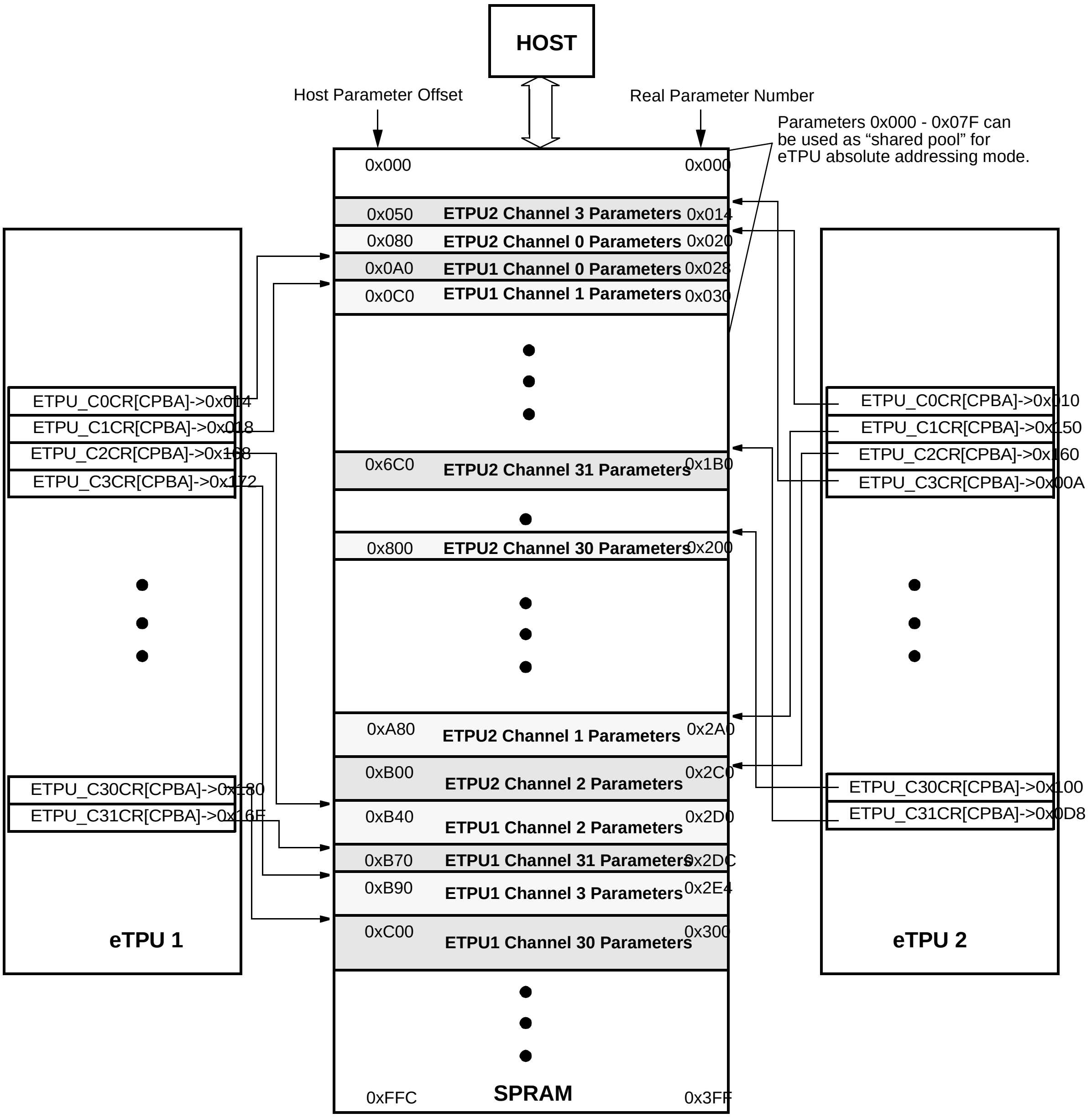


Figure 396. SPRAM organization example

A single-engine eTPU or dual-engine eTPU system may require less parameters than the maximum number provided by the SPRAM. Since the SPRAM partition is fully dynamic, there is no limitation of fixed channel addresses, and the reduced array can be fully utilized.

Host service requests

Host CPU can request immediate service from a channel by writing a non-zero value to the Host Service Request register field HSR (see *Section , ETPU_CxHSRR – eTPU Channel x*

Host Service Request Register). There is one HSR field for each channel, so that writing to it generates a Service Request to the respective channel only. A zero value in HSR means no Host Service Request is pending for the channel.

HSR value turns to 000 automatically at the end of microengine service for that channel, but only if the thread started due to an HSR.

The meaning of a non-zero HSR value depends on the Function assigned for the channel. These bits are part of the conditions which select the Function entry point, and cannot be tested by microcode. For more details, refer to *Section , Entry points*.

If Host writes HSR = 000 when a thread for the same channel is already running, the thread runs until the end and is not aborted. If Host writes HSR>000 when an HSR thread for the same channel is already running, HSR value resets at the end of the thread, and no new HSR will be pending. If HSR is written before its value is resolved by the scheduler during TST, the entry point will obey the new HSR value, and if this new value is 000, no service thread is executed for the HSR.

The scheduling of HSRs is completely asynchronous with Host accesses, and there is no race-free manner to change an HSR value before service thread execution, so generally the safe way is: write HSR>0 only when HSR = 0. Error recovery or emergency host procedures may require one to the safely abort service and reset channel state when an HSR is already pending or executing. In these cases, the procedure below should be followed:

1. Disable the channel, writing CPR = 00 in register ETPU_CxCR. That will prevent any pending HSR to be serviced.
2. Check if the channel is currently being serviced, reading its service status bit in register ETPU_CSSR. If it is, wait for the time necessary to finish the service pending, or check again until HSR == 0, or channel service bit in ETPU_CSSR is cleared.
3. Write HSR with the error recover value. This value should, possibly combined with other host-defined flags in SPRAM or FM bits, initiate a channel reset or error recovery procedure.
4. Re-enable the channel, writing CPR value > 0 in register ETPU_CxCR.

SCM access

Only Host can access SCM as data. Depending on the specific device, SCM may be implemented as a RAM or ROM. This determines Host accesses to the SCM as shown below.

SCM RAM implementations

When SCM is implemented as RAM, the Host may read or write to SCM by setting ETPU_MCR bit VIS = 1. If VIS = 0 and Host tries to access SCM space, a bus error is issued, writes are ineffective and read data is meaningless. Both engines must be stopped or halted to set VIS = 1.

Only 32-bit aligned writes are allowed to SCM from the Host. Write accesses of other sizes store unpredictable values into SCM.

Note: It is necessary to turn VIS bit on to set software breakpoints (see Section , Software breakpoints).

SCM low power

SCM turns off its internal clocks when both engines are stopped (ETPU_ECR bit STF asserted), VIS = 0 at ETPU_MCR, and MISC is not enabled (SCMMISEN = 0). The SCM clocks are automatically turned on if either one of the STF bits is negated or VIS turns to 1, or SCMMISEN turns to 1.

SCM clocks are not turned off if any of the engines is not stopped, even if they are both halted.

The conditions for SCM Clocks and MISC activation are summarized in *Table 382*.

Table 382. SCM clocks and MISC activation

ETPU_ECR_1 STF	ETPU_ECR_2 STF	ETPU_MCR VIS	ETPU_MCR SCMMISEN	SCM clocks	MISC
0	x	0(1)	1	On	On
0	x	0(1)	0	On	off
x	0	0(1)	1	On	On
x	0	0(1)	0	On	off
1	1	0	0	off	off
1	1	0	1	On	On
1(2)	1(2)	1	0	On	off
1(2)	1(2)	1	1	On	off(3)
0	0	x	0	On	off
0	0	x	1	On	On

1. VIS cannot be written 1 if ETPU_ECR_1 bit STF = 0 or ETPU_ECR_2 bit STF = 0, and both HLTF bits are 0.
2. If VIS = 1, neither MDIS can be written 0 nor the engine leave Stop Mode, regardless of device stop request.
3. MISC resets and stays so when VIS = 1, restarting automatically when VIS goes 0 if SCMMISEN = 1.

SCM off-range data

When read accesses are made, either by the Host or by a microengine, to addresses above the limit corresponding to the SCMSIZE value in ETPU_MCR, the value read comes from the register ETPU_SCMOFFDATAR. The Host can program the register at initialization with an opcode value with operations that try to protect or recover the system from runaway code, for instance: terminate the thread, clear channel flags, disable match and transition service requests, issue an interrupt, jump to an error recovery procedure[q]. Writes to unimplemented addresses do not return error and can write on unspecified mirror addresses, so they should be avoided.

22.5.3 Scheduler

Every Function is composed of one or more Threads. A Thread consists of a group of instructions that, once begins execution, cannot be interrupted by host or channel events. Each active channel intents to be serviced, being granted time for Thread execution. Since one microengine handles several channels operating concurrently, the Function threads must be executed serially.

---

[q] Only part of these suggested operations can be parallelized in a single instruction, see *Section , Microinstruction formats*.

The task of the Scheduler is to recognize and prioritize the channels needing service and to grant execution time to each channel. The time given to an individual Thread for execution or service is called a **Time Slot**. The duration of a time slot is determined by the number of instructions executed in the Thread plus SPRAM wait-states received, and varies in length.

At any time, an arbitrary number of channels can require service. To request service, channel logic, eTPU microcode or Host application notifies the Scheduler by issuing a Service Request.

Channel enabling and priority assignment

Every channel is assigned one of three priority levels—high, middle, or low—by the Host CPU, through the Channel Configuration Register field CPR (see *Section , ETPU_CxCR – eTPU Channel x Configuration Register*). These registers are also used to disable the channel, which is equivalent to assigning it a “null” priority. In this case, the Scheduler does not grant any of its Service Requests.

It is possible to change the channel priority level or disable it dynamically. If the Host disables a channel when it is currently being serviced, channel service thread will complete. This means that it is possible for the output level of a channel signal to change, or a Host interrupt occur, even after its priority register was written to “null”. For instance, if an output transition is scheduled, the transition will occur even after the channel is disabled.

Service requests previously pending or that occur while a channel is disabled remain asserted while the channel is disabled, and are serviced if the channel is enabled again, in due time determined by the priority scheme and concurrent requests from other channels. Channels are disabled after reset, and it is recommended to configure a Host Service Request for initialization of a channel before that channel is enabled to active priority (see *Section 22.6, Initialization/Application information*).

Channel priority schemes

The Scheduler holds a Service Grant register with one bit for each channel. Once the Scheduler grants a time slot to channel, the Service Grant bit for that channel is asserted in the Service Grant register. When the Service Grant bit of a channel is set, the channel may request new service but is not serviced again before its Service Grant bit is cleared.

When all channels in a same priority level are serviced, their Service Grant bits are cleared at the end of the thread, one system clock before the next serviced channel is calculated, according to the scheme below(r):

- Clear all grant bits of priority High if all channels of that priority that are requesting have their grant bits in 1.
- Clear all grant bits of priority Medium if all channels of that priority that are requesting have their grant bits in 1.
- Clear all grant bits of priority Low if all channels of that priority that are requesting have their grant bits in 1.
- Clear all grant bits of disabled channels.

This scheme assures that no channel is left with its grant bit forever asserted (preventing it from being serviced again), even if the channel priorities are reassigned during the execution.

---

r. Grant bits are also cleared in the next clock, when the service channel is chosen, or when the microengine is idle, using the same scheme.

Priority level is determined based on the maximum latency desired for each channel. A channel having a Function that requires the most frequent or more immediate service should be allocated a high priority level.

The eTPU employs a **primary** and a **secondary priority scheme**. These two schemes ensure frequent servicing of high-demand Functions and ensure a minimum time allocation to all channels requesting service, regardless of their priority level. The primary scheme prioritizes requesting channels that have different priority levels; the secondary scheme prioritizes requesting channels that have the same priority level.

Initially, a channel requests service and is granted a time slot by the Scheduler: Service Grant bit is asserted. If only high-level channels constantly receive service first because of their priority level, middle- and low-level channels would only be serviced by default, i.e., if no high-level channels request service. To ensure that each priority level receives an opportunity for servicing, every time slot has a fixed priority level that the Scheduler honors first. Divided into sets of seven, time slots are numbered from one to seven. *Figure 397* illustrates the numbered time slots in sets of seven (fields A and B) and identifies their assigned default priority level. The high level has more time slots than the middle and low levels. Out of every seven time slots available, four are assigned to honor high-level channels first, two are assigned to honor middle-level channels first, and one is assigned to honor low-level channels first. Only one request (in each engine) is serviced per time slot. When no channel requests service and the microengine is idle the priority scheme is initialized to time slot one, to prevent priority inversion on the next request(s).

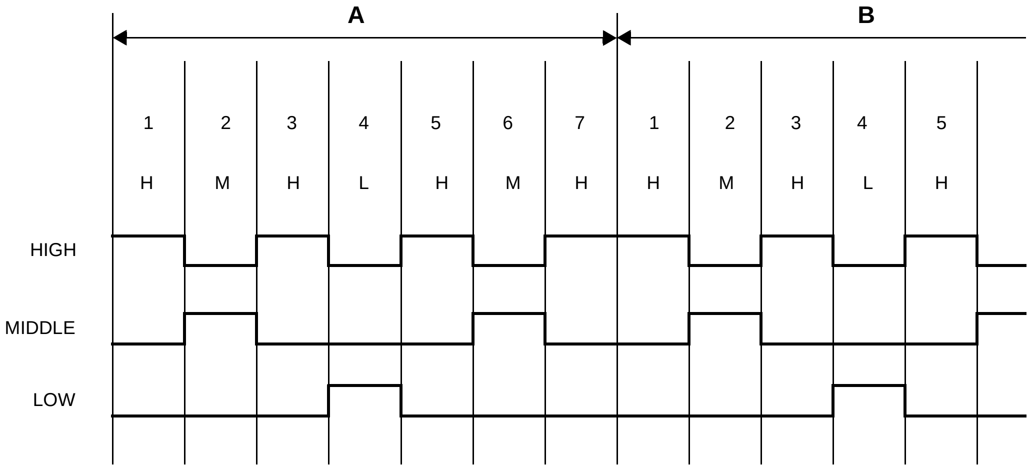


Figure 397. Time Slot Priority levels

Primary scheme – priority among channels on different levels

Although time slot priority assignment is fixed, the servicing priority is not. The primary scheme acknowledges the priority level assigned to a time slot, granting service first to a channel having the same priority. In *Figure 397*, time slot one has a high-level assignment;

(s) Priority inversion would occur in the following situation: no channel is requesting service, and the current time slot is primarily assigned to a low-priority channel. If the Scheduler was not reset to time slot one and two channels requested service at the same time, one with high priority and the other with low priority, the channel to be serviced would be the low-priority channel.

therefore, a high-level channel requesting service is recognized first. However, if no high-level channel requests service, the Scheduler recognizes a requesting middle-level channel. If this level has no request, the Scheduler continues to the low-level. If no requests occur, the Scheduler truncates the seven state cycle and starts a new cycle at time slot one, waiting for the first request. Granting service to a different-level channel is called priority passing. The order of passing always gives the highest priority to the assigned level, and the second priority to the higher of the remaining requesting priority levels as shown in *Table 383*.

Table 383. Priority passing

Assigned priority level	Next priority level	Next priority level
High	→ Middle	→ Low
Middle	→ High	→ Low
Low	→ High	→ Middle

When priority is passed to another level, that level is serviced and the fixed-priority-level sequence is resumed with the next time slot.

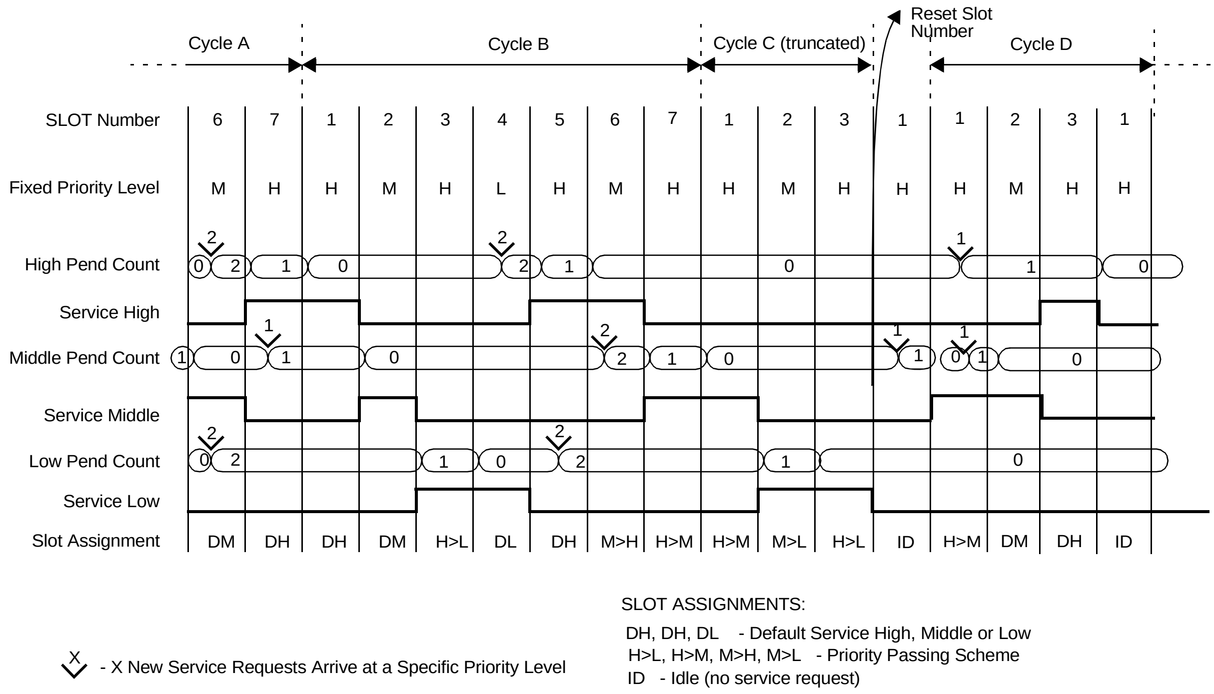


Figure 398. Priority Passing Example

Examples of priority passing are shown in *Figure 398*. Each cycle contains seven time slots (or less if no service request exist). In cycle B, no high-level or middle-level service requests are present before time slot three which is assigned by default to high-level priority. Thus, time slot three is passed to the low level. In cycle B there are also no middle-level service requests before time slot six, so it passes the priority to a requesting high-level channel.

During time slot six no more high level requests are left, but two new middle-level requests arrive, and there are also three low level pending service requests. Thus, time slot seven of cycle B and time slot one of cycle C are passed to the middle-level which is the next priority level after high. Time slots two and three of cycle C are passed to the low level which contains the three remaining channel service requests. At time slot three of cycle C the last low level request is serviced, and the Scheduler passes to idle state. At this point the cycle C is truncated and the Scheduler passes to time slot one of cycle D.

Priority passing disabling

The priority passing scheme allows a case where a high priority channel looses to a lower priority one right after another lower priority has been serviced, exemplified in the Cycle D on *Figure 398*. A middle priority channel wins time slot 1 due to priority passing from high to middle. While it is being serviced, two new service requests arrive, one high and one middle priority. The high priority request looses to the middle one on next time slot 2 by default priority assignment.

This priority inversion can be avoided by setting the ETPU_ECR bit SPPDIS (see *Section , ETPU_ECR – eTPU Engine Configuration Register*), which disables the priority passing mechanism. When priority passing is disabled, at the end of the thread the slot number is incremented until a time slot that matches the priority of one of the requesting channel(s). The time slot advance takes no extra clocks. If no channel requests service, the time slot counter stays at time slot 1. The priority selection scheme with disabled priority passing is summarized in *Table 384*.

Table 384. Priority Passing Disabling

At the end of time slot	servicing priority	if any request of priority	service it on time slot	else if any request of priority	service it on time slot	else if any request of priority	service it on time slot
1	High	Medium	2	High	3	Low	4
2	Medium	High	3	Low	4	Medium	6
3	High	Low	4	High	5	Medium	6
4	Low	High	5	Medium	6	Low	4
5	High	Medium	6	High	7	Low	4
6	Medium	High	7	Medium	2	Low	4
7	High	High	1	Medium	2	Low	4

An example of the priority passing disabling scheme is illustrated in *Figure 399*. The sequence of service requests is the same as in the example of *Figure 398*, and although the time slot incrementing differs, the priorities granted are the same for cycle B. Cycle C has one of the low priority channels serviced before the second middle one. Cycle D, however, no longer has the priority inversion.

In cycle B, after the time slot 2 only a low priority request remains, so the time slot count advances directly to 4, which has a low priority assigned. Time slot keeps on 4 for the next service, as only a low priority request remains also, and only time slot 4 is assigned to low. Two high priority services contend for the next time slot 5 (assigned to High). The second high priority channel is serviced on the next time slot, jumped to 7 because there is no middle request, ending cycle B. Cycle C starts with time slot 2, as there are no high priority requests and two middle and two low ones. After the first middle service, time slot count skips 3 assigned to high (no high requests), and services a low priority channel on time slot 4. It follows the same scheme until there are no other requests and cycle C is truncated, resetting the time slot counter to 1.

Cycle D begins with a middle request, jumping to time slot 2. During this service two requests arrive, one high and one middle. Unlike what happened with priority passing, the next serviced is the high priority channel, as the time slot increments to 3. The second middle priority channel request in cycle D is finally serviced next, on time slot 5.

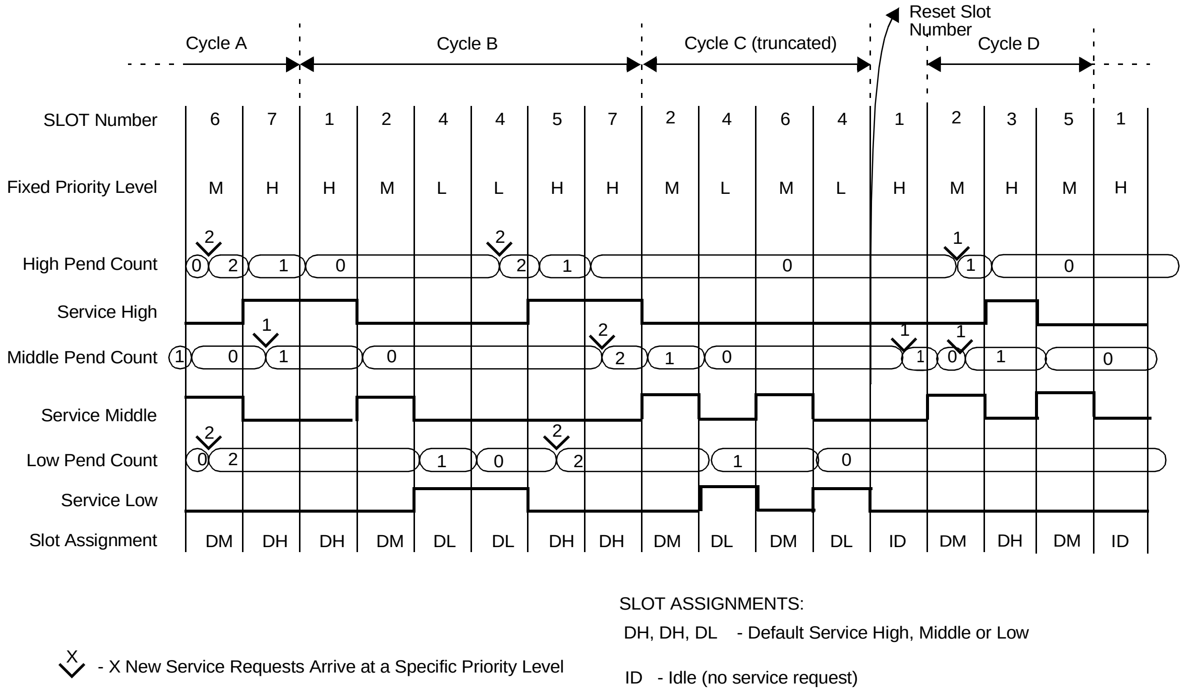


Figure 399. Priority Passing Disabling Example

Secondary scheme – priority among channels on the same level

Because channels can randomly request service, channels having the same priority level will inevitably request service simultaneously. A secondary scheme prioritizes these requests. The Scheduler services channels on each of the three priority levels, beginning with the lowest numbered channel on that level.

Priority scheme example

The overall priority scheme simultaneously incorporates both primary and secondary schemes. Combining both schemes in the following example conveys their correlation.

1. One high-priority and one low priority channels request service, while the Scheduler is in time slot one. Having its service request bit asserted, a single high-level channel is granted the time slot, which has high-level priority (primary scheme) and its service

grant bit is asserted. At the end of the thread, the service grant bit is negated (no more requests of high priority level channels).

2. The Scheduler proceeds to time slot two, which has middle-level priority; however, no middle-level channel is requesting service. Priority is passed to the high level, but no high-level channel is requesting service; therefore, priority is passed again, and service is granted to the single requesting low-level channel. Once serviced, this channel's grant bit is negated (no more low-level requests).
3. The Scheduler resumes with the fixed-priority sequence on time slot three; however, no channels are requesting service. The Scheduler returns to time slot one, waiting for requests.
4. Two high-level and two middle-level channels simultaneously request service. Being in time slot one which is assigned high priority, the Scheduler finds the lowest numbered high-level channel (secondary scheme) and selects it for service. This channel's service grant bit is asserted.
5. The Scheduler continues to time slot two, which has middle priority (primary scheme), and allocates the slot to the lowest numbered middle-level channel requesting service (secondary scheme). The Scheduler notes the still unserviced middle-level channel and proceeds to time slot three.
6. Time slot three is allocated for high priority. The slot is allocated to the remaining unserviced high-priority channel, and the channel's service grant bit is asserted. The Scheduler checks again at the end of the thread. All service grant bits of high-level requested channels are asserted; therefore, all high-priority channels that requested have been allocated execution time. Under this condition, all service grant bits of the high-level serviced channels are negated. The Scheduler proceeds to time slot four.
7. Time slot four is allocated for low-priority channel; however, no low-level channel is requesting service. Priority is passed to the high level, but no high-level channel is requesting service; therefore, priority is passed again, and service is granted to the remaining middle-level channel which requests service. This channel's service grant bit is asserted. The Scheduler checks again at the end of the thread. All grant bits of middle-level requested channels are asserted; therefore, all middle-priority channels have been allocated execution time. Under this condition, all service grant bits of the middle-level serviced channels are negated. The Scheduler proceeds to time slot five. Meanwhile a low priority channel requests service.
8. Time slot five is allocated for high-priority channels, but there are no more requests from high-priority or middle priority channels. The single low-level channel which required service is granted time slot five. Once serviced, the channel's service grant bit is asserted. Next, the service grant bit is negated (no more requests of low priority level channels).
9. The Scheduler resumes with the fixed-priority sequence on time slot six; however, no channels are requesting service. The Scheduler returns to time slot one and waits for requests.

Time Slot Latency

Latency is the amount of time between a service request and the beginning of service on that channel. The following factors affect latency:

- Number of active channels
- Number of channels on a priority level
- Number of available time slots on a priority level
- Number of microcycles required to execute a thread of a Function
- Number of parameter RAM accesses during execution of a Function thread
- System clock frequency.

Each time slot may require a different number of microcycles, depending on the thread of a Function to be executed. This variation is shown in *Figure 400*.

For more details on latency evaluation, see *Section 22.6.5, Estimating worst-case latency*.

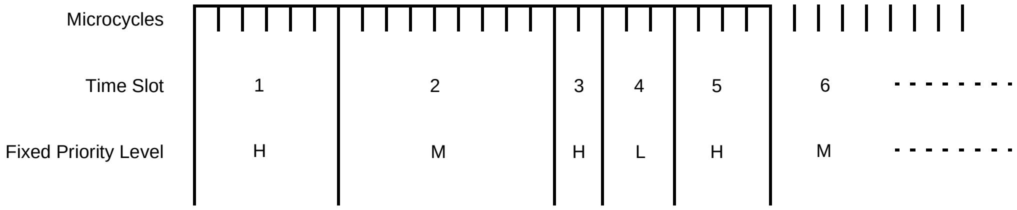


Figure 400. Time-slot variation

22.5.4 Parameter sharing and coherency

SPRAM can be concurrently accessed by Host and Microengines (two in a dual eTPU engine system). In general, there is no guaranteed order by which a group of parameters is accessed, which may lead to a lack of internal consistency if two or more related parameters are read when only part of them is updated.

eTPU provides mechanisms to guarantee parameter coherency. The most generic mechanisms for Host-eTPU coherency, suitable for any number of parameters, are:

- the use of Transfer Service Thread mechanism.
- the mailbox (or “software semaphore”) mechanism.

These mechanisms, described in *Section 22.6.3, Multiple parameter coherency methods*, use microcode to transfer parameters from temporary buffers in SPRAM to their definitive locations (or vice-versa). These methods have the disadvantage of wasting processing and code memory resources.

eTPU also provides a **Coherent Dual-parameter Controller (CDC)** mechanism. It is used by Host to coherently transfer pairs of parameters from/to a parameter buffer located on SPRAM to/from the locations on SPRAM where parameters are accessed directly by the channels. Coherency is guaranteed by SPRAM access arbitration. Although limited to two parameters only, it has lower latency and wastes no microengine resources(t). CDC usage is described in *Section , Coherent Dual-parameter Controller (CDC)*.

For parameters shared by both engines, eTPU provides **Hardware Semaphores**. Coherency is assured given the semaphores are used to prevent concurrent access to the changing parameters. Microengine can request semaphores using specific microinstructions (see *Section , Semaphore operations*). Hardware Semaphores are described in detail in *Section , Hardware Semaphores*.

Neither Host nor CDC have access to the hardware semaphores, but they can be combined with microcode transfer mechanisms if Host must coherently access parameters which are also shared by both engines.

In order to ensure coherent access to a group of parameters by two or more contenders, **each contender must have atomic access** to the shared parameters. Atomicity conditions are discussed in *Section , Host Side Atomic Access*, and *Section , Microengine Side Atomic Accesses*.

Host Side Atomic Access

Host side atomic accesses can be achieved by either of following ways:

- For one parameter, the SPRAM should be accessed by 32-bit-wide data transfers to ensure coherency
- For two parameters only, using the Coherent Dual-Parameter Controller.

indirectly, for any number of parameters, by requesting microcode to coherently access SPRAM in its behalf. The host side atomicity problem becomes, then, a microengine side atomicity problem. Some methods that use this approach to achieve coherency are described in *Section 22.6.3, Multiple parameter coherency methods*.

Microengine Side Atomic Accesses

Microengine single-parameter atomicity

SPRAM should be accessed by 32-bit-wide data transfers to ensure atomicity for 32-bit parameters. This applies either to Host-Microengine coherency or Microengine-Microengine coherency in a dual eTPU engine system.

Microengine dual-parameter atomicity

Microengine has the ability to access two parameters coherently in back-to-back accesses, at random addresses: once it accesses SPRAM, it has priority over Host for another access in the next microcycle (see *Section , SPRAM Arbitration*). Note that it applies **only to Microengine-Host coherency**. For Microengine-Microengine coherency in a dual eTPU engine system, one must use Hardware Semaphores (see *Section , Hardware Semaphores*).

Microengine dual back-to-back accesses are guaranteed to be atomic in relation to Host slave accesses or Coherent Dual-parameter Controller, regardless of semaphore usage: Host or CDC accesses cannot break-up a back-to-back Microengine access, neither Microengine can break a CDC transfer, due to the SPRAM arbitration mechanism described in *Section , SPRAM Arbitration*.

---

t. A microengine access to the SPRAM in the moment CDC is performing the transfer may suffer a maximum of two wait-states.

Atomicity is not guaranteed if microengine enters halt state in the middle of a back-to-back access (see *Section , Microengine halt state*): Host can access SPRAM while microengine is halted in the middle of a back-to-back access.

Microengine Side Multiple Atomicity

Hardware Semaphores must be used for Microengine-Microengine coherency (more than 1 parameter) since two or more accesses from one Microengine are not atomic with respect to the other.

For multiple Microengine-Host coherency, the software methods described in *Section 22.6.3, Multiple parameter coherency methods*, or similar ones, must be used. Some of these methods rely on the fact that parameter access of a thread is atomic in relation to another thread in the same engine, since a thread cannot be suspended (pre-empted).

For 1 parameter coherent access, or dual-parameter coherency between only one Microengine and Host, the alternatives shown in previous sections apply.

Coherent Dual-parameter Controller (CDC)

Dual-parameter coherency is supported by a Coherent Dual-parameter Controller hardware (CDC), which contends with microengine for SPRAM access. CDC atomically transfers, upon Host's command, two parameters from one area of the SPRAM to another. One area is a temporary (buffer) area, where the two parameters are directly read or written by the Host. This temporary area has to begin in an SPRAM address multiple of 2 words, and the two parameters must be sequential. The other area is the channel parameter area where the microcode normally accesses the parameters, usually with the channel relative address mode (see *Section , SPRAM Addressing Modes*). In this area, the parameters transferred by CDC don't have to be sequential. A transfer from the temporary area to the channel area, when the Host sends data to the channel, is called a **write transfer**. Inversely, in a **read transfer** the parameters are copied from the channel area to the temporary area (channel to Host).

Coherency is guaranteed by the SPRAM access contention rules implemented in the SPRAM arbiter (see *Section , SPRAM Arbitration*). CDC transfers are coherent in respect to the two engines, so the target parameters in the channel area may be shared by channels on them both. During CDC operation, the Host may suffer from 3 up to 11 system clocks wait states(u), and the Microengine(s) may suffer up to 2 microcycle wait-states(v). CDC accesses are atomic with respect to Microengine(s) accesses to the SPRAM. Even when neither engine is in TST, CDC may suffer up to 4 system clock internal wait-states from SPRAM arbiter, meaning 9 slave wait-states to Host, so that it does not break atomic back-to-back accesses from microengine(s). CDC also cannot break TST preload accesses. Host can initiate CDC back-to-back transfers: there is no need of idle slave cycles between two transfers.

---

u. The maximum number of Host wait states on CDC occurs when both microengines overlap their TSTs, delayed 3 system clocks from each other.

v. One microcycle takes two system clocks. Microengines get wait-states in multiples of microcycles, while Host and CDC wait-states are multiples of system clocks.

CDC Programming

The Coherent Dual-parameter Controller Register (see *Section , ETPU_CDCR – eTPU Coherent Dual-Parameter Controller Register*) is used to configure and initiate CDC transfers between the temporary area and channel parameter area. Host asserts STS bit in order to start the data transfer. CDC then contends for the SPRAM and starts the transfer. When the data transfer is complete, STS returns to 0. Host receives wait-states for writing STS = 1 while CDC contends for SPRAM and during the transfer. The write access ends when CDC finishes the transfer. Host receives wait-states during the CDC transfer. If Host writes ETPU_CDCR with STS = 0 or does not write the STS byte, the CDC transfer does not occur. CDC programming can be summarized as follows:

1. If it is a write transfer, i.e., from Host to channel, write the two parameters into temporary area.
2. Write ETPU_CDCR with STS = 1 and the remaining CDC programming parameters: parameter width (32 or 24 bits, field PWIDTH), transfer direction (read or write, field WR), temporary parameter area base address (field PBBASE), and the absolute addresses of the parameters to be transferred (concatenation of the fields CTBASE and PARAM0/1).
3. If it is a read transfer, i.e., from channel to host, read the two parameters from the temporary area into Host memory/registers.

Hardware Semaphores

eTPU provides Hardware Semaphores accessible by the Microengine only. It is the responsibility of the application to ensure proper use of the semaphores (i.e., agree upon a specific semaphore and use it properly, to ensure coherency).

The eTPU microinstruction set has support for locking and freeing the semaphores, described in *Section , Semaphore operations*, and this is the only way to access them.

There are four semaphores available, which reduces the amount of collisions by assigning unrelated data transfers to different semaphores. Semaphores are used for parameters which can be shared by channels in different engines, and for engine-to-engine synchronization. Semaphores are also the only way to ensure coherent access to parameters shared between the two Microengines.

Attempting to lock one semaphore (even not successfully) frees the other locked by the same engine, ensuring one can lock just one semaphore at a time. That prevents deadlock conditions between the two engines.

Microcode END command or engine being in idle state (no thread executing) automatically releases all semaphores from one engine side, even if a semaphore lock is done in parallel. However, it is recommended to write the microcode in a way which locks semaphores for the shortest required period, and frees them without waiting for the END command, to improve the performance of the other microengine. Semaphores are free after reset. An engine can only free a sempaphore locked by itself.

Semaphore lock requests are always **non-blocking**, in the sense that they do not suspend the requester in case the semaphore is already locked. The semaphore status after the lock request—indicating if it was successfully locked or not—must be tested through the SMLCK microengine branch condition (see *Section , Branch Conditions*).

SPRAM Arbitration

Up to four entities can access SPRAM:

- Two Microengines (in a dual eTPU engine system)
- The Coherent Dual-parameter Controller (CDC)
- The Host CPU (direct memory-mapped access)

The following rules specify the access priorities for contended access. They keep compatibility with the TPU3 dual-parameter access atomicity, but only between the microengine and CDC (not Host accesses through slave bus).

1. Microengine accesses from the two eTPU engines are interleaved between each other, but not with Host or CDC accesses;
2. The eTPU microengine(s) gives priority for SPRAM accesses to either the Host CPU or the CDC under any of the following conditions:
   a) The microengine has completed accessing the second parameter in a back-to-back SPRAM access(w).
   b) The SPRAM was not accessed during the last arbitration slot for the microengine and the host does not loose the access to the other engine in the current arbitration slot(x).
   c) CDC is transferring data, after its first (read) access. Note that the CDC can be in middle of a data transfer of another pair of parameters, unrelated to the ones that microengine tries to access.
3. The eTPU microengine takes priority for SPRAM accesses under either of the following conditions:
   a) The Host CPU or CDC has done a data transfer during the last access arbitration slot for the engine[x]. Also, the Host CPU does not hold a pending access against the other eTPU microengine.
   b) The microengine is arbitrating for the access of its second parameter in a back-to-back access[w]. All pairs of back-to-back parameter accesses are coherent with respect to Host and CDC (not to the other microengine).

The direction (read or write) of any individual access by Host or microengine is irrelevant to the arbitration. The use of Normal or PSE SPRAM area by the Host is also irrelevant to the arbitration.

The first parameter preloading in a TST is considered first access by the arbiter, regardless of any access made at the END microinstruction of the previous thread, i.e.: the last access of a thread and the first preload are never considered a back-to-back access. On the other hand, the TST preload accesses are considered back-to-back and are, therefore, atomic with respect to Host or CDC.

Note: *The Zero SPRAM operation (see Section , Zero SPRAM operation) is considered an SPRAM access for arbitration purposes both on writes and reads; the fact that read SPRAM data is discarded is irrelevant for arbitration.*

---

w. If microengine tries to access the SPRAM in the following microcycles, the third and fourth consecutive accesses are considered the first and second of a new back-to-back dual access.

x. The microengine access slot is between its own T4 and T2 edges (see *Section 22.7.1, Microcycle and I/O timing*).

22.5.5 Enhanced Channels

Enhanced Channels comprise hardware support for input digital signal processing and output signal generation. Each Channel is associated with one input and one output signal. Enhanced Channel logic is combined with Function microcode (and optionally Angle Mode logic) to implement Channel I/O functionality.

eTPU's Enhanced Channels are capable of **dual action**, meaning that each channel logic can handle two events at different times and/or cause two separated actions—these actions and events can be mutually dependent (with the first either blocking or enabling the other), or both independent, depending on the programmed **Channel Mode**.

Each Enhanced Channel contains event logic containing two **Event Register sets**, each set supporting one input and/or output action, the pair implementing dual-action support. Each Event Register set contains two 24 bit registers: Match and Capture. The **Match register** holds the pending match value which is compared against one of the two time bases by an equal-only/greater-equal comparator. The **Capture register** captures one of the two time bases as a result of a Match or Transition detection. Service Requests are issued on particular combination of match and capture events, defined by the selected Channel Mode.

In the context of the eTPU channels, a **Match** is a comparison between a time base value and a channel Match register. If those two values are coincident, or the time base value is greater than the value of the Match register, a **Match Event occurs**. Depending on the channel mode of operation and current state of the channel logic, the match event may be recognized, i.e., change the state of the channel, or be ignored. A match event recognized by the channel logic is called a **Match Recognition**. Match Recognitions can cause, also depending on Channel Mode and current state, the channel to request service, configuring a **Match Service Request**.

eTPU uses two kinds of comparator to assert a Match Event: an **Equal comparator**, in which both the Match Register and the value of the selected time base must match exactly, and a **Greater-Equal comparator**. The Greater-Equal comparator considers any time base value between the range [N: N+0x800000-1] as a valid match against the value of N in the Match Register, even when the value N+0x800000-1 wraps around the point of origin (0x0). Refer to *Figure 401* for an illustration of the matching values on a Greater-Equal comparator.

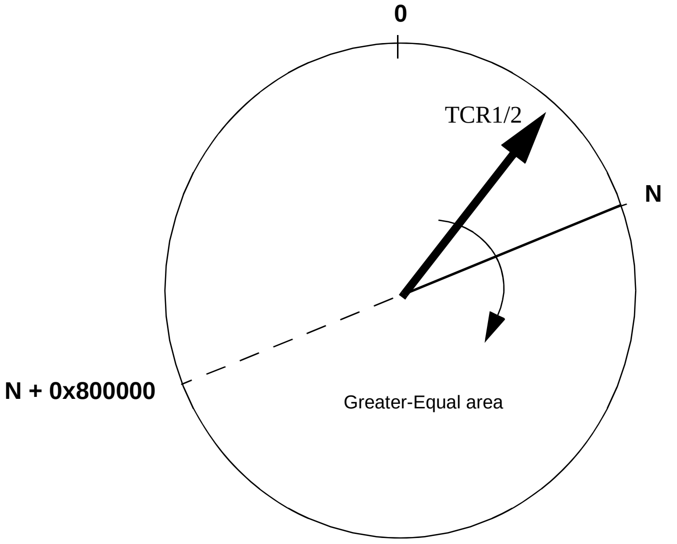
The second source of events for the eTPU channel is a **Transition** detected at the corresponding channel's input signal. Two distinct Transition detections can be programmed individually for each channel, allowing recognition of all possible combinations of edge detection. It is also possible to check the sampled state of an input signal upon the occurrence of a Match: the sampling of the expected value is treated as a Transition, even if the input signal did not necessarily toggled at the time of the Match, or at any time at all.

Like Match Events, **Transitions Events** may or not be recognized by the channel logic. When they are, a **Transition Detection** occurs. As well as Match Recognitions, Transition Detections can issue a Channel Service Request, depending on Channel Mode and current state.

Transition Detections and Match Recognitions are sometimes simply called Transitions and Matches throughout this document, for short.

Input and output signals can be processed separately by the channel logic and microcode, and can also be combined such that Matches and Transitions are used to cause **output signal actions**. The output signals can also be directly controlled by microcode. Many event combinations are allowed for a channel, given the possibility of configuring pairs of matches and transitions for the dual-action logic, where each event is able to block or enable the next one. There is a full set of Channel Modes described in *Section , Channel Modes*, exploring all the capabilities mentioned here.

Each channel has its own set of registers and flags. They are selected, and made accessible to the Microengine, according to the value written into the microengine **CHAN Register** that points to the desired channel. Every time the CHAN register is written, even if with the same previous value, a channel is selected and its flags and registers are updated. For further detail, see *Section , Channel Selection Register – CHAN*.



NOTE: the value opposed to N (N+0x800000) does not cause a match.

Figure 401. Greater-Equal Comparator

Beyond the request of services due to the signal and timing internal to each channel, one eTPU channel microcode can explicitly request service from another channel through the microengine **LINK Register**. A microcode write to the LINK Register asserts a service request to the channel whose number matches the contents of LINK. Refer to *Section , Channel Link* for a complete description of this mechanism.

These service requests originated in the eTPU Enhanced Channels (either time base match, input signal transition, or link service request) result in a call to the corresponding channel service routine, which is the sequence of microinstructions that is called a **Thread**. For further detail, refer to *Section 22.5.1, Functions and threads*.

In addition to Event Logic, each Channel has an Output Buffer Enable signal, controlled by microcode, and an Enhanced Digital Filter, which eliminates spurious glitches on input pin signal. Output Buffer Enable is meant to control output MCU pad signal driver.

A high level diagram of Channel logic and registers is shown in *Figure 402*.

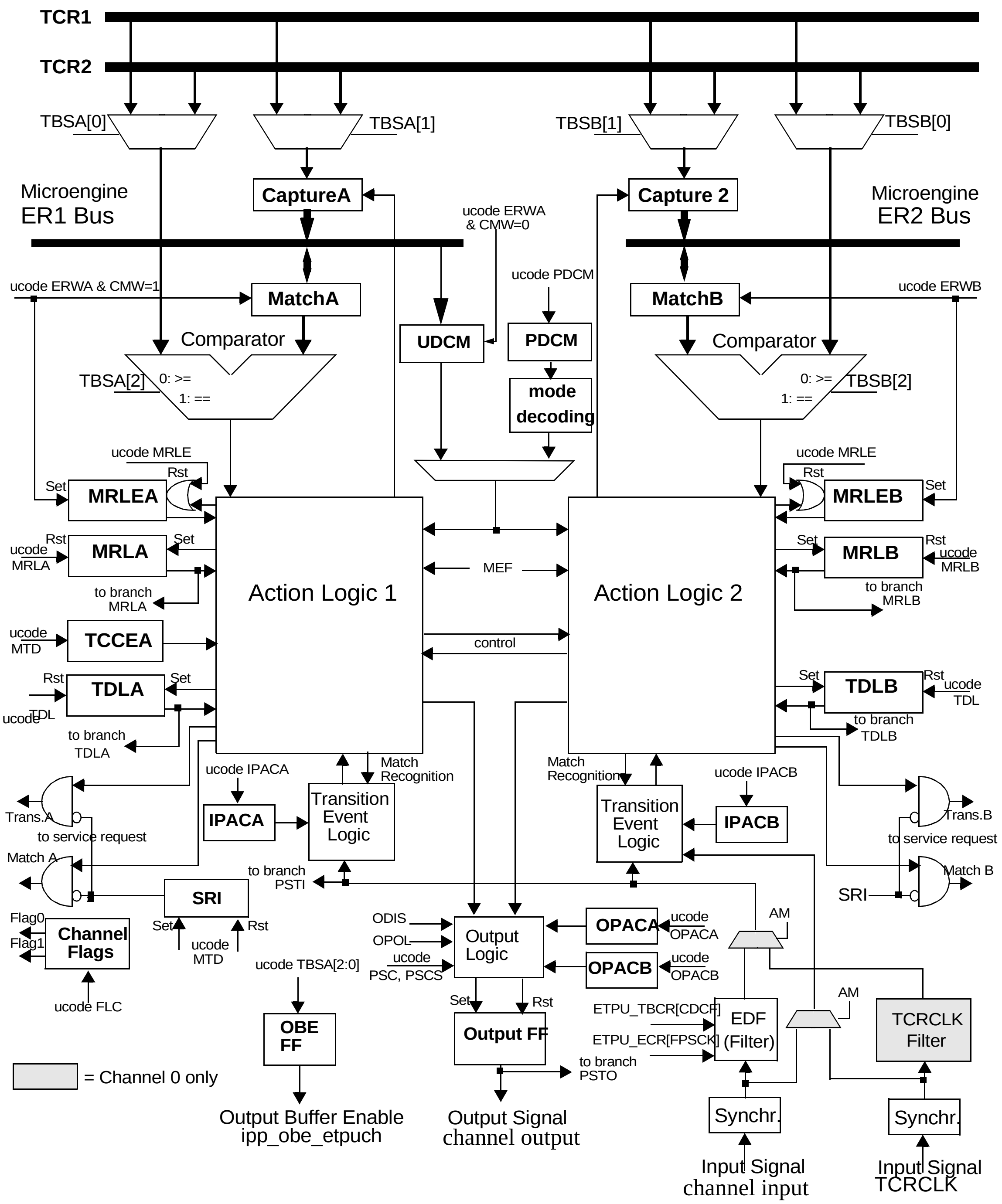


Figure 402. Channel Logic Block Diagram

Channel Registers and Flags

Channel configuration and control registers can be divided in the following groups:

- **Host Configuration and Control registers**, which define channel Function and parameter allocation in SPRAM, input signal filtering, manage Host interrupts, and are used for Host Service Requests; they can only be accessed by Host, except for the Function Mode bits which can be also tested by microcode.
- **Event Registers**, which can only be accessed by eTPU Microengine, through dedicated Channel Control microinstruction operations (see *Section , Channel control and configuration microoperations*); these registers are directly used to implement channel functionality, and include channel event status latches which can be directly tested by Microengine branch instructions.
- **Pin Control registers**, which basically define pin state and transition polarity (but not input signal filtering); they are accessible only by dedicated Channel Control microinstruction operations.
- **Link registers**, which implement the channel link mechanism that allows one channel to request service to another one; they are accessible only by microinstruction operations.
- **General Channel registers**: CHAN, SRI, Flag0/1, PDCM, UDCM.

Most of those registers are channel exclusive, i.e., there is one copy of them for each channel. Microcode can access registers from only one channel at a time. The Channel Selection (CHAN) register (see *Section , Channel Selection Register – CHAN*), accessible only by microcode, defines the channel whose registers are being accessed, with exception of link register and function mode. CHAN register assumes the value of the channel to be serviced at the beginning of TST.

The Service Request Inhibit (SRI) register controls the generation of Service Requests on matches and transitions, also affecting channel logic behavior. For a full description see *Section , SRI – Match/Transition Service Request Inhibit Latch*. Flag0/1 are used to select channel service threads based on channel software state. See *Section , Flag1,Flag0 – Channel "state resolution" flags*, for more details.

Host Configuration and Control registers are described in *Section 22.4.7, Channel configuration and control registers*.

Time Base configuration is common to all channels, and described in *Section 22.5.6, Time Bases*. Time Base selection for matches and captures, however, is individual to each channel, and is part of the Event Registers.

Link registers are described in *Section , Channel Link*.

The following sections describe the Event Registers and Pin Control registers.

ER – Event Registers

Each channel contains two identical Event Register sets, named ERA and ERB, corresponding to the two actions supported. Each Event Register set contains:

- A 24-bit Match register (Match A or Match B), which holds a match value. This value is compared against the selected match time base (TCR1 or TCR2).
- A 24-bit Capture register (CaptureA or CaptureB), which samples the selected capture time base (TCR1 or TCR2)
- A Time Base Selection register (TBSA or TBSB)
- A Match Recognition status flag (or latch) (MRLA or MRLB)
- A Match Recognition Enable latch (MRLEA or MRLEB)
- A Transition Detection flag (or latch) (TDLA or TDLB)
- A Transition Continuous Capture enable (TCCEA only)

ERA and ERB are associated with the first and second events in double action modes, not necessarily in that order. The order of Match events associated with ERA and ERB depends on the programmed channel mode, the MatchA and MatchB values, and the timebases selected by TBSA and TBSB. Similarly, the order of Transition events associated with ERA and ERB depends on the programmed channel mode, and the transition detection selected by IPACA and IPACB.

These registers are directly or indirectly accessed by the microcode. TBSA and TBSB registers are defined in *Section , TBSA and TBSB – Time Base Selection Registers*. The other registers are explained in *Section , Match Recognition* and *Section , Transition Detection and Time Base Capture*.

Access to the Event Registers is qualified by the channel currently selected by the microengine (i.e., the channel value currently in the CHAN register). During the channel transition period (automatic CHAN assignment), or whenever CHAN is written by microcode, Capture values of the new selected channel are sampled into Microengine registers ERTA and ERTB, therefore becoming visible to the microcode. At the same time, updated values of MRLA, MRLB, TDLA and TDLB are sampled into the branch logic, making the register values and the flags coherent with respect to each other and with the thread selected by the Scheduler(y).

Note: *The Function Mode bits are also sampled from the Host interface on Time Slot Transition, so that they remain constant to microengine even when Host changes them.*

During service, the microcode can access updated values of the Event Registers of any channel by writing the channel number to CHAN. Writing CHAN with the same value (CHAN := CHAN) updates ERTA and ERTB with the new captured values, the branch logic with updated MRLA/B and TDLA/B flags. Writing CHAN with a different value does the same with the values from the newly selected channel.

Match values are also accessed through ERTA and ERTB Microengine registers, which are copied to/from the channel MatchA and MatchB registers by specific microinstruction operations.

Microcode writes to the flags and selections (MRLA/B, TDLA/B and TBSA/B) are immediately effective to the channel. The MRLA/B and TDLA/B branch conditions are also

---

(y) The thread selected is determined by the Entry Point which, in turn, is determined partially by the channel latches. See *Section , Entry point address generation*.

immediately reset when their correspondent flags are reset by microcode. Match registers are indirectly written by microcode through ERTA/B. MRLEA/B is unconditionally asserted when respective Match register is updated from ERTA/B, and its negation is immediate.

Table 385 summarizes Event Registers accesses.

Table 385. Event Registers microcode accesses

Register	Access Type	Sampled from channel	Update to channel	Microcode fields[1]	Reset value[2]
CaptureA, CaptureB	read through ERTA/B	to ERTA/B on CHAN assignment	no	T2ABD	n.a.
MatchA, MatchB	read and write through ERTA/B	to ERTA/B by microcode	from ERTA/B by microcode	ERWA, CMW, ERWB, T4ABS	n.a.
MRLEA, MRLEB	write to 0 (negate) directly; write to 1 (assert) upon MatchA/B update from ERTA/B	no	immediate	MRLE, ERWA, ERWB	0, 0
TBSA, TBSB	write only	no	immediate	TBSA, TBSB	000, 000
MRLA, MRLB	flag test on branch, write to 0 (negate) only	on CHAN assignment	immediate	BCC (test) MRLA, MRLB (reset)	0, 0
TDLA, TDLB	flag test on branch, write to 0 (negate) only	on CHAN assignment	immediate	BCC (test) TDL (reset)	0, 0
TCCEA	write only	no	immediate	MTD	0

1. See *Section 22.5.9, Microinstruction set*.
2. n.a. means that value of the register is undetermined after reset.

MatchA and MatchB Registers

MatchA and MatchB registers hold a match value to be compared against the selected channel time base. A match value can only be written into the Match register by microcode, through ERTA/B microengine registers (see *Section , Write Channel Match and UDCM Registers*). Microcode can also read the Match register as a special T4ABS source operation, when T4ABS = 0101, and the source for T4ABS is selected from the second register set. In this operation, MatchA/B registers are copied into ERTA/B registers (see *Section , Selecting sources and destination*). For more information on time base matches, see *Section , Match Recognition*.

CaptureA and CaptureB Registers

CaptureA and CaptureB registers capture the selected channel time base. CaptureA/B registers cannot be directly written or read by microcode. During the Time Slot Transition (TST) or during CHAN assignment, CaptureA/B registers are copied into ERTA/B microengine registers. For more information, see *Section , Transition Detection and Time Base Capture*.

TBSA and TBSB – Time Base Selection Registers

TBSA/B are 3-bit registers which have the following effect on channel configuration:

- Selection of the timebase (TCR1 or TCR2) to be compared against the match values in MatchA and/or MatchB registers.
- Selection of the timebase (TCR1 or TCR2) to be captured in the CaptureA and/or CaptureB registers by a match or transition detection event.
- Selection of comparator mode to be used with MatchA and MatchB registers: equal-only or greater-equal.

After reset TBSA/B are 000. *Table 386* shows values of TBSA and TBSB for configuration selection. Note that the time base selection for capture is independent of the time base selected for matches.

Table 386. TBSA/B Programming States

TBS bit	2	1	0
Bit value	Comparator selection	Capture time base selection	Match time base selection
0	greater or equal	TCR1	TCR1
1	equal-only	TCR2	TCR2

TBSA/B are written through the microcode fields TBSA/B (see *Section , Comparator and time base selection*). Note that microcode field TBSA is also used to control the OBE pin control register (see *Section , Pin Control Registers*), which is separate from the TBSA register.

MRLA/B – Match Recognition Latches

See *Section , MRLA/B – Match Recognition Latches*.

MRLEA/B – Match Recognition Latch Enable

See *Section , MRLEA/B – Match Recognition Latch Enable*.

TDLA/B – Transition Detection Latch

See *Section , TDLA/B – Transition Detect Latches*.

TCCEA – Transition Continuous Capture Enable

See *Section , TCCEA – Transition Continuous Capture Enable*.

Pin Control Registers

Pin Control Registers are replicated one per channel, accessed only by microcode and qualified by the CHAN register in the same way as Event Registers. *Table 387* lists Pin Control Registers, explained in following subsections, and their accesses.

Table 387. Pin Control Registers microcode accesses

Register	Access Type	Sampled from channel	Update to channel	Microcode fields[(1)]	Reset value
IPACA, IPACB	write only	no	immediate	IPACA, IPACB	000,000
OPACA, OPACB	write only	no	immediate	OPACA, OPACB	000,000

Table 387. Pin Control Registers microcode accesses

Register	Access Type	Sampled from channel	Update to channel	Microcode fields[1]	Reset value
PSTI	flag test on branch	no	no	BCC	0
PSTO	flag test on branch, write	no	immediate	BCC (test) PSC, PSCS (set)	0
OBE	write only	no	immediate	TBSA values 1000,1001	0 (negated)
PSS	flag test on branch	on CHAN assignment	no	BCC	n.a.[2]
PRSS	flag test on branch	on CHAN assignment	no	BCC	n.a.[3]

1. See *Section 22.5.9, Microinstruction set*.
2. PSS is PSTI or PSTO sampled on CHAN assignments and at thread start.
3. PRSS is PSTI sampled on channel service request.

IPACA,IPACB and OPACA,OPACB – Input and Output Pin Action Control Registers

These registers determine the transition detections and output pin actions which takes place due to match or transition events. Each field is three bit wide. After reset, the IPACA/B and OPACA/B registers are set to 000. IPACA and IPACB registers are mutually independent and have identical encoding, and so do OPACA and OPACB. *Table 388* shows IPAC and OPAC encoding. Note that transition detections are independent from the output actions, but some options of output actions are triggered by transition detections. Output actions can also be triggered by Matches. For a detailed definition of Transition Detections, see *Section , Transition Detection and Time Base Capture*.

IPACA/B define the Transition events treated by channel logic. Although the name “transition” is generically used for the transition detections, IPAC options 100 and 101 do not really detect transitions: they actually sample the state of the input signal at the occurrence of the corresponding Match (Match A used for IPACA, Match B used for IPACB).

Table 388. IPACA/B and OPACA/B Encoding

Value	IPAC meaning	OPAC meaning
000	Do not detect transitions	Do not change output signal
001	Detect rising edge only	Match(1) sets output signal high
010	Detect falling edge only	Match(1) sets output signal low
011	Detect either edge	Match(1) toggles output signal
100	Detect input signal = 0 on Match(1)	Transition sets output signal low
101	Detect input signal = 1 on Match(1)	Transition detection sets output signal high
110	reserved	Transition detection toggles output signal
111	n.a.(2)	n.a.(2)

1. Match A is used for IPACA/OPACA, and Match B for IPACB/OPACB.
2. On the microinstruction fields IPACA/B and OPACA/B this value is neutral, meaning that IPAC/OPAC register values are not changed.

Output Pin Control Logic and Pin State Output Register – PSTO

The output signal control logic uses OPACA/B, the Pre-defined Channel Mode (PDCM) and the User Defined Channel Mode (UDCM), and the microcode Pin State Control (PSC and PSCS) fields. It is responsible for setting the Pin State Output (PSTO) register to the specified logic value required by microcode, by input events, or by Match A and/or Match B events. The PSTO register stores the driven pin state determined by the Pin Control logic. The Output Buffer Enable signal, if used at MCU integration, must be set by microcode (using TBSA field) to make the pad propagate the PSTO register output to the actual pin.

PSTO register output also goes to the microengine branch logic, enabling branching on the driven pin state (see *Figure 403*). PSTO is set to 0 on reset.

The PSC and PSCS microcode fields are used for setting the PSTO register to a fixed value, or to the value specified by the OPACA or OPACB microcode field, as shown in *Table 389*.

Table 389. PSC and PSCS encoding

PSC	Output Pin Action
00	Force pin state according to OPACA (PSCS = 0) or OPACB (PSCS = 1).
01	Force pin high.
10	Force pin low.
11	Do not change pin state.

For details refer to *Section , Channel control and configuration microoperations*.

On occurrence of match recognitions or transition detections, the pin control logic sets PSTO value according to the event number (Match A/Transition A or Match B/Transition B) and the contents of OPACA/IPACA or OPACB/IPACB registers. There are cases in which two match or transition events may occur at the same time, each of them trying to force a different pin action. The resolution of the selected match event which sets the value depends on the Pre-defined Channel Mode (PDCM) register and the User Defined Channel

Mode (UDCM). For details refer to *Section , Match/Transition Pin Action Conflict Resolution*.

PSTI and PSS – Pin State Input and Pin Sampled State Registers

During the time slot transition period, or whenever the CHAN register is written by microcode, the filtered(z) input signal PSTI or output signal PSTO (selected by the ETPU_CxCR bit ETPD) is sampled into the branch logic as the PSS flag (see *Figure 403* and *Table 457*). The microcode can then branch on either the currently driven (PSTO) or input (PSTI) pin state, or on sampled pin state (PSS, which is stable as long as CHAN does not change).

Note: *If a transition occurs simultaneously (after the filter) with the CHAN assignment, PSS samples the new pin value. Therefore, if TDLA/B is cleared simultaneously with the assignment CHAN := CHAN, the occurrence of a transition at this very moment can still be tested with PSS.*

PRSS – Pin Request Service Sample

Channel logic can, depending on its state and programmed mode, request service to the eTPU microengine (see *Section 22.5.1, Functions and threads*). When the channel logic issues a service request, the filtered input signal PSTI is sampled into an internal channel flag PRSS. There is one such PRSS flag for each channel (see *Figure 403*). Channel PRSS keeps its value until all its service request sources are cleared and until a new service request rises.

The channel PRSS flag is sampled into the branch logic as the PRSS flag (see *Table 457*) during the time slot transition period, or whenever the CHAN register is written by microcode.

OBE – Output Buffer Enable control latch

OBE latch drives the Output Buffer Enable signal, which can be used (depending on MCU integration) to control the output signal pad driver. Channel output comes disabled from reset. If ipp_obe_[1|2]([0-31])(aa) from eTPU is used on MCU integration and a signal is desired to be output in a channel, OBE signal must be set by microcode. Microcode field TBSA is used to set/reset the Output Buffer Enable control when microcode field TBSA bit 3 is 1, according to *Table 390*

Table 390. TBSA Output Buffer control

Microcode TBSA[2:0]	meaning when TBSA[3] = 1
000	enable output buffer
001	disable output buffer
111	do nothing
other values	reserved

(z) The filter can be bypassed.

(aa) Output Buffer Enable: there is one independent OBE signal for each channel.

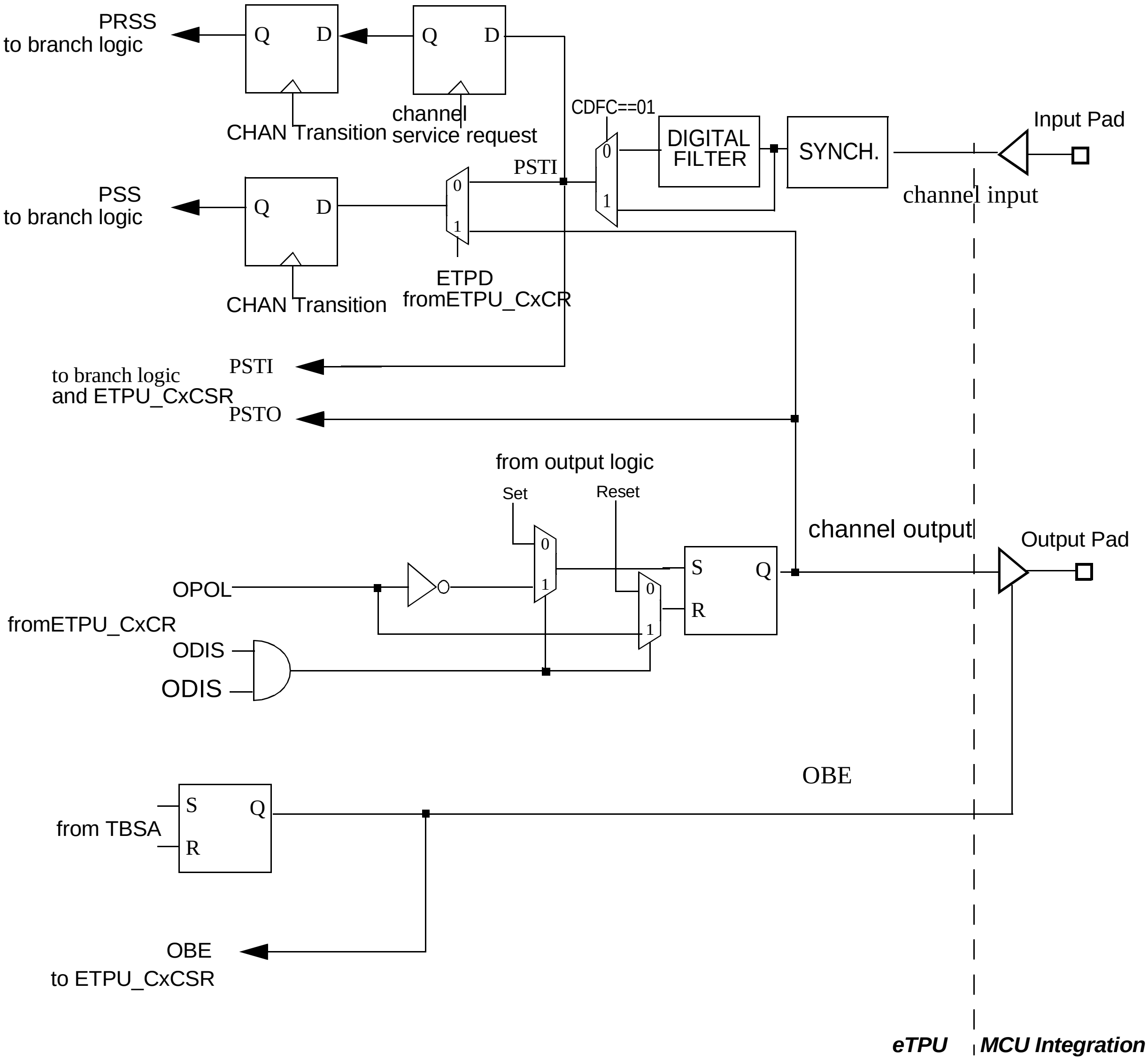


Figure 403. Pin State Input/Output Logic

General Channel Registers

These registers control other aspects of channel logic. Except for CHAN, they are unique per channel. *Table 391* summarizes the registers and access options.

Table 391. General Channel registers microcode access

Register	Access type	Sampled from channel	Update to channel	Microcode fields[1]	Reset value
CHAN	read/write	n.a.[2]	n.a.[2]	T4ABS, T4BBS, T2ABD	defaults to serviced channel at thread start
PDCM	write only	no	immediate	PDCM	1100 (sm_st)
UDCM	write only	no	from ERTA by microcode	CMW, ERWA	parameter value defined at integration
SRI	write only	no	immediate	MTD	1
Flag1,Flag0	branch flag test, write	no	immediate	BCC,FLC	0, 0

1. See *Section 22.5.9, Microinstruction set*.
2. CHAN is common to all channels in the engine.

Channel Selection Register – CHAN

CHAN is the register that holds the number of the channel that qualifies the context of most Channel Registers, including Pin Control and ER accesses, and is common to all channels in a same engine.

When a thread starts to be executed, the contents of CHAN register is automatically updated on Time Slot transition to the number of the channel to be serviced. Serviced channel is constant during channel servicing, but the selected channel can be changed any time by microengine writing into CHAN register.

Some microinstructions are affected by the serviced channel instead of CHAN. These are:

- Conditional branch using LSR (see *Section , LINK Register*) or Function Mode (*Section , ETPU_CxSCR – eTPU Channel x Status Control Register*).
- Negate channel flag LSR, Interrupt CPU and Data Transfer Request (see *Section , Channel interrupt and data transfer requests*).

When CHAN register is written, accesses are qualified by the new CHAN register value from the instruction following CHAN assignment on, except CaptureA/B sampling into ERTA/B and Match register writing from ERTA/B (see *Section , CHAN assignment, Read Match and ERWA/B*).

Writing CHAN (including with the same value, CHAN:= CHAN) updates ERTA and ERTB with the new captured values, the branch logic with updated MRLA/B and TDLA/B flags.

Table 392 shows the commands, flags and registers selected by the CHAN register value

Table 392. CHAN-selected features

Feature used	Selected by CHAN
Channel-relative SPRAM access	YES
Branch using PSS, PRSS, PSTI and PSTO channel flags	YES

Table 392. CHAN-selected features

Feature used	Selected by CHAN
Branch using MRLA/B, TDLA/B, Flag0/1(1)	YES
Branch on all other conditions(2)	No
ERTA/B value	YES
Configure (selected) channel	YES
Channel commands applied to: MRLA/B, TDLA/B, TBSA/B, IPACA/B,OPACA/B, PSC, PSCS, OBE, PDCM, MRLE, Flag0/1	YES
Channel command: set/reset SRI	YES
Channel command: write to Match registers (ERWA/B)	YES(3)
Channel command: read Match registers into ERTA/B	YES
Clear LSR	No
Channel interrupts and data transfer requests (CIRC)	opTionally

1. In TPU, these conditions retained the old values.
2. Refer to *Section , Branch operations*
3. If write Match (ERWA/B) occurs at the same time of a CHAN assignment, the channel which is target of the write is the one selected by the new CHAN value. See *Section , CHAN assignment, Read Match and ERWA/B*.

PDCM – Predefined Channel Mode

PDCM determines the channel mode assigned to the channel. Channel mode defines much of the channel logic behavior, specially how matches blocks and enables transitions and vice-versa, as well as occurrence of time base captures and service requests based on matches and transitions. For a complete description see *Section , Channel Modes*.

PDCM is a 4-bit register set by the microcode field of the same name (see *Section , Predefined channel modes*), and cannot be read or tested in branch instructions. *Table 393* relates the PDCM value with channel modes. The second column specifies the mnemonic used to reference the mode, introduced in *Section , Channel modes overview*. There is one PDCM for each channel, initialized with 1100 on reset.

PDCM is also used to select the User Programmable Channel Mode. If this selection is made, the channel behavior is defined by the settings of the UDCM register (see *Section , UDCM – User Defined Channel Mode*).

Table 393. PDCM encoding

PDCM	Channel mode
0000	em_b_st
0001	em_b_dt
0010	em_nb_st
0011	em_nb_dt
0100	m2_st
0101	m2_dt
0110	bm_st
0111	bm_dt
1000	m2_o_st
1001	m2_o_dt
1010	User Defined Channel Mode
1011	reserved
1100	sm_st(1)
1101	sm_dt
1110	sm_st_e
1111	n.a.(2)

1. This is the reset value, also compatible with TPU channel behavior.
2. This value is used as a neutral (do not change) value in the PDCM microinstruction field.

UDCM – User Defined Channel Mode

UDCM holds the control signals that define the channel logic behavior in terms of Match and Transition blocking and enabling, captures and service requests triggered by events. The register layout is defined in *Figure 404*. The individual fields are defined in *Section , Channel Mode Logic and Event Flags*. There is one UDCM register for each channel, which can be independently programmed.

UDCM can only be written into the Match register by microcode, through ERTA/B microengine registers (see *Section , Write Channel Match and UDCM Registers*).

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved	Reserved	MSR		MCAP	M1ET	M1EM2	M1BM2	M2BM1	M2BT	T1BM1	T2BM1	TBM2	T1ET2	TSR	TCAP
W																

Figure 404. UDCM Register

SRI – Match/Transition Service Request Inhibit Latch

SRI blocks channel service requests due to the assertion of MRLA/B and/or TDLA/B. SRI does not affect recognition of Link Service Requests or Host Service Requests, neither

MRLA/B or TDLA/B microcode branch tests nor Entry Table selection(ab). SRI is asserted during reset and is controlled by microcode field MTD.

To unburden the microengine, SRI asserted configures a channel “dumb” regarding the servicing of match and capture channel service requests. Even with SRI = 1, TDLA/B and MRLA/B can still be asserted, and the level specified by the OPAC (Output Pin Action Control) registers will be output to the pin.

Flag1,Flag0 – Channel “state resolution” flags

Each channel has a pair of flags, simply called Flag0 and Flag1, that can be set/reset by microcode through microinstruction field FLC. FLC sets/resets Flag0/1 of the channel selected by CHAN. These flags can be tested by microcode, and are also used to resolve the microcode entry point for the channel service (see *Section , Entry points*). Flag0 and Flag1 are, so, typically used for fast state resolution. FLC microinstruction field also allows Flag1,Flag0 to be copied from selected bits of P register high byte, which is also meant to be used to hold application state. Flag0 and Flag1 are both zero out of reset.

Match Recognition

The match operation is performed every microcycle by comparing the channel MatchA and MatchB registers against the value of the TCR bus specified for each match. TCR1 or TCR2 bus is selected according to TBSA and TBSB fields. Both results have effect on the next clock cycle (see *Section 22.7.1, Microcycle and I/O timing*).

A Match A/B event is qualified by a set of match enabling conditions to the Match Recognition Registers MRLA/B. To recognize the match and assert these registers, the following match enabling conditions are required:

- For IPACA/B = 0xx, Match Enable Flag (MEF), qualified by the channel currently being serviced must be asserted. For IPACA/B = 1xx, Match A/B is always enabled (even during Time Slot Transition (TST)), regardless of the state of the Match Enable Flag (MEF). See *Section , MEF – Match Enable Flag* for the conditions of MEF assertion.
- Match Recognition Latch Enable 1/2 (MRLEA/B) is asserted. A match event recognition may only occur if its corresponding MRLEA/B bit is set, which only happens upon a write to a channel match register by the microcode, copied from ERTA/B. MRLEA/B is negated when the respective match occurs or, in some double match channel modes, when a match for the other Match register occurs. It ensures that the greater-equal comparison will not cause additional matches(ac).
- In selected modes (see *Section , Channel Modes*), the particular conditions of MRL and TDL flags of the other event, i.e:
  - MRLA, TDLA enable or block MRLB;
  - MRLB, TDLB enable or block MRLA.
- The respective MRL is negated.
- In selected modes (see *Section , Channel Modes*), the state of its respective TDL flag.

If the Match A and/or Match B conditions are met, the channel immediately forces the pin state if specified by the appropriate OPACA/B registers (Output Pin Action Control 1/2) and, in some cases, by IPACA/B registers. Refer to *Section , IPACA,IPACB and OPACA,OPACB – Input and Output Pin Action Control Registers*.

---

ab. In TPU, SRI also blocked TDL and MDL branches and enabled any transition to capture time base.

ac. Microcode can also negate MRLEA/B.

If both Match A and Match B events occur at the same time, with conflicting pin actions, the priority over the pin action is mode dependent. For further details on pin action resolution refer to *Section , Match/Transition Pin Action Conflict Resolution*.

MRLA/B – Match Recognition Latches

MRLA/B indicate the recognition of a match event detected by the comparator. They can be asserted either on T2 or T4 (see *Section 22.7.1, Microcycle and I/O timing*). Assertion of MRLA/B issues a match Service Request in specific channel modes, depending on previous events and state of SRI. After reset MRLA and MRLB are both negated.

When MRLA or MRLB is asserted, it may change the output signal level according to the Input and Output Pin Action Control registers (refer to *Section , IPACA,IPACB and OPACA,OPACB – Input and Output Pin Action Control Registers*). Assertion of MRLA/B causes a capture of one or two time bases, according to the selected mode capturing scheme (see *Section , Transition Detection and Time Base Capture*).

A match recognition is self-blocking, regardless of Channel Mode: once MRLA (MRLB) has been asserted, it negates its associated MRLEA (MRLEB) register, preventing future match recognitions, until the associated match register is rewritten by microcode. The microcode has to enable new matches by updating the new match value in the MatchA (MatchB) register(ad). In addition, assertion of MRLA/B can block its twin MRLB/A, depending on the channel mode. In some double match blocking channel modes, Match A/B event blocks the occurrence of Match B/A in a “first win” scheme.

It is the transition from 0 to 1 in MRL that causes the Match actions: apart from MRLEA/B negation(s), no action due to a Match occurs if MRL was already set to 1, even if the other MRL assert conditions are satisfied. However, if a Match and a microoperation negating its corresponding MRL occur at the same time, MRL negation by microcode overrides its assertion, but any dependable captures and pin action occurs anyway (if MRL was already negated before), and also the negation of MRLE(s) (the respective one and, in some channel modes, the other, regardless of MRL state before). Note that MRLE must have been set before (by writing a new Match value).

MEF – Match Enable Flag

MEF is a one-bit latch that is unique for all channels in an engine.

MEF can selectively enable assertion of MRLA/B, depending on the IPACA/B field. For IPACA/B = 0xx, MEF = 1 enables assertion of MRLA/B for the scheduled channel during service. For IPACA/B = 1xx, Match A/B is always enabled, regardless the state of the MEF, but it still depends on the other Match recognition conditions. Matches of channels not being serviced are never disabled by MEF.

MEF is not accessible by Microengine or Host. MEF is negated for one microcycle in the middle of the time slot transition period. After two microcycles (plus wait-states) into TST, the ME bit in the entry point is copied to MEF to allow selective enabling of MRL for each thread (refer to *Section , Time slot transition*). MEF is asserted unconditionally soon after a thread ends.

If a channel service needs to postpone a programmed match, MEF assures that microcode wins the race against match event after time slot transition (only for IPAC = 0xx).

---

ad. Before that, microcode should also negate MRLA (MRLB), otherwise an old match may be recognized by the scheduler and serviced as a new one

Note that a match event may be lost during the periods when MEF is negated only if:

- the match comparator is configured for “equal-only” behavior, and
- IPACA/B = 0xx, and
- TCR increments at the rate of system clock divided by 2 or faster.

When the comparator is configured as “greater-equal”, the match event that occurred when MEF was negated may be recognized after MEF is asserted again, due to the “greater than” condition.

MRLEA/B – Match Recognition Latch Enable

MRLEA/B is negated upon the assertion of its respective MRLA/B. In blocking match channel modes it may also be negated together with the assertion of the twin MRLB/A. The MRLEA/B bits ensure that data captured due to the first match event will not be overwritten when MRLA/B is negated: due to greater-equal comparison, the match condition continues to be true, but should not cause another capture event.

In addition to negation by local match event, the microcode can negate both MRLEA and MRLEB, to block pending matches, and also MRLA/B, individually. This action will prevent future match events from the selected channel.

Writing the MatchA/B registers by microcode to schedule the next match values sets MRLEA and/or MRLEB and enables new matches. This setting overrides the MRLE negation conditions due to channel logic or microcode (see *Section , Channel Modes*). By combining write to Match A/B with MRLEA/B negation microinstructions, the microcode can negate one MRLE while asserting the other.

Note: *If the MRLE negation conditions continue after writing MatchA/B registers, the respective MRLE does not keep asserted. For instance, if MRL = 1 and a match is programmed for a time value in the past during a thread with MEF = 1, then MRLE will be cleared soon after MatchA/B is written, even though a match does not occur (because MRL was already asserted, neither captures nor pin toggles occur).*

When the match register is updated (with MRLE already asserted before) and field MRLA/B = 1 (no clear, see *Section , Clear transition/match event registers*) and MRLA/B flag is zero, the eTPU behaves exactly as the TPU, that is: a match that comes concurrently with the rewrite of the match register, matching the old value, sets the MRL, as if the setting of the MRLE due to match register write had precedence over its clear by the match at that moment. After this simultaneous operation, the MRLE value stays at 1, and the captured time base value, if any, reflects the match value.

When the match register is updated (with MRLE already asserted before) and field MRLA/B = 0 (clear MRL, see *Section , Clear transition/match event registers*) and MRLA/B flag is zero, the match captures will occur, the MRLA/B flag will keep negated, and MRLE will stay asserted. If a match is reprogrammed on TCR1 running at T2/T4 timing (TCR1CS = 1, see *Section , ETPU_TBCR – eTPU Time Base Configuration Register*), a match can occur after MRLA/B is cleared, together with the set of MRLE. In this case, both MRL and MRLE will be set, and a match service request will occur if enabled. However, the match happened on the old match value, not on the new (reprogrammed) one. In order to prevent this ambiguity to the code that services the match, it is advisable to clear the MRLE (besides MRL) together with the match reprogramming, avoiding the match on the old value to occur while the new match value is being written. The set of the MRLE due to match reprogramming prevails over the MRLE clear, thus allowing the new programmed match to occur.

Transition Detection and Time Base Capture

Time Base Capture(s) occur when the value of a specified TCR is sampled into the CaptureA and/or CaptureB register. TBSA[1] and TBSB[1] select which TCR will be captured in CaptureA and CaptureB, respectively.

A capture event may occur due to either of the following events:

- The assertion condition of Match Recognition Latch (MRL), even if MRL is simultaneously negated by microcode
- The assertion condition of Transition Detection Latch (TDL), even if TDL is simultaneously negated by microcode.
- Any Transition Event specified by IPACA if both the Transition Detection Latch TDLA and Transition Continuous Capture Enable TCCEA are asserted.

A capture event occurs together with the assertion of MRL or TDL either on T2 or T4 positive edges, and captures the time-base value that caused the match, even if TCR1/2 increments concurrently with the assertion (see *Section 22.7.1, Microcycle and I/O timing*)[ae]. MRLA/B and TDLA may, depending on the channel mode, inhibit the capture of the second event's TCR into CaptureA/B. As a general rule, values captured by signal transitions are not overwritten by values captured by match events.

When the enable bit TCCEA is asserted, captures due to Transition Events also occur after TDLA is asserted. Those captures happen on transition events specified by IPACA, and the TCR value is saved into CaptureA register only.

The capturing scheme is defined by the Channel Mode programmed at register PDCM, or at register UDCM when User Defined Channel Mode is selected. For more information on mode-dependent capture schemes refer to *Section , Channel Modes*.

TDLA/B – Transition Detect Latches

TDLA/B indicate detection of specific transition occurrences on a channel input signal. TDLA and TDLB assertion causes service request in single and double transition predefined modes, respectively. TDLB assertion does not cause Service Request in single transition predefined modes, and TDLA assertion does not cause Service request in double transition predefined modes. In single transition channel predefined modes TDLB can be asserted on the second transition, but does not generate Service Request. Yet on predefined modes, TDLB assertion is enabled only if TDLA is asserted to detect an ordered input signal double transition. All the restrictions above, however, may be overridden by using the User Defined Channel Mode. The IPACA and IPACB registers indicate the programmed edges of the first and second detected transition, respectively.

The sampling of a determined value (0 or 1) on the input signal due the occurrence of a Match is also treated as a “transition”, depending on IPACA/B programming (see *Section , IPACA,IPACB and OPACA,OPACB – Input and Output Pin Action Control Registers*). When using a channel mode where the transition1 is initially blocked and IPACA is programmed to detect such “transitions”, the occurrence of a Match A only unblocks the transition after the sampling. It means that the transition on the first Match (IPACA configurations 100 and 101) is not effective on predefined modes where Transition A is enabled by Match A (m2_st, m2_dt, m2_o_st and m2_o_dt) or user-defined modes with UDCM bit M1ET asserted. A

[ae] In TPU3, when TCR1 was counting at maximum rate of system clock divided by 2, the next value was captured.

Transition A can still happen after the Match A, however, if MatchA register is reprogrammed without clearing MRLA.

TDLA/B assertion conditions initiates a capture event of one or both selected TCR buses. TDLA or TDLB transition event generates a Service Request, depending on channel mode, previous events and the state of SRI. For more information on the service request scheme, refer to *Section , Entry point address generation*, and *Section , Channel Modes*.

Assertion of TDLA/B occurs on either T2 or T4 positive edges. The capture event occurs on the same clock, and captures the time base value present when TDLA/B was asserted[ae]. TDLA and TDLB are negated during reset and may also be negated independently by microcode. TDLA/B is reset by no way other than reset and microcode.

It is the transition from 0 to 1 in TDL that causes the Transition actions: even if TDL assert conditions are satisfied, no action due to a Transition occurs if TDL was already set to 1. However, if a Transition and a microoperation negating TDLs occur at the same time and TDL was already negated, TDL negation by microcode overrides its assertion, but any dependable captures and pin action occurs anyway.

TCCEA – Transition Continuous Capture Enable

TCCEA enables capture from transitions after the TDLA flag is set. TCCEA is negated on reset, so that a capture occurs only when TDLA asserts. TCCEA can be set and reset by microcode only, through the instruction field MTD (see *Section , Disable match and transition service requests*). It can only be set together with inhibiting of the channel service requests (SRI = 1)(af).

When TCCEA is asserted, the transition events specified by IPACA that occur after TDLA is set also cause captures into the CaptureA register only. However, output actions related to transition events are still blocked by TDLA.

Channel Modes

The Enhanced Channels support various modes of operation combining Match A/B recognition and transition detection events which set MRLA/B and TDLA/B. The channel mode is individually set for each channel by eTPU microcode, through the PDCM register (see *Section , PDCM – Predefined Channel Mode*). The PDCM register selects among a set of 13 predefined channel modes, and also a user-defined channel mode.

The order in which events occur, combined with assigned channel mode, establish which following event detections are inhibited and/or enabled, as well as the actions taken: Time Base capture, flag setting (MRLA/B, TDLA/B), match disabling (MRLEA/B), output signal transition, and Service Request. Those channel mode characteristics are fixed in the predefined modes, but can be individually programmed in the user-defined channel mode.

A generic description of channel modes from the usage point of view can be found in *Section , Channel modes overview*. Each mode is named with a mnemonic acronym for terse reference. The individual programmed attributes of the user-programmable channel mode are also described.

The modes are used differently for input and output signals, as explained in *Section , Predefined Channel Modes on Input Signal Processing*, and *Section , Channel Modes on Output Signal Generation*. Modes also allow combining input processing and output

---

af. TCCEA provides compatibility with TPU when service request is inhibited.

generation in a single channel, as exemplified in *Section , Combining Input and Output Signals*.

Section , Channel Mode Logic and Event Flags, shows a structural definition of channel logic and its relation to channel modes. A dynamic, event-oriented view of each channel mode can be found in *Section 22.7.3, Predefined channel mode summary*.

Channel Mode Logic and Event Flags

Figure 405 shows a more detailed diagram of channel mode logic. The logic shown here is not necessarily identical to the actual channel logic implementation, but is equivalent with respect to conditions for event blocking, enabling, capture, and service requests.

Signals MSR, TSR, MCAP, TCAP, M1ET, M1EM2, M1BM2, M2BM1, M2BT, T1BM1, T2BM1, TBM2 and T1ET2 are decoded from programmed channel mode PDCM in predefined modes, and come directly from the UDCM register when user-defined mode is selected:

- TSR (1 bit) defines Service Requests issued by Transitions, as shown in *Table 395*.
- MSR (2 bits) defines Service Requests issued by Matches, as shown in *Table 394*.
- TCAP (1 bit) defines time base captures caused by Transitions, as shown in *Table 395*.
- MCAP (1bit) defines time base captures caused by Matches, as shown in *Table 396*.
- M1ET, M1EM2, M1BM2, M2BM1, M2BT, T1BM1, T2BM1, TBM2, T1ET2 (1 bit each) define Match and Transition reciprocal blocking and enabling, as well as Transition ordering, as shown in *Table 397* and *Table 398*.

Table 399 shows the decoded values of those control signals for each predefined channel mode. The first column shows the mnemonic reference for the predefined channel modes described in the following sections.

Changing PDCM or the UDCM when user mode is selected may set or reset any of the channel flags, or issue captures and service requests, so it is advisable to switch channel modes only in a quiescent channel state, with channel flags MRLEA/B, TDLA/B, MRLA/B cleared. Furthermore, an event flag asserted in one mode may keep asserted after the mode programming change, even if the flag is impossible to be set in the new mode.

Table 394. MSR[1:0] signals – Match Service Requests

Value	MSR
00	issue no Service Requests on Matches
01	issue Service Request on Match B only
10	issue Service Request on 2nd[1] Match
11	issue Service Request on both Matches

1. 2nd Match means the Match that happens after the 1st Match in time, either Match A or Match B.

Table 395. TCAP and TSR signals – Transition Captures and Service Requests

Value	TCAP	TSR
0	1st[1] Transition captures Time Bases corresponding to Transition A and Transition B[2] Transition A captures corresponding Time Base if it is the second transition; Transition B captures corresponding Time Base if it is the second transition.	issue Service Request on the 1st[1] Transition
1	Transition A captures corresponding Time Base. Transition B captures corresponding Time Base.	issue Service Request on the 2nd[3] Transition

1. 1st Transition means the Transition that happens first in time, either Transition A or Transition B.
2. Match capture(s) never overwrites a Transition capture. Transition captures can always override a Match capture.
3. 2nd Transition means the Transition that happens second in time, either Transition A or Transition B.

Table 396. MCAP signal – Match Capture

Value	MCAP
0	Match A captures corresponding Time Base; Match B captures corresponding Time Base
1	either Match captures Time Bases corresponding to Match A and Match B[1]

1. Match capture(s) never overrides a Transition capture. Transition captures can always override a Match capture.

Table 397. TBM2 signal – Transition Blocks Match B

Value	TBM2
0	Transition A Blocks Match B
1	Transition B Blocks Match B

Table 398. M1ET, M1EM2, M1BM2, M2BM1, M2BT signals

Signal	Active value meaning
M1ET (Match A Enable Transitions)	Transitions are initially blocked, and Match A enables Transitions
M1EM2 (Match A Enables Match B)	Match B is initially blocked[1], and Match A enables Match B
M1BM2[2] (Match A Blocks Match B)	Match B is initially enabled[1], and Match A blocks Match B[3]
M2BM1[2] (Match B Blocks Match A)	Match A is initially enabled, and Match B blocks Match A[3]
M2BT (Match B Blocks Transitions)	Match B blocks Transitions
T1BM1 (Transition A Blocks Match A)	Transition A blocks Match A

Table 398. M1ET, M1EM2, M1BM2, M2BM1, M2BT signals

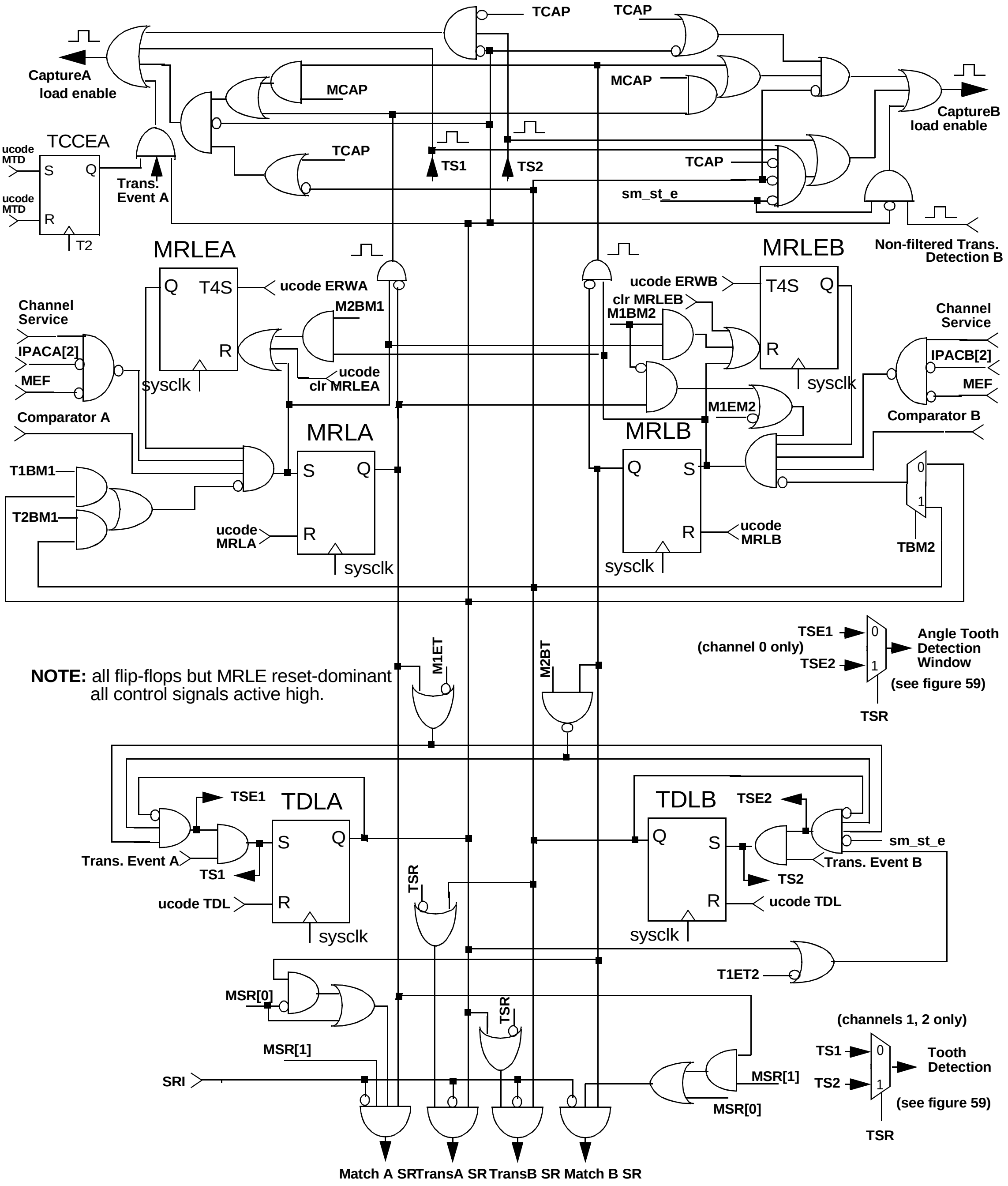
Signal	Active value meaning
T2BM1 (Transition B Blocks Match A)	Transition B blocks Match A
T1ET2 (Transition A Enables Transition B)	Transition B is initially blocked, and Transition A enables Transition B

1. The initial condition of M1EM2 prevails over M1BM2, while M1BM2 blocking prevails over M1EM2 enabling, so that Match B stays always blocked when both M1BM2 and M1EM2 are active. This combination is used in single-match modes (sm_*).
2. Blocking of one Match by the other is done through MRLEs.
3. Matches always block themselves by resetting their own MRLEs (Match A always blocks Match A, Match B always blocks Match B)

Table 399. Predefined channel mode control signals decoding

Predefined mode	MSR	MCAP	M1ET	M1EM2	M1BM2	M2BM1	M2BT	T1BM1	T2BM1	TBM2 (1)	T1ET2	TSR (1)	TCAP(1)
em_nb_st	11	0	off	off	off	off	off	On	off	0	On	0	0
em_nb_dt	11	0	off	off	off	off	off	On	off	1	On	1	1
em_b_st	11	1	off	off	On	On	off	On	off	0	On	0	0
em_b_dt	11	1	off	off	On	On	off	On	off	1	On	1	1
bm_st	10	0	off	off	off	off	off	On	off	0	On	0	0
bm_dt(2)	10	0	off	off	off	off	off	off	On	1	On	1	1
m2_st	01	0	On	off	off	On	off	On	off	0	On	0	0
m2_dt	01	0	On	off	off	On	off	On	off	1	On	1	1
m2_o_st	01	0	On	On	off	off	On	On	off	0	On	0	0
m2_o_dt	01	0	On	On	off	off	On	On	off	1	On	1	1
sm_st	11	1	off	On	On	On	off	On	off	0	On	0	0
sm_dt(2)	11	1	off	On	On	On	off	off	On	1	On	1	1
sm_st_e(3)	11	0	off	On	On	On	off	On	off	0	On	0	0

1. Signals TSR, TCAP and TBM2 replace the signal DTM used in previous eTPU versions.
2. bm_dt and sm_dt are exceptions in the match blocking logic by transitions. See *Section , Both Match Request Modes (bm_st, bm_dt)*, and *Section , Single match modes (sm_st, sm_dt)*.
3. sm_st_e is an exception in the capture scheme. See *Section , Single match enhanced mode (sm_st_e)*.



NOTE: all flip-flops but MRLE reset-dominant
all control signals active high.

Figure 405. Channel Mode Logic and Event Flags

Channel modes overview

Predefined channel modes are divided according to the way they treat transitions in two basic modes:

- **Single Transition Modes** (mnemonic suffix _**st**): In these modes the first transition (flagged in TDLA) issues a service request, and captures both time bases (selected by TBSA[1] and TBSB[1]) except on sm_st_e. The second transition (flagged in TDLB) does not issue a service request, but it captures time base selected by TBSB[1], except on sm_st_e.
- **Double Transition Modes** (mnemonic suffix _**dt**): In these modes the second transition (flagged in TDLB) issues a service request, and each transition captures its own selected time base (Transition A and Transition B capture time bases selected by TBSA[1] and TBSB[1], respectively).

In predefined modes, Transition B is always (but not only) enabled by Transition A, so that transitions are always ordered: TDLA is set on the first transition and TDLB on the second. Unordered transitions are possible with user-defined mode, when UDCM bit T1ET2 = 0. Matches are generally not ordered, except on specific ordered match modes m2_o_st and m2_o_dt. Match capture(s) never overrides a Transition capture, while Transition captures can always override a Match capture, either in predefined or user-defined modes.

The following general rules apply to both predefined and user-defined modes:

- Blocking of one Match by the other, when it occurs, is done through MRLEs.
- Matches always block themselves by resetting their own MRLEs (Match A always blocks Match A, Match B always blocks Match B).

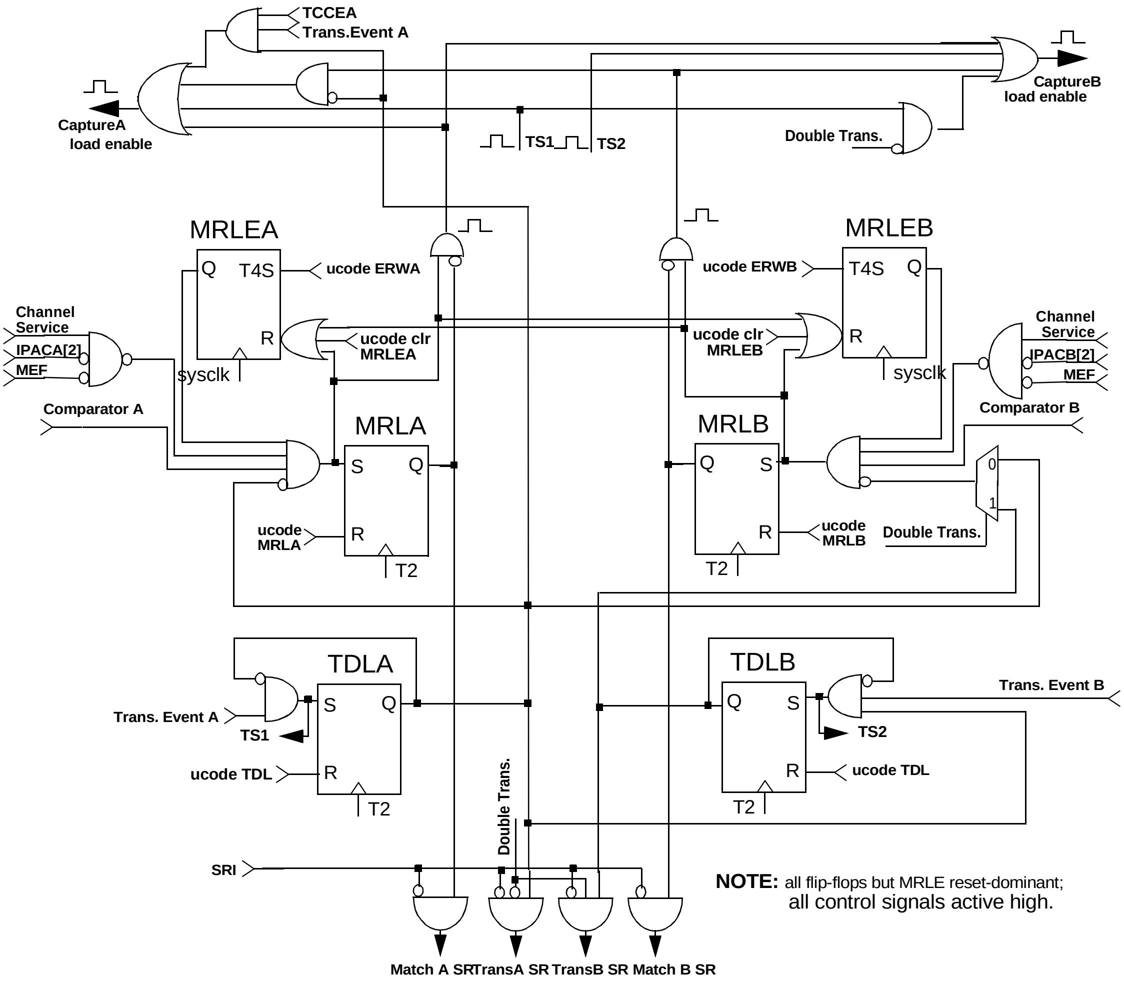
Predefined modes differ mostly by the way matches affects and are affected by other matches and transitions, as explained in next sections. However, some general rules on Match blocking apply:

- Match B is blocked by first transition (TDLA) in single transition modes, and by second transition (TDLB) in double transition modes.
- Both Matches are blocked by first transition in single transition modes.

Note: The rules above and in following sections may be overruled by the state of the channel latches if they are set/reset by microcode or if channel mode is changed. Care must be taken to change channel modes, and is advisable to reset channel flags MRLA/B, TDLA/B and MRLEA/B before writing PDCM, or to UDCM when user-defined mode is selected.

Either Match, Blocking Modes (em_b_st, em_b_dt)

In these modes the first match recognition that occurs blocks the other match recognition and generates a service request. They end up with one service request for two programmed match recognitions where only the first match recognition has an actual effect. If both match recognitions occur at the same time, both MRLA and MRLB are set, before the mutual blocking takes effect.

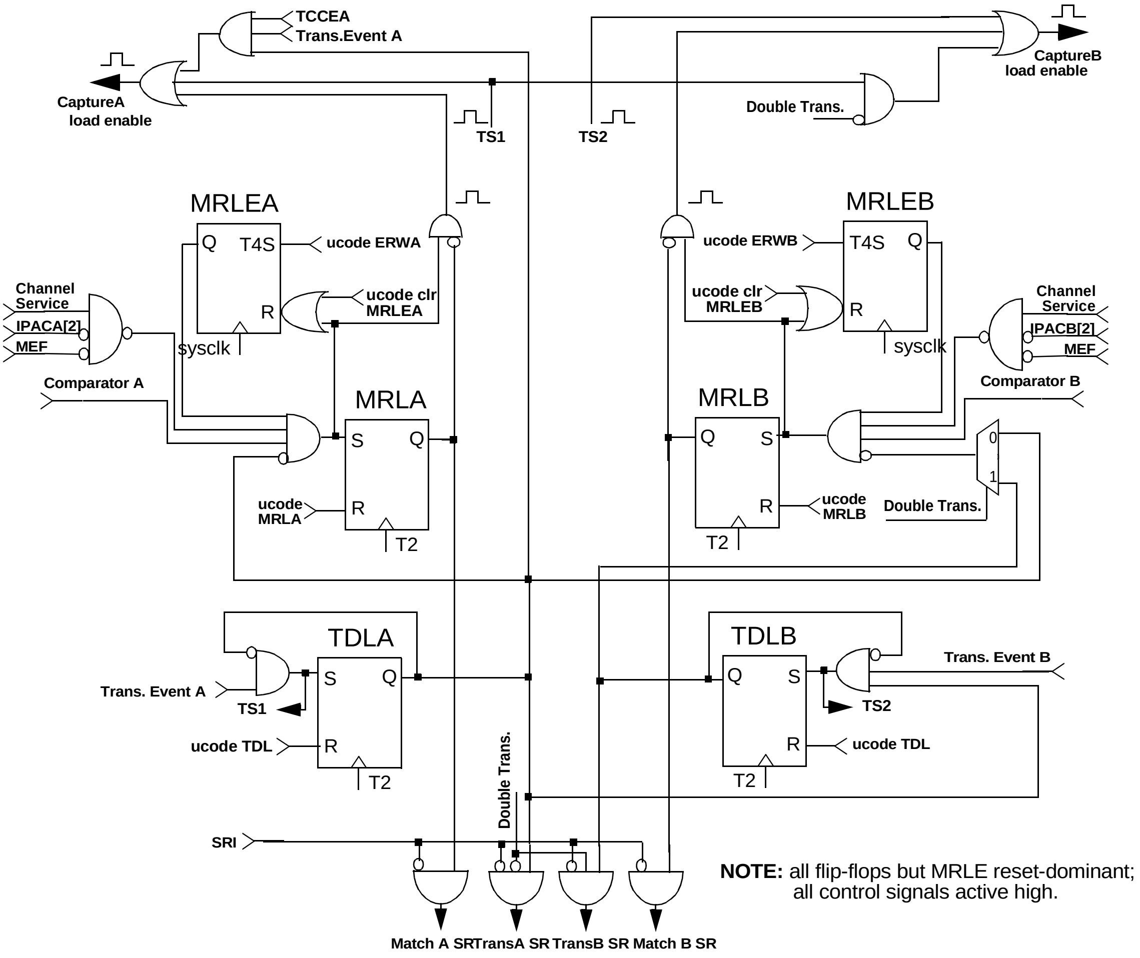


NOTE: all flip-flops but MRLE reset-dominant;
all control signals active high.

Figure 406. Either Match, Blocking Modes (em_b_st, em_b_dt)

Either Match, Non Blocking Modes (em_nb_st, em_nb_dt)

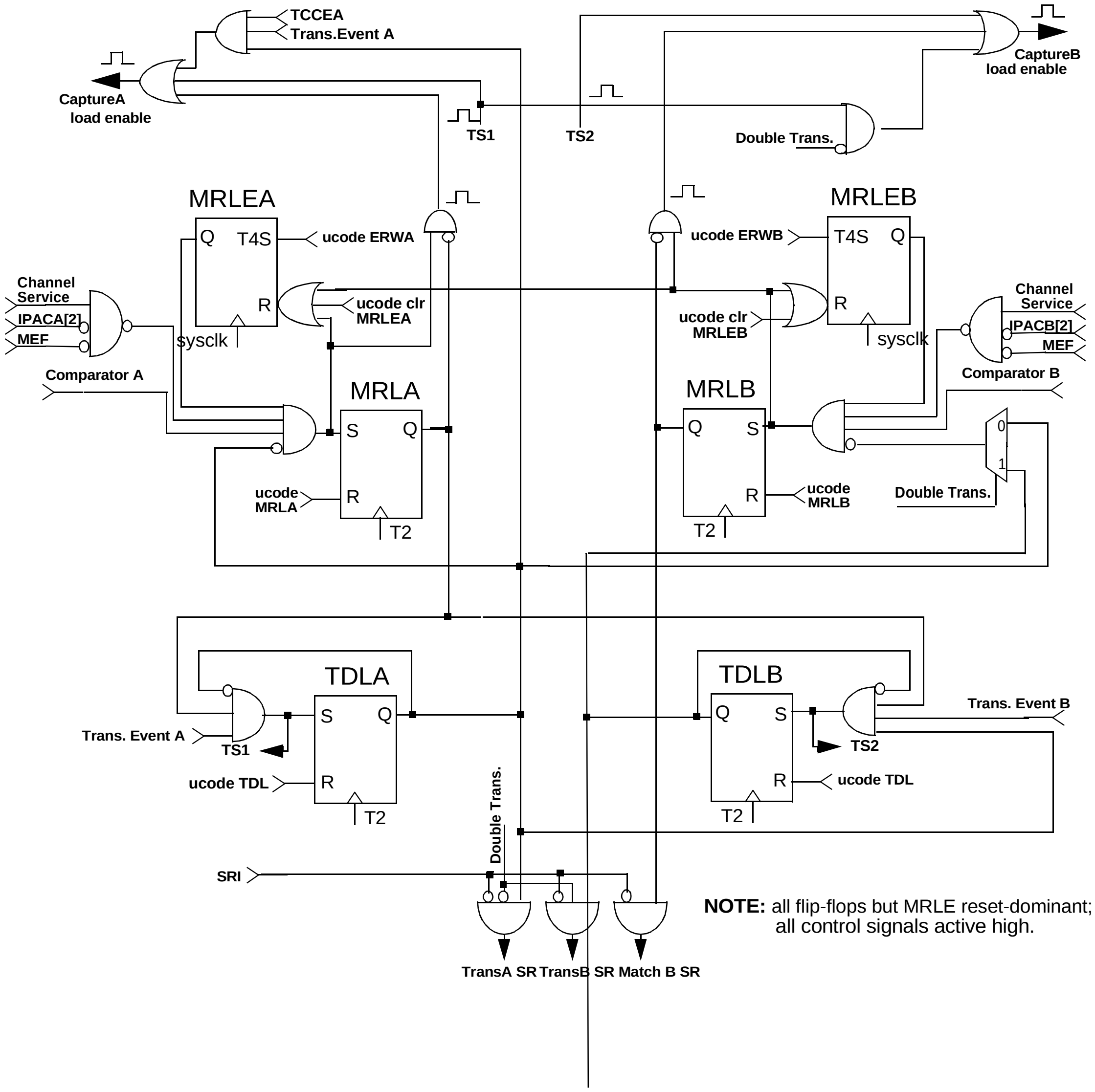
In these modes both match recognitions are independent and each of them generates service request. Each match recognition captures its related time base and does not block the other.



NOTE: all flip-flops but MRLE reset-dominant;
all control signals active high.

Figure 407. Either Match, Non Blocking Modes (em_nb_st, em_nb_dt)
Match B Request Modes (m2_st, m2_dt)

In these modes transitions are initially blocked, and are enabled by Match A. Match B recognition generates the match service request and disables Match A recognition. Each match recognition captures its own programmed timebase. In case of simultaneous match recognition, both MRLA and MRLB are set, and OPACB register has priority over OPACA for selecting the pin action.

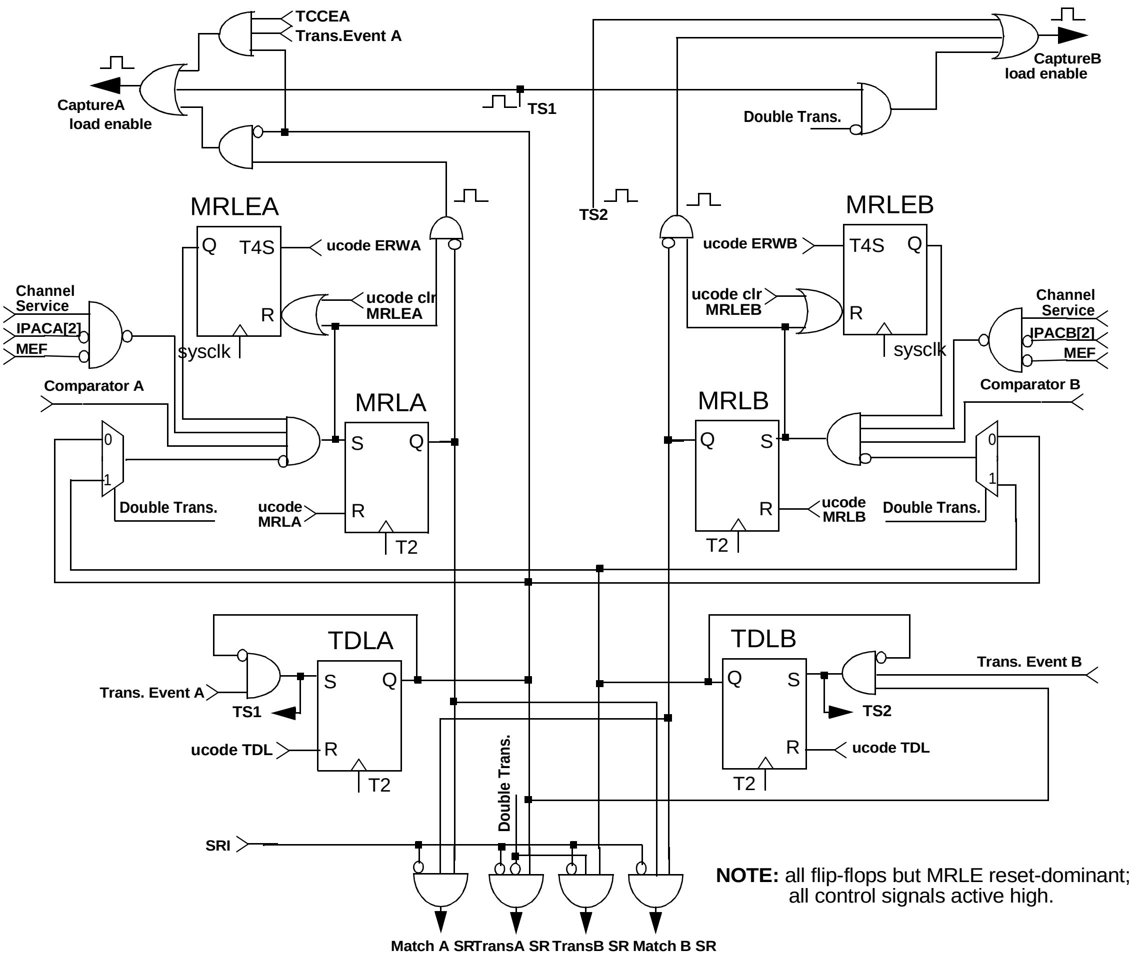


NOTE: all flip-flops but MRLE reset-dominant; all control signals active high.

Figure 408. Match B Request Modes (m2_st, m2_dt)

Both Match Request Modes (bm_st, bm_dt)

In these modes, match service request is generated only after both match recognitions occurred. By definition this is a non-blocking match mode: match recognitions do not block each other, implementing a last-served scheme. Unlike other double transition modes, bm_dt blocks Match A with Transition B (not with Transition A), so that the second transition blocks both matches.

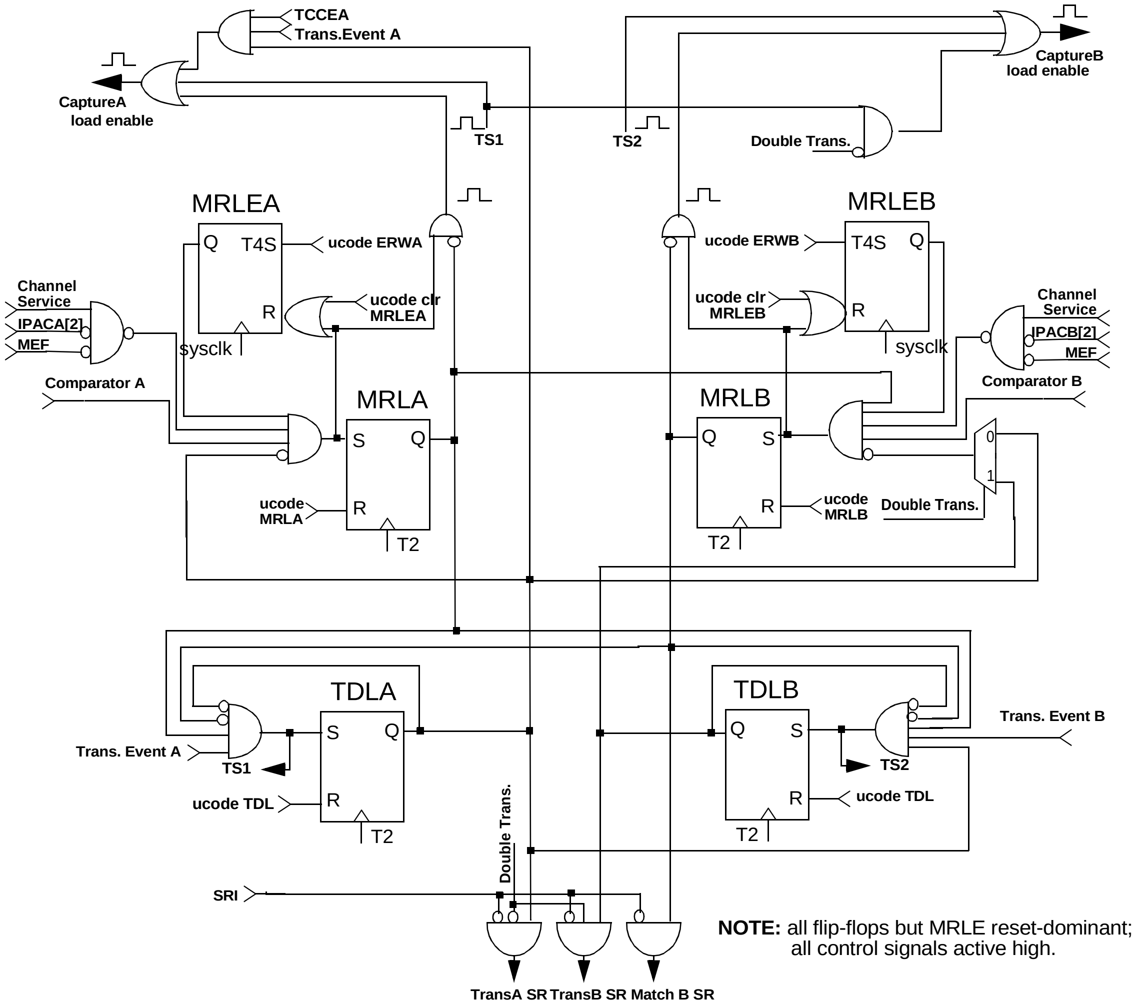


NOTE: all flip-flops but MRLE reset-dominant;
all control signals active high.

Figure 409. Both Match Request Modes (bm_st, bm_dt)

Ordered Modes with Match B Request (m2_o_st, m2_o_dt)

These are ordered match modes on which Match A recognition must precede Match B recognition (ordered 1->2). Match A asserts MRLA and enables Match B and transitions. Match B asserts MRLB, generates a match service request, and blocks both transitions.

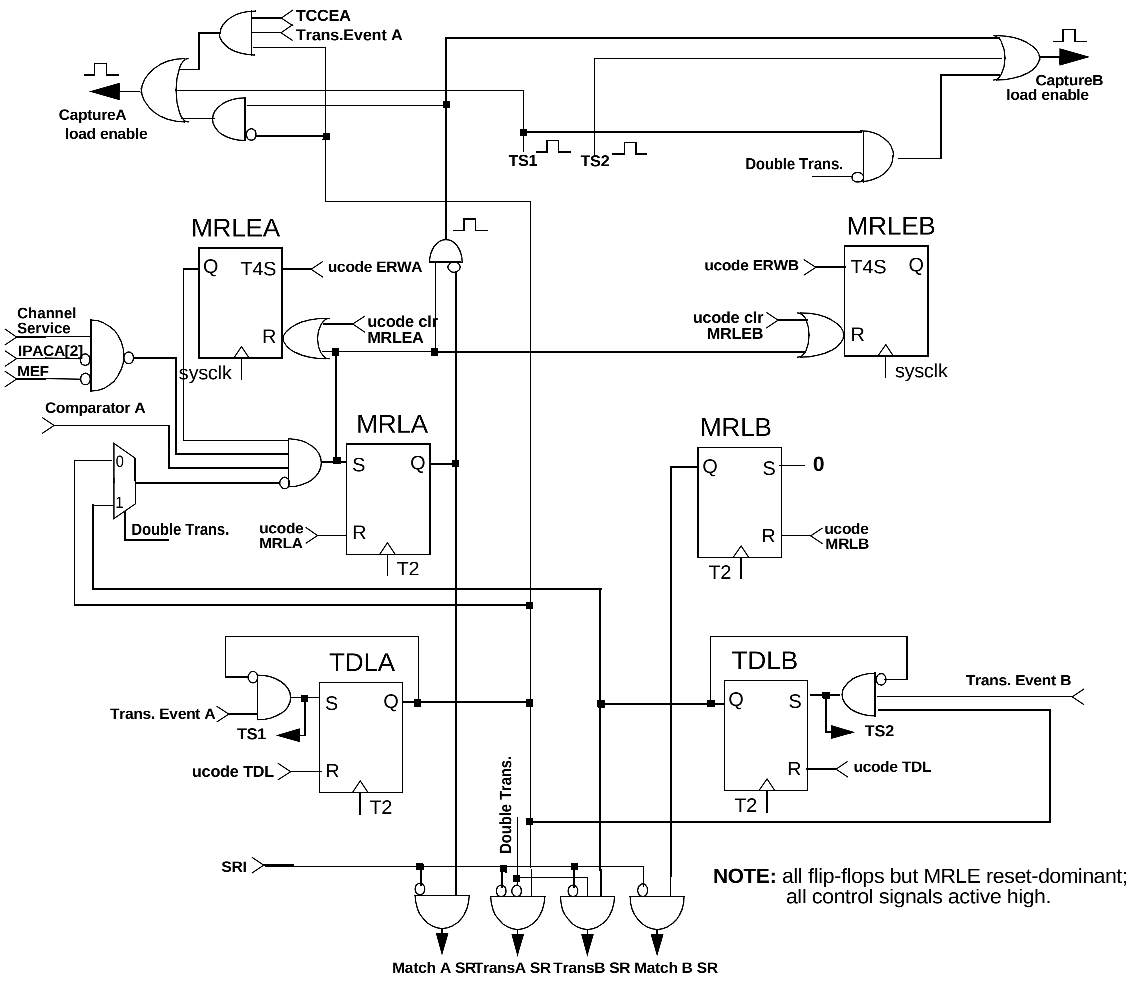


NOTE: all flip-flops but MRLE reset-dominant;
all control signals active high.

Figure 410. Ordered Modes with Match B Request (m2_o_st, m2_o_dt)

Single match modes (sm_st, sm_dt)

Single match modes support single or double transition with single match recognition. MRLB is never set, and MRLEB has no effect.

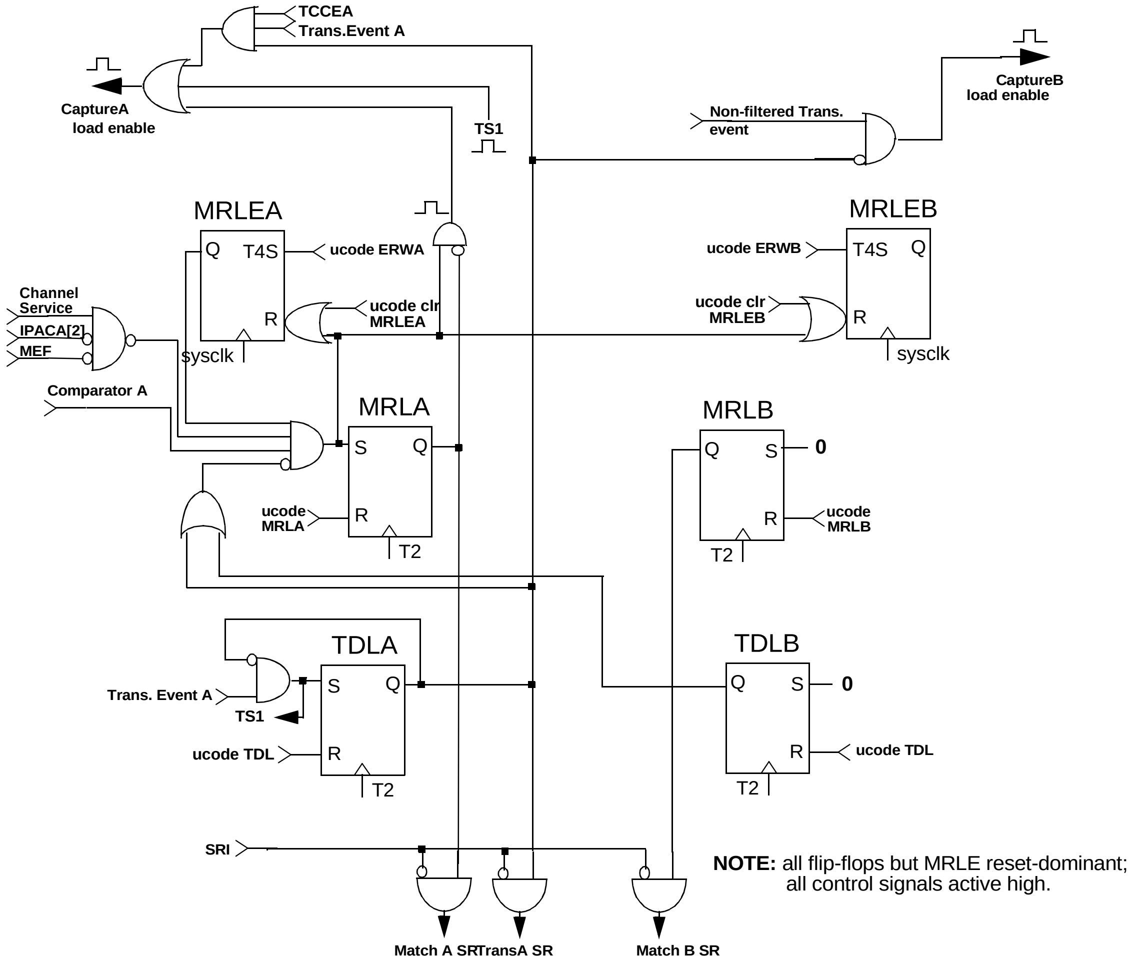


NOTE: all flip-flops but MRLE reset-dominant; all control signals active high.

Figure 411. Single match modes (sm_st, sm_dt)

Single match enhanced mode (sm_st_e)

This is an enhanced single transition and single capture mode, which provides non-filtered input captures in addition to the single capture, allowing one to measure the delay of the digital filter. In an output channel, it has the same functionality of sm_st (captures both time bases at once due to a match recognition).



NOTE: all flip-flops but MRLE reset-dominant;
all control signals active high.

Figure 412. Single match enhanced mode (sm_st_e)

Predefined Channel Modes on Input Signal Processing

When processing an input signal, the predefined channel modes can be classified in the following primary mode groups:

- Single Transition, Single Match: em_b_st, sm_st, sm_st_e
- Single Transition, Double Match: em_nb_st, bm_st, m2_st, m2_o_st
- Double Transition, Single Match: em_b_dt, sm_dt
- Double Transition, Double Match: em_nb_dt, bm_dt, m2_dt, m2_o_dt

In single transition modes, TDLA assertion may capture both time bases at once, while in double transition modes each transition captures its related time base in its related capture register.

Double transition is always ordered, i.e TDLB is enabled by TDLA and generates the service request.

The channel logic supports various input modes with combinations of single/double transition and single/double match, explained in the following subsections.

Either Match, Blocking, Single Transition (em_b_st)

On an input signal, this mode provides double timeout mechanism on a programmed transition edge with two timebases. The signal transition blocks both pending matches, indicating that no timeout condition occurred. The two match recognitions block each other, giving good separation in the entry table as to which match recognition caused the first timeout condition, and generating only one service request. Either match performs timebase captures which do not overwrite captures by transitions.

Either Match, Blocking, Double Transition (em_b_dt)

In double transition mode each transition is related to one match recognition. TDLA assertion captures its related timebase, blocks Match A and enables TDLB. TDLB assertion blocks Match B, captures its related timebase and generates a service request. Match recognitions block each other, so if there is a match timeout condition on TDLA, only one match service request is generated. This mode is good for qualifying two signal transitions by match timeout mechanisms, with one service request. Note that although TDLA assertion does not block Match B recognition, the value captured in CaptureA by TDLA assertion is not overwritten by this recognition. The second transition blocks Match B. Either match performs timebase captures which do not overwrite captures by transitions.

Either Match, Non Blocking, Single Transition (em_nb_st)

On an input signal, this is a double timeout mechanism of independent match recognitions of two different timebases. The match recognitions do not block each other, such that the microcode can check if one or two match recognitions occurred before their related signal transition. The signal transition detection (by IPACA) asserts TDLA, blocks both matches, captures both time bases and generates a transition service request, indicating that none of the two timeout conditions occurred. Any combination can be easily resolved by microcode (for example, signal transition after Match A and before Match B, or signal transition after both Match A and Match B).

Another possible use of this mode is allocating one match recognition for transition timeout and the other for another non-critical timed task, adding functionality to a single channel. Since the transition detection blocks both match recognitions, the match recognition of the other timed task is based on the fact that the comparator checks greater-equal conditions. It may be delayed if it occurs in the period between the signal transition detection (which blocks it) and the time TDLA is negated by microcode. If matches are enabled during the service, the same code can check if the match recognition of the timed task occurred in this period, by negating TDLA and writing to the CHAN register its own value (in order to update the MRLA flag in the branch logic).

Either Match, Non Blocking, Double Transition (em_nb_dt)

In this mode each transition is related to one match recognition, and the match recognitions are independent of each other. This mode can be used to give independent timeout conditions for the first and the second signal transition recognitions, and call service in any case of any timeout condition.

The first transition detection programmed in IPACA sets TDLA, captures its related timebase, blocks Match A recognition and enables TDLB assertion. The second transition detection programmed in IPACB sets TDLB, blocks Match B recognition, captures its related timebase and generates a service request. Any match recognition that occurs captures its related time base and generates a match service request, independent on the other match recognition.

Match B Request, Single Transition (m2_st)

On an input signal, this mode provides an open window filter for a single signal transition. MRLA assertion opens the window, and enables transition detection on TDLA from this time on. MRLB assertion blocks Match A (by negating MRLEA), providing conditional window opening, because transitions are indirectly blocked. It also generates service request, but if it happens after Match A it does not block transitions, providing a non-blocking timeout mechanism for the estimated signal transition time (typically it indicates a missing transition, or mis-prediction of the transition time).

Transitions can be detected from the microcycle following MRLA assertion. The Transition A detection asserts TDLA, blocks both matches, captures both timebases and generates service request.

Using this mode, the channel can replace software open window filtering of qualified transitions with the channel hardware window. The window opening and timeout can be scheduled for any of the two time bases or combination of them. Typically, Match A will be used to open a prediction window, and Match B will be used as a timeout condition which does not close the prediction window. This configuration improves noise immunity from early signal transitions, and reduces the probability for blocking late signal transitions due to timeout mis-prediction.

Using these conditions, the microcode can easily resolve the state:

- If TDLA and MRLA are asserted and MRLB negated, signal transition is in the expected range.
- If MRLA and MRLB are both asserted, and TDLA is asserted, the signal transition had a timeout condition due to Match B mis-prediction.
- If MRLB is asserted and TDLA negated, a timeout condition occurred, and the expected signal transition had not occurred yet.
- If MRLA is negated and MRLB is asserted, the conditional window did not open at all (for example: a time window is open only after a specific angle, otherwise it is not opened).

Match B Request, Double Transition (m2_dt)

This mode is used as an open window filter for two signal transitions. In this case the Match A recognition opens the window (unless Match B recognition occurred first), and Match B recognition blocks Match A and generates a match service request. It is similar to m2_st, but in this case, it is the second transition that blocks Match B. MRLB assertion is a global timeout condition for the two pulses. Like m2_st, MRLB can conditionally eliminate the window from opening.

Using the TDLA, TDLB, MRLA and MRLB conditions, the microcode can easily resolve the state, in a similar manner as m2_st, with additional information on the second transition (TDLB).

Both Match Request, Single Transition (bm_st)

On an input signal, this is a double timeout mechanism on two different time bases. Both match recognitions must occur before the signal transition to generate a match timeout service request. Assertion of TDLA blocks both Match A and Match B recognitions, and captures both time bases, indicating there was no double timeout condition from both time bases.

Using the same timebase implements two timeout conditions, the first only sets its related MRL and the second generates a service request. Using these flags allows the microcode to check if one or both match recognitions precede the signal transition.

Both Match Request, Double Transition (bm_dt)

In this mode the first transition detection does not block matches, since both match recognitions are required to generate a match service request. The second transition detection asserts TDLB, blocks Match A and Match B, captures its related timebase and generates transition service request. In this mode, a Match A recognition which occurs after the assertion of TDLA does not capture a new value in CaptureA, to preserve the actual signal transition time. Assertion of TDLA, however, always captures its related timebase.

This mode allows putting a double match timeout condition on the second transition. Typically, a pulse trailing edge timing can be checked against two time bases, to indicate if the pulse has not ended when both MRLA and MRLB are asserted. When a transition service request is generated by TDLB assertion, the state of MRLA and MRLB indicates which timeout condition occurred, if any.

Ordered Mode with Match B Request, Single Transition (m2_o_st)

On an input channel, this mode provides a closing window filter for a single signal transition. Match A assertion captures its programmed time base in CaptureA, opens the filter window (enables assertion of TDLA), and enables assertion of MRLB. Match B recognition captures its related timebase, closes the window (disables assertion of TDLA) and generates a service request. Due to Match A and Match B ordering, the window is opened for at least one microcycle. Match B recognition indicates a window timeout condition which blocks late signal transitions, outside the prediction window. Transition detection blocks both matches, indicating the transition occurred inside the estimated window. Transitions can be detected from the microcycle following MRLA assertion until the microcycle on which MRLB is asserted. When TDLA is asserted inside the window range it disables both matches, captures both time bases and generates a transition service request.

Using this mode, the channel can replace software window filtering of qualified transitions with the channel hardware window. The window opening and closing time can be scheduled for any of the two time bases or a combination of them.

Ordered Mode with Match B Request, Double Transition (m2_o_dt)

In this mode the channel logic implements a window filter for two detected signal transitions. MRLA assertion captures its related timebase and enables the assertion of both TDLA and TDLB. MRLB assertion captures its related timebase and disables the assertion of both TDLA and TDLB. Transitions can be detected from the microcycle following MRLA assertion until the microcycle on which MRLB is asserted. The first signal transition (following MRLA assertion) asserts TDLA, captures its related timebase and enables assertion of TDLB. The second signal transition detection asserts TDLB, blocks Match B, captures its related timebase and generates the service request.

If both signal transitions occur inside the scheduled window, Match B recognition is blocked. If one or both signal transitions do not occur inside the scheduled window, Match B recognition generates a match service request and blocks further transition detections. The microcode can resolve the state using MRLA, MRLB, TDLA and TDLB, which affect the entry point selection.

Single Match Enhanced Mode (sm_st_e)

This is an enhanced single transition and single match channel mode which provides timing information of the digital filter delay.

The CaptureA register captures the timebase selected by TBSA due to transition detection specified by IPACA or match recognition, as in sm_st mode. Initially, the CaptureB register continuously captures the unfiltered IPACB-selected signal transitions from the digital filter input, directly from the signal synchronizer. When an IPACA-qualified, filtered transition detection occurs, TDLA is set, MRLA assertion is blocked, and, in addition, captures into CaptureB are also blocked. On service, CaptureA and CaptureB (copied into ERTA and ERTB) holds the time of the qualified transition detection (ERTA), and the time of the last signal transition at the input of the digital filter (ERTB). Subtracting the time in ERTB from the time in ERTA provides the delay of the digital filter.

In a quiet environment, the two captures provide the accurate delay of the digital filter in granularity of two system clocks. In a noisy environment, false transitions may be detected at the input of the digital filter due to the noise, and the delay measurement may be reduced, especially if IPACB selects both edge detection. The microcode can do sanity checks on this value to recognize noise effects (for example: calculated delay is less than the minimum delay of the digital filter).

Note: In Channel 0, if ETPU_TBCR field AM = 01 (Angle Mode), the unfiltered input comes from TCRCLK input and the filtered input comes from the TCRCLK filter output. The edge is selected by IPACA/B, and is independent of the edge selection by ETPU_TBCR field TCR2CTL.

Single Match, Single Transition (sm_st)

In this mode the channel logic is functionally back-compatible to a TPU3 single action channel, but a match or transition detection captures at once both timebases. The mode recognizes a single transition with single match timeout. Either TDLA or MRLA generates service request and captures both timebases. Assertion of TDLA blocks future assertions of MRLA.

Single Match, Double Transition (sm_dt)

In this mode, the first transition detection asserts TDLA, captures a timebase in CaptureA and enables TDLB. The second signal transition asserts TDLB, blocks Match A, captures a timebase in CaptureB and generates a service request.

Match A (before TDLB) captures into CaptureB the timebase selected by TBSA, in order not to overwrite the captured value of TDLA.

This mode is used for scheduling one timeout condition on two input signal transitions (pulse timeout).

Channel Modes on Output Signal Generation

Since channel logic can generate output signal transitions based on Matches, the channel can be viewed as working in the following primary mode groups for signal generation:

- Single Match: em_b_st, sm_st, sm_st_e, em_b_dt, sm_dt
- Double Match: em_nb_st, bm_st, m2_st, m2_o_st, em_nb_dt, bm_dt, m2_dt, m2_o_dt

The channel logic supports various match channel modes with single/double match, as explained in the following subsections.

Either Match, Blocking Modes (em_b_st, em_b_dt)

On an output signal these modes are useful when using two different time bases to set a required signal transition. The first match condition which is met sets a required pin action, captures both time bases, blocks any effects of the other recognition, and generates a service request. Because the first match recognition blocks the other, the microcode can get good separation in the function entry table as to which match caused the timeout first, and both time bases are captured, enabling the microcode to compare one timebase to the other at the moment of the match recognition. These modes can be used for:

- Scheduling a required pin action to the first match recognition of two different time bases.
- Cancelling a programmed pin action scheduled on one time base by match on another timebase (as a consequence of *Table 400*). Microcode has to set the OPAC register of the cancelling match to no-action and the OPAC register of the other match to the required pin action which may be blocked. If Match A is the cancelling match, it blocks the pin action also in case of two matches at the same time, since it has priority in this case. If Match B is the cancelling match, it does not block the pin action in case of two matches at the same time.

Either Match, Non Blocking Modes (em_nb_st, em_nb_dt)

On an output signal these modes are useful in combination with the ME bit set on the entry point, to define an interlaced operation. For example, each match recognition can set a pin action, and the second pin action is not sensitive to microcode latency (ME bit asserted). Example for usage is PWM interlaced function on which the latency is determined by the period and not the duty cycle.

Another possibility is using one match for pin actions and the other match for an unrelated timed task without pin action (double the functionality of a single channel).

Match B Request Modes (m2_st, m2_dt)

On an output signal, these modes can generate narrow pulses or do conditional pin actions. A conditional pin action means that the pin state is changed only if the match recognitions occurred in the correct order, since the Match B recognition which generates the service request also has priority over the pin action and blocks future Match A recognitions.

Setting OPACA to a desired pin action and OPACB to no-action, and using different time bases for Match A and Match B defines a conditional OPACA pin action which can be blocked by Match B recognition. For example, setting Match A on time and Match B on angle can limit the pin action to a maximum angle value.

When pulses are generated, the service is requested at the trailing edge of the pulse, after MRLB is asserted.

Both Match Request Modes (bm_st, bm_dt)

On an output signal, each match recognition can affect the pin state, and capture its programmed time base. This way the pin action can be programmed separately for both match recognitions. For example, both match recognitions can negate the signal, and service request is generated after both conditions are met. This mechanism can set two conditions to do a required pin action, and the first recognition changes the signal, but service is called only after both conditions occur.

When using the same time base, these modes can generate narrow pulses in any required order. For example, in a PWM function, when duty cycle is below 50% the function can get service on the low time and program the pulse to the required duty cycle of the high time. When duty cycle is equal or above 50%, the function can get service on the high time and program a negative pulse with the width of the required low time. To switch between the two states the function can program once the same transition time to MatchA and Match B with a required pin action, and on the next service program double match for the new state.

Another usage is generating a required pin action on one programmed time and service request later on another time, after the second match recognition occurs, or capturing some timebase on one time and generating a required signal transition and service request later.

Ordered Modes with Match B Request (m2_o_st, m2_o_dt)

The order of the match recognitions imply that OPACA register programmed pin action always precede the OPACB register pin action. Setting OPACA to no-action, based on the greater-equal comparator, enables using Match A on one time base to delay the signal effect of Match B on the other time base. This method implements a conditional pulse extension or conditional delay on signal transition.

These modes can also be used for deferred pulse generation with microcode service request after its trailing edge (if Match A condition comes after Match B condition). Another option is having Match A recognition associated with output pin actions and Match B recognition for a timed microcode task which has to be scheduled at a programmed time which may be delayed by the Match A pin action.

Single Match Modes (sm_st, sm_dt, sm_st_e)

There is no difference between plain and enhanced single match modes on an output signal.

In this mode the channel logic is functionally back-compatible to a TPU3 single match output channel. Match A recognition generates service request and sets the pin state according to OPACA register. It captures at once the timebase selected by TBSA in CaptureA and the timebase selected by TBSB in CaptureB.

Match/Transition Pin Action Conflict Resolution

In output signals, matches and/or transitions automatically cause pin actions defined by the OPACA/B and/or IPACA/B channel control registers (see *Section , Pin Control Registers*). Simultaneous matches/transitions may be associated with different, possibly contradictory, pin actions. These conflicts are resolved according to the *Table 400*.

If an OPACA/B = 000 (no action) prevails over non-zero OPAC according to *Table 400*, then if Match A/Transition A and Match B/Transition B occur simultaneously, no output pin action occurs, that is: a match on the action logic with OPAC = 000 inhibits simultaneous actions of the other OPAC, if prevailing according to *Table 400*. That also applies when output actions are caused by inputs (OPAC = 1xx).

Table 400. Simultaneous match pin action priority

Channel mode	Priority
em_nb_st / em_nb_dt	OPACA
em_b_st / em_b_dt	OPACA
bm_st / bm_dt	OPACA
m2_st / m2_dt	OPACB
m2_o_st / m2_o_dt	In these modes there is no possibility of simultaneous matches
user-defined	OPACB if M2BM1 = 1 and M1BM2 = 0, OPACA otherwise

Combining Input and Output Signals

The processing of input signal can be combined with output signal generation. A detected input transition can trigger an output signal edge, even without microcode intervention, by using OPAC options 1xx.

The channel set-up examples below show these two capabilities combined (see *Figure 413*).

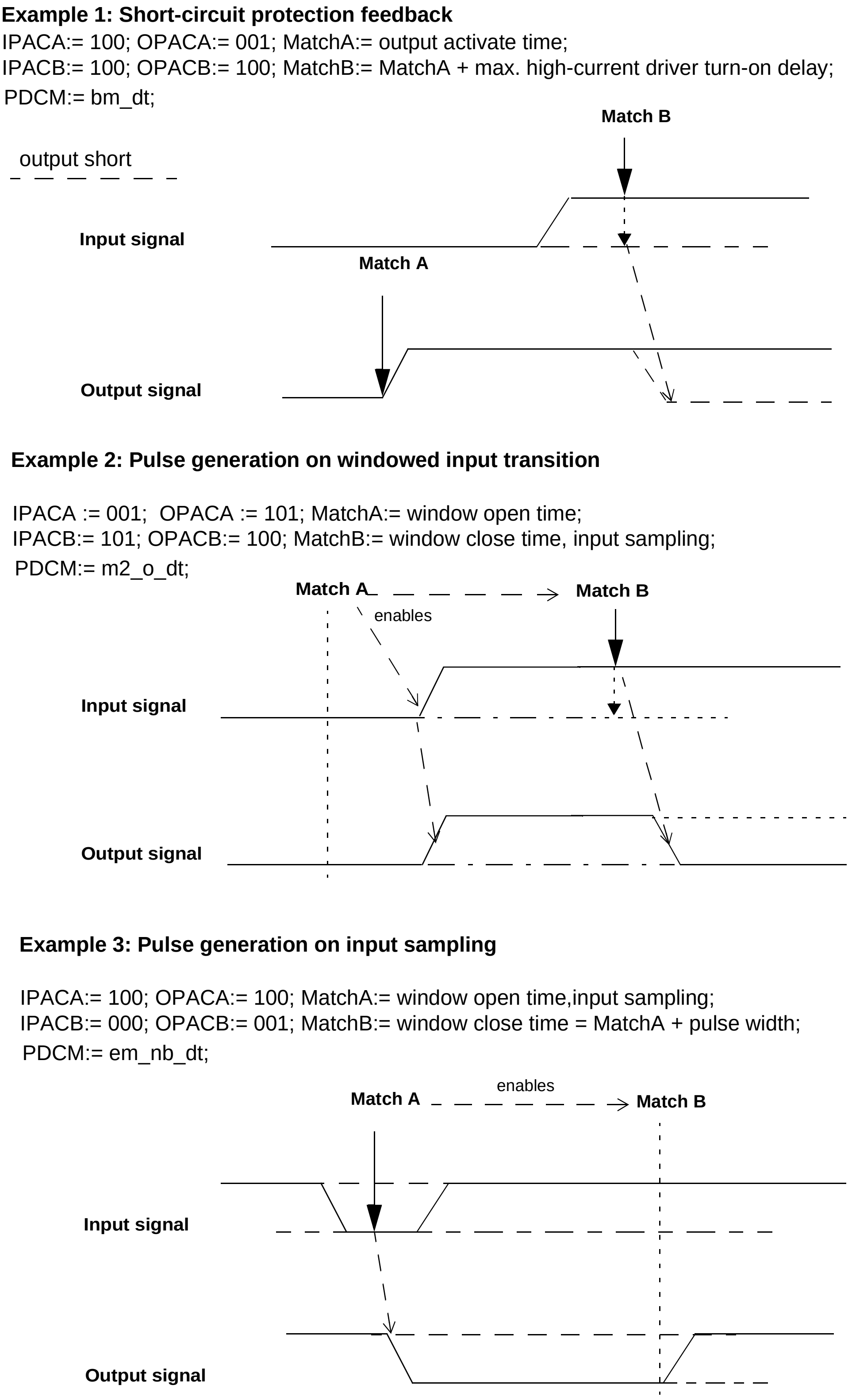
The **first example** implements a fast (no microcode intervention) short-circuit protection feedback mechanism for driving high-current output devices. The signal after the high-current driver feeds back to the channel input. The input signal is normally delayed from the output signal by the device turn-on delay. After the channel output turns on, the channel logic must check if the driver output (connected to the channel input) follows the driven value after the maximum device turn-on delay. If it does not, the driver output is probably shorted, and the channel output must be turned off immediately to avoid damaging the device.

Match A causes the output to be driven high (for simplicity the output and input signals are shown as positive logic). It also causes a transition A, because IPACA = 100 and the input is still low. Match B occurs after the expected driver delay, and causes a service request. If the output is shorted, a Transition B occurs on Match B because IPACB = 100. This will cause the output to go low immediately, because OPACB = 100.

In the **second example**, an output pulse is generated from an input transition without microcode intervention. Match A opens a window for transitions and also enables Match B. A rising edge on input sets output high. On Match B the window closes, and input signal is checked: if sampled high, the output resets; otherwise it stays high.

In the **third example**, a pulse is generated depending on the value sampled on the input signal at a predetermined time. Match A samples the input signal, causing a Transition. Low level on input sets output low, otherwise it stays high. Match B sets output high. In both cases a service request is issued (microcode intervention), at the beginning and at the end of the pulse (Match B), if required (SRI := 0).

Note: *When IPAC = 1xx, a match event can cause simultaneously a Match recognition and a Transition detection. Depending on the Channel Mode, these Match and Transition may have conflicting effects on other Transition/Match blocking or enabling. In these cases, blocking always prevails over enabling, effective on the next microcycle.*



Example 1: Short-circuit protection feedback

IPACA:= 100; OPACA:= 001; MatchA:= output activate time;
IPACB:= 100; OPACB:= 100; MatchB:= MatchA + max. high-current driver turn-on delay;
PDCM:= bm_dt;

Example 2: Pulse generation on windowed input transition

IPACA := 001; OPACA := 101; MatchA:= window open time;
IPACB:= 101; OPACB:= 100; MatchB:= window close time, input sampling;
PDCM:= m2_o_dt;

Example 3: Pulse generation on input sampling

IPACA:= 100; OPACA:= 100; MatchA:= window open time,input sampling;
IPACB:= 000; OPACB:= 001; MatchB:= window close time = MatchA + pulse width;
PDCM:= em_nb_dt;

Figure 413. Input/Output combination

Channel Link

A channel can issue service requests to other channels through microcode, by assigning to the write-only microengine register LINK (refer to *Section , LINK Register*) a value which specifies the target channel of the Link Service Request, as shown in *Figure 414*.

Writing to the LINK register issues a link request to the target channel, setting its LSR flag. Each channel has its own LSR flag, which can be tested as a microcode branch condition (see *Section , Conditional/Unconditional branch*) and reset through the microcode field LSR (see *Section , Clear link service request*). The link branch condition samples, at the TST start, the value used to calculate the Entry Point.

Writing LINK with another channel target value in the same thread issues a Link Service Request to the new target, without negating the service request to the former one. This allows a channel to issue service requests to any number of channels, including itself. Neither LINK nor LSR microengine accesses are qualified by the CHAN register, i.e., they always access the serviced channel LINK and LSR, regardless of the value written in CHAN.

If microcode executes an instruction with field LSR = 0 (clear Link Service Request), the link branch condition is cleared. However, the link service request itself is cleared only if no link was received by the serviced channel during the same thread(ag). If microengine clears LSR of its channel and, simultaneously, Link Service Request is issued to the current serviced channel, the branch condition is cleared but the link service request remains pending.

7	6	5	4	3	2	1	0
Engine Selection		reserved(1)	Channel Number				

1. Reserved bit must be written 0.

Figure 414. Microengine LINK Register

A channel can issue Link Service Requests to channels in any of two engines, determined by the LINK register field Engine Selection (2 bits), as shown in *Table 401*. In a single-engine eTPU, Link is ignored when sent to the other engine, or engine 2.

Table 401. LINK engine selection

Engine selection	Description
00	this Engine
01	Engine 1
10(1)	Engine 2
11(1)	the other Engine

1. Ignored in single-engine eTPU

The engine which receives the link cannot distinguish where the link comes from, except by some user-programmed protocol using SPRAM parameters. All links are negated on reset.

---

ag. That can only happen if the link service request came from the other engine or from the serviced channel itself.

Enhanced Digital Filter – EDF

The EDF eliminates passing of signal transitions which are caused by noise. Its purpose is to eliminate false transition service requests caused by noise pulses which are shorter than a programmed width.

The EDF has three modes of operations, selected by the CDFC field in the ETPU_ECR (see *Section , ETPU_ECR – eTPU Engine Configuration Register*). These modes offer selections of trade-off between noise immunity and signal latency. CDFC also allows the filter to be bypassed. *Table 402* gives an example of minimum detected signal pulse and maximum filtered noise pulse in the three EDF operation modes. In Angle Mode, if AM = 01, the EDF in channel 0 is replaced with the digital filter and synchronizer of the TCRCLK signal. In this mode, channel 0 works in combination with the Angle Counter logic, and their operation is fully synchronized.

Following subsections provide the functional description of the eTPU channel digital filter.

Two-Sample Mode

In this mode the EDF works like the TPU2/3 digital filter. It uses the filter clock which is the system clock divided by (2, 4, 8,.., 256) as a sampling clock. The filter clock is selected by the FPSCK field in the Engine Configuration Register (ETPU_ECR) (see *Section , ETPU_ECR – eTPU Engine Configuration Register*). The EDF compares two consecutive samples. If both samples have the same value, the input signal state is updated. Note that when the FPSCK field selects the system clock divided by two, the EDF works like the TPU1 four-clock digital filter.

Three-Sample Mode

In this mode, like in the TPU2/3 mode, the EDF uses the filter clock as a sampling clock. The EDF compares three consecutive samples. If all three samples have the same value, the input signal state is updated.

The Three-Sample mode gives more signal latency than the Two-Sample mode, but also better noise immunity and better ratio between minimum detected signal pulse to maximum filtered noise pulse. When a certain filter clock frequency is selected for Two-sample mode, double filter clock frequency can be selected to get better latency in Three-sample mode.

Continuous Mode

In this mode the EDF compares all the values sampled at the rate of system clock divided by two, between two consecutive filter clock pulses. If the signal is continuously stable for the entire digital filter clock period (i.e all the samples have the same signal value), the input signal state is updated.

This method gives the same latency and the same ratio between minimum detected signal pulse to maximum filtered noise pulse, as the Two-Sample mode, as long as there is no noise. Each sampled noise delays the signal transition detection by at least a whole digital filter clock period.

The Continuous mode gives the best noise immunity by comparing multiple samples of the noise. On the other hand, when a short noise pulse appears in the middle of the filter clock period at the same time of a real signal transition, the Continuous mode may reject a real signal transition and delay the response to the first filter clock period in which the signal is continuously stable. This may add to the latency and also to the minimum detected signal pulse in a noisy environment.

Bypass Mode

In bypass mode the signal that feeds the edge detection comes directly from the output of the synchronizer, not filtered. Bypass mode automatically makes the channel logic work in T2/T4 timing mode (see *Section , Channel Timing Modes*).

Filter Clock Prescaler

The TCRCLK signal and each channel configured as an input have an associated synchronizer followed by a digital filter connected to the signal that samples signal transitions. After reset, the digital filter filters out high and low pulse widths smaller than the period of two system clocks with ETPU_ECR bit FCSS = 0, or 1 system clock with FCSS = 1, preventing these transitions from being input to the transition detect logic. For FPSCK = 0 and FCSS = 0, the synchronizer and digital filter are guaranteed to pass pulses that are as wide as or wider than four system clocks, meaning a minimum period of eight system clocks. These figures are halved by setting FCSS = 1. By changing the FPSCK field in register ETPU_ECR the user can select a lower clock rate for the filter signal to define wider valid pulses and filter out wider noise pulses. The filter prescaler clock control is a division of the system clock. To guarantee pulse detection by the digital filter, the pulse must cover at least the stated number of samples at the filter clock rate. For example, a two sample digital filter must sample two points in the pulse to detect it. *Table 402* shows the minimum guaranteed detected pulse width and the maximum filtered noise pulse width. The table refers only to the digital filter operation. The external pulses may have to be wider (to ensure detection) or narrower (to ensure filtering) depending on the rise/fall delay differences in the MCU receivers and internal logic. Delays introduced by synchronizer, filter and edge detection logic are explained in *Section , Input/Output signal delays*.

Table 402. Pulse Widths and Delays

Filter Control (FPSCK)		**Sample on system clock divided by:**	**Min. Width Guaranteed Detected / Max. Width filtered (Min. Filter Delay / Max. Filter Delay)**[1]	
FCSS = 0	**FCSS = 1**		**Two-Sample or Continuous Mode**	**Three-Sample or Integrator**[2] **Mode**
not avail.	000	1	2 / 1 (2 / 3)	3 / 2 (3 / 4)
000	001	2	4 / 2 (3 / 3)	6 / 4 (5 / 5)
001	010	4	8 / 4 (5 / 7)	12 / 8 (9 / 11)
010	011	8	16 / 8 (9 / 15)	24 / 16 (17 / 23)
011	100	16	32 / 16 (17 / 31)	48 / 32 (33 / 47)
100	101	32	64 / 32 (33 / 63)	96 / 64 (65 / 95)
101	110	64	128 / 64 (65 / 127)	192 / 128 (129 / 191)
110	111	128	256 / 128 (129 / 255)	384 / 256 (257 / 383)
111	not avail.	256	512 / 256 (257 / 511)	768 / 512 (513 / 767)

1. This table shows pulse widths and delays in number of periods of the system clock.
2. Integrator mode is available for TCRCLK filtering only, see *Section , TCRCLK digital filter*.

Note: If the ETPU_TBCR field TCRCF selects the filter clock of the channels (see Section , ETPU_TBCR – eTPU Time Base Configuration Register), the TCRCLK filter will be clocked as if FCSS = 0 always dividing system clock /2 using FPSCK, regardless if FCSS is 0 or 1.

Channel Timing Modes

Channels can work on two different timing schemes, defining the period of channel clocking, tied to T2 and T4 microengine phases, as explained in subsections below. Microengine T2 and T4 phases are shown in *Section 22.7.1, Microcycle and I/O timing*.

T2 Channel Timing

In T2 timing mode the channel event state can only be updated every two system clocks (see *Figure 441*). Pin state, TDLs, MRLs and Capture registers are updated on the microengine's T2 clock phase. MRLE clears also happen on T2, but MRLE setting occurs on T4, together with the Match register write by microcode (see *Section , Write Channel Match and UDCM Registers*).

Channels work in T2 timing mode when all the following conditions are true:

- ETPU_TBCR bit TCR1CS is 0 (see *Section , ETPU_TBCR – eTPU Time Base Configuration Register*).
- the Enhanced Digital Filter is not configured as bypass (see *Section , Enhanced Digital Filter – EDF*).
- ETPU_ECR bit FCSS is 0 (see *Section , Filter Clock Prescaler*).

T2/T4 Channel Timing

In T2/T4 timing mode the channel event state can be updated on any system clock (see *Figure 442*). Pin state, TDLs, MRLs, MRLEs, and Capture registers are updated either on microengine's T2 or T4 clock phases. MRLE clears can happen on T2 or T4, but MRLE setting occurs on T4 only, together with the Match register write by microcode (see *Section , Write Channel Match and UDCM Registers*).

Channels work in T2/T4 timing mode when either one the following conditions are true:

- ETPU_TBCR bit TCR1CS is 1 (see *Section , ETPU_TBCR – eTPU Time Base Configuration Register*).
- the Enhanced Digital Filter is configured as bypass (see *Section , Enhanced Digital Filter – EDF*).
- ETPU_ECR bit FCSS is 1 (see *Section , Filter Clock Prescaler*).

22.5.6 Time Bases

Each eTPU engine has two Time Counter Registers, TCR1 and TCR2. They provide 24-bit time bases, shared by all 32 channels. Any channel can use both time bases to:

- Match channel's internal registers MatchA or MatchB;
- Capture time base value to channel's internal registers, CaptureA and/or CaptureB, when a match recognition or an Input transition detection occurs. For more information on channel events refer to *Section 22.5.5, Enhanced Channels*.

The TCR1 and TCR2 counters are accessible by the microcode for read and write operations. Its current value is used for getting the current time, and the captured values are used for channel relative time calculations of future events. When they are written at the same time they are incremented from any clock source, the written value prevails.

TCR1 with ETPU_TBCR[TCR1CS] = 0 and TCR2 values are updated in T2 and read in T4 (see *Section 22.7.1, Microcycle and I/O timing*). TCR1 can also work at full-speed system clock, and so be updated on both T2 and T4, when ETPU_TBCR[TCR1CS] = 1. Both TCR1 and TCR2 values can be imported from or exported to the STAC bus. When their values are imported (STAC Clients), these registers are written from the STAC bus and can only be read by microcode. For information on STAC bus protocol and definition of STAC modules refer to IPI STAC and *Section , STAC Interface*.

The TCR2 counters between the two engines are out of phase by 1 system clock, even when Time Bases are shared between them through STAC. It also applies to TCR1 counters if ETPU_TBCR[TCR1CS] = 0, but they can be in phase otherwise.

Timer Count Register 1 – TCR1

TCR1 can be used in the following modes:

- Internally Clocked Mode
- Externally Clocked Mode
- STAC Bus Client Mode

The host program can read TCR1 time base through the ETPU_TB1R (see *Section , ETPU_TB1R – eTPU Time Base 1 (TCR1) Visibility Register*).

The TCR1 bus runs through all the local engine channels. In channels which select TCR1 as MatchA and/or MatchB source, when its value is greater or equal to the programmed match value, a Match A and/or Match B event occurs on that channel. A recognized match event sets its related Match Recognition Latch 1 or 2, and according to the Predefined Channel Modes (PDCM) it may generate a channel service request. For details on eTPU channels refer to *Section 22.5.5, Enhanced Channels*.

Externally clocked mode

TCR1 can be driven externally by the TCRCLK input, after the digital filter. The TCR1 clock source is configured by the TCR1CTL bit, as shown in *Figure 415*. For more information on clock source selection, please refer to *Section , ETPU_TBCR – eTPU Time Base Configuration Register*.

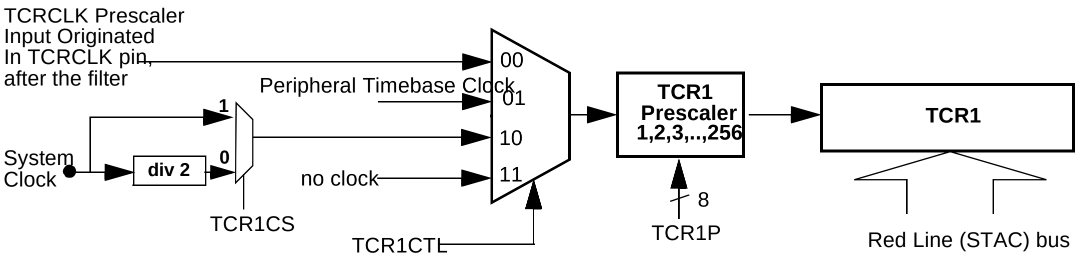


Figure 415. TCR1 Clock Selection

Internally clocked mode

TCR1 can be driven by the system clock or system clock divided by 2, before the prescaler. TCR1 can also be clocked by a Peripheral Timebase Clock generated within the MCU, also selected by TCR1CTL. The maximum frequency of the Peripheral Timebase Clock is system clock divided by two. TCR1 advances only on the rising edges of the Peripheral Timebase Clock. The use of this clock is MCU-dependent. TCR1CTL can also be used to freeze TCR1 clock independently of TCR2 (unlike GTBE).

TCR1 clock prescaling

Any clock source selected by TCR1CTL is prescaled by a factor of 1 to 256, selected by ETPU_TBCR field TCR1P. For more information on prescaler configuration refer to *Section , ETPU_TBCR – eTPU Time Base Configuration Register*. The TCR1 Prescaler resets when etpu_gtbe_in is negated. After reset, it starts counting up to TCR1P when etpu_gtbe_in is asserted. When TCR1 increments (etpu_gtbe_in = 1), the prescaler starts a new count and the new TCR1P becomes effective. When TCR1 is written by microcode, the prescaler is reloaded with TCR1P and it becomes effective, if etpu_gtbe_in is asserted.

STAC bus client mode

In this mode the TCR1 register is continuously updated from the STAC bus, and the clock selection and prescaling logic becomes ineffective. It is not writable by the microcode, and when read, it reflects the STAC bus imported value. The use of EAC is forbidden in client mode. This mode is configured through the register ETPU_REDCR (see *Section , ETPU_REDCR – eTPU STAC Configuration Register*).

STAC bus server mode

TCR1 bus can be exported to the STAC bus as a server, providing time information to other peripherals. This mode is configured through the register ETPU_REDCR (see *Section , ETPU_REDCR – eTPU STAC Configuration Register*).

Timer Count Register 2 – TCR2

The TCR2 is a 24-bit counter which can be used in the following modes:

- Pin Transition Mode: Count the rise, fall or both transitions of TCRCLK signal.
- Angle Clock Mode: Count internal tooth angle in combination with the eTPU Angle Counter (EAC) hardware which implements an Angle PLL, and generates angle information to the channels. This mode is targeted for angle based applications.
- STAC (STAC) Bus Client Mode: TCR2 is driven by an external source (see *Section , STAC bus client mode*).
- Gated Mode: Count with rate derived from the system clock divided by eight. The TCRCLK signal is used to gate this count, enabling pulse accumulator operations.
- Internally Clocked Modes: TCR2 is driven by internal clock, with count rate either system clock divided by eight or driven from the rising edge of a Peripheral Timebase Clock defined at MCU integration. The use and rate of the Peripheral Timebase Clock is MCU-dependent, but must not exceed system clock divided by two.

All clock sources pass through a prescaler. In addition, the TCR2 count can be originated from the EAC which is a hardware angle clock and angle counter. *Figure 416* shows the diagram for TCR2 clock control. When TCR2 is not driven by the EAC or STAC, the ETPU_TBCR field TCR2CTL selects the clock source, also allowing TCR2 to be frozen independently of TCR1 (see *Section , ETPU_TBCR – eTPU Time Base Configuration Register*). When in Angle Mode, TCR2CTL selects the TCRCLK edge sensitivity.

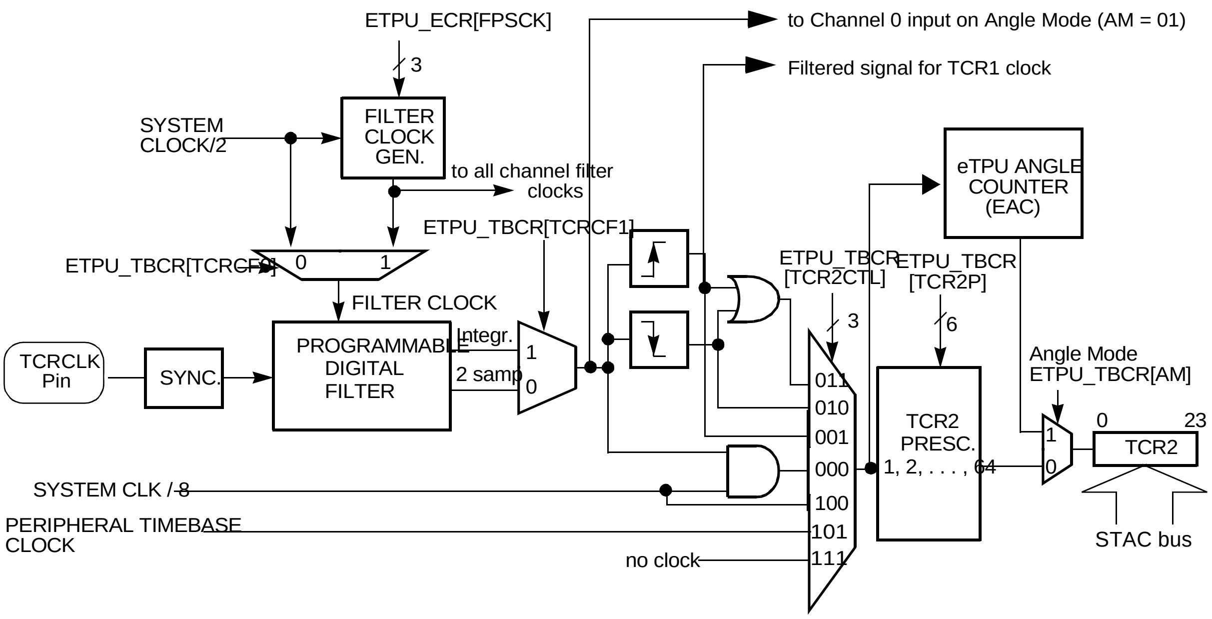


Figure 416. TCR2 Clock Control

The TCRCLK signal input is passed through a synchronizer and a programmable digital filter. In Angle Mode with AM = 01, synchronizer and filter are also used in Channel 0, replacing its input synchronizer and filter, to get the same timing in the EAC and Channel 0. The TCRCLK synchronizer is an improved filter that provides best latency while maintaining proper noise filtering (see *Section , ETPU_TBCR – eTPU Time Base Configuration Register*, field TCRCF[1:0]—TCRCLK Signal Filter Control).

The TCR2 bus runs through all the local engine channels. It transitions on clock T2 (see *Section 22.7.1, Microcycle and I/O timing*). In channels which select TCR2 as MatchA and/or MatchB source, TCR2 value is compared against MatchA and/or MatchB registers. A recognized match event sets its related Match Recognition Latch 1 or 2, and according to the Predefined Channel Modes (PDCM) it may generate a channel service request. For details on eTPU channels refer to *Section 22.5.5, Enhanced Channels*.

The TCR2 counter is accessible by the microcode for read and write operations. Its current value is used for getting the current counter value (representing signal transitions, time or angle), and the captured values are used for channel relative count calculations of future events. The TCR2 value is readable to the host through the ETPU_TB2R (refer to *Section , ETPU_TB2R – eTPU Time Base 2 (TCR2) Visibility Register*). When the TCR2 bus value is imported from the STAC bus (STAC client mode), TCR2 is not writable by the microcode, and read access from the microcode or from the host reflect the imported TCR2 value.

TCR2 clock prescaling

Except in Angle Mode, any clock source selected by TCR2CTL is prescaled by a factor of 1 to 64, selected by ETPU_TBCR field TCR2P. For more information on prescaler configuration refer to *Section , ETPU_TBCR – eTPU Time Base Configuration Register*. The TCR2 Prescaler resets when etpu_gtbe_in is negated. After reset, it starts counting up to TCR2P when etpu_gtbe_in is asserted. When TCR2 increments (etpu_gtbe_in = 1), the prescaler starts a new count and the new TCR2P becomes effective. When TCR2 is written by microcode, the prescaler is reloaded with TCR2P and it becomes effective, if etpu_gtbe_in is asserted.

The counter that divides the system clock by 8 before the prescaler also resets when etpu_gtbe_in is negated, or when TCR2 is written by microcode.

TCR2 gated mode

TCR2 Gated mode is selected in field TCR2CTL of register ETPU_TBCR. In this mode the TCRCLK signal enables or disables transfer of the system clock divided by 8 to the TCR2 prescaler. By programming the prescaler, TCR2 can run at rates from system clock divided by eight down to system clock divided by 512, in steps of eight system clock divisions. For more information refer to *Section , ETPU_TBCR – eTPU Time Base Configuration Register*.

TCR2 signal transition modes

These modes are selected when the TCR2CTL field in ETPU_TBCR is set to rise, fall or “rise-and-fall”. In these modes the TCRCLK signal is the TCR2 clock source, and its maximum transition rate depends on the TCRCLK digital filter mode of operation. The TCRCLK digital filter can be programmed to use the system clock divided by two, or use the same filter clock of the channels, controlled by the TCRCF field in ETPU_TBCR. It contains an up-down counter which operates as a digital integrator, optimizing signal latency in the selected mode and clock rate.

When system clock divided by two is selected, the synchronizer and the digital filter are guaranteed to pass pulses that are wider than four system clocks (two filter clocks). Otherwise the TCRCLK is filtered with the same filter clock as the channel input signals. For details on TCRCLK and channels digital filter control refer to *Section , ETPU_TBCR – eTPU Time Base Configuration Register*, and *Section , Enhanced Digital Filter – EDF*.

STAC bus client mode

In this mode the TCR2 register is continuously updated from the STAC bus, and the clock selection and prescaling logic becomes ineffective. It is not write accessible for the microcode, and when read, it reflects the STAC bus imported value. The use of EAC is forbidden in client mode. This mode is configured through the register ETPU_REDCR (see *Section , ETPU_REDCR – eTPU STAC Configuration Register*).

STAC bus server mode

When TCR2 bus is exported to the STAC bus as a server, it can provide either time or angle bus to other peripherals, according to its operation mode. This mode is configured through the register ETPU_REDCR (see *Section , ETPU_REDCR – eTPU STAC Configuration Register*). To provide sequential update of the STAC clients, the Angle tick rate must not be faster than the STAC programmed update rate. This requirement puts a limitation on the angle clock count rate on high rate mode. In this case the Angle and Angle Fraction accumulator (see *Section , Angle tick generator*, and *Figure 422*) are advanced at rate of system clock divided by eight. Therefore, the STAC update rate for the Angle Bus must not be slower than eight system clocks.

TCR2 bus in angle clock mode

In this mode the TCR2 counter operates as part of the eTPU Angle Counter (EAC). The TCR2 bus value reflects this angle representation in which it counts Angle Ticks. Angle Mode is selected when the AM bit is set in ETPU_TBCR.

Note that when TCR2 works in Angle Mode, it does not count directly from the TCR2 clock input which indicates tooth signal transition. Its Angle counter is controlled by the Count Control and High Rate logic (see *Section , Count control and high rate logic*), which provides the interpolated pin position, and handle cases of missing tooth, acceleration, de-acceleration and mechanical corrections.

The EAC uses the TCRCLK signal to get the tooth transition indications. The TCR2CTL field in ETPU_TBCR has to be set for the appropriate tooth edge detection rise, fall, “rise-and-fall” or none. TCR2 count clock comes from the EAC control and not directly from the physical tooth. This way the EAC control processes the signal transitions and handles missing teeth and flywheel mechanical corrections. Note that when TCR2CTL selects “none” for tooth edge selection, the TCR2 is not necessarily frozen, but can still be incremented by the EAC logic.

In Angle Mode, eTPU channel 0, 1 or 2 operation is combined with the EAC operation. When channel 0 is selected for EAC operation, the TCRCLK digital filter is used both by the EAC and by channel 0 to get full synchronization between the two logics.

The eTPU Angle Counter (EAC) logic runs continuously and updates the TCR2 Angle counter, eliminating the microcode latency in updating the TCR2 value.

STAC Interface

Both time bases TCR1 and TCR2 can be shared between the engines and with other blocks in the same MCU. Each one of both eTPU engines can drive their time bases to the STAC (Shared Time and Count) bus, acting as a server, while any other block can capture the value into its resources and behave like a client. For further reference about the STAC bus operation refer to *Section , ETPU_REDCR – eTPU STAC Configuration Register*.

The eTPU can export to the STAC bus or import from the STAC bus the following internal resources:

- TCR1: Can be exported to or imported from the STAC bus. TCR1 can only be imported from STAC bus when the engine is not in Angle Mode. When TCR1 is imported from the STAC bus, it becomes read-only for the microcode and reflects the imported values. For details refer to *Section , Timer Count Register 1 – TCR1*.
- TCR2: Can be exported to or imported from the STAC bus. TCR2 can only be imported from the STAC bus when engine is not in Angle Mode. When TCR2 is imported from the STAC bus, it becomes read-only for the microcode, and reflects the imported values. When exported to the STAC bus, TCR2 can work in either Angle Mode or as a free running counter associated with the TCRCLK signal. For details refer to *Section 22.5.7, EAC – eTPU angle counter*.

Proper configuration of the following bits is necessary to determine what can drive the STAC bus: ETPU_TBCR[AM] and ETPU_REDCR[REN2, RSC2], according to *Table 403*.

Table 403. STAC Bus and Host Read Sources

AM (ETPU_TBCR)	REN2,RSC2 (ETPU_REDCR)	TCR2 Bus Source (Host read of ETPU_TB2R)	STAC Bus Driver
00	0x (disabled)	TCR2/Time	x
01, 10 or 11	0x (disabled)	TCR2/Angle	x
00	11 (Server)	TCR2/Time	TCR2/Time

Table 403. STAC Bus and Host Read Sources

AM (ETPU_TBCR)	REN2,RSC2 (ETPU_REDCR)	TCR2 Bus Source (Host read of ETPU_TB2R)	STAC Bus Driver
01, 10 or 11	11 (Server)	TCR2/Angle	TCR2/Angle
01, 10 or 11	10 (Client)	Forbidden(1)	

1. **STAC** client configuration in Angle Mode is also forbidden for TCR1.

Note that Angle Mode is not available for STAC bus clients: configuring both at the same time brings unpredictable results. When TCR2 is a stand-alone counter or a STAC Bus server, the same value that is driven to the internal TCR2 bus is also exported to the STAC bus (either Time Count or Angle).

STAC bus configuration is provided by the ETPU_REDCR bits REN1/2 and RSC1/2. REN1/2 enable the STAC interface to interact with the resource (either TCR1 or TCR2 bus). RSC1/2 configure the resource (either TCR1 or TCR2 bus) as Server or Client.

Each time base / angle count resource from each engine receives a unique 4-bit hard-wired address that identifies it as a potential server. This address is used by the STAC Controller to coordinate which resource will drive the bus at a given STAC time-slot. For any time-slot there is a server driving the bus upon selection of the STAC Controller, and there may be a client linked to that server by the ETPU_REDCR bits SRV1/2 on each engine. When the server address on the STAC bus matches the value in SRV1/2, the client will load the STAC information into the appropriate resource. For information on eTPU STAC Bus configuration refer to *Section , ETPU_REDCR – eTPU STAC Configuration Register*.

The eTPU does not include a STAC Controller module, which is instantiated once in the system integration.

Note: Setting a timebase as client of itself is forbidden, and results are unpredictable.

GTBE – Global time base enable

The GTBE bit in ETPU_MCR enables time bases in both engines, allowing them to be started synchronously. GTBE is divided in two block interface signals: etpu_gtbe_out and etpu_gtbe_in. GTBE bit sets etpu_gtbe_out, and etpu_gtbe_in enables time bases to start. The etpu_gtbe_out signal can be used by MCU integration for synchronization between eTPU time bases and time bases from other modules. If the GTBE bit in ETPU_MCR must enable only the eTPU time bases, etpu_gtbe_out is simply connected to etpu_gtbe_in. These two cases are shown in *Figure 417*. Synchronization logic can be as simple as an OR or an AND logic gate.

Once etpu_gtbe_in transitions to 1, the Engine 1 Time Bases start one system clock earlier than Time Bases in Engine 2, except when TCRCLK is selected as clock source or TCR1 when ETPU_TBCR[TCR1CS] = 1. This happens independently of prescaler values as long as they are the same for both engines, because the prescalers also freeze when etpu_gtbe_in = 0. Microcode can always write to TCR1/2 registers, with either value of etpu_gtbe_in.

Note: The timebase prescalers are reset when the GTBE input is negated.

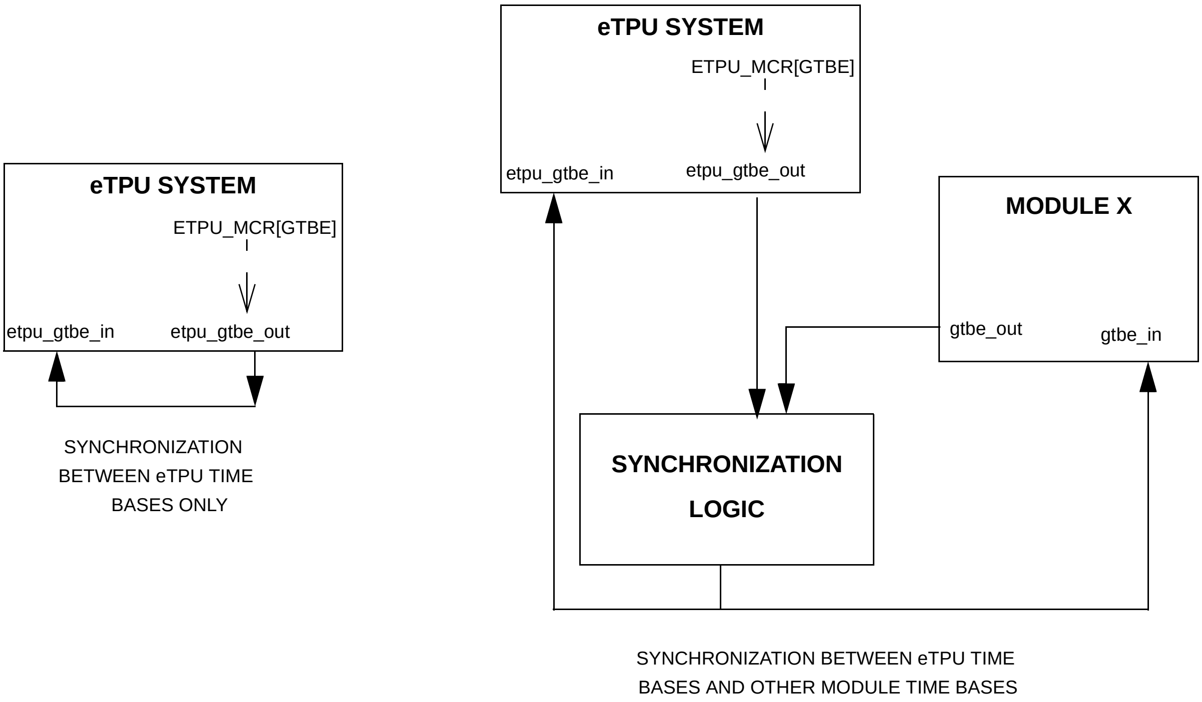


Figure 417. Time base synchronization

TCRCLK digital filter

The TCRCLK signal has an improved integrating digital filter with a 2-bit up-down counter. The counter counts up to 3 when a high signal level is detected, or down to 0 when a low level is detected. The signal state is updated to one when the counter stops at 3, or zero when the counter stops at 0. The field TCRCF in register ETPU_TBCR (see *Section , ETPU_TBCR – eTPU Time Base Configuration Register*) determines whether the TCRCLK signal input (after a synchronizer) is filtered with the same filter clock as the channel input signals (see *Section , Enhanced Digital Filter – EDF*) or uses the system clock divided by 2, and also whether the TCRCLK digital filter works in integrator mode or the same two sample mode as the channel filters (see *Table 359*).

The TCRCLK filter delay and prescaling determines the minimum detectable TCRCLK pulse widths and, therefore, its maximum frequency, as shown in *Section , Filter Clock Prescaler*, and *Table 402*. The TCRCLK signal delay from the module input to TCR1/TCR2 incrementing or detection in the EAC logic is explained in *Section , Input/Output signal delays*.

22.5.7 EAC – eTPU angle counter

General

The EAC logic contains a mechanism which follows the flywheel angle, based on the tooth rate. This hardware works in combination with the TCRCLK signal, the TCR2 counter and Channel 0, 1 or 2 (depending on the ETPU_TBCR field AM) to generate angle information on the TCR2 bus which is passed to all the local engine channels. The EAC helps to implement a digital angle PLL (see *Table 421*), which combines hardware with microcode processing at channel 0, 1 or 2. The angle measurement is based on history knowledge of the tooth period, for predicting the period of the next tooth. The tooth period is partitioned into a programmable number of **Angle Ticks**. The eTPU application will use the divider in the MAC/Divide unit to calculate an integer and a fraction part of the angle tick such that the full tooth period gets the correct programmed number of angle ticks with no accumulated error.

Each single tooth can be divided in angle ticks, up to 1024. In a 60-tooth flywheel, 128 Angle Ticks per tooth provide resolution of ~0.05 degrees per tick, which meets the accuracy requirement of 0.1 degrees in current automotive applications.

The measurement of one tooth in angle ticks is independent on engine RPM; it is the tooth period itself (and the corresponding tick period) that is re-calculated for each new tooth, based on the difference between the estimated tooth and the actual detection.

For these applications, one of the eTPU channels 0, 1 or 2 is dedicated to service the physical tooth detection. Channel 0 shares the same filtered input as the TCRCLK signal to get the same timing as the EAC. The TCRCLK edge detection is selected by ETPU_TBCR field TCR2CTL for the EAC, and by IPACA/B on channel 0, which must be set to detect the same edge(s). When channels 1 or 2 are selected to work with the EAC, IPACA/B is used to select the tooth signal edge detection for both the channel and the EAC, and the tooth signal that feeds the EAC is the same filtered input which feeds the channel.

Channel 0, 1 or 2 generates the signal transition service request, and can also be used for generation of a window filter on this transition, to qualify TCR2 clocks. For this purpose, the selected channel should be configured with double match window filtering mode (refer to *Section , Channel Modes*). Depending on the channel mode set for the channel, Match A recognition opens the window, and Match B recognition may close it or leave it open. See *Section , Angle logic and channel modes*, for details. Match B also generates a time-out service request. Its input signal transition comes from the tooth. The window can be defined by microcode to open at a predefined point inside the tooth period, and stay open for a desired percentage to the tooth period. The window can be measured in angle or time This method improves the noise immunity by allowing transition detection only on an expected period, a feature which was software responsibility in previous TPU versions.

The EAC supports deceleration, acceleration, last tooth and missing tooth scenarios. The large range of angle ticks per tooth can be used to cover longer tick counts caused by one or more missing teeth, or to provide extra resolution for future application requirements. In case of a missing tooth, the EAC can be configured to insert a dummy tooth or to simply measure a longer tooth.

Figure 422 shows the block diagram of the Angle Counter system. TCR1 is used as a time base which measures the tooth period and is used for partitioning the period to angle ticks.

Angle mode registers

In Angle Mode, the registers described below control eTPU angle operations. They are accessible only by microengine as source and destination registers in microinstructions. When eTPU is not in angle mode (AM bit is negated in ETPU_TBCR), all angle mode registers can be used as general purpose registers.

TPR – Tooth program register

TPR provides configuration for the Angle Counter circuit. In this register, the microcode can properly adjust the tooth count (controlling last tooth, missing teeth, dummy tooth insertion, halt until tooth detection) and the number of angle ticks per tooth (field TICKS). Note that this register is sampled into a temporary register in the EAC logic when the High Rate Mode is detected (see *Section , High rate mode (Acceleration)*, and *Section , TPR buffering*), which means that changes to this register may take effect only for the next tooth.

Refer to *Section , Count control and high rate logic*, and to *Section , Special cases of missing teeth and last tooth*, until *Section , Handling false tooth detection*, for a detailed explanation about the use of this register. *Figure 418* provides a detailed description of the TPR.

Several conflict issues on TPR writes are explained in *Section , Special TPR write cases*.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	LAST	MISS CNT		IPH	HOLD	TPR 10	TICKS									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 418. TPR Register

Field	Description
[9:0] TICKS	Angle Ticks Number in the Current Tooth This field defines the number of angle ticks in the current physical tooth. It partitions the tooth period to the required number of angle ticks. The actual number of angle ticks in a tooth is (TICKS+1). In High Rate mode (see *Section , High rate mode (Acceleration)*), TPR writes are immediately effective only for bits IPH and HOLD. All other fields changes are “buffered” and become effective when EAC leaves High Rate mode. See also *Section , Special TPR write cases*. Bits LAST, IPH and HOLD must not be asserted all at once.
10 TPR	TPR register Reserved bit. In Angle Mode, must always be written 0 by the user, but holds the value written, so that TPR can be used as a general purpose register bit when angle mode is off.
11 HOLD	Force EAC Hold This bit forces the EAC to halt its operation in a special EAC freeze mode until a new physical tooth (a real one or emulated with IPH = 1) is detected. Assertion of this bit immediately freezes the EAC in the middle of the tooth period. When a new physical tooth is detected, the bit is automatically negated by the EAC. The HOLD bit can be used for synchronizing the EAC tooth count, in case that a false physical tooth is detected due to noise. Normal Operation. Force EAC to halt until detection of a physical tooth.

Field	Description
12 IPH	Insert Physical Tooth This bit generates a dummy physical tooth which has the same effect as a real physical tooth, and resets itself subsequently. If EAC is in Halt mode, it switches back to Normal mode(1). If EAC is in Normal Mode, it switches to High Rate Mode. If Angle Logic is frozen by HOLD = 1 (see below), it returns to the state it was at the freezing moment. No Operation. Insert dummy physical tooth. IPH reads as 1 in the next microinstruction after it is asserted, negating subsequently. However, it can be set twice in two consecutive microinstructions to generate two teeth and make the EAC go from Halt to Normal to High Rate Mode.
13-14 MISSCNT	Missing Tooth Counter Decremented on each estimated tooth, stops at zero. Used for generation of “Dummy Tooth” whenever it holds a non-zero value. 00 No missing tooth 01 One missing tooth 10 Two missing teeth 11 Three missing teeth If the tooth is detected or inserted before the missing tooth tick count completes (going High Rate mode, see *Section , High rate mode (Acceleration)*), MISSCNT resets immediately, but missing teeth count continues in High Rate mode (see *Section , TPR buffering*).
15 LAST	Last Tooth Indication Asserted by microcode and negated when a tooth is detected or inserted via IPH. Not Last Tooth. Last Tooth - reset TCR2 Counter at the end of the tooth tick count (after physical tooth or IPH = 1) when MISSCNT = 0. If the tooth is detected or inserted before the tooth tick count completes (going High Rate mode, see *Section , High rate mode (Acceleration)*), LAST resets immediately, but TCR2 resets only when the tooth count completes and MISSCNT = 0

1. Missing a physical tooth naturally causes EAC to get into Halt mode.

TCR2 – Timer Counter 2

In Angle Mode TCR2 counts angle ticks instead of time.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	Angle Tick Counter[23:16]							
W																
Reset									0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Angle Tick Counter[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 419. TCR2 in Angle Mode

This 24-bit free-running counter is used to generate an accumulated Angle Fraction value. It is updated by the Angle Tick Generator (refer to *Section , Angle tick generator*, for more details). Refer to *Section , Count control and high rate logic*, for a detailed explanation about the use of this register in Angle Mode.

TCR2 provides continuous count of the angle in units of angle ticks. The Angle Tick Counter in TCR2 can be reset due to “Last Tooth” microcode indication. TCR2 Prescaling is disregarded in Angle Mode: physical tooth detection is done by EAC regardless of the value set in TCR2P.

TRR – Tick Rate Register

The exact period of the Angle Tick is programmed in the Tick Rate Register by microcode. The period of the Angle Tick is given in units of TCR1 clocks as system clocks divided by 2*(TCR1P + 1), even if TCR1CS = 1 (see *Section , ETPU_TBCR – eTPU Time Base Configuration Register*). Refer to *Section , Calculating the angle tick period integer and fraction*, for a complete description about the mechanism to calculate the value to be written into this register.

	31	30	29	26	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	INTEGER[14:7]							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	INTEGER[6:0]							FRACTION								
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 420. TRR Register

INTEGER[14:0]—The integer part of TCR1 clocks in one Angle Tick.

This number, decremented by one, works as a down-counter preload value. A value of INTEGER = 0 represents an integer of 32768. A new value written is reloaded into the counter (becoming effective) when a new tick starts or a tooth is detected or inserted via IPH.

FRACTION[8:0]—Nine-bit fractional part of TCR1 clocks in one angle tick.

The FRACTION value is accumulated in the EAC Fraction Accumulator, and whenever the result overflows (i.e., the accumulated fraction added up to an integer), the Tick Prescaler is halted for one TCR1 clock.

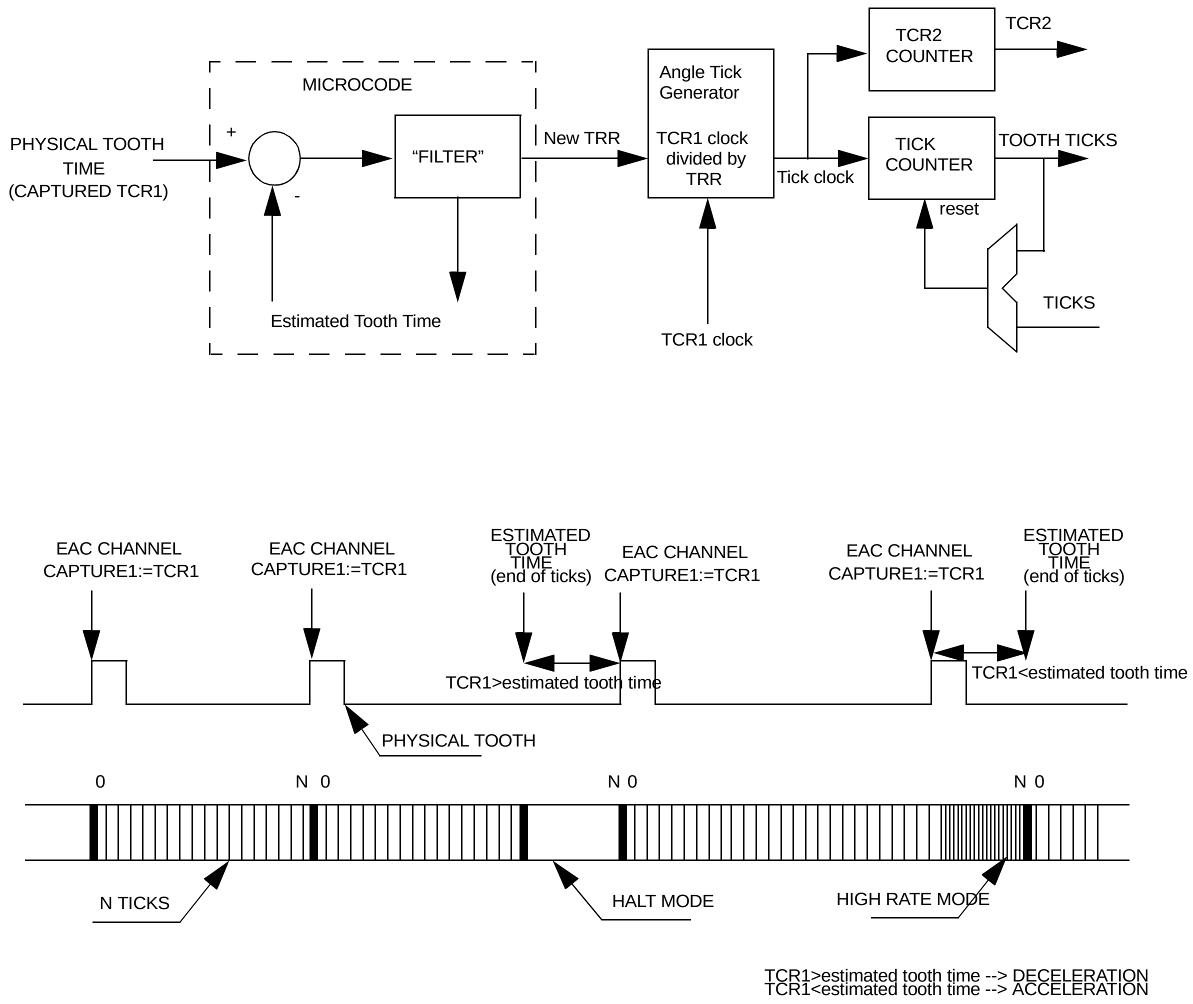


Figure 421. EAC "PLL"

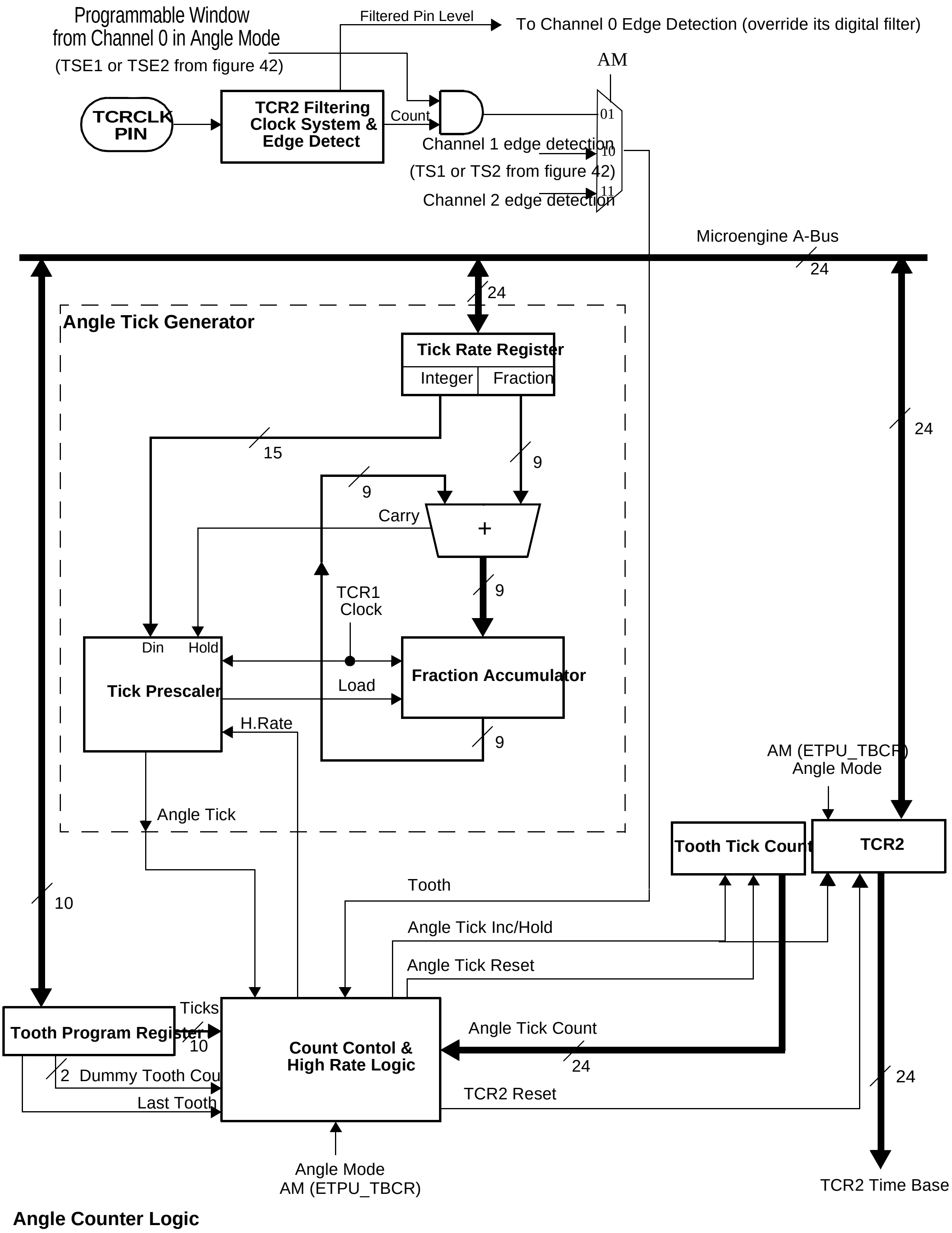


Figure 422. eTPU angle counter system

Acceleration and deceleration

Acceleration and deceleration affect the new tooth period relative to the known period of the last tooth. Changes in tooth period may be extreme at very low engine RPM (such as cold start and warm start). The worst case of tooth period changes is caused during missing teeth, since there is more time for changes in angular velocity to be unnoticed by the EAC hardware. For example, on cold start (~20 RPM) there may be extreme acceleration: the ratio between a known tooth period before two missing teeth and the new tooth period after the missing teeth can be very high (up to a factor of 75). Acceleration and deceleration effects from tooth to tooth are less extreme as the engine climbs to high RPM.

In case of deceleration, the estimated tooth period ends before the actual tooth detection arrives. In this case, the EAC hardware waits at the end of the current tooth period, when it is said to be in **Halt mode**, until the real tooth indication is received, then continues with normal operation (**Normal mode**). See *Table 421*.

In case of acceleration, the actual tooth period is shorter than the estimated tooth period. As a result, a new physical tooth indication arrives before the end of the estimated tooth period. In this case the EAC closes the gap on **High Rate mode** by counting on system clock divided by eight to the end of the tooth, advances to the next tooth, and switches back to normal operation mode. See *Table 421*.

The reason that the EAC does not jump directly to the next tooth is the need to provide sequential angle count throughout the whole tooth period, for channels or external STAC clients (if TCR2 is a STAC server) which compare angle in “equal” mode. These peripherals must get all the valid angle values in a sequential manner, to avoid missing angle matches.

TCR2 advancing from one tooth to another is a continuous count, and can be optionally reset at the end of the tooth. An estimated tooth is generated after the Tooth Tick Counter reaches the TICKS programmed value.

The EAC works continuously and switches automatically between Normal, Halt and High Rate modes. It relies on the microcode to calculate the estimated tooth period on every tooth, and to update the correct angle tick and tooth parameters in the EAC control registers. On high RPM, tooth period changes are reduced from tooth to tooth, and the EAC may follow the angle with good accuracy for several teeth without microcode intervention.

The EAC handles missing teeth by insertion of “dummy” teeth, or by enlarging the expected tooth period. It is a good practice to locate the flywheel missing teeth in non-critical angles, since a missing tooth may increase the angle measurement error (acceleration and deceleration is detected late).

Angle tick generator

The Angle Tick Generator is responsible for generating a programmed number of angle ticks in the tooth period. It generates the ticks in an average rate which ensures completion of the correct number of angle counts in the estimated period of the tooth, since the count of one tooth in angle ticks is independent on engine RPM. The main output of the Angle Tick Generator is the tick clock that feeds TCR2 in Angle Mode, as well as the internal Tooth Tick Counter (see *Figure 422*). The Tooth Tick Counter counts ticks within a tooth, from 0 up to TICKS, is controlled by the Angle Tick Generator logic and cannot be accessed by microcode. Refer to *Figure 423* for a generic presentation of the angle tick count and the measurement of a single tooth period.

Calculating the angle tick period integer and fraction

On each tooth the microcode has to update the exact period of a single angle tick used for counting the internal angle in the tooth. The period of an angle tick or a tooth is measured in units of TCR1 clocks (if TCR1CS = 0) or TCR1 clocks divided by 2 (if TCR1CS = 1). The microcode can use the eTPU MAC Divider unit (see *Section , MAC and Divide Unit (MDU)*) to divide the tooth period by the number of angle ticks per tooth, which is stored in the TICKS field of TPR (refer to *Section , Angle mode registers*). This division yields the integer part of the angle tick period and the remainder. Dividing again the remainder shifted left nine positions, by the number of angle ticks per tooth translates the remainder to a 9-bit fraction. The microcode concatenates the 15-bit integer and the 9-bit fraction to a 24-bit value and writes it to TRR. The new rate is effective immediately after the next Angle Tick is generated by the Angle Tick Generator(ah).

For high RPM, note that shifting the tooth period value nine positions to the left prior to the first divide operation would calculate, in one operation, the integer and the fraction. For example: On 60-tooth flywheel running at 1000 RPM, tooth period is 1 ms. If TCR1 counts @ 25 MHz, it counts 25,000 times in a tooth, which can be represented by 15 bits. Therefore the tooth period can be shifted nine positions to the left prior to divide operation, and be represented with 24 bits.

Using shift left nine positions and one divide operation would get the result in MACL register (in MDU) which holds the integer and nine bits of the fraction:

```
Angle_Tick_Rate {Integer[14:0], Fraction[8:0]} = (TCR1ToothPeriod(ai)<<9) /
Ticks
TRR = Angle_Tick_Rate {Integer[14:0], Fraction[8:0]}
```

On low RPM the initial tooth period, measured in TCR1 counts, may be too big to be shifted nine positions to the left. For lower RPM (for example 500 RPM) the tooth period cannot be represented by 15 bits, and shifting it nine positions to the left would lose the MSB. In this case, two divide operations are required as follows: first divide the Tooth Period by the number of TICKS—the integer is stored in MACL and remainder in MACH. MACL is saved in another register. MACH is shifted 9 positions to the left and divided again by TICKS. In parallel with the second divide, the register which saved the original MACL is shifted left 9 positions. After the divide MACL contains the 9 bits fraction and the other register contains the 15-bit integer, shifted left nine times. The logical OR of the two registers is written to the TRR:

```
Angle_Tick_Rate {Integer[14:0]} = (TCR1ToothPeriod) / Ticks
Remainder[9:0]                  = TCR1ToothPeriod) mod Ticks
Angle_Tick_Rate {Fraction[8:0]} = (Remainder[9:0] << 9) / Ticks

TRR = Angle_Tick_Rate {Integer[14:0], Fraction[8:0]} = (Integer[14:0] << 9)
| Fraction[8:0]
```

---

ah. In High-rate mode, the tick keeps being updated at the rate of system clock/8 until it goes back to Normal mode, when the new TRR value is used.

ai. The tooth period (TCR1ToothPeriod) is not, in general, the value of estimated tooth time. It is obtained by microcode by subtracting TCR1 values between two teeth detections. Its comparison with the estimated tooth time indicates acceleration (if minor) or deceleration (if greater) to the microcode.

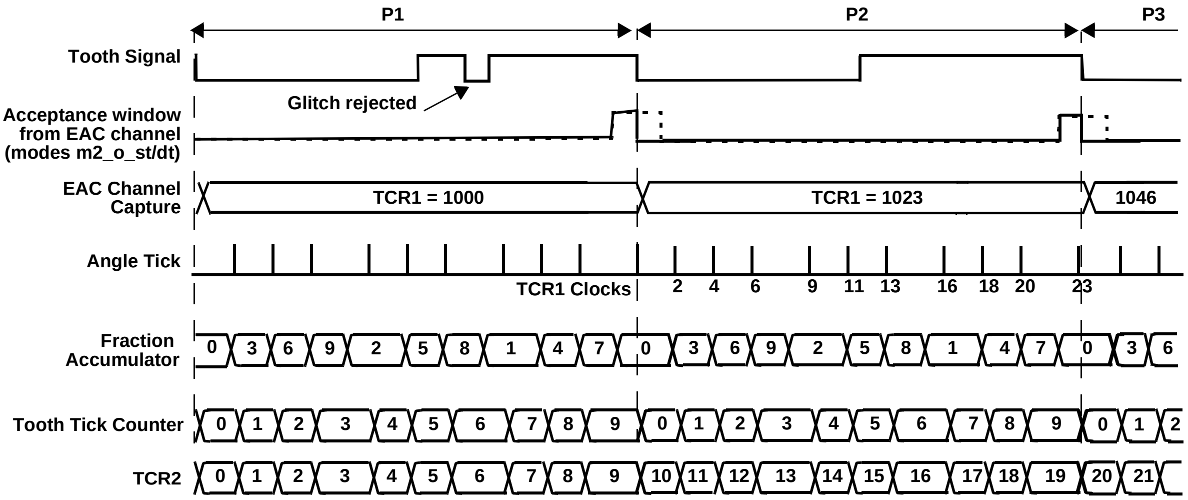


Figure 423. Angle Ticks Generation

Generating the angle ticks

The integer part of TRR is preloaded to a prescaler, which counts down at input clock rate equals to the TCR1 clock rate (TCR1CS = 0) or TCR1 clock rate divided by 2 (TCR1CS = 1) (see *Figure 422*). When the down counter reaches zero, it generates an angle tick pulse to the Angle Counter Logic and a Load pulse to the Fraction Accumulator. It is then preloaded with most updated TRR integer part. Due to the Load pulse, the 9-bit fraction is accumulated in a 9-bit Fraction Accumulator. If a fraction overflow condition occurs (the 9-bit adder asserts carry out), the accumulator saves the lower 9 bits of the addition result, which is the remaining fractional part. The carry out bit indicates an accumulated integer “one” which means that the angle tick is early by one input clock. It halts the prescaler operation for one input clock to compensate the accumulated error generated by the integer prescaler. As a result, the average angle tick period takes into account both the integer and the fraction parts. The accuracy depends on the bit count of the fraction. Using 9-bit fraction part while the width of the field TICKS in register TPR is 10 bits provides accuracy of two LSB on a full scale (TICKS = 1023) or one LSB on lower scale (TICKS<=511).

When the Tick Prescaler gets High Rate mode indication from the Angle Counter Logic, it generates angle ticks at a rate of system clock divided by eight. In this case it does not generate Load pulses to the Fraction Accumulator, ignores its “hold” input and preloads internally to a fixed period of eight system clocks. When High Rate mode is entered, the prescaler is preloaded to a period of eight system clocks before its first angle tick generation, ensuring separation of at least eight system clocks between the last Normal mode angle tick and the first High Rate mode angle tick. The fraction accumulator resets when the tick count advances to the next tooth, or when TRR is written by the microcode.

Count control and high rate logic

The Count Control and High Rate Logic controls TCR2 operation in Angle Mode, using the angle ticks generated by the Angle Tick Generator. Count Control logic is responsible for advancing, holding and resetting the TCR2 and Tooth Tick Counter in the proper timing, such that the TCR2 time base will reflect the correct estimated angle. This logic also includes the Tooth Program Register (TPR, see *Section , Angle mode registers*, for more information).

The Count Control and High Rate Logic handles deceleration, acceleration, missing teeth and last tooth. On High Rate (acceleration) it ensures that the angle bus scans all valid angle values in a rate which can be traced by the STAC bus. This operation enables external STAC clients (if TCR2 is a STAC server) or channels working in “equal-only” comparator mode to match the TCR2 exported angle information in “equal” mode, in an exact match.

Because the eTPU channels are capable of capturing either TCR1 or TCR2 due to signal transition, the microcode can get either the angle or time of the physical pin transition. Since the EAC channel (0, 1 or 2) is connected to the physical tooth, the microcode can get the EAC error in angle domain (tooth appears at the wrong angle) or time domain (physical tooth captured time into the EAC channel, relative to the estimated tooth time). Note that in angle mode, the transition detect logic of the channel 0, if selected as the EAC channel, is fed from the digital filter of the TCRCLK signal, and not from the channel 0 internal digital filter. This ensures synchronous operation of channel 0 and the EAC hardware.

Another feature of the eTPU channel, when working in Single Match And Single Transition Enhanced Mode (refer to *Section , Single Match Enhanced Mode (sm_st_e)*), is capturing a single time base due to signal transition before and after the digital filter. This option allows subtracting the digital filter delay to get accurate signal transition timing on the channel. This way, the TCRCLK signal may be programmed with a slow and reliable digital filter, and get accurate time measurement of the digital filter delay.

To assert the end of the estimated tooth period the Count Control and High Rate logic compares the TICKS field in TPR (refer to *Section , Angle mode registers*) with the current value of the tooth tick counter. When the Tooth Tick value is greater or equal to TICKS, it determines the end of the estimated tooth period. On acceleration this event occurs during High Rate mode operation, after the arrival of a physical tooth. In deceleration, this event occurs during Normal Mode, before the arrival of a physical tooth. On constant angular velocity, this event appears together with the arrival of a physical tooth.

The following sections describe the operation of the Counter Control and High Rate logic.

Normal mode

In Normal mode the Counter Control logic advances TCR2 and the Tooth Tick counter as if the engine has a constant speed during the tooth period. It receives the angle ticks from the Angle Tick Generator in an average rate which is determined by the Tooth Rate Register (TRR). This is the reset mode.

When the Tooth Tick Counter is about to reach the last value effectively stored in TPR field TICKS plus one, the hardware detects the end of the estimated tooth period. If the physical tooth and the estimated tooth arrive at the same time the EAC stays in Normal mode, the Tooth Tick Counter is reset, and TCR2 is incremented (depending on TPR bits LAST and MISSCNT). If the physical tooth and the estimated tooth do not arrive at the same time, either acceleration or deceleration is detected, and the EAC switches to the proper mode. See *Figure 424* for a detailed diagram of Normal Mode behavior.

The microcode which services the EAC channel physical tooth transition may update TRR according to various conditions to give the best estimation of the current tooth period, according to the previous tooth period and other engine parameters.

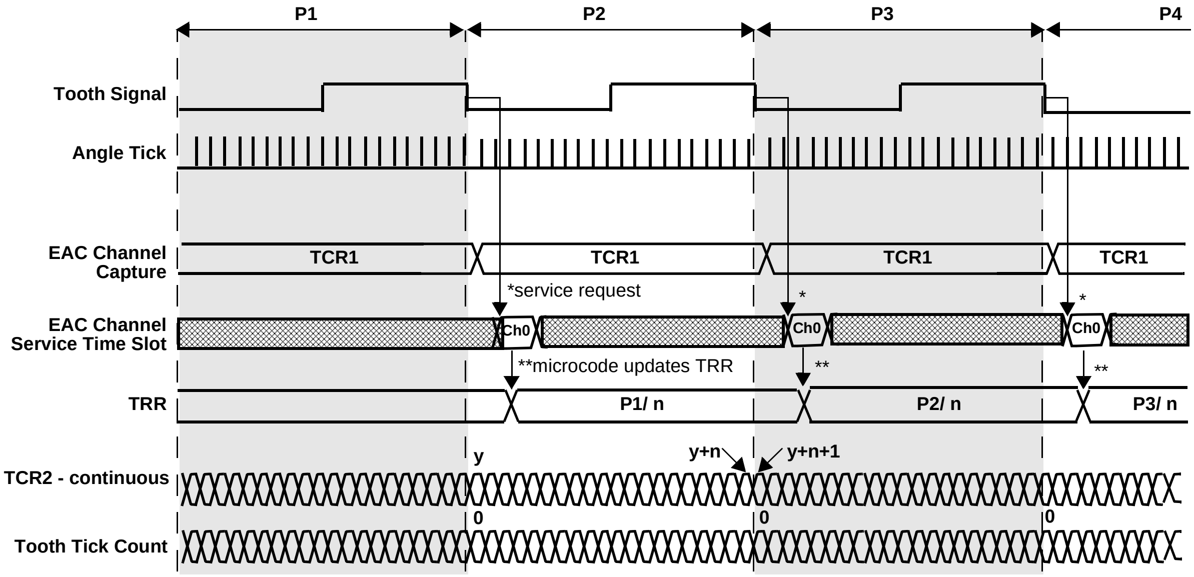


Figure 424. Normal Mode

Halt mode (Deceleration)

In case of deceleration, the Tooth Tick Counter reaches the TICKS value before the arrival of the next tooth. The Count Control logic does not reset neither advances TCR2 and Tooth Tick Counters. The Count Control logic halts TCR2 and Tooth Tick Counters at the end of the tooth, waiting for the physical tooth to arrive.

When the physical tooth is detected the EAC switches back to Normal Mode and releases TCR2 to count the angle ticks of the new tooth, also resetting the Tooth Tick Counter. Only then TCR2 may wrap to 0, if TPR bit LAST is asserted. See *Figure 425* for a detailed diagram of Halt Mode behavior.

The microcode service caused by the physical tooth determines the deceleration, calculates the new tooth period and Angle Tick period and updates TRR. This operation slows the angle tick rate generated by the Angle Tick Generator on-the-fly, to the rate required for the new tooth period.

Since the microcode service is initiated by the physical tooth edge, microcode latency may introduce a small angle error caused by using the TRR value of the previous tooth at the beginning of the current tooth. On high RPM, deceleration is relatively small but the microcode latency may take a significant percentage of the tooth period. On low RPM microcode service latency takes little percentage of the tooth period, but there may be cases of extreme acceleration and deceleration. The microcode latency can be calculated knowing TCR1 value during the service time, and TCR1 value captured in the EAC channel due to the physical tooth pin transition. The duration of the Halt mode is obtained using the estimated tooth time.

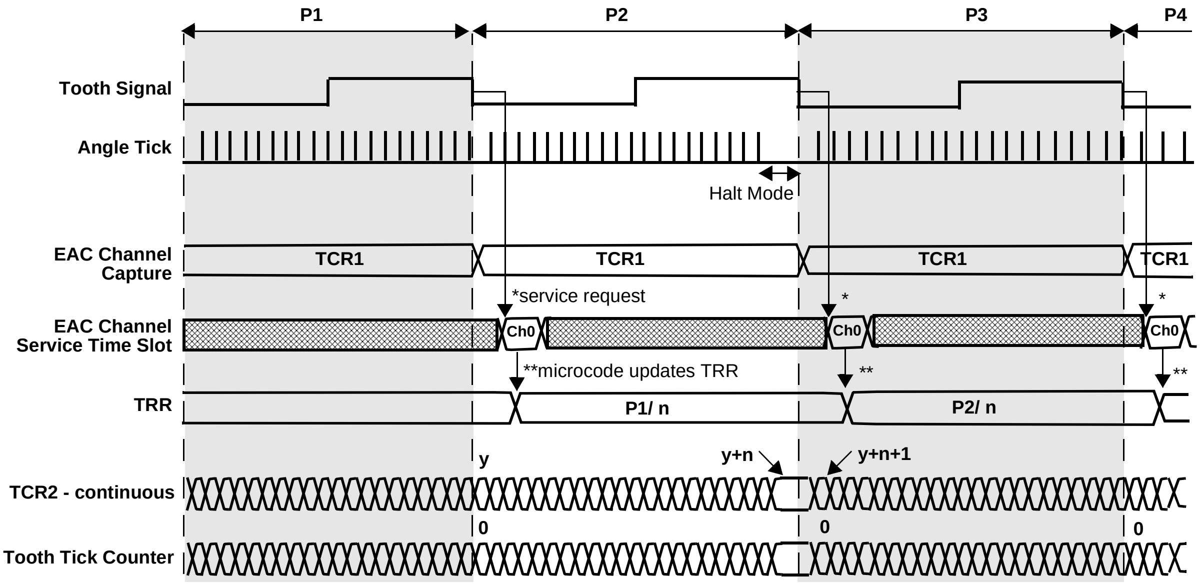


Figure 425. Halt Mode – Deceleration

High rate mode (Acceleration)

In case of acceleration, the next tooth arrives before the Tooth Tick Counter reaches the TICKS value. In this case the High Rate logic is responsible for closing the gap and advancing the tooth on the correct timing. The High Rate mode operates as follows:

- When the acceleration is detected (physical tooth arrives before the Tooth Tick Counter reaches the TICKS value), the Count Control and High Rate logic switches to High Rate mode in which both the Tooth Tick Counter and TCR2 count at rate of system clock divided by eight, until the Tooth Tick Counter reaches the current TICKS value. To ensure correct operation, the TICKS value is sampled in the logic at the beginning of the mode.
- At this point, which represents the estimated tooth edge, the logic resets the Tooth Tick Counter and advances TCR2 (or resets it if LAST is asserted and MISSCNT = 0).
- The control logic switches back to Normal Mode, using the most updated TRR value as input to the Angle Tick Generator. The logic samples the updated TICKS value for the tooth estimation, last tooth indication and number of missing teeth from TPR.

In High Rate mode the angle ticks are provided at high speed until the end of the current tooth. This operation is required to scan all the valid angle values of the current tooth, in a rate which is not too high for the STAC bus continuous update, but much higher than the rate dictated by TRR.

EAC channel microcode, which services the physical tooth transition detection, can start its service either before High Rate mode operation is complete (the Tooth Tick Counter has not reached the TICKS value) or after the EAC switched back to Normal mode. Any physical teeth received while the EAC is in High Rate Mode does not alter the immediate EAC state, but it is still detected by the EAC channel logic and can, therefore, alter future EAC behavior (for instance, closing the tooth detection window (see *Section , Angle logic and channel modes*).

At the beginning of High Rate mode operation, the TPR value is preloaded into a temporary register in the Counter Control Logic, used for scanning all the valid values to the end of the current tooth, with its appropriate LAST and MISSCNT attributes. While the EAC is in High Rate mode operation, the effect of microcode update of TPR fields LAST, MISSCNT and TICKS is delayed to the next estimated tooth, after the High Rate mode operation is complete[aj] (see *Section , Special TPR write cases*). This is because the current physical tooth represents the next estimated tooth. If the microcode updates this field after High Rate mode operation is complete, the current physical tooth and estimated tooth are the same, and the effect is immediate. Either in High-Rate mode or not, the value read by microengine is the same written, even if not yet effective, until the EAC resets LAST and/or IPH, or decrements MISSCNT. Typically the microcode service may occur during the High Rate mode on extreme acceleration situation at low RPM. Therefore, the microcode operations are always related to the real physical tooth. From the above it can be seen that the microcode updates of the TICKS field in TPR affect the end time of the current physical tooth. For correct operation, this field should be updated before the Tooth Tick Counter has reached either the old or the new TICKS value.

During High Rate mode operation, TRR is ignored and the Angle Tick Generator uses system clock divided by eight. Therefore, the TRR update by microcode will take effect only after the EAC switches back to Normal mode. If microcode service occurs after the Tooth Tick Counter has been reset, the EAC is already back in Normal mode, and some angle ticks may have been counted at the rate of the previous tooth. In this case the new TRR value will have immediate effect on the angle tick period, and the microcode should take into consideration the delay from the physical tooth to the estimated tooth in calculation of the next tooth period. See *Figure 426* for a detailed diagram of High Rate Mode behavior.

An angle error may be introduced by the duration of the High Rate mode. Also, the scheduler latency may introduce a small accumulated error by using TRR value of the previous estimated tooth at the beginning of the current tooth. After the estimated tooth has advanced, the duration of the High Rate mode operation is the actual delay from the physical tooth edge to the estimated tooth edge. This delay can be obtained by comparing the estimated tooth time with the EAC channel capture register which captured TCR1 on the physical pin transition.

[aj] The effect of microcode writes to fields HOLD and IPH is immediate in High Rate mode.

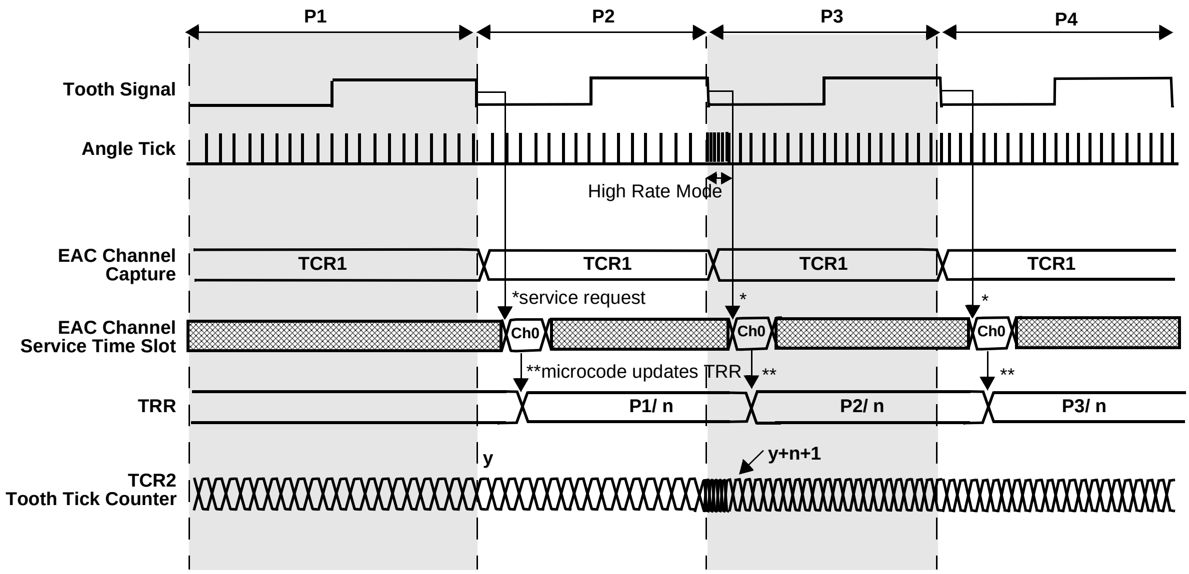


Figure 426. High Rate Mode – Acceleration

Special cases of missing teeth and last tooth

The EAC handles cases of up to three missing teeth and the last tooth in the engine cycle. The following paragraphs describe these functions.

Handling the last tooth

The microcode can set the TCR2 counter to work in engine periods (wrap-around count) or continuous angle measurement.

For periodic operation, during the last engine cycle tooth the EAC microcode has to set the LAST flag in TPR. As a result, when the tooth period is ended, the Counter Control Logic generates a reset command to both the Tooth Tick Counter and TCR2 instead of an advance command. The operation resets the TCR2 based angle count, indicating a new period of the engine cycle. This implementation provides an engine cycle based periodic angle measurement.

Handling missing teeth

The EAC can handle up to three missing teeth in two ways:

- Count the angle ticks relative to the last physical tooth. The microcode should update the TPR TICKS field to the number of angle ticks included in two, three or four teeth, according to the flywheel type (one, two or three missing teeth). EAC hardware works in its regular manner.
- Insert a “dummy” tooth instead of the missing tooth, at the estimated point in time. After the “dummy” tooth, the Angle Tick Counter is incremented as if there was a physical tooth. A “dummy” tooth can be inserted only during Normal or High Rate operation modes. The microcode inserts “dummy” teeth by writing to the MISSCNT field in TPR.

In the first option the missing tooth is not counted on the angle measurement. For example, a flywheel with 59 physical teeth and one missing tooth can be considered as 58 identical teeth numbered (0-57) and tooth number 58 has a double number of angle TICKS. In this case a 720 degrees engine cycle has 118 teeth. TCR2 reflects the real angle, since it counts angle ticks continuously.

In the second option, the missing teeth are counted as “regular” teeth by automatic insertion of “dummy” teeth. The microcode has to write a non-zero value to the MISSCNT field in TPR. This field is a 2-bit down counter which affects the operation of the Counter Control logic.

For example, a flywheel with 59 physical teeth (0-58) and one missing tooth (59) can be considered as 60 teeth numbered (0-59), all having the same number of angle ticks. The microcode has to write “01” to the MISSCNT bits during the period of tooth number 58 to indicate that next tooth (59) is missing.

When the Tooth Tick Counter reaches the TICKS value, TCR2 is incremented as if a physical tooth has been detected. In addition, the MISSCNT value initializes a “dummy tooth counter” which is decremented to indicate the number of left “dummy teeth” which still need to be generated. Because a dummy tooth was counted, EAC does not enter Halt Mode and Tooth Tick Counter continues incrementing in the absence of a physical tooth detection.

In case of extreme acceleration on very low RPM (cold start) there can be a situation that the first physical tooth after one or two missing teeth appears even before the “dummy” tooth is generated. Due to the acceleration the EAC switches to High Rate mode in order to run through all the valid angle values, including the dummy teeth. When the Tooth Tick Counter reaches the TICKS value on High Rate mode, and the “dummy tooth” down counter is not zero, the generated “dummy tooth” advances to the next tooth and decrements the “dummy tooth” counter, but does not switch the EAC back to Normal mode. The last “dummy tooth” decrements the counter to zero, indicating that no more dummy teeth are to be inserted, and the next tooth is an estimated physical tooth. The EAC continues at High Rate mode until the Tooth Tick Counter reaches the TICKS value again, then advances to the next tooth while switching back to Normal mode. When in High Rate mode, the TPR does not reflect the MISSCNT downcounting; see *Section , TPR buffering*, for details.

MISSCNT can be rewritten before it reaches 0, allowing it to count more than three missing teeth, as long as no physical tooth arrives between the first MISSCNT write and the rewrite.

Combining missing teeth and last tooth

The Last Tooth indication takes effect when there are no more missing teeth to be generated, i.e the “dummy tooth” counter value is zero. If, for example, the microcode sets the missing teeth counter to “10” (two missing teeth) and sets the LAST flag, the first and the second dummy teeth will increment TCR2, and the third estimated tooth, which correlates with the physical tooth (the first of the next cycle), will reset TCR2, because LAST was set. This scheme enables the microcode to define one or more missing teeth to be replaced by “dummy tooth” insertion, and the end of the engine cycle in one service request. It is assumed that the two missing teeth must come together in the same engine cycle, and not split between two engine cycles (either the missing teeth are both last in an engine cycle or both not last, but not last in one engine cycle and first in the next).

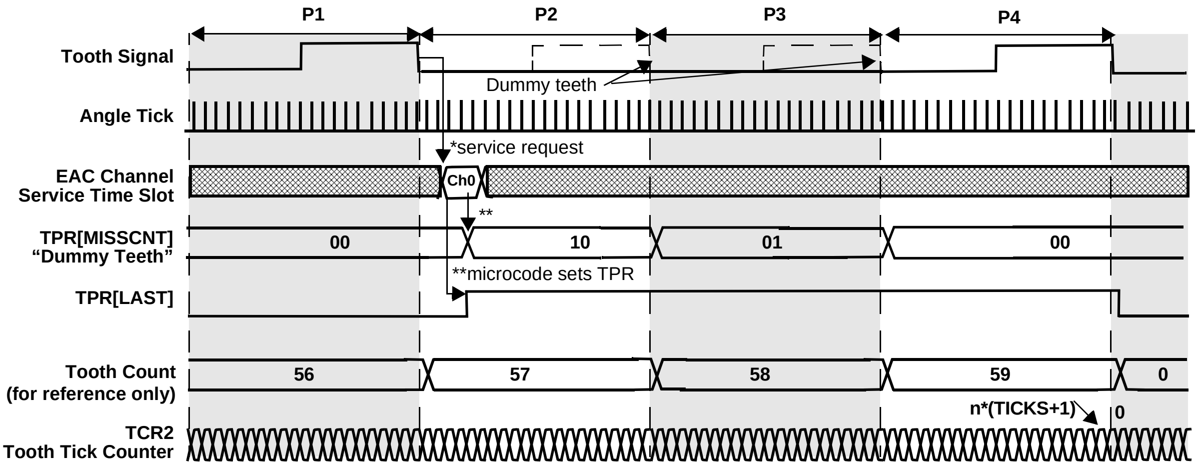


Figure 427. Missing Teeth and Last Tooth Combination

Handling mechanical tooth correction

The EAC can handle tooth edge detection errors caused by flywheel mechanical errors. The eTPU application can hold a vector of tooth mechanical errors with one entry per tooth. This error can be measured in angle ticks which are independent of engine RPM. The TRR can be updated to the fixed period of any tooth, including its mechanical error.

Because TCR2 counts continuously, without being reset, the mechanical correction is transparent. Though the tooth has its own programmed TICKS value, TCR2 simply counts angle ticks, disregarding the boundary between two adjacent teeth.

Handling mis-detected tooth

When a physical tooth signal is missed by the engine sensor, the EAC may get into Halt mode at the end of the estimated tooth period, expecting the physical arrival. In this case, a Match timeout event of EAC channel will call service which detects extreme deceleration. The microcode can assert the IPH bit in TPR, to force the detection of the missed physical tooth. It can also calculate the accumulated angle bus error, and fix the next estimated tooth period, to close the gap.

Handling false tooth detection

Most of the false tooth detection, caused by noises on the engine tooth sensor, can be eliminated by the window blanking filtering, timed by EAC channel match recognitions. The EAC also provides means of fixing false detection of an additional tooth which passed the window filter. When such an event occurs, the EAC switched to High Rate mode (advancing to the next tooth) and when the next physical tooth arrives, an extreme acceleration is detected: the EAC sees the remaining portion of the current tooth period as another tooth period. The microcode can detect the situation when the acceleration in not realistic, or when immediately after the detection of this extreme acceleration, the following tooth indicates extreme deceleration back to the original RPM.

When the microcode detects such a case, the Tooth Counter has been advanced by mistake to the next tooth. The microcode can set the HOLD bit in the TPR, forcing the EAC to freeze and wait for the next physical tooth to close the gap. When the next physical tooth arrives, HOLD is automatically negated and the EAC proceeds from that point to the remaining portion of the tooth period, in the same mode it was when HOLD bit was asserted.

Angle logic and channel modes

The TCRCLK transition detection is qualified by a signal that comes from channel 0 (see *Figure 422*) and depends on the particular mode (PDCM) programmed for that channel. This configures a window for TCRCLK detection for the angle logic which is the same (except on High Rate mode, see *Section , High rate mode (Acceleration)*) used to set the TDLA flag on single transition modes, and TDLB on double transition modes (see signals TSE1,TSE2 in *Figure 405*). The same applies when channels 1 or 2 are used to control EAC (see signals TS1, TS2 in *Figure 405*). As a consequence, the window depends on the channel mode as follows:

- On all modes, the window closes upon a tooth edge detection: TDLA asserted on single transition modes, TDLB asserted on double transition modes.
- On mode m2_st: the window opens on Match A (which enables Transition A) and does not close with Match B. If Match B comes before Match A, it blocks Match A and, hence, Transition A.
- On mode m2_o_st: the window opens on Match A (which enables Transition A) and closes on Match B. Match B is enabled by Match A, so it cannot come before.
- On all other Single Transition modes, the window is “always open”, independently of matches.
- On mode m2_dt: the window opens on Transition A, which is enabled by Match A. The window does not close with Match B, but if it comes before Match A the later gets blocked and, hence, blocks Transitions. Match A is also a condition for the window, so the microcode closes it by clearing MRLA.
- On mode m2_o_dt: the window opens on Transition A, which is enabled by Match A. The window closes on Match B, which is enabled by Match A. Match A is also a condition for the window, so the microcode closes it by clearing MRLA.
- On all other Double Transition modes, the window opens on Transition A.

Restarting angle logic

It is not advisable to toggle the ETPU_TBCR bit AM while GTBE = 1. However, if the Angle Logic must be restarted without interfering with the timebase count running on TCR1, the procedure below must be followed:

1. Write ETPU_TBCR setting AM = 00 and TCR2CTL = 111 at once. That prevents TCR2 from incrementing while the Angle logic is disabled. The Angle Logic state-machine resets to Normal mode and the tick prescaler to the initial count by AM = 00, but not the microengine registers TPR and TRR.
2. Start a thread to reconfigure the EAC. The thread must set the EAC controlling channel (0, 1 or 2) flags in a state, depending on the channel mode, that lets the channel tooth detection window open (see *Section , Angle logic and channel modes*). It can optionally write TCR2 with an angle preset value equivalent to the first tooth expected after restart. The thread must also set TPR bit HOLD = 1. The TPR bit IPH must be 0.
3. After the thread is finished, write ETPU_TBCR setting AM = 01, 10 or 11, and TCR2CTL according to the desired tooth edge selection if AM = 01.

The first tooth detected after this procedure restarts the TCR2 counting, unfreezing the Angle Mode logic into normal mode.

Special TPR write cases

This section describes how simultaneous modification of TPR fields are resolved, and how the effect of TPR writes depend on the EAC mode.

TPR buffering

In High Rate mode (see *Section , High rate mode (Acceleration)*), TPR writes are immediately effective only for bits IPH and HOLD. Writes to all other fields are “buffered” and become effective when EAC leaves High Rate mode. However, if TPR is written a second time right after IPH is asserted in Normal mode, this second write behaves as if EAC is still in Normal mode. Only in the next microcycle (after execution of a NOP, for instance) the TPR writes are buffered, acknowledging High Rate mode.

MISSCNT and LAST can be written any value during High Rate mode, and the value that prevails for the next tooth is the one sampled when EAC goes back to Normal mode (or the value written in Normal or Halt mode thereafter). If MISSCNT and/or LAST are not zero when High Rate mode begins, they are sampled into the internal EAC logic and are effective while High Rate lasts (missing teeth count continues and TCR2 is reset at the end of High Rate if LAST = 1). However, their values in TPR are reset when High Rate mode starts. After that and until the end of High Rate mode, the value read by microcode is the same written. This behavior prevents read-modify-writes to TPR from unwillingly rewriting LAST or MISSCNT.

IPH and LAST

If both IPH and LAST are asserted in the same microinstruction, the EAC acts as if LAST was set first and then IPH right after, so that:

- In Normal mode, it goes to High Rate with LAST = 1;
- In Halt mode, it goes to Normal Mode resetting LAST (and TCR2);
- In High Rate mode, IPH is ignored and LAST becomes effective in the next tooth (physical or inserted) after it goes back to Normal mode.

IPH and TICKS

Because of different results depending on the EAC mode at the time of TPR write, it is not advisable to write 1 to IPH and change TICKS at the same microinstruction. A consistent behavior is obtained if IPH is written fist and TICKS on the second microinstruction after (for instance with a NOP between them), making the new TICKS value valid for the next tooth regardless of the mode.

The mode-dependent behavior is:

- In Normal mode, the new TICKS value becomes valid before EAC goes High Rate due to the IPH;
- In Halt mode, the EAC goes to Normal mode, and new TICKS is valid for the next tooth;
- In High Rate mode, the new TICKS value is effective when EAC leaves High Rate mode, and IPH is ignored;

IPH and MISSCNT

If both IPH and MISSCNT are written non-zero values:

- In Normal mode, at the next microcycle the EAC goes to High Rate mode, the MISSCNT field in TPR goes to 0, and the missing teeth are counted in High Rate mode.
- In Halt mode, the EAC goes to Normal mode for one microcycle and then, yet another microcycle later, goes to High Rate mode, counting the missing teeth. The TPR fields IPH and MISSCNT are zeroed on the transition from Normal to High Rate mode.
- In High Rate mode, IPH is ignored (resetting at the next microcycle) and MISSCNT is buffered (see *Section , TPR buffering*).

IPH and HOLD

If IPH and HOLD are asserted at once, IPH cancels the HOLD and both reset. The EAC is not frozen, regardless of the mode.

LAST and HOLD

If LAST and HOLD are written 1 at once, LAST asserts and EAC is frozen. When a physical tooth is detected or IPH is asserted, the EAC is unfrozen in the same state it was before, and LAST is kept asserted.

22.5.8 Microengine

Each eTPU engine has a microengine that fetches, decodes and executes microinstructions. The Microengine only works when there are service requests to be attended, otherwise it turns to idle state, controlled by Hardware Scheduler (see *Section 22.5.3, Scheduler*).

Microcode is stored in Shared Code Memory (SCM) that is 32-bit wide. Microengine can access SPRAM using a different bus from the one used to accesses code memory, so that code and data can be accessed at the same time (Harvard Architecture).

Some of eTPU functionality can only be made through the microengine, like configuring channels and interrupting host. Microengine gives eTPU a high degree of flexibility, since any desired treatment for channel’s events can be implemented; however, that flexibility comes at the cost of channel service’s latency. Latency is worsened when channels from a same eTPU engine contend for microengine service. In *Figure 428* a block diagram of microengine architecture is shown.

Microengine features are summarized as follows:

- P, DIOB, A, B, C, D, SR, RAR, LINK, CHAN, MACL, MACH, ERTA, ERTB, TCR1, TCR2, TPR, TRR registers are accessible by microcode.
- 24-bit ALU and Post-ALU shifter performs basic arithmetic and logical operations described in *Section , ALU and Post-ALU Shifter*.
- MDU (MAC/Divide Unit) performs integer MAC, multiply and divide operations.
- Fixed Microinstruction Size of 32 bits.
- Fixed-length instruction execution (2 system clocks)
- Static superscalar operation

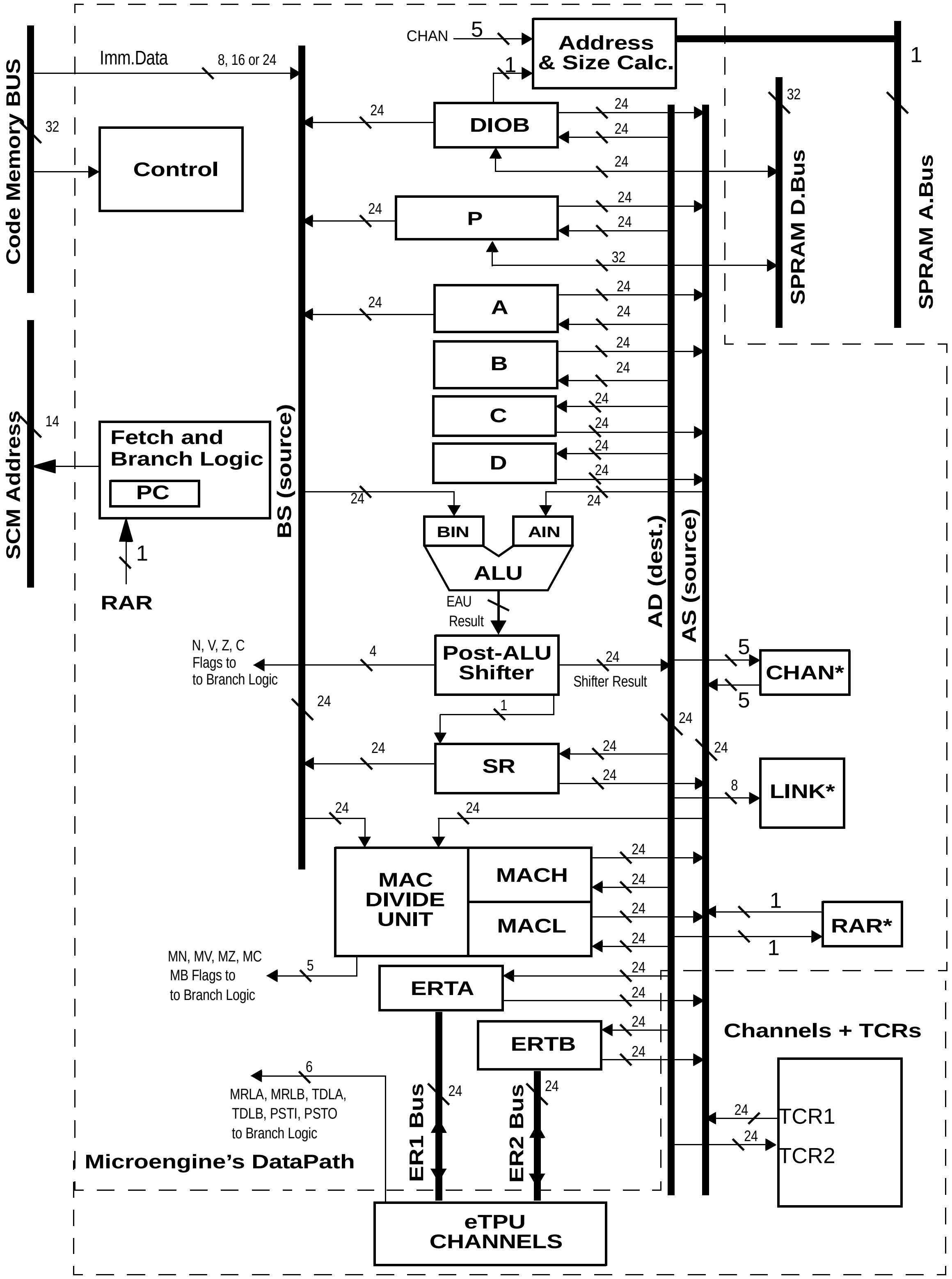


Figure 428. Microengine Block Diagram

Registers

eTPU microengine accesses a total of 18 registers. Fourteen of them are special purpose (registers A, B, C and D are for general use). Special purpose registers except CHAN and LINK can also be used as general use if the operation that use their contents are not performed. Register description is intended to just introduce their functionalities and not to provide detailed explanation of it since it will be described in *Section 22.5.9, Microinstruction set*. Registers less than 24 bits in size are right-justified.

None of the registers have guaranteed reset values. However, some are initialized just before the thread starts (see *Section , Time slot transition*).

P Register

P register is the only one that is 32-bit wide. It can be used as source and destination for arithmetic/logical operation, and as source and destination for SPRAM read/write operations.

For P source/destination possibilities in ALU/MDU microoperations, see *Section , Selecting sources and destination*.

When P is used as SPRAM read/write operations source or destination there are only 3 possibilities of access: all 32 bits, lower 24 bits and upper 8 bits. SPRAM operations are explained in detail in *Section , SPRAM microoperations*.

P is automatically loaded with one parameter before the thread starts (parameter preload). For more information see *Section , Entry point format*, and *Section , Time slot transition*.

Upper 8 bits of P register can be used as application state, since these bits can be tested as branch conditions. P[31:24] is also used in dispatch microoperation (see *Section , Dispatch microoperation*), and bit pairs P[29:28], P[27:26], P[25:24] can be directly copied into Channel flags 1 and 0 using field FLC. Together with Entry Table Condition Encoding, it provides fast state resolution without code execution.

DIOB – Data Input/Output Buffer Register

The DIOB register is 24-bit wide and can be used as source and destination for arithmetic/logical operations as well as SPRAM data source and destination. The DIOB only can be accessed as 24 bits, both in arithmetic/logical and SPRAM read/write operations. When using the DIOB to perform an SPRAM access, only the lower 24 bits of SPRAM will be accessible (SPRAM upper 8 bits always remain unchanged).

The DIOB can also be used as SPRAM addressing register, when the DIOB contents are used as absolute SPRAM address (14-bit wide). In this case the DIOB can also be pre-decremented or post-incremented (see *Section , Indirect addressing mode*).

The DIOB is automatically loaded with one parameter before the thread starts (parameter preload). For more information see *Section , Entry point format*, and *Section , Time slot transition*.

ERTA and ERTB Registers

ERTA/B registers are 24-bit wide and can be used as source or destination in arithmetic/logical operations. ERTA/B are the only source for channel’s match registers write (see *Section , Write Channel Match and UDCM Registers*). ERTA can also be the source for UDCM write.

When a thread starts to be executed, ERTA and ERTB are loaded with a copy of CaptureA and CaptureB registers respectively. ERTA/B can be used to receive a copy of MatchA and MatchB registers. ERTA/B are the only destination of MatchA/B read operation (see *Section , Special T4ABS source operation: Read match registers*).

ERTA and ERTB also receive a copy of CaptureA and CaptureB registers when CHAN register is written (see *Section , CHAN Register*). For more information about Capture and Match registers see *Section , MatchA and MatchB Registers*, and *Section , CaptureA and CaptureB Registers*.

SR – Shift Register

The SR is a 24-bit wide register that can be used as source and destination register for arithmetic/logical operations. The SR can shift right its contents by 1 bit at time and, at the same time, receive in its bit 23 the lost bit of a shift-right operation in post-ALU shifter (*Section , ALU and Post-ALU Shifter*), allowing the SR to be used to perform 48-bit shift right (see *Section , Shift operations*).

MACH and MACL Registers

Both MACH and MACL are 24-bit registers, part of MAC/Divide unit (see *Section , MAC and Divide Unit (MDU)*). They can be used as source and destination in most arithmetic/logic operations. When multiply or divide operations are used (multiply-accumulate included), MACH and MACL have special purpose and some restrictions apply, see *Section , MAC and Divide Unit (MDU)*, for more information.

LINK Register

Link Register is an 8-bit wide register and can be used only as destination in arithmetic operations. LINK is a write-only command register, which precludes its use as a source register for ALU operations. When LINK register is written, it issues a service request for the channel number and eTPU engine equal to the number written in LINK register (see *Section 22.5.1, Functions and threads*, and *Section , Channel Link*, for information about Link Service Request).

RAR – Report Address Register

The RAR is a 14-bit register and can be used as source and destination in arithmetic operations. The RAR also receives the contents of PC register when a subroutine call is executed. The contents of the RAR are loaded into PC when a return from subroutine is executed. The RAR is loaded with value 0x3FFF during TST. For more information about subroutine call and return see *Section , Branch operations*, and *Section , Return from subroutine*, respectively.

CHAN Register

CHAN is a 5-bit register that can be used as source and destination in arithmetic operations. The contents of CHAN register affects the execution of many channel-related microinstructions, because its number indicates the **selected channel**. CHAN register must not be used to store temporary values in arithmetic operations. For more details, refer to *Section , Channel Selection Register – CHAN*.

Counter Registers: TCR1, TCR2, TPR and TRR

All these registers are 24-bit wide except TPR, which is a 16-bit register. They can be read or written in arithmetic/logical operations, and have special-purpose uses for time base and angle mode operations. For more information about those registers see *Section 22.5.6, Time Bases*, and *Section 22.5.7, EAC – eTPU angle counter*.

General Purpose Registers: A, B, C and D

A, B, C and D are 24-bit general purpose registers, which can be used to store intermediate values and do not have other specific uses with any eTPU feature.

ALU and Post-ALU Shifter

The ALU executes 24-bit arithmetic and logical operations. ALU's output goes directly to a 1-bit shifter, called post-ALU shifter, so it is possible, for example, to add and shift using only one microinstruction.

In some microinstruction formats, it is not possible to specify the operation executed by ALU. In these cases ALU will always perform addition.

In formats which have the field ALUOP for ALU operation selection, all of them can be performed, including add/subtract using C (carry) flag as ALU's carry-in, bitwise AND/OR/NOT/XOR, and shift/rotate of 2, 4, 8 and 16 bits. See *Section , ALU/MDU Operation Selection*.

Subtraction, inversion, increment and decrement can be performed by combinations of source inversion and setting ALU's carry-in to 1.

ALU always performs 24-bit operations on its inputs, called **A-source** and **B-source**, and outputs a 24-bit result. 8-, and 16-bit inputs are zero padded to 24 bits. Likewise, ALU 24-bit output is always truncated to the destination register size.

ALU Flags

Four flags—Carry, Negative, Overflow, Zero—described below, are related to ALU and post-ALU shift operations. Operation size and shifting affect flags generation logic. Operation size determines the result boundary to be used for flags generation. Operation size is determined by size of sources and destination (see *Section , Flags sampling control*). For more information about flag generation, see *Section , Flags sampling control*. ALU flags can be used as branch condition (see *Section , Conditional/Unconditional branch*) or conditional ALU/MDU operation (see *Section , Conditional ALU/MDU operation execution*).

Field CCS/CCSV in microinstructions can force **no update** of all flags. Not all flags are updated in all ALU operations: Overflow is updated only on addition and absolute value operations, Carry flag is updated in most ALU operations, and only Zero and Negative are updated in all ALU operations.

ALU flags are never updated when microinstruction starts an MDU operation, regardless of CCS/CCSV, but are updated normally afterwards, on ALU operations that are executed in parallel with an ongoing MDU operation (MDU has its own flags).

Note: Operation size can be smaller than destination register. For example: 0xFFFF + 0x0001 (both 16-bit sources) stores 0x10000 in a 24-bit register and sets Zero and Carry flags because operation size is 16 bits.

Carry Flag (C)

In an unsigned addition without shifting, Carry Flag is the ALU carry from bit 7 to 8, 15 to 16, or 23 to 24 on 8, 16 and 24-bit operation sizes respectively. In an unsigned subtraction without shifting, Carry Flag represent the sign of ALU's result considering operation size (Carry Flag equal to 0 means a negative result).

Carry Flag definition is operation-dependent. The Carry flag in add/subtraction with Post-ALU shift is defined in *Table 408*. Find the definitions for other operations in the following sections.

Negative flag (N)

Negative flag indicates the sign of result based on the operation size, regardless of the operation performed, as shown in *Table 404*.

Table 404. Negative (N) flag behavior

Operation size	Value
8 bits	N = result[7]
16 bits	N = result[15]
24 bits	N = result[23]

Note: The N flag may not reflect the sign of the value actually written into the destination register, if it does not have the same size of the operation (see Section , Flags sampling control). This is always the case for registers RAR (14 bits) and CHAN (5 bits).

Overflow (V)

Overflow is updated only on addition (with or without carry) and absolute value operations. In signed operations, overflow flag indicates that the result of arithmetic operation (add or subtraction) can not be represented by a word of the size of the operation. Overflow flag behavior for addition is defined in *Table 405*. Overflow flag for Absolute operation is explained in *Section , Absolute value operation*. V Flag is calculated using ALU adder output (that is, it is not affected by 1-bit shift/rotate operations).

Table 405. Overflow flag on addition(1) – V

Operation size	Value(2)
8 bits	(AS[7] & BS[7] & !alu_adder_output[7]) | (!AS[7] & !BS[7] & alu_adder_output[7])
16 bits	(AS[15] & BS[15] &!alu_adder_output[15]) | (!AS[15] & !BS[15] & alu_adder_output[15])
24 bits	(AS[23] & BS[23] &!alu_adder_output[23]) | (!AS[23] & !BS[23] & alu_adder_output[23])

1. For V-flag definition on the absolute operation, see *Section , Absolute value operation*.
2. BS is taken after any inversion by the BINV field, but not added to the carry bit (CIN field)

Zero Flag (Z)

Zero flag equal to 1 indicates that the result from the ALU, limited to the operation size, is zero, regardless of the operation performed, whether the result is written, or where it is written. It depends on the operation size, as shown in *Table 406*.

Table 406. Zero Flag – Z

Operation size	Value
8 bits	Z = (result[7:0] == 0x00)

Table 406. Zero Flag – Z

Operation size	Value
16 bits	Z = (result[15:0] == 0x0000)
24 bits	Z = (result[23:0] == 0x000000)

ALU ADD Operation with and without shifting

ADD operation is selected by ALUOP or ALUOPI fields and when none of them is available in a microinstruction format[ak]. Optionally, result can be shifted or rotated right by 1 bit, which is selected by SHF, ALUOP or ALUOPI fields. See *Section 22.5.9, Microinstruction set*, for more details. *Table 407* describes how CIN and BINV fields change ADD operation behavior.

Table 407. Types of ADD operations

BINV	CIN	Operation (adder output)
1	1	AS + BS
1	0	AS + BS + 1
0	0	AS - BS
0	1	AS - BS - 1

ALU adder output can be 1-bit shifted or 1-bit rotated right as follows:

Shift right:

```
if BINV==1
  result[23:0] = adder_output[24:1]
else
  result[23:0] = adder_output[24:1] xor 0x800000
endif
```

Shift left:

```
result[23:1] = adder_output[22:0]
result[0]    = 0
```

Rotate right:

```
case(opsize/CCSV)
8-bit:
  result[6:0]    = adder_output[7:1]
  result[7]      = adder_output[0]
  result[23:8]   = adder_output[23:8]
16-bit:
  result[14:0]   = adder_output[15:1]
  result[15]     = adder_output[0]
  result[23:16]  = adder_output[23:16]
24-bit:
  result[22:0]   = adder_output[23:1]
  result[23]     = adder_output[0]
```

[ak] ALU operations only occur on formats where a destination field is found (T2ABD/T2D).

Note: Only for the Post-ALU rotate right, the operation size is determined by the field CCSV (see Section , Flags sampling control). For example: if CCSV = 00, T4ABS = P (24-bits), T4BBS = A (24 bits), T2ABD = B (24 bits), and ALUOP = “Add ROR”, then B gets A+ P with bits 7:0 rotated, even though the operation size is 24 bits.

Table 408 describes Carry flag behavior.

Table 408. Carry flag update on ADD operation

BINV[1]	Operation size	Shift/rotate	Value
1	8 bits	none	adder carry from bit 7 to bit 8
1	16 bits	none	adder carry from bit 15 to bit 16
1	24 bits	none	alu_adder_output[24]
0	8 bits	none	!adder carry from bit 7 to bit 8
0	16 bits	none	!adder carry from bit 15 to bit 16
0	24 bits	none	!alu_adder_output[24]
0 or 1	8 bits	shift left	alu_adder_output[7]
0 or 1	16 bits	shift left	alu_adder_output[15]
0 or 1	24 bits	shift left	alu_adder_output[23]
0 or 1	x	shift right	alu_adder_output[0]
1	8 bits	rotate right	adder carry from bit 7 to bit 8
1	16 bits	rotate right	adder carry from bit 15 to bit 16
1	24 bits	rotate right	alu_adder_output[24]
0	8 bits	rotate right	!adder carry from bit 7 to bit 8
0	16 bits	rotate right	!adder carry from bit 15 to bit 16
0	24 bits	rotate right	!alu_adder_output[24]

Except on max-constant generation (see *Section , Max constant generation with T4BBS = 111*)

[1] BINV has no effect on carry-out when used to code max constant (see *Section , Max constant generation with T4BBS = 111*).

Flags N and Z on shift are updated according to the result after shift. Flag V with shift is updated according to the ADD operation only, the same way as without shift.

ADC operation

ADC operation is selected by the ALUOP field. CIN field is ignored when this operation is selected. *Table 409* describes how BINV change ADC operation behavior.

Table 409. Types of ADC operations

BINV	CIN	Operation
1	x	AS + BS + C flag
0	x	AS - BS - C flag

Flags behave exactly the same way as for ADD operation without shift/rotate.

Bitwise Operations

Bitwise AND, OR and XOR are selected by ALUOP field. On these operations CIN field is ignored and BINV field inverts (bitwise NOT) BS. C and V Flags are never updated on these operations. *Table 410* Describes AND, OR and XOR bitwise operations.

Table 410. Types of Bitwise Operations

ALUOP	BINV	Operation
10000	1	AS | BS
10000	0	AS | (~BS)
10001	1	AS ^ BS
10001	0	AS ^ (~BS)
10010	1	AS & BS
10010	0	AS & (~BS)

Set Bit / Clear bit operations

These operations set or clear the AS bit determined by BS[4:0]. If the bit number resolves to a value greater than 23, no bit is set or cleared (i.e., result is equal to AS). On these operations CIN field is ignored and BINV field inverts (bitwise NOT) BS. C and V flags are never updated for set/clear bit operations. These operations **override** B-Source size to 8 bits.

set bit (BINV = 1):

```
result = AS | (1 << BS[4:0])
```

clear bit (BINV = 1):

```
result = AS & ~(1 << BS[4:0])
```

set bit (BINV = 0):

```
result = AS | (1 << (31 - BS[4:0]))
```

clear bit (BINV = 0):

```
result = AS & ~(1 << (31 - BS[4:0]))
```

Exchange bit

Exchange the AS bit determined by BS[4:0] with C flag. If the bit number resolves to a value greater than 23, no exchange is performed (i.e., result is equal to AS and C flag is not updated). This operation **overrides** BS size to 8 bits. On this operation, CIN field is ignored and BINV field inverts (bitwise NOT) BS. V flag is never updated on exchange bit operation. C flag is always updated, regardless of CCSV, unless BS[4:0] > 23.

Exchange Bit (BINV = 1):

```
if BS[4:0] <= 23
begin
  temp_C_flag = AS[BS[4:0]]

  if C_flag == 1
    result = AS | (1 << BS[4:0])
```

```
  else
    result = AS & ~(1 << BS[4:0])

  C_flag = temp_C_flag
end
```

Exchange Bit (BINV = 0):

```
if (31 - BS[4:0]) <= 23
begin
  temp_C_flag = AS[31 - BS[4:0]]
  if C_flag == 1
    result = AS | (1 << (31 - BS[4:0]))
  else
    result = AS & ~(1 << (31 - BS[4:0]))
  C_flag = temp_C_flag
end
```

Multibit shift/rotate operations

These operations shift or rotate AS by 2, 4, 8 or 16 bits. Size of shift/rotate is determined by BS[1:0]. *Table 411* describes the number of shifted/rotated bits depending on BS[1:0] value.

Table 411. Number of shifted/rotated bits for each BS[1:0] value

BS[1:0]	Bits shifted/rotated
0	2
1	4
2	8
3	16

Shift right is a logical operation (i.e., zeros are inserted on left). Multibit shift and rotate operations **overrides** BS size to 8 bits. The shifts and rotate operate on 24 bits, independently of the operation size.

V flag is never updated for multibit shift or rotate operations. Carry flag behavior is described on *Table 412*. CIN is ignored in these operations, but BINV is effective.

Table 412. Carry flag value on multibit shift/rotate operations

ALUOP	BS[1:0]	C flag value(1)
11001 (shift left)	0	AS[22]
11001 (shift left)	1	AS[20]
11001 (shift left)	2	AS[16]
11001 (shift left)	3	AS[8]
11010 (shift right)	0	AS[1]
11010 (shift right)	1	AS[3]
11010 (shift right)	2	AS[7]
11010 (shift right)	3	AS[15]
11011 (rotate right)	0	AS[2]

Table 412. Carry flag value on multibit shift/rotate operations

ALUOP	BS[1:0]	C flag value[1]
11011 (rotate right)	1	AS[4]
11011 (rotate right)	2	AS[8]
11011 (rotate right)	3	AS[16]

1. CCS/CCSV can disable flag update on multibit shift/rotate, but the specified flag size in CCSV is ignored for the C flag.

Absolute value operation

Absolute Value operation is selected by ALUOP field. On this operation, AS is interpreted as a signed number and its absolute value is the result. V and N flags are updated with the result signal determined by the operation size. AS bit 23 after size override and sign extension (if any, see *Section , A-Source size override*), regardless of A-source register size, is used to check the operand signal and is copied to C-flag. Note that if AS is 8-bit or 16-bit, its sign is taken into account and copied to C only if sign-extension is performed. *Table 413* summarizes flag updating for Absolute Value operation.

This operation is independent of B-source. Instruction fields T4BBS, BINV and CINV are ignored in this operation. The Absolute Value operation size is the minor between A-source size and Destination size.

Table 413. ALU Flags in Absolute Value operation

Operation size	V, N[1]	C	Z
8	alu_output[[7]	AS[23]	alu_output[7:0] == 0
16	alu_output[15]		alu_output[15:0] == 0
23	alu_output[23]		alu_output[23:0] == 0

1. V, N can be 1 on 8- and 16-bit Absolute Value, because the operand sign is always taken from bit 23. V, N can also be 1 in 23-bit Absolute Value (or 8-bit and 16-bit with sign extension), if the operand is 0x800000 (0x80, 0x8000).

MAC and Divide Unit (MDU)

MDU is an autonomous resource in the microengine which can carry out sequential multiply, multiply-accumulate, fractional multiplication and divide operations, selected through the microinstruction fields ALUOP or ALUOPI. The unit supports signed and unsigned multiply and fractional multiplication of any combination of 8, 16 or 24-bit operands[al], and also signed and unsigned 24-bit multiply-accumulate. Divide operation is unsigned, and both operands are always 24-bit wide.

Depending on the size of operands and the type of operation, MDU can take more than one microcycle to execute the operation, but microengine continues to execute microinstructions in parallel. When the microcode issues an END command, any MDU executing operation terminate immediately and is left incomplete. When selecting an operation that uses MDU, the result is always placed in MACH and MACL registers, and the register selected as destination does not have its value changed (*Section , Selecting sources and destination*).

[al] There is no distinct selection of 24-bit fractional multiplication, for it works exactly as a 24-bit ordinary multiplication.

During calculations, MACH and MACL holds temporary values and should not be written, otherwise the result is unpredictable. One must not start an MDU operation while MDU is already busy: the result is unpredictable for both the ongoing operation and the started one.

MDU Operations update its own set of five flags, described in *Section , MDU Flags*. MDU operations never update C, N, V and Z flags. CIN and BINV microinstruction fields affect MDU operations according to *Table 414*.

Table 414. CIN and BINV with MDU operations

B-source operand	BINV	CIN	Operation performed
signed	1	1	AS mdu_op BS
	0	0	AS mdu_op (-BS)
	1	0	reserved
	0	1	reserved
unsigned[1]	1	1	AS mdu_op BS
	1	0	AS mdu_op (BS+1)
	0	x	reserved

1. Includes the B-source (unsigned) in fmults (signed) operations.

Multiply and Multiply-Accumulate Operation Length

MDU needs two sources, A source and B source, to perform an operation. The time needed to perform a multiply or multiply-accumulate is:

- On 24-bit x 8 bit multiplies: 2 microcycles (one start-MDU plus one execution microcycle)
- On 24-bit x 16 bit multiplies: 3 microcycles (one start-MDU plus two execution microcycles)
- On 24 bit x 24 bit multiplies/macs: 4 microcycles (one start-MDU plus three execution microcycles)

An internal pipeline in MDU allows multiply-accumulate (or even non-fractional multiply) operations to start one microinstruction before a multiplication or multiply-accumulate (signed or unsigned) has been completed (e.g., one can start one multiply or multiply-accumulate once every three microinstructions). However, by doing that it is not possible to read the result in MACH and MACL (although the MDU flags can be tested), so this is intended to be used in a multiply-accumulate sequence. It is also allowed to mix different sizes in multiply/mac sequences.

Multiply-accumulate operations are similar to multiply operations, except that the contents of MACH and MACL registers are added to the multiplication result.

When multiply or multiply accumulate operations finish, MACL and MACH hold the least and the most significant 24-bit words, respectively.

Divide operation length

The divide operation is always unsigned. The division completes in 13 microcycles, meaning that after the start divide microinstruction, one has to wait for 12 microcycles and then read the result and the remainder in MACH and MACL registers. During the 12 execution microcycles, microengine can execute microinstructions unrelated to the MDU.

Signed multiplication (mults)

MDU signed multiplication is defined as follows:

```
(signed) MACH,MACL = (signed) AS * (signed) BS
```

MC and MV flags are reset. MZ is set if result is 0, resets otherwise. MN is set if result is negative.

Unsigned multiplication (multu)

MDU unsigned multiplication is defined as follows:

```
(unsigned) MACH|MACL = (unsigned) AS * (unsigned) BS
```

MC and MV flags are reset. MZ is set if result is 0, resets otherwise. MN is a copy of the most significant bit of result.

Signed multiply-accumulate (macs)

MDU signed multiply-accumulate is defined as follows:

```
(signed/unsigned) {MACH,MACL} += (signed) AS * (signed) BS
```

MC is not altered.

MV is set if result can not be represented by a 48-bit signed number. MACS never resets MV flag: it is left as is if no overflow occurs, or set it otherwise. This allows checking the overflow flag only once at the end of a series of multiply-accumulate operations in a scalar product calculation.

```
if (({MACH,MACL} += AS * BS < $-2^{47}$) || ({MACH,MACL} += AS * BS > $2^{47}$ - 1))
  MV = 1
```

MZ is set if result is 0, resets otherwise. MN is a copy of the most significant bit of result.

Note that only 24-bit multiply-accumulate is available.

Unsigned multiply-accumulate (macu)

MDU Unsigned Multiply-Accumulate is defined as follows:

```
(signed/unsigned) {MACH,MACL} += (unsigned) AS * (unsigned) BS
```

MC is set if result can not be represented by a 48-bit unsigned non-negative number. MACU never resets MC flag: MC flag is left as is if no carry occurs, or set otherwise. This allows checking the carry flag only once at the end of a series of multiply-accumulate operations in a scalar product calculation.

```
if (({MACH,MACL} += AS * BS < 0) || ({MACH,MACL} += AS * BS > $2^{48}$ - 1))
  MC = 1
```

MV is not altered.

MZ is set if result is 0, resets otherwise. MN is a copy of the most significant bit of result.

Note that only 24-bit multiply-accumulate is available.

Signed fractional multiplication (fmults)

MDU Signed Fractional Multiplication takes the B-Source as an **unsigned** 8- or 16-bit fraction between 0 and $(2^8 - 1)/2^8$ (inclusive) for the 8-bit operation, or between 0 and $(2^{16}-1)/2^{16}$ (inclusive) for the 16-bit operation. Only A-Source is taken as a signed number. The value of B-Source is considered the unsigned numerator of a fraction with denominator 2^8 or 2^{16} for the 8- and 16-bit operations, respectively.

The integer part of the result is stored in MACH, and the fractional part in MACL. The result is signed, so that the concatenation of MACH and MACL form a 48-bit fixed point number with a 24-bit mantissa, both for 8- and 16-bit operations. To calculate the unsigned numerator of the fractional part (with denominator 2^{24}) of the result, one must take the absolute value of MACL considering the signal of the result (not MACL alone), i.e.: if flag MN = 1, invert MACL and add 1.

MDU flags are updated in the same way as in the Signed Multiplication.

Unsigned Fractional Multiplication (fmultu)

MDU Unsigned Fractional Multiplication takes both A-Source and B-Source as unsigned operands. B-Source is taken as an 8- or 16-bit fraction between 0 and $(2^8 - 1)/2^8$ (inclusive) for the 8-bit operation, or between 0 and $(2^{16}-1)/2^{16}$ (inclusive) for the 16-bit operation. The value of B-Source is considered the numerator of a fraction with denominator 2^8 or 2^{16} for the 8 and 16-bit operations, respectively.

The integer part of the result is stored in MACH, and the fractional part in MACL. The fractional part in MACL is the numerator of a fraction with denominator 2^{24}. The concatenation of MACH and MACL form a 48-bit fixed point number with a 24-bit mantissa, both for 8 and 16-bit operations.

MDU flags are updated in the same way as in the Unsigned Multiplication.

Unsigned Divide (div)

At the end of a divide operation MACL holds the result of the division, taking A-source as numerator and B-source as denominator, while MACH holds the remainder. If a divide by 0 is executed, MACL holds the maximum unsigned number (0xFFFFFF) as result and flag MV is set to indicate division by 0 (otherwise reset). The contents of MACH become indetermined.

MC flag is always reset.

MZ flag is set if MACL equals 0, and reset otherwise.

MN receives a copy of MACH bit 23 (msb from the remainder).

Note that signed division is not available.

MDU Flags

MDU has its own flags to indicate the result and status of an MDU operation. They are: MC, MZ, MV, MN and MB. All MDU flags are updated with the final result at the end of the operation, and do not change until the next operation finishes. Therefore it is possible to start a new MDU operation and test the flags of the previous one in parallel, except for mult/mac with 8-bit operand (takes only 1 microcycle).

MDU Negative Flag – MN

MN flag is always a copy of MACH bit 23 at the end of the operation, either in signed or unsigned ones. Note that MACH holds the rest of a division operation, which is always unsigned.

MDU Carry Flag – MC

MDU carry flag indicates if the result cannot be represented by a 48-bit number, in Signed and Unsigned Multiply Accumulates. It is reset in the other operations.

MDU Zero Flag – MZ

In multiply and multiply-accumulate operations, MDU zero flag is asserted if MACH and MACL are equal to zero at the end of an operation. In divide operations, zero flag is asserted if MACL (result) is equal to 0.

MDU Overflow Flag – MV

In multiply operations, MV flag is negated and keeps negated in the end, because the result of a multiplication can always fit in a 48-bit result (MACH and MACL concatenated). In a multiply-accumulate operation, MV is asserted if the result size is wider than 48 bits. MV flag work in both signed and unsigned operations.

In divide operations it is only asserted if a divide-by-zero operation was executed.

MDU Busy Flag (MB)

MB tests as true at the next microinstruction after the MDU start operation, and as false at the last microcycle of any MDU operation execution.

Branch Conditions

Microengine allows conditional branch. There are five sets of flags that can be tested in a conditional branch: ALU flags, MDU flags, P flags, Channel flags, and Semaphore flag (flag SMLCK).

When a thread starts to be executed, the values in MDU and ALU flags are not initialized. ALU flags are described in *Section , ALU Flags*. MDU flags are described in *Section , MDU Flags*. MDU and ALU flags are updated during execution of microinstructions.

P flags are actually the upper byte of P register, which optionally can work as user defined flags (see *Section , P Register*).

Channel flags Flags0, Flag1, MRLA, MRLB, TDLA, TDLB, PSS, PSTI and PSTO are obtained from the selected channel (value in CHAN register), while channel flags, LSR, FM[0] and FM[1] are selected by the serviced channel, regardless of the CHAN value(am).

Flags TDLA/B, MRLA/B, LSR, FM[1:0] and PSS, are sampled at the beginning of a thread. Flag PSS does not change during its execution while CHAN register is not written. When a write in CHAN register is performed, all flags except LSR and FM[1:0] are updated according to the channel specified by CHAN value. Flags MRLA/B and TDLA/B are reset when their respective latches in channel are cleared by microcode.

Table 415. Channel flags as branch condition

Flag	Description	Service or selected channel
Flag0	State Resolution flag	Reflects the selected channel (CHAN)
Flag1	State Resolution flag	Reflects the selected channel (CHAN)

(am) Serviced channel does not change during execution of a thread, and it is the channel that requested a service (initial value of CHAN register when a thread starts).

Table 415. Channel flags as branch condition (continued)

Flag	Description	Service or selected channel
MRLA	Match A Recognition Latch	These flags reflect the selected channel (CHAN) see *Section , MRLA/B – Match Recognition Latches*, and *Section , TDLA/B – Transition Detect Latches*, for more information.
MRLB	Match B Recognition Latch	
TDLA	Transition A Detection Latch	
TDLB	Transition B Detection Latch	
LSR	Link Service Request	Reflects the serviced channel.
PSS	Sampled Input Pin State	Reflects the selected channel (CHAN). Does not change if CHAN is not changed (see *Section , Pin Control Registers*).
PSTI	Current Input Pin State.	Reflects the selected channel (CHAN). Changes any time.
PSTO	Current Output Pin State	Reflects the selected channel (CHAN). Changes any time.
FM[1:0]	Function Mode Bits	reflects the Function Mode for serviced channel (*Section , ETPU_CxSCR – eTPU Channel x Status Control Register*)

Semaphore condition SMLCK always indicates if a semaphore is locked for the engine, resolving as false before any lock attempt. For each trial, the SMLCK flag is updated. The SMLCK value set in one thread is not meaningful to the other. After a free, the SMLCK condition tests as false until a new lock attempt on the same thread.

Branch conditions are selected through instruction fields BCC and BCF (see *Section , Conditional/Unconditional branch*).

22.5.9 Microinstruction set

Each microinstruction can execute up to three microoperations in parallel. Microinstructions are grouped into formats, and there are four types of **microoperations**:

- ALU/MDU Operations
- SPRAM Operations
- Channel Configuration/Control Operations
- Flow Control Operations

Each microinstruction format is defined by a set of microinstruction fields, which determine the operations, each belonging to one of the groups above (there may be several in one group). Complete microinstruction formats are shown in *Section , Microinstruction formats*.

Parallelism conflicts may arise when two operations are executed in the same microinstruction. These situations are explained in *Section , Microinstruction parallelism issues*.

SPRAM microoperations

The access to SPRAM is made by providing an address and a register to perform a data transfer, except semaphore operations, which are also classified in the SPRAM group. Only

P and DIOB registers can exchange data with SPRAM. Microengine always addresses SPRAM in 32-bit boundaries, for 8, 24, or 32-bit wide data.

Direction is determined by the field RW in all addressing modes: RW = 0 selects read and RW = 1 selects write.

SPRAM Addressing Modes

The eTPU has four addressing modes:

- Absolute
- Selected Channel Relative
- Indirect
- Engine Relative

The addressing modes Absolute and Selected Channel Relative use immediate bits to form the physical address of SPRAM, which is identified in microinstruction as a field called AID. AID field can be 3, 7, or 8-bit wide depending on the addressing mode.

Absolute addressing mode

In Absolute addressing mode, the address range is 256 parameters, addressed by field AID, which in this mode is 8-bit wide. These parameters are located in SPRAM addresses from 0 to 255.

```
physical_address = AID[7:0]
```

Selected channel relative addressing mode

In Selected Channel Relative addressing mode, only the first 8 (with 3-bit AID) or 128 (with 7-bit AID) parameters of the selected channel are accessible, depending on the microinstruction format. Physical address is calculated using the channel parameter base address that is specified in field CPBA of ETPU_CxCR (see *Section , ETPU_CxCR – eTPU Channel x Configuration Register*). AID field is added to channel parameter base address to compose the physical address. The equation is:

```
physical_address = selected_channel_parameter_base_address + AID[6:0], or

physical_address = selected_channel_parameter_base_address + AID[2:0]
```

Indirect addressing mode

In Indirect Addressing mode the physical address is taken from DIOB register. Only DIOB bits 13 to 2 are relevant. Since the SPRAM word address is shifted two bits up in DIOB, its contents hold the same parameter address value used by Host. The equation is:

```
physical_address = DIOB[13:2], or

physical_address = (truncated) DIOB / 4
```

Indirect addressing mode can have post-increment or pre-decrement on DIOB, allowing stack operations. See *Section , DIOB stack operation*, for more information.

Engine relative addressing mode

In Engine Relative Addressing mode the physical address is the concatenation of the ETPU_ECR field ERBA (see *Section , ETPU_ECR – eTPU Engine Configuration Register*) with the 7-bit AID instruction field. This allows the same function microcode, when running on distinct engines, to access different address spaces, global to the engine only.

SPRam source/destination registers

When performing an SPRAM operation, only DIOB or P can be used as data source or destination. P is 32-bit wide, and DIOB is 24-bit wide. Microinstruction field P/D (1 bit) is used to choose between P and DIOB as data the source or destination. When the P/D field is not available in microinstructions that support SPRAM access, the source/destination is P.

Table 416. SPRAM source/destination register selection

P/D	Meaning
0	P access
1	DIOB access

SPRAM operation size

When using DIOB register to perform SPRAM data transfers, the operation size is always 24-bit wide (lower 24 bits of SPRAM). When using P register, the operation size can be 8, 24 or 32-bit wide, which is controlled by microcode RSIZ field (2 bits). RSIZ meaning is shown in *Table 417*.

Table 417. SPRAM P access size

RSIZ	P access	DIOB access
00	full 32-bit access (i.e. P[31:0]=SPRAM[addr] [31:0])	RESERVED
01	only upper 8 bits are transferred (i.e. P[31:24] = SPRAM[addr] [31:24])	RESERVED
10	only lower 24 bits are transferred (i.e. P[23:0] = SPRAM[addr] [23:0])	DIOB = SPRAM[addr] [23:0]
11	RESERVED	RESERVED

RSIZ is not available in all microinstructions that support SPRAM access. In microinstructions where RSIZ field is not available, SPRAM access will be 24 bits by default.

When performing a Zero SPRAM write operation (see *Section , Zero SPRAM operation*), RSIZ defines the size of operation regardless of the P/D field (*Section , SPRam source/destination registers*).

SPRAM access direction

RW field defines the direction of the access in the SPRAM. The access direction is summarized in *Table 418*.

Table 418. SPRAM access direction

R/W	Meaning
0	read SPRAM parameter into P or DIOB registers
1	write SPRAM parameter from P or DIOB registers

Zero SPRAM operation

Zero SPRAM operation is controlled by microcode field ZRO (1 bit). When ZRO field is 0, the data portion written in SPRAM or in P/DIOB (SPRAM read) registers will always be 0x0. When performing a Zero SPRAM write operation, the RSIZ is relevant regardless of the P/D field (usually RSIZ is meaningful only for P/D = 0), which means that zero SPRAM write operation can be performed with 32, 24 or 8 bits according to SPRAM operation size. These conditions are summarized in *Table 419*.

Table 419. Zero SPRAM operation

ZRO	RW	P/D	Meaning
0	0	0	Clear P register. Size is determined by RSIZ field. See *Section , SPRAM operation size*.
0	0	1	Clear DIOB (all 24 bits), independently of RSIZ
0	1	x	Clear SPRAM parameter. Size is determined by RSIZ field. See *Section , SPRAM operation size*.
1	RW	P/D	Regular SPRAM operation

Note: When field STC is present, STC = 11 also disables Zero SPRAM operation (see Table 420). The conflicts with DIOB operations (see Section , DIOB stack operation) and ALU operations are resolved like a normal SPRAM operation (see Table 461).

DIOB stack operation

SPRAM Indirect Addressing Mode (see *Section , Indirect addressing mode*) is used if STC field (2 bits) exists in the microinstruction, controlling automatic increment/decrement of DIOB register, as shown in *Table 420*, thus allowing stack operations. DIOB is incremented and decremented in word addresses, only from bits 15 downto 2, i.e.: the bits 23 to 16 and 1 to 0 are left untouched by STC pre-decrement and post-increment.

Table 420. DIOB Post-Increment / Pre-Decrement – STC

STC	Meaning
00	Post-Increment of DIOB
01	Pre-Decrement of DIOB
10	No Increment/Decrement (normal access)
11	No SPRAM Access(1)

1. Also disables Zero SPRAM operation

Semaphore operations

Semaphore lock and free operations are available through eTPU microcode. For more information about semaphores see *Section , Hardware Semaphores*. Two microinstruction fields control semaphore operations: FL (1 bit) and SMPR (2 bits). Serviced channel sees four semaphores, selected by field SMPR.

Table 421. Semaphore operations fields

Field	Meaning
FL	0 = free semaphore, 1 = lock semaphore
SMPR	semaphore number selector

When freeing a semaphore, the field SMPR has no meaning. This is because only one semaphore can be locked at a time by each engine, so when freeing a semaphore it is not necessary to specify its number.

Note: If microcode tries to lock a semaphore already locked for the same engine, the semaphore continues locked for the engine and the SMLCK branch condition resolves as true.

ALU/MDU operations

ALU/MDU microoperations mostly comprises two sources, one destination and one operation. The operation is generally selected through fields ALUOP, ALUOPI or SHF. In formats where there is no operation selection field (ALUOP, ALUOPI or SHF), the operation performed is always addition; however, it is possible to perform subtraction, increment or decrement using fields BINV (see *Section , B-Source inversion*) and CIN (see *Section , Carry-in Control*).

Source and destination register set selection

Microcode field T4ABS allows selection of a source from either one of two register sets, shown in *Table 425*. The same applies to T2ABD, used for ALU destination selection with other two register sets, as shown in *Table 426*. Microinstruction fields ABSE and ABDE control the register set selection for source and destination, respectively, when available at the format. In formats without ABSE/ABDE, the field T4BBS determines the register sets used by T2ABD and T4ABS, as shown in *Section , Microinstructions Without Fields ABSE and ABDE*.

Microinstructions with fields ABSE and ABDE

In microinstructions where ABSE and ABDE fields are available (1 bit each), ABSE controls register set selection for T4ABS and ABDE controls register set selection for T2ABD. *Table 422* shows the meaning of values for ABSE and ABDE fields.

Table 422. Register Set Selection by ABSE or ABDE

ABSE or ABDE	Register set selected
0	second
1	first

Microinstructions Without Fields ABSE and ABDE

When ABSE and ABDE are not available in a microinstruction format, register set is selected by T4BBS field. *Table 423* explains how to select a register set either for T4ABS and T2ABD.

Table 423. Register set selection by T4BBS w/o ABSE, ABDE

T4BBS	Register set for T2ABD	Register set for T4ABS
0xx	first	first
100	second	second
101	second	first
110	first	second
111	first	first
none[1]	first	first

1. Refers to operations with immediate data as B-source, without ABSE, ABDE.

Selecting sources and destination

All ALU/MDU operations need two sources (called AS and BS) and one destination (called AD), except for some of those that use immediate data (see *Section , Operations with immediate data*). Fields T4ABS (4 bits), ABSE (1 bit), T4BBS (3 bits) select sources, while T2ABD (4 bits) and ABDE (1 bit) select the destination. When MDU is used (multiply/divide), T2ABD destination selection is ignored and results are stored in MACH and MACL (see *Section , MAC and Divide Unit (MDU)*). ABSE and ABDE are not available in some microinstruction formats that support ALU/MDU operations. However, in all formats where ABSE is available, ABDE is also available and vice-versa. The existence of ABSE/ABDE fields changes the meaning of T4BBS field, as shown in *Table 424*. On instructions with immediate data, it is used as B-source (see *Section , Operations with immediate data*).

All sources and destinations have a size associated to them, and these sizes are used to select flag sample position (see *Section , Flags sampling control*). Sizes can be 8, 16 or 24 bits. Registers that are not exactly of one of these sizes are treated as the immediately upper size (e.g., CHAN[4:0] is an 8-bit source). See *Section , Flags sampling control*, for more information.

Some parallelism issues arise when selecting P, DIOB, ERTA or ERTB as destination registers, since they can be modified by other microoperations in the same microinstruction (see *Section , Microinstruction parallelism issues*, for details).

Table 424. B source selection – T4BBS

T4BBS	Meaning in microinstruction formats with ABSE/ABDE	Meaning in microinstruction formats without ABSE/ABDE[1]
000	BS[23:0] = P[23:0]	
001	BS[23:0] = A[23:0]	
010	BS[23:0] = SR[23:0]	
011	BS[23:0] = DIOB[23:0]	
100	reserved	BS = 0
101	reserved	BS = 0
110	reserved	BS = 0
111	BS = 0, or Max const., if CIN = 0 and BINV = 0 (see *Section , Generating "max" constant*).	

1. T4BBS also selects A-source and destination register set in this case, according to *Table 423*.

T4ABS selects one source from two register sets, shown in *Table 425*. ABSE and T4BBS control which set T4ABS field uses to select the source. For more information about how to select a register set for T4ABS and T2ABD see *Section , Source and destination register set selection*. All sources are zero-filled to 24 bits, unless sign-extension is specified (see *Section , A-Source size override*).

Table 425. A Source Selection – T4ABS

T4ABS	First register set		Second register set	
	Selected register	Size	Selected register	Size
0000	AS[7:0]=P[7:0]	8	AS[7:0] = 0	8
0001	AS[7:0]=P[15:8]	8	AS[23:0]=C[23:0]	24
0010	AS[7:0]=P[31:24]	8	AS[15:0] = TPR[15:0]	16
0011	AS[23:0] = ERTB[23:0]	24	AS[23:0] = B[23:0]	24
0100	AS[23:0] = D[23:0]	24	AS[23:0] = TRR[23:0]	24
0101	AS[15:0] = P[15:0]	16	AS[7:0] = 0, read_match(1)	8
0110	AS[15:0] = P[31:16]	16	AS[13:0] = RAR[13:0]	16
0111	AS[7:0] = P[23:16]	8	AS[23:0] = MACH[23:0]	24
1000	AS[23:0] = P[23:0]	24	AS[23:0] = MACL[23:0]	24
1001	AS[23:0] = A[23:0]	24	AS[4:0]=CHAN[4:0]	8
1010	AS[23:0] = SR[23:0]	24	AS[14:2] = CHAN_BASE(2)	16
1011	AS[23:0] = DIOB[23:0]	24	AS[13:0] = ENGINE_BASE(3)	16
1100	AS[23:0] = TCR1[23:0]	24	Reserved	-
1101	AS[23:0] = TCR2[23:0]	24	Reserved	-
1110	AS[23:0] = ERTA[23:0]	24	Reserved	-
1111	AS[23:0] = 0	24	Reserved	-

1. T4ABS = 0101 with second register set also reads MatchA/B registers into ERTA/B (see *Section , Special T4ABS source operation: Read match registers*).
2. CHAN_BASE is the selected channel's base SPRAM address in channel relative address mode (see *Section , CHAN_BASE as a Source*).
3. ENGINE_BASE is the ETPU_ECR field ERBA shifted left nine positions.

T2ABD selects the destination from one of two register sets, shown in *Table 426*. ABDE and T4BBS control which set T2ABD field uses to select the destination.

Table 426. Destination selection – T2ABD

T2ABD	First register set		Second register set	
	Selected register	Size	Selected register	Size
0000	A[23:0] = AD[23:0]	24	C[23:0] = AD[23:0]	24
0001	SR[23:0] = AD[23:0]	24	LINK[7:0] = AD[7:0]	8
0010	ERTA[23:0] = AD[23:0](1)	24	TPR[15:0] = AD[15:0]	16

Table 426. Destination selection – T2ABD

T2ABD	First register set		Second register set	
	Selected register	Size	Selected register	Size
0011	ERTB[23:0] = AD[23:0][2]	24	B[23:0] = AD[23:0]	24
0100	DIOB[23:0] = AD[23:0]	24	CHAN[4:0] = AD[4:0]	8
0101	P[15:0] = AD[15:0]	16	D[23:0] = AD[23:0]	24
0110	P[31:16] = AD[15:0]	16	RAR[12:0] = AD[12:0]	16
0111	P[23:0] = AD[23:0]	24	MACH[23:0] = AD[23:0]	24
1000	TCR1[23:0] = AD[23:0]	24	MACL[23:0] = AD[23:0]	24
1001	TCR2[23:0] = AD[23:0]	24	Reserved	-
1010	P[31:24] = AD[7:0]	8	Reserved	-
1011	P[23:16] = AD[7:0]	8	Reserved	-
1100	P[15:8] = AD[7:0]	8	Reserved	-
1101	P[7:0] = AD[7:0]	8	Reserved	-
1110	TRR[23:0] = AD[23:0]	24	Reserved	-
1111	no destination selected[3]	24	Reserved	-

1. T2ABD = 0010 with first register set also writes to MatchA or UDCM registers of the selected channel if field ERWA = 0 (see *Section , Write Channel Match and UDCM Registers*).
2. T2ABD = 0011 with first register set also writes to MatchB register of the selected channel if field ERWB = 0.
3. if no destination is selected, ALU flags are updated, although the result is lost.

Max constant generation with T4BBS = 111

When T4BBS = 111, BINV = 0, and CIN = 0, the value assigned to BS will be 0x800000, and not 0x0 as expected. See *Section , Generating “max” constant*, for a detailed explanation.

Special T4ABS source operation: Read match registers

When T4ABS = 0101 and the source for T4ABS is selected from the second register set, the constant 0x00 is used as AS (8-bit size) and the following register transfer is performed in parallel as well: match registers of the selected channel (value in CHAN register) are copied to ERTA/ERTB registers, where ERTA receives the value of MatchA register and ERTB receives the value of MatchB register (see *Section , ER – Event Registers*). Note that ALU destination can still be chosen by T2ABD in parallel. When ERTA or ERTB is selected by T2ABD, a parallelism issue arises (see *Section , ALU operations and read match registers*).

CHAN_BASE as a Source

Each channel has a parameter base address in SPRAM, which is configured in ETPU_CxCR registers, CPBA field (see *Section , ETPU_CxCR – eTPU Channel x Configuration Register*). CHAN_BASE, which represents a parameter address (CPBA*2), can be used as A-source using T4ABS = 1010 when T4ABS selects a source from the second register set. In this case, CHAN_BASE is loaded into AS[13:2] to form the byte address (AS[23:14] = 0, AS[1:0] = 0). For example, in Indirect addressing mode, where the destination register is DIOB, CHAN_BASE is loaded into DIOB[13:2], which is the parameter address, and DIOB[13:0] represents the byte address. CHAN_BASE is the base address of the selected channel (given by CHAN register).

Flags sampling control

This section explains how the flags Z (zero), C (carry), N (negative) and V (overflow) are updated in an ALU operation. When there are post-ALU shift operations, the ALU Carry Out is not directly sampled in Carry flag, but passed to the post-ALU shifter (see *Section , Shift operations*). Since the size of source operands in ALU operations is variable, flags can be sampled as an operation of 8, 16 or 24 bits wide. The operation size selection is automatic, based on defined sizes of sources and destination, using the equation:

`operation_size = minor(size_of(destination), greater(size_of(A-Source), size_of(B-Source)))`

Operation size can also be shown with the following table:

Table 427. Operation size determination

A Source	B Source	Destination	Operation size
x	x	8 bits	8 bits
8 bits	8 bits	x	8 bits
16 or 24 bits	x	16 bits	16 bits
16 bits	8 or 16 bits	24 bits	16 bits
8 or 16 bits	16 bits	24 bits	16 bits
24 bits	x	24 bits	24 bits
x	24 bits	24 bits	24 bits

Note: Whenever BS = (constant) 0, its size is considered 8 bits, and all 24 bits in B-bus are set to 0. Therefore, all operations with BS = (constant) 0 have their size determined by AS and Destination only.

CCS field (1 bit) controls whether flags will be updated or not (*Table 429*). When CCS bit exists in a microinstruction, the operation size will be used to sample flags. In some microinstructions CCS field is replaced by CCSV (2 bits, *Table 428*). Flag sampling according to CCSV can be set as defined by the operation size, or fixed as 8 or 16-bit operations.

Note: For the post-ALU rotate right operation, CCSV also determines the rotate size: whether 8-bit, 16-bit, or determined by the operation size.

Table 428. Flag Sampling Using CCSV field

CCSV	Meaning
00	sample flags as an 8 bit operation
01	sample flags as a 16 bit operation
10	sample flags as defined by operation size
11	do not sample flags

Table 429. Flag Sampling Using CCS field

CCS	Meaning
0	sample flags as defined by operation size
1	do not sample flags

When neither CCS nor CCSV are present in the microinstruction, flags are not sampled. CCS and CCSV do not affect the Carry update on Exchange Bit operation (see *Section , Exchange bit*), but does control the N and Z flags.

B-Source inversion

The data selected as second source (T4BBS) can be inverted (bitwise boolean NOT) before operation. This is controlled by microinstruction field BINV (1 bit, *Table 430*). A zero value for BINV activates B-source inversion.

Table 430. B-Source Inversion – BINV

BINV	Meaning
0	invert B-source(1)
1	keep B-source bus unchanged

1. Except on max-constant selection, see *Section , Generating “max” constant*.

BINV also selects between *adc* or *sbc* enhanced ALU operation, using inverted C flag as carry-in besides BS inversion for *sbc*. Note that BINV does not invert carry in fixed-carry operations (see *Section Table 431., ALU Carry-In Control*).

When BINV = 0, T4BBS = 111 and CIN = 0, the value assigned to BS is 0x800000, instead of 0x0. See *Section , Generating “max” constant*, for more details.

Carry-in Control

CIN field (1 bit, *Table 431*) controls the carry-in for addition/subtraction operations. Functionality of CIN field depends on the arithmetic operation selected by ALUOP. When ALUOP is not available in microinstruction, the operation selected is *add*. For carry-in control in MDU operations, see *Table 414*.

Table 431. ALU Carry-In Control

Operation	CIN = 0	CIN = 1
add (addition)	carry-in used is 1(1)	carry-in used is 0
adc (addition with carry)(2)	carry-in used is C flag	
sbc (subtraction with borrow)(3)	carry-in used is inverted C flag	

1. Except on max-constant selection, see *Section , Generating “max” constant*.
2. Selected by ALUOP = 11000 and BINV = 1
3. Selected by ALUOP = 11000 and BINV = 0

Generating “max” constant

When T4BBS = 111, CIN = 0 and BINV = 0, BS is assigned to 0x800000 (called “max constant”) instead of 0x000000. This is an exception for CIN and BINV fields: when “max constant” is selected, the carry in is 0 and B-source (“max constant” itself) is not inverted, neither the carry out.

“Max constant” is the value which, added to a time base value minus 1, gives the farthest wrapped time base value that satisfies a channel greater-equal comparison. See *Section 22.5.5, Enhanced Channels*, for more information.

Shift operations

There are three types of shift operations: ALU, post-ALU and Shift Register. ALU shift operations are covered in *Section , ALU/MDU Operation Selection*. Post-ALU and Shift Register are covered in the following sections.

Shift register operations

SR can be used as a general purpose register and it can easily shift-right its contents, combined or not to post-ALU shift operations. If field SRC (1 bit) in microcode is 0, SR will shift its contents 1 bit to the right according to the algorithmic description below. SR shifting operation depends also on SHF or ALUOP fields. ALUOP and SHF never exist both on the same microinstruction format.

Table 432. Shift Register Control – SRC

SRC	Meaning
0	shift right SR 1 bit
1	no shift

SR Operation:

```
SR[22:0] = SR[23:1];
if SHF == “01” or ALUOP == “10110” then
 SR[23] = ALU_OUT[0];
else
  SR[23] = 0;
endif;
```

Post-ALU shift operations

Post-ALU shift can be selected by SHF field (2 bits) or by some specific ALUOP field values. SHF and ALUOP fields are never both available in the same microinstruction format. When selecting post-ALU shift operation using ALUOP field, ALU will always add the sources before shifting the result.

Table 433. Post-ALU shift operation

Post ALU operation	SHF(1)	ALUOP
shift left (1 bit)	00	10101
shift right (1 bit)	01	10110
rotate right (1 bit)	10	10111
no shift/rotate	11	any other(2)

1. ALU performs AS+BS before shift/rotate for all SHF values.

2. Some ALUOP combinations perform shift/rotate, but not using the Post-ALU Shifter (see *Table 438*)

Carry flag is only updated when CCS or CCSV[1:0] fields allow it (see *Section , Flags sampling control*). Algorithmic descriptions of post-ALU shift operations are presented in *Section , ALU ADD Operation with and without shifting*.

Conditional ALU/MDU operation execution

The 3-bit field AS/CE allows conditional execution of arithmetic operation, as shown in *Table 434*. The same field can also be used for overriding the size of A-Source (see *Section , A-Source size override*).

Table 434. ALU/MDU conditional execution

AS/CE	Meaning
000	Used for A-Source size override (see *Section , A-Source size override*)
001	
010	execute if C = 1
011	execute if C = 0
100	execute if Z = 1
101	execute if Z = 0
110	execute if N = 1
111	execute unconditionally/no size override

Other operations not related to ALU/MDU in the same microinstruction are not affected by the AS/CE field.

If a conditional operation is selected, there is no A-Source size override; similarly, when size override for A-Source is selected, the ALU/MDU operation executes unconditionally.

When a conditional ALU/MDU operation is not executed:

- The destination register is not updated. If the destination is CHAN, no actions associated with CHAN assignment occur (see *Section , Channel Selection Register – CHAN*).
- The ALU and MDU flags are not updated.
- MDU does not start any operation, i.e., MACH and MACL are not updated.
- SR does not shift.
- T4ABS-selected read-match does not occur.

A-Source size override

Some values if the AS/CE field are used for A-Source Size Override, as shown in *Table 435*.

Table 435. A-Source size override

AS/CE	Meaning
000	A-source size override to 8 bits
001	A-source size override to 16 bits

Table 435. A-Source size override (continued)

AS/CE	Meaning
010	Used for conditional execution (see *Section , Conditional ALU/MDU operation execution*)
011	
100	
101	
110	
111	execute unconditionally/no size override

Register size override zero-pads an overridden source to 24-bits (if no sign extension is performed, see *Section , A-source sign extension*) and affects operation size calculation. When register source is wider than size override, most significant bits of selected register are not used as A source (zeros are used instead). When size override is wider than selected register, register value is padded to zeros.When size override is used with MDU operations, it affects only the operand values, but not the operation size: MDU operation size is fully determined by the operation definition (fields ALUOP, ALUOPI).

Table 436. AS/CE field A source size override funcionality

Size override	Size of selected register	AS value[1]
8 bits	8 bits	reg[7:0]
8-bits	16 bits	reg[7:0]
8bits	24 bits	reg[7:0]
16 bits	8 bits	reg[15:0]
16 bits	16 bits	reg[15:0]
16 bits	24 bits	reg[15:0]

1. All values are zero-padded to 24 bits

A-source sign extension

The SEXT microinstruction field forces sign extension of A source according to the size of A operand, either overridden or not by AS/CE field, according to *Table 437*. The sign is taken from the size-overridden value, not the original one.

Table 437. A source Sign Extension

SEXT	Meaning
0	extends sign of A source from its size to 24-bits
1	does not extend sign of A source

A-source sign is not extended in microinstructions without SEXT field, even if AS/CE field is present.

ALU/MDU Operation Selection

When field ALUOP is available in microinstruction, enhanced ALU operations shown in *Table 438* can be performed, otherwise addition is performed. The ALU operations are defined in *Section , ALU and Post-ALU Shifter*. The MDU operations are defined in *Section , MAC and Divide Unit (MDU)*.

Table 438. ALU Operation Selection – ALUOP

ALUOP	Operation	Comment
00000	AS mults BS[7:0]	signed multiplication
00001	AS multu BS[7:0]	unsigned multiplication
00010	AS fmults BS[7:0]	signed fractional multiplication
00011	AS fmultu BS[7:0]	unsigned fractional multiplication
00100	AS mults BS[15:0]	signed multiplication
00101	AS multu BS[15:0]	unsigned multiplication
00110	AS fmults BS[15:0]	signed fractional multiplication
00111	AS fmultu BS[15:0]	unsigned fractional multiplication
01000	AS mults BS[23:0]	signed multiplication
01001	AS multu BS[23:0]	unsigned multiplication
01010	AS macs BS[23:0]	signed multiply-accumulate
01011	AS macu BS[23:0]	unsigned multiply-accumulate
01100	AS div BS[7:0]	unsigned division by 8-bit value
01101	AS div BS[15:0]	unsigned division by 16-bit value
01110	AS div BS [23:0]	unsigned division by 24-bit value
01111	n.a.	RESERVED
10000	AS[23:0] | BS[23:0]	24 bit bitwise OR
10001	AS[23:0] ^ BS[23:0]	24 bit bitwise XOR
10010	AS[23:0] & BS[23:0]	24 bit bitwise AND
10011	abs(AS)	absolute value of AS
10100	AS + BS	arithmetic addition
10101	(AS + BS) shl 1	arithmetic addition with 1-bit post-ALU shift left. (*Section , Post-ALU shift operations*)
10110	(AS + BS) shr 1	arithmetic addition with 1-bit post-ALU shift right (*Section , Post-ALU shift operations*)
10111	(AS + BS) ror 1	arithmetic addition with 1-bit post-ALU rotate right (*Section , Post-ALU shift operations*)
11000	AS adc/sbc BS(1)	addition/subtraction with C flag (*Section , Carry-in Control*)
11001	AS shl (2^(BS[1:0]+1))	AS is shifted left: 2 bits for BS = 0; 4 for BS = 1; 8 for BS=2; 16 for BS=3
11010	AS shr (2^(BS[1:0]+1))	AS is shifted right: 2 bits for BS = 0; 4 for BS = 1; 8 for BS=2; 16 for BS=3

Table 438. ALU Operation Selection – ALUOP (continued)

ALUOP	Operation	Comment
11011	AS ror (2^(BS[1:0]+1))	AS is rotated right: 2 bits for BS = 0; 4 for BS = 1; 8 for BS=2; 16 for BS=3
11100	AS exch BS[4:0]	exchange C flag and AS bit determined by BS[4:0] (*Section , Exchange bit*)
11101	AS setb BS[4:0]	set bit in AS determined by BS[4:0] (2)
11110	AS clrb BS[4:0]	clear bit in AS determined by BS[4:0][2]
11111	n.a.	RESERVED

1. Addition/Subtraction is selected by field BINV (see *Section , B-Source inversion*)

2. In setb and clrb operations, the register that drives A source is not changed, unless selected as destination of the operation.

Operations with immediate data

Immediate data can be used with some specific microinstruction formats. eTPU microcode allows 8-, 16- or 24-bit immediate data. Immediate data is loaded as B-source, so T4BBS field is not available. When using 24-bit immediate, an add is performed with A-source = 0 and flags are not updated. ALU operations are available with 8-bit immediate, although the field that selects ALU operation in this case is ALUOPI.

24-bit immediate destination

When using 24-bit immediate data, the destination register is selected by T2D field (2 bits), according to *Table 439*.

Table 439. 24-bit Immediate Destination – T2D

T2D	Target Register
00	P[23:0]
01	A[23:0]
10	SR[23:0]
11	DIOB[23:0]

Enhanced ALU operations with immediate data

Enhanced operations with immediate data, selected by ALUOPI (5 bits) are allowed only with an 8 bit immediate operand (see *Table 440*).

Table 440. ALU Operation Selection With Immediate Data – ALUOPI

ALUOPI	Operation	Comment
00000	AS mults #imm8	signed multiplication
00001	AS multu #imm8	unsigned multiplication
00010	AS fmults #imm8	signed fractional multiplication
00011	AS fmultu #imm8	unsigned fractional multiplication
00100	AS div #imm8	unsigned division
00101	n.a.	reserved

Table 440. ALU Operation Selection With Immediate Data – ALUOPI (continued)

ALUOPI	Operation	Comment
00110	n.a.	reserved
00111	n.a.	reserved
01000	AD[7:0] = AS[7:0] | #imm8, AD[23:8] = AS[23:8]	bitwise OR
01001	AD[7:0] = AS[7:0] ^ #imm8, AD[23:8] = AS[23:8]	bitwise XOR
01010	AD[7:0] = AS[7:0] & #imm8, AD[23:8] = AS[23:8]	bitwise AND
01011	AD[7:0] = AS[7:0] & #imm8, AD[23:8] = 0x0	bitwise AND with clear
01100	AD[15:8] = AS[15:8] | #imm8, AD[23:16] = AS[23:16], AD[7:0] = AS[7:0]	bitwise OR
01101	AD[15:8] = AS[15:8] ^ #imm8, AD[23:16] = AS[23:16], AD[7:0] = AS[7:0]	bitwise XOR
01110	AD[15:8] = AS[15:8] & #imm8, AD[23:16] = AS[23:16], AD[7:0] = AS[7:0]	bitwise AND
01111	AD[15:8] = AS[15:8] & #imm8, AD[23:16] = 0x0, AD[7:0] = 0x0	bitwise AND with clear
10000	AD[23:16] = AS[23:16] | #imm8, AD[15:0] = AS[15:0]	bitwise OR
10001	AD[23:16] = AS[23:16] ^ #imm8, AD[15:0] = AS[15:0]	bitwise XOR
10010	AD[23:16] = AS[23:16] & #imm8, AD[15:0] = AS[15:0]	bitwise AND
10011	AD[23:16] = AS[23:16] & #imm8, AD[15:0] = 0x0	bitwise AND with clear
10100	AS + #imm8	arithmetic addition
10101	(AS + #imm8) shl 1	arithmetic addition with 1-bit shift left.
10110	(AS + #imm8) shr 1	arithmetic addition with 1-bit shift right
10111	(AS + #imm8) ror 1	arithmetic addition with 1-bit rotate right
11000	n.a.	reserved
11001	AS shl (2^(#imm8[1:0]+1))	AS is shifted left: 2 bits for #imm8 = 0; 4 for #imm8 = 1; 8 for #imm8=2; 16 for #imm8=3
11010	AS shr (2^(#imm8[1:0]+1))	AS is shifted right: 2 bits for #imm8 = 0; 4 for #imm8 = 1; 8 for #imm8=2; 16 for #imm8=3
11011	AS ror (2^(#imm8[1:0]+1))	AS is rotated right: 2 bits for #imm8 = 0; 4 for #imm8 = 1; 8 for #imm8=2; 16 for #imm8=3

Table 440. ALU Operation Selection With Immediate Data – ALUOPI (continued)

ALUOPI	Operation	Comment
11100	AS exch #imm8[4:0]	exchange C flag and AS bit determined by #imm8[4:0] (see *Section , Exchange bit*)
11101	n.a.	reserved
11110	n.a.	reserved
11111	n.a.	reserved

Channel control and configuration microoperations

Channel Control and Configuration fields set configuration values in the channel logic of the channel selected by the CHAN register, except fields LSR and CIRC.

Channel flags operations

Each channel has two associated hardware flags, called Channel Flag 0 and Channel Flag 1. Microcode field FLC (3 bits) allows them to be set or cleared, as shown in *Table 441*. These flags can be tested by microcode as a branch condition, and may also influence in the Entry Point taken, allowing fast state decoding. For more details, see *Section , Entry points*.

Table 441. P Flags Operation – FLC

FLC	Meaning
000	clear flag0
001	set flag0
010	clear flag1
011	set flag1
100	copy flag1:flag0 from P[25:24]
101	copy flag1:flag0 from P[27:26]
110	copy flag1:flag0 from P[29:28]
111	no operation (nil)

Comparator and time base selection

TBSA and TBSB fields (4-bit wide each) are used to configure the type of the comparator and the time bases used for match or capture (See *Table 442* and *Table 443*). TBSA can also be used to control the Output Buffer Enable signal (See *Table 390*).

Table 442. Time Base Selection 1 – TBSA

	TBSA bit	2	1	0
TBSA[3] = 0	**Bitfield**	**Comparator selection**	**Capture selection**	**Match TB selection**
	0	greater or equal	TCR1	TCR1
	1	equal-only	TCR2	TCR2

Table 442. Time Base Selection 1 – TBSA

	TBSA bit	2	1	0
TBSA[3] = 1	action	2	1	0
	set OBE = 1	0	0	0
	set OBE = 0	0	0	1
	do nothing	1	1	1
	reserved	all other values		

Table 443. Time Base Selection 2 – TBSB

	TBSB bit	2	1	0
TBSB[3] = 0	Bitfield	Comparator selection	Capture selection	Match TB selection
	0	greater or equal	TCR1	TCR1
	1	equal-only	TCR2	TCR2
TBSB[3] = 1	action	2	1	0
	do nothing	1	1	1
	reserved	all other values		

Transition detection and pin action control

IPACA/B and OPACA/B fields are used to configure transition detection sensitivity (for the channel input signal) or output pin action control (for the channel output signal), as defined in *Table 444*. IPACA and IPACB have the same format, where IPACA is related to Match A and first transition detection, and IPACB to Match B and second transition detection. The same applies in analogue way to OPACA and OPACB.

For the output signal, configuring OPAC registers does not change the current signal state, but defines the action to be done when a match or transition detection occurs. See *Section , Match Recognition*, and *Section , Channel Modes on Output Signal Generation*, for more information. IPACA/B = 1xx also enables assertion of MRLA/B during Time Slot Transition. See *Section , Match Recognition*.

Table 444. Input and Output Pin Action Control – IPACA/B and OPACA/B

Value	IPAC meaning	OPAC meaning
000	Do not detect transitions	Do not change output signal
001	Detect rising edge only	Match(1) sets output signal high
010	Detect falling edge only	Match(1) sets output signal low
011	Detect both edges	Match(1) toggles output signal
100	Detect input signal = 0 on Match(1)	Transition detection sets output signal low
101	Detect input signal = 1 on Match(1)	Transition detection sets output signal high

Table 444. Input and Output Pin Action Control – IPACA/B and OPACA/B

Value	IPAC meaning	OPAC meaning
110	reserved	Transition detection toggles output signal
111	do not change IPAC	do not change OPAC

1. Match A is used for IPACA/OPACA, and Match B for IPACB/OPACB.

Immediate pin state control

It is possible to change output signal state immediately by using PSC (2 bits) and PSCS (1 bit) fields.

Table 445. Immediate Pin State Control – PSC and PSCS

PSC	PSCS	Meaning
00	0	Set signal as specified by OPACA (see *Section , Transition detection and pin action control*)
00	1	Set signal as specified by OPACB (see *Section , Transition detection and pin action control*)
01	x	Set signal high
10	x	Set signal low
11	x	Don’t change signal state

Write Channel Match and UDCM Registers

Match registers can have their values changed using ERWA and ERWB fields (1 bit each). They also set their respective MRLE register (see *Section , Match Recognition*).

ERWA can also be used to program the UDCM register (see *Section , UDCM – User Defined Channel Mode*). The field CMW selects where the contents of ERTA is copied when ERWA is active (see *Table 446*).

Table 446. Write MatchA/B – ERWA/B

Field	CMW	Value	Action
ERWA	1	0	write ERTA value in MatchA. Enable matches for MatchA register (MRLEA = 1)
	0	0	write ERTA value into UDCM
	1	1	don’t change UDCM, MatchA and MRLEA
	0	1	reserved
ERWB	1	0	write ERTB value in MatchB. Enable matches for MatchB register (set MRLEB = 1)
	1	1	don’t change MatchB and MRLEB
	0		
	0	0	reserved

If ERTA or ERTB is a destination of an ALU operation and, at the same time, the respective ERWA/B field is active, the new ERTA value is the one written into the MatchA/B register or the UDCM register.

Clear transition/match event registers

Flags MRLA, MRLB, TDLA and TDLB (see *Section , ER – Event Registers*) indicate the state of matches and transitions detected in the selected channel, and it is possible to clear those flags using the microcode fields MRLA, MRLB (1 bit each) and TDL (1 or 2 bits, depending on the format). The flags cleared by these microcode fields are the actual channel flags, and also the ones sampled into the branch logic.

TDL can be one or two bits wide, depending on the microinstruction format (see *Section , Microinstruction formats*). Two-bit TDL allows independent clearing of TDLA and/or TDLB. *Table 448* defines the two-bit TDL field.

Table 447. Clear Transition/Match Event Registers – MRLA/B, TDL

Field	Meaning
MRLA	0 = clear MRLA event register, 1 = don’t change
MRLB	0 = clear MRLB event register, 1 = don’t change
TDL (1 bit)	0 = clear TDLA and TDLB flags, 1 = don’t change

Table 448. Independent TDLA/B clear – two-bit TDL

Value	Meaning
0 0	clear TDLA
0 1	clear TDLB
1 0	clear both TDLA and TDLB
1 1	do not clear TDLA or TDLB

Disable matches

Microcode field MRLE (1 bit) allows disabling matches on channel selected by CHAN register, for both MatchA and MatchB registers, by clearing their respective MRLE bits. Matches can be enabled for each Match register using ERWA and ERWB fields (see *Section , Write Channel Match and UDCM Registers*).

Table 449. Disable Matches – MRLE

MRLE	Meaning
0	Disable matches for Match A and Match B
1	don’t change match enabling

Some instruction formats have a two-bit MRLE field (see *Section , Microinstruction formats*) which allows independent disabling of Matches 1 and 2, as shown in *Table 450*.

Table 450. Two-bit MRLE

MRLE	Meaning
0 0	Disable Match A (clear MRLEA)
0 1	Disable both Matches (clear MRLEA and MRLEB)
1 0	Disable Match B (clear MRLEB)
1 1	nop

Disable match and transition service requests

Microcode field MTD (2 bits) disables match and transition service requests for the selected channel. MTD does not disable Link Service Request and Host Service Request. MTD sets or resets register SRI (for more details see *Section , SRI – Match/Transition Service Request Inhibit Latch*) and TCCEA (see *Section , TCCEA – Transition Continuous Capture Enable*).

Table 451. Disable Match and Transition Service Request – MTD

MTD	Action on SRI	Action on TCCEA
00	SRI = 0: enable service requests for match and transition	TCCEA = 0: disable transition captures(1) when TDLA = 1
01	SRI = 1: disable service requests for match and transition	
10	SRI = 1: disable service requests for match and transition	TCCEA = 1: enable transition captures(2) when TDLA = 1
11	don't change	

1. Disables only captures on transition events specified by IPACA.
2. Enables only captures into CaptureA register, on transition events specified by IPACA.

Predefined channel modes

Microcode PDCM field (4 bits) defines the channel mode (see *Section , Channel Modes*).

PDCM coding is shown in *Table 452*. Note that PDCM bit 0 selects between Single Transition (PDCM[0] = 0) and Double Transition (PDCM[0] = 1) predefined modes.

PDCM is also used to select the user-defined channel mode, defined by the channel register UDCM (see *Section , UDCM – User Defined Channel Mode*).

Table 452. Predefined Channel Modes

PDCM	Channel mode
0000	em_b_st
0001	em_b_dt
0010	em_nb_st
0011	em_nb_dt
0100	m2_st
0101	m2_dt

Table 452. Predefined Channel Modes

0110	bm_st
0111	bm_dt
1000	m2_o_st
1001	m2_o_dt
1010	**user-defined channel mode**
1011	reserved
1100	sm_st
1101	sm_dt
1110	sm_st_e
1111	keep current channel mode

Channel interrupt and data transfer requests

Microcode can issue Interrupt Requests, Data Transfer Requests and Global Exception through CIRC field. For more information see *Section , Interrupts and data transfer requests*.

Table 453. Channel and Data Transfer Requests – CIRC

CIRC	Meaning
000	Channel Interrupt Request from selected channel
001	Data Transfer Request from selected channel
010	Channel Interrupt and Data Transfer requests from selected channel
011	Channel Interrupt and Data Transfer requests from serviced channel
100	Channel Interrupt Request from service channel
101	Data Transfer Request from service channel
110	Global Exception
111	don’t request interrupt

Clear link service request

Microcode LSR field (1 bit) is used to clear the Link Service Request flag of the serviced channel (may not be the one selected by CHAN). The LSR branch condition is always cleared, but not the Link Service Request, if another channel link was received by the serviced channel during the executing thread. See *Section , Channel Link*, for more information.

Table 454. Link Service Request Negation Control – LSR

LSR	Meaning
0	clear Link Service Request (flag LSR)
1	don’t change

Flow control microoperations

eTPU has *jump* and *call* microoperations to change microcode flow. Besides, eTPU has *dispatch jump* and *dispatch call* that can be used to implement a jump table. In *call* (or *dispatch call*) microoperation, the return address is saved in the RAR. If nested sub-routine calls are necessary, return address values have to be saved in a stack, usually implemented with DIOB register.

Flow Control microoperations are also provided to repeat a given microinstruction, to finish the current thread execution, and to halt the microengine.

Ending current thread – END

Microcode END field (1 bit) finishes current thread and allows other channels to be serviced. If END field is 0, the current instruction is completed and the thread is finished. END = 1 has no effect, and the next microinstruction is executed Any MDU operation (see *Section , MAC and Divide Unit (MDU)*) that could be still pending when the thread is finished is left incomplete. END also releases any semaphore locked by the engine.

Branch operations

Branch operations can be **jump** or **call**. The target address of *jump* or *call* microoperations is always immediate and absolute. Branch microoperation is affected by FLS field (refer to *Section , Flush pipeline*).

Selecting jump or call microoperations

The only difference between *jump* and *call* microoperations is that when a *call* is executed the value of PC or PC+1 (depending on flush, see *Section , Flush pipeline*) is saved in the RAR. The microcode field J/C (1 bit) selects whether *jump* or a *call* is executed, according to *Table 455*.

Table 455. Jump / Call Selection – J/C

J/C	Meaning
0	jump
1	call

Branch target address

Microcode BAF field (14 bits) indicates the absolute address of a *jump*/*call* target.

Conditional/Unconditional branch

Jump and *call* can be conditional or unconditional, depending on the BCC (6 bits) and BCF (1 bit) fields, as shown in *Table 456* and *Table 457*. BCF determines whether branch is taken when condition specified by BCC is true or false. When a branch condition uses the channel flags, the channel context is related to the channel number written in CHAN register.

Table 456. Branch Condition Inversion – BCF

BCF	Meaning
0	branch if condition determined by BCC is false
1	branch if condition determined by BCC is true

Table 457. Branch Condition Selection – BCC

BCC	Meaning	BCC	Meaning
001110	Flag 0	001111	Flag 1
100000	V ALU flag	110000	PSS channel flag
100001	N ALU flag	110001	PRSS channel flag
100010	C ALU flag	110010	“Less Than” ALU flag combination (signed) (1)
100011	Z ALU flag	110011	“Lower or Equal” ALU flag combination (unsigned)(2)
100100	MV MDU flag	110100	P[24]
100101	MN MDU flag	110101	P[25]
100110	MC MDU flag	110110	P[26]
100111	MZ MDU flag	110111	P[27]
101000	TDLA channel flag	111000	P[28]
101001	TDLB channel flag	111001	P[29]
101010	MRLA channel flag	111010	P[30]
101011	MRLB channel flag	111011	P[31]
101100	LSR channel flag	111100	PSTO channel flag
101101	MB flag MDU flag	111101	PSTI channel flag
101110	FM[1] channel flag	111110	SMLCK semaphore flag
101111	FM[0] channel flag	111111	false
all other values reserved			

1. “less than” is a signed comparison, equal to the xor between ALU flags V and N; e.g., 0 < 0xFFFFFF tests as false (0 < -1).
2. “lower equal” is an unsigned comparison, equal to Z or C; e.g., 0 < 0xFFFFFF tests as true.

Dispatch microoperation

Dispatch microoperation is an unconditional branch where the target address is always PC+P[31:24] (unsigned). Dispatch is affected by FLS field (refer to *Section , Flush pipeline*). Dispatch microoperation is defined by R/D field (2 bits, *Table 458*). Field R/D can also be used to define return from sub-routine (see *Section , Return from subroutine*).

Table 458. Return and Dispatch – R/D

R/D	Meaning
00	return from subroutine (see *Section , Return from subroutine*)
01	dispatch jump
10	dispatch call
11	don’t change microinstruction flow

Return from subroutine

When a subroutine *call* or a *dispatch call* microoperation is executed, the return address is saved in the RAR. To return from a subroutine a microoperation is available to load the contents of the RAR back to the PC. Fields R/D (2 bits) or RTN (1 bit, *Table 459*) can be used to return from subroutine. For R/D field, see *Table 458*.

Return from subroutine microoperation is affected by FLS (see *Section , Flush pipeline*) when field R/D is used. Return execution through RTN always flushes the pipeline.

Table 459. Return from Sub-routine – RTN

RTN	Meaning
0	return with pipeline flush
1	do not return

Flush pipeline

When a branch, dispatch or subroutine return microoperation is executed, the next microinstruction can be executed unconditionally before the flow change takes effect, since microengine has a two-stage pipeline. Executing the next microinstruction after a branch maximizes execution performance. This feature is controlled by field FLS (1 bit, *Table 460*). When FLS = 0 the pipeline is flushed, so the next microinstruction placed after a branch is decoded as NOP if the branch is taken. If FLS = 1, the microinstruction placed after the branch is executed, either if the branch is taken or not, as shown in *Figure 429*.

Flush also controls which value is stored in RAR in a call: in case of no flush, it is the address of the branch/dispatch instruction + 2, even if RAR is the ALU destination of the instruction after the call; in case of a flush, it is the address of the instruction following branch/dispatch.

If a branch with no flush is followed by another branch with no flush, the instructions are executed in the following order:

1. First branch
2. Second branch
3. First branch's destination instruction
4. Second branch's destination instruction, and the flow proceeds normally from then on

The destination of the first branch must not be another flow changing instruction (branch, return or dispatch). Similar flows apply when returns or dispatches are used instead of branches. This scheme can be used to implement quick table look-ups with a dispatch replacing the first branch, for instance.

Table 460. Flush Pipeline – FLS

FLS	Meaning
0	flush pipeline when jump / call / dispatch jump / dispatch call / return is executed
1	do not flush pipeline when jump / call / dispatch jump / dispatch call / return is executed

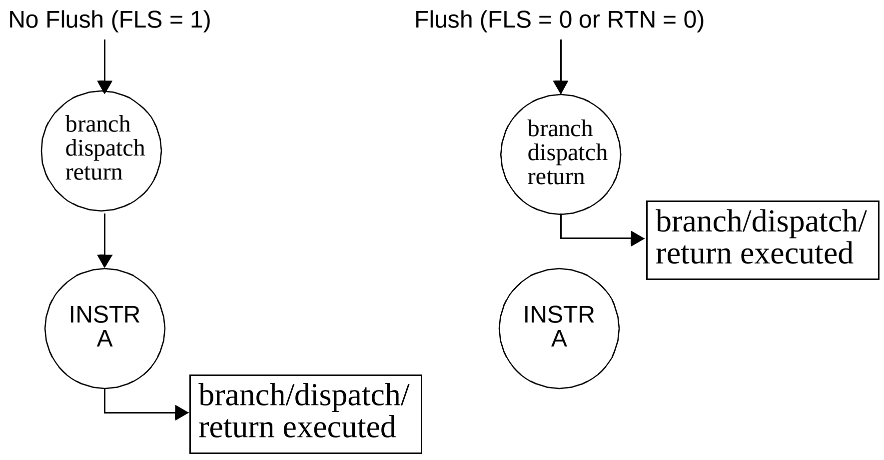


Figure 429. Flush Pipeline

HALT microinstruction

HALT is a microinstruction provided to implement software breakpoints (see *Section , Software breakpoints*). Note that HALT is coded as a microinstruction format, not a field (see *Section , Microinstruction formats*). The execution of this instruction puts the microengine in halt state. For more information about the implications of microengine halt state, see *Section , Microengine halt state*. HALT is valid only if software breakpoints are enabled at the Debug interface (signal ndedi_enable asserted). If software breakpoints are not enabled, HALT executes as a NOP and is treated as an Illegal Instruction (see *Section , Illegal Instructions*).

NOP microinstruction

There is not a unique microinstruction with an assigned opcode to do No Operation. NOP microinstruction is achieved through any of the formats shown on *Table 458* where the user can assign to each individual field the corresponding value for “No Operation”. However, to prevent future impacts of instruction changes on object code compatibility, the instruction value 0x4FFFFFFF should always be used for NOP.

Illegal Instructions

An instruction is considered illegal if any reserved field value is used, including when the fields marked rsv in the instruction formats (see *Table 462*) are assigned value 0. A HALT instruction is exceptionally considered illegal when executed with software breakpoints disabled (see *Section , HALT microinstruction*). Global Exception may be issued up to two microcycles after instruction fetch. The execution results of an illegal instruction on the microengine, channel logic or host interface are unpredictable, except for the HALT case.

If the microengine decodes an illegal instruction, the following actions are taken:

- a Global Exception is issued.
- flag ILF1/2 on register ETPU_MCR is set to indicate this occurrence to the Host.
- a breakpoint occurs, if NDEDI is present and configured to do so.

Microinstruction parallelism issues

This section clarifies parallelism issues that arise when two non-commutative microoperations appear in the same microinstruction.

ALU operations and read match registers

ALU operations have only one destination register, but there is one case where source selection determines destination: read Match register in ERTA and ERTB registers. In this case if ALU destination is ERTA or ERTB a conflict arises. The ALU destination value overwrites the value read from the match registers.

ALU and SPRAM operations

P and DIOB registers can be selected as destination by both ALU and SPRAM (read) microoperations in the same microinstruction. Since P and DIOB update from SPRAM data happens after P and DIOB update for ALU/MDU microoperations, the data read from SPRAM remains in P or DIOB after an operation when one of them is specified as destination for both ALU and SPRAM microoperations In this case, the value loaded into P or DIOB is the one read from SPRAM However, the ALU operation is executed and its flags are updated accordingly

When P or DIOB is destination of an SPRAM read and also an ALU source at the same microinstruction, the value before the read is used for the ALU operation

If DIOB is the ALU destination and P is loaded from SPRAM or vice-versa, no conflict occurs, and the result is the same as if operations occurred separately.

All the above also applies to Zero SPRAM operations.

When using P or DIOB as destination for ALU operations and also as source for a SPRAM write operation, the data written in SPRAM is the one calculated by ALU, which means it is possible to calculate a value and write it in an SPRAM address using only one microinstruction.

The old value of DIOB or the old value minus 4 (pre-decrement) is always used when DIOB is selected as address (indirect address mode), no matter if DIOB is selected as destination of either the SPRAM or ALU. For the value loaded into DIOB, refer to *Table 461*.

Table 461. DIOB load from SPRAM and ALU

DIOB selected as SPRAM read destination?	DIOB selected as ALU destination?	DIOB load value
no	no	DIOB, --DIOB (pre-decrement), or DIOB++ (post-increment)
yes	no	SPRAM read data (post-inc and pre-dec ignored)
yes	yes	SPRAM read data (post-inc, pre-dec and ALU result ignored)
no	yes	ALU result (post-inc an pre-dec ignored)

ERTA/B as ALU destination and ERWA/B

The value in ERTA and ERTB registers can be written in match registers of the selected channel by using fields ERWA and ERWB (*Section , Write Channel Match and UDCM Registers*). If, at the same microinstruction, ERTA or ERTB is the destination of an ALU/MDU microoperation, the value written in the Match registers is the ALU/MDU result.

The same applies to UDCM when ERTA is the destination of an ALU operation and instruction fields ERWA and CMW are active.

If an ALU operation occurs in parallel with ERWA/B but ERTA/B are not the destination of an ALU/MDU operation, then UDCM and MatchA/B receives the ERTA/B value.

ERWA/B and MRLE

ERWA/B automatically sets the MRLEA/B channel latch, respectively (see *Section , Write Channel Match and UDCM Registers*). Microinstruction fields ERWA/B independently sets MRLEA/B channel flags, regardless of MRLE.

CHAN assignment, Read Match and ERWA/B

When CHAN is a destination of an ALU operation it causes a read of the CaptureA/B register values into ERTA/B. The Capture registers loaded into ERTA/B are selected by the new CHAN value. The value of the CaptureA/B registers overwrites any read-match commanded simultaneously.

If CHAN assignment happens with an ERWA/B operation in the same instruction, the updated Match register(s) belong to the new selected channel.

Read Match and ERWA/B

If a read match operation is executed with ERWA/B in the same microinstruction, the MatchA/B registers receive the old values of ERTA/B, and the ERTA/B registers receive the old MatchA/B values simultaneously, i.e.: ERTA/B and MatchA/B swap their values.

If ERTA/B is the destination of an ALU operation at the same instruction, MatchA/B gets the ALU result (see *Section , ERTA/B as ALU destination and ERWA/B*), but the ERTA/B not being written still receives the old MatchA/B values.

Note: Read match, ERWA/B and CHAN assignment can be active at the same instruction. Combining rules Section , CHAN assignment, Read Match and ERWA/B, and Section , Read Match and ERWA/B, the result is: ERTA/B receives the CaptureA/B values of the new CHAN value, and MatchA/B of the new channel receives the old ERTA/B value(s).

Stack accesses and ALU operations

Post-increment is ignored in a stack operation (field STC) if DIOB is loaded from SPRAM: DIOB keeps the value read from SPRAM. Pre-decrement is ignored in a stack operation (field STC) if DIOB is destination of an ALU operation, for DIOB load value, but not for DIOB as address. Post-increment/pre-decrement remains valid in all other situations. These rules can be summarized in the following equivalent C code, and in *Table 461*.

```
DIOB = *DIOB[15:2];       // read without posinc/predec
*DIOB[15:2] = DIOB;       // write without posinc/predec
DIOB = *(--DIOB[15:2]);   // read with predec
DIOB = *DIOB[15:2];       // read with posinc (ignored)
*(--DIOB[15:2]) = DIOB;   // write with predec
*DIOB[15:2]++ = DIOB;     // write with posinc (value written is before
increment)
```

SRC and ALU/MDU operations

If operation SRC is active (field SRC = 0) and register SR is selected as destination of an ALU operation, the value of the ALU operation prevails over the shifted value.

The value of SR used as source in the ALU/MDU operation is the one before the shift.

Semaphore lock/free and SMLCK branch condition

When the SMLCK branch condition is tested at the same microinstruction of a semaphore lock or free, the condition is evaluated after the semaphore action (either free or lock) is taken.

Dispatch and SPRAM read

When the most significant byte of P is read from SPRAM (read 8 msb bits or 32 bits) and a dispatch instruction is executed simultaneously, the dispatch target address is calculated upon the P value before the read.

CHAN Assignment, PSC/PSCS, and clear MRLEA/B, MRLA/B, TDLA/B

When clear MRLEs, MRLA/B or TDLs is done and a CHAN assignment is done at the same time, the flag selected by the old CHAN value is cleared in the channel, but the branch conditions receive the state of the flags selected by the new CHAN.

When a pin action is commanded through PCS/PSCS and a CHAN assignment is done simultaneously, the output signal affected is selected with the old CHAN value.

Microinstruction formats

See *Table 462*.

Table 462. Microinstruction Formats

format	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A1	0 0 0			IMM[15:13]			IMM[7:2]						IMM[23:16]								IMM[12]	RTN	IMM[11:9]			IMM[1:0]		T2D		IMM[8]	0 0	
A2													T4ABS				T2ABD					CCS						AB SE	AB DE		0 1	
A3				ALUOP	CCSV																ALUOPI [3:2]		AS/CE					ALUOPI [1:0]		0	1 0	
A4				FLC	SHF																SRC	CCS	rsv	FLC [1:0]				AB SE	AB DE	1		
B1	1 0		0	END			CIN	BINV	T4BBS			RW										P/D		CCS	AID[7:0] (global param)							
B2			1																						ZRO	AID[6:0] (channel param)						
B3	0 0 0																									STC		AB SE	AB DE	rsv	1 1	
B4	0 0 1			0	CCSV																1		AS/CE					ALUOP				
B5												FL									0	SEXT				SMPR						
B6				1								rsv									SRC					AB SE	AB DE					
B7	0 1 1			END	SHF							TDL										PSC		MRLA	ERWA	MRLB	ERWB	ABSE	ABDE	CCS	MRLE	PSCS
C1	0 1 0			0	END	OPACA			OPACB				TBSA				TBSB				LSR							PDCM				
C2				1		IPACA			IPACB																							
D0	1 1 0			0		MRLE	0	CIRC	PSC		FLS	RW	PSCS	FLC			CIRC[1:0]		R/D		0	P/D	RSIZ		ZRO	AID[6:0] (engine param)						
D1							1																		AID[7:0] (global param)							
D2																					1				ZRO	AID[6:0] (channel param)						
D3	1 1 1			1		MRLE															1					STC		1 1		0	0	rsv
D4												FL									0	rsv				SMPR						
D5	1 1 0			1		MRLE	rsv	CMW	MTD		TDL	RW	TDL				MRLA	ERWA	MRLB	ERWB	0	P/D	RSIZ		AID[7:0] (global param)							
D6																					1				ZRO	AID[6:0] (channel param)						
D7	1 1 1			1		MRLE															1					STC		1 1		0	1	rsv
D8												FL									0	rsv				SMPR						
D9	1 1 0			0		MRLE	0					RW									1	P/D	RSIZ		ZRO	AID[6:0] (engine param)						

Table 462. Microinstruction Formats (continued)

format	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
E1	1 1 1			BCC[5]	J/C	BCC[4:0]					FLS	RW	BCF	BAF[13:0]														00		P/D	STC	
E2																												01		AID[2:0]		
E3												FL																10		rsv	SMPR	
E4												0																11		1	rsv	
F1				rsv								1	rsv															111			rsv	

ALU Operations	Channel Control/Config Operations
RAM Operations	Flow Control Operations

22.5.10 Test and Development Support

Overview

Following sections describe several features available to support development and test. Most debug features, described in *Section , Development support features*, are accessible through a separate debug bus, and are not available through registers in the standard eTPU memory map. The details of the access to this interface are MCU-dependent, but a separate IP block, called NDEDI, is provided so that these features are accessible by a Nexus interface. IP-bus Green line device debug request can also be used to put microengines in halt state. Conditions for the assertion of this line are also MCU-dependent.

Section , Test support features, describes embedded test features: the Multiple Input Signature Calculator (MISC) is an SCM test feature accessible through registers ETPU_MCR and ETPU_MISCCMPR (see *Section 22.4.2, System configuration registers*). MISC allows SCM test “on the fly, that is, while eTPU is running, with no impact on eTPU functionality or performance.

Development support features

Internal Debug Interface and Nexus Class 3 support

eTPU provides an Internal Debug Interface that exports real-time microengine states and values, including breakpoint/watchpoint information. It also provides inputs for breakpoint request from other blocks or outside MCU.

NDEDI is an IP block designed to support Nexus functionality for the eTPU. When Internal Debug Interface is connected to an NDEDI block, the MCU can provide Nexus Class 3 debug interface. Nexus is a development support external interface defined by the **IEEE standard ISTO 5001-1999**.

Some of the next subsections describe debug features provided by the Internal Debug Interface combined with the NDEDI block. NDEDI can be replaced by other block providing a different programming interface, such as a register debug interface, for instance.

Microengine halt state

Halt is a microengine state where it suspends execution during a thread, or does not start executing a scheduled thread from idle state. While Idle State is entered from END execution without any other scheduled thread, microengine enters Halt State by any of the following events:

- Execution of the HALT microinstruction (software breakpoint).
- External halt request through the Debug Interface (includes Nexus breakpoint request via EVTI input pin (see *Section , Internal Debug Interface and Nexus Class 3 support*).
- The other engine enters halt state and they are configured to halt simultaneously (bit HTWIN is asserted via Nexus Interface).
- IPI Green line device debug request assertion and NDEDI register NDEDIETPUx_DC field CBI = 1. If same register’s field CBT = 1, microengine halts at the next time-slot boundary, if CBT = 0 it halts immediately. As a particular case, microengines come halted out of reset if device debug request is asserted, since CBI reset value is 1. Microengine does not execute out of reset, either in halt (device debug request

asserted) or idle state (device debug request negated), but halt enables several other features (see below).

- Occurrence of any of the hardware breakpoint conditions. See *Section , Hardware breakpoints*, for details.
- Execution of a single-step microinstruction: microengine returns to halt state after executing a single microinstruction while in halt state. See *Section , Single-step execution*, and *Section , Forced microinstruction execution*, for details.

When microengine enters halt state, it automatically triggers the following actions:

- Suspends input signal sampling and filters (respective engine channels only), if signal ndedi_stop_pins is asserted at the Debug Interface.
- Releases the SPRAM arbitration for Host or CDC accesses, no matter if microengine was halted in the middle of a dual-parameter (back-to-back) access.
- Stops TCR1/2 clocks of the respective engine, if signal ndedi_stop_tcr is asserted at the Debug Interface.
- If the other engine is also in halt state or stopped, allows turning ETPU_MCR VIS bit to 1.

If all halt conditions are cleared when VIS = 1, microengine(s) keep on halt state until VIS = 0, when it automatically exits halt state, except on single-step (see *Section , Single-step execution*), so that single-step execution is ignored while VIS = 1.

MDU continues executing until it finishes any ongoing operation even if microengine is in halt state, except when the halted instruction is an END.

There are two kinds of halt state, depending on the previous microengine state when halted:

1. **halt_idle**, if the engine was not executing a thread when halted; the engine cannot leave halt_idle to fetch instructions, so one cannot single-step or follow a program flow; it can, however, execute forced instructions (see *Section , Forced microinstruction execution*).
2. **halt_exec**, if the engine was executing a thread when halted. The engine can single-step and continue a program flow from halt_exec.

When microengine exits halt state, any dependable action is suspended and, if exiting halt_exec, the instruction pointed by the PC is fetched, while the instruction already fetched before halt is executed. Note that both the PC and the prefetched instructions can be modified during halt state, with a forced execution of a branch instruction (see *Section , Forced microinstruction execution*).

Hardware breakpoints

Microengine can enter halt state through a command from the Debug Interface, configuring a hardware breakpoint. Hardware breakpoints can halt the microengine on specific conditions, listed below. These conditions depend on NDEDI configuration.

- CHAN register assignment (only by microcode, not by time slot transition).
- SPRAM read and/or write to a given address and/or write data. The breakpoint is always qualified by the SPRAM address, but the following variations are allowed:
  - break on write only, read only, or read-and-write.
  - break on higher-byte write data value, lower 24-bit write value, full word (32-bit) write value, or regardless of data. Break on read data is not supported.
- PC (program counter) value.
- Beginning of a thread with a Host Service Request pending.
- Beginning of a thread with a Link Service Request pending.
- Beginning of a thread with a Match Service Request pending.
- Beginning of a thread with a Transition Service Request pending.
- End of a thread.
- Illegal instruction execution.

All these conditions can also be qualified by the value of the CHAN register.

On any of these conditions, halt of one microengine does not depend on the halt of the other, unless the other engine is configured to do so, via Nexus Interface. Occurrence of any of these conditions halts the microengine, i.e., the conditions are logically “ORed” together, and they can be individually enabled.

While in halt state, the microengine can also execute any forced microinstruction not in the normal program flow (see *Section , Forced microinstruction execution*) or, if in halt_exec, in single-step (see *Section , Single-step execution*).

There are situations when requests for stopping an engine, breakpoint and service can occur simultaneously. Breakpoint requests always prevails over a stop request (ETPU_ECR bit MDIS = 1 or device debug request = 1). When the eTPU is idle: stop request prevails over Service Request if there is not a hardware breakpoint request; a hardware breakpoint request leads to debug mode immediately if there is no Service Request, and after TST if there is Service Request (regardless of stop requests). The rules above are summarized in the *Table 463*, showing the destination state of the microengine in each situation.

Table 463. Breakpoint, stop and service requests resolution from idle

Breakpoint request	MDIS	Service request	Final state
no	0	0	Idle
no	1	0	Stop
no	0	1	TST
no	1	1	Stop
Yes	0	0	Halt_idle
Yes	1	0	Halt_idle
Yes	0	1	TST(1)
Yes	1	1	TST(1)

1. Breaks after TST, if signal ndedi_sync_break is still asserted.

When a thread is ending, it goes to Idle or TST only if there is neither a hardware breakpoint request (signal ndedi_thread_break negated) nor a request to stop (MDIS = 1 or device debug request = 1). When thread is ending and there are simultaneous hardware breakpoint (ndedi_thread_break active) and stop (MDIS = 1 or device debug request = 1) requests, hardware breakpoint prevails and the engine enters Debug mode (Halt_idle state).

If the engine entered Debug mode after a thread finished (Halt_idle state) and a “go” command comes from the debug interface, the engine state machine goes to Idle and the rules above apply. It means that if a “go” is issued in Halt_idle state with MDIS = 1, the engine goes to Idle for one microcycle and then stops (if MDIS or device debug request keeps asserted and there is no other breakpoint request).

Note: *Hardware breakpoint requests are ignored for the first microinstruction executed when microengine leaves halt.*

Hardware watchpoints

Debug Interface allows watchpoints on the same conditions available for hardware breakpoints (see *Section , Hardware breakpoints*).

Software breakpoints

A software breakpoint occurs when microengine executes a HALT microinstruction. Any number of software breakpoints can be set in code, usually replacing an active microinstruction.

Like any other microinstruction, HALT increments the PC and pre-fetches the next instruction. So, before the halt state is suspended, if the original program flow must be followed, the original instruction at the HALT address must be executed, regardless if the software breakpoint is removed (replacing HALT by the original microinstruction) or not. The following is the procedure to resume execution from a software breakpoint:

1. Restore the original instruction in SCM (replace HALT).
2. Force a jump with flush to the original instruction (see *Section , Forced microinstruction execution*).
3. If the software breakpoint must be kept: single-step and replace the original instruction with a HALT.
4. Let the flow continue, issuing a GO command (leaving halt state).

Special care must be taken if HALT is followed by another HALT, and the second HALT is removed when microengine was halted by the first one. In this case, replacing the second HALT with the original microinstruction is not enough to remove the second breakpoint, because the second HALT was already prefetched and would be executed anyway when halt was suspended. The debugger must also do a forced execution of unconditional branch with flush to the original microinstruction address. That will clear the pipeline, replacing the prefetched instruction with a NOP, and load PC with the address of the removed breakpoint. So, when halt state is suspended, the original microinstruction will be fetched while NOP is executed, and program flow continues normally from then on.

Note: *A HALT instruction placed after a no-flushing branch, dispatch or return may be a problem from the debugger application standpoint: after the HALT is executed, the eTPU debug interface informs the address of the branch/dispatch/return destination, and the debugger application has no direct way to identify which HALT instruction was executed, if multiple HALTs lead to the same address. This can be solved if the debug support block (NDEDI) has a register holding the address of the last instruction executed, otherwise one should forbid non-flushed HALT instructions.*

Software breakpoint setting and removal is possible only with SCM RAM implementations or ROM implementations with SCM RAM emulation (see *Section , SCM emulation*). There is only one way of inserting software breakpoints into SCM RAM: writing bit VIS = 1 in register ETPU_MCR, and then accessing SCM as an ordinary RAM from the slave bus. This can be done only if both engines are halted or stopped.

Single-step execution

When microengine is already in halt_exec state, it can run the next microinstruction in the normal program flow and get back to halt state. PC is incremented, or assigned the BAF value in a branch with satisfied condition. Note that the executed instruction was already prefetched in the instruction pipeline, and a new microinstruction is fetched during its execution. The prefetched instruction may be cleared during halt state by the forced execution of a branch with flush (see *Section , Forced microinstruction execution*), making single-step execute a NOP instead of the next instruction in the program flow.

Single-step execution is controlled by the debug interface, and is a feature available from Nexus if eTPU is connected to the NDEDI block.The single-step execution of a NOP instruction can be useful to control input signal sampling and filtering, if signal ndedi_stop_pins = 1 at the Debug Interface. Single-step does not happen if VIS = 1.

Forced microinstruction execution

When microengine is already in halt state (either halt_idle or halt_exec), it can run forced microinstructions through the debug interface. This feature is available from Nexus if eTPU is connected to the NDEDI block. The microinstruction, specified by the user, is not fetched from SCM and comes directly from the debug interface. MDU start commands issued by forced instructions are executed, and the MDU runs the operation until the end, independently of the halt state. The microinstruction field END is ignored.

During forced execution of any instruction except Branches, Returns and Dispatches, the PC does not change, and the prefetched instruction in the pipeline is bypassed, but not discarded. When halt state is suspended, the prefetched instruction is executed and the instruction pointed by the PC is prefetched in parallel (two-stage pipeline).

Forced execution of a Branch, Dispatch or Return loads the PC with the BAF field (if branch condition is satisfied), PC+P or RAR, respectively. If branch condition is not satisfied, PC value stays unaltered. The flush control (field FLS) also works, so that a successful forced branch with flush replaces the prefetched instruction with a NOP. So, to clear the instruction pipeline during halt, all one has to do is an unconditional branch to the desired address with flush. HALT instructions must not be executed as forced.

Forced operations that depend on the serviced channel are unpredictable when executed in halt_idle.

Microengine register access

eTPU provides no direct access to microengine and channel registers from the slave bus or any other interface. However, these registers can be read and written in halt state by executing forced microinstructions (see *Section , Forced microinstruction execution*). Immediate data microinstructions may be used to set register values. Some registers are not selectable for immediate data destination, so intermediary register(s)—notably P—may have to be used to carry the desired new value to the target register in two or more microinstructions. Usually the previous values of intermediary register(s) must be previously saved and restored after the whole operation.

Similar procedures apply for register reads: their contents must be dumped to SPRAM, where they can be read from the slave bus.

Microengine flag access

Microengine halt state allows reading the branch conditions flags through forced microinstructions or, more easily, through the NDEDI register NDEDI_ENGINEx_CFSR. Flag conditions set by the user are seen by microengine for the next microinstruction execution. The flag set options are limited by the possibilities of forced microinstruction execution.

If the eTPU runs (not single-stepping) after exiting the halted state, the conditions modified during halt may remain only for the first microcycle after the halted state. After the first microcycle, branch conditions are altered only according to their regular update scheme.

Microengine stall

Microengine can get into a stall state, attending a request from a debug interface signal assertion. The reason for a Stall request from NDEDI (or from any other debug support block) should be a temporary lack of resources, for instance queue full. During stall the microengine suspends execution, but all the other engine logic continues operating: time bases, angle logic, channel logic, input sampling and filters. Stall differs from Halt, not enabling any of the debug features that Halt enables (see *Section , Microengine halt state*). It also does not break an atomic microengine access, unlike halt.

The Microengine can be stalled when idle and from the moment TST ends, before executing the first thread microinstruction, until just before the last thread microinstruction is executed. Stall requests are ignored in any other occasions. Microengines in a dual-engine system can be independently stalled. If a forced end is issued when microengine is in stall coming from execution, the END is executed only when the microengine resumes execution from stall.

SCM emulation

If SCM is implemented as ROM, an external RAM may be used to replace it, allowing code patching and software breakpoint setting for debugging purposes. SCM ROM replacement by Emulation RAM is MCU-dependent. The SCM may even be divided into a ROM part and a RAM part. In this case, both microengines can run code from both ROM and Emulation RAM. It is possible to make one engine run code from RAM and the other from ROM, by using different Entry Tables. The SCM visibility conditions also apply to Emulation RAM.

All SCM implementations, either RAM, ROM or Emulation RAM, are external to the eTPU block. eTPU provides a signal to enable the switching between external SCM banks. The conditions for this switching are:

1. Both engines stopped
2. VIS bit = 0

Note that these conditions also stop the clocks of the SCM interface and MISC logic.

Test support features

SCM Test – Multiple input signature calculator

The Multiple Input Signature Calculator (MISC) comprises special hardware that sequentially reads all SCM positions and calculates, in parallel, a 32-bit signature from a 32-input CRC signature calculator with the following polynomial:

$1 + x^{1} + x^{2} + x^{22} + x^{31}$

A complete description of the signature calculation procedure can be found in *Section 22.7.4, MISC algorithm*.

Once started by the Host the MISC runs continuously, restarting after the completion of each cycle, when it sets the ETPU_MCR flag SCMMISC (see *Section , ETPU_MCR – eTPU Module Configuration Register*). The average time for a MISC calculation can be measured by checking SCMMISC state at regular intervals, incrementing a counter and clearing SCMMISC if it is set.

MISC accesses to the SCM array are executed if none of the engines is accessing the SCM, to avoid degradation of the microengine performance: it happens while no channel is being serviced. An ongoing MISC operation can be aborted by writing 0 to SCMMISEN.

The Host must load the register ETPU_MISCCMPR (see *Section , ETPU_MISCCMPR – eTPU MISC Compare Register*) with the expected value to be found at the end of the MISC cycle, and then start the signature calculation writing bit SCMMISEN = 1 in register ETPU_MCR (see *Section , ETPU_MCR – eTPU Module Configuration Register*). MISC zeroes the signature accumulator and starts reading SCM data and calculating the signature. After last SCM position is read, MISC compares the value in signature accumulator against the value in ETPU_MISCCMPR: if there is a mismatch MISC stops, a Global Exception is issued and the bit SCMMISF in register ETPU_MCR assumes value 1. If no mismatch is found, MISC repeats the procedure automatically. When signature is being calculated, SCM address starts at the last SCM address and counts down to 0. The conditions for executing a MISC operation are (see also *Table 382*):

- Both microengines in idle state (no channel is being serviced) or stopped, in any combination (e.g., engine 1 idle with engine 2 stopped)
- ETPU_MCR bit VIS = 0
- ETPU_MCR bit SCMMISEN = 1

Note that MISC can run regardless of SCM implementation type (RAM or ROM).

If SCMMISEN = 0 or VIS = 1, the MISC logic stays at its initial state, with address counter pointing to the last SCM position and accumulator reset.

Performance monitoring features

Idle Counter

The Idle Counter Register ETPU_IDLE (see *Section , ETPU_IDLE – eTPU Idle Register*) continuously counts microcycles in which the microengine is not busy with channel service. It can be used to measure the microengine utilization by rating the count measured during a period of time to the number of microcycles contained in the period. The Idle counter does not count microcycles when the engine is stopped, or is in TST or halt states.

22.6 Initialization/Application information

22.6.1 Configuration sequence

After initial power-on reset the eTPU remains in an idle state[an], requiring initialization of several registers before any function can begin execution. Also, if the SCM is implemented in RAM, it should be initialized with the eTPU application code prior to configuring the eTPU. Configuration procedures are summarized as follows:

- If SCM is implemented as RAM, load the eTPU application code (see *Section , SCM access*).
- Initialize the SCM MISC logic (optional, see *Section , SCM Test – Multiple input signature calculator*).
- Initialize the eTPU time base configuration registers (ETPU_TBCR) to setup:
  - TCR1 and TCR2 prescalers and clock sources.
  - Select digital filtering mode.
  - TCRCLK signal filter control.
  - Angle mode operation (if necessary).
- Initialize the eTPU engine configuration register(s) (ETPU_ECR) to setup:
  - Entry table base.
  - Filter prescaler clock control.
- Initialize eTPU STAC configuration register(s) (ETPU_REDCR), if one needs to setup TCR1/2 resource Client/Server operation.
- Write to the Channel Configuration registers (ETPU_CxCR) to choose the Function to be performed by each channel, and its parameter base address.
- Write to channel status control register (ETPU_CxSCR) to choose among the possible variations within the function flow (FM bits).
- Write to SPRAM for parameter initialization of each configured channel.
- Write to register(s) ETPU_WDTR if one needs to enable and setup the Watchdog(s) mode and timeout.
- Write to channel x Host Service Request registers (ETPU_CxHSRR) to initialize the active channels.(ao)
- Write to the channel interrupt enable register (ETPU_CIER) if interrupts are to be enabled from the appropriate channels. Likewise for Data Transfer Requests (ETPU_CDTRER). This can also be done through ETPU_CxCR.
- Write to channel x configuration registers (ETPU_CxCR) to enable each channel by assigning it a high, middle, or low priority (CPR field).[ao]
- Monitor the Host service request registers (ETPU_CxHSRR) for completion of initialization.
- Write ETPU_MCR bit GTBE = 1 to start TCR1/TCR2 time base counting at same time in both engines (may be done before or never, depending on the particular application and use of Red Line bus).

See *Section 22.7.2, Initialization code example*.

[an] Except when device debug request is asserted on power-on reset: in this case, the microengines wake-up in halt state.

[ao] This operation is done before enabling active channels to avoid time events happening before the channel initialization.

22.6.2 Reset options

Hardware Reset

Hardware reset is achieved by assertion of device synchronous reset. Both engines and common logic is reset, and even the System Configuration and Global Channel registers assume their reset values.

Note: All eTPU input clocks must pulse during reset so that both engines are reset, even if they are in Module Disable or Stop mode.

Software reset

eTPU has no Software reset. To abort infinite microcode loops, the Force END mechanism must be used (see field FEND in *Section , ETPU_ECR – eTPU Engine Configuration Register*).

22.6.3 Multiple parameter coherency methods

Follows a description of two methods for coherent transfer of multiple parameters between Host and eTPU. Both methods involve the use of two parameter areas: the Transfer Parameter Area (hereafter called TPA), which is the SPRAM area directly accessed by the Host for reads and writes, and the Permanent Parameter Area (hereafter called PPA), which are the SPRAM positions where channel parameters are normally accessed by the Function microcode. Note that parameters in either TPA or PPA do not have to be in sequential addresses. TPAs and PPAs allocation are completely defined by the application, and there may be any number of them, independently of the channels.

The methods described here are not the only solutions for the coherent transfer problem, and both can co-exist in eTPU and even used in combination. Also note that for transfers of a pair of parameters, the Coherent Dual-parameter Controller is faster and have less impact on both eTPU and Host performance. That said, the methods are:

- Transfer Service

  A microengine thread transfers, upon Host Service Request, data from/to a TPA to/from a PPA. Coherency is guaranteed by the fact that a thread is atomic with respect to other threads in the same engine, and so are its transfers. If parameters in PPA are shared by both engines, hardware semaphores have to be used to access them.
- Mailbox

  For Host to eTPU transfers, the microcode checks a flag, set by the host, indicating the existence of new parameter data in the TPA. It can, then, either access TPA data directly or copy it to the PPA. For eTPU to Host transfers, when microcode changes PPA, it copies them to the TPA and flags updated TPA data to Host, possibly using an Interrupt or a Data Transfer Request. The Mailbox flag is reset when data is copied: by the eTPU microcode, when it transfers TPA to PPA (possibly followed by an Interrupt); by the Host, when it reads data from the TPA. This indicates that TPA is free for another transfer.

Transfer Service has the advantage of separating the task of data transfer from the functional service thread that accesses the parameters, with less impact to the latter. Compared to the Mailbox method, however, it has bigger average latency, because the Transfer Service thread has to contend for a time slot to execute. This latency can be minimized if Transfer Service thread is assigned to a separate channel with higher priority, but even so it does not guarantee that PPA is updated before the next execution of the functional thread that uses it.

Mailbox method, on the other hand, makes the functional thread check for the existence of new data (Host to eTPU). It does not have to be responsible for the transfer, though: it may access the TPA directly, and a Transfer Service can then be used to copy data from TPA to PPA.

22.6.4 Programming hints and caveats

Atomic dual access after a call, return

A dual, back-to-back parameter access is not atomic after a call, a jump, or a return if they occurred in parallel with an odd SPRAM access. It is safer to make a pair of parameter accesses that must be coherent begin at the second instruction after a call/jump/return.

Resource polling

The use of polling while waiting for a condition or a resource (except semaphore lock) should be avoided in order not to hang the microengine in long loops. This general programming guideline is greatly enforced in eTPU, as a thread cannot be preempted for any reason. Safer polling, albeit with long and indeterministic latency, can be obtained if one issues a channel link to itself and terminates the thread. The microengine is then free to other tasks, and the next poll happens at the next time the channel is serviced. This mechanism can be combined with finite (timed out) loops for better latency.

Changing channel function, parameter base, or entry table scheme

Channel Function, Parameter Base Address and Entry Table Scheme are determined by the ETPU_CxCR fields CFS, CPBA and ETCS. They cannot be changed when the channel is enabled. If the channel is disabled first, one may still have service requests from the previous function, so before the channel is enabled again one must be sure that:

- The first thread executed in the new function is the initialization one.
- The initialization thread of the new function clears any previously pending service request.

Follows a safe procedure for function changing:

1. Disable the channel (write ETPU_CxCR field CPR = 00).
2. Change the function configuration (ETPU_CxCR fields CFS and/or CPBA and/or ETCS).
3. Request the initialization thread, writing ETPU_CxHSRR with the initialization HSR (channel still disabled).
4. Enable the channel (write ETPU_CxCR field CPR > 0); the initialization HSR is serviced before any other formerly pending service requests, clearing them.

Checking and clearing interrupts of a stopped engine

An engine may be stopped with interrupts (or DMA requests) pending. This includes the case when the engine’s MDIS bit is set and a thread is still running: the thread will complete execution, possibly issuing an interrupt or DMA request before the engine stops, setting the STF bit.

As soon as the engine stops the channel registers become inaccessible, issuing bus errors when accessed. Interrupts and DMA requests can still be checked and cleared through the Global Channel Registers, though. DMA requests can also be cleared by the hardware handshaking with the DMA controller when the engine is stopped.

22.6.5 Estimating worst-case latency

Reliable systems are designed to work under worst-case conditions. This section explains how to estimate worst-case latency (WCL) for any eTPU function in any system. The appendix covers the following topics:

- Introduction to Worst-Case Latency
- Using Worst-Case Latency Estimates to Evaluate Performance
- Priority Scheme Details used in WCL Analyses
- First-Pass WCL Analysis
- Second-Pass WCL Analysis

The first-pass WCL analysis is based on a deterministic, generalized formula that is easy to apply. Because of the generalizations in the formula, the first analysis result is almost always much worse than the real worst case. If the desired system performance is within the limits of this first analysis, then no further analysis is required; the system is well within the performance limits of the eTPU. If the desired system performance exceeds that indicated by the first analysis, the second-pass WCL analysis should be applied. The second-pass analysis is not a generalized formula, but rather uses specific system details for a realistic worst-case estimation.

Introduction to worst-case latency

Note: In this Appendix the latency calculation and examples refer to old TPU functions such as PWM, DIO etc. These functions use single action channels which have single transition and single match functionality. They are not optimized for the eTPU hardware enhancement which support various double action modes. These examples are for reference only. New eTPU functions which are optimized for the new hardware will impose different latency calculations.

Worst-case latency for a channel is the longest amount of time that can elapse between the execution of any two function threads on that channel. For example, if in a particular system, channel 5 is running PWM, the worst-case latency for channel 5 is the longest possible time between the execution of two PWM threads. The worst-case time includes the time the execution unit takes to execute threads for other active channels, and other delays described later in this section. Refer to *Figure 430*.

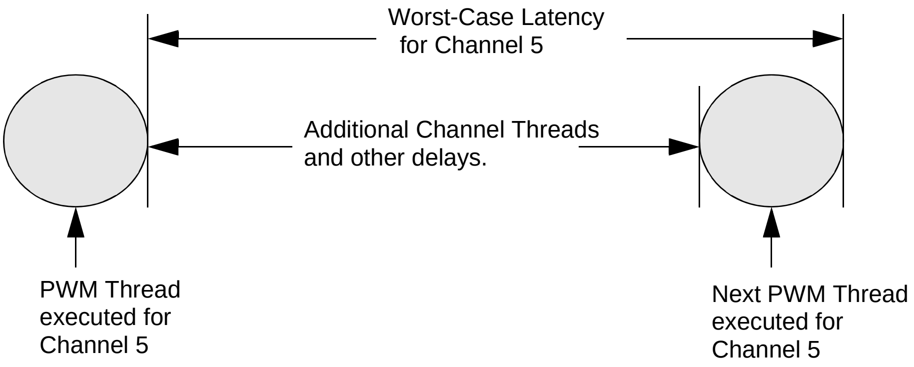


Figure 430. Worst-case latency for PWM

Worst-case latency for a channel depends both on the function running on that channel and on the activity on other channels. Since the 32 eTPU channels must all share the same execution unit, execution speed of a particular function varies with each system. The PWM thread response is faster if there are no other active channels than if other channels are also active. In addition, changing the priority scheme and channel number assignments can change performance for a function even if the same set of functions are still active.

Each function is divided into treads, as shown in *Figure 431* (see also *Section 22.5.1, Functions and threads*). The eTPU Microengine executes one thread of a function at a time. For example, the Microengine might execute thread 1 of PWM, then thread 3 of DIO, then thread 2 of PWM, then thread 2 of SM, and so on. The amount of time the eTPU Microengine grants a function to execute a thread varies with the number of microcode instructions in the thread.

Since there is only one eTPU Microengine (in each eTPU engine), the eTPU cannot actually execute the software for multiple functions simultaneously. However, the hardware for each of the channels is independent. This means that, for example, all 32 channel signals can change thread at the same moment, provided that the function software sets up the channel hardware to do so beforehand.

With Host CPU code, the system designer assigns functions to channels and initializes the functions. After initialization, functions typically run without Host intervention, except for eTPU channel interrupts to the Host to give or receive information. Most functions can run continuously with periodic servicing from the eTPU Microengine. As required, the channels request service from the eTPU Microengine, and the eTPU Scheduler determines the order in which the channels are serviced. Worst-case latency for a channel can be derived from the details of the priority scheme that the scheduler uses (see *Section 22.5.3, Scheduler*).

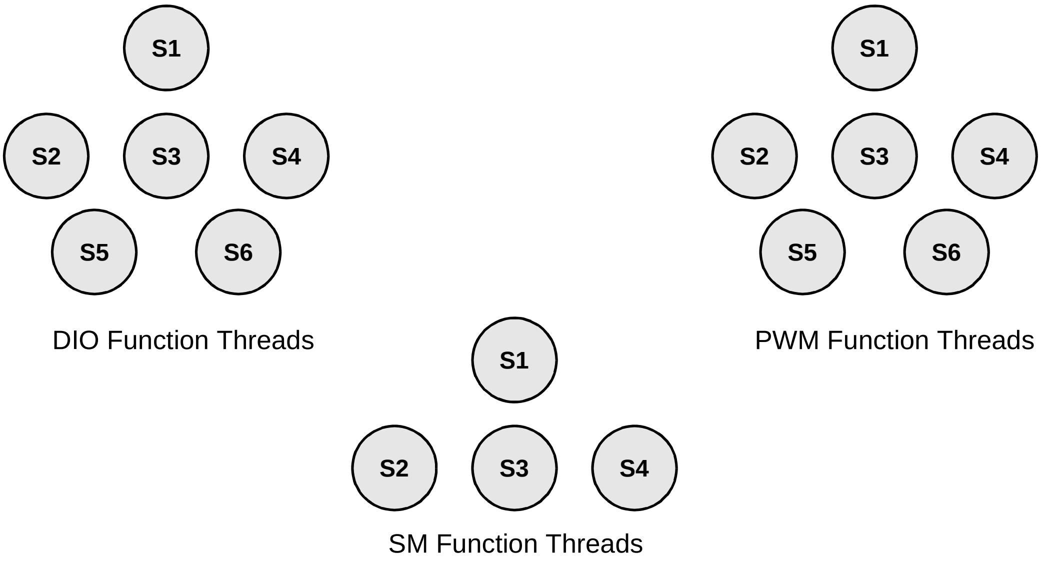


Figure 431. Function threads

Using worst-case latency estimates to evaluate performance

Once WCL is found for a channel, the user must determine how to use this number to analyze performance. To analyze the performance of a channel running the PWM function, for example, some information about what happens in each thread is necessary.

The following example refers to old TPU PWM function, which is not optimized to the eTPU enhanced hardware. For PWM, thread 1 is the initialization thread, and threads 2 and 3 are used during normal function execution. (PWM threads 4, 5, and 6 are for special modes and will be assumed to be unused on channel 5). Thread 2 writes a time into the channel 5 match register and performs other operations that will cause the channel 5 signal to go from low to high at the time indicated in the match register (match time). At match time, the signal goes high and channel 5 requests service from the eTPU Microengine to execute thread 3. Thread 3 writes a time into the channel 5 match register and performs other operations that will cause the channel 5 signal to go from high to low at match time. At match time, the signal goes low and channel 5 requests service from the eTPU Microengine to execute thread 2. A PWM wave is kept running on the system by the eTPU executing thread 2, then thread 3, then thread 2, then thread 3, and so on.

Since the definition of worse-case latency assumes a fully loaded running system, initialization threads are not part of worst-case calculations. For the channel 5 example, the two PWM threads in *Figure 430* are thus the two normal running threads, threads 2 and 3.

Figure 430 does not define which thread is thread 2 and which is thread 3. Since the worst-case latency derived from the first-pass analysis is the worst case between any two threads (not counting initialization threads), it is safe to say that the worst-case latency shown in *Figure 431* represents both the worst-case high time and the worst-case low time.

Notice in *Figure 430* that worst-case latency is drawn from the end of the execution of the first PWM thread to the end of the execution of the next PWM thread. It is drawn from end to end because the microcode instructions that make up the threads control the channel hardware. To make sure that all the microcode instructions needed to change the pin thread have been executed, it is necessary to include the execution time of the second thread.

Thread information for each function is found in the programming notes for individual TPU functions.

Priority scheme details used in WCL analysis

The user assigns functions to channel numbers and gives each active channel a priority level of high, middle, or low. The Scheduler uses the channel number and channel priority level to determine the order in which to grant service.

The scheduler allocates time slots to specific priority levels of high, middle, or low. One function thread is executed in each time slot. The length of a time slot varies according to the length of the executing thread. When fully loaded, the scheduler always assigns time slots in a seven-slot sequence (see *Figure 432*). After a seven-slot sequence is completed, another seven-slot sequence begins (see *Figure 433*). Note that in eTPU, when no service request exists, the scheduler goes to thread 1, but WCL calculation considers full load.

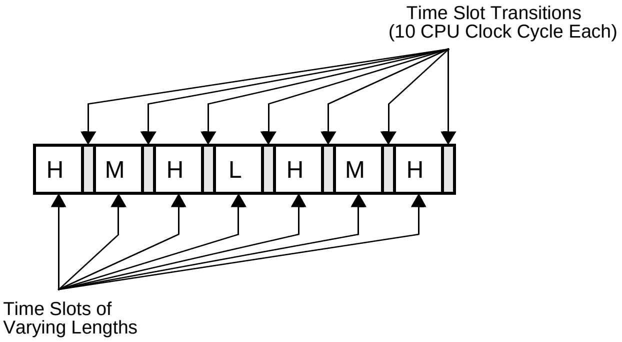


Figure 432. Time-slot sequence

This sequence scheme gives higher-priority channels more service time than lower-priority channels. High-priority channels are allocated four of seven time slots, middle-priority channels are allocated two of seven time slots, and low-priority channels are allocated one of seven time slots.

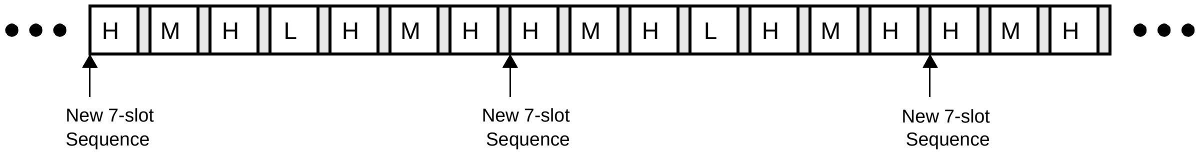


Figure 433. Multiple time-slot sequences

Priority passing

If no channel of the priority level assigned to the time slot is requesting service, the eTPU scheduler can pass priority to other levels. If no high-level channel is requesting service during a high level time slot, a middle-level channel is granted service; or, if no middle level-channel is requesting service, a low-level channel is granted service. If no middle-level channel is requesting service during a middle-level time slot, a high-level channel is granted service; or, if no high-level channel is requesting service, a low-level channel is granted service. If no low-level channel is requesting service during a low-level time slot, a high-level channel is granted service; or, if no high-level channel is requesting service, a middle-level channel is granted service. If no channel is requesting service, the time slot sequence is reset to state 1 and the scheduler idles until a request is received.

Priority passing is implemented in hardware and does not contribute to worst-case latency.

Time-slot transition

After each time slot, the eTPU must prepare for the next time slot. This preparation time between each time slot is called a time-slot transition. See *Section , Time slot transition*. Time-slot transitions can take from six up to ten system clocks.

Channel number priority

If more than one channel of a priority level is requesting service, the lowest numbered channel is granted service first. For example, if channels 0, 5, and 9 are all high-level channels requesting service during a high time slot, channel 0 is granted service first. Continuing this example, if channel 0 requests service again immediately after being serviced, it is not serviced again until channels 5 and 9 are serviced. This scheme is implemented so that continuously-requesting low numbered channels do not take all the time on the eTPU execution unit and leave no time for other channels.

The scheduler uses registers to keep track of which channels have been serviced and which require servicing. Each channel has two register bit: a service request register (SRR) and a service grant register (SGR). The SRR is set when a channel requests service. After the channel has been granted service, the SGR is set and the SRR is cleared.

SGRs are not cleared individually by channel, but rather as priority level groups. The clearing of a group of SGRs begins a new cycle for that priority level. An SGR group is cleared on the condition that a channel of that priority level has just been serviced, and no other channel of that priority level is requesting service (has a set SRR) and has not been granted service (has a clear SGR).

For example, if a middle-priority channel has just been serviced (either in a middle-priority time slot or a high or low-priority time slot gained by priority passing), the SRRs and SGRs of all middle-priority channels are compared. If there is no middle-priority channel with its SRR set and SGR cleared, the scheduler clears all middle-level SGRs. If there is a middle-level channel with its SRR set and SGR cleared, the scheduler does not clear the SGR group, and the requesting middle-level channel is serviced on the next middle-level time slot (or possibly sooner by priority passing).

SPRAM collision rate

Most function threads read or write to the eTPU SPRAM at least once. Because both the eTPU Microengine and Host can access the SPRAM but not at the same time, the Microengine may suspend execution during the SPRAM access while waiting for the Host to finish accessing the SPRAM. At other times the Host may wait for the Microengine. Wait states can take up to two system clocks, when the Host accesses the SPRAM directly, without using CDC. Microengine(s) wait-states must be added into the worst-case latency calculation. The system designer should estimate the percentage of SPRAM accesses in the system that will result in Microengine wait-states. This percentage is called the RAM collision rate (RCR). In each collision with direct Host accesses to the SPRAM the Microengine(s) wait for two system clocks.

In eTPU the Coherent Dual-parameter Controller (CDC) may also access the SPRAM for atomic transfers of two parameters. eTPU Microengine may wait on this operation (if it is in service time) until the transfer is complete. CDC always transfers two parameters, making four consecutive accesses (read, write, read, write) of one system clock each. The system designer should estimate the percentage of SPRAM accesses in the system that will result in a Microengine wait due to coherent transfer, and multiply it with the average number of system clocks the Microengine waits for each transfer. This percentage is called Coherent Parameter Collision Rate (CPCR).

In addition, Microengine to Microengine multiple parameter coherent communication, using the hardware semaphores, may hold one Microengine which waits to lock the semaphore while the other Microengine is holding it. This waiting is due to a software loop, not hardware wait-states. Note that single parameter access of one Microengine does not affect the timing of the other Microengine due to SPRAM time interlace. This implies that single parameter

Microengine to Microengine communication does not affect the performance. The Microengine which waits for the semaphore will loop until it is freed by the other Microengine. This time depends on the eTPU application. The system designer should estimate the percentage of Microengine to Microengine coherent parameter communication that will result in eTPU semaphore loops, and multiply it with the average number of system clocks the Microengine loops for each such transfer. This percentage is called CCR (Communication Collision Rate).

A 100% collision rate for a system is the theoretical worst case. In many systems, however, the RCR, CPCR and CCR would be very low, sometimes even near 0%. This is because the eTPU is an independent processor capable of servicing most function needs, so that the Host rarely needs to access the eTPU parameter RAM. Also coherent Microengine to Microengine communication of more than one parameter may be rare. To find a realistic RCR, CPCR the system designer should evaluate the Host code and find the percentage of time it accesses the eTPU parameter RAM with or without using the CDC. This percentage gives a good RCR and CPCR. The eTPU application provides a good estimation of CCR.

Note: The programming practice of polling a flag in the eTPU SPRAM causes a very high RCR and should be avoided in high-performance systems.

After the collision rate for a system is found, it can be applied to the WCL calculations for each channel. The system designer can use the collision percentage and the number of SPRAM accesses (with and without semaphores) to estimate the eTPU loop time for a function. Note that in old TPU functions CPCR and CCR are both zero.

The estimation of eTPU wait time is as follows:

Variables:

N1 = Number of simple RAM accesses in the longest thread

RCRWait = Maximal system clocks wait time for simple RAM collision = 2

CPCRWait = Average System Clocks for Coherent Parameter Transfer (using CDC).

N2 = Number of eTPU-eTPU semaphore RAM accesses in the longest thread

CCRWait = Average System Clocks for Microengine-Microengine communication transfer.

Estimated Wait Time:

Function eTPU maximal wait time =

N1 * (RCR * RCRWait + CPCR * CPCRWait) + N2 * CCR * CCRWait

First-pass worst-case latency analysis

Following is the first-pass calculation of worst-case latency for a channel. Remember that this analysis uses generalizations that usually produce a result much worse than the real worst case. If the worst-case result from the first analysis is too long for the desired performance, use the second analysis for a more realistic worst-case analysis.

Worst-case assumptions and formula

To estimate worst-case latency for a channel, assume this worst-case condition: the channel has just been serviced in a time slot of its priority level, and all other channels in the system are continuously requesting service and have cleared SGRs. The worst-case latency is the time from the end of the channel's service until the end of the channel's next service. See *Figure 434*.

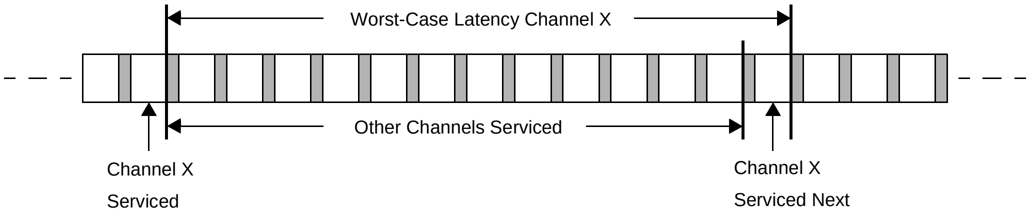


Figure 434. First-pass worst-case latency

To estimate worst-case latency:

- Find the worst-case service time for each active channel.
- Using the H-M-H-L-H-M-H time-slot sequence, map the channels that are granted for each time slot.
- Add time for six-clock time-slot transitions.

Finding the worst-case service time for each active channel

A table for eTPU functions should list the longest threads (not counting initialization threads) for the functions, and the number of eTPU SPRAM accesses in the longest thread (semaphored and non semaphored). These figures will be used for estimating Microengine wait time. *Table 464* is an example for old TPU functions in which there are only simple parameter RAM accesses. It does not take into consideration the CDC operation and Microengine to Microengine communication.

The worst-case service time for each channel is: (CPCR = CCR = 0)

Longest thread + ((number of RAM accesses in longest thread+1) * RCR * 2 clocks).

Note that the formula adds 1 RAM accesses for the parameter preload that occurs during TST. There are actually three accesses during TST, but only the first one can receive wait-states.

Table 464. Longest threads and RAM accesses for old TPU functions

Function	Longest thread	RAM accesses
DIO	10	4
ITC	40 (no linking) 42 (linking)	7
OC	40	7
PWM	24	4
SPWM Mode 0 Mode1 Mode 2	14 18 20 (no linking) 22 (linking	4 4 4 4
PMA	94	8
PMM	94	8
PSP Angle-Angle Mode Angle-Time Mode	76 50	6 3
SM(1)	160	21
PPWA Mode 0 Mode 1 Mode 2 Mode 3	44 50(2) 44 50	9 10 9 10

1. Assumes one master and one slave. For each additional slave
   a) Add 32 clocks and 2 RAM accesses, and
   b) Add (STEP_RATE_CNT * two clocks)
2. With one channel linked. Add two clocks for each additional channel linked.

Mapping the channels for each time slot

To determine when a channel will be serviced again, it is necessary to determine which other channels will be serviced first. Do this by assuming all channels are continuously requesting service and mapping the channels into the time-slot sequence.

Adding time for time-slot transitions

Add six system clocks for time-slot transitions which occur after each time slot.

First-pass analysis worst-case latency examples

The examples in this section assume the system configuration shown in *Table 465*.

Table 465. System configuration example

Channel	Priority	Function(1), (2)
0	High	PWM (driving a DC motor)
1	Middle	PPWA (Mode 0, measuring the DC motor speed)
2	Low	DIO (Input)

1. 9% RAM Collision Rate (RCR)
2. CPU clock rate = 40 MHz, or 25 ns per clock period

Finding the WCL for PWM on Channel 0

The following shows how to find the WCL for PWM on channel 0.

1. Find the worst-case service time for each active channel.
   a) Longest thread of PWM is 24 CPU clocks with four RAM accesses.

      24 + ((4 RAM accesses+1) * 0.09 * 2 CPU clock waits) = 24.9 CPU clocks, rounded up to 25 CPU clocks (since there are no partial clock periods)

      Channel 0 worst-case service time = 25 CPU clocks.

   b) Longest thread of PPWA in mode 0 is 44 CPU clocks with nine RAM accesses.

      44 + ((9 RAM accesses+1) * 0.09 * 2 CPU clock waits) = 45.8 CPU clocks, rounded up to 46 CPU clocks

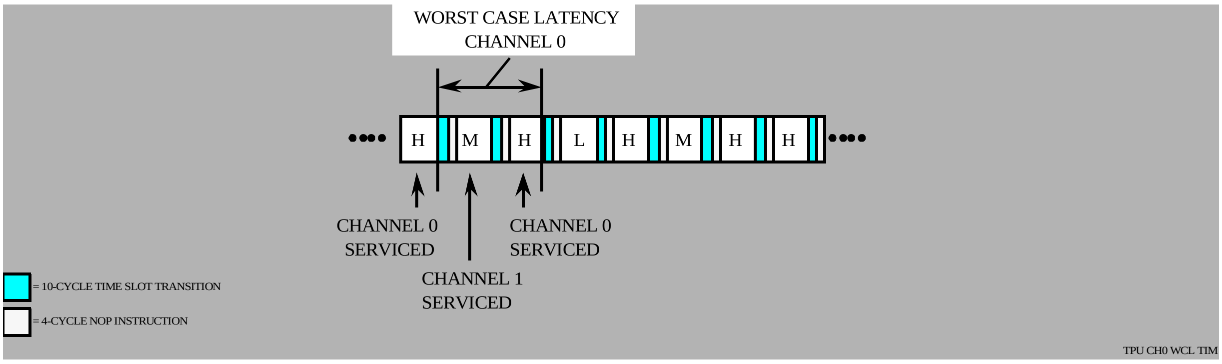
      Channel 1 worst-case service time = 46 CPU clocks.

   c) Longest thread of DIO is ten CPU clocks with four RAM accesses.

      10 + ((4 RAM accesses+1) * 0.09 * 2 CPU clock waits) = 10.9 CPU clocks, rounded up to 11 CPU clocks

      Channel 2 worst-case service time = 11 CPU clocks.
2. Assume channel 0 has just been serviced and that channels 1 and 2 are continuously requesting service. Using the H-M-H-L-H-M-H time-slot sequence, map the channels that are granted for each time slot. See *Figure 435*.

Figure 435. Next Servicing for Channel 0



Channel 1 will be serviced in the middle-priority time slot before channel 0 is serviced again.

3. Add time for the six-clock CPU time-slot transitions. See *Figure 435* and *Table 466*.

   A four-clock NOP occurs after each channel is serviced since there is one channel in each priority level, i.e., a new cycle for a priority level is started after each channel is serviced. Time-slot transitions occur after each time slot.

Table 466. Worst-case latency for channel 0

Channel 0 worst-case service time	25 clocks
Channel 1 worst-case service time	46 clocks
Two 6-clock time-slot transitions	12 clocks
Total clocks	83 clocks

83 clocks * 25 ns/clock = 2075 ns

Conclusion: in this system configuration PWM can run with a minimum high time or low time of 2075 ns.

Note that in double match eTPU system the PWM can be serviced once in each period, and there is no latency for minimum high time. The latency in eTPU PWM function will represent the minimum PWM period.

Finding the WCL for PPWA on channel 1

The following shows how to find the WCL for PPWA on channel 1.

1. Find the worst-case service time for each active channel. See step 1 of previous example.
2. Assume channel 1 has just been serviced and that channels 0 and 2 are continuously requesting service. Using the H-M-H-L-H-M-H time-slot sequence, map the channels that are granted for each time slot. See *Figure 436*.

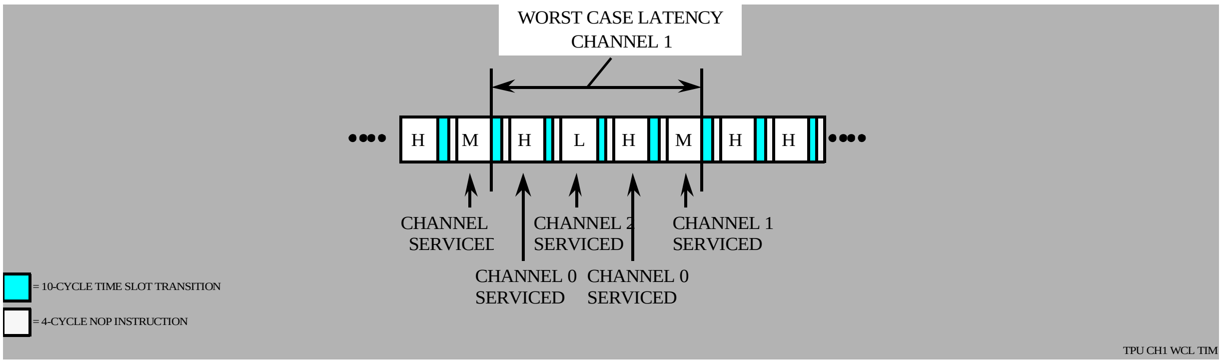


Figure 436. Next servicing for channel 1

Channel 0 will be serviced twice and channel 2 once before channel 1 is serviced again.

3. Add time for the six-clock CPU time-slot transitions. See *Figure 436* and *Table 467*.

Table 467. Worst-case latency for channel 1

Two Channel 0 worst-case service times	50 clocks
Channel 1 worst-case service time	46 clocks
Channel 2 worst-case service time	11 clocks
Four 6-clock time-slot transitions	24 clocks
Total clocks	131 clocks

131 clocks * 25 ns/clock = 3275 ns

Conclusion: In this system configuration PPWA can measure a period or pulse of minimum 3275 ns.

Note that PPWA function optimized for eTPU hardware can use double transition mode to measure very narrow pulses with one service after the second transition, and latency will affect only the minimum gap between two input pulses. Also the function threads would have more efficient coding.

Finding the WCL for DIO on Channel 2

The following shows how to find the WCL for DIO on channel 2.

1. Find the worst-case service time for each active channel. See step 1 of previous examples.
2. Assume channel 2 has just been serviced and that channels 0 and 1 are continuously requesting service. Using the H-M-H-L-H-M-H time-slot sequence, map the channels that are granted for each time slot. See *Figure 437*.

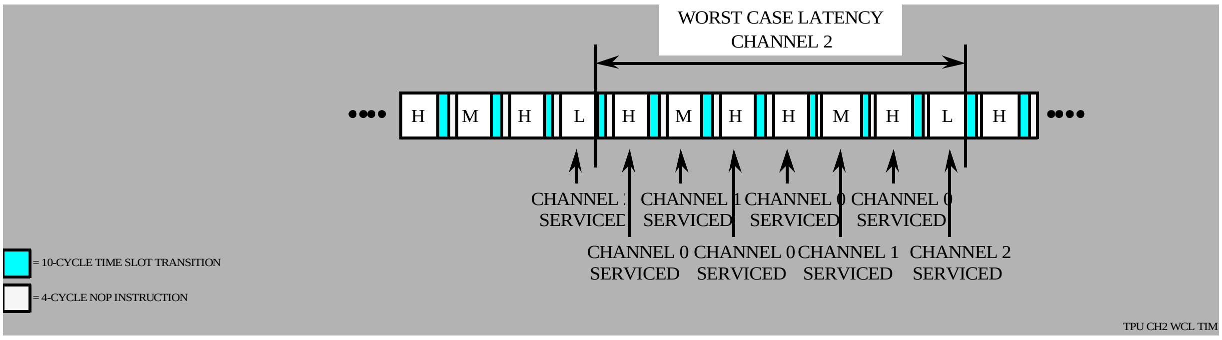


Figure 437. Next Servicing for Channel 2

Channel 0 will be serviced four times and channel 1 twice before channel 2 is serviced again.

3. Add time for the ten-clock CPU time-slot transitions and the four-clock NOPs. See *Figure 437* and *Table 468*.

Table 468. Worst-case latency for channel 2

Four Channel 0 worst-case service times	100 clocks
Two Channel 1 worst-case service time	92 clocks
Channel 2 worst-case service time	11 clocks
Seven 6-clock time-slot transitions	42 clocks
Total clocks	245 clocks

245 clocks * 25 ns/clock = 6125 ns

Conclusion: in this system configuration DIO can keep track of the input level at a minimum of every 6125 ns.

Note that DIO function optimized for eTPU hardware can use double transition mode to measure two pin transitions at a time and reduce the service time, improving the overall system performance and latency.

Second-pass worst-case latency analysis

Following is an example of a second-pass analysis for calculating worst-case latency for a channel. The second-pass analysis is useful for higher-performance systems, since it gives a more realistic worst-case latency result than first-pass analysis.

This example uses a relatively simple system in order to illustrate the basic principles of second-pass analysis. For a more complex example of second-pass analysis, refer to Multiphase Motor Commutation TPU Function (COMM)(TPUPN09/ D).

Second-pass analysis guidelines

Rather than use a fixed formula, a second-pass analysis relies on the application of the following guidelines.

1. The first-pass analysis makes the assumption that all channels in the system are continually requesting service. For many systems this is an unrealistic assumption. For example, if TCR1 is counting at a rate of 2 MHz (500 ns per count) and a channel is running the DIO function with a match rate of 20,000 TCR1 counts, the DIO will request

service every 10 ms (20,000 * 500 ns = 10,000,000 ns or 10 ms). It is therefore unrealistic to assume that the channel running this DIO function is continuously requesting service. Figure out a realistic service request rate for each channel. Time slots can then be mapped to each channel at the real rate of request.

2. If a function is active during system initialization but not during the high-speed running mode of the system, then that system does not need to be included in the high-speed worst-case latency calculations.
3. Use a realistic SPRAM collision rate.
4. Be careful when assigning functions priority levels and channel numbers. Decide which function or functions will be most difficult to make perform at the desired level. Assign those channels high priority and low channel numbers. Try different priority and channel assignments to see how it affects the system.
5. The seven-slot sequence of || H | M | H | L | H | M | H || is asymmetrical when put back-to-back with other seven-slot sequences. Note that in the following sequence there are two high-priority slots next to each other:

   || H | M | H | L | H | M | H |||| H | M | H | L | H | M | H ||
6. Make sure that when mapping out channels to the sequence, you choose a worst-case slot to start the mapping. For example, when estimating WCL for a high-priority channel, do not start the mapping in the last high-priority slot in a seven-slot sequence, as that is a best case for a high-priority channel since another high-priority time slot is next.
7. Instead of always using the longest thread in the function as the worst-case thread, evaluate the threads in the function that will be used in the system and use the appropriate worst-case threads. For example, in the preceding example of first-pass analysis, the PWM was shown to be able to achieve a high time and low time of 2475 ns under worst-case conditions. This was derived using the longest PWM thread of 24 CPU clocks. This longest thread is actually thread 2, the thread that is entered after the pin has just gone high. Thread 3, the thread that is entered after the pin has just gone low, requires only two CPU clocks. Therefore, in the first-pass example, the high time was correctly derived, but the low time is actually shorter than was estimated.

Second-Pass analysis example

This example requires three 50% PWM waveforms: one 5 kHz (200 ms/period) and two 50 kHz (20 ms/period), each running DC motors. (Remember that the PWM function requests service from the eTPU after each high time and after each low time, so the eTPU must handle a request every 100 ms for the 5 kHz PWM and every 10 ms for the 50 MHz PWM.)

Note: *This example uses square waves for simplicity. Notice that to use a PWM waveform in the typical way, in which the pulse is modulated, the pulse must not be modulated in a way that violates the worst-case latency requirements.*

This example also uses one DIO channel monitoring a signal level every millisecond and one PPWA channel in mode 0 monitoring the speed of the 5-kHz DC motor. The PPWA must measure periods of 5 kHz (200 ms/period).

The CPU is interrupted by the channel running the PPWA function after measuring 200 periods (every 40 ms). The interrupt service routine performs an averaging of the period accumulation and checks it against a known parameter. The interrupt service time is so short and infrequent that it is a tiny fraction of total system time. The interrupt service routine contains no polling of the parameter RAM. Therefore a realistic RCR = 0%.

First-Try system configuration

Try a system configuration that seems likely to work. If it does not, change priority levels or channel numbers.

The 5 kHz and 50 kHz PWMs are the most time-critical functions. Those are assigned high priority. PPWA is assigned middle priority. The DIO is low performance and is assigned low priority. Refer to *Table 469*.

Table 469. First-Try system configuration

Channel	Priority	Function[1],[2]
0	High	PWM at 50 kHz (needs a 4-µs WCL)
1	High	PWM at 50 kHz (needs a 4-µs WCL)
2	High	PWM at 5 kHz (needs a 40-µs WCL)
8	Middle	PPWA at 5 kHz (needs an 80-µs WCL)
15	Low	DIO as input at rate of 1 ms

1. 0% RAM collision rate
2. CPU clock rate = 40 MHz, or 60 ns per clock period

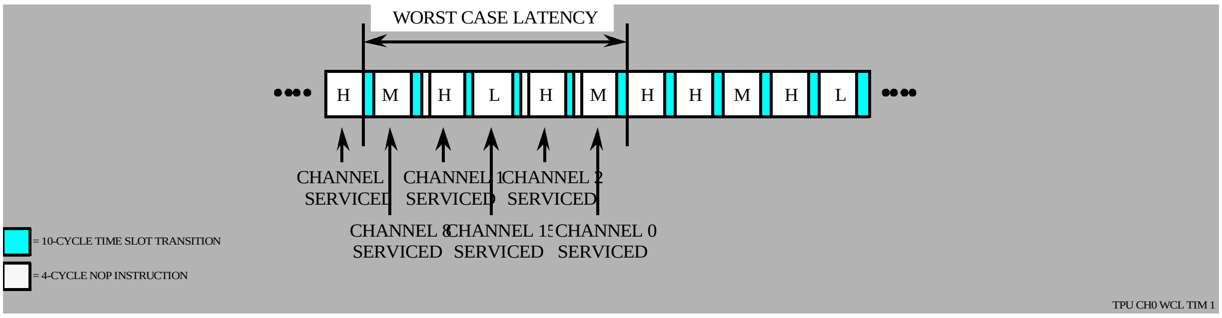
With this system configuration, worst-case service time for each active channel is determined as follows:

a) Longest thread of PWM is 24 CPU clocks with four RAM accesses.
24 + ((4 RAM accesses+1) * 0 * 2 CPU clock waits) = 24 CPU clocks
Channels 0-2 worst-case service time = 24 CPU clocks.

b) Longest thread of PPWA in mode 0 is 44 CPU clocks with nine RAM accesses.
44 + ((9 RAM accesses +1)* 0 * 2 CPU clock waits) = 44 CPU clocks
Channel 8 worst-case service time = 44 CPU clocks.

c) Longest thread of DIO is ten CPU clocks with four RAM accesses.
10 + ((4 RAM accesses+1) * 0 * 2 CPU clock waits) = 10 CPU clocks
Channel 15 worst-case service time = 10 CPU clocks.
To find the WCL for channel 0, assume channel 0 has just finished service.
Map the channels in the H-M-H-L-H-M-H sequence. See *Figure 438*.

Figure 438. Worst-case latency for channel 0 (first try)



Conclusion: with this system configuration, worst-case latencies for channels 0 and 1 are too high (WCL for channel 1 is the same as WCL for channel 0). Try a different system configuration.

Second-Try system configuration

The second-try system configuration is shown in *Table 470*.

Table 470. Second-Try system configuration

Channel	Priority	Function(1),(2)
0	High	PWM at 50 kHz (needs a 4-µs WCL)
1	High	PWM at 50 kHz (needs a 4-µs WCL)
2	Middle	PWM at 5 kHz (needs a 40-µs WCL)
8	Middle	PPWA at 5 kHz (needs an 80-µs WCL)
15	Low	DIO as input at rate of 1 ms

1. 0% RAM collision rate
2. CPU clock rate = 40 MHz, or 60 ns per clock period

To find the WCL for channel 0, assume channel 0 has just finished service. Map the channels in the H-M-H-L-H-M-H sequence. See *Figure 439*.

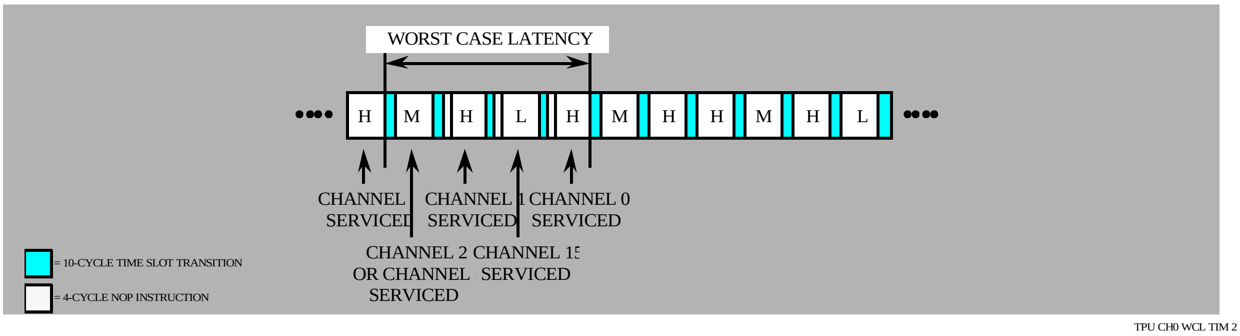


Figure 439. Worst-case latency for channel 0 (second try)

Conclusion: with this system configuration, the WCL of both channel 0 and channel 1 is 3.85 ms, which is within the limit of 4 ms needed for a 50-kHz PWM.

Next, find the WCL for channel 2. Assume channel 2 has just finished service. Map the channels in the H-M-H-L-H-M-H sequence. See *Figure 440*.

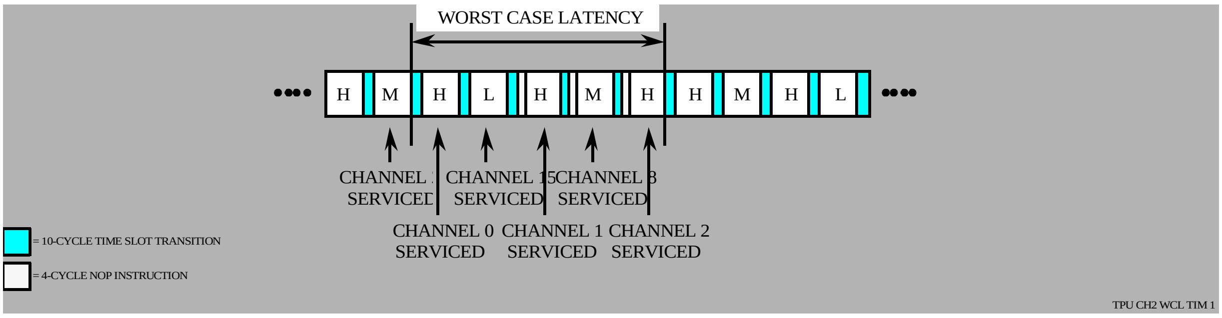


Figure 440. Worst-case latency for channel 2

Conclusion: with this system configuration, the WCL for channels 2 and 8 is 4.7 ms, which is within the 40 and 80 ms WCL requirements.

Notice that channels 2 and 8 are well within their WCL requirements. The system could be reconfigured as shown in *Table 471* to give channels 0 and 1 a larger margin while still keeping channels 2, 8 and 15 within their WCL requirements.

Table 471. Second-try system with channel 0 and 1 reconfigured

Channel	Priority	Function[(1),(2)]
0	High	PWM at 50 kHz (needs a 10-µs WCL)
1	High	PWM at 50 kHz (needs a 10-µs WCL)
2	Middle	PWM at 5 kHz (needs a 40-µs WCL)
8	Low	PPWA at 5 kHz (needs an 80-µs WCL)
15	Low	DIO as input at rate of 1 ms

1. 0% RAM collision rate
2. CPU clock rate = 40 MHz, or 60 ns per clock period

22.6.6 Endianness

The address of the 24-bit parameters and the most significant byte depends on the endianness of the MCU. *Table 472* shows the parameter addresses for big and little endian machines.

Table 472. Parameter addresses and endianness

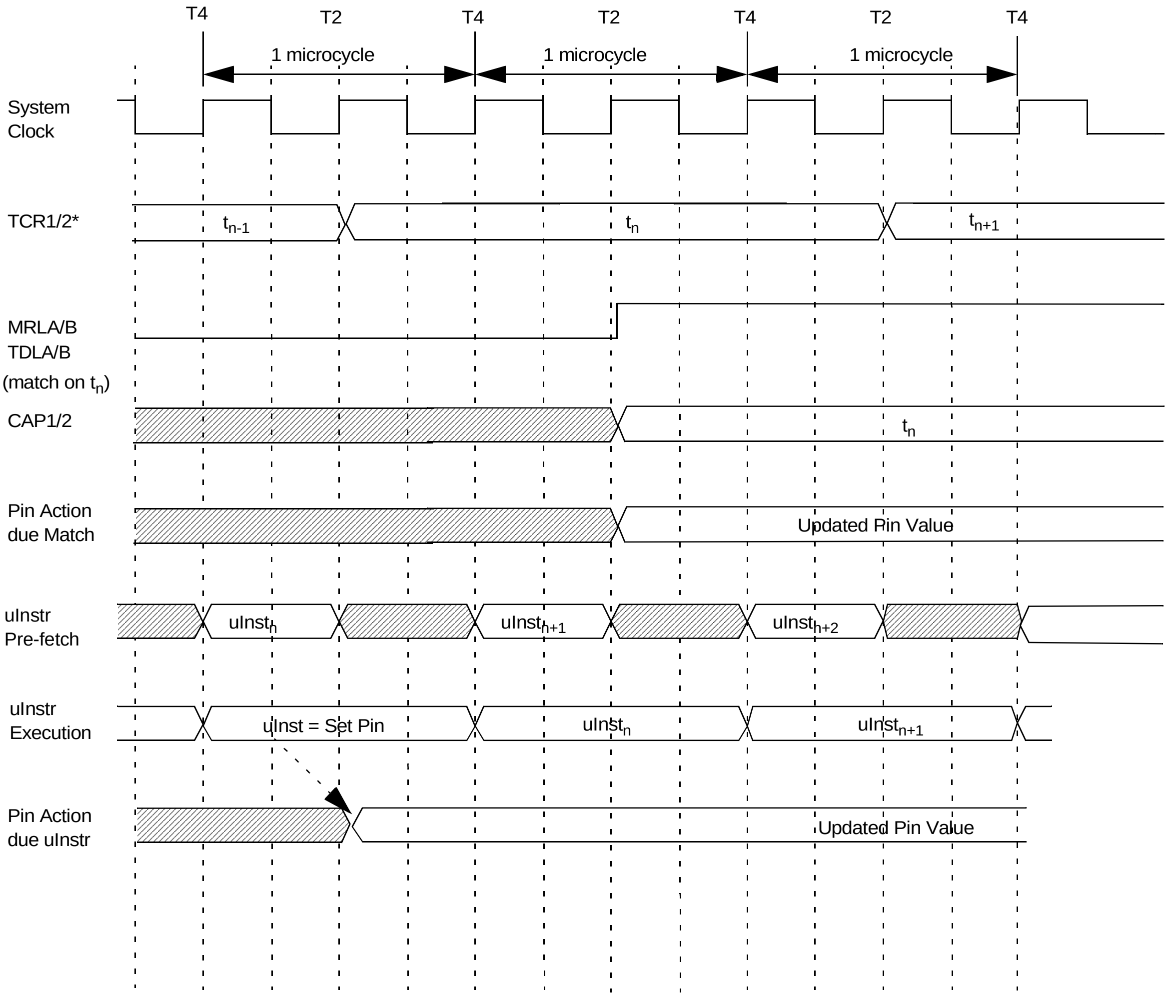
Parameter	Byte address offset (n = word address offset)	
	Big endian	Little endian
32-bit	4*n	
24-bit	4*n + 1	4*n
32-bit parameter’s most significant byte	4*n	4*n + 3
24-bit parameter’s most significant byte	4*n + 1	4*n + 2
Least significant byte	4*n + 3	4*n

22.7 Appendices

22.7.1 Microcycle and I/O timing

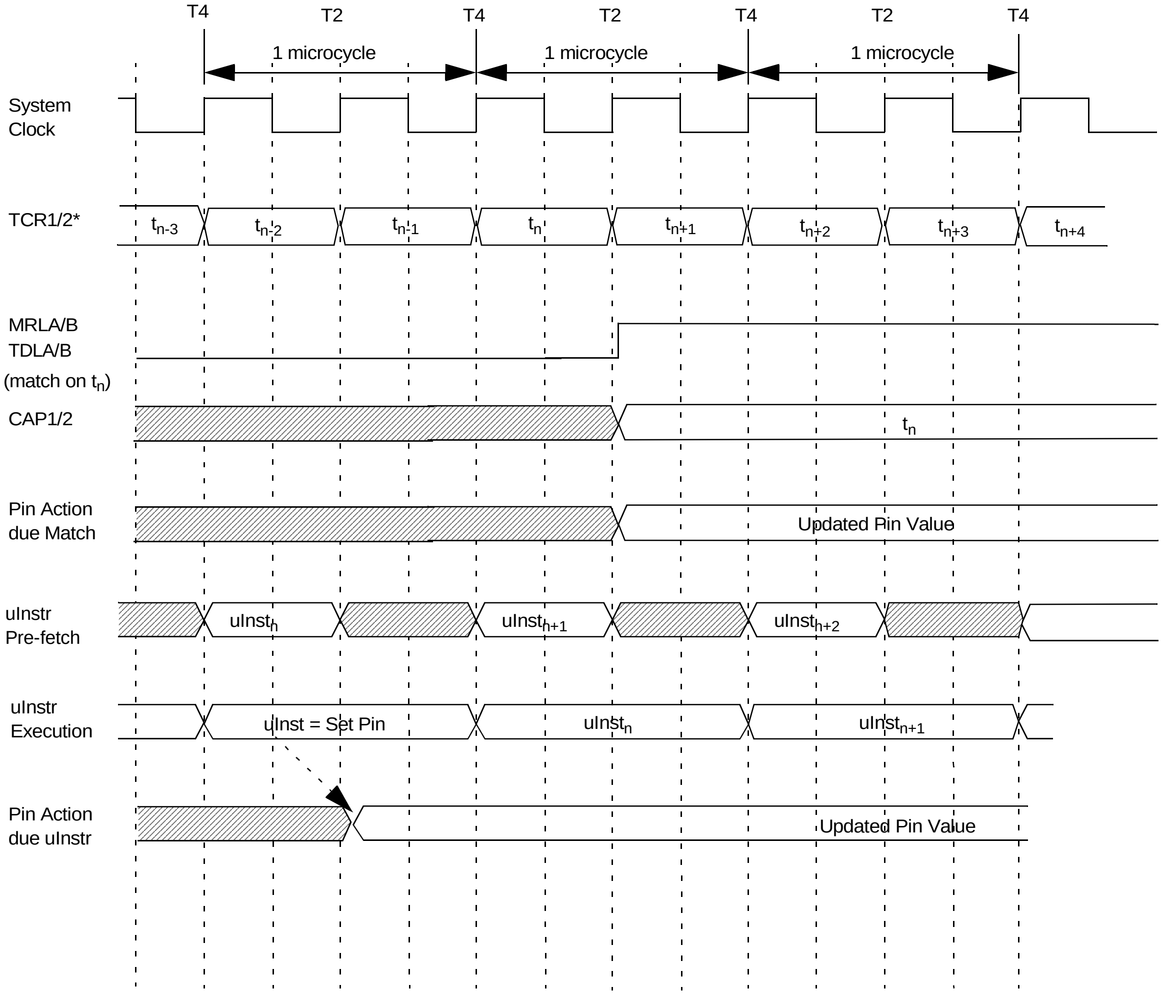
Execution and channel timing

Figure 441 shows the main timings related to microinstruction execution when channels and timebases run on T2 timing.



Note: *TCR clock/prescaler selection = 4x system clock

Figure 441. Execution, Timebase and Channel T2 Timing



Note: *TCR clock/prescaler selection = 1 x system clock

Figure 442. Execution, Timebase and Channel T2/T4 Timing

The sequential occurrence of the four T states (T1 – T4) constitutes a microcycle. Only T2 and T4 are taken as reference for timings, either internal or external. T2 and T4 have the timing of the positive system clock pulses, and are used in most of the eTPU logic in an edge-triggered design style.

Two additional T states are derived from the system clocks: $\overline{\text{T2}}$ and $\overline{\text{T4}}$. $\overline{\text{T2}}$ occurs when the eTPU loses SPRAM arbitration to a bus master. $\overline{\text{T4}}$ occurs in halt state (due to a breakpoint or device debug request assertion, for instance), or in stall state (due to an NDEDI queue full); see *Section , Development support features*, for more details.

$\overline{\text{T2}}$ and $\overline{\text{T4}}$ states are defined as microcycle timing states (not to be confounded with logic states) of one system clock in which the T clocks continue to run, but the control signals associated with the clocks are unaffected. That is, no operation occurs during these states. Both $\overline{\text{T2}}$ and $\overline{\text{T4}}$ states occur in multiples of two system clocks to keep the microengine synchronized with the free running channels, which are unaffected and keep on working as in T2 and T4.

Thus, the eTPU has two types of timing states:

- $\overline{T2}$: hold execution for SPRAM access, from clock pulse T2 until one of the next T2 clock pulses of another microcycle.
- $\overline{T4}$: hold execution in debug mode or stall, from clock pulse T4 until one of the next T4 clock pulses of another microcycle.

Figure 443 and *Figure 444* shows the timing of $\overline{T2}$ and $\overline{T4}$ timing states, respectively.

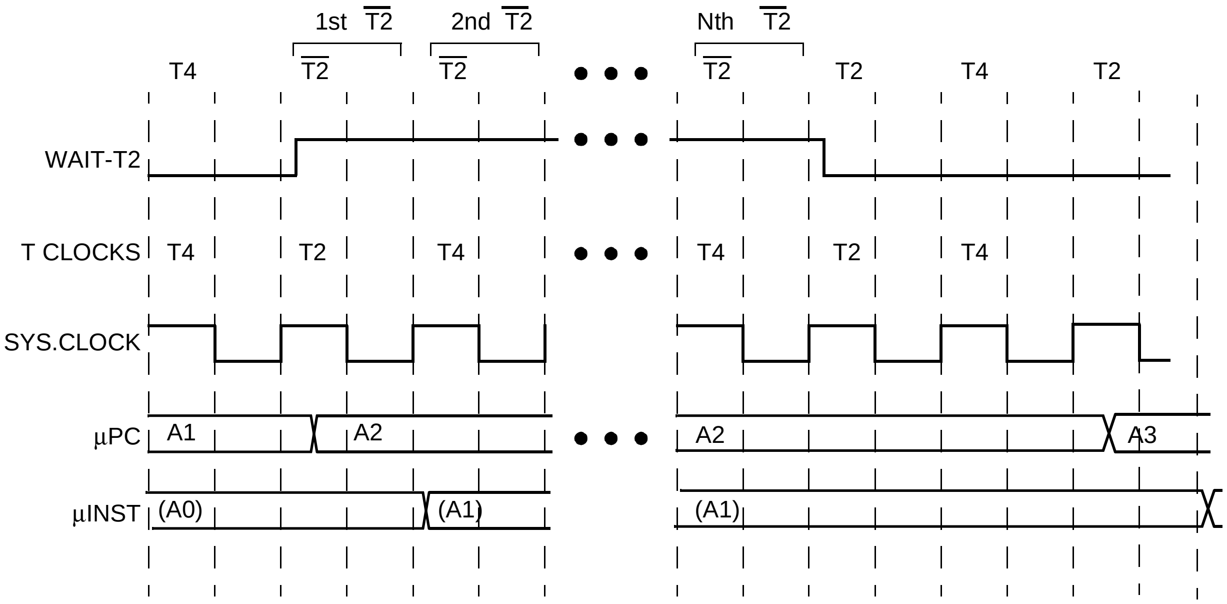


Figure 443. $\overline{T2}$ timing

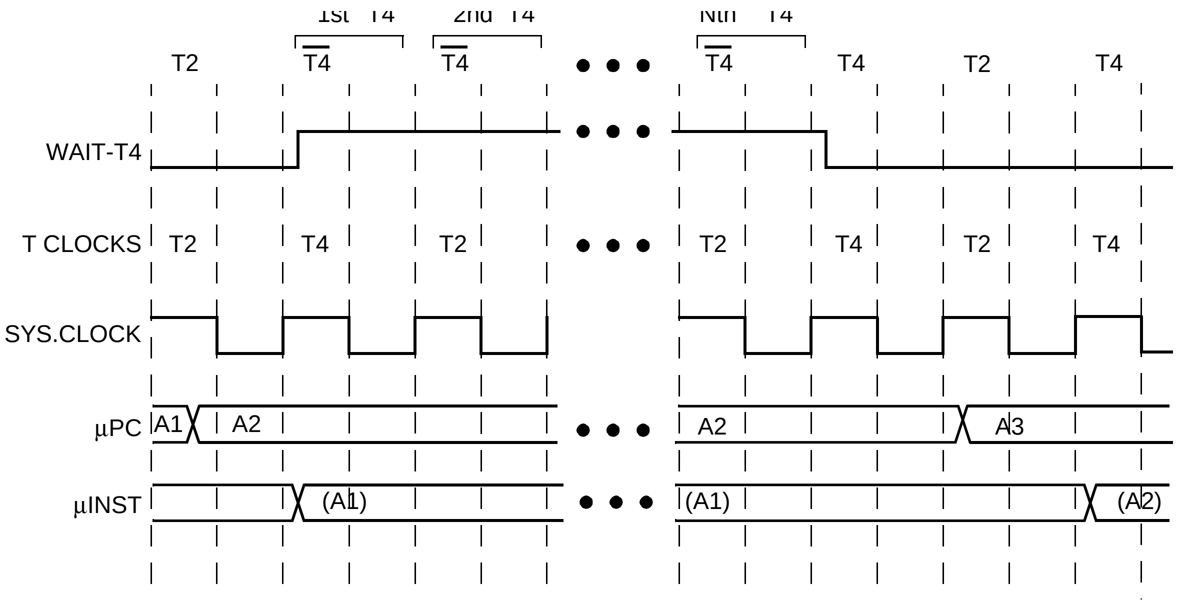


Figure 444. $\overline{T4}$ timing

Input/Output signal delays

The synchronizer, filter and edge detection logic delay the input signal transitions.

The Filter Delay varies with the filter clock (ETPU_ECR field FPSCK) and the filter mode used, as shown in the *Table 402*. For any given transition, it depends on the phase of the filter clock when the input transition happens. In integration mode (TCRCLK filtering only), it also depends on the state of the integrator counter. The Total Delay is defined as the number of system clock rising edges between the input transition and the setting of TDLA/B, TCR1/2 incrementing, or EAC tooth sensing (TCRCLK) in angle mode. The synchronizer delay is 2 or 3 system clocks, depending on the phase of the synchronizer when the input transition happens. The edge detection takes 1 more system clock. The total delays are, thus:

```
Min. Total Delay = Min. Synchronizer Delay + Min. Filter Delay + Edge
Detection Delay
Min. Total Delay = 3 + Min. Filter Delay

Max. Total Delay = Max. Synchronizer Delay + Max. Filter Delay + Edge
Detection Delay
Max. Total Delay = 4 + Max. Filter Delay
```

The channel filters can be bypassed, so nullifying the filter delays in the equations above.

The channel output flip-flops drive the eTPU output signals directly, without any synchronous delays. Consult the MCU Reference Manual for information on additional delays added at the integration.

22.7.2 Initialization code example

The code example below initializes ETPU_1 engine and configures eTPU UART FUNCTION to perform the receiver at channel 1 and the transmitter at channel 0. The function works without parity and the data word is 8 bits in size. The initialization code assumes the microcode function previously loaded into SCM.

```
**************************************************************************
*********
// Initilization program for eTPU engine 1, function microcode previously
loaded into SCM.
// No angle mode, eTPU UART FUNCTION configured to perform at channels 0 and
1.
// Channel0 - Tx_UART
// Channel1 - Rx_UART

// UART Specifications:
// Data word size: 8 bits
// Parity: disabled

// ***************************** Definitions
***********************************

//Bases
#define ETPU_BASE       0x000 //MCU-dependent
#define SPRAM_BASE      0x000           //MCU-dependent

//General Configuration Registers
#define ETPU_MCR_OFFSET        0x000     //Module Configuration Register
```

```
#define ETPU_TBCR_1_OFFSET     0x020     //Time Base Configuration Register
#define ETPU_ECR_1_OFFSET      0x014     //Engine Configuration Register
#define ETPU_CIER_1_OFFSET    0x240    //Channel Interrupt Enable Register
#define ETPU_CDTRER_1_OFFSET   0x250     //Data TransF Interrupt Enable
Register

//channel0 configuration registers
#define ETPU_C0CR_1_OFFSET     0x400     //Channel0 Configuration Register
#define ETPU_C0SCR_1_OFFSET    0x404     //Channel0 Status Control Register
#define ETPU_C0HSRR_1_OFFSET   0x408    //Channel0 Host Service Req. Register

//channel1 configuration registers
#define ETPU_C1CR_1_OFFSET     0x410     //Channel1 Configuration Register
#define ETPU_C1SCR_1_OFFSET    0x414     //Channel1 Status Control Register
#define ETPU_C1HSRR_1_OFFSET   0x418     //Channel1 Status Control Register

// Tx_UART SPRAM parameters
#define MATCH_RATE_TX_OFFSET   0x004    //Channel0 parameter 1
#define DATA_UART_TX_OFFSET    0x008    //Channel0 parameter 2
#define DATA_SIZE_TX_OFFSET    0x00C    //Channel0 parameter 3

// Rx_UART SPRAM parameters
#define MATCH_RATE_RX_OFFSET  0x024     //Channel1 parameter 1
#define DATA_UART_RX_OFFSET   0x028     //Channel1 parameter 2
#define DATA_SIZE_RX_OFFSET   0x02C     //Channel1 parameter 3

//
#define ETPU_MCR             (*((volatile unsigned int*)(ETPU_MCR_OFFSET +
ETPU_BASE)))
#define ETPU_TBCR_1          (*((volatile unsigned int*)(ETPU_TBCR_1_OFFSET
+ ETPU_BASE)))
#define ETPU_ECR_1          (*((volatile unsigned int*)(ETPU_ECR_1_OFFSET +
ETPU_BASE)))
#define ETPU_CIER_1          (*((volatile unsigned int*)(ETPU_CIER_1_OFFSET
+ ETPU_BASE)))
#define ETPU_CDTRER_1       (*((volatile unsigned int*)(ETPU_CDTRER_1_OFFSET
+ ETPU_BASE)))
#define ETPU_C0CR_1          (*((volatile unsigned int*)(ETPU_C0CR_1_OFFSET
+ ETPU_BASE)))
#define ETPU_C0SCR_1        (*((volatile unsigned int*)(ETPU_C0SCR_1_OFFSET
+ ETPU_BASE)))
#define ETPU_C0HSRR_1       (*((volatile unsigned int*)(ETPU_C0HSRR_1_OFFSET
+ ETPU_BASE)))
#define ETPU_C1CR_1          (*((volatile unsigned int*)(ETPU_C1CR_1_OFFSET
+ ETPU_BASE)))
#define ETPU_C1SCR_1        (*((volatile unsigned int*)(ETPU_C1SCR_1_OFFSET
+ ETPU_BASE)))
#define ETPU_C1HSRR_1       (*((volatile unsigned int*)(ETPU_C1HSRR_1_OFFSET
+ ETPU_BASE)))
#define MATCH_RATE_TX       (*((volatile unsigned int*)(MATCH_RATE_TX_OFFSET
+ SPRAM_BASE)))
#define DATA_UART_TX         (*((volatile unsigned int*)(DATA_UART_TX_OFFSET
+ SPRAM_BASE)))
#define DATA_SIZE_TX         (*((volatile unsigned int*)(DATA_SIZE_TX_OFFSET
+ SPRAM_BASE)))
#define MATCH_RATE_RX       (*((volatile unsigned int*)(MATCH_RATE_RX_OFFSET
+ SPRAM_BASE)))
```

```
#define DATA_UART_RX       (*((volatile unsigned int*)(DATA_UART_RX_OFFSET
+ SPRAM_BASE)))
#define DATA_SIZE_RX       (*((volatile unsigned int*)(DATA_SIZE_RX_OFFSET
+ SPRAM_BASE)))

// Macros
#define TCR2_PRESCALER(x)              ((x & 0x3F) <<  8)
#define TCR1_PRESCALER(x)              (x & 0xFF)
#define CHANNEL_FUNCTION(x)            ((x & 0x1F) << 16)
#define CHANNEL_PARAM_BASE_ADDR(x)     (x & 0xFF)
#define FUNCTION_MODE(x)               (x & 0x3)
#define MATCH_RATE_TRANS(x)            (x & 0xFFFF)
#define MATCH_RATE_REC(x)              (x & 0xFFFF)
#define DATA_WORD_Tx(x)                (x & 0x3FFF)
#define DATA_SIZE_TRANS(x)             (x & 0xF)
#define DATA_SIZE_REC(x)               (x & 0xF)
#define HOST_SERV_REQ(x)               (x & 0x7)
#define ENTRY_TABLE_BASE(x)            (x & 0x1F)


//ETPU_MCR fields - Module Configuration Register
#define PSE              0x00000002      //Parameter sign extension
#define SCMMISEN         0x00000200      //SCM MISC enable
#define VIS              0x00000040      //SCM visibility
#define GTBE             0x00000001      //Global time base enable

//ETPU_TBCR_1 fields - Time Base Configuration Register
#define TCRCLK_FILTER_TWOSAMPLE          0x00000000   //TCRCLK filter in Two
sample mode
#define TCRCLK_FILTER_INTEGRATOR         0x00800000   //TCRCLK filter in
Integrator mode
#define TCRCLK_FILTER_DIV2CLOCK          0x00000000   //TCRCLK filter uses
system clock divided by 2
#define TCRCLK_FILTER_CHANNELCLOCK       0x00400000   //TCRCLK filter uses
channel clock

#define TCR2_RISE                        0x00100000   //TCR2 inc rising edge
#define TCR2_FALL                        0x00200000   //TCR2 inc falling edge
#define TCR2_RISEFALL                    0x00300000   //TCR2 inc ris and fall
#define TCR2_GATEDDIV8                   0x00000000   //TCRCLK gates system
clock/8
#define TCR1CLK_SOURCE_DIV2              0x00000000   //TCR1 source system
clock/2
#define TRC1CLK_SOURCE_TCRCLK         0x00040000  //TCR1 source is TCRCLK pin
#define CHANNEL_FILTER_TWOSAMPLEMODE    0x00000000  //Filter:two sample mode
#define CHANNEL_FILTER_THREESAMPLEMODE  0x00008000  //Filter:three sample
mode
#define CHANNEL_FILTER_CONTMODE         0x0000C000  //Filter:continuous mode

//ETPU_ECR fields - Engine Configuration Register
#define FILTER_PRESCALER_CLOCK_DIV4      0x00010000   //System clock/4

//ETPU_CxCR fields - Channelx Configuration Register
#define CHANNEL_INT_ENABLE             0x80000000  //Channel Interrupt enable
#define CHANNEL_DATA_TRANSF_REQ_ENABLE  0x40000000  //Channel data transfer
req. enable
#define CHANNEL_PRIORITY_DISABLE         0x00000000   //Channel disable
```

```
#define CHANNEL_PRIORITY_LOW            0x10000000  //Low priority channel
#define CHANNEL_PRIORITY_MIDDLE        0x20000000  //Middle priority channel
#define CHANNEL_PRIORITY_HIGH           0x30000000  //High priority channel

//DATA_UART - SPRAM
#define CLEAR_TDRE     0x007FFFFF   //TDRE must be zero to signal new valid
                                    //data to be transmitted


void init_etpu( ){

volatile int temp;

//Initialize eTPU module configuration register(ETPU_MCR)
ETPU_MCR = 0x00070000; //SCMSIZE is 16K(7 2K blocks)

//Initialize eTPU time base configuration register(ETPU_TBCR)
ETPU_TBCR_1 = (TCR1CLK_SOURCE_DIV2 | CHANNEL_FILTER_TWOSAMPLEMODE |
TCR1_PRESCALER(8));

//Initialize eTPU engine configuration register(ETPU_ECR)
ETPU_ECR_1 =  (ENTRY_TABLE_BASE(0x1F) | FILTER_PRESCALER_CLOCK_DIV4);

//Write to the channel configuration Registers(ETPU_CxCR) to choose the
//function to be performed by the channel and its parameter base
address.Standard entry table //is selected.
ETPU_C0CR_1 = (CHANNEL_INT_ENABLE | CHANNEL_FUNCTION(15) |
CHANNEL_PARAM_BASE_ADDR(0x00));
ETPU_C1CR_1 = (CHANNEL_INT_ENABLE | CHANNEL_FUNCTION(15) |
CHANNEL_PARAM_BASE_ADDR(0x02));

//Write to the channel status control registers(ETPU_CxSCR) to choose
//variations within the function flow.
ETPU_C0SCR_1 = (FUNCTION_MODE(0));  // no parity for transmitter
ETPU_C1SCR_1 = (FUNCTION_MODE(0));  // no parity for receiver

//write to spram for parameter initialization of each configured
//channel
MATCH_RATE_TX = MATCH_RATE_TRANS(0x412); //setup match rate for transmitter
DATA_UART_TX = DATA_WORD_TX(0x000000AA); //load first byte to be
transmitted=AA
DATA_SIZE_TX = DATA_SIZE_TRANS(8);       //8-bit data word for transmitter
MATCH_RATE_RX = MATCH_RATE_REC(0x412);   //setup match rate for receiver
DATA_SIZE_RX = DATA_SIZE_REC(8);         //8-bit data word for receiver

//Write to Channel host service request registers(ETPU_CxHSRR) to
//initialize active channels(Channel 0 and 1)
ETPU_C0HSRR_1 = HOST_SERV_REQ(3);
ETPU_C1HSRR_1 = HOST_SERV_REQ(2);

//write to Channel priority field to enable each channel by
//assigning it a high,middle or low priority
ETPU_C0CR_1 =(ETPU_C0CR_1 | CHANNEL_PRIORITY_HIGH);
ETPU_C1CR_1 =(ETPU_C1CR_1 | CHANNEL_PRIORITY_HIGH);

//Monitor channel host service request register for completion
//of initialization
```

```
//HSR should be zero in the end of initialization
do
{
  temp = ETPU_C0HSRR_1;
} while (temp != 0);

do
{
  temp = ETPU_C1HSRR_1;
} while (temp != 0);


//Write GTBE bit to start TCR1 and TCR2 time bases counting
//at the same time
ETPU_MCR = (ETPU_MCR | GTBE);

}// end of etpu_initialization routine

***************************************************************************
*********
```

22.7.3 Predefined channel mode summary

Table 473 explains channel double match predefined submode functionality by showing all event sequence possibilities. The initial state considered for all submodes is channel flags MRLA, MRLB, TDLA and TDLB reset. From initial state one can follow the table and verify how each submode behaves in a determined sequence of events. Note that the actions performed by an event type depend on all previous events following the initial state, for a given channel submode.

There are three columns for each event: one for event type, one for enable/disable actions and one for capture. **Event type** column can be matchA, matchB, transA and transB (for double transition modes). **Enable/disable actions** column (identified as "**[blocks](enables)**" in column head) specifies which other events are enabled or disabled. Initially disabled events (specified in **"initially blocked"** column) are usually enabled by other events.

In double transition submodes, the first transition detected is always considered transA and the second is considered transB. This means that transA event actually enables the detection of transB event. This is not explicit in the table, since it is a general behavior for all double transition submodes.

A sequence of four events (two matches and two transitions) are necessary to describe the behavior of some channel submodes. When a determined sequence of events has less than four events, the other event columns are left blank.

Cells in an **"event type"** column that have light-grayed background indicate that a service request is generated. More than one event in the same event sequence can issue service request.

Note: The table does not exhaust all possibilities of channel logic event sequences, because it does not account for possible microcode interventions. For instance, if matches are blocked by first transition and microcode resets TDLA, the matches become enabled again, and from this point on the channel behaves as if the first transition had never occurred.

Table 473. Predefined channel mode summary

Mode	initially blocked	1st event			2nd event			3rd event			4th event		
		event type	[blocks] (enables(1))	Capt.	event type	[blocks] (enables)	Capt.	event type	[blocks] (enables)	Capt.	event type	[blocks] (enables)	Capt.
em_nb_st	none	matcha/b	none	1/2	matchB/A	none	2/1	transA	[matches]	both	transB	none	2
					transA	[matches]	both	transB	none	2			
		transA	[matches]	both	transB	none	2						
em_nb_dt	none	matchA/B	none	1/2	matchB/A	none	2/1	transA	none	1	transB	none	2
		matchA	none	1	transA	none	1	matchB	none	2	transB	none	2
								transB	[matchB]	2			
		matchB	none	2	transA	[matchA]	1	transB	[matchB]	2			
		transA	[matchA]	1	matchB	none	2	transB	none	2			
					transB	[matchB]	2						
em_b_st	none	matchA	[matchB]	both	transA	[matches]	both	transB	none	2			
		matchB	[matchA]	both									
		transA	[matches]	both	transB	none	2						
em_b_dt	none	matchA	[matchB]	both	transA	none	1	transB	none	2			
		matchB	[matchA]	both									
		transA	[matchA]	1	matchB	none	2	transB	none	2			
					transB	[matchB]	2						
bm_st	none	matchA/B	none	1/2	matchB/A	none	2/1	transA	[matches]	both	transB	none	2
					transA	[matches]	both	transB	none	2			
		transA	[matches]	both	transB	none	2						

Table 473. Predefined channel mode summary (continued)

Mode	initially blocked	1st event			2nd event			3rd event			4th event		
		event type	[blocks] (enables[1])	Capt.	event type	[blocks] (enables)	Capt.	event type	[blocks] (enables)	Capt.	event type	[blocks] (enables)	Capt.
bm_dt	none	matchA/B	none	1/2	matchB/A	none	2/1	transA	none	1	transB	[matches]	2
					transA	none	1	matchB/A	none	2/ none	transB	[matches]	2
								transB	[matches]	2			
		transA	none	1	matchA/B	none	none/2	matchB/A	none	2/ none	transB	[matches]	2
								transB	[matches]	2			
					transB	[matches]	2						
m2_st	transA	matchA	(transA)	1	matchB	none	2	transA	[matches]	both	transB	none	2
					transA	[matches]	both	transB	none	2			
		matchA and matchB	(transA)	both	transA	[matches]	both	transB	none	2			
		matchB	[matchA]	2									
m2_dt	transA	matchA	(transA)	1	transA	none	1	transB	[matchB]	2			
								matchB	none	2	transB	none	2
					matchB	none	2	transA	none	1	transB	none	2
		matchA and matchB	(transA)	both	transA	none	1	transB	none	2			
		matchB	[matchA]	2									

Table 473. Predefined channel mode summary (continued)

Mode	initially blocked	1st event			2nd event			3rd event			4th event		
		event type	[blocks] (enables(1))	Capt.	event type	[blocks] (enables)	Capt.	event type	[blocks] (enables)	Capt.	event type	[blocks] (enables)	Capt.
m2_o_st	transA matchB	matchA	(matchB) (transA)	1	transA	[matches]	both	transB	none	2			
					matchB	[transA]	2						
m2_o_dt	transA matchB	matchA	(matchB) (transA)	1	transA	none	1	transB	[matchB]	2			
								matchB	[transB]	2			
					matchB	[transA]	2						
sm_st(2)	matchB	matchA	none	both	transA	none	both	transB	none	2			
		transA	[matchA]	both	transB	none	2						
sm_dt	matchB	matchA	none	both	transA	none	1	transB	none	2			
		transA	none	1	matchA	none	2	transB	none	2			
		transA	none	1	transB	[matchA]	2						
sm_st_e(3)	matchB	matchA	none	1	transA	none	1						
	transB	transA	[matchA]	1									

▨ Generates Service Request

1. Transition A always enables Transition B
2. sm_st is compatible with TPU3 channel logic.
3. It is not possible to include all functionality of this submode in table. See *Section , Single Match Enhanced Mode (sm_st_e)*, for more details.

22.7.4 MISC algorithm

The MISC generator is based on the following polynomial:

$G(x) = 1 + x^{1} + x^{2} + x^{22} + x^{31}$ `(equivalent to feedback mask = 0x80400007)`

The MISC signature generation starts by clearing the MISC Accumulator value to 0 and preloading the MISC Counter with the highest SCM address. It then steps through each address decrementing the counter, reading 32 bit values and following the algorithm below:

```
If the least significant bit in MISC is 1 then
    MISC = MISC right shifted by 1 bit
    MISC = MISC XOR 0x80400007
else
    MISC = MISC right shifted by 1 bit
end if
MISC = MISC XOR RAM data
```

The code example below shows an excerpt of C code that calculates the MISC signature for a given array of data, based on the previous algorithm:

```
#define SCM_size   (MAX_SCM_ADDRESS / 4)   /* last byte address - converted
to 32-bit word */
#define POLY       0x80400007             /* G(x) = 1 + x^1 + x^2 + x^22 + x^31 */

/***************************************************************************
******
 FUNCTION : void calc_misc()
 PURPOSE : This function calculates the MISC value.
 INPUTS NOTES : none
 RETURNS NOTES : MISC value
 GENERAL NOTES : the array’unsigned int data[]’ represents the actual memory
                 array, organized in 32-bit words.
***************************************************************************
*****/
unsigned int calc_misc (void)
{
  int j; /* loop counter */

  unsigned int misc = 0;

  for (j = (SCM_size-1); j >= 0 ; j--) {   /* SCM_size has the number of 32-
bit words in SCM */

      if (misc & 0x1) {
          misc >>= 1;
          misc ^= POLY;
      }
      else {
          misc >>= 1;
      }
      misc ^= data[j];          /* data[j] is the actual 32-bit word taken
from the SCM array */
  }

  return (misc);          /* final signature calculated */

};
```

The value calculated by this algorithm must be loaded into register ETPU_MISCCMPR prior to activating the SCM MISC calculator in eTPU. Once the MISC calculator is activated (bit SCMMISEN in register ETPU_MCR is written to 1) eTPU itself will start this procedure(ap) reading the SCM whenever allowed by microengine. At the end of the cycle, when all the array has been read and the SCM signature is calculated, the Host CPU can be notified via Global Exception if the MISC Accumulator does not match the value in ETPU_MISCCMPR.

Equation 12 shows how the average time taken by MISC to complete the signature of the whole SCM can be calculated.

Equation 12 Average MISC period = S / (4 * f * (1 - L))

In *Equation 12*,

f = clock frequency

S = SCM size in bytes

L = eTPU load (as a percentage of execution clocks over a period of time, including TST clocks)

Further detail on MISC calculation can be found on *Section , SCM Test – Multiple input signature calculator*.

---

ap. eTPU MISC hardware is optimized to read 32-bit words from memory and to calculate this CRC in parallel, rather than shifting one bit at a time. The actual implementation inside eTPU, although bringing to the same results, does not match exactly the algorithm shown here.

23 Enhanced Queued Analog-to-Digital Converter (EQADC)

23.1 Information Specific to This Device

This section presents device-specific parameterization and customization information not specifically referenced in the remainder of this chapter.

23.1.1 Device-Specific Pin Configuration Features

The following eQADC pins are multiplexed and configuration of the corresponding Systems Integration Unit (SIU) registers is necessary.

AN12/MA0/SDS

These pins are configured by setting the Pad Configuration Register 215 (SIU_PCR215) on the SIU.

Note: Attempts to convert the input voltage applied to this pin while the MA0 or the SDS functions are selected will result in an undefined conversion result.

As this pin is also used by digital logic, it has reduced analog to digital conversion accuracy when compared to the AN[0:11,16:39] analog input pins.

AN13/MA1/SDO

These pins are configured by setting the Pad Configuration Register 216 (SIU_PCR216) on the SIU.

Note: Attempts to convert the input voltage applied to this pin while the MA1 or the SDO functions are selected will result in an undefined conversion result.

As this pin is also used by digital logic, it has reduced analog to digital conversion accuracy when compared to the AN[0:11,16:39] analog input pins.

AN14/MA2/SDI

These pins are configured by setting the Pad Configuration Register 217 (SIU_PCR217) on the SIU.

Note: Attempts to convert the input voltage applied to this pin while the MA2 or the SDI functions are selected will result in an undefined conversion result.

As this pin is also used by digital logic, it has reduced analog to digital conversion accuracy when compared to the AN[0:11,16:39] analog input pins.

AN15/FCK

These pins are configured by setting the Pad Configuration Register 218 (SIU_PCR218) on the SIU.

Note: Attempts to convert the input voltage applied to this pin while the FCK function is selected will result in an undefined conversion result.

As this pin is also used by digital logic, it has reduced analog to digital conversion accuracy when compared to the AN[0:11,16:39] analog input pins.

External Triggers

The source of the eQADC external triggers can be the eTPU, the eMIOS, or an external signal. The source is selected by configuring the eQADC Trigger Input Select Register (SIU_ETISR) on the SIU.

23.1.2 Availability of Analog Inputs

Analog inputs ANR, ANS, ANT and ANU are not available on SPC564A70 devices.

23.2 Introduction

23.2.1 Module overview

The Enhanced Queued Analog-to-Digital Converter (EQADC) block provides accurate and fast conversions for a wide range of applications. The EQADC provides a parallel interface to two on-chip analog-to-digital converters (ADCs), a single master to single slave serial interface to an off-chip external device, and a parallel side interface to one or more on-chip digital signal processing (DSP) modules (for example, a decimation filter). The two on-chip ADCs are architected to allow access to all the analog channels.

The EQADC transfers commands from multiple Command FIFOs (CFIFOs) to the on-chip ADCs or to the external device. The multiple Result FIFOs (RFIFOs) can receive data from the on-chip ADCs, from an off-chip external device or from an on-chip DSP module. Data from the on-chip ADCs can be routed to the side interface, processed by the on-chip DSP and then routed back through the side interface to the RFIFOs. The EQADC supports software and external hardware triggers from other blocks to initiate transfers of commands from the CFIFOs to the on-chip ADCs or to the external device. It also monitors the fullness of CFIFOs and RFIFOs, and accordingly generates DMA or interrupt requests to control data movement between the FIFOs and the system memory, which is external to the EQADC.

23.2.2 Block diagram

Figure 445 is the block diagram for the EQADC.

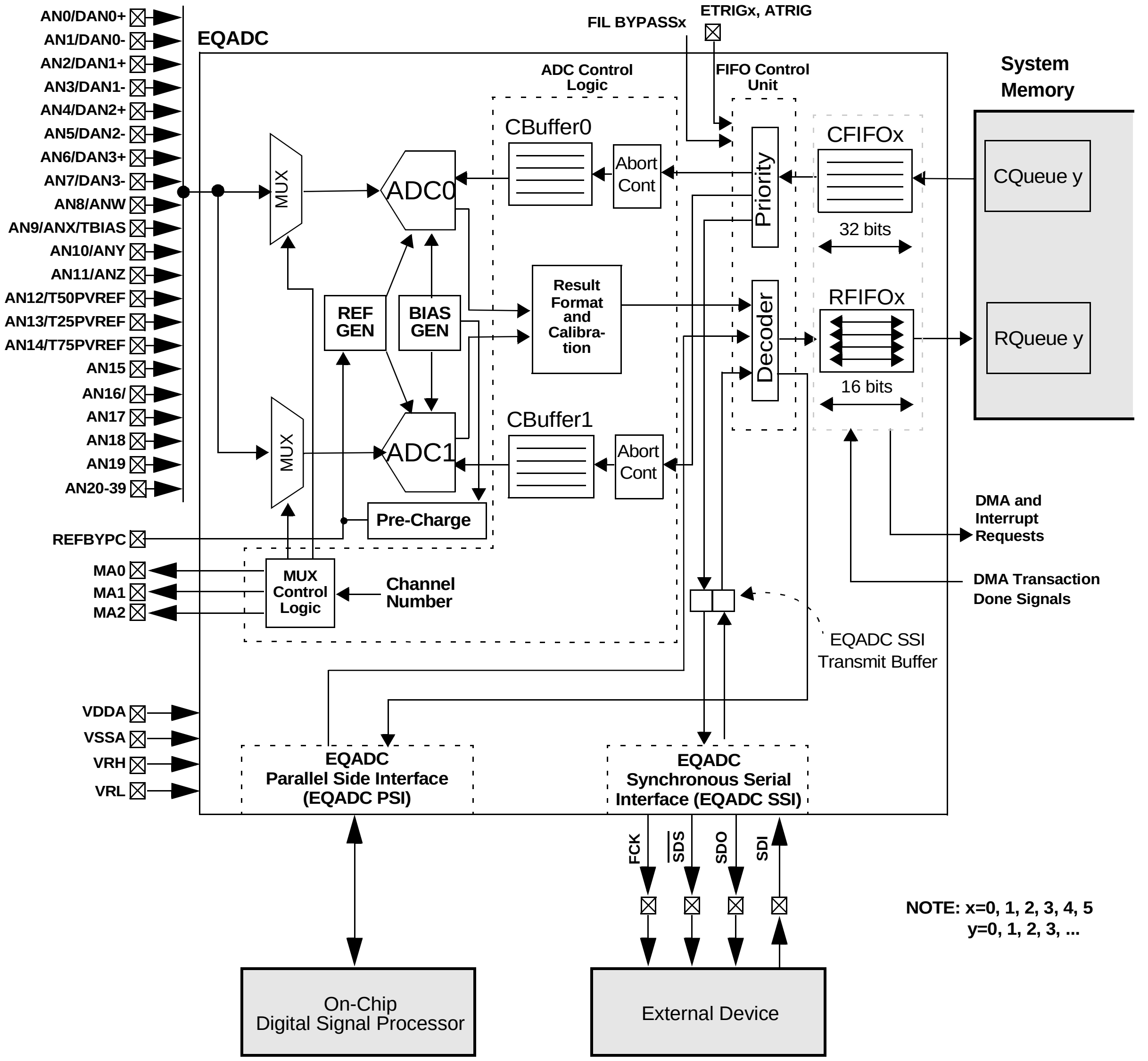


Figure 445. EQADC Block Diagram

Figure 445 shows the primary components inside the EQADC. The EQADC consists of the *FIFO Control Unit* which controls the CFIFOs and the RFIFOs, the *ADC Control Logic* which controls the two on-chip ADCs, the *EQADC Synchronous Serial Interface* (EQADC SSI) which allows communication with an external device, and the *EQADC Parallel Side Interface* (EQADC PSI) which allows communication with on-chip eQADC companion modules[aq]. There are 6 CFIFOs and 6 RFIFOs, each with 4 entries, except CFIFO0 that can have 8 entries.

The *FIFO Control Unit* performs the following functions:

- It prioritizes the CFIFOs to determine what CFIFOs will have their commands transferred.
- Supports software and hardware triggers to start command transfers from a particular CFIFO.
- Decodes command data from the CFIFOs, and accordingly, sends these commands to one of the two on-chip ADCs or to the external device.
- Decodes result data from on-chip ADCs or from the external device, and transfers data to the appropriate RFIFO or to the parallel side interface.

The *ADC Control Logic* manages the execution of commands bound for on-chip ADCs. It interfaces with the CFIFOs via two 2-entry command buffers (CBuffers) with abort control and with the RFIFOs and side interface via the *Result Format and Calibration Sub-Block*. The *ADC Control Logic* performs the following functions:

- Buffers command data for execution.
- Decodes command data and accordingly generates control signals for the two on-chip ADCs.
- Detects abort request, stores aborted commands and buffers immediate conversion commands.
- Formats and calibrates conversion result data coming from the on-chip ADCs.
- Generates the internal multiplexer control signals and the select signals used by the external multiplexers.

The EQADC SSI allows for a full duplex, synchronous, serial communication between the EQADC and an external device.

The EQADC PSI allows for a full duplex, synchronous, parallel communication between the EQADC and decimation filters A and B and reaction modules.

Figure 445 also depicts data flow through the EQADC. Commands are contained in system memory in a user defined data structure. The most likely data structure to be used is a queue as depicted in the *Figure 445*(ar). Command data is moved from the command queue (CQueue) to the CFIFOs by either the host CPU or by the DMAC. Once a CFIFO is triggered and becomes the highest priority CFIFO using a certain CBuffer, command data is transferred from the CFIFO to the on-chip ADCs, or to the external device. The ADC executes the command, and the result is moved through the *Result Format and Calibration Sub-Block* to either the side interface or to the RFIFO. Data from the external device or on-chip companion module bypasses the *Result Format and Calibration Sub-Block* and is moved directly to its specified RFIFO. When data is stored in an RFIFO, data is moved from the RFIFO by the host CPU or by the DMAC to a data structure in system memory depicted in the *Figure 445* as a result queue (RQueue).

For users familiar with the QADC, the EQADC system upgrades the functionality provided by that block. Refer to *Section 23.7.7, EQADC versus QADC*, for a comparison between the EQADC and QADC.

[aq] Decimation filters A and B and Reaction module

[ar] Command and result data can be stored in system memory in any user defined data structure. However, in this document it will be assumed that the data structure of choice is a queue, since it is the most likely data structure to be used and because queues are the only type of data structure supported by the DMAC.

23.2.3 Features

The EQADC Block includes these distinctive features:

- Two independent on-chip RSD Cyclic ADCs
  - 8, 10, and 12 bits AD Resolution
  - Targets up to 10 bit accuracy at 500KSample/s (ADC_CLK=7.5 MHz) and 8 bit accuracy at 1M Sample/s (ADC_CLK=15 MHz) for differential conversions
  - Selectable common mode conversion range (0 - 5V; 0 - 2.5V; 0 - 1.25V)
  - Differential conversions
  - Differential channels include variable gain amplifier for improved dynamic range (x1; x2; x4)
  - Differential channels include programmable pull-up and pull-down resistors for biasing and sensor diagnostics (200k ohms; 100k ohms; 5k ohms)
  - Sample times of 2 (default), 8, 64 or 128 ADC clock cycles
  - Provides time stamp information when requested
  - Parallel interface to EQADC CFIFOs and RFIFOs
  - Supports both right-justified unsigned and signed formats for conversion results
  - The REFBYPC pin stabilizes one of internal generated reference
  - Temperature sensor
  - Ability to measure directly Vdd
- Automatic application of ADC calibration constants
  - Provision of reference voltages (25%VREF and 75%VREF) for ADC calibration purposes
- 40 input channels available to the two on-chip ADCs
- 4 pairs of differential analog input channels
- Full duplex synchronous serial interface to an external device
  - Has a free-running clock for use by the external device
  - Supports a 26-bit message length
  - Transmits a null message when there are no triggered CFIFOs with commands bound for external CBuffers, or when there are triggered CFIFOs with commands bound for external CBuffers but the external CBuffers are full
- Parallel Side Interface to communicate with several on-chip companion modules
- STAC bus Client Interface to import an alternative timebase to the internal time stamp
- Priority Based CFIFOs
  - Supports six CFIFOs with fixed priority. The lower the CFIFO number, the higher its priority. When commands of distinct CFIFOs are bound for the same CBuffer, the higher priority CFIFO is always served first.
  - Immediate conversion command feature with conversion abort control
  - Streaming mode operation of CFIFO0 to execute some commands several times
  - Supports software and several hardware trigger modes to arm a particular CFIFO
  - Generates interrupt when command coherency is not achieved
- External Hardware Triggers
  - Supports rising edge, falling edge, high level and low level triggers
  - Supports configurable digital filter

  - Supports controls to bypass the trigger digital filters
- Two Triggers operation mode for queue0
  - Additional internal trigger (not filtered) called Advance trigger that is used to enable the external trigger of queue0 and to control the loop behavior of CFIFO0
- Supports 4 to 8 external 8-to-1 muxes which can expand the input channel number from 40 to 96
- Upgrades the functionality provided by the QADC

23.3 Modes of operation

This section describes the operation modes of the EQADC.

23.3.1 Normal mode

This is the default operational mode when the EQADC is not in streaming mode or background debug or stop mode.

23.3.2 Streaming mode

This mode is characterized by two main aspects: the abort action by CFIFO0 in any current conversion process started from another queue, and the loop behavior of the CFIFO0.

In some applications, there may be sequences of identical commands each spaced only by a few microseconds. To reduce the DMA data transfer, in this mode a short command queue can be stored in CFIFO0 and repeatedly be executed based on a timed trigger, but advance to the next (repeating) sequence of commands based on another device’s internal trigger.

The CFIFO0 delivers commands to the ADC as before, but those commands are not ‘invalidated’ after they are sent (in fact, they are ‘invalidated’ only because the Transfer Next Data Pointer has moved on). When it encounters these repeated commands the CFIFO0 only fills once, using the DMA as usual, until either it is full or a command with End-of-Queue is encountered. Thereafter the sub-queue repeats/wraps. The number of commands loaded is unaffected by the delivery of commands once the streaming mode is configured, since no commands loaded are invalidated even if sent before all the queue is loaded.

The number of entries in the CFIFO0 is extended to eight (configurable). This is to facilitate the targeted applications. The repeating subqueue must be contained within the eight CFIFO0 entries.

To maintain compatibility, CFIFO0 by default operates as it does before, without streaming and with four entries. Streaming, and additional entries, can be enabled independently.

Streaming mode is selected as another mode for queue 0 using the configuration bits in the EQADC_CFCR register. Streaming mode makes use of an additional bit in the Conversion Command Word (CCW); this bit is called ‘Repeat’. The purpose of this bit is to mark in the command queue, where to start a repeating sequence.

Streaming mode requires two trigger inputs. The standard queue 0 trigger, in this mode referred to as ‘Repeat Trigger’ and a second internal trigger input to the eQADC called ‘Advance’ trigger.

23.3.3 Debug mode

Upon a debug mode entry request, EQADC behavior will vary according to the status of the DBG field in *Section , EQADC Module Configuration Register (EQADC_MCR)*. If DBG is programmed to 0b00, the debug mode entry request is ignored. If DBG is programmed to 0b10 or to 0b11, the EQADC will enter debug mode. In case the EQADC SSI is enabled, the free running clock (FCK) output to external device will not stop when DBG is programmed to 0b11, but FCK will stop in low phase, when DBG is programmed to 0b10.

During debug mode, the EQADC will not transfer commands from any CFIFOs, no null messages will be transmitted to the external device, no data will be returned to any RFIFO, no hardware trigger event will be captured, and all EQADC registers can be accessed as in Normal mode. The latter implies that CFIFOs can still be triggered using software triggers, since no scheme is implemented to write-protect registers during debug mode. DMA and interrupt requests continue to be generated as in Normal Mode.

If at the time the debug mode entry request is detected, there are commands in the on-chip CBuffers that were already under execution, these commands will be completed but the generated results, if any, will not be sent to the RFIFOs until debug mode is exited. Commands whose execution has not started will not be executed until debug mode is exited.The clock associated with an on-chip ADC stops, during its low phase, after the ADC ceases executing commands. The time base counter will only stop after all on-chip ADCs cease executing commands.

When exiting debug mode, the EQADC relies on the CFIFO operation modes and on the CFIFO status to determine the next command entry to transfer.

The EQADC internal behavior after the debug mode entry request is detected differs depending on the status of command transfers.

- No command transfer is in progress.

  The EQADC immediately halts future command transfers from any CFIFO.

  If a null message is being transmitted, EQADC will complete the serial transmission before halting future command transfers. If valid data (conversion result or data read from an ADC register) is received at the end of transmission, it will not be sent to an RFIFO until debug mode is exited.

  If the null message transmission is aborted, the EQADC will complete the abort procedure before halting future command transfers from any CFIFO. The message of the CFIFO that caused the abort of the previous serial transmission will only be transmitted after debug mode is exited.
- Command transfer is in progress.

  EQADC will complete the transfer and update CFIFO status before halting future command transfers from any CFIFO. Command transfers to the internal CBuffers are considered completed when a command is written to the buffers.

  Command transfers to the external device are considered completed when the serial transmission of the command is completed. If valid data (conversion result or data read from an ADC register) is received at the end of a serial transmission, it will not be sent to an RFIFO until debug mode is exited. The CFIFO status bits will still be updated after the completion of the serial transmission, therefore, after debug mode entry request is detected, the EQADC status bits will only stop changing several system clock cycles after the on-going serial transmission completes.

  If the command message transmission is aborted, the EQADC will complete the abort procedure before halting future command transfers from any CFIFO. The message of the CFIFO that caused the abort of the previous serial transmission will only be transmitted after debug mode is exited.

- Command/Null message transfer through serial interface was aborted but next serial transmission did not start.

  If the debug mode entry request is detected between the time a previous serial transmission was aborted and the start of the next transmission, the EQADC will complete the abort procedure before halting future command transfers from any CFIFO. The message of the CFIFO that caused the abort of the previous serial transmission will only be transmitted after debug mode is exited.

23.3.4 Stop mode

Upon a stop mode entry request detection, the EQADC progressively halts its operations until it reaches a static, stable state from which it can recover when returning to Normal mode. EQADC then asserts an acknowledge signal, indicating that it is static and that the clock input can be stopped. In stop mode, the free running clock (FCK) output to external device will stop during its low phase if the EQADC SSI is enabled, and no hardware trigger events will be captured. The latter implies that, as long as the system clock is running, CFIFOs can still be triggered using software triggers, since no scheme is implemented to write-protect registers during stop mode.

If at the time the stop mode entry request is detected, there are commands in the on-chip CBuffers that were already under execution, these commands will be completed but the generated results, if any, will not be sent to the RFIFOs until stop mode is exited. Commands whose execution has not started will not be executed until stop mode is exited.

After these remaining commands are executed, the clock input to the ADCs is stopped. The ADC clock stops during its low phase. The time base counter will only stop after all on-chip ADCs cease executing commands. Only then, the stop acknowledge signal is asserted. When exiting stop mode, the EQADC relies on the CFIFO operation modes and on the CFIFO status to determine the next command entry to transfer.

The EQADC internal behavior after the stop mode entry request is detected differs depending on the status of the command transfer.

- No command transfer is in progress

  The EQADC immediately halts future command transfers from any CFIFO.

  If a null message is being transmitted, EQADC will complete the transmission before halting future command transfers. If valid data (conversion result or data read from an ADC register) is received at the end of the transmission, it will not be sent to an RFIFO until stop mode is exited.

  If the null message transmission is aborted, the EQADC will complete the abort procedure before halting future command transfers from any CFIFO. The message of the CFIFO that caused the abort of the previous serial transmission will only be transmitted after stop mode is exited.

- Command transfer is in progress

  EQADC will complete the transfer and update CFIFO status before halting future command transfers from any CFIFO. Command transfers to the internal CBuffers are considered completed when a command is written to the buffers.

  Command transfers to the external device are considered completed when the serial transmission of the command is completed. If valid data (conversion result or data read from an ADC register) is received at the end of a serial transmission, it will not be sent to an RFIFO until stop mode is exited. The CFIFO status bits will still be updated after the completion of the serial transmission, therefore, after stop mode entry request is detected, the EQADC status bits will only stop changing several system clock cycles after the on-going serial transmission completes.

If the command message transmission is aborted, the EQADC will complete the abort procedure before halting future command transfers from any CFIFO. The message of the CFIFO that caused the abort of the previous serial transmission will only be transmitted after stop mode is exited.

- Command/Null message transfer through serial interface was aborted but next serial transmission did not start.

  If the stop mode entry request is detected between the time a previous serial transmission was aborted and the start of the next transmission, the EQADC will complete the abort procedure before halting future command transfers from any CFIFO. The message of the CFIFO that caused the abort of the previous serial transmission will only be transferred after stop mode is exited.

23.4 External signal description

23.4.1 Overview

The following is a list of external pins.

Note: At chip integration level, some of the digital and analog signals listed here might share pins or not be available external to the chip. Refer to the Signals chapter for details.

Table 474. External Signals

Name	Port	Function	Reset State	Type
AN0/DAN0+	Input	Single-ended analog input / Differential analog input positive terminal	—	Analog
AN1/DAN0-	Input	Single-ended analog input / Differential analog input negative terminal	—	Analog
AN2/DAN1+	Input	Single-ended analog input / Differential analog input positive terminal	—	Analog
AN3/DAN1-	Input	Single-ended analog input / Differential analog input negative terminal	—	Analog
AN4/DAN2+	Input	Single-ended analog input / Differential analog input positive terminal	—	Analog
AN5/DAN2-	Input	Single-ended analog input / Differential analog input negative terminal	—	Analog
AN6/DAN3+	Input	Single-ended analog input / Differential analog input positive terminal	—	Analog

Table 474. External Signals (continued)

Name	Port	Function	Reset State	Type
AN7/DAN3-	Input	Single-ended analog input / Differential analog input negative terminal	—	Analog
AN8/ANW	Input	Single-ended analog input / Single-ended analog input from external multiplexers	—	Analog
AN9/ANX	Input	Single-ended analog input / Single-ended analog input from external multiplexers	—	Analog
AN10/ANY	Input	Single-ended analog input/Single-ended analog input from external multiplexers	—	Analog
AN11/ANZ	Input	Single-ended analog input / Single-ended analog input from external multiplexers	—	Analog
AN12	Input / Output	Single-ended analog input	—	Analog
AN13	Input / Output	Single-ended analog input	—	Analog
AN14	Input / Output	Single-ended analog input	—	Analog
AN15	Input	Single-ended analog input	—	Analog
AN16	Input	Single-ended analog input	—	Analog
AN17	Input	Single-ended analog input	—	Analog
AN18	Input	Single-ended analog input	—	Analog
AN19	Input	Single-ended analog input	—	Analog
AN20	Input	Single-ended analog input	—	Analog
AN21	Input	Single-ended analog input	—	Analog
AN22	Input	Single-ended analog input	—	Analog
AN23	Input	Single-ended analog input	—	Analog
AN24	Input	Single-ended analog input	—	Analog
AN25	Input	Single-ended analog input	—	Analog
AN26	Input	Single-ended analog input	—	Analog
AN27	Input	Single-ended analog input	—	Analog
AN28	Input	Single-ended analog input	—	Analog
AN29	Input	Single-ended analog input	—	Analog
AN30	Input	Single-ended analog input	—	Analog
AN31	Input	Single-ended analog input	—	Analog
AN32	Input	Single-ended analog input	—	Analog
AN33	Input	Single-ended analog input	—	Analog
AN34	Input	Single-ended analog input	—	Analog

Table 474. External Signals (continued)

Name	Port	Function	Reset State	Type
AN35	Input	Single-ended analog input	—	Analog
AN36	Input	Single-ended analog input	—	Analog
AN37	Input	Single-ended analog input	—	Analog
AN38	Input	Single-ended analog input	—	Analog
AN39	Input	Single-ended analog input	—	Analog
MA0	Output	External multiplexer control signal	0	Digital
MA1	Output	External multiplexer control signal	0	Digital
MA2	Output	External multiplexer control signal	0	Digital
FCK	Output	EQADC SSI free running clock	0	Digital
SDS	Output	EQADC SSI serial data select	1	Digital
SDI	Input	EQADC SSI serial data in		Digital
SDO	Output	EQADC SSI serial data out	0	Digital
VDDA	Input	Analog Positive Power Supply	—	Power
VSSA	Input	Analog Negative Power Supply	—	Power
VRH	Input	Voltage Reference High	—	Power
VRL	Input	Voltage Reference Low	—	Power
REFBYPC	Input	External Bypass capacitor Pin	—	Power
ETRIG0	Input	External trigger for CFIFO0	—	Digital
ETRIG1	Input	External trigger for CFIFO1	—	Digital
ETRIG2	Input	External trigger for CFIFO2	—	Digital
ETRIG3	Input	External trigger for CFIFO3	—	Digital
ETRIG4	Input	External trigger for CFIFO4	—	Digital
ETRIG5	Input	External trigger for CFIFO5	—	Digital

23.4.2 Detailed signal descriptions

AN0/DAN0+ — Single-ended analog input/Differential analog input positive terminal

AN0 is a single-ended analog input to the two on-chip ADCs. DAN0+ is the positive terminal of the differential analog input DAN0 (DAN0+ - DAN0-).

AN1/DAN0— — Single-ended analog input/Differential analog input negative terminal

AN1 is a single-ended analog input to the two on-chip ADCs. DAN0- is the negative terminal of the differential analog input DAN0 (DAN0+ - DAN0-).

AN2/DAN1+ — Single-ended analog input/Differential analog input positive terminal

AN2 is a single-ended analog input to the two on-chip ADCs. DAN1+ is the positive terminal of the differential analog input DAN1 (DAN1+ - DAN1-).

AN3/DAN1— — Single-ended analog input/Differential analog input negative terminal

AN3 is a single-ended analog input to the two on-chip ADCs. DAN1- is the negative terminal of the differential analog input DAN1 (DAN1+ - DAN1-).

AN4/DAN2+ — Single-ended analog input/Differential analog input positive terminal

AN4 is a single-ended analog input to the two on-chip ADCs. DAN2+ is the positive terminal of the differential analog input DAN2 (DAN2+ - DAN2-).

AN5/DAN2— — Single-ended analog input/Differential analog input negative terminal

AN5 is a single-ended analog input to the two on-chip ADCs. DAN2- is the negative terminal of the differential analog input DAN2 (DAN2+ - DAN2-).

AN6/DAN3+ — Single-ended analog input/Differential analog input positive terminal

AN6 is a single-ended analog input to the two on-chip ADCs. DAN3+ is the positive terminal of the differential analog input DAN3 (DAN3+ - DAN3-).

AN7/DAN3— — Single-ended analog input/Differential analog input negative terminal

AN7 is a single-ended analog input to the two on-chip ADCs. DAN3- is the negative terminal of the differential analog input DAN3 (DAN3+ - DAN3-).

AN8/ANW — Single-ended analog input/ Single-ended analog input from external multiplexers

AN8 is a single-ended analog input to the two on-chip ADCs. ANW is a single-ended analog input to one of the on-chip ADCs in external multiplexed mode.

AN9/ANX — Single-ended analog input/ Single-ended analog input from external multiplexers

AN9 is a single-ended analog input to the two on-chip ADCs. ANX is a single-ended analog input to one of the on-chip ADCs in external multiplexed mode.

AN10/ANY — Single-ended analog input/ Single-ended analog input from external multiplexers

AN10 is a single-ended analog input to the two on-chip ADCs. ANY is a single-ended analog input to one of the on-chip ADCs in external multiplexed mode.

AN11/ANZ — Single-ended analog input/ Single-ended analog input from external multiplexers

AN11 is a single-ended analog input to the two on-chip ADCs. ANZ is a single-ended analog input to one of the on-chip ADCs in external multiplexed mode.

AN12 — Single-ended analog input

AN12 is a single-ended analog input to the two on-chip ADCs.

AN13 — Single-ended analog input/

AN13 is a single-ended analog input to the two on-chip ADCs.

AN14 — Single-ended analog input

AN14 is a single-ended analog input to the two on-chip ADCs.

AN15 — Single-ended analog input

AN15 is a single-ended analog inputs to the two on-chip ADCs.

AN16 — Single-ended analog input/

AN16 is a single-ended analog input to the two on-chip ADCs.

AN17 — Single-ended analog input

AN17 is a single-ended analog input to the two on-chip ADCs.

AN18 — Single-ended analog input

AN18 is a single-ended analog input to the two on-chip ADCs.

AN19 — Single-ended analog input

AN19 is a single-ended analog input to the two on-chip ADCs.

AN20-AN39 — Single-ended analog input

AN20 through AN39 are single-ended analog inputs to the two on-chip ADCs.

INA_ADC0_0 - INA_ADC0_9 — Single-ended analog input

INA_ADC0_0 through INA_ADC0_9 are single-ended analog inputs to the on-chip ADC0.

INA_ADC1_0 - INA_ADC1_9 — Single-ended analog input

INA_ADC1_0 through INA_ADC1_9 are single-ended analog inputs to the on-chip ADC1.

MA0-MA2 — External multiplexer control signals

MA0, MA1, and MA2 combined form a select signal associated with external multiplexers.

FCK — EQADC SSI free-running clock

FCK is a free-running clock signal for synchronizing transmissions between the EQADC (master) and the external (slave) device.

$\overline{\text{SDS}}$ — EQADC SSI serial data select

$\overline{\text{SDS}}$ is the serial data select output. It is used to indicate to the external (slave) device when it can latch incoming serial data, when it can output its own serial data, and when it must abort a data transmission. $\overline{\text{SDS}}$ corresponds to the chip select signal in a conventional SPI interface.

SDI — EQADC SSI serial data in

SDI is the serial data input signal from the external (slave) device.

SDO — EQADC SSI serial data out

SDO is the serial data output signal to the external (slave) device.

VRH, VRL — Voltage reference high and voltage reference low

VRH and VRL are voltage references for the ADCs. VRH is the highest voltage reference, while VRL is the lowest voltage reference.

VDDA, VSSA — 5V VDD and VSS for the 5V analog components

VDDA is the positive power supply pin for the ADCs and VSSA is the negative power supply pin for the ADCs. Refer to electrical specifications.

REFBYPC — Reference Bypass Capacitor

The REFBYPC pin is used to connect an external bypass capacitor between REFBYPC and VRL. The value of this capacitor should be 100nf. This bypass capacitor is used to provide a stable reference voltage for the ADC.

ETRIG0—ETRIG5 — External triggers

The external trigger signals are for hardware triggering. The EQADC can detect rising edge, falling edge, high level and low level on each of the external trigger signals. ETRIGx triggers CFIFOx. The EQADC also supports configurable digital filters for these external trigger signals. These digital filters can be bypassed by individual input control signals called eqadc_intern_trig_selx.

23.5 Memory Map/Register Definition

This section provides memory maps and detailed descriptions of all registers. Data written to or read from reserved areas of the memory map is undefined.

23.5.1 EQADC Memory Map

This section provides memory maps for the EQADC block.

Table 475. EQADC Memory Map

Address	Use	Access
EQADC_BASE+0x000	EQADC Module Configuration Register (EQADC_MCR)	Unrestricted
EQADC_BASE+0x004	EQADC Test Register (EQADC_TST)	Test
EQADC_BASE+0x008	EQADC Null Message Send Format Register (EQADC_NMSFR)	Unrestricted

Table 475. EQADC Memory Map (continued)

Address	Use	Access
EQADC_BASE+0x00C	EQADC External Trigger Digital Filter Register (EQADC_ETDFR)	Unrestricted
EQADC_BASE+0x010	EQADC CFIFO Push Register 0 (EQADC_CFPR0)	Write only
EQADC_BASE+0x014	EQADC CFIFO Push Register 1 (EQADC_CFPR1)	Write only
EQADC_BASE+0x018	EQADC CFIFO Push Register 2 (EQADC_CFPR2)	Write only
EQADC_BASE+0x01C	EQADC CFIFO Push Register 3 (EQADC_CFPR3)	Write only
EQADC_BASE+0x020	EQADC CFIFO Push Register 4 (EQADC_CFPR4)	Write only
EQADC_BASE+0x024	EQADC CFIFO Push Register 5 (EQADC_CFPR5)	Write only
EQADC_BASE+0x028	Reserved	-
EQADC_BASE+0x02C	Reserved	-
EQADC_BASE+0x030	EQADC Result FIFO Pop Register 0 (EQADC_RFPR0)	Read only
EQADC_BASE+0x034	EQADC Result FIFO Pop Register 1 (EQADC_RFPR1)	Read only
EQADC_BASE+0x038	EQADC Result FIFO Pop Register 2 (EQADC_RFPR2)	Read only
EQADC_BASE+0x03C	EQADC Result FIFO Pop Register 3 (EQADC_RFPR3)	Read only
EQADC_BASE+0x040	EQADC Result FIFO Pop Register 4 (EQADC_RFPR4)	Read only
EQADC_BASE+0x044	EQADC Result FIFO Pop Register 5 (EQADC_RFPR5)	Read only
EQADC_BASE+0x048	Reserved	-
EQADC_BASE+0x04C	Reserved	-
EQADC_BASE+0x050	EQADC CFIFO Control Register 0 (EQADC_CFCR0)	Unrestricted
EQADC_BASE+0x054	EQADC CFIFO Control Register 1 (EQADC_CFCR1)	Unrestricted
EQADC_BASE+0x058	EQADC CFIFO Control Register 2 (EQADC_CFCR2)	Unrestricted
EQADC_BASE+0x05C	Reserved	-
EQADC_BASE+0x060	EQADC Interrupt and DMA Control Register 0 (EQADC_IDCR0)	Unrestricted
EQADC_BASE+0x064	EQADC Interrupt and DMA Control Register 1 (EQADC_IDCR1)	Unrestricted
EQADC_BASE+0x068	EQADC Interrupt and DMA Control Register 2 (EQADC_IDCR2)	Unrestricted
EQADC_BASE+0x06C	Reserved	-
EQADC_BASE+0x070	EQADC FIFO and Interrupt Status Register 0 (EQADC_FISR0)	Unrestricted
EQADC_BASE+0x074	EQADC FIFO and Interrupt Status Register 1 (EQADC_FISR1)	Unrestricted
EQADC_BASE+0x078	EQADC FIFO and Interrupt Status Register 2 (EQADC_FISR2)	Unrestricted
EQADC_BASE+0x07C	EQADC FIFO and Interrupt Status Register 3 (EQADC_FISR3)	Unrestricted
EQADC_BASE+0x080	EQADC FIFO and Interrupt Status Register 4 (EQADC_FISR4)	Unrestricted
EQADC_BASE+0x084	EQADC FIFO and Interrupt Status Register 5 (EQADC_FISR5)	Unrestricted
EQADC_BASE+0x088	Reserved	-
EQADC_BASE+0x08C	Reserved	-
EQADC_BASE+0x090	EQADC CFIFO Transfer Counter Register 0 (EQADC_CFTCR0)	Unrestricted
EQADC_BASE+0x094	EQADC CFIFO Transfer Counter Register 1 (EQADC_CFTCR1)	Unrestricted
EQADC_BASE+0x098	EQADC CFIFO Transfer Counter Register 2 (EQADC_CFTCR2)	Unrestricted
EQADC_BASE+0x09C	Reserved	-
EQADC_BASE+0x0A0	EQADC CFIFO Status Snapshot Register 0 (EQADC_CFSSR0)	Read only
EQADC_BASE+0x0A4	EQADC CFIFO Status Snapshot Register 1 (EQADC_CFSSR1)	Read only
EQADC_BASE+0x0A8	EQADC CFIFO Status Snapshot Register 2 (EQADC_CFSS2R)	Read only
EQADC_BASE+0x0AC	EQADC CFIFO Status Register (EQADC_CFSR)	Read only
EQADC_BASE+0x0B0	Reserved	-

Table 475. EQADC Memory Map (continued)

Address	Use	Access
EQADC_BASE+0x0B4	EQADC Synchronous Serial Interface Control Register (EQADC_SSICR)	Unrestricted
EQADC_BASE+0x0B8	EQADC Synchronous Serial Interface Receive Data Register (EQADC_SSIRDR)	Read Only
EQADC_BASE+0x0BC - EQADC_BASE+0x0CC	Reserved	-
EQADC_BASE+0x0D0	EQADC STAC bus Client Configuration Register (EQADC_REDLCCR)	Unrestricted
EQADC_BASE+0x0D4 - EQADC_BASE+0x0FC	Reserved	-
EQADC_BASE+0x100 - EQADC_BASE+0x10C	EQADC CFIFO0 Registers (EQADC_CF0Rw) (w=0, .., 3)	Read only
EQADC_BASE+0x110 - EQADC_BASE+0x11C	EQADC CFIFO0 Extension Registers (EQADC_CF0ERw) (w=0, .., 3)	Read only
EQADC_BASE+0x120 - EQADC_BASE+0x13C	Reserved	-
EQADC_BASE+0x140 - EQADC_BASE+0x14C	EQADC CFIFO1 Registers (EQADC_CF1Rw) (w=0, .., 3)	Read only
EQADC_BASE+0x150 - EQADC_BASE+0x17C	Reserved	-
EQADC_BASE+0x180 - EQADC_BASE+0x18C	EQADC CFIFO2 Registers (EQADC_CF2Rw) (w=0, .., 3)	Read only
EQADC_BASE+0x190 - EQADC_BASE+0x1BC	Reserved	-
EQADC_BASE+0x1C0 - EQADC_BASE+0x1CC	EQADC CFIFO3 Registers (EQADC_CF3Rw) (w=0, .., 3)	Read only
EQADC_BASE+0x1D0 - EQADC_BASE+0x1FC	Reserved	-
EQADC_BASE+0x200 - EQADC_BASE+0x20C	EQADC CFIFO4 Registers (EQADC_CF4Rw) (w=0, .., 3)	Read only
EQADC_BASE+0x210 - EQADC_BASE+0x23C	Reserved	-
EQADC_BASE+0x240 - EQADC_BASE+0x24C	EQADC CFIFO5 Registers (EQADC_CF5Rw) (w=0, .., 3)	Read only
EQADC_BASE+0x250 - EQADC_BASE+0x2FC	Reserved	-
EQADC_BASE+0x300 - EQADC_BASE+0x30C	EQADC RFIFO0 Registers (EQADC_RF0Rw) (w=0, .., 3)	Read only
EQADC_BASE+0x310 - EQADC_BASE+0x33C	Reserved	-
EQADC_BASE+0x340 - EQADC_BASE+0x34C	EQADC RFIFO1 Registers (EQADC_RF1Rw) (w=0, .., 3)	Read only
EQADC_BASE+0x350 - EQADC_BASE+0x37C	Reserved	-

Table 475. EQADC Memory Map (continued)

Address	Use	Access
EQADC_BASE+0x380 - EQADC_BASE+0x38C	EQADC RFIFO2 Registers (EQADC_RF2Rw) (w=0, .., 3)	Read only
EQADC_BASE+0x390 - EQADC_BASE+0x3BC	Reserved	-
EQADC_BASE+0x3C0 - EQADC_BASE+0x3CC	EQADC RFIFO3 Registers (EQADC_RF3Rw) (w=0, .., 3)	Read only
EQADC_BASE+0x3D0 - EQADC_BASE+0x3FC	Reserved	-
EQADC_BASE+0x400 - EQADC_BASE+0x40C	EQADC RFIFO4 Registers (EQADC_RF4Rw) (w=0, .., 3)	Read only
EQADC_BASE+0x410 - EQADC_BASE+0x43C	Reserved	-
EQADC_BASE+0x440 - EQADC_BASE+0x44C	EQADC RFIFO5 Registers (EQADC_RF5Rw) (w=0, .., 3)	Read only
EQADC_BASE+0x450 - EQADC_BASE+0x7FC	Reserved	-

23.5.2 EQADC Register Descriptions

EQADC Module Configuration Register (EQADC_MCR)

The EQADC Module Configuration Register (EQADC_MCR) contains bits used to control how the EQADC responds to a debug mode entry request, and to enable the EQADC SSI interface.

Register address: EQADC_BASE+0x000

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	ICEA0	ICEA1	0	ESSIE		0	DBG	
W	▨	▨	▨	▨	▨	▨	▨	▨			▨			▨		
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 446. EQADC Module Configuration Register (EQADC_MCR)

Table 476. EQADC Module Configuration Register (EQADC_MCR) field description

Field	Description
24-25 ICEA*n*	Immediate Conversion Command Enable ADC*n* (*n*=0,1) ICEA*n* enables the EQADC to abort on-chip ADC*n* current conversion and to start the immediate conversion command from CFIFO0 in the requested ADC*n*. 1 Enable immediate conversion command request. 0 Disable immediate conversion command request.
27-28 ESSIE[0:1]	EQADC Synchronous Serial Interface Enable Field The ESSIE field defines the EQADC synchronous serial interface operation according to *Table 477*.
30-31 DBG[0:1]	Debug enable The DBG field defines the EQADC response to a debug mode entry request as in *Table 478*.

Table 477. EQADC SSI Enable Field

ESSIE[0:1]	Meaning
0b00	EQADC SSI is disabled
0b01	Reserved
0b10	EQADC SSI is enabled, FCK is free running, and serial transmissions are disabled.
0b11	EQADC SSI is enabled, FCK is free running, and serial transmissions are enabled.

Note: Disabling the EQADC SSI (0b00 write to ESSIE) or serial transmissions from the EQADC SSI (0b10 write to ESSIE) while a serial transmission is in progress results in the abort of that transmission.

Note: When disabling the EQADC SSI, the FCK will not stop until it reaches its low phase.

Table 478. Debug Enable Field

DBG[0:1]	Meaning
0b00	Do not enter debug mode.
0b01	Reserved
0b10	Enter debug mode. If the EQADC SSI is enabled, FCK stops while the EQADC is in debug mode.
0b11	Enter debug mode. If the EQADC SSI is enabled, FCK is free running while the EQADC is in debug mode.

EQADC test register (EQADC_TST)

The EQADC Test Register (EQADC_TST) is used for test purposes only. This register can only be read/written in a test access, accessing the EQADC_TST register in any other way will result in a transfer error. In a non-test access to the EQADC_TST register, read data is undefined and written data is ignored.

Register address: EQADC_BASE+0x004

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 447. EQADC Test Register (EQADC_TST)

EQADC null message send format register (EQADC_NMSFR)

The EQADC Null Message Send Format Register (EQADC_NMSFR) defines the format of the null message sent to the external device.

Register address: EQADC_BASE+0x008

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	NMF									
W	▨	▨	▨	▨	▨	▨										
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	NMF															
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 448. EQADC null message send format register (EQADC_NMSFR)

Table 479. EQADC Null Message Send Format Register (EQADC_NMSFR) field description

Field	Description
6-31 NMF[0:25]	Null Message Format The NMF field contains the programmable null message send value for the EQADC. The value written to this register will be sent as a null message when serial transmissions from the EQADC SSI are enabled (ESSIE field configured to 0b11 in *Section , EQADC Module Configuration Register (EQADC_MCR)*) and either – there are no triggered CFIFOs with commands bound for external CBuffers, or; – there are triggered CFIFOs with commands bound for external CBuffers but the external CBuffers are full. Refer to *Section , Null Message Format for External Device Operation*, for more information on the format of a null message.

The EQADC Null Message Send Format Register only affects how the EQADC sends a null message, but it has no control on how the EQADC detects a null message on receiving data. The EQADC detects a null message by decoding the MESSAGE_TAG field on the receive data. Refer to page 927 for more information on the MESSAGE_TAG field.

Note: *Writing to the EQADC Null Message Send Format Register while the serial transmissions are enabled (ESSIE field configured to 0b11 in Section , EQADC Module Configuration Register (EQADC_MCR)) is not recommended.*

EQADC External Trigger Digital Filter Register (EQADC_ETDFR)

The EQADC External Trigger Digital Filter Register (EQADC_ETDFR) is used to set the minimum time a signal must be held in a logic state on the CFIFO triggers inputs to be recognized as an edge or level gated trigger. The Digital Filter Length field specifies the minimum number of system clocks that must be counted by the digital filter counter to recognize a logic state change. However, there is a control signal that can be used to bypass the digital filter when this is not needed.

Register address: EQADC_BASE+0x00C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	DFL			
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 449. EQADC External Trigger Digital Filter Register (EQADC_ETDFR)

Table 480. EQADC External Trigger Digital Filter Register (EQADC_ETDFR) field description

Field	Description
28-31 DFL[0:3]	Digital Filter Length The DFL field specifies the minimum number of system clocks that must be counted by the digital filter counter to recognize a logic state change. The count specifies the sample period of the digital filter which is calculated according to the following equation: $\mathrm{FilterPeriod} = (\mathrm{SystemClockPeriod} \times 2^{\mathrm{DFL}}) + 1(\mathrm{SystemClockPeriod})$ Minimum clock counts for which an ETRIG signal needs to be stable to be passed through the filter are shown in *Table 481*. Refer to *Section , External Trigger Event Detection*, for more information on the digital filter. The DFL field must only be written when the MODEx of all CFIFOs are configured to disabled.When the digital filter is bypassed by using the input control, the DFL is not considered and the trigger input signal is not filtered.

Table 481. Minimum Required Time to Valid ETRIG

DFL[0:3]	Minimum Clock Count	Minimum Time (ns) (system clock = 120 MHz)
0b0000	2	16.66
0b0001	3	25.00
0b0010	5	41.66
0b0011	9	75.00
0b0100	17	141.66
0b0101	33	275.00
0b0110	65	541.66
0b0111	129	1075.00
0b1000	257	2141.66
0b1001	513	4275.00
0b1010	1025	8541.66
0b1011	2049	17075.00
0b1100	4097	34141.00
0b1101	8193	68275.00
0b1110	16385	136541.66
0b1111	32769	273075.00

EQADC CFIFO Push Registers (EQADC_CFPR)

The EQADC CFIFO Push Registers (EQADC_CFPR) provide a mechanism to fill the CFIFOs with command messages from the CQueues. Refer to *Section 23.6.4, EQADC Command FIFOs*, for more information on the CFIFOs and to *Section , Message Format in EQADC*, for a description on command message formats.

Register address: EQADC_BASE+0x010
Register address: EQADC_BASE+0x014
Register address: EQADC_BASE+0x018
Register address: EQADC_BASE+0x01C
Register address: EQADC_BASE+0x020
Register address: EQADC_BASE+0x024

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	CF_PUSHx															
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	CF_PUSHx															
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 450. EQADC CFIFO Push Register x (EQADC_CFPRx)

Table 482. EQADC CFIFO Push Register x (EQADC_CFPRx) field description

Field	Description
0-31 CF_PUSHx [0:31]	CFIFO Push Data x When CFIFOx is not full, writing to the whole word or any bytes of EQADC_CFPRx will push the 32-bit CF_PUSHx value into CFIFOx. Writing to the CF_PUSHx field also increments the corresponding CFCTRx value by one in *Section , EQADC FIFO and Interrupt Status Registers (EQADC_FISR)*. When the CFIFOx is full, the EQADC ignores any write to the CF_PUSHx. Reading the EQADC_CFPRx always returns zero. Only whole words must be written to EQADC_CFPR. Writing half-words or bytes to EQADC_CFPR will still push the whole 32-bit CF_PUSH field into the corresponding CFIFO, but undefined data will fill the areas of CF_PUSH that were not specifically designated as target locations for the write.

EQADC Result FIFO Pop Registers (EQADC_RFPR)

The EQADC Result FIFO Pop Registers (EQADC_RFPR) provide a mechanism to retrieve data from RFIFOs.

Register address: EQADC_BASE+0x030
Register address: EQADC_BASE+0x034
Register address: EQADC_BASE+0x038
Register address: EQADC_BASE+0x03C
Register address: EQADC_BASE+0x040
Register address: EQADC_BASE+0x044

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	RF_POPx															
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 451. EQADC RFIFO Pop Register x (EQADC_RFPRx)

Table 483. EQADC RFIFO Pop Register x (EQADC_RFPRx) field description

Field	Description
16-31 RF_POPx [0:15]	Result FIFO Pop Data x When RFIFOx is not empty, the RF_POPx contains the next unread entry value of RFIFOx. Reading a word, a half-word, or any bytes from EQADC_RFPRx will pop one entry from RFIFOx and cause the RFCTRx field to be decremented by one in the *Section , EQADC FIFO and Interrupt Status Registers (EQADC_FISR)*. When the RFIFOx is empty, any read on EQADC_RFPRx returns undefined data value and does not decrement the RFCTRx value. Writing to EQADC_RFPRx has no effect.

EQADC CFIFO Control Registers (EQADC_CFCR)

The EQADC CFIFO Control Registers (EQADC_CFCR) contain bits that affect CFIFOs. These bits specify the CFIFO operation mode and can invalidate all of the CFIFO contents.

Register address: EQADC_BASE+0x050

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	CFEE0	STRME0	0	0	0	MODE0				AMODE0			
W	▨	▨	▨			SSE0	CFINV0	▨								
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	MODE1				0	0	0	0
W	▨	▨	▨	▨	▨	SSE1	CFINV1	▨					▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 452. EQADC CFIFO Control Register 0 (EQADC_CFCR0)

Register address: EQADC_BASE+0x054

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	MODE2				0	0	0	0
W	▨	▨	▨	▨	▨	SSE2	CFINV2	▨					▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	MODE3				0	0	0	0
W	▨	▨	▨	▨	▨	SSE3	CFINV3	▨					▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 453. EQADC CFIFO Control Register 1 (EQADC_CFCR1)

Register address: EQADC_BASE+0x058

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	MODE4				0	0	0	0
W						SSE 4	CFIN V4									
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	MODE5				0	0	0	0
W						SSE 5	CFIN V5									
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 454. EQADC CFIFO Control Register 2 (EQADC_CFCR2)

Table 484. EQADC CFIFO Control Register x (EQADC_CFCRx) field description

Field	Description
3 CFEEE0	CFIFO0 Entry Number Extension Enable The CFEEE0 bit is used to enable the extension of the CFIFO0 entries. When in extended mode, the CFIFO0 total entries is the sum of normal entries plus the defined value for extension. For more details, refer to *Section , CFIFO0 Streaming Mode Description*. 1 Enable the extension of CFIFO0 entries. 0 CFIFO0 has a normal value of entries.
4 STRME0	CFIFO0 Streaming Mode Operation Enable The STRME0 bit is used to enable the streaming mode of operation of CFIFO0. In this case, it is possible to repeat some sequence of commands of this FIFO. For more details, refer to *Section , CFIFO0 Streaming Mode Description*. 1 Enable the streaming mode of CFIFO0. 0 Streaming mode of CFIFO0 is disabled.
5 SSEx	CFIFO Single-Scan Enable Bit x The SSEx bit is used to set the SSSx bit in *Section , EQADC FIFO and Interrupt Status Registers (EQADC_FISR)*. Writing a “1” to SSEx will set the SSSx in *Section , EQADC FIFO and Interrupt Status Registers (EQADC_FISR)*, if the CFIFO is in single-scan mode. When SSSx is already asserted, writing a “1” to SSEx also has no effect. If the CFIFO is in continuous-scan mode or is disabled, writing a “1” to SSEx will not set SSSx. Writing a “0” to SSEx has no effect. SSEx always is read as “0”. 1 Set the SSSx bit. 0 No effect.

Table 484. EQADC CFIFO Control Register x (EQADC_CFCRx) field description (continued)

Field	Description
6 CFINVx	CFIFO Invalidate Bit x The CFINVx bit causes the EQADC to invalidate all entries of CFIFOx. Writing a “1” to CFINVx will reset the value of CFCTRx in *Section , EQADC FIFO and Interrupt Status Registers (EQADC_FISR)*. Writing a “1” to CFINVx also resets the Push Next Data Pointer, Transfer Next Data Pointer to the first entry of CFIFOx in *Figure 504*. CFINVx always is read as “0”. Writing a “0” has no effect. Invalidate all of the entries in the corresponding CFIFO. No effect. Writing CFINVx only invalidates commands stored in CFIFOx; previously transferred commands that are waiting for execution, that is commands stored in the CBuffers, will still be executed, and results generated by them will be stored in the appropriate RFIFO. CFINVx must not be written unless the MODEx is configured to disabled, and CFIFO status is IDLE.
MODEx [0:3]	CFIFO Operation Mode x The MODEx field selects the CFIFO operation mode for CFIFOx, see *Table 485*. Refer to *Section , CFIFO Scan Trigger Modes*, for more information on CFIFO trigger mode. If MODEx is not disabled, it must not be changed to any other mode besides disabled. If MODEx is disabled and the CFIFO status is IDLE, MODEx can be changed to any other mode.
12-15 AMODE0 [0:3]	CFIFO0 Advance Trigger Operation Mode 0 The AMODE0 field selects the trigger mode for the ATRIG0 trigger signal in streaming mode, see *Table 486*. The use of reserved values drives to unknown behavior of the block. If AMODE0 is not disabled, it must not be changed to any other mode besides disabled. If AMODE0 is disabled and the CFIFO0 status is IDLE, AMODE0 can be changed to any other mode. For the streaming mode of operation when the ATRIG0 is used to enable the ETRIG0 or to advance the command queue, the normal mode of operation is external trigger single scan. Other settings are not fully tested.

Table 485. CFIFO Operation Mode Table

MODE*x*[0:3]	CFIFO Operation Mode
0b0000	Disabled
0b0001	Software Trigger, Single Scan
0b0010	Low Level Gated External Trigger, Single Scan
0b0011	High Level Gated External Trigger, Single Scan
0b0100	Falling Edge External Trigger, Single Scan
0b0101	Rising Edge External Trigger, Single Scan
0b0110	Falling or Rising Edge External Trigger, Single Scan
0b0111 - 0b1000	Reserved
0b1001	Software Trigger, Continuous Scan
0b1010	Low Level Gated External Trigger, Continuous Scan

Table 485. CFIFO Operation Mode Table (continued)

MODE*x*[0:3]	CFIFO Operation Mode
0b1011	High Level Gated External Trigger, Continuous Scan
0b1100	Falling Edge External Trigger, Continuous Scan
0b1101	Rising Edge External Trigger, Continuous Scan
0b1110	Falling or Rising Edge External Trigger, Continuous Scan
0b1111	Reserved

Table 486. CFIFO0 Advance Trigger Operation Mode Table

AMODE0[0:3]	CFIFO0 Advance Trigger Operation Mode
0b0000	Disabled
0b0001	Reserved
0b0010	Reserved
0b0011	Reserved
0b0100	Falling Edge External Trigger, Single Scan
0b0101	Rising Edge External Trigger, Single Scan
0b0110	Falling or Rising Edge External Trigger, Single Scan
0b0111 - 0b1111	Reserved

EQADC Interrupt and DMA Control Registers (EQADC_IDCR)

The EQADC Interrupt Control Registers (EQADC_IDCR) contain bits to enable the generation of interrupt or DMA requests when the corresponding flag bits are set in *Section , EQADC FIFO and Interrupt Status Registers (EQADC_FISR)*.

Register address: EQADC_BASE+0x060

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	NCIE0	TORIE0	PIE0	EOQIE0	CFUIE0	0	CFFE0	CFFS0	0	0	0	0	RFOIE0	0	RFDE0	RFDS0
W						▨			▨	▨	▨	▨		▨		
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	NCIE1	TORIE1	PIE1	EOQIE1	CFUIE1	0	CFFE1	CFFS1	0	0	0	0	RFOIE1	0	RFDE1	RFDS1
W						▨			▨	▨	▨	▨		▨		
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 455. EQADC Interrupt and DMA Control Register 0 (EQADC_IDCR0)

Register address: EQADC_BASE+0x064

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	NCIE 2	TORI E2	PIE2	EOQ IE2	CFUI E2	0	CFF E2	CFF S2	0	0	0	0	RFOI E2	0	RFD E2	RFD S2
W						▨			▨	▨	▨	▨		▨		
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	NCIE 3	TORI E3	PIE3	EOQ IE3	CFUI E3	0	CFF E3	CFF S3	0	0	0	0	RFOI E3	0	RFD E3	RFD S3
W						▨			▨	▨	▨	▨		▨		
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 456. EQADC Interrupt and DMA Control Register 1 (EQADC_IDCR1)

Register address: EQADC_BASE+0x068

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	NCIE 4	TORI E4	PIE4	EOQ IE4	CFUI E4	0	CFF E4	CFF S4	0	0	0	0	RFOI E4	0	RFD E4	RFD S4
W						▨			▨	▨	▨	▨		▨		
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	NCIE 5	TORI E5	PIE5	EOQ IE5	CFUI E5	0	CFF E5	CFF S5	0	0	0	0	RFOI E5	0	RFD E5	RFD S5
W						▨			▨	▨	▨	▨		▨		
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 457. EQADC Interrupt and DMA Control Register 2 (EQADC_IDCR2)

Table 487. EQADC Interrupt and DMA Control Register x (EQADC_IDCRx) field description

Field	Description
NCIEx	Non-Coherency Interrupt Enable x NCIEx enables the EQADC to generate an interrupt request when the corresponding NCFx in *Section , EQADC FIFO and Interrupt Status Registers (EQADC_FISR)*, is asserted. 1 Enable non-coherency interrupt request. 0 Disable non-coherency interrupt request.
TORIEx	Trigger Overrun Interrupt Enable x TORIEx enables the EQADC to generate an interrupt request when the corresponding TORFx in *Section , EQADC FIFO and Interrupt Status Registers (EQADC_FISR)*, is asserted. Apart from generating an independent interrupt request for a CFIFOx Trigger Overrun event, the EQADC also provides a combined interrupt at which the Result FIFO Overflow Interrupt, the Command FIFO Underflow Interrupt, and the Command FIFO Trigger Overrun Interrupt requests of ALL CFIFOs are ORed. When RFOIEx, CFUIEx, and TORIEx are all asserted, this combined interrupt request is asserted whenever one of the following 18 flags becomes asserted: RFOFx, CFUFx, and TORFx (assuming that all interrupts are enabled). See *Section 23.6.8, EQADC DMA/Interrupt request*, for details. 1 Enable trigger overrun interrupt request. 0 Disable trigger overrun interrupt request.
PIEx	Pause Interrupt Enable x PIEx enables the EQADC to generate an interrupt request when the corresponding PFx in *Section , EQADC FIFO and Interrupt Status Registers (EQADC_FISR)*, is asserted. 1 Enable pause interrupt request. 0 Disable pause interrupt request.
EOQIEx	End of Queue Interrupt Enable x EOQIEx enables the EQADC to generate an interrupt request when the corresponding EOQFx in *Section , EQADC FIFO and Interrupt Status Registers (EQADC_FISR)*, is asserted. 1 Enable End of Queue interrupt request. 0 Disable End of Queue interrupt request.
CFUIEx	CFIFO Underflow Interrupt Enable x CFUIEx enables the EQADC to generate an interrupt request when the corresponding CFUFx in *Section , EQADC FIFO and Interrupt Status Registers (EQADC_FISR)*, is asserted. Apart from generating an independent interrupt request for a CFIFOx underflow event, the EQADC also provides a combined interrupt at which the Result FIFO Overflow Interrupt, the Command FIFO Underflow Interrupt, and the Command FIFO Trigger Overrun Interrupt requests of ALL CFIFOs are ORed. When RFOIEx, CFUIEx, and TORIEx are all asserted, this combined interrupt request is asserted whenever one of the following 18 flags becomes asserted: RFOFx, CFUFx, and TORFx (assuming that all interrupts are enabled). See *Section 23.6.8, EQADC DMA/Interrupt request*, for details. 1 Enable Underflow Interrupt request. 0 Disable Underflow Interrupt request.

Table 487. EQADC Interrupt and DMA Control Register x (EQADC_IDCRx) field description

Field	Description
CFFEx	CFIFO Fill Enable x CFFEx enables the EQADC to generate an interrupt request (CFFSx is asserted) or DMA request (CFFSx is negated) when CFFFx in *Section , EQADC FIFO and Interrupt Status Registers (EQADC_FISR)*, is asserted. 1 Enable CFIFO Fill DMA or Interrupt request. 0 Disable CFIFO Fill DMA or Interrupt request. CFFEx must not be negated while a DMA transaction is in progress.
CFFSx	CFIFO Fill Select x CFFSx selects if a DMA or interrupt request is generated when CFFFx in *Section , EQADC FIFO and Interrupt Status Registers (EQADC_FISR)*, is asserted. If CFFEx is asserted, the EQADC generates an interrupt request when CFFSx is negated, or it generates a DMA request if CFFSx is asserted. 1 Generate DMA request to move data from the system memory to CFIFOx. 0 Generate interrupt request to move data from the system memory to CFIFOx. CFFSx must not be negated while a DMA transaction is in progress.
RFOIEx	RFIFO Overflow Interrupt Enable x RFOIEx enables the EQADC to generate an interrupt request when the corresponding RFOFx in *Section , EQADC FIFO and Interrupt Status Registers (EQADC_FISR)*, is asserted. Apart from generating an independent interrupt request for an RFIFOx overflow event, the EQADC also provides a combined interrupt at which the Result FIFO Overflow Interrupt, the Command FIFO Underflow Interrupt, and the Command FIFO Trigger Overrun Interrupt requests of ALL CFIFOs are ORed. When RFOIEx, CFUIEx, and TORIEx are all asserted, this combined interrupt request is asserted whenever one of the following 18 flags becomes asserted: RFOFx, CFUFx, and TORFx (assuming that all interrupts are enabled). See *Section 23.6.8, EQADC DMA/Interrupt request*, for details. 1 Enable Overflow Interrupt request. 0 Disable Overflow Interrupt request.

Table 487. EQADC Interrupt and DMA Control Register x (EQADC_IDCRx) field description

Field	Description
RFDEx	RFIFO Drain Enable x RFDEx enables the EQADC to generate an interrupt request (RFDSx is asserted) or DMA request (RFDSx is negated) when RFDFx in *Section , EQADC FIFO and Interrupt Status Registers (EQADC_FISR)*, is asserted. 1 Enable RFIFO Drain DMA or Interrupt request. 0 Disable RFIFO Drain DMA or Interrupt request. RFDEx must not be negated while a DMA transaction is in progress.
RFDSx	RFIFO Drain Select x RFDSx selects if a DMA or interrupt request is generated when RFDFx in *Section , EQADC FIFO and Interrupt Status Registers (EQADC_FISR)*, is asserted. If RFDEx is asserted, the EQADC generates an interrupt request when RFDSx is negated, or it generates a DMA request when RFDSx is asserted. 1 Generate DMA request to move data from RFIFOx to the system memory. 0 Generate interrupt request to move data from RFIFOx to the system memory. RFDSx must not be negated while a DMA transaction is in progress.

EQADC FIFO and Interrupt Status Registers (EQADC_FISR)

The EQADC FIFO and Interrupt Status Registers (EQADC_FISR) contain flag and status bits for each CFIFO and RFIFO pair. Write “1” to a flag bit to clear it. Writing “0” has no effect. Status bits are read only. These bits indicate the status of the FIFO itself.

Register address: EQADC_BASE+0x070
Register address: EQADC_BASE+0x074
Register address: EQADC_BASE+0x078
Register address: EQADC_BASE+0x07C
Register address: EQADC_BASE+0x080
Register address: EQADC_BASE+0x084

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	NCFx	TORFx	PFx	EOQFx	CFUFx	SSSx	CFFFx	0	0	0	0	0	RFOFx	0	RFDFx	0
W						▨		▨	▨	▨	▨	▨		▨		▨
RESET:	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CFCTRx				TNXTPTRx				RFCTRx				POPNXTPTRx			
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 458. EQADC FIFO and Interrupt Status Register x (EQADC_FISRx)

Table 488. EQADC FIFO and Interrupt Status Register x (EQADC_FISRx) field description

Field	Description
0 NCFx	Non-Coherency Flag NCFx is set whenever a command sequence being transferred through CFIFOx becomes non coherent. If NCIEx in *Section , EQADC Interrupt and DMA Control Registers (EQADC_IDCR)*, and NCF*x* are asserted, an interrupt request will be generated. Write “1” to clear NCFx. Writing a “0” has no effect. For more information refer to *Section , Command Sequence Non-Coherency Detection*. 1 Command sequence being transferred by CFIFOx became non-coherent. 0 Command sequence being transferred by CFIFOx is coherent.
1 TORFx	Trigger Overrun Flag for CFIFOx TORFx is set when trigger overrun occurs for the specified CFIFO in edge or level trigger mode. Trigger overrun occurs when an already triggered CFIFO receives an additional trigger. When TORIE*x* in *Section , EQADC Interrupt and DMA Control Registers (EQADC_IDCR)*, and TORF*x* are asserted, an interrupt request will be generated. Apart from generating an independent interrupt request for a CFIFOx Trigger Overrun event, the EQADC also provides a combined interrupt at which the Result FIFO Overflow Interrupt, the Command FIFO Underflow Interrupt, and the Command FIFO Trigger Overrun Interrupt requests of ALL CFIFOs are ORed. When RFOIEx, CFUIEx, and TORIEx are all asserted, this combined interrupt request is asserted whenever one of the following 18 flags becomes asserted: RFOFx, CFUFx, and TORFx (assuming that all interrupts are enabled). See *Section 23.6.8, EQADC DMA/Interrupt request*, for details. Write “1” to clear the TORF*x* bit. Writing a “0” has no effect. 1 Trigger overrun occurred. 0 No trigger overrun occurred. The trigger overrun flag will not set for CFIFOs configured for software trigger mode.

Table 488. EQADC FIFO and Interrupt Status Register x (EQADC_FISRx) field description

Field	Description
2 PFx	Pause Flag x PF behavior changes according to the CFIFO trigger mode. In edge trigger mode, PFx is set when the EQADC completes the transfer of an entry with an asserted Pause bit from CFIFOx. In level trigger mode, when CFIFOx is in TRIGGERED status, PFx is set when CFIFO status changes from TRIGGERED due to the detection of a closed gate. An interrupt routine, generated due to the asserted PF, can be used to verify if a complete scan of the CQueue was performed. If a closed gate is detected while no command transfers are taking place, it will have immediate effect on the CFIFO status. If a closed gate is detected while a command transfer to an on-chip CBuffer is taking place, it will only affect the CFIFO status when the transfer completes. If a closed gate is detected during the serial transmission of a command to the external device, it will have no effect on the CFIFO status until the transmission completes. The transfer of entries bound for the on-chip ADCs is considered completed when they are stored in the appropriate CBuffer. The transfer of entries bound for the external device is considered completed when the serial transmission of the entry is completed. In software trigger mode, PFx will never become asserted. If PIE*x* in *Section , EQADC Interrupt and DMA Control Registers (EQADC_IDCR)*, and PFx are asserted, an interrupt will be generated. Write “1” to clear the PF*x*. Writing a “0” has no effect. Refer to *Section , Pause Status*, for more information on the Pause Flag. 1 Entry with asserted PAUSE bit was transferred from CFIFOx (CFIFO in edge trigger mode), or CFIFO status changes from TRIGGERED due to detection of a closed gate (CFIFO in level trigger mode). 0 Entry with asserted PAUSE bit was not transferred from CFIFOx (CFIFO in edge trigger mode), or CFIFO status did not change from TRIGGERED due to detection of a closed gate (CFIFO in level trigger mode). In edge trigger mode, an asserted PFx only implies that the EQADC has finished transferring a command with an asserted PAUSE bit from CFIFOx. It does not imply that result data for the current command and for all previously transferred commands has been returned to the appropriate RFIFO. In software or level trigger mode, when the EQADC completes the transfer of an entry from CFIFOx with an asserted PAUSE bit, PFx will not be set and transfer of commands will continue without pausing.
3 EOQFx	End of Queue Flag x EOQFx indicates that an entry with an asserted EOQ bit was transferred from CFIFOx to the on-chip ADCs or to the external device - see *Section , Message Format in EQADC*, for details about command message formats. When the EQADC completes the transfer of an entry with an asserted EOQ bit from CFIFOx, EOQFx will be set. The transfer of entries bound for the on-chip ADCs is considered completed when they are stored in the appropriate CBuffer. The transfer of entries bound for the external device is considered completed when the serial transmission of the entry is completed. If the EOQIEx bit in *Section , EQADC Interrupt and DMA Control Registers (EQADC_IDCR)*, and EOQFx are asserted, an interrupt will be generated. Write “1” to clear the EOQFx bit. Writing a “0” has no effect. Refer to *Section , CQueue Completion Status*, for more information on the End of Queue Flag. 1 Entry with asserted EOQ bit was transferred from CFIFOx. 0 Entry with asserted EOQ bit was not transferred from CFIFOx. An asserted EOQFx only implies that the EQADC has finished transferring a command with an asserted EOQ bit from CFIFOx. It does not imply that result data for the current command and for all previously transferred commands has been returned to the appropriate RFIFO.

Table 488. EQADC FIFO and Interrupt Status Register x (EQADC_FISRx) field description

Field	Description
4 CFUFx	CFIFO Underflow Flag x CFUFx indicates an underflow event on CFIFOx. CFUFx is set when CFIFOx is in TRIGGERED state and it becomes empty. No commands will be transferred from an underflowing CFIFO, nor will command transfers from lower priority CFIFOs be blocked. When CFUIEx in *Section , EQADC Interrupt and DMA Control Registers (EQADC_IDCR)*, and CFUFx are both asserted, the EQADC generates an interrupt request. Apart from generating an independent interrupt request for a CFIFOx underflow event, the EQADC also provides a combined interrupt at which the Result FIFO Overflow Interrupt, the Command FIFO Underflow Interrupt, and the Command FIFO Trigger Overrun Interrupt requests of ALL CFIFOs are ORed. When RFOIEx, CFUIEx, and TORIEx are all asserted, this combined interrupt request is asserted whenever one of the following 18 flags becomes asserted: RFOFx, CFUFx, and TORFx (assuming that all interrupts are enabled). See *Section 23.6.8, EQADC DMA/Interrupt request*, for details. Write “1” to clear CFUFx. Writing a “0” has no effect. 1 A CFIFO underflow event occurred. 0 No CFIFO underflow event occurred.
5 SSSx	CFIFO Single-Scan Status Bit x An asserted SSSx bit enables the detection of trigger events for CFIFOs programmed into single-scan level- or edge-trigger mode, and works as trigger for CFIFOs programmed into single-scan software-trigger mode. Refer to *Section , Single-Scan Mode*, for further details. The SSSx bit is set by writing a “1” to the SSEx bit in *Section , EQADC CFIFO Control Registers (EQADC_CFCR)*. The EQADC clears the SSSx bit when a command with an asserted EOQ bit is transferred from a CFIFO in single-scan mode, when a CFIFO is in single-scan level-trigger mode and its status changes from TRIGGERED due to the detection of a closed gate, or when the value of the CFIFO operation mode (MODEx) in *Section , EQADC CFIFO Control Registers (EQADC_CFCR)*, is changed to disabled. Writing to SSSx has no effect. SSSx has no effect in continuous-scan or in disabled mode. 1CFIFO in single-scan level- or edge-trigger mode will detect a trigger event, or CFIFO in single-scan software-trigger mode is triggered. 0CFIFO in single-scan level- or edge-trigger mode will ignore trigger events, or CFIFO in single-scan software-trigger mode is not triggered.
6 CFFFx	CFIFO Fill Flag x CFFFx is set when the CFIFOx is not full. When CFFE*x* in *Section , EQADC Interrupt and DMA Control Registers (EQADC_IDCR)*, and CFFFx are both asserted, an interrupt or a DMA request will be generated depending on the status of the CFFSx bit. When CFFSx is negated (interrupt requests selected), software clears CFFFx by writing a “1” to it. Writing a “0” has no effect. When CFFSx is asserted (DMA requests selected), CFFFx is automatically cleared by the EQADC when the CFIFO becomes full. 1 CFIFOx is not full. 0 CFIFOx is full. Writing “1” to CFFFx when CFFSx is asserted (DMA requests selected) is not allowed. When generation of interrupt requests is selected (CFFSx=0), CFFFx must only be cleared in the ISR after the CFIFOx push register is accessed.

Table 488. EQADC FIFO and Interrupt Status Register x (EQADC_FISRx) field description

Field	Description
12 RFOFx	RFIFO Overflow Flag x RFOFx indicates an overflow event on RFIFOx. RFOF*x* is set when RFIFOx is already full, and a new data is received from the on-chip ADCs or from the external device. The RFIFOx will not overwrite older data in the RFIFO, and the new data will be ignored. When RFOIEx in *Section , EQADC Interrupt and DMA Control Registers (EQADC_IDCR)*, and RFOFx are both asserted, the EQADC generates an interrupt request. Apart from generating an independent interrupt request for an RFIFOx overflow event, the EQADC also provides a combined interrupt at which the Result FIFO Overflow Interrupt, the Command FIFO Underflow Interrupt, and the Command FIFO Trigger Overrun Interrupt requests of ALL CFIFOs are ORed. When RFOIEx, CFUIEx, and TORIEx are all asserted, this combined interrupt request is asserted whenever one of the following 18 flags becomes asserted: RFOFx, CFUFx, and TORFx (assuming that all interrupts are enabled). See *Section 23.6.8, EQADC DMA/Interrupt request*, for details. Write “1” to clear RFOFx. Writing a “0” has no effect. 1 An RFIFO overflow event occurred. 0 No RFIFO overflow event occurred.
14 RFDF*x*	RFIFO Drain Flag x RFDFx indicates if RFIFOx has valid entries or not. RFDFx is set when the RFIFOx has at least one valid entry in it. When RFDEx in *Section , EQADC Interrupt and DMA Control Registers (EQADC_IDCR)*, and RFDFx are both asserted, an interrupt or a DMA request will be generated depending on the status of the RFDSx bit. When RFDSx is negated (interrupt requests selected), software clears RFDFx by writing a “1” to it. Writing a “0” has no effect. When RFDSx is asserted (DMA requests selected), RFDFx is automatically cleared by the EQADC when the RFIFO becomes empty. 1 RFIFOx has at least one valid entry. 0 RFIFOx is empty. Writing “1” to RFDFx when RFDSx is asserted (DMA requests selected) is not allowed. When the generation of interrupt requests is selected (RFDSx=0), RFDFx must only be cleared in the ISR after the RFIFOx pop register is accessed.
16-19 CFCTRx [0:3]	CFIFOx Entry Counter CFCTRx indicates the number of commands stored in the CFIFOx. When the EQADC completes transferring a piece of new data from the CFIFOx, it decrements CFCTRx by one. Writing a word or any bytes to the corresponding *Section , EQADC CFIFO Push Registers (EQADC_CFPR)*, increments CFCTRx by one. Writing any value to CFCTRx has no effect.
20-23 TNXTPTRx [0:3]	CFIFOx Transfer Next Pointer TNXTPTRx indicates the index of the next entry to be removed from CFIFOx when it completes a transfer. When TNXTPTRx is zero, it points to the entry with the smallest memory-mapped address inside CFIFOx. TNXTPTRx is only updated when a command transfer is completed. If the maximum index number (CFIFO depth minus one) is reached, TNXTPTRx is wrapped to zero, else, it is incremented by one. For details refer to *Section , CFIFO Basic Functionality*. Writing any value to TNXTPTRx has no effect.

Table 488. EQADC FIFO and Interrupt Status Register x (EQADC_FISRx) field description

Field	Description
24-27 RFCTRx [0:3]	RFIFOx Entry Counter RFCTRx indicates the number of data items stored in the RFIFOx. When the EQADC stores a piece of new data into RFIFOx, it increments RFCTRx by one. Reading the whole word, half-word or any bytes of the corresponding *Section , EQADC Result FIFO Pop Registers (EQADC_RFPR)*, decrements RFCTRx by one. Writing any value to RFCTRx itself has no effect.
28-31 POPNXTPTRx [0:3]	RFIFOx Pop Next Pointer POPNXTPTRx indicates the index of the entry that will be returned when EQADC_RFPRx is read. When POPNXTPTRx is zero, it points to the entry with the smallest memory-mapped address inside RFIFOx. POPNXTPTRx is updated when EQADC_RFPRx is read. If the maximum index number (RFIFO depth minus one) is reached, POPNXTPTRx is wrapped to zero, else, it is incremented by one. For details refer to *Section , RFIFO Basic Functionality*. Writing any value to POPNXTPTRx has no effect.

EQADC CFIFO Transfer Counter Registers (EQADC_CFTCR)

The EQADC CFIFO Transfer Counter Registers (EQADC_CFTCR) record the number of commands transferred from a CFIFO. The EQADC_CFTCR supports the monitoring of command transfers from a CFIFO.

Register address: EQADC_BASE+0x090

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	TC_CF0										
W	▨	▨	▨	▨	▨											
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	TC_CF1										
W	▨	▨	▨	▨	▨											
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 459. EQADC CFIFO Transfer Counter Register 0 (EQADC_CFTCR0)

Register address: EQADC_BASE+0x094

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	TC_CF2										
W	▨	▨	▨	▨	▨											
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	TC_CF3										
W	▨	▨	▨	▨	▨											
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 460. EQADC CFIFO Transfer Counter Register 1 (EQADC_CFTCR1)

Register address:EQADC_BASE+0x098

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	TC_CF4										
W	▨	▨	▨	▨	▨											
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	TC_CF5										
W	▨	▨	▨	▨	▨											
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 461. EQADC CFIFO Transfer Counter Register 2 (EQADC_CFTCR2)

Table 489. EQADC CFIFO Transfer Counter Register x (EQADC_CFTCRx) field description

Field	Description
TC_CFx [0:10]	Transfer Counter for CFIFOx TC_CFx counts the number of commands which have been completely transferred from CFIFOx. The transfer of entries bound for the on-chip ADCs is considered completed when they are stored in the appropriate CBuffer. The transfer of entries bound for an external device is considered completed when the serial transmission of the entry is completed. The EQADC increments the TC_CFx value by one after a command is transferred. TC_CFx resets to zero after EQADC completes transferring a command with an asserted EOQ bit. Writing any value to TC_CFx sets the counter to that written value. If CFIFOx is in TRIGGERED state when its MODEx field is programmed to disabled, the exact number of entries transferred from the CFIFO until that point - TC_CFx - is only known after the CFIFO status changes to IDLE, as indicated by CFSx. For details refer to *Section , Disabled Mode*.

EQADC CFIFO Status Snapshot Registers (EQADC_CFSSR)

The EQADC CFIFO Status Snapshot Registers (EQADC_CFSSR) contain status fields to track the operation status of each CFIFO and the transfer counter of the last CFIFO to initiate a command transfer to the internal/external CBuffers. EQADC_CFSSR0-1 are related to the on-chip CBuffers (CBuffer0-1) while EQADC_CFSSR2 is related to the external CBuffers. All fields of a particular EQADC_CFSSR register are captured at the beginning of a command transfer to the CBuffer associated with that register. Note that captured status register values are associated with previous command transfer. This means that the CFSSR registers capture the status registers before the status registers change because of the transfer of the current command that is about to be popped from the CFIFO. The EQADC_CFSSR registers are read only. Writing to the EQADC_CFSSR registers has no effect.

Register address: EQADC_BASE+0x0A0

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CFS0_TCB 0		CFS1_TCB 0		CFS2_TCB 0		CFS3_TCB 0		CFS4_TCB 0		CFS5_TCB 0		0	0	0	0
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	LCFTCB0				TC_LCFTCB0										
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 462. EQADC CFIFO Status Snapshot Register 0 (EQADC_CFSSR0)

Register address: EQADC_BASE+0x0A4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CFS0_TCB1		CFS1_TCB1		CFS2_TCB1		CFS3_TCB1		CFS4_TCB1		CFS5_TCB1		0	0	0	0
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	LCFTCB1				TC_LCFTCB1										
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 463. EQADC CFIFO Status Snapshot Register 1 (EQADC_CFSSR1)

Register address: EQADC_BASE+0x0A8

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CFS0_TSSI		CFS1_TSSI		CFS2_TSSI		CFS3_TSSI		CFS4_TSSI		CFS5_TSSI		0	0	0	0
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	ECBNI	LCFTSSI				TC_LCFTSSI										
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 464. EQADC CFIFO Status Snapshot Register 2 (EQADC_CFSSR2)

Table 490. EQADC CFIFO Status Snapshot Register x (EQADC_CFSSRx) field description

Field	Description
CFSx_TCB*n* [0:1]	CFIFO Status at Transfer to CBuffer*n (n=0,1)* CFSx_TCB*n* indicates the CFIFOx status of previously completed command transfer. CFSx_TCB*n* is a copy of the corresponding CFSx in the *Section , EQADC CFIFO Status Register (EQADC_CFSR)*, captured at the time a current command transfer to CBuffer*n* is initiated.
LCFTCB*n* [0:3]	Last CFIFO to Transfer to CBuffer*n (n=0,1)* LCFTCB*n* holds the CFIFO number to have completed a previous command transfer to CBuffer*n*, see *Table 491*.
TC_LCFTCB*n* [0:10]	Transfer Counter for Last CFIFO to transfer commands to CBuffer*n* TC_LCFTCB*n* indicates the number of commands which have been completely transferred from CFIFOx when a current command transfer from CFIFOx to CBuffer*n* is initiated. TC_LCFTCB*n* is a copy of the corresponding TC_CFx in the *Section , EQADC CFIFO Transfer Counter Registers (EQADC_CFTCR)*, captured at the time a current command transfer from CFIFOx to CBuffer*n* is initiated. This field has no meaning when LCFTCB*n* is 0b1111.
CFSx_TSSI [0:1]	CFIFO Status at Transfer through the EQADC SSI CFSx_TSSI indicates the CFIFOx status of previously completed serial transmission through the EQADC SSI. CFSx_TSSI is a copy of the corresponding CFSx in the *Section , EQADC CFIFO Status Register (EQADC_CFSR)*, capture at the time a current serial transmission through the EQADC SSI is initiated.
ECBNI	External CBuffer Number Indicator ECBNI indicates to which external CBuffer the previous command was transmitted. 1 Last command was transferred to CBuffer3. 0 Last command was transferred to CBuffer2.
LCFTSSI[0:3]	Last CFIFO to Transfer Commands through the EQADC SSI. LCFTSSI holds the CFIFO number to have completed a previous command transfer to an external CBuffer through the EQADC SSI, see *Table 492*. LCFTSSI does not indicate the transmission of null messages.
TC_LCFSSI [0:10]	Transfer Counter for Last CFIFO to Transfer commands through EQADC SSI TC_LCFTSSI indicates the number of commands which have been completely transferred from a particular CFIFO when a command transfer from that CFIFO to an external CBuffer is initiated. TC_LCFTSSI is a copy of the corresponding TC_CFx in the *Section , EQADC CFIFO Transfer Counter Registers (EQADC_CFTCR)*, captured at the time a current command transfer to an external CBuffer is initiated. This field has no meaning when LCFTSSI is 0b1111.

Table 491. LCFTCB*n* Description

LCFTCB*n*[0:3]	LCFTCB*n* Meaning
0b0000	Last command was transferred from CFIFO0
0b0001	Last command was transferred from CFIFO1
0b0010	Last command was transferred from CFIFO2
0b0011	Last command was transferred from CFIFO3
0b0100	Last command was transferred from CFIFO4

Table 491. LCFTCB*n* Description (continued)

LCFTCB*n*[0:3]	LCFTCB*n* Meaning
0b0101	Last command was transferred from CFIFO5
0b0110 - 0b1110	Reserved
0b1111	No command was transferred to CBuffer*n*

Table 492. LCFTSSI Description

LCFTSSI[0:3]	LCFTSSI Meaning
0b0000	Last command was transferred from CFIFO0
0b0001	Last command was transferred from CFIFO1
0b0010	Last command was transferred from CFIFO2
0b0011	Last command was transferred from CFIFO3
0b0100	Last command was transferred from CFIFO4
0b0101	Last command was transferred from CFIFO5
0b0110 - 0b1110	Reserved
0b1111	No command was transferred to an external CBuffer

EQADC CFIFO Status Register (EQADC_CFSR)

The EQADC CFIFO Status Register (EQADC_CFSR) contains the current CFIFO status. The EQADC_CFSR registers is read only. Writing to the EQADC_CFSR register has no effect.

Register address: EQADC_BASE+0x0AC

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CFS0		CFS1		CFS2		CFS3		CFS4		CFS5		0	0	0	0
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 465. EQADC CFIFO Status Register (EQADC_CFSR)

Table 493. field description

Field	Description
CFSx[0:1]	CFIFO Status CFSx indicates the current status of CFIFO*x*. Refer to *Table 494* for more information on CFIFO status.

Table 494. Current CFIFO Status

CFIFO Status	Field Value	Explanation
IDLE	0b00	CFIFO is disabled. CFIFO is in single-scan edge or level trigger mode and does not have SSS asserted. EQADC completed the transfer of the last entry of the CQueue in single-scan mode.
Reserved	0b01	Not applicable.
WAITING FOR TRIGGER	0b10	CFIFO Mode is modified to continuous-scan edge or level trigger mode. CFIFO Mode is modified to single-scan edge or level trigger mode and SSS is asserted. CFIFO Mode is modified to single-scan software trigger mode and SSS is negated. CFIFO is paused. EQADC transferred the last entry of the queue in continuous-scan edge trigger mode.
TRIGGERED	0b11	CFIFO is triggered

EQADC SSI Control Register (EQADC_SSICR)

The EQADC SSI Control Register (EQADC_SSICR) configures the SSI sub-block.

Register address: EQADC_BASE+0x0B4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	MDT			0	0	0	0	BR			
W	▨	▨	▨	▨	▨				▨	▨	▨	▨				
RESET:	0	0	0	0	0	1	1	1	0	0	0	0	1	1	1	1

▨ = Unimplemented or Reserved

Figure 466. EQADC SSI Control Register (EQADC_SSICR)

Table 495. EQADC SSI Control Register (EQADC_SSICR) field description

Field	Description
21-23 MDT[0:2]	Minimum Delay after Transmission MDT field defines the minimum delay after transmission time (t_{MDT}) expressed in serial clock (FCK) periods. t_{MDT} is minimum time $\overline{SDS}$ should be kept negated between two consecutive serial transmissions. *Table 496* lists the minimum delay after transfer time according to how MDT is set. The MDT field must only be written when the serial transmissions from the EQADC SSI are disabled - See ESSIE field in *Section , EQADC Module Configuration Register (EQADC_MCR)*.
28-31 BR[0:3]	Baud Rate Field The BR field selects system clock divide factor as shown in *Table 497*. The baud clock is calculated by dividing the system clock by the clock divide factor specified with the BR field. The BR field must only be written when the EQADC SSI is disabled - See ESSIE field in *Section , EQADC Module Configuration Register (EQADC_MCR)*.

Table 496. Minimum Delay After Transmission (t_{MDT}) Time

MDT	t_{MDT} (FCK period)
0b000	1
0b001	2
0b010	3
0b011	4

Table 496. Minimum Delay After Transmission (t_{MDT}) Time

MDT	t_{MDT} (FCK period)
0b100	5
0b101	6
0b110	7
0b111	8

Table 497. System Clock Divide Factor for Baud Clock

BR[0:3]	System Clock Divide Factor (1)
0b0000	2
0b0001	3
0b0010	4
0b0011	5
0b0100	6
0b0101	7
0b0110	8
0b0111	9
0b1000	10
0b1001	11
0b1010	12
0b1011	13
0b1100	14
0b1101	15
0b1110	16
0b1111	17

1. If the system clock is divided by a odd number then the serial clock will have a duty cycle different from 50%.

EQADC SSI Receive Data Register (EQADC_SSIRDR)

The EQADC SSI Receive Data Register (EQADC_SSIRDR) records the last message received from the external device.

Register address: EQADC_BASE+0x0B8

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	RDV	0	0	0	0	0	R_DATA									
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	R_DATA															
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 467. EQADC SSI Receive Data Register (EQADC_SSIRDR)

Table 498. EQADC SSI Receive Data Register (EQADC_SSIRDR) field description

Field	Description
0 RDV	Receive Data Valid Bit The RDV bit indicates if the last received data is valid. This bit is cleared automatically whenever the EQADC_SSIRDR register is read. Writes have no effect. 1 Receive data is valid. 0 Receive data is not valid.
6-31 R_DATA [0:25]	EQADC Receive DATA Field The R_DATA contains the last result message that was shifted in. Writes to the R_DATA have no effect. Messages that were not completely received due to a transmission abort will not be copied into EQADC_SSIRDR.

EQADC STAC Client Configuration Register (EQADC_REDLCCR)

The EQADC STAC Client Configuration Register (EQADC_REDLCCR) contains bits used to control which time slots the EQADC selects to obtain pre-defined external time bases.

Register address: EQADC_BASE+0x0D0

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	REDBS2				SRV2				REDBS1				SRV1			
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 468. EQADC STAC Client Configuration Register (EQADC_REDLCCR)

Table 499. EQADC STAC Client Configuration Register (EQADC_REDLCCR) field description

Field	Description
REDBS*m* [0:3]	STAC Timebase Bits Selection, where *m* (*m*=1,2) The REDBS*m* field selects 16 bits from the total of 24 bits that are received from the STAC interface as described in below. Consider TBASE*m*[0:23] the selected time base from slot SRV*m:*
SRV*m* [0:3]	STAC bus Server Data Slot Selector *m* (*m*=1,2) The field SRV*m* indicates the slot number that contains the desired time base value sent by the STAC server. It is possible to have up to 16 different sources to be selected.

Table 500. STAC Bus Timebase Bits Selection

REDBS*m*[0:3]	Selected Bits
0b0000	TBASE*m*[0:15]
0b0001	TBASE*m*[1:16]
0b0010	TBASE*m*[2:17]
0b0011	TBASE*m*[3:18]
0b0100	TBASE*m*[4:19]
0b0101	TBASE*m*[5:20]
0b0110	TBASE*m*[6:21]
0b0111	TBASE*m*[7:22]
0b1000	TBASE*m*[8:23]
Others	Reserved

Table 501. SRVm valid values

SRV*m*[0:3]	Time Base
0b0000	eTPU engine A, TCR1
0b0001	Reserved
0b0010	eTPU engine A, TCR2
0b0011	Reserved
0b0100–0b1111	Reserved

EQADC CFIFO Registers (EQADC_CFxRw) (x=0, ..,5; w=0, .., 3)

The EQADC CFIFO Registers (EQADC_CFxRw) (x=0, .., 5; w=0, .., 3) provide visibility of the contents of a CFIFO for debugging purposes. Each CFIFO has four registers which are uniquely mapped to its four 32-bit entries. Refer to *Section 23.6.4, EQADC Command FIFOs*, for more information on CFIFOs. These registers are read only. Data written to these registers is ignored.

Register address: EQADC_BASE+0x100
Register address: EQADC_BASE+0x104
Register address: EQADC_BASE+0x108
Register address: EQADC_BASE+0x10C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CFIFO0_DATAw															
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CFIFO0_DATAw															
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 469. EQADC CFIFO0 Registers (EQADC_CF0Rw) (w=0, .., 3)

Register address: EQADC_BASE+0x140
Register address: EQADC_BASE+0x144
Register address: EQADC_BASE+0x148
Register address: EQADC_BASE+0x14C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CFIFO1_DATAw															
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CFIFO1_DATAw															
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 470. EQADC CFIFO1 Registers (EQADC_CF1Rw) (w=0, .., 3)

Register address: EQADC_BASE+0x180
Register address: EQADC_BASE+0x184
Register address: EQADC_BASE+0x188
Register address: EQADC_BASE+0x18C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CFIFO2_DATAw															
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CFIFO2_DATAw															
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 471. EQADC CFIFO2 Registers (EQADC_CF2Rw) (w=0, .., 3)

Register address: EQADC_BASE+0x1C0
Register address: EQADC_BASE+0x1C4
Register address: EQADC_BASE+0x1C8
Register address: EQADC_BASE+0x1CC

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CFIFO3_DATAw															
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CFIFO3_DATAw															
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 472. EQADC CFIFO3 Registers (EQADC_CF3Rw) (w=0, .., 3)

Register address: EQADC_BASE+0x200
Register address: EQADC_BASE+0x204
Register address: EQADC_BASE+0x208
Register address: EQADC_BASE+0x20C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CFIFO4_DATAw															
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CFIFO4_DATAw															
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 473. EQADC CFIFO4 Registers (EQADC_CF4Rw) (w=0, .., 3)

Register address: EQADC_BASE+0x240
Register address: EQADC_BASE+0x244
Register address: EQADC_BASE+0x248
Register address: EQADC_BASE+0x24C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CFIFO5_DATAw															
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CFIFO5_DATAw															
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 474. EQADC CFIFO5 Registers (EQADC_CF5Rw) (w=0, .., 3)

Table 502. EQADC CFIFOx Registers (EQADC_CFxRw) (w=0, .., 3) field description

Field	Description
0-31 CFIFOx_DATAw [0:31]	CFIFOx Data w (w = 0, .., 3) Reading CFIFOx_DATAw returns the value stored on the wth entry of CFIFOx. Each CFIFO is composed of four 32-bit entries, with register 0 being mapped to the one with the smallest memory mapped address.

EQADC CFIFO0 Extension Registers (EQADC_CF0ERw) (w=0, .., 3)

The EQADC CFIFO0 Extension Registers (EQADC_CF0ERw) (w=0, .., 3) provide visibility of the contents of the extended portion of CFIFO0 for debugging purposes. There are four registers which are uniquely mapped to its four 32-bit entries. Refer to *Section 23.6.4, EQADC Command FIFOs*, for more information on CFIFOs. These registers are read only. Data written to these registers is ignored.

Register address: EQADC_BASE+0x110
Register address: EQADC_BASE+0x114
Register address: EQADC_BASE+0x118
Register address: EQADC_BASE+0x11C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CFIFO0_EDATAw															
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CFIFO0_EDATAw															
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 475. EQADC CFIFO0 Extension Registers (EQADC_CF0ERw) (w=0, .., 3)

Table 503. field description

Field	Description
0-31 CFIFOx_EDATAw [0:31]	CFIFOx Data w (w = 0, .., 3) Reading CFIFOx_DATAw returns the value stored on the wth entry of CFIFOx. Each CFIFO is composed of four 32-bit entries, with register 0 being mapped to the one with the smallest memory mapped address.

EQADC RFIFO Registers (EQADC_RFxRw) (x=0, .., 5; w=0, .., 3)

The EQADC RFIFO Registers (EQADC_RFxRw) (x=0, .., 5; w=0, .., 3) provide visibility of the contents of a RFIFO for debugging purposes. Each RFIFO has four registers which are uniquely mapped to its four 16-bit entries. Refer to *Section 23.6.5, EQADC Result FIFOs*, for more information on RFIFOs. These registers are read only. Data written to these registers is ignored.

Register address: EQADC_BASE+0x300
Register address: EQADC_BASE+0x304
Register address: EQADC_BASE+0x308
Register address: EQADC_BASE+0x30C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	RFIFO0_DATAw															
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 476. EQADC RFIFO0 Registers (EQADC_RF0Rw) (w=0, .., 3)

Register address: EQADC_BASE+0x340
Register address: EQADC_BASE+0x344
Register address: EQADC_BASE+0x348
Register address: EQADC_BASE+0x34C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	RFIFO1_DATAw															
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 477. EQADC RFIFO1 Registers (EQADC_RF1Rw) (w=0, .., 3)

Register address: EQADC_BASE+0x380
Register address: EQADC_BASE+0x384
Register address: EQADC_BASE+0x388
Register address: EQADC_BASE+0x38C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	RFIFO2_DATAw															
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 478. EQADC RFIFO2 Registers (EQADC_RF2Rw) (w=0, .., 3)

Register address: EQADC_BASE+0x3C0
Register address: EQADC_BASE+0x3C4
Register address: EQADC_BASE+0x3C8
Register address: EQADC_BASE+0x3CC

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	RFIFO3_DATAw															
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 479. EQADC RFIFO3 Registers (EQADC_RF3Rw) (w=0, .., 3)

Register address: EQADC_BASE+0x400
Register address: EQADC_BASE+0x404
Register address: EQADC_BASE+0x408
Register address: EQADC_BASE+0x40C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	RFIFO4_DATAw															
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 480. EQADC RFIFO4 Registers (EQADC_RF4Rw) (w=0, .., 3)

Register address: EQADC_BASE+0x440
Register address: EQADC_BASE+0x444
Register address: EQADC_BASE+0x448
Register address: EQADC_BASE+0x44C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	RFIFO5_DATAw															
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 481. EQADC RFIFO5 Registers (EQADC_RF5Rw) (w=0, .., 3)

Table 504. EQADC RFIFOx Registers (EQADC_RFxRw) (w=0, .., 3) field description

Field	Description
0-31 RFIFOx_DATAw [0:15]	RFIFOx Data w (w = 0, .., 3) Reading RFIFOx_DATAw returns the value stored on the wth entry of RFIFOx. Each RFIFO is composed of four 16-bit entries, with register 0 being mapped to the one with the smallest memory mapped address.

23.5.3 On-Chip ADC Registers

This section describes a list of registers that control on-chip ADC operation. The ADC registers are not part of the CPU accessible memory map. These registers can only be accessed indirectly through configuration commands. There are 4 non memory mapped registers per ADC, plus 12 registers shared by both ADCs. The address, usage, and access privilege of each register is shown in *Table 505*. Data written to or read from reserved areas of the memory map is undefined.

Their assigned addresses are the values used to set the ADC_REG_ADDRESS field of the read/write configurations commands bound for the on-chip ADCs. These are half-word addresses. Further, the following restrictions apply when accessing these registers:

- Registers ADC0_CR, ADC0_GCCR, ADC0_OCCR, ADC0_AGR1/2 and ADC0_AOR1/2 can only be accessed by configuration commands sent to CBuffer0.
- Registers ADC1_CR, ADC1_GCCR, ADC1_OCCR, ADC1_AGR1/2 and ADC1_AOR1/2 can only be accessed by configuration commands sent to CBuffer1.
- Registers ADC_TSCR, ADC_TBCR, ADC_ACR1-8 and ADC_PUDCR0-7 can be accessed by configuration commands sent to CBuffer0 or to CBuffer1. A data write to any of these registers through a configuration command sent to CBuffer0 will write the same memory location as when writing to it through a configuration command sent to CBuffer1.

Note: Simultaneous write accesses from CBuffer0 and CBuffer1 to any of the shared registers are not allowed.

Table 505. On-Chip ADC Memory Map

ADC Address	Use	Access
0x00	ADC0/ADC1[(1)] Conversion Command for Standard Configuration (See *Section , Conversion Command Format for the Standard Configuration*)	Write
0x01	ADC0/ADC1 Configuration Control Register (ADC0_CR, ADC1_CR)	Write/Read
0x02	Time Stamp Control Register (ADC_TSCR)	Write/Read
0x03	Time Base Counter Register (ADC_TBCR)	Write/Read
0x04	ADC0/ADC1 Gain Calibration Constant Register (ADC0_GCCR, ADC1_GCCR)	Write/Read
0x05	ADC0/ADC1 Offset Calibration Constant Register (ADC0_OCCR, ADC1_OCCR)	Write/Read
0x06- 0x07	Reserved	-
0x08	ADC0/ADC1 Conversion Command for Alternate Configuration 1 (See *Section , Conversion Command Format for Alternate Configurations*)	Write
0x09	ADC0/ADC1 Conversion Command for Alternate Configuration 2 (See *Section , Conversion Command Format for Alternate Configurations*)	Write
0x0A	ADC0/ADC1 Conversion Command for Alternate Configuration 3 (See *Section , Conversion Command Format for Alternate Configurations*)	Write

Table 505. On-Chip ADC Memory Map (continued)

ADC Address	Use	Access
0x0B	ADC0/ADC1 Conversion Command for Alternate Configuration 4 (See *Section , Conversion Command Format for Alternate Configurations*)	Write
0x0C	ADC0/ADC1 Conversion Command for Alternate Configuration 5 (See *Section , Conversion Command Format for Alternate Configurations*)	Write
0x0D	ADC0/ADC1 Conversion Command for Alternate Configuration 6 (See *Section , Conversion Command Format for Alternate Configurations*)	Write
0x0E	ADC0/ADC1 Conversion Command for Alternate Configuration 7 (See *Section , Conversion Command Format for Alternate Configurations*)	Write
0x0F	ADC0/ADC1 Conversion Command for Alternate Configuration 8 (See *Section , Conversion Command Format for Alternate Configurations*)	Write
0x10-0x2F	Reserved	-
0x30	Alternate Configuration 1 Control Register (ADC_ACR1)	Write/Read
0x31	ADC0/ADC1 Alternate Gain 1 Register (ADC0_AGR1, ADC1_AGR1)	Write/Read
0x32	ADC0/ADC1 Alternate Offset 1 Register (ADC0_AOR1, ADC1_AOR1)	Write/Read
0x33	Reserved	-
0x34	Alternate Configuration 2 Control Register (ADC_ACR2)	Write/Read
0x35	ADC0/ADC1 Alternate Gain 2 Register (ADC0_AGR2, ADC1_AGR2)	Write/Read
0x36	ADC0/ADC1 Alternate Offset 2 Register (ADC0_AOR2, ADC1_AOR2)	Write/Read
0x37	Reserved	-
0x38	Alternate Configuration 3 Control Register (ADC_ACR3)	Write/Read
0x39	Reserved	-
0x3A	Reserved	-
0x3B	Reserved	-
0x3C	Alternate Configuration 4 Control Register (ADC_ACR4)	Write/Read
0x3D	Reserved	-
0x3E	Reserved	-
0x3F	Reserved	-
0x40	Alternate Configuration 5 Control Register (ADC_ACR5)	Write/Read
0x41	Reserved	-
0x42	Reserved	-
0x43	Reserved	-
0x44	Alternate Configuration 6 Control Register (ADC_ACR6)	Write/Read
0x45	Reserved	-
0x46	Reserved	-
0x47	Reserved	-
0x48	Alternate Configuration 7 Control Register (ADC_ACR7)	Write/Read
0x49	Reserved	-
0x4A	Reserved	-
0x4B	Reserved	-
0x4C	Alternate Configuration 8 Control Register (ADC_ACR8)	Write/Read
0x4D-0x6F	Reserved	-
0x70	Pull Up/Down Control Register0 (ADC_PUDCR0)	Write/Read

Table 505. On-Chip ADC Memory Map (continued)

ADC Address	Use	Access
0x71	Pull Up/Down Control Register0 (ADC_PUDCR1)	Write/Read
0x72	Pull Up/Down Control Register0 (ADC_PUDCR2)	Write/Read
0x73	Pull Up/Down Control Register0 (ADC_PUDCR3)	Write/Read
0x74	Pull Up/Down Control Register0 (ADC_PUDCR4)	Write/Read
0x75	Pull Up/Down Control Register0 (ADC_PUDCR5)	Write/Read
0x76	Pull Up/Down Control Register0 (ADC_PUDCR6)	Write/Read
0x77	Pull Up/Down Control Register0 (ADC_PUDCR7)	Write/Read
0x78-0x97	Reserved for ADC_PUDCR8 to ADC_PUDCR39	-
0x98-0xFF	Reserved	-

1. Throughout the table, ADC0/ADC1 indicates that if the command is stored in CBuffer0 it will be applied to ADC0 and if in CBuffer1 it applies to ADC1. If this indication is omitted the register applies for both ADC0 and ADC1, independent of the CBuffer used.

ADC0/1 Control Registers (ADC0_CR and ADC1_CR)

The ADC0/1 Control Registers (ADC0/1_CR) is used to define the standard configuration of the ADC. In the standard configuration, the parameters contained in the Alternate Configuration Control Registers (ADC_ACR1-8) are fixed at their reset value. A conversion uses the standard configuration when the conversion command (with the standard format) is written to address 0x00 of the on-chip ADC memory map. Refer to *Section , Conversion Command Format for the Standard Configuration*.

Figure 482. ADC0/1 Control Registers (ADC0/1_CR)

ADC0 Register address: 0x01

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	ADC0_EN	0	0	0	ADC0_EMUX	0	ADC0_TBSEL		ADC0_ODD_PS	ADC0_CLK_DTY	ADC0_CLK_SEL	ADC0_CLK_PS				
W		▨	▨	▨		▨										
RESET:	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1

ADC1 Register address: 0x01

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	ADC1_EN	0	0	0	ADC1_EMUX	0	ADC1_TBSEL		ADC1_ODD_PS	ADC1_CLK_DTY	ADC1_CLK_SEL	ADC1_CLK_PS				
W		▨	▨	▨		▨										
RESET:	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1

▨ = Unimplemented or Reserved

Table 506. ADC0/1 Control Registers (ADC0/1_CR) field description

Field	Description
0 ADC0/1_EN	Enable bit for ADC0/1 ADC0/1_EN enables ADC0/1 to perform A/D conversions. Refer to *Section , Enabling and Disabling the On-chip ADCs*, for details. 1 ADC is enabled and ready to perform A/D conversions. 0 ADC is disabled. Clock supply to ADC0/1 is stopped. Conversion commands sent to the CBuffer of a disabled ADC are ignored by the ADC control hardware. When the ADC0/1_EN status is changed from asserted to negated, the ADC Clock will not stop until it reaches its low phase.
4 ADC0/1_EMUX	External Multiplexer enable for ADC0/1 When ADC0/1_EMUX is asserted, the MA pins will output digital values according to the number of the external channel being converted for selecting external multiplexer inputs. Refer to *Section 23.6.7, Internal/External Multiplexing*, for a detailed description about how ADC0/1_EMUX affects channel number decoding. 1 External multiplexer enabled; external multiplexer channels can be selected. 0 External multiplexer disabled; no external multiplexer channels can be selected. Both ADC0 and ADC1 of an eQADC module pair must be enabled before calibrating or using either ADC0 or ADC1 of the pair. Failure to enable both ADC0 and ADC1 of the pair can result in inaccurate conversions. Both ADC0/1_EMUX bits must not be asserted at the same time. The ADC0/1_EMUX bit must only be written when the ADC0/1_EN bit is negated. ADC0/1_EMUX can be set during the same write cycle used to set ADC0/1_EN.
6-7 ADC0/1_TBSEL [0:1]	Timebase Selection for ADC0/1 The ADC0/1_TBSEL[0:1] field selects the time information to be used as timestamp according to *Table 507*. This selection is overriden by the corresponding field ATBSEL in the ADC_ACR1-8 registers when the alternate conversion command is used.
8 ADC0/1_ODD_PS	Clock Prescaler Odd Rates Selector for ADC0/1 The ADC0/1_CLK_DTY field controls the duty rate of the ADC0/1 clock when the ADC0/1_CLK_PS field is asserted. The generated clock has an odd number of system clock cycles, therefore this field is used to select a clock duty higher or lower than 50%. 1 Odd divide factor is selected. The final divide factor is dependent of ADC0/1_CLK_PS field. 0 Even divide factor is selected. The final divide factor is dependent of ADC0/1_CLK_PS field.
9 ADC0/1_CLK_DTY	Clock Duty Rate Selector for ADC0/1 (for odd divide factors) The ADC0/1_ODD_PS field is used together with the ADC0/1_CLK_PS field to define even/odd divide factors in the generation of the ADC0/1 clocks. Refer to *Table 508* for available divide factors. 1 clock high pulse is longer 1 clock cycle than low portion. 0 clock low interval is longer 1 clock cycle than high pulse.

Table 506. ADC0/1 Control Registers (ADC0/1_CR) field description (continued)

Field	Description
10 ADC0/1_CLK_SEL	Clock Selector for ADC0/1 The ADC0/1_CLK_SEL is used to select between the system clock signal or the prescaler output signal. The prescaler provides the system clock signal divided by a even factor from 2 to 64. This is required to permit the ADC to run as fast as possible when the device is in Low Power Active mode and system clock is around 1 MHz. 1 System clock is selected - maximum frequency. 0 Prescaler output clock is selected. The ADC0/1_CLK_SEL bits must only be written when the ADC0/1_EN bit is negated. ADC0/1_CLK_SEL can be set during the same write cycle used to set ADC0/1_EN.
11-15 ADC0/1_CLK_PS [0:4]	Clock Prescaler Field for ADC0/1 The ADC0/1_CLK_PS field controls the system clock divide factor for the ADC0/1 clock as in *Table 508*. See *Section , ADC Clock and Conversion Speed,* for details about how to set ADC0/1_CLK_PS. The ADC0/1_CLK_PS field must only be written when the ADC0/1_EN bit is negated. This field can be configured during the same write cycle used to set ADC0/1_EN.

Table 507. Timebase Selection

ADC0/1_TBSEL[0:1]	Definition
00	Selects internally generated time base as time stamp.
01	Selects imported time base 1 indicated by SRV1 bit field of EQADC_REDLCCR register.
10	Selects imported time base 2 indicated by SRV2 bit field of EQADC_REDLCCR register.
11	Reserved

Table 508. System Clock Divide Factor for ADC Clock

ADC0/1_CLK_PS[0:4]	System Clock Divide Factor	
	ADC0/1_ODD_PS = 0	ADC0/1_ODD_PS = 1
0b00000	2	3
0b00001	4	5
0b00010	6	7
0b00011	8	9
0b00100	10	11
0b00101	12	13
0b00110	14	15
0b00111	16	17
0b01000	18	19

Table 508. System Clock Divide Factor for ADC Clock (continued)

ADC0/1_CLK_PS[0:4]	System Clock Divide Factor	
	ADC0/1_ODD_PS = 0	ADC0/1_ODD_PS = 1
0b01001	20	21
0b01010	22	23
0b01011	24	25
0b01100	26	27
0b01101	28	29
0b01110	30	31
0b01111	32	33
0b10000	34	35
0b10001	36	37
0b10010	38	39
0b10011	40	41
0b10100	42	43
0b10101	44	45
0b10110	46	47
0b10111	48	49
0b11000	50	51
0b11001	52	53
0b11010	54	55
0b11011	56	57
0b11100	58	59
0b11101	60	61
0b11110	62	63
0b11111	64	65

ADC Time Stamp Control Register (ADC_TSCR)

The ADC Time Stamp Control Register (ADC_TSCR) contains a system clock divide factor used in the making of the time base counter clock. It determines at what frequency the time base counter will run. ADC_TSCR can be accessed by configuration commands sent to CBuffer0 or to CBuffer1. A data write to ADC_TSCR through a configuration command sent to CBuffer0 will write the same memory location as when writing to it through a configuration command sent to CBuffer1.

Note: Simultaneous write accesses from CBuffer0 and CBuffer1 to ADC_TSCR are not allowed.

Figure 483. ADC Time Stamp Control Register (ADC_TSCR)

ADC0/1 Register address: 0x02

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	TBC_CLK_PS			
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨				
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Table 509. ADC Time Stamp Control Register (ADC_TSCR) field description

Field	Description
12-15 TBC_CLK_PS [0:3]	Time Base Counter Clock Prescaler The TBC_CLK_PS field contains the system clock divide factor for the time base counter. It controls the accuracy of the time stamp. The prescaler is disabled when TBC_CLK_PS is set to 0b0000.

Table 510. Clock Divide Factor for Time Stamp

TBC_CLK_PS[0:3]	System Clock Divide Factor	Clock to Time Stamp Counter for a 120 MHz System Clock (MHz)
0b0000	Disabled	Disabled
0b0001	1	120
0b0010	2	60
0b0011	4	30
0b0100	6	20
0b0101	8	15
0b0110	10	12
0b0111	12	10
0b1000	16	7.5
0b1001	32	3.75
0b1010	64	1.88
0b1011	128	0.94
0b1100	256	0.47
0b1101	512	0.23
0b1110 - 0b1111	Reserved	-

Note: If TBC_CLK_PS is not set to disabled, it must not be changed to any other value besides disabled. If TBC_CLK_PS is set to disabled it can be changed to any other value.

ADC Time Base Counter Registers (ADC_TBCR)

The ADC Time Base Counter Register (ADC_TBCR) contains the current value of the time base counter. ADC_TBCR can be accessed by configuration commands sent to CBuffer0 or to CBuffer1. A data write to ADC_TBCR through a configuration command sent to CBuffer0 will write the same memory location as when writing to it through a configuration command sent to CBuffer1.

Note: Simultaneous write accesses from CBuffer0 and CBuffer1 to ADC_TBCR are not allowed.

Figure 484. ADC Time Base Counter Register (ADC_TBCR)

ADC0/1 Register address: 0x03

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	TBC_VALUE															
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Table 511. ADC Time Base Counter Register (ADC_TBCR) field description

Field	Description
0-15 TBC_VALUE [0:15]	Time Base Counter VALUE Field The TBC_VALUE field contains the current value of the time base counter. Reading TBC_VALUE returns the current value of time base counter. Writes to TBC_VALUE register load the written data to the counter. The time base counter counts from 0x0000 to 0xFFFF and wraps when reaching 0xFFFF.

ADC0/1 Gain Calibration Constant Registers (ADC0_GCCR and ADC1_GCCR)

The ADC0/1 Gain Calibration Constant Register (ADC0/1_GCCR) contains the gain calibration constant used to fine-tune the ADC0/1 conversion results. Refer to *Section , ADC Calibration Feature*, for details about the calibration scheme used in the EQADC.

Figure 485. ADC0/1 Gain Calibration Constant Registers (ADC0/1_GCCR)

ADC0 Register address: 0x04

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	GCC0														
W	▨															
RESET:	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

ADC1 Register address: 0x04

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	GCC1														
W	▨															
RESET:	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Table 512. ADC0/1 Gain Calibration Constant Registers (ADC0/1_GCCR) field description

Field	Description
1-15 GCC0/1 [0:14]	Gain calibration constant for ADC0/1 GCC0/1 contains the gain calibration constant used to fine-tune ADC0/1 conversion results. It is a unsigned 15-bit fixed pointed value. The gain calibration constant is an unsigned fixed point number expressed in the *GCC_INT.GCC_FRAC* binary format. The integer part of the gain constant (GCC_INT) contains a single binary digit while its fractional part (GCC_FRAC) contains fourteen digits. For details about the GCC data format refer to *Section , MAC Unit and Operand Data Format*.

ADC0/1 Offset Calibration Constant Registers (ADC0_OCCR and ADC1_OCCR)

The ADC0/1 Offset Calibration Constant Register (ADC0/1_OCCR) contains the offset calibration constant used to fine-tune of ADC0/1 conversion results. The offset constant is a signed 14-bit integer value. Refer to *Section , ADC Calibration Feature*, for details about the calibration scheme used in the EQADC.

Figure 486. ADC0/1 Offset Calibration Constant Registers (ADC0/1_OCCR)

ADC0 Register address: 0x05

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	OCC0													
W	▨	▨														
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

ADC1 Register address: 0x05

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	OCC1													
W	▨	▨														
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Table 513. ADC0/1 Offset Calibration Constant Registers (ADC0/1_OCCR) field description

Field	Description
2-15 OCC0/1 [0:13]	Offset Calibration Constant of ADC0/1 OCC0/1 contains the offset calibration constant used to fine-tune ADC0/1 conversion results. Negative values should be expressed using the two’s complement representation.

Alternate Configuration 1-8 Control Registers (ADC_ACR1-8)

The Alternate Configuration Control Registers (ADC_ACR1-8) are used to configure the alternate configurations of the ADC. There are 8 possible alternate configurations, each one associated with one of the ADC_ACR1-8 registers. All alternate configurations share the same standard configuration parameters from the ADC0/1_CR registers, plus additional configuration parameters contained in the ADC_ACR1-8. A conversion uses one of the alternate configurations when the conversion command (with the alternate configuration format) is written to an address in the range 0x08-0x0F of the on-chip ADC memory map. Refer to *Section , Conversion Command Format for Alternate Configurations*.

Figure 487. Alternate Configuration 1-8 Control Registers (ADC_ACR1-8)

ADC0/1 Register address: 0x30
ADC0/1 Register address: 0x34
ADC0/1 Register address: 0x38
ADC0/1 Register address: 0x3C
ADC0/1 Register address: 0x40
ADC0/1 Register address: 0x44
ADC0/1 Register address: 0x48
ADC0/1 Register address: 0x4C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	RET_INH	0	DEST				FMTA	0	RESSEL		0	0	ATBSEL		PRE_GAIN	
W		▨						▨			▨	▨				
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Table 514. Alternate Configuration 1-8 Control Registers (ADC_ACR1-8) field description

Field	Description
0 RET_INH	Result Transfer Inhibit / Decimation Filter Pre-Fill This bit is used to inhibit the transfer of the result data from the peripheral module to the result queue. When the module is a Decimation Filter, this bit sets the filter in a special mode (PRE-FILL) in which it does not generate decimated samples out from the conversion results received from the EQADC block, but the conversion samples are used by the filter algorithm. This feature allows a proper initialization of the Decimation Filter without generating any decimated result. Or this bit is useful for sending the result of the ADC to the STAC bus master but not putting the result in the result queue. 1 No result transfer to result queue / Decimation Filter PRE-FILL mode 0 Result transfer to result queue / Decimation Filter in filtering mode
2-5 DEST [0:3]	Conversion Result Destination Selection The DEST[0:3] field selects the destination of the conversion result generated by the Alternate Conversion Command as shown in *Table 515*. This field also affects the behavior of the FMTA bit and the FFMT bit of the conversion command for alternate configurations (see *Section , Conversion Command Format for Alternate Configurations*).
6 FMTA	Conversion Data Format for Alternate Configuration If the DEST field is not 0b000, the FMTA bit specifies how the 12-bit conversion data returned by the ADCs is formatted into the 16-bit data which is sent to the parallel side interface. 1 Right justified signed 0 Right justified unsigned
8-9 RESSEL [0:1]	ADC Resolution Selection The RESSEL[0:1] field selects the resolution of the ADC according to *Table 516*.

Table 514. Alternate Configuration 1-8 Control Registers (ADC_ACR1-8) field description

Field	Description
12-13 ATBSEL [0:1]	Alternate Command Timebase Selector The ATBSEL[0:1] field selects the time information to be used as timestamp according to *Table 517*. This selection overrides the corresponding fields ADC0/1_TBSEL in the ADC0/1_CR registers when the alternate conversion command is used.
14-15 PRE_GAIN [0:1]	ADC Pre-gain control The PRE_GAIN[0:1] controls the gain of the ADC input stage by changing the internal ADC iterations in the gain stage. The gain is selected according to *Table 518*.

Table 515. Conversion Destination Selection

DEST[0:3]	Description
0000	The conversion result is sent to the RFIFOs. The data format is specified by the FFMT bit in the conversion command.
0001	The conversion result is sent to the Parallel Side Interface of Decimation filter A. The data format is specified by the FMTA bit in the Alternate Configuration Control Register.
0010	The conversion result is sent to the Parallel Side Interface of Decimation filter B. The data format is specified by the FMTA bit in the Alternate Configuration Control Register.
0011 - 1110	Unused.
1111	The conversion result is sent to the Parallel Side Interface of Reaction module. The data format is specified by the FMTA bit in the Alternate Configuration Control Register.

Table 516. Resolution Selection

RESSEL[0:1]	Definition
00	ADC set to 12-bits resolution
01	ADC set to 10-bits resolution
10	ADC set to 8-bits resolution
11	Reserved

Table 517. Timebase Selection

ATBSEL[0:1]	Definition
00	Selects internally generated time base as time stamp.
01	Selects imported time base 1 indicated by SRV1 bit field of EQADC_REDLCCR register.
10	Selects imported time base 2 indicated by SRV2 bit field of EQADC_REDLCCR register.
11	Reserved

Table 518. ADC Pre-Gain Control Bits

Pre_gain[0:1]	Description
00	X1 gain
01	X2 gain
10	X4 gain
11	Reserved

ADC0/1 Alternate Gain Registers (ADC0_AGR1-2 and ADC1_AGR1-2)

The Alternate Gain Registers (ADC0_AGRx and ADC1_AGRx, x=1-2) contain the gain calibration constants used to fine-tune the ADCs conversion results for alternate configurations 1 or 2. A conversion from an ADC uses the corresponding ADC0_AGRx or ADC1_AGRx register when the conversion command (with the alternate configuration format) is written to an address in the range 0x08-0x09 of the on-chip ADC memory map. Refer to *Section , ADC Calibration Feature*, for details about the calibration scheme used in the EQADC.

Figure 488. ADC0/1 Alternate x Gain Register (ADC0/1_AGRx, x=1-2)

ADC0 Register address: 0x31
ADC0 Register address: 0x35

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	ALTGCC0x														
W	▨															
RESET:	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

ADC1 Register address: 0x31
ADC1 Register address: 0x35

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	ALTGCC1x														
W	▨															
RESET:	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 489. ADC0/1 Alternate x Gain Register (ADC0/1_AGRx, x=1-2) field description

Field	Description
1-15 ALTGCC0/1x [0:14]	Alternate Gain Calibration Constant ALTGCC0/1x[0:14] contain the gain calibration constants used to fine-tune ADC0/1 conversion results for alternate configurations 1 and 2. The gain calibration constants are 15-bit unsigned fixed point numbers expressed in the *GCC_INT.GCC_FRAC* binary format. The integer part of the gain constants (GCC_INT) contain a single binary digit while their fractional part (GCC_FRAC) contain fourteen digits. For details about the GCC data format refer to *Section , MAC Unit and Operand Data Format*.

ADC0/1 Alternate Offset Register (ADC0_AOR1-2 and ADC1_AOR1-2)

The Alternate Offset Registers (ADC0_AORx and ADC1_AORx, x=1-2) contain the offset calibration constants used to fine-tune ADCs conversion results for alternate configurations 1 and 2. The offset constants are signed 14-bit integer values. Refer to *Section , ADC Calibration Feature*, for details about the calibration scheme used in the EQADC.

Figure 490. ADC0/1 Alternate x Offset Registers (ADC0/1_AORx, x=1-2)

ADC0 Register address: 0x32
ADC0 Register address: 0x36

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	ALTOCC0x													
W	▨	▨														
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

ADC1 Register address: 0x32
ADC1 Register address: 0x36

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	ALTOCC1x													
W	▨	▨														
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Table 519. ADC0/1 Alternate x Offset Registers (ADC0/1_AORx, x=1-2) field description

Field	Description
2-15 ALTOCC0/1x [0:13]	Alternate Offset Calibration Constant ALTOCC0/1x[0:13] contain the offset calibration constants used to fine-tune ADCs conversion results for alternate configurations 1 or 2. Negative values should be expressed using the two’s complement representation.

ADC Pull Up/Down Control Register *x* (ADC_PUDCRx, *x*=0-7)

The ADC Pull Up/Down Control Register *x* (ADC_PUDCR*x*) contains configuration bits for pull up and pull down resistors present at ADC input channels *x*, *x*=0 to 7.

Figure 491. ADC Pull Up/Down Control Register *x* (ADC_PUDCR*x*, *x*=0-7)

ADC0/1 Register address: 0x70-0x77

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	CH_PULL*x*		0	0	PULL_STR *x*		0	0	0	0	0	0	0	0
W	▨	▨			▨	▨			▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Table 520. ADC Pull Up/Down Control Register *x* (ADC_PUDCR*x*, *x*=0-7) field description

Field	Description
2-3 CH_PULL*x* [0:1]	Channel *x* Pull Up/Down Control bits The CH_PULL*x*[0:1] field controls the pull up/down configuration of the channel *x* according to *Table 521*.
6-7 PULL_STR*x* [0:1]	Pull Up/Down Strength Control bits of channel *x* The PULL_STR*x*[0:1] bit field defines the strength of the channel *x* pull up or down resistors, according to *Table 522*.

Table 521. Channel *x* Pull Up/Down Field Definition

CH_PULL*x*[0:1]	Definition
00	No Pull resistors connected to the channel
01	Pull Up resistor connected to the channel
10	Pull Down resistor connected to the channel
11	Pull Up and Pull Down resistors connected to the channel

Table 522. Pull Up/Down Strength Field Definition

PULL_STR*x*[0:1]	Definition
00	Reserved
01	200 Kohms pull resistor
10	100 Kohms pull resistor
11	5 Kohms (Approx.) pull resistor(1)

1. This set is not available for CH_PULL_*x* = 11.

23.6 Functional Description

23.6.1 Overview

The EQADC provides a parallel interface to two on-chip ADCs, a single master to single slave serial interface to an off-chip external device and a parallel side interface to an on-chip companion module, like a decimation filter. The two on-chip ADCs are architected to allow access to all the analog channels.

Initially, command data is contained in system memory in a user defined data structure which is likely to be a queue as depicted in *Figure 445*(as). Command data is moved between the CQueues and CFIFOs by the host CPU or by the DMAC which respond to interrupt and DMA requests generated by the EQADC. The EQADC supports software and hardware triggers from other blocks or external pins to initiate transfers of commands from the multiple CFIFOs to the on-chip ADCs or to the external device.

---

as. Command and result data can be stored in the system memory in any user defined data structure. However, in this document it will be assumed that the data structure of choice is a queue, since it is the most likely data structure to be used and because queues are the only type of data structure supported by the DMAC.

CFIFOs can be configured in single-scan or continuous-scan mode. When a CFIFO is configured in single-scan mode, the EQADC scans the CQueue one time. The EQADC stops transferring commands from the triggered CFIFO after detecting the EOQ bit set in the last transfer. After an EOQ bit is detected, software involvement is required to rearm the CFIFO so that it can detect new trigger events.

When a CFIFO is configured for continuous-scan mode, the whole CQueue is scanned multiple times. After the detection of an asserted EOQ bit in the last command transfer, command transfers can continue or not depending on the mode of operation of the CFIFO.

CFIFO0 has a special configuration option to allow a repetitive sequence of conversion commands (streaming mode) with high priority characteristics (abort operation) or not. This feature is useful with the immediate conversion command feature that allows the immediate execution of a conversion command or a sequence of commands with critical timing even with the possibility of abortion of some current ADC conversion in progress. The aborted command is stored and executed again as soon as the critical timing commands have been finished.

The multiple Result FIFOs (RFIFOs) can receive data from the on-chip ADCs, from an off-chip external device or from an on-chip companion module. Data from the on-chip ADCs can be routed to the side interface, processed by the on-chip companion module and then routed back through the side interface to the RFIFOs.

23.6.2 Data Flow in EQADC

Overview and Basic Terminology

Figure 492 shows how command data flows inside the EQADC system. A Command Message is the predefined format at which command data is stored on the CQueues. A Command message has 32 bits and is composed of two parts: a CFIFO header and an ADC Command. Command messages are moved from the CQueues to the CFIFOs by the host CPU or by the DMAC as they respond to interrupt and DMA requests generated by the EQADC. The EQADC generates these requests whenever a CFIFO is not full. The *FIFO Control Unit* will only transfer to a CBuffer the ADC command part of the Command Message. Information in the CFIFO header together with the upper bit of the ADC command is used by the *FIFO Control Unit* to arbitrate which triggered CFIFO will be transferring the next command. Since command transfer through the serial interface can take significantly more time than a parallel transfer to the on-chip ADCs, command transfers for on-chip ADCs occur concurrently with the ones through the serial interface.

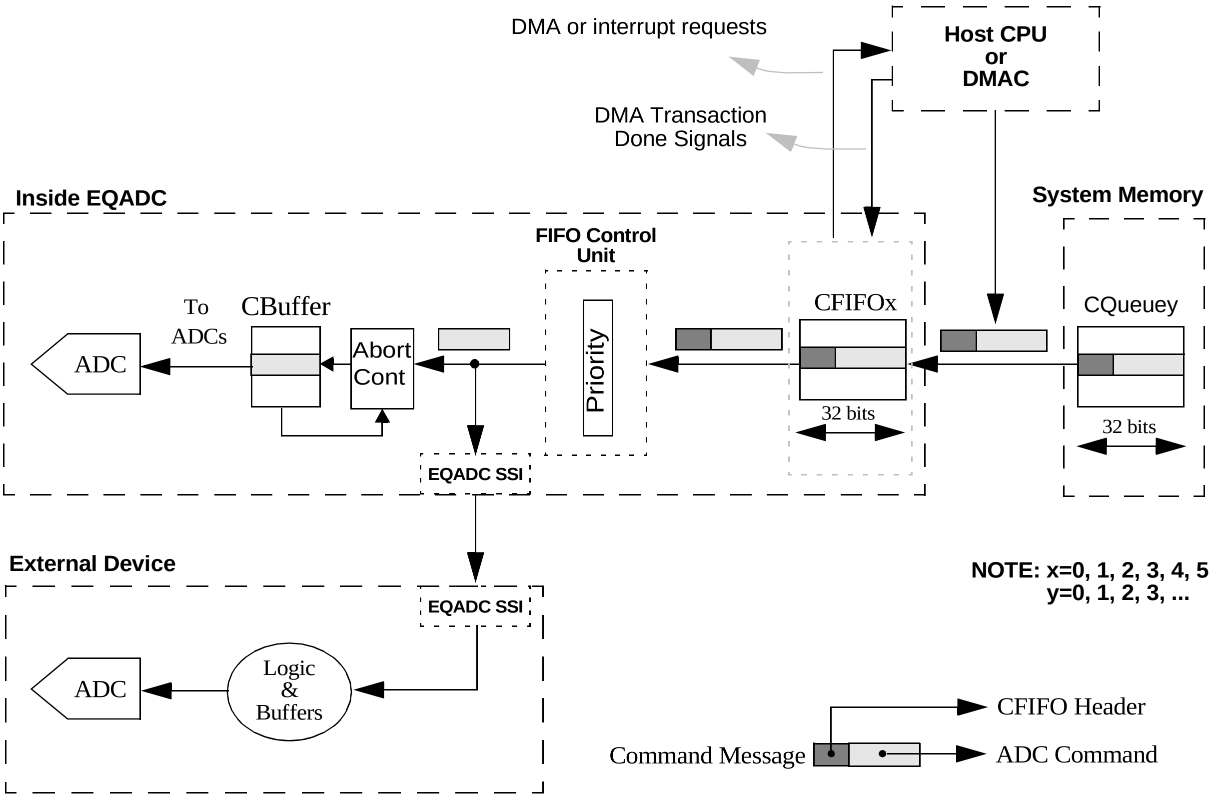


Figure 492. Command Flow during EQADC operation

ADC commands sent to the on-chip CBuffers are executed in a first-in-first-out basis with exception when the immediate conversion command function is enabled. Three types of results can be expected: data read from an ADC register, a conversion result, or a time stamp. The order at which ADC commands sent to the external device are executed, and the type of results that can be expected depends on the architecture of that device with the exception of unsolicited data like null messages for example.

Note: While the EQADC pops commands out from a CFIFO, it also is checking the number of entries in the CFIFO and generating requests to fill it. The process of pushing and popping commands to and from a CFIFO can occur simultaneously. However, this is not true for CFIFO0 when configured to operate in streaming mode for popping.

The *FIFO Control Unit* expects all incoming results to be shaped in a predefined Result Message format. *Figure 493* shows how result data flows inside the EQADC system. Results generated on the on-chip ADCs are adjusted considering the selected resolution of the ADC and are formatted into result messages inside the *Result Format and Calibration Sub-Block*. This result message can be routed directly to one of the RFIFOs or to an on-chip companion module via the parallel side interface. After the data is processed by the companion module, it can be routed back to one of the RFIFOs via the side interface with the correct format. Results returning from the external device are already formatted into result messages and therefore bypass the *Result Format and Calibration Sub-Block*. A result message is composed of an RFIFO header and an ADC Result. The *FIFO Control Unit* decodes the information contained in the RFIFO header to determine the RFIFO to which the ADC result should be sent. Once in an RFIFO, the ADC result is moved to the corresponding RQueue by the host CPU or by the DMAC as they respond to interrupt and DMA requests generated by the EQADC. The EQADC generates these requests whenever an RFIFO has at least one entry.

Note: While conversion results are returned, the EQADC is checking the number of entries in the RFIFO and generating requests to empty it. The process of pushing and popping ADC results to and from an RFIFO can occur simultaneously.

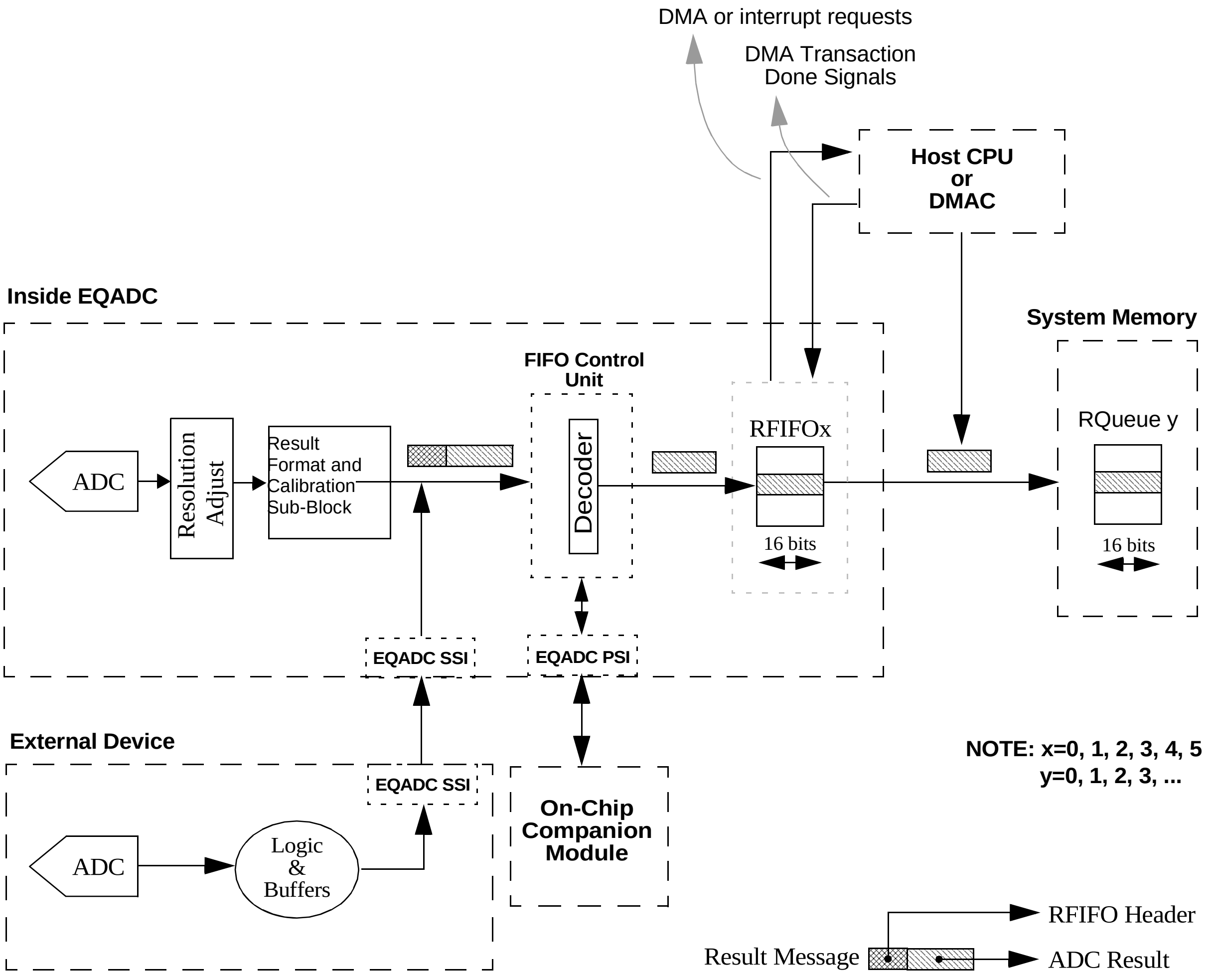


Figure 493. Result Flow during EQADC operation

Assumptions/Requirements Regarding the External Device

The external device exchanges command and result data with the EQADC through the EQADC SSI interface. This section explains the minimum requirements an external device has to meet to properly interface with the EQADC. Some assumptions about the architecture of the external device are also described.

EQADC SSI Protocol Support

The external device must fully support the EQADC SSI protocol as specified in *Section 23.6.9, EQADC Synchronous Serial Interface (SSI) Sub-Block*.. Support for the abort feature is optional. When aborts are not supported, all command messages bound for an external CBuffer must have the ABORT_ST bit negated - see *Section , Command Message Format for External Device Operation*.

Number of Command Buffers and Result Buffers

The external device should have a minimum of one and a maximum of two Command Buffers (CBuffer) to store command data sent from the EQADC. Even if more than two CBuffers are implemented in the external device, they are not recognized by the EQADC as valid destinations for commands. In this document, these two CBuffers will be referred as CBuffer2 and CBuffer3. The external device decides to which external CBuffer a command should go by decoding the upper bit (BN bit) of the ADC command - see *Section , Command Message Format for External Device Operation*. An external device that only implements one CBuffer can ignore the BN bit.

The limit of two CBuffers does not limit the number of RBuffers in the slave device.

Command Execution and Result Return

Commands sent to an specific CBuffer should be executed in that order they were received.

Results generated by the execution of commands of a CBuffer should be returned in the order the CBuffer received these commands.

Null and Result Messages

The external device must be capable of correctly processing null messages as specified in the *Section , EQADC null message send format register (EQADC_NMSFR)*.

In case no valid result data is available to be sent to the EQADC, the external device must send data in the format specified in *Section , Null Message Format for External Device Operation*.

In case valid result data is available to sent to the EQADC, the external device must send data in the format specified in *Section , Result Message Format for External Device Operation*.

The BUSY0/1 fields of all messages sent from the external device to the EQADC must be correctly encoded according to the latest information on the fullness state of the CBuffers. For example, if the CBuffer2 is empty before the end of the current serial transmission and if at the end of this transmission the external device receives a command to CBuffer2, then the BUSY0 field, that is to be sent to the EQADC on the next serial transmission, should be encoded assuming that CBuffer2 has one entry.

Message Format in EQADC

This section explains the command and result message formats used for on-chip ADC operation and for external device operation.

A Command Message is the predefined format at which command data is stored on the CQueues. A Command message has 32 bits and is composed of two parts: a CFIFO header and an ADC Command. The size of the CFIFO header is fixed to 6 bits, and it works as inputs to the *FIFO Control Unit*. It controls when a CQueue ends, when it pauses, if commands are sent to internal or external buffers, and if it can abort a serial data transmission. Information contained in the CFIFO header, together with the upper bit of the ADC Command is used by the *FIFO Control Unit* to arbitrate which triggered CFIFO will transfer the next command. ADC commands are encoded inside the least significant 26 bits of the command message.

A Result message is composed of an RFIFO header and an ADC Result. The *FIFO Control Unit* decodes the information contained in the RFIFO header to determine the RFIFO to which the ADC result should be sent. An ADC result is always 16 bits long.

Message Formats for On-Chip ADC Operation

This section describes the Command/Result message formats used for on-chip ADC operation.

Note: Although this subsection describes how the command and result messages are formatted to communicate with the on-chip ADCs, nothing prevents the programmer from using a different format when communicating with an external device through the serial interface. Refer to Section , Message Formats for External Device Operation. Apart from the BN bit, the ADC Command of a command message can be formatted to communicate to an arbitrary external device provided that the device returns an RFIFO header in the format expected by the EQADC. When the FIFO Control Unit receives return data message, it decodes the message tag field and stores the 16-bit data into the corresponding RFIFO.

Conversion Command Format for the Standard Configuration

Figure 494 describes the format for conversion commands when interfacing with the on-chip ADCs in the standard configuration. The standard configuration is selected when the lowest byte (bits 24-31) of the conversion command is set to zero. In the standard configuration, the conversion result is always routed to one of the RFIFOs. A time stamp information can be optionally requested.

Figure 494. Conversion Command Format for the Standard Configuration

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
EOQ	PAUSE	REP	RESERVED		EB (0b0)	BN	CAL	MESSAGE_TAG				LST		TSR	FMT
CFIFO Header						ADC Command									

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
CHANNEL_NUMBER								0	0	0	0	0	0	0	0
ADC Command															

Table 523. Conversion Command Format for the Standard Configuration field description

Field	Description
0 EOQ	End Of Queue Bit The EOQ bit is asserted in the last command of a CQueue to indicate to the EQADC that a scan of the CQueue is completed. EOQ instructs the EQADC to reset its current CFIFO transfer counter value (TC_CF) to zero. Depending on the CFIFO mode of operation, the CFIFO status will also change upon the detection of an asserted EOQ bit on the last transferred command - see *Section , CFIFO Scan Trigger Modes*, for details. 1 Last entry of the CQueue. 0 Not the last entry of the CQueue. If both the PAUSE and EOQ bits are asserted in the same command message the respective flags are set, but the CFIFO status changes as if only the EOQ bit were asserted.
1 PAUSE	Pause Bit The Pause bit allows software to create sub-queues within a CQueue. When the EQADC completes the transfer of a command with an asserted Pause bit, the CFIFO enters the WAITING FOR TRIGGER state. Refer to *Section , CFIFO Operation Status*, for a description of the state transitions. The Pause bit is only valid when CFIFO operation mode is configured to single or continuous-scan edge trigger mode. 1 Enter WAITING FOR TRIGGER state after transfer of the current Command Message. 0 Do not enter WAITING FOR TRIGGER state after transfer of the current Command Message. If both the PAUSE and EOQ bits are asserted in the same command message the respective flags are set, but the CFIFO status changes as if only the EOQ bit were asserted.
2 REP	Repeat/loop Start Point Indication Bit The REP bit is asserted in the command to indicate where is the start point of the sub-queue to be repeated when the streaming mode is enabled. The PAUSE bit indicates the end point of the sub-queue. Therefore, both can occur in the same command or in separated ones. If two or more REP bits are read before a PAUSE bit, this is an error case and the intermediary REP bits are ignored. 1 Indicates the start point of the sub-queue to be repeated. 0 It is not the start point of a loop.
5 EB	External Buffer Bit A negated EB bit indicates that the command is sent to an internal CBuffer. Command is sent to an internal buffer.
6 BN	Buffer Number Bit BN indicates which buffer the message will be stored in. Buffers 1 and 0 can either internal or external depending on the EB bit setting. Message stored in buffer 1. Message stored in buffer 0.
7 CAL	CALibration Bit CAL indicates if the returning conversion result must be calibrated. Calibrate conversion result. Do not calibrate conversion result.

Table 523. Conversion Command Format for the Standard Configuration field description

Field	Description
8-11 MESSAGE_TAG [0:3]	MESSAGE_TAG Field The MESSAGE_TAG allows the EQADC to separate returning results into different RFIFOs. *Table 524* describes the meaning of the MESSAGE_TAG. When the EQADC transfers a command, the MESSAGE_TAG is included as part of the command. Eventually the external device/on-chip ADC returns the result with the same MESSAGE_TAG. The EQADC separates incoming messages into different RFIFOs by decoding the MESSAGE_TAG of the incoming data.
12-13 LST [0:1]	Long Sampling Time These two bits determine the duration of the sampling time in ADC clock cycles.
14 TSR	Time Stamp Request TSR indicates the request for a time stamp. When TSR is asserted, the on-chip *ADC Control Logic* returns a time stamp for the current conversion command after the conversion result is sent to the RFIFOs. See *Section , Time Stamp Feature*, for details. Return conversion time stamp after the conversion result. Return conversion result only.
15 FMT	Conversion Data Format FMT specifies to the EQADC how to format the 12-bit conversion data returned by the ADCs into the 16-bit format which is sent to the RFIFOs. See *Section , ADC Result Format for On-Chip ADC Operation*, for details. Right justified signed. Right justified unsigned.
16-23 CHANNEL_NUMBER [0:7]	Channel Number Field The CHANNEL_NUMBER field selects the analog input channel. The software programs this field with the channel number corresponding to the analog input pin to be sampled and converted. See *Section , Channel assignment*, for details.

Table 524. MESSAGE_TAG Description

MESSAGE_TAG[0:3]	MESSAGE_TAG Meaning
0b0000	Result is sent to RFIFO 0
0b0001	Result is sent to RFIFO 1
0b0010	Result is sent to RFIFO 2
0b0011	Result is sent to RFIFO 3
0b0100	Result is sent to RFIFO 4
0b0101	Result is sent to RFIFO 5
0b0110 - 0b0111	Reserved
0b1000	Null Message Received
0b1001	Reserved for customer use(1)
0b1010	Reserved for customer use[1]
0b1011 - 0b1111	Reserved

1. These messages are treated as null messages. Therefore, they must obey the format for incoming null messages and return valid BUSY0/1 fields. Refer to *Section , Null Message Format for External Device Operation*.

Table 525. Sampling Time

LST[0:1]	Sampling cycles (ADC Clock Cycles)
0b00	2
0b01	8
0b10	64
0b11	128

Conversion Command Format for Alternate Configurations

Figure 495 describes the format for conversion commands when interfacing with the on-chip ADCs in one of the 8 alternate configurations. An alternate configuration is selected when the lowest byte (bits 24-31) of the conversion command is set to a value in the range 0x08-0x0F. Each value in this range selects one of the 8 alternate configuration (0x08 selects Alternate Configuration 1, 0x0F selects Alternate Configuration 8). In the alternate configurations, the conversion result can be routed to one of the RFIFOs or to the parallel side interface to communicate with an on-chip companion module. A bit field in the corresponding Alternate Configuration Control Register selects the Internal RFIFO or Parallel Side Interface as the destination for the conversion result. A time stamp information can be optionally requested.

All fields, except FFMT and ALT_CONFIG_SEL, are identical to the ones in the standard configuration format. Only the fields which are different from the standard format will be described here.

Figure 495. Conversion Command Format for Alternate Configurations

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
EOQ	PAUSE	REP	RESERVED		EB (0b0)	BN	CAL	MESSAGE_TAG				LST		TSR	FFMT

CFIFO Header ADC Command

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
CHANNEL_NUMBER								ALT_CONFIG_SEL							

ADC Command

Table 526. Conversion Command Format for Alternate Configurations field description

Field	Description
15 FFMT	Flush or Format The function of this bit depends on the DEST field of the Alternate Configuration Control Register. If DEST is equal to 0b000, then FFMT defines the format in which the 12-bit conversion result are stored in the RFIFOs. If DEST is not equal to 0b000, then the FFMT bit is used to send a flush (soft-reset) signal through the parallel side interface to the companion module addressed by the DEST field. In case DEST is not equal to 0b000, the FMTA bit in the Alternate Configuration Control register is used to define the conversion result format. 1 Conversion Result Format set to right justified signed if DEST is equal to 0b000. A flush signal is sent through the side interface if DEST is not equal to 0b000. 0 Conversion Result Format set to right justified unsigned if DEST is equal to 0b000. No flush signal is sent through the side interface if DEST is not equal to 0b000. The flush signal can be asserted along with a valid conversion result. In this case the companion module should execute the software-reset first and then consider the conversion result as a valid data for the filtering algorithm.
24-31 ALT_CONFIG_SEL	Alternate Configuration Selection This field selects one of the alternate configurations according to *Table 527*.

Table 527. Alternate Configuration Selection

ALT_CONFIG_SEL[0:7]	Alternate Configuration
0x08	1
0x09	2
0x0A	3
0x0B	4
0x0C	5
0x0D	6
0x0E	7
0x0F	8

Write Configuration Command Format for On-Chip ADC Operation

Figure 495 describes the command message format for a write configuration command when interfacing with the on-chip ADCs. A write configuration command is used to set the control registers of the on-chip ADCs. No conversion data will be returned for a write configuration command. Write configuration commands are differentiated from read configuration commands by a negated R/W bit.

Figure 496. Write Configuration Command Format for On-Chip ADC Operation

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
EOQ	PAUSE	REP	RESERVED		EB (0b0)	BN	R/W (0b0)	ADC_REGISTER HIGH BYTE							

CFIFO Header ADC Command

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
ADC_REGISTER LOW BYTE								ADC_REG_ADDRESS							

ADC Command

Table 528. Write Configuration Command Format for On-Chip ADC Operation field description

Field	Description
0 EOQ	End Of Queue Bit
1 PAUSE	Pause Bit
2 REP	Repeat/loop Start Point Indication Bit
5 EB	External Buffer Bit
6 BN	Buffer Number Bit Refer to *Section , Conversion Command Format for the Standard Configuration.*
7 R/W	Read/Write bit A negated R/W indicates a write configuration command. Write
8-11 ADC_REGISTER_HIGH_BYTE [0:7]	ADC Register High Byte Field REGISTER_HIGH_BYTE is the value to be written into the most significant 8 bits of control/configuration register when the R/W bit is negated.
16-23 ADC_REGISTER_LOW_BYTE [0:7]	Register Low Byte Field REGISTER_LOW_BYTE is the value to be written into the least significant 8 bits of a control/configuration register when the R/W bit is negated.
24-31 ADC_REG_ADDRESS [0:7]	ADC Register Address The ADC_REG_ADDRESS field selects a register on the ADC register set to be written or read. Only half-word addresses can be used.

Read Configuration Command Format for On-Chip ADC Operation

Figure 497 describes the command message format for a read configuration command when interfacing with the on-chip ADCs. A read configuration command is used to read the contents of the on-chip ADC registers which are only accessible via command messages. Read configuration commands are differentiated from write configuration commands by an asserted R/W bit.

Figure 497. Read Configuration Command Format for On-Chip ADC Operation

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
EOQ	PAUSE	REP	RESERVED		EB (0b0)	BN	R/W (0b1)	MESSAGE_TAG				RESERVED			
CFIFO Header								ADC Command							

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
RESERVED								ADC_REG_ADDRESS							
ADC Command															

Table 529. Read Configuration Command Format for On-Chip ADC Operation field description

Field	Description
0 EOQ	End Of Queue Bit
1 PAUSE	Pause Bit
2 REP	Repeat/loop Start Point Indication Bit
5 EB	External Buffer Bit
6 BN	Buffer Number Bit Refer to *Section , Conversion Command Format for the Standard Configuration*..
7 R/W	R/W - Read/Write bit An asserted R/W bit indicates a read configuration command. 1 Read
8-11 MESSAGE_TAG [0:3]	MESSAGE_TAG Field Refer to *Section , Conversion Command Format for the Standard Configuration*..
24-31 ADC_REG_ADDRESS [0:7]	ADC Register Address The ADC_REG_ADDRESS field selects a register on the ADC register set to be written or read. Only half-word addresses can be used.

ADC Result Format for On-Chip ADC Operation

When the *FIFO Control Unit* receives a return data message, it decodes the MESSAGE_TAG field and stores the 16-bit data into the appropriate RFIFO. This section describes the *ADC result* portion of the *result message* returned by the on-chip ADCs. The 16-bit data stored in the RFIFOs can be:

- Data read from an ADC register with a read configuration command. In this case, the stored 16-bit data corresponds to the contents of the ADC register that was read.
- A time stamp. In this case, the stored 16-bit data is the value of the time base counter latched when the EQADC detects the end of the analog input voltage sampling. For

details see *Section , Time Stamp Feature*.

- A conversion result, coming directly from the ADCs. In this case, the stored 16-bit data contains a right justified 14-bit result data. The conversion result can be calibrated or not depending on the status of CAL bit in the command that requested the conversion[at]. When the CAL bit is negated, this 14-bit data is obtained by executing a 2-bit left-shift on the 12-bit data resultant from the resolution adjustment on the 8 or 10 or 12-bit data received from the ADC. The resolution adjustment consists of changing the conversion result input from 8, 10 or 12 bits right aligned to a 12-bit word left aligned - refer to *Section , ADC resolution selection feature*, for details. When the CAL bit is asserted, this 14-bit data is the result of the calculations performed in the EQADC MAC unit using the 12-bit data result of the resolution adjustment and the calibration constants GCC and OCC, or ALTGCC and ALTOCC - refer to *Section , ADC Calibration Feature*, for details. Then, this 14-bit data is further formatted into a 16-bit format according to the status of the FMT bit in conversion command of the standard configuration or FFMT bit in the conversion command of the alternate configurations[au]. When FMT/FFMT is asserted, the 14-bit result data is reformatted to look as if it was measured against an imaginary ground at VREF/2 (the MSB bit of the 14-bit result is inverted), and is sign-extended to a 16-bit format as in *Figure 498*. When FMT/FFMT is negated, the EQADC zero-extends the 14-bit result data to a 16-bit format as in *Figure 499*. Correspondence between the analog voltage in a channel and the calculated digital values is shown in *Table 531*.

Figure 498. ADC Result Format when FMT=1 (Right Justified Signed)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
SIGN_EXT		RESOLUTION ADJUSTED CONVERSION_RESULT (With inverted MSB bit)													

ADC Result

Figure 499. ADC Result Format when FMT=0 (Right Justified Unsigned)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	RESOLUTION ADJUSTED CONVERSION_RESULT													

ADC Result

[at] In case the conversion result is routed through an on-chip DSP via side interface, the calibration is applied before the data is sent to the DSP.

[au] For simplicity, the following text will refer to FMT only, but when using alternate configurations, refer to *Section , Conversion Command Format for Alternate Configurations*.

Table 530. ADC Result Format (Right Justified Signed) field description

Field	Description
0-1 SIGN_EXT [0:1]	Sign Extension field SIGN_EXT only has meaning when FMT is asserted. SIGN_EXT is 0b00 when CONVERSION_RESULT is positive, and 0b11 when CONVERSION_RESULT is negative.
2-15 CONVERSION_RESULT [0:13]	Conversion Result field CONVERSION_RESULT is a digital value corresponding to the analog input voltage in a channel when the conversion command was initiated. The two's complement representation is used to express negative values.

Table 531. Correspondence between analog voltages and digital values(1), (2)

	Voltage Level on Channel (V)	Corresponding 8-bit Conversion Result Returned by the ADC	Corresponding 10-bit Conversion Result Returned by the ADC	Corresponding 12-bit Conversion Result Returned by the ADC	16-bit Result Sent to RFIFOs (FMT=0) (3)	16-bit Result Sent to RFIFOs (FMT=1) (3)
Single-Ended Conversions	5.12	-	-	0xFFF	0x3FFC	0x1FFC
		-	0x3FF	-	0x3FF0	0x1FF0
		0xFF	-	-	0x3FC0	0x1FC0
	5.12 - LSB	-	-	0xFFF	0x3FFC	0x1FFC
		-	0x3FF	-	0x3FF0	0x1FF0
		0xFF	-	-	0x3FC0	0x1FC0
	...	...	...	...	...	...
	2.56	-	-	0x800	0x2000	0x0000
		-	0x200	-	0x2000	0x0000
		0x80	-	-	0x2000	0x0000
	...	...	...	...	...	...
	1 LSB	-	-	0x001	0x0004	0xE004
		-	0x001	-	0x0010	0xE010
		0x01	-	-	0x0040	0xE040
	0	0x00	0x000	0x000	0x0000	0xE000

Table 531. Correspondence between analog voltages and digital values[(1), (2)]

	Voltage Level on Channel (V)	Corresponding 8-bit Conversion Result Returned by the ADC	Corresponding 10-bit Conversion Result Returned by the ADC	Corresponding 12-bit Conversion Result Returned by the ADC	16-bit Result Sent to RFIFOs (FMT=0) [(3)]	16-bit Result Sent to RFIFOs (FMT=1) [(3)]
Differential Conversions	2.56	-	-	0xFFF	0x3FFC	0x1FFC
		-	0x3FF	-	0x3FF0	0x1FF0
		0xFF	-	-	0x3FC0	0x1FC0
	2.56 - LSB	-	-	0xFFF	0x3FFC	0x1FFC
		-	0x3FF	-	0x3FF0	0x1FF0
		0xFF	-	-	0x3FC0	0x1FC0
	...	...	...	...	...	...
	0	-	-	0x800	0x2000	0x0000
		-	0x200	-	0x2000	0x0000
		0x80	-	-	0x2000	0x0000
	...	...	...	...	...	...
	2.56 - LSB	-	-	0x001	0x0004	0xE004
		-	0x001	-	0x0010	0xE010
		0x01	-	-	0x0040	0xE040
	-2.56	0x00	0x000	0x000	0x0000	0xE000

1. VREF=VRH-VRL=5.12V. Resulting in one 12-bit count (LSB) =1.25mV.
2. The two's complement representation is used to express negative values.
3. Assuming uncalibrated conversion results.

Message Formats for External Device Operation

This section describes the Command Messages, Data Messages, and Null Messages formats used for external device operation.

Command Message Format for External Device Operation

Figure 500 describes the command message format for external device operation. Command message formats for on-chip operation and for external device operation share the same CFIFO header format. However, there are no limitations regarding the format an ADC Command used to communicate to an arbitrary external device. Only the upper bit of an ADC Command has a fixed format (BN field) to indicate to the *FIFO Control Unit*/external device to which CBuffer the corresponding command should be sent. The remaining 25 bits can be anything decodable by the external device. Only the ADC Command portion of a command message is transferred to the external device.

Figure 500. Command Message Format for External Device Operation

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
EOQ	PAUSE	RESERVED		ABORT_ST	EB (0b1)	BN	OFF_CHIP_COMMAND								

CFIFO Header ADC Command

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
OFF_CHIP_COMMAND															

ADC Command

Table 532. Command Message Format for External Device Operation field description

Field	Description
0 EOQ	End Of Queue Bit
1 PAUSE	Pause Bit Refer to *Section , Conversion Command Format for the Standard Configuration*.
4 ABORT_ST	ABORT Serial Transmission Bit ABORT_ST indicates whether an on-going serial transmission should be aborted or not. All CFIFOs can abort null message transmissions when triggered but only CFIFO0 can abort command transmissions of lower priority CFIFOs. For more on serial transmission aborts see *Section , CFIFO Common Prioritization and Command Transfer*. Abort current serial transmission. Do not abort current serial transmission.
5 EB	External Buffer Bit An asserted EB bit indicates that the command is sent to an external CBuffer. Command is sent to an external CBuffer.
6 BN	Buffer Number Bit Refer to *Section , Conversion Command Format for the Standard Configuration*.
7-31 OFF_CHIP_COMMAND [0:24]	OFF-CHIP COMMAND Field The OFF_CHIP_COMMAND field can be anything decodable by the external device. It is 25 bits long and it is transferred together with the BN bit to the external device when the CFIFO is triggered. Refer to *Section , Conversion Command Format for the Standard Configuration*, for a description of the command message used when interfacing with the on-chip ADCs.

Result Message Format for External Device Operation

Data is returned from the ADCs in the form of Result Messages. A result message is composed of an RFIFO header and an ADC Result. The *FIFO Control Unit* decodes the information contained in the RFIFO header and sends the contents of the ADC Result to the appropriate RFIFO. Only data stored on the ADC_RESULT field is stored in the RFIFOs/RQueues. The ADC result of any received message with a Null Data Message Tag will be ignored. The format of a Result Message returned from the external device is shown in *Figure 501*. It is 26 bits long, and is composed of a MESSAGE_TAG field, information about the status of the CBuffers (BUSY fields), and result data. The BUSY fields are needed to inform the EQADC about when it is appropriate to transfer commands to the external CBuffers.

Figure 501. Result Message Format for External Device Operation

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
RESERVED								MESSAGE_TAG				BUSY1		BUSY0	

RFIFO Header

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
ADC_RESULT															

ADC Result

Table 533. Result Message Format for External Device Operation field description

Field	Description
8-11 MESSAGE_TAG [0:3]	MESSAGE_TAG Field Refer to *Section , Conversion Command Format for the Standard Configuration*.
12-13 BUSY1[0:1]	BUSY Status field The BUSY fields indicate if the external device can receive more commands. *Table 534* shows how these two bits are encoded. When an external device cannot accept any more new commands, it must set BUSYx to a value indicating “Do not send commands” in the returning message. The BUSY fields of values 0b10 and 0b10 can be freely encoded by the external device to allow visibility of the status of the external CBuffers for debug, they could indicate the number of entries in a external CBuffer for example. After reset, the EQADC always assumes that the external CBuffers are full and cannot receive commands.

Table 533. Result Message Format for External Device Operation field description (continued)

Field	Description
14-15 BUSY0[0:1]	BUSY Status field The BUSY fields indicate if the external device can receive more commands. *Table 534* shows how these two bits are encoded. When an external device cannot accept any more new commands, it must set BUSYx to a value indicating “Do not send commands” in the returning message. The BUSY fields of values 0b10 and 0b10 can be freely encoded by the external device to allow visibility of the status of the external CBuffers for debug, they could indicate the number of entries in a external CBuffer for example. After reset, the EQADC always assumes that the external CBuffers are full and cannot receive commands.
16-31 ADC_RESULT [0:15]	ADC RESULT Field ADC_RESULT is the result data received from the external device or on-chip ADC. This can be the result of a conversion command, data requested via a read configuration command, or time stamp value. The ADC_RESULT of any incoming message with a Null Message tag will be ignored. When the MESSAGE_TAG is for an RFIFO, the EQADC extracts the 16-bit ADC_RESULT from the raw message and stores it into the appropriate RFIFO.

Table 534. Command BUFFERx BUSY Status

BUSYx[0:1]	Meaning
0b00	Send available commands - CBuffer is empty
0b01	Send available commands
0b10	Send available commands
0b11	Do not send commands

Null Message Format for External Device Operation

Null Messages are only transferred through the serial interface to allow results and unsolicited control data, like the status of the external CBuffers, to return when there are no more commands pending to transfer. Null Messages are only transmitted when serial transmissions from the EQADC SSI are enabled (See ESSIE field in *Section , EQADC Module Configuration Register (EQADC_MCR)*), and when one of the following conditions apply:

1. there are no triggered CFIFOs with commands bound for external CBuffers, or;
2. there are triggered CFIFOs with commands bound for external CBuffers but the external CBuffers are full. The EQADC detected returning BUSYx fields indicating “Do not send commands”.

Figure 502 illustrates the null message send format. When the EQADC transfers a null message, it directly shifts out the 26-bit data content inside the *Section , EQADC null message send format register (EQADC_NMSFR)*. The register must be programmed with the null message send format of the external device.

Figure 503 illustrates the null message receive format. It has the same fields found in a Result Message with the exception that the ADC result is not used. Refer to *Section , Result Message Format for External Device Operation*, for more information. The MESSAGE_TAG field must be set to the Null Message tag (0b1000). The EQADC does not store into an RFIFO any incoming message with a Null Message tag.

Figure 502. Null Message Send Format for External Device Operation

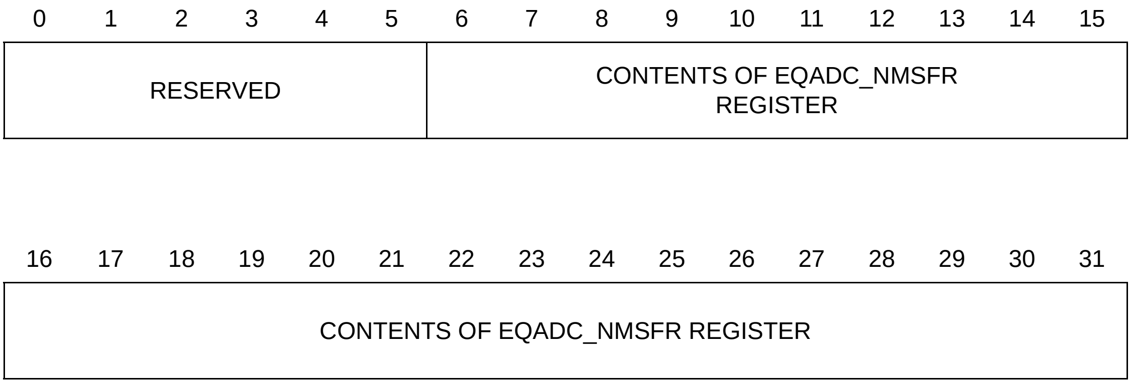


Figure 503. Null Message Receive Format for External Device Operation

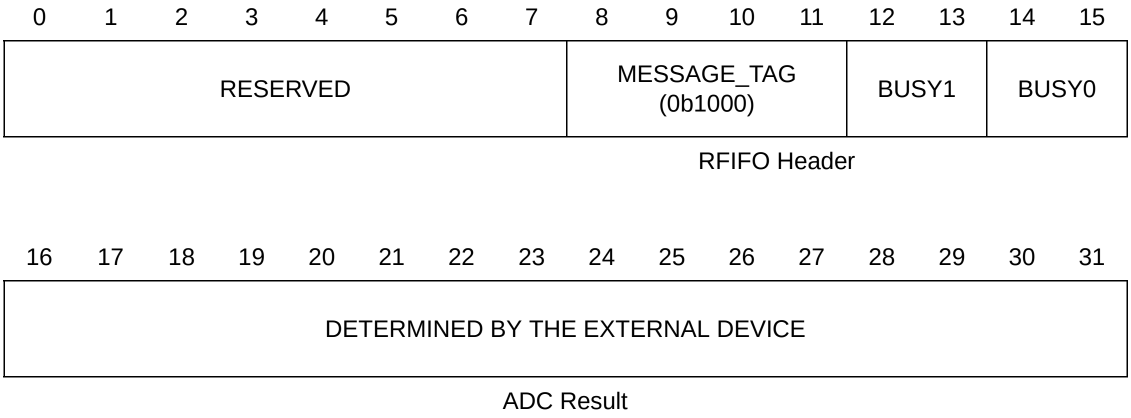


Table 535. field description

Field	Description
8-11 MESSAGE_TAG [0:3]	MESSAGE_TAG Field Refer to *Section , Conversion Command Format for the Standard Configuration*.
21-13 BUSY1 [0:1]	BUSY Status field Refer to *Section , Result Message Format for External Device Operation*.
14-15 BUSY0 [0:1]	BUSY Status field Refer to *Section , Result Message Format for External Device Operation*.

23.6.3 Command/Result Queues

The Command and Result queues (CQueues and RQueues) are actually part of the EQADC system although they are not hardware implemented inside the EQADC. Each CQueue entry is a 32-bit Command Message.The last entry of a CQueue has the EOQ bit asserted to indicate that it is the last entry of the CQueue. RQueue entry is a 16-bit data.

See *Section , Overview and Basic Terminology*, for a description of the message formats and their flow in EQADC.

Refer to *Section 23.7.5, CQueue and RQueues usage*, for examples of how CQueues and RQueues can be used.

23.6.4 EQADC Command FIFOs

CFIFO Basic Functionality

There are six prioritized CFIFOs located in the EQADC. Each CFIFO is four entries deep, except CFIFO0 that can be configured to eight entries deep in extended mode, and each CFIFO entry is 32 bits long. A CFIFO serves as a temporary storage location for the command messages stored on the CQueues in the system memory. When a CFIFO is not full, the EQADC sets the corresponding CFFF bit in *Section , EQADC FIFO and Interrupt Status Registers (EQADC_FISR)*. If CFFE is asserted in *Section , EQADC Interrupt and DMA Control Registers (EQADC_IDCR)*, the EQADC generates requests for more commands from a CQueue. An interrupt request, served by the host CPU, is generated when CFFS is negated, and a DMA request, served by the DMAC, is generated when CFFS is asserted. The host CPU or the DMAC respond to these requests by writing to the *Section , EQADC CFIFO Push Registers (EQADC_CFPR)*, to fill the CFIFO.

Note: *The DMAC should be configured to write a single command (32-bit data) to the CFIFO push registers for every asserted DMA request it acknowledges. Refer to Section 23.7.2, EQADC/DMAC Interface, for DMAC configuration guidelines.*

Note: *CFIFO0 can be configured to work in an alternative way called Streaming Mode. This mode is very different from the mode described here because it maintains some stored commands to execute them several times in sequence and in loop.*

Note: *Only whole words must be written to EQADC_CFPR. Writing half-words or bytes to EQADC_CFPR will still push the whole 32-bit CF_PUSH field into the corresponding CFIFO, but undefined data will fill the areas of CF_PUSH that were not specifically designated as target locations for writing.*

Figure 504 describes the important components in the CFIFO. Each CFIFO is implemented as a circular set of registers to avoid the need to move all entries at each push/pop operation. The Push Next Data Pointer points to the next available CFIFO location for storing data written into the EQADC Command FIFO Push Register. The Transfer Next Data Pointer points to the next entry to be removed from CFIFOx when it completes a transfer. The *CFIFO Transfer Counter Control Logic* counts the number of entries in the CFIFO and generates DMA or interrupt requests to fill the CFIFO. TNXTPTR in *Section , EQADC FIFO and Interrupt Status Registers (EQADC_FISR)*, indicates the index of the entry that is currently being addressed by the Transfer Next Data Pointer, and CFCTR, in the same register, provides the number of entries stored in the CFIFO. Using TNXTPTR and CFCTR, the absolute addresses for the entries indicated by the Transfer Next Data Pointer and by the Push Next Data Pointer can be calculated using the following formulas:

Transfer Next Data Pointer Address = CFIFOx_BASE_ADDRESS + TNXTPTRx*4

Push Next Data Pointer Address = CFIFOx_BASE_ADDRESS +
[(TNXTPTRx+CFCTRx) mod CFIFO_DEPTH] * 4

where

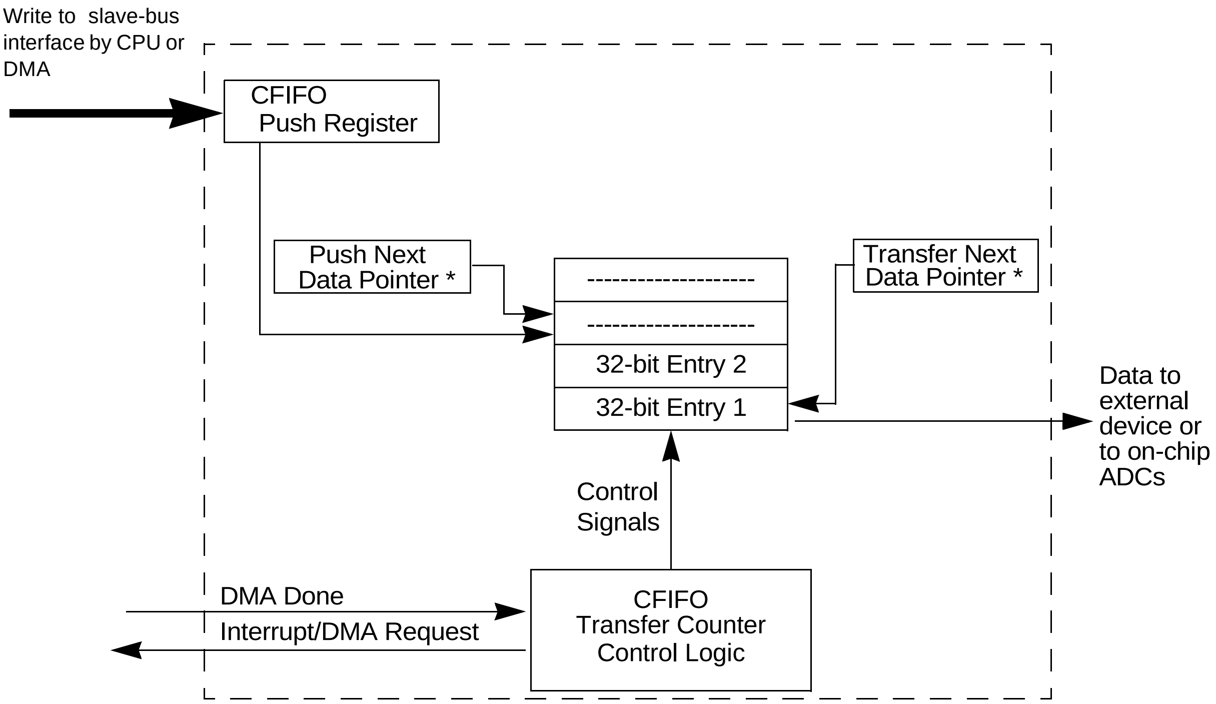
- *a mod b* returns the remainder of the division of *a* by *b*.
- CFIFOx_BASE_ADDRESS is the smallest memory mapped address allocated to a CFIFOx entry.
- CFIFO_DEPTH is the number of entries contained in a CFIFO - four in this implementation.

When CFS*x* in *Section , EQADC CFIFO Status Register (EQADC_CFSR)*, is TRIGGERED, the EQADC generates the proper control signals for the transfer of the entry pointed by Transfer Next Data Pointer. CFUFx in *Section , EQADC FIFO and Interrupt Status Registers (EQADC_FISR)*, is set when a CFIFOx underflow event occurs. A CFIFO underflow occurs when the CFIFO is in TRIGGERED state and it becomes empty. No commands will be transferred from an underflowing CFIFO, nor will command transfers from lower priority CFIFOs be blocked. CFIFOx is empty when the Transfer Next Data Pointer x equals the Push Next Data Pointer x and CFCTRx is zero. CFIFOx is full when the Transfer Next Data Pointer x equals the Push Next Data Pointer x and CFCTRx is not zero.

When the EQADC completes the transfer of an entry from CFIFOx: the transferred entry is popped from CFIFOx, the CFIFO counter CFCTR in the *Section , EQADC FIFO and Interrupt Status Registers (EQADC_FISR)*, is decremented by one, and Transfer Next Data Pointer x is incremented by one (or wrapped around) to point to the next entry in the CFIFO. The transfer of entries bound for the on-chip ADCs is considered completed when they are stored in the appropriate CBuffer. The transfer of entries bound for the external device is considered completed when the serial transmission of the entry is completed.

When the EQADC_CFPRx is written and CFIFOx is not full, the CFIFO counter CFCTRx is incremented by one, and the Push Next Data Pointer x then is incremented by one (or wrapped around) to point to the next entry in the CFIFO.

When the EQADC_CFPRx is written but CFIFOx is full, the EQADC will not increment the counter value and will not overwrite any entry in CFIFOx.



* All CFIFO entries are memory mapped and the entries addressed by these pointers can have their absolute addresses calculated using TNXTPTR and CFCTR.

Figure 504. CFIFO Diagram

The detailed behavior of the Push Next Data Pointer and Transfer Next Data Pointer is described in the example shown in *Figure 505* where a CFIFO with 16 entries is shown for clarity of explanation, the actual hardware implementation has only four entries. In this example, CFIFOx with 16 entries is shown in sequence after pushing and transferring entries.

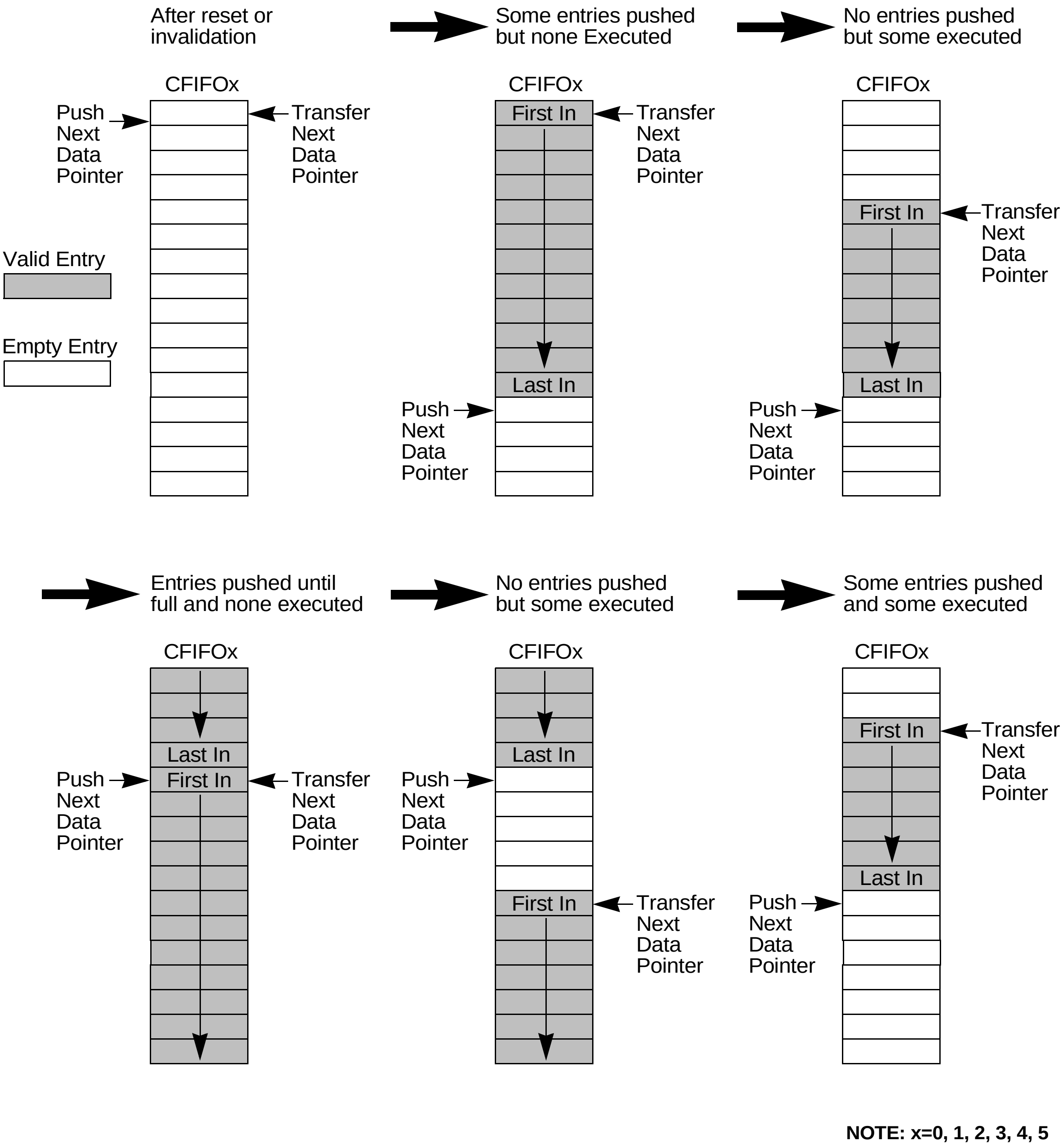


Figure 505. CFIFO Entry Pointer Example

CFIFO0 Streaming Mode Description

CFIFO0 can be configured to operate in streaming mode to allow repetition of a group of commands several times without the need of refilling the registers as in the normal mode of operation of CFIFOs. This mode makes use of the additional bit in the Conversion

Command Word (CCW) called ‘Repeat’ (REP bit). The purpose of this bit is to mark in the command queue, where to start a repeating sequence. This location is stored in an additional pointer ‘Repeat Pointer’.

Streaming mode requires 2 trigger inputs. The standard queue 0 trigger, in this mode referred to as Repeat Trigger and a new internal trigger input to the eQADC called Advance Trigger (no filter available).

CFIFO0 is configured to operate in streaming mode by setting the bit STRME0 as described in *Section , EQADC CFIFO Control Registers (EQADC_CFCR)*. CFIFO0 is eight entries deep in extended mode by setting the bit CFEEE0 in the same EQADC_CFCR register, and each entry is 32 bits long. This CFIFO0 serves as a local storage of a few commands that need to be executed sequentially as in a FIFO but can contain sub-queues that need to be executed several times. The CFFF0 bit in *Section , EQADC FIFO and Interrupt Status Registers (EQADC_FISR)*, is used to assure the CFIFO0 is not full and command messages are stored from address 0x0 to 0x7.

CFIFO0 Operation in Streaming Mode

In Streaming mode, the CFIFO0 is filled with CCWs using the DMA exactly the same as existing modes. The CFIFO executes commands as per the existing modes until it executes a Conversion Command Word with the Repeat bit set. When this CCW is executed, the Repeat Pointer is set to point to this FIFO location and from this CCW onwards, CFIFO0 entries is not invalidated, that is, the Repeat Pointer prevents this and subsequent entries from being overwritten.

The queue continues to execute until a CCW with an asserted Pause bit is completed; then the queue stops and enters the Pause state, waiting for a trigger. This is the same as normal behavior.

The Pause state is exited in one of two ways: Repeat Trigger or Repeat Trigger with Advance Trigger. The Repeat trigger with no Advance trigger causes the Transfer Next Data Pointer to be loaded with the Repeat Pointer location and CCWs are then executed from the Repeat Pointer back to the Pause bit. This means that a section of the CFIFO0 is repeatedly executed every time a Repeat Trigger occurs.

The Repeat trigger with the Advance trigger pending causes all CCWs from the Repeat pointer to the Pause bit to be invalidated and the CCW after the pause bit to be executed. This is achieved by invalidating the Repeat Pointer. The effect is that the queue advances beyond the repeating section of the CFIFO0 to execute new CCWs.

Note that the Advance trigger can occur at any time between Repeat triggers, but is only actioned when the next Repeat trigger occurs. Prior to that it is pending.

In a typical application, the queue is made of some configuration commands to the ADC (to flush the decimator or turn on pad pull-up/down) followed by a repeating section of ADC conversions on one or more ADC channels from one or more sensors; followed by a few more configuration commands; then more repeating ADC conversions, until the entire engine cycle is complete; when the queue is restarted. The mechanism described permits any number of repeating sub-queues to be loaded and executed, interspersed by configuration commands.

Triggering Description in Streaming Mode

The additional trigger signal ATRIG0 is detected by a separate circuit that is configured by the bit field AMODE0 as described in *Section , EQADC CFIFO Control Registers (EQADC_CFCR)*. This trigger signal is used as an advance control of pop pointer of

CFIFO0. In addition, it is used as the enable trigger for the Repeat trigger. This means it is necessary to have an Advance trigger first to enable the detection of the Repeat trigger. When the Repeat trigger is enabled, the Advance trigger is used to advance the pop pointer beyond some loop sub-queue. And it is to disable the Repeat trigger by executing a Pause without a previous REP bit.

A typical sequence of events is presented below to describe the relationship between the triggers.

In Streaming mode, the CFIFO0 is filled with CCWs using the DMA as usual. The two triggers are configured to positive edge and single scan mode.

The SSS bit is asserted and the trigger detector of the Repeat trigger is disabled in the start of the queue. It is necessary to receive the first Advance trigger to enable the detector of the other trigger. This enable is useful when the Repeat trigger is received all the time and the trigger signal can be disabled when it is not desired.

The Advance trigger is received and detected and the Repeat trigger detector is enabled. No commands are executed until now.

The Repeat trigger is detected and the commands start to be executed in sequence. If a REP bit is decoded with the PAUSE bit, the loop is configured and the CFIFO0 commands stop to be executed. The next Repeat trigger is waited to start the execution of the loop again, or the Advance trigger can be detected to break the loop and advance the queue in CFIFO0. The Repeat trigger detector remains enabled.

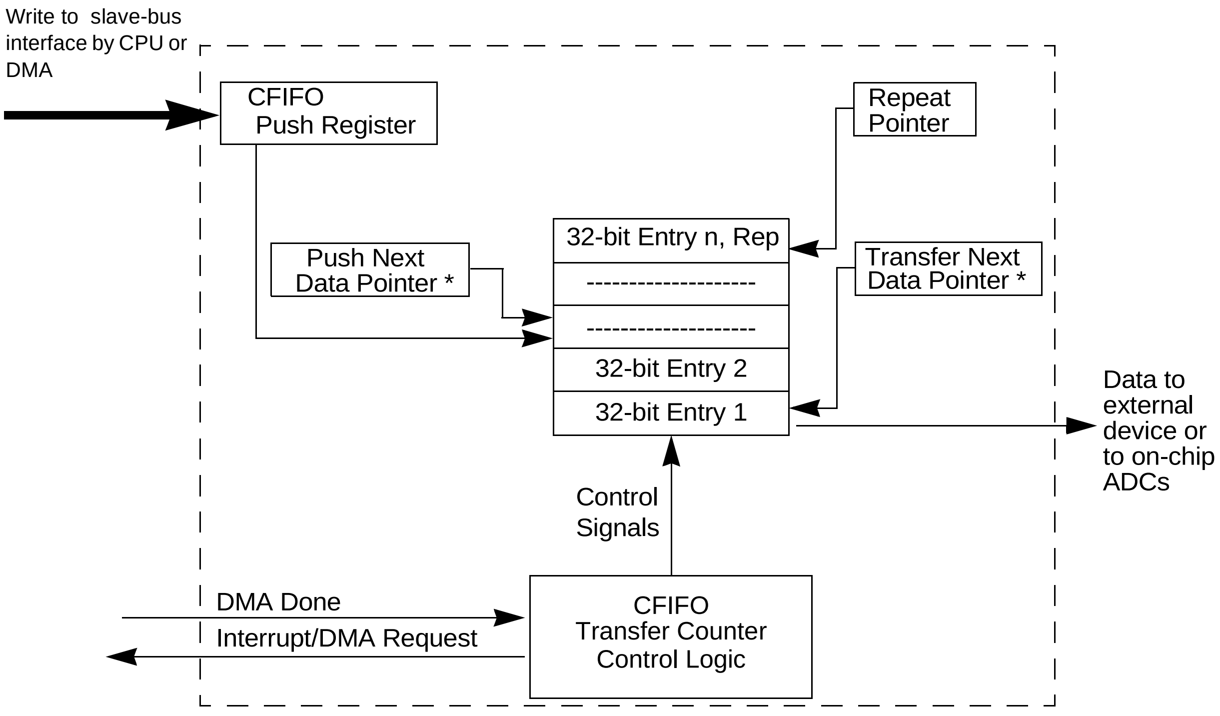
If the Advance trigger is received and the next command in the CFIFO0 does not present the REP bit set, this means the CFIFO0 is not starting a new loop. In this case (outside a loop) if a PAUSE bit is decoded, this means to disable the Repeat trigger detector. This can be useful if the Repeat trigger is not required for some interval of time. The Repeat trigger detector is enabled again when the next Advance trigger event is detected.

CFIFO0 Diagram Description in Streaming Mode

Figure 506 represents the main components of CFIFO0 in streaming mode. However, some signals behave in a different way from the common operation. The Push Next Data Pointer points to the next available CFIFO0 location for storing data written into the EQADC Command FIFO Push Register. The Transfer Next Data Pointer points to the next entry to be transferred to Cbuffer. The Repeat Pointer points to the first entry of the repeating sub-queue. TNXTPTR in *Section , EQADC FIFO and Interrupt Status Registers (EQADC_FISR)*, indicates the index of the entry that is currently being addressed by the Transfer Next Data Pointer, and CFCTR, in the same register, provides the number of entries stored in the CFIFO.

When CFS0 in *Section , EQADC CFIFO Status Register (EQADC_CFSR)*, is TRIGGERED, the EQADC generates the proper control signals for the transfer of the entry pointed by Transfer Next Data Pointer. CFUF0 in *Section , EQADC FIFO and Interrupt Status Registers (EQADC_FISR)*, is set when CFIFO0 underflow event occurs. A CFIFO underflow occurs when the CFIFO is in TRIGGERED state and it is empty. No commands will be transferred from an underflowing CFIFO, nor will command transfers from lower priority CFIFOs be blocked. CFIFO0 is empty when CFCTR0 is zero. CFIFO0 is full when (CFCTR0 mod CFIFO_DEPTH) is zero but CFCTR0 is not zero.

When the EQADC completes the transfer of an entry from CFIFO0 in loop condition: the transferred entry is not popped from CFIFO0, the CFIFO counter CFCTR in the *Section , EQADC FIFO and Interrupt Status Registers (EQADC_FISR)*, is not decremented by one, and Transfer Next Data Pointer 0 is incremented by one (or wrapped around) to point to the next entry in the CFIFO0.



* All CFIFO entries are memory mapped and the entries addressed by these pointers can have their absolute addresses calculated using TNXTPTR and CFCTR.

Figure 506. CFIFO0 in Streaming Mode Diagram

The detailed behavior of the Push Next Data Pointer and Transfer Next Data Pointer is described in the example shown in *Figure 507* where a CFIFO with 16 entries is shown for clarity of explanation, the actual hardware implementation has only four/eight entries. In this example, CFIFO0 with 16 entries is shown in sequence after pushing and transferring entries.

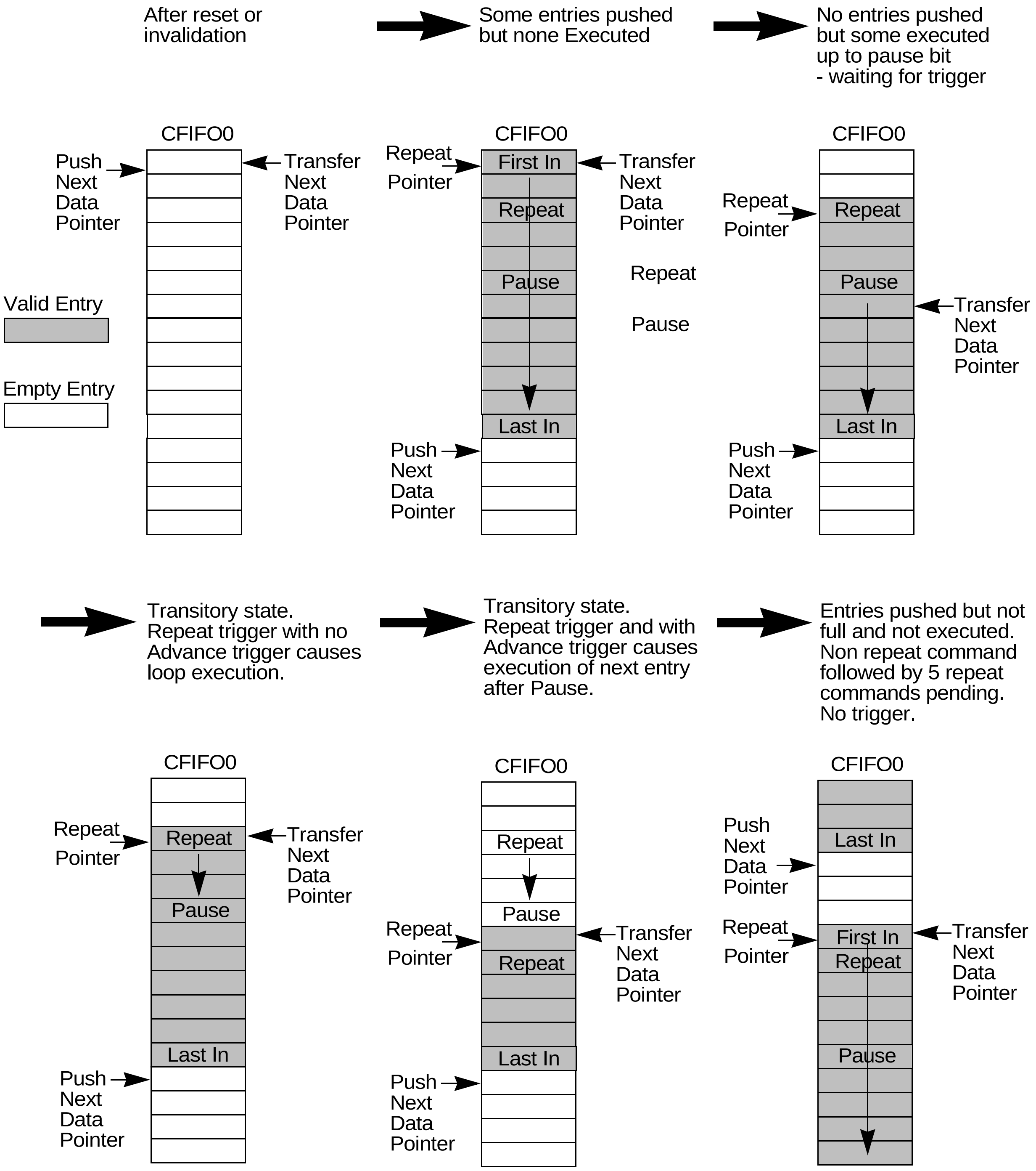


Figure 507. CFIFO0 in Streaming Mode Entry Pointer Example

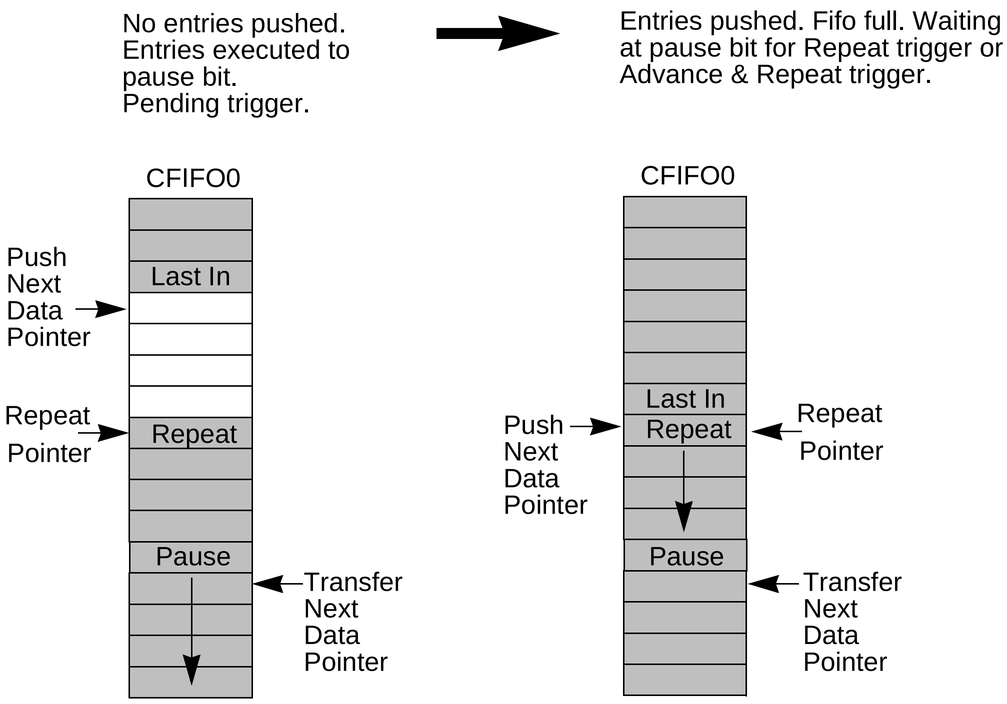


Figure 508. CFIFO0 in Streaming Mode Entry Pointer Example (Cont.)

Streaming Mode Error Conditions

In the repeat state, the existing error conditions still apply, but now there are new ways to trigger them. Now, the CCWs are not being invalidated so the DMA is not able to load more CCWs into those locations. So a queue overflow becomes more likely, and occurs if the repeat loop is longer than 8 entries. If all CCWs in the CFIFO0 are executed and no Pause bit or EOQ bit is detected, the eQADC will signal an underflow error. In practice this may limit a repeating queue to 7 entries since otherwise an underflow will occur at the point a Repeat with Advance trigger occurs, and there is no command in the CFIFO0 to execute. The exception is a final command with both a Pause and an EOQ bit set. The End of Queue bit EOQ continues to operate as in normal mode, unless the Repeat mode is enabled. In this case the Pause bit takes precedence and a Repeat trigger causes the jump back described. A Repeat trigger with Advance trigger causes the queue to end.

Another error condition occur when the repeat trigger is in the TRIGGERED state and a new repeat trigger is received. In this case, a trigger overflow occurs but the CFIFO0 is defined to not restart the loop. The trigger in this case is not used in the CFIFO0, but the overflow is indicated.

CFIFO Common Prioritization and Command Transfer

The CFIFO priority is fixed according to the CFIFO number. A CFIFO with a smaller number has a higher priority. When commands of distinct CFIFOs are bound for the same destination (CBuffer), the higher priority CFIFO is always served first. A TRIGGERED, not-underflowing CFIFO will start the transfer of its commands when:

- its commands are bound for an internal CBuffer that is not full, and it is the highest priority triggered CFIFO sending commands to that CBuffer.
- its commands are bound for an external CBuffer that is not full, and it is the highest priority triggered CFIFO sending commands to an external CBuffer that is not full.

A triggered CFIFO with commands bound for a certain CBuffer consecutively transfers its commands to it until:

- an asserted End Of Queue bit is reached, or;
- an asserted Pause bit is encountered and the CFIFO is configured for edge trigger mode, or;
- CFIFO is configured for level trigger mode and a closed gate is detected, or;
- in case its commands are bound for an internal CBuffer, a higher priority CFIFO that uses the same internal CBuffer is triggered, or;
- in case its commands are bound for an external CBuffer, a higher priority CFIFO that uses an external CBuffer is triggered.

The prioritization logic of the EQADC, depicted in *Figure 509*, is composed of three independent sub-blocks: one prioritizing CFIFOs with commands bound for CBuffer0, another prioritizing CFIFOs with commands for CBuffer1, and a last one prioritizing CFIFOs with commands for CBuffer2 and CBuffer3 which reside inside the external device. As these three sub-blocks are independent, simultaneous writes to CBuffer0, to CBuffer1, and to EQADC SSI transmit buffer are allowed. The hardware identifies the destination of a command by decoding the EB and BN bits in the command message - see *Section , Message Format in EQADC*, for details.

Note: *Triggered but empty CFIFOs, underflowing CFIFOs, are not considered for prioritization. No data from these CFIFOs will be sent to the CBuffers and nor will they stop lower priority CFIFOs from transferring commands.*

Whenever CBuffer0 is able to receive new entries, the prioritization sub-block selects the highest-priority triggered CFIFO with a command bound for CBuffer0, and writes its command into the buffer. In case CBuffer0 is able to receive new entries but there are no triggered CFIFOs with commands bound for it, nothing is written to the buffer. The sub-block prioritizing CBuffer1 usage behaves in the same way.

When the EQADC SSI is enabled and ready to start serial transmissions, the sub-block prioritizing EQADC SSI usage writes command or null messages into the EQADC SSI transmit buffer, data written to the EQADC SSI transmit buffer is subsequently transmitted to the external device through the EQADC SSI link. The sub-block writes commands to the EQADC SSI transmit buffer when there are triggered CFIFOs with commands bound for not-full external CBuffers. The command written to the transmit buffer belongs to the highest priority CFIFO sending commands to a external CBuffer that is not full. This implies that a lower priority CFIFO can have its commands sent if a higher priority CFIFO cannot send its commands due to a full CBuffer. The sub-block writes null messages to the EQADC SSI transmit buffer when there are no triggered CFIFOs with commands bound for external CBuffers, or when there are triggered CFIFOs with commands bound for external CBuffers but the external CBuffers are full. The EQADC monitors the status of the external CBuffers by decoding the BUSY fields of the incoming result messages from the external device - see *Section , Result Message Format for External Device Operation*, for details.

Note: *When a lower priority CFIFO is served first because a higher priority CFIFO cannot send its commands due to a full external CBuffer, there is a possibility that command transfers from the lower priority CFIFO will be interrupted and the CFIFO will become non-coherent, when the higher priority CFIFO again becomes ready to send commands. If the lower priority CFIFO becomes non-coherent or not depends on the rate at which commands on the external CBuffers are executed, on the rate at which commands are transmitted to the external CBuffers, and on the depth of those buffers.*

Once a serial transmission is started, the sub-block monitors triggered CFIFOs and manages the abort of serial transmissions. In case a null message is being transmitted, the serial transmission is aborted when all following conditions are met:

- A not-underflowing CFIFO in TRIGGERED state has commands bound for an external CBuffer that is not full, and it is the highest priority CFIFO sending commands to an external CBuffer that is not full.
- the ABORT_ST bit of the command to be transmitted is asserted.
- the 26th bit of currently transmitting null message has not being shifted out.

The command from the CFIFO is then written into EQADC SSI transmit buffer, allowing for a new serial transmission to initiate.

In case a command is being transmitted, the serial transmission is aborted when all following conditions are met:

- CFIFO0 is in TRIGGERED state, is not underflowing, and its current command is bound for an external CBuffer that is not full.
- the ABORT_ST bit of the command to be transmitted is asserted.
- the 26th bit of currently transmitting command has not being shifted out.

The command from CFIFO0 is then written into EQADC SSI transmit buffer, allowing for a new serial transmission to initiate.

Note: *The aborted command is not popped from the preempted CFIFO and will be retransmitted as soon as its CFIFO becomes the highest priority CFIFO sending commands to an external CBuffer that is not full.*

After a serial transmission is completed, the EQADC prioritizes the CFIFOs and schedules a command or a null message to be sent in the next serial transmission. After the data for the next transmission has been defined and scheduled, the EQADC can, under certain conditions, stretch the SDS negation time in order to allow the schedule of new data for that transmission. This occurs when the EQADC acknowledges that the status of a higher-priority CFIFO changed to TRIGGERED and attempts to schedule that CFIFO command before SDS is asserted. Only commands of CFIFOs that have the ABORT_ST bit asserted can be scheduled in this manner. Under such conditions:

1. a CFIFO0 command is scheduled for the next transmission independently of the type of data that was previously scheduled. The time during which SDS is negated is stretched in order to allow the EQADC to load the CFIFO0 command and start its transmission.
2. CFIFO1-5 commands are only scheduled for the next transmission if the previously scheduled data was a null message. The time during which SDS is negated is stretched in order to allow the EQADC to load that command and start its transmission. However, if the previously scheduled data was a command, no rescheduling occurs and the next transmission starts without delays.

If a CFIFO becomes TRIGGERED while SDS is negated, but the EQADC only attempts to reschedule that CFIFO command after SDS is asserted, then the current transmission is aborted depending on if the conditions for that are met or not.

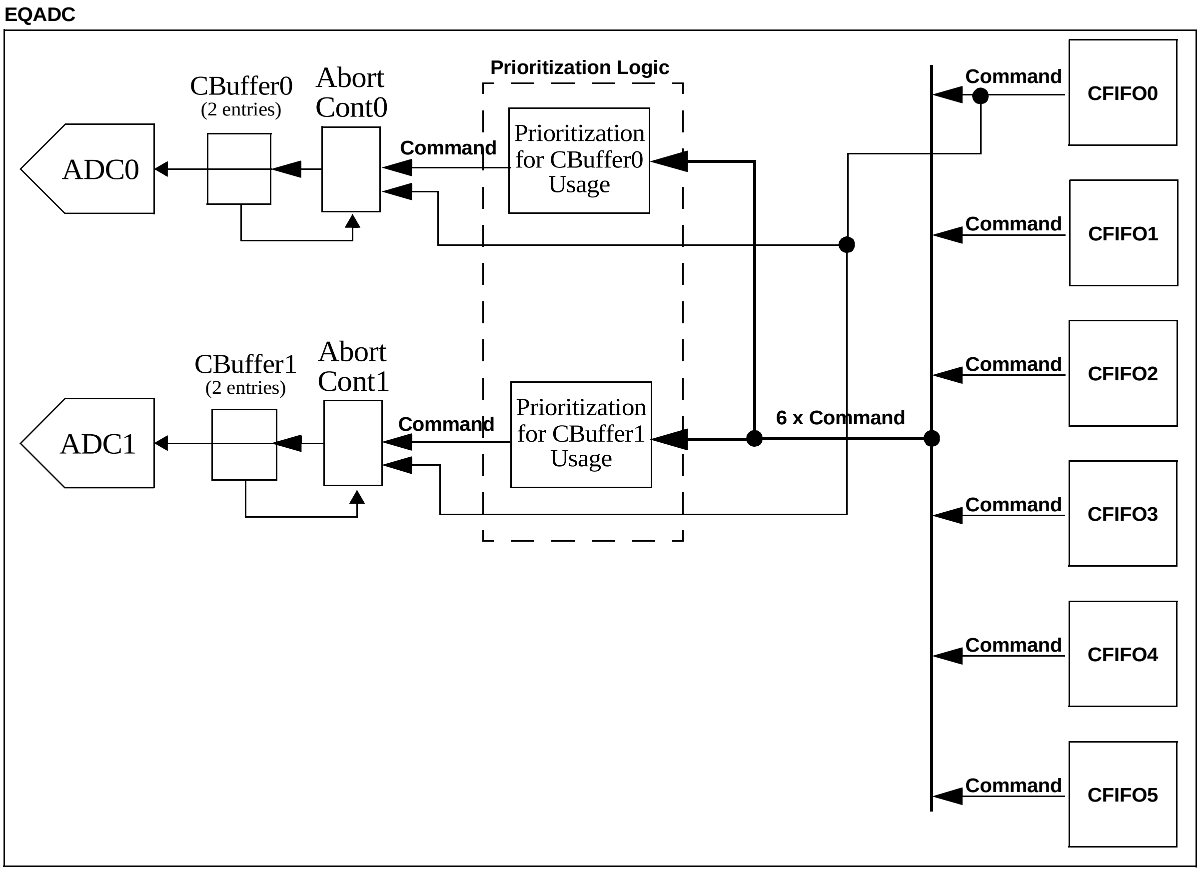


Figure 509. CFIFO Prioritization Logic

CFIFO Prioritization in Abort Mode

The CFIFO priority does not change when the EQADC is configured to allow abortion of conversion execution in on-chip ADC analog blocks. However, CFIFO0 is the only one that can be enabled to abort conversions.

This feature is necessary when the timing of some conversion is very important. In normal priority scheme, when CFIFO0 is triggered, its conversion command can be put behind 2 pending conversion commands in the Cbuffer due to the queue structure. Considering that these 2 pending commands are from lower priority CFIFOs and that the delay between the trigger and the sampling of the command from CFIFO0 can be unacceptable, EQADC can be configured to permit immediate conversion commands from CFIFO0 with abort function.

When CFIFO0 is triggered and abort is enabled, up to 2 commands in Cbuffer0 or Cbuffer1 are stored in a side register. The abort request signal is generated to ADC0 or ADC1 and the confirmation of ADC reset/ready is waited to send the command from CFIFO0 to the decoded Cbuffer.

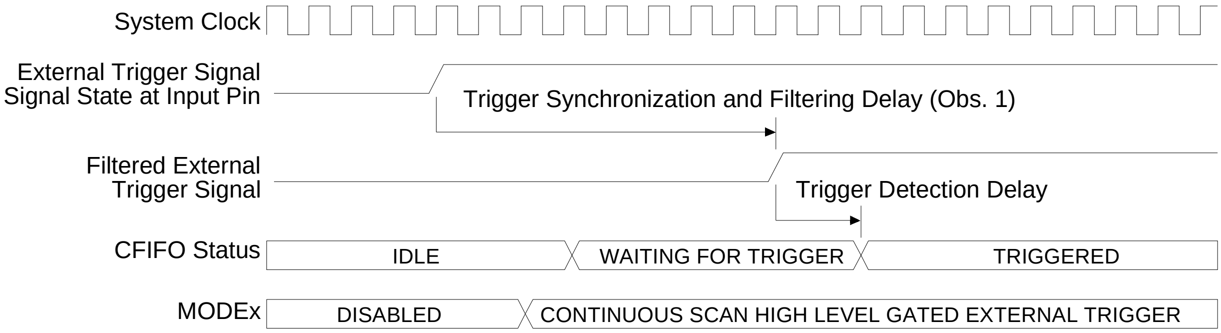
After the transfer of all commands from CFIFO0, the recovery phase restores the up to 2 commands that were in Cbuffer when the abort occurred. After this recovery phase, it is established the normal process of prioritization of commands from CFIFOs.

External Trigger Event Detection

The digital filters for trigger signals can be individually bypassed by asserting the input control signals eqadc_intern_trig_sel5-0. When the filter is bypassed, the ETRIG input signal is not filtered and the logic after the filter receives a copy of this input trigger signal.

The Digital Filter Length field in *Section , EQADC External Trigger Digital Filter Register (EQADC_ETDFR)*, specifies the minimum number of system clocks that the ETRIG0-5 signals must be held at a logic level to be recognized as valid. All ETRIG signals are filtered. A counter for each queue trigger is implemented to detect a transition between logic levels. The counter counts at the system clock rate. The corresponding counter is cleared and restarted each time the signal transitions between logic levels. When the corresponding counter matches the value specified by the Digital Filter Length field in *Section , EQADC External Trigger Digital Filter Register (EQADC_ETDFR)*, the EQADC considers the ETRIG logic level to be valid and passes that new logic level to the rest of the EQADC.

The filter is only for filtering the ETRIG signal. Logic after the filter checks for transitions between filtered values, such as for detecting the transition from a filtered logic level zero to a filter logic level one in rising edge external trigger mode. The EQADC can detect rising edge, falling edge, or level gated external triggers. The digital filter will always be active independently of the status of the MODEx field in *Section , EQADC CFIFO Control Registers (EQADC_CFCR)*, but the edge, level detection logic is only active when MODEx is set to a value different from disabled, and in case MODEx is set to single scan mode, when the SSS bit is asserted. Note that the time necessary for a external trigger event to result into a CFIFO status change is not solely determined by the DFL field in the *Section , EQADC External Trigger Digital Filter Register (EQADC_ETDFR)*. After being synchronized to the system clock and filtered, a trigger event is checked against the CFIFO trigger mode. Only then, after a valid trigger event is detected, the EQADC accordingly changes the CFIFO status. Refer to *Figure 510* for an example.



Obs.

- 1: This delay is about 2 clocks when the filter bypass control is asserted.

Figure 510. ETRIG Event Propagation Example

CFIFO Scan Trigger Modes

The EQADC supports two different scan modes, single-scan and continuous-scan. Refer to *Table 536* for a summary of these two scan modes. When a CFIFO is triggered, the EQADC scan mode determines whether the EQADC will stop command transfers from a CFIFO, and wait for software intervention to rearm the CFIFO to detect new trigger events, upon detection of an asserted EOQ bit in the last transfer. Refer to *Section , Message Format in EQADC*, for details about command formats.

CFIFOs can be configured in single-scan or continuous-scan mode. When a CFIFO is configured in single-scan mode, the EQADC scans the CQueue one time. The EQADC stops future command transfers from the triggered CFIFO after detecting the EOQ bit set in the last transfer. After a EOQ bit is detected, software involvement is required to rearm the CFIFO so that it can detect new trigger events.

When a CFIFO is configured for continuous-scan mode, no software involvement is necessary to rearm the CFIFO to detect new trigger events after an asserted EOQ is detected. In continuous-scan mode the whole CQueue is scanned multiple times.

The EQADC also supports different triggering mechanisms for each scan mode. The EQADC will not transfer commands from a CFIFO until the CFIFO is triggered. The combination of scan modes and triggering mechanisms allows the support of different requirements for scanning input channels. The scan mode and trigger mechanism are configured by programming the MODEx field in *Section , EQADC CFIFO Control Registers (EQADC_CFCR)*.

Enabled CFIFOs can be triggered by software or external trigger events. The elapsed time from detecting a trigger to transferring a command is a function of clock frequency, trigger synchronization, trigger filtering or not, programmable trigger events, command transfer, CFIFO prioritization, CBuffer availability, etc. Fast and predictable transfers can be achieved by ensuring that the CFIFO is not underflowing and that the target CBuffer is not full when the CFIFO is triggered.

Disabled Mode

The MODE*x* field in *Section , EQADC CFIFO Control Registers (EQADC_CFCR)*, for all of the CFIFOs can be changed from any other mode to disabled at any time. No trigger event can initiate command transfers from an CFIFO which has its MODE field programmed to disabled.

Note: *If MODEx is not disabled, it must not be changed to any other mode besides disabled. If MODEx is disabled and the CFIFO status is IDLE, MODEx can be changed to any other mode.*

If MODE*x* is changed to disabled:

- The CFIFO execution status will change to IDLE. The timing of this change depends on whether a command is being transferred or not:
  - When no command transfer is in progress, the EQADC switches the CFIFO to IDLE status immediately.
  - When a command transfer to an on-chip CBuffer is in progress, the EQADC will complete the transfer, update TC_CF, and switch CFIFO status to IDLE. Command transfers to the internal CBuffers are considered completed when a command is written to the buffers.
  - When a command transfer to an external CBuffer is in progress, the EQADC will abort the transfer and switch CFIFO status to IDLE. If the EQADC cannot abort the transfer, that is when the 26th bit of the serial message has being already shifted out, the EQADC will complete the transfer, update TC_CF and then switch CFIFO status to IDLE.

- The CFIFOs are not invalidated automatically. The CFIFO still can be invalidated by writing a “1” to the CFINVx bit in *Section , EQADC CFIFO Control Registers (EQADC_CFCR)*. Certify that CFS has changed to IDLE before setting CFINVx.
- The TC_CFx value also is not reset automatically, but it can be reset by writing “0” to it.
- The SSS bit in *Section , EQADC FIFO and Interrupt Status Registers (EQADC_FISR)*, is negated. The SSS bit can be set even if a “1” is written to the SSE bit in *Section , EQADC CFIFO Control Registers (EQADC_CFCR)*, in the same write that the MODEx field is changed to a value other than disabled.
- The trigger detection hardware is reset. If MODEx is changed from disabled to an edge trigger mode, a new edge, matching that edge trigger mode, is needed to trigger the command transfers from the CFIFO.

Note: CFIFO fill requests, which generated when CFFF is asserted, are not automatically halted when MODEx is changed to disabled. CFIFO fill requests will still be generated until CFFE is cleared in Section , EQADC Interrupt and DMA Control Registers (EQADC_IDCR).

Single-Scan Mode

In single-scan mode, a single pass through a sequence of command messages in a CQueue is performed.

In single-scan software trigger mode, the CFIFO is triggered by an asserted Single-Scan Status bit (SSS) in *Section , EQADC FIFO and Interrupt Status Registers (EQADC_FISR)*. The SSS bit is set by writing “1” to the Single-Scan Enable bit (SSE) in *Section , EQADC CFIFO Control Registers (EQADC_CFCR)*.

In single-scan edge- or level-trigger mode, the respective triggers are only detected when the SSS bit is asserted. When the SSS bit is negated, all trigger events for that CFIFO are ignored. Writing a “1” to the SSE bit can be done during the same write cycle that the CFIFO operation mode is configured.

Only the EQADC can clear the SSS bit. Once SSS is asserted, it remains asserted until the EQADC completes the CQueue scan, or the CFIFO operation mode (MODEx) in *Section , EQADC CFIFO Control Registers (EQADC_CFCR)*, is changed to disabled. The SSSx bit will be negated while MODE*x* is disabled.

Single-Scan Software Trigger

When single-scan software trigger mode is selected, the CFIFO is triggered by an asserted SSS bit. The SSS bit is asserted by writing “1” to the SSE bit. Writing to SSE while SSS is already asserted will not have any effect on the state of the SSS bit, nor will it cause a trigger overrun event.

The CFIFO commands start to be transferred when the CFIFO becomes the highest priority CFIFO using a not-full on-chip CBuffer or an not-full external CBuffer. When an asserted EOQ bit is encountered, the EQADC will clear the SSS bit. Setting the SSS bit is required for the EQADC to start the next scan of the queue.

The Pause bit has no effect in single-scan software trigger mode.

Single-Scan Edge Trigger

When SSS is asserted and an edge triggered mode is selected for a CFIFO, an appropriate edge on the associated trigger signal causes the CFIFO to become TRIGGERED. For example, if rising-edge trigger mode is selected, the CFIFO becomes TRIGGERED when a rising edge is sensed on the trigger signal. The CFIFO commands start to be transferred when the CFIFO becomes the highest priority CFIFO using a not-full on-chip CBuffer or an not-full external CBuffer.

When an asserted EOQ bit is encountered, the EQADC clears SSS and stops command transfers from the CFIFO. An asserted SSS bit and a subsequent edge trigger event are required to start the next scan for the CFIFO. When an asserted Pause bit is encountered, the EQADC stops command transfers from the CFIFO, but SSS remains set. Another edge trigger event is required for command transfers to continue. A trigger overrun happens when the CFIFO is in TRIGGERED state and an edge trigger event is detected.

Single-Scan Level Trigger

When SSS is asserted and a level gated trigger mode is selected, the input level on the associated trigger signal puts the CFIFO in TRIGGERED state. When the CFIFO is asserted to high-level gated trigger, a high level signal opens the gate, and a low level closes the gate. When the CFIFO is set to low-level gated trigger mode, a low level signal opens the gate, and a high level closes the gate. If the corresponding level is already present, setting the SSS bit triggers the CFIFO. The CFIFO commands start to be transferred when the CFIFO becomes the highest priority CFIFO using a not-full on-chip CBuffer or a not -full external CBuffer.

The EQADC clears the SSS bit and stops transferring commands from a TRIGGERED CFIFO when an asserted EOQ bit is encountered or when CFIFO status changes from TRIGGERED due to the detection of a closed gate. If a closed gate is detected while no command transfers are taking place and the CFIFO status is TRIGGERED, the CFIFO status is immediately changed to IDLE, the SSS bit is negated, and the PF flag is asserted. If a closed gate is detected during the serial transmission of a command to the external device, it will have no effect on the CFIFO status until the transmission completes. Once the transmission is completed, the TC_CF counter is updated, the SSS bit is negated, the PF flag is asserted, and the CFIFO status is changed to IDLE. An asserted SSS bit and a level trigger are required to restart the CFIFO. Command transfers will restart from the point they have stopped.

If the gate closes and opens during the same serial transmission of a command to the external device, it will have no effect on the CFIFO status or on the PF flag, but the TORF flag will become asserted as was exemplified in *Figure 512*. Therefore, closing the gate for a period less than a serial transmission time interval does not guarantee that the closure will affect command transfers from a CFIFO.

The Pause bit has no effect in single-scan level-trigger mode.

Continuous-Scan Mode

In continuous-scan mode, multiple passes looping through a sequence of command messages in a CQueue are executed. When a CFIFO is programmed for a continuous-scan mode, the SSE bit in the *Section , EQADC CFIFO Control Registers (EQADC_CFCR)*, does not have any effect.

Continuous-Scan Software Trigger

When a CFIFO is programmed to continuous-scan software trigger mode, the CFIFO is triggered immediately. The CFIFO commands start to be transferred when the CFIFO becomes the highest priority CFIFO using a not-full on-chip CBuffer or an not-full external CBuffer. When a CFIFO is programmed to run in continuous-scan software trigger mode, the EQADC will not halt transfers from the CFIFO until the CFIFO operation mode is modified to disabled or a higher priority CFIFO preempts it. Although command transfers will not stop upon detection of an asserted EOQ bit, the EOQF is set and, if enabled, an EOQ interrupt request is generated.

The Pause bit has no effect in continuous-scan software trigger mode.

Continuous-Scan Edge Trigger

When rising, falling, or either edge trigger mode is selected for a CFIFO, a corresponding edge on the associated ETRIG signal places the CFIFO in TRIGGERED state. The CFIFO commands start to be transferred when the CFIFO becomes the highest priority CFIFO using a not-full on-chip CBuffer or an not-full external CBuffer

When an EOQ or a Pause is encountered, the EQADC halts command transfers from the CFIFO and, if enabled, the appropriate interrupt requests are generated. Another edge trigger event is required to resume command transfers but no software involvement is required to rearm the CFIFO in order to detect such event.

A trigger overrun happens when the CFIFO is already in TRIGGERED state and a new edge trigger event is detected.

Continuous-Scan Level Trigger

When high or low level gated trigger mode is selected, the input level on the associated trigger signal places the CFIFO in TRIGGERED state. When high-level gated trigger is selected, a high-level signal opens the gate, and a low level closes the gate. The CFIFO commands start to be transferred when the CFIFO becomes the highest priority CFIFO using a not-full on-chip CBuffer or an not-full external CBuffer. Although command transfers will not stop upon detection of an asserted EOQ bit at the end of a command transfer, the EOQF is asserted and, if enabled, an EOQ interrupt request is generated.

The EQADC stops transferring commands from a TRIGGERED CFIFO when CFIFO status changes from TRIGGERED due to the detection of a closed gate. If a closed gate is detected while no command transfers are taking place and the CFIFO status is TRIGGERED, the CFIFO status is immediately changed to WAITING FOR TRIGGER and the PF flag is asserted. If a closed gate is detected during the serial transmission of a command to the external device, it will have no effect on the CFIFO status until the transmission completes. Once the transmission is completed, the TC_CF counter is updated, the PF flag is asserted, and the CFIFO status is changed to WAITING FOR TRIGGER. Command transfers will restart as the gate opens.

If the gate closes and opens during the same serial transmission of a command to the external device, it will have no effect on the CFIFO status or on the PF flag, but the TORF flag will become asserted as was exemplified in *Figure 512*. Therefore, closing the gate for a period less than a serial transmission time interval does not guarantee that the closure will affect command transfers from a CFIFO.

The Pause bit has no effect in continuous-scan level-trigger mode.

CFIFO Scan Trigger Mode Start/Stop Summary

Table 536 summarizes the start and stop conditions of command transfers from CFIFOs for all of the single-scan and continuous-scan trigger modes.

Table 536. CFIFO Scan Trigger Mode - Command Transfer Start/Stop Summary

Trigger Mode	Requires Asserted SSS to Recognize Trigger Events?	Command Transfer Start/Restart Condition	Stop on asserted EOQ bit(1)?	Stop on asserted Pause bit(2)?	Other Command Transfer Stop Condition(3)(4)
Single Scan Software	Don't Care	Asserted SSS bit.	Yes	No	None.
Single Scan Edge	Yes	A corresponding edge occurs when the SSS bit is asserted.	Yes	Yes	None.
Single Scan Level	Yes	Gate is opened when the SSS bit is asserted.	Yes	No	EQADC also stops transfers from the CFIFO when CFIFO status changes from TRIGGERED due to the detection of a closed gate.(5)
Continuous Scan Software	No	CFIFO starts automatically after being configured into this mode.	No	No	None.
Continuous Scan Edge	No	A corresponding edge occurs.	Yes	Yes	None.
Continuous Scan Level	No	Gate is opened.	No	No	EQADC also stops transfers from the CFIFO when CFIFO status changes from TRIGGERED due to the detection of a closed gate.(5)

1. Refer to *Section , CQueue Completion Status*, for more information on EOQ.
2. Refer to *Section , Pause Status*, for more information on Pause.
3. EQADC always stops command transfers from a CFIFO when the CFIFO operation mode is disabled.
4. EQADC always stops command transfers from a CFIFO when a higher priority CFIFO is triggered. Refer to *Section , CFIFO Common Prioritization and Command Transfer*, for information on CFIFO priority.
5. If a closed gate is detected while no command transfers are taking place, it will have immediate effect on the CFIFO status. If a closed gate is detected during the serial transmission of a command to the external device, it will have no effect on the CFIFO status until the transmission completes.

CFIFO and Trigger Status

CFIFO Operation Status

Each CFIFOs has its own CFIFO status field. CFIFO status (CFS) can be read from *Section , EQADC CFIFO Status Register (EQADC_CFSR)*. *Figure 511* and *Table 537* indicate the CFIFO status switching condition. Refer to *Table 494* for the meaning of each CFIFO operation status. The last CFIFO to transfer a command to an on-chip CBuffer can be read from the LCFTCB*n* (n=0,1) fields in the *Section , EQADC CFIFO Status Snapshot Registers (EQADC_CFSSR)*. The last CFIFO to transfer a command to a specific external CBuffer can be identified by reading the LCFTSSI and ECBNI fields in the *Section , EQADC CFIFO Status Snapshot Registers (EQADC_CFSSR)*.

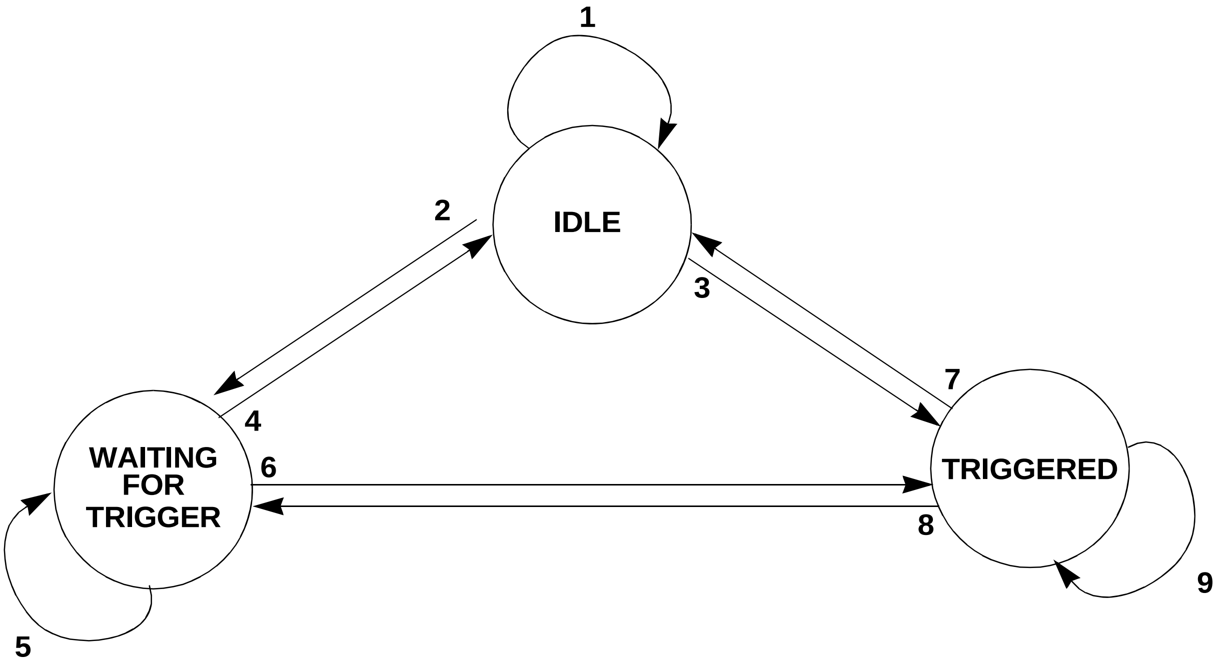


Figure 511. State Machine of CFIFO Status

Table 537. Command FIFO Status Switching Condition

No.	From Current CFIFO Status (CFS)	To New CFIFO Status (CFS)	Status Switching Condition
1	IDLE (00)	IDLE (0b00)	— CFIFO Mode is programmed to disabled, OR — CFIFO Mode is programmed to single-scan edge or level trigger mode and SSS is negated.
2		WAITING FOR TRIGGER (0b10)	— CFIFO Mode is programmed to continuous-scan edge or level trigger mode, OR — CFIFO Mode is programmed to single-scan edge or level trigger mode and SSS is asserted, OR — CFIFO Mode is programmed to single-scan software trigger mode.
3		TRIGGERED (0b11)	— CFIFO Mode is programmed to continuous-scan software trigger mode
4	WAITING FOR TRIGGER (10)	IDLE (0b00)	— CFIFO Mode is modified to disabled mode.
5		WAITING FOR TRIGGER (0b10)	— No trigger occurred.
6		TRIGGERED (0b11)	— Appropriate edge or level trigger occurred, OR — CFIFO Mode is programmed to single-scan software trigger mode and SSS bit is asserted.

Table 537. Command FIFO Status Switching Condition (continued)

No.	From Current CFIFO Status (CFS)	To New CFIFO Status (CFS)	Status Switching Condition
7	TRIGGERED (11)	IDLE (0b00)	— CFIFO in single-scan mode, EQADC detects the EOQ bit asserted at end of command transfer, and CFIFO Mode is not modified to disabled.OR — CFIFO, in single-scan level trigger mode, and the gate closes while no commands are being transferred from the CFIFO, and CFIFO Mode is not modified to disabled. OR — CFIFO, in single-scan level trigger mode, and EQADC detects a closed gated at end of command transfer, and CFIFO Mode is not modified to disabled. OR — CFIFO Mode is modified to disabled mode and CFIFO was not transferring commands. —CFIFO Mode is modified to disabled mode while CFIFO was transferring commands, and CFIFO completes or aborts the transfer.
8		WAITING FOR TRIGGER (0b10)	— CFIFO in single or continuous-scan edge trigger mode, EQADC detects the Pause bit asserted at the end of command transfer, the EOQ bit in the same command is negated, and CFIFO Mode is not modified to disabled, OR — CFIFO in continuous-scan edge trigger mode, EQADC detects the EOQ bit asserted at the end of command transfer, and CFIFO Mode is not modified to disabled, OR — CFIFO, in continuous-scan level trigger mode, and the gate closes while no commands are being transferred from the CFIFO, and CFIFO Mode is not modified to disabled, OR — CFIFO, in continuous-scan level trigger mode, and EQADC detects a closed gated at end of command transfer, and CFIFO Mode is not modified to disabled.
9		TRIGGERED (0b11)	— No event to switch to IDLE or WAITING FOR TRIGGER status has happened.

CQueue Completion Status

The End of Queue Flag (EOQF) in *Section , EQADC FIFO and Interrupt Status Registers (EQADC_FISR)*, is asserted when the EQADC completes the transfer of a CFIFO entry with an asserted EOQ bit. Software sets the EOQ bit in the last Command Message of a CQueue to indicate that this entry is the end of the CQueue - see *Section , Message Format in EQADC*, for information on command message formats. The transfer of entries bound for the on-chip ADCs is considered completed when they are stored in the appropriate CBuffer. The transfer of entries bound for the external device is considered completed when the serial transmission of the entry is completed.

The command with a EOQ bit asserted is valid and will be transferred. When EOQIE in *Section , EQADC CFIFO Control Registers (EQADC_CFCR)*, and EOQF are asserted, the EQADC will generate an End of Queue interrupt request.

In single-scan modes, command transfers from the corresponding CFIFO will cease when EQADC completes the transfer of a entry with an asserted EOQ. Software involvement is required to rearm the CFIFO so that it can detect new trigger events.

Note: An asserted EOQFx only implies that EQADC has finished transferring a command with an asserted EOQ bit from CFIFOx. It does not imply that result data for the current command and for all previously transferred commands has been returned to the appropriate RFIFO.

Pause Status

In edge trigger mode, when the EQADC completes the transfer of a CFIFO entry with an asserted Pause bit, the EQADC will stop future command transfers from the CFIFO and set the corresponding Pause Flag (PF) in *Section , EQADC FIFO and Interrupt Status Registers (EQADC_FISR)*. Refer to *Section , Message Format in EQADC*, for information on command message formats. The EQADC ignores the Pause bit in command messages in any software and external level trigger mode. The EQADC sets the PF flag upon detection of an asserted Pause bit only in single or continuous-scan edge trigger mode. When the PF flag is set for a CFIFO in single-scan edge trigger mode, the SSS bit will not be cleared in *Section , EQADC FIFO and Interrupt Status Registers (EQADC_FISR)*.

In level trigger mode, the definition of the PF flag has been redefined. In level trigger mode, when CFIFOx is in TRIGGERED status, PFx is set when CFIFO status changes from TRIGGERED due to detection of a closed gate. The pause flag interrupt routine can be used to verify if the a complete scan of the CQueue was performed. If a closed gate is detected while no command transfers are taking place, it will have immediate effect on the CFIFO status. If a closed gate is detected during the serial transmission of a command to the external device, it will have no effect on the CFIFO status until the transmission completes.

When PIE in *Section , EQADC CFIFO Control Registers (EQADC_CFCR)*, and PF are asserted, the EQADC will generate a Pause interrupt request.

Note: In edge trigger mode, an asserted PFx only implies that the EQADC finished transferring a command with an asserted PAUSE bit from CFIFOx. It does not imply that result data for the current command and for all previously transferred commands has been returned to the appropriate RFIFO.

Note: In software or level trigger mode, when the EQADC completes the transfer of an entry from CFIFOx with an asserted Pause bit, PFx will not be set and command transfers will continues without pausing.

Trigger Overrun Status

When a CFIFO is configured for edge- or level-trigger mode and is in TRIGGERED state, an additional trigger occurring for the same CFIFO results in a trigger overrun. The trigger overrun bit for the corresponding CFIFO will be set (TORFx = 1) in *Section , EQADC FIFO and Interrupt Status Registers (EQADC_FISR)*. When TORIE in *Section , EQADC CFIFO Control Registers (EQADC_CFCR)*, and TORF are asserted, the EQADC generates a trigger overrun interrupt request.

For CFIFOs configured for level-trigger mode, a trigger overrun event is only detected when the gate closes and opens during a single serial command transmission as shown in *Figure 512*.

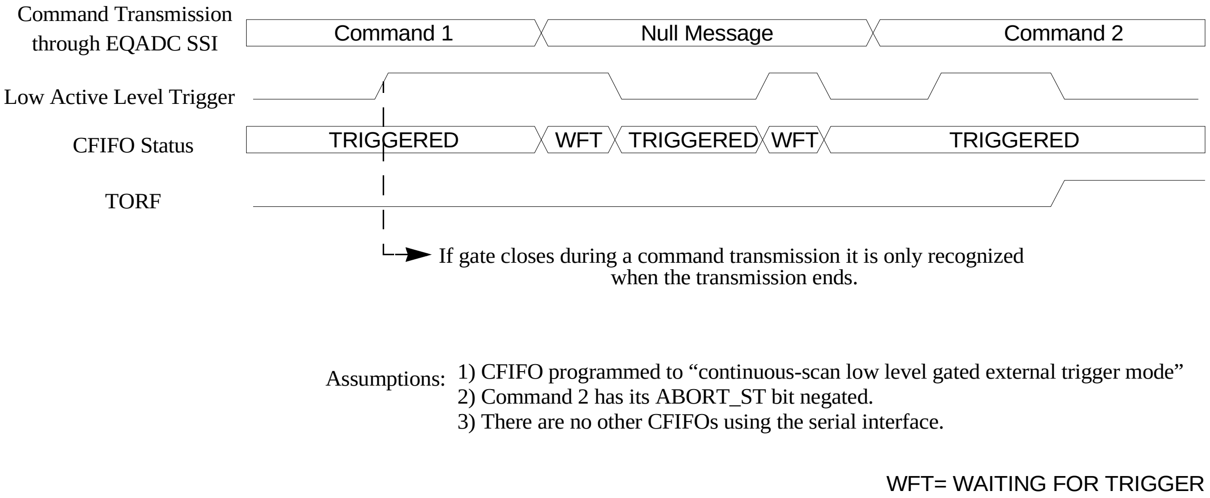


Figure 512. Trigger Overrun on Level-Trigger Mode CFIFOs

Note: The trigger overrun flag will not set for CFIFOs configured for software trigger mode.

Command Sequence Non-Coherency Detection

The EQADC provides a mechanism to indicate if a command sequence has been completely executed without interruptions. A command sequence is defined as a group of consecutive commands bound for the same CBuffer and it is expected to be executed without interruptions. A command sequence is coherent if its commands are executed in order without interruptions. Since commands are stored in the CBuffers before being executed in the EQADC, a command sequence is coherent if, while it is transferring commands to a CBuffer, the buffer is only fed with commands from that sequence without ever becoming empty.

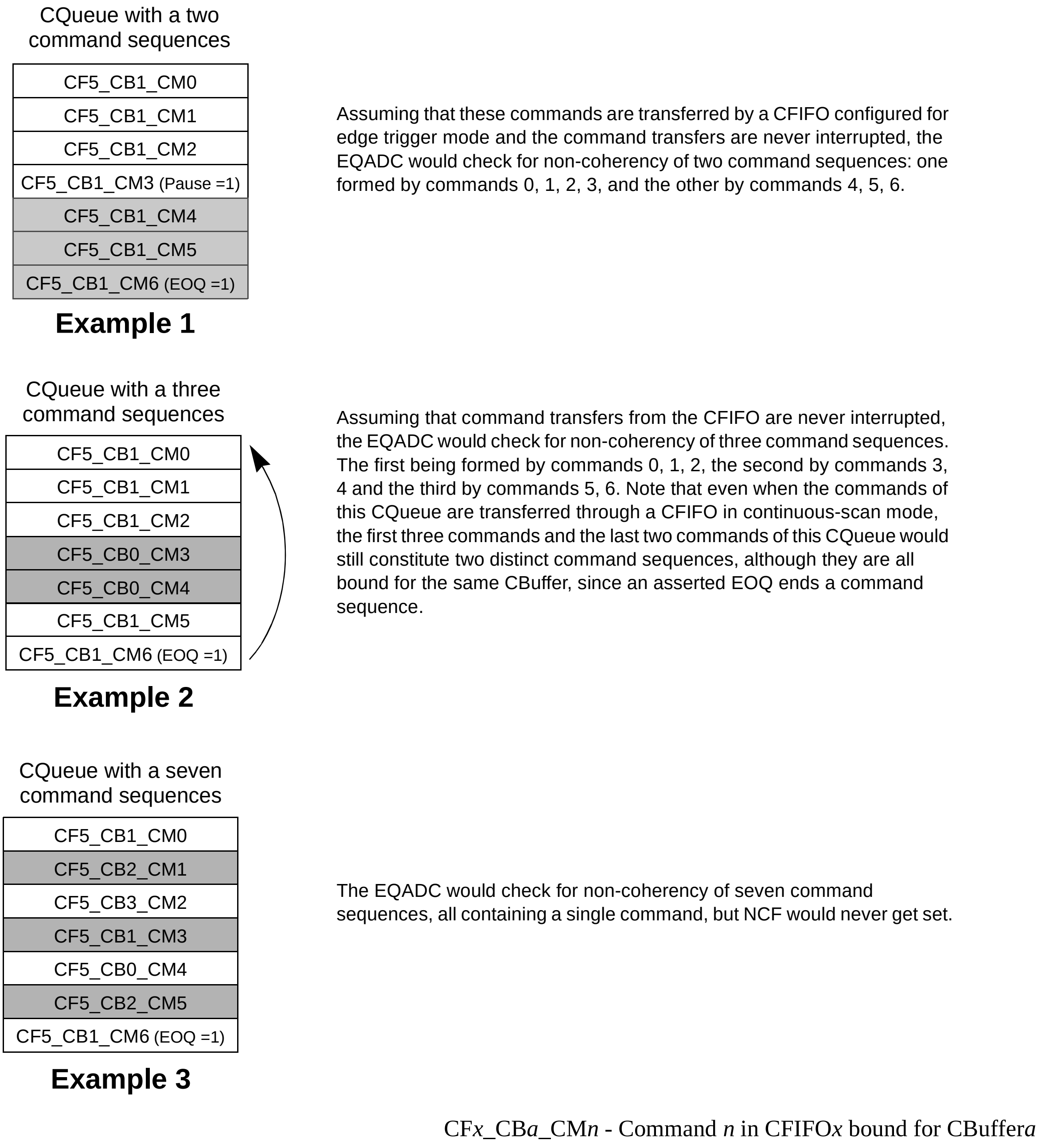
A command sequence starts when:

- a CFIFO in TRIGGERED state transfers its first command to CBuffer.
- the CFIFO is constantly transferring commands and the previous command sequence ended.
- the CFIFO resumes command transfers after being interrupted.

And a command sequence ends when:

- an asserted EOQ bit is detected on the last transferred command.
- CFIFO is in edge-trigger mode and asserted PAUSE bit is detected on the last transferred command.
- the CBuffer to which the next command is bound is different from the one to which the last command was transferred.

Figure 513 shows examples of how the EQADC would detect command sequences when transferring commands from a CFIFO to a CBuffer. The smallest possible command sequence can have a single command as shown in example 3 of *Figure 513*.



Assuming that these commands are transferred by a CFIFO configured for edge trigger mode and the command transfers are never interrupted, the EQADC would check for non-coherency of two command sequences: one formed by commands 0, 1, 2, 3, and the other by commands 4, 5, 6.

Assuming that command transfers from the CFIFO are never interrupted, the EQADC would check for non-coherency of three command sequences. The first being formed by commands 0, 1, 2, the second by commands 3, 4 and the third by commands 5, 6. Note that even when the commands of this CQueue are transferred through a CFIFO in continuous-scan mode, the first three commands and the last two commands of this CQueue would still constitute two distinct command sequences, although they are all bound for the same CBuffer, since an asserted EOQ ends a command sequence.

The EQADC would check for non-coherency of seven command sequences, all containing a single command, but NCF would never get set.

CF*x*_CB*a*_CM*n* - Command *n* in CFIFO*x* bound for CBuffer*a*

Figure 513. Command Sequence Examples

The NCF flag is used to indicate command sequence non-coherency. When the NCFx flag is asserted, it indicates that the command sequence being transferred through CFIFOx became non-coherent. The NCF flag only becomes asserted for CFIFOs in TRIGGERED state.

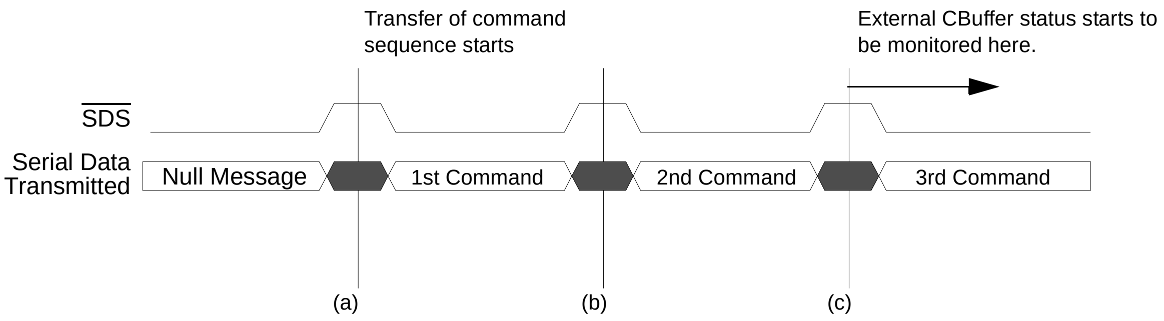
A command sequence is non-coherent when, after transferring the first command of a sequence from a CFIFO to a CBuffer, it cannot successively send all the other commands of the sequence before any of the following conditions are true:

- The CFIFO through which commands are being transferred is preempted by a higher priority CFIFO which sends commands to the same CBuffer. The NCF flag becomes asserted immediately after the first command transfer from the preempting CFIFO, that is the higher priority CFIFO, to the CBuffer in use is completed. See *Figure 515*.
- The external CBuffer in use becomes empty(av). This case happens when different CFIFOs attempt to use different external CBuffers and the higher priority CFIFO bars the lower priority one from sending new commands to its CBuffer - see *Figure 516*. An external CBuffer is considered empty when the corresponding BUSY field in the last result message received from external device is encoded as “Send available commands - CBuffer is empty”. Refer to *Section , Result Message Format for External Device Operation*. The NCF flag becomes asserted immediately after the EQADC detects that the external CBuffer in use becomes empty.

Note: *After the transfer of a command sequence to an external CBuffer starts, the EQADC ignores, for non-coherency detection purposes, the BUSY fields captured at the end of the first serial transmission. Thereafter, all BUSY fields captured at the end of consecutive serial transmissions are used to check the fullness of that external CBuffer. This is done because the EQADC only updates its external CBuffers status record when it receives a serial message, resulting that the record kept by the EQADC is always outdated by, at least, the length of one serial transmission. This prevents a CFIFO from immediately becoming non-coherent when it starts transferring commands to an empty external CBuffer. Refer to Figure 514 for an example.*

---

av. Only the fullness of external CBuffers is monitored because the fill rate for internal CBuffers is many times faster than the drain rate, and each has a dedicated priority engine.



External CBuffer Status

Capture Point at EQADC	CBuffer Status at External Device	CBuffer Status as Captured by the EQADC	Used for NCF detection on the EQADC?
(a)	EMPTY	EMPTY	Don’t care
(b)	1 ENTRY	EMPTY	No
(c)	2 ENTRY	1 ENTRY	Yes

Assumptions:

1. The CFIFO starts sending commands to an empty external CBuffer when triggered.
2. Execution of a command on the external device takes longer than the time to complete three serial transmissions.

Figure 514. External CBuffer Status Detection at Command Sequence Transfer Start

Once a command sequence starts to be transferred, the EQADC will check for the command sequence coherency until the command sequence ends or until one of the conditions below becomes true.

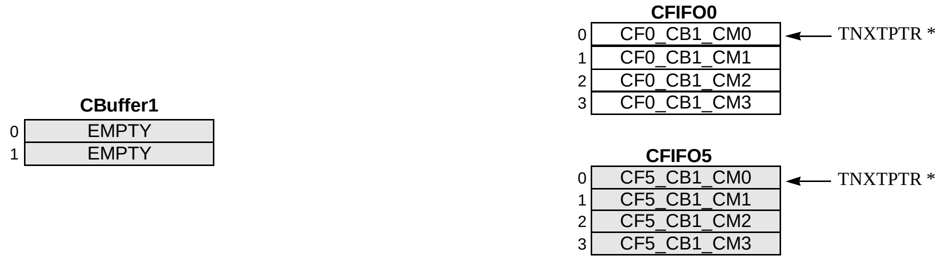
- The command sequence became non-coherent.
- The CFIFO status changed from TRIGGERED.
- The CFIFO underflowed.

Note: The NCF flag still becomes asserted if an external CBuffer empty event is detected at the same time the EQADC stops checking for the coherency of a command sequence.

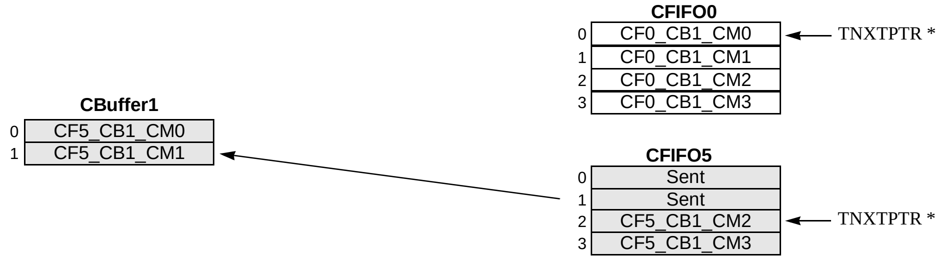
Once command transfers restart/continue, the non-coherency hardware will behave as if the command sequence started from that point. *Figure 517* depicts how the non-coherency hardware will behave when a non-coherency event is detected.

Note: If MODEx is changed to disabled while a CFIFO is transferring commands, the NCF flag for that CFIFO will not become asserted.

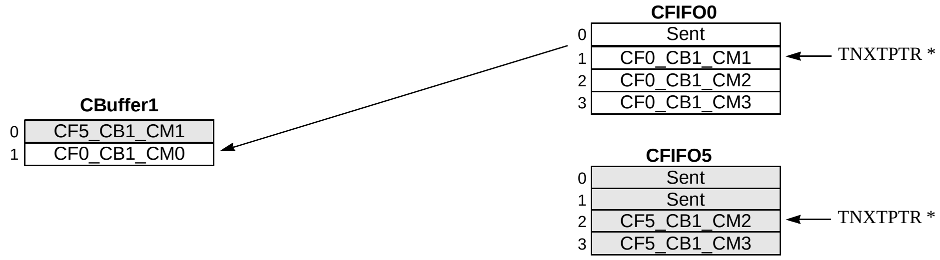
Note: When the EQADC enters debug or stop mode while a command sequence is being executed, the NCF will become asserted if an empty external CBuffer is detected after debug/stop mode is exited.



(a) CFIFO0 and CFIFO5 both have commands to be sent to CBuffer1, and both are not triggered



(b) CFIFO5 becomes triggered and transfers two commands to CBuffer1

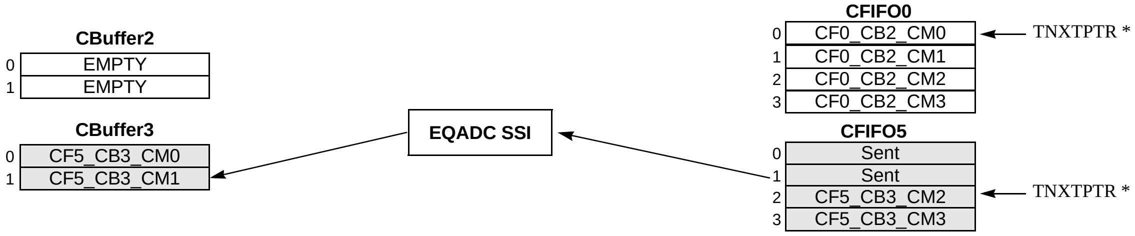


(c) CFIFO0 becomes triggered and transfers a command to CBuffer1. The sequence sent through CFIFO5 becomes non-coherent.

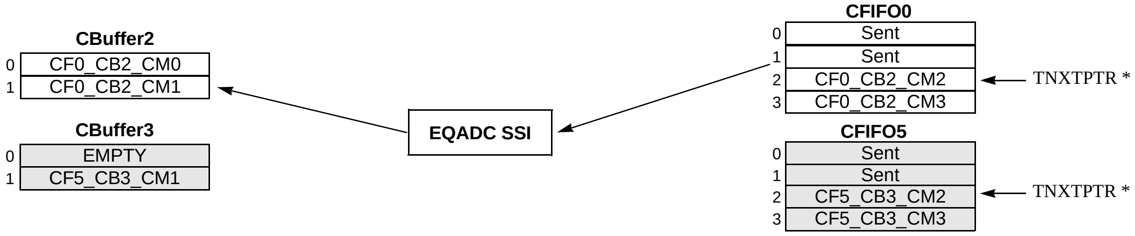
* TNXTPTR - Transfer Next Data Pointer
CF*x*_CB*a*_CM*n* - Command *n* in CFIFO*x* bound for CBu

Figure 515. Non-Coherency Event when Different CFIFOs use the same CBuffer

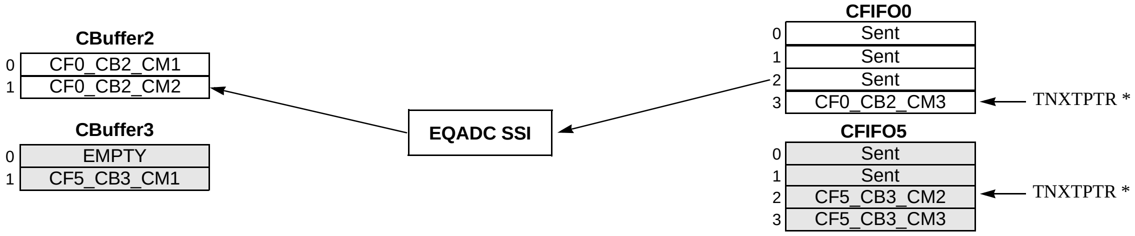
Figure 516. Non-Coherency Event when Different CFIFOs are using Different External CBuffers



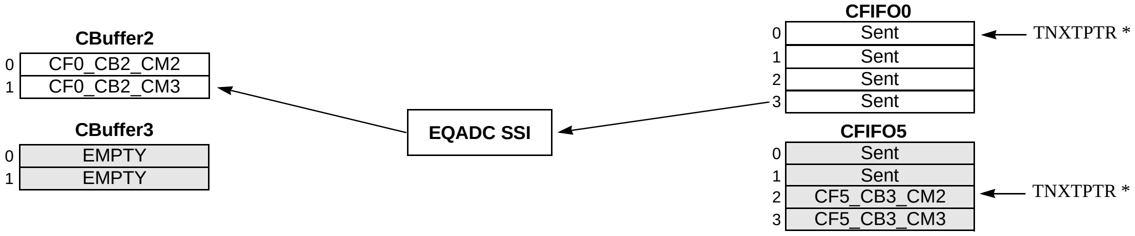
(a) CFIFO0 and CFIFO5 both have commands to be sent to external CBuffers. CFIFO0 is not triggered. CFIFO5 is triggered and sent two commands to CBuffer3



(b) CFIFO0 is triggered and sent two commands to CBuffer2. CFIFO5 cannot send commands to CBuffer3 because the EQADC SSI is busy transferring commands from CFIFO0. Execution of first command of CFIFO5 is completed.

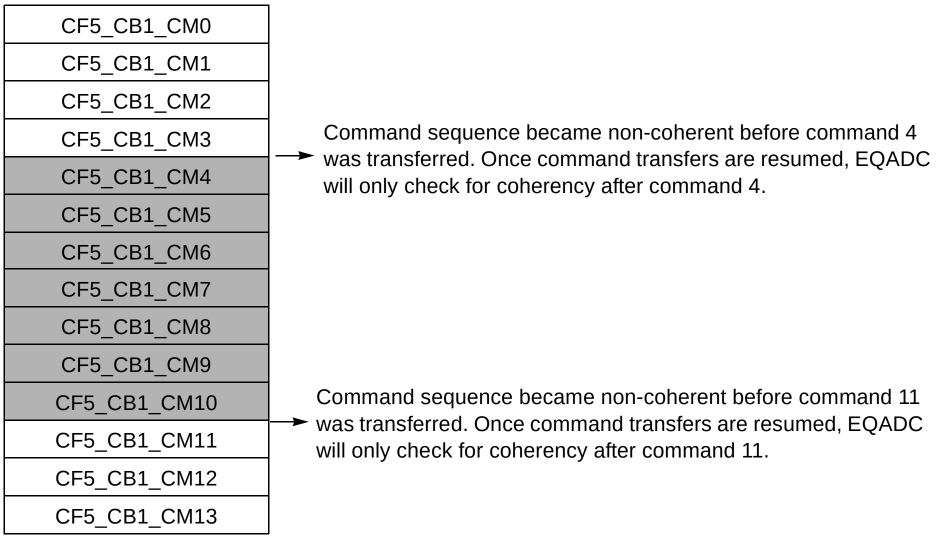


(c) Execution of first command of CFIFO0 is completed and CFIFO0 sends new command to CBuffer2.



(d) Second command in CBuffer3 completes. CBuffer3 became empty before the complete command sequence in CFIFO5 is sent to it. NCF5 becomes asserted when the EQADC receives an indication that CBuffer3 is empty, by the BUSY fields in the returning serial message.

* TNXTPTR - Transfer Next Data Pointer
CF*x*_CB*a*_CM*n* - Command *n* in CFIFO*x* bound for CBuffer*a*



Command sequence became non-coherent before command 4 was transferred. Once command transfers are resumed, EQADC will only check for coherency after command 4.

Command sequence became non-coherent before command 11 was transferred. Once command transfers are resumed, EQADC will only check for coherency after command 11.

Figure 517. Non-coherency Detection when Transfers from a Command Sequence are Interrupted

23.6.5 EQADC Result FIFOs

RFIFO Basic Functionality

There are six RFIFOs located in the EQADC. Each RFIFO is four entries deep, and each RFIFO entry is 16 bits long. Each RFIFO serves as a temporary storage location for the one of the RQueues allocated in system memory. Result data is saved in the RFIFOs before being moved into the system RQueues. When an RFIFO is not empty, the EQADC sets the corresponding RFDF bit in *Section , EQADC FIFO and Interrupt Status Registers (EQADC_FISR)*. If RFDE is asserted in *Section , EQADC Interrupt and DMA Control Registers (EQADC_IDCR)*, the EQADC generates a request so that an RFIFO entry is moved to the RQueue. An interrupt request, served by the host CPU, is generated when RFDS is negated, and a DMA request, served by the DMAC, is generated when RFDS is asserted. The host CPU or the DMAC responds to these requests by reading *Section , EQADC Result FIFO Pop Registers (EQADC_RFPR)*, to retrieve data from the RFIFO.

Note: The DMAC should be configured to read a single result (16-bit data) from the RFIFO pop registers for every asserted DMA request it acknowledges. Refer to Section 23.7.2, EQADC/DMAC Interface, for DMAC configuration guidelines.

Note: Reading a word, a half-word, or any bytes from EQADC_RFPRx will pop an entry from RFIFOx, and the RFCTRx field will be decremented by one.

Figure 518 describes the important components in the RFIFO. Each RFIFO is implemented as a circular set of registers to avoid the need to move all entries at each push/pop operation. The Pop Next Data Pointer always points to the next RFIFO message to be retrieved from the RFIFO when reading EQADC_RFPR. The Receive Next Data Pointer points to the next available RFIFO location for storing the next incoming message from the on-chip ADCs or from the external device. The *RFIFO Counter Logic* counts the number of entries in RFIFO and generates interrupt or DMA requests to drain the RFIFO.

POPNXTPTR in *Section , EQADC FIFO and Interrupt Status Registers (EQADC_FISR)*, indicates which entry is currently being addressed by the Pop Next Data Pointer, and RFCTR, in the same register, provides the number of entries stored in the RFIFO. Using POPNXTPTR and RFCTR, the absolute addresses for Pop Next Data Pointer and Receive Next Data Pointer can be calculated using the following formulas:

Pop Next Data Pointer Address= RFIFOx_BASE_ADDRESS + POPNXTPTRx*4

Receive Next Data Pointer Address = RFIFOx_BASE_ADDRESS + [(POPNXTPTRx+RFCTRx) mod RFIFO_DEPTH] * 4

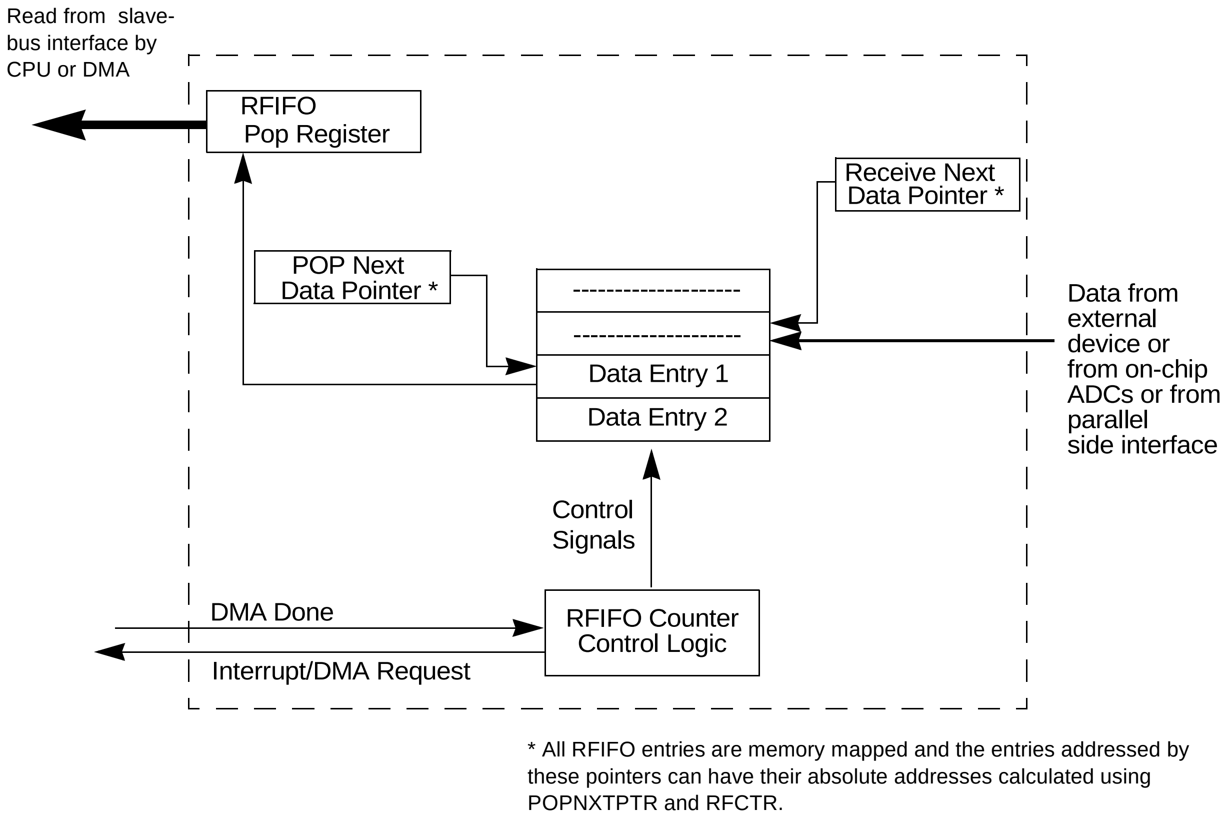
where

- *a mod b* returns the remainder of the division of *a* by *b*.
- RFIFOx_BASE_ADDRESS is the smallest memory mapped address allocated to an RFIFOx entry.
- RFIFO_DEPTH is the number of entries contained in a RFIFO - four in this implementation.

When a new message arrives and RFIFOx is not full, the EQADC copies its contents into the entry pointed by the Receive Next Data Pointer. The RFIFO counter RFCTRx in *Section , EQADC FIFO and Interrupt Status Registers (EQADC_FISR)*, is incremented by one, and the Receive Next Data Pointer x is also incremented by one (or wrapped around) to point to the next empty entry in RFIFOx. However, if the RFIFOx is full, the EQADC sets the RFOF in *Section , EQADC FIFO and Interrupt Status Registers (EQADC_FISR)*. The RFIFOx will not overwrite the older data in the RFIFO, the new data will be ignored, and the Receive Next Data Pointer x is not incremented or wrapped around. RFIFOx is full when the Receive Next Data Pointer x equals the Pop Next Data Pointer x and RFCTRx is not zero. RFIFOx is empty when the Receive Next Data Pointer x equals the Pop Next Data Pointer x and RFCTRx is zero.

When the EQADC RFIFO Pop Register x is read and the RFIFOx is not empty, the RFIFO counter RFCTRx is decremented by one, and the POP Next Data Pointer is incremented by one (or wrapped around) to point to the next RFIFO entry.

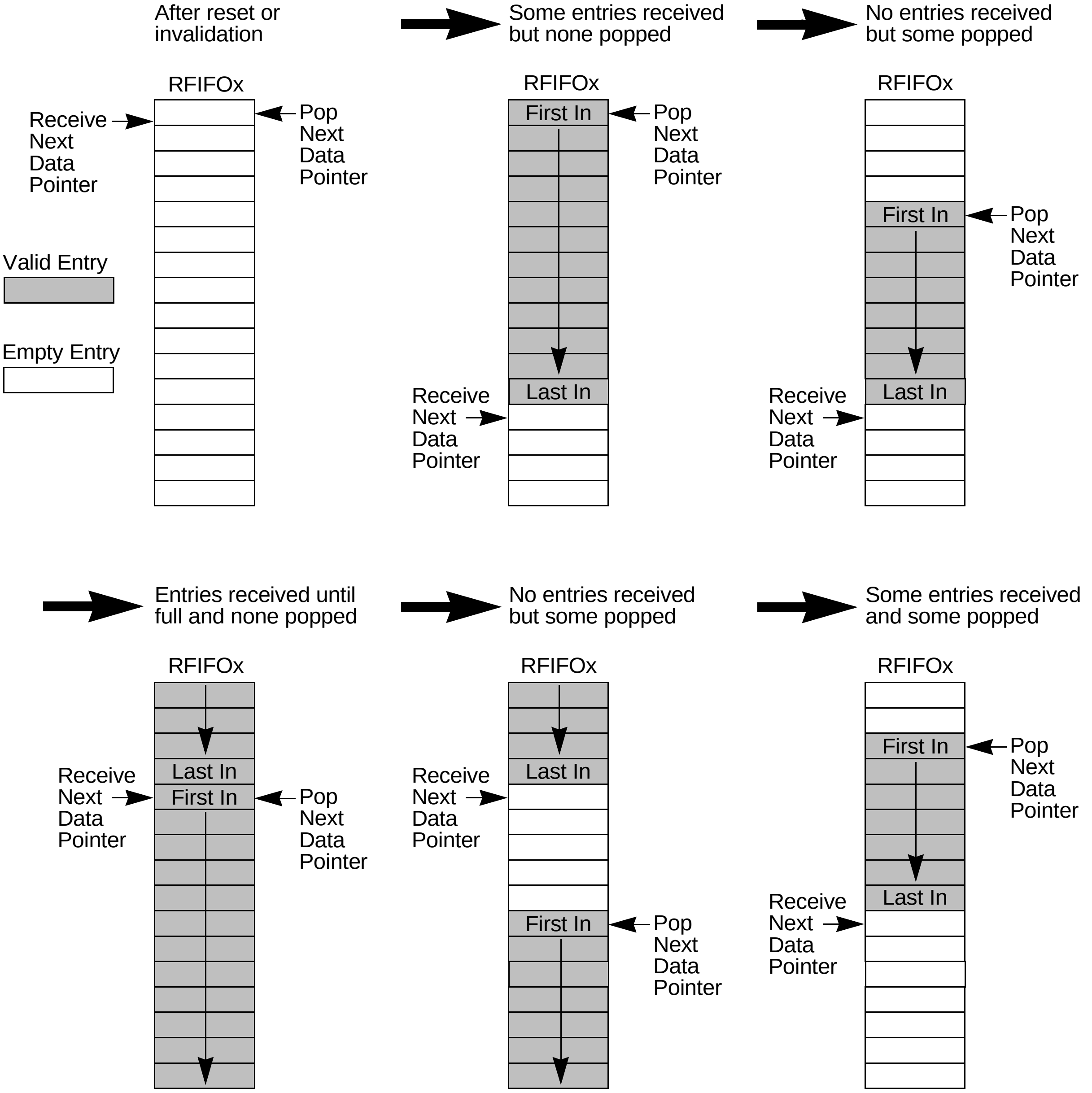
When the EQADC RFIFO Pop Register x is read and RFIFOx is empty, EQADC will not decrement the counter value and the POP Next Data Pointer x will not be updated. The read value will be undefined.



* All RFIFO entries are memory mapped and the entries addressed by these pointers can have their absolute addresses calculated using POPNXTPTR and RFCTR.

Figure 518. RFIFO Diagram

The detailed behavior of the Pop Next Data Pointer and Receive Next Data Pointer is described in the example shown in *Figure 519* where an RFIFO with 16 entries is shown for clarity of explanation, the actual hardware implementation has only four entries. In this example, RFIFOx with 16 entries is shown in sequence after popping or receiving entries.



NOTE: x=0, 1, 2, 3, 4, 5

Figure 519. RFIFO Entry Pointer Example

Distributing Result Data into RFIFOs

Data to be moved into the RFIFOs can come from four sources: from ADC0, from ADC1, from the external device or from the decimation filter A or B, or reaction module through the PSI. All result data comes with a MESSAGE_TAG field and a DEST field defining what should be done with the received data. The EQADC hardware decodes the MESSAGE_TAG and DEST fields and:

- stores the 16-bit data into the appropriate RFIFO if the MESSAGE_TAG indicates a valid RFIFO number, or;
- sends the 16-bit data, the MESSAGE_TAG and the DEST data through the PSI to decimation filter A or B or reaction module, or;
- ignores the data in case of a null or “reserved for customer use” MESSAGE_TAG.

In general received data is moved into RFIFOs as they become available, while an exception happens when multiple results from different sources become available at the same time. In that case, result data from ADC0 is processed first, result data from ADC1 is only process after all ADC0 data is processed, result data from the external device is only processed after all data from ADC0/1 is processed, and finally returned data from companion module is only processed after all data from ADC0/1 and external device is processed.

When time-stamped results return from the on-chip ADCs, the conversion result and the time stamp are always moved to the RFIFOs in consecutive clock cycles in order to guarantee they are always stored in consecutive RFIFO entries.

23.6.6 On-Chip ADC Configuration and Control

Enabling and Disabling the On-chip ADCs

The on-chip ADCs have an enable bit (ADC0/1_EN) in the *Section , ADC0/1 Control Registers (ADC0_CR and ADC1_CR)*, which allows the enabling of the ADCs only when necessary. When the enable bit for an ADC is negated, the clock input to that ADC is stopped. The ADCs are disabled out of reset - ADC0/1_EN bits are negated - to allow for their safe configuration. The ADC must only be configured when its enable bit is negated. Once the enable bit of an ADC is asserted, clock input to is started.

Note: Conversion commands sent to the CBuffer of a disabled ADC are ignored by the ADC control hardware.

Note: A 8ms wait time from VDDA power up to enabling ADC is required to pre-charge the external 100nf capacitor on REFBYPC pin. This time must be guaranteed by crystal startup time plus reset duration or user.

Note: Due to legacy reasons, the EQADC will always wait 120 ADC clocks before issuing the first conversion command following the enabling of one of on-chip ADCs, or the exiting of stop mode. There are two independent counters checking for this delay: one clocked by ADC0_CLK and another by ADC1_CLK. Conversion commands can start to be executed whenever one of these counters completes counting 120 ADC clocks.

ADC Clock and Conversion Speed

The clock input to the ADCs is defined by setting the ADC0/1_ODD_PS, the ADC0/1_CLK_SEL and the ADC0/1_CLK_PS fields in the *Section , ADC0/1 Control Registers (ADC0_CR and ADC1_CR)*. When the ADC0/1_CLK_SEL is set, the ADC clock frequency is the same as the system clock, but it has the inverted phase. When it is clear, the ADC0/1_ODD_PS and the ADC0/1_CLK_PS fields select the clock divide factor by which the system clock will be divided as showed in *Table 508*. The ADC clock frequency is calculated as below and it must not exceed 15 MHz. This is also the maximum frequency of system clock when the ADC0/1_CLK_SEL is asserted.

$$\mathrm{ADCClockFrequency} = \frac{\mathrm{SystemClockFrequency(MHz)}}{\mathrm{SystemClockDivideFactor}};(\mathrm{ADCClockFrequency} \le 15\mathrm{MHz})$$

Figure 520 depicts how the ADC clocks for ADC0 and ADC1 are generated.

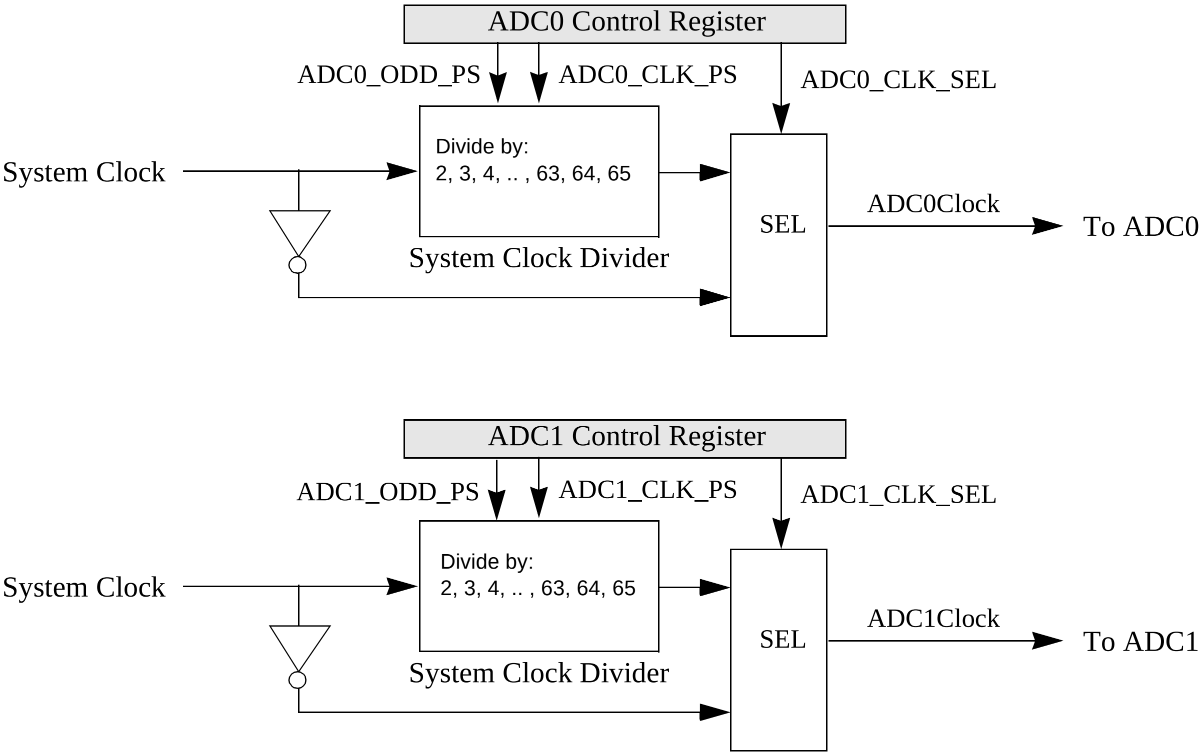


Figure 520. ADC0/1 Clock Generation

The ADC conversion speed (in K samples per second - Ksps) is calculated by the following formula. T*he number of sampling cycles* is determined by the LST bits in the command message - see *Section , Conversion Command Format for the Standard Configuration* - and it can take one of the following values: 2, 8, 64, or 128 ADC clock cycles. The *number of AD conversion cycles* is 13 for differential conversions and 14 for single-ended 12-bit resolution and unitary input gain. The maximum conversion speed is achieved when the ADC Clock frequency is set to its maximum (15 MHz) and the number of sampling cycles set to its minimum (2 cycles). The maximum conversion speed for differential and single-ended conversions are 1Msps and 937.5Ksps, respectively.

$$\text{ADCConversionSpeed} = \frac{\text{ADCClockFrequency(MHz)}}{(\text{NumberOfSamplingCycles} + \text{NumberOfADConversionCycles})}$$

Table 538 shows an example of how the ADC0/1_CLK_PS can be set when using a 120 MHz system clock and the corresponding conversion speeds for all possible ADC clock frequencies. The table also shows that according to the system clock frequency, certain clock divide factors are invalid (2, 4, 6, 8 clock divide factors in the example) since their use would result in a ADC clock frequency higher than the maximum one supported by the ADC. ADC clock frequency must not exceed 15 MHz.

Table 538. ADC Clock Configuration Example (System Clock Frequency=120 MHz)

ADC0/1_CLK_PS [0:4]	ADC0/1_ ODD_PS	System Clock Divide Factor	ADC Clock (System Clock = 120 MHz)	Differential Conversion Speed with Default Sampling Time (2 cycles)	Single-Ended Conversion Speed with Default Sampling Time (2 cycles)
0b00000	0	2	N/A	N/A	N/A
	1	3	N/A	N/A	N/A
0b00001	0	4	N/A	N/A	N/A
	1	5	N/A	N/A	N/A
0b00010	0	6	N/A	N/A	N/A
	1	7	N/A	N/A	N/A
0b00011	0	8	15.0 MHz	1.0 Msps	938 Ksps
	1	9	13.3 MHz	889 Ksps	833 Ksps
0b00100	0	10	12.0 MHz	800 Ksps	750 Ksps
	1	11	10.9 MHz	727 Ksps	682 Ksps
0b00101	0	12	10.0 MHz	667 Ksps	625 Ksps
	1	13	9.23 MHz	615 Ksps	577 Ksps
0b00110	0	14	8.57 MHz	571 Ksps	536 Ksps
	1	15	8.0 MHz	533 Ksps	500 Ksps
0b00111	0	16	7.5 MHz	500 Ksps	469 Ksps
	1	17	7.06 MHz	471 Ksps	441 Ksps
0b01000	0	18	6.67 MHz	444 Ksps	417 Ksps
	1	19	6.32 MHz	421 Ksps	395 Ksps
0b01001	0	20	6.0 MHz	400 Ksps	375 Ksps
	1	21	5.71 MHz	381 Ksps	357 Ksps
0b01010	0	22	5.45 MHz	364 Ksps	341 Ksps
	1	23	5.22 MHz	348 Ksps	326 Ksps
0b01011	0	24	5.0 MHz	333 Ksps	313 Ksps
	1	25	4.80 MHz	320 Ksps	300 Ksps

Table 538. ADC Clock Configuration Example (System Clock Frequency=120 MHz) (continued)

ADC0/1_CLK_PS [0:4]	ADC0/1_ ODD_PS	System Clock Divide Factor	ADC Clock (System Clock = 120 MHz)	Differential Conversion Speed with Default Sampling Time (2 cycles)	Single-Ended Conversion Speed with Default Sampling Time (2 cycles)
0b01100	0	26	4.62 MHz	308 Ksps	288 Ksps
	1	27	4.44 MHz	296 Ksps	278 Ksps
0b01101	0	28	4.29 MHz	286 Ksps	268 Ksps
	1	29	4.14 MHz	276 Ksps	259 Ksps
0b01110	0	30	4.0 MHz	267 Ksps	250 Ksps
	1	31	3.87 MHz	258 Ksps	242 Ksps
0b01111	0	32	3.75 MHz	250 Ksps	234 Ksps
	1	33	3.64 MHz	242 Ksps	227 Ksps
0b10000	0	34	3.53 MHz	235 Ksps	221 Ksps
	1	35	3.43 MHz	229 Ksps	214 Ksps
0b10001	0	36	3.33 MHz	222 Ksps	208 Ksps
	1	37	3.24 MHz	216 Ksps	203 Ksps
0b10010	0	38	3.16 MHz	211 Ksps	198 Ksps
	1	39	3.08 MHz	205 Ksps	192 Ksps
0b10011	0	40	3.0 MHz	200 Ksps	188 Ksps
	1	41	2.93 MHz	195 Ksps	183 Ksps
0b10100	0	42	2.86 MHz	190 Ksps	179 Ksps
	1	43	2.79 MHz	186 Ksps	174 Ksps
0b10101	0	44	2.73 MHz	182 Ksps	170 Ksps
	1	45	2.67 MHz	178 Ksps	167 Ksps
0b10110	0	46	2.61 MHz	174 Ksps	163 Ksps
	1	47	2.55 MHz	170 Ksps	160 Ksps
0b10111	0	48	2.5 MHz	167 Ksps	156 Ksps
	1	49	2.45 MHz	163 Ksps	153 Ksps
0b11000	0	50	2.4 MHz	160 Ksps	150 Ksps
	1	51	2.35 MHz	157 Ksps	147 Ksps
0b11001	0	52	2.31 MHz	154 Ksps	144 Ksps
	1	53	2.26 MHz	151 Ksps	142 Ksps
0b11010	0	54	2.22 MHz	148 Ksps	139 Ksps
	1	55	2.18 MHz	145 Ksps	136 Ksps
0b11011	0	56	2.14 MHz	143 Ksps	134 Ksps
	1	57	2.11 MHz	140 Ksps	132 Ksps

Table 538. ADC Clock Configuration Example (System Clock Frequency=120 MHz) (continued)

ADC0/1_CLK_PS [0:4]	ADC0/1_ ODD_PS	System Clock Divide Factor	ADC Clock (System Clock = 120 MHz)	Differential Conversion Speed with Default Sampling Time (2 cycles)	Single-Ended Conversion Speed with Default Sampling Time (2 cycles)
0b11100	0	58	2.07 MHz	138 Ksps	129 Ksps
	1	59	2.03 MHz	136 Ksps	127 Ksps
0b11101	0	60	2.0 MHz	133 Ksps	125 Ksps
	1	61	1.97 MHz	131 Ksps	123 Ksps
0b11110	0	62	1.94 MHz	129 Ksps	121 Ksps
	1	63	1.90 MHz	127 Ksps	119 Ksps
0b11111	0	64	1.88 MHz	125 Ksps	117 Ksps
	1	65	1.85 MHz	123 Ksps	115 Ksps

ADC Sampling Delay after Power-Up

To guarantee accuracy specifications, a delay of at least 8 ms must be present between the power-up of the VDDA supply and the start of the first ADC conversion. This delay allows internal ADC references to settle. The accuracy of a conversion during the first 8 ms is not guaranteed by the specifications, however conversion within the first 8 ms will be possible on the eQADC if this delay is not implemented in software.

Time Stamp Feature

The on-chip ADCs can provide a time stamp for the conversions they execute. A time stamp is the value of the time base counter latched when the EQADC detects the end of the analog input voltage sampling. A time stamp for a conversion command is requested by setting the TSR bit in the corresponding command. When TSR is negated, that is a time stamp is not requested, the ADC returns a single result message containing the conversion result. When TSR is asserted, that is a time stamp is requested, the ADC returns two result messages; one containing the conversion result, and afterwards another containing the time stamp for that conversion. The result messages are sent in this order to the RFIFOs and both messages are sent to the same RFIFO was specified in the MESSAGE_TAG field of the executed conversion command.

The time stamp can be provided by an external source using the STAC bus interface (more details in *Section , STAC Client Submodule (REDLC)*) or by the internal time base counter. The selection between the two sources is done by field ADC0/1_TBSEL in the ADC0/1_CR register or by field ATBSEL in registers ADC_ACR1-8. Refer to *Table 507* and *Table 517* for selection details.

The time base counter is a 16-bit up counter that wraps after reaching 0xFFFF. It is disabled after reset and it is enabled according to the setting of TBC_CLK_PS field in *Section , ADC Time Stamp Control Register (ADC_TSCR)*. TBC_CLK_PS defines if the counter is enabled or disabled, and, if enabled, at what frequency it is incremented. The time stamps are returned regardless of whether the time base counter is enabled or disabled. The time base counter can be reset by writing 0x0000 to the *Section , ADC Time Base Counter Registers (ADC_TBCR)*, with a write configuration command.

STAC Client Submodule (REDLC)

The shared time and angle count (STAC) bus provides access to one external time base, imported from the STAC bus to the eQADC. The eTPU2 module's time bases and angle count can be exported through the STAC client submodule interface. Time base and/or angle information of the eTPU2 engine can be exported to the eMIOS module and the eQADC, which are STAC clients.

The device’s STAC server identification assignment is shown in *Table 539*. The time slot assignment is fixed, so only time bases running at system clock divided by four or slower can be integrally exported. The STAC client submodule runs with the system clock, and its time slot timing is synchronized with the eTPU2 timing on reset. The time slot sequence is 0-1-2-3, such that they alternate between eTPU time bases.

Table 539. STAC Client Submodule Server Slot Assignment

eTPU2 Engine Time Base	Server ID
TCR1	0
TCR2	2

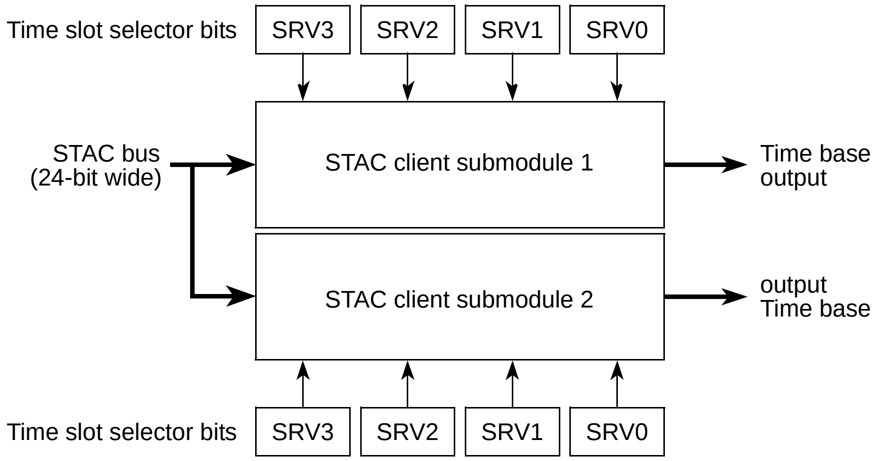
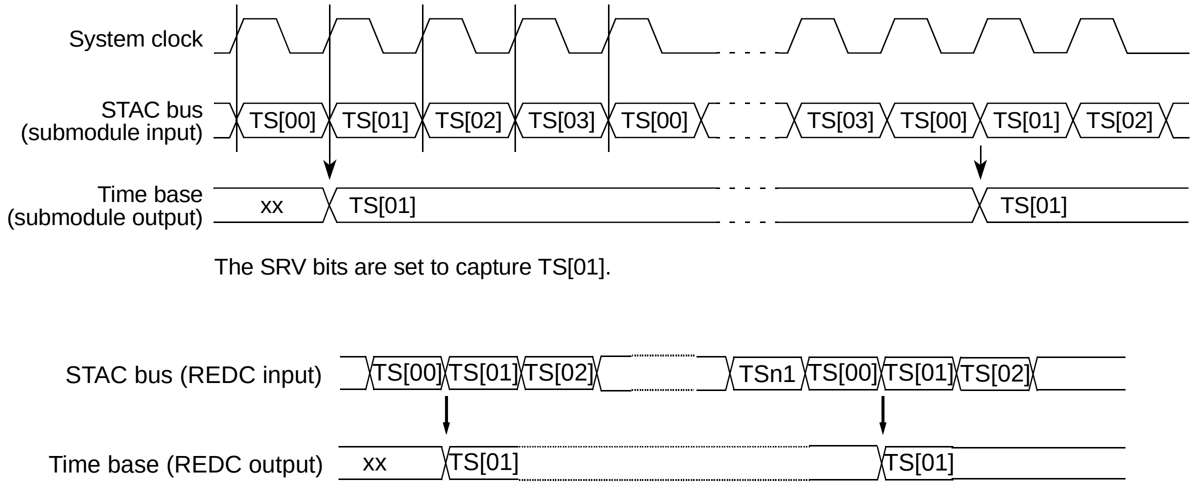


Figure 521. REDLC Block Diagram

The eQADC_REDLCCR[SRV1] bit selects the time slot of the STAC timebase 1 output and the eQADC_REDLCCR[SRV2] bit selects the time slot of the STAC timebase 2 output.

Figure 522 shows a timing diagram for the STAC client submodule.



NOTES: 1. Maximum of 16 time slots (TSn)
2. The SRV bits capture TS[01]

Figure 522. Timing Diagram for the STAC Bus and STAC Client Submodule Output

Every time the selected time slot changes, the STAC client submodule output is updated.

After the slot selection is done and the timebase data is extracted, the STAC client submodule selects 16 bits from the original 24-bit timebase data. These selected bits are the timebase to be used internal to the EQADC.

ADC pre-gain feature

Each ADC can be configured to have a selectable input gain as defined in *Section , Alternate Configuration 1-8 Control Registers (ADC_ACR1-8)*. This means the input signal is sampled and the result is amplified by factor 2, or 4 before the conversion phase. In present implementation of this feature, the conversion is 1 or 2 ADC clock cycles longer for gain 2 or gain 4, respectively.

ADC resolution selection feature

The ADCs conversion resolutions can be 8 bits, 10 bits or 12 bits as described in *Section , Alternate Configuration 1-8 Control Registers (ADC_ACR1-8)*. For conversions at a resolution less than 12, the ADC is executing less operations and the conversion time is smaller. In this ADC, it is verified that there is 1 ADC clock cycle for each bit of resolution. Therefore, for the same ADC clock frequency, the ADC sample frequency is higher for lower resolutions.

When a conversion is undertaken at a resolution less than 12, the result is presented by the ADC in right justified format in the 12-bit input bus e.g.: 0000xxxxxxxx for 8 bits and 00xxxxxxxxxx for 10 bits. The EQADC inverts the result to left justified format i.e.: xxxxxxxx0000 for 8 bits and xxxxxxxxxx00 for 10 bits. This is because the same calibration coefficients in the MAC can then be used. The left shift operation is done just after the conversion result enters the EQADC, in the Resolution Adjustment block prior to the MAC, as illustrated in *Figure 525*.

ADC Calibration Feature

Overview

There are three sets of calibration coefficients for each ADC. Each set is composed by a gain factor and an offset factor: GCCn/OCCn, ALTGCCn1/ALTGCCn1, and ALTGCCn2/ALTGCCn2, where n is the ADC number 0 or 1. The pair GCCn/OCCn is selected when it is used the normal configuration or the alternate configurations 3 to 8. The pair ALTGCCn1/ALTGCCn1 is used only when the alternate configuration 1 is selected. And the pair ALTGCCn2/ALTGCCn2 is for the alternate configuration 2. The description below is for a generic pair of gain/offset GCC/OCC.

The EQADC provides a calibration scheme to remove the effects of gain and offset errors from the results generated by the on-chip ADCs. Only results generated by the on-chip ADCs are calibrated. The results generated by ADCs on the external device are directly sent to RFIFOs unchanged. The main component of calibration hardware is a Multiply-and-Accumulate (MAC) unit, one per on-chip ADC, that is used to calculate the following transfer function which relates a calibrated result to a raw, uncalibrated one.

CAL_RES = GCC * RAW_RES + OCC+2;

where:

- CAL_RES is the calibrated result corresponding the input voltage V_i.
- GCC is the gain calibration constant.
- RAW_RES is the raw, uncalibrated result with resolution adjustment corresponding to an specific input voltage V_i.
- OCC is the offset calibration constant.
- The addition of two reduces the maximum quantization error of the ADC. See *Section , Quantization Error Reduction During Calibration*.

Calibration constants GCC and OCC are determined by taking two samples of known reference voltages and using these samples to calculate the values for the constants. For details and an example about how to calculate the calibration constants and use them in result calibration refer to *Section 23.7.6, ADC Result Calibration*. Once calculated, GCC is stored in the *Section , ADC0/1 Gain Calibration Constant Registers (ADC0_GCCR and ADC1_GCCR)*, and OCC in *Section , ADC0/1 Offset Calibration Constant Registers (ADC0_OCCR and ADC1_OCCR)*, from where their values are fed to the MAC unit. The alternate gain values are stored in *Section , ADC0/1 Alternate Gain Registers (ADC0_AGR1-2 and ADC1_AGR1-2)*, and the alternate offset values in *Section , ADC0/1 Alternate Offset Register (ADC0_AOR1-2 and ADC1_AOR1-2)*. Since the analog characteristics of each on-chip ADCs differs, each ADC has an independent pair of calibration constants.

A conversion result is calibrated according to the status of CAL bit in the command that initiated the conversion. If the CAL bit is asserted, the EQADC will automatically calculate the calibrated result before sending the result to the appropriate RFIFO or companion module. If the CAL bit is negated, the result is not calibrated, it bypasses the calibration hardware, and is directly sent to the appropriate RFIFO or companion module.

MAC Unit and Operand Data Format

The MAC unit diagram is shown in *Figure 523*. Each on-chip ADC has a separate MAC unit to fine-tune its conversion results. The description below considers the general calibration constant registers but it is the same for the alternate calibration constants.

The OCC0/1 operand is a 14-bit signed value stored in the *Section , ADC0/1 Offset Calibration Constant Registers (ADC0_OCCR and ADC1_OCCR)*. The RAW_RES operand is the raw uncalibrated result, and it is the direct output from the on-chip ADCs but passing through the resolution adjustment block. The GCC0/1 operand is a 15-bit fixed point unsigned value stored in the *Section , ADC0/1 Gain Calibration Constant Registers (ADC0_GCCR and ADC1_GCCR)*. The GCC is expressed in the *GCC_INT.GCC_FRAC* binary format. The integer part of the GCC (GCC_INT=GCC[1]) contains a single binary digit while its fractional part (GCC_FRAC=GCC[2:15]) contains 14 bits - see *Figure 524*. The gain constant equivalent decimal value ranges from 0 to 1.999938..., as shown in *Table 540*. Two is always added to the MAC output - see *Section , Quantization Error Reduction During Calibration*. CAL_RES output is the calibrated result, and it is a 14-bit unsigned value. CAL_RES is truncated to 0x3FFF, in case of a overflow, and to 0x0000, in case of an underflow.

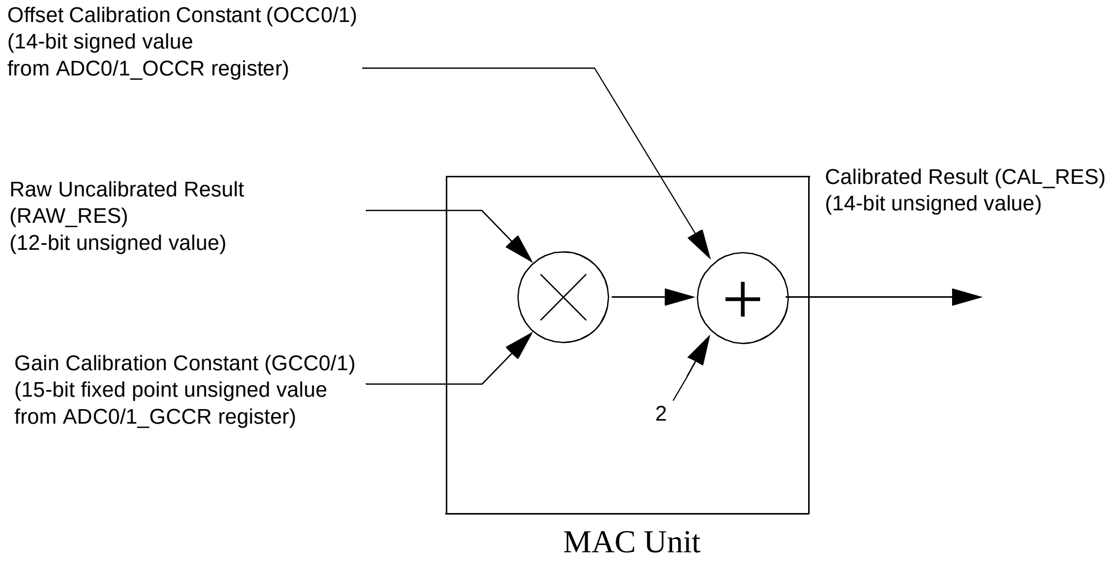


Figure 523. MAC Unit Diagram

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
GCC_INT	GCC_FRAC													
GCC[1]	GCC[2:15]													

Figure 524. Gain Calibration Constant Format

GCC_INT - Integer part of the gain calibration constant for ADC0/1

GCC_INT is the integer part of the gain calibration constant for ADC0/1.

GCC_FRAC[1:14] - Fractional part of the gain calibration constant for ADC0/1

GCC_FRAC is the fractional part of the gain calibration constant for ADC0/1. GCC_FRAC expresses decimal values ranging from 0 to 0.999938...

Table 540. Binary and Decimal Representations of the Gain Constant

Gain Constant (GCC_INT.GCC_FRAC binary format)	Corresponding Decimal Value
0.0000_0000_0000_00	0
...	...
0.1000_0000_0000_00	0.5
...	...
0.1111_1111_1111_11	0.999938...
1.0000_0000_0000_00	1
...	...
1.1100_0000_0000_00	1.75
...	...
1.1111_1111_1111_11	1.999938...

ADC Control Logic overview and command execution

Figure 525 shows the basic logic blocks involved in the ADC Control and how they interact. CFIFOs/RFIFOs interact with CBuffers/*Abort Cont*/*Result Message Return Logic* through the *FIFO Control Unit*. The EB and BN bits in the Command Message uniquely identify the CBuffer to which a command should be sent. The *FIFO Control Unit* decodes these bits and sends the ADC command to the proper CBuffer. Other blocks of logic are the *Resolution Adjustment*, *Result Format and Calibration Sub-Block*, the *Time Stamp Logic,* and the *MUX Control Logic*.

The *Resolution Adjustment Sub-Block* receives the 12-bit data bus directly from the ADC and changes the received conversion results from right aligned format of ADC to the left aligned format depending on the selected resolution of the conversion. This operation helps the calibration processing to use the calibration coefficients always with the same format.

The *Result Format and Calibration Sub-Block* formats the returning data into Result Messages and sends them to the RFIFOs[aw]. The returning data can be data read from an ADC register, a conversion result, or a time stamp. The formatting and calibration of conversion results also take place inside this sub-block.

The *Time Stamp Logic* latches the value of the time base counter or the STAC bus time base when detecting the end of the analog input voltage sampling, and sends it to the *Result Format and Calibration Sub-Block* as time stamp information.

The *MUX Control Logic* generates the proper MUX control signals and, when the ADC0/1_EMUX bits are asserted, the MA signals based on the channel numbers extracted from the ADC Command.

When the on-chip ADC abort feature is not enabled, ADC Commands are stored in the CBuffers as they come and they are executed in the first-in-first-out basis. After the execution of a command in ENTRY1 finishes all commands are shifted one entry. After the

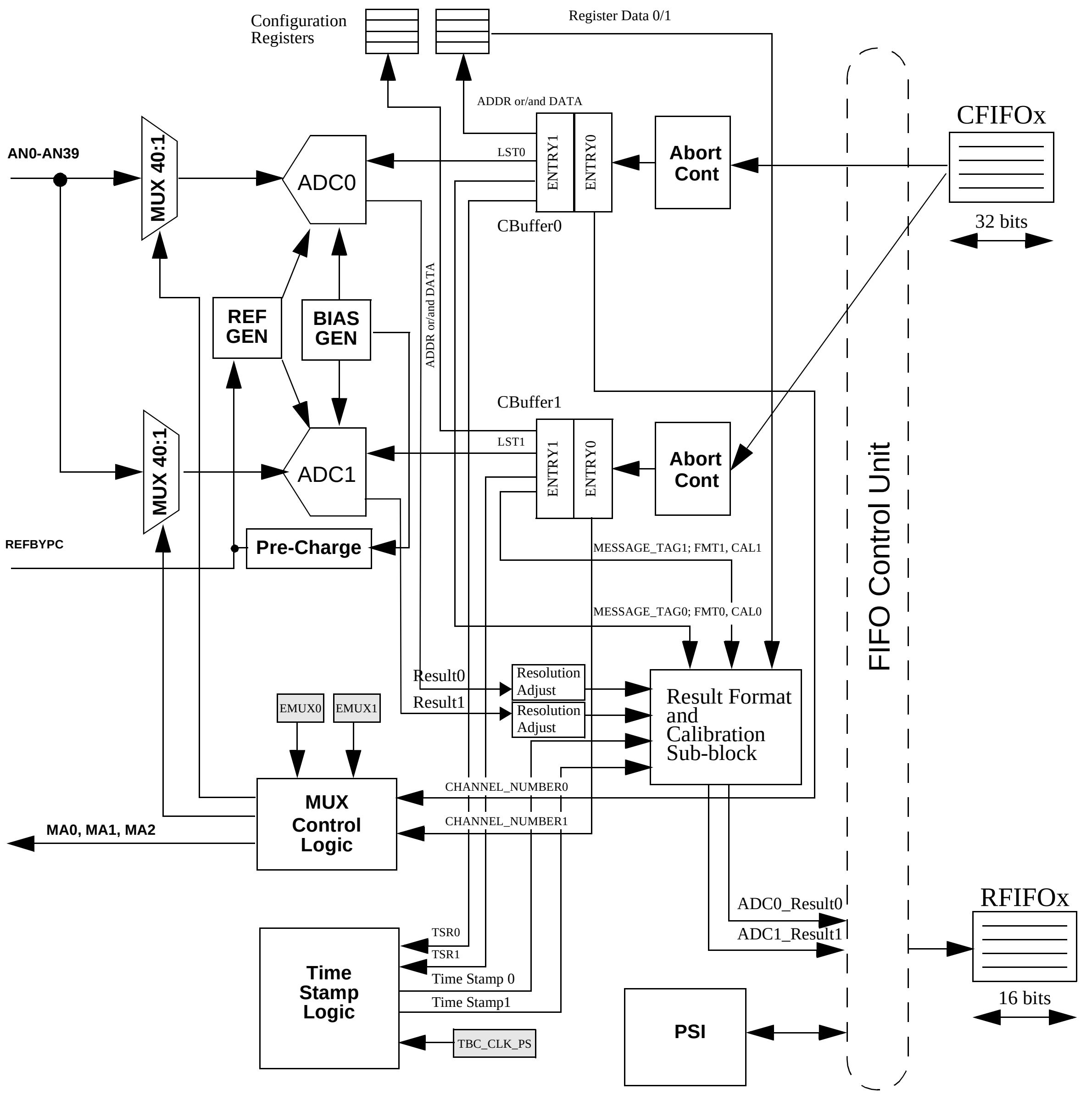
---

[aw] The result messages may also be routed to an on-chip companion module via the side interface, and then fed back to the RFIFOs.

shift, ENTRY0 is always empty and ready to receive a new command. Execution of configuration commands only start when they reach ENTRY1. Consecutive conversion commands are pipelined and their execution can start while in ENTRY0. This is explained below.

AD conversion accuracy can be affected by the settling time of the input channel multiplexers. Some time is required for the channel multiplexers internal capacitances to settle after the channel number is changed. If the time prior to sampling is not long enough to absorb this settling, then the settling time will take from ADC sampling time which may result in inaccurate sampling and ultimately compromise conversion result accuracy - see *Figure 526* (a). This could be avoided by switching the multiplexers in preparation for the next command's sampling during the AD conversion phase of the current command as showed in *Figure 526* (b). In EQADC, this is done in the following way; when a conversion command is in buffer ENTRY1 and another conversion command is identified in ENTRY0, then the channel number of ENTRY0 is sent to the *MUX Control Logic* some cycles before the sampling phase of the command in ENTRY0 starts. In this way, sampling for the next command can promptly start after the current conversion finishes because the internal capacitance of the multiplexers will be settled by that time, allowing for more accurate sampling. This is specially important for applications that require high conversion speeds, that is with the ADC running at maximum clock frequency and with the analog input voltage sampling time set to a minimum (2 ADC clock cycles), when the short sampling time does not allow the multiplexers to completely settle. The second advantage of pipelining conversion commands is to provide precise conversion intervals, which means the time intervals between two consecutive conversions are the same. This is important for any digital signal process application.

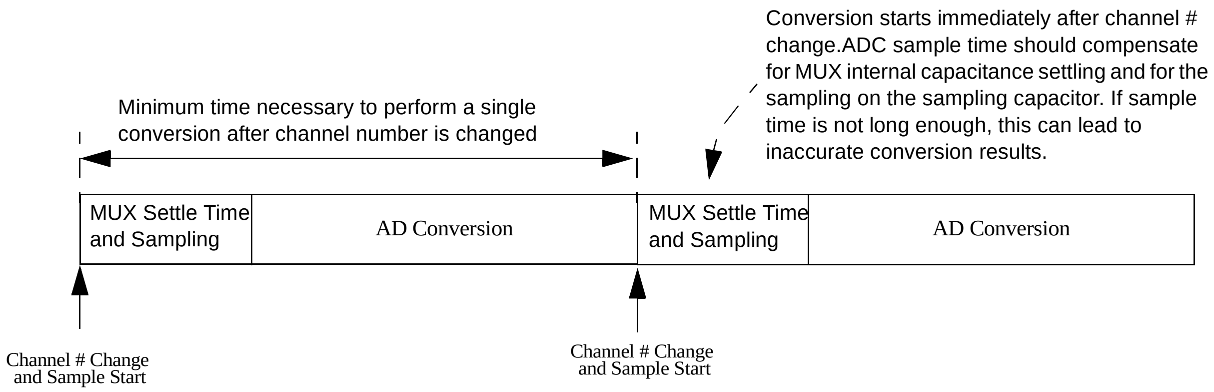
When the on-chip ADC abort feature is enabled, ADC Commands from CFIFO0 should be considered immediately, even stopping the execution of some command that is already in ENTRY1. When the abort request is sent to the ADC, the already stored commands in the CBuffers are copied in a temporary set of registers. The first ADC command from CFIFO0 is sent after the abort acknowledge indication from ADC. The process is the same as usual until the transfer of the last command from CFIFO0. Then the temporarily stored commands that were postponed by the abortion are recovered and they are pipelined for execution. After the last command from this temporary memory is transferred, the next commands are pipelined from the CFIFOs.



REGISTER FIELD Words in shaded boxes represent configuration register fields

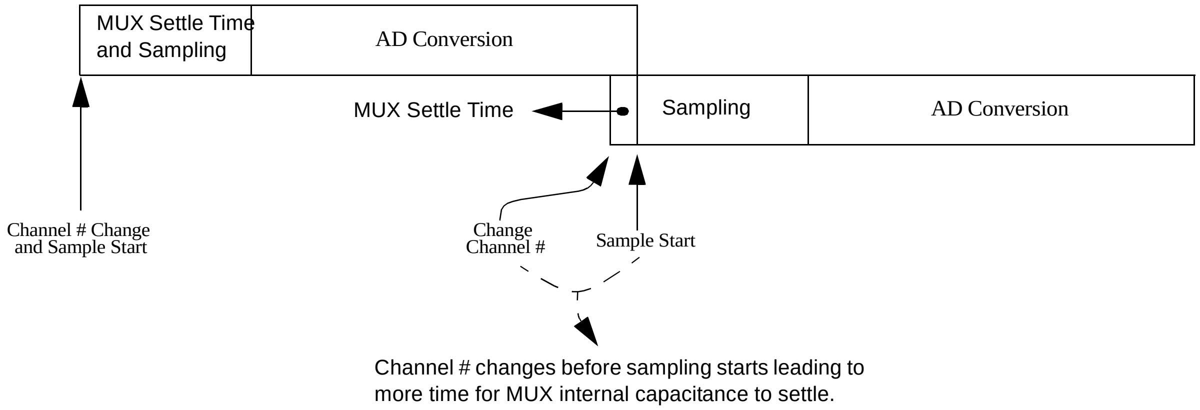
NOTE: x=0, 1, 2, 3, 4, 5

Figure 525. On-Chip ADC Control Scheme



Conversion starts immediately after channel # change.ADC sample time should compensate for MUX internal capacitance settling and for the sampling on the sampling capacitor. If sample time is not long enough, this can lead to inaccurate conversion results.

(a) Command Execution Sequence for Two Non-Overlapped Commands



Channel # changes before sampling starts leading to more time for MUX internal capacitance to settle.

(b) Command Execution Sequence for Two Overlapped Commands

Figure 526. Overlapping Consecutive Conversion Commands

23.6.7 Internal/External Multiplexing

Channel assignment

The internal analog multiplexers select one of the 40 analog input pins for conversion, based on the CHANNEL_NUMBER field of a Command Message. The analog input pin channel number assignments and the pin definitions vary depending on how the ADC0/1_EMUX are configured. Allowed combinations of ADC0/1_EMUX bits are shown in *Table 541* together with references to tables indicating how CHANNEL_NUMBER field of each conversion command must be set to avoid channel selection conflicts.

During differential conversions the analog multiplexer passes differential signals to both the positive and negative terminals of the ADC. The differential conversions can only be initiated on four channels: DAN0, DAN1, DAN2, and DAN3. Refer to *Table 542* and *Table 543* for the channel numbers used to select differential conversions.

Table 541. ADC0/1_EMUX Bits Combinations

ADC0_EMUX	ADC1_EMUX	CHANNEL_NUMBER should be set as in	
		ADC0	ADC1
0	0	Refer to *Table 542*	Refer to *Table 542*
0	1	Refer to *Table 542*	Refer to *Table 543*
1	0	Refer to *Table 543*	Refer to *Table 542*
1	1	Reserved (1)	

1. ADC0_EMUX and ADC1_EMUX must not be asserted at the same time.

Table 542 shows the channel number assignments for the non-multiplexed mode. The 43 single-ended channels and 4 differential pairs are shared between the two ADCs.

Table 542. Non-multiplexed Channel Assignments(1)

Input Pins			ADC	Channel Number in CHANNEL_NUMBER Field	
Analog Pin Name	**Other Functions**	**Conversion Type**	**ADC Number**	**Binary**	**Decimal**
AN0 to AN39		Single-ended	ADC0/ADC1	0000_0000 to 0010_0111	0 to 39
VRH		Single-ended	ADC0/ADC1	0010_1000	40
VRL		Single-ended	ADC0/ADC1	0010_1001	41
	50% x VREF (2),(3) (do not use for calibration)	Single-ended	ADC0/ADC1	0010_1010	42
	75% x VREF (2)	Single-ended	ADC0/ADC1	0010_1011	43
	25% x VREF (2)	Single-ended	ADC0/ADC1	0010_1100	44
INA_ADC0/1_0	Buffered bandgap	Single-ended	ADC0/ADC1	0010_1101	45
Reserved				0010_1110 to 0101_1111	46 to 95
DAN0+ and DAN0-		Differential	ADC0/ADC1	0110_0000	96
DAN1+ and DAN1-		Differential	ADC0/ADC1	0110_0001	97
DAN2+ and DAN2-		Differential	ADC0/ADC1	0110_0010	98
DAN3+ and DAN3-		Differential	ADC0/ADC1	0110_0011	99
Reserved				0110_0100 to 0111_1111	100 to 127
INA_ADC0/1_1	Temp Sensor	Single-ended	ADC0/ADC1	1000_0000	128
INA_ADC0/1_2	Spare	Single-ended	ADC0/ADC1	1000_0001	129
Reserved				1000_0010 to 1000_1111	130 to 143

Table 542. Non-multiplexed Channel Assignments[1] (continued)

Input Pins			ADC	Channel Number in CHANNEL_NUMBER Field	
Device Specific Use			ADC0	1001_0000 to 1001_0011	144 to 147
Reserved			ADC1	1001_0000 to 1001_0011	144 to 147
Reserved				1001_0100 to 1010_0001	148 to 161
Reserved			ADC1	1010_0010 to 1010_0111	162 to 167
INA_ADC0_3	Device Specific	Single-ended	ADC0	1010_0010	162
INA_ADC0_4	Device Specific	Single-ended	ADC0	1010_0011	163
INA_ADC0_5	Device Specific	Single-ended	ADC0	1010_0100	164
INA_ADC0_6	Device Specific	Single-ended	ADC0	1010_0101	165
INA_ADC0_7	Device Specific	Single-ended	ADC0	1010_0110	166
INA_ADC0_8	Device Specific	Single-ended	ADC0	1010_0111	167
Reserved				1010_1000 to 1100_0001	168 to 193
Reserved			ADC0	1100_0010 to 1100_0111	194 to 199
INA_ADC1_3	Device Specific	Single-ended	ADC1	1100_0010	194
INA_ADC1_4	Device Specific	Single-ended	ADC1	1100_0011	195
INA_ADC1_5	Device Specific	Single-ended	ADC1	1100_0100	196
INA_ADC1_6	Device Specific	Single-ended	ADC1	1100_0101	197
INA_ADC1_7	Device Specific	Single-ended	ADC1	1100_0110	198
INA_ADC1_8	Device Specific	Single-ended	ADC1	1100_0111	199
Reserved				1100_1000 to 1111_1111	200 to 255

1. The two on-chip ADCs can access the same analog input pins but simultaneous conversions are not allowed. Also, when one ADC is performing a differential conversion on a pair of pins, the other ADC must not access either of these two pins as single-ended channels.
2. VREF=VRH-VRL.
3. 50% x VREF = 50% ref = (VRH / VRL)/2, but this only applies before calibration. After calibration, the 50% reference point will actually return approximately 20 mV lower than the expected 50% of the difference between the High Reference Voltage (VRH) and the Low Reference Voltage (VRL).The 50% reference point should not be used to calibrate ADC. For calibration of the ADC only the 25% and 75% points should be used as described in *Section 23.7.6, ADC Result Calibration*.

Table 543 shows the channel number assignments for multiplexed mode. The ADC with the ADC0/1_EMUX bit asserted can access 4 differential pairs, 39 single-ended, and, at most, 64 externally multiplexed channels. Refer to *Section , External multiplexing*, for a detailed explanation about how external multiplexing can be achieved.

Table 543. Multiplexed Channel Assignments(1)

Input Pins			ADC	Channel Number in CHANNEL_NUMBER Field	
Analog Pin Name	**Other Functions**	**Conversion Type**	**ADC Number**	**Binary**	**Decimal**
AN0 to AN39 (2)		Single-ended	ADC0/ADC1	0000_0000 to 0010_0111	0 to 39
VRH		Single-ended	ADC0/ADC1	0010_1000	40
VRL		Single-ended	ADC0/ADC1	0010_1001	41
	50% x VREF (3),(4)	Single-ended	ADC0/ADC1	0010_1010	42
	75% x VREF (3)	Single-ended	ADC0/ADC1	0010_1011	43
	25% x VREF (3)	Single-ended	ADC0/ADC1	0010_1100	44
INA_ADC0/1_0	Buffered bandgap	Single-ended	ADC0/ADC1	0010_1101	45
Reserved				0010_1110 to 0011_1111	46 to 63
ANW	—	Single-ended	ADC0/ADC1	0100_0xxx	64 to 71
ANX	—	Single-ended	ADC0/ADC1	0100_1xxx	72 to 79
ANY	—	Single-ended	ADC0/ADC1	0101_0xxx	80 to 87
ANZ	—	Single-ended	ADC0/ADC1	0101_1xxx	88 to 95
DAN0+ and DAN0-		Differential	ADC0/ADC1	0110_0000	96
DAN1+ and DAN1-		Differential	ADC0/ADC1	0110_0001	97
DAN2+ and DAN2-		Differential	ADC0/ADC1	0110_0010	98
DAN3+ and DAN3-		Differential	ADC0/ADC1	0110_0011	99
Reserved				0110_0100 to 0111_1111	100 to 127
INA_ADC0/1_1	Temp Sensor	Single-ended	ADC0/ADC1	1000_0000	128
INA_ADC0/1_2	Spare	Single-ended	ADC0/ADC1	1000_0001	129
Reserved				1000_0010 to 1000_1111	130 to 143
Device Specific Use			ADC0	1001_0000 to 1001_0011	144 to 147
Reserved			ADC1	1001_0000 to 1001_0011	144 to 147
Reserved				1001_0100 to 1010_0001	148 to 161

Table 543. Multiplexed Channel Assignments[1] (continued)

Input Pins			ADC	Channel Number in CHANNEL_NUMBER Field	
Reserved			ADC1	1010_0010 to 1010_0111	162 to 167
INA_ADC0_3	Device Specific	Single-ended	ADC0	1010_0010	162
INA_ADC0_4	Device Specific	Single-ended	ADC0	1010_0011	163
INA_ADC0_5	Device Specific	Single-ended	ADC0	1010_0100	164
INA_ADC0_6	Device Specific	Single-ended	ADC0	1010_0101	165
INA_ADC0_7	Device Specific	Single-ended	ADC0	1010_0110	166
INA_ADC0_8	Device Specific	Single-ended	ADC0	1010_0111	167
Reserved				1010_1000 to 1100_0001	168 to 193
Reserved			ADC0	1100_0010 to 1100_0111	194 to 199
INA_ADC1_3	Device Specific	Single-ended	ADC1	1100_0010	194
INA_ADC1_4	Device Specific	Single-ended	ADC1	1100_0011	195
INA_ADC1_5	Device Specific	Single-ended	ADC1	1100_0100	196
INA_ADC1_6	Device Specific	Single-ended	ADC1	1100_0101	197
INA_ADC1_7	Device Specific	Single-ended	ADC1	1100_0110	198
INA_ADC1_8	Device Specific	Single-ended	ADC1	1100_0111	199
Reserved				1100_1000 to 1101_1111	200 to 223
Reserved				1110_0xxx to 1111_1xxx	224 to 255

1. The two on-chip ADCs can access the same analog input pins but simultaneous conversions are not allowed. Also, when one ADC is performing a differential conversion on a pair of pins, the other ADC must not access either of these two pins as single-ended channels.
2. Old version has reserved values for channel numbers 8 to 11 when EMUX =1. Therefore, now the behavior is different because it is converted the signal at AN8 to AN11, respectively.
3. VREF=VRH-VRL.
4. 50% x VREF = 50% ref = (VRH / VRL)/2, but this only applies before calibration. After calibration, the 50% reference point will actually return approximately 20 mV lower than the expected 50% of the difference between the High Reference Voltage (VRH) and the Low Reference Voltage (VRL). For calibration of the ADC only the 25% and 75% points should be used as described in *Section 23.7.6, ADC Result Calibration*.

External multiplexing

The EQADC can use from one to eight external multiplexer chips to expand the number of analog signals that may be converted. Up to 64 analog channels can be converted through external multiplexer selection. The externally multiplexed channels are automatically selected by the CHANNEL_NUMBER field of a Command Message, in the same way done with internally multiplexed channels. The software selects the external multiplexed mode by setting the ADC0/1_EMUX bit in either ADC0_CR or ADC1_CR depending on which ADC will perform the conversion. *Table 543* shows the channel number assignments for the multiplexed mode. There are 4 differential pairs, 39 single-ended, and, at most, 64 externally multiplexed channels which can be selected. Only one ADC can have its ADC0/1_EMUX bit asserted at a time.

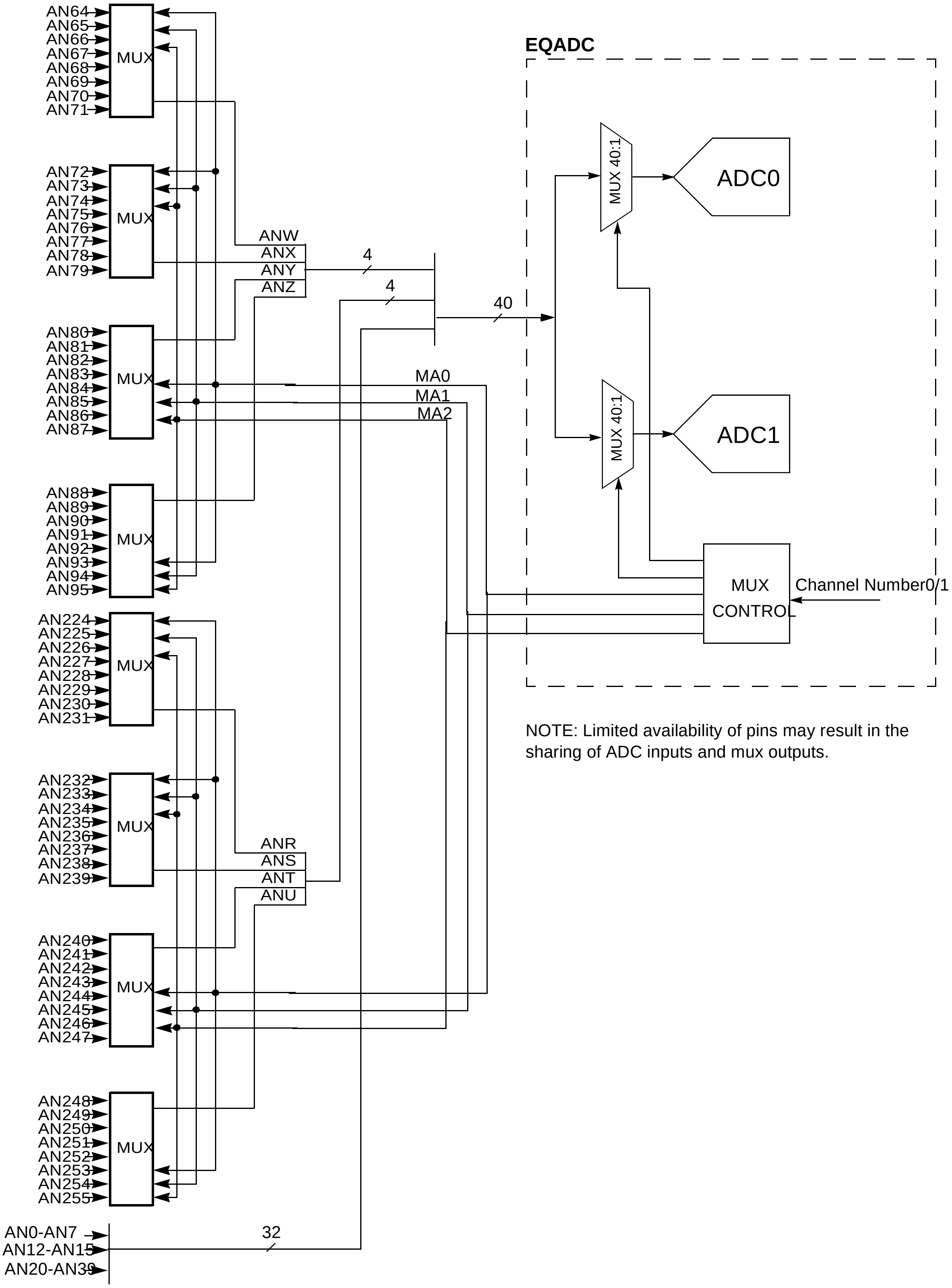
Figure 527 shows the maximum configuration of eight external multiplexer chips connected to the EQADC. The external multiplexer chip selects one of eight analog inputs and connects it to a single analog output, which is fed to a specific input of the EQADC. The EQADC provides three multiplexed address signals, MA0, MA1, and MA2, to select one of eight inputs. These three multiplexed address signals are connected to all eight external multiplexer chips. The analog output of the eight multiplex chips are each connected to eight separate EQADC inputs, ANR, ANS, ANT, ANU, ANW, ANX, ANY, and ANZ. The MA pins correspond to the three least significant bits of the channel number that selects ANR, ANS, ANT, ANU, ANW, ANX, ANY, and ANZ with MA0 being the most significant bit - See *Table 544*.

Table 544. Encoding of MA Pins(1)

Channel Number selecting ANW, ANX, ANY, ANZ (decimal)				MA0	MA1	MA2
ANW	ANX	ANY	ANZ			
64	72	80	88	0	0	0
65	73	81	89	0	0	1
66	74	82	90	0	1	0
67	75	83	91	0	1	1
68	76	84	92	1	0	0
69	77	85	93	1	0	1
70	78	86	94	1	1	0
71	79	87	95	1	1	1

1. ‘0’ means pin is driven LOW and ‘1’ that pin is driven HIGH.

When the external multiplexed mode is selected for either ADC, the EQADC automatically creates the MA output signals from CHANNEL_NUMBER field of a Command Message. The EQADC also converts the proper input channel (ANW, ANX, ANY, and ANZ) by interpreting the CHANNEL_NUMBER field. As a result, up to 64 externally multiplexed channels appear to the conversion queues as directly connected signals.



NOTE: Limited availability of pins may result in the sharing of ADC inputs and mux outputs.

Figure 527. Example of External Multiplexing

23.6.8 EQADC DMA/Interrupt request

Table 545 lists methods to generate interrupt requests in the EQADC queuing control and triggering control. The DMA/interrupt request select bits and the DMA/interrupt enable bits are described in *Section , EQADC Interrupt and DMA Control Registers (EQADC_IDCR)*, and the interrupt flag bits are described in *Section , EQADC FIFO and Interrupt Status Registers (EQADC_FISR)*. *Table 528* depicts all interrupts and DMA requests generated by the EQADC.

Table 545. EQADC FIFO Interrupt Summary(1)

Interrupt	Condition	Clearing Mechanism
Non Coherency Interrupt	NCIEx = 1 NCFx = 1	Clear NCFx bit by writing a “1” to the bit.
Result FIFO Overflow Interrupt(2)	RFOIEx = 1 RFOFx = 1	Clear RFOFx bit by writing a “1” to the bit.
Command FIFO Underflow Interrupt(2)	CFUIEx = 1 CFUFx = 1	Clear CFUFx bit by writing a “1” to the bit.
Result FIFO Drain Interrupt	RFDEx = 1 RFDSx = 0 RFDFx = 1	Clear RFDFx bit by writing a “1” to the bit.
Command FIFO Fill Interrupt	CFFEx = 1 CFFSx = 0 CFFFx = 1	Clear CFFFx bit by writing a “1” to the bit.
End of Queue Interrupt	EOQIEx = 1 EOQFx = 1	Clear EOQFx bit by writing a “1” to the bit.
Pause Interrupt	PIEx = 1 PFx =1	Clear PFx bit by writing a “1” to the bit.
Trigger Overrun Interrupt(2)	TORIEx = 1 TORFx =1	Clear TORFx bit by writing a “1” to the bit.

1. For details refer to *Section , EQADC FIFO and Interrupt Status Registers (EQADC_FISR)*, and *Section , EQADC Interrupt and DMA Control Registers (EQADC_IDCR)*.
2. Apart from generating an independent interrupt request for when a RFIFO Overflow Interrupt, a CFIFO Underflow Interrupt, and a CFIFO Trigger Overrun Interrupt occurs, the EQADC also provides a combined interrupt request at which these requests from ALL CFIFOs are ORed. Refer to *Figure 528* for details.

Table 546 describes a list of methods to generate DMA requests in the EQADC.

Table 546. EQADC FIFO DMA Summary(1)

DMA Request	Condition	Clearing Mechanism
Result FIFO Drain DMA Request	RFDEx = 1 RFDSx = 1 RFDFx = 1	The EQADC automatically clears the RFDFx when RFIFOx becomes empty. Writing “1” to the RFDFx bit is not allowed.
Command FIFO Fill DMA Request	CFFEx = 1 CFFSx = 1 CFFFx = 1	The EQADC automatically clears the CFFFx when CFIFOx becomes full. Writing “1” to the CFFFx bit is not allowed.

1. For details refer to *Section , EQADC FIFO and Interrupt Status Registers (EQADC_FISR)*, and *Section , EQADC Interrupt and DMA Control Registers (EQADC_IDCR)*.

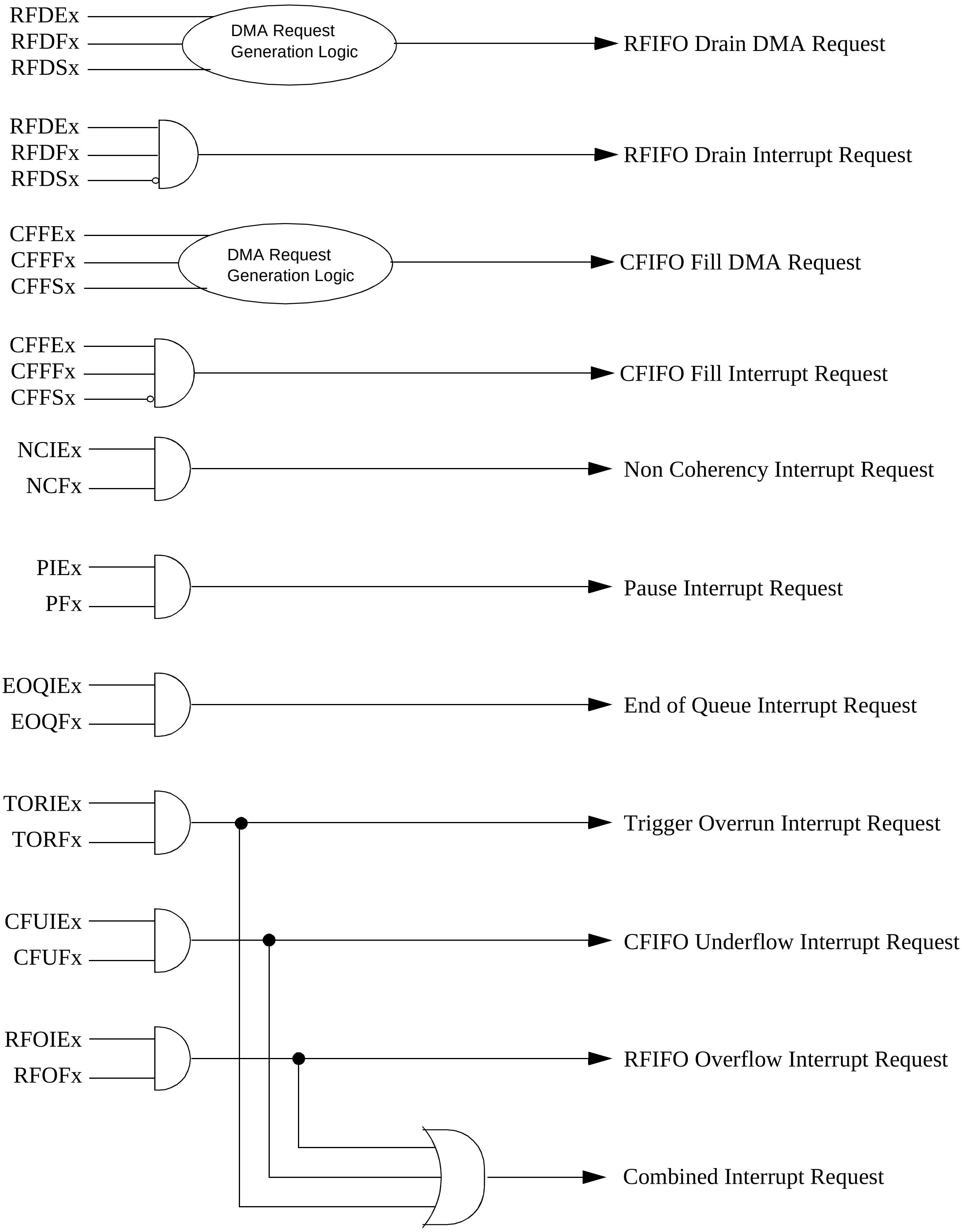


Figure 528. EQADC DMA and Interrupt Requests

23.6.9 EQADC Synchronous Serial Interface (SSI) Sub-Block

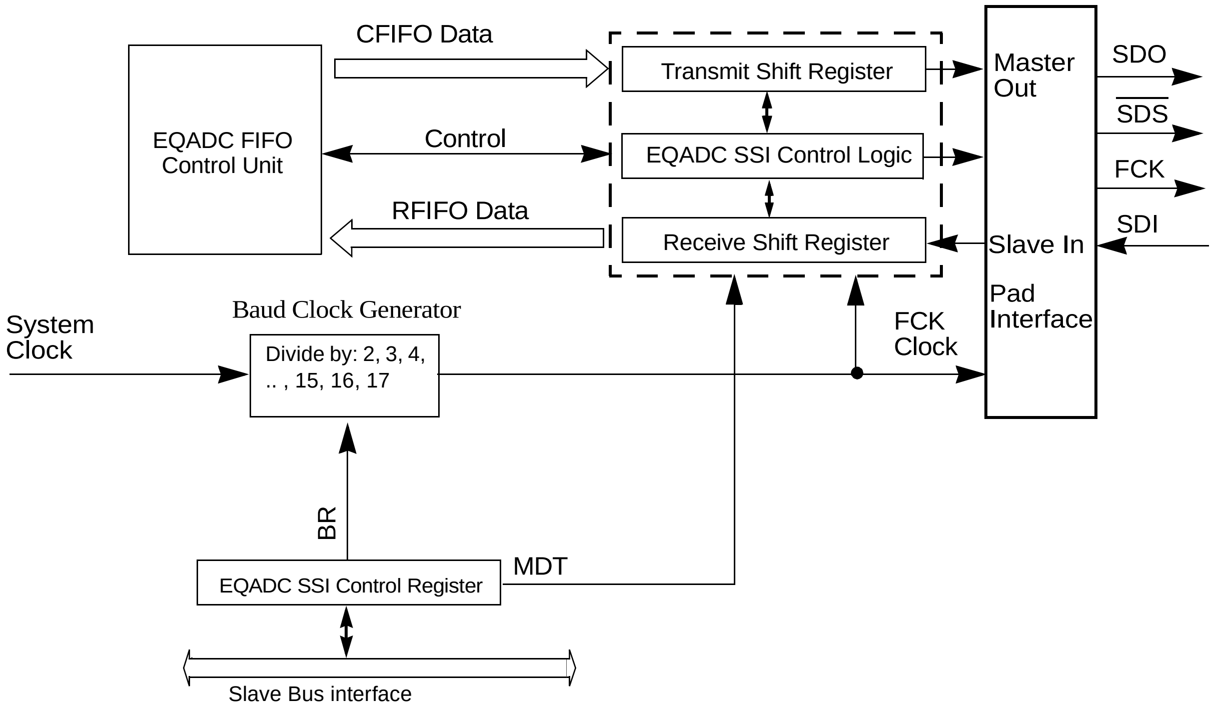


Figure 529. EQADC Synchronous Serial Interface Block Diagram

The EQADC SSI protocol allows for a full duplex, synchronous, serial communication between the EQADC and a single external device. *Figure 529* shows the different components inside the EQADC SSI block. The EQADC SSI sub-block on the EQADC is always configured as a master. The EQADC SSI has four associated port pins:

- Free running Clock (FCK)
- Serial Data Select ($\overline{\text{SDS}}$)
- Serial Data In (SDI)
- Serial Data Out (SDO)

The FCK clock signal times the shifting and sampling of the two serial data signals and it is free running between transmissions, allowing it to be used as the clock for the external device. The $\overline{\text{SDS}}$ signal will be asserted to indicate the start of a transmission, and negated to indicate the end or the abort of a transmission. SDI is the master serial data input and SDO the master serial data output.

The EQADC SSI sub-block is enabled by setting the ESSIE field in the *Section , EQADC Module Configuration Register (EQADC_MCR)*. When enabled, the EQADC SSI can be optionally capable of starting serial transmissions. When serial transmissions are disabled (ESSIE set to 0b10), no data will be transmitted to the external device but FCK will be free-running. This operation mode permits the control of the timing of the first serial transmission, and can be used to avoid the transmission of data to an unstable external device, for example, a device that is not fully reset. This mode of operation is specially important for the reset procedure of an external device that uses the FCK as its main clock.

The main elements of the EQADC SSI block are the shift registers. The 26-bit transmit shift register in the master and 26-bit receive shift register in the slave are linked by the SDO pin. In a similar way, the 26-bit transmit shift register in the slave and 26-bit receive shift register in the master are linked by the SDI pin. See *Figure 530*. When a data transmission operation is performed, data in the transmit registers is serially shifted twenty-six bit positions into the receive registers by the FCK clock from the master; data is exchanged between the master and the slave. Data in the master transmit shift register in the beginning of a transmission operation becomes the output data for the slave, and data in the master receive shift register after a transmission operation is the input data from the slave.

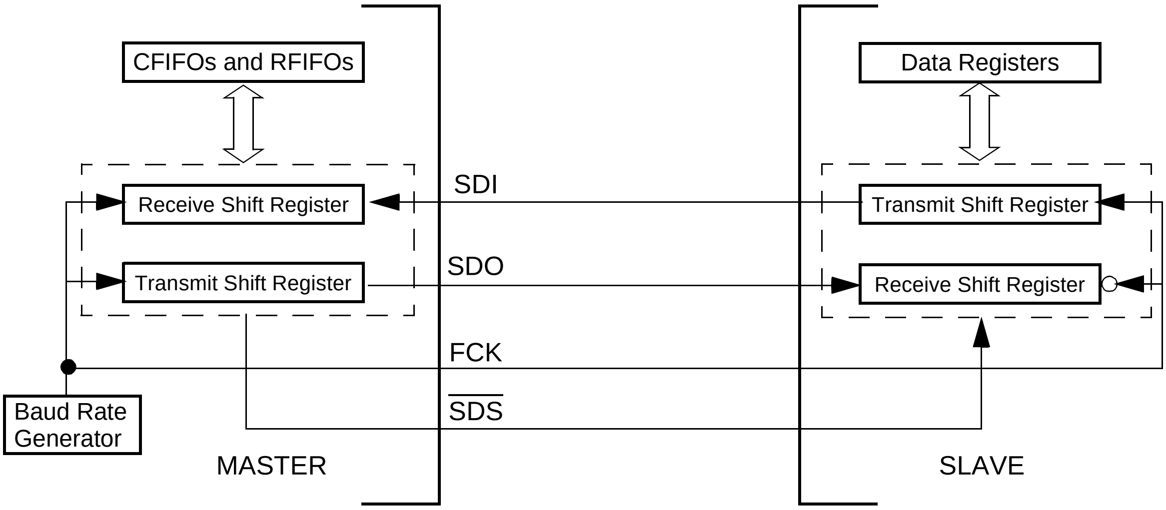


Figure 530. Full Duplex Pin Connection

EQADC SSI data transmission protocol

Figure 531 shows the timing of an EQADC SSI transmission operation. The main characteristics of this protocol are:

- FCK is free running, it does not stop between data transmissions. FCK will be driven low:
  - When the serial interface is disabled.
  - In stop/debug mode.
  - Immediately after reset.
- Frame size is fixed to 26 bits.
- MSB bit is always transmitted first.
- Master drives data on the positive edge of FCK and latches incoming data on the next positive edge of FCK.
- Slave drives data on the positive edge of FCK and latches incoming data on the negative edge of FCK.

Master initiates a data transmission by driving $\overline{\text{SDS}}$ low, and its MSB bit on SDO on the positive edge of FCK. Once an asserted $\overline{\text{SDS}}$ is detected, the slave shifts its data out, one bit at a time, on every FCK positive edge. Both the master and the slave drive new data on the serial lines on every FCK positive edge. This process continues until all the initial 26-bits in the master shift register are moved into the slave shift register. t_{DT} is the delay between two consecutive serial transmissions, time during which SDS is negated. When ready to start of the next transmission, the slave must drive the MSB bit of the message on every positive edge of FCK regardless of the state of the SDS signal. On the next positive edge, the second bit of the message is conditionally driven according to if an asserted SDS was detected by the slave on the preceding FCK negative edge. This is an important requisite since the SDS and the FCK are not synchronous. The SDS signal is not generated by FCK, rather both are generated by the system clock, so that it is not guaranteed that FCK edges will precede SDS ones. While SDS is negated, the slave continuously drives its MSB bit on every positive edge of FCK until it detects an asserted SDS on the immediately next FCK negative edge. See *Figure 532* for three situations showing how the slave should behave according to when SDS is asserted.

Note: *On the master, the FCK is not used as a clock. Although, the EQADC SSI behavior is described in terms of the FCK positive and negative edges, all EQADC SSI related signals (SDI, SDS, SDO, and FCK) are synchronized by the system clock on the master side. There are no restrictions regarding the use of the FCK as a clock on the slave device.*

Abort Feature

The master indicates it is aborting the current transfer by negating SDS before the whole data frame has being shifted out, that is the 26th bit of data being transferred has not being shifted out. The EQADC ignores the incompletely received message. The EQADC resends the aborted message whenever the corresponding CFIFO becomes again the highest priority CFIFO with commands bound for not-full external CBuffer. Refer to *Section , CFIFO Common Prioritization and Command Transfer*, for more information on aborts and CFIFO priority.

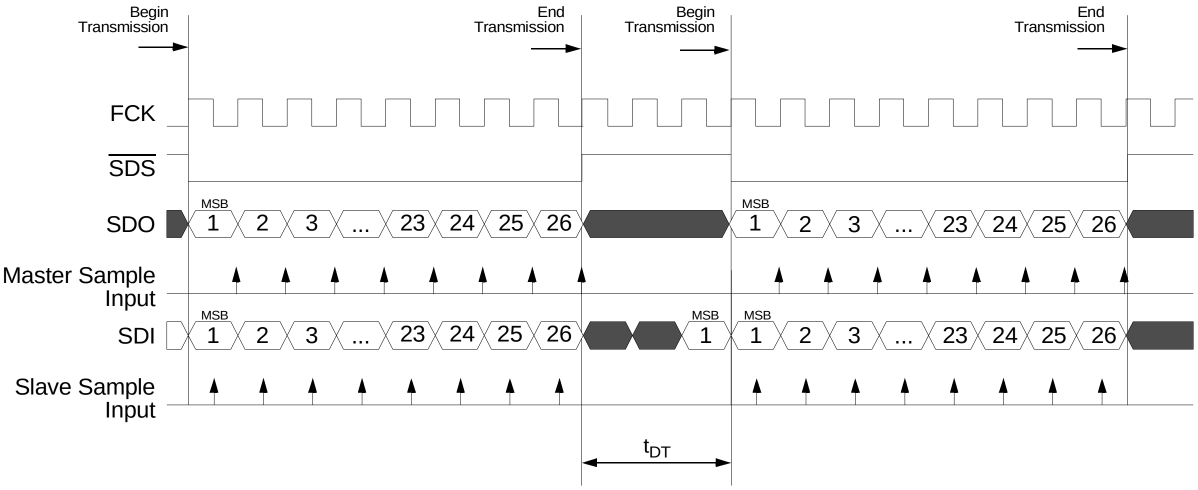
Baud clock generation

As shown in *Figure 529*, the baud clock generator divides the system clock to produce the baud clock. The BR field in *Section , EQADC SSI Control Register (EQADC_SSICR)*, selects the system clock divide factor as in *Table 497*. (ax)

$$\mathrm{BaudClockFrequency} = \frac{\mathrm{SystemClockFrequency(MHz)}}{\mathrm{SystemClockDivideFactor}}$$

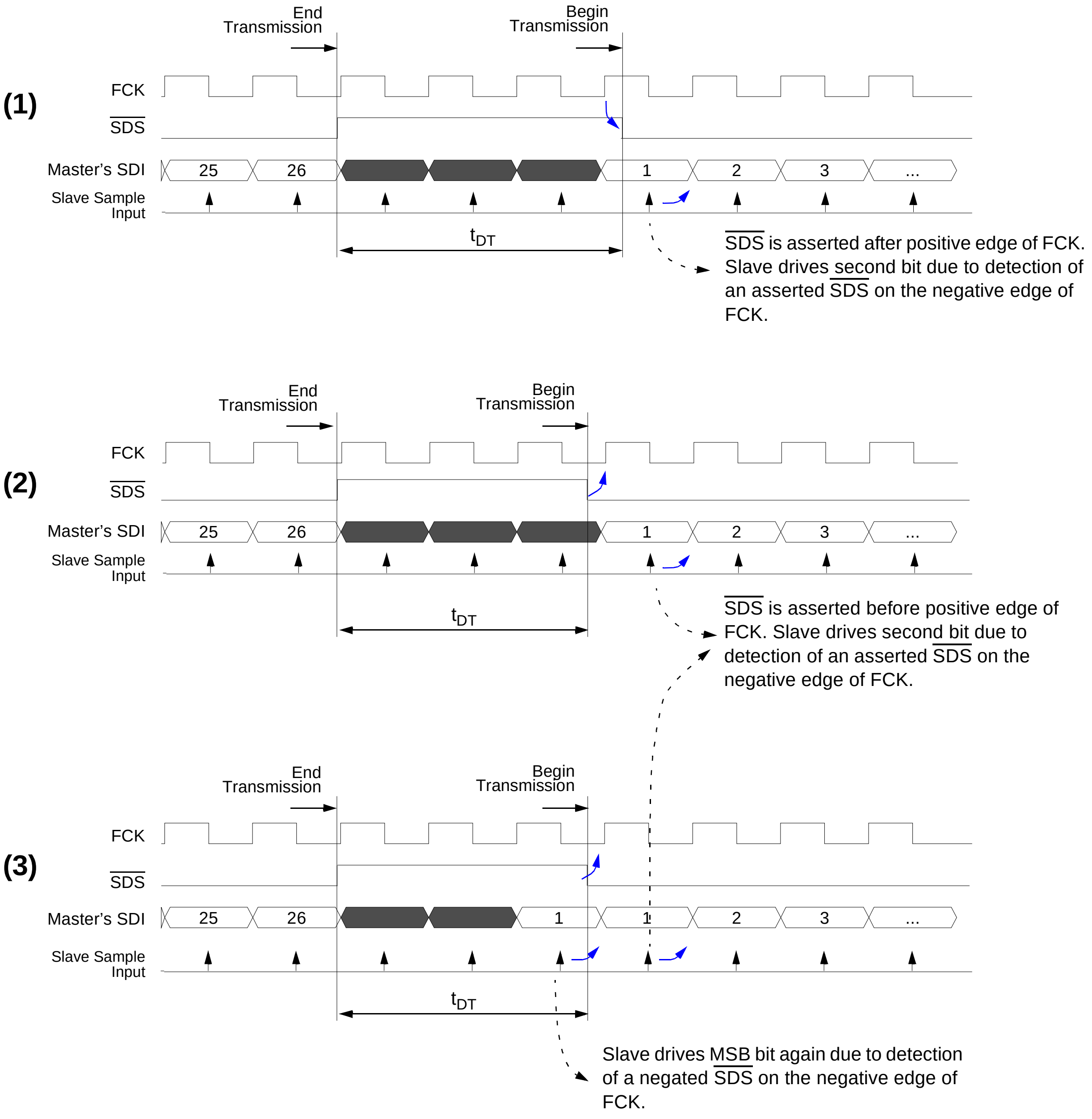
---

ax. Maximum FCK frequency is highly dependable on track delays, master pad delays, and slave pad delays.



t_{MDT} = Minimum t_{DT} is programmable and defined in the *Section , EQADC SSI Control Register (EQADC_SSICR)*

Figure 531. Synchronous Serial Interface Protocol Timing



SDS is asserted after positive edge of FCK. Slave drives second bit due to detection of an asserted SDS on the negative edge of FCK.

SDS is asserted before positive edge of FCK. Slave drives second bit due to detection of an asserted SDS on the negative edge of FCK.

Slave drives MSB bit again due to detection of a negated SDS on the negative edge of FCK.

Figure 532. Slave Driving the MSB and Consecutive Bits in a Data Transmission

23.6.10 EQADC Parallel Side Interface (PSI) Sub-Block

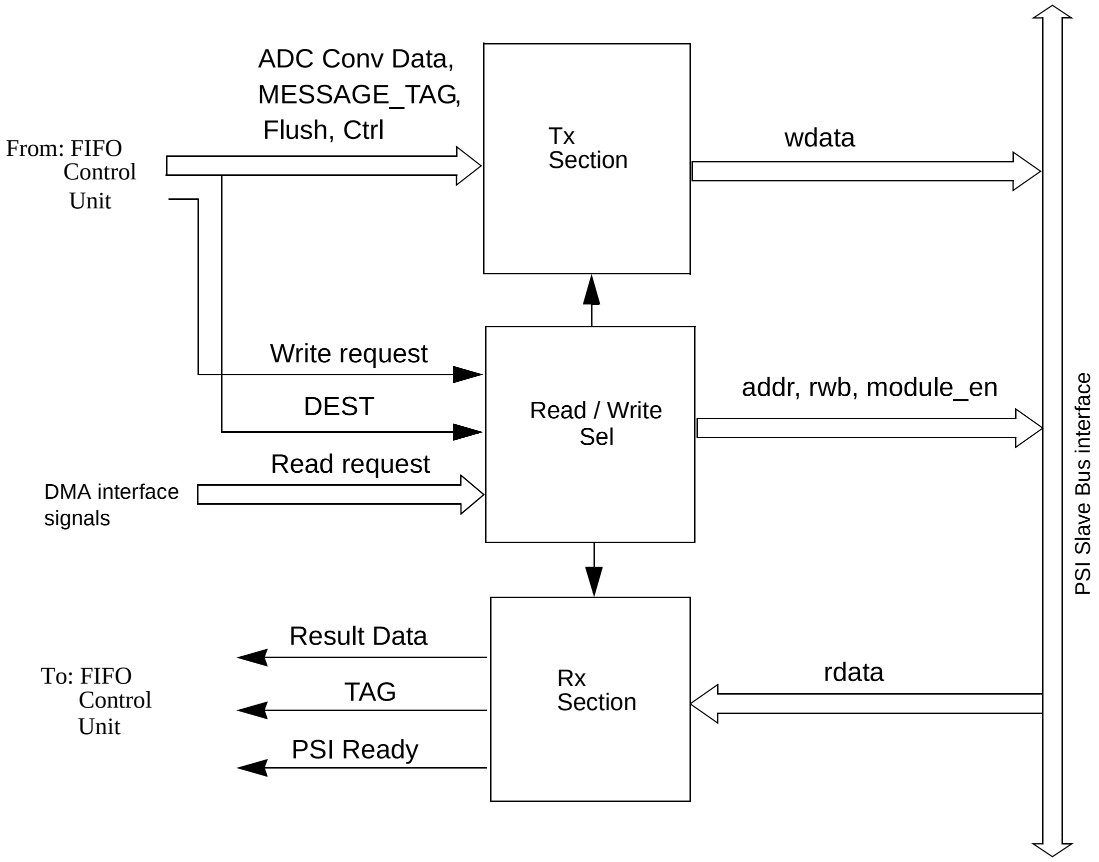


Figure 533. EQADC Parallel Side Interface Block Diagram

The EQADC PSI sub-block allows communication between the EQADC and the companion modules through a local slave bus that has the EQADC block as the master and the companion modules as the slaves. However, due to the poor processing capability of EQADC, it is defined a very limited number of addresses and a maximum of 15 different slave blocks to be enabled.

The conversion result goes through the Result Formatting and Calibration sub-block and the corresponding MESSAGE_TAG and DEST fields are decoded to decide the destination of the conversion result data. When the DEST field is not zero as described in *Section , Alternate Configuration 1-8 Control Registers (ADC_ACR1-8)*, the MESSAGE_TAG bits, and some control bits, and the conversion result data are sent to the transmission section of the PSI sub-block. This set of data is processed and sent to the corresponding on-chip companion module.

The companion module can also send back to EQADC the result of some processing in the received data. In this case, the receiver section of he PSI sub-block treats this request and receives the result data with the corresponding MESSAGE_TAG or TAG field. This TAG is used by the decoder to select which RFIFO will be written with the received companion module data.

Input / Output signals description

The information content of the input and the output data buses from/to the companion modules are described in *Figure 534*. The input data format is used in the Read cycle and corresponds to the data supplied by some companion module. The output data format is used in the Write cycle of this interface block and contains information / controls to some companion module.

Read (Input) / Write (Output) Data Buses Content

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	▨	▨	▨	▨	▨	▨	▨	▨	0	0	0	0	TAG**[0:3]**			
W	▨	▨	▨	▨	▨	▨	▨	▨	FLUSH	CTRL**[0:1]**		▨	MESSAGE_TAG**[0:3]**			
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	RESULT_DATA**[0:15]**															
W	ADC_CONV_RESULT**[0:15]**															
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 534. PSI Input and Output Data Buses Content

FLUSH — Master block Flush request/control bit

The FLUSH signal is used to request a Flush flow in the slave block. More details is presented in item *Section , Conversion Command Format for Alternate Configurations*.

1 = Flush request
0 = No flush request.

CTRL[0:1] — Control bits

Table 547 describes the CTRL[0:1] field functions. This field is used for the control of the companion module.

Table 547. CTRL[0:1] field description

CTRL[0:1]	Description
00	INH_RET or PREFILL - This mode indicates that the companion module should not send back some RESULT_DATA corresponding to the accompanying ADC_CONV_RESULT data. When the slave module is the Decimation Filter, this control enables the PREFILL filter mode.
01	CONVERSION RESULT - It indicates that the ADC_CONV_RESULT field should be treated as a valid sample data. This control mode is useful to Decimation Filter to put the filter em normal mode instead of prefill mode.
10	TIME STAMP o REGISTER READ - A time stamp indicates that the ADC_CONV_RESULT field should follow a bypass flow in the companion module, returning back to the EQADC without any modification.
11	reserved

MESSAGE_TAG[0:3] — Message tag bits field

This field indicates the RFIFO destination associated with the ADC_CONV_RESULT sample. This value is stored by the companion module and is used to address the destination RFIFO register when a RESULT_DATA is generated due to that ADC_CONV_RESULT sample.

TAG[0:3] — Companion module tag bits

This bit field is used to address the appropriate destination RFIFO in the EQADC for the accompanying RESULT_DATA bits.

In eQADC application, this is used to address the appropriate RFIFO in the eQADC block. In this case, the possible values are only from 0000 to 0101.

ADC_CONV_RESULT[0:15] — ADC Conversion Result Data

This bit field is the ADC conversion result data after passing through the calibration and formatting block.

RESULT_DATA[0:15] — Companion Module Result Data

This bit field corresponds to the companion module data processing result to EQADC.

PSI transmitter / Write section

The transmission sub-block formats the data bus from RFIFO control sub-block to send to the PSI slave wdata bus. The transmission data is registered and its content is described in *Section , Input / Output signals description.* The transmission has higher priority than reception. This is done to avoid the use of memory to store transmission data and it is not used waiting time for transmission.

The destination companion module for the transmission data is obtained by decoding the DEST[0:3] bits. The not null decimal value of DEST[0:3] is used to uniquely set the corresponding module enable signal. For example, DEST[0:3] value equal to 0xF that corresponds to the decimal value 15 is going to set only the module enable 15. All other module enable from 1 to 14 are not set.

PSI Receiver / Read section

The receiver sub-block receives data from some companion module using the PSI slave bus interface. The companion module sends a read request to EQADC using the DMA read request line. The PSI logic sends a read command if there is no transmission request. The received data has the structure described in item *Section , Input / Output signals description*.

23.6.11 Analog Sub-Block

Analog to Digital Converter (ADC)

ADC architecture

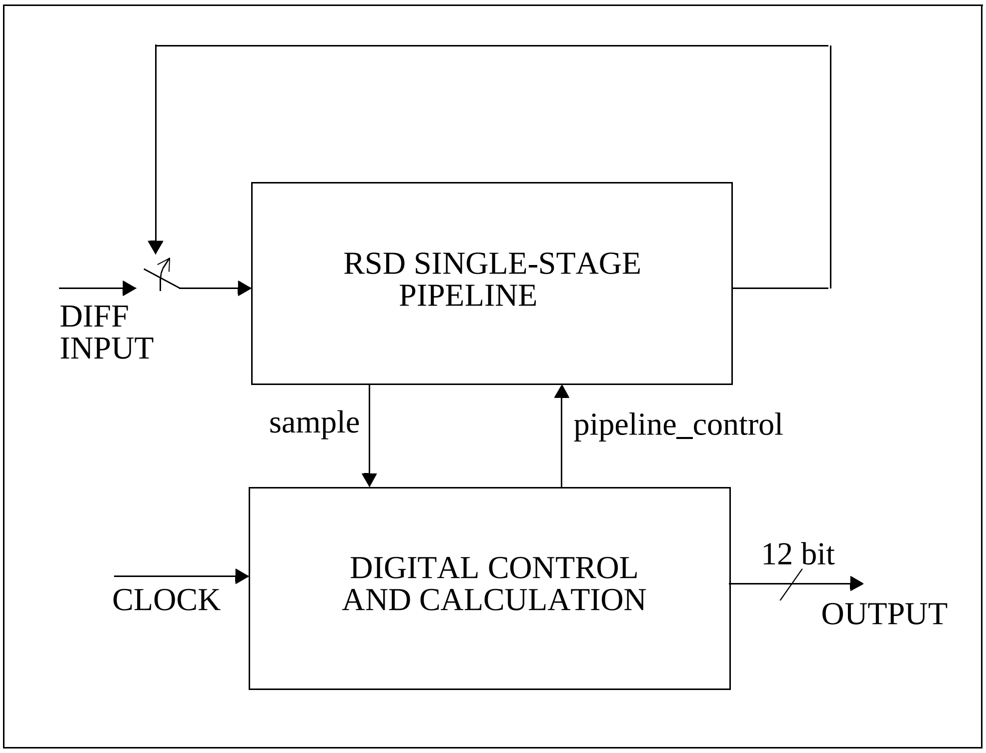


Figure 535. RSD ADC Block Diagram

The RSD Cyclic ADC consists of two main portions, the analog RSD Stage, and the digital control and calculation block, as shown in *Figure 535*. To begin an analog to digital conversion, a differential input is passed into the analog RSD stage. The signal is passed through the RSD stage, and then from the RSD stage output, back to its input to be passed again. To complete a 12-bit conversion, the signal must pass through the RSD stage 12 times. For 10-bit and 8-bit resolution, the signal must pass 10 or 8 times through the RSD. Each time an input signal is read into the RSD stage, a digital sample is taken by the digital control/calculation block. The digital control/calculation block uses this sample to tell the analog block how to condition the signal. The digital block also saves each successive sample and adds them according to the RSD algorithm at the end of the entire conversion cycle.

RSD overview

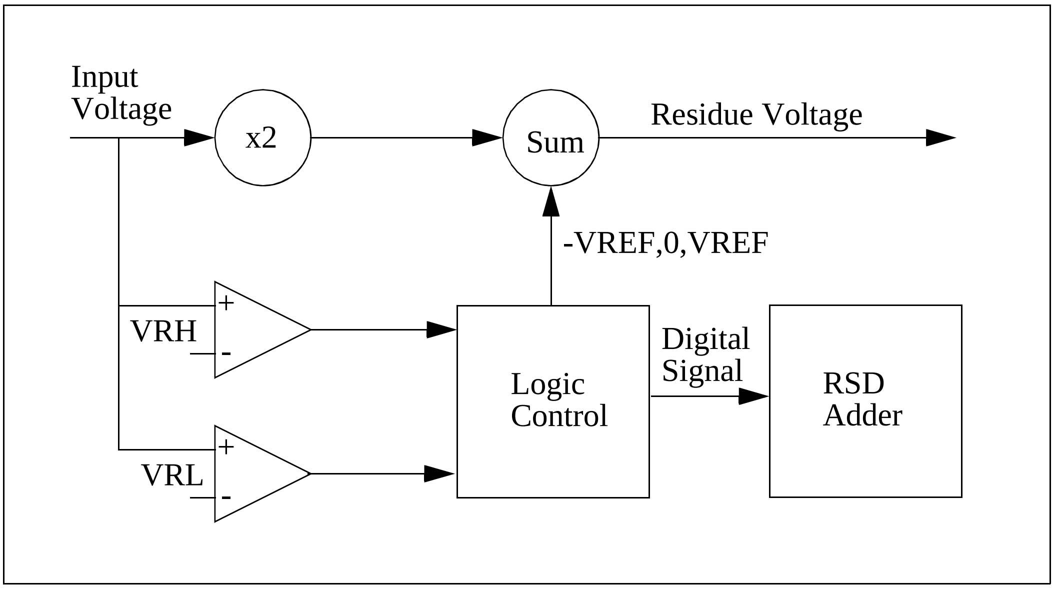


Figure 536. RSD Stage Block Diagram

On each pass through the RSD stage, the input signal will be multiplied by exactly two, and summed with either -VREF, 0, or VREF, depending on the Logic Control. The Logic Control will determine -VREF, 0, or VREF depending on the two comparator inputs. As the Logic Control sets the summing operation, it also sends a digital value to the RSD adder. Each time an analog signal passes through the RSD single-stage, a digital value is collected by the RSD adder. At the end of an entire AD conversion cycle, the RSD adder uses these collected values to calculate the 12-bit/10-bit/8-bit digital output.

Figure 537 shows the transfer function for the RSD stage. Note how the digital value (a, b) is dependent on the two comparator inputs.

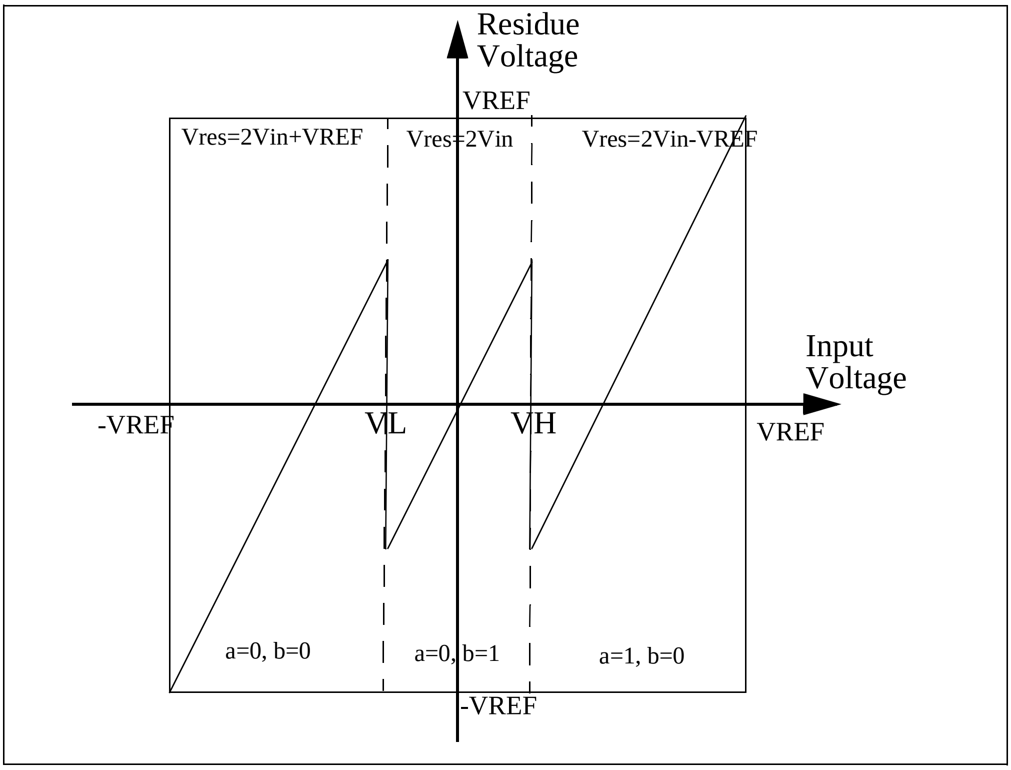


Figure 537. RSD Stage Transfer Function

In each pass through the RSD stage, the residue will be sent back to be the new input, and the digital signals, a and b, will be stored. For the 12-bit ADC, input signal is sampled during the input phase, and after each of the 12 passes through the RSD stage. Thus, 13 total a and b values are collected. Upon collecting all these values, they will be added according to the RSD algorithm to create the 12-bit digital representation of the original analog input. The bits are added in the following manner:

RSD Adder

The array, s1 to s12,will be the digital output of the RSD ADC with s1 being the MSB and s12 being the LSB.

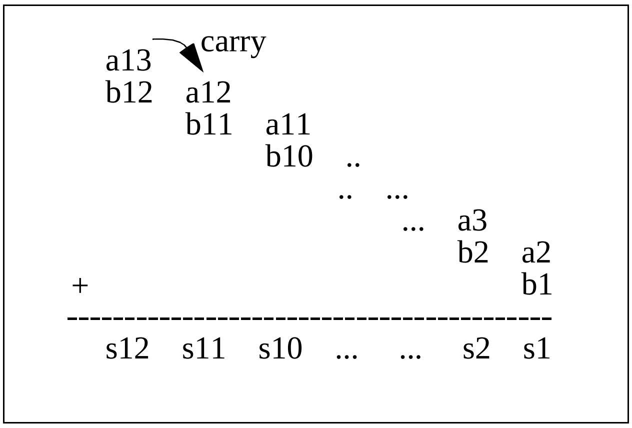


Figure 538. RSD Adder

Variable Gain Amplification (VGA) for Pre-gain

The VGA starts after sampling completes. It is enabled by a 2-bit signal PRE_GAIN described in *Section , Alternate Configuration 1-8 Control Registers (ADC_ACR1-8)*.

The ADC takes 2, 8, 64 or 128 clock cycles to do sampling which is selected by the LST[0:1] field in the conversion command message. After the sampling, if 2x VGA is enabled, there is a 2x gain stage without comparison before the regular conversion cycles. When 4x VGA is enabled, there are the 2x gain stage without comparison by 2 times before the normal conversion processing.

23.7 Initialization/Application information

23.7.1 Multiple queues control setup example

This section provides an example of how to configure multiple CQueues. *Table 548* describes how each CQueue can be used for a different application. Also documented in this section are general guidelines on how to initialize the on-chip ADCs and the external device, and how to configure the CQueues and the EQADC.

Table 548. Application of Each CQueue

CQueue Number	CQueue Type	Running Speed	Number of Contiguous Conversions	Example
0	Very fast burst time-based CQueue	every 2 µs for 200 µs; pause for 300 µs and then repeat	2	Injector current profiling
1	Fast hardware-triggered CQueue	every 900 µs	3	Current sensing of PWM controlled actuators
2	Fast repetitive time-based CQueue	every 2 ms	8	Throttle position

Table 548. Application of Each CQueue

CQueue Number	CQueue Type	Running Speed	Number of Contiguous Conversions	Example
3	Software-triggered CQueue	every 3.9 ms	3	Command triggered by software strategy
4	Repetitive angle-based CQueue	every 625 µs	7	Airflow read every 30 degrees at 8000 RPM
5	Slow repetitive time-based CQueue	every 100 ms	10	Temperature sensors

EQADC initialization

The following steps provide an example about how to configure the EQADC controls and how to initialize the on-chip ADCs and the external device. In this example, all conversion commands will be transferred through CFIFO0.

1. Load all required configuration commands in the RAM in such way that they form a queue; this data structure will be referred below as CQueue0. *Figure 539* shows an example of a CQueue able to configure the on-chip ADCs and external device at the same time. Although, this example uses the DMAC to store commands in CFIFO0, configuration commands could have also been directly written to the CFIFO0 push register.
2. Select source driving EQADC hardware trigger ports (ETRIG). Before proceeding to next step, allow some time (minimum of two system clocks - filter period is set to minimum after reset) so that the logic level at the source is filtered and reaches the EQADC control logic.

Note: ETRIG ports could be driven by an external pin or by the output port of other blocks in the device, such as timers. In order to avoid unexpected triggering of CFIFOs in hardware trigger modes, the source driving the ETRIG port must be selected and set to a known logic level before putting the CFIFOs into the WAITING FOR TRIGGER state.

The trigger filter bypass control inputs must be set considering the characteristics of the trigger signal. A particular case to assert the bypass control is when a device's internal signal with one clock width pulse is used.

3. Configure *Section , EQADC External Trigger Digital Filter Register (EQADC_ETDFR)*.
4. Configure *Section , EQADC null message send format register (EQADC_NMSFR)*.
5. Configure *Section , EQADC SSI Control Register (EQADC_SSICR)*, to communicate with the external device.
6. Enable the EQADC SSI by programming the ESSIE field in the *Section , EQADC Module Configuration Register (EQADC_MCR)*.
   a) Write 0b10 to ESSIE field to enable the EQADC SSI. FCK is free running but serial transmissions are not started.
   b) Wait until the external device becomes stable after reset.
   c) Write 0b11 to ESSIE field to enable the EQADC SSI to start serial transmissions.
7. Configure the DMAC to transfer data from CQueue0 to CFIFO0 in the EQADC.
8. Configure *Section , EQADC Interrupt and DMA Control Registers (EQADC_IDCR)*.
   a) Set CFFS0 to configure the EQADC to generate a DMA request to load commands from CQueue 0 to the CFIFO0.
   b) Set CFFE0 to enable the EQADC to generate a DMA request to transfer commands from CQueue0 to CFIFO0; Command transfers from the RAM to the CFIFO0 will start immediately.
   c) Set EOQIE0 to enable the EQADC to generate an interrupt after transferring all of the commands of CQueue0 through CFIFO0.
9. Configure *Section , EQADC CFIFO Control Registers (EQADC_CFCR)*.
   a) Write 0b0001 to the MODE0 field in EQADC_CFCR0 to program CFIFO0 for software single-scan mode.
   b) Write "1" to SSE0 to assert SSS0 and trigger CFIFO0.
10. Since CFIFO0 is in single-scan software mode and it is also the highest priority CFIFO, the EQADC starts to transfer configuration commands to the on-chip ADCs and to the external device.
11. When all of the configuration commands have been transferred, CF0 in *Section , EQADC FIFO and Interrupt Status Registers (EQADC_FISR)*, will be set. The EQADC generates a End of Queue interrupt. The initialization procedure is complete.

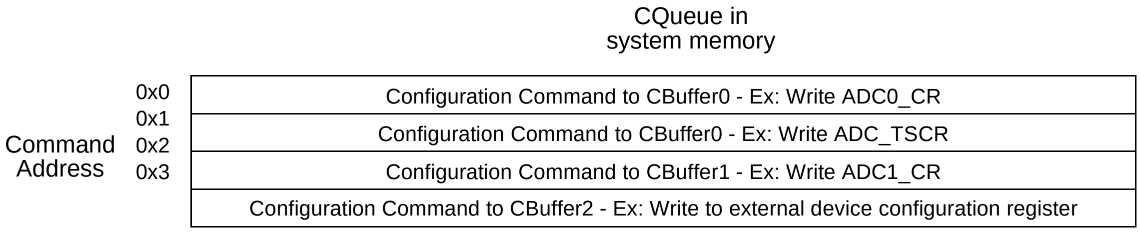


Figure 539. Example of a CQueue Configuring the On-Chip ADCs/External Device

The initialization procedure described above does not generate ADC clocks that are in phase because the timing at which the ADC0/1_EN bits, in the *Section , ADC0/1 Control Registers (ADC0_CR and ADC1_CR)*, are set is different. Below follows an example on how to simultaneously set these bits so that in-phase ADC clocks are generated. In this example, ADC0/1_CLK are configured to the same frequency.

1. Push an ADC0_CR write configuration command in CFIFO0 that enables ADC0 (ADC0_EN=1) and that sets the ADC0_CLK_PS to an appropriate value. For example, 0x80800801.
2. Push an ADC1_CR write configuration command in CFIFO1 that enables ADC1 (ADC1_EN=1) and that sets the ADC1_CLK_PS to an appropriate value. For example, 0x82800801.
3. Configure CFIFO0 and CFIFO1 to single scan software trigger mode and simultaneously trigger them by writing 0x04100410 to the EQADC_CFCR0 register - see *Section , EQADC CFIFO Control Registers (EQADC_CFCR)*.

Configuring EQADC for applications

This section provides an example based on the applications in *Table 548*. The example describes how to configure multiple CQueues to be used for those applications and provides a step-by-step procedure to configure the EQADC and the associated CQueue structures. In the example, the “Fast hardware-triggered CQueue”, described on the second row of *Table 548*, will have its commands transferred to CBuffer1; the conversion commands will be executed by ADC1. The generated results will be returned to RFIFO3 before being transferred to the RQueues in the RAM by the DMAC.

Note: There is no fixed relationship between CFIFOs and RFIFOs with the same number. The results of commands being transferred through CFIFO1 can be returned to any RFIFO, regardless of its number. The destination of a result is determined by the MESSAGE_TAG field of the command that requested the result. See Section , Message Format in EQADC, for details.

Step One: Setup the CQueues and RQueues.

1. Load the RAM with configuration and conversion commands. *Table 549* is an example of how CQueue1 commands should be set.
    a) Each trigger event will cause four commands to be executed. When the EQADC detects the Pause bit asserted, it will wait for another trigger to restart transferring commands from the CFIFO.
    b) At the end of the CQueue, the “EOQ” bit is asserted as shown in *Table 549*.
    c) Results will be returned to RFIFO3 as specified in the MESSAGE_TAG field of commands.
2. Reserve memory space for storing results.

Table 549. Example of CQueue Commands(1)

Bit #	0	1	2	3 4	5	6	7	8 9 10 11	12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
Bit Name	**EOQ**	**PAUSE**	**REP**	**RESERVED**	**EB**	**BN**	**RESERVED**	**MESSAGE TAG**	**ADC COMMAND**
CMD 1	0	0	0	0	0	1	0	0b0011	Conversion Command
CMD 2	0	0	0	0	0	1	0	0b0011	Conversion Command
CMD 3	0	0	0	0	0	1	0	0b0011	Conversion Command
CMD 4	0	1	0	0	0	1	0	0b0011 (2)	Configure peripheral device for next conversion sequence
CMD 5	0	0	0	0	0	1	0	0b0011	Conversion Command
CMD 6	0	0	0	0	0	1	0	0b0011	Conversion Command
CMD 7	0	0	0	0	0	1	0	0b0011	Conversion Command
CMD 8	0	1	0	0	0	1	0	0b0011 (2)	Configure peripheral device for next conversion sequence
				0			etc. ...		
CMD EOQ	1	0	0	0	0	1	0	0b0011	EOQ Message

CFIFO Header ADC Command

1. Fields LST, TSR, FMT, and CHANNEL_NUMBER are not showed for clarity. See *Section , Conversion Command Format for the Standard Configuration*, for details.
2. MESSAGE_TAG field is only defined for read configuration commands.

Step Two: Configure the DMAC to handle data transfers between the CQueues/RQueues in RAM and the CFIFOs/RFIFOs in the EQADC.

1. For transferring, set the source address of the DMAC to point to the start address of CQueue1. Set the destination address of the DMAC to point to EQADC_CFPR1. Refer to *Section , EQADC CFIFO Push Registers (EQADC_CFPR)*.
2. For receiving, set the source address of the DMAC to point to EQADC_RFPR3. Refer to *Section , EQADC Result FIFO Pop Registers (EQADC_RFPR)*. Set the destination address of the DMAC to point to the starting address of RQueue1.

Step Three: Configure the EQADC Control Registers.

1. Configure *Section , EQADC Interrupt and DMA Control Registers (EQADC_IDCR)*.
    a) Set EOQIE1 to enable the End of Queue Interrupt request.
    b) Set CFFS1 and RFDS3 to configure the EQADC to generate DMA requests to push commands into CFIFO1 and to pop result data from RFIF03.
    c) Set CFINV1 to invalidate the contents of CFIFO1.
    d) Set RFDE3 and CFFE1 to enable the EQADC to generate DMA requests. Command transfers from the RAM to the CFIFO1 will start immediately.
    e) Set RFOIE3 to indicate if RFIFO3 overflows.
    f) Set CFUIE1 to indicate if CFIFO1 underflows.
2. Configure MODE1 to continuous-scan rising edge external trigger mode in *Section , EQADC CFIFO Control Registers (EQADC_CFCR)*.

Step Four: Command transfer to ADCs and Result data reception.

When an external rising edge event occurs for CFIFO1, the EQADC automatically will begin transferring commands from CFIFO1 when it becomes the highest priority CFIFO trying to send commands to CBuffer1. The received results will be placed in RFIFO3 and then moved to RQueue1 by the DMAC.

23.7.2 EQADC/DMAC Interface

This section provides an overview about the EQADC/DMAC interface and general guidelines about how the DMAC should be configured in order for it to correctly transfer data between the queues in system memory and the EQADC FIFOs.

Note: *Advanced DMACs provide more functionality then the ones discussed in this section.*

CQueue/CFIFO transfers

In transfers involving CQueues and CFIFOs, the DMAC moves data from a queued source to a single destination as showed in *Figure 540*. The location of the data to be moved is indicated by the source address, and the final destination for that data, by the destination address. The DMAC contains a data structure containing these addresses and other parameters used in the control of data transfers. For every DMA request issued by the EQADC, the DMAC has to be configured to transfer a single command (32-bit data) from the CQueue, pointed to by the source address, to the CFIFO push register, pointed to by the destination address. After the service of a DMA request is completed, the source address has to be updated to point to the next valid command. The destination address remains unchanged. When the last command of a queue is transferred one of the following actions is recommended.

- The corresponding DMA channel should be disabled. This might be desirable for CFIFOs in single scan mode.
- The source address should be updated to pointed to a valid command which can be the first command in the queue that has just been transferred (cyclic queue), or the first command of any other CQueue. This is desirable for CFIFOs in continuous scan mode, and at some cases, for CFIFOs in single scan mode.

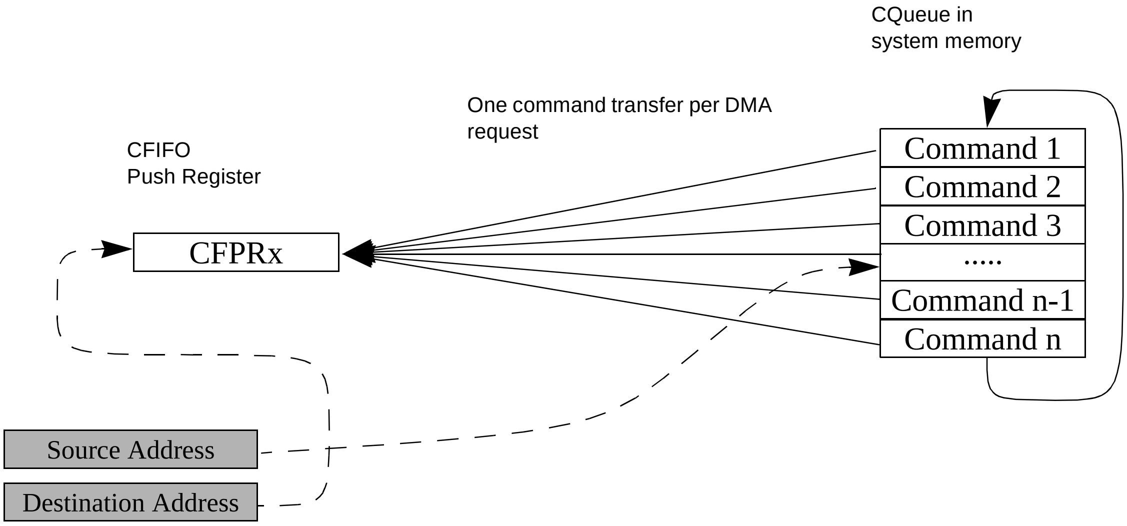


Figure 540. CQueue/CFIFO Interface

RQueue/RFIFO transfers

In transfers involving RQueues and RFIFOs, the DMAC moves data from a single source to a queue destination as showed in *Figure 541*. The location of the data to be moved is indicated by the source address, and the final destination for that data, by the destination address. For every DMA request issued by the EQADC, the DMAC has to be configured to transfer a single result (16-bit data), pointed to by the source address, from the RFIFO pop register to the RQueue, pointed to by the destination address. After the service of a DMA request is completed, the destination address has to be updated to point to the location where the next 16-bit result will be stored. The source address remains unchanged. When the last expected result is written to the RQueue, one of the following actions is recommended.

- The corresponding DMA channel should be disabled.
- The destination address should be updated pointed to the next location where new coming results are stored, which can be the first entry of the current RQueue (cyclic queue), or the beginning of a new RQueue.

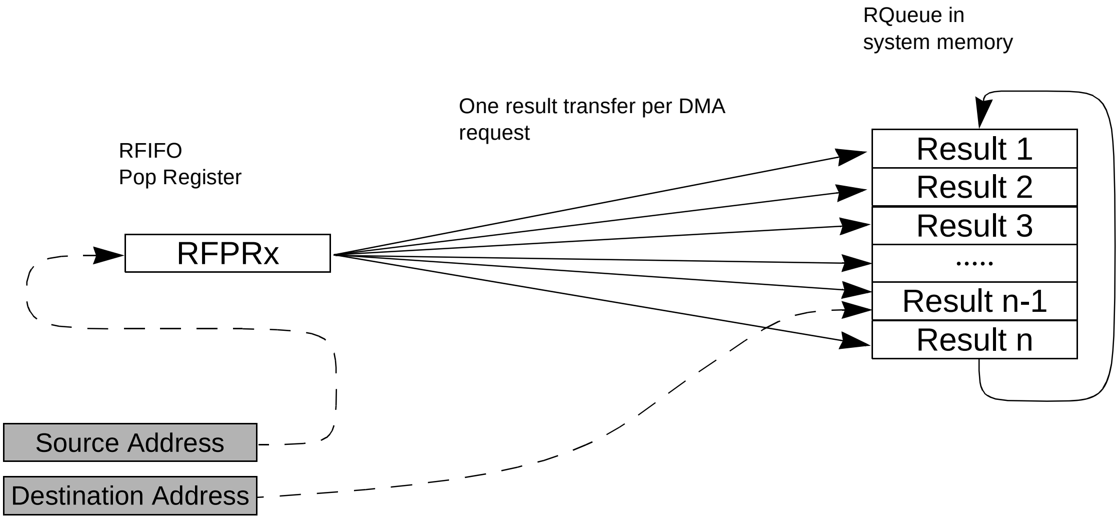


Figure 541. RQueue/RFIFO Interface

23.7.3 Sending immediate command setup example

In the EQADC, there is no immediate command register for sending a command immediately after writing to that register. However, a CFIFO can be configured to perform the same function as an immediate command register. The following steps illustrate how to configure CFIFO5 as an immediate command CFIFO. The results will be returned to RFIFO5.

1. Configure the *Section , EQADC Interrupt and DMA Control Registers (EQADC_IDCR)*.
   a) Clear CFIFO Fill Enable5 (CFFE5 = 0) in EQADC_IDCR2.
   b) Clear CFIFO Underflow Interrupt Enable5 (CFUIE5 = 0) in EQADC_IDCR2.
   c) Clear RFDS5 to configure the EQADC to generate interrupt requests to pop result data from RFIF05.
   d) Set RFIFO Drain Enable5 (RFDE5 = 1) in EQADC_IDCR2.
2. Configure the *Section , EQADC CFIFO Control Registers (EQADC_CFCR)*.
   a) Write “1” to CFINV5 in EQADC_FCR2. This will invalidate the contents of CFIFO5.
   b) Set MODE5 to Continuous-Scan Software Trigger mode in EQADC_CFCR2.
3. To transfer a command, write it to EQADC CFIFO Push Register 5 (EQADC_CFPR5) with Message Tag = 0b0101. Refer to *Section , EQADC CFIFO Push Registers (EQADC_CFPR)*.
4. Up to four commands can be queued in CFIFO5. Check the CFCTR5 status in EQADC_FISR5 before pushing another command to avoid overflowing the CFIFO. Refer to *Section , EQADC FIFO and Interrupt Status Registers (EQADC_FISR)*.
5. When the EQADC receives a conversion result for RFIFO5, it generates an interrupt request. RFIFO Pop Register 5 (EQADC_RFPR5) can be popped to read the result. Refer to *Section , EQADC Result FIFO Pop Registers (EQADC_RFPR)*.

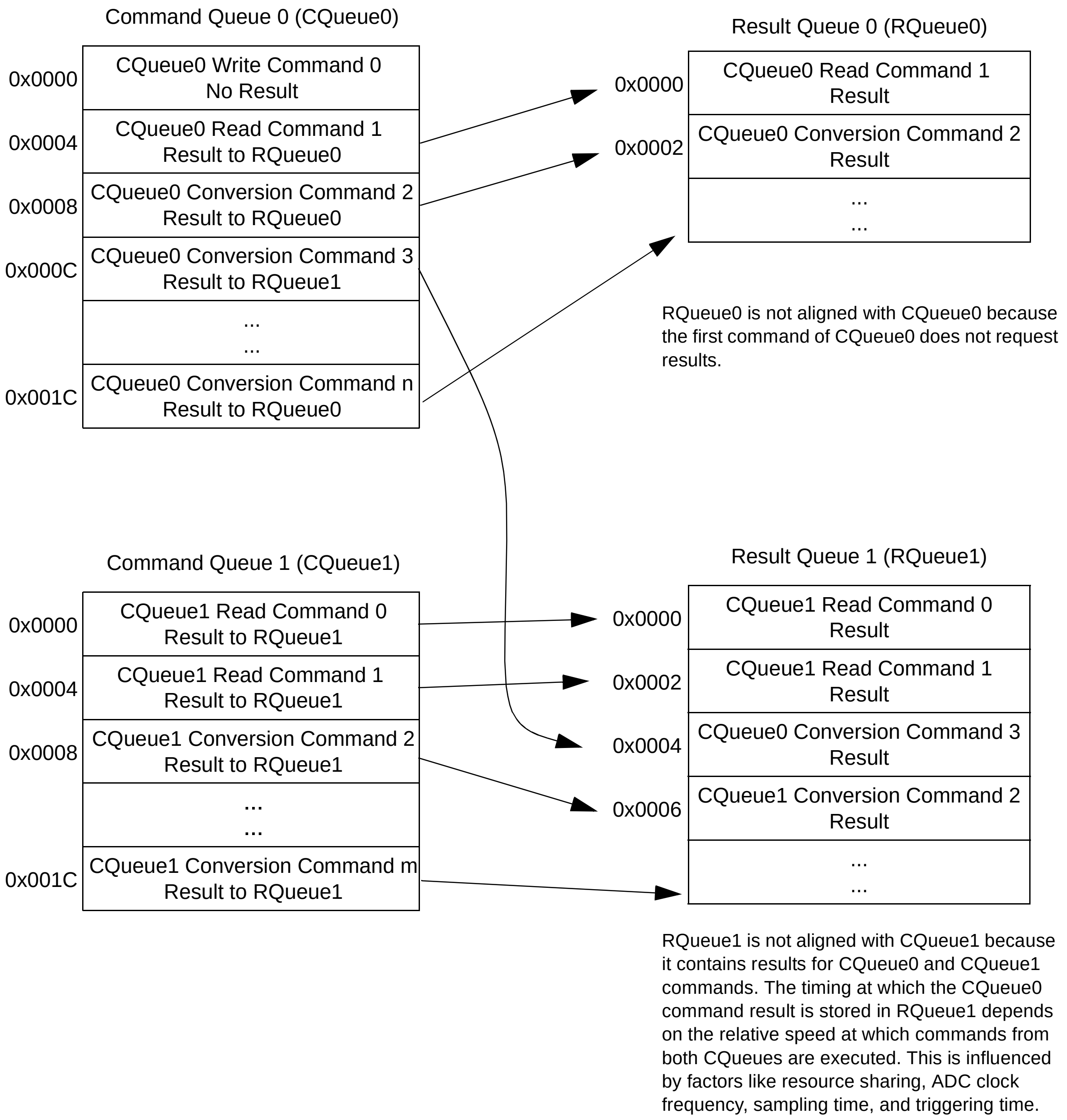
23.7.4 Modifying queues

More CQueues may be needed than the six supported by the EQADC. These additional CQueues can be supported by interrupting command transfers from a configured CFIFO, even if it is TRIGGERED and transferring, modifying the corresponding CQueue in the RAM or associating another CQueue to it, and restarting the CFIFO. More details on disabling a CFIFO are described in *Section , Disabled Mode*.

1. Determine the resumption conditions when later resuming the scan of the CQueue at the point before it was modified.
    a) Change MODEx in *Section , EQADC CFIFO Control Registers (EQADC_CFCR)*, to Disabled. Refer to *Section , Disabled Mode*, for a description of what happens when MODEx is changed to Disabled.
    b) Poll CFSx until it becomes IDLE in *Section , EQADC CFIFO Status Register (EQADC_CFSR)*.
    c) Read and save TC_CFx in *Section , EQADC CFIFO Transfer Counter Registers (EQADC_CFTCR)*, for later resuming the scan of the queue. The TC_CFx provides the point of resumption.
    d) Since all result data may not have being stored in the appropriate RFIFO at the time MODEx is changed to disable, wait for all expected results to be stored in the RFIFO/RQueue before reconfiguring the DMAC to work with the modified RQueue. The number of results that must return can be estimated from the TC_CFx value obtained above.
2. Disable the DMAC from responding to the DMA request generated by CFFFx and RFDFx in *Section , EQADC FIFO and Interrupt Status Registers (EQADC_FISR)*.
3. Write “0x0000” to the TC_CFx field.
4. Load the new configuration and conversion commands into RAM. Configure the DMAC to support the new CQueue/RQueue, but do not configure it yet to respond to DMA requests from CFIFOx/RFIFOx.
5. If necessary, modify *Section , EQADC Interrupt and DMA Control Registers (EQADC_IDCR)*, to suit the modified CQueue.
6. Write “1” to CFINVx in *Section , EQADC CFIFO Control Registers (EQADC_CFCR)*, to invalidate the entries of CFIFOx. Perform any other modifications to EQADC_CFCR except changing MODEx from Disabled.
7. Configure the DMAC to respond to DMA requests generated by CFFFx and RFDFx.
8. Change MODEx to the modified CFIFO operation mode. Write “1” to SSEx to trigger CFIFOx if MODEx is software trigger.

23.7.5 CQueue and RQueues usage

Figure 542 is an example of CQueue and RQueue usage. It shows the CQueue0 commands requesting results that will be stored in RQueue0 and RQueue1, and CQueue1 commands requesting results that will be stored only in RQueue1. Some Command Messages request data to be returned from the on-chip ADC/external device, but some only configure them and do not request returning data. When a CQueue contains both write and read commands like CQueue0, the CQueue and RQueue entries will not be aligned; as shown in *Figure 542*, the result for the second command of CQueue0 is the first entry of RQueue0. The figure also shows that CQueue and RQueue entries can also become unaligned even if all commands in a CQueue request data as CQueue1. CQueue1 entries became unaligned to RQueue1 entries because a result requested by the forth CQueue0 command was sent to RQueue1. This happens because the system can be configured so that several CQueues can have its results sent to a single RQueue.



RQueue0 is not aligned with CQueue0 because the first command of CQueue0 does not request results.

RQueue1 is not aligned with CQueue1 because it contains results for CQueue0 and CQueue1 commands. The timing at which the CQueue0 command result is stored in RQueue1 depends on the relative speed at which commands from both CQueues are executed. This is influenced by factors like resource sharing, ADC clock frequency, sampling time, and triggering time.

Figure 542. EQADC Command and Result Queues

23.7.6 ADC Result Calibration

The ADC result calibration process consists of two steps: determining the gain and offset calibration constants, and calibrating the raw results generated by the on-chip ADCs by solving the following equation discussed in *Section , ADC Calibration Feature*.

Equation 13 $CAL\_RES = GCC * RAW\_RES + OCC+2;$

The calibration constants GCC and OCC can be calculated from equation *Equation 13* provided that two pairs of expected (CAL_RES) and measured (RAW_RES) result values are available for two different input voltages. Most likely calibration points to be used are 25% VREF[ay] and 75% VREF since they are far apart but not too close to the end points of the full input voltage range. This allows for calculations of more representative calibration constants. The EQADC provides these voltages via channel numbers 43 and 44. The raw, uncalibrated results for these input voltages are obtained by converting these channels with conversion commands that have the CAL bit negated.

The transfer equations for when sampling these reference voltages are:

$CAL\_RES_{75\%VREF} = GCC * RAW\_RES_{75\%VREF} + OCC+2;$

$CAL\_RES_{25\%VREF} = GCC * RAW\_RES_{25\%VREF} + OCC+2;$

Thus;

Equation 14 $GCC = (CAL\_RES_{75\%VREF} - CAL\_RES_{25\%VREF}) / (RAW\_RES_{75\%VREF} - RAW\_RES_{25\%VREF});$

Equation 15 $OCC = CAL\_RES_{75\%VREF} - GCC*RAW\_RES_{75\%VREF} - 2;$

or

Equation 16 $OCC = CAL\_RES_{25\%VREF} - GCC*RAW\_RES_{25\%VREF} - 2;$

After being calculated, the GCC and OCC values must be written to ADC registers: *Section , ADC0/1 Gain Calibration Constant Registers (ADC0_GCCR and ADC1_GCCR)*, and *Section , ADC0/1 Offset Calibration Constant Registers (ADC0_OCCR and ADC1_OCCR)*, using write configuration commands.

The EQADC will automatically calibrate the results, according to equation *Equation 13*, of every conversion command that has its CAL bit asserted using the GCC and OCC values stored in the ADC calibration registers.

Note: For accurate calibration, the 25% VREF channel must be converted using the Long Sample Time (LST) setting for either 64 or 128 ADC sample cycles in the ADC Conversion Command Message (LST = 0b10 or 0b11).

MAC Configuration Procedure

The following steps illustrate how to configure the calibration hardware, namely, determining the values of the gain and offset calibration constants, and the writing of these constants to the calibration registers. The procedure below should be performed for ADC0 and for ADC1.

[ay] VREF=VRH-VRL

1. Convert channel 44 with a command that has its CAL bit negated and obtain the raw, uncalibrated result for 25%VREF ($RAW\_RES_{25\%VREF}$).
2. Convert channel 43 with a command that has its CAL bit negated and obtain the raw, uncalibrated result for 75%VREF ($RAW\_RES_{75\%VREF}$).
3. Since the expected values for the conversion of these voltages are known ($CAL\_RES_{25\%VREF}$ and $CAL\_RES_{75\%VREF}$), GCC and OCC values can be calculated from equations *Equation 14* and *Equation 15* using these values, and the ones determined in steps 1 and 2.
4. Reformat GCC and OCC to the proper data formats as specified in *Section , MAC Unit and Operand Data Format*. GCC is an unsigned 15-bit fixed point value and OCC is a signed 14-bit value.
5. Write GCC value to *Section , ADC0/1 Gain Calibration Constant Registers (ADC0_GCCR and ADC1_GCCR)*, and OCC value to *Section , ADC0/1 Offset Calibration Constant Registers (ADC0_OCCR and ADC1_OCCR)*, using write configuration commands.

Example

The raw results obtained when sampling reference voltages 25%VREF and 75%VREF were, respectively, 3798 and 11592. The results that should have been obtained from the conversion of these reference voltages are, respectively, 4096 and 12288. Therefore, using equations *Equation 14* and *Equation 15*, the gain and offset calibration constants are:

GCC=(12288-4096)/(11592-3798) = 1.05106492-> 1.05102539 = 0x4388

OCC=12288 - 1.05106492*11592 - 2 = 102.06-> 102 = 0x0066

Table 550 shows, for this particular case, examples of how the result values change according to GCC and OCC when result calibration is executed (CAL=1) and when it is not (CAL=0).

Table 550. Calibration example

Input Voltage	Raw result (CAL=0)		Calibrated result (CAL=1)	
	Hexadecimal	Decimal	Hexadecimal	Decimal
25% VREF(1)	0x0ED6	3798	0x1000	4095.794
75% VREF	0x2D48	11592	0x3000	12287.486

1. For accurate calibration, the 25% VREF channel must be converted using the Long Sample Time (LST) setting for either 64 or 128 ADC sample cycles in the ADC Conversion Command Message (LST = 0b10 or 0b11).

Quantization Error Reduction During Calibration

Figure 543 shows how the ADC transfer curve changes due to the addition of two to the MAC output during the calibration - see MAC output equation at *Section , Overview*. The maximum absolute quantization error is reduced by half leading to an increase in accuracy.

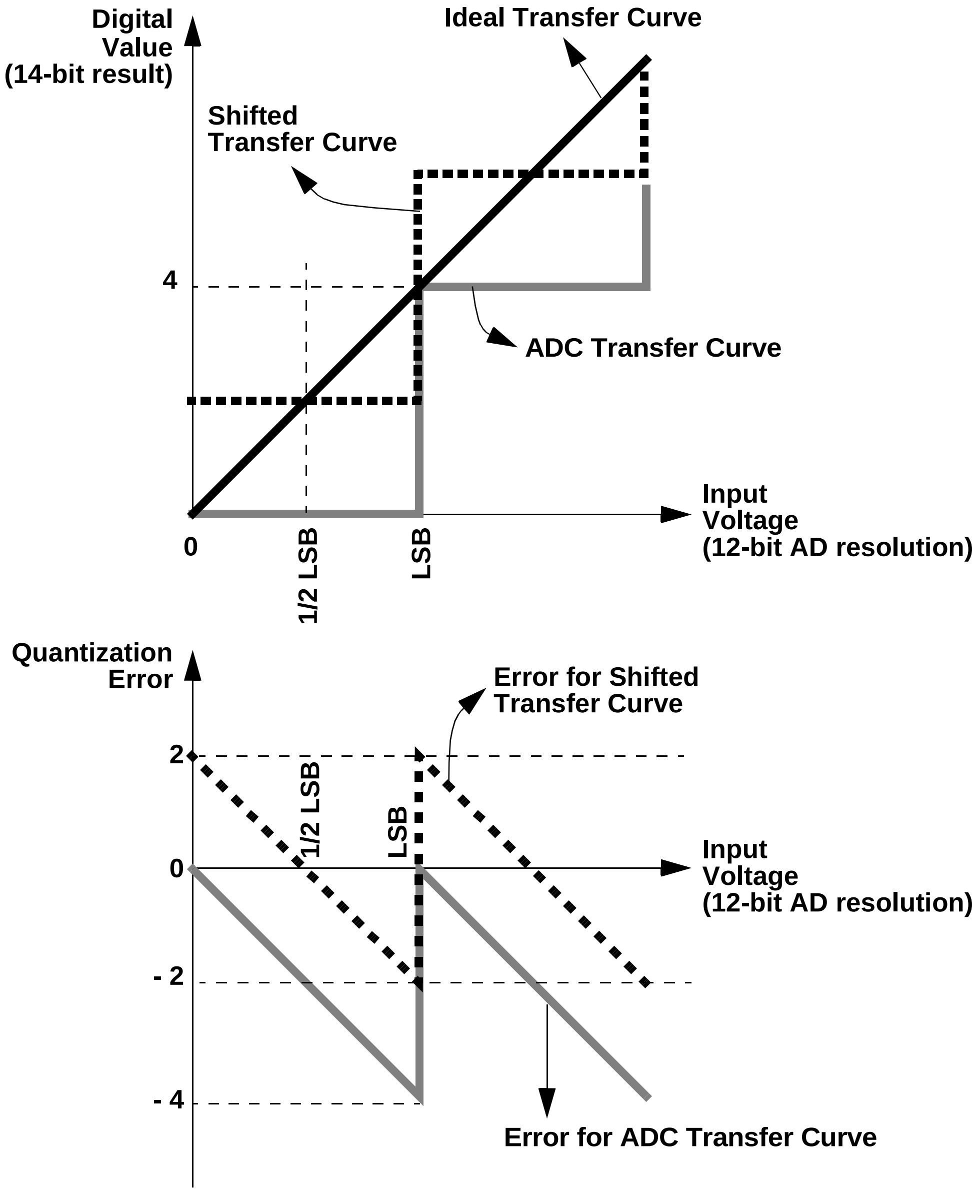


Figure 543. Quantization error reduction during calibration

23.7.7 EQADC versus QADC

This section describes how the EQADC upgrades the QADC functionality. The section also provides a comparison between the EQADC and QADC in terms of their functionality. This section targets the users familiar with terminology in QADC. *Figure 544* is an overview of a QADC. *Figure 545* is an overview of the EQADC system.

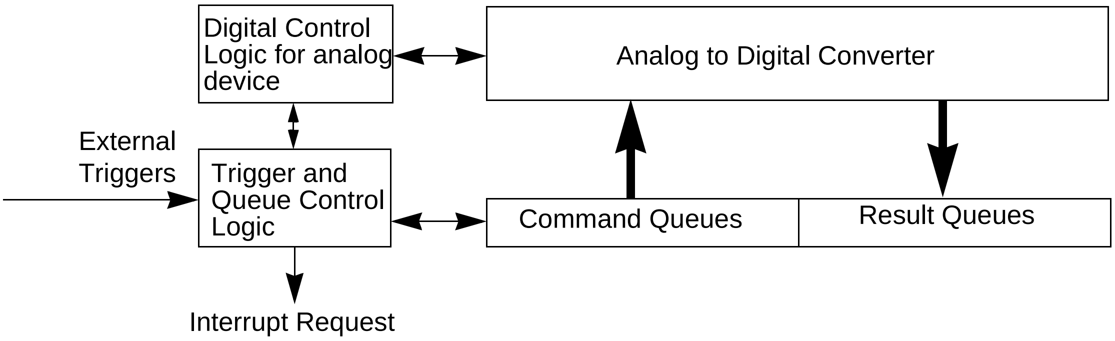


Figure 544. QADC Overview

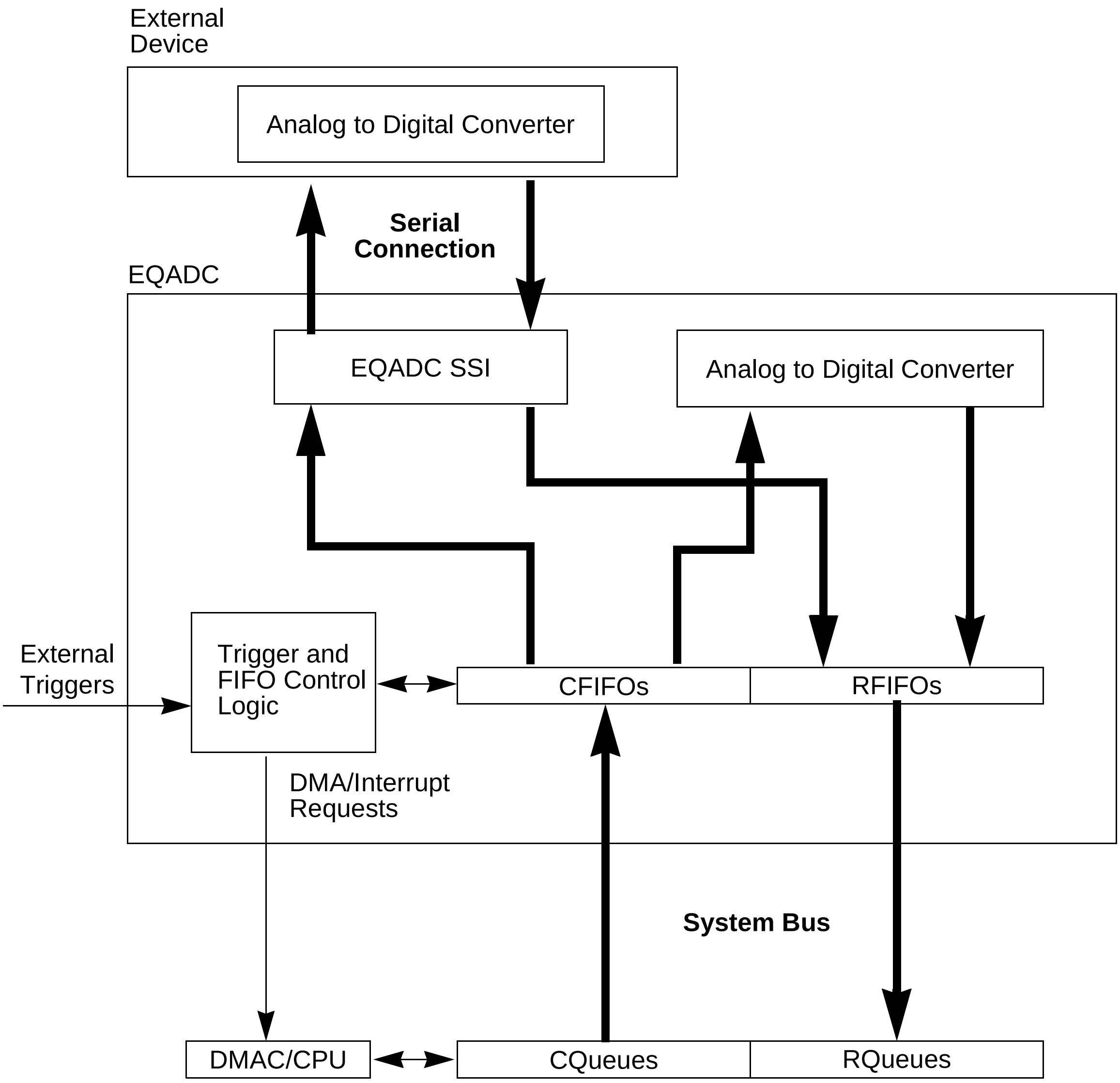


Figure 545. EQADC System Overview

The EQADC system consists of four parts: queues in RAM, the EQADC, on-chip ADCs, and an external device. As compared with the QADC, the EQADC system requires two pieces of extra hardware.

1. A DMA or an MCU is required to move data between the EQADC's FIFOs and Queues in the system memory.
2. A serial interface (EQADC Synchronous Serial Interface - EQADC SSI) is implemented to transmit and receive data between the EQADC and the external device.

Since there are only FIFOs inside the EQADC, much of the terminology or use of the register names, register contents, and signals of the EQADC involve "FIFO" instead of "Queue". These register names, register contents, and signals are functionally equivalent to the "Queue" counterparts in the QADC. *Table 551* lists how the EQADC register, register contents, and signals are related to QADC.

Table 551. Terminology Comparison between QADC and EQADC

QADC Terminology	EQADC Terminology	Function
CCW	Command Message	In the QADC, the hardware only executes conversion command words. In the EQADC, not all commands are conversion commands; some are configuration commands.
Queue Trigger	CFIFO Trigger	In the QADC, a trigger event is required to start the execution of a queue. In the EQADC, a trigger event is required to start command transfers from a CFIFO. When a CFIFO is TRIGGERED and transferring, commands are continuously moved from CQueues to CFIFOs. Thus, the trigger event initiates the “execution of a queue” indirectly.
Current Word Pointer Queue x (CWPQx)	Counter Value of Commands Transferred from Command FIFOx (TC_CFx)	In the QADC, CWPQx allows the last executed command on queue x to be determined. In the EQADC, the TC_CFx value allows the last transferred command on CQueue x to be determined.
Queue Pause Bit (P)	CFIFO Pause Bit	In the QADC, detecting a pause bit in the CCW will pause the queue execution. In the EQADC, detecting a pause bit in the Command will pause command transfers from a CFIFO.
Queue Operation Mode (MQx)	CFIFO Operation Mode (MODEx)	The EQADC supports all queue operation modes in the QADC except operation modes related to a periodic timer. A timer elsewhere in the system can provide the same functionality if it is connected to ETRIGx.
Queue Status (QS)	CFIFO Status (CFSx)	In the QADC, the Queue Status is read to check whether a queue is idle, active, paused, suspended, or trigger pending. In the EQADC, the CFIFO Status is read to check whether a queue is IDLE, WAITING FOR TRIGGER (idle or paused in QADC), or TRIGGERED (suspended or trigger pending in QADC). What CFIFO is currently “active” can be determined by reading the LCFTCBz field on the EQADC_CFSSR registers.

The EQADC and QADC also have similar procedures for the configuration or execution of applications. *Table 552* shows the steps required for the QADC versus the steps required for the EQADC system.

Table 552. Usage Comparison between QADC and EQADC System

Procedure	QADC	EQADC System
Analog Control Configuration	Configure analog device by writing to the QADC registers.	Program configuration commands into command queues.
Prepare Scan Sequence	Program scan commands into command queues.	Program scan commands into command queues.
Queue Control Configuration	Write to the QADC Control Registers.	Write to the EQADC Control Registers.
Data Transferred between Queues and Buffers	Not Required.	Program the DMAC or the CPU to handle the data transfer.

Table 552. Usage Comparison between QADC and EQADC System (continued)

Procedure	QADC	EQADC System
Serial Interface Configuration	Not Required.	Write to the EQADC SSI Registers.
Queue Execution	Require Software or External Trigger events to start queue execution.	Require Software or External Trigger events to start command transfers from a CFIFO.

24 System Information Module and Trim (SIM)

24.1 Overview

The System Information Module loads configuration data for the device, and trims data used by various analog IP blocks to calibrate current/voltage references or other tunable circuits from flash module test rows.

Configuration data is maintained by the System Information Module. Some, but not all, of the data is readable by the user.

24.2 User trim values

Table 553 specifies the SPC564A70 microcontroller temperature sensor calibration values from address offset 0x00 – 0x3C.

Note: *The SIM base address is 0xFFFE_C000 (same as the temperature sensor base). The temperature sensor calibration values are available as for read-only access.*

Table 553. User trim values

<table>
<tr><th>Offset from TSENS_Base</th><th>Bits 0:15</th><th>Bits 16:31</th></tr>
<tr><td>0x00</td><td>Temperature sensor calibration(1) 16 bits</td><td>Temperature sensor calibration(1) 16 bits</td></tr>
<tr><td>0x04</td><td>Temperature sensor calibration(1) 16 bits</td><td>Reserved</td></tr>
<tr><td>0x08</td><td colspan="2" rowspan="2">Reserved</td></tr>
<tr><td>0x0C</td></tr>
<tr><td>0x10</td><td colspan="2" rowspan="4">Unique device ID(2)</td></tr>
<tr><td>0x14</td></tr>
<tr><td>0x18</td></tr>
<tr><td>0x1c</td></tr>
<tr><td>0x20 – 0x3C</td><td colspan="2">Reserved</td></tr>
</table>

1. The temperature sensor calibration values, are available as a read-only slave-bus access in user mode. See the temperature sensor chapter for further details.
2. The unique device ID is a serial number that is different for every device manufactured.

25 Decimation Filter

25.1 Information specific to this device

This section presents device-specific parameterization and customization information not specifically referenced in the remainder of this chapter.

different ADC setup configurations which are identified by an specific eQADC Control address within a conversion command. Conversion commands with Register Address set to zero use the standard configuration setup. The samples generated by the standard configuration setup are sent to one of the local eQADC RFIFO buffers. The samples generated by the Alternate Configurations, with address from 1 to 6, can be sent to the internal RFIFO or to the eQADC dedicated slave-bus interface to communicate with the external Decimation Filter IP block or any other block that can communicate with this interface. A bit field in the Alternate Configuration Control Register selects the Internal RFIFO or this slave-bus interface as the destination for the conversion result.

25.1.1 Device-specific parameters

MDIS_DEFAULT resets MDIS bit in Decimation Filter Module Configuration Register to 0.

Table 554. Decimation Filter Parameters for SPC564A70

Parameter Name	Description	Value
MDIS_DEFAULT	DECFILTER_MCR[31], MDIS reset state	0

25.1.2 Device-specific features

Two Decimation Filter modules are implemented in SPC564A70. The Decimation filter is used to decimate conversion results from the eQADC block. A dedicated slave-bus interface provides bidirectional communication between the eQADC and Decimation filters. Another slave-bus interface is also provided for setting up the filter parameters and configuration registers. The Decimation Filter receives conversion results generated by the eQADC block. These results can be generated from eight different ADC setup configurations which are identified by an specific eQADC Control address within a conversion command. Conversion commands with Register Address set to zero use the standard configuration setup. The samples generated by the standard configuration setup are sent to one of the local eQADC RFIFO buffers. The samples generated by the Alternate Configurations, with address from 1 to 6, can be sent to the internal RFIFO or to the eQADC dedicated slave-bus interface to communicate with the external Decimation Filter IP block or any other block that can communicate with this interface. A bit field in the Alternate Configuration Control Register selects the Internal RFIFO or this slave-bus interface as the destination for the conversion result.

25.1.3 Device-specific parameters

MDIS_DEFAULT resets MDIS bit in Decimation Filter Module Configuration Register to 0.

Table 555. Decimation filter parameters for SPC564A70

Parameter Name	Description	Value
MDIS_DEFAULT	DECFILTER_MCR[31], MDIS reset state	0

25.2 Introduction

25.2.1 Overview

The decimation filter is a dedicated hardware block, designed to decimate fixed point sample conversion results, generated by master block, usually an eQADC. A dedicated parallel side interface (PSI) provides bi-directional communication between the master block and the filter. A second interface is provided for use by the CPU, allowing setup of the filter parameters and read/write of the configuration registers.

The Decimation Filter receives data samples from the master block (eQADC) in the PSI RX sub-block. Each sample arrives at the decimation filter with an identifier tag and associated commands. The input information is decoded by the PSI RX and control logic sub-blocks. When receiving a filtering command, the data is transferred to the filter tap register's sub-block and is processed by the filter using the MAC, the coefficient register, and the control logic sub-blocks. Then the result is returned to the master block by the PSI TX sub-block. This result is accompanied by the corresponding tag information that provides an address for the data.

To summarize normal mode, the decimation filter works as a slave block on this second slave-bus line, and there is a PSI master block such as the eQADC to send and read data. This is illustrated in the application example in *Section 25.7, Application information*.

The decimation filter can also work in a standalone mode. In this mode, the input data is supplied and the output results are read by the chip core processor (CPU) using status and interrupt signals or DMA requests.

Mixed modes are also provided, allowing input data fed from the PSI interface, and output results read by the CPU or DMA, or input fed from CPU or DMA and output directed to the PSI interface.

An integrator unit independently accumulates the values of filter outputs. The integration can be restricted to time windows defined by hardware signals or software.

Two or more decimation filter blocks can also be configured to work in the cascade mode of operation to obtain a more complex filtering function. The output result of a filter block is connected to the input of the next filter by means of a dedicated interface.

All signals in the interface are generated in the system clock domain.

Figure 546 is the block diagram for the decimation filter.

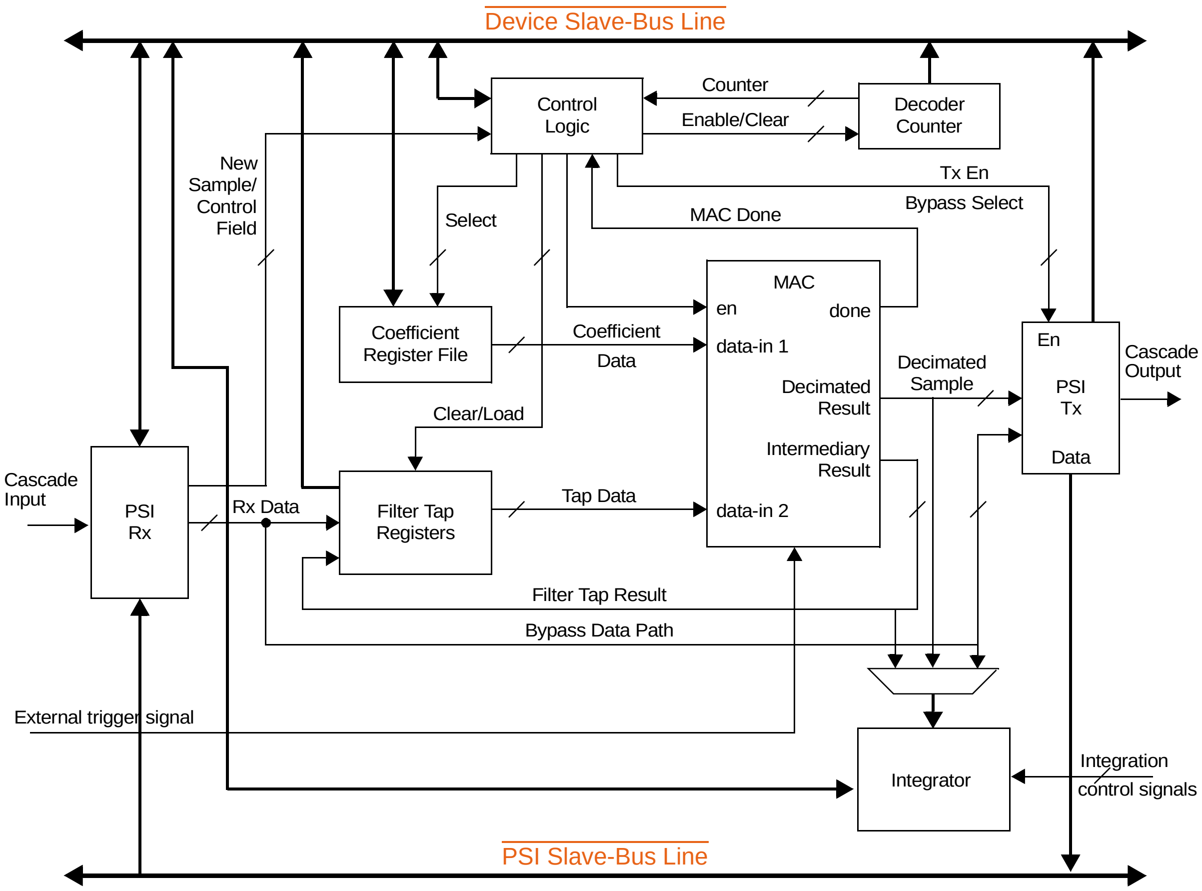


Figure 546. Decimation filter block diagram

25.2.2 Features

The Decimation Filter block includes these distinctive features:

- Selectable 4th order IIR filter, or an 8th order FIR filter
  - Input/output with 16-bit (fixed point) two’s complement signed values
  - Internal taps with 16-bit (feed-forward portion of first IIR) and 24-bit (feedback portion) resolutions (fixed point) for two’s complement signed value
  - 24-bit programmable filter coefficients (fixed point) for two’s complement signed value
  - MAC unit with 51-bit fixed point accumulator
  - Convergent rounding methodology
  - Two’s complement overflow or saturation selection
  - 58 clock cycles to process the input
- Implements a local slave-bus interface to a master block (e.g. the eQADC block)
- Input and output buffers with DMA capability
- Slave-bus interface to device
- Filter taps access for debug
- Filter initialization (flush) and stabilization (prefill) commands
- Timestamp support
- Decimation controlled by an internal counter or from an on-chip independent trigger signal (triggered output result)
- Integrator unit accumulates filter output values, signaled or absolute, with 32-bit resolution. The integrator can be controlled by software or hardware signals.
- Cascade of 2 or more individual blocks to compose a more complex filter

25.2.3 Modes of operation

This section describes the operation modes of the Decimation Filter. The modes are selected using the DECFILTER_MCR fields MDIS, FREN, FRZ, ISEL, MIXM, and CASCD (see *Section , Decimation Filter Module Configuration Register (DECFILTER_MCR)*). The mode selection is summarized in *Table 556*.

Table 556. Operation mode selection

Mode	MDIS	FREN, FRZ	ISEL	MIXM	CASCD
Normal	0	(0, 0) or (0, 1) or (1, 0)	0	0	00
Standalone			1	0	
PSI Input Mixed			0	1	
PSI Output Mixed			1	1	
Cascade			0 or 1	0	01 or 10 or 11
Freeze(1)		1, 1	X	X	X
Low Power	1	X	X	X	X

1. Freeze mode can also be activated from outside the Decimation Filter, depending on the MCU, if FREN = 1.

Normal mode

This is the default operational mode of the decimation filter block. It corresponds to the prefill/filter operation with input data supplied through the PSI slave-bus interface (i.e. its input data is the ADC conversion result), with output going to the same PSI interface.

Standalone mode

Standalone mode differs from normal mode because the input data is not supplied by the master block through the PSI slave-bus interface. In this case, the data is provided by the central processor using the device slave-bus interface or DMA interface signals. Once the data is filtered the decimated result is available in the Output Buffer register. The filter output is also consumed by a CPU or DMA mastering the same device slave-bus interface. This operation mode can be used to debug the filter stability or to decimate data in System RAM.

PSI Input Mixed mode

In this mode the input is selected from the PSI slave-bus interface, but the output is directed to the device slave-bus interface, where it can be read by the CPU or DMA.

PSI Output Mixed mode

This mode works inverted from the PSI Input Mixed Mode: the input is selected from the device slave-bus interface, fed either by the CPU or DMA, and the output is directed to the PSI slave-bus interface. If an eQADC is connected to the PSI interface, the output is directed to an RFIFO selected by the tag field in the DECFILTER_IB register (see *Section , Decimation Filter Interface Input Buffer Register (DECFILTER_IB)*).

Cascade mode

Cascade mode is a filter structure mode with two or more individual filter blocks connected in a chain to form a more complex filter function. The output result of the first block (head block) is connected to the input of the next block (middle or tail block) to be filtered again. More details in *Section 25.5.16, Cascade mode description*.

Low Power mode

Low power mode corresponds to the module disable mode or stop mode. In the module disable mode the PSI slave-bus line is disabled and it is not possible to enter Freeze mode. The system clock is stopped. And in stop mode, the system clock is also stopped.

Freeze mode

This mode is also known as debug mode. All filter action is frozen, either through software or by the hardware SoC debug request signal. If a freeze request comes when the filter is processing an input, it enters freeze mode only after the processing finishes.

25.3 External signal description

Note: The Decimation Filter does not provide metastability protection nor filtering for these signals.

25.3.1 Decimation trigger signal

This signal is used to control the output of the decimation filter, allowing decimation to be driven externally. For more details, see *Section , Triggered output result description*.

25.3.2 Integrator enable signal

This signal is used to enable the hardware integrator. For more details, see *Section , Integrator enabling and halting*.

25.3.3 Integrator halt signal

This signal is used to halt the hardware integrator. For more details, see *Section , Integrator enabling and halting*.

25.3.4 Integrator reset signal

This signal is used to reset the hardware integrator. For more details, see *Section , Integrator reset*.

25.3.5 Integrator output request signal

This signal is used to update the integrator output result. For more details, see *Section , Integrator outputs*.

25.4 Memory map and register definition

This section provides the memory maps and detailed descriptions of all registers. Accesses to reserved areas of the memory map can return a bus error, depending on the device integration.

This module communicates with two distinct slave-bus lines. One is related to the device integration and the second is related to a PSI master block for data transfers. Below both memory maps are described.

25.4.1 Decimation filter device memory map

The addresses of the Decimation Filter registers are specified as offsets from the module’s base address, described in *Table 557*. The registers allocated in this memory map are sufficient for a 4th order IIR filter implementation.

Table 557. Decimation filter device memory map

Offset from DECFILTER_ BASE 0xFFF8_8000 (Filter A) **0xFFF8_C000 (Filter B)**	**Register**	**Location**
0x000	Decimation Filter Module Configuration Register (DECFILTER_MCR)	*on page 25-1028*
0x004	Decimation Filter Module Status Register (DECFILTER_MSR)	*on page 25-1034*
0x008	Decimation Filter Module Extended Configuration Register (DECFILTER_MXCR)	*on page 25-1037*
0x00C	Decimation Filter Module Extended Status Register (DECFILTER_MXSR)	*on page 25-1041*

Table 557. Decimation filter device memory map (continued)

Offset from DECFILTER_BASE 0xFFF8_8000 (Filter A) 0xFFF8_C000 (Filter B)	Register	Location
0x010	Decimation Filter Interface Input Buffer Register (DECFILTER_IB)	*on page 25-1043*
0x014	Decimation Filter Interface Output Buffer Register (DECFILTER_OB)	*on page 25-1044*
0x018–0x01F	Reserved	
0x020	Decimation Filter Coefficient 0 Register (DECFILTER_COEF0)	*on page 25-1045*
0x024	Decimation Filter Coefficient 1 Register (DECFILTER_COEF1)	
0x028	Decimation Filter Coefficient 2 Register (DECFILTER_COEF2)	
0x02C	Decimation Filter Coefficient 3 Register (DECFILTER_COEF3)	
0x030	Decimation Filter Coefficient 4 Register (DECFILTER_COEF4)	
0x034	Decimation Filter Coefficient 5 Register (DECFILTER_COEF5)	
0x038	Decimation Filter Coefficient 6 Register (DECFILTER_COEF6)	
0x03C	Decimation Filter Coefficient 7 Register (DECFILTER_COEF7)	
0x040	Decimation Filter Coefficient 8 Register (DECFILTER_COEF8)	
0x044–0x077	Reserved	
0x078	Decimation Filter TAP0 Register (DECFILTER_TAP0)(1)	*on page 25-1045*
0x07C	Decimation Filter TAP1 Register (DECFILTER_TAP1)	
0x080	Decimation Filter TAP2 Register (DECFILTER_TAP2)	
0x084	Decimation Filter TAP3 Register (DECFILTER_TAP3)	
0x088	Decimation Filter TAP4 Register (DECFILTER_TAP4)	
0x08C	Decimation Filter TAP5 Register (DECFILTER_TAP5)	
0x090	Decimation Filter TAP6 Register (DECFILTER_TAP6)	
0x094	Decimation Filter TAP7 Register (DECFILTER_TAP7)	
0x098–0x0CF	Reserved	
0x0D0	Decimation Filter Interface Enhanced Debug Input Data Register (DECFILTER_EDID)	*on page 25-1046*
0x0D4–0x0DF	Reserved	

Table 557. Decimation filter device memory map (continued)

Offset from DECFILTER_ BASE 0xFFF8_8000 (Filter A) 0xFFF8_C000 (Filter B)	Register	Location
0x0E0	Decimation Filter Final Integration Value Register (DECFILTER_FINTVAL)	*on page 25-1046*
0x0E4	Decimation Filter Final Integration Count Value Register (DECFILTER_FINTCNT)	*on page 25-1047*
0x0E8	Decimation Filter Current Integration Value Register (DECFILTER_CINTVAL)	*on page 25-1048*
0x0EC	Decimation Filter Current Integration Count Value Register (DECFILTER_CINTCNT)	*on page 25-1048*
0x0F0–0x1FF	Reserved	

1. The TAP register stores, on each filter node, the input sample data and, for the IIR type, the filter intermediary results.

25.4.2 Decimation filter register descriptions

All registers are 32-bit wide.

Decimation Filter Module Configuration Register (DECFILTER_MCR)

The Decimation Filter module configuration register provides configuration control bits for the Decimation Filter internal logic.

Note: One must not modify this register contents when the status bit BSY is set, except for fields FREN, FRZ and IDIS. To guarantee that BSY does not set during the read-modify-write operation, it is advisable to set IDIS = 1 and wait for BSY = 0 beforehand.

Figure 547. Decimation Filter Module Configuration Register (DECFILTER_MCR)

Address: DECFILTER_BASE + 0x000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MDIS	FREN	0	FRZ	0	CASCD[1:0]		IDEN	ODEN	ERREN	0	FTYPE[1:0]		0	SCAL[1:0]	
W					SRES											
Reset	—(1)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	IDIS	SAT	ISEL	MIXM	DEC_RATE[3:0]				SDIE	DSEL	IBIE	OBIE	EDME	TORE	TMODE[1:0]	
W																
Reset	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1. Reset value is defined by the MDIS_DEFAULT parameter value.

Table 558. DECFILTER_MCR Register Field Descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>0
MDIS</td><td>Module Disable. The MDIS bit puts the Decimation Filter in low power mode. Communication through the PSI slave-bus Interface is ignored in this mode. Writes to the configuration register are allowed with the exception of writes to the FREN and SRES bits, which are ignored. Writes to the Coefficient registers are also allowed. The Decimation Filter cannot enter Freeze mode once in disable mode. Once the module is disabled it no longer receives the system clock.
1 Low Power Mode
0 Normal Mode</td></tr>
<tr><td>1
FREN</td><td>Freeze Enable. The FREN bit enables the Decimation Filter to enter freeze mode if the SoC debug request signal or the FRZ bit is asserted. See <i>Section 25.5.13, Freeze mode description</i>, for more details.
1 Decimation Filter Freeze mode enabled
0 Decimation Filter Freeze mode disabled</td></tr>
<tr><td>2</td><td>Reserved, should be cleared.</td></tr>
<tr><td>3
FRZ</td><td>Freeze Mode
The FRZ bit controls the freeze mode of the Decimation Filter. For this bit to take effect the FREN freeze enable bit also needs to be asserted. While in freeze mode the MAC operations are halted. See <i>Section 25.5.13, Freeze mode description</i>, for more details.
1 Decimation Filter in Freeze Mode
0 Decimation Filter in Normal Mode</td></tr>
<tr><td>4
SRES</td><td>Software-reset bit
The SRES is a self-negated bit which provides the CPU with the capability to initialize the Decimation Filter through the slave-bus interface. This bit always reads as zero. See <i>Section 25.5.10, Soft-reset command description</i>, for more details.
1 Software-Reset
0 No action</td></tr>
<tr><td>5–6
CASCD[1:0]</td><td>Cascade Mode Configuration. The CASCD[1:0] bit field configures the block to work in cascade mode of operation according to <i>Table 559</i>. For more details about the cascade mode, see <i>Section 25.5.16, Cascade mode description</i>.
Table 559. CASCD[1:0] – Filter Cascade mode configuration selection
<table>
<tr><th>CASCD[1:0]</th><th>Description</th></tr>
<tr><td>00</td><td>No cascade mode (single block)</td></tr>
<tr><td>01</td><td>Cascade Mode, Head block configuration</td></tr>
<tr><td>10</td><td>Cascade Mode, Tail block configuration</td></tr>
<tr><td>11</td><td>Cascade Mode, Middle block configuration</td></tr>
</table>
Any change to this field must follow the procedure described in the <i>Section , Cascade freeze, stop, and configuration change procedures</i>.</td></tr>
<tr><td>IDEN
7</td><td>Input Data Interrupt Enable. The IDEN bit enables the Decimation Filter to generate interrupt requests on all new input data written to the Interface Input Buffer register or Input/Output Buffers register.
1 Input Data Interrupt Enabled
0 Input Data Interrupt Disabled</td></tr>
</table>

Table 558. DECFILTER_MCR Register Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>ODEN
8</td><td>Output Data Interrupt Enable. The ODEN bit enables the Decimation Filter to generate interrupt requests on all new data written to the filter Output buffer. It is independent of how ISEL and MIXM are set.
1 Output Data Interrupt Enabled
0 Output Data Interrupt Disabled</td></tr>
<tr><td>ERREN
9</td><td>Error Interrupt Enable. The ERREN bit enables the Decimation Filter to generate interrupt requests based on the assertion of the DECFILTER_MSR error flags OVF, DIVR, SVR, OVR or IVR.
1 Error Interrupts Enabled
0 Error Interrupts Disabled</td></tr>
<tr><td>10</td><td>Reserved, should be cleared.</td></tr>
<tr><td>FTYPE[1:0]
11–12</td><td>Filter Type Selection bits. The FTYPE[1:0] bits select the filter type according to <i>Table 560</i>.
Table 560. FTYPE[1:0] – Filter type selection
<table>
<tr><th>FTYPE[1:0]</th><th>Description</th></tr>
<tr><td>00</td><td>Filter Bypass(1)</td></tr>
<tr><td>01</td><td>IIR Filter - 1 x 4th order</td></tr>
<tr><td>10</td><td>FIR Filter - 1 x 8th order</td></tr>
<tr><td>11</td><td>reserved</td></tr>
</table>
1. In Bypass configuration the filter is disabled.
Bypass must not be configured in cascade mode (see field CASCD).</td></tr>
<tr><td>13</td><td>Reserved, should be cleared.</td></tr>
<tr><td>14–15
SCAL[1:0]</td><td>Filter Scaling Factor. The SCAL[1:0] bit field selects the scaling factor used by the filter algorithm according to <i>Table 561</i>.
Table 561. SCAL[1:0] – Filter scaling factor definition
<table>
<tr><th>SCAL[1:0]</th><th>Description</th></tr>
<tr><td>00</td><td>Scaling Factor = 1</td></tr>
<tr><td>01</td><td>Scaling Factor = 4</td></tr>
<tr><td>10</td><td>Scaling Factor = 8</td></tr>
<tr><td>11</td><td>Scaling Factor = 16</td></tr>
</table></td></tr>
</table>

Table 558. DECFILTER_MCR Register Field Descriptions (continued)

Field	Description
16 IDIS	Input Disable. The IDIS bit disables the block input, so that writes to the input buffer have no effect (either from the device slave-bus or from the PSI interface), and input DMA or interrupt requests are not issued. Input disabling is needed to change the block configuration to or from cascade mode. See *Section 25.5.16, Cascade mode description* for more details. 1 Input disabled 0 Input enabled IDIS resets in 1 (hardware reset only), and the module configuration procedures for previous versions (without IDIS bit) are upward compatible provided that the DECFILTER_MCR is written with 0 into the IDIS position (previously reserved).
17 SAT	Saturation Enable. The SAT bit enables the saturation of the filter output. See *Section , Saturation*, for more details. 1 Enable Saturation 0 Disables Saturation
ISEL 18	Input Selection. The ISEL bit selects the source of input data to the Filter. Possible data sources are the master block of the PSI slave-bus interface, or the CPU/DMA on the device slave-bus interface. Each device slave-bus write to the Interface Input Buffer register or DMA transfer to the input buffer is interpreted as a new sample to be processed by the filter. The output interface used is the same as the one selected by ISEL if the output selection bit MIXM = 0. When MIXM = 1, the output selection (slave-bus or device slave-bus) is contrary to the input selection (see *Table 562* and the MIXM bit definition), configuring a mixed mode operation. The slave-bus interface can always read the input/output buffers, however the PSI slave-bus interface can only read the output buffer by request of the decimation filter, in normal or input mixed modes. This behavior is outlined in detail in *Table 562*. 1 Filter input from the device slave-bus interface 0 Filter input from PSI slave-bus interface ISEL completely selects the output when the filter is configured as cascade tail, and is ignored when it is configured as cascade middle. ISEL must not be modified during the filter operation (when the status bit BSY is set). In addition, the interface not selected by ISEL must not be used to write into the input buffer.

Table 558. DECFILTER_MCR Register Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>ISEL
18
(cont)</td><td>

Table 562. ISEL/MIXM definition — Read/Write from/to Input/Output buffers

<table>
<tr><th rowspan="2">ISEL</th><th rowspan="2">MIXM</th><th rowspan="2">Mode</th><th rowspan="2">Operation</th><th colspan="2">Device slave-bus Interface</th><th colspan="2">PSI slave-bus interface</th></tr>
<tr><th>Input Buffer</th><th>Output Buffer</th><th>Input Buffer</th><th>Output Buffer</th></tr>
<tr><td rowspan="4">0</td><td>0</td><td>Normal</td><td rowspan="2">Read</td><td rowspan="2">Always, by DMA or interrupt (1) request if EDME = 1</td><td>Always, no DMA or interrupt</td><td rowspan="2">Forbidden, write only</td><td>By decfil request(2)</td></tr>
<tr><td>1</td><td>PSI Input Mixed</td><td>Always, issues DMA or interrupt(1)</td><td>Disabled</td></tr>
<tr><td>0</td><td>Normal</td><td rowspan="2">Write</td><td rowspan="2">Forbidden</td><td rowspan="2">No effect, read only</td><td rowspan="2">Enabled</td><td rowspan="2">(read only)</td></tr>
<tr><td>1</td><td>PSI Input Mixed</td></tr>
<tr><td rowspan="4">1</td><td>0</td><td>Standalone</td><td rowspan="2">Read</td><td rowspan="2">Always, no DMA or interrupt</td><td>Always, issues DMA or interrupt(1)</td><td rowspan="2">Forbidden, write only</td><td>Disabled</td></tr>
<tr><td>1</td><td>PSI Output Mixed</td><td>Always, no DMA or interrupt</td><td>By decfil request(2)</td></tr>
<tr><td>0</td><td>Standalone</td><td rowspan="2">Write</td><td rowspan="2">Enabled, by DMA or interrupt request(1)</td><td rowspan="2">No effect, read only</td><td rowspan="2">Forbidden</td><td rowspan="2">read only</td></tr>
<tr><td>1</td><td>PSI Output Mixed</td></tr>
</table>

1. Bit DSEL selects between interrupt or DMA request
2. Decimation filter issues a read request to the master block

</td></tr>
<tr><td>19
MIXM</td><td>Mixed Mode. The MIXM field selects the interface used for filter output, either device slave-bus or the PSI slave-bus, in relation to the interface selected by ISEL (see ISEL bit definition and *Table 562* for more details):
1 Interface NOT selected by ISEL is used for output, configuring mixed mode.
0 Interface selected by ISEL is used for output, configuring normal or standalone mode
MIXM must be set to 0 (zero) when the filter is configured as cascade mode.</td></tr>
</table>

Table 558. DECFILTER_MCR Register Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>20–23
DEC_RATE[3:0]</td><td>Decimation Rate Selection. The DEC_RATE[3:0] field selects the decimation rate used by the Decimation Filter. The decimation rate defines the number of data samples from the master block that is required to generate one decimated result in the Decimation Filter output.
Table 563. DEC_RATE[3:0] definition
<table>
<tr><th>DEC_RATE[3:0]</th><th>Description</th></tr>
<tr><td>0000</td><td>No Decimation: one filter output for each sample input</td></tr>
<tr><td>0001 – 1111</td><td>One filter output for each (DEC_RATE+1) sample inputs</td></tr>
</table></td></tr>
<tr><td>24
SDIE</td><td>Integrator Data Interrupt Enable. The SDIE field enables output buffer interrupts due to integrator data result being ready (at registers DECFILTER_FINTVAL and DECFILTER_FINTCNT):
1 Integration ready causes an output interrupt
0 Integration ready does not cause an output interrupt.</td></tr>
<tr><td>25
DSEL</td><td>DMA Selection. The DSEL bit determines whether the data transfers — to the input buffer (write to) and from the output buffer (read from) — are performed by DMA requests or by interrupt requests. This bit can also be active when PSI input is selected with Enhanced Debug (ISEL = 0, EDME = 1), in which case the input buffer generates read requests only (see <i>Section 25.5.14, Enhanced debug monitor description</i>).
1 DMA requests are generated
0 Interrupt requests are generated</td></tr>
<tr><td>26
IBIE</td><td>Input Buffer Interrupt Request Enable. The IBIE bit enables the Decimation Filter to generate interrupt requests when:
– device slave-bus input is selected (ISEL = 1) and DSEL = 0 when the input buffer is available to receive new data;
– PSI input is selected with Enhanced debug (ISEL = 0, EDME = 1) and DSEL = 0 when the input buffer has data to be read by the device CPU.
1 Input Buffer Interrupt Request Enabled
0 Input Buffer Interrupt Request Disabled</td></tr>
<tr><td>27
OBIE</td><td>Output Buffer Interrupt Request Enable
The OBIE bit enables the Decimation Filter interrupt requests when outputs are directed to the device slave-bus (ISEL!= MIXM) and DMA is not selected (DSEL = 0).
1 Output Buffer Interrupt Request Enabled
0 Output Buffer Interrupt Request Disabled</td></tr>
<tr><td>28
EDME</td><td>Enhanced Debug Monitor Enable
The EDME bit defines the enhanced debug monitor when input selection is from PSI (ISEL = 0). In this case, the raw data fed from the PSI Master block is also, in parallel, made available in the register DECFILTER_EDID (see <i>Section 25.5.14, Enhanced debug monitor description</i>), generating and input interrupt or DMA request.
1 Enhanced debug monitor enabled (read requests of input data from master block enabled)
0 Enhanced debug monitor disabled</td></tr>
</table>

Table 558. DECFILTER_MCR Register Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>29
TORE</td><td>Triggered Output Result Enable. The TORE bit enables an input trigger signal to force the decimation filter to send the next result of the filter back to the master block. For more details, see <i>Section , Triggered output result description</i>.
1 Output buffer update using an external signal is enabled
0 Output buffer update using an external signal is disabled
TORE must only be asserted when PSI is selected as output (normal or PSI output mixed modes).TORE must not be asserted with the filter bypassed (FTYPE = 00).</td></tr>
<tr><td>30–31
TMODE[1:0]</td><td>Trigger Mode. The TMODE field selects the way the trigger signal controls the output result sampling function enabled by the TORE bit, as shown in <i>Table 564</i>.
Table 564. TMODE[1:0] definition
<table>
<tr><th>TMODE[1:0]</th><th>Description</th></tr>
<tr><td>00</td><td>Output is posted at the rising edge of the trigger signal</td></tr>
<tr><td>01</td><td>Output is posted whenever the trigger signal is a logical 0</td></tr>
<tr><td>10</td><td>Output is posted at the falling edge of the trigger signal</td></tr>
<tr><td>11</td><td>Output is posted whenever the trigger signal is a logical 1</td></tr>
</table>
The TMODE definition replaces, and is upward compatible with, the TRFE bit definition found in previous versions of the Decimation Filter.</td></tr>
</table>

Decimation Filter Module Status Register (DECFILTER_MSR)

Figure 548. Decimation Filter Status Register (DECFILTER_MSR)

Address: DECFILTER_BASE + 0x004

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	BSY	0	DEC_COUNTER[3:0]				0	0	0	0	0	0	0	0	0	0
W							IDFC	ODFC		IBIC	OBIC		DIVRC	OVFC	OVRC	IVRC
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	IDF	ODF	0	IBIF	OBIF	0	DIVR	OVF	OVR	IVR
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 565. DECFILTER_MSR Register Field Descriptions

Field	Description
0 BSY	Decimation Filter Busy indication. The BSY bit indicates that the Decimation Filter is processing new input data from the master block in normal mode or from the device core in standalone mode. BSY is not asserted when the filter is disabled (FTYPE = 00). However, the BSY bit is asserted when the soft reset is executed. 1 Decimation Filter Busy 0 Decimation Filter Idle
1	Reserved, should be cleared.
2–5 DEC_COUNTER [3:0]	Decimation Counter. The DEC_COUNTER[3:0] field indicates the current value of the DEC_COUNTER Decimation Counter (see *Figure 546*), which counts the number of input data samples received by the Decimation Filter. When the value of this counter matches the DEC_RATE[3:0] Configuration Register field, one decimated result is generated and the DEC_COUNTER counter is reinitialized at zero. This register is cleared by a soft reset or a flush command.
6 IDFC	Input Data Flag Clear bit. The IDFC bit clears the IDF Flag bit in the Status Register. This bit is self negated, therefore it is always read as zero. 1 Clears IDF 0 No action
7 ODFC	Output Data Flag Clear bit. The ODFC bit clears the ODF Flag bit in the Status Register. This bit is self negated, therefore it is always read as zero. 1 Clears ODF 0 No action
8	Reserved, should be cleared.
9 IBIC	Input Buffer Interrupt Request Clear bit. The IBIC bit clears the IBIF Flag bit in the Status Register. This bit is self negated, therefore it is always read as zero. 1 Clears IBIF 0 No action
10 OBIC	Output Buffer Interrupt Request Clear bit. The OBIC bit clears the OBIF Flag bit in the Status Register. This bit is self negated, therefore it is always read as zero. 1 Clears OBIF 0 No action
11	Reserved, should be cleared.
12 DIVRC	DIVR Clear bit. The DIVRC bit clears the DIVR Debug Filter Input Data Read Overrun indication bit in the Status Register. This bit is self negated, therefore it is always read as zero. 1 Clears DIVR 0 No action
13 OVFC	OVF Clear bit. The OVFC bit clears the OVF Output Overflow bit in the Status Register. This bit is self negated, therefore it is always read as zero. 1 Clears OVF 0 No action
14 OVRC	OVR Clear bit. The OVRC bit clears the OVR Output Overrun bit in the Status Register. This bit is self negated, therefore it is always read as zero. 1 Clears OVR 0 No action

Table 565. DECFILTER_MSR Register Field Descriptions (continued)

Field	Description
15 IVRC	IVR Clear bit. The IVRC bit clears the IVR Filter Input Overrun indication bit in the Status Register. This bit is self negated, therefore it is always read as zero. 1 Clears IVR 0 No action
16–21	Reserved, should be cleared.
22 IDF	Input Data Flag. The IDF bit flag indicates when new data is available at the DECFILTER_IB register or at the DECFILTER_IOB register. This flag generates an Interrupt Request if enabled by the IDEN bit in the Configuration Register. This Flag is cleared by the IDFC Status bit or by a soft reset of the decimation filter. 1 New Sample received 0 Sample not received OBS: This flag is not used for read / write requests. It is used only to announce the input data event. For read / write request flag, refer to IBIF.
23 ODF	Output Data Flag. The ODF bit flag indicates when a new decimated sample is available at the DECFILTER_OB register or at the DECFILTER_IOB register. This flag generates an Interrupt Request if enabled by the ODEN bit in the Configuration Register. This Flag is cleared by the ODFC Status bit or by a soft reset of the decimation filter. 1 New Decimated Output Sample available 0 No new Decimated Output Sample available OBS: This flag is not used for read requests. It is used only to announce the output data event. For read request flag, refer to OBIF.
24	Reserved, should be cleared.
25 IBIF	Input Buffer Interrupt Request Flag. The IBIF bit flag indicates that the input buffer DECFILTER_IB is available to be filled with new data, when Enhanced Debug Monitor is off. In Enhanced Debug Monitor, it indicates the input buffer DECFILTER_IB was filled with a new sample and is ready to be read. IBIF assertion also asserts the interrupt signal when enabled by the IBIE bit in the Configuration Register when DMA is not selected (DSEL = 0) and the input buffer requires access from the device slave-bus (ISEL!= EDME). This Flag is cleared by the IBIC Status bit or by a soft reset of the decimation filter. 1 New Sample is requested (ISEL = 1, EDME = 0) or new sample is available in Enhanced Debug Monitor (ISEL = 0, EDME = 1). 0 No action
26 OBIF	Output Buffer Interrupt Request Flag. The OBIF bit flag indicates that either a new decimated sample is available at the DECFILTER_OB register. This flag generates an Interrupt Request if enabled by the OBIE bit in the Configuration Register and with ISEL!=MIXM and DSEL = 0. This Flag is cleared by the OBIC Status bit or by a soft reset of the decimation filter. 1 New Decimated Output available 0 No new Decimated Output available
27	Reserved, should be cleared.
28 DIVR	Enhanced Debug Monitor Input Data Read Overrun. The DIVR bit indicates that a received sample in the Filter Interface Input Register was overwritten by a new sample and was not read by the Core. This flag generates an Interrupt Request if enabled by the ERREN bit in the Configuration Register. This Flag is cleared by the DIVRC Status bit or by a soft reset of the decimation filter. 1 Enhanced Debug Monitor Input Data Read Overrun occurred 0 Input Data Read Overrun did not occur in Enhanced Debug monitor

Table 565. DECFILTER_MSR Register Field Descriptions (continued)

Field	Description
29 OVF	Filter Overflow Flag. The OVF bit indicates that an overflow occurred in the filtered sample result. This flag generates an Interrupt Request if enabled by the ERREN bit in the Configuration Register. This Flag is cleared by the OVFC Status bit or by a soft reset of the decimation filter. 1 Overflow occurred 0 No overflow
30 OVR	Output Interface Buffer Overrun. The OVR bit indicates that a decimated sample was overwritten by a new sample in the Interface Output Buffer Register. This flag generates an Interrupt Request if enabled by the ERREN bit in the Configuration Register. This Flag is cleared by the OVRC Status bit or by a soft reset of the decimation filter. 1 Filter Output Overrun occurred 0 No Output Overrun
31 IVR	Input Interface Buffer Overrun. The IVR bit indicates that a received sample in the Filter Interface Input Register was overwritten by a new sample. This was probably caused by a violation of the Decimation Filter maximum throughput. This flag generates an Interrupt Request if enabled by the ERREN bit in the Configuration Register. This Flag is cleared by the IVRC Status bit or by a soft reset of the decimation filter. 1 Input Buffer Overrun occurred 0 Input Buffer Overrun did not occur IVR does not set due to input register writes when input is disabled (DECFILTER_MCR bit IDIS = 1).

Decimation Filter Module Extended Configuration Register (DECFILTER_MXCR)

Figure 549. Decimation Filter Extended Configuration Register (DECFILTER_MXCR)

Address: DECFILTER_BASE + 0x008 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SDMAE	SSIG	SSAT	SCSAT	0	0	0	0	0	0	0	0	0	0	0	0
W															SRQ	SZRO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SISEL	0	SZROSEL		0	0	SHLTSEL		0	SRQSEL			0	0	SENSEL	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 566. DECFILTER_MXCR Register Field Descriptions

Field	Description
0 SDMAE	Integrator DMA Enable. The SDMAE bit enables the DMA request when an integrator output is requested (see *Section , Integrator outputs*). 1 integrator DMA request enabled 0 integrator DMA request disabled The DMA channel used is the same one used for filter outputs, and any configuration that generates DMA requests from both of those sources is not allowed.
1 SSIG	Integrator Signal operation selection. The SSIG bit defines how the filtered data signal is treated for integration: 1 integrator input takes signaled filter output 0 integrator input takes the absolute value of filter output
2 SSAT	Integrator Saturated operation selection. The SSAT bit defines how the integrator accumulator behaves in case of an overflow. 1 integrator accumulator saturates on an overflow 0 integrator accumulator holds a modulo 2^{17} value (considering the 15-bit fractional part) on an overflow. In saturated operation the overflown integration sum holds the value 0xFFFFFFFF for absolute integration (SSIG = 0), or values 0x7FFFFFFF (positive saturation) and 0x80000000 (negative saturation) for signaled integration (SSIG = 1). Non-saturated mode is not supported with signaled integration, therefore one must not configure SSIG = 1 and SSAT = 0.
3 SCSAT	Integrator Counter Saturated operation selection. The SCSAT bit defines how the integrator sample counter behaves in case of an overflow. 1 integrator sample counter saturates on an overflow, holding a value of 0xFFFFFFFF. 0 integrator sample counter holds a modulo 2^{32} value on an overflow.
4–13	Reserved, should be cleared.
14 SRQ	Integrator Output Request. The SRQ bit is used to command the update of the integrator output, reflected in the registers DECFILTER_FINTVAL and DECFILTER_FINTCNT. It may also cause a DMA or interrupt request, depending on the DECFILTER_MCR bit SDIE and DECFILTER_MXCR bit SDMAE. This is a write-only bit, so reads always return 0. For more details see *Section , Integrator outputs*. 1 requests integrator output update 0 no integrator output update request
15 SZRO	Integrator Zero. The SZRO bit is used to zero the integrator sum. This is a write-only bit, reads always return 0. For more details see *Section , Integrator reset*. 1 zeroes integrator sum 0 does not zero integrator sum If bits SRQ and SZRO are both written 1 at the same time, the integrator is reset only after the registers DECFILTER_FINTVAL and DECFILTER_FINTCNT are updated.
16 SISEL	Integrator Input Selection. The SISEL bit selects the input of the integrator. For more details see *Section , Integrator inputs*. 1 filter outputs before the decimation feed the integrator 0 decimated filter outputs feed the integrator
17	Reserved, should be cleared.

Table 566. DECFILTER_MXCR Register Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>18–19
SZROSEL[1:0](1)</td><td>Integrator Zero Control Mode Selection
The SZROSEL field defines the use of the integrator zero hardware input signal, according to <i>Table 567</i>. For more details see <i>Section , Integrator reset</i>.
Table 567. SZROSEL – Integrator Zero mode
<table>
<tr><th>SZROSEL[1:0]</th><th>Description</th></tr>
<tr><td>00</td><td>Hardware integrator zero request disabled</td></tr>
<tr><td>01</td><td>Integrator zero on toggle of hardware signal</td></tr>
<tr><td>10</td><td>Integrator zero on rising edge of hardware signal</td></tr>
<tr><td>11</td><td>Integrator zero on falling edge of hardware signal</td></tr>
</table></td></tr>
<tr><td>20–21</td><td>Reserved, should be cleared.</td></tr>
<tr><td>22–23
SHLTSEL[1:0](2)</td><td>Integrator Halt Control Selection. The SHLTSEL field defines the integrator halting mechanism, according to <i>Table 568</i>. When the integrator is halted, the integration accumulator remains unaltered on filter outputs independently of the enabling selected by SENSEL. For more details see <i>Section , Integrator enabling and halting</i>.
Table 568. SHLTSEL – Integrator halt control selection
<table>
<tr><th>SHLTSEL[1:0]</th><th>Description</th></tr>
<tr><td>00</td><td>Hardware halt control signal disabled</td></tr>
<tr><td>01</td><td>Integrator halted, independently of the hardware signal</td></tr>
<tr><td>10</td><td>Integrator halted when signal is at logical 0</td></tr>
<tr><td>11</td><td>Integrator halted when signal is at logical 1</td></tr>
</table></td></tr>
<tr><td>24</td><td>Reserved, should be cleared.</td></tr>
</table>

Table 566. DECFILTER_MXCR Register Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>25–27
SRQSEL[2:0][1]</td><td>Integrator Output Read Request Mode Selection. The SRQSEL field defines the use of the integrator output request hardware input signal, according to <i>Table 569</i>. An integrator output request updates the registers DECFILTER_FINTVAL and DECFILTER_FINTCNT, also causing a DMA or interrupt request. Note that DMA or interrupt requests due to integrator output updates depend on the DECFILTER_MXCR bit SDMAE and DECFILTER_MCR bit SDIE.
When continuous output is on, an integrator output request is issued whenever a new filter output is accumulated. For more details see <i>Section , Integrator outputs</i>.
Table 569. SRQSEL – Integrator output request mode
<table>
<tr><th>SRQSEL[2:0]</th><th>Description</th></tr>
<tr><td>000</td><td>Hardware output request disabled</td></tr>
<tr><td>001</td><td>Integrator output request on toggle of hardware signal</td></tr>
<tr><td>010</td><td>Integrator output request on rising edge of hardware signal</td></tr>
<tr><td>011</td><td>Integrator output request on falling edge of hardware signal</td></tr>
<tr><td>100</td><td>Reserved</td></tr>
<tr><td>101</td><td>Continuous output request on, independently of hardware signal</td></tr>
<tr><td>110</td><td>Continuous output request on when signal is at logical 0</td></tr>
<tr><td>111</td><td>Continuous output request on when signal is at logical 1</td></tr>
</table></td></tr>
<tr><td>28–29</td><td>Reserved, should be cleared.</td></tr>
<tr><td>30–31
SENSEL[1:0][1]</td><td>Integrator Enable Control Selection. The SENSEL field defines the integrator enabling mechanism, according to <i>Table 570</i>. When the integrator is enabled, filter outputs selected by the SISEL bit are added to the integration accumulator. When the integrator is disabled, the integration accumulator remains unaltered on filter outputs. For more details see <i>Section , Integrator enabling and halting</i>.
Table 570. SENSEL – Integrator enable control selection
<table>
<tr><th>SENSEL[1:0]</th><th>Description</th></tr>
<tr><td>00</td><td>Integrator disabled, independently of the hardware enable control signal</td></tr>
<tr><td>01</td><td>Integrator enabled, independently of the hardware signal</td></tr>
<tr><td>10</td><td>Integrator enabled when signal is at logical 0</td></tr>
<tr><td>11</td><td>Integrator enabled when signal is at logical 1</td></tr>
</table></td></tr>
</table>

[1] The hardware input signals are ZSELA for Decimation filter A and ZSELB for Decimation filter B, defined in *Section 15.6.24, IMUX Select Register 10 (SIU_ISEL10)*.

[2] The hardware input signals are HSELA for Decimation filter A and HSELB for Decimation filter B, defined in *Section 15.6.24, IMUX Select Register 10 (SIU_ISEL10)*.

Decimation Filter Module Extended Status Register (DECFILTER_MXSR)

Figure 550. Decimation Filter Extended Status Register (DECFILTER_MXSR)

Address: DECFILTER_BASE + 0x00C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W								SDFC			SSEC	SCEC		SSOVFC	SCOVFC	SVRC
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	SDF	0	0	SSE	SCE	0	SSOVF	SCOVF	SVR
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 571. DECFILTER_MXSR Register Field Descriptions

Field	Description
0–6	Reserved, should be cleared.
7 SDFC	Integrator Output Data Flag Clear bit. The SDFC bit clears the SDF Flag bit in the Status Register. This bit is self negated, therefore it is always read as zero. 1 Clears SDF 0 No action
8–9	Reserved, should be cleared.
10 SSEC	Integrator Sum Exception Clear bit. The SSEC bit clears the SSE flag bit in the Status Register. This bit is self negated, therefore it is always read as zero. 1 Clears SSE 0 No action
11 SCEC	Integrator Count Exception Clear bit. The SCEC bit clears the SCE flag bit in the Status Register. This bit is self negated, therefore it is always read as zero. 1 Clears SCE 0 No action
12	Reserved, should be cleared.
13 SSOVFC	Integrator Sum Overflow Clear bit. The SSOVFC bit clears the SSOVF Flag bit in the Status Register. This bit is self negated, therefore it is always read as zero. 1 Clears SSOVF 0 No action
14 SCOVFC	Integrator Count Overflow Clear bit. The SCOVFC bit clears the SCOVF Flag bit in the Status Register. This bit is self negated, therefore it is always read as zero. 1 Clears SCOVF 0 No action

Table 571. DECFILTER_MXSR Register Field Descriptions (continued)

Field	Description
15 SVRC	SVR Clear bit. The SVRC bit clears the SVR Integrator Data Overrun indication bit in the Status Register. This bit is self negated, therefore it is always read as zero. 1 Clears SVR 0 No action
16–22	Reserved, should be cleared.
23 SDF	Integrator Data Flag. The SDF bit flag indicates when a new integrator result is available at the DECFILTER_FINTVAL register. This flag generates an Interrupt Request if enabled by the SDIE bit in the Configuration Register. This Flag is cleared by the SDFC Status bit or by a soft reset of the decimation filter. 1 New integrator result available 0 No new integrator result available
24–25	Reserved, should be cleared.
26 SSE	Integrator Sum Exception flag. The SSE bit indicates an exceptional condition of the integrator accumulator. This flag generates an Interrupt Request if enabled by the DECFILTER_MCR bit ERREN, and it is cleared by the SSEC bit or by a soft reset. Integrator exceptions are defined in *Section , Integrator exceptions*. 1 Integrator accumulator exception. 0 No exception in the integrator accumulator.
27 SCE	Integrator Count Exception flag. The SCE bit indicates an exceptional condition of the integrator counter. This flag generates an Interrupt Request if enabled by the DECFILTER_MCR bit ERREN, and it is cleared by the SCEC bit or by a soft reset. Integrator exceptions are defined in *Section , Integrator exceptions*. 1 Integrator counter exception. 0 No exception in the integrator counter.
28	Reserved, should be cleared.
29 SSOVF	Integrator Sum Overflow Flag. The SSOVF bit indicates an overflow of the integrator accumulator. This Flag is cleared by the SSOVFC bit or by a soft reset. 1 Integrator accumulator overflown. 0 No overflow in the integrator accumulator. The SSOVF bit samples the integrator accumulator overflow condition when and only when either registers DECFILTER_FINTVAL or DECFILTER_CINTCNT are updated. Therefore, only one of the register pairs (DECFILTER_FINTVAL/DECFILTER_FINTCNT and DECFILTER_CINTVAL/DECFILTER_CINTCNT) must be used by the application, in order to avoid races.

Table 571. DECFILTER_MXSR Register Field Descriptions (continued)

Field	Description
30 SCOVF	Integrator Count Overflow Flag. The SCOVF bit flag indicates an overflow of the internal integrated sample counter. This Flag is cleared by the SCOVFC bit or by a soft reset. 1 Integrator sample counter overflown. 0 No overflow in the integrator sample counter. The SCOVF bit samples the integrator accumulator overflow condition when and only when either registers DECFILTER_FINTVAL or DECFILTER_CINTCNT are updated. Therefore, only one of the register pairs (DECFILTER_FINTVAL/DECFILTER_FINTCNT and DECFILTER_CINTVAL/DECFILTER_CINTCNT) must be used by the application, in order to avoid races.
31 SVR	Integrator Data Overrun. The SVR bit indicates that an integration value and count in the registers DECFILTER_FINTVAL and DECFILTER_FINTCNT was overwritten by a new integrator output request and was not read by the CPU or DMA. This flag generates an Interrupt Request if enabled by the ERREN bit in the Configuration Register. This Flag is cleared by the SVRC bit or by a soft reset. 1 Integrator Data Overrun occurred. 0 Integrator Data Overrun did not occur.

Decimation Filter Interface Input Buffer Register (DECFILTER_IB)

The Input Buffer Register provides access to the Input buffer of the decimation filter when the filter is in the standalone or PSI output mixed modes of operation. Writes to this register are interpreted as requests to the Decimation Filter to process new sample data. Writes to this register when ISEL = 0 are not allowed.

Figure 551. Decimation Filter Interface Input Buffer Register (DECFILTER_IB)

Address: DECFILTER_BASE + 0x010

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	INTAG[3:0]				0	0	0	0	0	0	PREFILL	FLUSH
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	INPBUF[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 572. DECFILTER_IB Register Field Descriptions

Field	Description
0–3	Reserved, should be cleared.
4–7 INTAG[3:0]	Decimation filter input tag bits The INTAG[3:0] bit field is defined as a selector signal and it is used to identify different destinations for the INBUF[15:0] data. When the PSI master block is an eQADC, it is used in PSI output mixed mode to address the appropriate RFIFO in the eQADC block.

Table 572. DECFILTER_IB Register Field Descriptions (continued)

Field	Description
8–13	Reserved, should be cleared.
14 PREFILL	Decimation Filter Prefill/Filter control bit The PREFILL bit selects the Decimation Filter operation mode. For more details, see *Section 25.5.7, Filter prefill control description*. 1 Decimation Filter prefill sample 0 Decimation Filter normal sample
15 FLUSH	Decimation Filter Flush control bit Assertion of the FLUSH bit initializes the Decimation Filter to a initial state, as defined in *Section 25.5.9, Flush command description*. This bit is self negated and it is cleared only when the data is read and the flush is executed. 1 Flush request 0 No flush request
16–31 INPBUF[15:0]	Input Buffer Data The INPBUF[15:0] bit field carries the sample data to be filtered. This data buffer can be written from the PSI slave-bus interface or by the device slave-bus interface. See *Section 25.5.3, Input buffer description*, for more details.

Decimation Filter Interface Output Buffer Register (DECFILTER_OB)

Figure 552. Decimation Filter Interface Output Buffer Register (DECFILTER_OB)

Address: DECFILTER_BASE + 0x014 Access: User read only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	OUTTAG[3:0]			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	OUTBUF[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 573. DECFILTER_OB Register Field Descriptions

Field	Description
0–11	Reserved, should be cleared.
12–15 OUTTAG[3:0]	Decimation filter output tag bits. The OUTTAG[3:0] bit field is defined as a selector signal and it is used to identify different destinations for the OUTBUF[15:0] data. When an eQADC is the PSI block master, it holds the same number used to address the destination RFIFO.
15–31 OUTBUF[15:0]	Output Buffer Data. The OUTPBUF[15:0] bit field is the result data in the decimation filter Output Buffer. It represents a fixed point signed number in two’s complement format and is updated only when a decimated result is ready to be transmitted, meaning it contains the last decimated result from the filter.

Decimation Filter Coefficient n Register (DECFILTER_COEFn)

Figure 553. Decimation Filter Coefficient n Register (DECFILTER_COEFn)

Address: DECFILTER_BASE + 0x020–0x040 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	8{COEFn[23]}								COEFn[23:16]							
W	▨	▨	▨	▨	▨	▨	▨	▨								
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	COEFn[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 574. DECFILTER_COEFn Register Field Descriptions

Field	Description
0–7	
8–31 COEFn[23:0]	Coefficient n field The COEFn[23:0] bit fields are the digital filter coefficients registers. The coefficients are fractional signed values in two’s complement format, in the range (-1 ≤ coef < 1). Reads to this register are sign-extended, meaning the coefficient’s sign bit is copied to all eight most significant register bits.Writing to these fields when BSY = 1 is not allowed.

Decimation Filter TAPn Register (DECFILTER_TAPn)

Figure 554. Decimation Filter TAPn Register (DECFILTER_TAPn)

Address: DECFILTER_BASE + 0x078–0x094 Access: User read only

	0	1	2	3		5	6	7	8	9	10	11	12	13	14	15
R	8{TAPn[23]}								TAPn[23:16]							
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	TAPn[15:0]															
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 575. DECFILTER_TAPn Register Field Descriptions

Field	Description
8–31 TAPn[23:0]	TAPn Register The read-only TAPn[23:0] bit fields shows the contents of the digital filter tap registers, as fractional signed values in two’s complement format, in the range (-1 ≤ coef < 1).

Note: Reads to this register are sign-extended, meaning the coefficient’s sign bit is copied to all eight most significant register bits.

The content of these registers is meaningless when BSY = 1.

Decimation Filter Interface Enhanced Debug Input Data Register (DECFILTER_EDID)

The Enhanced Debug Input Data Register provides read-only access to the sample data received by the decimation filter when the input is selected from the device slave-bus (ISEL = 0), allowing the monitoring of filter operation. See *Section 25.5.14, Enhanced debug monitor description* for more details. Writes to this register are not allowed.

Figure 555. Decimation Filter Interface Input Buffer Register (DECFILTER_EDID)

Address: DECFILTER_BASE + 0x0D0 Access: User read only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SAMP_DATA[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 576. DECFILTER_EDID Register Field Descriptions

Field	Description
0–31 SAMP_DATA [15:0]	Conversion Sample Data. The SAMP_DATA[15:0] bit field carries the data that was loaded in the decimation filter to be processed by the FIR/IIR sub-block. This conversion data is supplied by the PSI slave-bus interface only. See *Section 25.5.11, Interrupts requests description*, and *Section 25.5.12, DMA requests description*, for more details.

Decimation Filter Final Integration Value Register (DECFILTER_FINTVAL)

Figure 556. Decimation Filter Final Integration Value Register (DECFILTER_FINTVAL)

Address: DECFILTER_BASE + 0x0E0 Access: User read only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SUM_VALUE[31:16]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SUM_VALUE[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 577. DECFILTER_FINTVAL Register Field Descriptions

Field	Description
0–31 SUM_VALUE [31:0]	Integration Sum Value. The SUM_VALUE[31:0] field holds the sum of filtered output values. The 17 most significant bits hold the integer part, and the 15 least significant ones the fractional part of the integration value. The control of the integration sum and update of this register is determined by the register DECFILTER_MXCR (see *Section , Decimation Filter Module Extended Configuration Register (DECFILTER_MXCR)*). The register is updated only upon an integration output request. SUM_VALUE should be taken as an unsigned number when the integrator is configured for absolute operation (DECFILTER_MXCR bit SSIG = 0), and a two's complement signed number otherwise.

Note: If the DEFILTER_MXCR bit SSAT = 1, the integration sum is saturated, so that if the accumulation overflows DECFILTER_FINTVAL holds the value 0xFFFFFFFF for absolute integration (SSIG = 0), or values 0x7FFFFFFF (positive saturation) and 0x80000000 (negative saturation) for signaled integration (SSIG = 1).

If SSAT = 0, DECFILTER_FINTVAL holds the integration sum modulo 2^{17} (considering the 15-bit fractional part).

Decimation Filter Final Integration Count Value Register (DECFILTER_FINTCNT)

Figure 557. Decimation Filter Final Integration Count Value Register (DECFILTER_FINTCNT)

Address: DECFILTER_BASE + 0x0E4 Access: User read only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	COUNT[31:16]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	COUNT[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 578. DECFILTER_FINTCNT Register Field Descriptions

Field	Description
0–31 COUNT[31:0]	Integration Count Value The COUNT field holds the count of filtered outputs integrated. The control of the integration sum and update of this register is determined by the register DECFILTER_MXCR (see *Section , Decimation Filter Module Extended Configuration Register (DECFILTER_MXCR)*). The register is updated together with DECFILTER_FINTVAL, only upon an integration output request.

Decimation Filter Current Integration Value Register (DECFILTER_CINTVAL)

Figure 558. Decimation Filter Current Integration Value Register (DECFILTER_CINTVAL)

Address: DECFILTER_BASE + 0x0E8 Access: User read only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SUM_VALUE[31:16]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SUM_VALUE[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 579. DECFILTER_CINTVAL Register Field Descriptions

Field	Description
0–31 SUM_VALUE [31:0]	Integration Sum Value. The SUM_VALUE[31:0] field holds an unsigned number representing the sum of filtered output values, continuously updated as the integration proceeds. The control of the integration sum is determined by the register DECFILTER_MXCR (see *Section , Decimation Filter Module Extended Configuration Register (DECFILTER_MXCR)*).

Note: *If the DEFILTER_MXCR bit SSAT = 1, the integration sum is saturated, so that if the accumulation overflows DECFILTER_CINTVAL holds the value 0xFFFFFFFF for absolute integration (SSIG = 0), or values 0x7FFFFFFF (positive saturation) and 0x80000000 (negative saturation) for signaled integration (SSIG = 1).*

If SSAT = 0, DECFILTER_FINTVAL holds the integration sum modulo 2^{17} *(considering the 15-bit fractional part).*

Note: *A read on this register automatically commands an update of the register DECFILTER_CINTCNT.*

Decimation Filter Current Integration Count Value Register (DECFILTER_CINTCNT)

Figure 559. Decimation Filter Current Integration Count Value Register (DECFILTER_CINTCNT)

Address: DECFILTER_BASE + 0x0EC Access: User read only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	COUNT[31:16]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	COUNT[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 580. DECFILTER_CINTCNT Register Field Descriptions

Field	Description
0–31 COUNT[31:0]	Integration Count Value. The COUNT field holds the count of filtered outputs integrated. The value is updated only when register DECFILTER_CINTVAL is read, to keep the coherency between the integration and count values.

25.4.3 Decimation Filter Memory Map for Parallel Side Interface

The Decimation Filter exchanges data with the master block through the PSI. The master block sends data to the input buffers and reads data from the output buffers of the filter. To implement this exchange, only a single register is required as described in *Table 581*, therefore the PSI address is ignored.

Table 581. Parallel side interface memory map for decfilter data exchange

Decimation filter address	Description	Access
0x0	DECFILTER_IOB - Decimation Filter[1] input/output Registers	R/W

1. The input registers are addressed for write only and output registers are addressed for read only.

25.4.4 PSI Register Description

This register is defined as 24-bit.

Decimation Filter Input/Output Buffers Register (DECFILTER_IOB)

The Input/Output Buffers Register is used by the PSI master block to access the Input and Output buffers of the decimation filter. Writes to this register are interpreted as requests to the Decimation Filter to process new sample data or to bypass timestamp data. Reads from this register frees the decimation filter output buffer from filtered data or from bypass timestamp data.

Figure 560. Decimation Filter Interface Input/Output Buffers Register (DECFILTER_IOB)

Address: PSI_BASE + 0x0 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R									0	0	0	0	OUT_TAG[3:0]			
W									M_FLUSH	M_CTRL [1:0]			INP_TAG[3:0]			
Reset	—	—	—	—	—	—	—	—	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	OUT_BUFF[15:0]															
W	INP_BUFF[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 582. DECFILTER_IOB Register Field Descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>0–7</td><td>Reserved, should be cleared.</td></tr>
<tr><td>8
M_FLUSH</td><td>Master block Flush request/control bit. Assertion of the M_FLUSH bit initializes the Decimation Filter to a initial state, as defined in <i>Section 25.5.9, Flush command description</i>. The sample or timestamp data written with a flush request in the same IOB register write is processed normally after the flush.
1 flush request
0 no flush request</td></tr>
<tr><td>9–10
M_CTRL[1:0]</td><td>Decimation Filter mode control bits. <i>Table 583</i> describes the M_CTRL[1:0] field functions. This field is used for control of the Decimation Filter.
Table 583. M_CTRL[1:0] – Decimation filter control functions
<table>
<tr><th>M_CTRL[1:0]</th><th>Description</th></tr>
<tr><td>00</td><td>PREFILL — A prefill indicates to the Decimation Filter to accept INP_BUFF[15:0] as valid data but no decimated samples are generated out of these master samples. The prefill function is used to initialize and stabilize the Decimation Filter without generating decimated samples.</td></tr>
<tr><td>01</td><td>CONVERSION RESULT — A conversion result indicates that the INP_BUFF[15:0] field is data to be treated as valid sample data and it is considered for decimation counting and output buffer update.</td></tr>
<tr><td>10</td><td>TIMESTAMP — A timestamp indicates that the INP_BUFF[15:0] field has data that bypasses the flow in the decimation filter logic, returning back to the master block without any modification when:
– the previous accompanying data is not for prefill, and
– when the previous accompanying input data is generating decimated filter output.
Also, bit M_FLUSH is always 0 for timestamp data type.</td></tr>
<tr><td>11</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>11</td><td>Reserved, should be cleared.</td></tr>
<tr><td>12–15
INP_TAG[3:0]</td><td>Decimation filter input tag bits. The INP_TAG[3:0] field indicates the destination associated with the INPBUFF[15:0] sample. This value is stored by the Decimation Filter and used to address the destination register when a decimated sample is available to be read by the master block. Since several input data samples can be received before a decimated result is generated, the INP_TAG[3:0] used for the decimated sample corresponds to the latest INP_TAG[3:0] received. Therefore it is expected that the tag field be constant during the decimation process.</td></tr>
<tr><td>12–15
OUT_TAG[3:0]</td><td>Decimation filter output tag bits. The OUT_TAG[3:0] bit field is used to address the appropriate destination register in the master block for the accompanying OUTBUF[15:0] data. When this value is updated, it is a copy of the INP_TAG[3:0] value that was received with the last processed input data.
When an eQADC is the PSI master block, this is used to address the appropriate RFIFO in the eQADC block.</td></tr>
</table>

Table 582. DECFILTER_IOB Register Field Descriptions (continued)

Field	Description
16–31 INP_BUFF[15:0]	Input Buffer Data. The INP_BUFF[15:0] bit field is the data input from the master block. The input register can be written with this data when ISEL = 0. This data can be timestamp information that is not processed by the filter, or sample data that is processed by the digital filter. In this case, the information is a signed signal in two's complement format.
16–31 OUT_BUFF[15:0]	Output Buffer Data. The OUT_BUFF[15:0] bit field corresponds to the data result of the decimation filter that has been processed to the master block. This data can be timestamp information or a digital filter result. In this case, the information is a signed signal in two's complement format.

25.5 Functional description

25.5.1 Overview

Figure 546 shows the block diagram of the Decimation Filter. The Control Logic provides the control signals for all other sub-modules. The PSI data interface is subdivided into two sub-modules, transmitter and receiver, that are accessed by the PSI slave-bus interface. The bypass path is used when the filter is disabled and the incoming data can be transmitted back to the master block without being processed by the Filter algorithm. The Filter hardware is implemented in such a way that an IIR (1 x 4 poles) or FIR filter type can be implemented. The selection between the two types of filter algorithms is implemented by the Control Logic sub-block.

The Coefficient register file provides the digital filter coefficients. This block is a register bank with read/write access by the device slave-bus interface. The Filter TAP registers are also accessed through the device slave-bus line interface, providing additional debug capabilities to the Decimation Filter block. The MAC (Multiply Accumulate) sub-block executes the filter arithmetic operations controlled by the Control Logic. The MAC results are routed to the Filter TAP registers and to the output buffer when the result is a decimated filter sample.

25.5.2 Parallel Side Interface (PSI) description

This section describes the operation of the Parallel Side Interface (PSI) sub-block which is responsible for communication and data exchange between the master block (for instance, the eQADC block) and the Decimation Filter block.

The decimation filter receives sample data from the master block. The input data bus format is presented in *Figure 560*. The sample data arrives along with the control bits. These control bits are decoded and the proper action is decided in the Control Logic sub-block. When the decimation filter finishes its processing and a result is available, the read request signal is issued to the master block. This data transfer request remains set until the result is read by the master block.

When using two or more decimation filter blocks in the device, the output of the second block is connected to the input data of the next block, and the output of the first block is connected to the read data input of the PSI master block.

25.5.3 Input buffer description

The decimation filter receives data samples for filtering from a master block (e.g., eQADC) using the PSI interface, or from the CPU using the device slave-bus interface. The data source is selected by the ISEL bit of the module configuration register DECFILTER_MCR.

When the device slave-bus interface is selected and DMA operation is chosen (DECFILTER_MCR bit DSEL = 1), the input data request signal is asserted when the input buffer is empty. When DMA operation is not chosen (DSEL = 0) in standalone or PSI output mixed modes, the logic asserts an input interrupt request and the input buffer waits for data from the device slave-bus.

Input buffer filling is flagged by the DECFILTER_MSR bit IDF. The IDF flag remains set, even after the input data has been consumed and the buffer is free, until it is cleared by software.

Input buffer overrun is detected and flagged by the DECFILTER_MSR bit IVR. The idle (BSY = 0) Decimation Filter is able to receive two consecutive input writes without input overrun. For more details, see *Section , Input buffer overrun*.

When the selected input source is the PSI master block, the PSI master may send timestamp data after related sample data for filtering. The timestamp is sent back to the PSI following the respective filter output, using the same request mechanism. As the decimation filter takes several clock cycles to process a sample, the timestamp is copied into an internal timestamp register until the filtered output is sent out, therefore freeing the input buffer for yet another sample data.

When the input buffer is loaded with a sample data coming from the PSI interface and enhanced debug monitor enabled is enabled (DECFILTER_MCR bits ISEL = 0, EDME = 1), input DMA or interrupt requests are asserted, so that the PSI input samples can be monitored from the device slave-bus interface. These interrupt or DMA requests result in input read accesses from the DECFILTER_IB register, unlike write accesses needed when the same requests are made and the device slave-bus interface is selected for input (ISEL = 1). Only the sample data input (not timestamps), can be monitored in this way. See also *Section 25.5.14, Enhanced debug monitor description*.

Soft reset clears the input buffer indication flags and any data write/read request that was generated in any mode.

Input buffer overrun

An input overrun occurs when the input buffer is holding input data and new data is received by the filter. See *Section 25.5.3, Input buffer description*, for details of the input buffer.

The input buffer overrun can occur only when the input is enabled (DECFILTER_MCR bit IDIS = 0), and in the following cases:

- When the input buffer has sample data to be processed but the filter is busy and another input (data or timestamp) is received.
- When the input buffer has a timestamp, the internal timestamp register is loaded and the next input data is received.

As an example of the input data sequence, assume that the filter is enabled and not busy, and all registers are empty. Then a word of sample data is received followed by a timestamp and another word of sample data. No input overrun occurs in this case, because the first sample is immediately transferred to the tap input register, the timestamp is immediately transferred to the internal timestamp storage register, and the second sample can be held in the input buffer until the end of the processing of the sample data by the filter. The input overrun may occur if more input is received before the end of the processing, or if the filter is busy at the beginning of the received sequence.

When the filter is in bypass/disable mode (DECFILTER_MCR field FTYPE = 00), the data from the input buffer is transferred to the output buffer, if it is not already full. If the output buffer is full, the input buffer is loaded, and another word of input data is sent, then an input overrun occurs.

Note: Configuring ISEL = 1, MIXM = 1 and FTYPE = 00 (bypass), writes to the DECFILTER_IB are routed directly to the PSI output.

25.5.4 Output buffer description

The decimation filter has an output buffer to send filtering results to a master block using the PSI, or to the CPU using the device slave-bus, as selected through the DECFILTER_MCR bits ISEL and MIXM bits.

Filtering of prefill inputs do not update the output buffer, so the flag ODF is not set.

When filter types IIR and FIR are selected and the input source is the PSI master block (normal or PSI input mixed modes), the output buffer receives data from the MAC sub-block or from the timestamp storage register. The result from the MAC is written immediately after the processing if the decimation is enabled. However, the timestamp data is enabled to be written in the output buffer only when the output buffer is empty, the decimation count is reached, and the corresponding data was also ready to be transmitted. When a new word of data is available in the output buffer, a read request signal is sent to the master block. The master block has to send the corresponding read commands to clear the read request signal. In this configuration, the core can always read the output buffer. The flag ODF is set when the buffer is updated.

When in filter operation mode with input from the device slave-bus (ISEL = 1), only sample data is processed by the filter, as there is no way to input timestamps. When the filter result from the MAC is ready, this is immediately written to the output buffer when the decimation count is reached. The flag ODF is set when the buffer is updated. It also generates a DMA read request if DSEL = 1 in standalone or PSI input mixed modes.

When the filter is bypassed (FTYPE = 00), and PSI is selected as output (normal mode or PSI output mixed mode), the data written into the input buffer waits until the output buffer is empty before passing the data. This is needed because the master block takes some clock cycles to send the read commands to the decimation filter after the read request signal is asserted. When the filter is bypassed and the device slave-bus is selected as output (standalone mode or PSI input mixed mode), the data written into the input buffer is immediately written into the output buffer. The flag ODF is set when the buffer is updated and a DMA read request is asserted if DSEL = 1.

Soft reset clears the output buffer, as well as any data read request generated in any mode.

Output buffer overrun

An output overrun occurs when the output buffer is holding output data (sample or timestamp) that has not been read is overwritten with another word of data (sample or timestamp).

Output overruns are flagged by the DECFILTER_MSR bit OVR, if the output buffer is updated when not empty. The output buffer empty condition depends on the mode and output selection as follows:

- if PSI is selected as output (normal mode or PSI output mixed mode), the output buffer is considered empty when the last output has been read.
- if cascade mode is selected, the output buffer is considered empty if the last output request was acknowledged.
- if the device slave-bus is selected (standalone or PSI input mixed mode) for output and DMA is selected (DSEL = 1), the output buffer is considered empty if the last output has been read.
- if the device slave-bus is selected (standalone or PSI input mixed mode) for output and DMA is not selected (DSEL = 0), the output buffer is considered empty when the ODF flag is negated.

Note: *When the device slave-bus is selected (standalone or PSI input mixed mode) for output and DMA is not selected (DSEL = 0), the ODF flag must be cleared to avoid overrun, even if its corresponding interrupt is not used (ODEN = 0).*

Prefill inputs do not cause IIR or FIR output overrun in standalone, normal, and cascade tail modes, but can cause overruns in cascade head or middle configurations (see *Section , Cascade mode*).

When bypass is selected, the output overrun does not occur because the data written into the input buffer is written into the output buffer only when this buffer is empty, but an input overrun may still occur (see *Section , Input buffer overrun*).

Triggered output result description

The posting of a filter output, either to the PSI interface or to the device slave-bus can be controlled by an additional input trigger signal (see *Section 25.3.1, Decimation trigger signal*). It allows the decimation to be controlled by another circuit, instead of the internal decimation counter.

Triggered output operation is enabled by the bit TORE in the configuration register DECFILTER_MCR. When triggered output is enabled, the decimation count configured by DECFILTER_MCR field DEC_COUNTER is ignored. The decimation filter detects the rising edge or the falling edge of the trigger signal as selected by the configuration field TMODE in the DECFILTER_MCR. When the corresponding edge is detected, the output buffer is enabled to receive the next filter result.

The input trigger signal can also be used as a simple filtered output enable: in this trigger mode, when the input trigger signal is asserted, every filtered output is posted to the output buffer; no output is posted when the signal is negated. This trigger mode and the assertion polarity (active 0 or 1) of the input trigger signal is also defined in the DECFILTER_MCR field TMODE.

25.5.5 Bypass configuration description

Bypass operation is configured by setting the field FTYPE[1:0] of module configuration register DECFILTER_MCR to 00. In this case, the input sample and tag are sent to the output with no change. This behavior is independent of the ISEL/MIXM setting. The following applies to the bypass configuration:

- flush is ignored
- prefill is ignored
- trigger or counted decimation is ignored
- BSY bit is not set
- the input and output flags are set

Note: Bypass must not be configured in cascade mode (see Section 25.5.16, Cascade mode description).

25.5.6 IIR and FIR filter

This section describes the IIR filters implemented in the Decimation Filter block. *Figure 561* shows the filters functional diagrams. The figure shows a IIR filter of fourth order (4 poles). The filter topology, not the hardware, is represented. The hardware implementation is based on a MAC unit controlled by an FSM which implements the filter algorithm.

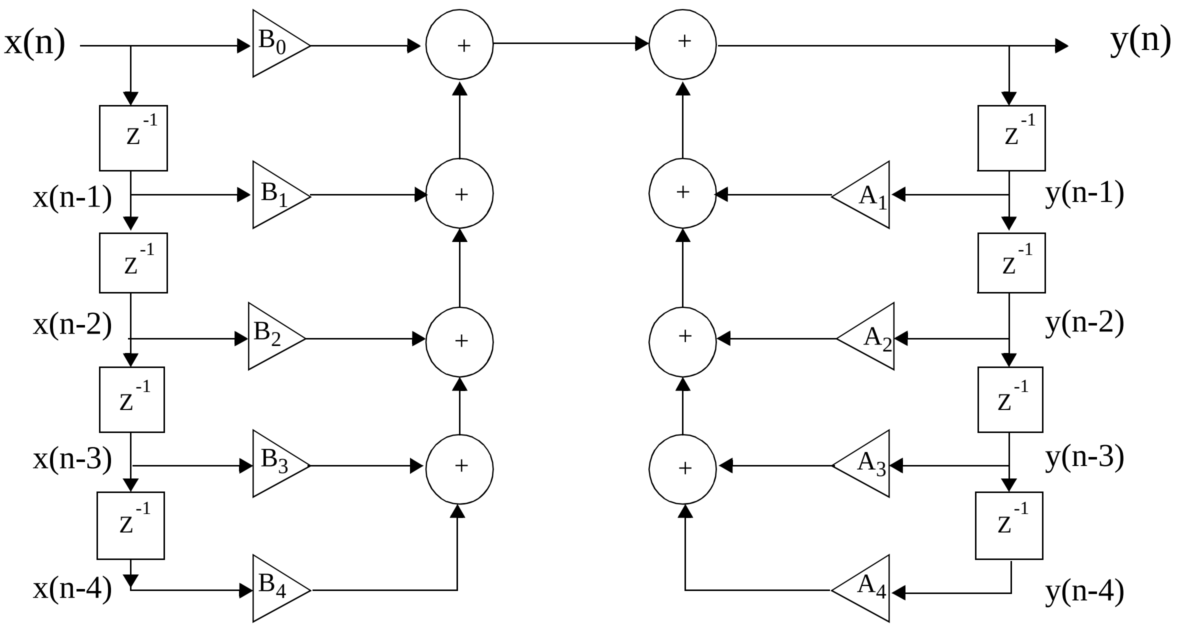


Figure 561. 1 x 4 poles IIR filter functional diagram

The difference equation for the IIR filter of *Figure 561* can be written as:

Equation 17

$$y(n) = \sum_{i=0}^{N} B_i x(n-i) + \sum_{j=1}^{M} A_j y(n-j)$$

where $x(n)$ is the filter input at time n, $y(n)$ is the filter output at time n, N is the number of feed-forward filter coefficients minus one, B_i are the feed-forward filter coefficients, M is the number of feed-back filter coefficients, and A_j are the feedback filter coefficients.

Equation 17 can be written as:

Equation 18

$$w(n) = \sum_{i=0}^{N} \frac{B_i}{S} x(n-i)$$

Equation 19

$$y(n) = S\left\{ w(n) + \sum_{j=1}^{M} \frac{A_j}{S} y(n-j) \right\}$$

Where all the coefficients are scaled down by S. The block diagram for Equation 18 and Equation 19 is shown in Figure 562 in a fourth-order IIR filter implementation where the coefficients A_j and B_i are called *coefficient n*, where n = 0-8.

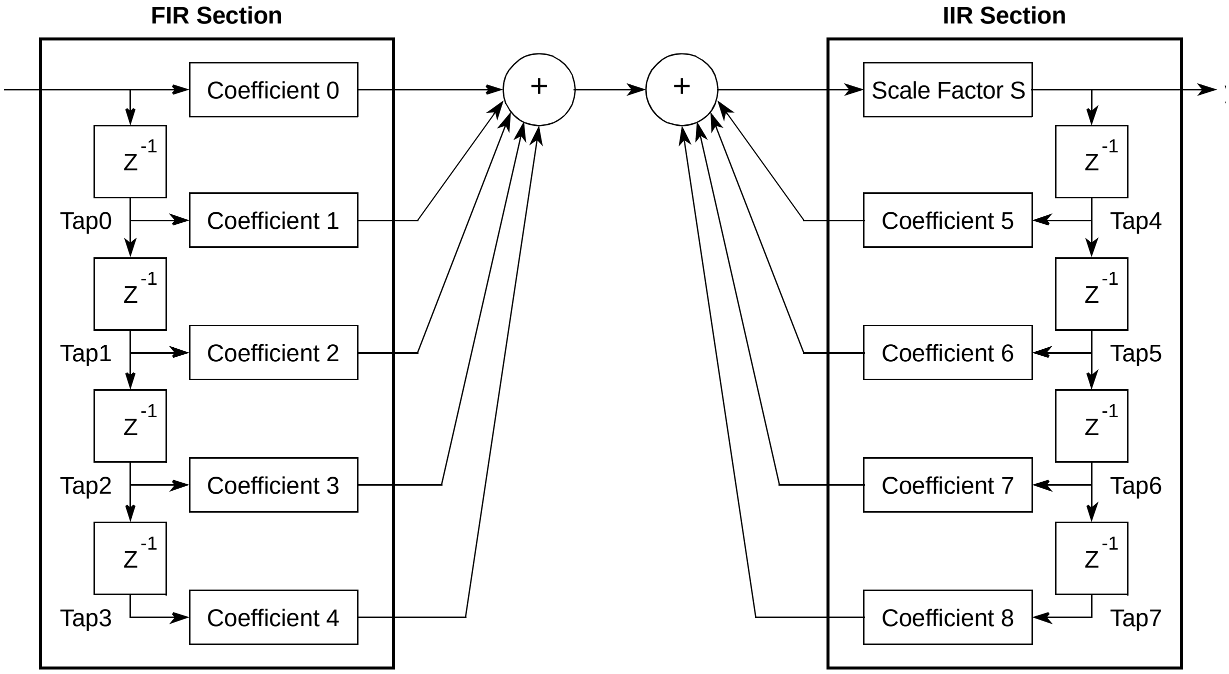


Figure 562. Fourth order IIR filter implementation block diagram

The fourth order IIR filter is implemented with a FIR section followed by an IIR section. If the FIR type filter mode is selected, the IIR section is converted into an FIR section. In this case the order of the FIR filter is twice the IIR filter order, since all the TAP and coefficient registers are allocated for the FIR section. The Filter configuration paths are shown in

Figure 563. Multiplexer A controls the *bypass* filter path and multiplexer B controls/selects the filter mode of operation, to either IIR mode or FIR mode. The selection is controlled by the FTYPE[1:0] bits in the Filter Module Configuration register. The order of the filter can be controlled by setting the appropriate *filter coefficients* to *zero.*

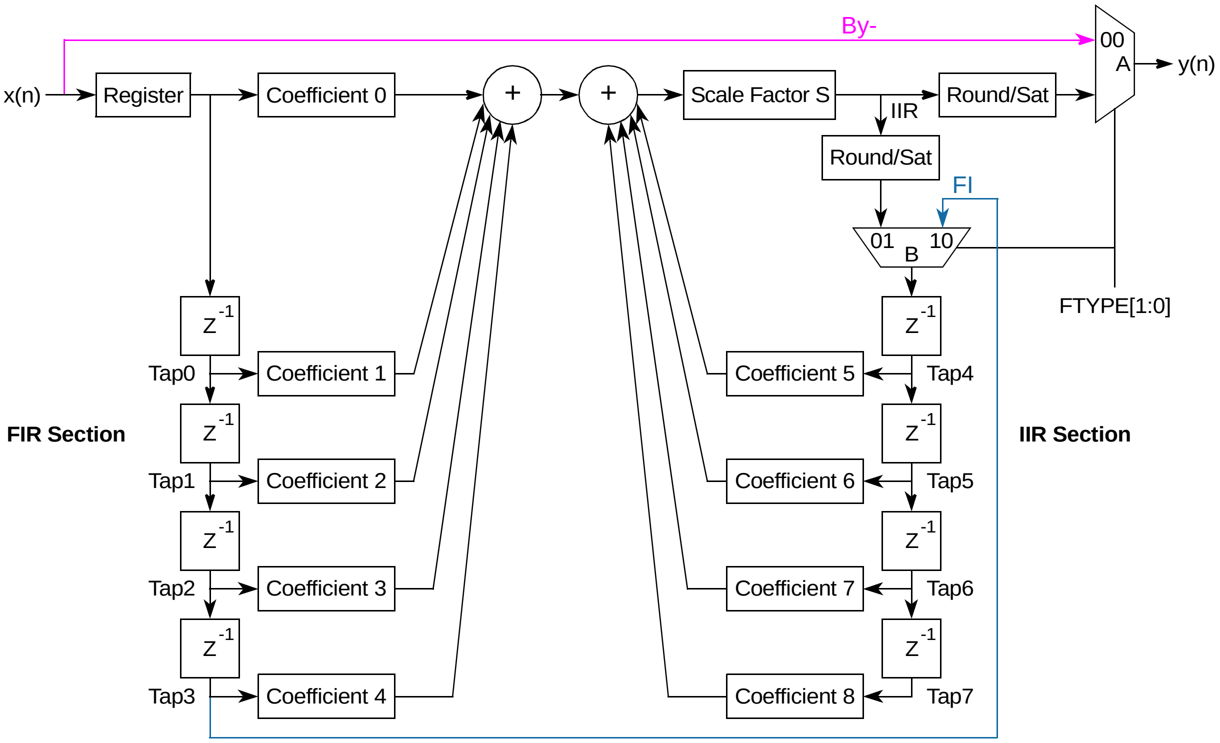


Figure 563. Filter configuration paths (FIR or 1x4 poles IIR)

Rounding

The Decimation Filter performs rounding operations in two different locations, as shown in *Figure 563*:

- to obtain the filter output result with 16 bits
- to obtain the IIR feedback result to be stored in tap4 registers with 24 bits

The rounding mechanism implements the Convergent Rounding methodology (also known as round-to-nearest even number), which makes the decision on rounding up or down based on the value of the lower portion of data to be rounded (LS_WORD). The rounding up/down condition is equal to the traditional rounding except when the LS_WORD has the format {1000...00}. In this particular case, the rounding procedure is like the example of *Figure 564*. If the MS_WORD is odd, the value is rounded up. Otherwise the value is rounded down.

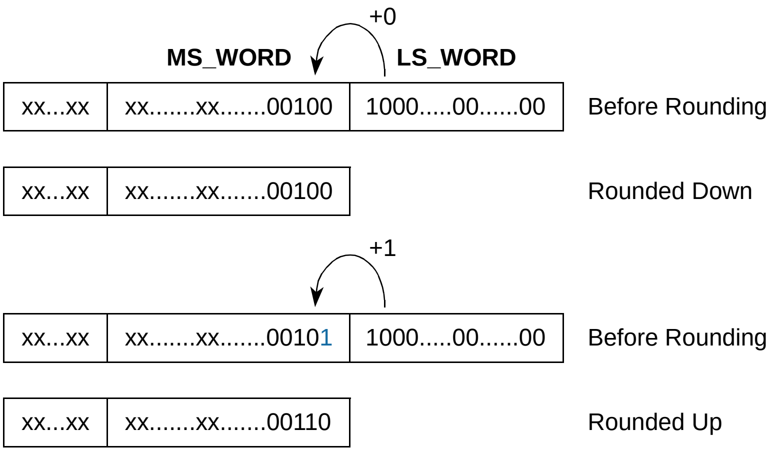


Figure 564. Convergent rounding methodology

Saturation

Filter output saturation occurs when an overflow or underflow condition of the filter is detected by dedicated logic, and if it is enabled by the SAT control bit of the configuration register DECFILTER_MCR. In this condition, the filter output is set to a saturated value equal to the maximum or minimum value that can be represented by the 16-bit output port. Also, for the IIR filter an equivalent logic is used to assert the saturation for the 24-bit feedback result.

25.5.7 Filter prefill control description

A prefill indicates that the input data should be accepted by the Decimation Filter, but no decimated output should be generated while the control field indicates prefill. Therefore the prefill function is used in the beginning of the filter operation to initialize and stabilize the Decimation Filter without generating decimated samples. In addition, the prefill does not operate when the filter is in bypass (FTYPE = 0b00).

The prefill is controlled by the value in the M_CTRL[1:0] field in the DECFILTER_IOB register. When ISEL = 0, the M_CTRL[1:0] field in the DECFILTER_IOB register controls the prefill. When ISEL = 1, it is controlled by the PREFILL field in the DECFILTER_IB register. The prefill control is usually activated only in a certain number of words of sample data in the beginning of the input data sequence.

When the prefill control is set, the decimation filter block operates as follows:

- Input data is processed normally by the digital filter and tap values are updated.
- The decimation counter is maintained in reset value.
- The output buffer is not updated and no output interrupt or read request is generated.
- The accompanying timestamp for the identified prefill conversion data is not bypassed.
- The overflow detector/flag operates normally and the error interrupt request is set if enabled.

In Cascade Mode, prefill control is ignored in all cascaded filters except the Tail one: the prefill samples are filtered, decimated and forwarded to the next block with the prefill indication. The Tail block takes the samples with prefill indication, filter them and discard the result.

Note: The combined decimation count of the cascade combo may not be reset after the last prefill sample. Unlike in a single (non-cascaded) filter, the number of non-prefill inputs until an output comes out depends on the number of past prefill inputs.

25.5.8 Timestamp data transmission

The timestamp information is identified by the master block using the M_CTRL[1:0] bit field of the register DECFILTER_IOB. For timestamp data, the input data and tag values that come with M_CTRL[1:0] bits set to 0b10 are sent back to the master block without changing. However, some additional conditions are considered:

- The timestamp is additional information that accompanies a sample conversion data. The PSI master block sends the decimation filter the conversion data with control bits for either prefill or filter operation. This data may optionally be followed by the corresponding timestamp data. When the corresponding conversion data is marked for prefill, the timestamp data is not sent to the output buffer. This occurs because the filter result is not sent to the output buffer.
- Similarly, when the filter is decimating the results, the timestamp is only sent to the output buffer if the corresponding received conversion data has generated a filter output that is selected by the decimation counter to be sent to the output buffer. Other received timestamps that come with data not selected by the decimator are discarded.
- Sending two consecutive words of timestamp data is not allowed: there must be at least one conversion data between two timestamp inputs. In normal operation, the filter should receive only one timestamp for each word of conversion data to be filtered.
- Timestamps are not allowed in PSI Input Mixed Mode.

25.5.9 Flush command description

The flush signal is used by the Decimation Filter to execute a partial reset of the filter. This is useful when the same filter is used on a new set of data samples after finishing the filtering of another set of data.

When the flush control is detected, all filter TAPs are cleared and the DEC_COUNTER[3:0] field in the status register DECFILTER_MSR is reset.

The flush function does not clear the Coefficient registers file in the Decimation Filter, thus it is not required to re-write these registers after a flush. The integrator accumulator and sample count are not cleared either. The output buffer also keeps the last result and may be retrieved until the next output is posted.

The flush control precedes the input data to be filtered. Therefore, the corresponding sample data is processed by the block after the flush. When ISEL = 0, the field M_FLUSH in the DECFILTER_IOB register is processed. When ISEL = 1, the field FLUSH in the DECFILTER_IB register is processed. Note that a word of valid sample data can be available at the same time the flush signal is asserted. In this case the flush is executed and the sample is processed after the flush.

When ISEL = 0, flush bit M_FLUSH must not be asserted when the input data is a timestamp (M_CTRL = 10).

When the filter is disabled by the FTYPE[1:0] control bit field, the flush command is not executed.

Note: *In Cascade Mode, the flush command is forwarded to the next cascaded block together with the output, after the decimation count. Therefore it is possible that, in a given moment, the taps of a cascaded block are zeroed after a flush input, while the following ones still retain the old values.*

25.5.10 Soft-reset command description

The Soft-Reset command is requested through the self-negated bit SRES of the DECFILTER_MCR register and provides the CPU with the capability to initialize the Decimation Filter through the slave-bus interface. The procedure below must be performed for a software reset when the filter is active:

1. disable filter inputs, writing DECFIL_MCR bit IDIS = 1.
2. poll the register DECFIL_MSR until the bit BSY is 0.
3. repeat the step 2 polling; this is necessary to cover the case when a sample is left in the input buffer.
4. write DECFILTER_MCR bit SRES = 1.

After the software reset is issued, all internal Filter TAP registers, the decimation counter, the integrator outputs (except DECFILTER_CINTCNT) and the state machine are put in the initial state. The status register DECFILTER_MSR is also cleared. The Coefficient registers are not affected by the SRES. In case there is some filter processing, the filter process is aborted and the last sample is discarded. In addition, data in the input buffer waiting to be processed, and data in the output buffer waiting to be read, are discarded (the requests of service are cleared). The software reset command has high priority and the BSY bit is set during its operation.

The configuration registers DECFILTER_MXCR and DECFILTER_MCR are also not affected by a soft reset, except the bit SRES that is self-negated and is always read as zero.

When in debug or freeze mode, the soft reset is executed but the filter remains in debug or freeze mode.

Note: *It is recommended to clear the IBIE bit before a software reset, especially if ISEL changes, in order to avoid unwanted interrupt requests.*

Note: *DMA transfers must not be active during soft reset. Data loss can occur.*

25.5.11 Interrupts requests description

Block interrupt request

There are several interrupt request events that can be enabled using the module configuration register DECFILTER_MCR. Basically, the interrupt request can be issued under any of the following conditions:

- when a word of input data is received
- when a word of output data is available
- when an error has occurred.

The input data flag IDF is set when a word of data is received from the CPU when in standalone mode, or when a word of data is received in the PSI when it is not in standalone mode (normal mode). It is not used to generate the read or write requests (as defined in *Section , Input buffer interrupt request*) when DSEL = 0.

Output data is available and its flag (ODF) is set when the input data sample is processed by the filter and the decimation counter matches the decimation rate value. It is not used to generate the read requests (as defined in *Section , Output buffer interrupt request*) when DSEL = 0.

An error event in the decimation filter block is defined as one of these events:

- Overflow in the filter, flagged by OVF
- Overrun in the decimation filter input, flagged by IVR
- Overrun in the decimation filter output, flagged by OVR
- Overrun in enhanced debug monitor, flagged by DIVR
- Integrator overrun, flagged by SVR
- Integrator value exception, flagged by SSE
- Integrator count exception, flagged by SCE

An overflow occurs when the two's-complement result value from the MAC accumulator is out of the range of values that can be stored in tap register 4 (IIR) or in the output register.

An input overrun occurs when the input buffer is holding a word of input data and one more word of data is received by the filter. See *Section , Input buffer overrun*, for more details.

An output overrun occurs when a new word of data is sent to the output buffer but the previous word of data has not been handled yet. See *Section , Output buffer overrun*, for more details.

These flags can be set by the PSI events, however they are only cleared by

- the CPU, or
- by the soft reset command in the DECFILTER_MCR, or
- by the clear flag fields in the DECFILTER_MSR.

Input buffer interrupt request

This interrupt is enabled by the register IBIE in the DECFILTER_MCR and is asserted only when DSEL = 0. This request is flagged in the DECFILTER_MSR by field IBIF.

In standalone and PSI output mixed modes, the input buffer interrupt request is asserted when the input buffer is available to receive a conversion sample and DMA operation is not selected (DSEL = 0), meaning the block is requesting data to be written into the input buffer. The interrupt request is cleared when the CPU writes a one in field IBIC of the DECFILTER_MSR, or by the soft reset command.

When in normal, cascade or PSI input mixed modes with enhanced debug enabled (ISEL = 0 and EDME = 1, DSEL = 0), the input sample data can be read by the CPU when this interrupt request is asserted. The interrupt is asserted just when a new word of sample data is supplied to the filter sub-block to be processed. As this filter register is overwritten by the next word of sample data, an input read overrun event can occur (the DIVR bit in DECFILTER_MSR is asserted) if the interrupt request is not cleared before, or at the same time as, the new sample arrives to set the interrupt. This DIVR bit is cleared by the DIVRC bit in the status register DECFILTER_MSR. However, in enhanced debug monitoring, the set condition has higher priority than the clear. This means that if the set condition and the clear bit IBIC occur at the same time, the interrupt remains asserted.

Output buffer interrupt request

This interrupt is enabled by the register OBIE in the DECFILTER_MCR and is asserted only when DSEL = 0. This request is also indicated in the field OBIF of the DECFILTER_MSR.

When in standalone mode, the output buffer can be read by the CPU with the DMA disabled. The output buffer interrupt request is asserted when the output buffer receives a new result from the filter sub-block. This means the block is requesting data to be read by the CPU.

The output buffer interrupt request can also be asserted due to an integrator result being ready, as flagged by the DECFILTER_MXSR bit SDF, when the DECFILTER_MCR bit SDIE = 1. Note that both the filter output and integrator share the same interrupt source.

This interrupt request is cleared when a one is written in the bit OBIC and/or bit SDFC of the DECFILTER_MXSR, or by the soft reset command.

25.5.12 DMA requests description

The DMA function for integrator result, input and output buffers is enabled using DSEL = 1 in the DECFILTER_MCR.

Input Buffer DMA request

This DMA request is enabled by the ISEL and DSEL bits in the DECFILTER_MCR.

When in standalone mode, the input buffer can be written by the DMA. The input buffer DMA request is asserted when the input buffer is available to receive a conversion sample (it is not holding a word of data). This DMA request is cleared when an input data word is written in the input buffer. Therefore, the DMA request is always cleared before it is asserted again.

When in normal mode with enhanced debug enabled (ISEL = 0 and EDME = 1, DSEL = 1), the input sample data can be read by the DMA when this DMA request is asserted. The request is asserted when a new word of sample data is written into the input buffer to be processed. As this filter register is overwritten by the next word of sample data, a DMA read overrun event can occur (the DIVR bit in DECFILTER_MSR is asserted) if the DMA request is not cleared before, or at the same time as, a new sample arrives to set the DMA request. This DIVR bit is cleared by the DIVRC bit in the status register DECFILTER_MSR or by soft reset.

This DMA request is cleared when the DMA transfer is complete or by soft reset.

Output buffer DMA request

This DMA request is enabled by the ISEL, MIXM and DSEL bits in the DECFILTER_MCR.

When in standalone mode, the output buffer can be read using the DMA. The output buffer DMA request is asserted when the output buffer receives a new result from the internal filter sub-block. This DMA request is cleared when the output buffer is read by the processor.

The output buffer DMA request is also asserted when an integrator output is ready, if DECFILTER_MXCR bit SDMAE = 1 and the filter output is directed to the PSI interface (ISEL = MIXM).

The DMA request is also cleared by soft reset.

25.5.13 Freeze mode description

The freeze mode operation is asserted using the FREN enable bit and FRZ bit in the DECFILTER_MCR or by the modules debug input.

It is not possible to enter freeze mode when the module is disabled by the configuration bit MDIS.

In case of a freeze mode request during the processing of an input sample, the current processing is finished and then the module enters freeze mode.

Access to input and output buffers remain operational in freeze, as well as their related flags.

Note: *For cascaded blocks (see Section 25.5.16, Cascade mode description), it is recommended to freeze only the Head block. If it is really necessary to freeze all the blocks, it is recommended that they are frozen sequentially starting from the Head block towards the Tail one. Unfreezing is recommended to be done sequentially from the Tail block towards the Head one.*

25.5.14 Enhanced debug monitor description

This feature is enabled by the EDME bit in the configuration register DECFILTER_MCR. The monitoring operation works either in normal or cascade modes, when the sample data is supplied by a master block through the PSI slave-bus interface (ISEL = 0) or another cascaded block (see *Section 25.5.16, Cascade mode description*). The Enhanced Debug Monitor feature makes the input sample data also available in the DECFILTER_EDID register. A DMA or interrupt request (selected by DSEL) indicates a new input was fed and DECFILTER_EDID was updated. The input is processed normally by the filter.

An Enhanced Debug Input Data Register (DECFILTER_EDID) overrun can occur if a sample is not read by the CPU or DMA before overwritten by a new sample. The overrun is indicated in a separate flag DIVR in the status register DECFILTER_MSR. If the ERREN bit is set in the DECFILTER_MCR configuration register, this overrun asserts the module interrupt request.

25.5.15 Integrator

The hardware sample integrator accumulates the filter output values for determined periods.

Integrator inputs

The integrator can be fed either by raw or decimated filter outputs, selected by the DECFILTER_MXCR bit SISEL (see *Section , Decimation Filter Module Extended Configuration Register (DECFILTER_MXCR)*). The accumulated input value taken can be the filtered sample “as is” (signaled), or its absolute value, depending on the DECFILTER_MXCR bit SSIG.

Note: *The integrator accumulates input samples when bypass is selected.*

Integrator outputs

The integrator output is either:

- the 32-bit, fixed point unsigned accumulation of the absolute values from the filter output, when the integrator is configured for absolute operation (DECFILTER_MXCR bit SSIG = 0). This resolution allows a minimum of 131071 samples to be integrated before an overflow occurs in absolute operation, or 65536 samples in signed operation.

- the 32-bit, fixed point signed two's complement accumulation of the signed values from the filter output, when the integrator is configured for signed operation (SSIG = 1).

The fractional part of the accumulation is 15 bits wide in both cases.

An accumulation overflow is flagged by the DECFILTER_MXSR bit SSOVF. The accumulator can overflow in either of the ways below, selected through the DECFILTER_MXCR bit SSAT:

- saturated accumulation (SSAT = 1), so that an overflow results in the value of 0xFFFFFFFF for absolute value accumulation (SSIG = 0), or 0x7FFFFFFF (positive) and 0x80000000 (negative) for signaled accumulation (SSIG = 1).
- non-saturated accumulation (SSAT = 0), so that an overflow results in the modulo 2^{17} accumulation value. This operation is only allowed in absolute accumulation (SSIG = 0).

The integrator output value becomes available in register DECFILTER_FINTVAL (see *Section , Decimation Filter Final Integration Value Register (DECFILTER_FINTVAL)*) when an integrator output request is issued. The integrator output request can be issued in the following ways:

- by hardware, controlled by an external signal; the enabling and the selection of the signal request modes is done through the DECFILTER_MXCR field SRQSEL (see *Section , Decimation Filter Module Extended Configuration Register (DECFILTER_MXCR)*);
- by software, writing 1 to the DECFILTER_MXCR bit SRQ;

The SSOVF flag is asserted upon an integrator output request, based on the overflow state of the internal accumulator. This internal overflow state is cleared upon an output request, just after SSOVF is asserted, or upon an integrator reset (see *Section , Integrator reset*). The internal overflow state is also cleared by writing SSOVFC to 1, but only in saturated accumulation. Therefore, a non-saturated overflow that occurs before an SSOVF clear is still flagged in the next output request.

The integrator output request also updates the register DECFILTER_FINTCNT, which holds the number of samples accumulated into the register DECFILTER_FINTVAL. This internal accumulated sample counter can operate either in a saturated or "wrapped" count mode, as selected by the DECFILTER_MXCR bit SCSAT. In both cases, the counter overflow is flagged by the DECFILTER_MXSR bit SCOVF.

The SCOVF flag is asserted upon an integrator output request, based on the overflow state of the internal counter. This internal overflow state is negated upon an output request, just after SCOVF is asserted, or upon an integrator reset (see *Section , Integrator reset*). The internal overflow state is also negated by writing SCOVFC to 1, but only in saturated count. Therefore, a non-saturated overflow that occurs before an SCOVF clear is still flagged in the next output request.

An integrator output update can also issue a DMA or interrupt request. The interrupt and DMA requests are the same ones used for the filter output buffer (see *Section , Output buffer interrupt request* and *Section , Output buffer DMA request*). The DECFILTER_MCR bit SDIE is used to enable integrator interrupts, and the DECFILTER_MXCR bit SDMAE enables the DMA integrator requests. The integrator DMA request uses the same signal as the filter output DMA request, so one must never use any configuration that allows both the integrator and filter output to make DMA requests.

Integrator output updates are flagged by the DECFILTER_MXSR bit SDF. The integrator overrun is detected in the same way as a filter output buffer overrun, and is flagged by DECFILTER_MXSR bit SVR. An integrator overrun also generates an error interrupt if the DECFILTER_MCR bit ERREN = 1 (see *Section , Output buffer overrun*).

Registers DECFILTER_CINTVAL and DECFILTER_CINTCNT provide a way to poll intermediate integration values and sample counts, respectively (see *Section , Decimation Filter Current Integration Value Register (DECFILTER_CINTVAL)* and *Section , Decimation Filter Current Integration Count Value Register (DECFILTER_CINTCNT)*). DECFILTER_CINTVAL is updated whenever the integrator is reset or a new sample is accumulated. DECFILTER_CINTCNT is updated only when DECFILTER_CINTVAL is read, so that coherency between the value and count values is guaranteed. Therefore, the read access order of that pair of registers must be DECFILTER_CINTVAL first, followed by DECFILTER_CINTCNT.

Note: The flags SSOVF and SCOVF can also asserted when DECFILTER_CINTVAL is read. The SSOVF and SCOVF set and clearing rules apply for the DECFILTER_CINTVAL read the same way as for an integrator output request.

Integrator reset

The integration value is reset to the value of zero, in the following ways:

- by hardware: on hardware reset, or controlled by an external signal; the enabling and the selection of the zero signal modes is done through the DECFILTER_MXCR field SZROSEL (see *Section , Decimation Filter Module Extended Configuration Register (DECFILTER_MXCR)*);
- by software: on software reset, or writing 1 to the DECFILTER_MXCR bit SZRO;

The integrator reset also zeroes the internal counter of accumulated samples and the internal overflow state (but not SSOVF and SCOVF). Software and hardware reset resets all integrator registers immediately. Integrator zero command from external signal or by software (SZRO) affects the integrator registers and flags as follows:

- DECFILTER_CINTVAL resets immediately;
- DECFILTER_CINTCNT does not reset immediately; it is updated only upon a DECFILTER_CINTVAL read, loaded with the number of integrated samples occurred after the reset;
- DECFILTER_FINTVAL and DECFILTER_FINTCNT do not reset immediately; being updated only upon a new output request (see *Section , Integrator outputs*); if a integrator software zero command (through SZRO bit) and an integrator output request (through SRQ bit) are made at the same time, the registers DECFILTER_FINTVAL and DECFILTER_FINTCNT are updated with the last internal values before reset; the same applies to simultaneous integrator zero command and output request by hardware signal;
- the SSOVF and SCOVF flags do not negate; however, the internal overflow states which assert SSOVF and SCOVF do reset immediately, so that the next output update (either by hardware request, software request or DECFILTER_CINTVAL read) before an overflow does not assert SSOVF/SCOVF.

Note: The integrator reset does not depend on the integrator enabling (see Section , Integrator enabling and halting).

Integrator enabling and halting

Two mechanisms, enabling and halting, drive the integrator accumulation, allowing it to be controlled by a combination of two distinct sources, both software, both hardware, or one hardware and other software. Values are accumulated when the integrator is enabled and not halted. The integrator halt and enable states can be controlled in the following ways:

- by hardware, through external signals; the enabling and the selection of the signal request modes is done through the DECFILTER_MXCR fields SENSEL and SHLTSEL, respectively (see *Section , Decimation Filter Module Extended Configuration Register (DECFILTER_MXCR)*);
- by software, through the same DECFILTER_MXCR fields SENSEL and SHLTSEL. Note that these fields are in different bytes, so that two distinct, concurrent software tasks can avoid coherency problems by changing the fields using byte read-modify-write accesses.

Note: Enabling and halting does not affect output requests or integrator reset.

Integrator exceptions

Integrator may run into exception states due to overflow, either of the accumulated value or the sample counter. Exceptions are flagged by the DECFILTER_MXSR bits SSE, for sum value exception, and SCE, for counter exception. These flags generate an error interrupt, if it is enabled (see *Section , Block interrupt request*).

The accumulator exception condition depends on whether it operates in saturated mode or not, as follows:

- In Saturated operation (DECFILTER_MXCR bit SSAT = 1): a sum exception occurs (SSE = 1) whenever an overflow is flagged; SSE asserts together with SSOVF.
- In Non-saturated operation (DECFILTER_MXCR bit SSAT = 0): a sum exception occurs (SSE = 1) when an overflow is flagged and the DECFILTER_MXSR bit SSOVF is already set to 1.
- In Non-saturated operation, an accumulator exception also occurs if the accumulator overflows twice without any update of the final integrator value DECFILTER_FINTVAL or the current integrator counter DECFILTER_CINTCNT (by a read to the DECFILTER_CINTVAL register), neither an integrator reset occurs. The SSOVF flag does not assert in this situation.

Note: The SSOVF flag can only be asserted upon a hardware request, a software request, or when DECFILTER_CINTVAL is read, based on the internal accumulator overflow state.

Similarly, the sample counter exception condition depends on whether it operates in saturated mode or not, as follows:

- In Saturated operation (DECFILTER_MXCR bit SCSAT = 1): a counter exception occurs (SCE = 1) whenever an overflow is flagged; SCE asserts together with SCOVF.
- In Non-saturated operation (DECFILTER_MXCR bit SCSAT = 0): a counter exception occurs (SCE = 1) when an overflow is flagged and the DECFILTER_MXSR bit SCOVF is already set to 1.
- In Non-saturated operation, a counter exception also occurs if the counter overflows twice without any update of the final count DECFILTER_FINTCNT or the current integrator counter DECFILTER_CINTCNT (by a read to the DECFILTER_CINTVAL register), neither an integration reset occurs. The SCOVF flag does not assert in this situation.

Note: *The Scovf Flag Can Only Be Asserted Upon A Hardware Request, A Software Request, Or When Decfilter_cintval Is Read (Also Updating Decfilter_cintcnt), Based On The Internal Counter Overflow State.*

25.5.16 Cascade mode description

The cascade mode is defined as a configuration mode of the decimation filter to work together with other ones in a chain arrangement. All blocks in the arrangement, hereafter called a **cascade combo**, are configured to operate in cascade mode by the CASCD[1:0] field in the DECFILTER_MCR. *Figure 565* shows an example of cascade combo:

- The figure shows PSI being used for both data input and output, but cascade can also work in standalone or mixed modes.
- The leftmost block, named Head. receives the raw data to be filtered from the PSI master block (or from the device slave-bus interface in standalone/PSI output modes). The head type is configured using CASCD[1:0] = 01 in the configuration register DECFILTER_MCR.
- The rightmost block, named Tail, is the last filter block in the chain. It sends the output result back to the PSI master block (or to the device slave-bus interface in standalone/PSI input modes). This type of cascaded block is configured using CASCD[1:0] = 10.
- The blocks in between, named Middle, do not exchange data (receive / transmit) with the PSI master, only with other decimation filter blocks. This type of block is configured by setting CASCD[1:0] = 11. Middle blocks are optional in a cascaded combo: two blocks, one Head block feeding a Tail one, can be used in cascade.

Note: *The values passed between cascaded blocks can be monitored using Enhanced Debug Monitor (see Section 25.5.14, Enhanced debug monitor description).*

- Each decimation filter block has one cascade-in and one cascade-out ports, besides the PSI connections used in normal mode. The format of the cascade bus is described in *Section , Cascade Mode Data/Control Bus description*. The arrows show how they are physically connected. The bold arrows show the connections used in cascade mode.
- The block configurations as Head, Tail or Middle must respect their physical connections such that all the following apply:
  - a Head block feeds a Middle or Tail one
  - a Middle block feeds another Middle block or a Tail one
  - Tail feeds no other block, and Head is fed by no other block.
- As a consequence of the conditions above, there must be one and only one Head block and one and only one Tail block in a cascade combo.
- A group of physically chained blocks can form more than one cascade combo. For instance, *Figure 566* shows the same physical chain, but now with blocks 1 and 2 configured as Head and Tail, respectively, forming one combo. The remaining blocks form another combo starting with block 3 (Head), and ending with block N (Tail). Note that the chain-in inputs of blocks from block 3 on are used to carry output data from the first cascaded combo (Tail block 2) to the PSI master block.
- Blocks not used in a cascaded chain can be used normally, isolated (DEFILTER_MCR field CASCD[1:0] = 00), as exemplified in *Figure 567*.
- The optional connection show from block N to block 1 in *Figure 565* allows block N to be configured as Head or Middle, feeding block 1 configured as Middle or Tail, yielding

more flexibility, as in the last example of *Figure 567*.

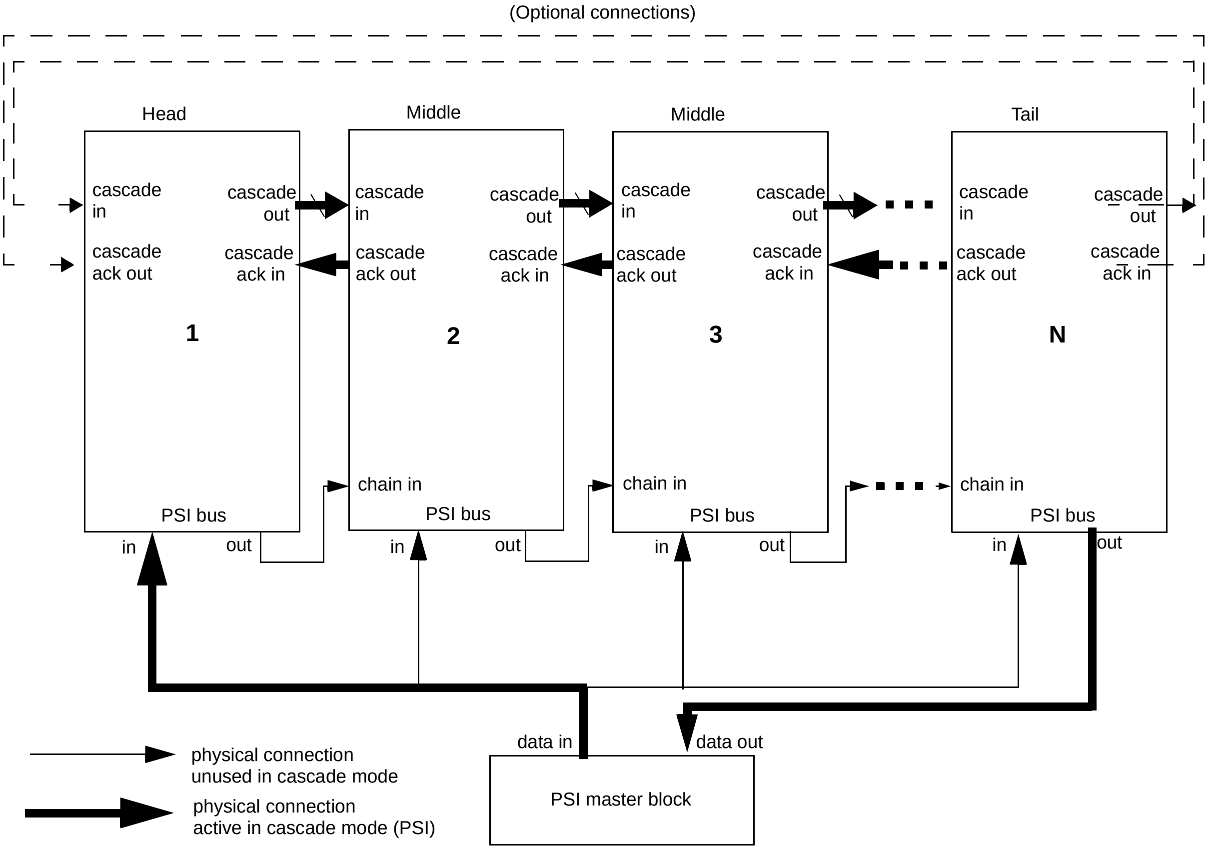


Figure 565. Cascade mode chain structure

The input to a cascaded configuration is selected by the DECFILTER_MCR bit ISEL of the Head block. The output target of the cascaded blocks is selected by the DECFILTER_MCR bit ISEL of the Tail block, with the same values used for input selection (ISEL = 0 for PSI, ISEL = 1 for device slave-bus). DECFILTER_MCR bit MIXM must be written 0 for all cascaded blocks.

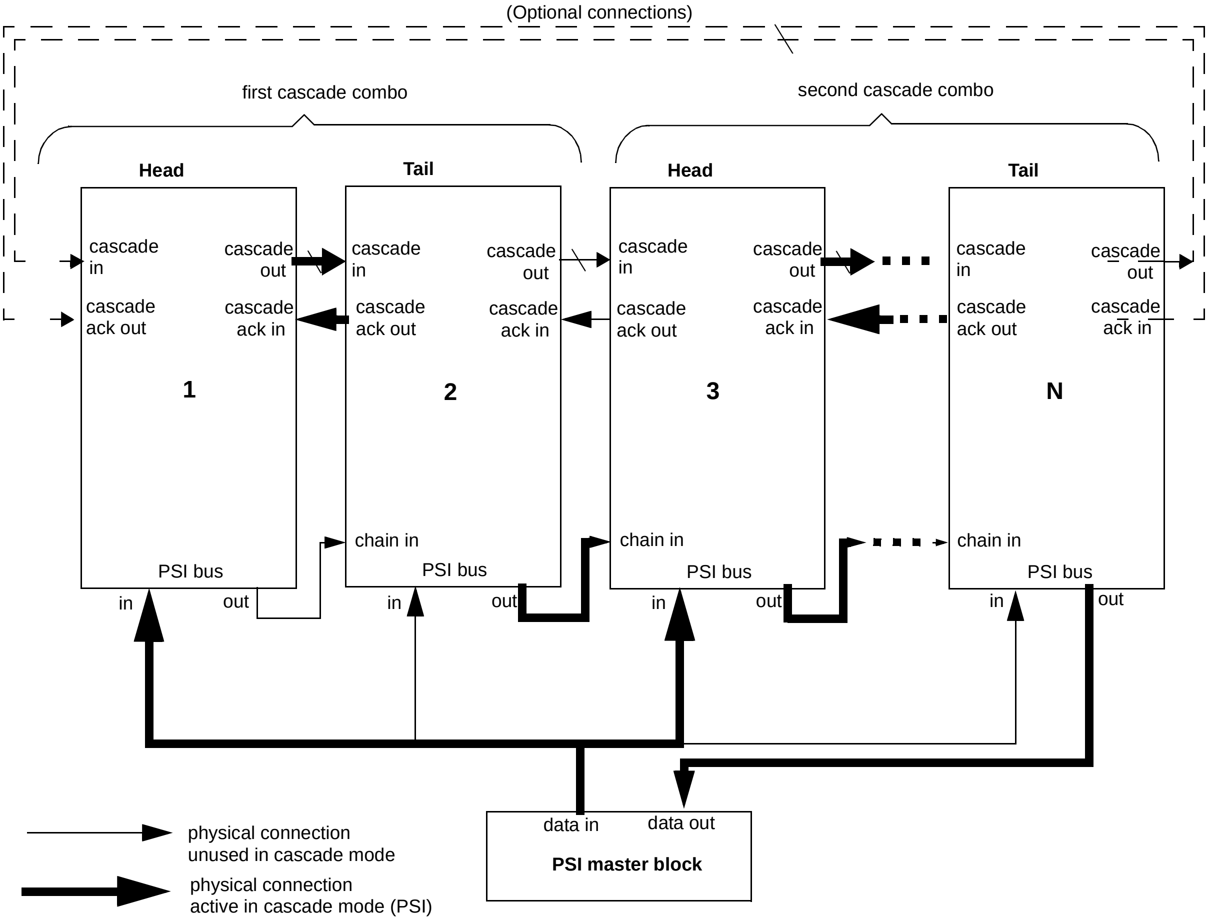


Figure 566. Multiple cascade mode chain structure

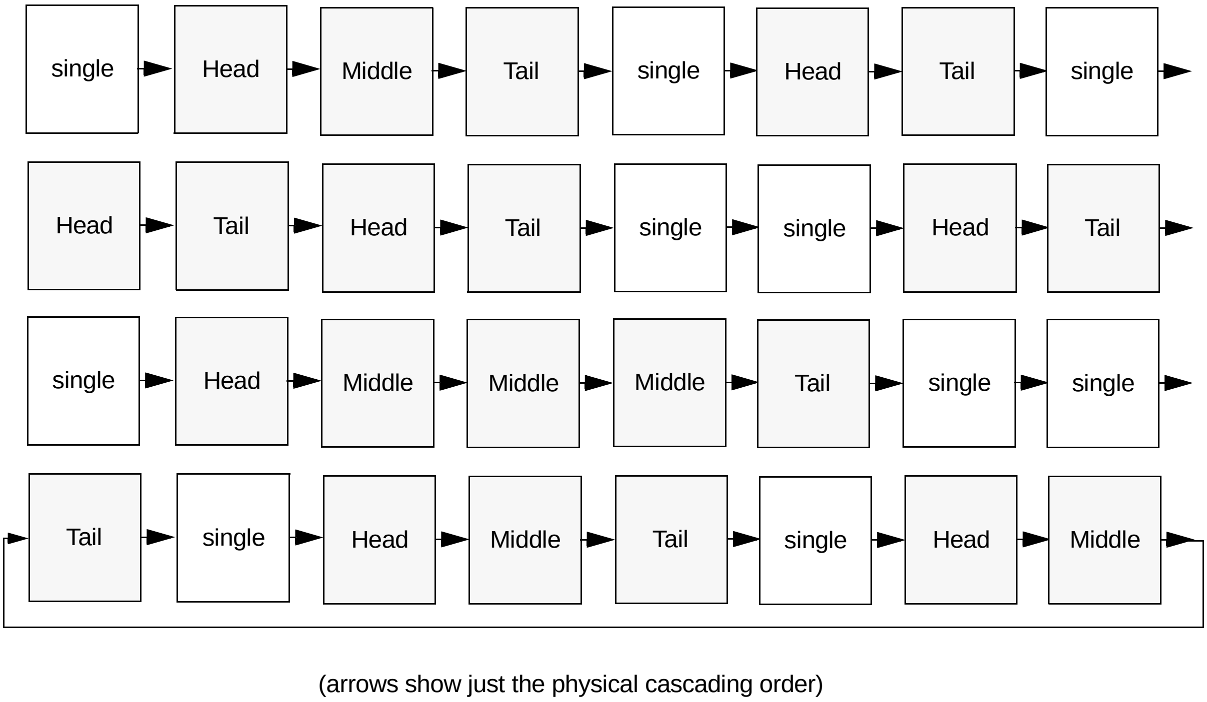


Figure 567. Examples of mixed cascaded and single blocks

Cascade freeze, stop, and configuration change procedures

To change a block configuration mode to or from cascade mode, the following safe procedures must be observed:

- To modify a cascade combo, either to single or any other cascade combo combination, all the cascade combo blocks must have their inputs disabled (using DECFILTER_MCR bit IDIS), in order, from the Head to the Tail block. After a block IDIS bit has been set to 1 (one), one must wait for its DECFILTER_MSR bit BSY to be 0 (zero) before disabling the input of the next block in the sequence.
- Each block in a new cascade combo must be configured with its input disabled. When the mode configuration is done, the combo blocks must have their inputs enabled in order, from the Tail towards the Head block.
- A single block must also be reconfigured the same way, to or from a cascade combo configuration: first disabling its input, and then waiting for a non-busy state before writing DECFILTER_MCR field CASCD.

To take cascade combo blocks to or from freeze or low power modes, a similar procedure must be used:

- Take the Head to freeze or low-power first, wait for DECFILTER_MSR bit BSY = 0, and repeat the procedure for the other blocks in the chain in sequence, towards the Tail block.
- Take the blocks out of freeze or low-power modes in the inverse sequence, from Tail to Head.

Cascade Mode Data/Control Bus description

A separate data bus is used to cascade the filters. The bus content is presented in *Figure 568* and the signals descriptions are present below.

Figure 568. Decimation filter cascade mode data bus

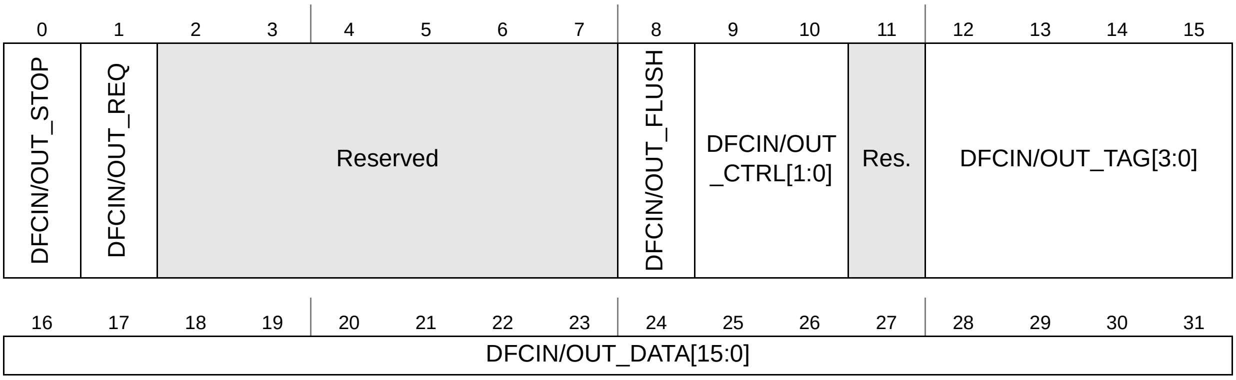


Table 584. Decimation filter cascade mode data bus field description

Field	Description
0 DFCIN/OUT_STOP	Decimation Filter Cascade Input/Output Stop flag The DFCIN/OUT_STOP bit indicates that a cascade bus driver block in a cascade configuration is stopped. When the block is configured as Cascade Middle or Tail, it only stops when this bit is asserted.
1 DFCIN/OUT_REQ	Decimation Filter Cascade Request The DFCIN/OUT_REQ bit indicates that a cascade bus driver block in a cascade configuration has data ready to be sent. The driven block responds to the request asserting its decfil_cascade_ack signal at the same time it copies the relevant cascade data bus fields.
8 DFCIN/OUT_FLUSH	Decimation Filter Cascade Input/Output Flush control bit The DFCIN/OUT_FLUSH bit indicates to the receiver Decimation Filter block that it should execute a flush command — thus some internal registers are placed in the initial state.
9-10 DFCIN/OUT_CTRL [1:0]	Decimation Filter Cascade Input/Output Control bits The DFCIN/OUT_CTRL[1:0] field has the same function as the M_CTRL[1:0] control bits described in *Table 583*. This field defines the operation to be executed with the DFCIN/OUT_DATA[15:0] data.
12-15 DFCIN/OUT_TAG [3:0]	Decimation Filter Cascade Input/Output Tag bits The DFCIN/OUT_TAG[3:0] field indicates the destination associated with the DFCIN/OUT_DATA[15:0] data. This value is stored by the Decimation Filter and used to address the destination register when a decimated sample is available to be read by the master block.
DFCIN/OUT_DATA [15:0]	Decimation Filter Cascade Input/Output Data The DFCIN/OUT_DATA[15:0] bit field carries the data to be transmitted in the chain of cascaded decimation filter blocks.

Cascade Middle and Tail blocks do not make input feed requests, either on PSI or slave-bus interfaces. Similarly, cascade Head and Middle block do not make filtered (not integrator) output data requests, either on PSI or slave-bus interfaces.

Cascaded blocks can be configured with different filter types (DECFILTER_MCR field FTYPE), including bypass. The *Table 585* shows how Decimation Filter features work in each of the Cascade Mode configurations.

Table 585. Features in cascade mode

Feature	Head	Middle	Tail
Prefill	Output and prefill command are forwarded to the next cascaded block		Effective
Flush	Effective, and forwarded to the next cascaded block		Effective
Decimation	Effective in each block		
Timestamp	Forwarded to the next cascaded block		Effective
Enhanced Debug	Effective in each block		
Integrator	Effective in each block		

25.6 Initialization information

Following are some simple initialization steps to be done before using the decimation filter block. These steps assume that the user has already calculated the filter coefficients using a filter design tool.

25.6.1 Initialization procedure

The sequence of steps for the block initialization is as follows:

1. Program the configuration registers DECFILTER_MCR and DECFILTER_MXCR as desired for your application.
2. Write all filter coefficient registers DECFILTER_COEFn with the previously calculated values.
3. Enable the filter input, writing DECFILTER_MCR bit IDIS = 0.
4. Run a soft-reset cycle if necessary.
5. The module is ready to receive data from PSI or from the device slave-bus interface.

25.7 Application information

25.7.1 eQADC IP as the PSI master block

The system block diagram for the eQADC application is shown in *Figure 569*. In this case, the Decimation Filter receives conversion results generated by the eQADC block. These results can be generated from eight different ADC setup configurations which are identified by a specific eQADC Control address within a Conversion command. Conversion commands with Register Address set to zero use the standard configuration setup. The samples generated by the standard configuration setup are sent to one of the local eQADC RFIFO buffers. The samples generated by the Alternate Configurations, with an address from 1–8, can be sent to the internal RFIFO or to the eQADC dedicated slave-bus interface (Parallel Side Interface PSI) to communicate with the external Decimation Filter IP block or any other block that can communicate with this interface. A bit field in the Alternate Configuration Control Register selects the Internal RFIFO or this slave-bus interface as the destination for the conversion result. The eQADC can send either conversion data or timestamp data. The conversion data is filtered by the decimation filter and the timestamp is bypassed and sent back to the eQADC.

In the eQADC application, the TAG field is used to address the appropriate RFIFO in the eQADC block. In this case, only addresses 0–5 are used since there are only six RFIFOS available in the eQADC block.

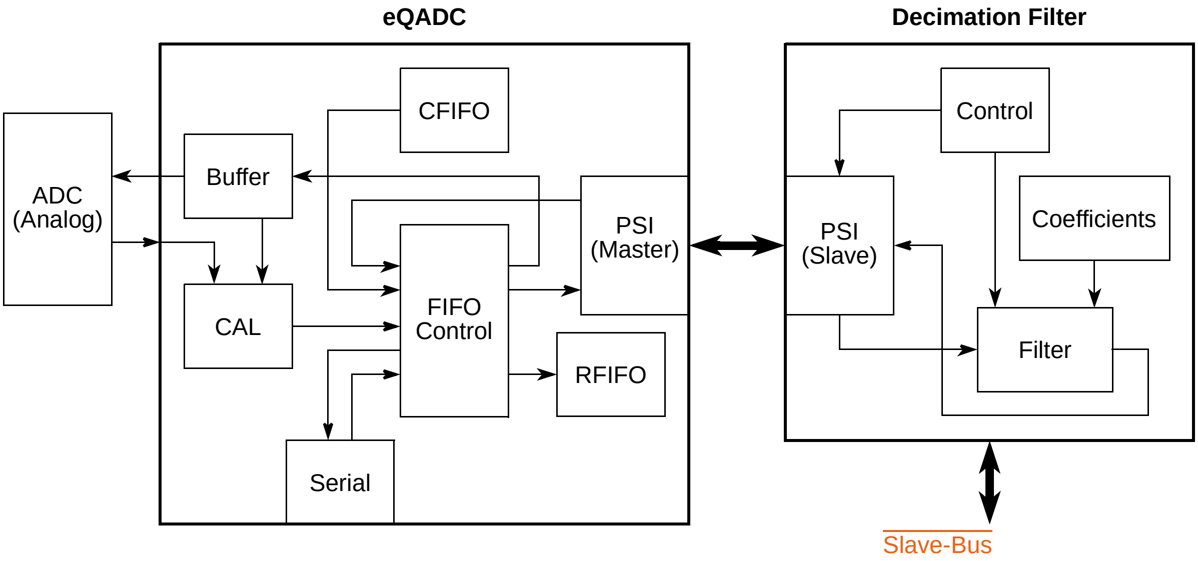


Figure 569. Decimation filter/eQADC interface

25.8 Filter example simulation

The decimation filter block operation was checked in a Verilog simulation using calculated filter coefficient values and noisy input data. The expected output values and the RMS error were then calculated.

25.8.1 Coefficients calculation

The coefficients were calculated using a digital filter design tool. We have supplied some hypothetical filter parameters to the tool and obtained the filter coefficients. The input parameters are:

- Filter characteristics: elliptic/low pass
- Filter type: 4th order IIR
- Input sample rate: 800k sample/s
- Passband edge: 100 kHz
- Stopband edge: 150 kHz
- Passband attenuation: £ 1 dB

The software tool gives the IIR filter coefficients in the Z-transform format expressed by *Equation 20*:

Equation 20

$$\frac{Y(s)}{X(s)} = \frac{B_0 + B_1 s + B_2 s^2 + B_3 s^3 + B_4 s^4}{A_0 - A_1 s - A_2 s^2 - A_3 s^3 - A_4 s^4}$$

The coefficient results in fixed point decimal representation are shown in *Table 586*.

Table 586. Coefficient values given by SPW digital filter design tool

Coefficient	Decimal value	Coefficient	Decimal value
B0	0.0221455	A0	1.0
B1	0.00445582948893748	A1	-2.69772868375858
B2	0.0318517846509088	A2	3.234056294853
B3	0.00445582948893748	A3	-1.92028561712454
B4	0.0221455	A4	0.47939080709495

Comparing *Equation 19* with *Equation 20*, we obtain the relationship between the calculated values of coefficients and the values to be loaded in the DECFILTER_COEFn registers. See *Table 587* below to obtain the coefficients. A scale factor of eight is used, being the smallest divider factor to have all coefficient values in the range (–1 £ Coef < +1). Also note that the signal of the An coefficients has signals inverted.

Table 587. Coefficient values for decimation filter

SCALE	S = 1	S = 8	
COEFn	Decimal value	Decimal value	Hexadecimal values (24 bits)
Coef0 = B0/S	0.0221455	0.00276815891266	0x005AB5
Coef1 = B1/S	0.00445582948893748	0.00055694580078	0x001240
Coef2 = B2/S	0.0318517846509088	0.00398147106171	0x008277
Coef3 = B3/S	0.00445582948893748	0.00055694580078	0x001240
Coef4 = B4/S	0.0221455	0.00276815891266	0x005AB5
Coef5 = –A1/S	2.69772868375858	0.33721613883972	0x2B29E6
Coef6 = –A2/S	–3.234056294853	–0.40425717830658	0xCC414E
Coef7 = –A3/S	1.92028561712454	0.24003565311432	0x1EB97D
Coef8 = –A4/S	–0.47939080709495	–0.05992400646210	0xF8546A

25.8.2 Input data calculation

The 24-bit words of input data are samples of a sum of two tones: one tone of 30 kHz and another tone of 250 kHz. The samples were calculated at the rate of 800k samples per second. The tones have the same amplitude and it is assured that the resultant amplitude is smaller than 1 so as to obtain samples in the range (–1 ≤ sample < +1). It is supposed the input data are signed values in the two's complement format in the range (–1 ≤ sample < +1).

25.8.3 Filter results

The decimation filter block was used in a Verilog simulator using the calculated coefficients and the input data samples. A scaling factor of eight in the configuration register DECFILTER_MCR, and no decimation factor, were used to obtain the maximum of output results from the filter.The theoretical expected values from this filter were also calculated, and these results were compared with those from the decimation filter. The resultant RMS error when considering about 500 samples was about –97 dB.

26 Cyclic Redundancy Checker (CRC) Unit

26.1 Overview

The CRC module provides a fast on-chip capability for verifying code and data integrity. This capability is particularly important in safety applications. Examples include:

- Verifying memory integrity by setting it to a known value, calculating a checksum and comparing the calculated checksum against a stored checksum value
- Verifying code integrity by comparing its calculated checksum to its stored checksum value
- Verifying the integrity of data received from a network by comparing its received checksum to its calculated checksum

CRC functionality can be implemented in software but there are significant speed advantages to be gained by offloading CRC computation tasks from the processor core to the CRC module. Further gains are made when data is written to the CRC module via DMA.

Note: *This chapter does not discuss the details of computing CRC checksums but there are many articles to be found via internet searches. One that might be of particular interest is “A Painless Guide to CRC Error Detection Algorithms” by Ross Williams.*

26.2 Features

The CRC module on the SPC564A70 includes the following features:

- 3 “contexts”—A context is a CRC engine with its own independent set of configuration and data registers. The SPC564A70 CRC module can process up to three separate data streams concurrently.
- Each context supports CRC-16-CCITT and CRC-32 ethernet polynomials
- Bit-swap and bit-inversion operations can be applied on the final CRC signature
- Support for byte/half-word/word width of the input data stream
- Computation is performed with zero wait states

26.2.1 Access and performance

All CRC registers are accessible (read/write) in each access mode: user, supervisor or test.

The following bus operations (contiguous byte enables) are supported:

- 32-bit data read/write operations to any register
- Low and high half-word read/write operations to any register(az)
- Byte data write/read operations to any register(ba)

---

az. 16-bit operations must be aligned to 16-bit boundaries, i.e., bits 0–15 or bits 16–31. Any unaligned operation results in a bus error.

ba. Byte operations must be aligned to 8-bit boundaries, i.e., bits 0–7, bits 8–15, bits 16–23, or bits 24–31. Any unaligned operation results in a bus error.

Bus performance of the operations is as follows:

- Zero wait state (single bus cycle) for each read/write to the CRC_CFG and CRC_INP registers
- Zero wait state (single bus cycle) for each write operation to the CRC_ CSTAT register
- Double wait state (3 bus cycles) for each read operation to the CRC_ CSTAT or CRC_OUTP registers immediately following (next clock cycle) a write operation to the CRC_CSTAT, CRC_INP or CRC_CFG registers belonging to the same context. In all the other cases no wait states are inserted.

The following will result in transfer errors:

- Unaligned reads or writes
- Any attempt to read or write an address that is assigned to the CRC module but not actually mapped to a register.

26.3 Calculating a CRC checksum

The SPC564A70 CRC module has three independent sets of CRC engines and registers, each set called a context. Each context supports a single data stream, structured as a sequence of bytes, half-words or words, written to its input register. Since the context operate independently, the CRC module can process up to three data streams concurrently.

Figure 570 illustrates the steps to calculating a CRC checksum (also called a signature) for a data stream:

1. Configure the context to be used.
2. Write a seed value into the CRC Current Status Register (CRC_CSTAT).
3. Write the data to the CRC Input Register (CRC_INP), until the end of the data to be checked.
4. Retrieve the calculated checksum from the CRC Output Register (CRC_OUTP) and verify the checksum against a stored value.

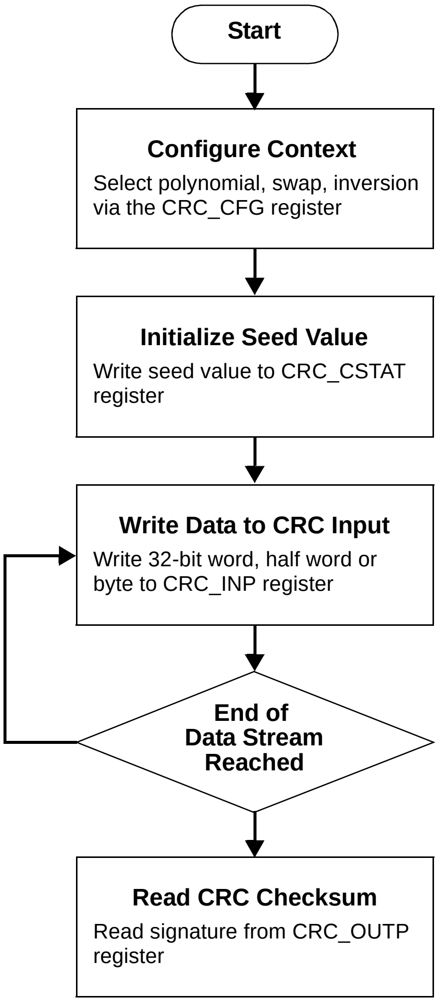


Figure 570. CRC checksum processing flow

The following sections describe each step in the process.

26.3.1 Configuring the context

A context consists of a CRC engine and a dedicated set of registers. The SPC564A70 CRC module includes three contexts.

The configuration step consists of:

- Selecting the polynomial
- Specifying whether a swap operation is to be performed on the output
- Specifying whether a bit inversion is to be performed on the output

Selections are made by writing the appropriate values to fields in the CRC_CFG register.

Two standard polynomials are provided by the CRC module: CRC-16-CCITT (x25 protocol) and CRC-32 (ethernet protocol). They are illustrated in *Equation 21* and *Equation 22*.

$$X^{16} + X^{12} + X^{5} + 1$$

Equation 21 CRC-CCITT (x25 protocol)

$$X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^{8} + X^{7} + X^{5} + X^{4} + X^{2} + X + 1$$

Equation 22 CRC-32 (ethernet protocol)

The polynomial to be used is based on system requirements.

In case of usage of the CRC signature for encapsulation in the data frame of a communication protocol (e.g., SPI) a bit swap (MSB → LSB, LSB → MSB) and/or bit inversion of the final CRC signature can be applied (CRC_OUTP register).

26.3.2 Initializing the context seed value

A CRC checksum can be thought of as the remainder of a division of a long, arbitrary number (the data stream) by a known fixed value. The known fixed value is known as the seed value. The same seed value must be used to generate the checksums that are to be compared to each other.

The seed value is specified in the CRC current status register (CRC_CSTAT), which as a dual purpose. Before CRC checksum calculation is performed, i.e., during the configuration phase, the CRC_CSTAT register is used to program the seed value. During CRC checksum calculation, the register contains the current checksum value.

The seed value can be any arbitrary 32-bit value.

Note: As with the CRC configuration register (CRC_CFG) the CRC_CSTAT register can only be written during the configuration phase. A write protection error generated by a write operation to this register indicates it is currently in use.

26.3.3 Writing the data stream to the context input

After the context is configured and a seed is written, the data stream is written to the context’s input register (CRC_INP). The CRC_INP register can be written at byte, half-word (high and low) or word in any sequence. In case of half-word write operation, the bytes must be contiguous.

Note: The CRC_INP register only supports aligned writes. Half-word (16-bit) writes must be either to bits 0–15 or 16–31. Byte writes must be to bits 0–7, bits 8–15, bits 16–23, or bits 24–31.

The writes can be by the processor core or by DMA transfer. The writes continue until the end of the data stream is reached.

26.3.4 Reading the checksum

After writing of the data stream to the input register has been completed, the checksum is read from the output register (CRC_OUTP). The CRC_OUTP register includes the final checksum (signature) corresponding to the CRC_CSTAT register value with swap and inversion operations applied, if selected via the CRC_CFG register.

In case of CRC-16-CCITT polynomial only the 16 least significant bits have meaning. The 16 most significant bits are set to 0 during the computation.

26.4 Register descriptions

Table 588. CRC register map

Context	Address[1]	Register	Location
1	CRC_BASE + 0x0000	CRC Configuration Register (CRC_CFG)	*on page 26-1081*
	CRC_BASE + 0x0004	CRC Input Register (CRC_INP)	*on page 26-1082*
	CRC_BASE + 0x0008	CRC Current Status Register (CRC_CSTAT)	*on page 26-1083*
	CRC_BASE + 0x000C	CRC Output Register (CRC_OUTP)	*on page 26-1084*
2	CRC_BASE + 0x0010	CRC Configuration Register (CRC_CFG)	*on page 26-1081*
	CRC_BASE + 0x0014	CRC Input Register (CRC_INP)	*on page 26-1082*
	CRC_BASE + 0x0018	CRC Current Status Register (CRC_CSTAT)	*on page 26-1083*
	CRC_BASE + 0x001C	CRC Output Register (CRC_OUTP)	*on page 26-1084*
3	CRC_BASE + 0x0020	CRC Configuration Register (CRC_CFG)	*on page 26-1081*
	CRC_BASE + 0x0024	CRC Input Register (CRC_INP)	*on page 26-1082*
	CRC_BASE + 0x0028	CRC Current Status Register (CRC_CSTAT)	*on page 26-1083*
	CRC_BASE + 0x002C	CRC Output Register (CRC_OUTP)	*on page 26-1084*

1. CRC_BASE for the SPC564A70 is 0xFFE6_8000

26.4.1 CRC Configuration Register (CRC_CFG)

Figure 571. CRC Configuration Register (CRC_CFG)

Offset: PMC_BASE + 0x0000
PMC_BASE + 0x0010
PMC_BASE + 0x0020

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	POLYG	SWAP	INV
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0/1(1)	0	0

1. Reset value is 1 for Context 2 and 0 for Context 1 and Context 3.

Table 589. CRC_CFG field descriptions

Field	Description
0–28	Reserved
29 POLYG	POLYG: *Polynomial selection* 0: CRC-CCITT polynomial. 1: CRC-32 polynomial. This bit can be read and written by software. This bit can be written only during the configuration phase.
30 SWAP	SWAP: *SWAP selection* 0: No swap selection applied on the CRC_OUTP content 1: Swap selection (MSB ⟶ LSB, LSB ⟶ MSB) applied on the CRC_OUTP content. In case of CRC-CCITT polynomial the swap operation is applied on the 16 LSB bits. This bit can be read and written by software. This bit can be written only during the configuration phase.
31 INV	INV: *INV selection* 0: No inversion selection applied on the CRC_OUTP content 1: Inversion selection (bit x bit) applied on the CRC_OUTP content. In case of CRC-CCITT polynomial the inversion operation is applied on the 16 LSB bits. This bit can be read and written by software. This bit can be written only during the configuration phase.

26.4.2 CRC Input Register (CRC_INP)

Figure 572. CRC Input Register (CRC_INP)

PMC_BASE + 0x0004
Offset: PMC_BASE + 0x0014
PMC_BASE + 0x0024

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	INP															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	INP															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 590. CRC_INP field descriptions

Field	Description
0–31	INP: *Input data for the CRC computation* The INP register can be written at byte, half-word (high and low) or word in any sequence. In case of half-word write operation, the bytes must be contiguous. This register can be read and written by software.

26.4.3 CRC Current Status Register (CRC_CSTAT)

Figure 573. CRC Current Status Register (CRC_CSTAT)

PMC_BASE + 0x0008
Offset: PMC_BASE + 0x0018
PMC_BASE + 0x0028

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CSTAT															
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CSTAT															
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Table 591. CRC_CSTAT field descriptions

Field	Description
0–31	CSTAT: *Status of the CRC signature* The CSTAT register includes the current status of the CRC signature. No bit swap and inversion are applied to this register. In case of CRC-CCITT polynomial only the16 least significant bits have meaning. The 16 most significant bits are tied to 0 during the computation. The CSTAT register can be written at byte, half-word or word. This register can be read and written by software. This register can be written only during the configuration phase.

26.4.4 CRC Output Register (CRC_OUTP)

Figure 574. CRC Output Register (CRC_OUTP)

Offset: PMC_BASE + 0x000C
PMC_BASE + 0x001C
PMC_BASE + 0x002C

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	OUTP															
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	OUTP															
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Table 592. CRC_OUTP field descriptions

Field	Description
0–31	OUTP: *Final CRC signature* The OUTP register includes the final signature corresponding to the CRC_CSTAT register value after swap/inversion operations, if specified. In case of CRC-CCITT polynomial only the16least significant bits have meaning. The 16 most significant bits are tied to 0 during the computation. This register can be read by software.

26.5 Use cases and limitations

Two main use cases are considered:

- Calculation of the CRC of the configuration registers during the process safety time
- Calculation of the CRC on the incoming and outgoing frames for the communication protocols (not protected with CRC by definition of the protocol itself) used as a safety-relevant peripheral.

26.5.1 Checksums for configuration registers

The checksum (signature) of configuration registers is computed in a correct way only if these registers do not contain any status bits, i.e., configuration register contents must not dynamically change during, or as a result of, a CRC checksum calculation.

26.5.2 Calculations on incoming/outgoing protocol frames

The following sections show the sequence for managing CRC checksums as part of a communication external to the device.

Calculating checksums on data to be transmitted

Figure 575 illustrates the sequence used to calculate a CRC checksum on a data stream, append the checksum and transfer it to the peripheral to be used for transmission. The sequence is as follows:

1. Software configures the DMA channel and CRC context to be used.
2. DMA copies the data to be transmitted to the CRC context’s input register (CRC_INP) to calculate the CRC signature (Phase 1)
3. Software copies the CRC checksum (signature) from the CRC module (CRC_OUTP register) to the memory location immediately following the transmission data. (Phase 2)
4. DMA transfers the data block (payload + CRC checksum from memory to the peripheral module (e.g., SPI Tx FIFO) (Phase 3)

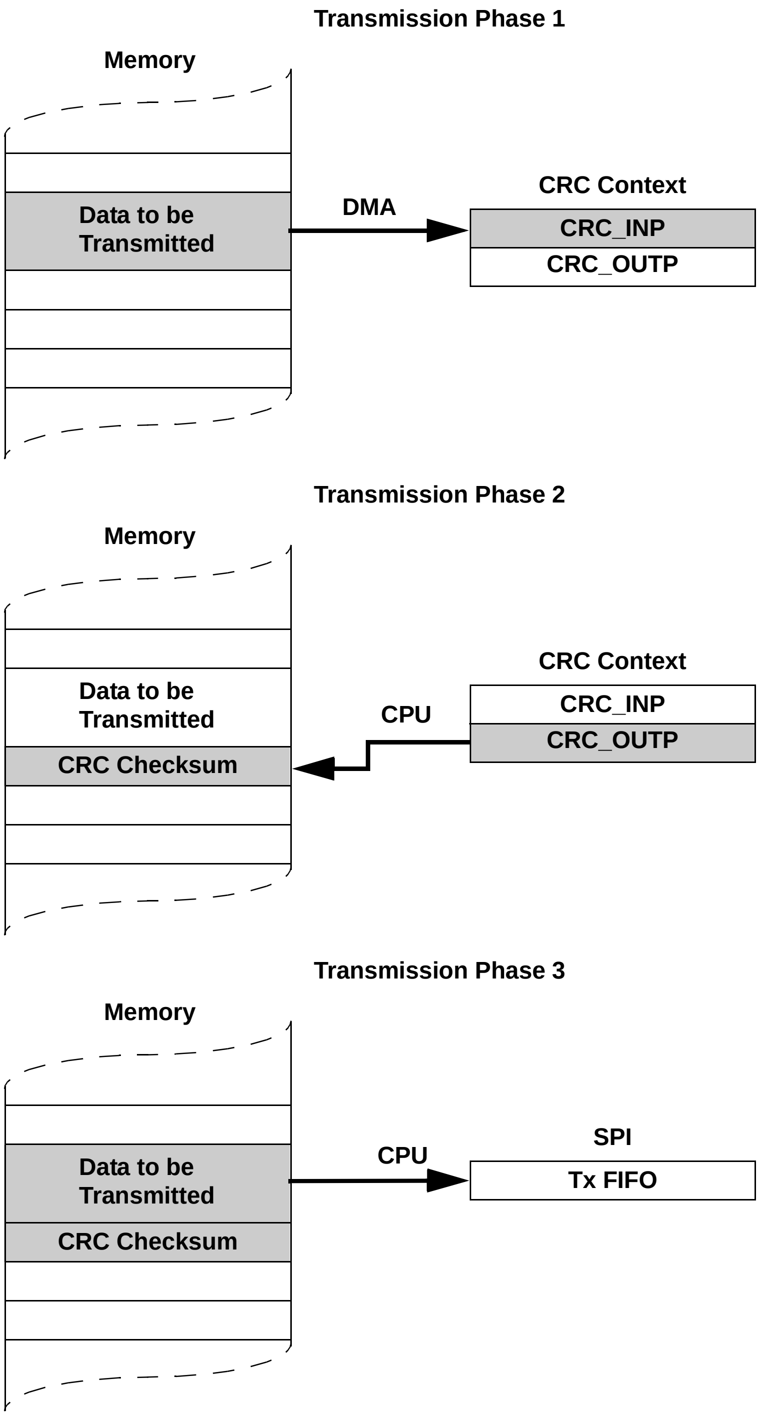


Figure 575. Transmission sequence

Calculating checksums on received data

Figure 576 illustrates the sequence used to calculate a CRC checksum on a received data stream. The sequence is as follows:

1. Software configures the DMA channel and /CRC context to be used
2. DMA copies the received data block (payload + CRC) from the peripheral (e.g., SPI Rx FIFO) module to memory (Phase 1)
3. DMA copies the received data block transfer (payload + CRC) from memory to the CRC context (CRC_INP register) to calculate the CRC signature (Phase 2)
4. The CRC signature is read from the CRC context (CRC_OUTP register) by software (Phase 3)

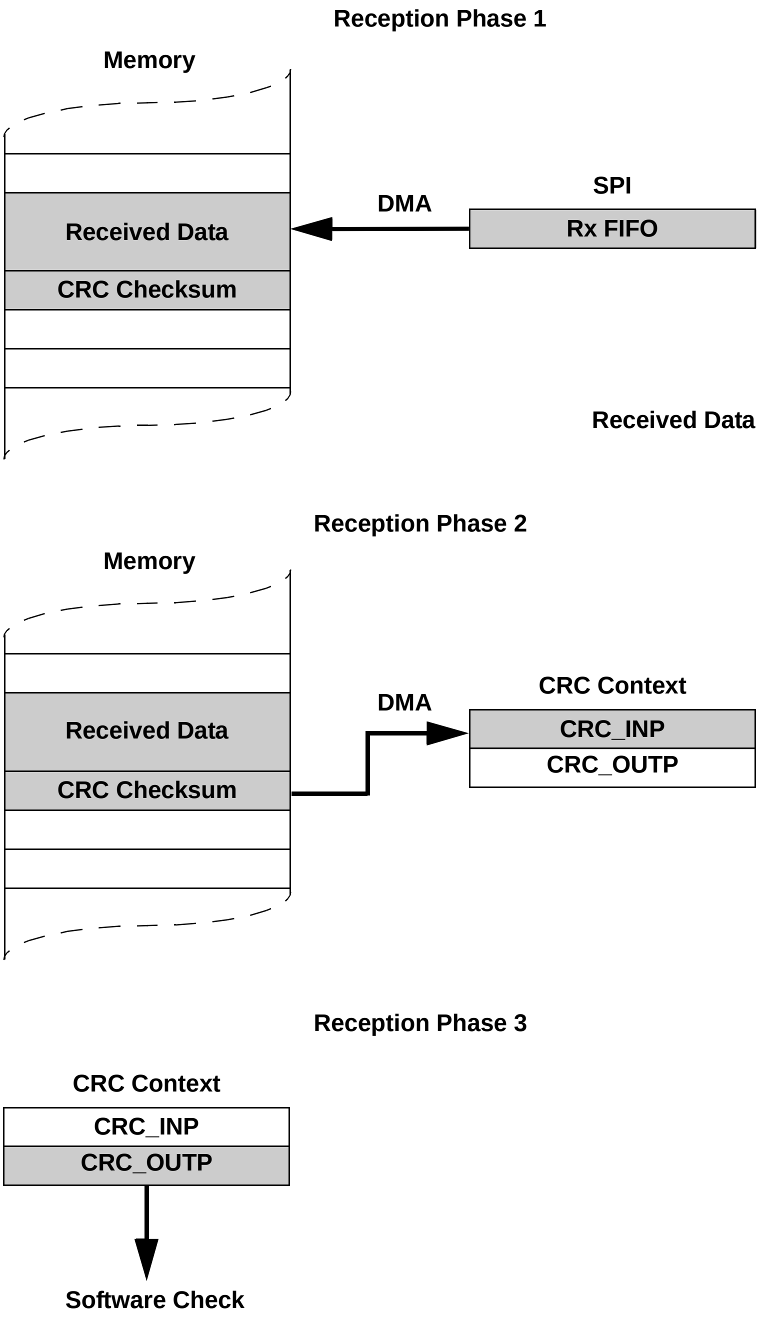


Figure 576. Reception sequence

27 Deserial Serial Peripheral Interface (DSPI)

27.1 Introduction

Figure 577 is a block diagram of the Deserial Serial Peripheral Interface (DSPI) module.

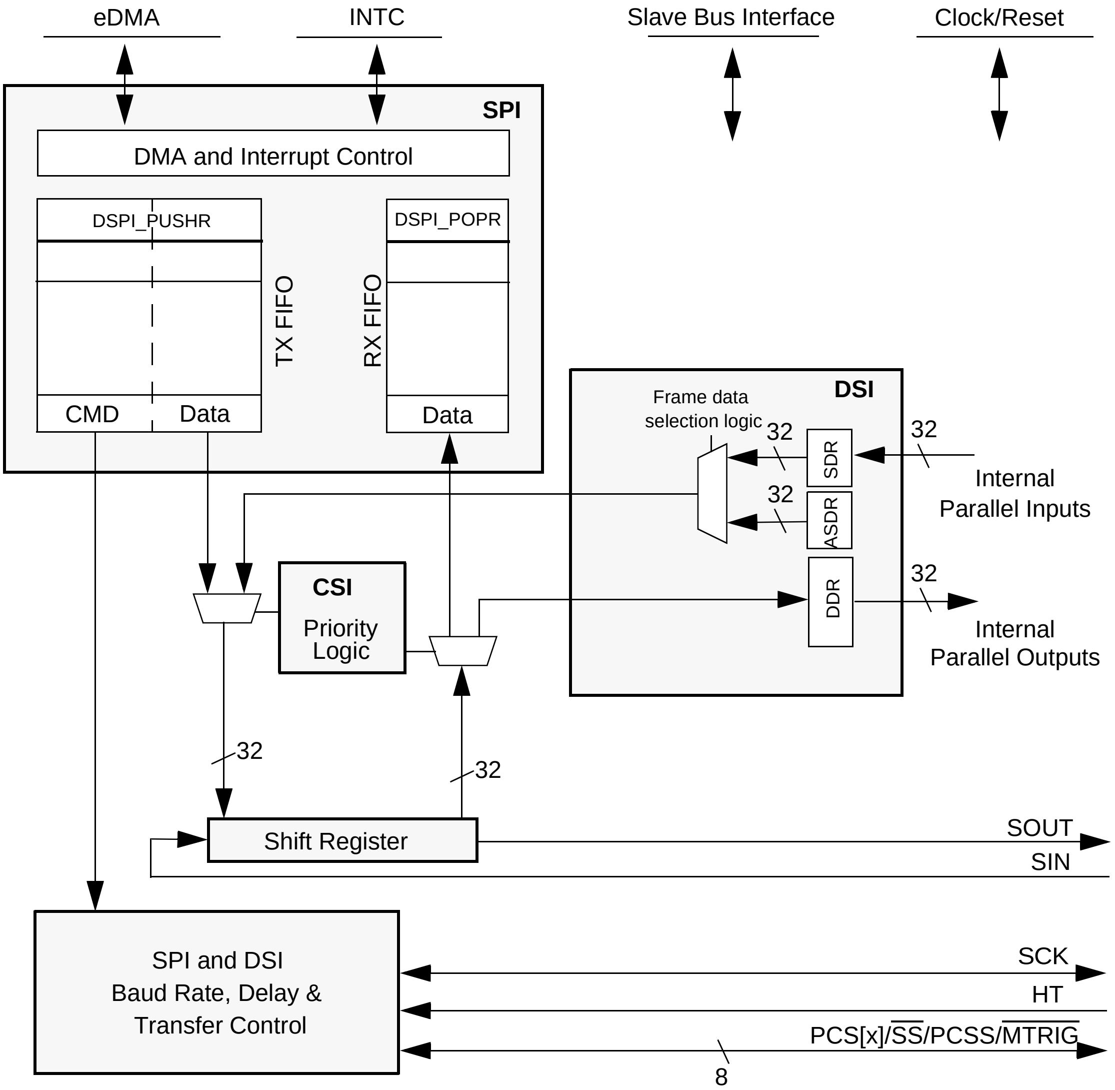


Figure 577. DSPI block diagram

27.2 Overview

The Deserial Serial Peripheral Interface (DSPI) module provides a synchronous serial interface for communication between the SPC564A70 and external devices. The DSPI supports pin count reduction through serialization and deserialization of eTPU channels, eMIOS channels and memory-mapped registers. Incoming deserialized data can also be used to trigger external interrupt requests. The channels and register content are transmitted using a SPI-like protocol. There are three identical DSPI modules (DSPI_B, DSPI_C and DSPI_D) on the SPC564A70.

The DSPIs have three configurations:

- Serial Peripheral Interface (SPI)—DSPI operates as a SPI with support for queues
- Deserial Serial Interface (DSI)—DSPI serializes eTPU and eMIOS output channels and deserializes the received data by placing it on the eTPU and eMIOS input channels and as inputs to the External Interrupt Request sub-block of the SIU
- Combined Serial Interface (CSI)—DSPI operates in both SPI and DSI configurations interleaving DSI frames with SPI frames, giving priority to SPI frames

For queued operations, the SPI queues reside in system memory external to the DSPI. Data transfers between the memory and the DSPI FIFOs are accomplished through the use of the eDMA controller or through host software.

27.3 Features

The DSPI supports these SPI features:

- Full-duplex, synchronous transfers
- Selectable LVDS Pads working at 40 MHz for SOUT and SCK pins (only in DSPI_B and DSPI_C)
- Master and Slave Mode
- Buffered transmit operation using the TX FIFO with depth of 4 entries
- Buffered receive operation using the RX FIFO with depth of 4 entries
- TX and RX FIFOs can be disabled individually for low-latency updates to SPI queues
- Visibility into the TX and RX FIFOs for ease of debugging
- FIFO Bypass Mode for low-latency updates to SPI queues
- Programmable transfer attributes on a per-frame basis:
  - Parameterized number of transfer attribute registers (from 2 to 8)
  - Serial clock with programmable polarity and phase
  - Various programmable delays:

    PCS to SCK delay

    SCK to PCS delay

    Delay between frames
  - Programmable serial frame size of 4 to 32 bits, expandable with software control
  - Continuously held chip select capability
- 8 Peripheral Chip Selects, expandable to 256 with external demultiplexer
- Deglitching support for up to 128 Peripheral Chip Selects with external demultiplexer
- DMA support for adding entries to TX FIFO and removing entries from RX FIFO:
  - TX FIFO is not full (TFFF)
  - RX FIFO is not empty (RFDF)
- 6 interrupt conditions:
  - End of queue reached (EOQF)
  - TX FIFO is not full (TFFF)
  - Transfer of current frame complete (TCF)
  - Attempt to transmit with an empty Transmit FIFO (TFUF) ‘OR’ Serial frame received while RX FIFO is full (RFOF). These two interrupts are ORed and given out as FIFO Overrun interrupt.
  - RX FIFO is not empty (RFDF)
  - FIFO Underrun (slave only and SPI mode, the slave is asked to transfer data when the Tx FIFO is empty)
- Modified transfer formats for communication with slower peripheral devices
- Continuous Serial Communications Clock (SCK)
- Power-saving architectural features
  - Support for IPI Green-line Stop Mode
- Enhanced DSI logic to implement a 32-bit Timed Serial Bus (TSB) configuration, supporting the Micro Second Channel (MSC) bus downstream frame format

The DSPIs also support these features unique to the DSI and CSI configurations:

- 2 sources of the serialized data:
  - eTPU_A and eMIOS output channels
  - Memory-mapped register in the DSPI
- Destinations for the deserialized data:
  - eTPU_A and eMIOS input channels
  - SIU External Interrupt Request inputs
  - Memory-mapped register in the DSPI
- Deserialized data is provided as Parallel Output signals and as bits in a memory-mapped register
- Transfer initiation conditions:
  - Continuous
  - Edge sensitive hardware triggered
  - Change in data
- Pin serialization/deserialization with interleaved SPI frames for control and diagnostics
- Continuous serial communications clock
- Support for parallel and serial chaining of up to 3 DSPI modules
- Parity generation and checking

27.4 DSPI configurations

The DSPI module can operate in three configurations: SPI, DSI and CSI.

27.4.1 SPI configuration

The SPI configuration allows the DSPI to send and receive serial data. This configuration allows the DSPI to operate as a basic SPI block with internal FIFOs supporting external queues operation. Transmit data and received data reside in separate FIFOs. The host CPU or a DMA controller read the received data from the receive FIFO and write transmit data to the transmit FIFO.

For queued operations the SPI queues can reside in system RAM, external to the DSPI. Data transfers between the queues and the DSPI FIFOs are accomplished by a DMA controller or host CPU. *Figure 578* shows a system example with DMA, DSPI and external queues in system RAM.

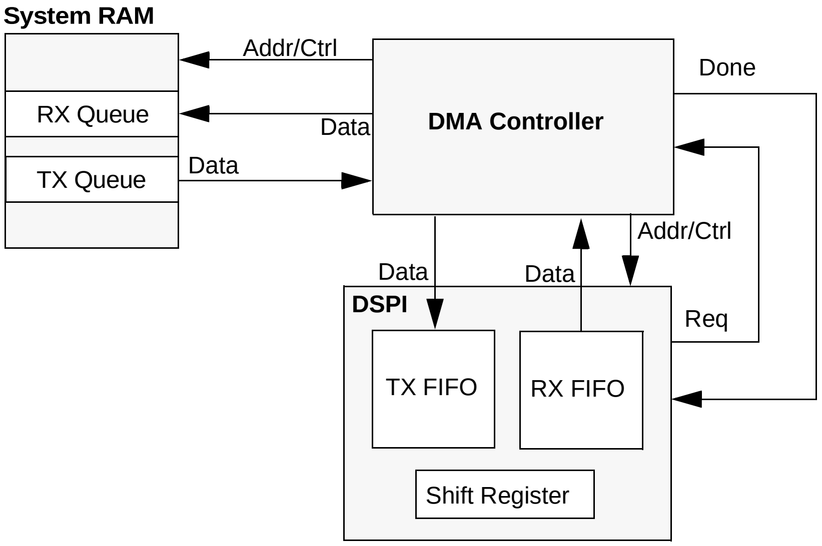


Figure 578. DSPI with queues and DMA

27.4.2 DSI configuration

The DSI configuration supports pin count reduction by serializing eTPU and eMIOS output channels or bits from a memory-mapped register and shifting them out with a SPI-like protocol. The DSPI deserializes the received data, and provides the received data to the eTPU's and eMIOS' input channels, the SIU IRQ inputs, or to a memory-mapped register in the DSPI. See *Section 27.9.17, DSPI connections to eTPU_A, eMIOS and SIU* for the source of the serialization data for each DSPI module.

Figure 579 shows an example of how a master DSPI block connects to a DSI slave in DSI configuration.

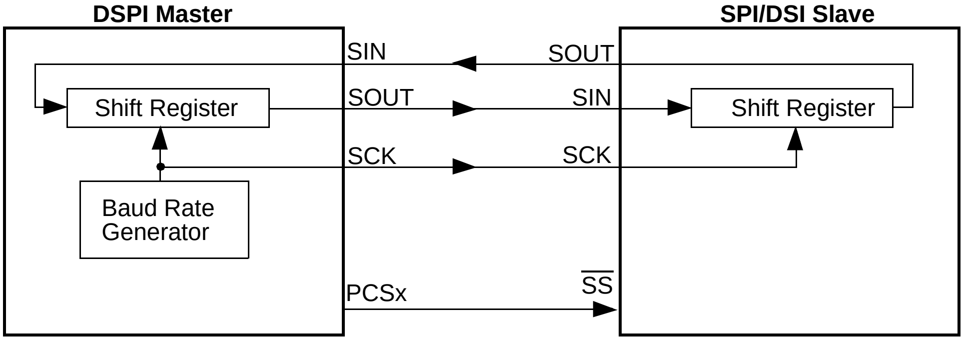


Figure 579. DSPI connections for SPI and DSI transfers

Specifically in the TSB configuration, detailed in *Section 27.9.8, Timed serial bus (TSB)*, the DSPI serializes from 4 to 32 Parallel Input signals or register bits. The TSB downstream frame used to communicate with a single slave is shown in *Figure 614*.

27.4.3 CSI configuration

The CSI configuration allows serialized data to be interleaved with configuration or diagnostic data and be transferred to a slave device using only one serial link. The CSI configuration supports SPI and DSI functionality on a frame by frame basis. CSI configuration allows interleaving of DSI data frames from the eTPU's and eMIOS' output channels with SPI commands and data from the TX FIFO. In the CSI configuration, transmission of SPI data has higher priority than DSI data. The data returned from the bus slave is either used to drive the eTPUs or eMIOS input channels, or the data is stored in the RX FIFO. The DSPI only supports CSI configuration in Master Mode. *Figure 580* shows an example of how a DSPI can be used with a deserializing peripheral that supports SPI control for control and diagnostic frames.

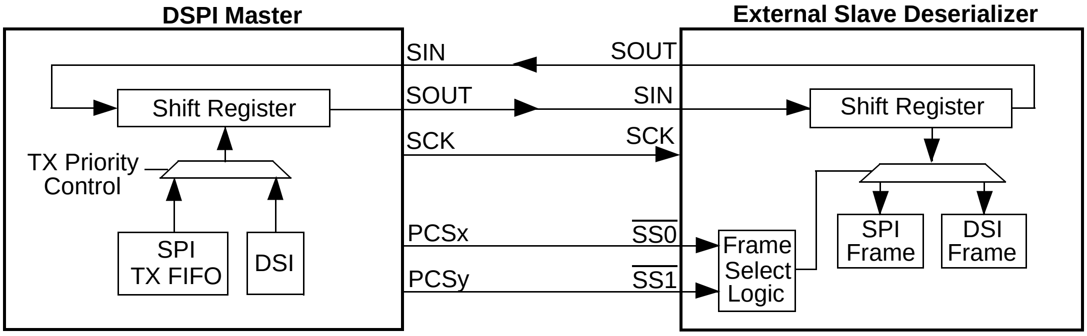


Figure 580. DSPI Connections for CSI Transfer

27.5 DSPI frequency support

The DSPI supports frequencies up to 40 MHz when used with LVDS outputs (DSPI_B and DSPI_C only), and frequencies up to 20 MHz in non-LVDS mode. *Table 593* shows possible divider settings to achieve maximum frequency for different system clock frequencies.

Table 593. DSPI channel frequency support

System clock (MHz)	DSPI use mode	Max. usable frequency (MHz)	Notes
150	LVDS	37.5	Use sysclock /4 divide ratio
	Non-LVDS	18.75	Use sysclock /8 divide ratio
120	LVDS	40	Use sysclock /3 divide ratio. Gives 33/66 duty cycle. Use DSPI configuration DBR = 0b1 (double baud rate), BR = 0b0000 (scaler value 2) and PBR = 0b01 (prescaler value 3).
	Non-LVDS	20	Use sysclock /6 divide ratio
80	LVDS	40	Use sysclock /2 divide ratio
	Non-LVDS	20	Use sysclock /4 divide ratio

27.6 Modes of operation

The DSPI has four modes of operation that can be divided into two categories: module-specific modes and an MCU-specific mode. Master Mode, Slave Mode and Module Disable Mode are the module-specific modes, and Debug Mode is the MCU-specific mode.

The module-specific modes are entered by host software writing to a register bit. The MCU-specific mode is selected by a signal external to the DSPI. The MCU-specific mode is a mode that SPC564A70 may enter in parallel to the DSPI being in one of its module-specific modes.

27.6.1 Master mode

Master Mode allows the DSPI to initiate and control serial communication. In this mode the SCK, DSPI_x_PCS and SOUT signals are controlled by the DSPI and configured as outputs.

27.6.2 Slave mode

Slave Mode allows the DSPI to communicate with SPI/DSI bus masters. In this mode the DSPI responds to externally controlled serial transfers. The DSPI cannot initiate serial transfers in Slave Mode.

27.6.3 Module Disable mode

The Module Disable mode is used for MCU power management. The clock to the non-memory mapped logic in the DSPI is stopped while in Module Disable Mode. The DSPI enters the Module Disable Mode when bit DSPI_MCR[MDIS] is set.

27.6.4 Debug mode

Debug Mode is used for system development and debugging. If the SPC564A70 MCU enters Debug Mode while bit DSPI_MCR[FRZ] is set, the DSPI halts operation on the next frame boundary. If the SPC564A70 enters Debug Mode while the FRZ bit is negated, the DSPI behavior is unaffected and remains dictated by the module-specific mode and configuration of the DSPI.

27.7 External signal description

27.7.1 Overview

Table 594 lists the signals that may connect off-chip depending on the device implementation.

Table 594. Signal properties

Name	I/O type	Function	
		Master mode	**Slave mode**
DSPI_x_PCS[0]/$\overline{SS}$	Output / Input	Peripheral Chip Select 0	Slave Select
DSPI_x_PCS[1] – PCS[3]	Output	Peripheral Chip Select 1 – 3	Unused

Table 594. Signal properties (continued)

Name	I/O type	Function	
		Master mode	Slave mode
DSPI_x_PCS[4]/$\overline{\text{MTRIG}}$	Output	Peripheral Chip Select 4	Master Trigger
DSPI_x_PCS[5]/$\overline{\text{PCSS}}$	Output	Peripheral Chip Select 5 / Peripheral Chip Select Strobe	Unused
DSPI_x_SIN	Input	Serial Data In	Serial Data In
DSPI_x_SOUT	Output	Serial Data Out	Serial Data Out
DSPI_x_SCK	Output / Input	Serial Clock (output)	Serial Clock (input)
HT	Input	Hardware Trigger	Hardware Trigger

27.7.2 Detailed signal description

DSPI_x_PCS[0]/$\overline{\text{SS}}$ — Peripheral Chip Select/Slave Select

In master mode, the DSPI_x_PCS[0] signal is a Peripheral Chip Select output that selects which slave device the current transmission is intended for.

In slave mode, the active low $\overline{\text{SS}}$ signal is a Slave Select input signal that allows a SPI master to select the DSPI as the target for transmission.

DSPI_x_PCS[1] – PCS[3] — Peripheral Chip Selects 1 – 3

DSPI_x_PCS[1] – PCS[3] are Peripheral Chip Select output signals in master mode. In slave mode these signals are unused.

DSPI_x_PCS[4]/$\overline{\text{MTRIG}}$ — Peripheral Chip Select 4/Master Trigger

In master mode, DSPI_x_PCS[4] is a Peripheral Chip Select output signal.

In slave mode, the active low $\overline{\text{MTRIG}}$ is an output trigger signal that indicates that a change in data to be serialized has occurred. The MTRIG provides a pulse in DSI configuration when a change in data to be serialized occurs. The MTRIG pulse is four system clock cycles in duration. If the DSPI is in slave mode and the MTO is disabled, the DSPI_x_PCS[4]/MTRIG signal is unused.

DSPI_x_PCS[5]/$\overline{\text{PCSS}}$ — Peripheral Chip Select 5/Peripheral Chip Select Strobe

DSPI_x_PCS[5] is a Peripheral Chip Select output signal. When the DSPI is in master mode and the DSPI_MCR[PCSSE] bit is cleared, this signal selects which slave device the current transfer is intended for.

When the DSPI is in master mode and the DSPI_MCR[PCSSE] bit is set, the $\overline{\text{PCSS}}$ signal acts as a strobe to external peripheral chip select demultiplexer, which decodes the DSPI_x_PCS[0] – PCS[4] signals, preventing glitches on the demultiplexer outputs.

This signal is not used in slave mode.

DSPI_x_SIN — Serial input

DSPI_x_SIN is a serial data input signal.

DSPI_x_SOUT — Serial output

DSPI_x_SOUT is a serial data output signal.

DSPI_x_SCK — Serial clock

DSPI_x_SCK is a serial communication clock signal. In master mode, the DSPI generates the SCK. In slave mode, SCK is an input from an external bus master.

HT — Hardware trigger

HT is a trigger input signal that is used with Multiple Transfer Operations in DSI configuration.

In master mode while in DSI or CSI configurations, the HT signal initiates a data transfer when the TRRE bit in the DSPI_DSICR is set and a rising or falling edge is detected on HT. Which edge to trigger on is determined by the TPOL bit in the DSPI_DSICR.

In slave mode, the DSPI generates a trigger pulse on the $\overline{\mathrm{MTRIG}}$ pin, when a rising or falling edge is detected on HT. Which edge that generates an output pulse is selected by the TPOL bit in the DSPI_DSICR.

27.8 Memory map and register definition

27.8.1 Memory map

Register accesses to memory addresses that are reserved or undefined result in a transfer error. Write access to the DSPI_POPR also result in a transfer error.

Table 595 shows the DSPI memory map.

Table 595. Memory map

Address	Register name	Location
DSPI_BASE	DSPI Module Configuration Register (DSPI_MCR)	*on page 27-1099*
DSPI_BASE+0x4	DSPI Hardware Configuration Register (DSPI_HCR)	*on page 27-1101*
DSPI_BASE+0x8	DSPI Transfer Count Register (DSPI_TCR)	*on page 27-1102*
DSPI_BASE+0xC – DSPI_BASE+0x28	DSPI Clock and Transfer Attributes Register 0 (DSPI_CTAR0) – DSPI Clock and Transfer Attributes Register 7 (DSPI_CTAR7)	*on page 27-1103*
DSPI_BASE+0x2C	DSPI Status Register (DSPI_SR)	*on page 27-1109*
DSPI_BASE+0x30	DSPI DMA/Interrupt Request Select and Enable Register (DSPI_RSER)	*on page 27-1111*
FIFO Registers		
DSPI_BASE+0x34	DSPI Push TX FIFO Register (DSPI_PUSHR)	*on page 27-1113*
DSPI_BASE+0x38	DSPI Pop RX FIFO Register (DSPI_POPR)	*on page 27-1115*

Table 595. Memory map (continued)

Address	Register name	Location
DSPI_BASE+0x3C – DSPI_BASE+0x48	DSPI Transmit FIFO Register 0 (DSPI_TXFR0) – DSPI Transmit FIFO Register 3 (DSPI_TXFR3)	*on page 27-1116*
DSPI_BASE+0x4C – DSPI_BASE+0x78	Reserved	
DSPI_BASE+0x7C – DSPI_BASE+0x88	DSPI Receive FIFO Register 0 (DSPI_RXFR0) – DSPI Receive FIFO Register 3 (DSPI_RXFR3)	*on page 27-1117*
DSPI_BASE+0x8C – DSPI_BASE+0xB8	Reserved	
DSI Registers		
DSPI_BASE+0xBC	DSPI DSI Configuration Register (DSPI_DSICR)	*on page 27-1117*
DSPI_BASE+0xC0	DSPI DSI Serialization Data Register (DSPI_SDR)	*on page 27-1120*
DSPI_BASE+0xC4	DSPI DSI Alternate Serialization Data Register (DSPI_ASDR)	*on page 27-1120*
DSPI_BASE+0xC8	DSPI DSI Transmit Comparison Register (DSPI_COMPR)	*on page 27-1121*
DSPI_BASE+0xCC	DSPI DSI Deserialization Data Register (DSPI_DDR)	*on page 27-1122*
DSPI_BASE+0xD0	DSPI DSI Configuration Register 1 (DSPI_DSICR1)	*on page 27-1122*
DSPI_BASE+0xD4	DSPI DSI Serialization Source Select Register (DSPI_SSR)	*on page 27-1123*
DSPI_BASE+0xD8	DSPI DSI Parallel Input Select Register 0 (DPSI_PISR0)(1)	*on page 27-1124*
DSPI_BASE+0xDC	DSPI DSI Parallel Input Select Register 1 (DPSI_PISR1)(1)	*on page 27-1124*
DSPI_BASE+0xE0	DSPI DSI Parallel Input Select Register 2 (DPSI_PISR2)(1)	*on page 27-1124*
DSPI_BASE+0xE4	DSPI DSI Parallel Input Select Register 3 (DPSI_PISR3)(1)	*on page 27-1124*
DSPI_BASE+0xE8	DSPI DSI Deserialized Data Interrupt Mask Register (DSPI_DIMR)	*on page 27-1128*
DSPI_BASE+0xEC	DSPI DSI Deserialized Data Polarity Interrupt Register (DSPI_DPIR)	*on page 27-1129*

1. DSPI_PISR0-3 registers and assosiated with them functionality may be not implemented in particular DSPI instances of the SOC.

27.8.2 Register descriptions

DSPI Module Configuration Register (DSPI_MCR)

The DSPI_MCR contains bits which configure various attributes associated with DSPI operation. The HALT and MDIS bits can be changed at any time, but only take effect on the next frame boundary. Only the HALT and MDIS bits in the DSPI_MCR are allowed to be changed, while the DSPI is in the Running state.

Figure 581. DSPI Module Configuration Register (DSPI_MCR)

Address: DSPI_BASE

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MSTR	CONT_SCKE	DCONF		FRZ	MTFE	PCSSE	ROOE	PCSIS7	PCSIS6	PCSIS5	PCSIS4	PCSIS3	PCSIS2	PCSIS1	PCSIS0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	MDIS	DIS_TXF	DIS_RXF	0	0	SMPL_PT		0	0	0	0	0	0	PES	HALT
W					CLR_TXF	CLR_RXF										
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Table 596. DSPI_MCR field description

Field	Description
0 MSTR	Master/Slave Mode Select The MSTR bit configures the DSPI for either master mode or slave mode. 0 DSPI is in slave mode 1 DSPI is in master mode
1 CONT_SCKE	Continuous SCK Enable The CONT_SCKE bit enables the Serial Communication Clock (SCK) to run continuously. See *Section 27.9.7, Continuous serial communications clock* for details. 0 Continuous SCK disabled 1 Continuous SCK enabled
2–3 DCONF[0:1]	DSPI Configuration The DCONF field selects between the three different configurations of the DSPI: 00 SPI 01 DSI 10 CSI 11 Reserved

Table 596. DSPI_MCR field description (continued)

Field	Description
4 FRZ	Freeze The FRZ bit enables the DSPI transfers to be stopped on the next frame boundary when the device enters Debug mode. 0 Do not stop serial transfers 1 Stop serial transfers
5 MTFE	Modified Timing Format Enable The MTFE bit enables a modified transfer format to be used. See *Section , Modified SPI/DSI transfer format (MTFE = 1, CPHA = 1)* for more information. 0 Modified SPI transfer format disabled 1 Modified SPI transfer format enabled
6 PCSSE	Peripheral Chip Select Strobe Enable The PCSSE bit enables the DSPI_x_PCS[5]/$\overline{\text{PCSS}}$ to operate as a PCS Strobe output signal. See *Section , Peripheral chip select strobe enable (PCSS)* for more information. 0 DSPI_x_PCS[5]/$\overline{\text{PCSS}}$ is used as the Peripheral Chip Select[5] signal 1 DSPI_x_PCS[5]/$\overline{\text{PCSS}}$ is used as an active-low PCS Strobe signal
7 ROOE	Receive FIFO Overflow Overwrite Enable The ROOE bit enables in RX FIFO overflow condition to ignore the incoming serial data or to overwrite existing data. If the RX FIFO is full and new data is received, the data from the transfer, generated the overflow, is ignored or shifted in to the shift register. See *Section , Receive FIFO overflow interrupt request* for more information. 0 Incoming data is ignored 1 Incoming data is shifted in to the shift register
8–15 PCSIS*x*	Peripheral Chip Select Inactive State The PCSIS bit determines the inactive state of the PCS*x* signal. 0 The inactive state of PCS*x* is low 1 The inactive state of PCS*x* is high
17 MDIS	Module Disable The MDIS bit allows the clock to be stopped to the non-memory mapped logic in the DSPI effectively putting the DSPI in a software controlled power-saving state. See *Section 27.9.18, Power saving features* for more information. The reset value of the MDIS bit is parameterized, with a default reset value of ‘0’. 0 Enable DSPI clocks. 1 Allow external logic to disable DSPI clocks.
18 DIS_TXF	Disable Transmit FIFO When the TX FIFO is disabled, the transmit part of the DSPI operates as a simplified double-buffered SPI. See *Section , FIFO disable operation* for details. 0 TX FIFO is enabled 1 TX FIFO is disabled
19 DIS_RXF	Disable Receive FIFO When the RX FIFO is disabled, the receive part of the DSPI operates as a simplified double-buffered SPI. See *Section , FIFO disable operation* for details. 0 RX FIFO is enabled 1 RX FIFO is disabled

Table 596. DSPI_MCR field description (continued)

Field	Description
20 CLR_TXF	Clear TX FIFO CLR_TXF is used to flush the TX FIFO. Writing a ‘1’ to CLR_TXF clears the TX FIFO Counter. The CLR_TXF bit is always read as zero. 0 Do not clear the TX FIFO Counter 1 Clear the TX FIFO Counter
21 CLR_RXF	Clear RX FIFO CLR_RXF is used to flush the RX FIFO. Writing a ‘1’ to CLR_RXF clears the RX Counter. The CLR_RXF bit is always read as zero. 0 Do not clear the RX FIFO Counter 1 Clear the RX FIFO Counter
22–23 SMPL_PT	Sample Point SMPL_PT field controls when the DSPI master samples SIN in Modified Transfer Format. *Figure 616* shows where the master can sample the SIN pin. 00 DSPI samples SIN at driving SCK edge. 01 DSPI samples SIN one system clock after driving SCK edge 10 DSPI samples SIN two system clocks after driving SCK edge 11 Reserved
24–29	Reserved, should be cleared.
30 PES	Parity Error Stop PES bit controls SPI operation when a parity error detected in received SPI frame. 0 SPI frames transmission continue. 1 SPI frames transmission stop.
31 HALT	Halt The HALT bit starts and stops DSPI transfers. See *Section 27.9.1, Start and stop of DSPI transfers* for details on the operation of this bit. 0 Start transfers 1 Stop transfers

DSPI Hardware Configuration Register (DSPI_HCR)

DSPI Hardware Configuration Register provides particular implementation details about the DSPImodule, i.e. number of Receive and Transmit FIFO entries, number of CTAR registers and if DSI featuresare implemented in the module or not. It is read only register.

Figure 582. DSPI Hardware Configuration Register (DSPI_HCR)

Address: DSPI_BASE + 0x4 Access:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	DSI	PISR	0	0	0	CTAR			TXFR				RXFR			
W																
Reset	-[1]	-	0	0	0	-	-	-	-	-	-	-	-	-	-	-

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1. The reset bits in the DSPI_HCR are set by configuration parameters in the SOC.

Table 597. DSPI_HCR field description

Field	Description
0 DSI	DSI features are implemented for the module. 0 - DSI features are not implemented, DSI registers don't exist. 1 - DSI features are implemented
1 PISR	PISR, PISR0-3 and parallel inputs frame positions selection logic are implemented for the module. 0 - DSPI_PISR0-3 registers are not implemented. 1 - DSPI_PISR0-3 registers are implemented
2–4	Reserved, should be cleared.
5–7 CTAR[0:2]	CTAR, Maximum implemented DSPI_CTAR register number.
8–11 TXFR0:3]	TXFR, Maximum implemented DSPI_TXFR register number.
12–15 RXFR0:3]	RXFR, Maximum implemented DSPI_RXFR register number.
16–31	Reserved, should be cleared.

DSPI Transfer Count Register (DSPI_TCR)

The DSPI_TCR contains a counter that indicates the number of SPI transfers made. The transfer counter is intended to assist in queue management. Do not write the DSPI_TCR, when the DSPI is in the Running state.

Figure 583. DSPI Transfer Count Register (DSPI_TCR)

Address: DSPI_BASE + 0x8

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TCNT															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 598. DSPI_TCR field description

Field	Description
0–15 TCNT[0:15]	SPI Transfer Counter The SPI_TCNT field counts the number of SPI transfers the DSPI makes. The SPI_TCNT field increments every time the last bit of a SPI frame is transmitted. A value written to SPI_TCNT presets the counter to that value. SPI_TCNT is reset to zero at the beginning of the frame when the CTCNT field is set in the executing SPI command. The Transfer Counter ‘wraps around’ i.e. incrementing the counter past 65535 resets the counter to zero.
16–31	Reserved, should be cleared.

DSPI Clock and Transfer Attributes Registers 0–7 (DSPI_CTAR0–DSPI_CTAR7)

The DSPI_CTAR registers are used to define different transfer attributes. The number of CTAR registersis parameterized in the RTL and can be from two to eight registers. Do not write to the DSPI_CTARregisters, while the DSPI is in the Running state.

In master mode, the DSPI_CTAR0 - DSPI_CTAR7 registers define combinations of transfer attributessuch as frame size, clock phase and polarity, data bit ordering, baud rate, and various delays. In slave mode,a subset of the bitfields in the DSPI_CTAR0 and DSPI_CTAR1 registers are used to set the slave transfer attributes.

When the DSPI is configured as a SPI master, the CTAS field in the command portion of the TX FIFOentry selects which of the DSPI_CTAR register is used. When the DSPI is configured as a SPI bus slave,the DSPI_CTAR0 register is used.

When the DSPI is configured as a DSI master, the DSICTAS field in the DSPI DSI Configuration Register (DSPI_DSICR), selects which of the DSPI_CTAR register is used. When the DSPI is configured as a DSI bus slave, the DSPI_CTAR1 register is used.

In CSI Configuration, the transfer attributes are selected based on whether the current frame is SPI data or DSI data. SPI transfers in CSI Configuration follow the protocol described for SPI Configuration, and DSI transfers in CSI Configuration follow the protocol described for DSI Configuration. CSI Configuration is only valid in conjunction with master mode. See Section 1.5.4, “Combined Serial Interface (CSI) Configuration,” for more details..

TSB mode sets some limitations on transfer attributes:

- Clock phase is forced to be CPHA = 1 and the CPHA bit setting has no effect.
- PCS lines are driven at the driving edge of the SCK clock together with SOUT, so PCS assertion and negation delays control is unavailable and PCSSCK, PASC, CSSCK and ASC fields have no effect.
- Delay after transfer can be set from 1 to 64 serial clocks with help of PDT and DT fields.

Figure 584. DSPI Clock and Transfer Attributes Register 0–7 (DSPI_CTAR0–DSPI_CTAR7) in the master mode

Address: DSPI_BASE + 0xC–DSPI_BASE + 0x28

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	DBR	FMSZ				CPOL	CPHA	LSBFE	PCSSCK		PASC		PDT		PBR	
Reset	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CSSCK				ASC				DT				BR			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 585. DSPI Clock and Transfer Attributes Register 0 (DSPI_CTAR0) in the slave mode

Address: DSPI_BASE + 0xC

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	FMSZ					CPOL	CPHA	PE	PP	Not used						
Reset	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	Not used															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 599. DSPI_CTAR*n* field description in master mode

Field	Descriptions
0 DBR	Double Baud Rate The DBR bit doubles the effective baud rate of the Serial Communications Clock (SCK). This field is only used in master mode. It effectively halves the Baud Rate division ratio supporting faster frequencies and odd division ratios for the Serial Communications Clock (SCK). When the DBR bit is set, the duty cycle of the Serial Communications Clock (SCK) depends on the value in the Baud Rate Prescaler and the Clock Phase bit as listed in *Table 600*. See the BR field description for details on how to compute the baud rate. 0 The baud rate is computed normally with a 50/50 duty cycle 1 The baud rate is doubled with the duty cycle depending on the Baud Rate Prescaler
1–4 FMSZ[0:3]	Frame Size The number of bits transferred per frame is equal to FMSZ field value plus 1. Minimum valid FMSZ field value is 3. When operating in TSB mode, detailed in *Section 27.9.8, Timed serial bus (TSB)* the FMSZ field value plus 1 is equal the data frame bit number, where control of the PCS assertion switches from the DSPI_DSICR to the DSPI_DSICR1 register.
5 CPOL	Clock Polarity The CPOL bit selects the inactive state of the Serial Communications Clock (SCK). This bit is used in both master and slave mode. For successful communication between serial devices, the devices must have identical clock polarities. When the Continuous selection format is selected, switching between clock polarities without stopping the DSPI can cause errors in the transfer due to the peripheral device interpreting the switch of clock polarity as a valid clock edge. 0 The inactive state value of SCK is low 1 The inactive state value of SCK is high
6 CPHA	Clock Phase The CPHA bit selects which edge of SCK causes data to change and which edge causes data to be captured. This bit is used in both master and slave mode. For successful communication between serial devices, the devices must have identical clock phase settings. In Continuous SCK mode or TSB mode the bit value is ignored and the transfers are done as CPHA bit is set to 1. 0 Data is captured on the leading edge of SCK and changed on the following edge 1 Data is changed on the leading edge of SCK and captured on the following edge
7 LSBFE	LSB First The LSBFE bit selects if the LSB or MSB of the frame is transferred first. When operating in TSB configuration, this bit should be set to be compliant to MSC specification. 0 Data is transferred MSB first 1 Data is transferred LSB first
8–9 PCSSCK[0:1]	PCS to SCK Delay Prescaler The PCSSCK field selects the prescaler value for the delay between assertion of PCS and the first edge of the SCK. See the CSSCK field description how to compute the PCS to SCK delay. In the TSB mode the PCSSCK field has no effect. 00 PCS to SCK prescaler value is 1 01 PCS to SCK prescaler value is 3 10 PCS to SCK prescaler value is 5 11 PCS to SCK prescaler value is 7

Table 599. DSPI_CTAR*n* field description in master mode (continued)

Field	Descriptions
10–11 PASC[0:1]	After SCK Delay Prescaler The PASC field selects the prescaler value for the delay between the last edge of SCK and the negation of PCS. See the ASC field description how to compute the After SCK delay. In the TSB mode the PASC field has no effect. 00 After SCK delay prescaler value is 1 01 After SCK delay prescaler value is 3 10 After SCK delay prescaler value is 5 11 After SCK delay prescaler value is 7
12–13 PDT[0:1]	Delay after Transfer Prescaler The PDT field selects the prescaler value for the delay between the negation of the PCS signal at the end of a frame and the assertion of PCS at the beginning of the next frame. The PDT field is only used in master mode. In the TSB mode the PDT field defines two MSB bits of the Delay after Transfer. See the DT field description for details on how to compute the Delay after Transfer. 00 Delay after Transfer prescaler value is 1 01 Delay after Transfer prescaler value is 3 10 Delay after Transfer prescaler value is 5 11 Delay after Transfer prescaler value is 7
14–15 PBR[0:1]	Baud Rate Prescaler The PBR field selects the prescaler value for the baud rate. This field is only used in master mode. The Baud Rate is the frequency of the Serial Communications Clock (SCK). The system clock is divided by the prescaler value before the baud rate selection takes place. See the BR field description for details on how to compute the baud rate. 00 Baud Rate prescaler value is 2 01 Baud Rate prescaler value is 3 10 Baud Rate prescaler value is 5 11 Baud Rate prescaler value is 7
16–19 CSSCK[0:3]	PCS to SCK Delay Scaler The CSSCK field selects the scaler value for the PCS to SCK delay. This field is only used in master mode. The PCS to SCK delay is the delay between the assertion of PCS and the first edge of the SCK. *Table 601* list the scaler values.The PCS to SCK delay is a multiple of the system clock period and it is computed according to the following equation: **Equation 23** $t_{CSC} = \frac{1}{f_{SYS}} \times PCSSCK \times CSSCK$ See *Section , PCS to SCK delay (t_{CSC})* for more details.In the TSB mode the field has no effect.
20–23 ASC[0:3]	After SCK Delay Scaler The ASC field selects the scaler value for the After SCK Delay. This field is only used in master mode. The After SCK Delay is the delay between the last edge of SCK and the negation of PCS. *Table 601* list the scaler values.The After SCK Delay is a multiple of the system clock period, and it is computed according to the following equation: **Equation 24** $t_{ASC} = \frac{1}{f_{SYS}} \times PASC \times ASC$ See *Section , After SCK delay (t_{ASC})* for more details. In the TSB mode the field has no effect.

Table 599. DSPI_CTAR*n* field description in master mode (continued)

Field	Descriptions
24–27 DT[0:3]	Delay after Transfer Scaler The DT field selects the Delay after Transfer Scaler. This field is only used in master mode. The Delay after Transfer is the time between the negation of the PCS signal at the end of a frame and the assertion of PCS at the beginning of the next frame. *Table 601* lists the scaler values. In the Continuous Serial Communications Clock operation the DT value is fixed to one SCK clock period, The Delay after Transfer is a multiple of the system clock period and it is computed according to the following equation: **Equation 25** $t_{DT} = \frac{1}{f_{SYS}} \times PDT \times DT$ In the TSB mode the Delay after Transfer is equal to a number formed by concatenation of PDT and DT fields plus 1 of the SCK clock periods. See *Section , Delay after transfer (tDT)* for more details.
28–31 BR[0:3]	Baud Rate Scaler The BR field selects the scaler value for the baud rate. This field is only used in master mode. The prescaled system clock is divided by the Baud Rate Scaler to generate the frequency of the SCK. *Table 602* lists the Baud Rate Scaler values.The baud rate is computed according to the following equation: **Equation 26** $\text{SCK baud rate} = \frac{f_{SYS}}{PBR} \times \frac{1 + DBR}{BR}$ See *Section , Baud rate generator* for more details.

Table 600. DSPI SCK duty cycle

DBR	CPHA	PBR	SCK duty cycle
0	any	any	50/50
1	0	00	50/50
1	0	01	33/66
1	0	10	40/60
1	0	11	43/57
1	1	00	50/50
1	1	01	66/33
1	1	10	60/40
1	1	11	57/43

Table 601. Delay scaler encoding

Field value	Scaler value	Field value	Scaler value
0000	2	1000	512
0001	4	1001	1024
0010	8	1010	2048

Table 601. Delay scaler encoding (continued)

Field value	Scaler value	Field value	Scaler value
0011	16	1011	4096
0100	32	1100	8192
0101	64	1101	16384
0110	128	1110	32768
0111	256	1111	65536

Table 602. DSPI baud rate scaler

BR	Baud rate scaler value	BR	Baud rate scaler value
0000	2	1000	256
0001	4	1001	512
0010	6	1010	1024
0011	8	1011	2048
0100	16	1100	4096
0101	32	1101	8192
0110	64	1110	16384
0111	128	1111	32768

Table 603. DSPI_CTAR0 field description in slave mode

Field	Descriptions
0–4 FMSZ[0:4]	Frame Size The number of bits transferred per frame is equal FMSZ field value plus 1. Minimum valid FMSZ field value is 3.
5 CPOL	Clock Polarity The CPOL bit selects the inactive state of the Serial Communications Clock (SCK). 0 The inactive state value of SCK is low 1 The inactive state value of SCK is high
6 CPHA	Clock Phase The CPHA bit selects which edge of SCK causes data to change and which edge causes data to be captured. 0 Data is captured on the leading edge of SCK and changed on the following edge 1 Data is changed on the leading edge of SCK and captured on the following edge
7 PE	Parity Enable PE bit enables parity bit transmission and reception for the frame 0 No parity bit included/checked. 1 Parity bit is transmitted instead of last data bit in frame, parity checked for received frame.

Table 603. DSPI_CTAR0 field description in slave mode (continued)

Field	Descriptions
8 PP	Parity Polarity PP bit controls polarity of the parity bit transmitted and checked 0 Even Parity: number of “1” bits in the transmitted frame is even. The DSPI_SR[SPEF] bit is set if in the received frame number of “1” bits is odd. 1 Odd Parity: number of “1” bits in the transmitted frame is odd. The DSPI_SR[SPEF] bit is set if in the received frame number of “1” bits is even.
29–31 —	Not used, write always zero to keep software compatible with future updates.

DSPI Status Register (DSPI_SR)

The DSPI_SR contains status and flag bits. The bits reflect the status of the DSPI and indicate the occurrence of events that can generate interrupt or DMA requests. Software can clear flag bits in the DSPI_SR by writing a ‘1’ to it. Writing a ‘0’ to a flag bit has no effect. This register may not be writable in module disable mode due to the use of power saving mechanisms.

Figure 586. DSPI Status Register (DSPI_SR)

Address: DSPI_BASE + 0x2C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TCF	TXRXS	0	EOQF	TFUF	0	TFFF	0	0	DPEF	SPEF	DDIF	RFOF	0	RFDF	0
W	w1c	w1c		w1c	w1c		w1c			w1c	w1c	w1c	w1c		w1c	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	TXCTR				TXNXTPTR				RXCTR				POPNXTPTR			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 604. DSPI_SR field description

Field	Description
0 TCF	Transfer Complete Flag The TCF bit indicates that all bits in a frame have been shifted out. The TCF bit remains set until cleared by writing 1 to it. 0 Transfer not complete 1 Transfer complete
1 TXRXS	TX & RX Status The TXRXS bit reflects the run status of the DSPI. *Section 27.9.1, Start and stop of DSPI transfers* explains what causes this bit to be set or cleared. 0 TX and RX operations are disabled (DSPI is in STOPPED state) 1 TX and RX operations are enabled (DSPI is in RUNNING state)

Table 604. DSPI_SR field description (continued)

Field	Description
2	Reserved, should be cleared.
3 EOQF	End of Queue Flag The EOQF bit indicates that the last entry in a queue has been transmitted when the DSPI in the master mode. The EOQF bit is set when TX FIFO entry has the EOQ bit set in the command halfword and the end of the transfer is reached. The EOQF bit remains set until cleared by writing 1 to it. When the EOQF bit is set, the TXRXS bit is automatically cleared. 0 EOQ is not set in the executed command 1 EOQ bit is set in the executed SPI command
4 TFUF	Transmit FIFO Underflow Flag The TFUF bit indicates that an underflow condition in the TX FIFO has occurred. The transmit underflow condition is detected only for DSPI modules operating in slave mode and SPI configuration. The TFUF bit is set when the TX FIFO of a DSPI operating in SPI slave mode is empty, and a transfer is initiated by an external SPI master. The TFUF bit remains set until cleared by writing 1 to it. 0 TX FIFO underflow has not occurred 1 TX FIFO underflow has occurred
5	Reserved, should be cleared.
6 TFFF	Transmit FIFO Fill Flag The TFFF bit provides a method for the DSPI to request more entries to be added to the TX FIFO. The TFFF bit is set while the TX FIFO is not full. The TFFF bit can be cleared by writing 1 to it or by acknowledgement from the DMA controller to the TX FIFO full request. 0 TX FIFO is full 1 TX FIFO is not full
7–8	Reserved, should be cleared.
9 DPEF	DSI Parity Error Flag The DPEF flag indicates that a DSI frame with parity error had been received. The bit remains set until cleared by writing 1 to it. 0 Parity Error has not occurred 1 Parity Error has occurred
10 SPEF	SPI Parity Error Flag The SPEF flag indicates that a SPI frame with parity error had been received. The bit remains set until cleared by writing 1 to it. 0 Parity Error has not occurred 1 Parity Error has occurred
11 DDIF	DSI data received with active bits The DDIF flag indicates that DSI frame had been received with bits, selected by DSPI_DIMR with active polarity, defined by DSPI_DPIR. The bit remains set until cleared by writing 1 to it. 0 No DSI data with active bits was received 1 DSI data with active bits was received

Table 604. DSPI_SR field description (continued)

Field	Description
12 RFOF	Receive FIFO Overflow Flag The RFOF bit indicates that an overflow condition in the RX FIFO has occurred. The bit is set when the RX FIFO and shift register are full and a transfer is initiated. The bit remains set until cleared by writing 1 to it. 0 RX FIFO overflow has not occurred 1 RX FIFO overflow has occurred
13	Reserved, should be cleared.
14 RFDF	Receive FIFO Drain Flag The RFDF bit provides a method for the DSPI to request that entries be removed from the RX FIFO. The bit is set while the RX FIFO is not empty. The RFDF bit can be cleared by writing 1 to it or by acknowledgement from the DMA controller when the RX FIFO is empty. 0 RX FIFO is empty 1 RX FIFO is not empty
15	Reserved.
16–20 TXCTR	TX FIFO Counter The TXCTR field indicates the number of valid entries in the TX FIFO. The TXCTR is incremented every time the DSPI _PUSHR is written. The TXCTR is decremented every time a SPI command is executed and the SPI data is transferred to the shift register.
20–23 TXNXTPTR	Transmit Next Pointer The TXNXTPTR field indicates which TX FIFO Entry is transmitted during the next transfer. The TXNXTPTR field is updated every time SPI data is transferred from the TX FIFO to the shift register. See *Section , Transmit FIFO underflow interrupt request* for more details.
24–27 RXCTR	RX FIFO Counter The RXCTR field indicates the number of entries in the RX FIFO. The RXCTR is decremented every time the DSPI _POPR is read. The RXCTR is incremented every time data is transferred from the shift register to the RX FIFO.
28–31 POPNXTPTR	Pop Next Pointer The POPNXTPTR field contains a pointer to the RX FIFO entry that will be returned when the DSPI_POPR is read. The POPNXTPTR is updated when the DSPI_POPR is read. See *Section , Receive first-in first-out (RX FIFO) buffering mechanism* for more details.

DSPI DMA/Interrupt Request Select and Enable Register (DSPI_RSER)

The DSPI_RSER controls DMA and interrupt requests. Do not write to the DSPI_RSER while the DSPI is in the Running state.

Figure 587. DSPI DMA/Interrupt Request Select and Enable Register (DSPI_RSER)

Address: DSPI_BASE + 0x30

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TCF_RE	0	0	EOQFRE	TFUFRE	0	TFFFRE	TFFFDIRS	0	DPEFRE	SPEFRE	DDIFRE	RFOFRE	0	RFDFRE	RFDFDIRS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 605. DSPI_RSER field description

Field	Description
0 TCF_RE	Transmission Complete Request Enable The TCF_RE bit enables TCF flag in the DSPI_SR to generate an interrupt request. 0 TCF interrupt requests are disabled 1 TCF interrupt requests are enabled
1–2	Reserved, should be cleared.
3 EOQFRE	DSPI Finished Request Enable The EOQFRE bit enables the EOQF flag in the DSPI_SR to generate an interrupt request. 0 EOQF interrupt requests are disabled 1 EOQF interrupt requests are enabled
4 TFUFRE	Transmit FIFO Underflow Request Enable The TFUFRE bit enables the TFUF flag in the DSPI_SR to generate an interrupt request. 0 TFUF interrupt requests are disabled 1 TFUF interrupt requests are enabled
5	Reserved, should be cleared.
6 TFFFRE	Transmit FIFO Fill Request Enable The TFFFRE bit enables the TFFF flag in the DSPI_SR to generate a request. The TFFFDIRS bit selects between generating an interrupt request or a DMA requests. 0 TFFF interrupt requests or DMA requests are disabled 1 TFFF interrupt requests or DMA requests are enabled
7 TFFFDIRS	Transmit FIFO Fill DMA or Interrupt Request Select The TFFFDIRS bit selects between generating a DMA request or an interrupt request. When the TFFF flag bit in the DSPI_SR is set, and the TFFFRE bit in the DSPI_RSER is set, this bit selects between generating an interrupt request or a DMA request. 0 Interrupt request will be generated 1 DMA request will be generated
8	Reserved, should be cleared.

Table 605. DSPI_RSER field description (continued)

Field	Description
9 DPEFRE	DSI Parity Error Request Enable The DPEFRE bits enables DPEF flag in the DSPI_SR to generate an interrupt requests. 0 PEF interrupt requests are disabled 1 PEF interrupt requests are enabled
10 SPEFRE	SPI Parity Error Request Enable The SPEFRE bits enables SPEF flag in the DSPI_SR to generate an interrupt requests. 0 PEF interrupt requests are disabled 1 PEF interrupt requests are enabled
11 DDIFRE	DSI data received with active bits Request Enable The DDIFRE bit enables the DDIF flag in the DSPI_SR to generate an interrupt requests. 0 DDIF interrupt requests are disabled 1 DDIF interrupt requests are enabled
12 RFOFRE	Receive FIFO Overflow Request Enable The RFOFRE bit enables the RFOF flag in the DSPI_SR to generate an interrupt requests. 0 RFOF interrupt requests are disabled 1 RFOF interrupt requests are enabled
13	Reserved, should be cleared.
14 RFDFRE	Receive FIFO Drain Request Enable The RFDFRE bit enables the RFDF flag in the DSPI_SR to generate a request. The RFDFDIRS bit selects between generating an interrupt request or a DMA request. 0 RFDF interrupt requests or DMA requests are disabled 1 RFDF interrupt requests or DMA requests are enabled
15 RFDFDIRS	Receive FIFO Drain DMA or Interrupt Request Select The RFDFDIRS bit selects between generating a DMA request or an interrupt request. When the RFDF flag bit in the DSPI_SR is set, and the RFDFRE bit in the DSPI_RSER is set, the RFDFDIRS bit selects between generating an interrupt request or a DMA request. 0 Interrupt request will be generated 1 DMA request will be generated

DSPI PUSH TX FIFO Register (DSPI_PUSHR)

The DSPI_PUSHR provides means to write to the TX FIFO. Data written to this register is transferred to the TX FIFO. See *Section , Transmit first-in first-out (TX FIFO) buffering mechanism* for more information. Eight or 16-bit write accesses to the DSPI_PUSHR transfers all 32 register bits to the TX FIFO.

The register structure is different in master and slave modes. In master mode the register provides 16-bit commands and 16-bit data to the TX FIFO. In slave mode all 32 register bits can be used as data, supporting up to 32-bit SPI frame operation.

Figure 588. DSPI PUSH TX FIFO Register (DSPI_PUSHR) in master mode

Address: DSPI_BASE + 0x34

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CONT	CTAS			EOQ	CTCNT	PE	PP	PCS 7	PCS 6	PCS 5	PCS 4	PCS 3	PCS 2	PCS 1	PCS 0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	TXDATA															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 606. DSPI_PUSHR field description in master mode

Field	Descriptions
0 CONT	Continuous Peripheral Chip Select Enable The CONT bit selects a Continuous Selection Format. The bit is used in SPI master mode. The bit enables the selected PCS signals to remain asserted between transfers. See *Section , Continuous selection format* for more information. 0 Return Peripheral Chip Select signals to their inactive state between transfers 1 Keep Peripheral Chip Select signals asserted between transfers
1–3 CTAS[0:2]	Clock and Transfer Attributes Select The CTAS field selects the number of the DSPI_CTAR to be used to set the transfer attributes for the associated SPI frame. The field is only used in SPI master mode. In SPI slave mode DSPI_CTAR0 is used. The number of DSPI_CTAR registers is implementation specific and the CTAS should be set to select only implemented one.
4 EOQ	End Of Queue The EOQ bit provides a means for host software to signal to the DSPI that the current SPI transfer is the last in a queue. At the end of the transfer the EOQF bit in the DSPI_SR is set. 0 The SPI data is not the last data to transfer 1 The SPI data is the last data to transfer
5 CTCNT	Clear Transfer Counter The CTCNT bit clears field DSPI_TCR[TCNT]. The TCNT field is cleared before transmission of the current SPI frame begins. 0 Do not clear field DSPI_TCR[TCNT] 1 Clear field DSPI_TCR[TCNT]
6 PE	Parity Enable PE bit enables parity bit transmission and parity reception check for the SPI frame 0 No parity bit included/checked. 1 Parity bit is transmitted instead of last data bit in frame, parity checked for received frame.

Table 606. DSPI_PUSHR field description in master mode (continued)

Field	Descriptions
7 PP	Parity Polarity PP bit controls polarity of the parity bit transmitted and checked 0 Even Parity: number of “1” bits in the transmitted frame is even. The DSPI_SR[SPEF] bit is set if in the received frame number of “1” bits is odd. 1 Odd Parity: number of “1” bits in the transmitted frame is odd. The DSPI_SR[SPEF] bit is set if in the received frame number of “1” bits is even.
8–15 PCS*x*	Peripheral Chip Select 0–7 The PCS bits select which PCS signals will be asserted for the transfer. 0 Negate the PCS[x] signal 1 Assert the PCS[x] signal
16–31 TXDATA[0:15]	Transmit Data The TXDATA field holds SPI data to be transferred according to the associated SPI command.

Figure 589. DSPI PUSH TX FIFO Register (DSPI_PUSHR) in slave mode

Address: DSPI_BASE + 0x34

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	TXDATA															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	TXDATA															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 607. DSPI_PUSHR field description in slave mode

Field	Descriptions
0–31 TXDATA[0:31]	Transmit Data The TXDATA field holds SPI data to be transferred.

DSPI POP RX FIFO Register (DSPI_POPR)

The DSPI_POPR provides the means to read the RX FIFO. See *Section , Receive first-in first-out (RX FIFO) buffering mechanism* for a description of the RX FIFO operations. Eight or 16-bit read accesses to the DSPI_POPR have the same effect on the RX FIFO as 32-bit read access.

Figure 590. DSPI POP RX FIFO Register (DSPI_POPR)

Address: DSPI_BASE + 0x38

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	RXDATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	RXDATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 608. DSPI_POPR field description

Field	Description
0–31 RXDATA[0:31]	Received Data The RXDATA field contains the SPI data from the RX FIFO entry pointed to by the Pop Next Data Pointer.

DSPI Transmit FIFO Registers 0–15 (DSPI_TXFR0–DSPI_TXFR15)

The DSPI_TXFR0 – DSPI_TXFR15 registers provide visibility into the TX FIFO for debugging purposes. Each register is an entry in the TX FIFO. The registers are read-only and cannot be modified. Reading the DSPI_TXFRx registers does not alter the state of the TX FIFO. The number of registers used to implement the TX FIFO is device specific. If a four-entry TX FIFO is implemented, DSPI_TXFR0 – DSPI_TXFR3 are accessible.

Figure 591. DSPI Transmit FIFO Register 0–15 (DSPI_TXFR0–DSPI_TXFR15)

Address: DSPI_BASE+0x3C–DSPI_BASE+0x78

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TXCMD/TXDATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	TXDATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 609. DSPI_TXFR*n* field description

Field	Description
0–15 TXCMD[0:15]/ TXDATA[0:15]	Transmit Command or Transmit Data In master mode the TXCMD field contains the command that sets the transfer attributes for the SPI data. See *Section , DSPI PUSH TX FIFO Register (DSPI_PUSHR)* for details on the command field. In slave mode the TXDATA contains 16 MSB bits of the SPI data to be shifted out
16–31 TXDATA[16:31]	Transmit Data The TXDATA field contains the SPI data to be shifted out.

DSPI Receive FIFO Registers 0–15 (DSPI_RXFR0–DSPI_RXFR15)

The DSPI_RXFR0 – DSPI_RXFR15 registers provide visibility into the RX FIFO for debugging purposes. Each register is an entry in the RX FIFO. The DSPI_RXFR registers are read-only. Reading the DSPI_RXFRx registers does not alter the state of the RX FIFO. The number of registers used to implement the RX FIFO is device specific. If a four-entry RX FIFO is implemented, DSPI_RXFR0 – DSPI_RXFR3 exist, for example.

Figure 592. DSPI Receive FIFO Registers 0–15 (DSPI_RXFR0–DSPI_RXFR15)

Address: DSPI_BASE + 0x7C–DSPI_BASE + 0xB8

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	RXDATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	RXDATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 610. DSPI_RXFR*n* field description

Field	Description
0–31 RXDATA[0:31]	Receive Data The RXDATA field contains the received SPI data.

DSPI DSI Configuration Register (DSPI_DSICR)

The DSI Configuration Register selects various attributes associated with DSI and CSI Configurations. Do not write to the DSPI_DSICR, while the DSPI is in the Running state.

Figure 593. DSPI DSI Configuration Register (DSPI_DSICR)

Address: DSPI_BASE + 0xBC

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MTOE	FMSZ[4]	MTOCNT						0	0	0	TSBC	TXSS	TPOL	TRRE	CID
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	DCONT	DSICTAS			DMS	PES	PE	PP	DPCS7	DPCS6	DPCS5	DPCS4	DPCS3	DPCS2	DPCS1	DPCS0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 611. DSPI_DSICR field description

Field	Description
0 MTOE	Multiple Transfer Operation Enable The MTOE bit enables multiple DSPIs to be connected in a parallel or serial configuration. See *Section , Multiple transfer operation (MTO)* for more information. 0 Multiple Transfer Operation disabled 1 Multiple Transfer Operation enabled The MTOE and TSB bits should not be set simultaneously.
1	MSB of the Frame Size If the bit is set, 16 is added to the frame size, defined by field DSPI_CTARn[FMSZ]. DSPI_CTARn register is selected by field DSPI_DSICR[DSICTAS].
2–7 MTOCNT[0:5]	Multiple Transfer Operation Count The MTOCNT field selects number of bits to be shifted out during a transfer in Multiple Transfer Operation. The field sets the number of SCK cycles that the bus master will generate to complete the transfer. The number of SCK cycles used will be one more than the value in the MTOCNT field. The number of SCK cycles defined by MTOCNT must be equal to or greater than the frame size. When TSBC is set, MTOCNT field has no effect.
8–10	Reserved, should be cleared.
11 TSBC	Timed Serial Bus Configuration The TSBC bit enables the Timed Serial Bus Configuration. This configuration allows 32-bit data to be used. It also allows t_{DT} to be programmable. See *Section 27.9.8, Timed serial bus (TSB)* for detailed information. 0 Timed Serial Bus Configuration disabled 1 Timed Serial Bus Configuration enabled If this bit is clear the DSPI_DSICR1 register value has no effect.
12 TXSS	Transmit Data Source Select The TXSS bit selects the source of data to be serialized. The source can be either data from host Software written to the DSPI DSI Alternate Serialization Data Register (DSPI_ASDR), or Parallel Input pin states latched into the DSPI DSI Serialization Data Register (DSPI_SDR). 0 Source of serialized data is the DSPI_SDR 1 Source of serialized data is the DSPI_ASDR

Table 611. DSPI_DSICR field description (continued)

Field	Description
13 TPOL	Trigger Polarity The TPOL bit selects the active edge of the hardware trigger input signal (HT). initiating DSI frames transfer. See *Section , DSI transfer initiation control* for more information. 0 Falling edge will initiate a transfer 1 Rising edge will initiate a transfer
14 TRRE	Trigger Reception Enable The TRRE bit enables the DSPI to initiate DSI frames transfer with external trigger signal. See *Section , DSI transfer initiation control* for more information. 0Trigger signal reception disabled 1 Trigger signal reception enabled
15 CID	Change In Data Transfer Enable The CID bit enables a change in serialization data to initiate DSI frames transfer. in DSI and CSI configurations. When the CID bit is set, DSI frames are initiated when the current DSI data differs from the previous DSI data shifted out. Refer to *Section , DSI transfer initiation control* for more information.
16 DCONT	DSI Continuous Peripheral Chip Select Enable The DCONT bit enables the PCS signals to remain asserted between transfers. The DCONT bit only affects the PCS signals in DSI master mode. See *Section , Continuous selection format* for details. When TSBC bit is set, DCONT bit has no effect. 0 Return Peripheral Chip Select signals to their inactive state after transfer is complete 1 Keep Peripheral Chip Select signals asserted after transfer is complete
17–19 DSICTAS[0:2]	DSI Clock and Transfer Attributes Select The DSICTAS field selects which of the DSPI_CTAR registers is used to provide transfer attributes for DSI frames. The DSICTAS field is used in DSI master mode. In DSI slave mode, the DSPI_CTAR1 is always selected.
20 DMS	Data Match Stop. DMS bit if set stops DSI frames transmissions if DDIF flag is set in the DSPI_SR register. 0 DDIF flag does not have effect on DSI frames transmissions. 1 DDIF flag stops DSI frame transmissions.
21 PES	Parity Error Stop. PES bit if set stops DSI operation if the parity error had happened in received DSI frame. 0 parity error does not stop DSI frame transmissions. 1 parity error stops all DSI frame transmissions
22 PE	Parity Enable. PE bit enables parity bit transmission and parity reception check for the DSI frames 0 No parity bit included/checked. 1 Parity bit is transmitted instead of last data bit in frame, parity checked for received frame

Table 611. DSPI_DSICR field description (continued)

Field	Description
23 PP	Parity Polarity. PP bit controls polarity of the parity bit transmitted and checked 0 Even Parity: number of “1” bits in the transmitted frame is even. The DSPI_SR[DPEF] bit is set if in the received frame number of “1” bits is odd. 1 Odd Parity: number of “1” bits in the transmitted frame is odd. The DSPI_SR[DPEF] bit is set if in the received frame number of “1” bits is even
24–31 DPCS*x*	DSI Peripheral Chip Select 0–7 The DPCS bits select which of the PCS signals to assert during a DSI master mode transfer. 0 Negate PCS[x] 1 Assert PCS[x]

DSPI DSI Serialization Data Register (DSPI_SDR)

The DSPI_SDR contains the states of the Parallel Input signals. The states of the Parallel Input signals are latched into the DSPI_SDR on the rising edge of every system clock. The DSPI_SDR is read-only. When the TXSS bit in the DSPI_DSICR is cleared, the data in the DSPI_SDR is used as the source of the DSI frames.

Figure 594. DSPI DSI Serialization Data Register (DSPI_SDR)

Address: DSPI_BASE + 0xC0

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SER_DATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SER_DATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 612. DSPI_SDR field description

Field	Description
0–31 SER_DATA [30:31]	Serialized Data The SER_DATA field contains the signal states of the Parallel Input signals.

DSPI DSI Alternate Serialization Data Register (DSPI_ASDR)

The DSPI_ASDR provides means for host software to write the data to be serialized. When the TXSS bit in the DSPI_DSICR is set, the data in the DSPI_ASDR is the source of the DSI frames. Writes to the DSPI_ASDR take effect on the next frame boundary.

Figure 595. DSPI DSI Alternate Serialization Data Register (DSPI_ASDR)

Address: DSPI_BASE + 0xC4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	ASER_DATA															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	ASER_DATA															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 613. DSPI_ASDR field description

Field	Descriptions
0–31 ASER_DATA [0:31]	Alternate Serialized Data The ASER_DATA field holds the alternate data to be serialized.

DSPI DSI Transmit Comparison Register (DSPI_COMPR)

The DSPI_COMPR holds a copy of the last transmitted DSI data. The DSPI_COMPR is read-only. DSI data is transferred to this register as it is loaded into the TX Shift Register.

Figure 596. DSPI DSI Transmit Comparison Register (DSPI_COMPR)

Address: DSPI_BASE + 0xC8

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	COMP_DATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	COMP_DATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 614. DSPI_COMPR field description

Field	Description
0–31 COMP_DATA[0:31]	Compare Data The COMP_DATA field holds the last serialized DSI data.

DSPI DSI Deserialization Data Register (DSPI_DDR)

The DSPI_DDR holds the signal states for the Parallel Output signals. The DSPI_DDR is read-only and host software can read data from incoming DSI frames.

Figure 597. DSPI Deserialization Data Register (DSPI_DDR)

Address: DSPI_BASE + 0xCC

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	DESER_DATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	DESER_DATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 615. DSPI_DDR field description

Field	Descriptions
0–31 DESER_DATA [0:31]	Deserialized Data The DESER_DATA field holds deserialized data which is presented as signal states to the Parallel Output signals.

DSPI DSI Configuration Register 1 (DSPI_DSICR1)

The DSI Configuration Register 1 selects various attributes associated with TSB Configuration. The user must not write to the DSPI_DSICR1 while the DSPI is in the Running state. If TSBC bit is cleared the register value is ignored.

Figure 598. DSPI DSI Configuration Register 1 (DSPI_DSICR1)

Address: DSPI_BASE + 0xD0

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	TSBCNT					0	0	0	0	0	0	DSE1	DSE2
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	DPCS1_7	DPCS1_6	DPCS1_5	DPCS1_4	DPCS1_3	DPCS1_2	DPCS1_1	DPCS1_0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 616. DSPI_DSICR1 field description

Field	Description
0–2	Reserved, should be cleared.
3–7 TSBCNT[0:4]	Timed Serial Bus Operation Count When TSBC is set, TSBCNT defines the length of the data frame. TSBCNT field valid value is from 3 to 31. The TSBCNT field selects number of data bits to be shifted out during a transfer in TSB mode. The number of data bits in the data frame is one more than the value in the TSBCNT field.
8–13	Reserved, should be cleared.
14 DSE1	Data Select Enable1. When TBSC bit is set, the DSE1 bit controls insertion of the zero bit (Data Select) in the middle of the data frame. The insertion bit position is defined by FMSZ field of DSPI_CTARn register, selected by DSICTAS field of the DSPI_DSICR register. 0 No Zero bit inserted in the middle of the data frame. 1 Zero bit is inserted at the middle of the data frame. Total number of bits in the data frame is increased by 1.
15 DSE0	Data Select Enable0. When TBSC bit is set, the DSE0 bit controls insertion of the zero bit (Data Select) in the beginning of the data frame. 0 No Zero bit inserted in the beginning of the frame. 1 Zero bit is inserted at the beginning of the data frame. Total number of bits in the data frame is increased by 1.
16–23	Reserved, should be cleared.
24–31 DPCS1_*x*	DSI Peripheral Chip Select 0–7 These bits define the PCSs to assert for the second part of the DSI frame when operating in TSB configuration with dual receiver. The DPCS1 bits select which of the PCS signals to assert during the second part of the DSI frame. The DPCS1 bits only control the assertions of the PCS signals in TSB mode. 0 Negate PCS[x] 1 Assert PCS[x]

DSPI DSI Serialization Source Select Register (DSPI_SSR)

DSPI DSI Serialization Source Select Register provides means to create combined frame for transmission, containing bits from DSPI_ASDR register and from DSPI_SDR register. Each bit in the DSPI_SSR register selects corresponding bit to be serialized. When DSPI_DSICR[TXSS] is set, the DSPI_SSR register value has no effect.

Figure 599. DSPI DSI Serialization Source Select Register (DSPI_SSR)

Address: DSPI_BASE + 0xD4 Access:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	SS															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	SS															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 617. DSPI_SSR Field Descriptions

Field	Description
0–31 SS[0:31]	Source Select. The SS bits select serialization source for DSI frame. Each SS bit selects data for corresponded bit in the transmitted frame. 0 the bit in transmitted frame is taken from Parallel Input pin; 1 the bit in transmitted frame is taken from DSPI_ASDR register

DSPI DSI Parallel Input Select Registers 0 - 3 (DPSI_PISR0 - DPSI_PISR3)

DSPI DSI Parallel Input Select Registers 0 - 3 provide means to select each data bit for transmitted frame from 16 Parallel Input pins. Each Input Pin Select (IPS) field controls one bit in the transmitted frame. Each register contains control fields for 8 bits in the frame. The select field value is defined as 4 bits signed integer number. Selected Parallel Input pin number is defined as a difference between the field number and field value.

For example, if IPS16 is equal binary number 1111 (minus 1 decimal) bit 16 in the frame will be taken from Parallel Input pin number 17. When the IPS0 is equal -1, the bit 0 in the frame is taken from Parallel Input 1. When the IPS0 is equal +1, the bit 0 in the frame is taken from Parallel Input 31 and etc.

Please, note that the DSPI_PISR0-3 only preselect Parallel Input pins, final selection to the transmitted frame is done by DSPI_SSR register bits or DSPI_DSICR[TXSS] bit.

Figure 600. DSPI DSI Parallel Input Select Register 0 (DSPI_PISR0)

Address: DSPI_BASE + 0xD8 Access:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	IPS24				IPS25				IPS26				IPS27			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	IPS28				IPS29				IPS30				IPS31			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 618. DSPI_PISR0 Field Descriptions

Field	Description
0–3 IPS24	Input Pin Select 24. The IPS24 field selects Parallel Input pin for transmitted frame bit 24.
4–7 IPS25	Input Pin Select 25. The IPS25 field selects Parallel Input pin for transmitted frame bit 25.
8–11 IPS26	Input Pin Select 26. The IPS26 field selects Parallel Input pin for transmitted frame bit 26.
12–15 IPS27	Input Pin Select 27. The IPS27 field selects Parallel Input pin for transmitted frame bit 27.
16–19 IPS28	Input Pin Select 28. The IPS28 field selects Parallel Input pin for transmitted frame bit 28.
20–23 IPS29	Input Pin Select 29. The IPS29 field selects Parallel Input pin for transmitted frame bit 29.
24–27 IPS30	Input Pin Select 30. The IPS30 field selects Parallel Input pin for transmitted frame bit 30.
28–31 IPS31	Input Pin Select 31. The IPS31 field selects Parallel Input pin for transmitted frame bit 31.

Figure 601. DSPI DSI Parallel Input Select Register 1 (DSPI_PISR1)

Address: DSPI_BASE + 0xDC Access:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	IPS16				IPS17				IPS18				IPS19			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	IPS20				IPS21				IPS22				IPS23			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 619. DSPI_PISR1 Field Descriptions

Field	Description
0–3 IPS16	Input Pin Select 16. The IPS16 field selects Parallel Input pin for transmitted frame bit 16.
4–7 IPS17	Input Pin Select 17. The IPS17 field selects Parallel Input pin for transmitted frame bit 17.
8–11 IPS18	Input Pin Select 18. The IPS18 field selects Parallel Input pin for transmitted frame bit 18.
12–15 IPS19	Input Pin Select 19. The IPS19 field selects Parallel Input pin for transmitted frame bit 19.
16–19 IPS20	Input Pin Select 20. The IPS20 field selects Parallel Input pin for transmitted frame bit 20.
20–23 IPS21	Input Pin Select 21. The IPS21 field selects Parallel Input pin for transmitted frame bit 21.
24–27 IPS22	Input Pin Select 22. The IPS22 field selects Parallel Input pin for transmitted frame bit 22.
28–31 IPS23	Input Pin Select 23. The IPS23 field selects Parallel Input pin for transmitted frame bit 23.

Figure 602. DSPI DSI Parallel Input Select Register 2 (DSPI_PISR2)

Address: DSPI_BASE + 0xE0 Access:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	IPS8				IPS9				IPS10				IPS11			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	IPS12				IPS13				IPS14				IPS15			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 620. DSPI_PISR2 Field Descriptions

Field	Description
0–3 IPS8	Input Pin Select 8. The IPS8 field selects Parallel Input pin for transmitted frame bit 8.
4–7 IPS9	Input Pin Select 9. The IPS9 field selects Parallel Input pin for transmitted frame bit 9.
8–11 IPS10	Input Pin Select 10. The IPS10 field selects Parallel Input pin for transmitted frame bit 10.
12–15 IPS11	Input Pin Select 11. The IPS11 field selects Parallel Input pin for transmitted frame bit 11.
16–19 IPS12	Input Pin Select 12. The IPS12 field selects Parallel Input pin for transmitted frame bit 12.
20–23 IPS13	Input Pin Select 13. The IPS13 field selects Parallel Input pin for transmitted frame bit 13.
24–27 IPS14	Input Pin Select 14. The IPS14 field selects Parallel Input pin for transmitted frame bit 14.
28–31 IPS15	Input Pin Select 15. The IPS15 field selects Parallel Input pin for transmitted frame bit 15.

Figure 603. DSPI DSI Parallel Input Select Register 3 (DSPI_PISR3)

Address: DSPI_BASE + 0xE4 Access:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	IPS0				IPS1				IPS2				IPS3			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	IPS4				IPS5				IPS6				IPS7			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 621. DSPI_PISR3 Field Descriptions

Field	Description
0–3 IPS0	Input Pin Select 0. The IPS0 field selects Parallel Input pin for transmitted frame bit 0.
4–7 IPS1	Input Pin Select 1. The IPS1 field selects Parallel Input pin for transmitted frame bit 1.
8–11 IPS2	Input Pin Select 2. The IPS2 field selects Parallel Input pin for transmitted frame bit 2.
12–15 IPS3	Input Pin Select 3. The IPS3 field selects Parallel Input pin for transmitted frame bit 3.
16–19 IPS4	Input Pin Select 4. The IPS4 field selects Parallel Input pin for transmitted frame bit 4.
20–23 IPS5	Input Pin Select 5. The IPS5 field selects Parallel Input pin for transmitted frame bit 5.
24–27 IPS6	Input Pin Select 6. The IPS6 field selects Parallel Input pin for transmitted frame bit 6.
28–31 IPS7	Input Pin Select 7. The IPS7 field selects Parallel Input pin for transmitted frame bit 7.

DSPI DSI Deserialized Data Interrupt Mask Register (DSPI_DIMR)

The DSPI DSI Deserialized Data Interrupt Mask Register selects bits in the received DSI frame to be checked to generate the DDI interrupt.

Figure 604. DSPI DSI Deserialized Data Interrupt Mask Register (DSPI_DIMR)

Address: DSPI_BASE + 0xE8 Access:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MASK															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	MASK															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 622. DSPI_DIMR Field Descriptions

Field	Description
0–31 MASK[0:31]	MASK. The MASK bits define which bits in received deserialization data should be checked to produce the Deserialized Data Interrupt (DDI). 0 the bit in received DSI frame does not produce DDI interrupt. 1 the bit in received DSI frame can produce DDI interrupt if the data bit matches to configured polarity.

DSPI DSI Deserialized Data Polarity Interrupt Register (DSPI_DPIR)

The DSPI DSI Deserialized Data Polarity Interrupt Register defines what data bits value in thereceived DSI frame generates the DDI interrupt.

Figure 605. DSPI DSI Deserialized Data Polarity Interrupt Register (DSPI_DIPR)

Address: DSPI_BASE + 0xEC Access:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	DP															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	DP															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 623. DSPI_DIPR Field Descriptions

Field	Description
0–31 DP[0:31]	Data Polarity. The DP bits define what value of the received deserialization data sets the DSPI_SR[DDIF] bit. 0 if received bit is 0 the DSPI_SR[DDIF] bit is set. 1 if received bit is 1 the DSPI_SR[DDIF] bit is set.

27.9 Functional description

The Deserial Serial Peripheral Interface (DSPI) module supports full-duplex, synchronous serial communications between MCUs and peripheral devices. The DSPI can also be used to reduce the number of pins required for I/O by serializing and deserializing up to 32 Parallel Input/Output signals. All communications are done with SPI-like protocol.

The DSPI has three configurations:

- SPI configuration in which the DSPI operates as a basic SPI or a queued SPI.
- DSI configuration in which the DSPI serializes and deserializes Parallel Input/Output signals or bits from memory mapped register.
- CSI configuration in which the DSPI combines the functionality of the SPI and DSI configurations.

Field DSPI_MCR[DCONF] determines the DSPI configuration. See *Table 596* for the DSPI configuration values.

Registers DSPI_CTAR0 – DSPI_CTAR7 hold clock and transfer attributes. The SPI configuration allows to select which DSPI_CTAR to use on a frame by frame basis by setting a field in the SPI command. The DSI configuration statically selects which DSPI_CTAR to use. In CSI configuration priority logic determines if SPI data or DSI data is transferred and dictates what DSPI_CTAR is used for the data transfer. See *Section , DSPI Clock and Transfer Attributes Registers 0–7 (DSPI_CTAR0–DSPI_CTAR7)* for information on the fields of the DSPI_CTAR registers.

Typical master to slave connections are shown in *Figure 606*. When a data transfer operation is performed, data is serially shifted a predetermined number of bit positions. Because the modules are linked, data is exchanged between the master and the slave. The data that was in the master shift register is now in the shift register of the slave, and vice versa. At the end of a transfer, bit DSPI_SR[TCF] is set to indicate a completed transfer.

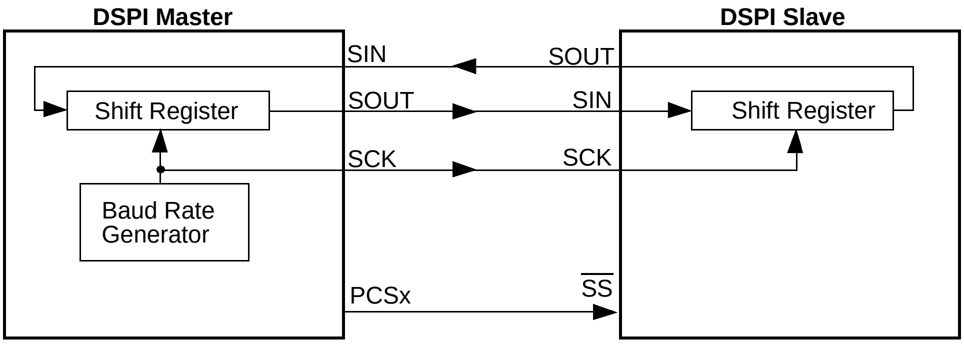


Figure 606. SPI and DSI Serial Protocol Overview

Generally more than one slave device can be connected to the DSPI master. Eight Peripheral Chip Select (PCS) signals of the DSPI masters can be used to select which of the slaves to communicate with.

The three DSPI configurations share transfer protocol and timing properties which are described independently of the configuration in *Section 27.9.6, Transfer formats*. The transfer rate and delay settings are described in *Section 27.9.5, DSPI baud rate and clock delay generation*.

27.9.1 Start and stop of DSPI transfers

The DSPI has two operating states: STOPPED and RUNNING. The states are independent of DSPI configuration. The default state of the DSPI is STOPPED. In the STOPPED state no serial transfers are initiated in master mode and no transfers are responded to in slave mode. The STOPPED state is also a safe state for writing the various configuration registers of the DSPI without causing undetermined results. In the RUNNING state serial transfers take place.

Bit DSPI_SR[TXRXS] indicates the DSPI’s operating state. The bit is set if the module is in RUNNING state.

The DSPI is started (DSPI transitions to RUNNING) when all of the following conditions are true:

- DSPI_SR[EOQF] bit is clear
- Device is not in the debug mode is or the DSPI_MCR[FRZ] bit is clear
- DSPI_MCR[HALT] bit is clear

The DSPI stops (transitions from RUNNING to STOPPED) after the current frame when any one of the following conditions exist:

- DSPI_SR[EOQF] bit is set
- Device in the debug mode and the DSPI_MCR[FRZ] bit is set
- DSPI_MCR[HALT] bit is set

State transitions from RUNNING to STOPPED occur on the next frame boundary if a transfer is in progress, or immediately if no transfers are in progress.

27.9.2 Serial peripheral interface (SPI) configuration

The SPI configuration transfers data serially using a shift register and a selection of programmable transfer attributes. The DSPI is in SPI configuration when field DSPI_MCR[DCONF] is 0b00. The SPI frames can be from 4 to 16 bits long. Host CPU or a DMA controller transfer the SPI data from the external to DSPI RAM queues to a transmit First-In First-Out (TX FIFO) buffer. The received data is stored in entries in the Receive FIFO (RX FIFO) buffer. Host CPU or the DMA controller transfer the received data from the RX FIFO to memory external to the DSPI. The FIFO buffers operation is described in *Section , Transmit first-in first-out (TX FIFO) buffering mechanism* and *Section , Receive first-in first-out (RX FIFO) buffering mechanism*. The interrupt and DMA request conditions are described in *Section 27.9.10, Interrupts/DMA requests*.

The SPI configuration supports two module-specific modes: master mode and slave mode. The FIFO operations are similar for both modes. The main difference is that in master mode the DSPI initiates and controls the transfer according to the fields in the SPI command field of the TX FIFO entry. In slave mode the DSPI only responds to transfers initiated by a bus master external to the DSPI and the SPI command field space is used for the 16 most significant bitS of the transmit data.

Master mode

In SPI master mode the DSPI initiates the serial transfers by controlling the Serial Communications Clock (SCK) and the Peripheral Chip Select (PCS) signals. The SPI command field in the executing TX FIFO entry determines which of the DSPI_CTAR registers will be used to set the transfer attributes and which PCS signal to assert. The command field also contains various bits that help with queue management and transfer protocol. See *Section , DSPI PUSH TX FIFO Register (DSPI_PUSHR)* for details on the SPI command fields. The data field in the executing TX FIFO entry is loaded into the shift register and shifted out on the Serial Out (SOUT) pin. In SPI master mode, each SPI frame to be transmitted has a command associated with it allowing for transfer attribute control on a frame by frame basis.

Slave mode

In SPI slave mode the DSPI responds to transfers initiated by a SPI bus master. The DSPI does not initiate transfers. Certain transfer attributes such as clock polarity, clock phase and frame size must be set for successful communication with a SPI master. The SPI slave mode transfer attributes are set in the DSPI_CTAR0.

FIFO disable operation

The FIFO disable mechanisms allow SPI transfers without using the TX FIFO or RX FIFO. The DSPI operates as a double-buffered simplified SPI when the FIFOs are disabled. The FIFOs are disabled separately; setting the DSPI_MCR[DIS_TXF] bit disables the TX FIFO, and setting the DSPI_MCR[DIS_RXF] bit disables the RX FIFO.

The FIFO Disable mechanisms are transparent to the user and to host software; Transmit data and commands are written to the DSPI_PUSHR and received data is read from the DSPI_POPR.

When the TX FIFO is disabled the TFFF, TFUF and TXCTR fields in DSPI_SR behave as if there is a one-entry FIFO but the contents of the DSPI_TXFR registers and TXNXTPTR are undefined. Likewise, when the RX FIFO is disabled, the RFDF, RFOF and RXCTR fields in the DSPI_SR behave as if there is a one-entry FIFO, but the contents of the DSPI_RXFR registers and POPNXTPTR are undefined.

Transmit first-in first-out (TX FIFO) buffering mechanism

The TX FIFO functions as a buffer of SPI data and SPI commands for transmission. The TX FIFO holds from 1 to 16 words, each consisting of a command field and a data field. The number of entries in the TX FIFO is device specific. SPI commands and data are added to the TX FIFO by writing to the DSPI PUSH TX FIFO Register (DSPI_PUSHR). TX FIFO entries can only be removed from the TX FIFO by being shifted out or by flushing the TX FIFO.

The TX FIFO Counter field DSPI_SR[TXCTR] indicates the number of valid entries in the TX FIFO. Field DSPI_SR[TXCTR] is updated every time the DSPI _PUSHR is written or SPI data is transferred into the shift register from the TX FIFO.

Field DSPI_SR[TXNXTPTR] indicates which TX FIFO Entry will be transmitted during the next transfer. Field DSPI_SR[TXNXTPTR] contains the positive offset from DSPI_TXFR0 in number of 32-bit registers. For example, TXNXTPTR equal to two means that the DSPI_TXFR2 contains the SPI data and command for the next transfer. Field DSPI_SR[TXNXTPTR] is incremented every time SPI data is transferred from the TX FIFO to the shift register. The maximum value of the field is equal to DSPI_HCR[TXFR] and it rolls over after reaching the maximum.

Filling the TX FIFO

Host software or other intelligent blocks can add (push) entries to the TX FIFO by writing to the DSPI_PUSHR. When the TX FIFO is not full, the TX FIFO Fill Flag (TFFF) in the DSPI_SR is set. The TFFF bit is cleared when TX FIFO is full and the DMA controller indicates that a write to DSPI_PUSHR is complete. Writing a ‘1’ to the TFFF bit also clears it. The TFFF can generate a DMA request or an interrupt request. See *Section , Transmit FIFO fill interrupt or DMA request* for details.

The DSPI ignores attempts to push data to a full TX FIFO, the state of the TX FIFO does not change and no error condition is indicated.

Draining the TX FIFO

The TX FIFO entries are removed (drained) by shifting SPI data out through the shift register. Entries are transferred from the TX FIFO to the shift register and shifted out as long as there are valid entries in the TX FIFO. Every time an entry is transferred from the TX FIFO to the shift register, the TX FIFO Counter decrements by one. At the end of a transfer, bit DSPI_SR[TCF] is set to indicate the completion of a transfer. The TX FIFO is flushed by writing a ‘1’ to bit DSPI_MCR[CLR_TXF].

If an external bus master initiates a transfer with a DSPI slave while the slave’s DSPI TX FIFO is empty, the Transmit FIFO Underflow Flag (TFUF) in the slave’s DSPI_SR is set. See *Section , Transmit FIFO underflow interrupt request* for details.

Receive first-in first-out (RX FIFO) buffering mechanism

The RX FIFO functions as a buffer for data received on the SIN pin. The RX FIFO holds from 1 to 16 received SPI data frames. The number of entries in the RX FIFO is device specific. SPI data is added to the RX FIFO at the completion of a transfer when the received data in the shift register is transferred into the RX FIFO. SPI data are removed (popped) from the RX FIFO by reading the DSPI POP RX FIFO Register (DSPI_POPR). RX FIFO entries can only be removed from the RX FIFO by reading the DSPI_POPR or by flushing the RX FIFO.

The RX FIFO Counter field DSPI_SR[RXCTR] indicates the number of valid entries in the RX FIFO. Field DSPI_SR[RXCTR] is updated every time the DSPI _POPR is read or SPI data is copied from the shift register to the RX FIFO.

Field DSPI_SR[POPNXTPTR] points to the RX FIFO entry that is returned when the DSPI_POPR is read. Field DSPI_SR[POPNXTPTR] contains the positive offset from DSPI_RXFR0 in number of 32-bit registers. For example, POPNXTPTR equal to two means that the DSPI_RXFR2 contains the received SPI data that will be returned when DSPI_POPR is read. Field DSPI_SR[POPNXTPTR] is incremented every time the DSPI_POPR is read. The maximum value of the field is equal to DSPI_HCR[RXFR] and it rolls over after reaching the maximum.

Filling the RX FIFO

The RX FIFO is filled with the received SPI data from the shift register. While the RX FIFO is not full, SPI frames from the shift register are transferred to the RX FIFO. Every time a SPI frame is transferred to the RX FIFO the RX FIFO Counter is incremented by one.

If the RX FIFO and shift register are full and a transfer is initiated, the RFOF bit in the DSPI_SR is set indicating an overflow condition. Depending on the state of the ROOE bit in the DSPI_MCR, the data from the transfer that generated the overflow is either ignored or shifted in to the shift register. If the ROOE bit is set, the incoming data is shifted in to the shift register. If the ROOE bit is cleared, the incoming data is ignored.

Draining the RX FIFO

Host CPU or a DMA can remove (pop) entries from the RX FIFO by reading the DSPI POP RX FIFO Register (DSPI_POPR). A read of the DSPI_POPR decrements the RX FIFO Counter by one. Attempts to pop data from an empty RX FIFO are ignored and the RX FIFO Counter remains unchanged. The data, read from the empty RX FIFO, is undetermined.

When the RX FIFO is not empty, the RX FIFO Drain Flag (RFDF) in the DSPI_SR is set. The RFDF bit is cleared when the RX_FIFO is empty and the DMA controller indicates that a read from DSPI_POPR is complete or by writing a ‘1’ to it.

27.9.3 Deserial serial interface (DSI) configuration

The DSI configuration supports pin count reduction by serializing Parallel Input signals or register bits and shifting them out in a SPI-like protocol. The timing and transfer protocol is described in *Section 27.9.6, Transfer formats*. The received serial frames are converted to a parallel form (deserialized) and placed on the Parallel Output signals or in the DSPI_DDR. The various features of the DSI configuration are set in the DSPI DSI Configuration Register (DSPI_DSICR).

The DSI frames can be from 4 to 32 bits. With Multiple Transfer Operation (MTO) the DSPI supports serial chaining of DSPI modules within a device to create DSI frames up to 64 bits, consisting of concatenated bits from multiple DSPIs. The DSPI also supports parallel chaining allowing several DSPIs and off-chip SPI devices to share the same Serial Communications Clock (SCK) and Peripheral Chip Select (PCS) signals. See *Section , Multiple transfer operation (MTO)* for details on the serial and parallel chaining support.

DSI Master mode

In DSI master mode the DSPI initiates and controls the DSI transfers. The DSI master has four different conditions that can initiate a transfer:

- Continuous
- Change in data
- Trigger signal
- Trigger signal combined with a change in data

The four transfer initiation conditions are described in *Section , DSI transfer initiation control*. Transfer attributes are set during initialization. Field DSPI_DSICR[DSICTAS] determines which of the DSPI_CTAR registers will control the transfer attributes.

Slave mode

In DSI slave mode the DSPI responds to transfers initiated by a SPI or DSI bus master. In this mode the DSPI does not initiate DSI transfers. Certain transfer attributes such as clock polarity and phase must be set for successful communication with a DSI master. The DSI slave mode Transfer attributes are set in the DSPI_CTAR1.

If the CID bit in the DSPI_DSICR is set and the data in the DSPI_COMPR differs from the selected source of the serialized data, the slave DSPI will assert the MTRIG signal. If the slave's HT signal is asserted and the TRRE is set, the slave DSPI asserts MTRIG. These features are included to support chaining of several DSPI. Details about the MTRIG signal is found in *Section , Multiple transfer operation (MTO)*.

DSI serialization

In the DSI configuration from 4 to 16 bits can be serialized using 2 different sources. The TXSS bit in the DSPI_DSICR selects between the DSPI DSI Serialization Data Register (DSPI_SDR) and the DSPI DSI Alternate Serialization Data Register (DSPI_ASDR) as the source of the serialized data. The DSPI_SDR holds the latest Parallel Input signal values which is sampled at every rising edge of the system clock. The DSPI_ASDR is written by host software and used as an alternate source of serialized data.

The DSPI_PISR0–3 registers allow to change relative position of the Parallel input pins in the transmitted frame. Each transmitted frame bit can be selected from 16 adjacent Parallel Inputs by writing IPSn fields. The IPSn field is treated as a 4-bit integer number, representing numbers from –8 to 7. The Parallel Input pin number, selected by IPSn field is defined by the difference between sum IPSn field number (n) and the IPSn field value. If the operation result is negative the number 32 should be added. If the result is higher than 32, 32 should be subtracted from the result.

For example, IPS0, set to minus 1 (binary 1111), preselects Parallel Input 1 to 0 position in the transmitted frame.

IPS6, set to 3 (binary 0011), preselects Parallel Input 3 to be bit number 6 in the transmitted frame, while the value minus 2 (1110) preselects Parallel Input 8.

IPS31, set to minus 8 (binary 1000), preselects Parallel Input 7 to be bit number 31 in the transmitted frame.

(Of course, the Parallel Input pin state, to be transmitted, should be selected by TXSS and the frame size should be higher than the bit position in the preselected frame.)

The DSPI_SSR provides additional way to create the frame for transmission. Each bit from this register is OR'd with the TXSS bit and controls individual transmitted bit source. This way, the transmitted frame can have any combination of the DSPI_SDR and DSPI_ASDR bits. This feature allows control SPI based devices, requiring control and data fields in the frame. Control field may come from DSPI_ASDR, set by the device's CPU, while data field can be generated by device peripheral modules, such as PWM timers.

A copy of the last 32-bit DSI frame shifted out of the Shift Register is stored in the DSPI DSI Transmit Comparison Register (DSPI_COMPR). This register provides added visibility for debugging and it serves as a reference for transfer initiation control. *Figure 607* shows the DSI Serialization logic.

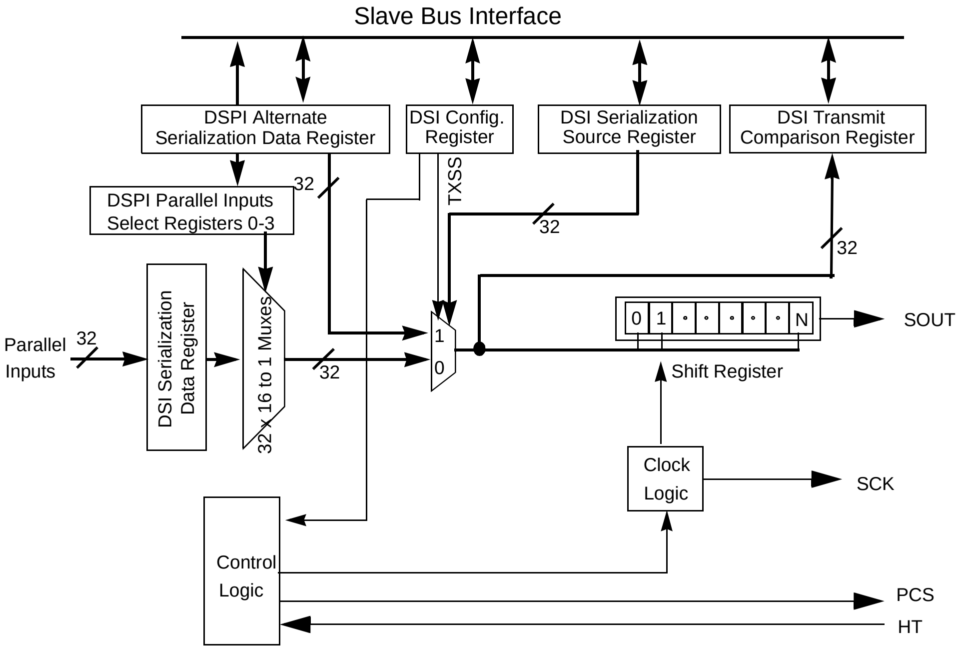


Figure 607. DSI serialization diagram

DSI deserialization

When all bits in a DSI frame have been shifted in, the frame is copied to the DSPI DSI Deserialization Data Register (DSPI_DDR). This register presents the deserialized data as Parallel Output signal values. The DSPI_DDR is memory mapped to allow host software to read the deserialized data directly.

The received data is bit-wise compared to the value of the DSI Deserialized Data Polarity Interrupt Register, bit-wise AND'ed with DSI Deserialized Interrupt Mask Register and the results OR'ed to produce the DDIF flag in the DSPI_SR, which in turn can cause a DDI interrupt request if the DDIFRE bit of DSPI_RSER is set.

Figure 608 shows the DSI deserialization logic.

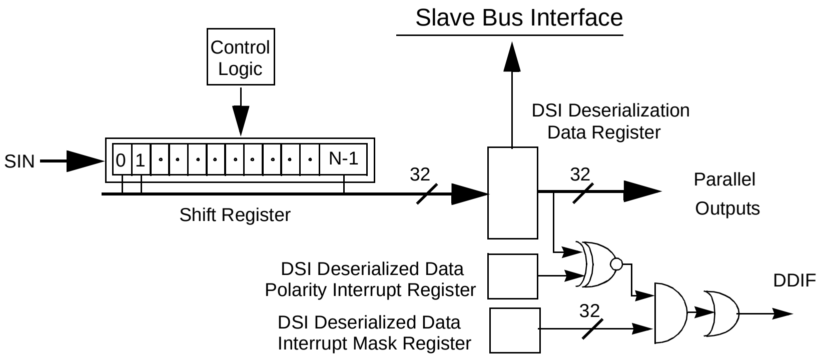


Figure 608. DSI deserialization diagram

DSI transfer initiation control

Data transfers for a master DSPI in DSI configuration are initiated by a condition. The transfer initiation conditions are selected by the TRRE and CID bits in the DSPI_DSICR. *Table 624* lists the four transfer initiation conditions.

Table 624. DSI data transfer initiation control

DSPI_DSICR bits		Transfer initiation control
TRRE	**CID**	
0	0	Continuous
0	1	Change in Data
1	0	Triggered
1	1	Triggered or Change in Data

Continuous control

For Continuous Control a new DSI frame shifts out when the previous transfer cycle has completed and the Delay after Transfer (t_{DT}) has elapsed.

Change in data control

For Change in Data Control a transfer is initiated when the data to be serialized has changed since the transfer of the last DSI frame. A copy of the previously transferred DSI data is stored in the DSPI_COMPR. When the data selected for the transfer from the DSPI_SDR and DSPI_ASDR registers is different from the data in the DSPI_COMPR a new DSI frame is transmitted. The $\overline{\text{MTRIG}}$ output signal is asserted every time a change in data is detected.

Triggered control

For Triggered Control initiation of a transfer is controlled by the Hardware Trigger signal (HT). The TPOL bit in the DSPI_DSICR selects the active edge of HT. For HT to have any affect, the TRRE bit in the DSPI_DSICR must be set.

Triggered or change in data control

For Triggered or Change in Data Control initiation of a transfer is controlled by the HT signal or by the detection of a change in data to be serialized.

Multiple transfer operation (MTO)

In DSI configuration the MTO feature allows for multiple DSPIs within a device to be chained together in a parallel or serial configuration. The parallel chaining allows multiple DSPIs internal to a device and multiple SPI devices external to a device to share SCK and PCS signals thereby helping to minimize device pin count. The serial chaining allows bits from multiple DSPIs to be concatenated into a single DSI frame. MTO is enabled by setting the MTOE bit in the DSPI_DSICR.

In parallel and serial chaining there is one bus master and multiple bus slaves. The bus master initiates and controls the transfers, but the DSPI slaves generate trigger signals for the bus DSPI master when an internal condition in the slave warrants a transfer. The DSPI slaves also propagate triggers from other slaves to the master. When a DSPI slave detects a trigger signal on its HT input, the slave generates a trigger signal on the MTRIG output.

Serial and parallel chaining require multiplexing of signals external to the DSPI.

Note: TSB operation is not available in MTO mode. TSBC and MTOE bits of DSPI_DSICR should not be set simultaneously.

Parallel chaining

Parallel chaining allows the PCS and SCK signals from a Master DSPI to be shared by internal Slave DSPIs and external Slave SPI devices, thus reducing pin utilization of the SPC564A70 MCU. Signal sharing reduces DSPI pin utilization. An example of a parallel chain is shown in *Figure 609*. In this example, the SOUT and SIN of the three DSPIs connect to separate external SPI devices, which share a common PCS and SCK.

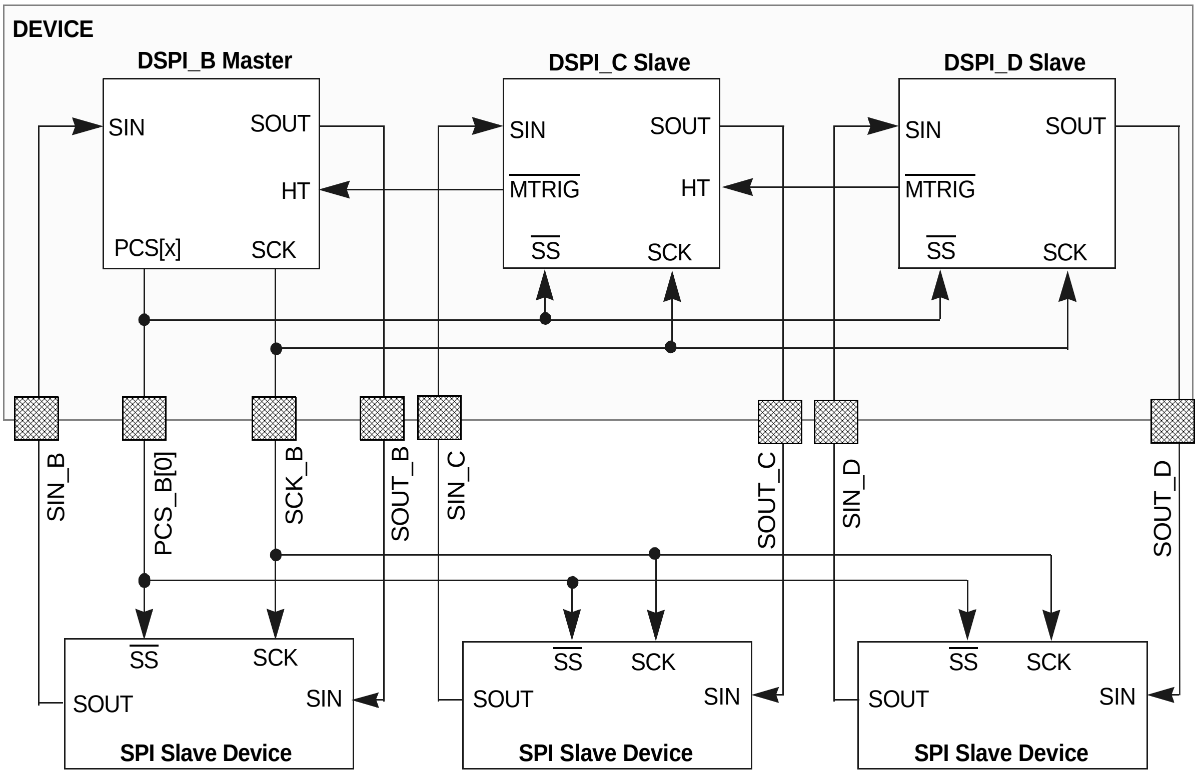


Figure 609. DSPI parallel chaining example

In the parallel chaining example, the SOUT and SIN of the three DSPIs connect to separate external SPI devices. All internal and external SPI blocks share PCS and SCK signals. DSPI_B controls and initiates all transfers, but the DSPI slaves each have a trigger output signal MTRIG that indicates to DSPI_B that a trigger condition has occurred in the DSPI slaves. When the slave DSPI has a change in data to be serialized, it asserts the MTRIG signal that propagates to DSPI_B which initiates the transfer.

Serial chaining

Serial chaining allows SPI operation with an external device that has more bits than one DSPI module. In a serial chain, one DSPI module operates as a master, the other DSPI modules operate as slaves.

The data output (SOUT) of the master is connected to the data input (SIN) of the slave. The SOUT of a slave is connected to the SIN of subsequent slaves until the last block in the chain, where the SOUT is connected to an external pin, which connects to the input of an external SPI device. The slave DSPI and external SPI device use the master peripheral chip select (PCS) and clock (SCK).

The Trigger input of the master allows a slave DSPI to trigger a transfer when a data change occurs in the slave DSPI and the slave DSPI is operating in Change in Data mode. The Trigger input of the master is connected to MTRIG output of the slave.

The concatenated frames can be from 8 to 64 bits long. *Figure 610* shows an example of how the blocks can be connected in the SPC564A70.

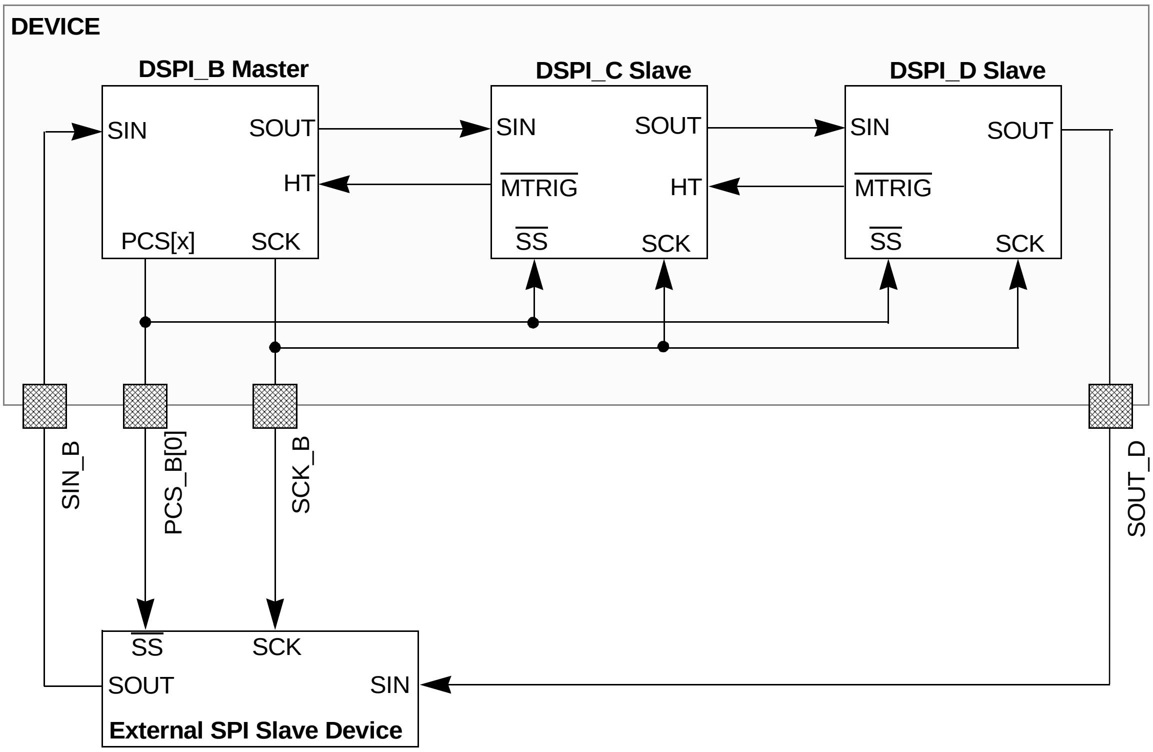


Figure 610. DSPI serial chaining example

The SOUT of DSPI_B is connected to the SIN of DSPI_C, the SOUT of DSPI_C is connected to the SIN of DSPI_D and the SOUT of the DSPI_D is connected to the SIN of the external SPI slave. The SOUT of the external SPI slave is connected to the SIN of DSPI_B.

DSPI_B controls and initiates all transfers, but the slave DSPIs use the trigger output signal MTRIG to indicate to DSPI_B that a trigger condition has occurred. When an on-chip DSPI slave has a change in data to be serialized it can assert the MTRIG signal to the DSPI master which initiates the transfer. The DSPI slaves also propagate trigger signals from other slaves to the DSPI master.

Field DSPI_DSICR[MTOCNT] in DSPI_B must be written with the total number of bits to be transferred. Field DSPI_DSICR[MTOCNT] must equal the sum of all FMSZ fields in the selected DSPI_CTAR registers for DSPI_B and all on-chip DSPI slaves. For example, if one 16-bit DSI frame is created by concatenating 8 bits from DSPI_B and 4 bits from DSPI_C and DSPI_D each, then DSPI_B's frame size must be set to 8, and the DSPI slaves' frame size must be set to 4 each. Field DSPI_DSICR[MTOCNT] in DSPI_B must be set to 16.

IMUX/SIU support for serial and parallel chaining

To support MTO, each DSPI in the SPC564A70 has multiplexers on the SIN, $\overline{\text{SS}}$, SCK, and HT inputs. The Internal Multiplexers (IMUX) reside in the SIU module on the SPC564A70.

27.9.4 Combined serial interface (CSI) configuration

The CSI configuration of the DSPI is used to support SPI and DSI functions on a frame by frame basis. CSI configuration allows interleaving of DSI data frames from the Parallel Input signals with SPI commands and data from the TX FIFO. The data returned from the bus slave is either used to drive the Parallel Output signals or it is stored in the RX FIFO. The CSI configuration allows serialized data and configuration or diagnostic data to be transferred to a slave device using only one serial link. The DSPI is in CSI configuration when field DSPI_MCR[DCONF] is 0b10. *Figure 611* shows an example of how a DSPI can be used with a deserializing peripheral that supports SPI control for control and diagnostic frames.

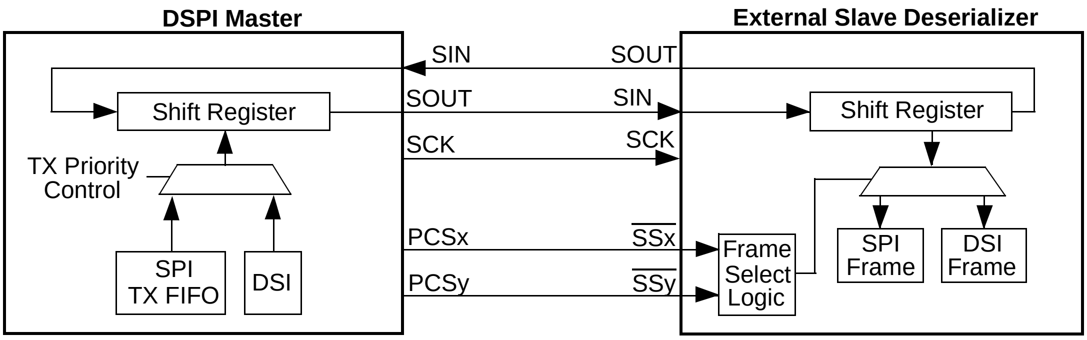


Figure 611. Example of system using DSPI in CSI configuration

In CSI configuration the DSPI transfers DSI data based on DSI transfer initiation control. When there are SPI commands in the TX FIFO, the SPI data has priority over the DSI frames. When the TX FIFO is empty, DSI transfer resumes.

Two peripheral chip select signals indicate whether DSI data or SPI data is transmitted. The user must configure the DSPI so that the two DSPI_CTAR registers associated with DSI data and SPI data assert different peripheral chip select signals denoted in the figure as PCSx and PCSy. The CSI configuration is only supported in master mode.

Data returned from the external slave while a DSI frame is transferred is placed on the Parallel Output signals. Data returned from the external slave while a SPI frame is transferred is moved to the RX FIFO. The TX FIFO and RX FIFO are fully functional in CSI mode.

CSI serialization

Serialization in the CSI configuration is similar to serialization in DSI configuration. The transfer attributes for SPI frames are determined by the DSPI_CTAR selected by the CTAS field in the SPI command halfword. The transfer attributes for the DSI frames are determined by the DSPI_CTAR selected by field DSPI_DSICR[DSICTAS].

The Parallel Inputs signal states are latched into the DSPI DSI Serialization Data Register (DSPI_SDR) on the rising edge of every system clock and serialized based on the transfer initiation control settings in the DSPI_DSICR. When SPI frames are written to the TX FIFO they have priority over DSI data from the DSPI_SDR and are transferred at the next frame boundary. A copy of the most recently transferred DSI frame is stored in the DSPI_COMPR. The Transfer Priority Logic selects the source of the serialized data and asserts the appropriate PCS signal.

CSI deserialization

The deserialized frames in CSI configuration goes into the DSPI_DDR or the RX FIFO based on the transfer priority logic. When DSI frames are transferred the returned frames are deserialized and latched into the DSPI_DDR. When SPI frames are transferred the returned frames are deserialized and written to the RX FIFO.

27.9.5 DSPI baud rate and clock delay generation

The SCK frequency and the delay values for serial transfer are generated by dividing the system clock frequency by a prescaler and a scaler with the option for doubling the baud rate. *Figure 612* shows conceptually how the SCK signal is generated.

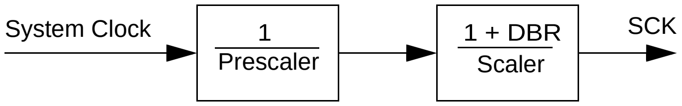


Figure 612. Communications clock prescalers and scalers

Baud rate generator

The baud rate is the frequency of the Serial Communication Clock (SCK). The system clock is divided by a prescaler (PBR) and scaler (BR) to produce SCK with the possibility of halving the scaler division. The DBR, PBR and BR fields in the DSPI_CTAR registers select the frequency of SCK by the formula in the BR field description. *Table 625* shows an example of how to compute the baud rate.

Table 625. Baud rate computation example

f_{sys}	PBR	Prescaler	BR	Scaler	DBR	Baud rate
100 MHz	0b00	2	0b0000	2	0	25 Mb/s
20 MHz	0b00	2	0b0000	2	1	10 Mb/s

PCS to SCK delay (t_{CSC})

The PCS to SCK delay is the length of time from assertion of the PCS signal to the first SCK edge. See *Figure 614* for an illustration of the PCS to SCK delay. The PCSSCK and CSSCK fields in the DSPI_CTAR*x* registers select the PCS to SCK delay by the formula in the CSSCK field description. *Table 626* shows an example of how to compute the PCS to SCK delay.

Table 626. PCS to SCK delay computation example

f_{sys}	PCSSCK	Prescaler	CSSCK	Scaler	PCS to SCK delay
100 MHz	0b01	3	0b0100	32	0.96 µs

PCSCSK and CSSCK fields have no effect in TSB configuration.

After SCK delay (t_{ASC})

The After SCK Delay is the length of time between the last edge of SCK and the negation of PCS. See *Figure 614* and *Figure 615* for illustrations of the After SCK delay. The PASC and ASC fields in the DSPI_CTAR*x* registers select the After SCK Delay by the formula in the ASC field description. *Table 627* shows an example of how to compute the After SCK delay.

Table 627. After SCK delay computation example

f_{sys}	PASC	Prescaler	ASC	Scaler	After SCK delay
100 MHz	0b01	3	0b0100	32	0.96 µs

PCASC and ASC fields have no effect in TSB configuration.

Delay after transfer (t_{DT})

The Delay after Transfer is the minimum time between negation of the PCS signal for a frame and the assertion of the PCS signal for the next frame. See *Figure 614* for an illustration of the Delay after Transfer. The PDT and DT fields in the DSPI_CTAR*x* registers select the Delay after Transfer by the formula in the DT field description. *Table 628* shows an example of how to compute the Delay after Transfer.

Table 628. Delay after transfer computation example

f_{sys}	PDT	Prescaler	DT	Scaler	Delay after transfer
100 MHz	0b01	3	0b1110	32768	0.98 ms

When in non-continuous clock mode the t_{DT} delay is configured according *Equation 25*. When in continuous clock mode and TSB is not enabled the delay is fixed at 1 SCK period.

In TSB mode the Delay after Transfer is equal to a number formed by concatenation of PDT and DT fields plus 1 of the SCK clock periods. See detailed information in *Section 27.9.8, Timed serial bus (TSB)*.

Peripheral chip select strobe enable ($\overline{\text{PCSS}}$)

The $\overline{\text{PCSS}}$ signal provides a delay to allow the PCS signals to settle after a transition occurs thereby avoiding glitches. When the DSPI is in master mode and PCSSE bit is set in the DSPI_MCR, $\overline{\text{PCSS}}$ provides a signal for an external demultiplexer to decode the DSPI_x_PCS[0] – PCS[4] signals into as many as 128 glitch-free PCS signals. *Figure 613* shows the timing of the PCSS signal relative to PCS signals.

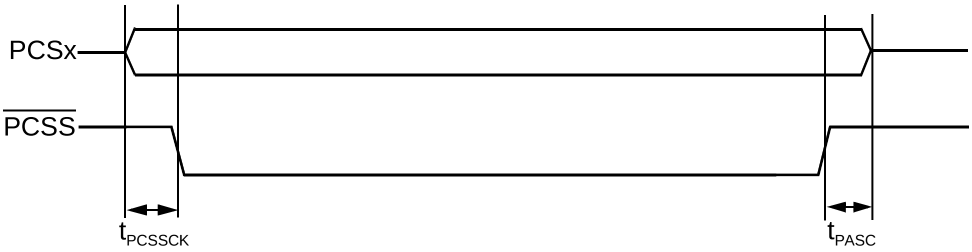


Figure 613. Peripheral chip select strobe timing

The delay between the assertion of the PCS signals and the assertion of $\overline{\text{PCSS}}$ is selected by field DSPI_CTAR[PCSSCK] based on the following formula:

Equation 27

$$t_{PCSSCK} = \frac{1}{f_{SYS}} \times PCSSCK$$

At the end of the transfer the delay between $\overline{\text{PCSS}}$ negation and PCS negation is selected by field DSPI_CTAR[PASC] based on the following formula:

Equation 28

$$t_{PASC} = \frac{1}{f_{SYS}} \times PASC$$

Table 629 shows an example of how to compute the t_{pcssck} delay.

Table 629. Peripheral chip select strobe assert computation example

f_{sys}	PCSSCK	Prescaler	Delay before transfer
100 MHz	0b11	7	70.0 ns

Table 630 shows an example of how to compute the t_{pasc} delay.

Table 630. Peripheral chip select strobe negate computation example

f_{sys}	PASC	Prescaler	Delay after transfer
100 MHz	0b11	7	70.0 ns

The $\overline{\text{PCSS}}$ signal is not supported when Continuous Serial Communication SCK or TSB mode are enabled.

27.9.6 Transfer formats

The SPI serial communication is controlled by the Serial Communications Clock (SCK) signal and the PCS signals. The SCK signal provided by the master device synchronizes shifting and sampling of the data on the SIN and SOUT pins. The PCS signals serve as enable signals for the slave devices.

When the DSPI is the bus master, the CPOL and CPHA bits in the DSPI Clock and Transfer Attributes Registers (DSPI_CTARx) select the polarity and phase of the serial clock, SCK. The polarity bit selects the idle state of the SCK. The clock phase bit selects if the data on SOUT is valid before or on the first SCK edge.

When the DSPI is the bus slave, CPOL and CPHA bits in the DSPI_CTAR0 (SPI) or DSPI_CTAR1 (DSI) select the polarity and phase of the serial clock. Even though the bus slave does not control the SCK signal, clock polarity, clock phase and number of bits to transfer must be identical for the master and the slave devices to ensure proper transmission.

The DSPI supports four different transfer formats:

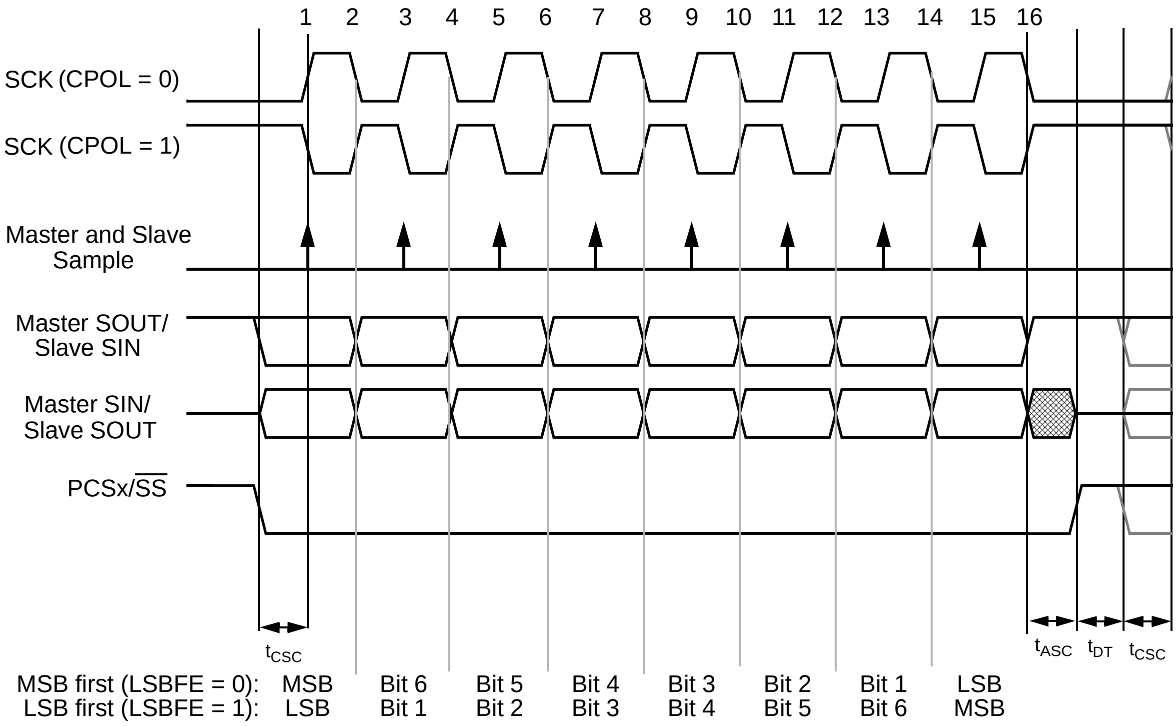
- Classic SPI with CPHA = 0
- Classic SPI with CPHA = 1
- Modified Transfer format with CPHA = 0
- Modified Transfer format with CPHA = 1

A modified transfer format is supported to allow for high-speed communication with peripherals that require longer setup times. The DSPI can sample the incoming data later than halfway through the cycle to give the peripheral more setup time. The MTFE bit in the DSPI_MCR selects between Classic SPI Format and Modified Transfer Format.

In the SPI and DSI configurations, the DSPI provides the option of keeping the PCS signals asserted between frames. See *Section , Continuous selection format* for details.

Classic SPI transfer format (CPHA = 0)

The transfer format shown in *Figure 614* is used to communicate with peripheral SPI slave devices where the first data bit is available on the first clock edge. In this format, the master and slave sample their SIN pins on the odd-numbered SCK edges and change the data on their SOUT pins on the even-numbered SCK edges.



t_{CSC} = PCS to SCK delay

t_{ASC} = After SCK delay

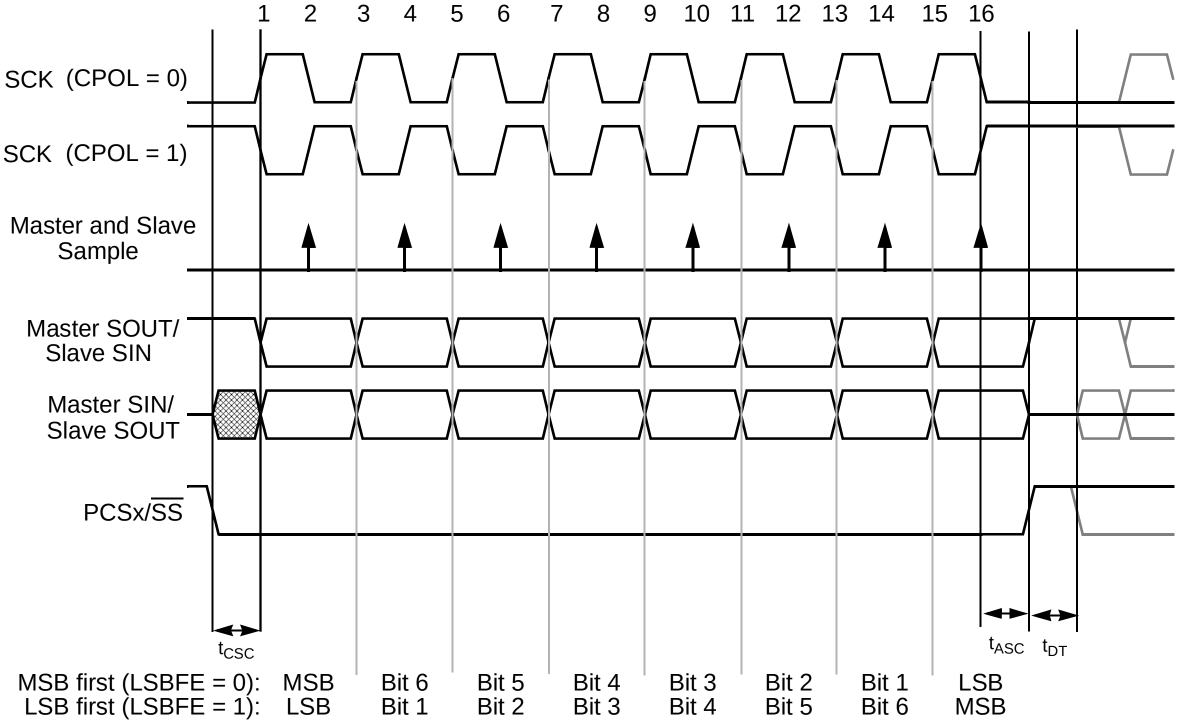
t_{DT} = Delay after Transfer (Minimum CS idle time)

Figure 614. DSPI transfer timing diagram (MTFE = 0, CPHA = 0, FMSZ = 8)

The master initiates the transfer by placing its first data bit on the SOUT pin and asserting the appropriate peripheral chip select signals to the slave device. The slave responds by placing its first data bit on its SOUT pin. After the t_{CSC} delay elapses, the master outputs the first edge of SCK. The master and slave devices use this edge to sample the first input data bit on their serial data input signals. At the second edge of the SCK the master and slave devices place their second data bit on their serial data output signals. For the rest of the frame the master and the slave sample their SIN pins on the odd-numbered clock edges and changes the data on their SOUT pins on the even-numbered clock edges. After the last clock edge occurs a delay of t_{ASC} is inserted before the master negates the PCS signals. A delay of t_{DT} is inserted before a new frame transfer can be initiated by the master.

Classic SPI transfer format (CPHA = 1)

This transfer format shown in *Figure 615* is used to communicate with peripheral SPI slave devices that require the first SCK edge before the first data bit becomes available on the slave SOUT pin. In this format the master and slave devices change the data on their SOUT pins on the odd-numbered SCK edges and sample the data on their SIN pins on the even-numbered SCK edges.



t_{CSC} = PCS to SCK delay

t_{ASC} = After SCK delay

t_{DT} = Delay after Transfer (minimum CS negation time)

Figure 615. DSPI Transfer Timing Diagram (MTFE = 0, CPHA = 1, FMSZ = 8)

The master initiates the transfer by asserting the PCS signal to the slave. After the t_{CSC} delay has elapsed, the master generates the first SCK edge and at the same time places valid data on the master SOUT pin. The slave responds to the first SCK edge by placing its first data bit on its slave SOUT pin.

At the second edge of the SCK the master and slave sample their SIN pins. For the rest of the frame the master and the slave change the data on their SOUT pins on the odd-numbered clock edges and sample their SIN pins on the even-numbered clock edges. After the last clock edge occurs a delay of t_{ASC} is inserted before the master negates the PCS signal. A delay of t_{DT} is inserted before a new frame transfer can be initiated by the master.

Modified SPI/DSI transfer format (MTFE = 1, CPHA = 0)

In this Modified Transfer Format both the master and the slave sample later in the SCK period than in Classic SPI mode to allow tolerate more delays in device pads and board traces. These delays become a more significant fraction of the SCK period as the SCK period decreases with increasing baud rates.

The master and the slave place data on the SOUT pins at the assertion of the PCS signal. After the PCS to SCK delay has elapsed the first SCK edge is generated. The slave samples the master SOUT signal on every odd numbered SCK edge. The DSPI in the slave mode when the MTFE bit is set also places new data on the slave SOUT on every odd numbered clock edge. Regular external slave, configured with CPHA = 0 format drives its SOUT output at every even numbered SCK clock edge.

The DSPI master places its second data bit on the SOUT line one system clock after odd numbered SCK edge if the system frequency to SCK frequency ratio is higher than three. If this ratio is below four the master changes SOUT at odd numbered SCK edge. The point where the master samples the SIN is selected by field DSPI_MCR[SMPL_PT]. *Table 596* lists the number of system clock cycles between the active edge of SCK and the master Sample point. The master sample point can be delayed by one or two system clock cycles. Field DSPI_MCR[SMPL_PT] should be set to ‘0’ if the system to SCK frequency ratio is less than 4.

The following timing diagrams illustrate the DSPI operation with MTFE = 1. Timing delays shown are:

- T_{csc} = PCS to SCK assertion delay
- T_{acs} = After SCK PCS negation delay
- T_{su_ms} = Master SIN setup time
- T_{hd_ms} = Master SIN hold time
- T_{vd_sl} = Slave data output valid time, time between slave data output SCK driving edge and data becomes valid.
- T_{su_sl} = Data setup time on slave data input
- T_{hd_sl} = Data hold time on slave data input
- T_{sys} = System clock period

Figure 616 shows the modified transfer format for CPHA = 0 and f_{sys}/f_{sck} = 4. Only the condition where CPOL = 0 is illustrated. Solid triangles show the data sampling clock edges. The two possible slave behaviors are shown.

- Signal, marked “SOUT of Ext Slave”, presents regular SPI slave serial output.
- Signal, marked “SOUT of DSPI Slave”, presents DSPI in the slave mode with MTFE bit set.

Other MTFE = 1 diagrams show DSPI SIN input as being driven by a regular external SPI slave, configured according DSPI master CPHA programming.

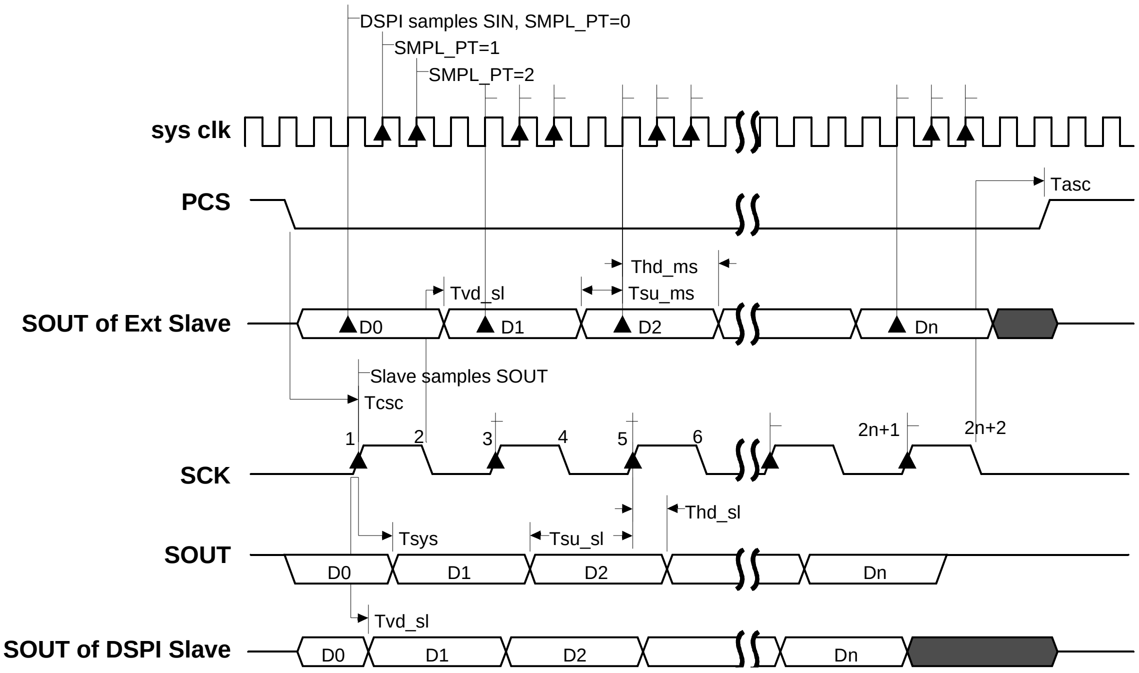


Figure 616. DSPI Modified Transfer Format (MTFE = 1, CPHA = 0, $f_{sck} = f_{sys}/4$)

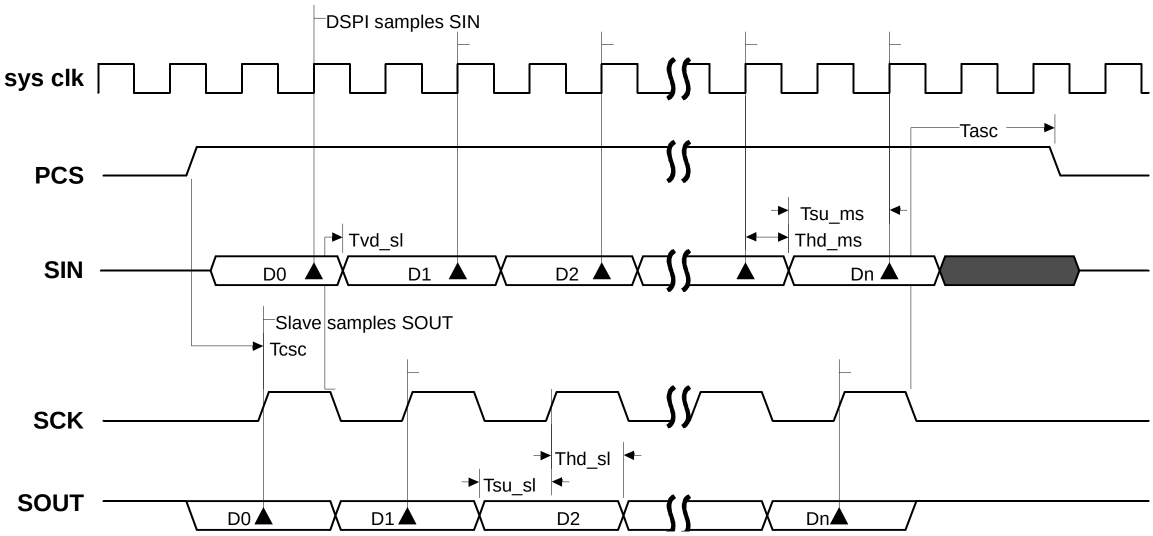


Figure 617. DSPI Modified Transfer Format (MTFE = 1, CPHA = 0, $f_{sck} = f_{sys}/2$)

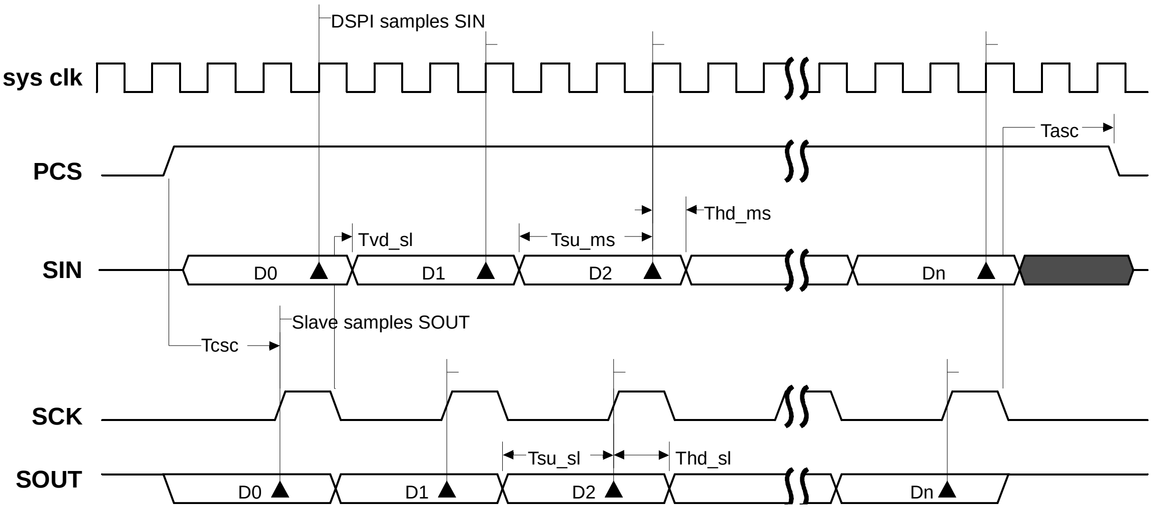


Figure 618. DSPI modified transfer format (MTFE = 1, CPHA = 0, $f_{sck} = f_{sys}/3$)

Modified SPI/DSI transfer format (MTFE = 1, CPHA = 1)

Figure 619 – *Figure 621* show the Modified Transfer Format for CPHA = 1. Only the condition, where CPOL = 0 is shown. At the start of a transfer the DSPI asserts the PCS signal to the slave device. After the PCS to SCK delay has elapsed the master and the slave put data on their SOUT pins at the first edge of SCK. The slave samples the master SOUT signal on the even numbered edges of SCK. The master samples the slave SOUT signal on the odd numbered SCK edges starting with the third SCK edge. The slave samples the last bit on the last edge of the SCK. The master samples the last slave SOUT bit one half SCK cycle after the last edge of SCK. No clock edge will be visible on the master SCK pin during the sampling of the last bit. **The SCK to PCS delay must be greater or equal to half of the SCK period.**

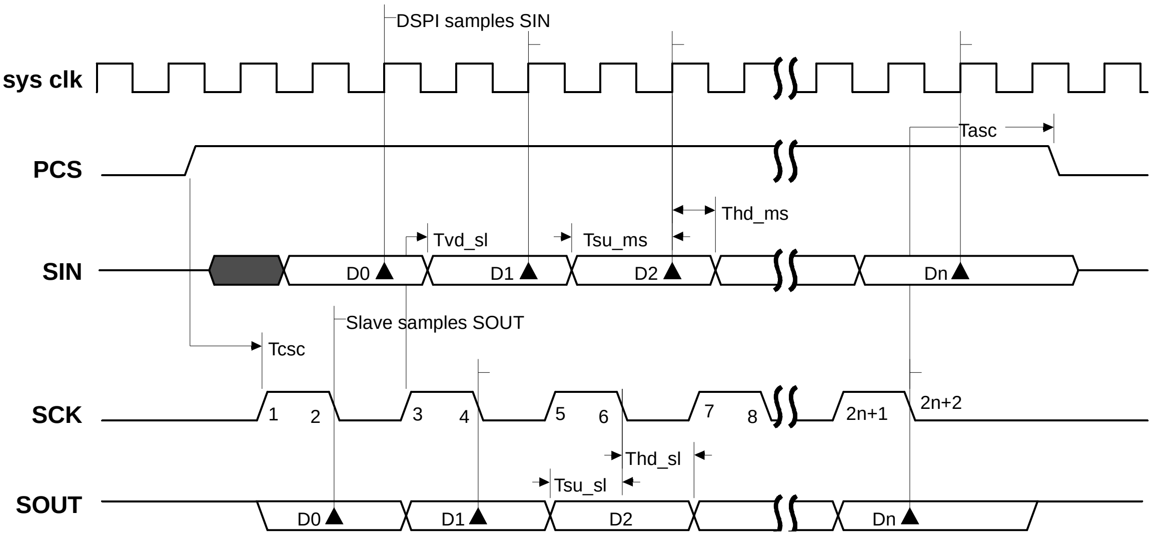


Figure 619. DSPI modified transfer format (MTFE = 1, CPHA = 1, $f_{sck} = f_{sys}/2$)

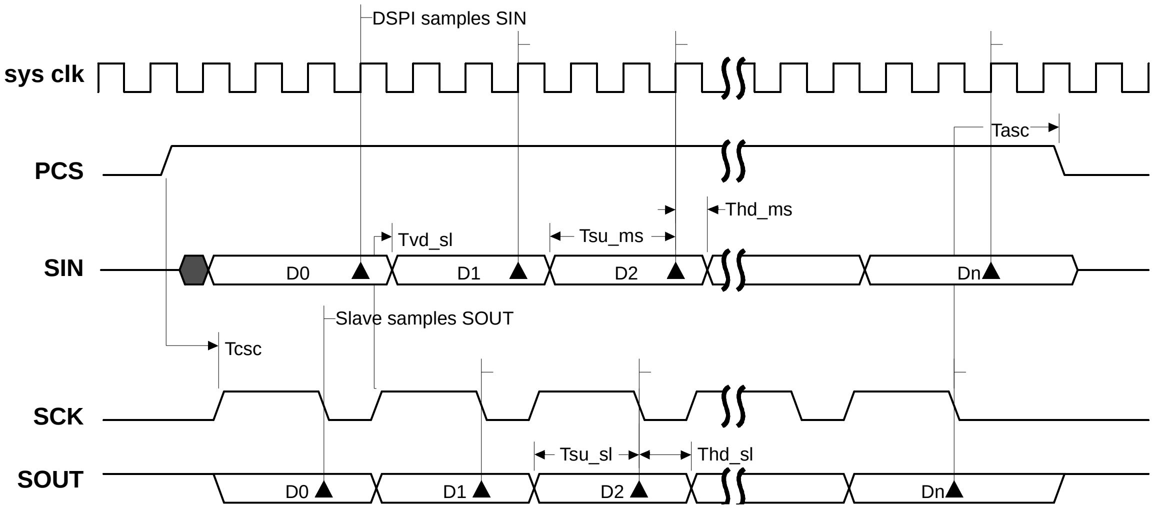


Figure 620. DSPI modified transfer format (MTFE = 1, CPHA = 1, $f_{sck} = f_{sys}/3$)

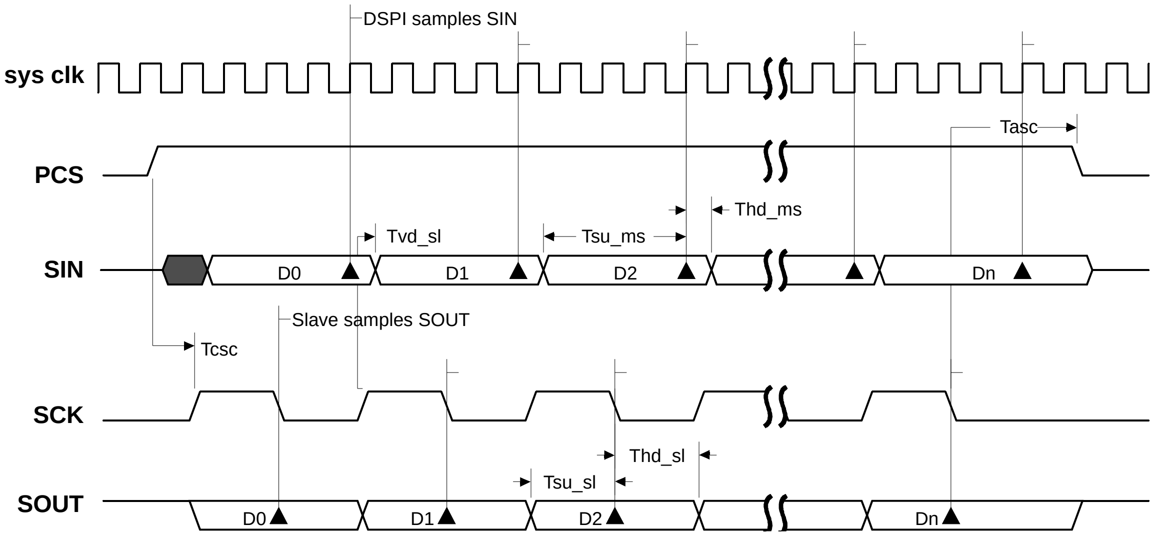
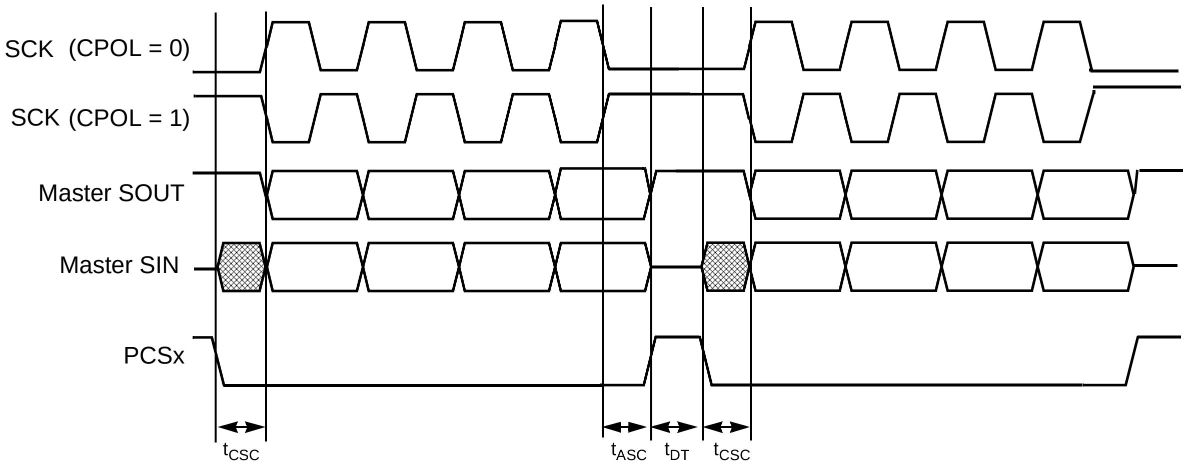


Figure 621. DSPI Modified transfer format (MTFE = 1, CPHA = 1, $f_{sck} = f_{sys}/4$)

Continuous selection format

Some peripherals must be deselected between every transfer. Other peripherals must remain selected between several sequential serial transfers. The Continuous Selection Format provides the flexibility to handle both cases. The Continuous Selection Format is enabled for the SPI configuration by setting the CONT bit in the SPI command. Continuous Selection is enabled for the DSI configuration by setting the DCONT bit in the DSPI_DSICR. The behavior of the PCS signals in the two configurations is identical so only SPI configuration will be described.

When the CONT bit = 0, the DSPI drives the asserted Chip Select signals to their idle states in between frames. The idle states of the Chip Select signals are selected by the PCSISn bits in the DSPI_MCR. *Figure 622* shows the timing diagram for two 4-bit transfers with CPHA = 1 and CONT = 0.



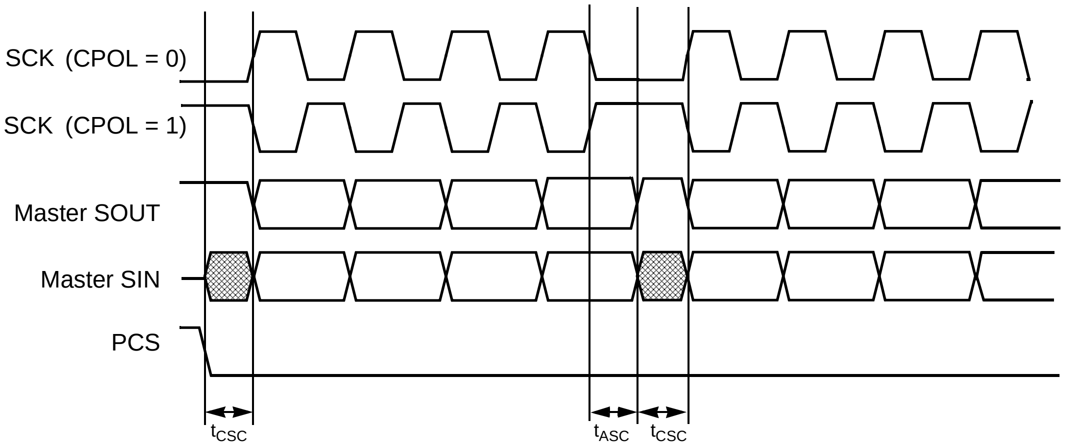
t_{CSC} = PCS to SCK delay

t_{ASC} = After SCK delay

t_{DT} = Delay after Transfer (minimum CS negation time)

Figure 622. Example of non-continuous format (CPHA = 1, CONT = 0)

When the CONT bit = 1, the PCS signal remains asserted for the duration of the two transfers. The Delay between Transfers (t_{DT}) is not inserted between the transfers. *Figure 623* shows the timing diagram for two 4-bit transfers with CPHA = 1 and CONT = 1.



t_{CSC} = PCS to SCK delay

t_{ASC} = After SCK delay

Figure 623. Example of continuous transfer (CPHA = 1, CONT = 1)

When using DSPI with continuous selection follow these rules:

- all transmit commands must have the same PCSn bits programming
- the DSPI_CTARs, selected by transmit commands, must be programmed with the same transfer attributes. Only field FMSZ can be programmed differently in these DSPI_CTARs.

Note: It is mandatory to fill the TXFIFO with the number of entries that will be concatenated together under one PCS assertion for both master and slave before the TXFIFO becomes empty. For example, while transmitting in master mode, it should be ensured that the last entry in the TXFIFO, after which TXFIFO becomes empty, must have the CONT bit in command frame as deasserted (i.e. CONT bit = 0).While operating in slave mode, it should be ensured that when the last-entry in the TXFIFO is comp.letely transmited (i.e. the corresponding TCF flag is asserted and TXFIFO is empty) the slave should be deselected for any further serial communication; else an underflow error occurs.

27.9.7 Continuous serial communications clock

The DSPI provides the option of generating a continuous SCK signal for slave peripherals that require a continuous clock.

Continuous SCK is enabled by setting bit DSPI_MCR[CONT_SCKE]. Continuous SCK is valid in all configurations.

Continuous SCK is only supported for CPHA = 1. Clearing CPHA is ignored if bit DSPI_MCR[CONT_SCKE] is set. Continuous SCK is supported for Modified Transfer Format.

Clock and transfer attributes for the Continuous SCK mode are set according to the following rules:

- The TX FIFO must be cleared before initiating any SPI configuration transfer.
- When the DSPI is in SPI configuration, CTAR0 shall be used initially. At the start of each SPI frame transfer, the CTAR specified by the CTAS for the frame should be CTAR0.
- When the DSPI is in DSI configuration, the CTAR specified by the DSICTAS field shall be used at all times.
- When the DSPI is in CSI configuration, the CTAR selected by the DSICTAS field shall be used initially. At the start of an SPI frame transfer, the CTAR specified by the CTAS value (which is CTAR0) for the frame shall be used. At the start of a DSI frame transfer, the CTAR specified by the DSICTAS field shall be used.
- In all configurations, the currently selected DSPI_CTAR remains in use until the start of a frame with a different DSPI_CTAR specified, or the Continuous SCK mode is terminated.

It is recommended to keep the baud rate the same while using the Continuous SCK. Switching clock polarity between frames while using Continuous SCK can cause errors in the transfer. Continuous SCK operation is not guaranteed if the DSPI is put into the External Stop mode or Module Disable mode.

Enabling Continuous SCK disables the PCS to SCK delay and the Delay after Transfer (t_{DT}) is fixed to one SCK cycle. When TSB configuration is enabled the t_{DT} is programmable from 1 to 65 SCK cycles. *Figure 624* shows timing diagram for Continuous SCK format with Continuous Selection disabled.

Note: When in Continuous SCK mode, for the SPI transfer CTAR0 should always be used, and the TX-FIFO must be clear using the DSPI_MCR[CLR_TXF] field before initiating transfer.

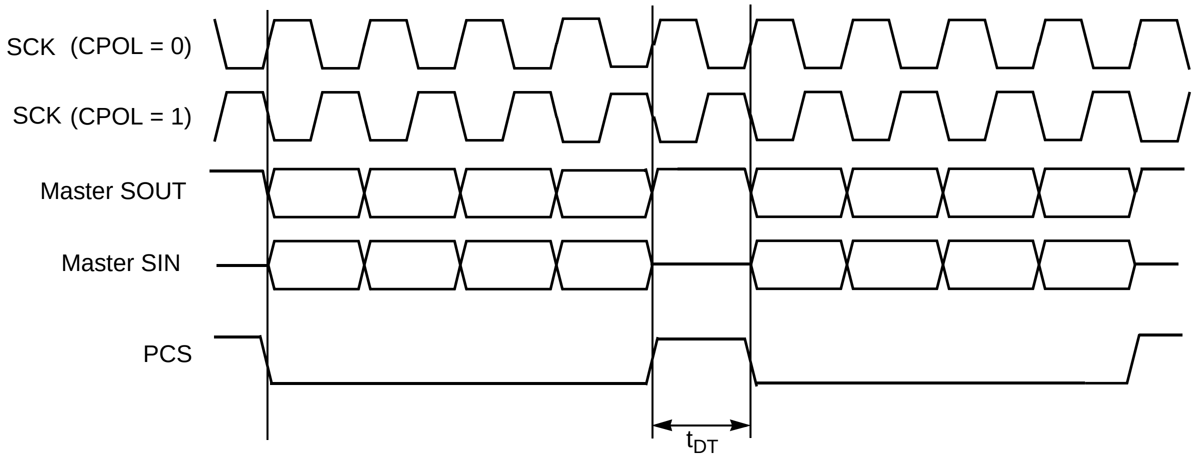


Figure 624. Continuous SCK timing diagram (CONT = 0)

If the CONT bit in the TX FIFO entry is set or the DCONT in the DSPI_DSICR is set, PCS remains asserted between the transfers. Under certain conditions, SCK can continue with PCS asserted, but with no data being shifted out of SOUT (SOUT pulled high). This can cause the slave to receive incorrect data. Those conditions include:

- Continuous SCK with CONT bit set, but no data in the transmit FIFO.
- Continuous SCK with CONT bit set and entering STOPPED state (refer to *Section 27.9.1, Start and stop of DSPI transfers*).
- Continuous SCK with CONT bit set and entering Stop mode or Module Disable mode.

Figure 625 shows timing diagram for Continuous SCK format with Continuous Selection enabled.

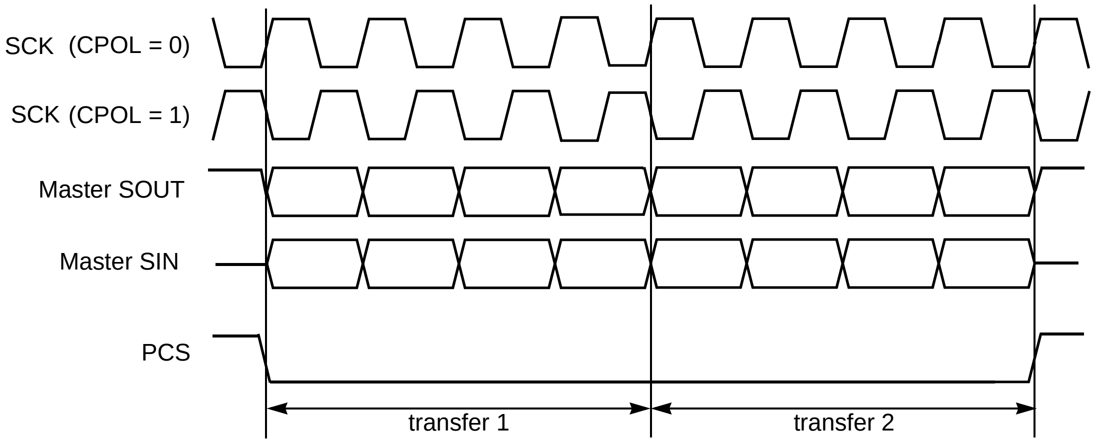


Figure 625. Continuous SCK timing diagram (CONT = 1)

27.9.8 Timed serial bus (TSB)

The DSPI can be programmed in Timed Serial Bus configuration by setting the TSBC bit in the DSPI_DSICR. See *Section , DSPI DSI Configuration Register (DSPI_DSICR)* for details.

TSB configuration provides the Micro Second Channel (MSC) downstream channel support.

The MSC upstream channel is not supported by the DSPI, but can be supported by any available Serial Communication Controller (SCI or UART) in the device.

To work in TSB mode the DSPI must be in master mode and in DSI (DCONF = 0b01) or CSI (DCONF = 0b10) configuration. Both Continuous and Non Continuous Serial Communication Clock (controlled by bit DSPI_MCR[CONT_SCKE]) are supported in the TSB mode.

Figure 626 shows the signals used in the TSB interface.

In the TSB configuration the DSPI is able to send from 4 to 34 bits MSC data frames (4 to 32 serialized data bits and up to 2 Data Selection zero bits). The serialized data bits source can be either:

- the DSPI DSI Alternate Serialization Data Register (DSPI_ASDR), written by the host software,
- Parallel Input pin states latched into the DSPI DSI Serialization Data Register (DSPI_SDR).

DSPI_DSICR TXSS bit or DSPI_SSR bits define the source of the data.

The Least Significant Bits of the DSPI_ASDR or DSPI_SDR registers are selected to be serialized if the data frame is set to less than 32 bits.

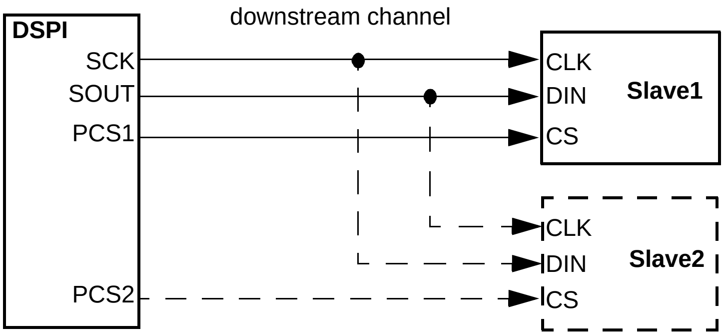


Figure 626. DSPI usage in the TSB configuration

The PCS signals are driven together with SOUT. The t_{CSC} and t_{ASC} delays are not available. Delay after Transfer (DT) is set in SCK clock periods as a binary number formed by concatenation of the DSPI_CTARn PDT and DT fields plus one, allowing to set DT from 1 to 64 serial clock periods. DT field provides least significant bits and PDT field provides most significant bits of the Delay after Transfer.

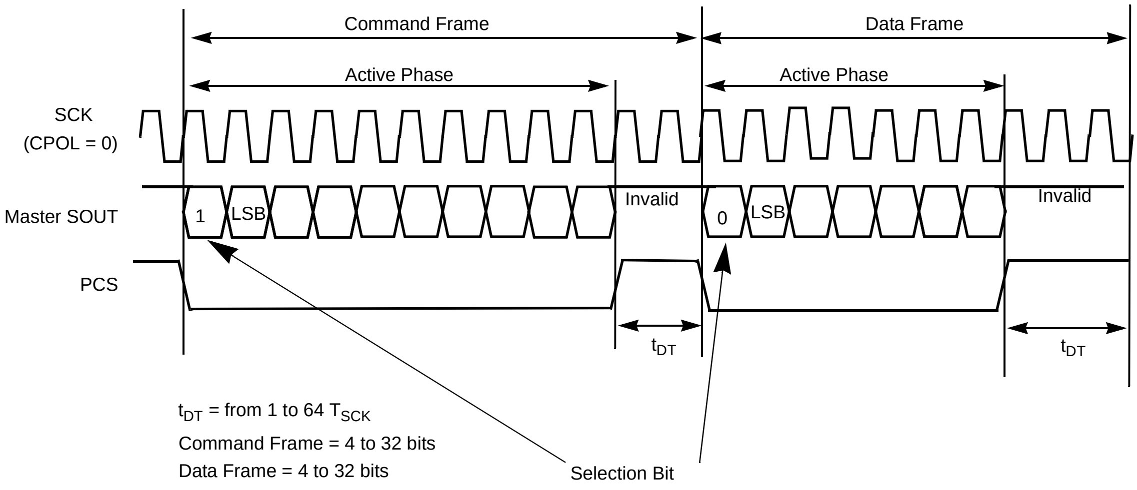


Figure 627. TSB Downstream frames

Figure 627 shows the two types of MSC downstream frames: command frame and data frame.

The first transmitted bit, called the selection bit, determines the frame type:

- The selection bit “0” indicates a data frame
- The selection bit “1” indicates a command frame

Data frame may contain up to two selection bits to support two external slave devices, (so called dual receiver configuration) or no selection bits at all.

The command frame can be written by software, through SPI TX FIFO, using one or two FIFO entries with help of the CONT bit. The data frame consists of up to 32 bits from the DSPI_SDR or DSPI_ASDR registers and up to two zero selection bits. The number of data bits in the data frame is defined by field DSPI_DSCICR1[TSBCNT].

The selection bit of the MSC command frames (1) can be implemented by software.

To comply with MSC specification, set DSPI_CTARn[LSBFE] to transmit the least significant bit first.

Regardless of the LSBFE bit setting, the Data Frame Selection Bits, if enabled, are always transmitted first, before the corresponding data subframes.

MSC dual receiver support with PCS switchover

When in TSB mode it is possible to switch the set of PCS signals that are driven during the first part of the frame to a different set of PCS signals during the second part of the frame. The bit, at which this switchover occurs, is defined by field FMSZ of the DSPI_CTARn register, which is selected by field DSPI_DSICR[DSICTAS].

Number of the bits, not including the Data Selection Bit, in the first part of the frame is equal to value of the FMSZ field plus one. During this part of the frame the PCS signal levels are controlled by DSPI_DSICR DPCSn bits, after that by DSPI_DSICR1 DPCS1_n bits.

The PCS switchover occurs at driving edge of the SCK clock output.

The second Data Selection Bit is inserted after the PCS switchover if enabled.

Data Frame with PCS switchover is shown in *Figure 628*.

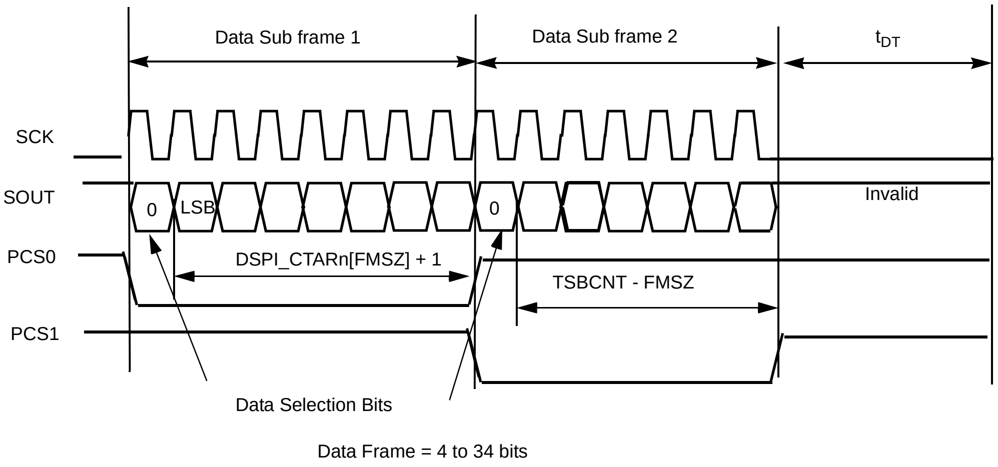


Figure 628. TSB data frame format for MSC dual receiver operation

27.9.9 Parity generation and check

The DSPI module can generate and check parity in the serial frame. The parity bit replaces the last transmitted bit in the frame. The parity is calculated for all transmitted data bits in frame, not including the last, would be transmitted, data bit. The parity generation/control is done on frame basis. The registers fields, setting frame size defines the total number of bits in the frame, including the parity bit. Thus, to transmit/receive the same number of data bits with parity check, increase the frame size by one versus the same data size frame without the parity check.

Parity can be selected as odd or even. Parity Errors in the received frame set Parity Error flags in the Status register. The Parity Error Interrupt Requests are generated if enabled. The DSPI module can be programmed to stop SPI or/and DSI frame transmission in case of a frame reception with parity error.

Parity for SPI frames

When the DSPI is in the master mode the parity generation is controlled by PE and PP bits of the TX FIFO entries (DSPI_PUSHR). Setting the PE bit enables parity generation for transmitted SPI frames and parity check for received frames. PP bit defines polarity of the parity bit.

When continuous PCS selection is used to transmit SPI data, two parity generation scenarios are available:

- Generate/check parity for the whole frame
- Generate/check parity for each subframe separately.

To generate/check parity for the whole frame set PE bit only in the last command/TX FIFO entry, forming this frame (with the DSPI_PUSHR).

To generate/check parity for each subframe set PE bit in each command/TX FIFO entry, forming this frame.

If the parity error occurs for received SPI frame, the DSPI_SR[SPEF] bit is set. If DSPI_MCR[PES] bit is set, the DSPI stops SPI frames transmission. To resume SPI operation clear the DSPI_SR[SPEF] or the DSPI_MCR[PES] bits.

In slave mode the parity is controlled by the PE and PP bits of the DSPI_CTAR0 register similar to the master mode parity generation without continuous PCS selection.

Parity for DSI frames

Parity generation is controlled by PE and PP bits of the DSPI_DSICR similar to the SPI frames. The parity is calculated and checked for each DSI frame. (DSPI_DSICR[DCONT] bit has no effect on parity generation.)

If the parity error occurs for received DSI frame, the DSPI_SR[DPEF] bit is set. To resume DSI operation clear the DSPI_SR[DPEF] bit.

27.9.10 Interrupts/DMA requests

The DSPI has several conditions that can generate interrupt requests and two conditions that can generate both interrupt or DMA requests. *Table 631* lists these conditions. The ‘x’ in the request type columns indicates which signals are connected on the SPC564A70.

Table 631. Interrupt and DMA request conditions

Condition	Flag	Request type	
		Interrupt	DMA
End of Queue (EOQ)	EOQF	x	
TX FIFO Fill	TFFF	x	x
Transfer Complete	TCF	x	
TX FIFO Underflow	TFUF	x	
RX FIFO Drain	RFDF	x	x
RX FIFO Overflow	RFOF	x	
SPI Parity Error	SPEF	x	
DSI Parity Error	DPEF	x	
DSI Deserialized Data Match	DDIF	x	

Each condition has a flag bit in the DSPI Status Register (DSPI_SR) and an Request Enable bit in the DSPI DMA/Interrupt Request Select and Enable Register (DSPI_RSER). The TX FIFO Fill Flag (TFFF) and RX FIFO Drain Flag (RFDF) generate interrupt requests or DMA requests depending on the TFFFDIRS and RFDFDIRS bits in the DSPI_RSER.

The DSPI module also provides a global interrupt request line, which is asserted when any of individual interrupt requests lines is asserted.

End of queue interrupt request

The End of Queue Request indicates that the end of a transmit queue is reached. The End of Queue Request is generated when the EOQ bit in the executing SPI command is set and bit DSPI_RSER[EOQFRE] is set.

Transmit FIFO fill interrupt or DMA request

The Transmit FIFO Fill Request indicates that the TX FIFO is not full. The Transmit FIFO Fill Request is generated when the number of entries in the TX FIFO is less than the maximum number of possible entries, and the TFFFRE bit in the DSPI_RSER is set. The TFFFDIRS bit in the DSPI_RSER selects whether a DMA request or an interrupt request is generated.

Transfer complete interrupt request

The Transfer Complete Request indicates the end of the transfer of a serial frame. The Transfer Complete Request is generated at the end of each frame transfer when the TCF_RE bit is set in the DSPI_RSER.

Transmit FIFO underflow interrupt request

The Transmit FIFO Underflow Request indicates that an underflow condition in the TX FIFO has occurred. The transmit underflow condition is detected only for the DSPI, operating in slave mode and SPI configuration. The TFUF bit is set when the TX FIFO of a DSPI is empty, and a transfer is initiated from an external SPI master. If the TFUF bit is set while the TFUFRE bit in the DSPI_RSER is set, an interrupt request is generated.

Receive FIFO drain interrupt or DMA request

The Receive FIFO Drain Request indicates that the RX FIFO is not empty. The Receive FIFO Drain Request is generated when the number of entries in the RX FIFO is not zero, and the RFDFRE bit in the DSPI_RSER is set. The RFDFDIRS bit in the DSPI_RSER selects whether a DMA request or an interrupt request is generated.

Receive FIFO overflow interrupt request

The Receive FIFO Overflow Request indicates that an overflow condition in the RX FIFO has occurred. A Receive FIFO Overflow request is generated when RX FIFO and shift register are full and a transfer is initiated. The RFOFRE bit in the DSPI_RSER must be set for the interrupt request to be generated.

Depending on the state of the ROOE bit in the DSPI_MCR, the data from the transfer that generated the overflow is either ignored or shifted in to the shift register. If the ROOE bit is set, the incoming data is shifted in to the shift register. If the ROOE bit is cleared, the incoming data is ignored.

SPI frame parity error interrupt request

The SPI Frame Parity Error Flag indicates that a SPI frame with parity error had been received. The SPEFRE bit in the DSPI_RSER must be set for the interrupt request to be generated.

DSI frame parity error interrupt request

The DSI Frame Parity Error Flag indicates that a DSI frame with parity error has been received. The DPEFRE bit in the DSPI_RSER must be set for the interrupt request to be generated.

Deserialized data match interrupt request

The Deserialized Data Match Flag (DDIF) indicates that a DSI frame with data matches DSPI_DPIR data, masked with DSPI_DIMR, had been received. The DDIFRE bit in the DSPI_RSER must be set for the interrupt request to be generated.

27.9.11 Buffered SPI operation

The DSPI can use a FIFO buffering mechanism to transmit and receive commands and data to and from external devices. The Transmit FIFO buffers SPI commands and data to be transferred. The Receive FIFO buffers incoming serial data. Both FIFOs are four entries deep. The TX FIFO stores 32-bit words when the DSPIs are configured for Master Mode. The 32-bit words are composed of 16-bit command fields and data fields up to 16 bits wide. The RX FIFOs store 16-bit words of received data from external devices. When the DSPI is configured for Slave Mode, the DSPI ignores the SPI command in the TX FIFO.

For queued operations, the SPI queues reside in system memory external to the DSPI. Data transfers between the memory and the DSPI FIFOs are accomplished through the use of the eDMA controller or through Host software. See *Figure 629* for a conceptual diagram of the queue data transfer control in the SPC564A70 MCU.

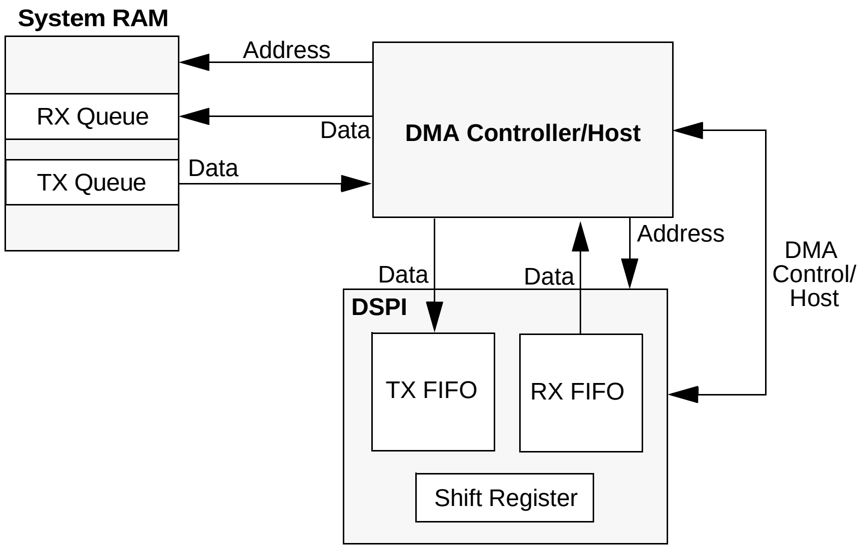


Figure 629. DSPI queue transfer control in the SPC564A70

27.9.12 Continuous peripheral chip select

For peripherals that must remain selected between sequential serial transfers, the DSPI provides the option of having the PCS signals asserted between transfers. For SPI transfers, the CONT bit in the SPI command fields selects the continuous PCS feature. For DSI and CSI transfers, the DCONT bit in the DSPI_DSICR selects the continuous PCS feature.

27.9.13 Peripheral chip select expansion and deglitching

The DSPI supports up to 256 Peripheral Chip Select Signals with the use of an external demultiplexer. Up to 128 Peripheral Chip Select Signals can be used if deglitching is desired. The PCSS signal provides the appropriate timing to enable and disable the demultiplexer for the DSPI_x_PCS[0:7] signals.

Figure 630 shows how an external 8-to-256 demultiplexer (on-board decoder) can be connected to the DSPI.

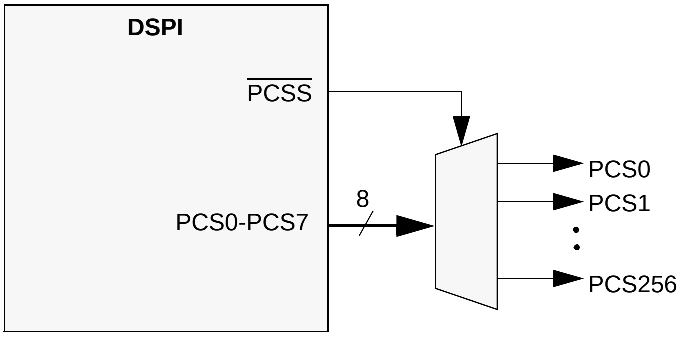


Figure 630. DSPI PCS expansion and deglitching

27.9.14 DMA and interrupt conditions

The DSPI has six conditions that can generate interrupt requests and two conditions that can generate both interrupt or DMA requests. *Table 632* lists the conditions. The ‘x’ in the request type columns indicates which signals are connected on the SPC564A70.

Table 632. DSPI interrupt and DMA request conditions

Condition	Flag	Request type	
		Interrupt	DMA
End of queue reached	EOQF	x	
TX FIFO is not full	TFFF	x	x
Transfer of current frame complete	TCF	x	
Attempt to transmit with an empty Transmit FIFO	TFUF	x	
RX FIFO is not empty	RFDF	x	x
Frame received while Receive FIFO is full	RFOF	x	

All request conditions are detected in the SPI configuration and in the CSI configuration. In DSI configuration only the transfer of current frame complete condition is detected.

Transmit FIFO underflow flag (TFUF)

The Transmit FIFO Underflow Flag indicates that an underflow condition in the TX FIFO has occurred. The transmit underflow condition is detected only for DSPI modules operating in slave mode and SPI configuration. The transmit underflow condition is detected when the TX FIFO of a DSPI operating as a SPI slave is empty, and a transfer is initiated from an external SPI master.

Receive FIFO overflow flag (RFOF)

The Receive FIFO Overflow Flag indicates that an overflow condition in the RX FIFO has occurred, and that data may be lost. The Receive FIFO Overflow Flag is asserted when the RX FIFO is full, a new frame has been received in the shift register, and a transfer is initiated.

27.9.15 Modified SPI transfer format

In Modified Transfer Format, the slave peripheral has more time to place data on the SOUT pin before the DSPI samples the data. In the Modified Transfer Format, the Master samples the incoming data towards the end of the transfer cycle. For correct operation of the Modified Transfer Format, the user must thoroughly analyze the SPI link timing budget.

27.9.16 LVDS pad usage

The differential transmitter pad driver LVDS support data rate up to 40 MHz. *Figure 631* describes the pad signals interface.

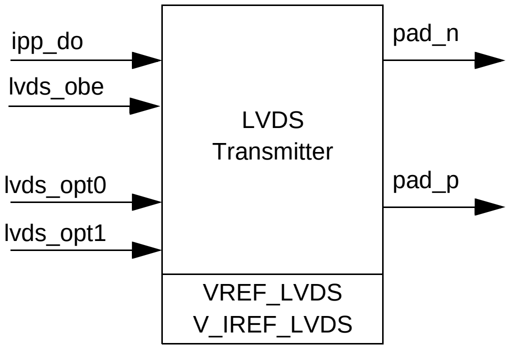


Figure 631. LVDS transmitter pad block diagram

Signals lvds_opt0 and lvds_opt1 control the voltage swing on the LVDS pad. These two signals are controlled by bits SRC[1:0] of the respective SIU_PCR. *Table 633* gives the configuration for these bits.

Table 633. LVDS pads voltage swing

SIU_PCRx		Current flowing in the driver	Differential voltage across pad_p and pad_n
SRC[0]	SRC[1]		
0	0	normal	default
0	1	increased	increased
1	0	decreased	decreased
1	1	normal	same as default

27.9.17 DSPI connections to eTPU_A, eMIOS and SIU

The three DSPI modules connect to the input and output channels of the eTPUs and the eMIOS. Some of the DSPI outputs connect to the External Interrupt Input Multiplexing sub-block in the SIU. See *Section , External interrupts* and *Section 15.6.19, External IRQ Input Select Register (SIU_EIISR)* for details on how the DSPI deserialized outputs can be used to trigger external interrupt requests.

DSPI_B connectivity

The DSPI_B connects to the eMIOS, eTPU_A and SIU as shown in *Figure 632*.

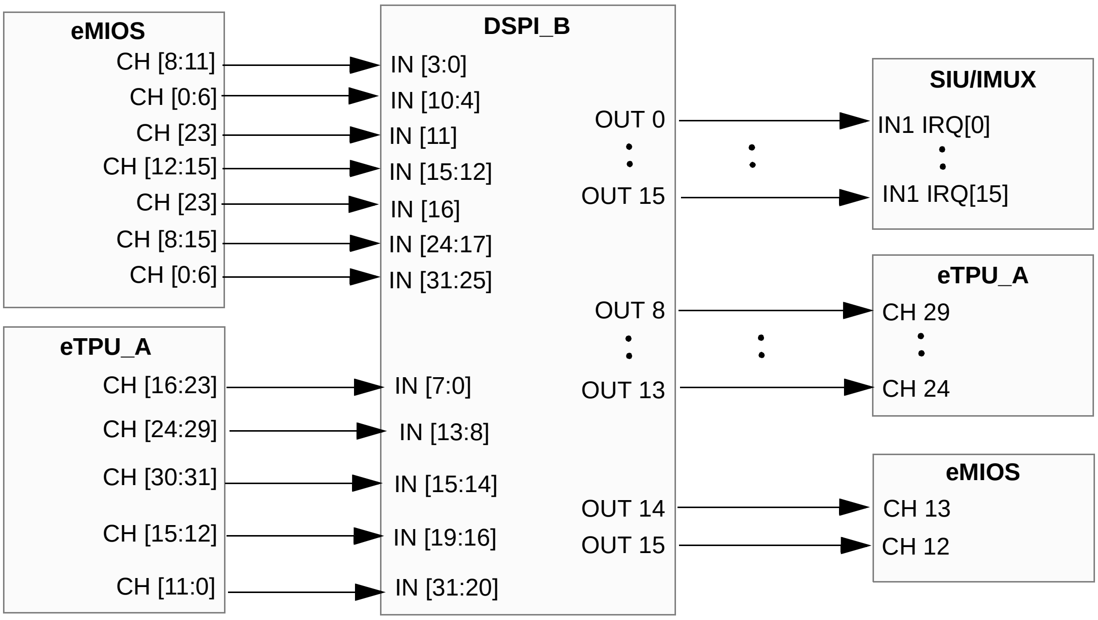


Figure 632. DSPI_B connectivity

Table 634 lists the DSPI_B connections.

Table 634. DSPI_B connectivity table

DSPI_B input	Connected to:
0	eMIOS Output Channel 11 eTPU_A Output Channel 23 GPDO350
1	eMIOS Output Channel 10 eTPU_A Output Channel 22 GPDO351
2	eMIOS Output Channel 9 eTPU_A Output Channel 21 GPDO352
3	eMIOS Output Channel 8 eTPU_A Output Channel 20 GPDO353
4	eMIOS Output Channel 6 eTPU_A Output Channel 19 GPDO354
5	eMIOS Output Channel 5 eTPU_A Output Channel 18 GPDO355

DSPI_B output	Connected to:
0	Input 1 on IMUX for External IRQ[0]
1	Input 1 on IMUX for External IRQ[1]
2	Input 1 on IMUX for External IRQ[2]
3	Input 1 on IMUX for External IRQ[3]
4	Input 1 on IMUX for External IRQ[4]
5	Input 1 on IMUX for External IRQ[5]

Table 634. DSPI_B connectivity table (continued)

DSPI_B input	Connected to:	DSPI_B output	Connected to:
6	eMIOS Output Channel 4 eTPU_A Output Channel 17 GPDO356	6	Input 1 on IMUX for External IRQ[6]
7	eMIOS Output Channel 3 eTPU_A Output Channel 16 GPDO357	7	Input 1 on IMUX for External IRQ[7]
8	eMIOS Output Channel 2 eTPU_A Output Channel 29 GPDO358	8	eTPU_A Input Channel 29, Input 1 on IMUX for External IRQ[8]
9	eMIOS Output Channel 1 eTPU_A Output Channel 28 GPDO359	9	eTPU_A Input Channel 28, Input 1 on IMUX for External IRQ[9]
10	eMIOS Output Channel 0 eTPU_A Output Channel 27 GPDO360	10	eTPU_A Input Channel 27, Input 1 on IMUX for External IRQ[10]
11	eMIOS Output Channel 23 eTPU_A Output Channel 26 GPDO361	11	eTPU_A Input Channel 26, Input 1 on IMUX for External IRQ[11]
12	eMIOS Output Channel 15 eTPU_A Output Channel 25 GPDO362	12	eTPU_A Input Channel 25, Input 1 on IMUX for External IRQ[12]
13	eMIOS Output Channel 14 eTPU_A Output Channel 24 GPDO363	13	eTPU_A Input Channel 24, Input 1 on IMUX for External IRQ[13]
14	eMIOS Output Channel 13 eTPU_A Output Channel 31 GPDO364	14	eMIOS Input Channel 13, Input 1 on IMUX for External IRQ[14]
15	eMIOS Output Channel 12 eTPU_A Output Channel 30 GPDO365	15	eMIOS Input Channel 12, Input 1 on IMUX for External IRQ[15]
16	eMIOS Output Channel 23 eTPU_A Output Channel 12 GPDO366	16	NC
17	eMIOS Output Channel 15 eTPU_A Output Channel 13 GPDO367	17	NC
18	eMIOS Output Channel 14 eTPU_A Output Channel 14 GPDO368	18	NC

Table 634. DSPI_B connectivity table (continued)

DSPI_B input	Connected to:	DSPI_B output	Connected to:
19	eMIOS Output Channel 13 eTPU_A Output Channel 15 GPDO369	19	NC
20	eMIOS Output Channel 12 eTPU_A Output Channel 0 GPDO370	20	NC
21	eMIOS Output Channel 11 eTPU_A Output Channel 1 GPDO371	21	NC
22	eMIOS Output Channel 10 eTPU_A Output Channel 2 GPDO372	22	NC
23	eMIOS Output Channel 9 eTPU_A Output Channel 3 GPDO373	23	NC
24	eMIOS Output Channel 8 eTPU_A Output Channel 4 GPDO374	24	NC
25	eMIOS Output Channel 6 eTPU_A Output Channel 5 GPDO375	25	NC
26	eMIOS Output Channel 5 eTPU_A Output Channel 6 GPDO376	26	NC
27	eMIOS Output Channel 4 eTPU_A Output Channel 7 GPDO377	27	NC
28	eMIOS Output Channel 3 eTPU_A Output Channel 8 GPDO378	28	NC
29	eMIOS Output Channel 2 eTPU_A Output Channel 9 GPDO379	29	NC
30	eMIOS Output Channel 1 eTPU_A Output Channel 10 GPDO380	30	NC
31	eMIOS Output Channel 0 eTPU_A Output Channel 11 GPDO381	31	NC

DSPI_C connectivity

The DSPI_C connects to eTPU_A and SIU as shown in *Figure 634*.

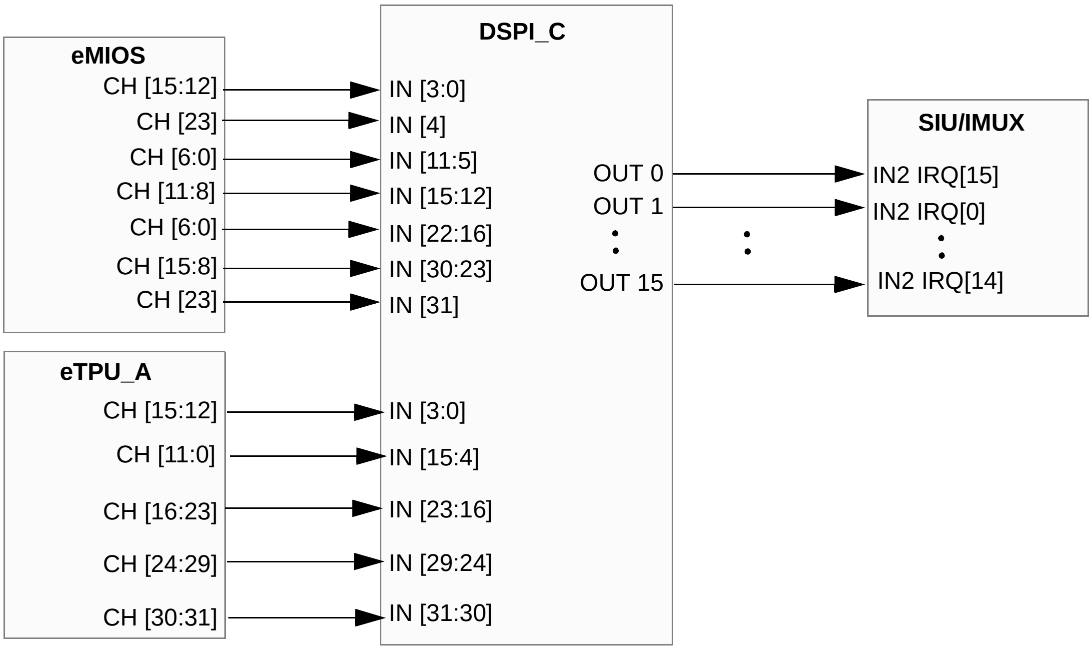


Figure 633. DSPI_C connectivity

Table 636 lists the DSPI_C connections.

Table 635. DSPI_C connectivity table

DSPI_C input	Connected to:
0	eMIOS Output Channel 7 eMIOS Output Channel 12 eTPU_A Output Channel 12 GPDO382
1	eMIOS Output Channel 16 eMIOS Output Channel 13 eTPU_A Output Channel 13 GPDO383
2	eMIOS Output Channel 17 eMIOS Output Channel 14 eTPU_A Output Channel 14 GPDO384

DSPI_C output	Connected to:
0	Input 2 on IMUX for External IRQ[15]
1	Input 2 on IMUX for External IRQ[0]
2	Input 2 on IMUX for External IRQ[1]

Table 635. DSPI_C connectivity table (continued)

DSPI_C input	Connected to:	DSPI_C output	Connected to:
3	eMIOS Output Channel 18 eMIOS Output Channel 15 eTPU_A Output Channel 15 GPDO385	3	Input 2 on IMUX for External IRQ[2]
4	eMIOS Output Channel 19 eMIOS Output Channel 23 eTPU_A Output Channel 0 GPDO386	4	Input 2 on IMUX for External IRQ[3]
5	eMIOS Output Channel 20 eMIOS Output Channel 0 eTPU_A Output Channel 1 GPDO387	5	Input 2 on IMUX for External IRQ[4]
6	eMIOS Output Channel 21 eMIOS Output Channel 1 eTPU_A Output Channel 2 GPDO388	6	Input 2 on IMUX for External IRQ[5]
7	eMIOS Output Channel 22 eMIOS Output Channel 2 eTPU_A Output Channel 3 GPDO389	7	Input 2 on IMUX for External IRQ[6]
8	eMIOS Output Channel 3 eTPU_A Output Channel 4 GPDO390	8	Input 2 on IMUX for External IRQ[7]
9	eMIOS Output Channel 4 eTPU_A Output Channel 5 GPDO391	9	Input 2 on IMUX for External IRQ[8]
10	eMIOS Output Channel 5 eTPU_A Output Channel 6 GPDO392	10	Input 2 on IMUX for External IRQ[9]
11	eMIOS Output Channel 6 eTPU_A Output Channel 7 GPDO393	11	Input 2 on IMUX for External IRQ[10]
12	eMIOS Output Channel 8 eTPU_A Output Channel 8 GPDO394	12	Input 2 on IMUX for External IRQ[11]
13	eMIOS Output Channel 9 eTPU_A Output Channel 9 GPDO395	13	Input 2 on IMUX for External IRQ[12]
14	eMIOS Output Channel 10 eTPU_A Output Channel 10 GPDO396	14	Input 2 on IMUX for External IRQ[13]

Table 635. DSPI_C connectivity table (continued)

DSPI_C input	Connected to:	DSPI_C output	Connected to:
15	eMIOS Output Channel 11 eTPU_A Output Channel 11 GPDO397	15	Input 2 on IMUX for External IRQ[14]
16	eMIOS Output Channel 0 eTPU_A Output Channel 23 GPDO398	16	NC
17	eMIOS Output Channel 1 eTPU_A Output Channel 22 GPDO399	17	NC
18	eMIOS Output Channel 2 eTPU_A Output Channel 21 GPDO400	18	NC
19	eMIOS Output Channel 3 eTPU_A Output Channel 20 GPDO401	19	NC
20	eMIOS Output Channel 4 eTPU_A Output Channel19 GPDO402	20	NC
21	eMIOS Output Channel 5 eTPU_A Output Channel 18 GPDO403	21	NC
22	eMIOS Output Channel 6 eTPU_A Output Channel 17 GPDO404	22	NC
23	eMIOS Output Channel 8 eTPU_A Output Channel 16 GPDO405	23	NC
24	eMIOS Output Channel 9 eTPU_A Output Channel 29 GPDO406	24	NC
25	eMIOS Output Channel 10 eTPU_A Output Channel 28 GPDO407	25	NC
26	eMIOS Output Channel 11 eTPU_A Output Channel 27 GPDO408	26	NC
27	eMIOS Output Channel 12 eTPU_A Output Channel 26 GPDO409	27	NC

Table 635. DSPI_C connectivity table (continued)

DSPI_C input	Connected to:	DSPI_C output	Connected to:
28	eMIOS Output Channel 13 eTPU_A Output Channel 25 GPDO410	28	NC
29	eMIOS Output Channel 14 eTPU_A Output Channel 24 GPDO411	29	NC
30	eMIOS Output Channel 15 eTPU_A Output Channel 31 GPDO412	30	NC
31	eMIOS Output Channel 23 eTPU_A Output Channel 30 GPDO413	31	NC

DSPI_D connectivity

The DSPI_D connects to SIU as shown in *Figure 634*.

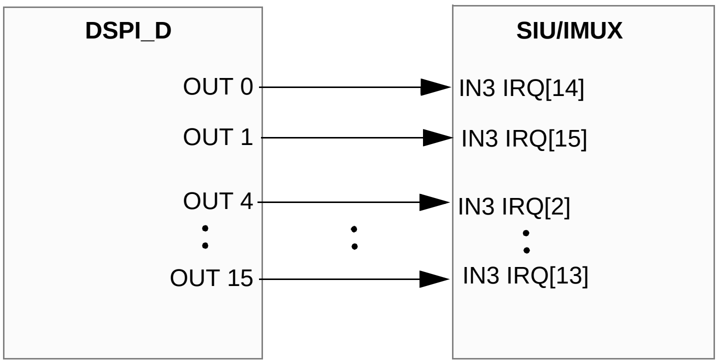


Figure 634. DSPI_D connectivity

Table 636 lists the DSPI_D connections.

Table 636. DSPI_D connectivity table

DSPI_D input	Connected to:	DSPI_D output	Connected to:
0	NC	0	Input 3 on IMUX for External IRQ[14]
1	NC	1	Input 3 on IMUX for External IRQ[15]
2	NC	2	NC
3	NC	3	NC

Table 636. DSPI_D connectivity table

DSPI_D input	Connected to:	DSPI_D output	Connected to:
4	NC	4	Input 3 on IMUX for External IRQ[2]
5	NC	5	Input 3 on IMUX for External IRQ[3]
6	NC	6	Input 3 on IMUX for External IRQ[4]
7	NC	7	Input 3 on IMUX for External IRQ[5]
8	NC	8	Input 3 on IMUX for External IRQ[6]
9	NC	9	Input 3 on IMUX for External IRQ[7]
10	NC	10	Input 3 on IMUX for External IRQ[8]
11	NC	11	Input 3 on IMUX for External IRQ[9]
12	NC	12	Input 3 on IMUX for External IRQ[10]
13	NC	13	Input 3 on IMUX for External IRQ[11]
14	NC	14	Input 3 on IMUX for External IRQ[12]
15	NC	15	Input 3 on IMUX for External IRQ[13]
16–31	NC	16–31	NC

27.9.18 Power saving features

The DSPI supports two power-saving strategies:

- External Stop mode
- Module Disable mode—Clock gating of non-memory mapped logic

Stop mode (External Stop mode)

The DSPI supports the stop mode protocol. When a request is made to enter external stop mode, the DSPI module acknowledges the request. If a serial transfer is in progress, the DSPI waits until it reaches the frame boundary before it is ready to have its clocks shut off. While the clocks are shut off, the DSPI memory-mapped logic is not accessible. The states of the interrupt and DMA request signals cannot be changed while in External Stop mode.

Module disable mode

Module disable mode is a module-specific mode that the DSPI can enter to save power. Host CPU can initiate the module disable mode by setting bit DSPI_MCR[MDIS]. The module disable mode can also be initiated by hardware.

When the MDIS bit is set, the DSPI negates Clock Enable signal at the next frame boundary. If implemented, the Clock Enable signal can stop the clock to the non-memory mapped logic. When Clock Enable is negated, the DSPI is in a dormant state, but the memory mapped registers are still accessible. Certain read or write operations have a different effect when the DSPI is in the module disable mode. Reading the RX FIFO Pop Register does not change the state of the RX FIFO. Likewise, writing to the TX FIFO Push Register does not change the state of the TX FIFO. Clearing either of the FIFOs has no effect in the module disable mode. Changes to the DIS_TXF and DIS_RXF fields of the DSPI_MCR have no effect in the module disable mode. In the module disable mode, all status bits and register flags in the DSPI return the correct values when read, but writing to them has no effect.

Writing to the DSPI_TCR during module disable mode has no effect. Interrupt and DMA request signals cannot be cleared while in the module disable mode.

27.10 Initialization/Application information

27.10.1 How to manage DSPI queues

The queues are not part of the DSPI, but the DSPI includes features in support of queue management. Queues are primarily supported in SPI configuration.

1. When DSPI executes last command word from a queue, the EOQ bit in the command word is set to indicate to the DSPI that this is the last entry in the queue.
2. At the end of the transfer, corresponding to the command word with EOQ set is sampled, the EOQ flag DSPI_SR[EOQF] is set.
3. The setting of the EOQ flag disables serial transmission and reception of data, putting the DSPI in the STOPPED state. The TXRXS bit is cleared to indicate the STOPPED state.
4. The DMA can continue to fill TX FIFO until it is full or step 5 occurs.
5. Disable DSPI DMA transfers by disabling the DMA enable request for the DMA channel assigned to TX FIFO and RX FIFO. This is done by clearing the corresponding DMA enable request bits in the DMA controller.
6. Ensure all received data in RX FIFO has been transferred to memory receive queue by reading DSPI_SR[RXCNT] or by checking DSPI_SR[RFDF] after each read operation of the DSPI_POPR.
7. Modify DMA descriptor of TX and RX channels for new queues
8. Flush TX FIFO by writing a ‘1’ to bit DSPI_MCR[CLR_TXF]. Flush RX FIFO by writing a ‘1’ to bit DSPI_MCR[CLR_RXF].
9. Clear transfer count either by setting CTCNT bit in the command word of the first entry in the new queue or via CPU writing directly to field DSPI_TCR[TCNT].
10. Enable DMA channel by enabling the DMA enable request for the DMA channel assigned to the DSPI TX FIFO, and RX FIFO by setting the corresponding DMA set enable request bit.
11. Enable serial transmission and serial reception of data by clearing the EOQF bit.

27.10.2 Switching master and slave mode

When changing modes in the DSPI, follow the steps below to guarantee proper operation.

1. Halt the DSPI by setting DSPI_MCR[HALT].
2. Clear the transmit and receive FIFOs by writing a 1 to the CLR_TXF and CLR_RXF bits in DSPI_MCR.
3. Set the appropriate mode in DSPI_MCR[MSTR] and enable the DSPI by clearing DSPI_MCR[HALT].

27.10.3 Baud rate settings

Table 637 shows the baud rate that is generated based on the combination of the baud rate prescaler PBR and the baud rate scaler BR in the DSPI_CTAR registers. The values calculated assume a 100 MHz system frequency and the double baud rate DBR bit is clear.

Table 637. Baud rate values (bit/s)

		Baud rate divider prescaler values			
		2	**3**	**5**	**7**
Baud rate scaler values	**2**	25.0M	16.7M	10.0M	7.14M
	4	12.5M	8.33M	5.00M	3.57M
	6	8.33M	5.56M	3.33M	2.38M
	8	6.25M	4.17M	2.50M	1.79M
	16	3.12M	2.08M	1.25M	893k
	32	1.56M	1.04M	625k	446k
	64	781k	521k	312k	223k
	128	391k	260k	156k	112k
	256	195k	130k	78.1k	55.8k
	512	97.7k	65.1k	39.1k	27.9k
	1024	48.8k	32.6k	19.5k	14.0k
	2048	24.4k	16.3k	9.77k	6.98k
	4096	12.2k	8.14k	4.88k	3.49k
	8192	6.10k	4.07k	2.44k	1.74k
	16384	3.05k	2.04k	1.22k	872
	32768	1.53k	1.02k	610	436

27.10.4 Delay settings

Table 638 shows the values for the Delay after Transfer (t_{DT}) and CS to SCK Delay (T_{CSC}) that can be generated based on the prescaler values and the scaler values set in the DSPI_CTAR registers. The values calculated assume a 100 MHz system frequency.

Table 638 does not apply for TSB Continuous mode.

Table 638. Delay values

		Delay prescaler values			
		1	**3**	**5**	**7**
Delay scaler values	**2**	20.0 ns	60.0 ns	100.0 ns	140.0 ns
	4	40.0 ns	120.0 ns	200.0 ns	280.0 ns
	8	80.0 ns	240.0 ns	400.0 ns	560.0 ns
	16	160.0 ns	480.0 ns	800.0 ns	1.1 µs
	32	320.0 ns	960.0 ns	1.6 µs	2.2 µs
	64	640.0 ns	1.9 µs	3.2 µs	4.5 µs
	128	1.3 µs	3.8 µs	6.4 µs	9.0 µs
	256	2.6 µs	7.7 µs	12.8 µs	17.9 µs
	512	5.1 µs	15.4 µs	25.6 µs	35.8 µs
	1024	10.2 µs	30.7 µs	51.2 µs	71.7 µs
	2048	20.5 µs	61.4 µs	102.4 µs	143.4 µs
	4096	41.0 µs	122.9 µs	204.8 µs	286.7 µs
	8192	81.9 µs	245.8 µs	409.6 µs	573.4 µs
	16384	163.8 µs	491.5 µs	819.2 µs	1.1 ms
	32768	327.7 µs	983.0 µs	1.6 ms	2.3 ms
	65536	655.4 µs	2.0 ms	3.3 ms	4.6 ms

27.10.5 Calculation of FIFO pointer addresses

Complete visibility of the TX and RX FIFO contents is available through the FIFO registers, and valid entries can be identified through a memory mapped pointer and a memory mapped counter for each FIFO. The pointer to the first-in entry in each FIFO is memory mapped. For the TX FIFO the first-in pointer is the Transmit Next Pointer (TXNXTPTR). For the RX FIFO the first-in pointer is the Pop Next Pointer (POPNXTPTR). *Figure 635* illustrates the concept of first-in and last-in FIFO entries along with the FIFO Counter. The TX FIFO is chosen for the illustration, but the concepts carry over to the RX FIFO. See *Section , Transmit first-in first-out (TX FIFO) buffering mechanism* and *Section , Receive first-in first-out (RX FIFO) buffering mechanism* for details on the FIFO operation.

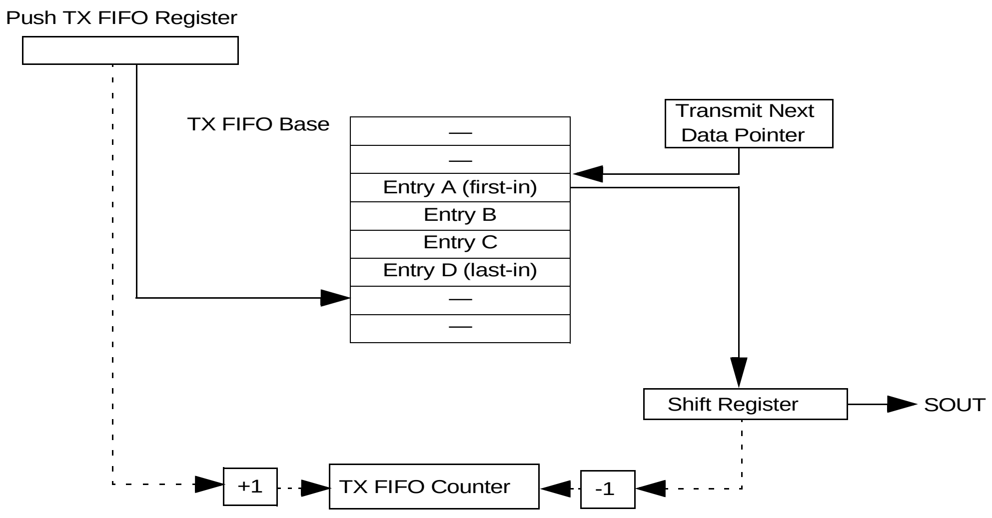


Figure 635. TX FIFO pointers and counter

Address calculation for the first-in entry and last-in entry in the TX FIFO

The memory address of the first-in entry in the TX FIFO is computed by the following equation:

Equation 29

$$\text{First-in Entry Address} = \text{TX FIFO Base} + (4 \times \text{TXNXTPTR})$$

The memory address of the last-in entry in the TX FIFO is computed by the following equation:

Equation 30

$$\text{Last-in Entry address} = \text{TX FIFO Base} + 4 \times (\text{TXCTR} + \text{TXNXTPTR} - 1)\text{mod}(\text{TXFIFOdepth})$$

TX FIFO Base: Base address of TX FIFO

TXCTR: TX FIFO Counter

TXNXTPTR: Transmit Next Pointer

TX FIFO Depth: Transmit FIFO depth, implementation-specific

Address calculation for the first-in entry and last-in entry in the RX FIFO

The memory address of the first-in entry in the RX FIFO is computed by the following equation:

Equation 31

$$\text{First-in Entry Address} = \text{RX FIFO Base} + (4 \times \text{POPNXTPTR})$$

The memory address of the last-in entry in the RX FIFO is computed by the following equation:

Equation 32

$$\text{Last-in Entry address} = \text{RX FIFO Base} + 4 \times (\text{RXCTR} + \text{POPNXTPTR} - 1)\text{mod (RXFIFOdepth)}$$

RX FIFO Base: Base address of RX FIFO
RXCTR: RX FIFO counter
POPNXTPTR: Pop Next Pointer
RX FIFO Depth: Receive FIFO depth, implementation specific

28 Enhanced Serial Communication Interface (eSCI)

28.1 Introduction

The eSCI block is an enhanced SCI block with a LIN master interface layer and DMA support. The LIN master layer complies with the specifications LIN 1.3, LIN 2.0, LIN 2.1, and SAE J2602/1.

28.1.1 Bibliography

- LIN Specification Package Revision 1.3; December 12, 2002
- LIN Specification Package Revision 2.0; September 23, 2003
- LIN Network for Vehicle Applications, SAE J2602/1, September 1, 2005
- LIN Specification Package Revision 2.1; November 24, 2006

28.1.2 Acronyms and abbreviations

Table 639 contains acronyms and abbreviations used in this document.

Table 639. Acronyms and abbreviations

Term	Description
eSCI	Enhanced SCI block with LIN support and DMA support
SCI	Serial Communications Interface
LIN	Local Interconnect Network - A protocol for low-cost automobile networks
LIN PE	LIN Protocol Engine, Finite State Machine to control logic of the LIN hardware.
MCLK	Module Clock, defined in *Section , Module clock*
TCLK	Transmitter Clock, defined in *Section , Transmitter clock*
RCLK	Receiver Clock, defined in *Section , Receiver clock*
RSC	Receiver Sample Counter, defined in *Section , Receiver clock*

28.1.3 Glossary

Table 640. Glossary

Term	Definition
Logic level one	The voltage that corresponds to Boolean true (1) state.
Logic level zero	The voltage that corresponds to Boolean false (0) state.
Set	To set a bit or bits means to establish logic level one on the bit or bits.
Clear	To clear a bit or bits means to establish logic level zero on the bit or bits.
Asserted	A signal that is asserted is in its active state. An active low signal changes from logic level one to logic level zero when asserted, and an active high signal changes from logic level zero to logic level one.

Table 640. Glossary (continued)

Term	Definition
Preamble	The term preamble describes an idle character which is *transmitted* by the eSCI module.
Bit time	Duration of a single bit in a transmitted byte field or character, equivalent to the duration of one transmitter clock cycle defined in *Section , Transmitter clock*
frame	Entity that consists of the start bit followed by payload bits followed by one ore more stop bits
LIN byte field	Special instance of a frame
SCI frame	Special instance of a frame
LIN frame	Sequence of break character followed by LIN byte fields
LIN TX frame	A LIN frame with the frame header, data byte fields, and checksum field transmitted by the eSCI module
LIN RX frame	A LIN frame with the header field transmitted by the eSCI module and the data byte fields and checksum field received by the eSCI module
module is idle	Module is idle, described in *Section , Module idle condition*

28.1.4 Overview

The eSCI block allows asynchronous serial communications with peripheral devices and other CPUs. It includes special support to interface to LIN slave devices.

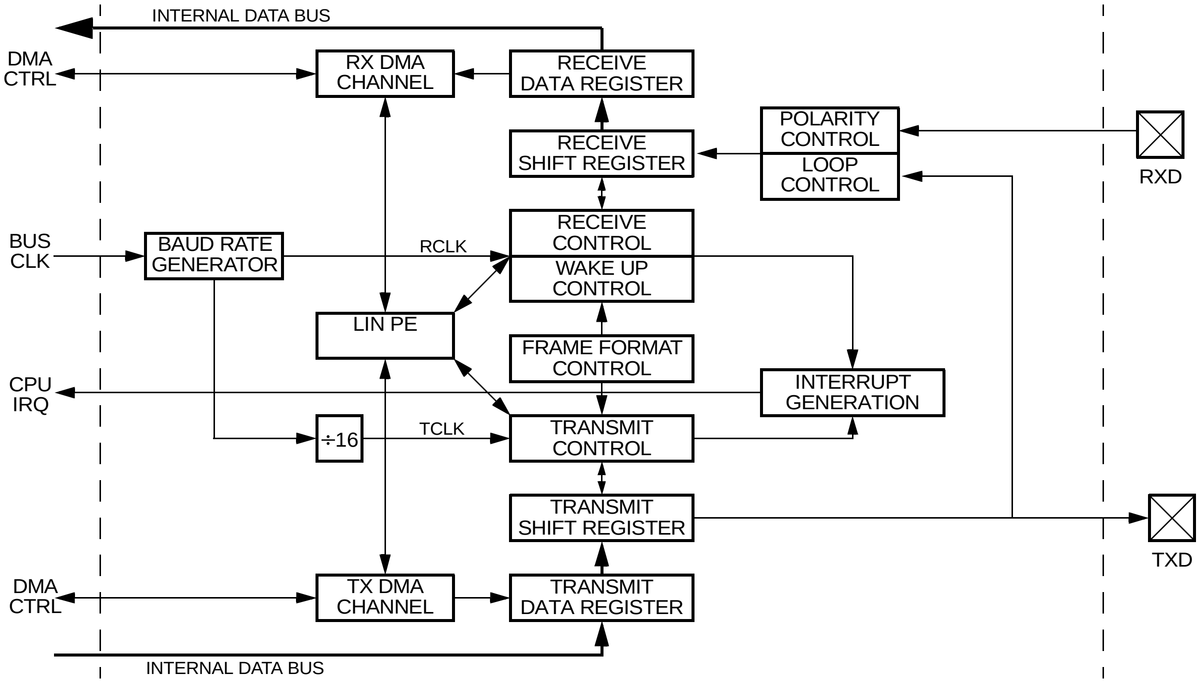


Figure 636. eSCI Block Diagram

28.1.5 Features

The eSCI block includes these distinctive features:

- Full-duplex operation
- Standard mark/space non-return-to-zero (NRZ) format
- 13-bit baud rate selection
- Programmable frame, payload, and character format
- Support of 2 stop bits in receiver path
- Hardware parity generation and checking
  - Programmable even or odd parity
- Programmable polarity of RXD pin
- Separately enabled transmitter and receiver
- Two receiver wake-up methods:
  - Idle line wake up
  - Address mark wake up
- Interrupt-driven operation with eight flags:
  - Transmitter empty
  - Transmission complete
  - Receiver full
  - Idle receiver input
  - Receiver overrun
  - Noise error
  - Framing error
  - Parity error
- Receiver framing error detection
- 1/16 bit-time noise detection
- 2 channel DMA interface
- LIN support
  - LIN Master Node functionality (master and slave task)
  - Compatible with LIN slaves from revisions 1.x and 2.0 of the LIN standard
  - Detection of Bit Errors, Physical Bus Errors and Checksum Errors
  - All status bit can generate maskable interrupts
  - Application layer CRC support
  - Programmable CRC polynom
  - Detection and generation of wake-up characters
  - Programmable wake-up delimiter time
  - Programmable slave timeout
  - Can be configured to include header bits in checksum
  - LIN DMA interface

28.1.6 Modes of operation

The eSCI module has two functional operational modes, SCI and LIN mode, and low power modes. The availability of register bits and fields depends on the selected operational mode.

Module idle condition

Some modes can only be entered if the module is idle. The module is idle if

- all five active status bits TACT, and RAF in the *Status register (eSCI_SR)* are 0, and
- no interrupt request is pending, i.e either the interrupt flag or its related interrupt enable is 0.

To ensure that the module goes idle, the application should clear all interrupt enable bits before triggering the mode change.

SCI mode

The SCI mode is the default functional operational mode and is described in *Section 28.4.5, SCI mode*.

LIN mode

The LIN mode is the second functional operational mode and is described in *Section 28.4.6, LIN mode*.

Disabled mode

In the Disabled mode the eSCI module indicates to the clocking system, that all module clocks can be turned off.

The eSCI module is in the Disabled Mode, if the MDIS bit in the *Control register 2 (eSCI_CR2)* is set and the module is idle.

28.2 External signal description

The eSCI module is connected two a total of two external pins.

28.2.1 Detailed signal descriptions

eSCI transmit pin (TXD)

This pin serves as transmit data output and as the receive data input of eSCI.

eSCI receive pin (RXD)

This pin serves as receive data input of the eSCI.

28.3 Memory map and register definition

This section provides the memory map and a detailed description of the memory mapped registers.

28.3.1 Memory map

Table 641. Block memory map

Offset	Register	Location
0x00	*Control register 1 (eSCI_CR1)*	*on page 28-1179*
0x04	*Control register 2 (eSCI_CR2)*	*on page 28-1182*
0x06	*SCI data register (eSCI_DR)*	*on page 28-1184*
0x08	*Status register (eSCI_SR)*	*on page 28-1185*
0x0C	*LIN control register (eSCI_LCR)*	*on page 28-1187*
0x10	*LIN transmit register (eSCI_LTR)*	*on page 28-1189*
0x14	*LIN receive register (eSCI_LRR)*	*on page 28-1191*
0x18	*LIN CRC polynomial register (eSCI_LPR)*	*on page 28-1191*
0x1A	*Control register 3 (eSCI_CR3)*	*on page 28-1192*

Table 642 provides a key for register figures and tables.

Table 642. Register Conventions

Convention	Description
▨	Depending on its placement in the read or write row, indicates that the bit is not readable or not writeable.
FIELDNAME	Identifies the field. Its presence in the read or write row indicates that it can be read or written.
Register Field Types	
rwm	A read/write bit that may be modified by a eSCI module in some fashion other than by a reset.
w1c	Write one to clear. A flag bit that can be read, is cleared by writing a one, writing 0 has no effect.
Reset Values	
0	Resets to zero.
1	Resets to one.

28.3.2 Register descriptions

This section consists of register descriptions in address order. Each description includes a standard register diagram with an associated figure number. Writes to a reserved register location do not have any effect and reads of these locations return a zero. Details of register bit and field function follow the register diagrams, in bit order.

Control register 1 (eSCI_CR1)

This register provides: the control value for the serial baud rate, bits to configure the functionality of the module, the interrupt enable bits for the interrupt flags provided in *Status register (eSCI_SR)* and provides the control bits for the transmitter and receiver.

The baud rate and clock generation is specified in *Section 28.4.3, Baud rate and clock generation*.

A byte write access to only the upper byte of the high half work of this register (eSCI_CR1[0:7]) will not change the content of the register, instead, the written byte is stored internally into a shadow register. A subsequent byte write access to only the lower byte of the high half work of this register (eSCI_CR1[8:15]) updates the lower byte of the high half work and copies the content of the shadow register into the upper byte.

A byte write access to only the lower byte of the high half work of this register (eSCI_CR1[8:15]) without a preceding byte write access to only the upper byte copies a value of all zero into the upper byte.

A word write access to this register updates both the lower and upper byte immediately and is the recommended write access type for this register.

Figure 637. Control register 1 (eSCI_CR1)

eSCI_BASE + 0x0000 Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	SBR												
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	LOOPS	0	RSRC	M	WAKE	ILT	PE	PT	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK
W															rwm	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 643. eSCI_CR1 field descriptions

Field	Description
SBR	Serial Baud Rate. This field provides the baud rate control value SBR.
LOOPS	Loop Mode Select. This control bit together with the RSRC control bit defines the receiver source mode. The mode coding is defined in *Table 644* and the modes are described in *Section , Receiver input mode selection*.
RSRC	Receiver Source Control. This control bit together with the LOOPS control bit defines the receiver source mode. The mode coding is defined in *Table 644* and the modes are described in *Section , Receiver input mode selection*.
M	Frame Format Mode. This control bit together with the M2 bit of the *Control register 3 (eSCI_CR3)* controls the frame format used. The supported frame formats and the related settings are defines in *Section 28.4.2, Frame formats*.
WAKE	Receiver Wake-up Condition. This control bit defines the wake-up condition for the receiver. The receiver wake-up is described in *Section , Multiprocessor communication*. 0 Idle line wake-up. 1 Address mark wake-up
ILT	Idle Line Type. This control bit defines the type of idle line detection for the receiver wake-up. The two types are described in *Section , Idle-Line wake up*. 0 Idle line detection starts after reception of a low bit. 1 Idle line detection starts after reception of the last stop bit.

Table 643. eSCI_CR1 field descriptions (continued) (continued)

Field	Description
PE	Parity Enable. This control bit enables the parity bit generation and checking. The location of the parity bits is shown in *Section 28.4.2, Frame formats*. 0 Parity bit generation and checking disabled. 1 Parity bit generation and checking enabled.
PT	Parity Type. This control bit defines whether even or odd parity has to be used. 0 Even parity (even number of ones in character clears the parity bit). 1 Odd parity (odd number of ones in character clears the parity bit).
TIE	Transmitter Interrupt Enable. This bit controls the eSCI_SR[TRDE] interrupt request generation. 0 TDRE interrupt request generation disabled. 1 TDRE interrupt request generation enabled.
TCIE	Transmission Complete Interrupt Enable. This bit controls the eSCI_SR[TC] interrupt request generation. 0 TC interrupt request generation disabled. 1 TC interrupt request generation enabled.
RIE	Receiver Full Interrupt Enable. This bit controls the eSCI_SR[RDRF] interrupt request generation. 0 RDRF interrupt request generation disabled. 1 RDRF interrupt request generation enabled.
ILIE	Idle Line Interrupt Enable. This bit controls theeSCI_SR[IDLE] interrupt request generation. 0 IDLE interrupt request generation disabled. 1 IDLE interrupt request generation enabled.
TE	Transmitter Enable. This control bit enables and disables the transmitter. The control features of the transmitter are described in *Section , Transmitter states and transitions*. 0 Transmitter disabled. 1 Transmitter enabled.
RE	Receiver Enable.This control bit enables and disables the receiver. The control features of the receiver are described in *Section , Receiver states and transitions*. 0 Receiver disabled. 1 Receiver enabled.
RWU	Receiver Wake-Up Mode. This bit controls and indicates the receiver wake-up mode, which is described in *Section , Multiprocessor communication*. 0 Normal receiver operation. 1 Receiver is in wake-up mode. This bit should be set in SCI mode only.
SBK	Send Break Character. This bit controls the transmission of break characters, which is described in *Section , Break character transmission*. 0 No break characters will be transmitted. 1 Break characters will be transmitted. This bit should be set in SCI mode only.

Table 644. Receive source mode selection

LOOPS	RSCR	Receiver Input Mode
0	0	Dual Wire Mode
0	1	Reserved
1	0	Loop Mode
1	1	Single Wire Mode

Control register 2 (eSCI_CR2)

Figure 638. Control register 2 (eSCI_CR2)

eSCI_BASE + 0x0004 Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MDIS	FBR	BSTP	IEBERR	RXDMA	TXDMA	BRK13	TXDIR	BESM13	SBSTP	RXPOL	PMSK	ORIE	NFIE	FEIE	PFIE
Reset	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0

This register provides bits to configure the functionality of the module, and interrupt enable bits for the interrupt flags provided in *Status register (eSCI_SR)* and control bits for the transmitter and receiver.

Table 645. eSCI_CR2 field descriptions

Field	Description
MDIS	Module Disabled Mode. This bit controls the Module Mode of Operation, which is described in *Section 28.1.6, Modes of operation*. 0 Module is not in Disabled Mode. 1 Module is in Disabled Mode, if module is idle.
FBR	Fast Bit Error Detection. This bit controls the Bit Error Detection mode. 0 Standard Bit error detection performed as described in *Section , Standard bit error detection*. 1 Fast Bit error detection performed as described in *Section , Fast bit error detection*. This bit is used in LIN mode only.
BSTP	DMA Stop on Bit Error or Physical Bus Error. This bit controls the transmit DMA requests generation in case of bit errors or physical bus errors. Bit errors are indicated by the BERR flag in the *Status register (eSCI_SR)* and physical bus errors are indicated by the PBERR flag in the *Status register (eSCI_SR)*. 0 Transmit DMA requests generated regardless of bit errors or physical bus errors. 1 Transmit DMA requests are *not* generated if eSCI_SR[BERR] flag or eSCI_SR[PBERR] flags are set. This bit is used in LIN mode only.
IBERRIE	Bit Error Interrupt Enable. This bit controls the BERR interrupt request generation. 0 BERR interrupt request generation disabled. 1 BERR interrupt request generation enabled.

Table 645. eSCI_CR2 field descriptions (continued)

Field	Description
RXDMA	Receive DMA Control. This bit enables the receive DMA feature. When this bit is cleared, a pending receive DMA request is deasserted. 0 Receive DMA disabled. 1 Receive DMA enabled.
TXDMA	Transmit DMA Control. This bit enables the transmit DMA feature. When this bit is cleared, a pending transmit DMA request is deasserted. 0 Transmit DMA disabled. 1 Transmit DMA enabled.
BRK13	Break Character Length. This bit is used to define the length of the break character to be transmitted. The settings are specified in *Section , Break character formats*.
TXDIR	TXD pin output driver control. This bit controls the output driver of the TXD pin in Single Wire Mode. 0 TXD pin output driver disabled. 1 TXD pin output driver enabled.
BESM13	Fast Bit Error Detection Sample Mode. This bit defines the sample point for the Fast Bit Error Detection Mode. 0 Sample point is RS9. 1 Sample point is RS13. **Note:** This bit is used in LIN mode only.
SBSTP	Bit Error Transmit Stop. This control bit defines the behavior of the eSCI Transmit Pin TXD while the bit error flag eSCI_SR[BERR] is 1. 0 Application Data Values driven onto TXD pin. 1 Recessive Data Value 1 driven onto TXD pin. **Note:** This bit is used in LIN mode only.
RXPOL	RXD Pin polarity. This bit controls the polarity of the RXD pin. See *Section , inverted data frame formats*. 0 Normal Polarity. 1 Inverted Polarity.
PMSK	Parity Bit Masking. This bit defines whether the received parity bit is presented in the related bit position in the *SCI data register (eSCI_DR)*. 0 The received parity bit is presented in the bit position related to the parity bit. 1 The value 0 is presented in the bit position related to the parity bit.
ORIE	Overrun Interrupt Enable. This bit controls the eSCI_SR[OR] interrupt request generation. 0 OR interrupt request generation disabled. 1 OR interrupt request generation enabled.
NFIE	Noise Interrupt Enable. This bit controls the eSCI_SR[NF] interrupt request generation. 0 NF interrupt request generation disabled. 1 NF interrupt request generation enabled.
FEIE	Frame Error Interrupt Enable. This bit controls the eSCI_SR[FE] interrupt request generation. 0 FE interrupt request generation disabled. 1 FE interrupt request generation enabled.
PFIE	Parity Error Interrupt Enable. This bit controls the eSCI_SR[PF] interrupt request generation. 0 PF interrupt request generation disabled. 1 PF interrupt request generation enabled.

SCI data register (eSCI_DR)

Figure 639. SCI data register (eSCI_DR)

eSCI_BASE + 0x0006 Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	R8	T8	ERR	0	R				D							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

In SCI mode this register is used to provide transmit data and retrieve received data. In LIN mode any write access to this register is ignored and any read access returns unspecified data. In case of data transmission this register is used to provide a part of the transmit data. In case of data reception this register provides a part of the received data and related error information.If the application writes to the lower byte of this register (eSCI_DR[8:15]), the internal commit flag iCMT, which is not visible to the application, is set to indicate that the register has been updated and ready to transmit new data.

If the application reads from the lower byte of this register (eSCI_DR[8:15]), a signal is send to the internal receiver unit to indicate that the register was read and is ready to receive new data. The read access will not change the content of any register.

Table 646. eSCI_DR field descriptions

Field	Description
R8	Received Most Significant Bit. The semantic of this bit depends on the frame format selected by eSCI_CR3[M2], eSCI_CR1[M], and eSCI_CR1[PE]. [M2=0,M=1,PE=0]: value of received data bit 8 or address bit. [M2=0,M=1,PE=1]: value of received parity bit if eSCI_CR2[PMSK]=0, 0 otherwise. [M2=1,M=0,PE=1]: value of received parity bit if eSCI_CR2[PMSK]=0, 0 otherwise. [M2=1,M=1,PE=1]: value of received parity bit if eSCI_CR2[PMSK]=0, 0 otherwise. It is 0 for all other frame formats.
T8	Transmit Most Significant Bit. The semantic of this bit depends on the frame format selected by eSCI_CR3[M2], eSCI_CR1[M], and eSCI_CR1[PE]. [M2=0,M=1,PE=0]: value to be transmitted as data bit 8 or address bit. It is not used for all other frame formats.
ERR	Receive Error Bit. This bit indicates the occurrence of the errors selected by the *Control register 3 (eSCI_CR3)* during the reception of the frame presented in *SCI data register (eSCI_DR)*. In case of an overrun error for subsequent frames this bit is set too. 0 None of the selected errors occured. 1 At least one of the selected errors occured.
R	Received Data Bits[11:8] The semantic of this field depends on the frame format selected by SCI_LPR[M2] and SCI_CR1[M]. [M2 = 1, M = 1]: value of the received data bits 11:8. (Rx = BITx). It is all 0 for all other frame formats.
D	Received or transmitted data

Status register (eSCI_SR)

This register provides interrupt flags that indicate the occurrence of module and LIN events. The related interrupt enable bits are located in *Control register 1 (eSCI_CR1)*, *Control register 2 (eSCI_CR2)* and *LIN control register (eSCI_LCR)*.

All interrupt flags in this register will be set in LIN mode only.

Figure 640. Status register (eSCI_SR)

eSCI_BASE + 0x0008

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TDRE	TC	RDRF	IDLE	OR	NF	FE	PF	0	0	0	BERR	0	0	TACT	RAF
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	▨	▨	▨	w1c	▨	▨	▨	▨
Reset	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	RXRDY	TXRDY	LWAKE	STO	PBERR	CERR	CKERR	FRC	0	0	0	0	0	0	UREQ	OVFL
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	▨	▨	▨	▨	▨	▨	w1c	w1c
Reset	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented

Table 647. eSCI_SR field descriptions

Field	Description
TDRE	Transmit data register Empty Interrupt Flag. This interrupt flag is set when the content of the *SCI data register (eSCI_DR)* was transferred into internal shift register. This flag is set in SCI mode only.
TC	Transmit Complete Interrupt Flag. This interrupt flag is set when a frame, break or idle character transmission has been completed and no data were written into *SCI data register (eSCI_DR)* after the last setting of the TDRE flag and the SBK bit in *Control register 1 (eSCI_CR1)* is 0. This flag is set in LIN mode, if the preamble was transmitted after the enabling of the transmitter.
RDRF	Receive data register Full Interrupt Flag. This interrupt flag is set when the payload data of a received frame was transferred into the *SCI data register (eSCI_DR)* and the receive DMA is disabled. This flag is set in SCI mode only.
IDLE	Idle Line Interrupt Flag. This interrupt flag is set when an idle character was detected and the receiver is not in the wake-up state. This flag is set in SCI mode only.
OR	Overrun Interrupt Flag. This interrupt flag is set when an overrun was detected as described in *Section , Receiver overrun*. This flag is set in SCI mode only.

Table 647. eSCI_SR field descriptions (continued)

Field	Description
NF	Noise Interrupt Flag. This interrupt flag is set when the receiver has detected noise during the reception of a frame, as described in *Section , Bit sampling*.
FE	Framing Error Interrupt Flag. This interrupt flag is set when the payload data of a received frame was transferred into the *SCI data register (eSCI_DR)* or *LIN receive register (eSCI_LRR)* and the receiver has detected a framing error during the reception of that frame, as described in *Section , Stop bit verification*.
PF	Parity Error Interrupt Flag. This interrupt flag is set when the payload data of a received frame was transferred into the *SCI data register (eSCI_DR)* and the receiver has detected a parity error for the character, as described in *Section , Reception error reporting*. This flag is set in SCI mode only.
BERR	Bit Error Interrupt Flag. This flag is set when a bit error was detected as described in *Section , Standard bit error detection*. **Note:** This flag is set in LIN mode only.
TACT	Transmitter Active. This status bit is set as long as the transmission of a frame or special character is ongoing. 0 No transmission in progress. 1 Transmission in progress.
RAF	Receiver Active. This status bit is set as long as the receive is active. The set and clear conditions for the SCI mode are described in *Section , Receiver states and transitions*.The set and clear conditions for the LIN mode are described in *Section , LIN byte field reception*. 0 No reception in progress. 1 Reception in progress.
RXRDY	Receive Data Ready Interrupt Flag. This interrupt flag is set when the payload data of a received frame was transferred into the *LIN receive register (eSCI_LRR)* and the receive DMA is disabled.
TXRDY	Transmit Data Ready Interrupt Flag. This interrupt flag is set when a) the content of the *LIN transmit register (eSCI_LTR)* was processed by the LIN PE to generate frame header or frame transmit data, or b) when the module has transmitted a LIN wakeup signal frame.
LWAKE	LIN Wake-up Received Interrupt Flag. This interrupt flag is set when a LIN Wake-up character was received, as described in *Section , LIN wake up*.
STO	Slave Timeout Interrupt Flag. This interrupt flag is set when a Slave-Not-Responding-Error is detected. A detailed description is given in *Section , Slave-Not-Responding-Error detection*.
PBERR	Physical Bus Error Interrupt Flag. This interrupt flag is set when the receiver input remains unchanged for at least 31 RCLK clock cycles after the start of a byte transmission, as described in *Section , LIN error reporting*.
CERR	CRC Error Interrupt Flag. This interrupt flag is set when an incorrect CRC pattern was detected for a received LIN frame.
CKERR	Checksum Error Interrupt Flag. This interrupt flag is set when a checksum error was detected for a received LIN frame.

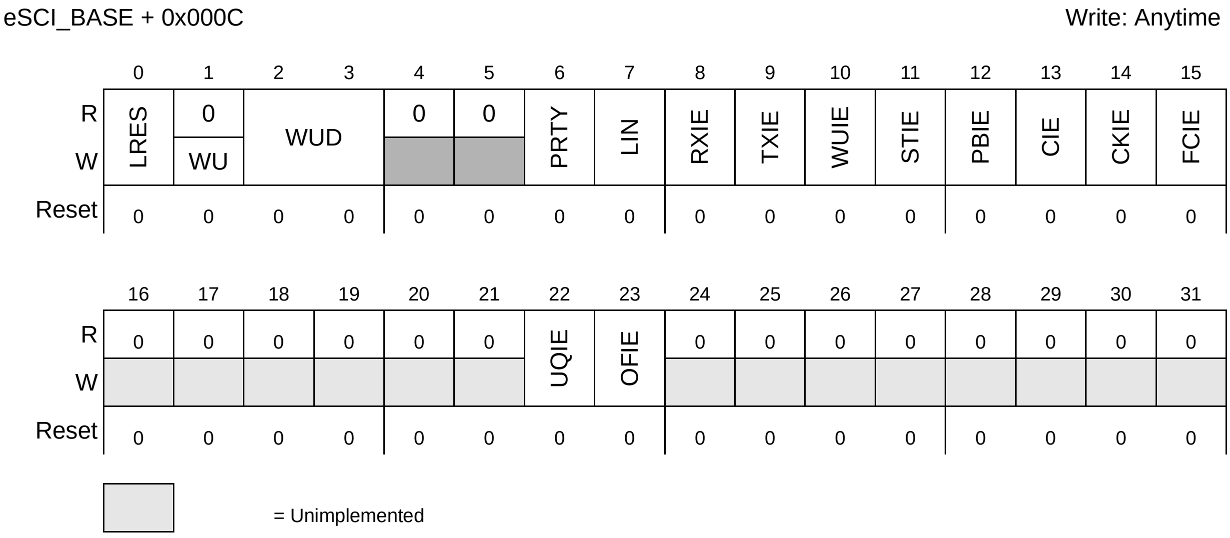
Table 647. eSCI_SR field descriptions (continued)

Field	Description
UREQ	Unrequested Data Received Interrupt Flag. This interrupt flag is set when unrequested activity has been detected on the LIN bus, as described in *Section , LIN error reporting*.
OVFL	Overflow Interrupt Flag. This interrupt flag is set when an overflow as described in *Section , Overflow detection* was detected.

LIN control register (eSCI_LCR)

This register provides control bits to control and configure the LIN hardware and the interrupt enable bits for the interrupt flags in *Status register (eSCI_SR)*.

Figure 641. LIN control register (SCI_LCR)



eSCI_BASE + 0x000C

Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	LRES	0	WUD		0	0	PRTY	LIN	RXIE	TXIE	WUIE	STIE	PBIE	CIE	CKIE	FCIE
W		WU			▨	▨										
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	UQIE	OFIE	0	0	0	0	0	0	0	0
W	▨	▨	▨	▨	▨	▨			▨	▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented

Table 648. eSCI_LCR field descriptions

Field	Description
LRES	LIN Protocol Engine Stop and Reset. This bit is used to stop and reset the LIN protocol engine as described in *Section , LIN protocol engine stop and reset*. 0 LIN protocol engine is operational. 1 LIN protocol engine is reset and stopped.
WU	LIN Bus Wake-Up Trigger. This bit is used to trigger the generation of a wake-up signal frame on the LIN bus, as described in *Section , LIN wake up*. 0 Write has no effect. 1 Write triggers the generation of a wake-up signal.
WUD	LIN Bus Wake-Up Delimiter Time. This field determines how long the LIN protocol engine waits after the end of the transmitted wake-up signal, before starting the next LIN frame transmission. 00 4 bit times. 01 8 bit times. 10 32 bit times. 11 64 bit times.

Table 648. eSCI_LCR field descriptions (continued)

Field	Description
PRTY	Parity Generation Control. This bit controls the generation of the two parity bits in the LIN header. 0 Parity bits generation disabled. 1 Parity bits generation enabled.
LIN	LIN Mode Control. This bit controls whether the device is in SCI or LIN Mode. 0 SCI Mode. 1 LIN Mode.
RXIE	Receive Data Ready Interrupt Enable. This bit controls the eSCI_SR[RXRDY] interrupt request generation. 0 RXRDY interrupt request generation disabled. 1 RXRDY interrupt request generation enabled.
TXIE	Transmit Data Ready Interrupt Enable. This bit controls the eSCI_SR[TXRDY] interrupt request generation. 0 TXRDY interrupt request generation disabled. 1 TXRDY interrupt request generation enabled.
WUIE	LIN Wake-up Received Interrupt Enable. This bit controls the eSCI_SR[LWAKE] interrupt request generation. 0 LWAKE interrupt request generation disabled. 1 LWAKE interrupt request generation enabled.
STIE	Slave Timeout Flag Interrupt Enable. This bit controls the eSCI_SR[STO] interrupt request generation. 0 STO interrupt request generation disabled. 1 STO interrupt request generation enabled.
PBIE	Physical Bus Error Interrupt Enable. This bit controls the eSCI_SR[PBERR] interrupt request generation. 0 PBERR interrupt request generation disabled. 1 PBERR interrupt request generation enabled.
CIE	CRC Error Interrupt Enable. This bit controls the eSCI_SR[CERR] interrupt request generation. 0 CERR interrupt request generation disabled. 1 CERR interrupt request generation enabled.
CKIE	Checksum Error Interrupt Enable. This bit controls the eSCI_SR[CKERR] interrupt request generation. 0 CKERR interrupt request generation disabled. 1 CKERR interrupt request generation enabled.
FCIE	Frame Complete Interrupt Enable. This bit controls the eSCI_SR[FRC] interrupt request generation. 0 FRC interrupt request generation disabled. 1 FRC interrupt request generation enabled.

Table 648. eSCI_LCR field descriptions (continued)

Field	Description
UQIE	Unrequested Data Received Interrupt Enable This bit controls the SCI_SR[UREQ] interrupt request generation. 0 UREQ interrupt request generation disabled 1 UREQ interrupt request generation enabled
OFIE	Overflow Interrupt Enable This bit controls the SCI_SR[OVFL] interrupt request generation. 0 OVFL interrupt request generation disabled 1 OVFL interrupt request generation enabled

LIN transmit register (eSCI_LTR)

Figure 642. LIN transmit register (eSCI_LTR) - LIN TX frame generation

eSCI_BASE + 0x0010 Write: LIN Mode

	0	1	2	3	4	5	6	7
R	DATA							
1st W	P		ID					
2nd W	LEN							
3rd W	CSM	CSE	CRC	TD (=1)	TO (ignored)			
4th+ W	D							
Reset	0	0	0	0	0	0	0	0

Figure 643. LIN transmit register (eSCI_LTR) - LIN RX frame generation

	0	1	2	3	4	5	6	7
R	DATA							
1st W	P[1:0]		ID[5:0]					
2nd W	LEN							
3rd W	CSM	CSE	CRC	TD (=0)	TO[11:8]			
4th W	TO[7:0]							
Reset	0	0	0	0	0	0	0	0

This register is used by the application to initiate the LIN frame header generation for both LIN TX frames and LIN RX frames. If a LIN TX frame is generated, this register is used to provide the payload data for the LIN TX frame.

If the LIN PE is in the idle state (eSCI_LCR[LRES] = 1) or performs a wakeup, each write access to this register is ignored.

In case of an read access, the register provides the last data written into this register in the DATA field.

If the application initiates a LIN TX frame transfer, i.e the TD bit is set to 1, the content and usage shown in *LIN transmit register (eSCI_LTR) - LIN TX frame generation* applies. The initiation and transmit of a TX frame is described in *Section , LIN TX frame generation*.

If the application initiates an LIN RX frame, i.e the TD bit is set to 0, the content and usage shown in *LIN transmit register (eSCI_LTR) - LIN RX frame generation* applies. The initiation and transmit of a RX frame is described in *Section , LIN RX frame generation*.

Each successful write access to this register increments the internal write access counter and enables the writing to the next field. The write access counter is reset if

- the LIN PE is in the idle state (eSCI_LCR[LRES] = 1)
- a LIN TX frame was completely transmitted (eSCI_SR[FRC] was set to 1)
- a LIN RX frame was completely received (eSCI_SR[FRC] was set to 1)

Table 649. eSCI_LTR field descriptions

Field	Description
DATA	Value written in the most recent successful write access.
P	Identifier Parity. This field provides the identifier parity which is used to create the protected identifier if the automatic identifier parity generation is disabled, i.e the PRTY bit in *LIN control register (eSCI_LCR)* is 0.
ID	Identifier. This field is used for the identifier field in the protected identifier.
LEN	Frame Length. This field defines the number of data bytes to be transmitted or received.
CSM	Checksum Model. This bit controls the checksum calculation model used. 0 Classic Checksum Model (LIN 1.3). 1 Enhanced Checksum Model (LIN 2.0).
CSE	Checksum Enable. This bit control the generation and checking of the checksum byte. 0 No generation and checking of checksum byte. 1 Generation and checking of checksum byte.
CRC	CRC Enable. This bit controls the generation of checking standard or enhanced LIN frames, which are described in *Section , LIN frame formats* 0 Standard LIN frame generation and checking. 1 Enhanced LIN frame generation and checking.
TD	Transfer Direction. This bit control the transfer direction of the data, crc, and checksum byte fields. 0 Data, CRC, and Checksum byte fields received, described in *Section , LIN RX frame generation*. 1 Data, CRC, and Checksum byte fields transmitted, described in *Section , LIN TX frame generation*.
TO	Timeout Value. The content of the field depends on the transfer direction. RX frame: Defines the time available for a complete RX frame transfer, as described in *Section , Slave-Not-Responding-Error detection* TX frame: Must be set to 0.
D	Transmit Data. Data bits for transmission.

LIN receive register (eSCI_LRR)

Figure 644. LIN receive register (eSCI_LRR)

eSCI_BASE + 0x0014 Read Only

	0	1	2	3	4	5	6	7
R	D							
W								
Reset	0	0	0	0	0	0	0	0

This register provides the data bytes of received in case of an LIN RX frame was initiated.

Table 650. eSCI_LRR field descriptions

Field	Description
D	Receive Data. This field provides the data bytes of received LIN RX frames.

LIN CRC polynomial register (eSCI_LPR)

Figure 645. LIN CRC polynomial register (eSCI_LPR)

This register provides the CRC polynom for generation and processing of CRC-enhanced LIN frames.

eSCI_BASE + 0x0018 Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	P															
Reset	1	1	0	0	0	1	0	1	1	0	0	1	1	0	0	1

Table 651. eSCI_LPR field descriptions

Field	Description
P	Polynomial bit $x^{P[n]}$. Used to define the LIN polynomial. Reset value results in $x^{15} + x^{14} + x^{10} + x^{8} + x^{7} + x^{4} + x^{3} + 1$, which is the polynomial used for the CAN protocol.

Control register 3 (eSCI_CR3)

Figure 646. Control register 3 (eSCI_CR3)

eSCI_BASE + 0x001A Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	EROE	ERFE	ERPE	M2	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register is used to control the frame formats and the generation of the ERR bit in the *SCI data register (eSCI_DR)*.

Table 652. eSCI_CR3 field descriptions

Field	Description
3 EROE	ERR flag overrun enable. 0 eSCI_DR[ERR] flag not affected by overrun detection. 1 eSCI_DR[ERR] flag is set on overrun detection during frame reception.
2 ERFE	ERR flag frame error enable. 0eSCI_DR[ERR] flag not affected by frame error detection. 1eSCI_DR[ERR] flag is set on frame error detection for the data provided in eSCI_DR.
1 ERPE	ERR flag parity error enable. 0 eSCI_DR[ERR] flag not affected by parity error detection. 1 eSCI_DR[ERR] flag is set on parity error detection for the data provided in eSCI_DR.
0 M2	Frame Format Mode 2. This control bit together with the M bit of the *Control register 1 (eSCI_CR1)* controls the frame format used. The supported frame formats and the related settings are defines in *Section 28.4.2, Frame formats*.

28.4 Functional description

This section provides a complete functional description of the eSCI block, detailing the operation of the design from the end user perspective in a number of subsections.

28.4.1 Module control

The operational mode of the module is controlled by the MDIS bit in the *Control register 2 (eSCI_CR2)*. The module can transmit and receives data when it is enabled, i.e MDIS=0.

28.4.2 Frame formats

The eSCI module uses the standard NRZ mark/space data format. The eSCI supports three basic frame types, which are the data frames, break characters, and idle characters.

Data frame formats

Each data frame contains a character that is surrounded by a start bit, an optional parity or address bit, and one or two stop bits. The supported data frame formats for transmission and reception are specified in *Table 653*. The supported data frame formats for reception only are specified in *Table 654*.

Table 653. Supported data frame formats for RX and TX

Control				**Frame Content**				
eSCI_CR3	**eSCI_CR1**				**Payload Bits**			
M2	**M**	**PE**	**WAKE**	**Start Bits**	**Character Bits**	**Address Bits(1)**	**Parity Bits**	**Stop Bits**
LIN byte fields (*Figure 647*)								
0	0	0	0	1	8	0	0	1
SCI Frames (8 payload bits)(*Figure 648*)								
0	0	0	0	1	8	0	0	1
0	0	0	1	1	7	1	0	1
0	0	1	0	1	7	0	1	1
SCI Frames (9 payload bits) (*Figure 649*)								
0	1	0	0	1	9	0	0	1
0	1	0	1	1	8	1	0	1
0	1	1	0	1	8	0	1	1

1. The address bit identifies the frame as an address character. See *Section , Multiprocessor communication*.

Table 654. Supported data frame formats for RX only

Control				Frame Content				
eSCI_CR3	eSCI_CR1			Start Bits	Payload Bits			Stop Bits
M2	M	PE	WAKE		Character Bits	Address Bits	Parity Bits	
SCI Frames (2 stop bits) (see *Figure 650*)								
1	0	1	0	1	8	0	1	2
1	1	1	0	1	12	0	1	2

The structure of the LIN byte fields in normal polarity is shown in *Figure 647*.

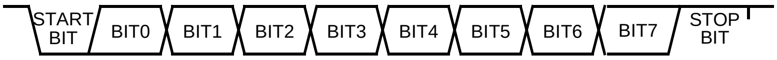


Figure 647. LIN byte field format

The structures of the supported SCI frame formats with 8 payload bits are shown in *Figure 648*.

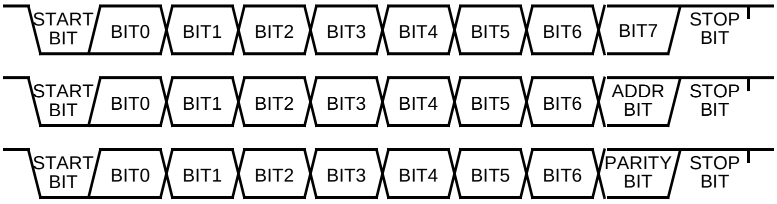


Figure 648. SCI frame formats (8 payload bits)

The structures of the supported SCI frame formats with 9 payload bits are shown in *Figure 649*.

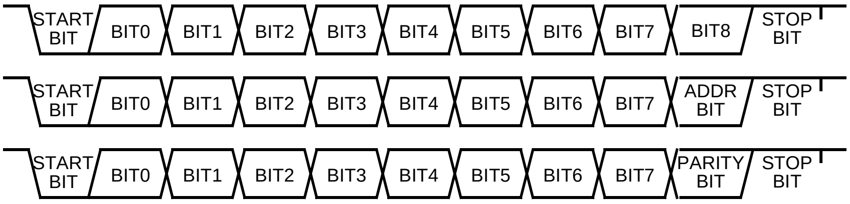


Figure 649. SCI frame formats (9 payload bits)

The structures of the supported SCI frame formats with 2 stop bits in normal polarity are shown in *Figure 650*. This frame format is supported for reception only.

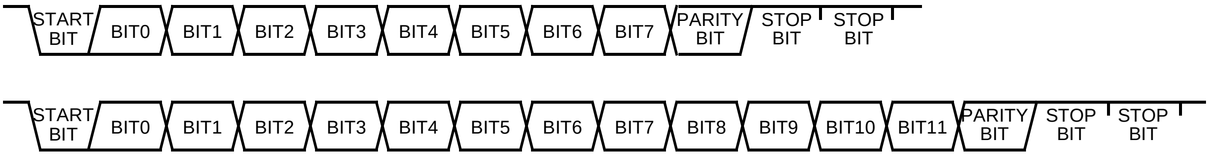


Figure 650. SCI frame formats (2 stop bits)

inverted data frame formats

The structures of the supported data frame formats in inverted polarity are shown in *Figure 651*. These frame types are supported for reception only. The polarity of the RXD pin is controlled by the RXPOL bit in the *Control register 2 (eSCI_CR2)*.

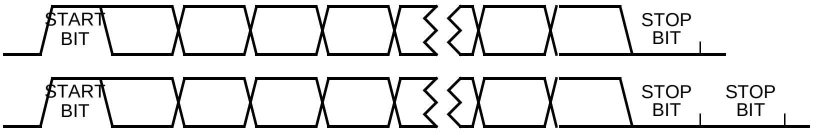


Figure 651. Inverted SCI frame formats

Break character formats

The supported break character formats are specified in *Table 655*.

Table 655. Supported break character formats

Control[1]			Break Character Content		
eSCI_CR3	**eSCI_CR1**	**eSCI_CR2**	**Start Bit**	**Character Bits**	**Delemit Bits**
M2	**M**	**BRK13**			
LIN Break Symbol (see *Figure 652*)					
0	0	0	1	9	1
0	0	1	1	12	1
SCI Break Character (see *Figure 653*)					
0	0	0	1	9	0
0	0	1	1	12	0
0	1	0	1	10	0
0	1	1	1	13	0

1. All codings which are not listed are reserved and must not be used.

The structure and content of the LIN break symbols is shown in *Figure 652*.

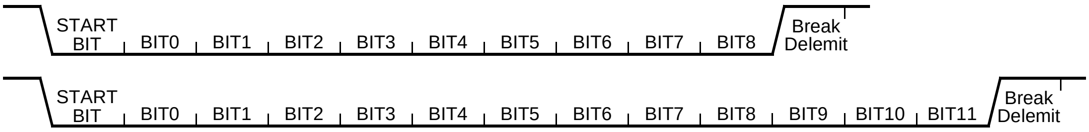


Figure 652. LIN break symbol format

The structure and content of the SCI break characters is shown in *Figure 653*.

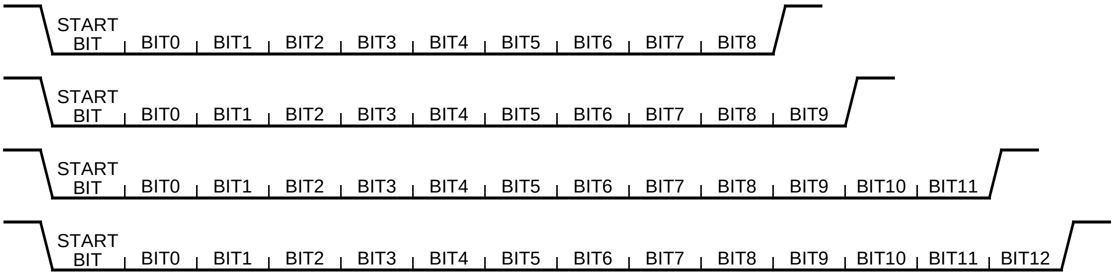


Figure 653. SCI break character formats

Idle character formats

An idle character is a sequence of bits with the value 1. The supported idle character formats are specified in *Table 656*. The preamble has the same structure and content as an idle character.

Table 656. Supported idle character formats

Control		Idle Character Length
eSCI_CR3[M2]	**eSCI_CR1[M]**	
Idle Characters (see *Figure 654*)		
0	0	10
0	1	11
1	0	12
1	1	16

The structure and content of the idle characters is shown in *Figure 654*.

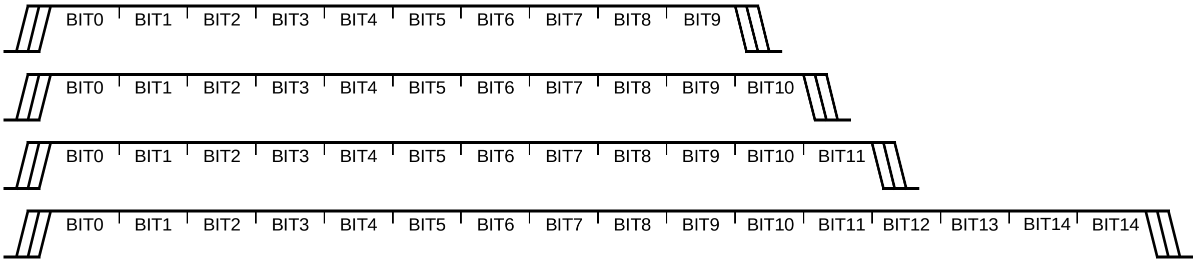


Figure 654. Idle character formats

28.4.3 Baud rate and clock generation

A 13-bit modulus counter in the baud rate generator derives the baud rate for both the receiver and the transmitter. The value written to the SBR field in the *Control register 1 (eSCI_CR1)* determines the module clock divisor. The baud rate clock is synchronized with the bus clock and drives the receiver. The baud rate clock divided by 16 drives the transmitter. The receiver has an acquisition rate of 16 samples per bit time.

The baud rate generator is enabled when the TE bit or RE bit in the *Control register 1 (eSCI_CR1)* is set to 1 for the first time. The baud rate generator is disabled when SBR = 0.

Baud rate generation is subject to one source of error:

- Integer division of the module clock may not give the exact required target baud rate.

Figure 657 lists some examples of achieving target baud rates with a module clock frequency of MCLK = 10.2 MHz.

Table 657. Baud rates error example (MCLK = 10.2 MHz)

eSCI_CR1[SBR]	RCLK (Hz)	TCLK (Hz)	Target Baud Rate	Error (%)
17	600,000.0	37,500.0	38,400	2.3
33	309,090.9	19,318.2	19,200	.62
66	154,545.5	9659.1	9600	.62
133	76,691.7	4793.2	4800	.14
266	38,345.9	2396.6	2400	.14
531	19,209.0	1200.6	1200	.11
1062	9604.5	600.3	600	.05
2125	4800.0	300.0	300	.00
4250	2400.0	150.0	150	.00
5795	1760.1	110.0	110	.00

Module clock

The module clock MCLK is derived from the system bus clock. It has the same phase and frequency.

Transmitter clock

The transmitter clock TCLK is used to drive the data to the serial bus via the TXD pin. It is derived from the system bus clock by the baud rate generator. The baud rate generator is controlled by the value of the SBR field in the *Control register 1 (eSCI_CR1)*. The frequency of the transmitter clock is determined by *Equation 33* and defines the length of the transmitted bits, which is denoted as the *bit time*.

Equation 33

$$f_{TCLK} = \frac{f_{MCLK}}{16 \cdot SBR}$$

Receiver clock

The receiver clock RCLK is used to sample the data received on the RXD or TXD pin. It is derived from the system bus clock by the baud rate generator. The baud rate generator is controlled by the value of the SBR field in the *Control register 1 (eSCI_CR1)*. The frequency of the receiver sample clock is determined by *Equation 34*.

Equation 34

$$f_{RT} = \frac{f_{MCLK}}{SBR}$$

The frequency of the receiver clock is 16 times the frequency of the transmitter clock, this each bit is sampled with 16 samples. Each of the 16 samples of a bit has a sample number assigned, which is defined by the receiver sample counter RSC. The n-th sample is denoted by RSn. The receiver sample counter RSC is updated with each rising edge of the receiver clock RCLK.

28.4.4 Baud rate tolerance

A transmitting device may be operating at a baud rate below or above the receiver baud rate. Accumulated bit time misalignment can cause one of the three stop bit data samples RS8, RS9, and RS10 to fall outside the actual stop bit. A noise error will occur if the stop bit sample RS8, RS9, and RS10 samples are not all the same logical value 1. A framing error will occur if the receiver clock is misaligned in such a way that the majority of the RS8, RS9, and RS10 stop bit samples are a logic zero.

Faster receiver tolerance

In this case the receiver has a higher baud rate than the transmitter, thus the stop bit sampling starts already in the last transmitted payload bit. To ensure the correct, noise and framing error free reception of the stop bit, the samples RS8, RS9, and RS10 must be located in the transmitted stop bit as shown in *Figure 655*.

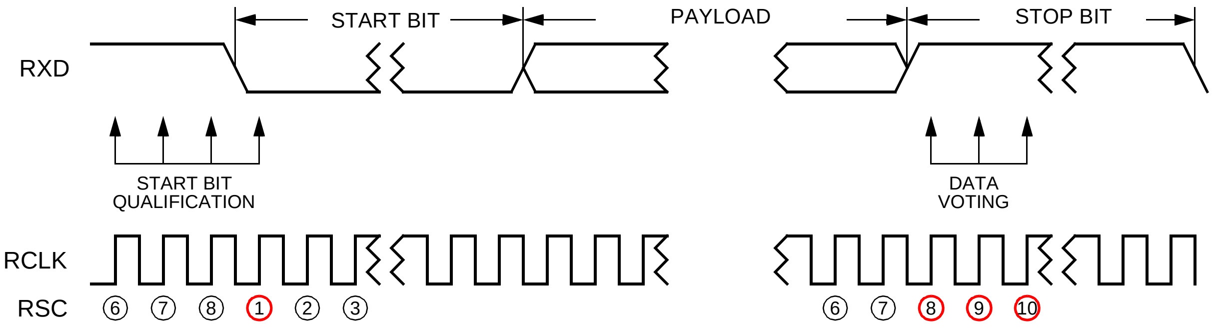


Figure 655. Faster Receiver

The maximum tolerance that ensures error free reception can be calculated with the assumption, that RS7 is sampled during the last transmitted payload bit and RS8 is sampled in the stop bit.

For an frame with *n* payload bits the transmitter starts the transmission of the stop bit

Equation 35

$$tx_{STOP} = (n+1) \cdot 16 \cdot RT_{TR}$$

after the start of the transmission of the start bit.

For an frame with *n* payload bits the receiver samples the RS8 sample of the stop bit

Equation 36

$$rx_{STOP} = (n+1) \cdot 16 \cdot RT_{RE} + 7 \cdot RT_{RE}$$

after the successful qualification of the start bit.

To ensure error free reception of the stop bit, the transmitter must start the transmission of the stop bit before the receiver samples RSC8.

Equation 37

$$tx_{STOP} < rx_{STOP}$$

The maximum percent difference between the receiver baud rate and the transmitter baud rate is:

Equation 38

$$\Delta baudrate \le \left(\frac{rx_{STOP} - tx_{STOP}}{rx_{STOP}}\right) \times 100$$

The maximum percent differences for the supported frames is given in *Table 658*

Table 658. Faster receiver maximum tolerance

payload bits	max baudrate difference	tx_{STOP}	rx_{STOP}
8	4.63%	144	151
9	4.19%	160	167
13	3.03%	224	231

Slower receiver tolerance

In this case the receiver has a slower baud rate than the transmitter, thus the stop bit sampling is still running while the next start bit is already transmitted. To ensure the correct, noise and framing error free reception of the stop bit, the samples RS8, RS9, and RS10 must be located in the transmitted stop bit as shown in *Figure 656*.

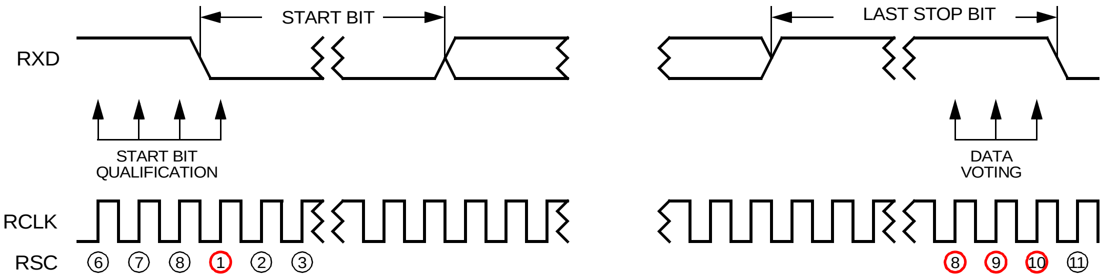


Figure 656. Slower receiver

The maximum tolerance that ensures error free reception can be calculated with the assumption, that RS11 is sampled in the transmitted start bit and RS10 is sampled in the last stop bit.

For an frame with *n* payload bits and *s* stop bits, the transmitter starts the transmission of the next start bit

Equation 39

$$tx_{START} = (n + s + 1) \cdot 16 \cdot RT_{TR}$$

after the start of the transmission of the previous start bit.

For an frame with *n* payload bits and s stop bits, the receiver samples the RS10 sample of the last stop bit

Equation 40

$$rx_{STOP} = (n + s) \cdot 16 \cdot RT_{RE} + 9 \cdot RT_{RE}$$

after the successful qualification of the start bit.

To ensure error free reception of the last stop bit, the transmitter must start the transmission of the start bit after the receiver samples RS10.

Equation 41

$$rx_{STOP} < tx_{START}$$

The maximum percent difference between the receiver baud rate and the transmitter baud rate is:

Equation 42

$$\Delta baudrate \le \left(\frac{tx_{START} - rx_{STOP}}{tx_{START}}\right) \times 100$$

The maximum percent differences for the supported frames is given in *Table 659*

Table 659. Slower receiver maximum tolerance

payload bits	stop bits	max baudrate difference	rx_{STOP}	tx_{START}
8	1	4.37%	153	160
9	1	3.97%	169	176
9	2	3.57%	185	196
13	2	2.73%	249	256

28.4.5 SCI mode

SCI mode configuration

The application must configure the following bits and fields in order to achieve correct SCI operation.

- enable *SCI* Mode
  - *LIN control register (eSCI_LCR)*[LIN]:= 0
- select *baud rate*
  - *Control register 1 (eSCI_CR1)*[SBR]
- select *receiver input* mode
  - *Control register 1 (eSCI_CR1)*[LOOPS]
  - *Control register 1 (eSCI_CR1)*[RSRC]
- select *frame format*
  - *Control register 1 (eSCI_CR1)*[M]
  - *Control register 1 (eSCI_CR1)*[PE]
  - *Control register 1 (eSCI_CR1)*[WAKE]
  - *Control register 3 (eSCI_CR3)*[M2]
- select *parity type*
  - *Control register 1 (eSCI_CR1)*[PT]

Transmitter

The transmitter supports the transmission of all frame types defined in *Table 653*, of all break characters defined in *Table 655*, and of all idle characters defined in *Table 656*.

Transmitter states and transitions

The transmitter has four basic states which are shown and described in *Table 660*. The state transitions that can triggered by the application commands are shown in *Table 661*. The state transitions that can triggered by the module are shown in *Table 662*. The state diagram of the transmitter is shown in *Figure 657*.

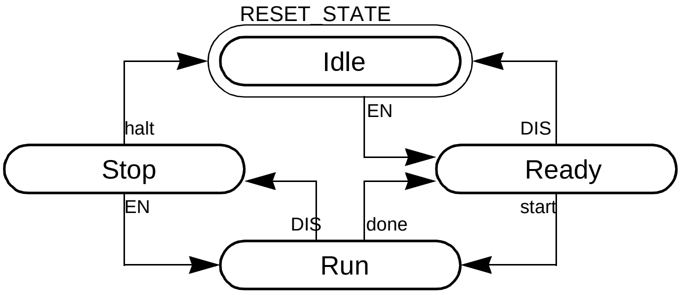


Figure 657. Transmitter state diagram

The current state of the transmitter can be determined by the TE control bit in the *Control register 1 (eSCI_CR1)* and the TACT status bit in *Status register (eSCI_SR)*.

Table 660. Transmitter states

State	Indication		Description
	eSCI_CR1[TE]	eSCI_SR[TACT]	
Idle	0	0	Transmitter is *disabled* and *no* transmission is running
Ready	1	0	Transmitter is *enabled* and *no* transmission is running
Run	1	1	Transmitter is *enabled* and transmission is running
Stop	0	1	Transmitter is *disabled* and transmission is running

The application triggers a transition described in *Table 661* when it issues a command by writing to the TE bit in the *Control register 1 (eSCI_CR1)*. The transition is triggered only if the conditions are fulfilled. As a result of the transition the state of the transmitter is changed as shown in *Figure 657* and the action given in *Table 661* is executed.

Table 661. Transmitter application transitions

Transition	Command	Precondition	Action	Description
EN	eSCI_CR1[TE]:=1	eSCI_CR1[TE]=0	iPRE:=1	Transmitter is *enabled* by application command.
DIS	eSCI_CR1[TE]:=0	eSCI_CR1[TE]=1		Transmitter is *disabled* by application command

The module transition shown in *Table 662* are triggered when the described condition or event occurs. The send break bit SBK in the *Control register 1 (eSCI_CR1)* is check for the start condition. The internal commit bit iCMT, the transmitter active bit TACT in the *Status register (eSCI_SR)*, the TDRE, and the TC flag in the *Status register (eSCI_SR)* are changed as a action result of the transition.

Table 662. Transmitter module transitions

Transition	Condition	Action	Description
start	(State=Ready) and (SBK=1 or iPRE=1 or iCMT=1)	TACT:=1	Start of transmission of data frame or special character when data are available or character transmission request is pending.
done	State=Run and last stop bit transmitted	TACT:=0 TC:= (SBK=0 & iPRE=0 & iCMT=0)	Finished transmission of data frame or special character and transmitter still enabled. Transmission is complete if no transmit request is pending.
halt	State=Stop and last stop bit transmitted	TACT:=0 TC:=1 iCMT:=0	Finished transmission of data frame or special character and transmitter was disabled.

Frame and character transmission

The transmitter starts the transmission of a data frame or special character when the condition for the *start* transition as described in *Table 662* is fulfilled. There are three source for data or character transmission. The priority among these source are specified in *Table 663*. All three sources can be available at one point in time.

Table 663. Transmit Source Priority

Priority	Indication	Transmission Source
(highest) 0	iPRE=1	Preamble.
1	eSCI_CR1[SBK]=1	Break character.
(lowest) 2	iCMT=1	*SCI data register (eSCI_DR)* frame.

CPU controlled SCI data frame transmission

The transmission of a data frame is started when the transmitter is in its Ready state and only the commit bit iCMT is set.

As the first step, the content of the *SCI data register (eSCI_DR)* is transferred into the internal transmitter shift register. When this transfer is finished, the internal commit bit iCMT is cleared and the transmit data register empty flag TDRE in the *Status register (eSCI_SR)* is set. If the transmit interrupt enable bit TIE in the *Control register 1 (eSCI_CR1)* is also set, the TDRE flag generates a transmitter interrupt request.

The transmitter shift register then shifts a frame out through the TXD output signal, which is prefaced with a start bit and appended with the parity bit, if configured, and the configured number of stop bits.

When the last stop bit has been transmitted and the application has not disabled the transmitter, the transmitter returns to the Ready state via the *done* transition. If no frame or character transmit request is pending, the transfer complete flag TC in the *Status register (eSCI_SR)* is set.

If the application has disabled the transmitter while the frame is transmitted and stop bit has been transmitted, the transmitter goes into the Idle state via the *halt* transition. The transfer complete flag TC in the *Status register (eSCI_SR)* is set and the internal commit bit iCMT is cleared.

DMA controlled SCI data frame transmission

In this mode, the eSCI module handles the generation of Data Frames internally.

When new data required for transmission, the module generates the transmit DMA request and the DMA controller delivers the required data via write accesses to *SCI data register (eSCI_DR)*. The write access to the low byte of *SCI data register (eSCI_DR)* triggers the transmission of the data. The write access to the high byte of *SCI data register (eSCI_DR)* triggers no internal operation.

The application request the eSCI module to enter this mode by setting the TXDMA bit in the *Control register 2 (eSCI_CR2)*. From this point in time, the module starts the generation of DMA requests and frame transmission. Before entering this mode, the application should perform the following actions:

1. Configure the module for SCI mode.
2. Enable the transmitter by setting TE in *Control register 1 (eSCI_CR1)* to 1.
3. Set up the DMA controller channel and provide frame data in system memory

A block diagram which presents an overview of the DMA Controlled Date Frame Transmission is shown in *Figure 658*.

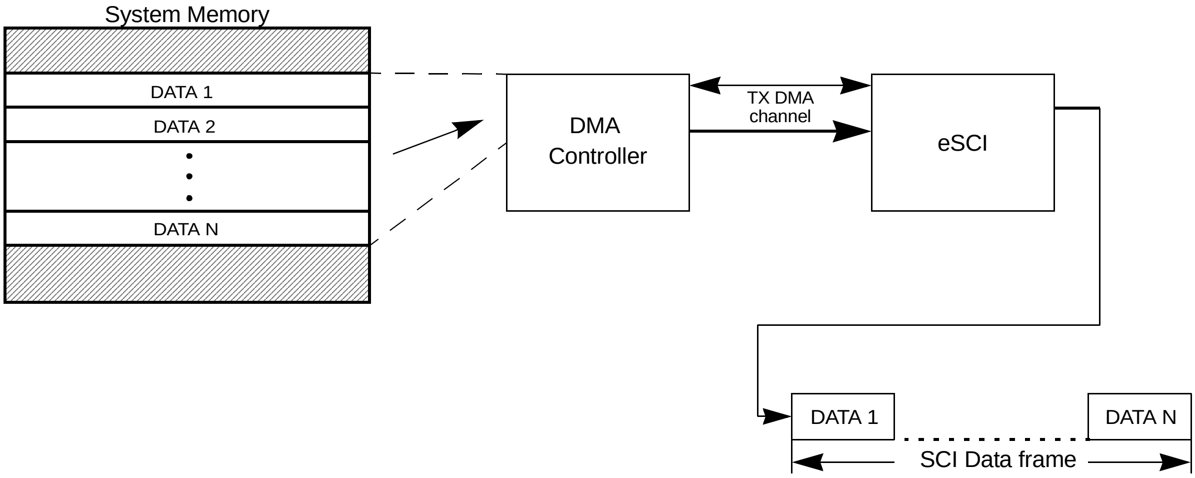


Figure 658. DMA controlled SCI data frame generation

Parity generation

The eSCI module generates the parity bit in transmitted data frame when the parity enable bit PE in the *Control register 1 (eSCI_CR1)* is set. The parity type bit PT in the *Control register 1 (eSCI_CR1)* defines whether the odd or even parity is generated.

Preamble transmission

The transmission of a preamble is started when the transmitter is in Ready state, the internal iPRE bit, which is not visible to the application, is set, and the SBK in the *Control register 1 (eSCI_CR1)* is clear.

After the transmission of the stop bit and if the application has not disabled the transmitter, the transmitter returns to the Ready state via the *done* transition. If no frame or character transmit request is pending, the transfer complete flag TC in the *Status register (eSCI_SR)* is set.

If the application has disabled the transmitter while the preamble is transmitted and if the stop bit has been transmitted, the transmitter goes into the Idle state via the halt transition. The transfer complete flag TC in the *Status register (eSCI_SR)* is set and the internal commit bit iCMT is cleared.

Break character transmission

The transmission of a break character is started when the transmitter is in Ready state and the send break character bit SBK in the *Control register 1 (eSCI_CR1)* is set. After the transmission of the break character and if the application has not disabled the transmitter, the transmitter returns to the Ready state via the *done* transition and restarts the transmission. As long as SBK bit remains set, the transmitter continues to send break characters.

When the application has cleared the SBK bit or has disabled the transmitter, the transmitter continues to transmit the current break character and after it has finished the transmission of this break character it transmits a stop bit. The stop bit at the end of a break character sequence guarantees the recognition of the start bit of the next data frame.

After the transmission of the stop bit and if the application has not disabled the transmitter, the transmitter returns to the Ready state via the *done* transition. If no frame or character transmit request is pending, the transfer complete flag TC in the *Status register (eSCI_SR)* is set.

If the application has disabled the transmitter while the break character is transmitted and if the stop bit has been transmitted, the transmitter goes into the Idle state via the *halt* transition. The transfer complete flag TC in the *Status register (eSCI_SR)* is set and the internal commit bit iCMT is cleared.

Receiver

The receiver supports the reception of all data frame types defined in *Table 653* and *Table 654*, of all break characters defined in *Table 655*, and of all idle characters defined in *Table 656*.

Receiver states and transitions

The receiver has four basic states which are shown and described in *Table 661*. The state transitions that can triggered by the application commands are shown in *Table 661*. The state transitions that can triggered by the module are shown in *Table 662*. The state diagram of the transmitter is shown in *Figure 657*.

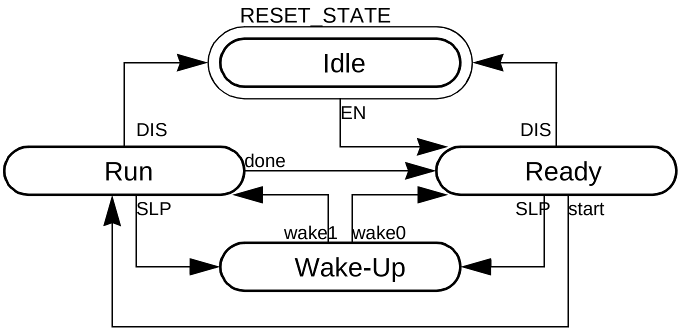


Figure 659. Receiver state diagram

The current state of the receiver can be determined by the RE and RWU bit in the *Control register 1 (eSCI_CR1)* and the RAF status bit in *Status register (eSCI_SR)*.

Table 664. Receiver states

State	Indication			Description
	RE	RAF	RWU	
Idle	0	0	0	Receiver is *disabled* and *no* reception is running
Ready	1	0	0	Receiver is *enabled* and *no* reception is running
Run	1	1	0	Receiver is *enabled* and reception is running
Wake-up	1	—	1	Receiver is in wake-up mode

The application triggers a transition described in *Table 665* when it issues a command by writing to the RE bit in the *Control register 1 (eSCI_CR1)*. The transition is triggered only if the conditions are fulfilled. As a result of the transition the state of the receiver is changed as shown in *Figure 659* and the action given in *Table 665* is executed.

Table 665. Receiver application transition

Transition	Command	Condition	Action	Description
EN	RE:=1	RE=0		Receiver is *enabled* by application command.
DIS	RE:=0	RE=1		Receiver is *disabled* by application command
SLP	RWU:=1	RE=1		Receiver is set into wake-up mode

The module transitions shown in *Table 666* are triggered when the described event occurs.

Table 666. Receiver module transition

Transition	Condition	Action	Description
start	(State=Ready,Run) and (start bit qualified)	RAF:=1	Start of reception of data frame or break character.
done	(State=Run) and (start bit not verified or idle character received)	RAF:=0	Start Bit not Verified or Idle Character received.
wake0	(State=Wake-up) and (idle character received)	RWU:=0	Wake-up Idle Character received.
wake1	(State=Wake-up) and (address frame received)	RWU:=0	Wake-up address frame received.

Receiver input mode selection

This section describes the three receiver input modes supported by the eSCI module. The modes are selected by the LOOPS and RSRC control bits in the *Control register 1 (eSCI_CR1)*.

Dual wire mode

In Dual Wire Mode, the eSCI uses the TXD pin for transmitting and the RXD pin for data receiving.

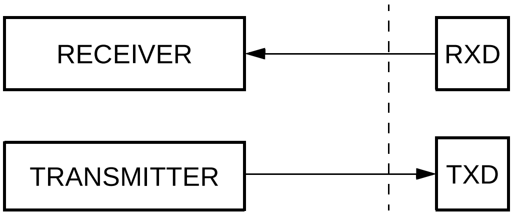


Figure 660. Dual Wire Mode

Single wire mode

In Single Wire Mode, the RXD pin is disconnected from the eSCI module and the TXD pin is used for both receiving and transmitting.

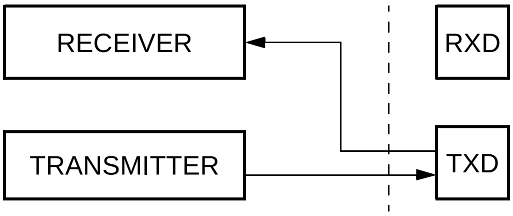


Figure 661. Single Wire Mode

The TXDIR bit (eSCI_CR2[1]) determines whether the TXD pin is going to be used as an input (TXDIR = 0) or an output (TXDIR = 1) in this mode of operation.

Loop mode

In Loop Mode, the input of the receiver is driven by the output of the transmitter. The RXD pin is disconnected from the eSCI module.

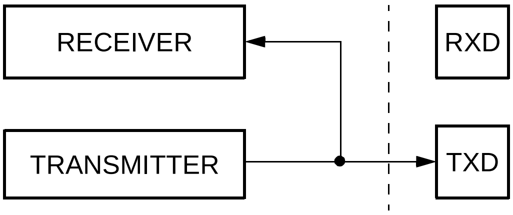


Figure 662. Loop Mode

Frame and character reception

The receiver is started when it is in Ready or Wake-up state and on the selected receiver input (see *Section , Receiver input mode selection*) an active signal is sampled. The receiver enters the Run or Wake-up state. The received bits are recovered by the bit sampling described in *Section , Bit sampling*. During the reception, the received bits are shifted into the internal shift register.

Break character detection

The receiver does not provide any means to detect the reception of a break character. Instead, break characters are processed as data frames. Due to the received 0 at the stop bit location, the reception of a break character causes at least a framing error. The error reporting is performed as described in *Section , Reception error reporting*.

Idle character detection

The Idle character detection starts after the reception of the last stop bit.

CPU controlled SCI data frame reception

This section describes the reception process when the receiver is in the Run state.

When the required number of frame bits have been received, the payload bits of the received frame are transferred into *SCI data register (eSCI_DR)* if the RDRF flag is 0. The receive data register full flag RDRF in *Status register (eSCI_SR)* is set. If the receive interrupt enable bit RIE in the *Status register (eSCI_SR)* is set, the RDRF interrupt request is generated.

If an idle character has been detected, the IDLE flag in the *Status register (eSCI_SR)* is set. If the idle line interrupt enable bit ILIE in the *Control register 1 (eSCI_CR1)* is set, the IDLE interrupt request is generated.

If any of the receiver errors described in *Section , Reception error reporting* have been occurred, that corresponding flags will be set.

If the application disabled the receiver by clearing the receiver enable bit RE in the *Status register (eSCI_SR)* the current frame is discarded and no flags will be updated.

DMA controlled SCI data frames reception

In this mode, the eSCI module controls the reception of SCI Data frames automatically and utilizes the connected DMA channels. A block diagram which presents an overview of the DMA Controlled SCI Data Frame reception is shown in *Figure 663*. The RX DMA channel is used to transfer the received frame data into the memory.

When new data was received, the module generates the receive DMA request and the DMA controller retrieves the provided data from the *SCI data register (eSCI_DR)*. The read access from the low byte of the *SCI data register (eSCI_DR)* signals the end of the DMA cycle for the current data and triggers the reception of new data. The read access from the *SCI data register (eSCI_DR)* triggers no internal action.

The application requests the eSCI module to enter this mode by setting the RXDMA bit in the *Control register 2 (eSCI_CR2)*. From this point in time, the module starts the generation of DMA requests and frame transmission and reception. Before entering this mode, the application should perform the following actions:

1. Configure the module for SCI mode.
2. Enable the receiver by setting RE in *Control register 1 (eSCI_CR1)* to 1.
3. Set up the DMA controller channel.

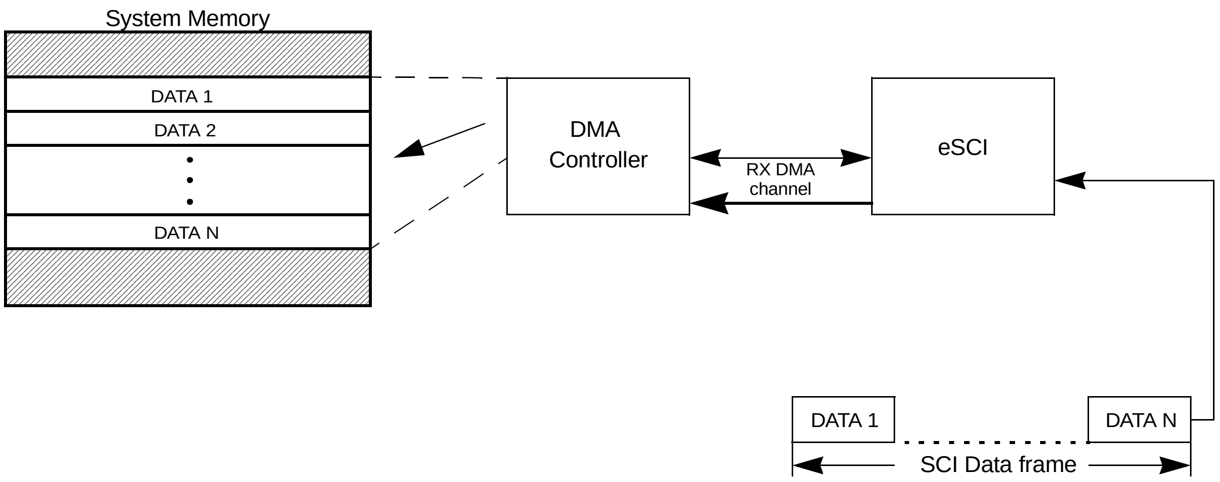


Figure 663. DMA controlled SCI data frame reception

Receiver overrun

When the eSCI module has received a frame and attempts to transfer the payload data of the received frame into the *SCI data register (eSCI_DR)* but neither the application nor the DMA controller has read the *SCI data register (eSCI_DR)* since its last update, the overrun flag OR in the *Status register (eSCI_SR)* is set. The data contained in *SCI data register (eSCI_DR)* are not changed and the received data are lost.

Wake-up frame reception

This section describes the reception process when the receiver is in the Wake-up state.

When the required number of frame bits have been received, the payload bits of the received frame are transferred into *SCI data register (eSCI_DR)* if the RDRF flag is 0.

If the *address-mark* wake-up mode is selected and the received frame has the address bit set, the receive data register full flag RDRF in *Status register (eSCI_SR)* is set. If the receive interrupt enable bit RIE in the *Status register (eSCI_SR)* is set, the RDRF interrupt request is generated. The RWU bit is cleared, and the receiver enters the Run state via the wake1 transition.

If the *idle line* wake-up mode is selected and the receiver has detected an idle character, The RWU bit is cleared, and the receiver enters the Ready state via the wake0 transition.

If any of the receiver errors described in *Section , Reception error reporting* have occurred, the corresponding flags will be set.

Bit sampling

The receiver samples the selected receiver input (see *Section , Receiver input mode selection*) with the receiver clock RCLK. The bit sampling for start bit detection is shown in *Figure 664*, the bit sampling for data and stop bit reception is shown in *Figure 665*. The samples indicated by dashed arrows are not used by the receiver. The received data bits are transferred into the internal shift register after the data strobing. If noise or framing errors were detected, this is flagged as described in *Section , Reception error reporting*.

Bit synchronization

To adjust for baud rate mismatch, a synchronization of the cyclic receive sample counter RSC is performed during start bit reception as described in *Section , Start bit sampling*.

Start bit sampling

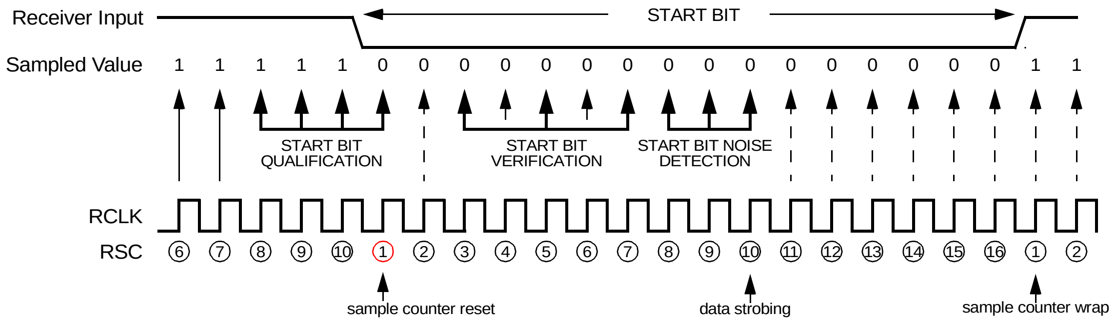


Figure 664. Start Bit Sampling and Strobing

The sampling of the start bit consists of three phases, the start bit qualification, the start bit verification, and the start bit noise detection.

Start bit qualification

To adjust for baud rate mismatch, the cyclic receive sample counter RSC is re-synchronized after a successful start bit qualification.

A start bit is successfully qualified if

- the start qualification is active, and
- a low sample is read, and
- the low sample was preceded by three consecutive high samples.

The start bit qualification becomes active

- after module reset, or
- after receiver disable and subsequent enable, or
- after the 7-th sample if the start bit verification failed, or
- after the 10-th sample of last stop bit of the preceding frame (example shown in *Figure 664*).

The start bit qualification becomes inactive

- after successful start bit qualification.

Start bit verification

After the successful start bit qualification the receiver starts to verify the start bit by a two out of three samples majority voting.

A start bit is verified if at least two out of the three sample RSC3, RSC5, and RSC7 are sampled low. Noise is detected when exactly one out of the three samples is high. In this case, the noise flag eSCI_SR[NF] is set. The result of the start bit verification is summarized in *Table 667*.

Table 667. Start bit verification result

[RSC3, RSC5, RSC7]	Start Bit Verified	Verification Noise Detected
000	Yes	No
001	Yes	Yes
010	Yes	Yes
100	Yes	Yes
011	No	No
101	No	No
110	No	No
111	No	No

If the start bit verification was *not successful*, the receiver activates the start bit qualification. If the start bit verification was *successful*, the receiver continues sampling to perform data noise detection on the samples at RSC8, RSC9, and RSC10. The result of the start bit data noise detection is summarized in *Table 668*. If noise is detected, the noise flag eSCI_SR[NF] is set.

Table 668. Start bit noise detection

[RSC8, RSC9, RSC10]	Noise Detected
000	No
001	Yes
010	Yes
100	Yes
011	Yes
101	Yes
110	Yes
111	Yes

Data bit sampling

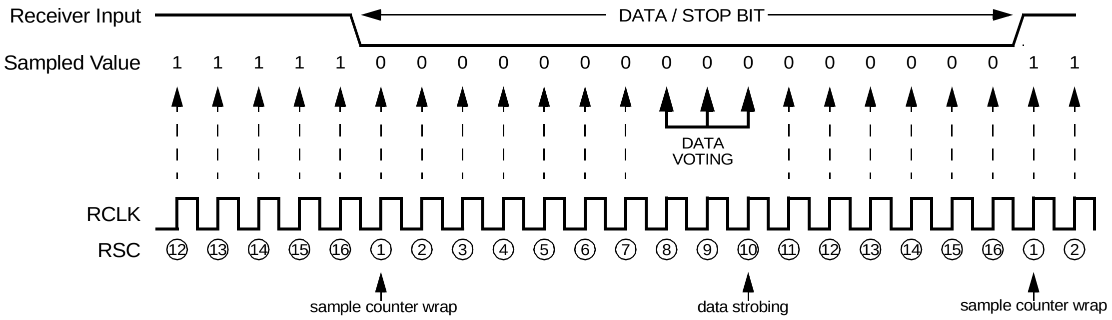


Figure 665. Data and stop bit sampling and strobing

To determine the value of a data bit and to detect noise, a two out of three majority voting is performed on the samples RSC8, RSC9, and RSC10. *Table 669* summarizes the results of the data bit sample. The receiver detects the number of data bit according to the selected frame format. If noise is detected, the noise flag eSCI_SR[NF] is set.

Table 669. Data bit sampling

[RSC8, RSC9, RSC10]	Data Bit Value	Noise Detected
000	0	No
001	0	Yes
010	0	Yes
100	0	Yes
011	1	Yes
101	1	Yes
110	1	Yes
111	1	No

Stop bit verification

The reception of a valid stop bit is verified if at least two out of the sample RSC8, RSC9, and RSC10 are sampled high. If this is not that case, a framing error is detected. Noise is detected if not all of the samples are of the same value. In this case, the noise flag eSCI_SR[NF] is set. The result of the stop bit verification is summarized in *Table 670*.

Table 670. Stop bit verification

[RSC8, RSC9, RSC10]	Stop Bit Verified	Framing Error Detected	Noise Detected
000	No	Yes	No
001	No	Yes	Yes
010	No	Yes	Yes
100	No	Yes	Yes
011	Yes	No	Yes
101	Yes	No	Yes
110	Yes	No	Yes
111	Yes	No	No

Parity checking

The eSCI module calculates the parity of a received character and checks is versus the received parity bit in the received data frame when the parity enable bit PE in the *Control register 1 (eSCI_CR1)* is set. The parity type bit PT in the *Control register 1 (eSCI_CR1)* defines whether to check for odd or even parity is generated. If an parity error is detected, this is reported as described in *Section , Reception error reporting*.

Reception error reporting

The receiver can detect four error types: parity errors, framing errors, noise errors, and the overrun error.

The receiver reports the errors detected during frame reception at the end of the reception of the last stop bit of a frame. For error reporting the receiver utilizes the OR, NF, FE, and PF flags in the *Status register (eSCI_SR)*.

If the receiver has detected an overrun as described in *Section , Receiver overrun*, only the OR flag is set. All other error flags are not updated.

If the receiver has detected noise as described in *Section , Bit sampling* the NF flag is set.

If the receiver has *not* detected an overrun and has detected a framing error as described in *Section , Bit sampling* the FE flag is set.

If the receiver has *not* detected an overrun and has detected a parity error as described in *Section , Parity checking* the PF flag is set.

Multiprocessor communication

The multiprocessor communication allows one processor to send blocks of frames to other processors on the same serial link. To avoid the received data interrupt for frames not intended for the processor, the eSCI receiver can be put into the Wake-up state. If the receiver is in the Wake-up state, the eSCI will still load the received data into the *SCI data register (eSCI_DR)*, but will not set the RDRF flag and consequently not request the RDRF interrupt.

The receiver leaves the Wake-up state and clears the RWU bit in the *Control register 1 (eSCI_CR1)* when the wake-up pattern configured by WAKE bit in *Control register 1 (eSCI_CR1)* is received. The eSCI module supports two types of wake-up patterns, the idle-line wakup pattern and the address-mark wake-up pattern.

Idle-Line wake up

The idle-line wake-up mode is selected when the WAKE bit in *Control register 1 (eSCI_CR1)* is 0. In this mode, the receiver leaves the wake-up state, when an idle character is detected as described in *Section , Idle character detection*. The next received frame is the address frame that contains address information which can be evaluated by the application. If the application decides not to receive the frame block, it can set the RWU bit in the *Control register 1 (eSCI_CR1)* and return the receiver to the wake-up state.

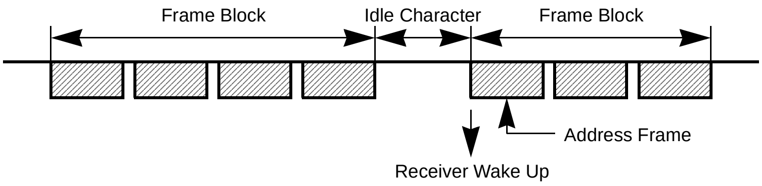


Figure 666. Idle-Line wake up

Address-Mark wake up

The address-mark wake-up mode is selected when the WAKE bit in *Control register 1 (eSCI_CR1)* is 1. If the WAKE bit is set, the address bit is added to the frame format. In this mode, the receiver leaves the wake-up state, when a data frame with the address bit value of 1 was received. This frame is the address frame and contains address information which can be evaluated by the application. If the application decides not to receive the frame block, it can set the RWU bit in the *Control register 1 (eSCI_CR1)* and return the receiver to the wake-up state. All data frames that belong to the frame block must have the address bit cleared.

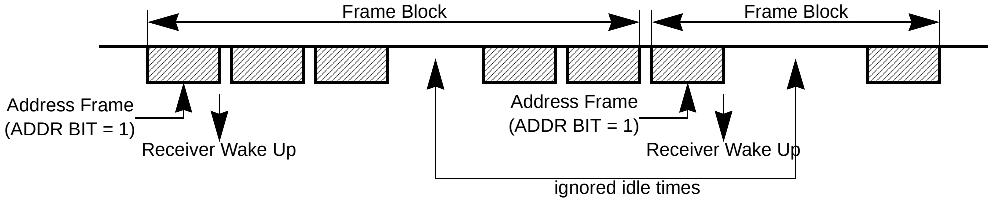


Figure 667. Address-Mark wake up

28.4.6 LIN mode

The eSCI provides support for the LIN protocol. It can be used to automate most tasks of a LIN master. In conjunction with the DMA interface it is possible to transmit entire LIN frames and sequences of LIN frames as well as to receive data from LIN slaves without application intervention. There is no special support for LIN slave mode.

LIN mode configuration

The application must configure the following bits and fields in order to achieve correct LIN operation. The configuration of bits and fields not mentioned in this section depend on the connected LIN slaves and the current application.

- enable *LIN* Mode
  - *LIN control register (eSCI_LCR)*[LIN]:= 1
- select *RXD* pin as receiver input
  - *Control register 1 (eSCI_CR1)*[LOOPS]:= 0
  - *Control register 1 (eSCI_CR1)*[RSRC]:= 0
- select *LIN byte fields* as used frame format
  - *Control register 1 (eSCI_CR1)*[M]:= 0
  - *Control register 1 (eSCI_CR1)*[PE]:= 0
  - *Control register 1 (eSCI_CR1)*[WAKE]:= 0
  - *Control register 3 (eSCI_CR3)*[M2]:= 0
- select *break character length* of 13 bit as required by LIN 2.0
  - *Control register 2 (eSCI_CR2)*[BRK13]:= 1
- select transmission stop on *bit error* detection
  - *Control register 2 (eSCI_CR2)*[SBSTP]:= 1
- select transmission DMA stop on *bit error* detection
  - *Control register 2 (eSCI_CR2)*[BSTP]:= 1
- enable both *transmitter* and *receiver*
  - *Control register 1 (eSCI_CR1)*[TE]:= 1
  - *Control register 1 (eSCI_CR1)*[RE]:= 1

LIN frame formats

The term LIN frame refers to a sequence of LIN byte fields preceded by a break character, both are described in *Section 28.4.2, Frame formats*. The eSCI module allows to generate LIN frames for LIN slaves of LIN standards 1.3 and 2.0.

LIN byte field reception

The reception of a LIN byte field starts with the successful start bit qualification and is finished with the reception of the 16-th sample of the stop bit when no start bit start bit qualification pattern has been detected. If a start bit start bit qualification pattern has been detected at or after the 10-th sample of the stop bit, the reception ends at this sample. An ongoing reception is indicated by the RAF status bit in *Status register (eSCI_SR)*.

The RAF flag is set if all of the following conditions are fulfilled,

1. the receiver is enabled (eSCI_CR1[RE] = 1), and
2. the LIN task is not in reset (eSCI_LCR[LRES] = 0), and
3. the start bit start bit qualification pattern has been received (see *Section , Start bit sampling*).

The RAF flag is cleared if at least one of the following conditions is fulfilled,

1. the receiver is disabled (eSCI_CR1[RE] = 0), or
2. the LIN task is in reset (eSCI_LCR[LRES] = 1), or
3. the start bit verification fails at sample 7 according to *Table 667*, or
4. the 16-th sample of the stop bit has been received and no start bit qualification pattern has been detected at or after the 10-th sample.

Standard LIN frames

A standard LIN frame, shown in *Figure 668* consists of a break character, a sync field, an ID field, zero or more data fields, and a checksum field. The data fields and the checksum field are generated by the LIN master for TX LIN frames and generated by the LIN slave for RX LIN frames. The header fields will always be generated by the LIN master.

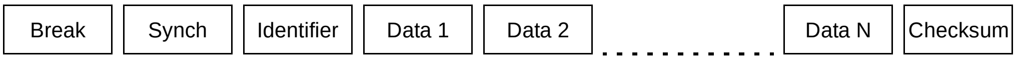


Figure 668. Standard LIN frame format

CRC Enhanced LIN frames

The CRC Enhanced LIN frames shown in *Figure 669* contain two additional CRC byte fields. These fields are located between the last data field and the Checksum field. The value of the CRC is calculated on the same byte fields as the Checksum is calculated on. The polynom used for the CRC calculation is defined by *LIN CRC polynomial register (eSCI_LPR)*. The eSCI module generates the CRC fields for TX frames and checks the CRC fields for RX frames if the CRC bit in the *LIN transmit register (eSCI_LTR)* was written with a value of 1.

Figure 669. CRC Enhanced LIN frame format

The CRC Enhanced LIN frames are not part of the LIN standard.

LIN TX frame generation

The eSCI module supports two modes of LIN TX Frame generation, the CPU controlled mode and the DMA controlled mode. In the CPU controlled mode, the application provides the required frame configuration and frame data by subsequent CPU write accesses to the *LIN transmit register (eSCI_LTR)*. In the DMA controlled mode, the DMA controller provides the required frame configuration and frame data in response to DMA requests generated by the eSCI module.

CPU controlled LIN TX frame generation

In this mode, the application initiates the generation of an LIN TX Frame and provides the data to be transmitted by a sequence of subsequent CPU write accesses to the *LIN transmit register (eSCI_LTR)*. When the eSCI module has processed the data written into *LIN transmit register (eSCI_LTR)*, the TXRDY interrupt flag in the *Status register (eSCI_SR)* will be set.

The application should clear the TXRDY interrupt flag *before* writing data into the *LIN transmit register (eSCI_LTR)* because the eSCI module will set the TXRDY one clock cycle after the write access.

The first data written to the *LIN transmit register (eSCI_LTR)* provides the Identifier and Identifier Parity fields. The second data written defines the number of data bytes to be transmitted. The third data written defines the CRC and checksum generation. The TD bit has to set to 1 in order to invoke the LIN TX frame generation. The value of the TO field is ignored by the eSCI module for LIN TX frames.

After the third data was written the generation of a LIN TX frame is started. Firstly, a break field is transmitted, then the synch field and the protected identifier field.

All subsequent write accesses to the *LIN transmit register (eSCI_LTR)* provide data bytes to be transmitted via the LIN bus. A data byte field will be transmitted as soon as data are available. After the last data byte, defined by the value written to the LEN field, was send out, the configured CRC and checksum fields will be send out.

After the transmission of the checksum field of the LIN TX frame, the write access counter for the *LIN transmit register (eSCI_LTR)* is reset and the FRC interrupt flag in the *Status register (eSCI_SR)* is set.

DMA controlled LIN TX frame generation

In this mode, the eSCI module controls the generation of an LIN TX Frame. When new data required for transmission, the eSCI module generates the transmit DMA request and the DMA controller delivers the required data. The application request the eSCI module to enter this mode by setting the TXDMA bit in the *Control register 2 (eSCI_CR2)*. From this point in time, the module start the generation of DMA requests and initiates the frame transmission. Before entering this mode, the application should perform the following actions:

1. Configure the module for LIN mode.
2. Enable the transmitter by setting TE in *Control register 1 (eSCI_CR1)* to 1.
3. Setup the DMA controller channel and provide frame data in system memory

A block diagram which presents an overview of the DMA Controlled LIN TX Frame is shown in *Figure 670*. The content of the fields in the memory is the same as described in *LIN transmit register (eSCI_LTR) - LIN TX frame generation*.

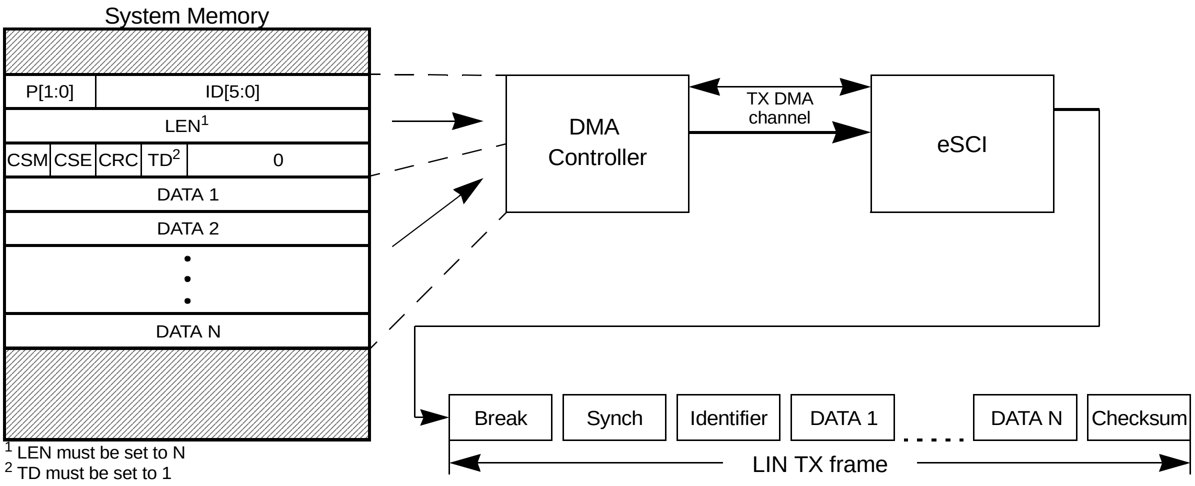


Figure 670. DMA controlled LIN TX frame generation

LIN RX frame generation

The eSCI module supports two modes of LIN RX Frame generation and reception, the CPU controlled mode and the DMA controlled mode. In the CPU controlled mode, the application provides the required data by subsequent CPU write accesses to the *LIN transmit register (eSCI_LTR)* and retrieves the received data by subsequent CPU read accesses to the *LIN receive register (eSCI_LRR)*. In the DMA controlled mode, the DMA controller provides the required frame configuration data in response to DMA requests generated by the eSCI module and transfers the received frame data to the memory in response to DMA requests generated by the eSCI module.

CPU controlled LIN RX frames generation

In this mode, the application initiates the generation of an LIN RX Frame by a sequence of subsequent CPU write accesses to the *LIN transmit register (eSCI_LTR)*. When the eSCI module has processed the data written into *LIN transmit register (eSCI_LTR)*, the TXRDY interrupt flag in the *Status register (eSCI_SR)* will be set.

The application should clear the TXRDY interrupt flag *before* writing data into the *LIN transmit register (eSCI_LTR)* because the eSCI module will set the TXRDY one clock cycle after the write access.

The first data written to the *LIN transmit register (eSCI_LTR)* provides the Identifier and Identifier Parity fields. The second data written defines the number of data bytes requested from the LIN slave. The third data written defines the CRC and checksum generation. The TD bit has to set to 0 to invoke the RX frame generation. The TO field defines the upper part of the timeout value. The fourth byte written defines the lower part of the timeout value.

After the fourth byte was written the generation of a LIN RX frame is started. Firstly, a break field is transmitted, then the synch field and the protected identifier field. After the transmission of the protected identifier, the eSCI module starts to receive the frame data transmitted by the LIN slave. When the module has received a complete byte field, the received data are transferred into the *LIN receive register (eSCI_LRR)* and the receive data ready flag RXRDY in the *Status register (eSCI_SR)* is set.

The application can retrieve the received data by subsequent read access from *LIN receive register (eSCI_LRR)* after checking the RXRDY flag. The application should clear the RXRDY flag immediately after reading the *LIN receive register (eSCI_LRR)*.

After the reception of the configured number of data from the slave, the module starts the reception of the configured CRC and Checksum byte fields. These data are not transferred into the *LIN receive register (eSCI_LRR)*. The CRC and Checksum checking is performed internally. In case of errors, they will be reported as described in *Section , LIN error reporting*.

After the reception of the checksum field of the LIN RX frame, the FRC interrupt flag in the *Status register (eSCI_SR)* is set.

DMA controlled LIN RX frames generation

In this mode, the eSCI module controls the generation of LIN RX frame header and the reception of the frame data automatically and utilizes the two connected DMA channels. A block diagram which presents an overview of the DMA Controlled LIN RX Frame generation and reception is shown in *Figure 670*. The content of the header fields in the memory is the same as described in *LIN transmit register (eSCI_LTR) - LIN RX frame generation*. The TX DMA channel is used the fetch the LIN RX frame header and control information. The RX DMA channel is used to transfer the received frame data into the memory.

When new data required for transmission, the module generates the transmit DMA request and the DMA controller delivers the required data. When new data was received, the module generates the receive DMA request and the DMA controller retrieves the provided data.

The application request the eSCI module to enter this mode by setting the RXDMA bit in the *Control register 2 (eSCI_CR2)*. From this point in time, the module start the generation of DMA requests and frame transmission and reception. Before entering this mode, the application should perform the following actions:

1. Configure the module for LIN mode.
2. Enable transmitter and receiver by setting TE and RE in *Control register 1 (eSCI_CR1)* to 1.
3. Setup the two DMA controller channels and provide frame header data in system memory.

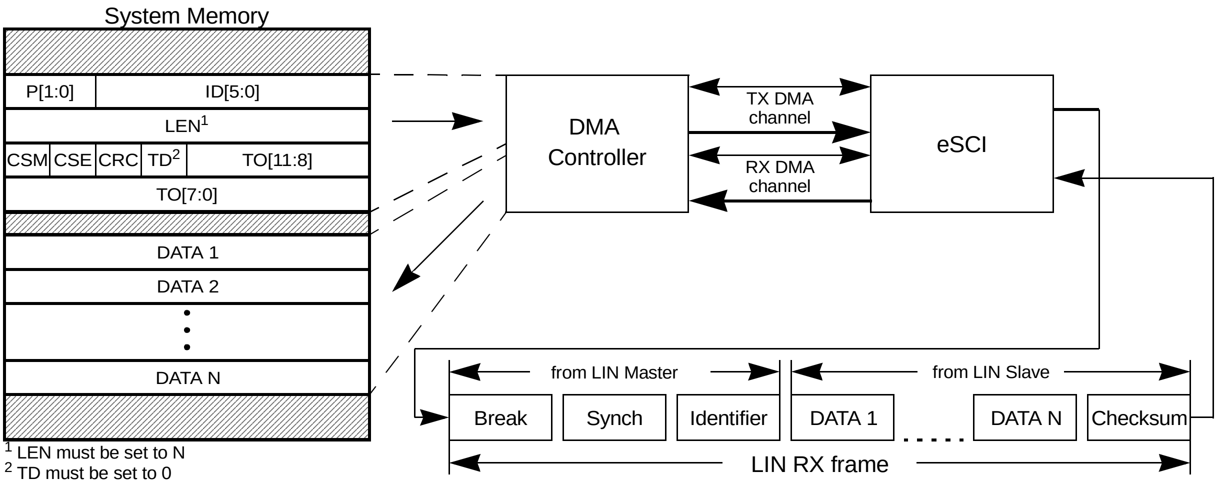


Figure 671. DMA controlled LIN RX frame generation and reception

LIN error reporting

This section describes error checking and the signaling of detected errors in LIN mode.

Physical bus error detection

If the receiver input is sampled 0 for at least 31 sample clock cycles after the start of the transmission of a LIN frame, the physical bus error flag PBERR in the *Status register (eSCI_SR)* will be set.

Unrequested activity detection

If an unrequested byte is received (i.e. a byte which is not part of an RX frame) which is not recognized as a wake-up or break character, the bit error flag BERR in the *Status register (eSCI_SR)* is set. In addition the RXRDY flag will also be set, the LINRX register must be read before normal operations can proceed.

Standard bit error detection

The standard bit error detection is enabled when the fast bit error detection control bit FBR in the *Control register 2 (eSCI_CR2)* is 0. The standard bit error detection is performed after each LIN byte field transmission.

During the transmission of the LIN frame header and LIN frame data, the receiver is running and receives the signal values on the serial bus. After the complete transmission and the related reception of a LIN byte field, the eSCI compares the data that was transmitted and the data that has been received. If they do not match, the bit error interrupt flag BERR in the *Status register (eSCI_SR)* is set.

Fast bit error detection

Fast Bit Error Detection has been designed to allow flagging of LIN bit errors while they occur, rather than flagging them after a byte transmission has completed (see *Figure 672*).

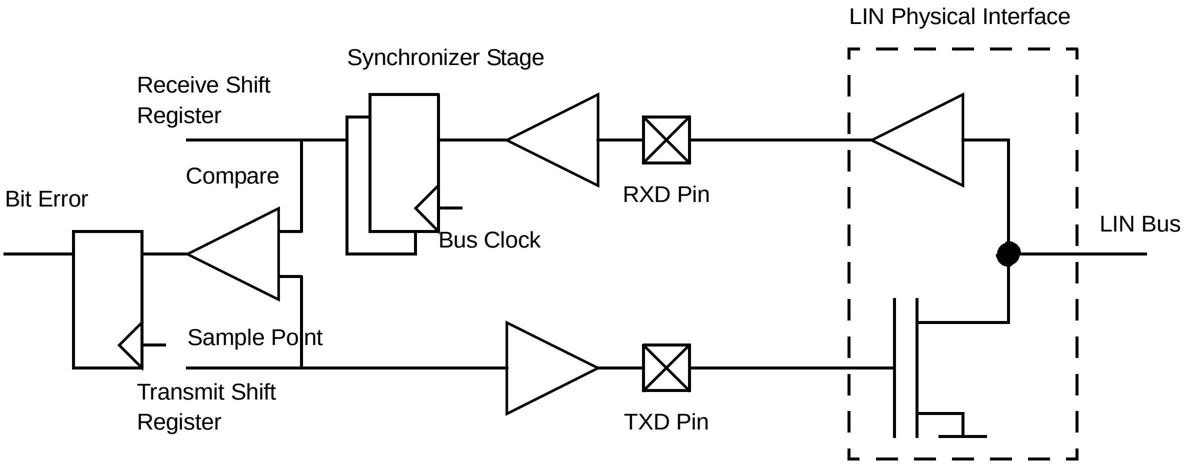


Figure 672. Fast bit error detection on a LIN bus

If fast bit error detection bit FBR in the *Control register 2 (eSCI_CR2)* is set the eSCI will compare the transmitted and the received data stream while the transmitter is active (not idle). Once a mismatch between the transmitted data and the received data is detected the following actions are performed the bit error flag BERR will be set.

To adjust to different bus loads the sample point at which the incoming bit is compared to the one which was transmitted can be selected with the BESM bit in the *Control register 2 (eSCI_CR2)*. If eSCI_CR2[BESM] = 1, the comparison will be performed with sample RS13, otherwise with RS9 (see *Figure 673*) (also see *Section , Bit sampling*).

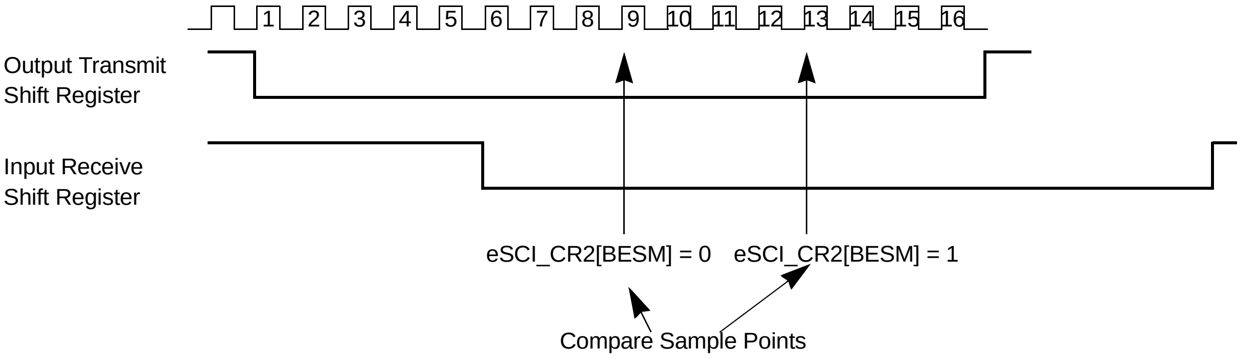


Figure 673. Timing diagram fast bit error detection

Note: To calculate the exact position of the sample point with regard to the RX pin, the delays through the pads and the two Bus Clock cycle delay through the input synchronizer also needs to be taken into account.

Slave-Not-Responding-Error detection

The Slave-Not-Responding-Error is defined in *LIN Specification Package Revision 1.3; December 12, 2002*; 6 ERROR AND EXCEPTION HANDLING. The LIN specification requires that a NO_RESPONSE_ERROR has to be detected if a message frame is not fully completed within the maximum length T_{FRAME_MAX} by any slave task upon transmission of the SYNCH and IDENTIFIER fields. The maximum frame length TFRAME_MAX is defined in *LIN Specification Package Revision 1.3; December 12, 2002*; 3.3 LENGTH OF MESSAGE FRAME AND BUS SLEEP DETECT, as

Equation 43

$$T_{FRAME\_MAX} = (10 \cdot N_{DATA} + 45) \cdot 1.4$$

where N_{DATA} is the number of data byte fields of the message frame.

The STO interrupt flag in the *Status register (eSCI_SR)* will be set, if an LIN RX frame was not fully received in the amount of time specified in the timeout value field TO in the *LIN transmit register (eSCI_LTR)*. The time period starts with the falling edge of the transmitted LIN break character and is specified in units of transmit bits.

To achieve LIN compliant Slave-Not-Responding-Error detection, the timeout value TO in the *LIN transmit register (eSCI_LTR)* field has to be set to T_{FRAME_MAX} when a LIN RX frame is initiated.

Checksum error detection

If the checksum enable bit CSE in the *LIN transmit register (eSCI_LTR)* was set, the checksum checking is performed based on the received checksum byte. The checksum mode is selected by the CSM bit in the *LIN transmit register (eSCI_LTR)*. If the value received in the checksum bytes did not match the calculated checksum, the checksum error flag CKERR in the *Status register (eSCI_SR)* will be set.

CRC error detection

The CRC checking is performed on the two received CRC bytes CRC1 and CRC2 if the CRC Enhanced LIN frame format was selected by the CRC bit in the *LIN transmit register (eSCI_LTR)*. If the value received in the two CRC bytes did not match the calculated CRC pattern, the CRC error flag CERR in the *Status register (eSCI_SR)* will be set.

Overflow detection

When the receiver has received the next byte field, which should be transferred into the *LIN receive register (eSCI_LRR)*, but neither the application nor the RX DMA channel have read data from this register since the last update, the received data overflow flag OVFL in the *Status register (eSCI_SR)* will be set. In this case the content of the *LIN receive register (eSCI_LRR)* is not changed. The data received most recently are lost.

LIN wake up

The section describes the LIN Wake Up behavior of the eSCI module.

LIN Wake-Up request generation

The eSCI module can cause the LIN bus to exit the sleep mode by sending a wake-up signal frame, which consists of a wake-up signal 0x80 (consisting of 8 dominant bits followed by 1 recessive bit), followed by the wake-up delimiter period as defined by the WUD0 and WUD1 fields in the *LIN control register (eSCI_LCR)*.

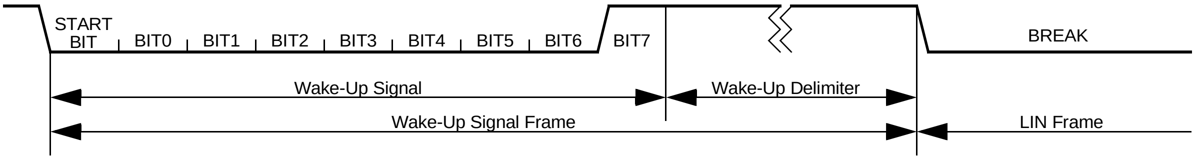


Figure 674. LIN Wake-Up Signal Frame

The application triggers the transmission of a wake-up signal frame by writing 1 to the LIN bus wake-up trigger WU in the *LIN control register (eSCI_LCR)*.

The LIN Specification 2.0 requires the generation of LIN wake-up signals as dominant pulses longer than 250 µs and shorter than 5 ms. To achieve this, the eSCI module has to programmed to a baud rate between 32 kBaud and 1.6 kBaud. With each of these baud rate settings, the wake-up signal is transmitted as a dominant pulse longer than 250 µs and shorter than 5 ms.

LIN Wake-Up request detection

The eSCI module detects a LIN wake-up requests when

e) one of the characters 0x00, 0x80, or 0xC0 has been received,

f) followed by zero or more low bits,

g) followed by at least one high bit, and

h) no LIN frame transmission or reception is started or running during the reception above

If a LIN wake-up request has been detected, the LIN wake-up flag LWAKE in the *Status register (eSCI_SR)* will be set after the reception of the first high bit.

The LIN Specification 2.0 requires the detection of LIN wake-up requests as dominant pulses longer than 150 µs. To achieve this, the eSCI module has to programmed to the maximum baud rate that is not greater than 43.77 kBaud. With this baud rate setting, any dominant pulse longer than 150 µs is decoded as at least 7 dominant bits (one start and 6 data bits) and consequently as one of the characters 0xC0, 0x80, or 0x00.

LIN protocol engine stop and reset

The LIN protocol engine is stopped and reset when the application set the LRES control bit in the *LIN control register (eSCI_LCR)* to 1. In this case, the LIN protocol engine will stop immediately. No new transmissions or receptions are initiated, the LIN serial bus is driven with the recessive value 1. Additionally to the stop and reset of the LIN protocol engine the receiver and transmitter modules are stopped and reset as well, and the receive and transmit DMA requests are deasserted.

In order to start the LIN Protocol Engine with idle transmitter and receiver processes, the LRES bit should be asserted until all of the status bits TACT, and RAF in the *Status register (eSCI_SR)* are cleared. Theses status bits are cleared within one bit time after assertion of the LRES bit.

28.4.7 Interrupts

This section describes the interrupt sources and interrupt request generation.

Interrupt flags and enables

All interrupt sources, interrupt flags, and interrupt enable bits are listed in *Table 671*. This table indicates the operational modes, where the interrupt flags can be set by the eSCI module.

Table 671. eSCI iterrupt flags and interrupt enable bits

Interrupt Source	Operational Mode	Interrupt Flag	Interrupt Enable Bit
Transmitter	SCI	eSCI_SR[TDRE]	eSCI_CR1[TIE]
Transmitter	SCI, LIN	eSCI_SR[TC]	eSCI_CR1[TCIE]
Receiver	SCI	eSCI_SR[RDRF]	eSCI_CR1[RIE]
Receiver	SCI	eSCI_SR[IDLE]	eSCI_CR1[ILIE]
Receiver	SCI	eSCI_SR[OR]	eSCI_CR2[ORIE]
Receiver	SCI, LIN	eSCI_SR[NF]	eSCI_CR2[NFIE]
Receiver	SCI, LIN	eSCI_SR[FE]	eSCI_CR2[FEIE]
Receiver	SCI	eSCI_SR[PF]	eSCI_CR2[PFIE]
Receiver	LIN	eSCI_SR[BERR]	eSCI_CR2[BERRIE]
Receiver	LIN	eSCI_SR[RXRDY]	eSCI_LCR[RXIE]
Transmitter	LIN	eSCI_SR[TXRDY]	eSCI_LCR[TXIE]
Receiver	LIN	eSCI_SR[LWAKE]	eSCI_LCR[WUIE]
Receiver	LIN	eSCI_SR[STO]	eSCI_LCR[STIE]
Receiver	LIN	eSCI_SR[PBERR]	eSCI_LCR[PBIE]
Receiver	LIN	eSCI_SR[CERR]	eSCI_LCR[CIE]
Receiver	LIN	eSCI_SR[CKERR]	eSCI_LCR[CKIE]
Receiver	LIN	eSCI_SR[FRC]	eSCI_LCR[FCIE]
Receiver	LIN	eSCI_SR[UREQ]	eSCI_LCR[UQIE]
Transmitter, Receiver	LIN	eSCI_SR[OVFL]	eSCI_LCR[OFIE]

Interrupt request generation

The eSCI module provides one hardware interrupt request signal to the systems interrupt controller. This interrupt request signal is asserted if and only if at least one of the interrupt flags and the corresponding interrupt enables are set to 1. Otherwise the interrupt line is deasserted.

28.5 Application information

28.5.1 SCI data frames separated by preamble

To separate SCI data frame with preambles with minimum idle line time, use this sequence between messages:

1. write to *SCI data register (eSCI_DR)*
   - this sets the internal iCMT bit which requests the data transmission
2. wait until TDRE in *Status register (eSCI_SR)* is set
   - this indicates the start of transmission; the iCMT bit was cleared
3. clear and subsequently set the TE bit in *Control register 1 (eSCI_CR1)*
   - this set the internal iPRE bit which requests the preamble transmission
4. write to *SCI data register (eSCI_DR)*
   - this sets the internal iCMT bit which requests the data transmission

The priority scheme of the transmitter which is described in *Table 663* ensures, that the preamble is transmitted before the data frame.

29 FlexCAN Module

29.1 Information specific to this device

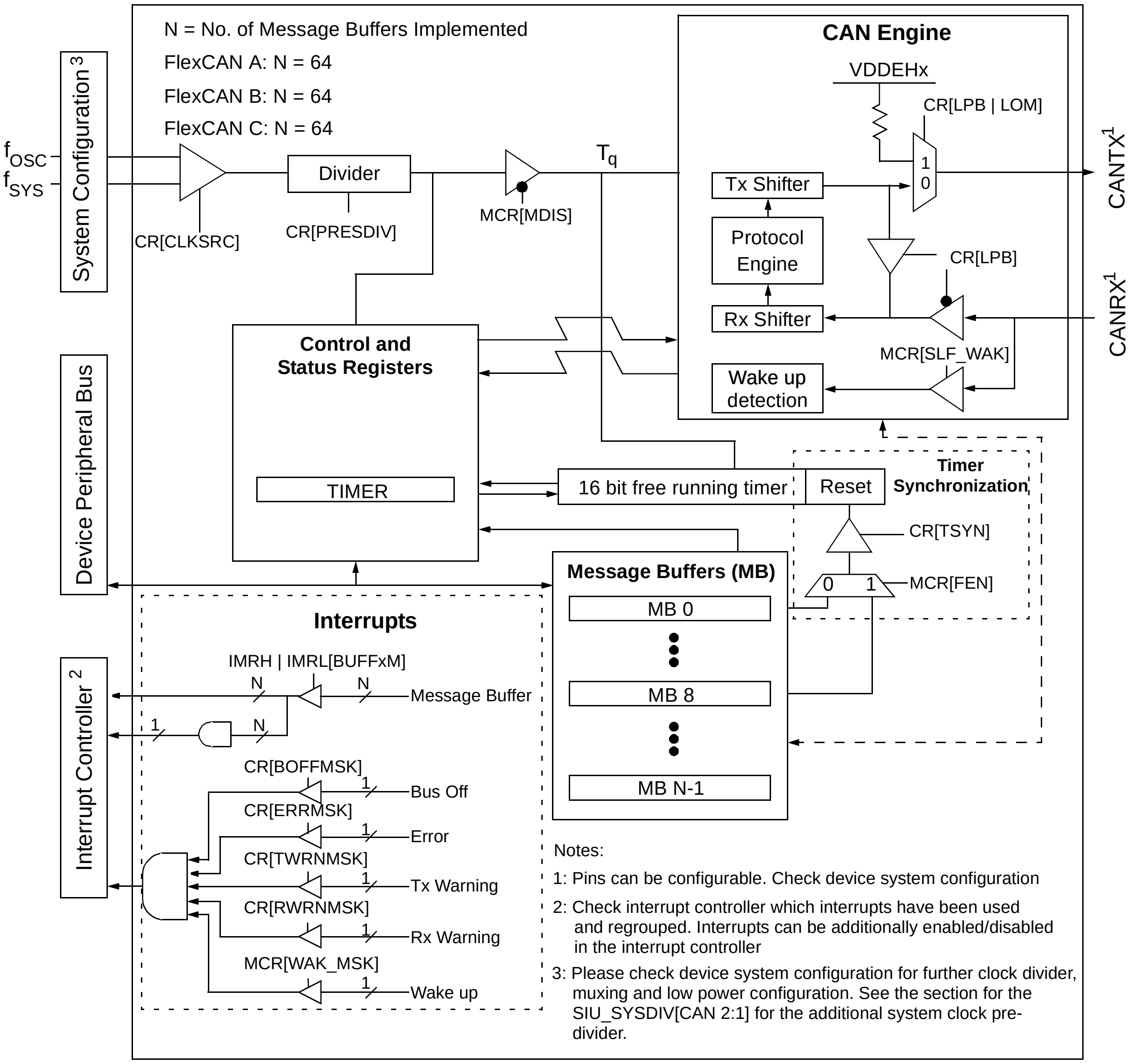
This section presents device-specific parameterization and customization information not specifically referenced in the remainder of this chapter.

29.1.1 Device-specific features

- 3 FlexCAN modules with 64 message buffers each
- The device has 3 Controller Area Network (FlexCAN) blocks referred as FlexCAN_A, FlexCAN_B and FlexCAN_C.
  - Each FlexCAN module contains an embedded memory capable of storing 64 Message Buffers (MB).
  - Although the FlexCAN module provides a differentiation between Supervisor and User access types, all accesses will be always considered of the Supervisor type. As a consequence, the SUPV bit in the Module Configuration Register (MCR) Register has no effect on the module behavior.
  - All 4 FlexCAN functional modes are supported: Normal, Freeze, Listen-Only and Loop-Back.
  - Only two power modes are supported, the Disable and Stop Mode. Doze Mode is not supported.
  - Low-pass filter (glitch filter)

29.2 Introduction

The FlexCAN module is a communication controller implementing the CAN protocol according to the CAN 2.0B protocol specification. The block diagram in *Figure 675* shows the main sub-blocks implemented in the FlexCAN module. Support for up to 64 Message Buffers is provided. The functions of the submodules are described in subsequent sections.



Notes:

1: Pins can be configurable. Check device system configuration

2: Check interrupt controller which interrupts have been used and regrouped. Interrupts can be additionally enabled/disabled in the interrupt controller

3: Please check device system configuration for further clock divider, muxing and low power configuration. See the section for the SIU_SYSDIV[CAN 2:1] for the additional system clock pre-divider.

Figure 675. FlexCAN block diagram

29.2.1 Overview

The CAN protocol was primarily, but not only, designed to be used as a vehicle serial data bus, meeting the specific requirements of this field: real-time processing, reliable operation in the EMI environment of a vehicle, cost-effectiveness and required bandwidth. The FlexCAN module is a full implementation of the CAN protocol specification, Version 2.0 B, which supports both standard and extended message frames. A flexible number of Message Buffers (16, 32 or 64) is also supported. The Message Buffers are stored in an embedded RAM dedicated to the FlexCAN module.

The CAN Protocol Interface (CPI) submodule manages the serial communication on the CAN bus, requesting RAM access for receiving and transmitting message frames, validating received messages and performing error handling. The Message Buffer Management (MBM) submodule handles Message Buffer selection for reception and transmission, taking care of arbitration and ID matching algorithms. The Bus Interface Unit (BIU) submodule controls the access to and from the internal interface bus, in order to establish connection to the CPU and to other blocks. Clocks, address and data buses, interrupt outputs and test signals are accessed through the Bus Interface Unit.

A typical CAN system is shown below in *Figure 676*. Each CAN station is connected physically to the CAN bus through a transceiver. The transceiver provides the transmit drive, waveshaping, and receive/compare functions required for communicating on the CAN bus. It can also provide protection against damage to the FlexCAN caused by a defective CAN bus or defective stations.

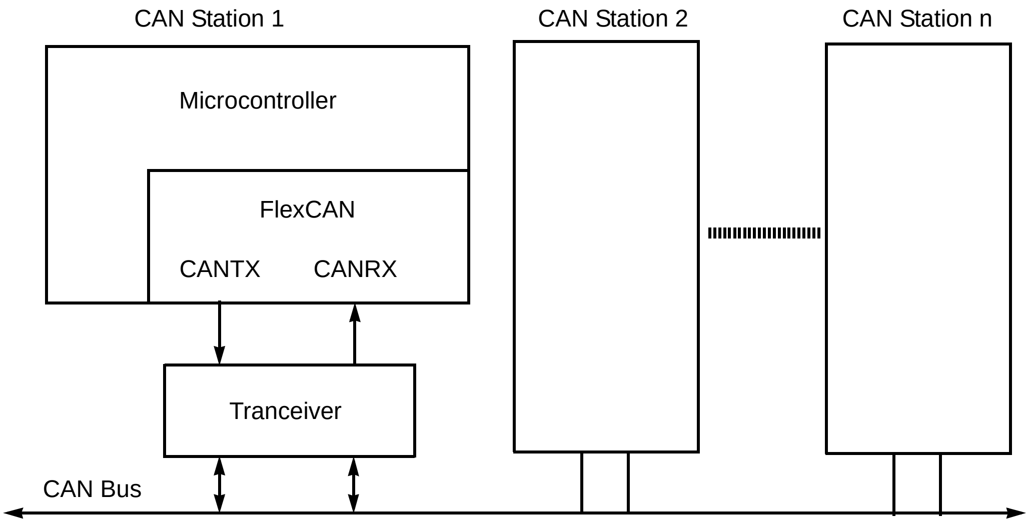


Figure 676. Typical CAN system

29.2.2 FlexCAN module features

The FlexCAN module includes these distinctive features:

- Full Implementation of the CAN protocol specification, Version 2.0B
  - Standard data and remote frames
  - Extended data and remote frames
  - Zero to eight bytes data length
  - Programmable bit rate up to 1 Mbit/s
  - Content-related addressing
- Flexible Message Buffers (up to 64) of zero to eight bytes data length
- Each message buffer configurable as Rx or Tx, all supporting standard and extended messages
- Individual Rx Mask Registers per Message Buffer
- Includes either 1056 bytes (64 message buffers), 544 bytes (32 message buffers) or 288 bytes (16 message buffers) of RAM used for message buffer storage
- Includes either 256 bytes (64 message buffers), 128 bytes (32 message buffers) or 64 bytes (16 message buffers) of RAM used for individual Rx Mask Registers
- Full featured Rx FIFO with storage capacity for 6 frames and internal pointer handling
- Powerful Rx FIFO ID filtering, capable of matching incoming IDs against either 8 extended, 16 standard or 32 partial (8 bits) IDs, with individual masking capability
- Selectable backwards compatibility with previous FlexCAN version
- Programmable clock source to the CAN Protocol Interface, either bus clock or crystal oscillator
- Unused message buffer and Rx Mask Register space can be used as general purpose RAM space
- Listen only mode capability
- Programmable loop-back mode supporting self-test operation
- Programmable transmission priority scheme: lowest ID, lowest buffer number or highest priority
- Time Stamp based on 16-bit free-running timer
- Global network time, synchronized by a specific message
- Maskable interrupts
- Independent of the transmission medium (an external transceiver is assumed)
- Short latency time due to an arbitration scheme for high-priority messages
- Low power modes, with programmable wake up on bus activity

29.2.3 Modes of operation

The FlexCAN module has four functional modes: Normal Mode (User and Supervisor), Freeze Mode, Listen-Only Mode and Loop-Back Mode. There are also two low power modes: Disable Mode and Stop Mode.

- Normal Mode (User or Supervisor):

  In Normal Mode, the module operates receiving and/or transmitting message frames, errors are handled normally and all the CAN Protocol functions are enabled. User and Supervisor Modes differ in the access to some restricted control registers.
- Freeze Mode:

  It is enabled when MCR[FRZ] is asserted. If enabled, Freeze Mode is entered when MCR[HALT] is set or when Debug Mode is requested at MCU level. In this mode, no transmission or reception of frames is done and synchronicity to the CAN bus is lost. See *Section , Freeze Mode* for more information.
- Listen-Only Mode:

  The module enters this mode when CR[LOM] is asserted. In this mode, transmission is disabled, all error counters are frozen and the module operates in a CAN Error Passive mode. Only messages acknowledged by another CAN station will be received. If FlexCAN detects a message that has not been acknowledged, it will flag a BIT0 error (without changing the REC), as if it was trying to acknowledge the message.
- Loop-Back Mode:

  The module enters this mode when CR[LPB] is asserted. In this mode, FlexCAN performs an internal loop back that can be used for self test operation. The bit stream output of the transmitter is internally fed back to the receiver input. The Rx CAN input pin is ignored and the Tx CAN output goes to the recessive state (logic ‘1’). FlexCAN behaves as it normally does when transmitting and treats its own transmitted message as a message received from a remote node. In this mode, FlexCAN ignores the bit sent during the ACK slot in the CAN frame acknowledge field to ensure proper reception of its own message. Both transmit and receive interrupts are generated.
- Module Disable Mode:

  This low power mode is entered when the MDIS bit in the MCR Register is asserted by the CPU. When disabled, the module sends a request to disable the clocks to the CAN Protocol Interface and Message Buffer Management sub-modules. Exit from this mode is done by negating MCR[MDIS]. See *Section , Module Disable Mode*, for more information.
- Stop Mode:

  This low power mode is entered when Stop Mode is requested at MCU level. When in Stop Mode, the module puts itself in an inactive state and then informs the CPU that the clocks can be shut down globally. Exit from this mode happens when the Stop Mode request is removed or when activity is detected on the CAN bus and the Self Wake Up mechanism is enabled. See *Section , Stop Mode*, for more information.

29.3 External signal description

29.3.1 Overview

The FlexCAN module has two I/O signals connected to the external MCU pins. These signals are summarized in *Table 672* and described in more detail in the next subsections.

Table 672. FlexCAN signals

Signal name[1]	Direction	Description
CAN_*x*_RX	Input	CAN receive pin
CAN_*x*_TX	Output	CAN transmit pin

1. The actual MCU pins may have different names.

29.3.2 Signal descriptions

CAN RX

This pin is the receive pin from the CAN bus transceiver. Dominant state is represented by logic level ‘0’. Recessive state is represented by logic level ‘1’.

CAN TX

This pin is the transmit pin to the CAN bus transceiver. Dominant state is represented by logic level ‘0’. Recessive state is represented by logic level ‘1’.

29.4 Memory map/Register definition

This section describes the registers and data structures in the FlexCAN module. The base address of the module depends on the particular memory map of the MCU. The addresses presented here are relative to the base address.

The address space occupied by FlexCAN has 96 bytes for registers starting at the module base address, followed by message buffer storage space in embedded RAM starting at address 0x0060, and an extra ID Mask storage space in a separate embedded RAM starting at address 0x0880.

29.4.1 FlexCAN memory mapping

The complete memory map for a FlexCAN module with 64 message buffers capability is shown in *Table 673*. Each individual register is identified by its complete name and the corresponding mnemonic. The access type can be Supervisor (S) or Unrestricted (U). Most of the registers can be configured to have either Supervisor or Unrestricted access by programming MCR[SUPV]. These registers are identified as S/U in the Access column of *Table 673*.

The IFRH and IMRH registers are considered reserved space when FlexCAN is configured with 16 or 32 message buffers. The Rx Global Mask (RXGMASK), Rx Buffer 14 Mask (RX14MASK) and the Rx Buffer 15 Mask (RX15MASK) registers are provided for backwards compatibility, and are not used when MCR[MBFEN] is asserted.

The address ranges 0x0060–0x047F and 0x0880–0x097F are occupied by two separate embedded memories. These two ranges are completely occupied by RAM (1056 and 256 bytes, respectively) only when FlexCAN is configured with 64 message buffers. When it is configured with 16 message buffers, the memory sizes are 288 and 64 bytes, so the address ranges 0x0180–0x047F and 0x08C0–0x097F are considered reserved space. When it is configured with 32 message buffers, the memory sizes are 544 and 128 bytes, so the address ranges 0x0280–0x047F and 0x0900–0x097F are considered reserved space.

Furthermore, if MCR[MBFEN] is negated, then the whole Rx Individual Mask Registers address range (0x0880–0x097F) is considered reserved space.

Table 673. Module memory map

Address	Use	Access Type	Affected by Hard Reset	Affected by Soft Reset
Base + 0x0000	Module Configuration (MCR)	S	Yes	Yes
Base + 0x0004	Control Register (CR)	S/U	Yes	No
Base + 0x0008	Free Running Timer (TIMER)	S/U	Yes	Yes
Base + 0x000C	Reserved			
Base + 0x0010	Rx Global Mask (RXGMASK)	S/U	Yes	No
Base + 0x0014	Rx Buffer 14 Mask (RX14MASK)	S/U	Yes	No
Base + 0x0018	Rx Buffer 15 Mask (RX15MASK)	S/U	Yes	No
Base + 0x001C	Error Counter Register (ECR)	S/U	Yes	Yes
Base + 0x0020	Error and Status Register (ESR)	S/U	Yes	Yes
Base + 0x0024	Interrupt Masks 2 (IMRH)	S/U	Yes	Yes
Base + 0x0028	Interrupt Masks 1 (IMRL)	S/U	Yes	Yes
Base + 0x002C	Interrupt Flags 2 (IFRH)	S/U	Yes	Yes
Base + 0x0030	Interrupt Flags 1 (IFRL)	S/U	Yes	Yes
Base + 0x0034–0x005F	Reserved			
Base + 0x0060–0x007F	Reserved			
Base + 0x0080–0x017F	Message Buffers MB0–MB15	S/U	No	No
Base + 0x0180–0x027F	Message Buffers MB16–MB31	S/U	No	No
Base + 0x0280–0x047F	Message Buffers MB32–MB63	S/U	No	No
Base + 0x0480–087F	Reserved			
Base + 0x0880–0x08BF	Rx Individual Mask Registers RXIMR0–RXIMR15	S/U	No	No
Base + 0x08C0–0x08FF	Rx Individual Mask Registers RXIMR16–RXIMR31	S/U	No	No
Base + 0x0900–0x097F	Rx Individual Mask Registers RXIMR32–RXIMR63	S/U	No	No

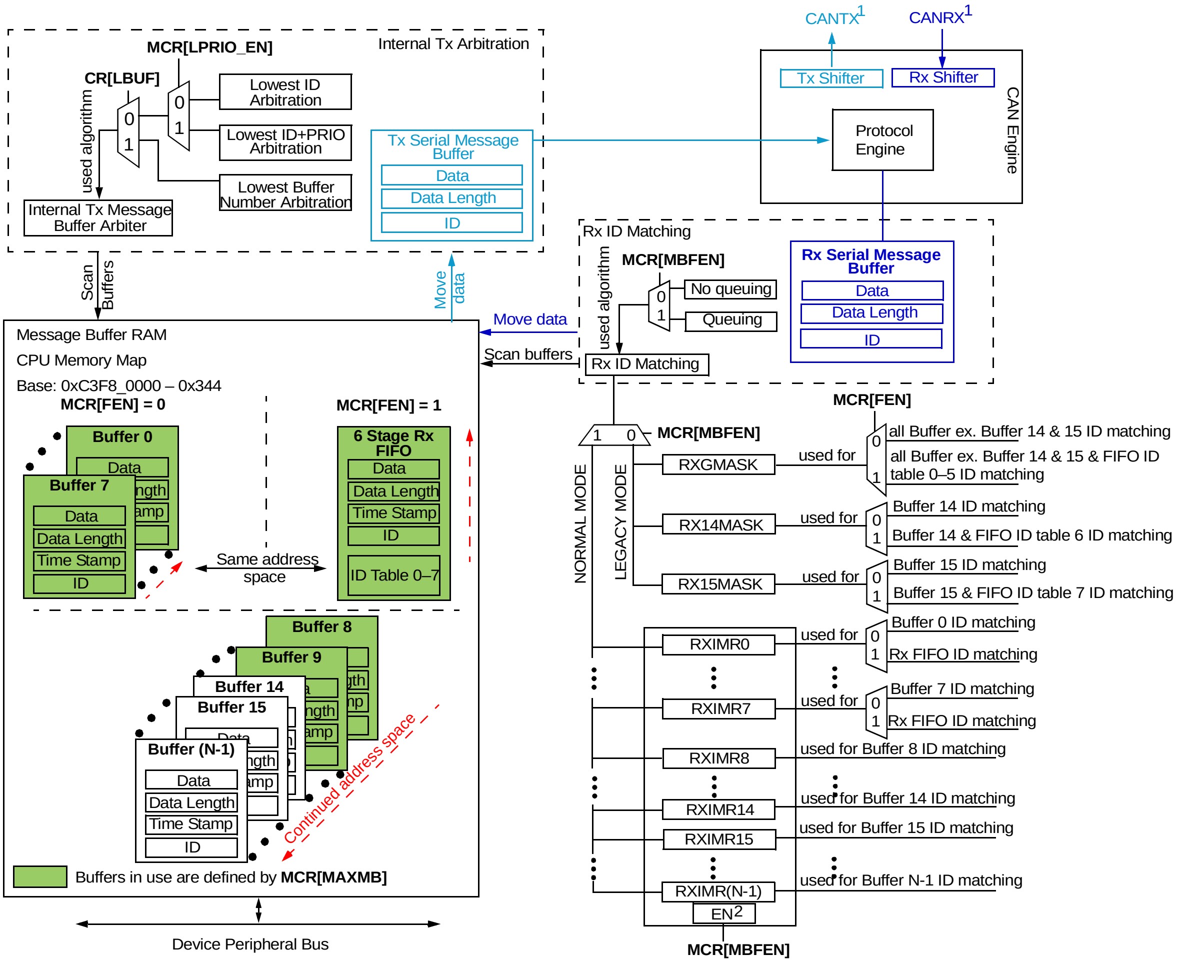
The FlexCAN module stores CAN messages for transmission and reception using a Message Buffer structure. Each individual message buffer is formed by 16 bytes mapped on memory as described in *Table 674*. *Table 674* shows a Standard/Extended Message Buffer (MB0) memory map, using 16 bytes total (0x80–0x8F space).

Table 674. Message buffer MB0 memory mapping

Address offset	MB field
0x80	Control and Status (C/S)
0x84	Identifier Field
0x88–0x8F	Data Field 0 – Data Field 7 (1 byte each)

29.4.2 Message buffer architecture

The message buffer architecture is shown in *Figure 677*.



Notes:
1: Pins can be configurable. Check device system configuration.
2: If disabled the RXIMRx registers are not memory mapped.
Any access in this case will cause data access error.

Figure 677. FlexCAN message buffer architecture

29.4.3 Message buffer structure

The Message Buffer structure used by the FlexCAN module is represented in *Figure 678*. Both Extended and Standard Frames (29-bit Identifier and 11-bit Identifier, respectively) used in the CAN specification (Version 2.0 Part B) are supported. The buffer is a 4-word (128-bit) structure summarized in *Figure 678*.

Figure 678. Message Buffer Structure

<table>
<tr><th></th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th><th>16</th><th>17</th><th>18</th><th>19</th><th>20</th><th>21</th><th>22</th><th>23</th><th>24</th><th>25</th><th>26</th><th>27</th><th>28</th><th>29</th><th>30</th><th>31</th></tr>
<tr><td>0x0</td><td colspan="4">▨</td><td colspan="4">CODE</td><td>▨</td><td>SRR</td><td>IDE</td><td>RTR</td><td colspan="4">LENGTH</td><td colspan="16">TIME STAMP</td></tr>
<tr><td>0x4</td><td colspan="3">PRIO</td><td colspan="11">ID (Standard/Extended)</td><td colspan="18">ID (Extended)</td></tr>
<tr><td>0x8</td><td colspan="8">Data Byte 0</td><td colspan="8">Data Byte 1</td><td colspan="8">Data Byte 2</td><td colspan="8">Data Byte 3</td></tr>
<tr><td>0xC</td><td colspan="8">Data Byte 4</td><td colspan="8">Data Byte 5</td><td colspan="8">Data Byte 6</td><td colspan="8">Data Byte 7</td></tr>
</table>

▨ = Unimplemented or Reserved

Table 675. Message Buffer Structure (Word 0—0x0)

Field	Description
CODE	This 4-bit field can be accessed (read or write) by the CPU and by the FlexCAN module itself, as part of the message buffer matching and arbitration process. The encoding for Rx buffers is shown in *Table 676* and the encoding for Tx buffers is shown in *Table 677*. See *Section 29.5, Functional description* for additional information.
SRR	Substitute Remote Request Fixed recessive bit, used only in extended format. It must be set to ‘1’ by the user for transmission (Tx Buffers) and will be stored with the value received on the CAN bus for Rx receiving buffers. It can be received as either recessive or dominant. If FlexCAN receives this bit as dominant, then it is interpreted as arbitration loss. 1: Recessive value is compulsory for transmission in Extended Format frames 0: Dominant is not a valid value for transmission in Extended Format frames
IDE	ID Extended Bit This bit identifies whether the frame format is standard or extended. 1: Frame format is extended 0: Frame format is standard
RTR	Remote Transmission Request This bit is used for requesting transmissions of a data frame. If FlexCAN transmits this bit as ‘1’ (recessive) and receives it as ‘0’ (dominant), it is interpreted as arbitration loss. If this bit is transmitted as ‘0’ (dominant), then if it is received as ‘1’ (recessive), the FlexCAN module treats it as bit error. If the value received matches the value transmitted, it is considered as a successful bit transmission. 1: Indicates the current message buffer has a Remote Frame to be transmitted 0: Indicates the current message buffer has a Data Frame to be transmitted
LENGTH	Length of Data in Bytes This 4-bit field is the length (in bytes) of the Rx or Tx data, which is located in offset 0x8 through 0xF of the message buffer space (see *Figure 678*). In reception, this field is written by the FlexCAN module, copied from the DLC (Data Length Code) field of the received frame. In transmission, this field is written by the CPU and corresponds to the DLC field value of the frame to be transmitted. When RTR=1, the Frame to be transmitted is a Remote Frame and does not include the data field, regardless of the Length field.

Table 675. Message Buffer Structure (Word 0—0x0)

Field	Description
TIME STAMP	Free-Running Counter Time Stamp This 16-bit field is a copy of the Free-Running Timer, captured for Tx and Rx frames at the time when the beginning of the Identifier field appears on the CAN bus.
PRIO	Local priority This 3-bit field is only used when MCR[LPRIO_EN] is set and it only makes sense for Tx buffers. These bits are not transmitted. They are appended to the regular ID to define the transmission priority. See *Section 29.5.3, Arbitration process*.
ID	Frame Identifier In Standard Frame format, only the 11 most significant bits (3 to 13) are used for frame identification in both receive and transmit cases. The 18 least significant bits are ignored. In Extended Frame format, all bits are used for frame identification in both receive and transmit cases.
DATA	Data Field Up to eight bytes can be used for a data frame. For Rx frames, the data is stored as it is received from the CAN bus. For Tx frames, the CPU prepares the data field to be transmitted within the frame.

Table 676. Message Buffer Code for Rx buffers

Rx Code BEFORE Rx New Frame	Description	Rx Code AFTER Rx New Frame	Comment
0000	INACTIVE: buffer is not active.	—	MB does not participate in the matching process.
0100	EMPTY: buffer is active and empty.	0010	MB participates in the matching process. When a frame is received successfully, the code is automatically updated to FULL.
0010	FULL: buffer is full.	0010	The act of reading the C/S word followed by unlocking the MB does not make the code return to EMPTY. It remains FULL. If a new frame is written to the MB after the C/S word was read and the MB was unlocked, the code still remains FULL.
		0110	If the MB is FULL and a new frame is overwritten to this MB before the CPU had time to read it, the code is automatically updated to OVERRUN. Refer to *Section 29.5.5, Matching process* for details about overrun behavior.
0110	OVERRUN: a frame was overwritten into a full buffer.	0010	If the code indicates OVERRUN but the CPU reads the C/S word and then unlocks the MB, when a new frame is written to the MB the code returns to FULL.
		0110	If the code already indicates OVERRUN, and yet another new frame must be written, the MB will be overwritten again, and the code will remain OVERRUN. Refer to *Section 29.5.5, Matching process* for details about overrun behavior.

Table 676. Message Buffer Code for Rx buffers

Rx Code BEFORE Rx New Frame	Description	Rx Code AFTER Rx New Frame	Comment
0XY1[1]	BUSY: Flexcan is updating the contents of the MB. The CPU must not access the MB.	0010	An EMPTY buffer was written with a new frame (XY was 01).
		0110	A FULL/OVERRUN buffer was overwritten (XY was 11).

1. Note that for Tx MBs (see *Table 677*), the BUSY bit should be ignored upon read, except when MCR[AEN] is set.

Table 677. Message Buffer Code for Tx buffers

RTR	Initial Tx code	Code after successful transmission	Description
X	1000	—	INACTIVE: MB does not participate in the arbitration process.
X	1001	—	ABORT: MB was configured as Tx and CPU aborted the transmission. This code is only valid when MCR[AEN] is asserted. MB does not participate in the arbitration process.
0	1100	1000	Transmit data frame unconditionally once. After transmission, the MB automatically returns to the INACTIVE state.
1	1100	0100	Transmit remote frame unconditionally once. After transmission, the MB automatically becomes an Rx MB with the same ID.
0	1010	1010	Transmit a data frame whenever a remote request frame with the same ID is received. This MB participates simultaneously in both the matching and arbitration processes. The matching process compares the ID of the incoming remote request frame with the ID of the MB. If a match occurs this MB is allowed to participate in the current arbitration process and the Code field is automatically updated to ‘1110’ to allow the MB to participate in future arbitration runs. When the frame is eventually transmitted successfully, the Code automatically returns to ‘1010’ to restart the process again.
0	1110	1010	This is an intermediate code that is automatically written to the MB by the MBM as a result of match to a remote request frame. The data frame will be transmitted unconditionally once and then the code will automatically return to ‘1010’. The CPU can also write this code with the same effect.

29.4.4 Rx FIFO structure

When MCR[FEN] is set, the memory area from 0x80 to 0xFC (which is normally occupied by MBs 0 to 7) is used by the reception FIFO engine. *Figure 679* shows the Rx FIFO data structure. The region 0x80–0x8C contains a message buffer structure which is the port through which the CPU reads data from the FIFO (the oldest frame received and not read yet). The region 0x90–0xDC is reserved for internal use of the FIFO engine. The region 0xE0–0xFC contains an 8-entry ID table that specifies filtering criteria for accepting frames into the FIFO. *Figure 680* shows the three different formats that the elements of the ID table can assume, depending on field MCR[IDAM]. Note that all elements of the table must have the same format. See *Section 29.5.7, Rx FIFO* for more information.

Figure 679. Rx FIFO Structure

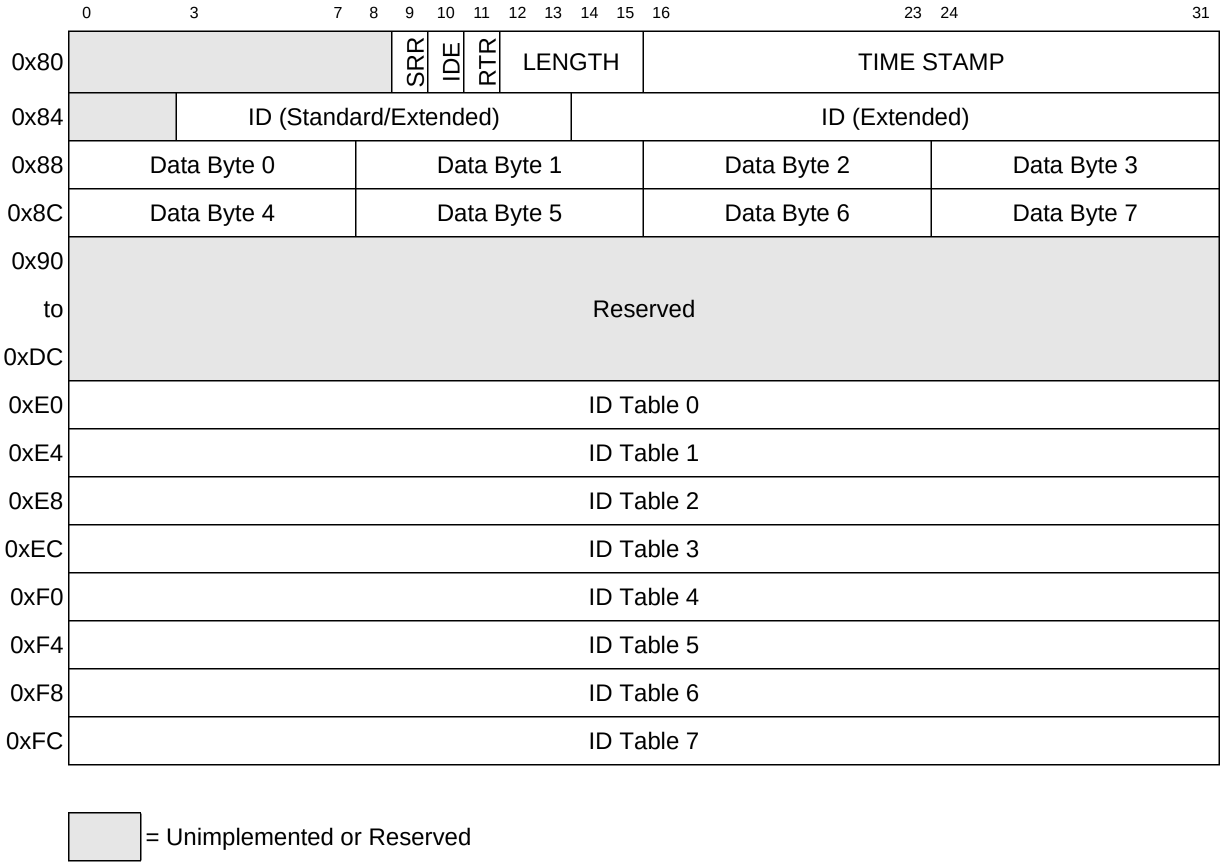


Figure 680. ID Table 0 – 7

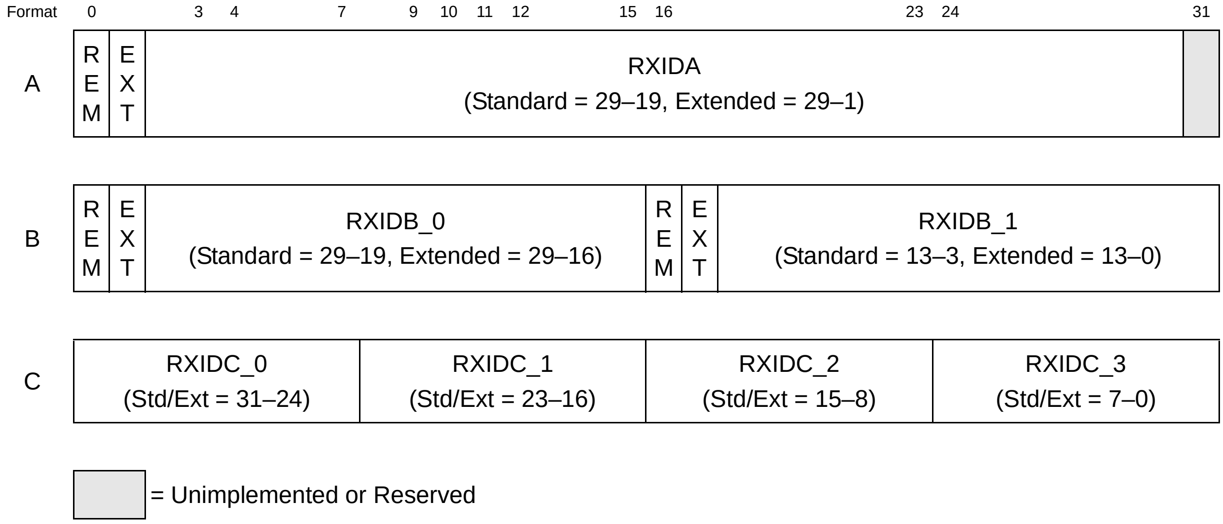


Table 678. Rx FIFO Structure

Field	Description
REM	Remote Frame This bit specifies if Remote Frames are accepted into the FIFO if they match the target ID. 1: Remote Frames can be accepted and data frames are rejected 0: Remote Frames are rejected and data frames can be accepted
EXT	Extended Frame Specifies whether extended or standard frames are accepted into the FIFO if they match the target ID. 1: Extended frames can be accepted and standard frames are rejected 0: Extended frames are rejected and standard frames can be accepted
RXIDA	Rx Frame Identifier (Format A) Specifies an ID to be used as acceptance criteria for the FIFO. In the standard frame format, only the 11 most significant bits (3 to 13) are used for frame identification. In the extended frame format, all bits are used.
RXIDB_0, RXIDB_1	Rx Frame Identifier (Format B) Specifies an ID to be used as acceptance criteria for the FIFO. In the standard frame format, the 11 most significant bits (a full standard ID) (3 to 13)are used for frame identification. In the extended frame format, all 14 bits of the field are compared to the 14 most significant bits of the received ID.
RXIDC_0, RXIDC_1, RXIDC_2, RXIDC_3	Rx Frame Identifier (Format C) Specifies an ID to be used as acceptance criteria for the FIFO. In both standard and extended frame formats, all 8 bits of the field are compared to the 8 most significant bits of the received ID.

29.4.5 Register descriptions

The FlexCAN registers are described in this section in ascending address order.

Module Configuration Register (MCR)

This register defines global system configurations, such as the module operation mode (e.g., low power) and maximum message buffer configuration. Most of the fields in this register can be accessed at any time, except the MAXMB field, which should only be changed while the module is in Freeze Mode.

Base + 0x0000

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MDIS	FRZ	FEN	HALT	NOTRDY	WAK_MSK	SOFTRST	FRZACK	SUPV	SLF_WAK	WRNEN	MDISACK	WAK_SRC	0	SRXDIS	MBFEN
W					▨			▨				▨		▨		
RESET:	0	1	0	1	1	0	0	1	1	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	LPRIO_EN	AEN	0	0	IDAM		0	0	MAXMB					
W	▨	▨			▨	▨			▨	▨						
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

▨ = Unimplemented or Reserved

Figure 681. Module Configuration Register (MCR)

Table 679. Module Configuration Register (MCR) field descriptions

Field	Description
MDIS	Module Disable This bit controls whether FlexCAN is enabled or not. When disabled, FlexCAN shuts down the clocks to the CAN Protocol Interface and Message Buffer Management submodules. This is the only bit in MCR not affected by soft reset. See *Section , Module Disable Mode* for more information. 1: Disable the FlexCAN module 0: Enable the FlexCAN module
FRZ	Freeze Enable The FRZ bit specifies the FlexCAN behavior when MCR[HALT] is set or when Debug Mode is requested at MCU level. When FRZ is asserted, FlexCAN is enabled to enter Freeze Mode. Negation of this bit field causes FlexCAN to exit from Freeze Mode. 1: Enabled to enter Freeze Mode 0: Not enabled to enter Freeze Mode

Table 679. Module Configuration Register (MCR) field descriptions

Field	Description
FEN	FIFO Enable This bit controls whether the FIFO feature is enabled or not. When FEN is set, MBs 0 to 7 cannot be used for normal reception and transmission because the corresponding memory region (0x80–0xFF) is used by the FIFO engine. See *Section 29.4.4, Rx FIFO structure* and *Section 29.5.7, Rx FIFO* for more information. 1: FIFO enabled 0: FIFO not enabled
HALT	Halt FlexCAN Assertion of this bit puts the FlexCAN module into Freeze Mode. The CPU should clear it after initializing the Message Buffers and Control Register. No reception or transmission is performed by FlexCAN before this bit is cleared. While in Freeze Mode, the CPU has write access to the Error Counter Register, that is otherwise read-only. Freeze Mode can not be entered while FlexCAN is in any of the low power modes. See *Section , Freeze Mode* for more information. 1: Enters Freeze Mode if the FRZ bit is asserted. 0: No Freeze Mode request.
NOTRDY	FlexCAN Not Ready This read-only bit indicates that FlexCAN is either in Disable Mode, Stop Mode or Freeze Mode. It is negated once FlexCAN has exited these modes. 1: FlexCAN module is either in Disable Mode, Stop Mode or Freeze Mode 0: FlexCAN module is either in Normal Mode, Listen-Only Mode or Loop-Back Mode
WAK_MSK	Wake Up Interrupt Mask This bit enables the Wake Up Interrupt generation. 1: Wake Up Interrupt is enabled 0: Wake Up Interrupt is disabled

Table 679. Module Configuration Register (MCR) field descriptions

Field	Description
SOFTRST	Soft Reset When this bit is asserted, FlexCAN resets its internal state machines and some of the memory mapped registers. The following registers are reset: MCR (except the MDIS bit), TIMER, ECR, ESR, IMRL, IMRH, IFRL, IFRH. Configuration registers that control the interface to the CAN bus are not affected by soft reset. The following registers are unaffected: – CR – RXIMR0–RXIMR63 – RXGMASK, RX14MASK, RX15MASK – all Message Buffers The SOFTRST bit can be asserted directly by the CPU when it writes to the MCR, but it is also asserted when global soft reset is requested at MCU level. Since soft reset is synchronous and has to follow a request/acknowledge procedure across clock domains, it may take some time to fully propagate its effect. The SOFTRST bit remains asserted while reset is pending, and is automatically negated when reset completes. Therefore, software can poll this bit to know when the soft reset has completed. Soft reset cannot be applied while clocks are shut down in any of the low power modes. The module should be first removed from low power mode, and then soft reset can be applied. 1: Resets the registers marked as “affected by soft reset” in *Table 673* 0: No reset request
FRZACK	Freeze Mode Acknowledge This read-only bit indicates that FlexCAN is in Freeze Mode and its prescaler is stopped. The Freeze Mode request cannot be granted until current transmission or reception processes have finished. Therefore the software can poll the FRZACK bit to know when FlexCAN has actually entered Freeze Mode. If Freeze Mode request is negated, then this bit is negated once the FlexCAN prescaler is running again. If Freeze Mode is requested while FlexCAN is in any of the low power modes, then the FRZACK bit will only be set when the low power mode is exited. See *Section , Freeze Mode* for more information. 1: FlexCAN in Freeze Mode, prescaler stopped 0: FlexCAN not in Freeze Mode, prescaler running
SUPV	Supervisor Mode This bit configures some of the FlexCAN registers to be either in Supervisor or Unrestricted memory space. The registers affected by this bit are marked as S/U in the Access Type column of *Table 673*. Reset value of this bit is ‘1’, so the affected registers start with Supervisor access restrictions. 1: Affected registers are in Supervisor memory space. Any access without supervisor permission behaves as though the access was done to an unimplemented register location 0: Affected registers are in Unrestricted memory space

Table 679. Module Configuration Register (MCR) field descriptions

Field	Description
SLF_WAK	Self Wake Up This bit enables the Self Wake Up feature when FlexCAN is in Stop Mode. If this bit had been asserted by the time FlexCAN entered Stop Mode, then FlexCAN will look for a recessive to dominant transition on the bus during these modes. If a transition from recessive to dominant is detected during Stop Mode, then FlexCAN generates, if enabled to do so, a Wake Up interrupt to the CPU so that it can resume the clocks globally. This bit can not be written while the module is in Stop Mode. 1: FlexCAN Self Wake Up feature is enabled 0: FlexCAN Self Wake Up feature is disabled
WRNEN	Warning Interrupt Enable When asserted, this bit enables the generation of the TWRNINT and RWRNINT flags in the Error and Status Register. If WRNEN is negated, the TWRNINT and RWRNINT flags will always be zero, independent of the values of the error counters, and no warning interrupt will ever be generated. 1: TWRNINT and RWRNINT bits are set when the respective error counter transition from <96 to ≥ 96. 0: TWRNINT and RWRNINT bits are zero, independent of the values in the error counters.
MDISACK	Low Power Mode Acknowledge This read-only bit indicates that FlexCAN is either in Disable Mode or Stop Mode. Either of these low power modes can not be entered until all current transmission or reception processes have finished, so the CPU can poll the MDISACK bit to know when FlexCAN has actually entered low power mode. See *Section , Module Disable Mode* and *Section , Stop Mode* for more information. 1: FlexCAN is either in Disable Mode or Stop mode 0: FlexCAN not in any of the low power modes
WAK_SRC	Wake Up Source This bit defines whether the integrated low-pass filter is applied to protect the Rx CAN input from spurious wake up. See *Section , Stop Mode* for more information. 1: FlexCAN uses the filtered Rx input to detect recessive to dominant edges on the CAN bus 0: FlexCAN uses the unfiltered Rx input to detect recessive to dominant edges on the CAN bus.
SRX_DIS	Self Reception Disable This bit defines whether FlexCAN is allowed to receive frames transmitted by itself. If this bit is asserted, frames transmitted by the module will not be stored in any message buffer, regardless if the message buffer is programmed with an ID that matches the transmitted frame, and no interrupt flag or interrupt signal will be generated due to the frame reception. 1: Self reception disabled 0: Self reception enabled

Table 679. Module Configuration Register (MCR) field descriptions

Field	Description
MBFEN	Backwards Compatibility Configuration This bit is provided to support Backwards Compatibility with previous FlexCAN versions. When this bit is negated, the following configuration is applied: – For MCUs supporting individual Rx ID masking, this feature is disabled. Instead of individual ID masking per message buffer, FlexCAN uses its previous masking scheme with RXGMASK, RX14MASK and RX15MASK. – The reception queue feature is disabled. Upon receiving a message, if the first message buffer with a matching ID that is found is still occupied by a previous unread message, FlexCAN will not look for another matching message buffer. It will override this message buffer with the new message and set the CODE field to ‘0110’ (overrun). Upon reset this bit is negated, allowing legacy software to work without modification. 1: Individual Rx masking and queue feature are enabled. 0: Individual Rx masking and queue feature are disabled.
LPRIO_EN	Local Priority Enable This bit is provided for backwards compatibility reasons. It controls whether the local priority feature is enabled or not. It is used to extend the ID used during the arbitration process. With this extended ID concept, the arbitration process is done based on the full 32-bit word, but the actual transmitted ID still has 11-bit for standard frames and 29-bit for extended frames. 1: Local Priority enabled 0: Local Priority disabled
AEN	Abort Enable This bit is supplied for backwards compatibility reasons. When asserted, it enables the Tx abort feature. This feature guarantees a safe procedure for aborting a pending transmission, so that no frame is sent in the CAN bus without notification. 1: Abort enabled 0: Abort disabled
IDAM	ID Acceptance Mode This 2-bit field identifies the format of the elements of the Rx FIFO filter table, as shown in *Table 680*. Note that all elements of the table are configured at the same time by this field (they are all the same format). See *Section 29.4.4, Rx FIFO structure*.
MAXMB	Maximum Number of Message Buffers This 6-bit field defines the maximum number of message buffers that will take part in the matching and arbitration processes. The reset value (0x0F) is equivalent to 16 message buffer configuration. This field should be changed only while the module is in Freeze Mode. Maximum message buffers in use = MAXMB + 1. MAXMB must be programmed with a value smaller or equal to the number of available Message Buffers, otherwise FlexCAN can transmit and receive wrong messages.

Table 680. IDAM coding

IDAM	Format	Explanation
00	A	One full ID (standard or extended) per filter element
01	B	Two full standard IDs or two partial 14-bit extended IDs per filter element
10	C	Four partial 8-bit IDs (standard or extended) per filter element
11	D	All frames rejected

Control Register (CR)

This register is defined for specific FlexCAN control features related to the CAN bus, such as bit-rate, programmable sampling point within an Rx bit, Loop Back Mode, Listen Only Mode, Bus Off recovery behavior and interrupt enabling (Bus-Off, Error, Warning). It also determines the Division Factor for the clock prescaler. Most of the fields in this register should only be changed while the module is in Disable Mode or in Freeze Mode. Exceptions are the BOFFMSK, ERRMSK, TWRNMSK, RWRNMSK and BOFFREC bits, that can be accessed at any time.

Base + 0x0004

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	PRESDIV								RJW		PSEG1			PSEG2		
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	BOFFMSK	ERRMSK	CLKSRC	LPB	TWRNMSK	RWRNMSK	0	0	SMP	BOFFREC	TSYN	LBUF	LOM	PROPSEG		
W							▨	▨								
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 682. Control Register (CR)

Table 681. CR Register field descriptions

Field	Description
RESDIV	Prescaler Division Factor This 8-bit field defines the ratio between the CPI clock frequency and the Serial Clock (Sclock) frequency. The Sclock period defines the time quantum of the CAN protocol. For the reset value, the Sclock frequency is equal to the CPI clock frequency. The maximum value of this register is 0xFF, that gives a minimum Sclock frequency equal to the CPI clock frequency divided by 256. For more information refer to *Section , Protocol timing*. Sclock frequency = CPI clock frequency / (PRESDIV + 1)
PRJW	Resync Jump Width This 2-bit field defines the maximum number of time quanta(1) that a bit time can be changed by one resynchronization. The valid programmable values are 0–3. Resync Jump Width = RJW + 1.
PSEG1	Phase Segment 1 This 3-bit field defines the length of Phase Buffer Segment 1 in the bit time. The valid programmable values are 0–7. Phase Buffer Segment 1 = (PSEG1 + 1) x Time-Quanta.
PSEG2	Phase Segment 2 This 3-bit field defines the length of Phase Buffer Segment 2 in the bit time. The valid programmable values are 1–7. Phase Buffer Segment 2 = (PSEG2 + 1) x Time-Quanta.
BOFFMSK	Bus Off Mask This bit provides a mask for the Bus Off Interrupt. 1: Bus Off interrupt enabled 0: Bus Off interrupt disabled
ERRMSK	Error Mask This bit provides a mask for the Error Interrupt. 1: Error interrupt enabled 0: Error interrupt disabled
CLKSRC	CAN Engine Clock Source This bit selects the clock source to the CAN Protocol Interface (CPI) to be either the peripheral clock (driven by the PLL) or the crystal oscillator clock. The selected clock is the one fed to the prescaler to generate the Serial Clock (Sclock). In order to guarantee reliable operation, this bit should only be changed while the module is in Disable Mode. See *Section , Protocol timing* for more information. 1: The CAN engine clock source is the bus clock 0: The CAN engine clock source is the oscillator clock

Table 681. CR Register field descriptions

Field	Description
TWRNMSK	Tx Warning Interrupt Mask This bit provides a mask for the Tx Warning Interrupt associated with the TWRNINT flag in the Error and Status Register. This bit has no effect if MCR[WRNEN] is negated and it is read as zero when MCR[WRNEN] is negated. 1: Tx Warning Interrupt enabled 0: Tx Warning Interrupt disabled
RWRNMSK	Rx Warning Interrupt Mask This bit provides a mask for the Rx Warning Interrupt associated with the RWRNINT flag in the Error and Status Register. This bit has no effect if MCR[WRNEN] is negated and it is read as zero when MCR[WRNEN] is negated. 1: Rx Warning Interrupt enabled 0: Rx Warning Interrupt disabled
LPB	Loop Back This bit configures FlexCAN to operate in Loop-Back Mode. In this mode, FlexCAN performs an internal loop back that can be used for self test operation. The bit stream output of the transmitter is fed back internally to the receiver input. The Rx CAN input pin is ignored and the Tx CAN output goes to the recessive state (logic ‘1’). FlexCAN behaves as it normally does when transmitting, and treats its own transmitted message as a message received from a remote node. In this mode, FlexCAN ignores the bit sent during the ACK slot in the CAN frame acknowledge field, generating an internal acknowledge bit to ensure proper reception of its own message. Both transmit and receive interrupts are generated. 1: Loop Back enabled 0: Loop Back disabled
SMP	Sampling Mode This bit defines the sampling mode of CAN bits at the Rx input. 1: Three samples are used to determine the value of the received bit: the regular one (sample point) and 2 preceding samples, a majority rule is used 0: Just one sample is used to determine the bit value
BOFFREC	Bus Off Recovery Mode This bit defines how FlexCAN recovers from Bus Off state. If this bit is negated, automatic recovering from Bus Off state occurs according to the CAN Specification 2.0B. If the bit is asserted, automatic recovering from Bus Off is disabled and the module remains in Bus Off state until the bit is negated by the user. If the negation occurs before 128 sequences of 11 recessive bits are detected on the CAN bus, then Bus Off recovery happens as if the BOFFREC bit had never been asserted. If the negation occurs after 128 sequences of 11 recessive bits occurred, then FlexCAN will resynchronize to the bus by waiting for 11 recessive bits before joining the bus. After negation, the BOFFREC bit can be re-asserted again during Bus Off, but it will only be effective the next time the module enters Bus Off. If BOFFREC was negated when the module entered Bus Off, asserting it during Bus Off will not be effective for the current Bus Off recovery. 1: Automatic recovering from Bus Off state disabled 0: Automatic recovering from Bus Off state enabled, according to CAN Spec 2.0 part B

Table 681. CR Register field descriptions

Field	Description
TSYN	Timer Sync Mode This bit enables a mechanism that resets the free-running timer each time a message is received in Message Buffer 0. This feature provides means to synchronize multiple FlexCAN stations with a special “SYNC” message (i.e., global network time). If the FEN bit in MCR is set (FIFO enabled), MB8 is used for timer synchronization instead of MB0. 1: Timer Sync feature enabled 0: Timer Sync feature disabled
LBUF	Lowest Buffer Transmitted First This bit defines the ordering mechanism for Message Buffer transmission. When asserted, MCR[LPRIO_EN] does not affect the priority arbitration. 1: Lowest number buffer is transmitted first 0: Buffer with highest priority is transmitted first
LOM	Listen-Only Mode This bit configures FlexCAN to operate in Listen Only Mode. In this mode, transmission is disabled, all error counters are frozen and the module operates in a CAN Error Passive mode. Only messages acknowledged by another CAN station will be received. If FlexCAN detects a message that has not been acknowledged, it will flag a BIT0 error (without changing the REC), as if it was trying to acknowledge the message. 1: FlexCAN module operates in Listen Only Mode 0: Listen Only Mode is deactivated
PROPSEG	Propagation Segment This 3-bit field defines the length of the Propagation Segment in the bit time. The valid programmable values are 0–7. Propagation Segment Time = (PROPSEG + 1) * Time-Quanta. Time-Quantum = one Sclock period.

1. One time quantum is equal to the Sclock period.

Free running timer (TIMER)

This register represents a 16-bit free running counter that can be read and written by the CPU. The timer starts from 0x0000 after Reset, counts linearly to 0xFFFF, and wraps around.

The timer is clocked by the FlexCAN bit-clock (which defines the baud rate on the CAN bus). During a message transmission/reception, it increments by one for each bit that is received or transmitted. When there is no message on the bus, it counts using the previously programmed baud rate. During Freeze Mode, the timer is not incremented.

The timer value is captured at the beginning of the identifier field of any frame on the CAN bus. This captured value is written into the Time Stamp entry in a message buffer after a successful reception or transmission of a message.

Writing to the timer is an indirect operation. The data is first written to an auxiliary register and then an internal request/acknowledge procedure across clock domains is executed. All this is transparent to the user, except for the fact that the data will take some time to be actually written to the register. If desired, software can poll the register to discover when the data was actually written.

Base + 0x0008

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	TIMER															
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 683. Free Running Timer (TIMER)

Rx Global Mask (RXGMASK)

This register is provided for legacy support and for MCUs that do not have the individual masking per Message Buffer feature. For MCUs supporting individual masks per message buffer, setting MCR[MBFEN] causes the RXGMASK Register to have no effect on the module operation. For MCUs not supporting individual masks per message buffer, this register is always effective.

RXGMASK is used as acceptance mask for all Rx message buffers, excluding message buffers 14–15, which have individual mask registers. When MCR[FEN] is set (FIFO enabled), the RXGMASK also applies to all elements of the ID filter table, except elements 6–7, which have individual masks.

The contents of this register must be programmed while the module is in Freeze Mode, and must not be modified when the module is transmitting or receiving frames.

Base + 0x0010

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MI31	MI30	MI29	MI28	MI27	MI26	MI25	MI24	MI23	MI22	MI21	MI20	MI19	MI18	MI17	MI16
W																
RESET:	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	MI15	MI14	MI13	MI12	MI11	MI10	MI9	MI8	MI7	MI6	MI5	MI4	MI3	MI2	MI1	MI0
W																
RESET:	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

▨ = Unimplemented or Reserved

Figure 684. Rx Global Mask Register (RXGMASK)

Table 682. RXGMASK Register field descriptions

Field	Description
MI31–MI0	Mask Bits For normal Rx message buffers, the mask bits affect the ID filter programmed on the message buffer. For the Rx FIFO, the mask bits affect all bits programmed in the filter table (ID, IDE, RTR). 1: The corresponding bit in the filter is checked against the one received 0: The corresponding bit in the filter is "don't care"

Rx 14 Mask (RX14MASK)

This register is provided for legacy support and for low cost MCUs that do not have the individual masking per Message Buffer feature. For MCUs supporting individual masks per message buffer, setting MCR[MBFEN] causes the RX14MASK Register to have no effect on the module operation.

RX14MASK is used as acceptance mask for the Identifier in Message Buffer 14. When MCR[FEN] is set (FIFO enabled), the RXG14MASK also applies to element 6 of the ID filter table. This register has the same structure as the Rx Global Mask Register. It must be programmed while the module is in Freeze Mode, and must not be modified when the module is transmitting or receiving frames.

- Address Offset: 0x14
- Reset Value: 0xFFFF_FFFF

Rx 15 Mask (RX15MASK)

This register is provided for legacy support and for low cost MCUs that do not have the individual masking per Message Buffer feature. For MCUs supporting individual masks per message buffer, setting MCR[MBFEN] causes the RX15MASK Register to have no effect on the module operation.

When MCR[MBFEN] is negated, RX15MASK is used as acceptance mask for the Identifier in Message Buffer 15. When MCR[FEN] is set (FIFO enabled), the RXG14MASK also applies to element 7 of the ID filter table. This register has the same structure as the Rx Global Mask Register. It must be programmed while the module is in Freeze Mode, and must not be modified when the module is transmitting or receiving frames.

- Address Offset: 0x18
- Reset Value: 0xFFFF_FFFF

Error Counter Register (ECR)

This register has two 8-bit fields reflecting the value of two FlexCAN error counters: Transmit Error Counter (TXECNT) and Receive Error Counter (RXECNT). The rules for increasing and decreasing these counters are described in the CAN protocol and are completely implemented in the FlexCAN module. Both counters are read only except in Freeze Mode, where they can be written by the CPU.

Writing to the Error Counter Register while in Freeze Mode is an indirect operation. The data is first written to an auxiliary register and then an internal request/acknowledge procedure across clock domains is executed. All this is transparent to the user, except for the fact that the data will take some time to be actually written to the register. If desired, software can poll the register to discover when the data was actually written.

FlexCAN responds to any bus state as described in the protocol, e.g. transmit ‘Error Active’ or ‘Error Passive’ flag, delay its transmission start time (‘Error Passive’) and avoid any influence on the bus when in ‘Bus Off’ state. The following are the basic rules for FlexCAN bus state transitions.

- If the value of TXECNT or RXECNT increases to be greater than or equal to 128, ESR[FLTCONF] is updated to reflect ‘Error Passive’ state.
- If the FlexCAN state is ‘Error Passive’, and either TXECNT or RXECNT decrements to a value less than or equal to 127 while the other already satisfies this condition, ESR[FLTCONF] is updated to reflect ‘Error Active’ state.
- If the value of TXECNT increases to be greater than 255, ESR[FLTCONF] is updated to reflect ‘Bus Off’ state, and an interrupt may be issued. The value of TXECNT is then reset to zero.
- If FlexCAN is in ‘Bus Off’ state, then TXECNT is cascaded together with another internal counter to count the 128th occurrences of 11 consecutive recessive bits on the bus. Hence, TXECNT is reset to zero and counts in a manner where the internal counter counts 11 such bits and then wraps around while incrementing the TXECNT. When TXECNT reaches the value of 128, ESR[FLTCONF] is updated to be ‘Error Active’ and both error counters are reset to zero. At any instance of dominant bit following a stream of less than 11 consecutive recessive bits, the internal counter resets itself to zero without affecting the TXECNT value.
- If during system start-up, only one node is operating, then its TXECNT increases in each message it is trying to transmit, as a result of acknowledge errors (indicated by ESR[ACKERR]). After the transition to ‘Error Passive’ state, the TXECNT does not increment anymore by acknowledge errors. Therefore the device never goes to the ‘Bus Off’ state.
- If the RXECNT increases to a value greater than 127, it is not incremented further, even if more errors are detected while being a receiver. At the next successful message reception, the counter is set to a value between 119 and 127 to resume to ‘Error Active’ state.

Base + 0x001C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	RXECNT								TXECNT							
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

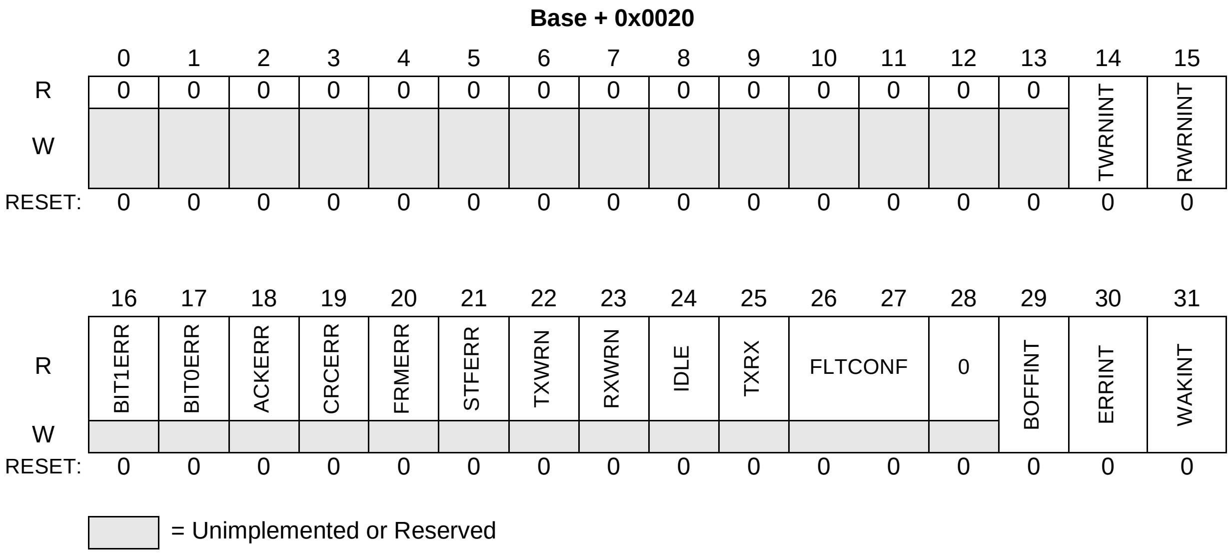
▨ = Unimplemented or Reserved

Figure 685. Error Counter Register (ECR)

Error and Status Register (ESR)

This register reflects various error conditions, some general status of the device and it is the source of four interrupts to the CPU. The reported error conditions (bits 16–21) are those that occurred since the last time the CPU read this register. The CPU read action clears bits16–21. Bits 22–28 are status bits.

Most bits in this register are read only, except TWRNINT, RWRNINT, BOFFINT, WAKINT and ERRINT, that are interrupt flags that can be cleared by writing ‘1’ to them (writing ‘0’ has no effect). See *Section 29.5.10, Interrupts* for more details.



Base + 0x0020

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	TWRNINT	RWRNINT
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨		
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	BIT1ERR	BIT0ERR	ACKERR	CRCERR	FRMERR	STFERR	TXWRN	RXWRN	IDLE	TXRX	FLTCONF		0	BOFFINT	ERRINT	WAKINT
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨		▨			
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 686. Error and Status Register (ESR)

Table 683. ESR Register field descriptions

Field	Description
TWRNINT	Tx Warning Interrupt Flag If the WRNEN bit in MCR is asserted, the TWRNINT bit is set when the TXWRN flag transition from ‘0’ to ‘1’, meaning that the Tx error counter reached 96. If the corresponding mask bit (CR[TWRNMSK]) is set, an interrupt is generated to the CPU. This bit is cleared by writing it to ‘1’. Writing ‘0’ has no effect. 1: The Tx error counter transition from < 96 to ≥ 96 0: No such occurrence
RWRNINT	Rx Warning Interrupt Flag If the WRNEN bit in MCR is asserted, the RWRNINT bit is set when the RXWRN flag transition from ‘0’ to ‘1’, meaning that the Rx error counters reached 96. If the corresponding mask bit (CR[RWRNMSK]) is set, an interrupt is generated to the CPU. This bit is cleared by writing it to ‘1’. Writing ‘0’ has no effect. 1: The Rx error counter transition from < 96 to ≥ 96 0: No such occurrence

Table 683. ESR Register field descriptions

Field	Description
BIT1ERR	Bit1 Error This bit indicates when an inconsistency occurs between the transmitted and the received bit in a message. 1: At least one bit sent as recessive is received as dominant 0: No such occurrence This bit is not set by a transmitter in case of arbitration field or ACK slot, or in case of a node sending a passive error flag that detects dominant bits.
BIT0ERR	Bit0 Error This bit indicates when an inconsistency occurs between the transmitted and the received bit in a message. 1: At least one bit sent as dominant is received as recessive 0: No such occurrence
ACKERR	Acknowledge Error This bit indicates that an Acknowledge Error has been detected by the transmitter node, i.e., a dominant bit has not been detected during the ACK SLOT. 1: An ACK error occurred since last read of this register 0: No such occurrence
CRCERR	Cyclic Redundancy Check Error This bit indicates that a CRC Error has been detected by the receiver node, i.e., the calculated CRC is different from the received. 1: A CRC error occurred since last read of this register. 0: No such occurrence
FRMERR	Form Error This bit indicates that a Form Error has been detected by the receiver node, i.e., a fixed-form bit field contains at least one illegal bit. 1: A Form Error occurred since last read of this register 0: No such occurrence
STFERR	Stuffing Error This bit indicates that a Stuffing Error has been detected. 1: A Stuffing Error occurred since last read of this register. 0: No such occurrence.
TXWRN	TX Error Warning This bit indicates when repetitive errors are occurring during message transmission. 1: TX_Err_Counter ≥ 96 0: No such occurrence

Table 683. ESR Register field descriptions

Field	Description
RXWRN	Rx Warning This bit indicates when repetitive errors are occurring during message reception. 1: Rx_Err_Counter ≥ 96 0: No such occurrence
IDLE	CAN bus IDLE state This bit indicates when CAN bus is in IDLE state. 1: CAN bus is now IDLE 0: No such occurrence
TXRX	Current FlexCAN status (transmitting/receiving) This bit indicates if FlexCAN is transmitting or receiving a message when the CAN bus is not in IDLE state. This bit has no meaning when IDLE is asserted. 1: FlexCAN is transmitting a message (IDLE=0) 0: FlexCAN is receiving a message (IDLE=0)
FLTCONF	Fault Confinement State This 2-bit field indicates the Confinement State of the FlexCAN module. 00: Error Active 01: Error Passive 1x: Bus Off If the LOM bit in the Control Register is asserted, the FLTCONF field will indicate “Error Passive”. Since the Control Register is not affected by soft reset, the FLTCONF field will not be affected by soft reset if the LOM bit is asserted.
BOFFINT	Bus Off’ Interrupt This bit is set when FlexCAN enters ‘Bus Off’ state. If the corresponding mask bit (CR[BOFFMSK]) is set, an interrupt is generated to the CPU. This bit is cleared by writing it to ‘1’. Writing ‘0’ has no effect. 1: FlexCAN module entered ‘Bus Off’ state 0: No such occurrence

Table 683. ESR Register field descriptions

Field	Description
ERRINT	Error Interrupt This bit indicates that at least one of the Error Bits (bits 16–21) is set. If the corresponding mask bit (CR[ERRMSK]) is set, an interrupt is generated to the CPU. This bit is cleared by writing it to ‘1’.Writing ‘0’ has no effect. 1: Indicates setting of any Error Bit in the Error and Status Register 0: No such occurrence
WAKINT	Wake-Up Interrupt When FlexCAN is in Stop Mode and a recessive to dominant transition is detected on the CAN bus and if MCR[WAK_MSK] is set, an interrupt is generated to the CPU. This bit is cleared by writing it to ‘1’. Writing ‘0’ has no effect. 1: Indicates a recessive to dominant transition received on the CAN bus when the FlexCAN module is in Stop Mode 0: No such occurrence

Interrupt Masks 2 Register (IMRH)

This register allows any number of a range of 32 Message Buffer Interrupts to be enabled or disabled. It contains one interrupt mask bit per buffer, enabling the CPU to determine which buffer generates an interrupt after a successful transmission or reception (i.e. when the corresponding bit in the IFRH register is set).

Base + 0x0024

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	BUF 63M	BUF 62M	BUF 61M	BUF 60M	BUF 59M	BUF 58M	BUF 57M	BUF 56M	BUF 55M	BUF 54M	BUF 53M	BUF 52M	BUF 51M	BUF 50M	BUF 49M	BUF 48M
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	BUF 47M	BUF 46M	BUF 45M	BUF 44M	BUF 43M	BUF 42M	BUF 41M	BUF 40M	BUF 39M	BUF 38M	BUF 37M	BUF 36M	BUF 35M	BUF 34M	BUF 33M	BUF 32M
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 687. Interrupt Masks 2 Register (IMRH)

Table 684. IMRH Register field descriptions

Field	Description
BUF63M–BUF32M	Buffer MB_i Mask Each bit enables or disables the respective FlexCAN Message Buffer (MB32 to MB63) Interrupt. 1: The corresponding buffer Interrupt is enabled 0: The corresponding buffer Interrupt is disabled Setting or clearing a bit in the IMRH register can assert or negate an interrupt request, if the corresponding bit in the IFRH register is set.

Interrupt Masks 1 Register (IMRL)

This register allows to enable or disable any number of a range of 32 Message Buffer Interrupts. It contains one interrupt mask bit per buffer, enabling the CPU to determine which buffer generates an interrupt after a successful transmission or reception (i.e., when the corresponding bit in the IFRL register is set).

Base + 0x0028

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	BUF 31M	BUF 30M	BUF 29M	BUF 28M	BUF 27M	BUF 26M	BUF 25M	BUF 24M	BUF 23M	BUF 22M	BUF 21M	BUF 20M	BUF 19M	BUF 18M	BUF 17M	BUF 16M
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	BUF 15M	BUF 14M	BUF 13M	BUF 12M	BUF 11M	BUF 10M	BUF 9M	BUF 8M	BUF 7M	BUF 6M	BUF 5M	BUF 4M	BUF 3M	BUF 2M	BUF 1M	BUF 0M
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 688. Interrupt Masks 1 Register (IMRL)

Table 685. IMRL Register field descriptions

Field	Description
BUF31M–BUF0M	Buffer MB_i Mask Each bit enables or disables the respective FlexCAN Message Buffer (MB0 to MB31) Interrupt. 1: The corresponding buffer Interrupt is enabled 0: The corresponding buffer Interrupt is disabled Setting or clearing a bit in the IMRL Register can assert or negate an interrupt request, if the corresponding bit in the IFRL register is set.

Interrupt Flags 2 Register (IFRH)

This register defines the flags for 32 Message Buffer interrupts. It contains one interrupt flag bit per buffer. Each successful transmission or reception sets the corresponding bit in IFRH.

If the corresponding bit in IMRH is set, an interrupt will be generated. The interrupt flag must be cleared by writing it to ‘1’. Writing ‘0’ has no effect.

When MCR[AEN] is set (Abort enabled), while the IFRH bit is set for a message buffer configured as Tx, the writing access done by CPU into the corresponding message buffer will be blocked.

Base + 0x002C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	BUF 63I	BUF 62I	BUF 61I	BUF 60I	BUF 59I	BUF 58I	BUF 57I	BUF 56I	BUF 55I	BUF 54I	BUF 53I	BUF 52I	BUF 51I	BUF 50I	BUF 49I	BUF 48I
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	BUF 47I	BUF 46I	BUF 45I	BUF 44I	BUF 43I	BUF 42I	BUF 41I	BUF 40I	BUF 39I	BUF 38I	BUF 37I	BUF 36I	BUF 35I	BUF 34I	BUF 33I	BUF 32I
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 689. Interrupt Flags 2 Register (IFRH)

Table 686. IFRH Register field descriptions

Field	Description
BUF32I– BUF63I	Buffer MB$_i$ Interrupt Each bit flags the respective FlexCAN Message Buffer (MB32 to MB63) interrupt. 1: The corresponding buffer has successfully completed transmission or reception 0: No such occurrence

Interrupt Flags 1 Register (IFRL)

This register defines the flags for 32 Message Buffer interrupts and FIFO interrupts. It contains one interrupt flag bit per buffer. Each successful transmission or reception sets the corresponding bit in the IFRL register. If the corresponding bit in the IMRL register is set, an interrupt will be generated. The Interrupt flag must be cleared by writing it to ‘1’. Writing ‘0’ has no effect.

When MCR[AEN] is set (Abort enabled), while the bit in the IFRL is set for a message buffer configured as Tx, the writing access done by CPU into the corresponding message buffer will be blocked.

When MCR[FEN] is set (FIFO enabled), the function of the 8 least significant interrupt flags (BUF7I– BUF0I) is changed to support the FIFO operation. BUF7I, BUF6I and BUF5I indicate operating conditions of the FIFO, while BUF4I to BUF0I are not used.

Base + 0x0030

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	BUF 31I	BUF 30I	BUF 29I	BUF 28I	BUF 27I	BUF 26I	BUF 25I	BUF 24I	BUF 23I	BUF 22I	BUF 21I	BUF 20I	BUF 19I	BUF 18I	BUF 17I	BUF 16I
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	BUF 15I	BUF 14I	BUF 13I	BUF 12I	BUF 11I	BUF 10I	BUF 9I	BUF 8I	BUF 7I	BUF 6I	BUF 5I	BUF 4I	BUF 3I	BUF 2I	BUF 1I	BUF 0I
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 690. Interrupt Flags 1 Register (IFRL)

Table 687. IFRL Register field descriptions

Field	Description
BUF31I–BUF8I	Buffer MB$_i$ Interrupt Each bit flags the respective FlexCAN Message Buffer (MB8 to MB31) interrupt. 1: The corresponding message buffer has successfully completed transmission or reception 0: No such occurrence
BUF7I	Buffer MB7 Interrupt or “FIFO Overflow” If the FIFO is not enabled, this bit flags the interrupt for MB7. If the FIFO is enabled, this flag indicates an overflow condition in the FIFO (frame lost because FIFO is full). 1: MB7 completed transmission/reception or FIFO overflow 0: No such occurrence
BUF6I	Buffer MB6 Interrupt or “FIFO Warning” If the FIFO is not enabled, this bit flags the interrupt for MB6. If the FIFO is enabled, this flag indicates that 5 out of 6 buffers of the FIFO are already occupied (FIFO almost full). 1: MB6 completed transmission/reception or FIFO almost full 0: No such occurrence
BUF5I	Buffer MB5 Interrupt or “Frames available in FIFO” If the FIFO is not enabled, this bit flags the interrupt for MB5. If the FIFO is enabled, this flag indicates that at least one frame is available to be read from the FIFO. 1: MB5 completed transmission/reception or frames available in the FIFO 0: No such occurrence
BUF4I–BUF0I	Buffer MB$_i$ Interrupt or “reserved” If the FIFO is not enabled, these bits flag the interrupts for MB0 to MB4. If the FIFO is enabled, these flags are not used and must be considered as reserved locations. 1: Corresponding message buffer completed transmission/reception 0: No such occurrence

Rx Individual Mask Registers (RXIMR0–RXIMR63)

These registers are used as acceptance masks for ID filtering in Rx message buffers and the FIFO. If the FIFO is not enabled, one mask register is provided for each available Message Buffer, providing ID masking capability on a per Message Buffer basis. When the FIFO is enabled (MCR[FEN] is set), the first eight Mask Registers apply to the eight elements of the FIFO filter table (on a one-to-one correspondence), while the rest of the registers apply to the regular message buffers, starting from MB8.

The Individual Rx Mask Registers are implemented in RAM, so they are not affected by reset and must be explicitly initialized prior to any reception. Furthermore, they can only be accessed by the CPU while the module is in Freeze Mode. Out of Freeze Mode, write accesses are blocked and read accesses will return “all zeros”. Furthermore, if MCR[MBFEN] is negated, any read or write operation to these registers results in access error.

Base + 0x0880–0x097F

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
MI31	MI30	MI29	MI28	MI27	MI26	MI25	MI24	MI23	MI22	MI21	MI20	MI19	MI18	MI17	MI16

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MI15	MI14	MI13	MI12	MI11	MI10	MI9	MI8	MI7	MI6	MI5	MI4	MI3	MI2	MI1	MI0

Figure 691. Rx Individual Mask Registers (RXIMR0 – RXIMR63)

Table 688. RXIMR0 – RXIMR63 Register field descriptions

Field	Description
MI31–MI0	Mask Bits For normal Rx message buffers, the mask bits affect the ID filter programmed on the message buffer. For the Rx FIFO, the mask bits affect all bits programmed in the filter table (ID, IDE, RTR). 1: The corresponding bit in the filter is checked against the one received 0: the corresponding bit in the filter is “don’t care”

29.5 Functional description

29.5.1 Overview

The FlexCAN module is a CAN protocol engine with a very flexible mailbox system for transmitting and receiving CAN frames. The mailbox system is composed by a set of up to 64 Message Buffers (MB) that store configuration and control data, time stamp, message ID and data (see *Section 29.4.3, Message buffer structure*). The memory corresponding to the first eight message buffers can be configured to support a FIFO reception scheme with a powerful ID filtering mechanism, capable of checking incoming frames against a table of IDs (up to eight extended IDs or sixteen standard IDs or thirty-two 8-bit ID slices), each one with its own individual mask register. Simultaneous reception through FIFO and mailbox is supported. For mailbox reception, a matching algorithm makes it possible to store received frames only into message buffers that have the same ID programmed on its ID field. A masking scheme makes it possible to match the ID programmed on the message buffer with a range of IDs on received CAN frames. For transmission, an arbitration algorithm decides the prioritization of message buffers to be transmitted based on the message ID (optionally augmented by three local priority bits) or the message buffer ordering.

Before proceeding with the functional description, an important concept must be explained. A Message Buffer is said to be “active” at a given time if it can participate in the matching and arbitration algorithms that are happening at that time. An Rx message buffer with a ‘0000’ code is inactive (refer to *Table 676*). Similarly, a Tx message buffer with a ‘1000’ or ‘1001’ code is also inactive (refer to *Table 677*). A message buffer not programmed with ‘0000’, ‘1000’ or ‘1001’ will be temporarily deactivated (will not participate in the current arbitration or matching run) when the CPU writes to the C/S field of that message buffer (see *Section , Message buffer deactivation*).

29.5.2 Transmit process

In order to transmit a CAN frame, the CPU must prepare a Message Buffer for transmission by executing the following procedure:

- If the message buffer is active (transmission pending), write ‘1000’ to the Code field to inactivate the message buffer.The deactivated message buffer can transmit without setting IFLAG and without updating the CODE field (see *Section , Message buffer deactivation*).
- Write the ID word.
- Write the data bytes.
- Write the Length, Control and Code fields of the Control and Status word to activate the message buffer.

Once the message buffer is activated in the fourth step, it will participate into the arbitration process and eventually be transmitted according to its priority. At the end of the successful transmission, the value of the Free Running Timer is written into the Time Stamp field, the Code field in the Control and Status word is updated, a status flag is set in the Interrupt Flag Register and an interrupt is generated if allowed by the corresponding Interrupt Mask Register bit. The new Code field after transmission depends on the code that was used to activate the message buffer in step four (see *Table 676* and *Table 677* in *Section 29.4.3, Message buffer structure*). When the Abort feature is enabled (MCR[AEN] is asserted), after the Interrupt Flag is asserted for a message buffer configured as transmit buffer, the message buffer is blocked, therefore the CPU is not able to update it until the Interrupt Flag be negated by CPU. It means that the CPU must clear the corresponding IFRL or IFRH register before starting to prepare this message buffer for a new transmission or reception.

29.5.3 Arbitration process

The arbitration process is an algorithm executed by the MBM that scans the whole message buffer memory looking for the highest priority message to be transmitted. All message buffers programmed as transmit buffers will be scanned to find the lowest ID(bb) or the

---

bb. Actually, if CR[LBUF] is negated, the arbitration considers not only the ID, but also the RTR and IDE bits placed inside the ID at the same positions they are transmitted in the CAN frame.

lowest MB number or the highest priority, depending on bits CR[LBUF] and MCR[LPRIO_EN]. The arbitration process is triggered in the following events:

- During the CRC field of the CAN frame
- During the error delimiter field of the CAN frame
- During Intermission, if the winner message buffer defined in a previous arbitration was deactivated, or if there was no message buffer to transmit, but the CPU wrote to the C/S word of any message buffer after the previous arbitration finished
- When MBM is in Idle or Bus Off state and the CPU writes to the C/S word of any message buffer
- Upon leaving Freeze Mode

When CR[LBUF] is asserted, MCR[LPRIO_EN] has no effect and the lowest number buffer is transmitted first. When CR[LBUF] and MCR[LPRIO_EN] are both negated, the message buffer with the lowest ID is transmitted first but if CR[LBUF] is negated and MCR[LPRIO_EN] is asserted, the PRIO bits augment the ID used during the arbitration process. With this extended ID concept, arbitration is done based on the full 32-bit ID and the PRIO bits define which message buffer should be transmitted first, therefore message buffers with PRIO = 000 have higher priority. If two or more message buffers have the same priority, the regular ID will determine the priority of transmission. If two or more message buffers have the same priority (three extra bits) and the same regular ID, the lowest message buffer will be transmitted first.

Once the highest priority message buffer is selected, it is transferred to a temporary storage space called Serial Message Buffer (SMB), which has the same structure as a normal message buffer but is not user accessible. This operation is called “move-out” and after it is done, write access to the corresponding message buffer is blocked (if MCR[AEN] is asserted). The write access is released in the following events:

- After the message buffer is transmitted
- FlexCAN enters in HALT or BUS OFF
- FlexCAN loses the bus arbitration or there is an error during the transmission

At the first opportunity window on the CAN bus, the message on the SMB is transmitted according to the CAN protocol rules. FlexCAN transmits up to eight data bytes, even if the DLC (Data Length Code) value is bigger.

29.5.4 Receive process

To be able to receive CAN frames into the mailbox message buffers, the CPU must prepare one or more Message Buffers for reception by executing the following steps:

1. If the message buffer has a pending transmission, write an ABORT code (‘1001’) to the Code field of the Control and Status word to request an abortion of the transmission, then read back the Code field and the IFRL or IFRH register to check if the transmission was aborted (see *Section , Transmission abort mechanism*). If backwards compatibility is desired (AEN in MCR negated), just write ‘1000’ to the Code field to inactivate the message buffer, but then the pending frame may be transmitted without notification (see *Section , Message buffer deactivation*). If the message buffer already programmed as a receiver, just write ‘0000’ to the Code field of the Control and Status word to keep the message buffer inactive.
2. Write the ID word
3. Write ‘0100’ to the Code field of the Control and Status word to activate the message buffer

Once the message buffer is activated in the third step, it will be able to receive frames that match the programmed ID. At the end of a successful reception, the message buffer is updated by the MBM as follows:

1. The value of the Free Running Timer is written into the Time Stamp field
2. The received ID, Data (8 bytes at most) and Length fields are stored
3. The Code field in the Control and Status word is updated (see *Table 676* and *Table 677* in *Section 29.4.3, Message buffer structure*)
4. A status flag is set in the Interrupt Flag Register and an interrupt is generated if allowed by the corresponding Interrupt Mask Register bit

Upon receiving the MB interrupt, the CPU should service the received frame using the following procedure:

1. Read the Control and Status word (mandatory – activates an internal lock for this buffer)
2. Read the ID field (optional – needed only if a mask was used)
3. Read the Data field
4. Read the Free Running Timer (optional – releases the internal lock)

Upon reading the Control and Status word, if the BUSY bit is set in the Code field, then the CPU should defer the access to the message buffer until this bit is negated. Reading the Free Running Timer is not mandatory. If not executed the message buffer remains locked, unless the CPU reads the C/S word of another message buffer. Note that only a single message buffer is locked at a time. The only mandatory CPU read operation is the one on the Control and Status word to assure data coherency (see *Section 29.5.6, Data coherence*).

The CPU should synchronize to frame reception by the status flag bit for the specific message buffer in the corresponding IFRL or IFRH register and not by the Code field of that message buffer. Polling the Code field does not work because once a frame was received and the CPU services the message buffer (by reading the C/S word followed by unlocking the message buffer), the Code field will not return to EMPTY. It will remain FULL, as explained in *Table 676*. If the CPU tries to workaround this behavior by writing to the C/S word to force an EMPTY code after reading the message buffer, the message buffer is actually deactivated from any currently ongoing matching process. As a result, a newly received frame matching the ID of that message buffer may be lost. In summary: **never do polling by reading directly the C/S word of the message buffers. Instead, read the corresponding IFRL or IFRH register**.

Note that the received ID field is always stored in the matching message buffer, thus the contents of the ID field in a message buffer may change if the match was due to masking. Note also that FlexCAN does receive frames transmitted by itself if there exists an Rx matching message buffer, provided MCR[SRX_DIS] is not asserted. If MCR[SRX_DIS] is asserted, FlexCAN will not store frames transmitted by itself in any message buffer, even if it contains a matching message buffer, and no interrupt flag or interrupt signal will be generated due to the frame reception.

To be able to receive CAN frames through the FIFO, the CPU must enable and configure the FIFO during Freeze Mode (see *Section 29.5.7, Rx FIFO*). Upon receiving the frames available interrupt from FIFO, the CPU should service the received frame using the following procedure:

1. Read the Control and Status word (optional – needed only if a mask was used for IDE and RTR bits)
2. Read the ID field (optional – needed only if a mask was used)
3. Read the Data field
4. Clear the frames available interrupt (mandatory – release the buffer and allow the CPU to read the next FIFO entry)

29.5.5 Matching process

The matching process is an algorithm executed by the MBM that scans the message buffer memory looking for Rx message buffers programmed with the same ID as the one received from the CAN bus. If the FIFO is enabled, the 8-entry ID table from FIFO is scanned first and then, if a match is not found within the FIFO table, the other message buffers are scanned. In the event that the FIFO is full, the matching algorithm will always look for a matching message buffer outside the FIFO region.

When the frame is received, it is temporarily stored in a hidden auxiliary message buffer called Serial Message Buffer (SMB). The matching process takes place during the CRC field of the received frame. If a matching ID is found in the FIFO table or in one of the regular message buffers, the contents of the SMB will be transferred to the FIFO or to the matched message buffer during the 6th bit of the End-Of-Frame field of the CAN protocol. This operation is called “move-in”. If any protocol error (CRC, ACK, etc.) is detected, than the move-in operation does not happen.

For the regular mailbox message buffers, a message buffer is said to be “free to receive” a new frame if the following conditions are satisfied:

- The message buffer is not locked (see *Section , Message buffer lock mechanism*)
- The Code field is either EMPTY or else it is FULL or OVERRUN but the CPU has already serviced the message buffer (read the C/S word and then unlocked the message buffer)

If the first message buffer with a matching ID is not “free to receive” the new frame, then the matching algorithm keeps looking for another free message buffer until it finds one. If it can not find one that is free, then it will overwrite the last matching message buffer (unless it is locked) and set the Code field to OVERRUN (refer to *Table 676* and *Table 677*). If the last matching message buffer is locked, then the new message remains in the SMB, waiting for the message buffer to be unlocked (see *Section , Message buffer lock mechanism*).

Suppose, for example, that the FIFO is disabled and there are two message buffers with the same ID, and FlexCAN starts receiving messages with that ID. Let us say that these message buffers are the second and the fifth in the array. When the first message arrives, the matching algorithm will find the first match in MB number 2. The code of this message buffer is EMPTY, so the message is stored there. When the second message arrives, the matching algorithm will find MB number 2 again, but it is not “free to receive”, so it will keep looking and find MB number 5 and store the message there. If yet another message with the same ID arrives, the matching algorithm finds out that there are no matching message buffers that are “free to receive”, so it decides to overwrite the last matched message buffer, which is number 5. In doing so, it sets the Code field of the message buffer to indicate OVERRUN.

The ability to match the same ID in more than one message buffer can be exploited to implement a reception queue (in addition to the full featured FIFO) to allow more time for the CPU to service the message buffers. By programming more than one message buffer with the same ID, received messages will be queued into the message buffers. The CPU can examine the Time Stamp field of the message buffers to determine the order in which the messages arrived.

The matching algorithm described above can be changed to be the same one used in previous versions of the FlexCAN module. When the MBFEN bit in MCR is negated, the matching algorithm stops at the first message buffer with a matching ID that it founds, whether this message buffer is free or not. As a result, the message queueing feature does not work if the MBFEN bit is negated.

Matching to a range of IDs is possible by using ID Acceptance Masks. FlexCAN supports individual masking per message buffer. Please refer to *Section , Rx Individual Mask Registers (RXIMR0–RXIMR63)*. During the matching algorithm, if a mask bit is asserted, then the corresponding ID bit is compared. If the mask bit is negated, the corresponding ID bit is “don’t care”. Please note that the Individual Mask Registers are implemented in RAM, so they are not initialized out of reset. Also, they can only be programmed if the MBFEN bit is asserted and while the module is in Freeze Mode.

FlexCAN also supports an alternate masking scheme with only three mask registers (RGXMASK, RX14MASK and RX15MASK) for backwards compatibility. This alternate masking scheme is enabled when MCR[MBFEN] is negated.

29.5.6 Data coherence

In order to maintain data coherency and FlexCAN proper operation, the CPU must obey the rules described in *Section 29.5.2, Transmit process* and *Section 29.5.4, Receive process*. Any form of CPU accessing a message buffer structure within FlexCAN other than those specified may cause FlexCAN to behave in an unpredictable way.

Transmission abort mechanism

The abort mechanism provides a safe way to request the abortion of a pending transmission. A feedback mechanism is provided to inform the CPU if the transmission was aborted or if the frame could not be aborted and was transmitted instead. In order to maintain backwards compatibility, the abort mechanism must be explicitly enabled by asserting MCR[AEN].

In order to abort a transmission, the CPU must write a specific abort code (1001) to the Code field of the Control and Status word. When the abort mechanism is enabled, the active message buffers configured as transmission must be aborted first and then they may be updated. If the abort code is written to a message buffer that is currently being transmitted, or to a message buffer that was already loaded into the SMB for transmission, the write operation is blocked and the message buffer is not deactivated, but the abort request is captured and kept pending until one of the following conditions are satisfied:

- The module loses the bus arbitration
- There is an error during the transmission
- The module is put into Freeze Mode

If none of conditions above are reached, the message buffer is transmitted correctly, the interrupt flag is set in the corresponding IFRL or IFRH register and an interrupt to the CPU is generated (if enabled). The abort request is automatically cleared when the interrupt flag is set. In the other hand, if one of the above conditions is reached, the frame is not transmitted, therefore the abort code is written into the Code field, the interrupt flag is set in the corresponding IFRL or IFRH register and an interrupt is (optionally) generated to the CPU.

If the CPU writes the abort code before the transmission begins internally, then the write operation is not blocked, therefore the message buffer is updated and no interrupt flag is set. In this way the CPU just needs to read the abort code to make sure the active message buffer was deactivated. Although the AEN bit is asserted and the CPU wrote the abort code, in this case the message buffer is deactivated and not aborted, because the transmission did not start yet. One message buffer is only aborted when the abort request is captured and kept pending until one of the previous conditions are satisfied.

The abort procedure can be summarized as follows:

- CPU writes 1001 into the code field of the C/S word
- CPU reads the CODE field and compares it to the value that was written
- If the CODE field that was read is different from the value that was written, the CPU must read the corresponding IFRL or IFRH register to check if the frame was transmitted or it is being currently transmitted. If the corresponding IFRL or IFRH is set, the frame was transmitted. If the corresponding IFRL or IFRH is reset, the CPU must wait for it to be set, and then the CPU must read the CODE field to check if the message buffer was aborted (CODE = 1001) or it was transmitted (CODE = 1000).

Note: An Abort request to a TxMB can block any write operation into its CODE field. As a consequence, the TxMB cannot be aborted or deactivated anymore until it completes the transmission by winning the CAN bus arbitration.

Message buffer deactivation

Deactivation is mechanism provided to maintain data coherence when the CPU writes to the Control and Status word of active message buffers out of Freeze Mode. Any CPU write access to the Control and Status word of a message buffer causes that message buffer to be excluded from the transmit or receive processes during the current matching or arbitration round. The deactivation is temporary, affecting only for the current match/arbitration round.

The purpose of deactivation is data coherency. The match/arbitration process scans the message buffers to decide which message buffer to transmit or receive. If the CPU updates the message buffer in the middle of a match or arbitration process, the data of that message buffer may no longer be coherent, therefore deactivation of that message buffer is done.

Even with the coherence mechanism described above, writing to the Control and Status word of active message buffers when not in Freeze Mode may produce undesirable results. Examples are:

- Matching and arbitration are one-pass processes. If message buffers are deactivated after they are scanned, no re-evaluation is done to determine a new match/winner. If an Rx message buffer with a matching ID is deactivated during the matching process after it was scanned, then this message buffer is marked as invalid to receive the frame, and FlexCAN will keep looking for another matching message buffer within the ones it has not scanned yet. If it can not find one, then the message will be lost. Suppose, for example, that two message buffers have a matching ID to a received frame, and the user deactivated the first matching message buffer after FlexCAN has scanned the second. The received frame will be lost even if the second matching message buffer was “free to receive”.
- If a Tx message buffer containing the lowest ID is deactivated after FlexCAN has scanned it, then FlexCAN will look for another winner within the message buffers that it has not scanned yet. Therefore, it may transmit a message buffer with ID that may not

be the lowest at the time because a lower ID might be present in one of the message buffers that it had already scanned before the deactivation.

- There is a point in time until which the deactivation of a Tx message buffer causes it not to be transmitted (end of move-out). After this point, it is transmitted but no interrupt is issued and the Code field is not updated. In order to avoid this situation, the abort procedures described in *Section , Transmission abort mechanism* should be used.

Message buffer lock mechanism

Besides message buffer deactivation, FlexCAN has another data coherence mechanism for the receive process. When the CPU reads the Control and Status word of an “active not empty” Rx message buffer, FlexCAN assumes that the CPU wants to read the whole message buffer in an atomic operation, and thus it sets an internal lock flag for that message buffer. The lock is released when the CPU reads the Free Running Timer (global unlock operation), or when it reads the Control and Status word of another message buffer. The message buffer locking is done to prevent a new frame to be written into the message buffer while the CPU is reading it.

Note: The locking mechanism only applies to Rx message buffers which have a code different than INACTIVE (‘0000’) or EMPTY(bc) *(‘0100’). Also, Tx message buffers can not be locked.*

Suppose, for example, that the FIFO is disabled and the second and the fifth message buffers of the array are programmed with the same ID, and FlexCAN has already received and stored messages into these two message buffers. Suppose now that the CPU decides to read MB number 5 and at the same time another message with the same ID is arriving. When the CPU reads the Control and Status word of MB number 5, this MB is locked. The new message arrives and the matching algorithm finds out that there are no “free to receive” message buffers, so it decides to override MB number 5. However, this message buffer is locked, so the new message can not be written there. It will remain in the SMB waiting for the message buffer to be unlocked, and only then will be written to the message buffer. If the message buffer is not unlocked in time and yet another new message with the same ID arrives, then the new message overwrites the one on the SMB and there will be no indication of lost messages either in the Code field of the message buffer or in the Error and Status Register.

While the message is being moved-in from the SMB to the message buffer, the BUSY bit on the Code field is asserted. If the CPU reads the Control and Status word and finds out that the BUSY bit is set, it should defer accessing the message buffer until the BUSY bit is negated.

Note: If the BUSY bit is asserted or if the message buffer is empty, then reading the Control and Status word does not lock the message buffer.

Deactivation takes precedence over locking. If the CPU deactivates a locked Rx message buffer, then its lock status is negated and the message buffer is marked as invalid for the current matching round. Any pending message on the SMB will not be transferred anymore to the message buffer.

---

bc. In previous FlexCAN versions, reading the C/S word locked the message buffer even if it was EMPTY. In current FlexCAN versions, this behavior is maintained when the MBFEN bit is negated.

29.5.7 Rx FIFO

The receive-only FIFO is enabled by asserting MCR[FEN]. The reset value of this bit is zero to maintain software backwards compatibility with previous versions of the module that did not have the FIFO feature. When the FIFO is enabled, the memory region normally occupied by the first eight message buffers (0x80–0xFF) is now reserved for use of the FIFO engine (see *Section 29.4.4, Rx FIFO structure*). Management of read and write pointers is done internally by the FIFO engine. The CPU can read the received frames sequentially, in the order they were received, by repeatedly accessing a Message Buffer structure at the beginning of the memory.

The FIFO can store up to 6 frames pending service by the CPU. An interrupt is sent to the CPU when new frames are available in the FIFO. Upon receiving the interrupt, the CPU must read the frame (accessing a message buffer in the 0x80 address) and then clear the interrupt. The act of clearing the interrupt triggers the FIFO engine to replace the message buffer in 0x80 with the next frame in the queue, and then issue another interrupt to the CPU. If the FIFO is full and more frames continue to be received, an OVERFLOW interrupt is issued to the CPU and subsequent frames are not accepted until the CPU creates space in the FIFO by reading one or more frames. A warning interrupt is also generated when 5 frames are accumulated in the FIFO.

A powerful filtering scheme is provided to accept only frames intended for the target application, thus reducing the interrupt servicing work load. The filtering criteria is specified by programming a table of 8 32-bit registers that can be configured to one of the following formats (see also *Section 29.4.4, Rx FIFO structure*):

- Format A: 8 extended or standard IDs (including IDE and RTR)
- Format B: 16 standard IDs or 16 extended 14-bit ID slices (including IDE and RTR)
- Format C: 32 standard or extended 8-bit ID slices

Note: A chosen format is applied to all 8 registers of the filter table. It is not possible to mix formats within the table.

The eight elements of the filter table are individually affected by the first eight Individual Mask Registers (RXIMR0 – RXIMR7), allowing very powerful filtering criteria to be defined. The rest of the RXIMR, starting from RXIM8, continue to affect the regular message buffers, starting from MB8. If the MBFEN bit is negated (or if the RXIMR are not available for the particular MCU), then the FIFO filter table is affected by the legacy mask registers as follows: element 6 is affected by RX14MASK, element 7 is affected by RX15MASK and the other elements (0 to 5) are affected by RXGMASK.

Precautions when using Global Mask and Individual Mask registers

Table 689. Recommended FEN and BCC settings

Case	MCR[FEN] RxFIFO	MCR[BCC] Rx Individual Mask	Notes
Case 1	FEN = 0	BCC = 0	RXGMASK, RX14MASK, and RX15MASK can safely be used. This allows backwards compatibility to older devices (e.g., devices without the individual masks feature). In this case, individual masks are not used.
Case 2	FEN = 1	BCC = 0	1st alternative: Do not use RXGMASK, RX14MASK, and RX15MASK in this case, leave the masks in their reset state.
Case 3	FEN = 1	BCC = 0	2nd alternative: Do not configure any MB as Rx (i.e., let all MBs as either Tx or inactive). In this case, RXGMASK, RX14MASK, and RX15MASK can be used to affect ID Tables without affecting the filtering process for Rx MBs.
Case 4	Don’t care	BCC = 1	If MCR[BCC] = 1, then the RXIMRs are enabled. Thus, RXGMASK, RX14MASK, and RX15MASK are not used. Particularly, when MCR[FEN] = 0, RxFIFO is disabled; RXGMASK, RX14MASK, and RX15MASK do not affect filtering. Individual masks are used.

29.5.8 CAN protocol related features

Remote frames

Remote frame is a special kind of frame. The user can program a message buffer to be a Request Remote Frame by writing the message buffer as Transmit with the RTR bit set to ‘1’. After the Remote Request frame is transmitted successfully, the message buffer becomes a Receive Message Buffer, with the same ID as before.

When a Remote Request frame is received by FlexCAN, its ID is compared to the IDs of the transmit message buffers with the Code field ‘1010’. If there is a matching ID, then this message buffer frame will be transmitted. Note that if the matching message buffer has the RTR bit set, then FlexCAN will transmit a Remote Frame as a response.

A received Remote Request Frame is not stored in a receive buffer. It is only used to trigger a transmission of a frame in response. The mask registers are not used in remote frame matching, and all ID bits (except RTR) of the incoming received frame should match.

In the case that a Remote Request Frame was received and matched a message buffer, this message buffer immediately enters the internal arbitration process, but is considered as normal Tx message buffer, with no higher priority. The data length of this frame is independent of the DLC field in the remote frame that initiated its transmission.

If the Rx FIFO is enabled (bit FEN set in MCR), FlexCAN will not generate an automatic response for Remote Request Frames that match the FIFO filtering criteria. If the remote frame matches one of the target IDs, it will be stored in the FIFO and presented to the CPU. Note that for filtering formats A and B, it is possible to select whether remote frames are accepted or not. For format C, remote frames are always accepted (if they match the ID).

Overload frames

FlexCAN does transmit overload frames due to detection of following conditions on CAN bus:

- Detection of a dominant bit in the first/second bit of Intermission
- Detection of a dominant bit at the 7th bit (last) of End of Frame field (Rx frames)
- Detection of a dominant bit at the 8th bit (last) of Error Frame Delimiter or Overload Frame Delimiter

Time stamp

The value of the Free Running Timer is sampled at the beginning of the Identifier field on the CAN bus, and is stored at the end of “move-in” in the TIME STAMP field, providing network behavior with respect to time.

Note that the Free Running Timer can be reset upon a specific frame reception, enabling network time synchronization. Refer to TSYN description in *Section , Control Register (CR)*.

Protocol timing

Figure 692 shows the structure of the clock generation circuitry that feeds the CAN Protocol Interface (CPI) submodule. The clock source bit (CLKSRC) in the CR Register defines whether the internal clock is connected to the output of a crystal oscillator (Oscillator Clock) or to the Peripheral Clock (generally from a PLL). In order to guarantee reliable operation, the clock source should be selected while the module is in Disable Mode (bit MDIS set in the Module Configuration Register).

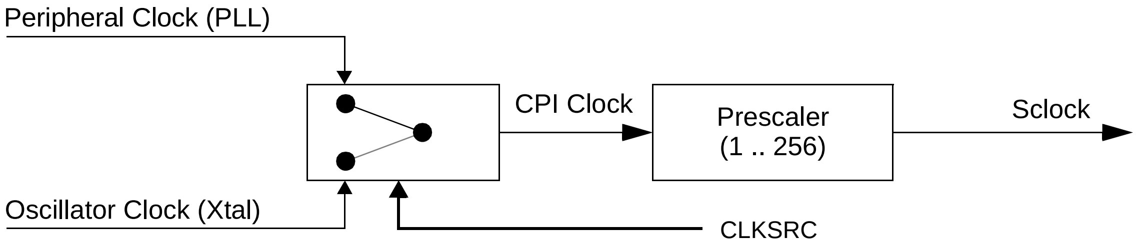


Figure 692. CAN engine clocking scheme

The crystal oscillator clock should be selected whenever a tight tolerance (up to 0.1%) is required in the CAN bus timing. The crystal oscillator clock has better jitter performance than PLL generated clocks.

The FlexCAN module supports a variety of means to setup bit timing parameters that are required by the CAN protocol. The Control Register has various fields used to control bit timing parameters: PRESDIV, PROPSEG, PSEG1, PSEG2 and RJW. See *Section , Control Register (CR)*.

The PRESDIV field controls a prescaler that generates the Serial Clock (Sclock), whose period defines the ‘time quantum’ used to compose the CAN waveform. A time quantum is the atomic unit of time handled by the CAN engine.

$$f_{Tq} = \frac{f_{CANCLK}}{(Prescaler\ \ value)}$$

A bit time is subdivided into three segments[bd] (reference *Figure 693* and *Table 690*):

- SYNC_SEG: This segment has a fixed length of one time quantum. Signal edges are expected to happen within this section
- Time Segment 1: This segment includes the Propagation Segment and the Phase Segment 1 of the CAN standard. It can be programmed by setting the PROPSEG and the PSEG1 fields of the CR Register so that their sum (plus 2) is in the range of 4 to 16 time quanta
- Time Segment 2: This segment represents the Phase Segment 2 of the CAN standard. It can be programmed by setting the PSEG2 field of the CR Register (plus 1) to be 2 to 8 time quanta long

$$Bit\ \ Rate = \frac{f_{Tq}}{(number\ \ of\ \ Time\ \ Quanta)}$$

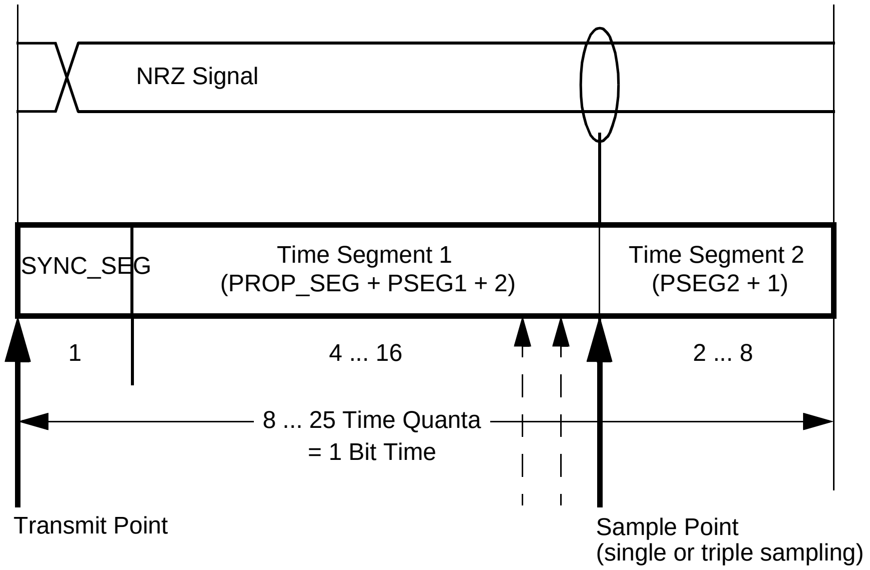


Figure 693. Segments within the Bit Time

[bd] For further explanation of the underlying concepts please refer to ISO/DIS 11519–1, Section 10.3. Reference also the Bosch CAN 2.0A/B protocol specification dated September 1991 for bit timing.

Table 690. Time Segment Syntax

Syntax	Description
SYNC_SEG	System expects transitions to occur on the bus during this period.
Transmit Point	A node in transmit mode transfers a new value to the CAN bus at this point.
Sample Point	A node samples the bus at this point. If the three samples per bit option is selected, then this point marks the position of the third sample.

Table 691 gives an overview of the CAN compliant segment settings and the related parameter values.

Table 691. CAN standard compliant bit time segment settings

Time segment 1	Time segment 2	Resynchronization jump width
5 .. 10	2	1 .. 2
4 .. 11	3	1 .. 3
5 .. 12	4	1 .. 4
6 .. 13	5	1 .. 4
7 .. 14	6	1 .. 4
8 .. 15	7	1 .. 4
9 .. 16	8	1 .. 4

Note: It is the user's responsibility to ensure the bit time settings are in compliance with the CAN standard. For bit time calculations, use an IPT (Information Processing Time) of 2, which is the value implemented in the FlexCAN module.

Arbitration and matching timing

During normal transmission or reception of frames, the arbitration, matching, move-in and move-out processes are executed during certain time windows inside the CAN frame, as shown in *Figure 694*.

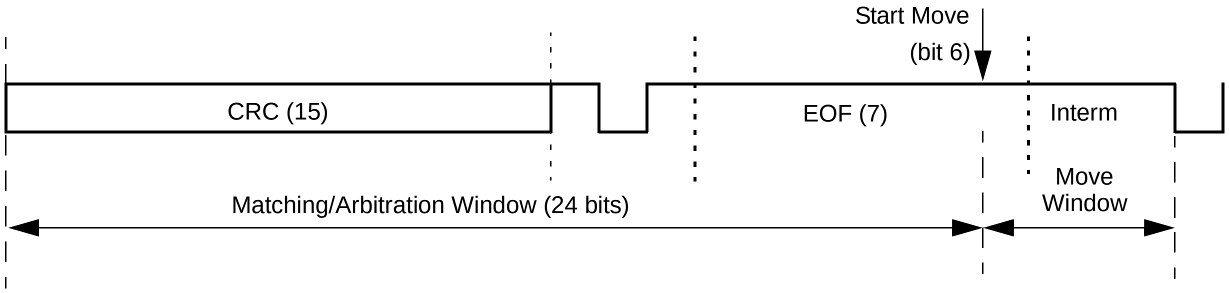


Figure 694. Arbitration, match and move time windows

When doing matching and arbitration, FlexCAN needs to scan the whole Message Buffer memory during the available time slot. In order to have sufficient time to do that, the following requirements must be observed:

- A valid CAN bit timing must be programmed, as indicated in *Table 691*
- The peripheral clock frequency can not be smaller than the oscillator clock frequency, i.e. the PLL can not be programmed to divide down the oscillator clock
- There must be a minimum ratio between the peripheral clock frequency and the CAN bit rate, as specified in *Table 692*

Table 692. Minimum Ratio Between Peripheral Clock Frequency and CAN Bit Rate

Number of Message Buffers	Minimum Ratio
16	8
32	8
64	16

A direct consequence of the first requirement is that the minimum number of time quanta per CAN bit must be 8, so the oscillator clock frequency should be at least 8 times the CAN bit rate. The minimum frequency ratio specified in *Table 692* can be achieved by choosing a high enough peripheral clock frequency when compared to the oscillator clock frequency, or by adjusting one or more of the bit timing parameters (PRESDIV, PROPSEG, PSEG1, PSEG2). As an example, taking the case of 64 message buffers, if the oscillator and peripheral clock frequencies are equal and the CAN bit timing is programmed to have 8 time quanta per bit, then the prescaler factor (PRESDIV + 1) should be at least 2. For prescaler factor equal to one and CAN bit timing with 8 time quanta per bit, the ratio between peripheral and oscillator clock frequencies should be at least 2.

29.5.9 Modes of operation details

Freeze Mode

This mode is entered by asserting MCR[HALT] or when the MCU is put into Debug Mode. In both cases it is also necessary that MCR[FRZ] is asserted and the module is not in either of the low power modes (Disable or Stop). When Freeze Mode is requested during transmission or reception, FlexCAN does the following:

- Waits to be in either Intermission, Passive Error, Bus Off or Idle state
- Waits for all internal activities like arbitration, matching, move-in and move-out to finish
- Ignores the Rx input pin and drives the Tx pin as recessive
- Stops the prescaler, thus halting all CAN protocol activities
- Grants write access to the Error Counters Register, which is read-only in other modes
- Sets the NOTRDY and FRZACK bits in MCR

After requesting Freeze Mode, the user must wait for MCR[FRZACK] to be asserted before executing any other action, otherwise FlexCAN may operate in an unpredictable way. In Freeze mode, all memory mapped registers are accessible.

Exiting Freeze Mode is done in one of the following ways:

- CPU negates MCR[FRZ]
- The MCU is removed from Debug Mode and/or the HALT bit is negated

Once out of Freeze Mode, FlexCAN tries to resynchronize to the CAN bus by waiting for 11 consecutive recessive bits.

Module Disable Mode

This low power mode is entered when the MDIS bit in the MCR Register is asserted. If the module is disabled during Freeze Mode, the module sends a request to disable the clocks to the CAN Protocol Interface (CPI) and Message Buffer Management (MBM) sub-modules, sets the LPM_ACK bit and negates the FRZ_ACK bit. If the module is disabled during transmission or reception, FlexCAN does the following:

- Waits to be in either Idle or Bus Off state, or else waits for the third bit of Intermission and then checks it to be recessive
- Waits for all internal activities like arbitration, matching, move-in and move-out to finish
- Ignores its Rx input pin and drives its Tx pin as recessive
- Shuts down the clocks to the CPI and MBM submodules
- Sets the NOTRDY and MDISACK bits in MCR

The Bus Interface Unit continues to operate, enabling the CPU to access memory mapped registers, except the Free Running Timer, the Error Counter Register and the Message Buffers, which cannot be accessed when the module is in Disable Mode. Exiting from this mode is done by negating MCR[MDIS], which will resume the clocks and negate MCR[MDISACK].

Stop Mode

This is a system low power mode in which all MCU clocks are stopped for maximum power savings. If FlexCAN receives the global Stop Mode request during Freeze Mode, it sets MCR[MDISACK], negates MCR[FRZACK] and then sends a Stop Acknowledge signal to the CPU, in order to shut down the clocks globally. If Stop Mode is requested during transmission or reception, FlexCAN does the following:

- Waits to be in either Idle or Bus Off state, or else waits for the third bit of Intermission and checks it to be recessive
- Waits for all internal activities like arbitration, matching, move-in and move-out to finish
- Ignores its Rx input pin and drives its Tx pin as recessive
- Sets the NOTRDY and MDISACK bits in MCR
- Sends a Stop Acknowledge signal to the CPU, so that it can shut down the clocks globally

Exiting Stop Mode is done in one of the following ways:

- CPU resuming the clocks and removing the Stop Mode request
- CPU resuming the clocks and Stop Mode request as a result of the Self Wake mechanism

In the Self Wake mechanism, if MCR[SLF_WAK] was set at the time FlexCAN entered Stop Mode, then upon detection of a recessive to dominant transition on the CAN bus, FlexCAN sets ESR[WAKINT] and, if enabled by MCR[WAK_MSK], generates a Wake Up interrupt to the CPU. Upon receiving the interrupt, the CPU should resume the clocks and remove the Stop Mode request. FlexCAN will then wait for 11 consecutive recessive bits to synchronize to the CAN bus. As a consequence, it will not receive the frame that woke it up. *Table 693* details the effect of MCR[SLF_WAK] and MCR[WAK_MSK] upon wake-up from Stop Mode. Note that wake-up from Stop Mode only works when both bits are asserted.

Table 693. Wake-up from Stop Mode

SLF_WAK	WAK_MSK	MCU clocks enabled	Wake-up interrupt generated
0	0	No	No
0	1	No	No
1	0	No	No
1	1	Yes	Yes

The sensitivity to CAN bus activity can be modified by applying a low-pass filter function to the Rx CAN input line while in Stop Mode. See the WAK_SRC bit in *Section , Module Configuration Register (MCR)*. This feature can be used to protect FlexCAN from waking up due to short glitches on the CAN bus lines. Such glitches can result from electromagnetic interference within noisy environments.

29.5.10 Interrupts

The module can generate up to 70 interrupt sources (64 interrupts due to message buffers and 6 interrupts due to Ored interrupts from message buffers, Bus Off, Error, Tx Warning, Rx Warning and Wake Up). The number of actual sources depends on the configured number of Message Buffers.

Each one of the message buffers can be an interrupt source, if its corresponding bit in the IMRL or IMRH register is set. There is no distinction between Tx and Rx interrupts for a particular buffer, under the assumption that the buffer is initialized for either transmission or reception. Each of the buffers has assigned a flag bit in the IFRL or IFRH register. The bit is set when the corresponding buffer completes a successful transmission/reception and is cleared when the CPU writes it to ‘1’ (unless another interrupt is generated at the same time).

Note: It must be guaranteed that the CPU only clears the bit causing the current interrupt. For this reason, bit manipulation instructions (BSET) must not be used to clear interrupt flags. These instructions may cause accidental clearing of interrupt flags which are set after entering the current interrupt service routine.

If the Rx FIFO is enabled (MCR[FEN] set), the interrupts corresponding to MBs 0 to 7 have a different behavior. Bit 7 of the IFRL becomes the “FIFO Overflow” flag; bit 6 becomes the FIFO Warning flag, bit 5 becomes the “Frames Available in FIFO flag” and bits 4–0 are unused. See *Section , Interrupt Flags 1 Register (IFRL)* for more information.

A combined interrupt for all message buffers is also generated by an Or of all the interrupt sources from message buffers. This interrupt gets generated when any of the message buffers generates an interrupt. In this case the CPU must read the IFRL or IFRH register to determine which message buffer caused the interrupt.

The other five interrupt sources (Bus Off, Error, Tx Warning, Rx Warning and Wake Up) generate interrupts like the message buffer ones, and can be read from the Error and Status Register. The Bus Off, Error, Tx Warning and Rx Warning interrupt mask bits are located in the Control Register, and the Wake-Up interrupt mask bit is located in the MCR.

29.5.11 Bus interface

The CPU access to FlexCAN registers are subject to the following rules:

- Read and write access to supervisor registers in User Mode results in access error.
- Read and write access to unimplemented or reserved address space also results in access error. Any access to unimplemented message buffer or Rx Individual Mask Register locations results in access error. Any access to the Rx Individual Mask Register space when MCR[MBFEN] is negated results in access error.
- If MCR[MAXMB] is programmed with a value smaller than the available number of message buffers, then the unused memory space can be used as general purpose RAM space. Note that the Rx Individual Mask Registers can only be accessed in Freeze Mode, and this is still true for unused space within this memory. Note also that reserved words within RAM cannot be used. As an example, suppose FlexCAN is configured with 64 message buffers and MCR[MAXMB] is programmed with zero. The maximum number of message buffers in this case becomes one. The message buffer memory starts at 0x0060, but the space from 0x0060 to 0x007F is reserved (for SMB usage), and the space from 0x0080 to 0x008F is used by the one message buffer. This leaves us with the available space from 0x0090 to 0x047F. The available memory in the Mask Registers space would be from 0x0884 to 0x097F.

Note: Unused message buffer space must not be used as general purpose RAM while FlexCAN is transmitting and receiving CAN frames.

29.6 Initialization/Application information

This section provide instructions for initializing the FlexCAN module.

29.6.1 FlexCAN initialization sequence

The FlexCAN module may be reset in three ways:

- MCU level hard reset, which resets all memory mapped registers asynchronously
- MCU level soft reset, which resets some of the memory mapped registers synchronously (refer to *Table 673* to see what registers are affected by soft reset)
- SOFTRST bit in MCR, which has the same effect as the MCU level soft reset

Soft reset is synchronous and has to follow an internal request/acknowledge procedure across clock domains. Therefore, it may take some time to fully propagate its effects. The SOFTRST bit remains asserted while soft reset is pending, so software can poll this bit to know when the reset has completed. Also, soft reset can not be applied while clocks are shut down in any of the low power modes. The low power mode should be exited and the clocks resumed before applying soft reset.

The clock source (CLKSRC bit) should be selected while the module is in Disable Mode. After the clock source is selected and the module is enabled (MDIS bit negated), FlexCAN automatically goes to Freeze Mode. In Freeze Mode, FlexCAN is unsynchronized to the CAN bus, the HALT and FRZ bits in the MCR are set, the internal state machines are disabled and the FRZACK and NOTRDY bits in the MCR are set. The Tx pin is in recessive state and FlexCAN does not initiate any transmission or reception of CAN frames. Note that the Message Buffers and the Rx Individual Mask Registers are not affected by reset, so they are not automatically initialized.

For any configuration change/initialization it is required that FlexCAN is put into Freeze Mode (see *Section , Freeze Mode*). The following is a generic initialization sequence applicable to the FlexCAN module:

- Initialize the Module Configuration Register (MCR)
  - Enable the individual filtering per message buffer and reception queue features by setting the MBFEN bit
  - Enable the warning interrupts by setting the WRNEN bit
  - If required, disable frame self reception by setting the SRX_DIS bit
  - Enable the FIFO by setting the FEN bit
  - Enable the abort mechanism by setting the AEN bit
  - Enable the local priority feature by setting the LPRIO_EN bit
- Initialize the Control Register (CR)
  - Determine the bit timing parameters: PROPSEG, PSEG1, PSEG2, RJW
  - Determine the bit rate by programming the PRESDIV field
  - Determine the internal arbitration mode (bit CR[LBUF])
- Initialize the Message Buffers
  - The Control and Status word of all Message Buffers must be initialized
  - If FIFO was enabled, the 8-entry ID table must be initialized
  - Other entries in each Message Buffer should be initialized as required
- Initialize the Rx Individual Mask Registers
- Set required interrupt mask bits in the corresponding IMRL or IMRH register (for all message buffer interrupts), in the CR (for Bus Off and Error interrupts) and in the MCR for Wake-Up interrupt
- Negate the HALT bit in MCR

Starting with the last event, FlexCAN attempts to synchronize to the CAN bus.

29.6.2 FlexCAN addressing and RAM size configurations

There are three RAM configurations that can be implemented within the FlexCAN module. The possible configurations are:

- For 16 message buffers: 288 bytes for message buffer memory and 64 bytes for Individual Mask Registers
- For 32 message buffers: 544 bytes for message buffer memory and 128 bytes for Individual Mask Registers
- For 64 message buffers: 1056 bytes for message buffer memory and 256 bytes for Individual Mask Registers

In each configuration the user can program the maximum number of message buffers that will take part in the matching and arbitration processes using field MCR[MAXMB]:

- For 16 message buffer configuration, MCR[MAXMB] can be any number between 0–15.
- For 32 message buffer configuration, MCR[MAXMB] can be any number between 0–31.
- For 64 message buffer configuration, MCR[MAXMB] can be any number between 0–63.

30 Periodic Interrupt Timer (PIT_RTI)

30.1 Information specific to this device

This section presents device-specific parameterization and customization information not specifically referenced in the remainder of this chapter.

30.1.1 Device-specific features

- 1 Periodic Interrupt Timer Module (PIT/RTI)
  - 32-bit counter
  - 4 Timer Channels
  - 1 Real Time Interrupt (RTI): timer channel clocked from the crystal oscillator that can be used to wake the part from stop mode.
- The counter period of a running timer can be modified, by first disabling the timer, setting a new load value and then enabling the timer again (see *Figure 702*). In the case of the RTI, because of the different clock domains (system clock / oscillator clock), a delay must be respected between setting the new value and re-enabling the RTI. (Modification to *Section , Timers / RTI*).

30.2 Introduction

Figure 695 shows the PIT block diagram.

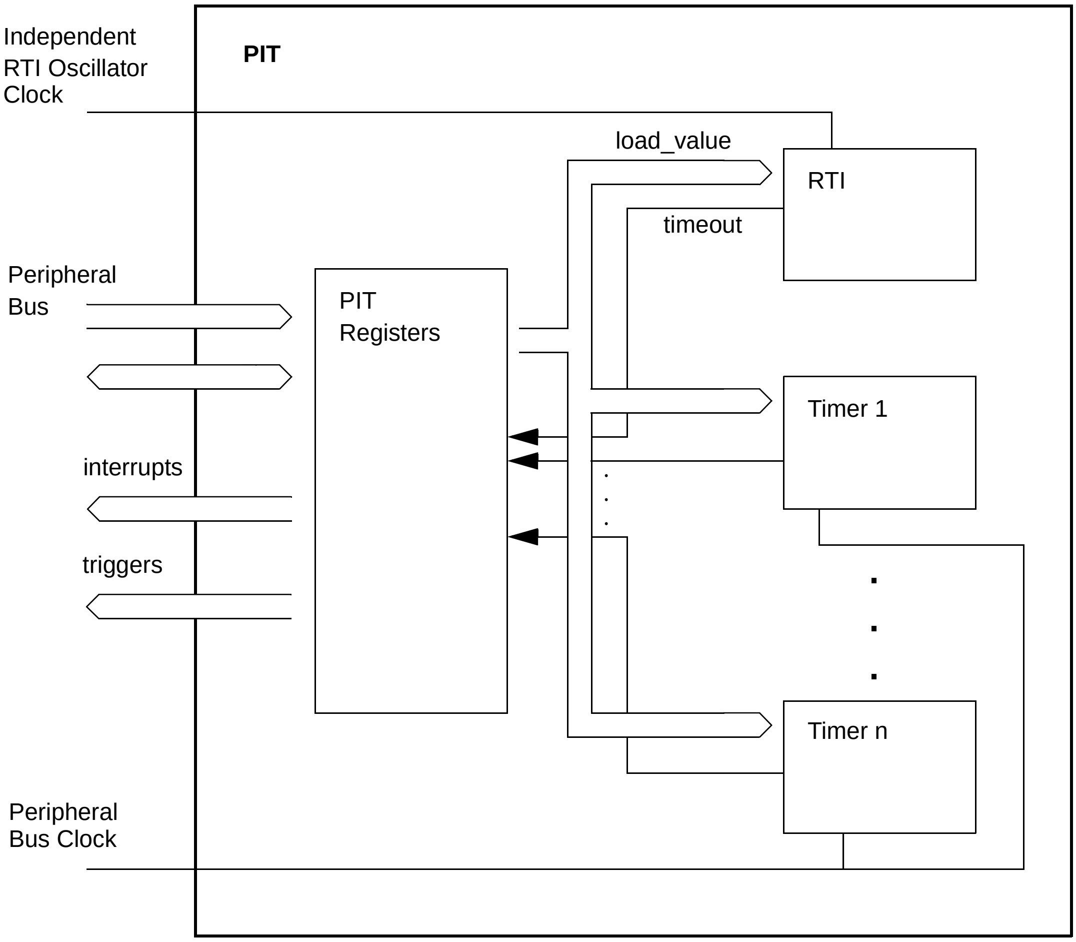


Figure 695. Block diagram of PIT_RTI

30.2.1 Overview

This specification describes the function of the Periodic Interrupt Timer block (PIT_RTI). The PIT is an array of timers that can be used to raise interrupts and trigger DMA channels. Real Time Interrupt Timer (RTI) is a dedicated interrupt timer, which runs on a separate clock and can be used for system wakeup.

30.2.2 Features

The main features of this block are:

- Timers can generate DMA trigger pulses
- Timers can generate interrupts
- All interrupts are maskable
- Independent timeout periods for each timer

30.3 Signal description

The PIT module has no external pins.

30.4 Memory map and register description

This section provides a detailed description of all registers accessible in the PIT_RTI module.

30.4.1 Memory map

Table 694 gives an overview of all PIT_RTI registers.

Table 694. PIT_RTI memory map

Address offset	Use	Access	Location
0x000	PIT Module Control Register (PITMCR)	R/W	*on page 30-1279*
0x004–0x0EC	Reserved	R	—
0x0F0–0x0FC	RTI channel	(1)	
0x100–0x10C	Timer Channel 0	(1)	
0x110–0x11C	Timer Channel 1	(1)	
0x120–0x12C	Timer Channel 2	(1)	
0x130–0x13C	Timer Channel 3	(1)	

1. See *Table 695*.

Table 695. Timer channel n / RTI channel registers

Address offset	Use	Access	Location
Channel + 0x00	Timer Load Value Register n (LDVALn)	R/W	*on page 30-1280*
Channel + 0x04	Current Timer Value Register n (CVALn)	R	*on page 30-1281*
Channel + 0x08	Timer Control Register n (TCTRLn)	R/W	*on page 30-1282*
Channel + 0x0C	Timer Flag Register n (TFLGn)	R/W	*on page 30-1283*

Note: Register Address = Base Address + Address Offset, where the Base Address is defined at the MCU level and the Address Offset is defined at the module level.

Note: Reserved registers will read as 0, writes will have no effect.

30.4.2 Register descriptions

This section describes in address order all the PIT_RTI registers and their individual bits.

PIT Module Control Register (PITMCR)

This register controls whether the timer clocks should be enabled and whether the timers should run in debug mode.

Figure 696. PIT Module Control Register (PITMCR)

Offset 0x000 Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	MDIS_RTI	MDIS	FRZ
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 696. PITMCR field descriptions

Field	Description
MDIS_RTI	Module Disable—RTI section This is used to disable the RTI timer. This bit should be enabled before any RTI setup is done. 0: Clock for RTI is enabled (default) 1: Clock for RTI disabled
MDIS	Module Disable—PIT section This is used to disable the standard timers. The RTI timer is not affected by this bit. This bit should be enabled before any other setup is done. 0: Clock for PIT Timers is enabled (default) 1: Clock for PIT Timers is disabled
FRZ	Freeze Allows the timers to be stopped when the device enters debug mode. 0: Timers continue to run in debug mode. 1: Timers are stopped in debug mode.

Timer Load Value Register *n* (LDVAL*n*)

These registers select the timeout period for the timer interrupts. In the case of the RTI, it will take several cycles until this value is synchronized into the RTI clock domain. For all other timers the value change is visible immediately. The synchronization mechanism allows 0 wait states in this case.

Figure 697. Timer Load Value Register (LDVAL)

Offset channel_base + 0x00 Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	TSV31	TSV30	TSV29	TSV28	TSV27	TSV26	TSV25	TSV24	TSV23	TSV22	TSV21	TSV20	TSV19	TSV18	TSV17	TSV16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	TSV15	TSV14	TSV13	TSV12	TSV11	TSV10	TSV9	TSV8	TSV7	TSV6	TSV5	TSV4	TSV3	TSV2	TSV1	TSV0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 697. LDVAL field descriptions

Field	Description
TSV*n*	Time Start Value Bits These bits set the timer start value. The timer will count down until it reaches 0, then it will generate an interrupt and load this register value again. Writing a new value to this register will not restart the timer, instead the value will be loaded once the timer expires. To abort the current cycle and start a timer period with the new value, the timer must be disabled and enabled again (see *Figure 702*). NOTE: For the RTI, the timer should not be set to a value lower than 32 cycles, otherwise interrupts may be lost, as it takes several cycles to clear the RTI interrupt. For the other timers, this limit does not apply, however there will be practical limits, since the processor will require several cycles to service an interrupt.

Current Timer Value Register *n* (CVAL*n*)

These registers indicate the current timer position. In the case of the RTI, this will show a value which is several cycles old, since it originates from a potentially different clock domain.

Figure 698. Current Timer Value Register (CVAL)

Offset channel_base + 0x04 Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TVL31	TVL30	TVL29	TVL28	TVL27	TVL26	TVL25	TVL24	TVL23	TVL22	TVL21	TVL20	TVL19	TVL18	TVL17	TVL16
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	TVL15	TVL14	TVL13	TVL12	TVL11	TVL10	TVL9	TVL8	TVL7	TVL6	TVL5	TVL4	TVL3	TVL2	TVL1	TVL0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 698. CVAL field descriptions

Field	Description
TVL*n*	Current Timer Value These bits represent the current timer value. Note that the timer uses a downcounter. NOTE: The timer values will be frozen in Debug mode if the FRZ bit is set in the PIT Module Control Register (see *Figure 696*).

Timer Control Register *n* (TCTRL*n*)

These registers contain the control bits for each timer.

Figure 699. Timer Control Register (TCTRL)

Offset channel_base + 0x08 Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	TIE	TEN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 699. TCTRL field descriptions

Field	Description
TIE	Timer Interrupt Enable Bit 0: Interrupt requests from Timer x are disabled 1: Interrupt will be requested whenever TIF is set When an interrupt is pending (TIF set), enabling the interrupt will immediately cause an interrupt event. To avoid this, the associated TIF flag must be cleared first.
TEN	Timer Enable Bit 0: Timer will be disabled 1: Timer will be active

Timer Flag Register *n* (TFLG*n*)

These registers hold the PIT interrupt flags.

Figure 700. Timer Flag Register (TFLG)

Offset channel_base + 0x0C Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	TIF
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 700. TFLG field descriptions

Field	Description
TIF	Time Interrupt Flag TIF is set to 1 at the end of the timer period.This flag can be cleared only by writing it with a 1. Writing a 0 has no effect. If enabled (TIE = 1), TIF causes an interrupt request. 0: Timeout has not yet occurred 1: Timeout has occurred

30.5 Functional description

30.5.1 General

This section gives detailed information on the internal operation of the module. Each timer can be used to generate trigger pulses as well as to generate interrupts, each interrupt will be available on a separate interrupt line. Additionally the RTI timer can be used to wakeup the processor.

Timers / RTI

The timers generate triggers at periodic intervals, when enabled. They load their start values, as specified in their LDVAL registers, then count down until they reach 0. Then they load their respective start value again. Each time a timer reaches 0, it will generate a trigger pulse, and set the interrupt flag.

All interrupts can be enabled or masked (by setting the TIE bits in the TCTRL registers). A new interrupt can be generated only after the previous one is cleared. Since in the case of the RTI, clearing the interrupt crosses clock domains, a minimum load value of 32 should be maintained.

If desired, the current counter value of the timer can be read via the CVAL registers. The value of the RTI counter can be delayed considerably, as it is synchronized to the bus clock from the RTI clock domain.

The counter period can be restarted, by first disabling, then enabling the timer with the TEN bit (see *Figure 701*).

The counter period of a running timer can be modified, by first disabling the timer, setting a new load value and then enabling the timer again (see *Figure 702*).

It is also possible to change the counter period without restarting the timer by writing the LDVAL register with the new load value. This value will then be loaded after the next trigger event (see *Figure 703*).

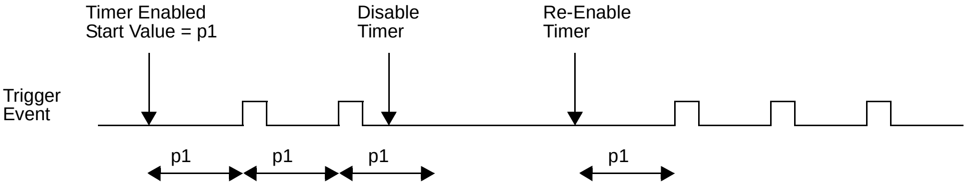


Figure 701. Stopping and starting a timer

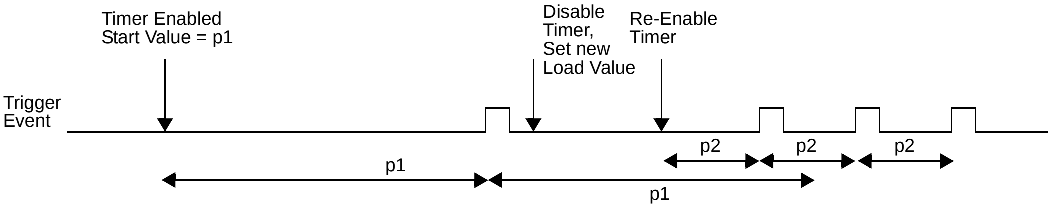


Figure 702. Modifying running timer period

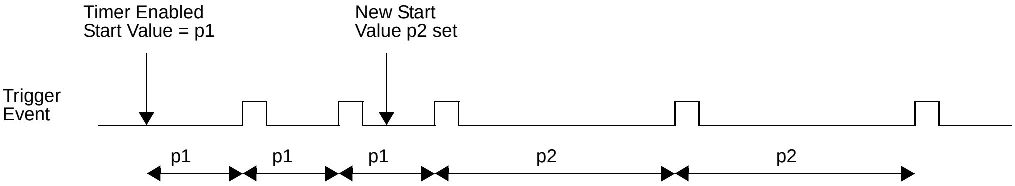


Figure 703. Dynamically setting a new load value

Debug mode

In Debug mode the timers will be frozen—this is intended to aid software development, allowing the developer to halt the processor, investigate the current state of the system (e.g. the timer values) and then continue the operation.

30.5.2 Interrupts

All of the timers support interrupt generation. The RTI is typically used for system wakeup, but can be used for interrupt generation as well. Refer to the section “Functional description” in *Chapter 14: Interrupt Controller (INTC)* for related vector addresses and priorities.

Timer interrupts can be disabled by setting the TIE bits to zero. The timer interrupt flags (TIF) are set to 1 when a timeout occurs on the associated timer, and are cleared to 0 by writing a 1 to that TIF bit.

The PIT_RTI generates a real time interrupt when the selected interrupt time period elapses. The RTI interrupt is disabled locally by setting the TIE bit to zero. The real time interrupt flag (TIF) is set to 1 when a timeout occurs, and is cleared by writing a 1 to the TIF bit. (The flag will be set regardless of whether the interrupt is enabled.)

The RTI can be used for periodic wakeup from a low power mode. It can also be used to generate a general purpose interrupt.

30.6 Initialization and application information

30.6.1 Example configuration

In the example configuration

- the PIT clock has a frequency of 50 MHz
- the RTI clock has a frequency of 10 MHz
- the RTI shall be set up to create a wakeup interrupt every 500 ms
- timer 1 shall create an interrupt every 5.12 ms
- timer 3 shall create a trigger event every 30 ms

First the PIT module needs to be activated by writing a 0 to the MDIS bit in the PITMCR.

The 50 MHz clock frequency equates to a clock period of 20 ns and the 10 MHz frequency equates to a clock period of 100 ns. Therefore the RTI timer needs to trigger every 500 ms/100 ns = 5000000 cycles. Timer 1 needs to trigger every 5.12 ms/20 ns = 256000 cycles and timer 3 every 30 ms/20 ns = 1500000 cycles. The value for the LDVAL register trigger would be calculated as (period / clock period) – 1.

This means that RTI LDVAL will be written with 0x004C_4B3F, LDVAL1 with 0x0003_E7FF and LDVAL3 with 0x0016_E35F.

To generate the wakeup interrupt, the interrupt line must be enabled by writing a 1 to the RTI TIE bit in the TCTRL register. To start the RTI, the TEN bit in the RTI TCTRL register must also be set.

The interrupt for Timer 1 is enabled by setting the TIE bit in the TCTRL1 register. The timer is started by writing a 1 to bit TEN in the TCTRL1 register.

Timer 3 shall be used only for triggering. Therefore Timer 3 is started by writing a 1 to bit TEN in the TCTRL3 register, bit TIE stays at 0.

The following example code matches the described setup:

```
// turn on PIT
PIT_CTRL = 0x00;

// RTI
PIT_RTI_LDVAL = 0x004C4B3F; // setup RTI for 5000000 cycles
PIT_RTI_TCTRL = PIT_TIE; // let RTI generate interrupts
PIT_RTI_TCTRL |= PIT_TEN; // start RTI

// Timer 1
PIT_LDVAL1 = 0x0003E7FF; // setup timer 1 for 256000 cycles
PIT_TCTRL1 = TIE; // enable Timer 1 interrupts
PIT_TCTRL1 |= TEN; // start timer 1

// Timer 3
PIT_LDVAL3 = 0x0016E35F; // setup timer 3for 1500000 cycles
PIT_TCTRL3 = TEN; // start timer 3
```

31 FlexRay Communication Controller (FlexRay)

31.1 Introduction

31.1.1 Reference

The following documents are referenced.

- FlexRay Communications System Protocol Specification, Version 2.1 Rev A(be)
- FlexRay Communications System Electrical Physical Layer Specification, Version 2.1 Rev A

31.1.2 Glossary

This section provides a list of terms used in this chapter.

---

be. The FlexRay Specifications have been developed for automotive applications.The FlexRay Specifications have been neither developed nor tested for non-automotive applications.

Table 701. List of terms

Term	Definition
BCU	Buffer Control Unit—Handles message buffer access
BMIF	Bus Master Interface—Provides master access to FlexRay memory area
CC	FlexRay Communication Controller—Module described in this chapter
CDC	Clock Domain Crosser
CHI	Controller Host Interface
Cycle length in µT	The actual length of a cycle in µT for the ideal CC (+/- 0 ppm)
FlexRay memory area	Memory area to store the physical message buffer payload data, frame header, frame andslot status, and synchronization frame related tables
System memory	Memory that contains the FlexRay memory area
System Bus	Bus that connects the CC and system memory
FSS	Frame Start Sequence
HIF	Host Interface—Provides host access to the CC
Host	The FlexRay CC host CPU

Table 701. List of terms (continued)

Term	Definition
LUT	Look Up Table—Stores message buffer header index value
MB	Message Buffer
MBIDX	Message Buffer Index—The position of a header field entry within the header area. If the header area is accessed as an array, this is the same as the array index of the entry.
MBNum	Message Buffer Number—Position of message buffer configuration registers within the register map. For example, Message Buffer Number 5 corresponds to the MBCCS5 register.
MCU	Microcontroller Unit
µT	Microtick
MT	Macrotick
MTS	Media Access Test Symbol
NIT	Network Idle Time
PE	Protocol Engine
POC	Protocol Operation Control—Each state of the POC is denoted by *POC:state*
Rx	Reception
SEQ	Sequencer Engine
TCU	Time Control Unit
Tx	Transmission
sync frame	Null frame or message frame with *Sync Frame Indicator* set to 1
startup frame	Null frame or message frame with both *Sync Frame Indicator* and *Startup Frame Indicator* set to 1
normal frame	Null frame or message frame with both *Sync Frame Indicator* and *Startup Frame Indicator* set to 0
null frame	Frame with *Null Frame Indicator* set to 0
message frame	Frame with *Null Frame Indicator* set to 1

31.1.3 Color coding

Throughout this chapter types of items are highlighted through the use of an italicized color font.

FlexRay protocol parameters, constants and variables are highlighted with *blue italics*. An example is the parameter *gdActionPointOffset*.

FlexRay protocol states are highlighted in *green italics*. An example is the state *POC:normal active*.

31.1.4 Overview

The CC is a FlexRay communication controller that implements the *FlexRay Communications System Protocol Specification, Version 2.1 Rev A.*

The CC has three main components:

- Controller host interface (CHI)
- Protocol engine (PE)
- Clock domain crossing unit (CDC)

A block diagram of the CC with its surrounding modules is given in *Figure 704*.

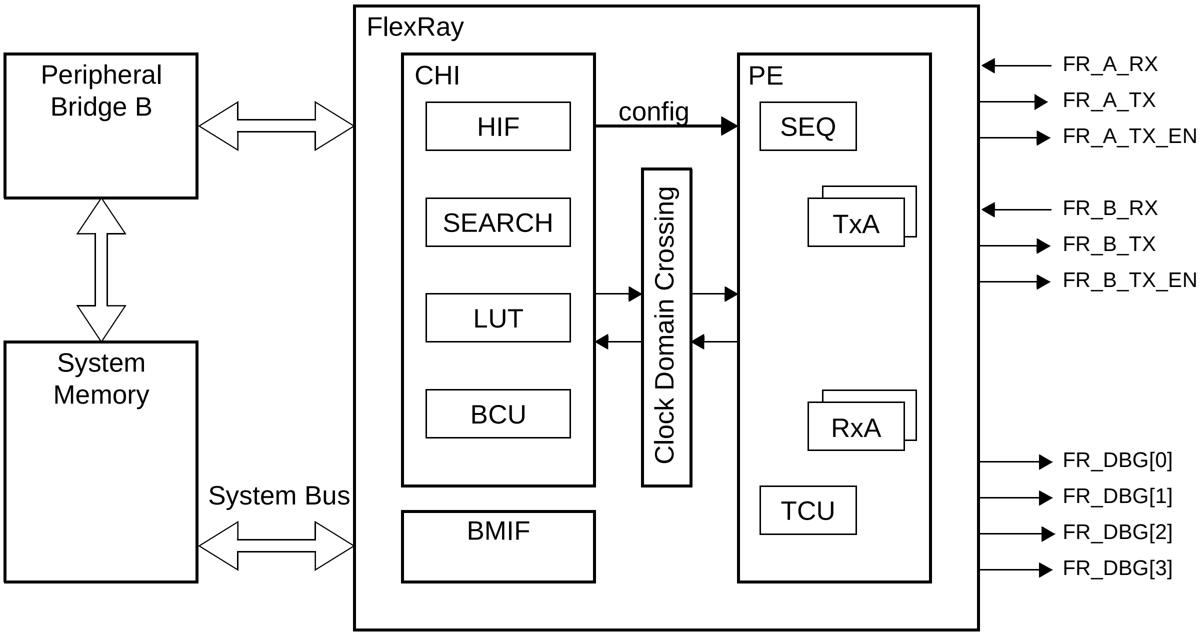


Figure 704. FlexRay block diagram

The protocol engine has two transmitter units TxA and TxB and two receiver units RxA and RxB for sending and receiving frames through the two FlexRay channels. The time control unit (TCU) is responsible for maintaining global clock synchronization to the FlexRay network. The overall activity of the PE is controlled by the sequencer engine (SEQ).

The CC host interface provides host access to the module’s configuration, control, and status registers, as well as to the message buffer configuration, control, and status registers. The message buffers themselves, which contain the frame header and payload data received or to be transmitted, and the slot status information, are stored in the FlexRay memory area.

The clock domain crossing unit implements signal crossing from the CHI clock domain to the PE clock domain and vice versa, to allow for asynchronous PE and CHI clock domains.

The CC stores the frame header and payload data of frames received or of frames to be transmitted in the FlexRay memory area. The application accesses the FlexRay memory area to retrieve and provide the frames to be processed by the CC. In addition to the frame header and payload data, the CC stores the synchronization frame related tables in the FlexRay memory area for application processing.

The FlexRay memory area is located in the system memory of the MCU. The CC has access to the FlexRay memory area via its bus master interface (BMIF). The host provides the start address of the FlexRay memory area within the system memory by programming the *System Memory Base Address Register (FR_SYMBADR)*. All FlexRay memory area related offsets are stored in offset registers. The physical address pointer into the flexray memory area of the MCU system memory is calculated using the offset values the FlexRay memory area base address.

Note: The CC does not provide a memory protection scheme for the FlexRay memory area.

31.1.5 Features

The CC provides the following features:

- *FlexRay Communications System Protocol Specification, Version 2.1 Rev A* compliant protocol implementation
- *FlexRay Communications System Electrical Physical Layer Specification, Version 2.1 Rev A* compliant bus driver interface
- Single channel support
  - FlexRay Port A can be configured to be connected either to physical FlexRay channel A or physical FlexRay channel B.
- FlexRay bus data rates of 10 Mbit/s, 8 Mbit/s, 5 Mbit/s, and 2.5 Mbit/s supported
- 128 configurable message buffers with
  - Individual frame ID filtering
  - Individual channel ID filtering
  - Individual cycle counter filtering
- Message buffer header, status and payload data stored in dedicated FlexRay memory area
  - Allows for flexible and efficient message buffer implementation
  - Consistent data access ensured by means of buffer locking scheme
  - Application can lock multiple buffers at the same time
- Size of message buffer payload data section configurable from 0 up to 254 bytes
- 2 independent message buffer segments with configurable size of payload data section
  - Each segment can contain message buffers assigned to the static segment and message buffers assigned to the dynamic segment at the same time
- Zero padding for transmit message buffers in static segment
  - Applied when the frame payload length exceeds the size of the message buffer data section
- Transmit message buffers configurable with state/event semantics
- Message buffers can be configured as
  - Receive message buffer
  - Single buffered transmit message buffer
  - Double buffered transmit message buffer (combines two single buffered message buffer)
- Individual message buffer reconfiguration supported
  - Means provided to safely disable individual message buffers
  - Disabled message buffers can be reconfigured
- 2 independent receive FIFOs
  - 1 receive FIFO per channel
  - Up to 255 entries for each FIFO
  - Global frame ID filtering, based on both value/mask filters and range filters
  - Global channel ID filtering
  - Global message ID filtering for the dynamic segment
- 4 configurable slot error counters
- 4 dedicated slot status indicators

  - Used to observe slots without using receive message buffers
- Measured value indicators for the clock synchronization
  - internal synchronization frame ID and synchronization frame measurement tables can be copied into the FlexRay memory area
- Fractional macroticks are supported for clock correction
- Maskable interrupt sources provided via individual and combined interrupt lines
- 1 absolute timer
- 1 timer that can be configured to absolute or relative
- SECDED for protocol engine data RAM
- SEDDED for CHI lookup table RAM

31.1.6 Modes of operation

This section describes the basic operational power modes of the CC.

Disabled mode

The CC enters the Disabled Mode during hard reset. The CC indicates that it is in the Disabled Mode by negating the module enable bit MEN in the *Module Configuration Register (FR_MCR)*.

In the Disabled Mode no communication is performed on the FlexRay bus.

All registers with the write access conditions *Any Time* and *Disabled Mode* can be accessed for writing as stated in *Section 31.5.2, Register descriptions*.

The application configures the CC by accessing the configuration bits and fields in the *Module Configuration Register (FR_MCR)*.

Leave Disabled mode

The CC leaves the Disabled Mode and enters the Normal Mode, when the application writes 1 to the module enable bit MEN in the *Module Configuration Register (FR_MCR)*

Note: Once the CC is enabled it can only be disabled via a device reset.

Normal mode

In this mode the CC is fully functional. The CC indicates that it is in Normal Mode by asserting the module enable bit MEN in the *Module Configuration Register (FR_MCR)*.

Enter Normal mode

This mode is entered when the application requests the CC to leave the Disabled Mode. If the Normal Mode was entered by leaving the Disabled Mode, the application has to perform the protocol initialization described in *Section , Protocol Initialization* to achieve full FlexRay functionality.

Depending on the values of the SCM, CHA, and CHB bits in the *Module Configuration Register (FR_MCR)*, the corresponding FlexRay bus driver ports are enabled and driven.

31.2 External signal description

This section lists and describes the CC signals, connected to external pins. These signals are summarized in *Table 702* and described in detail in *Section 31.2.1, Detailed signal descriptions*.

Note: The off-chip signals FR_A_RX, FR_A_TX, and FR_A_TX_EN are available on each package option. The availability of the other off-chip signals depends on the package option.

Table 702. External signal properties

Name	Direction	Active	Reset	Function
FR_A_RX	Input	—	—	Receive Data Channel A
FR_A_TX	Output	—	1	Transmit Data Channel A
FR_A_TX_EN	Output	Low	1	Transmit Enable Channel A
FR_B_RX	Input	—	—	Receive Data Channel B
FR_B_TX	Output	—	1	Transmit Data Channel B
FR_B_TX_EN	Output	Low	1	Transmit Enable Channel B
FR_DBG[0]	Output	—	0	Debug Strobe Signal 0
FR_DBG[1]	Output	—	0	Debug Strobe Signal 1
FR_DBG[2]	Output	—	0	Debug Strobe Signal 2
FR_DBG[3]	Output	—	0	Debug Strobe Signal 3

31.2.1 Detailed signal descriptions

This section provides a detailed description of the CC signals, connected to external pins.

FR_A_RX — Receive Data Channel A

The FR_A_RX signal carries the receive data for channel A from the corresponding FlexRay bus driver.

FR_A_TX — Transmit Data Channel A

The FR_A_TX signal carries the transmit data for channel A to the corresponding FlexRay bus driver.

FR_A_TX_EN — Transmit Enable Channel A

The FR_A_TX_EN signal indicates to the FlexRay bus driver that the CC is attempting to transmit data on channel A.

FR_B_RX — Receive Data Channel B

The FR_B_RX signal carries the receive data for channel B from the corresponding FlexRay bus driver.

FR_B_TX — Transmit Data Channel B

The FR_B_TX signal carries the transmit data for channel B to the corresponding FlexRay bus driver

FR_B_TX_EN — Transmit Enable Channel B

The FR_B_TX_EN signal indicates to the FlexRay bus driver that the CC is attempting to transmit data on channel B.

FR_DBG[3], FR_DBG[2], FR_DBG[1], FR_DBG[0] — Strobe Signals

These signals provide the selected debug strobe signals. For details on the debug strobe signal selection refer to *Section 31.6.16, Strobe signal support*.

31.3 Controller host interface clocking

The clock for the CHI is derived from the system bus clock and has the same phase and frequency as the system bus clock. There are two constraints for the minimum CHI clock frequency:

- The first constraint corresponds to the number of utilized message buffers and is specified in *Section 31.7.6, Number of usable message buffers*.
- The second constraint corresponds to the value of the TIMEOUT field in the *System Memory Access Time-Out Register (FR_SYMATOR)* and is specified in *Section , Configure System Memory Access Time-Out Register (FR_SYMATOR)*.

31.4 Protocol engine clocking

The clock for the protocol engine can be generated by two sources. The first source is the internal crystal oscillator and the second source is an internal PLL. The clock source to be used is selected by the clock source select bit CLKSEL in the *Module Configuration Register (FR_MCR)*.

31.4.1 Oscillator clocking

If the protocol engine is clocked by the internal crystal oscillator, an 40 MHz crystal or CMOS compatible clock must be connected to the oscillator pins. The crystal or clock must fulfill the requirements given by the *FlexRay Communications System Protocol Specification, Version 2.1 Rev A.*

31.4.2 PLL clocking

If the protocol engine is clocked by the internal PLL, the frequency of the PE clock source is system clock / 3. The system clock frequency has to be 120 MHz.

31.5 Memory map and register description

The CC occupies 1280 bytes of address space starting at the base address of the CC is defined by the memory map of the MCU.

31.5.1 Memory map

The complete memory map of the CC is shown in *Table 703*. The addresses presented here are the offsets relative to the CC base address which is defined by the MCU address map.

Table 703. FlexRay memory map

Offset	Register	Access	Location
Module Configuration and Control			
0x0000	*Module Version Register (FR_MVR)*	R	*on page 31-1305*
0x0002	*Module Configuration Register (FR_MCR)*	R/W	*on page 31-1305*
0x0004	*System Memory Base Address High Register (FR_SYMBADHR)*	R/W	*on page 31-1307*
0x0006	*System Memory Base Address Low Register (FR_SYMBADLR)*	R/W	*on page 31-1307*
0x0008	*Strobe Signal Control Register (FR_STBSCR)*	R/W	*on page 31-1308*
0x000A	Reserved	R	—
0x000C	*Message Buffer Data Size Register (FR_MBDSR)*	R/W	*on page 31-1310*
0x000E	*Message Buffer Segment Size and Utilization Register (FR_MBSSUTR)*	R/W	*on page 31-1311*
PE Access Registers			
0x0010	*PE DRAM Access Register (FR_PEDRAR)*	R/W	*on page 31-1312*
0x0012	*PE DRAM Data Register (FR_PEDRDR)*	R/W	*on page 31-1313*
Interrupt and Error Handling			
0x0014	*Protocol Operation Control Register (FR_POCR)*	R/W	*on page 31-1313*
0x0016	*Global Interrupt Flag and Enable Register (FR_GIFER)*	R/W	*on page 31-1315*
0x0018	*Protocol Interrupt Flag Register 0 (FR_PIFR0)*	R/W	*on page 31-1317*
0x001A	*Protocol Interrupt Flag Register 1 (FR_PIFR1)*	R/W	*on page 31-1320*
0x001C	*Protocol Interrupt Enable Register 0 (FR_PIER0)*	R/W	*on page 31-1321*
0x001E	*Protocol Interrupt Enable Register 1 (FR_PIER1)*	R/W	*on page 31-1323*
0x0020	*CHI Error Flag Register (FR_CHIERFR)*	R/W	*on page 31-1324*
0x0022	*Message Buffer Interrupt Vector Register (FR_MBIVEC)*	R	*on page 31-1326*

Table 703. FlexRay memory map (continued)

Offset	Register	Access	Location
0x0024	*Channel A Status Error Counter Register (FR_CASERCR)*	R	*on page 31-1327*
0x0026	*Channel B Status Error Counter Register (FR_CBSERCR)*	R	*on page 31-1327*
Protocol Status			
0x0028	*Protocol Status Register 0 (FR_PSR0)*	R	*on page 31-1328*
0x002A	*Protocol Status Register 1 (FR_PSR1)*	R	*on page 31-1330*
0x002C	*Protocol Status Register 2 (FR_PSR2)*	R	*on page 31-1331*
0x002E	*Protocol Status Register 3 (FR_PSR3)*	R/W	*on page 31-1333*
0x0030	*Macrotick Counter Register (FR_MTCTR)*	R	*on page 31-1335*
0x0032	*Cycle Counter Register (FR_CYCTR)*	R	*on page 31-1335*
0x0034	*Slot Counter Channel A Register (FR_SLTCTAR)*	R	*on page 31-1336*
0x0036	*Slot Counter Channel B Register (FR_SLTCTBR)*	R	*on page 31-1336*
0x0038	*Rate Correction Value Register (FR_RTCORVR)*	R	*on page 31-1337*
0x003A	*Offset Correction Value Register (FR_OFCORVR)*	R	*on page 31-1337*
0x003C	*Combined Interrupt Flag Register (FR_CIFR)*	R	*on page 31-1338*
0x003E	*System Memory Access Time-Out Register (FR_SYMATOR)*	R/W	*on page 31-1339*
Sync Frame Counter and Tables			
0x0040	*Sync Frame Counter Register (FR_SFCNTR)*	R	*on page 31-1340*
0x0042	*Sync Frame Table Offset Register (FR_SFTOR)*	R/W	*on page 31-1341*
0x0044	*Sync Frame Table Configuration, Control, Status Register (FR_SFTCCSR)*	R/W	*on page 31-1341*
Sync Frame Filter			
0x0046	*Sync Frame ID Rejection Filter Register (FR_SFIDRFR)*	R/W	*on page 31-1343*

Table 703. FlexRay memory map (continued)

Offset	Register	Access	Location
0x0048	*Sync Frame ID Acceptance Filter Value Register (FR_SFIDAFVR)*	R/W	*on page 31-1343*
0x004A	*Sync Frame ID Acceptance Filter Mask Register (FR_SFIDAFMR)*	R/W	*on page 31-1344*
	Network Management Vector		
0x004C	*Network Management Vector Register 0 (FR_NMVR0)*	R	*on page 31-1344*
0x004E	*Network Management Vector Register 1 (FR_NMVR1)*	R	*on page 31-1344*
0x0050	*Network Management Vector Register 2 (FR_NMVR2)*	R	*on page 31-1344*
0x0052	*Network Management Vector Register 3 (FR_NMVR3)*	R	*on page 31-1344*
0x0054	*Network Management Vector Register 4 (FR_NMVR4)*	R	*on page 31-1344*
0x0056	*Network Management Vector Register 5 (FR_NMVR5)*	R	*on page 31-1344*
0x0058	*Network Management Vector Length Register (FR_NMVLR)*	R/W	*on page 31-1345*
	Timer Configuration		
0x005A	*Timer Configuration and Control Register (FR_TICCR)*	R/W	*on page 31-1346*
0x005C	*Timer 1 Cycle Set Register (FR_TI1CYSR)*	R/W	*on page 31-1347*
0x005E	*Timer 1 Macrotick Offset Register (FR_TI1MTOR)*	R/W	*on page 31-1347*
0x0060	*Timer 2 Configuration Register 0 (FR_TI2CR0)*	R/W	*on page 31-1348*
0x0062	*Timer 2 Configuration Register 1 (FR_TI2CR1)*	R/W	*on page 31-1349*
	Slot Status Configuration		
0x0064	*Slot Status Selection Register (FR_SSSR)*	R/W	*on page 31-1349*
0x0066	*Slot Status Counter Condition Register (FR_SSCCR)*	R/W	*on page 31-1350*
	Slot Status		
0x0068	*Slot Status Register 0 (FR_SSR0)*	R	*on page 31-1352*

Table 703. FlexRay memory map (continued)

Offset	Register	Access	Location
0x006A	*Slot Status Register 1 (FR_SSR1)*	R	*on page 31-1352*
0x006C	*Slot Status Register 2 (FR_SSR2)*	R	*on page 31-1352*
0x006E	*Slot Status Register 3 (FR_SSR3)*	R	*on page 31-1352*
0x0070	*Slot Status Register 4 (FR_SSR4)*	R	*on page 31-1352*
0x0072	*Slot Status Register 5 (FR_SSR5)*	R	*on page 31-1352*
0x0074	*Slot Status Register 6 (FR_SSR6)*	R	*on page 31-1352*
0x0076	*Slot Status Register 7 (FR_SSR7)*	R	*on page 31-1352*
0x0078	*Slot Status Counter Register 0 (FR_SSCR0)*	R	*on page 31-1354*
0x007A	*Slot Status Counter Register 1 (FR_SSCR1)*	R	*on page 31-1354*
0x007C	*Slot Status Counter Register 2 (FR_SSCR2)*	R	*on page 31-1354*
0x007E	*Slot Status Counter Register 3 (FR_SSCR3)*	R	*on page 31-1354*
MTS Generation			
0x0080	*MTS A Configuration Register (FR_MTSACFR)*	R/W	*on page 31-1355*
0x0082	*MTS B Configuration Register (MTSBCFR)*	R/W	*on page 31-1355*
Shadow Buffer Configuration			
0x0084	*Receive Shadow Buffer Index Register (FR_RSBIR)*	R/W	*on page 31-1356*
Receive FIFO – Configuration			
0x0086	*Receive FIFO Watermark and Selection Register (FR_RFWMSR)*	R/W	*on page 31-1358*
0x0088	*Receive FIFO Start Index Register (FR_RFSIR)*	R/W	*on page 31-1359*
0x008A	*Receive FIFO Depth and Size Register (RFDSR)*	R/W	*on page 31-1359*
Receive FIFO – Control			

Table 703. FlexRay memory map (continued)

Offset	Register	Access	Location
0x008C	*Receive FIFO A Read Index Register (FR_RFARIR)*	R	*on page 31-1360*
0x008E	*Receive FIFO B Read Index Register (FR_RFBRIR)*	R	*on page 31-1360*
Receive FIFO – Filter			
0x0090	*Receive FIFO Message ID Acceptance Filter Value Register (FR_RFMIDAFVR)*	R/W	*on page 31-1361*
0x0092	*Receive FIFO Message ID Acceptance Filter Mask Register (FR_RFMIDAFMR)*	R/W	*on page 31-1362*
0x0094	*Receive FIFO Frame ID Rejection Filter Value Register (FR_RFFIDRFVR)*	R/W	*on page 31-1362*
0x0096	*Receive FIFO Frame ID Rejection Filter Mask Register (FR_RFFIDRFMR)*	R/W	*on page 31-1363*
0x0098	*Receive FIFO Range Filter Configuration Register (FR_RFRFCFR)*	R/W	*on page 31-1363*
0x009A	*Receive FIFO Range Filter Control Register (FR_RFRFCTR)*	R/W	*on page 31-1364*
Dynamic Segment Status			
0x009C	*Last Dynamic Transmit Slot Channel A Register (FR_LDTXSLAR)*	R	*on page 31-1365*
0x009E	*Last Dynamic Transmit Slot Channel B Register (FR_LDTXSLBR)*	R	*on page 31-1366*
Protocol Configuration			
0x00A0 ... 0x00DC	*Protocol Configuration Register 0 (FR_PCR0)* ... *Protocol Configuration Register 30 (FR_PCR30)*	R/W – R/W	*on page 31-1369* ... *on page 31-1376*
0x00DE ... 0x00E6	Reserved	R	—
Receive FIFO – Configuration (cont.)			
0x00E8	*Receive FIFO System Memory Base Address High Register (FR_RFSYMBADHR)*	R/W	*on page 31-1357*
0x00EA	*Receive FIFO System Memory Base Address Low Register (FR_RFSYMBADLR)*	R/W	*on page 31-1357*
0x00EC	*Receive FIFO Periodic Timer Register (FR_RFPTR)*	R/W	*on page 31-1357*

Table 703. FlexRay memory map (continued)

Offset	Register	Access	Location
Receive FIFO – Control (cont.)			
0x00EE	*Receive FIFO Fill Level and POP Count Register (FR_RFFLPCR)*	R/W	*on page 31-1361*
ECC Registers			
0x00F0	*ECC Error Interrupt Flag and Enable Register (FR_EEIFER)*	R/W	*on page 31-1377*
0x00F2	*ECC Error Report and Injection Control Register (FR_EERICR)*	R/W	*on page 31-1380*
0x00F4	*ECC Error Report Address Register (FR_EERAR)*	R	*on page 31-1381*
0x00F6	*ECC Error Report Data Register (FR_EERDR)*	R	*on page 31-1381*
0x00F8	*ECC Error Report Code Register (FR_EERCR)*	R	*on page 31-1382*
0x00FA	*ECC Error Injection Address Register (FR_EEIAR)*	R/W	*on page 31-1383*
0x00FC	*ECC Error Injection Data Register (FR_EEIDR)*	R/W	*on page 31-1383*
0x00FE	*ECC Error Injection Code Register (FR_EEICR)*	R/W	*on page 31-1384*
Message Buffers Configuration, Control, Status			
0x0100	*Message Buffer Configuration, Control, Status Register 0 (FR_MBCCSR0)*	R/W	*on page 31-1384*
0x0102	*Message Buffer Cycle Counter Filter Register 0 (FR_MBCCFR0)*	R/W	*on page 31-1386*
0x0104	*Message Buffer Frame ID Register 0 (FR_MBFIDR0)*	R/W	*on page 31-1388*
0x0106	*Message Buffer Index Register 0 (FR_MBIDXR0)*	R/W	*on page 31-1388*
...	*...*	...	...
0x04F8	*Message Buffer Configuration, Control, Status Register 127 (FR_MBCCSR127)*	R/W	*on page 31-1384*
0x04FA	*Message Buffer Cycle Counter Filter Register 127 (FR_MBCCFR127)*	R/W	*on page 31-1386*
0x04FC	*Message Buffer Frame ID Register 127 (FR_MBFIDR127)*	R/W	*on page 31-1388*
0x04FE	*Message Buffer Index Register 127 (FR_MBIDXR127)*	R/W	*on page 31-1388*

31.5.2 Register descriptions

This section provides detailed descriptions of all registers in ascending address order, presented as 16-bit wide entities

Table 704 provides a key for the register figures and register tables.

Table 704. Register access conventions

Convention	Description
▨	Depending on its placement in the read or write row, indicates that the bit is not readable or not writeable
R*	Reserved bit or field; will not be changed—Application must not write any value different from the reset value
FIELDNAME	Identifies the field—Its presence in the read or write row indicates that it can be read or written.
Register field types	
rwm	A read/write bit that may be modified by a hardware in some fashion other than by a reset
w1c	Write one to clear—A flag bit that can be read, is cleared by writing a one; writing 0 has no effect
Reset value	
0	Resets to zero
1	Resets to one
—	Not defined after reset and not affected by reset

Register reset

All registers except the *Message Buffer Cycle Counter Filter Registers (FR_MBCCFRn)*, *Message Buffer Frame ID Registers (FR_MBFIDRn)*, and *Message Buffer Index Registers (FR_MBIDXRn)* are reset to their reset value on system reset. The registers mentioned above are located in physical memory blocks and, thus, they are not affected by reset. For some register fields, additional reset conditions exist. These additional reset conditions are mentioned in the detailed description of the register. The additional reset conditions are explained in *Table 705*.

Table 705. Additional register reset conditions

Condition	Description
Protocol RUN Command	The register field is reset when the application writes to RUN command “0101” to the POCCMD field in the *Protocol Operation Control Register (FR_POCR)*.
Message Buffer Disable	The register field is reset when the application has disabled the message buffer. This happens when the application writes 1 to the message buffer disable trigger bit FR_MBCCSRn[EDT] while the message buffer is enabled (FR_MBCCSRn[EDS] = 1) and the CC grants the disable to the application by clearing the FR_MBCCSRn[EDS] bit.

Register write access

This section describes the write access restriction terms that apply to all registers.

Register write access restriction

For each register bit and register field, the write access conditions are specified in the detailed register description. A description of the write access conditions is given in *Table 706*. If, for a specific register bit or field, none of the given write access conditions is fulfilled, any write attempt to this register bit or field is ignored without any notification. The values of the bits or fields are not changed. The condition term [A or B] indicates that the register or field can be written to if at least one of the conditions is fulfilled.The condition term [A and B] indicates that the register or field can be written to if both conditions are fulfilled.

Table 706. Register write access restrictions

Condition	Indication	Description
Any Time	—	No write access restriction
Disabled Mode	FR_MCR[MEN] = 0	Write access only when CC is in Disabled Mode
Normal Mode	FR_MCR[MEN] = 1	Write access only when CC is in Normal Mode
POC:config	FR_PSR0[PROTSTATE] = *POC:config*	Write access only when Protocol is in the *POC:config* state
MB_DIS	FR_MBCCSR[EDS] = 0	Write access only when related Message Buffer is disabled
MB_LCK	FR_MBCCSRn[LCKS] = 1	Write access only when related Message Buffer is locked
IDL	FR_EEIRICR[BSY] = 0	Write access only when ECC configuration is idle

Register write access requirements

All registers can be accessed with 8-bit, 16-bit and 32-bit wide operations. For some of the registers, at least a 16-bit wide write access is required to ensure correct operation. This write access requirement is stated in the detailed register description for each register affected

Internal register access

The following memory mapped registers are used to access multiple internal registers.

- *Strobe Signal Control Register (FR_STBSCR)*
- *Slot Status Selection Register (FR_SSSR)*
- *Slot Status Counter Condition Register (FR_SSCCR)*
- *Receive Shadow Buffer Index Register (FR_RSBIR)*

Each of these memory mapped registers provides a SEL field and a WMD bit. The SEL field is used to select the internal register. The WMD bit controls the write mode. If the WMD bit is set to 0 during the write access, all fields of the internal register are updated. If the WMD bit set to 1, only the SEL field is changed. All other fields of the internal register remain unchanged. This allows for reading back the values of the selected internal register in a subsequent read access.

Module Version Register (FR_MVR)

Figure 705. Module Version Register (FR_MVR)

Base + 0x0000

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CHIVER								PEVER							
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	1	0	1	0	0	0	1	0	0	1	1	0	1	0	0	0

This register provides the CC version number. The module version number is derived from the CHI version number and the PE version number.

Table 707. FR_MVR field description

Field	Description
CHIVER	**CHI Version Number** — This field provides the version number of the CC host interface.
PEVER	**PE Version Number** — This field provides the version number of the protocol engine.

Module Configuration Register (FR_MCR)

Figure 706. Module Configuration Register (FR_MCR)

Base + 0x0002

Write: MEN, SBFF, SCM, CHB, CHA, ECCE, FUM, FAM, CLKSEL, BITRATE: Disabled Mode
SFFE: Disabled Mode or *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MEN	SBFF	SCM	CHB	CHA	SFFE	ECCE	0	FUM	FAM	0	CLKSEL	BITRATE			0
W								▨			▨					▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register defines the global configuration of the CC.

Table 708. FR_MCR field description

Field	Description
MEN	**Module Enable** — This bit indicates whether or not the CC is in the Disabled Mode. The application requests the CC to leave the Disabled Mode by writing 1 to this bit Before leaving the Disabled Mode, the application must configure the SCM, SBFF, CHB, CHA, TMODE, BITRATE values. For details see *Section 31.1.6, Modes of operation*. 0 Write: ignored, CC disable not possible Read: CC disabled 1 Write: enable CC Read: CC enabled If the CC is enabled it can not be disabled.

Table 708. FR_MCR field description (continued)

Field	Description
SBFF	**System Bus Failure Freeze** — This bit controls the behavior of the CC in case of a system bus failure. 0 Continue normal operation 1 Transition to freeze mode
SCM	**Single Channel Device Mode** — This control bit defines the channel device mode of the CC as described in *Section 31.6.10, Channel device modes*. 0 CC works in dual channel device mode 1 CC works in single channel device mode
CHB CHA	**Channel Enable** — protocol related parameter: *pChannels* The semantic of these control bits depends on the channel device mode controlled by the SCM bit and is given *Table 709*.
SFFE	**Synchronization Frame Filter Enable** — This bit controls the filtering for received synchronization frames. For details see *Section 31.6.15, Sync frame filtering*. 0 Synchronization frame filtering disabled 1 Synchronization frame filtering enabled
ECCE	**ECC Functionality Enable** — This bit controls the ECC memory error detection functionality. For details see *Section 31.6.24, Memory content error detection*. 0 ECC functionality (injection, detection, reporting, response) disabled 1 ECC functionality enabled
FUM	**FIFO Update Mode** — This bit controls the FIFO update behavior when the interrupt flags FR_GIFER[FAFAIF] and FR_GIFER[FAFBIF] are written by the application (see *Section , FIFO update*) 0 FIFOA/FIFOB is updated on writing 1 to FR_GIFER[FAFAIF] /FR_GIFER[FAFBIF] 1 FIFOA/FIFOB) is **not** updated on writing 1 to FR_GIFER[FAFAIF]/FR_GIFER[FAFBIF]
FAM	**FIFO Address Mode** — This bit controls the location of the system memory base address for the FIFOs. (see *Section , FIFO configuration*) 0 FIFO Base Address located in *System Memory Base Address Register (FR_SYMBADR)* 1 FIFO Base Address located in *Receive FIFO System Memory Base Address Register (FR_RFSYMBADR)*
CLKSEL	**Protocol Engine Clock Source Select** — This bit is used to select the clock source for the protocol engine. 0 PE clock source is generated by on-chip crystal oscillator 1 PE clock source is generated by on-chip PLL
BITRATE	**FlexRay Bus Bit Rate** — This bit field defines the FlexRay bus bit rate. 000 10.0 Mbit/s 001 5.0 Mbit/s 010 2.5 Mbit/s 011 8.0 Mbit/s 100 Reserved 101 Reserved 110 Reserved 111 Reserved

Table 709. FlexRay channel selection

SCM	CHB	CHA	Description
Dual Channel Device Modes			
0	0	0	ports FR_A_RX, FR_A_TX, and FR_A_TX_EN not driven by CC ports FR_B_RX, FR_B_TX, and FR_A_TX_EN not driven by CC
	0	1	ports FR_A_RX, FR_A_TX, and FR_A_TX_EN driven by CC - connected to FlexRay channel A ports FR_B_RX, FR_B_TX, and FR_A_TX_EN not driven by CC
	1	0	ports FR_A_RX, FR_A_TX, and FR_A_TX_EN not driven by CC ports FR_B_RX, FR_B_TX, and FR_A_TX_EN driven by CC - connected to FlexRay channel B
	1	1	ports FR_A_RX, FR_A_TX, and FR_A_TX_EN driven by CC - connected to FlexRay channel A ports FR_B_RX, FR_B_TX, and FR_A_TX_EN driven by CC - connected to FlexRay channel B
Single Channel Device Mode			
1	0	0	ports FR_A_RX, FR_A_TX, and FR_A_TX_EN not driven by CC ports FR_B_RX, FR_B_TX, and FR_A_TX_EN not driven by CC
	0	1	ports FR_A_RX, FR_A_TX, and FR_A_TX_EN driven by CC - connected to FlexRay channel A ports FR_B_RX, FR_B_TX, and FR_A_TX_EN not driven by CC
	1	0	ports FR_A_RX, FR_A_TX, and FR_A_TX_EN driven by CC - connected to FlexRay channel B ports FR_B_RX, FR_B_TX, and FR_A_TX_EN not driven by CC
	1	1	Reserved

System Memory Base Address Register (FR_SYMBADR)

Figure 707. System Memory Base Address High Register (FR_SYMBADHR)

Base + 0x0004 Write: Disabled Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	SMBA[31:16]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 708. System Memory Base Address Low Register (FR_SYMBADLR)

Base + 0x0006 Write: Disabled Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SMBA[15:4]												0	0	0	0
W													▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Note: The system memory base address must be set before the CC is enabled.

The system memory base address registers define the base address of the FlexRay memory area within the system memory. The base address is used by the BMIF to calculate the physical memory address for system memory accesses.

Table 710. FR_SYMBADR field description

Field	Description
SMBA	**System Memory Base Address** — This is the value of the system memory base address for the individual message buffers and sync frame table. This is the value of the system memory base address for the receive FIFO if the FIFO address mode bit FR_MCR[FAM] is set to 1. It is defines as a byte address.

Strobe Signal Control Register (FR_STBSCR)

Figure 709. Strobe Signal Control Register (FR_STBSCR)

Base + 0x0008 16-bit write access required Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	SEL				0	0	0	ENB	0	0	STBPSEL	
W	WMD															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register is used to assign the individual protocol timing related strobe signals given in *Table 712* to the external strobe ports. Each strobe signal can be assigned to at most one strobe port. Each write access to registers overwrites the previously written ENB and STBPSEL values for the signal indicated by SEL. If more than one strobe signal is assigned to one strobe port, the current values of the strobe signals are combined with a binary OR and presented at the strobe port. If no strobe signal is assigned to a strobe port, the strobe port carries logic 0. For more detailed and timing information refer to *Section 31.6.16, Strobe signal support*.

Note: In single channel device mode, channel B related strobe signals are undefined and should not be assigned to the strobe ports.

Table 711. FR_STBSCR field description

Field	Description
WMD	**Write Mode** — This control bit defines the write mode of this register. 0 Write to all fields in this register on write access. 1 Write to SEL field only on write access.
SEL	**Strobe Signal Select** — This control field selects one of the strobe signals given in *Table 712* to be enabled or disabled and assigned to one of the four strobe ports given in *Table 712*.
ENB	**Strobe Signal Enable** — The control bit is used to enable and to disable the strobe signal selected by STBSSEL. 0 Strobe signal is disabled and not assigned to any strobe port. 1 Strobe signal is enabled and assigned to the strobe port selected by STBPSEL.
STBPSEL	**Strobe Port Select** — This field selects the strobe port that the strobe signal selected by the SEL is assigned to. All strobe signals that are enabled and assigned to the same strobe port are combined with a binary OR operation. 00 assign selected signal to FR_DBG[0] 01 assign selected signal to FR_DBG[1] 10 assign selected signal to FR_DBG[2] 11 assign selected signal to FR_DBG[3]

Table 712. Strobe signal mapping

SEL		Description	Channel	Type	Offset[1]	Reference
dec	hex					
0	0x0	arm	—	value	+1	MT start
1	0x1	mt	—	value	+1	MT start
2	0x2	cycle start	—	pulse	0	MT start
3	0x3	minislot start	—	pulse	0	MT start
4	0x4	slot start	A	pulse	0	MT start
5	0x5		B			
6	0x6	receive data after glitch filtering	A	value	+4	FR_A_RX
7	0x7		B			FR_B_RX
8	0x8	channel idle indicator	A	level	+5	FR_A_RX
9	0x9		B			FR_B_RX
10	0xA	syntax error detected	A	pulse	+4	FR_A_RX
11	0xB		B			FR_B_RX
12	0xC	content error detected	A	level	+4	FR_A_RX
13	0xD		B			FR_B_RX
14	0xE	receive FIFO almost-full interrupt signals	A	value	n.a.	RX FIFO A Almost Full Interrupt
15	0xF		B			RX FIFO B Almost Full Interrupt

1. Given in PE clock cycles

Message Buffer Data Size Register (FR_MBDSR)

Figure 710. Message Buffer Data Size Register (FR_MBDSR)

Base + 0x000C

Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	MBSEG2DS							0	MBSEG1DS						
W	▨								▨							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register defines the size of the message buffer data section for the two message buffer segments in a number of two-byte entities.

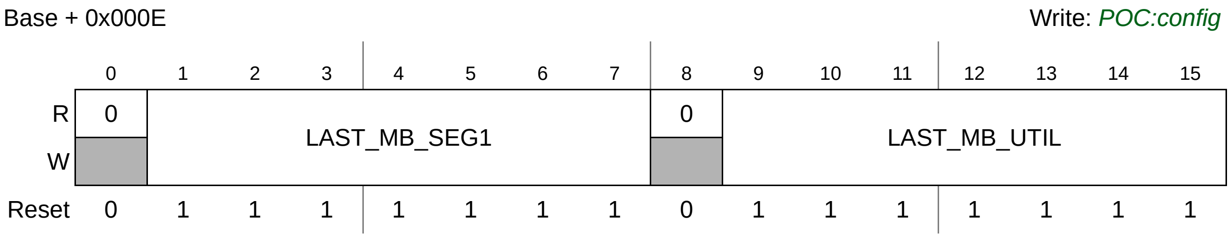
The CC provides two independent segments for the individual message buffers. All individual message buffers within one segment have to have the same size for the message buffer data section. This size can be different for the two message buffer segments.

Table 713. FR_MBDSR field description

Field	Description
MBSEG2DS	**Message Buffer Segment 2 Data Size** — The field defines the size of the message buffer data section in two-byte entities for message buffers within the *second* message buffer segment.
MBSEG1DS	**Message Buffer Segment 1 Data Size** — The field defines the size of the message buffer data section in two-byte entities for message buffers within the *first* message buffer segment.

Message Buffer Segment Size and Utilization Register (FR_MBSSUTR)

Figure 711. Message Buffer Segment Size and Utilization Register (FR_MBSSUTR)



This register is used to define the last individual message buffer that belongs to the first message buffer segment and the number of the last used individual message buffer.

Table 714. FR_MBSSUTR field description

Field	Description
LAST_MB_SEG1	**Last Message Buffer In Segment 1** — This field defines the message buffer number of the last individual message buffer that is assigned to the *first* message buffer segment. The individual message buffers in the *first* segment correspond to the message buffer control registers FR_MBCCSRn, FR_MBCCFRn, FR_MBFIDRn, FR_MBIDXRn with n ≤ LAST_MB_SEG1. The first message buffer segment contains LAST_MB_SEG1 + 1 individual message buffers. The *first* message buffer segment contains *at least* one individual message buffer. The individual message buffers in the *second* message buffer segment correspond to the message buffer control registers FR_MBCCSRn, FR_MBCCFRn, FR_MBFIDRn, FR_MBIDXRn with LAST_MB_SEG1 < n < 128. If LAST_MB_SEG1 = 127 all individual message buffers belong to the *first* message buffer segment and the *second* message buffer segment is empty.
LAST_MB_UTIL	**Last Message Buffer Utilized** — This field defines the message buffer number of last utilized individual message buffer. The message buffer search engine examines all individual message buffer with a message buffer number n ≤ LAST_MB_UTIL. If LAST_MB_UTIL=LAST_MB_SEG1 all individual message buffers belong to the *first* message buffer segment and the *second* message buffer segment is empty.

PE DRAM Access Register (FR_PEDRAR)

Figure 712. PE DRAM Access Register (FR_PEDRAR)

Base + 0x0010 16-bit write access required Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	INST				ADDR											DAD
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register is used to trigger write and read operations on the PE data memory (PE DRAM). These operations are used for memory error injection and memory error observation.

Each write access to this registers initiates a read or write operation on the PE DRAM. The access done status bit DAD is cleared after the write access and is set if the PE DRAM access has been finished.

In case of an PE DRAM write access, the data provided in *PE DRAM Data Register (FR_PEDRDR)* are written into the PE DRAM, read back from the PE DRAM and are stored into the *PE DRAM Data Register (FR_PEDRDR)*.

In case of an PE DRAM read access, the requested data are read from PE DRAM and stored into the *PE DRAM Data Register (FR_PEDRDR)*.

For a detailed description refer to *Section 31.6.24, Memory content error detection*"

Table 715. FR_PEDRAR field description

Field	Description
INST	**PE DRAM Access Instruction** — This field defines the operation to be executed on the PE DRAM. 0011 PE DRAM write: Write FR_PEDRDR[DATA] to PE DRAM address ADDR (16 bit) 0101 PE DRAM read: Read Data from PE DRAM address ADDR (16 bit) into FR_PEDRDR[DATA] other Reserved
ADDR	**PE DRAM Access Address** — This field defines the address in the PE DRAM to be written to or read from.
DAD	**PE DRAM Access Done** — This status bit is cleared when the application has written to this register and is set when the PE DRAM access has finished. 0 PE DRAM access running 1 PE DRAM access done

PE DRAM Data Register (FR_PEDRDR)

Figure 713. PE DRAM Data Register (FR_PEDRDR)

Base + 0x0012

16-bit write access required

Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	DATA															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register provides the data to be written to or read from the PE DRAM by the access initiated by write access to the *PE DRAM Access Register (FR_PEDRAR)*.

Protocol Operation Control Register (FR_POCR)

Figure 714. Protocol Operation Control Register (FR_POCR)

Base + 0x0014

Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	EOC_AP		ERC_AP		BSY	0	0	0	POCCMD			
W	WME	▨	▨	▨					WMC	▨	▨	▨				
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

The application uses this register to issue

- protocol control commands
- external clock correction commands

Protocol control commands are issued by writing to the POCCMD field. For more information on protocol control commands, see *Section 31.7.7, Protocol control command execution*.

External clock correction commands are issued by writing to the EOC_AP and ERC_AP fields. For more information on external clock correction, refer to *Section 31.6.11, External clock synchronization*.

Table 716. FR_POCR field description

Field	Description
WME	**Write Mode External Correction** — This bit controls the write mode of the EOC_AP and ERC_AP fields. 0 Write to EOC_AP and ERC_AP fields on register write. 1 No write to EOC_AP and ERC_AP fields on register write.
EOC_AP	**External Offset Correction Application** — This field is used to trigger the application of the external offset correction value defined in the *Protocol Configuration Register 29 (FR_PCR29)*. 00 do not apply external offset correction value 01 Reserved 10 subtract external offset correction value 11 add external offset correction value
ERC_AP	**External Rate Correction Application** — This field is used to trigger application of the external rate correction value defined in the *Protocol Configuration Register 21 (FR_PCR21)* 00 do not apply external rate correction value 01 Reserved 10 subtract external rate correction value 11 add external rate correction value
BSY	**Protocol Control Command Write Busy** — This status bit indicates the acceptance of the protocol control command issued by the application via the POCCMD field. The CC sets this status bit when the application has issued a protocol control command via the POCCMD field. The CC clears this status bit when protocol control command was accepted by the PE.When the application issues a protocol control command while the BSY bit is asserted, the CC ignores this command, sets the protocol command ignored error flag PCMI_EF in the *CHI Error Flag Register (FR_CHIERFR)*, and will not change the value of the POCCMD field. 0 Command write idle, command accepted and ready to receive new protocol command. 1 Command write busy, command not yet accepted, not ready to receive new protocol command.
WMC	**Write Mode Command** — This bit controls the write mode of the POCCMD field. 0 Write to POCCMD field on register write. 1 Do not write to POCCMD field on register write.
POCCMD	**Protocol Control Command** — The application writes to this field to issue a protocol control command to the PE. The CC sends the protocol command to the PE immediately. While the transfer is running, the BSY bit is set. 0000 ALLOW_COLDSTART — Immediately activate capability of node to cold start cluster. 0001 ALL_SLOTS — Delayed(1) transition to the all slots transmission mode. 0010 CONFIG — Immediately transition to the *POC:config* state. 0011 FREEZE — Immediately transition to the *POC:halt* state. 0100 READY, CONFIG_COMPLETE — Immediately transition to the *POC:ready* state. 0101 RUN — Immediately transition to the *POC:startup start* state. 0110 DEFAULT_CONFIG — Immediately transition to the *POC:default config* state. 0111 HALT — Delayed transition to the *POC:halt* state 1000 WAKEUP — Immediately initiate the wakeup procedure. 1001 Reserved 1010 Reserved 1011 Reserved 1100 Reserved 1101 Reserved 1110 Reserved 1111 Reserved

1. Delayed means on completion of current communication cycle.

Global Interrupt Flag and Enable Register (FR_GIFER)

Figure 715. Global Interrupt Flag and Enable Register (FR_GIFER)

Base + 0x0016

Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MIF	PRIF	CHIF	WUPIF	FAFBIF	FAFAIF	RBIF	TBIF	MIE	PRIE	CHIE	WUPIE	FAFBIE	FAFAIE	RBIE	TBIE
W				w1c	w1c	w1c										
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register provides the means to control some of the interrupt request lines and provides the corresponding interrupt flags. The interrupt flags MIF, PRIF, CHIF, RBIF, and TBIF are the outcome of a binary OR of the related individual interrupt flags and interrupt enables. The generation scheme for these flags is depicted in *Figure 863*. For more details on interrupt generation, see *Section 31.6.20, Interrupt support*. These flags are cleared automatically when all of the corresponding interrupt flags or interrupt enables in the related interrupt flag and enable registers are cleared by the application.

Table 717. FR_GIFER field description

Field	Description
MIF	**Module Interrupt Flag** — This flag is asserted if at least one of the other interrupt flags in this register and its related interrupt enable is asserted. 0 No interrupt flag is asserted or no interrupt enable is set 1 At least one of the other interrupt flags in this register is asserted and the related interrupt bit is asserted, too
PRIF	**Protocol Interrupt Flag** — This flag is set if at least one of the individual protocol interrupt flags in the *Protocol Interrupt Flag Register 0 (FR_PIFR0)* and *Protocol Interrupt Flag Register 1 (FR_PIFR1)* is asserted and the related interrupt enable flag is asserted. 0 All individual protocol interrupt flags are equal to 0 or no interrupt enable bit is set. 1 At least one of the individual protocol interrupt flags and the related interrupt enable is equal to 1.
CHIF	**CHI Interrupt Flag** — This flag is set if at least one of the individual CHI error flags in the *CHI Error Flag Register (FR_CHIERFR)* is asserted and the CHI error interrupt enable FR_GIFER[CHIE] is asserted. 0 All CHI error flags are equal to 0 or the CHI error interrupt is disabled 1 At least one CHI error flag is asserted and CHI error interrupt is enabled
WUPIF	**Wakeup Interrupt Flag** — This flag is set when the CC has received a wakeup symbol on the FlexRay bus. The application can determine on which channel the wakeup symbol was received by reading the related wakeup flags WUB and WUA in the *Protocol Status Register 3 (FR_PSR3).* 0 No wakeup condition or interrupt disabled 1 Wakeup symbol received on FlexRay bus and interrupt enabled

Table 717. FR_GIFER field description (continued)

Field	Description
FAFBIF	**Receive FIFO Channel B Almost Full Interrupt Flag** — This flag is set when one of the following events occurs a) the current number of FIFO B entries is equal to or greater than the watermark defined by the WM field in the *Receive FIFO Watermark and Selection Register (FR_RFWMSR)*, and the CC writes a received message into the FIFO B, or b) the current number of FIFO B entries is at least 1 and the periodic timer as defined by *Receive FIFO Periodic Timer Register (FR_RFPTR)* expires. 0 no such event 1 FIFO B almost full event has occurred
FAFAIF	**Receive FIFO Channel A Almost Full Interrupt Flag** — This flag is set when one of the following events occurs a) the current number of FIFO A entries is equal to or greater than the watermark defined by the WM field in the *Receive FIFO Watermark and Selection Register (FR_RFWMSR)*, and the CC writes a received message into the FIFO A, or b) the current number of FIFO B entries is at least 1 and the periodic timer as defined by *Receive FIFO Periodic Timer Register (FR_RFPTR)* expires. 0 no such event 1 FIFO A almost full event has occurred
RBIF	**Receive Message Buffer Interrupt Flag** — This flag is set if for at least one of the individual receive message buffers (FR_MBCCSRn[MTD] = 0) both the interrupt flag MBIF and the interrupt enable bit MBIE in the corresponding *Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn)* are asserted. The application can not clear this RBIF flag directly. This flag is cleared by the CC when all of the interrupt flags MBIF of the individual receive message buffers are cleared by the application or if the application has cleared the interrupt enables bit MBIE. 0 None of the individual receive message buffers has the MBIF and MBIE flag asserted. 1 At least one individual receive message buffer has the MBIF and MBIE flag asserted.
TBIF	**Transmit Message Buffer Interrupt Flag** — This flag is set if for at least one of the individual single or double transmit message buffers (FR_MBCCSRn[MTD] = 1) both the interrupt flag MBIF and the interrupt enable bit MBIE in the corresponding *Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn)* are equal to 1. The application can not clear this TBIF flag directly. This flag is cleared by the CC when either all of the individual interrupt flags MBIF of the individual transmit message buffers are cleared by the application or the host has cleared the interrupt enables bit MBIE. 0 None of the individual transmit message buffers has the MBIF and MBIE flag asserted. 1 At least one individual transmit message buffer has the MBIF and MBIE flag asserted.
MIE	**Module Interrupt Enable** — This flag controls if the Module Interrupt line is asserted when the MIF flag is set. 0 Disable interrupt line 1 Enable interrupt line
PRIE	**Protocol Interrupt Enable** — This flag controls if the Protocol Interrupt line is asserted when the PRIF flag is set. 0 Disable interrupt line 1 Enable interrupt line
CHIE	**CHI Interrupt Enable** — This flag controls if the CHI Interrupt line is asserted when the CHIF flag is set. 0 Disable interrupt line 1 Enable interrupt line

Table 717. FR_GIFER field description (continued)

Field	Description
WUPIE	**Wakeup Interrupt Enable** — This flag controls if the Wakeup Interrupt line is asserted when the WUPIF flag is set. 0 Disable interrupt line 1 Enable interrupt line
FAFBIE	**Receive FIFO Channel B Almost Full Interrupt Enable** — This flag controls if the RX FIFO B Almost Full Interrupt line is asserted when the FAFBIF flag is set. 0 Disable interrupt line 1 Enable interrupt line
FAFAIE	**Receive FIFO Channel A Almost Full Interrupt Enable** — This flag controls if the RX FIFO A Almost Full Interrupt line is asserted when the FAFAIF flag is set. 0 Disable interrupt line 1 Enable interrupt line
RBIE	**Receive Message Buffer Interrupt Enable** — This flag controls if the Receive Message Buffer Interrupt line is asserted when the RBIF flag is set. 0 Disable interrupt line 1 Enable interrupt line
TBIE	**Transmit Message Buffer Interrupt Enable** — This flag controls if the Transmit Message Buffer Interrupt line is asserted when the TBIF flag is set. 0 Disable interrupt line 1 Enable interrupt line

Protocol Interrupt Flag Register 0 (FR_PIFR0)

Figure 716. Protocol Interrupt Flag Register 0 (FR_PIFR0)

Base + 0x0018 Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FATL_IF	INTL_IF	ILCF_IF	CSA_IF	MRC_IF	MOC_IF	CCL_IF	MXS_IF	MTX_IF	LTXB_IF	LTXA_IF	TBVB_IF	TBVA_IF	TI2_IF	TI1_IF	CYS_IF
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

The register holds one set of the protocol-related individual interrupt flags.

Table 718. FR_PIFR0 field description

Field	Description
FATL_IF	**Fatal Protocol Error Interrupt Flag** — This flag is set when the protocol engine has detected a fatal protocol error. In this case, the protocol engine goes into the *POC:halt* state immediately. The fatal protocol errors are: 1) *pLatestTx* violation, as described in the MAC process of the FlexRay protocol 2) transmission across slot boundary violation, as described in the FSP process of the FlexRay protocol 0 No such event. 1 Fatal protocol error detected.
INTL_IF	**Internal Protocol Error Interrupt Flag** — This flag is set when the protocol engine has detected an internal protocol error. In this case, the protocol engine goes into the *POC:halt* state immediately. An internal protocol error occurs when the protocol engine has not finished a calculation and a new calculation is requested. This can be caused by a hardware error. 0 No such event. 1 Internal protocol error detected.
ILCF_IF	**Illegal Protocol Configuration Interrupt Flag** — This flag is set when the protocol engine has detected an illegal protocol configuration parameter setting. In this case, the protocol engine goes into the *POC:halt* state immediately. The protocol engine checks the *listen_timeout* value programmed into the *Protocol Configuration Register 14 (FR_PCR14)* and *Protocol Configuration Register 15 (FR_PCR15)* when the CONFIG_COMPLETE command was sent by the application via the *Protocol Operation Control Register (FR_POCR)*. If the value of *listen_timeout* is equal to zero, the protocol configuration setting is considered as illegal. 0 No such event. 1 Illegal protocol configuration detected.
CSA_IF	**Cold Start Abort Interrupt Flag** — This flag is set when the configured number of allowed cold start attempts is reached and none of these attempts was successful. The number of allowed cold start attempts is configured by the coldstart_attempts field in the *Protocol Configuration Register 3 (FR_PCR3)*. 0 No such event. 1 Cold start aborted and no more coldstart attempts allowed.
MRC_IF	**Missing Rate Correction Interrupt Flag** — This flag is set when an insufficient number of measurements is available for rate correction at the end of the communication cycle. 0 No such event 1 Insufficient number of measurements for rate correction detected
MOC_IF	**Missing Offset Correction Interrupt Flag** — This flag is set when an insufficient number of measurements is available for offset correction. This is related to the MISSING_TERM event in the CSP process for offset correction in the FlexRay protocol. 0 No such event. 1 Insufficient number of measurements for offset correction detected.
CCL_IF	**Clock Correction Limit Reached Interrupt Flag** — This flag is set when the internal calculated offset or rate calculation values have reached or exceeded its configured thresholds as given by the *offset_coorection_out* field in the *Protocol Configuration Register 9 (FR_PCR9)* and the *rate_correction_out* field in the *Protocol Configuration Register 14 (FR_PCR14)*. 0 No such event. 1 Offset or rate correction limit reached.

Table 718. FR_PIFR0 field description (continued)

Field	Description
MXS_IF	**Max Sync Frames Detected Interrupt Flag** — This flag is set when the number of synchronization frames detected in the current communication cycle exceeds the value of the *node_sync_max* field in the *Protocol Configuration Register 30 (FR_PCR30)*. 0 No such event. 1 More than *node_sync_max* sync frames detected. Only synchronization frames that have passed the synchronization frame acceptance and rejection filters are taken into account.
MTX_IF	**Media Access Test Symbol Received Interrupt Flag** — This flag is set when the MTS symbol was received on channel A or channel B. 0 No such event. 1 MTS symbol received.
LTXB_IF	*pLatestTx* **Violation on Channel B Interrupt Flag** — This flag is set when the frame transmission on channel B in the dynamic segment exceeds the dynamic segment boundary. This is related to the *pLatestTx* violation, as described in the MAC process of the FlexRay protocol. 0 No such event. 1 *pLatestTx* violation occurred on channel B.
LTXA_IF	*pLatestTx* **Violation on Channel A Interrupt Flag** — This flag is set when the frame transmission on channel A in the dynamic segment exceeds the dynamic segment boundary. This is related to the *pLatestTx* violation as described in the MAC process of the FlexRay protocol. 0 No such event. 1 *pLatestTx* violation occurred on channel A.
TBVB_IF	**Transmission across boundary on channel B Interrupt Flag** — This flag is set when the frame transmission on channel B crosses the slot boundary. This is related to the transmission across slot boundary violation as described in the FSP process of the FlexRay protocol. 0 No such event. 1 Transmission across boundary violation occurred on channel B.
TBVA_IF	**Transmission across boundary on channel A Interrupt Flag** — This flag is set when the frame transmission on channel A crosses the slot boundary. This is related to the transmission across slot boundary violation as described in the FSP process of the FlexRay protocol. 0 No such event. 1 Transmission across boundary violation occurred on channel A.
TI2_IF	**Timer 2 Expired Interrupt Flag** — This flag is set whenever timer 2 expires. 0 No such event. 1 Timer 2 has reached its time limit.
TI1_IF	**Timer 1 Expired Interrupt Flag** — This flag is set whenever timer 1 expires. 0 No such event 1 Timer 1 has reached its time limit
CYS_IF	**Cycle Start Interrupt Flag** — This flag is set when a communication cycle starts. 0 No such event 1 Communication cycle started.

Protocol Interrupt Flag Register 1 (FR_PIFR1)

Figure 717. Protocol Interrupt Flag Register 1 (FR_PIFR1)

Base + 0x001A Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	EMC_IF	IPC_IF	PECF_IF	PSC_IF	SSI3_IF	SSI2_IF	SSI1_IF	SSI0_IF	0	0	EVT_IF	ODT_IF	0	0	0	0
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c			w1c	w1c				
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

The register holds one set of the protocol-related individual interrupt flags.

Table 719. FR_PIFR1 field description

Field	Description
EMC_IF	**Error Mode Changed Interrupt Flag** — This flag is set when the value of the ERRMODE bit field in the *Protocol Status Register 0 (FR_PSR0)* is changed by the CC. 0 No such event. 1 ERRMODE field changed.
IPC_IF	**Illegal Protocol Control Command Interrupt Flag** — This flag is set when the PE tries to execute a protocol control command, which was issued via the POCCMD field of the *Protocol Operation Control Register (FR_POCR)*, and detects that this protocol control command is not allowed in the current protocol state. In this case the command is not executed. For more details, see *Section 31.7.7, Protocol control command execution*. 0 No such event. 1 Illegal protocol control command detected.
PECF_IF	**Protocol Engine Communication Failure Interrupt Flag** — This flag is set if the CC has detected a communication failure between the protocol engine and the CC host interface 0 No such event. 1 Protocol Engine Communication Failure detected.
PSC_IF	**Protocol State Changed Interrupt Flag** — This flag is set when the protocol state in the PROTSTATE field in the *Protocol Status Register 0 (FR_PSR0)* has changed. 0 No such event. 1 Protocol state changed.
SSI3_IF SSI2_IF SSI1_IF SSI0_IF	**Slot Status Counter Incremented Interrupt Flag** — Each of these flags is set when the SLOTSTATUSCNT field in the corresponding *Slot Status Counter Registers (FR_SSCR0–FR_SSCR3)* is incremented. 0 No such event. 1 The corresponding slot status counter has incremented.
EVT_IF	**Even Cycle Table Written Interrupt Flag** — This flag is set if the CC has written the sync frame measurement / ID tables into the FlexRay memory area for the even cycle. 0 No such event. 1 Sync frame measurement table written

Table 719. FR_PIFR1 field description (continued)

Field	Description
ODT_IF	**Odd Cycle Table Written Interrupt Flag** — This flag is set if the CC has written the sync frame measurement / ID tables into the FlexRay memory area for the odd cycle. 0 No such event. 1 Sync frame measurement table written

Protocol Interrupt Enable Register 0 (FR_PIER0)

Figure 718. Protocol Interrupt Enable Register 0 (FR_PIER0)

Base + 0x001C Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	FATL_IE	INTL_IE	ILCF_IE	CSA_IE	MRC_IE	MOC_IE	CCL_IE	MXS_IE	MTX_IE	LTXB_IE	LTXA_IE	TBVB_IE	TBVA_IE	TI2_IE	TI1_IE	CYS_IE
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register defines whether or not the individual interrupt flags defined in the *Protocol Interrupt Flag Register 0 (FR_PIFR0)* can generate a protocol interrupt request.

Table 720. FR_PIER0 field description

Field	Description
FATL_IE	**Fatal Protocol Error Interrupt Enable** — This bit controls FATL_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
INTL_IE	**Internal Protocol Error Interrupt Enable** — This bit controls INTL_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
ILCF_IE	**Illegal Protocol Configuration Interrupt Enable** — This bit controls ILCF_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
CSA_IE	**Cold Start Abort Interrupt Enable** — This bit controls CSA_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
MRC_IE	**Missing Rate Correction Interrupt Enable** — This bit controls MRC_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
MOC_IE	**Missing Offset Correction Interrupt Enable** — This bit controls MOC_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled

Table 720. FR_PIER0 field description (continued)

Field	Description
CCL_IE	**Clock Correction Limit Reached Interrupt Enable** — This bit controls CCL_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
MXS_IE	**Max Sync Frames Detected Interrupt Enable** — This bit controls MXS_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
MTX_IE	**Media Access Test Symbol Received Interrupt Enable** — This bit controls MTX_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
LTXB_IE	*pLatestTx* **Violation on Channel B Interrupt Enable** — This bit controls LTXB_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
LTXA_IE	*pLatestTx* **Violation on Channel A Interrupt Enable** — This bit controls LTXA_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
TBVB_IE	**Transmission across boundary on channel B Interrupt Enable** — This bit controls TBVB_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
TBVA_IE	**Transmission across boundary on channel A Interrupt Enable** — This bit controls TBVA_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
TI2_IE	**Timer 2 Expired Interrupt Enable** — This bit controls TI1_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
TI1_IE	**Timer 1 Expired Interrupt Enable** — This bit controls TI1_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
CYS_IE	**Cycle Start Interrupt Enable** — This bit controls CYC_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled

Protocol Interrupt Enable Register 1 (FR_PIER1)

Figure 719. Protocol Interrupt Enable Register 1 (FR_PIER1)

Base + 0x001E Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	EMC_IE	IPC_IE	PECF_IE	PSC_IE	SSI3_IE	SSI2_IE	SSI1_IE	SSI0_IE	0	0	EVT_IE	ODT_IE	0	0	0	0
W									▨	▨			▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register defines whether or not the individual interrupt flags defined in *Protocol Interrupt Flag Register 1 (FR_PIFR1)* can generate a protocol interrupt request.

Table 721. FR_PIER1 field description

Field	Description
EMC_IE	**Error Mode Changed Interrupt Enable** — This bit controls EMC_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
IPC_IE	**Illegal Protocol Control Command Interrupt Enable** — This bit controls IPC_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
PECF_IE	**Protocol Engine Communication Failure Interrupt Enable** — This bit controls PECF_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
PSC_IE	**Protocol State Changed Interrupt Enable** — This bit controls PSC_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
SSI3_IE SSI2_IE SSI1_IE SSI0_IE	**Slot Status Counter Incremented Interrupt Enable** — This bit controls SSI[3:0]_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
EVT_IE	**Even Cycle Table Written Interrupt Enable** — This bit controls EVT_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
ODT_IE	**Odd Cycle Table Written Interrupt Enable** — This bit controls ODT_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled

CHI Error Flag Register (FR_CHIERFR)

Figure 720. CHI Error Flag Register (FR_CHIERFR)

Base + 0x0020 Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FRLB_EF	FRLA_EF	PCMI_EF	FOVB_EF	FOVA_EF	MBS_EF	MBU_EF	LCK_EF	DBL_EF	SBCF_EF(1)	FID_EF	DPL_EF	SPL_EF	NML_EF	NMF_EF	ILSA_EF[1]
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1. The FlexRay controller should be stopped via a FREEZE or HALT command and subsequently restarted when any of the error flags CHIERFR[SBCF_EF] or CHIERFR[ILSA_EF] is set.

This register holds the CHI related error flags. The interrupt generation for each of these error flags is controlled by the CHI interrupt enable bit CHIE in the *Global Interrupt Flag and Enable Register (FR_GIFER)*.

Table 722. FR_CHIERFR field description

Field	Description
FRLB_EF	**Frame Lost Channel B Error Flag** — This flag is set if a complete frame was received on channel B but could not be stored in the selected individual message buffer because this message buffer is currently locked by the application. In this case, the frame and the related slot status information are lost. 0 No such event 1 Frame lost on channel B detected
FRLA_EF	**Frame Lost Channel A Error Flag** — This flag is set if a complete frame was received on channel A but could not be stored in the selected individual message buffer because this message buffer is currently locked by the application. In this case, the frame and the related slot status information are lost. 0 No such error 1 Frame lost on channel A detected
PCMI_EF	**Protocol Command Ignored Error Flag** — This flag is set if the application has issued a POC command by writing to the POCCMD field in the *Protocol Operation Control Register (FR_POCR)* while the BSY flag is equal to 1. In this case the command is ignored by the CC and is lost. 0 No such error 1 POC command ignored
FOVB_EF	**Receive FIFO Overrun Channel B Error Flag** — This flag is set when an overrun of the FIFO for channel B occurred. This error occurs if a semantically valid frame was received on channel B and matches the all criteria to be appended to the FIFO for channel B but the FIFO is full. In this case, the received frame and its related slot status information is lost. 0 No such error 1 FIFO overrun on channel B has been detected

Table 722. FR_CHIERFR field description (continued)

Field	Description
FOVA_EF	**Receive FIFO Overrun Channel A Error Flag** — This flag is set when an overrun of the FIFO for channel A occurred. This error occurs if a semantically valid frame was received on channel A and matches the all criteria to be appended to the FIFO for channel A but the FIFO is full. In this case, the received frame and its related slot status information is lost. 0 No such error 1 FIFO overrun on channel B has been detected
MSB_EF	**Message Buffer Search Error Flag** — This flag is set if the message buffer search engine is still running while the next search cycle must be started due to the FlexRay protocol timing. In this case, not all message buffers are considered while searching. 0 No such event 1 Search engine active while search start appears
MBU_EF	Message Buffer Utilization Error Flag — This flag is asserted if the application writes to a message buffer control field that is beyond the number of utilized message buffers programmed in the *Message Buffer Segment Size and Utilization Register (FR_MBSSUTR)*. If the application writes to a FR_MBCCSRn register with n > LAST_MB_UTIL, the CC ignores the write attempt and asserts the message buffer utilization error flag MBU_EF in the *CHI Error Flag Register (FR_CHIERFR)*. 0 No such event 1 Non-utilized message buffer enabled
LCK_EF	**Lock Error Flag** — This flag is set if the application tries to lock a message buffer that is already locked by the CC due to internal operations. In that case, the CC does not grant the lock to the application. The application must issue the lock request again. 0 No such error 1 Lock error detected
DBL_EF	**Double Transmit Message Buffer Lock Error Flag** — This flag is set if the application tries to lock the transmit side of a double transmit message buffer. In this case, the CC does not grant the lock to the transmit side of a double transmit message buffer. 0 No such event 1 Double transmit buffer lock error occurred
SBCF_EF	**System Bus Communication Failure Error Flag** — This flag is set if a system bus access was not finished within the required amount of time (see *Section , System bus access timeout*). 0 No such event 1 System bus access not finished in time
FID_EF	**Frame ID Error Flag** — This flag is set if the frame ID stored in the message buffer header area differs from the frame ID stored in the message buffer control register. 0 No such error occurred 1 Frame ID error occurred
DPL_EF	**Dynamic Payload Length Error Flag** — This flag is set if the payload length written into the message buffer header field of a single or double transmit message buffer assigned to the dynamic segment is greater than the maximum payload length for the dynamic segment as it is configured in the corresponding protocol configuration register field max_payload_length_dynamic in the *Protocol Configuration Register 24 (FR_PCR24)*. 0 No such error occurred 1 Dynamic payload length error occurred

Table 722. FR_CHIERFR field description (continued)

Field	Description
SPL_EF	**Static Payload Length Error** Flag — This flag is set if the payload length written into the message buffer header field of a single or double transmit message buffer assigned to the static segment is different from the payload length for the static segment as it is configured in the corresponding protocol configuration register field payload_length_static in the *Protocol Configuration Register 19 (FR_PCR19)*. 0 No such error occurred 1 Static payload length error occurred
NML_EF	**Network Management Length Error Flag** — This flag is set if the payload length written into the header structure of a receive message buffer assigned to the static segment is less than the configured length of the Network Management Vector as configured in the *Network Management Vector Length Register (FR_NMVLR)*. In this case the received part of the Network Management Vector will be used to update the Network Management Vector. 0 No such error occurred 1 Network management length error occurred
NMF_EF	**Network Management Frame Error Flag** — This flag is set if a received message in the static segment with a Preamble Indicator flag PP asserted has its Null Frame indicator flag NF asserted as well. In this case, the Global Network Management Registers (see *Network Management Vector Registers (FR_NMVR0–FR_NMVR5)*) are not updated. 0 No such error occurred 1 Network management frame error occurred
ILSA_EF	**Illegal System Bus Address Error Flag** — This flag is set if the external system bus subsystem has detected an access to an illegal system bus address from the CC (see *Section , System bus illegal address access*). 0 No such event 1 Illegal system bus address accessed

Message Buffer Interrupt Vector Register (FR_MBIVEC)

Figure 721. Message Buffer Interrupt Vector Register (FR_MBIVEC)

Base + 0x0022

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	TBIVEC							0	RBIVEC						
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register indicates the lowest numbered receive message buffer and the lowest numbered transmit message buffer that have their interrupt status flag MBIF and interrupt enable MBIE bits asserted. This means that message buffers with lower message buffer numbers have higher priority.

Table 723. FR_MBIVEC field description

Field	Description
TBIVEC	**Transmit Buffer Interrupt Vector** — This field provides the number of the lowest numbered enabled transmit message buffer that has its interrupt status flag MBIF and its interrupt enable bit MBIE set. If there is no transmit message buffer with the interrupt status flag MBIF and the interrupt enable MBIE bits asserted, the value in this field is set to 0.
RBIVEC	**Receive Buffer Interrupt Vector** — This field provides the message buffer number of the lowest numbered receive message buffer which has its interrupt flag MBIF and its interrupt enable bit MBIE asserted. If there is no receive message buffer with the interrupt status flag MBIF and the interrupt enable MBIE bits asserted, the value in this field is set to 0.

Channel A Status Error Counter Register (FR_CASERCR)

Figure 722. Channel A Status Error Counter Register (FR_CASERCR)

Base + 0x0024 Additional Reset: RUN Command

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	STATUS_ERR_CNT															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register provides the channel status error counter for channel A. The protocol engine generates a slot status vector for each static slot, each dynamic slot, the symbol window, and the NIT. The slot status vector contains the four protocol related error indicator bits *vSS!SyntaxError, vSS!ContentError, vSS!BViolation*, and *vSS!TxConflict*. The CC increments the status error counter by 1 if, for a slot or segment, at least one error indicator bit is set to 1. The counter wraps around after it has reached the maximum value. For more information on slot status monitoring, see *Section 31.6.18, Slot status monitoring*.

Table 724. FR_CASERCR field description

Field	Description
STATUS_ERR_CNT	**Channel Status Error Counter** — This field provides the current value channel status error counter. The counter value is updated within the first macrotick of the following slot or segment.

Channel B Status Error Counter Register (FR_CBSERCR)

Figure 723. Channel B Status Error Counter Register (FR_CBSERCR)

Base + 0x0026 Additional Reset: RUN Command

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	STATUS_ERR_CNT															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register provides the channel status error counter for channel B. The protocol engine generates a slot status vector for each static slot, each dynamic slot, the symbol window, and the NIT. The slot status vector contains the four protocol related error indicator bits *vSS!SyntaxError*, *vSS!ContentError*, *vSS!BViolation*, and *vSS!TxConflict*. The CC increments the status error counter by 1 if, for a slot or segment, at least one error indicator bit is set to 1. The counter wraps around after it has reached the maximum value. For more information on slot status monitoring see *Section 31.6.18, Slot status monitoring*.

Table 725. FR_CBSERCR field description

Field	Description
STATUS_ERR_CNT	**Channel Status Error Counter** — This field provides the current channel status error count. The counter value is updated within the first macrotick of the following slot or segment.

Protocol Status Register 0 (FR_PSR0)

Figure 724. Protocol Status Register 0 (FR_PSR0)

Base + 0x0028

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	ERRMODE		SLOTMODE		0	PROTSTATE			STARTUPSTATE				0	WAKEUPSTATUS		
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register provides information about the current protocol status.

Table 726. FR_PSR0 field description

Field	Description
ERRMODE	**Error Mode** — protocol related variable: *vPOC!ErrorMode*. This field indicates the error mode of the protocol. 00 ACTIVE 01 PASSIVE 10 COMM_HALT 11 Reserved
SLOTMODE	**Slot Mode** — protocol related variable: *vPOC!SlotMode*. This field indicates the slot mode of the protocol. 00 SINGLE 01 ALL_PENDING 10 ALL 11 Reserved

Table 726. FR_PSR0 field description (continued)

Field	Description
PROTSTATE	**Protocol State** — protocol related variable: *vPOC!State.* This field indicates the state of the protocol. 000 *POC:default config* 001 *POC:config* 010 *POC:wakeup* 011 *POC:ready* 100 *POC:normal passive* 101 *POC:normal active* 110 *POC:halt* 111 *POC:startup*
STARTUP STATE	**Startup State** — protocol related variable: *vPOC!StartupState.* This field indicates the current substate of the startup procedure. 0000 Reserved 0001 Reserved 0010 *POC:coldstart collision resolution* 0011 *POC:coldstart listen* 0100 *POC:integration consistency check* 0101 *POC:integrationi listen* 0110 Reserved 0111 *POC:initialize schedule* 1000 Reserved 1001 Reserved 1010 *POC:coldstart consistency check* 1011 Reserved 1100 Reserved 1101 *POC:integration coldstart check* 1110 *POC:coldstart gap* 1111 *POC:coldstart join*
WAKEUP STATUS	**Wakeup Status** — protocol related variable: *vPOC!WakeupStatus*. This field provides the outcome of the execution of the wakeup mechanism. 000 UNDEFINED 001 RECEIVED_HEADER 010 RECEIVED_WUP 011 COLLISION_HEADER 100 COLLISION_WUP 101 COLLISION_UNKNOWN 110 TRANSMITTED 111 Reserved

Protocol Status Register 1 (FR_PSR1)

Figure 725. Protocol Status Register 1 (FR_PSR1)

Base + 0x002A Additional Reset: CSAA, CSP, CPN: RUN Command Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CSAA	CSP	0	REMCSAT					CPN	HHR	FRZ	APTAC				
W	w1c															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 727. FR_PSR1 field description

Field	Description
CSAA	**Cold Start Attempt Aborted Flag** — protocol related event: ‘set coldstart abort indicator in CHI’ This flag is set when the CC has aborted a cold start attempt. 0 No such event 1 Cold start attempt aborted
CSP	**Leading Cold Start Path** — This status bit is set when the CC has reached the *POC:normal active* state via the leading cold start path. This indicates that this node has started the network 0 No such event 1 *POC:normal active* reached from *POC:startup* state via leading cold start path
REMCSAT	**Remaining Coldstart Attempts** — protocol related variable: *vRemainingColdstartAttempts* This field provides the number of remaining cold start attempts that the CC will execute.
CPN	**Leading Cold Start Path Noise** — protocol related variable: *vPOC!ColdstartNoise* This status bit is set if the CC has reached the *POC:normal active* state via the leading cold start path under noise conditions. This indicates there was some activity on the FlexRay bus while the CC was starting up the cluster. 0 No such event 1 *POC:normal active* state was reached from *POC:startup* state via noisy leading cold start path
HHR	**Host Halt Request Pending** — protocol related variable: *vPOC!CHIHaltRequest* This status bit is set when CC receives the HALT command from the application via the *Protocol Operation Control Register (FR_POCR)*. The CC clears this status bit after a hard reset condition or when the protocol is in the *POC:default config* state. 0 No such event 1 HALT command received
FRZ	**Freeze Occurred** — protocol related variable: *vPOC!Freeze* This status bit is set when the CC has reached the *POC:halt* state due to the host FREEZE command or due to an internal error condition requiring immediate halt. The CC clears this status bit after a hard reset condition or when the protocol is in the *POC:default config* state. 0 No such event 1 Immediate halt due to FREEZE or internal error condition
APTAC	**Allow Passive to Active Counter** — protocol related variable: *vPOC!vAllowPassivetoActive* This field provides the number of consecutive even/odd communication cycle pairs that have passed with valid rate and offset correction terms, but the protocol is still in the *POC:normal passive* state due to an application configured delay to enter *POC:normal active* state. This delay is defined by the allow_passive_to_active field in the *Protocol Configuration Register 12 (FR_PCR12)*.

Protocol Status Register 2 (FR_PSR2)

Figure 726. Protocol Status Register 2 (FR_PSR2)

Base + 0x002C Additional Reset: RUN Command

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	NBVB	NSEB	STCB	SBVB	SSEB	MTB	NBVA	NSEA	STCA	SBVA	SSEA	MTA	CLKCORRFAILCNT			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register provides a snapshot of status information about the Network Idle Time NIT, the Symbol Window and the clock synchronization. The NIT related status bits NBVB, NSEB, NBVA, and NSEA are updated by the CC after the end of the NIT and before the end of the first slot of the next communication cycle. The Symbol Window related status bits STCB, SBVB, SSEB, MTB, STCA, SBVA, SSEB, and MTA are updated by the CC after the end of the symbol window and before the end of the current communication cycle. If no symbol window is configured, the symbol window related status bits remain in their reset state. The clock synchronization related CLKCORRFAILCNT is updated by the CC after the end of the static segment and before the end of the current communication cycle.

Table 728. FR_PSR2 field description

Field	Description
NBVB	**NIT Boundary Violation on Channel B** — protocol related variable: *vSS!BViolation* for NIT on channel B This status bit is set when there was some media activity on the FlexRay bus channel B at the end of the NIT. 0 No such event 1 Media activity at boundaries detected
NSEB	**NIT Syntax Error on Channel B** — protocol related variable: *vSS!SyntaxError* for NIT on channel B This status bit is set when a syntax error was detected during NIT on channel B. 0 No such event 1 Syntax error detected
STCB	**Symbol Window Transmit Conflict on Channel B** — protocol related variable: *vSS!TxConflict* for symbol window on channel B This status bit is set if there was a transmission conflict during the symbol window on channel B. 0 No such event 1 Transmission conflict detected
SBVB	**Symbol Window Boundary Violation on Channel B** — protocol related variable: *vSS!BViolation* for symbol window on channel B This status bit is set if there was some media activity on the FlexRay bus channel B at the start or at the end of the symbol window. 0 No such event 1 Media activity at boundaries detected

Table 728. FR_PSR2 field description (continued)

Field	Description
SSEB	**Symbol Window Syntax Error on Channel B** — protocol related variable: *vSS!SyntaxError* for symbol window on channel B This status bit is set when a syntax error was detected during the symbol window on channel B. 0 No such event 1 Syntax error detected
MTB	**Media Access Test Symbol MTS Received on Channel B** — protocol related variable: *vSS!ValidMTS* for Symbol Window on channel B This status bit is set if the Media Access Test Symbol MTS was received in the symbol window on channel B. 0 No such event 1 MTS symbol received
NBVA	**NIT Boundary Violation on Channel A** — protocol related variable: *vSS!BViolation* for NIT on channel A This status bit is set when there was some media activity on the FlexRay bus channel A at the end of the NIT. 0 No such event 1 Media activity at boundaries detected
NSEA	**NIT Syntax Error on Channel A** — protocol related variable: *vSS!SyntaxError* for NIT on channel A This status bit is set when a syntax error was detected during NIT on channel A. 0 No such event 1 Syntax error detected
STCA	**Symbol Window Transmit Conflict on Channel A** — protocol related variable: *vSS!TxConflict* for symbol window on channel A This status bit is set if there was a transmission conflicts during the symbol window on channel A. 0 No such event 1 Transmission conflict detected
SBVA	**Symbol Window Boundary Violation on Channel A** — protocol related variable: *vSS!BViolation* for symbol window on channel A This status bit is set if there was some media activity on the FlexRay bus channel A at the start or at the end of the symbol window. 0 No such event 1 Media activity at boundaries detected
SSEA	**Symbol Window Syntax Error on Channel A** — protocol related variable: *vSS!SyntaxError* for symbol window on channel A This status bit is set when a syntax error was detected during the symbol window on channel A. 0 No such event 1 Syntax error detected
MTA	**Media Access Test Symbol MTS Received on Channel A** — protocol related variable: *vSS!ValidMTS* for symbol window on channel A This status bit is set if the Media Access Test Symbol MTS was received in the symbol window on channel A. 1 MTS symbol received 0 No such event

Table 728. FR_PSR2 field description (continued)

Field	Description
CLKCORR-FAILCNT	**Clock Correction Failed Counter** — protocol related variable: *vClockCorrectionFailed* This field provides the number of consecutive even/odd communication cycle pairs that have passed without clock synchronization having performed an offset or a rate correction due to lack of synchronization frames. It is not incremented when it has reached the configured value of either max_without_clock_correction_fatal or max_without_clock_correction_passive as defined in the *Protocol Configuration Register 8 (FR_PCR8)*. The CC resets this counter on a hard reset condition, when the protocol enters the *POC:normal active* state, or when both the rate and offset correction terms have been calculated successfully.

Protocol Status Register 3 (FR_PSR3)

Figure 727. Protocol Status Register 3 (FR_PSR3)

Base + 0x002E Additional Reset: RUN Command Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	WUB	ABVB	AACB	ACEB	ASEB	AVFB	0	0	WUA	ABVA	AACA	ACEA	ASEA	AVFA
W			w1c	w1c	w1c	w1c	w1c	w1c			w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register provides aggregated channel status information as an accrued status of channel activity for all communication slots, regardless of whether they are assigned for transmission or subscribed for reception. It provides accrued information for the symbol window, the NIT, and the wakeup status.

Table 729. FR_PSR3 field description

Field	Description
WUB	**Wakeup Symbol Received on Channel B** — This flag is set when a wakeup symbol was received on channel B. 0 No wakeup symbol received 1 Wakeup symbol received
ABVB	**Aggregated Boundary Violation on Channel B** — This flag is set when a boundary violation has been detected on channel B. Boundary violations are detected in the communication slots, the symbol window, and the NIT. 0 No boundary violation detected 1 Boundary violation detected
AACB	**Aggregated Additional Communication on Channel B** — This flag is set when at least one valid frame was received on channel B in a slot that also contained an additional communication with either syntax error, content error, or boundary violations. 0 No additional communication detected 1 Additional communication detected

Table 729. FR_PSR3 field description (continued)

Field	Description
ACEB	**Aggregated Content Error on Channel B** — This flag is set when a content error has been detected on channel B. Content errors are detected in the communication slots, the symbol window, and the NIT. 0 No content error detected 1 Content error detected
ASEB	**Aggregated Syntax Error on Channel B** — This flag is set when a syntax error has been detected on channel B. Syntax errors are detected in the communication slots, the symbol window and the NIT. 0 No syntax error detected 1 Syntax errors detected
AVFB	**Aggregated Valid Frame on Channel B** — This flag is set when a syntactically correct valid frame has been received in any static or dynamic slot through channel B. 1 At least one syntactically valid frame received 0 No syntactically valid frames received
WUA	**Wakeup Symbol Received on Channel A** — This flag is set when a wakeup symbol was received on channel A. 0 No wakeup symbol received 1 Wakeup symbol received
ABVA	**Aggregated Boundary Violation on Channel A** — This flag is set when a boundary violation has been detected on channel A. Boundary violations are detected in the communication slots, the symbol window, and the NIT. 0 No boundary violation detected 1 Boundary violation detected
AACA	**Aggregated Additional Communication on Channel A** — This flag is set when a valid frame was received in a slot on channel A that also contained an additional communication with either syntax error, content error, or boundary violations. 0 No additional communication detected 1 Additional communication detected
ACEA	**Aggregated Content Error on Channel A** — This flag is set when a content error has been detected on channel A. Content errors are detected in the communication slots, the symbol window, and the NIT. 0 No content error detected 1 Content error detected
ASEA	**Aggregated Syntax Error on Channel A** — This flag is set when a syntax error has been detected on channel A. Syntax errors are detected in the communication slots, the symbol window, and the NIT. 0 No syntax error detected 1 Syntax errors detected
AVFA	**Aggregated Valid Frame on Channel A** — This flag is set when a syntactically correct valid frame has been received in any static or dynamic slot through channel A. 0 No syntactically valid frames received 1 At least one syntactically valid frame received

Macrotick Counter Register (FR_MTCTR)

Figure 728. Macrotick Counter Register (FR_MTCTR)

Base + 0x0030

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	MTCT													
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register provides the macrotick count of the current communication cycle.

Table 730. FR_MTCTR field description

Field	Description
MTCT	**Macrotick Counter** — protocol related variable: *vMacrotick* This field provides the macrotick count of the current communication cycle.

Cycle Counter Register (FR_CYCTR)

Figure 729. Cycle Counter Register (FR_CYCTR)

Base + 0x0032

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	CYCCNT					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register provides the number of the current communication cycle.

Table 731. FR_CYCTR field description

Field	Description
CYCCNT	**Cycle Counter** — protocol related variable: *vCycleCounter* This field provides the number of the current communication cycle. If the counter reaches the maximum value of 63, the counter wraps and starts from zero again.

Slot Counter Channel A Register (FR_SLTCTAR)

Figure 730. Slot Counter Channel A Register (FR_SLTCTAR)

Base + 0x0034

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	SLOTCNTA										
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register provides the number of the current slot in the current communication cycle for channel A.

Table 732. FR_SLTCTAR field description

Field	Description
SLOTCNTA	**Slot Counter Value for Channel A** — protocol related variable: *vSlotCounter* for channel A This field provides the number of the current slot in the current communication cycle.

Slot Counter Channel B Register (FR_SLTCTBR)

Figure 731. Slot Counter Channel B Register (FR_SLTCTBR)

Base + 0x0036

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	SLOTCNTB										
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register provides the number of the current slot in the current communication cycle for channel B.

Table 733. FR_SLTCTBR field description

Field	Description
SLOTCNTA	**Slot Counter Value for Channel B** — protocol related variable: *vSlotCounter* for channel B This field provides the number of the current slot in the current communication cycle.

Rate Correction Value Register (FR_RTCORVR)

Figure 732. Rate Correction Value Register (FR_RTCORVR)

Base + 0x0038 Additional Reset: RUN Command

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	RATECORR															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register provides the sign extended rate correction value in microticks as it was calculated by the clock synchronization algorithm. The CC updates this register during the NIT of each odd numbered communication cycle.

Table 734. FR_RTCORVR field description

Field	Description
RATECORR	**Rate Correction Value** — protocol related variable: *vRateCorrection* (before value limitation and external rate correction) This field provides the sign extended rate correction value in microticks as it was calculated by the clock synchronization algorithm. The value is represented in 2’s complement format. This value does not include the value limitation and the application of the external rate correction. If the magnitude of the internally calculated rate correction value exceeds the limit given by rate_correction_out in the *Protocol Configuration Register 13 (FR_PCR13)*, the clock correction reached limit interrupt flag CCL_IF is set in the *Protocol Interrupt Flag Register 0 (FR_PIFR0)*. If the CC was not able to calculate a new rate correction term due to a lack of synchronization frames, the RATECORR value is not updated.

Offset Correction Value Register (FR_OFCORVR)

Figure 733. Offset Correction Value Register (FR_OFCORVR)

Base + 0x003A Additional Reset: RUN Command

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	OFFSETCORR															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register provides the sign extended offset correction value in microticks as it was calculated by the clock synchronization algorithm. The CC updates this register during the NIT.

Table 735. FR_OFCORVR field description

Field	Description
OFFSETCORR	**Offset Correction Value** — protocol related variable: *vOffsetCorrection* (before value limitation and external offset correction) This field provides the sign extended offset correction value in microticks as it was calculated by the clock synchronization algorithm. The value is represented in 2's complement format. This value does not include the value limitation and the application of the external offset correction. If the magnitude of the internally calculated rate correction value exceeds the limit given by offset_correction_out field in the *Protocol Configuration Register 29 (FR_PCR29)*, the clock correction reached limit interrupt flag CCL_IF is set in the *Protocol Interrupt Flag Register 0 (FR_PIFR0)*. If the CC was not able to calculate an new offset correction term due to a lack of synchronization frames, the OFFSETCORR value is not updated.

Combined Interrupt Flag Register (FR_CIFR)

Figure 734. Combined Interrupt Flag Register (FR_CIFR)

Base + 0x003C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	MIF	PRIF	CHIF	WUPIF	FAFBIF	FAFAIF	RBIF	TBIF
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register provides five combined interrupt flags and a copy of three individual interrupt flags. The combined interrupt flags are the result of a binary OR of the values of other interrupt flags regardless of the state of the interrupt enable bits. The generation scheme for the combined interrupt flags is depicted in *Figure 865*. The individual interrupt flags WUPIF, FAFBIF, and FAFAIF are copies of corresponding flags in the *Global Interrupt Flag and Enable Register (FR_GIFER)* and are provided here to simplify the application interrupt flag check. To clear the individual interrupt flags, the application must use the *Global Interrupt Flag and Enable Register (FR_GIFER)*.

Note: The meanings of the combined status bits MIF, PRIF, CHIF, RBIF, and TBIF are different from those mentioned in the Global Interrupt Flag and Enable Register (FR_GIFER).

Table 736. FR_CIFR field description

Field	Description
MIF	**Module Interrupt Flag** — This flag is set if there is at least one interrupt source that has its interrupt flag asserted. 0 No interrupt source has its interrupt flag asserted 1 At least one interrupt source has its interrupt flag asserted
PRIF	**Protocol Interrupt Flag** — This flag is set if at least one of the individual protocol interrupt flags in the *Protocol Interrupt Flag Register 0 (FR_PIFR0)* or *Protocol Interrupt Flag Register 1 (FR_PIFR1)* is equal to 1. 0 All individual protocol interrupt flags are equal to 0 1 At least one of the individual protocol interrupt flags is equal to 1
CHIF	**CHI Interrupt Flag** — This flag is set if at least one of the individual CHI error flags in the *CHI Error Flag Register (FR_CHIERFR)* is equal to 1. 0 All CHI error flags are equal to 0 1 At least one CHI error flag is equal to 1
WUPIF	**Wakeup Interrupt Flag** — Provides the same value as FR_GIFER[WUPIF]
FAFBIF	**Receive FIFO Channel B Almost Full Interrupt Flag** — Provides the same value as FR_GIFER[FAFBIF]
FAFAIF	**Receive FIFO Channel A Almost Full Interrupt Flag** — Provides the same value as FR_GIFER[FAFAIF]
RBIF	**Receive Message Buffer Interrupt Flag** — This flag is set if for at least one of the individual receive message buffers (FR_MBCCSRn[MTD] = 0) the interrupt flag MBIF in the corresponding *Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn)* is equal to 1. 0 None of the individual receive message buffers has the MBIF flag asserted. 1 At least one individual receive message buffers has the MBIF flag asserted.
TBIF	**Transmit Message Buffer Interrupt Flag** — This flag is set if for at least one of the individual single or double transmit message buffers (FR_MBCCSRn[MTD] = 1) the interrupt flag MBIF in the corresponding *Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn)* is equal to 1. 0 None of the individual transmit message buffers has the MBIF flag asserted. 1 At least one individual transmit message buffers has the MBIF flag asserted.

System Memory Access Time-Out Register (FR_SYMATOR)

Figure 735. System Memory Access Time-Out Register (FR_SYMATOR)

Base + 0x003E Write: Disabled Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	TIMEOUT							
W	▨	▨	▨	▨	▨	▨	▨	▨								
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0

Table 737. FR_SYMATOR field description

Field	Description
TIMEOUT	**System Memory Access Time-Out** — This value defines when a system bus access timeout is detected. For a detailed description see *Section , Configure System Memory Access Time-Out Register (FR_SYMATOR)* and *Section , System bus access timeout*.

Sync Frame Counter Register (FR_SFCNTR)

Figure 736. Sync Frame Counter Register (FR_SFCNTR)

Base + 0x0040 Additional Reset: RUN Command

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SFEVB				SFEVA				SFODB				SFODA			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register provides the number of synchronization frames that are used for clock synchronization in the last even and in the last odd numbered communication cycle. This register is updated after the start of the NIT and before 10 MT after offset correction start.

Note: If the application has locked the even synchronization table at the end of the static segment of an even communication cycle, the CC will not update the fields SFEVB and SFEVA.

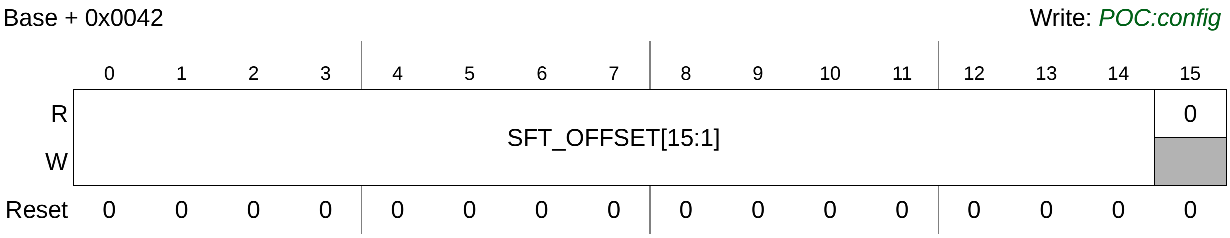
If the application has locked the odd synchronization table at the end of the static segment of an odd communication cycle, the CC will not update the values SFODB and SFODA.

Table 738. FR_SFCNTR field description

Field	Description
SFEVB	Sync Frames Channel B, even cycle — protocol related variable: size of (*vsSyncIdListB* for even cycle) This field provides the size of the internal list of frame IDs of received synchronization frames used for clock synchronization.
SFEVB	Sync Frames Channel A, even cycle — protocol related variable: size of (*vsSyncIdListA* for even cycle) This field provides the size of the internal list of frame IDs of received synchronization frames used for clock synchronization.
SFODB	Sync Frames Channel B, odd cycle — protocol related variable: size of (*vsSyncIdListB* for odd cycle) This field provides the size of the internal list of frame IDs of received synchronization frames used for clock synchronization.
SFODA	Sync Frames Channel A, odd cycle — protocol related variable: size of (*vsSyncIdListA* for odd cycle) This field provides the size of the internal list of frame IDs of received synchronization frames used for clock synchronization.

Sync Frame Table Offset Register (FR_SFTOR)

Figure 737. Sync Frame Table Offset Register (FR_SFTOR)



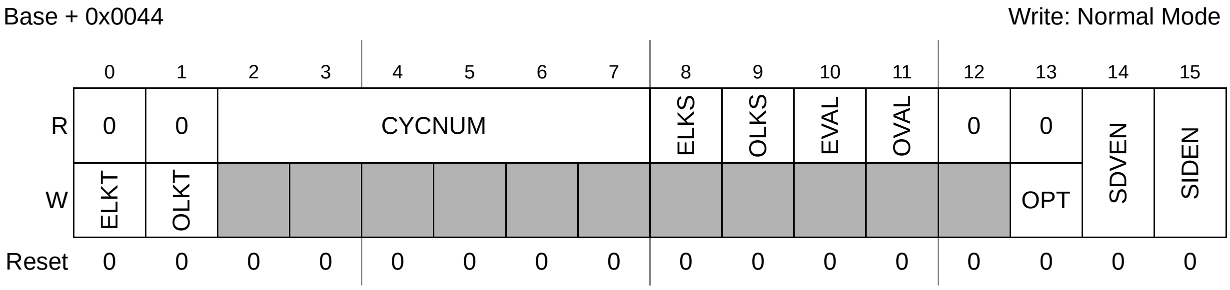
This register defines the FlexRay memory area related offset for sync frame tables. For more details, see *Section 31.6.12, Sync frame ID and sync frame deviation tables*.

Table 739. FR_SFTOR Field Description

Field	Description
SFT_OFFSET	**Sync Frame Table Offset** — The offset of the Sync Frame Tables in the FlexRay memory area. This offset is required to be 16-bit aligned. Thus STF_OFFSET[0] is always 0.

Sync Frame Table Configuration, Control, Status Register (FR_SFTCCSR)

Figure 738. Sync Frame Table Configuration, Control, Status Register (FR_SFTCCSR)



This register provides configuration, control, and status information related to the generation and access of the clock sync ID tables and clock sync measurement tables. For a detailed description, see *Section 31.6.12, Sync frame ID and sync frame deviation tables*.

Table 740. FR_SFTCCSR field description

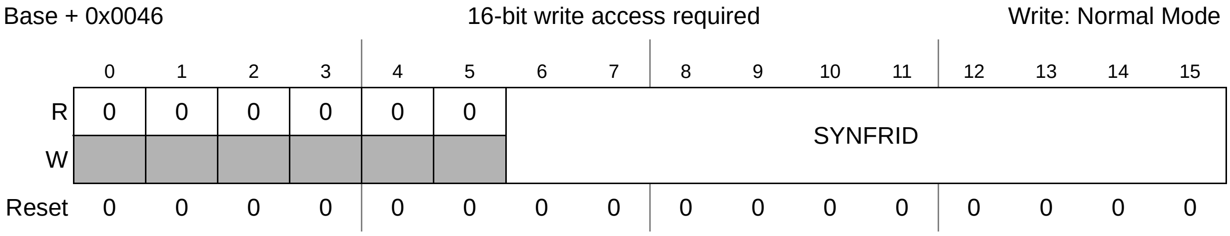
Field	Description
ELKT	**Even Cycle Tables Lock/Unlock Trigger** — This trigger bit is used to lock and unlock the even cycle tables. 0 No effect 1 Triggers lock/unlock of the even cycle tables.
OLKT	**Odd Cycle Tables Lock/Unlock Trigger** — This trigger bit is used to lock and unlock the odd cycle tables. 0 No effect 1 Triggers lock/unlock of the odd cycle tables.

Table 740. FR_SFTCCSR field description (continued)

Field	Description
CYCNUM	**Cycle Number** — This field provides the number of the cycle in which the currently locked table was recorded. If none or both tables are locked, this value is related to the even cycle table.
ELKS	**Even Cycle Tables Lock Status** — This status bit indicates whether the application has locked the even cycle tables. 0 Application has not locked the even cycle tables. 1 Application has locked the even cycle tables.
OLKS	**Odd Cycle Tables Lock Status** — This status bit indicates whether the application has locked the odd cycle tables. 0 Application has not locked the odd cycle tables. 1 Application has locked the odd cycle tables.
EVAL	**Even Cycle Tables Valid** — This status bit indicates whether the Sync Frame ID and Sync Frame Deviation Tables for the even cycle are valid. The CC clears this status bit when it starts updating the tables, and sets this bit when it has finished the table update. 0 Tables are not valid (update is ongoing) 1 Tables are valid (consistent).
OVAL	**Odd Cycle Tables Valid** — This status bit indicates whether the Sync Frame ID and Sync Frame Deviation Tables for the odd cycle are valid. The CC clears this status bit when it starts updating the tables, and sets this bit when it has finished the table update. 0 Tables are not valid (update is ongoing) 1 Tables are valid (consistent).
OPT	**One Pair Trigger** — This trigger bit controls whether the CC writes continuously or only one pair of Sync Frame Tables into the FlexRay memory area. If this trigger is set to 1 while SDVEN or SIDEN is set to 1, the CC writes only one pair of the enabled Sync Frame Tables corresponding to the next even-odd-cycle pair into the FlexRay memory area. In this case, the CC clears the SDVEN or SIDEN bits immediately. If this trigger is set to 0 while SDVEN or SIDEN is set to 1, the CC writes continuously the enabled Sync Frame Tables into the FlexRay memory area. 0 Write continuously pairs of enabled Sync Frame Tables into FlexRay memory area. 1 Write only one pair of enabled Sync Frame Tables into FlexRay memory area.
SDVEN	**Sync Frame Deviation Table Enable** — This bit controls the generation of the Sync Frame Deviation Tables. The application must set this bit to request the CC to write the Sync Frame Deviation Tables into the FlexRay memory area. 0 Do not write Sync Frame Deviation Tables 1 Write Sync Frame Deviation Tables into FlexRay memory area If SDVEN is set to 1, then SIDEN must also be set to 1.
SIDEN	**Sync Frame ID Table Enable** — This bit controls the generation of the Sync Frame ID Tables. The application must set this bit to 1 to request the CC to write the Sync Frame ID Tables into the FlexRay memory area. 0 Do not write Sync Frame ID Tables 1 Write Sync Frame ID Tables into FlexRay memory area

Sync Frame ID Rejection Filter Register (FR_SFIDRFR)

Figure 739. Sync Frame ID Rejection Filter Register (FR_SFIDRFR)



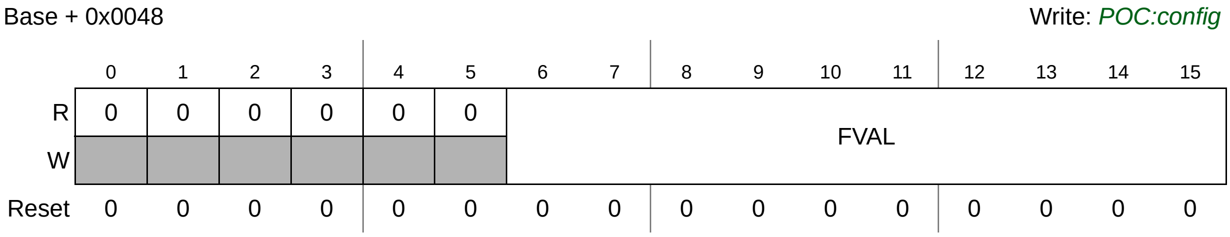
This register defines the Sync Frame Rejection Filter ID. The application must update this register outside of the static segment. If the application updates this register in the static segment, it can appear that the CC accepts the sync frame in the current cycle.

Table 741. FR_SFIDRFR field description

Field	Description
SYNFRID	**Sync Frame Rejection ID** — This field defines the frame ID of a frame that must not be used for clock synchronization. For details see *Section , Sync frame rejection filtering*.

Sync Frame ID Acceptance Filter Value Register (FR_SFIDAFVR)

Figure 740. Sync Frame ID Acceptance Filter Value Register (FR_SFIDAFVR)



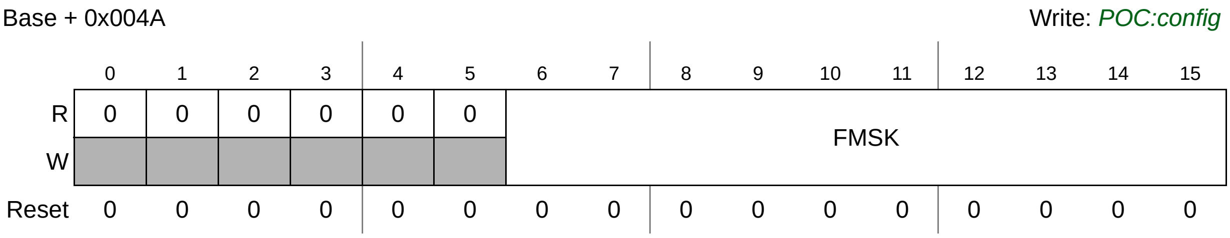
This register defines the sync frame acceptance filter value. For details on filtering, see *Section 31.6.15, Sync frame filtering*.

Table 742. FR_SFIDAFVR field description

Field	Description
FVAL	**Filter Value** — This field defines the value for the sync frame acceptance filtering.

Sync Frame ID Acceptance Filter Mask Register (FR_SFIDAFMR)

Figure 741. Sync Frame ID Acceptance Filter Mask Register (FR_SFIDAFMR)



This register defines the sync frame acceptance filter mask. For details on filtering see *Section , Sync frame acceptance filtering*.

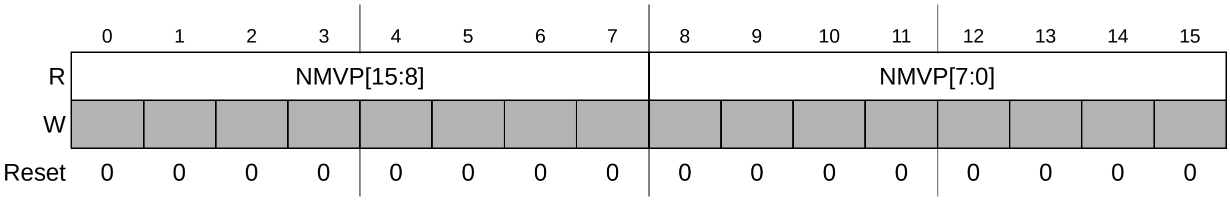
Table 743. FR_SFIDAFMR field description

Field	Description
FMSK	**Filter Mask** — This field defines the mask for the sync frame acceptance filtering.

Network Management Vector Registers (FR_NMVR0–FR_NMVR5)

Figure 742. Network Management Vector Registers (FR_NMVR0–FR_NMVR5)

Base + 0x004C (FR_NMVR0)
Base + 0x004E (FR_NMVR1)
Base + 0x0050 (FR_NMVR2)
Base + 0x0052 (FR_NMVR3)
Base + 0x0054 (FR_NMVR4)
Base + 0x0056 (FR_NMVR5)



Each of these six registers holds one part of the Network Management Vector. The length of the Network Management Vector is configured in the *Network Management Vector Length Register (FR_NMVLR)*. If FR_NMVLR is programmed with a value that is less than 12 bytes, the remaining bytes of the *Network Management Vector Registers (FR_NMVR0–FR_NMVR5)*, which are not used for the Network Management Vector accumulating, will remain 0.

The NMVR provides accrued information over all received NMVs in the last communication cycle. All NMVs received in one cycle are ORed into the NMVR. The NMVR is updated at the end of the communication cycle.

Table 744. NMVR[0:5] field description

Field	Description
NMVP	**Network Management Vector Part** — The mapping between the *Network Management Vector Registers (FR_NMVR0–FR_NMVR5)* and the receive message buffer payload bytes in NMV[0:11] is depicted in *Table 745*.

Table 745. Mapping of NMVRn to the received payload bytes NMVn

NMVRn register	NMVn received payload
FR_NMVR0[NMVP[15:8]]	NMV0
FR_NMVR0[NMVP[7:0]]	NMV1
FR_NMVR1[NMVP[15:8]]	NMV2
FR_NMVR1[NMVP[7:0]]	NMV3
...	
FR_NMVR5[NMVP[15:8]]	NMV10
FR_NMVR5[NMVP[7:0]]	NMV11

Network Management Vector Length Register (FR_NMVLR)

Figure 743. Network Management Vector Length Register (FR_NMVLR)

Base + 0x0058 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	NMVL			
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨				
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register defines the length of the network management vector in bytes.

Table 746. FR_NMVLR field description

Field	Description
NMVL	**Network Management Vector Length** — protocol related variable: *gNetworkManagementVectorLength* This field defines the length of the Network Management Vector in bytes. Legal values are between 0 and 12.

Timer Configuration and Control Register (FR_TICCR)

Figure 744. Timer Configuration and Control Register (FR_TICCR)

Base + 0x005A

Write: T2_CFG: *POC:config*
T2_REP, T1_REP, T1SP, T2SP, T1TR, T2TR: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	T2_CFG	T2_REP	0	0	0	T2ST	0	0	0	T1_REP	0	0	0	T1ST
W						T2SP	T2TR							T1SP	T1TR	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register is used to configure and control the two timers T1 and T2. For timer details, see *Section 31.6.17, Timer support*. The Timer T1 is an absolute timer. The Timer T2 can be configured as an absolute or relative timer.

Table 747. FR_TICCR field description

Field	Description
T2_CFG	**Timer T2 Configuration** — This bit configures the timebase mode of Timer T2. 0 T2 is absolute timer. 1 T2 is relative timer.
T2_REP	**Timer T2 Repetitive Mode** — This bit configures the repetition mode of Timer T2. 0 T2 is non repetitive 1 T2 is repetitive
T2SP	**Timer T2 Stop** — This trigger bit is used to stop timer T2. 0 no effect 1 stop timer T2
T2TR	**Timer T2 Trigger** — This trigger bit is used to start timer T2. 0 no effect 1 start timer T2
T2ST	**Timer T2 State** — This status bit provides the current state of timer T2. 0 timer T2 is idle 1 timer T2 is running
T1_REP	**Timer T1 Repetitive Mode** — This bit configures the repetition mode of timer T1. 0 T1 is non repetitive 1 T1 is repetitive
T1SP	**Timer T1 Stop** — This trigger bit is used to stop timer T1. 0 no effect 1 stop timer T1
T1TR	**Timer T1 Trigger** — This trigger bit is used to start timer T1. 0 no effect 1 start timer T1

Table 747. FR_TICCR field description (continued)

Field	Description
T1ST	**Timer T1 State** — This status bit provides the current state of timer T1. 0 timer T1 is idle 1 timer T1 is running

Note: Both timers are deactivated immediately when the protocol enters a state different from POC:normal active or POC:normal passive.

Timer 1 Cycle Set Register (FR_TI1CYSR)

Figure 745. Timer 1 Cycle Set Register (FR_TI1CYSR)

Base + 0x005C Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	T1_CYC_VAL						0	0	T1_CYC_MSK					
W	▨	▨							▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register defines the cycle filter value and the cycle filter mask for timer T1. For a detailed description of timer T1, refer to *Section , Absolute timer T1*.

Table 748. FR_TI1CYSR field description

Field	Description
T1_CYC_VAL	**Timer T1 Cycle Filter Value** — This field defines the cycle filter value for timer T1.
T1_CYC_MSK	**Timer T1 Cycle Filter Mask** — This field defines the cycle filter mask for timer T1.

Note: If the application modifies the value in this register while the timer is running, the change becomes effective immediately and timer T1 will expire according to the changed value.

Timer 1 Macrotick Offset Register (FR_TI1MTOR)

Figure 746. Timer 1 Macrotick Offset Register (FR_TI1MTOR)

Base + 0x005E Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	T1_MTOFFSET													
W	▨	▨														
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register holds the macrotick offset value for timer T1. For a detailed description of timer T1, refer to *Section , Absolute timer T1*.

Table 749. FR_TI1MTOR field description

Field	Description
T1_MTOFFSET	**Timer 1 Macrotick Offset** — This field defines the macrotick offset value for timer 1.

Note: If the application modifies the value in this register while the timer is running, the change becomes effective immediately and timer T1 will expire according to the changed value.

Timer 2 Configuration Register 0 (FR_TI2CR0)

Figure 747. Timer 2 Configuration Register 0 (FR_TI2CR0)

Base + 0x0060 Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	R*		T2_CYC_VAL						R*		T2_CYC_MSK					
R W	T2_MTCNT[31:16]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

The content of this register depends on the value of the T2_CFG bit in the *Timer Configuration and Control Register (FR_TICCR)*. For a detailed description of timer T2, refer to *Section , Absolute / Relative timer T2*.

Table 750. FR_TI2CR0 field description

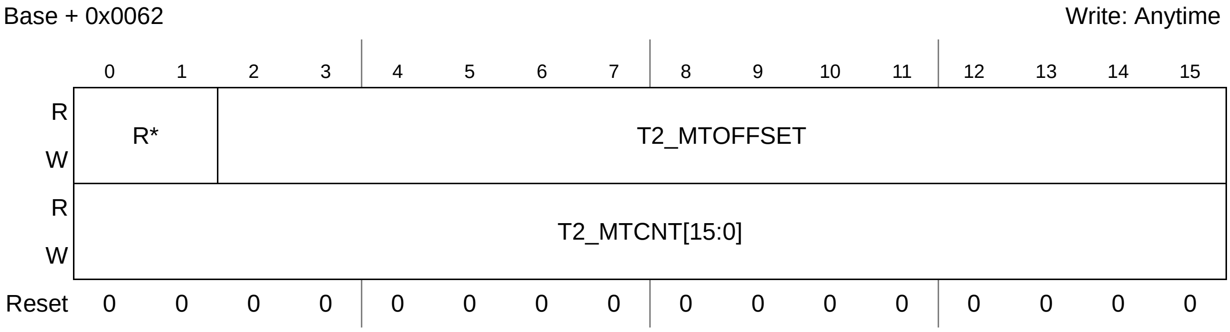
Field	Description
Fields for absolute timer T2 (FR_TICCR[T2_CFG] = 0)	
T2_CYC_VAL	**Timer T2 Cycle Filter Value** — This field defines the cycle filter value for timer T2.
T2_CYC_MSK	**Timer T2 Cycle Filter Mask** — This field defines the cycle filter mask for timer T2.
Fields for relative timer T2 (FR_TICCR[T2_CFG = 1)	
T2_MTCNT[31:16]	**Timer T2 Macrotick High Word** — This field defines the high word of the macrotick count for timer T2.

Note: If timer T2 is configured as an absolute timer and the application modifies the values in this register while the timer is running, the change becomes effective immediately and timer T2 will expire according to the changed values.

If timer T2 is configured as a relative timer and the application changes the values in this register while the timer is running, the change becomes effective when the timer has expired according to the old values.

Timer 2 Configuration Register 1 (FR_TI2CR1)

Figure 748. Timer 2 Configuration Register 1 (FR_TI2CR1)



The content of this register depends on the value of the T2_CFG bit in the *Timer Configuration and Control Register (FR_TICCR)*. For a detailed description of timer T2, refer to *Section , Absolute / Relative timer T2*.

Table 751. FR_TI2CR1 field description

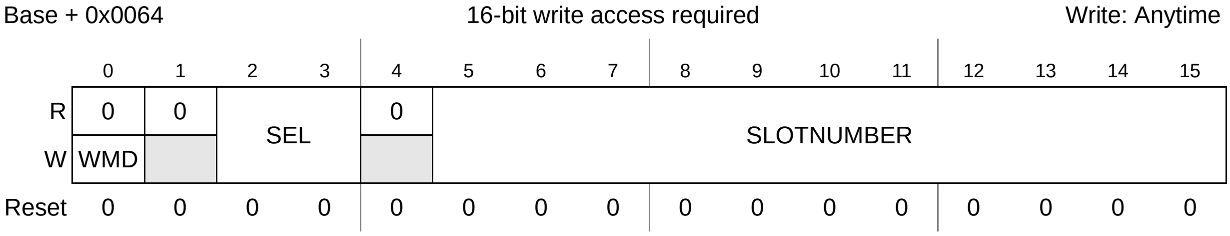
Field	Description
Fields for absolute timer T2 (FR_TICCR[T2_CFG] = 0)	
T2_MTOFFSET	**Timer T2 Macrotick Offset** — This field holds the macrotick offset value for timer T2.
Fields for relative timer T2 (FR_TICCR[T2_CFG] = 1)	
T2_MTCNT[15:0]	**Timer T2 Macrotick Low Word** — This field defines the low word of the macrotick value for timer T2.

Note: If timer T2 is configured as an absolute timer and the application modifies the values in this register while the timer is running, the change becomes effective immediately and the timer T2 will expire according to the changed values.

If timer T2 is configured as a relative timer and the application changes the values in this register while the timer is running, the change becomes effective when the timer has expired according to the old values.

Slot Status Selection Register (FR_SSSR)

Figure 749. Slot Status Selection Register (FR_SSSR)



This register is used to access the four internal non memory-mapped slot status selection registers FR_SSSR0 to FR_SSSR3. Each internal registers selects a slot, or symbol window/NIT, whose status vector will be saved in the corresponding *Slot Status Registers (FR_SSR0–FR_SSR7)* according to *Table 753*. For a detailed description of slot status monitoring, refer to *Section 31.6.18, Slot status monitoring*.

Note: Slot status information of the message buffers should not be used when any one of the the error flags FR_CHIERFR[SBCF_EF] or FR_CHIERFR[ILSA_EF] is set.

Table 752. FR_SSSR field description

Field	Description
WMD	**Write Mode** — This control bit defines the write mode of this register. 0 Write to all fields in this register on write access. 1 Write to SEL field only on write access.
SEL	**Selector** — This field selects one of the four internal slot status selection registers for access. 00 select FR_SSSR0. 01 select FR_SSSR1. 10 select FR_SSSR2. 11 select FR_SSSR3.
SLOTNUMBER	**Slot Number** — This field specifies the number of the slot whose status will be saved in the corresponding slot status registers. If this value is set to 0, the related slot status register provides the status of the symbol window after the NIT start, and provides the status of the NIT after the cycle start.

Table 753. Mapping between FR_SSSRn and FR_SSRn

Internal Slot Status Selection Register	Write the slot status of the slot selected by FR_SSSRn for each			
	Even communication cycle		Odd communication cycle	
	For Channel B to	For Channel A to	For Channel B to	For Channel A to
FR_SSSR0	FR_SSR0[15:8]	FR_SSR0[7:0]	FR_SSR1[15:8]	FR_SSR1[7:0]
FR_SSSR1	FR_SSR2[15:8]	FR_SSR2[7:0]	FR_SSR3[15:8]	FR_SSR3[7:0]
FR_SSSR2	FR_SSR4[15:8]	FR_SSR4[7:0]	FR_SSR5[15:8]	FR_SSR5[7:0]
FR_SSSR3	FR_SSR6[15:8]	FR_SSR6[7:0]	FR_SSR7[15:8]	FR_SSR7[7:0]

Slot Status Counter Condition Register (FR_SSCCR)

Figure 750. Slot Status Counter Condition Register (FR_SSCCR)

Base + 0x0066 16-bit write access required Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	SEL		0	CNTCFG		MCY	VFR	SYF	NUF	SUF	STATUSMASK[3:0]			
W	WMD															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register is used to access and program the four internal non-memory mapped Slot Status Counter Condition Registers FR_SSCCR0 to FR_SSCCR3. Each of these four internal slot status counter condition registers defines the mode and the conditions for incrementing the counter in the corresponding *Slot Status Counter Registers (FR_SSCR0–FR_SSCR3)*. The correspondence is given in *Table 755*. For a detailed description of slot status counters, refer to *Section , Slot status counter registers*.

Table 754. FR_SSCCR field description

Field	Description
WMD	**Write Mode** — This control bit defines the write mode of this register. 0 Write to all fields in this register on write access. 1 Write to SEL field only on write access.
SEL	**Selector** — This field selects one of the four internal slot counter condition registers for access. 00 select FR_SSCCR0. 01 select FR_SSCCR1. 10 select FR_SSCCR2. 11 select FR_SSCCR3.
CNTCFG	**Counter Configuration** — These bit field controls the channel related incrementing of the slot status counter. 00 increment by 1 if condition is fulfilled on channel A. 01 increment by 1 if condition is fulfilled on channel B. 10 increment by 1 if condition is fulfilled on at least one channel. 11 increment by 2 if condition is fulfilled on both channels channel. increment by 1 if condition is fulfilled on only one channel.
MCY	**Multi Cycle Selection** — This bit defines whether the slot status counter accumulates over multiple communication cycles or provides information for the previous communication cycle only. 0 The Slot Status Counter provides information for the previous communication cycle only. 1 The Slot Status Counter accumulates over multiple communication cycles.
VFR	**Valid Frame Restriction** — This bit is used to restrict the counter to received valid frames. 0 The counter is not restricted to valid frames only. 1 The counter is restricted to valid frames only.
SYF	**Sync Frame Restriction** — This bit is used to restrict the counter to received frames with the sync frame indicator bit set to 1. 0 The counter is not restricted with respect to the sync frame indicator bit. 1 The counter is restricted to frames with the sync frame indicator bit set to 1.
NUF	**Null Frame Restriction** — This bit is used to restrict the counter to received frames with the null frame indicator bit set to 0. 0 The counter is not restricted with respect to the null frame indicator bit. 1 The counter is restricted to frames with the null frame indicator bit set to 0.
SUF	**Startup Frame Restriction** — This bit is used to restrict the counter to received frames with the startup frame indicator bit set to 1. 0 The counter is not restricted with respect to the startup frame indicator bit. 1 The counter is restricted to received frames with the startup frame indicator bit set to 1.

Table 754. FR_SSCCR field description

Field	Description
STATUS MASK[3:0]	**Slot Status Mask** — This bit field is used to enable the counter with respect to the four slot status error indicator bits. **STATUSMASK[3]** – This bit enables the counting for slots with the syntax error indicator bit set to 1. **STATUSMASK[2]** – This bit enables the counting for slots with the content error indicator bit set to 1. **STATUSMASK[1]** – This bit enables the counting for slots with the boundary violation indicator bit set to 1. **STATUSMASK[0]** – This bit enables the counting for slots with the transmission conflict indicator bit set to 1.

Table 755. Mapping between internal FR_SSCCRn and FR_SSCRn

Condition register	Condition defined for register
FR_SSCCR0	FR_SSCR0
FR_SSCCR1	FR_SSCR1
FR_SSCCR2	FR_SSCR2
FR_SSCCR3	FR_SSCR3

Slot Status Registers (FR_SSR0–FR_SSR7)

Figure 751. Slot Status Registers (FR_SSR0–FR_SSR7)

Base + 0x0068 (FR_SSR0)
Base + 0x006A (FR_SSR1)
Base + 0x006C (FR_SSR2)
Base + 0x006E (FR_SSR3)
Base + 0x0070 (FR_SSR4)
Base + 0x0072 (FR_SSR5)
Base + 0x0074 (FR_SSR6)
Base + 0x0076 (FR_SSR7)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VFB	SYB	NFB	SUB	SEB	CEB	BVB	TCB	VFA	SYA	NFA	SUA	SEA	CEA	BVA	TCA
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Each of these eight registers holds the status vector of the slot specified in the corresponding internal slot status selection register, which can be programmed using the *Slot Status Selection Register (FR_SSSR)*. Each register is updated after the end of the corresponding slot as shown in *Figure 861*. The register bits are directly related to the protocol variables and described in more detail in *Section 31.6.18, Slot status monitoring*.

Table 756. FR_SSR0–FR_SSR7 field description

Field	Description
VFB	**Valid Frame on Channel B** — protocol related variable: *vSS!ValidFrame* channel B 0 *vSS!ValidFrame* = 0 1 *vSS!ValidFrame* = 1
SYB	**Sync Frame Indicator Channel B** — protocol related variable: *vRF!Header!SyFIndicator* channel B 0 *vRF!Header!SyFIndicator* = 0 1 *vRF!Header!SyFIndicator* = 1
NFB	**Null Frame Indicator Channel B** — protocol related variable: *vRF!Header!NFIndicator* channel B 0 *vRF!Header!NFIndicator* = 0 1 *vRF!Header!NFIndicator* = 1
SUB	**Startup Frame Indicator Channel B** — protocol related variable: *vRF!Header!SuFIndicator* channel B 0 *vRF!Header!SuFIndicator* = 0 1 *vRF!Header!SuFIndicator* = 1
SEB	**Syntax Error on Channel B** — protocol related variable: *vSS!SyntaxError* channel B 0 *vSS!SyntaxError* = 0 1 *vSS!SyntaxError* = 1
CEB	**Content Error on Channel B** — protocol related variable: *vSS!ContentError* channel B 0 *vSS!ContentError* = 0 1 *vSS!ContentError* = 1
BVB	**Boundary Violation on Channel B** — protocol related variable: *vSS!BViolation* channel B 0 *vSS!BViolation* = 0 1 *vSS!BViolation* = 1
TCB	**Transmission Conflict on Channel B** — protocol related variable: *vSS!TxConflict* channel B 0 *vSS!TxConflict* = 0 1 *vSS!TxConflict* = 1
VFA	**Valid Frame on Channel A** — protocol related variable: *vSS!ValidFrame* channel A 0 *vSS!ValidFrame* = 0 1 *vSS!ValidFrame* = 1
SYA	**Sync Frame Indicator Channel A** — protocol related variable: *vRF!Header!SyFIndicator* channel A 0 *vRF!Header!SyFIndicator* = 0 1 *vRF!Header!SyFIndicator* = 1
NFA	**Null Frame Indicator Channel A** — protocol related variable: *vRF!Header!NFIndicator* channel A 0 *vRF!Header!NFIndicator* = 0 1 *vRF!Header!NFIndicator* = 1
SUA	**Startup Frame Indicator Channel A** — protocol related variable: *vRF!Header!SuFIndicator* channel A 0 *vRF!Header!SuFIndicator* = 0 1 *vRF!Header!SuFIndicator* = 1
SEA	**Syntax Error on Channel A** — protocol related variable: *vSS!SyntaxError* channel A 0 *vSS!SyntaxError* = 0 1 *vSS!SyntaxError* = 1

Table 756. FR_SSR0–FR_SSR7 field description

Field	Description
CEA	**Content Error on Channel A** — protocol related variable: *vSS!ContentError* channel A 0 *vSS!ContentError* = 0 1 *vSS!ContentError* = 1
BVA	**Boundary Violation on Channel A** — protocol related variable: *vSS!BViolation* channel A 0 *vSS!BViolation* = 0 1 *vSS!BViolation* = 1
TCA	**Transmission Conflict on Channel A** — protocol related variable: *vSS!TxConflict* channel A 0 *vSS!TxConflict* = 0 1 *vSS!TxConflict* = 1

Slot Status Counter Registers (FR_SSCR0–FR_SSCR3)

Figure 752. Slot Status Counter Registers (FR_SSCR0–FR_SSCR3)

Base + 0x0078 (FR_SSCR0)
Base + 0x007A (FR_SSCR1)
Base + 0x007C (FR_SSCR2)
Base + 0x007E (FR_SSCR3)

Additional Reset: RUN Command

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SLOTSTATUSCNT															
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Each of these four registers provides the slot status counter value for the previous communication cycle(s) and is updated at the cycle start. The provided value depends on the control bits and fields in the related internal slot status counter condition register FR_SSCCRn, which can be programmed by using the *Slot Status Counter Condition Register (FR_SSCCR)*. For more details, see *Section , Slot status counter registers*.

Note: *If the counter has reached its maximum value 0xFFFF and is in the multicycle mode, that is, FR_SSCCRn[MCY] = 1, the counter is not reset to 0x0000. The application can reset the counter by clearing the FR_SSCCRn[MCY] bit and waiting for the next cycle start, when the CC clears the counter. Subsequently, the counter can be set into the multicycle mode again.*

Table 757. FR_SSCR0–FR_SSCR3 field description

Field	Description
SLOTSTATUSCNT	**Slot Status Counter** — This field provides the current value of the Slot Status Counter.

MTS A Configuration Register (FR_MTSACFR)

Figure 753. MTS A Configuration Register (FR_MTSACFR)

Base + 0x0080

Write: MTE: Anytime
CYCCNTMSK,CYCCNTVAL:*POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MTE	0	CYCCNTMSK						0	0	CYCCNTVAL					
W		▨							▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register controls the transmission of the Media Access Test Symbol MTS on channel A. For more details, see *Section 31.6.13, MTS generation*.

Table 758. FR_MTSACFR field description

Field	Description
MTE	**Media Access Test Symbol Transmission Enable** — This control bit is used to enable and disable the transmission of the Media Access Test Symbol in the selected set of cycles. 0 MTS transmission disabled 1 MTS transmission enabled
CYCCNTMSK	**Cycle Counter Mask** — This field provides the filter mask for the MTS cycle count filter.
CYCCNTVAL	**Cycle Counter Value** — This field provides the filter value for the MTS cycle count filter.

MTS B Configuration Register (MTSBCFR)

Figure 754. MTS B Configuration Register (MTSBCFR)

Base + 0x0082

Write: MTE: Anytime
CYCCNTMSK,CYCCNTVAL:*POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MTE	0	CYCCNTMSK						0	0	CYCCNTVAL					
W		▨							▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register controls the transmission of the Media Access Test Symbol MTS on channel B. For more details, see *Section 31.6.13, MTS generation*.

Table 759. MTSBCFR field description

Field	Description
MTE	**Media Access Test Symbol Transmission Enable** — This control bit is used to enable and disable the transmission of the Media Access Test Symbol in the selected set of cycles. 0 MTS transmission disabled 1 MTS transmission enabled
CYCCNTMSK	**Cycle Counter Mask** — This field provides the filter mask for the MTS cycle count filter.
CYCCNTVAL	**Cycle Counter Value** — This field provides the filter value for the MTS cycle count filter.

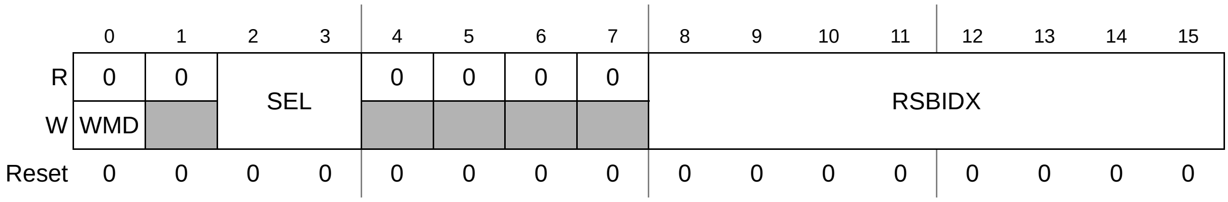
Receive Shadow Buffer Index Register (FR_RSBIR)

Figure 755. Receive Shadow Buffer Index Register (FR_RSBIR)

Base + 0x0084

16-bit write access required

Write: WMD, SEL: Any Time
RSBIDX: *POC:config*



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	SEL		0	0	0	0	RSBIDX							
W	WMD															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register is used to provide and retrieve the indices of the message buffer header fields currently associated with the receive shadow buffers. For more details on the receive shadow buffer concept, refer to *Section , Receive shadow buffers concept*.

Table 760. FR_RSBIR field description

Field	Description
WMD	**Write Mode** — This bit controls the write mode for this register. 0 update SEL and RSBIDX field on register write 1 update only SEL field on register write
SEL	**Selector** — This field is used to select the internal receive shadow buffer index register for access. 00 FR_RSBIR_A1 — receive shadow buffer index register for channel A, segment 1 01 FR_RSBIR_A2 — receive shadow buffer index register for channel A, segment 2 10 FR_RSBIR_B1 — receive shadow buffer index register for channel B, segment 1 11 FR_RSBIR_B2 — receive shadow buffer index register for channel B, segment 2
RSBIDX	**Receive Shadow Buffer Index** — This field contains the current index of the message buffer header field of the receive shadow message buffer selected by the SEL field. The CC uses this index to determine the physical location of the shadow buffer header field in the FlexRay memory area. The CC will update this field during receive operation.The application provides initial message buffer header index value in the configuration phase. CC: Updates the message buffer header index after successful reception. Application: Provides initial message buffer header index.

Receive FIFO System Memory Base Address Register (FR_RFSYMBADR)

Figure 756. Receive FIFO System Memory Base Address High Register (FR_RFSYMBADHR)

Base + 0x00E8 Write: Disabled Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	SMBA[31:16]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 757. Receive FIFO System Memory Base Address Low Register (FR_RFSYMBADLR)

Base + 0x00EA Write: Disabled Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SMBA[15:4]												0	0	0	0
W													▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

These registers define the system memory base address for the receive FIFO if the FIFO address mode bit FR_MCR[FAM] is set to 1. The system memory base address is used by the BMIF to calculate the physical memory address for system memory accesses for the FIFOs.

Table 761. FR_RFSYMBADR field description

Field	Description
SMBA	**System Memory Base Address** — This is the value of the system memory base address for the receive FIFO if the FIFO address mode bit FR_MCR[FAM] is set to 1. It is defines as a byte address.

Receive FIFO Periodic Timer Register (FR_RFPTR)

Figure 758. Receive FIFO Periodic Timer Register (FR_RFPTR)

Base + 0x00EC Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	PTD													
W	▨	▨														
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register holds periodic timer duration for the periodic FIFO timer. The periodic timer applies to both FIFOs (see *Section , FIFO periodic timer*).

Table 762. FR_RFPTR field description

Field	Description
PTD	**Periodic Timer Duration** — This value defines the periodic timer duration in terms of macroticks. 0000 timer stays expired 3FFF timer never expires other timer expires after specified number of macroticks, expires and is restarted at each cycle start

Receive FIFO Watermark and Selection Register (FR_RFWMSR)

Figure 759. Receive FIFO Watermark and Selection Register (FR_RFWMSR)

Base + 0x0086 Write: *WM_A/WM_B: POC:config, SEL: Anytime*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	WM_A/WM_B								0	0	0	0	0	0	0	SEL
W									▨	▨	▨	▨	▨	▨	▨	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register is used to

- select a receiver FIFO for subsequent programming access through the receiver FIFO configuration registers summarized in *Table 763*.
- to define the watermark for the selected FIFO.

Table 763. SEL Controlled Receiver FIFO Registers

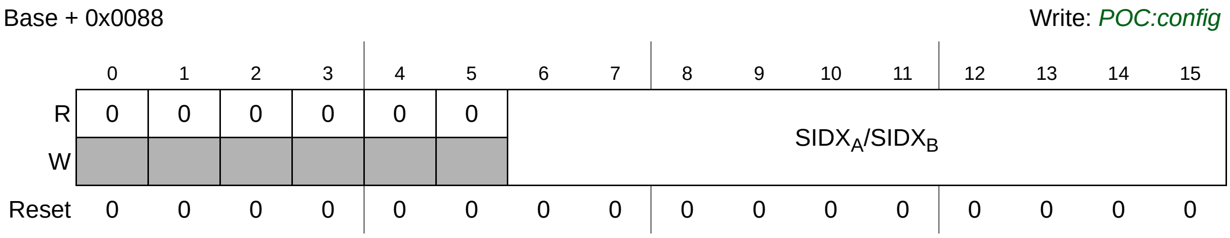
Register
Receive FIFO Start Index Register (FR_RFSIR)
Receive FIFO Depth and Size Register (RFDSR)
Receive FIFO Message ID Acceptance Filter Value Register (FR_RFMIDAFVR)
Receive FIFO Message ID Acceptance Filter Mask Register (FR_RFMIDAFMR)
Receive FIFO Frame ID Rejection Filter Value Register (FR_RFFIDRFVR)
Receive FIFO Frame ID Rejection Filter Mask Register (FR_RFFIDRFMR)
Receive FIFO Range Filter Configuration Register (FR_RFRFCFR)
Receive FIFO Range Filter Control Register (FR_RFRFCTR)

Table 764. FR_RFWMSR field description

Field	Description
WM_A WM_B	**Watermark** — This field defines the watermark value for the selected FIFO. This value is used to control the generation of the almost full interrupt flags.
SEL	**Select** — This control bit selects the receiver FIFO for subsequent programming. 0 Receiver FIFO for channel A selected 1 Receiver FIFO for channel B selected

Receive FIFO Start Index Register (FR_RFSIR)

Figure 760. Receive FIFO Start Index Register (FR_RFSIR)



Base + 0x0088 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	$SIDX_A/SIDX_B$									
W	▨	▨	▨	▨	▨	▨										
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

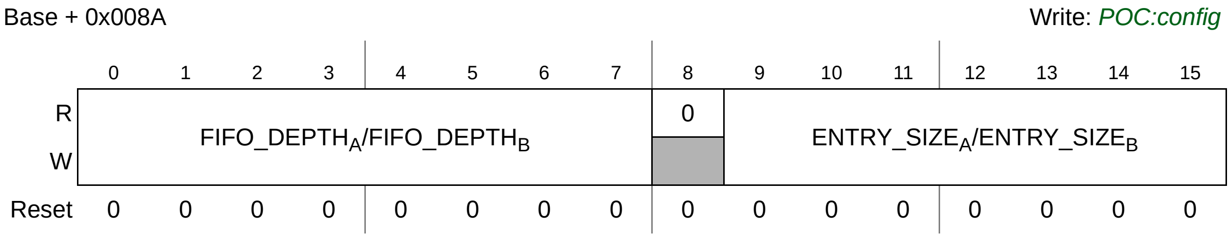
This register defines the message buffer header index of the first message buffer of the selected FIFO.

Table 765. FR_RFSIR field description

Field	Description
$SIDX_A$ $SIDX_B$	**Start Index** — This field defines the number of the message buffer header field of the first message buffer of the selected FIFO. The CC uses the value of the SIDX field to determine the physical location of the receiver FIFO's first message buffer header field.

Receive FIFO Depth and Size Register (RFDSR)

Figure 761. Receive FIFO Depth and Size Register (RFDSR)



Base + 0x008A Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	$FIFO\_DEPTH_A/FIFO\_DEPTH_B$								0	$ENTRY\_SIZE_A/ENTRY\_SIZE_B$						
W									▨							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register defines the structure of the selected FIFO, that is, the number of entries and the size of each entry.

Table 766. RFDSR field description

Field	Description
$FIFO\_DEPTH_A$ $FIFO\_DEPTH_B$	**FIFO Depth** — This field defines the depth of the selected FIFO, that is, the number of entries. **Note**: If the FIFO_DEPTH is configured to 0, FR_RFFIDRFMR[FIDRFMSK] must be configured to 0 too, to ensure that no frames are received into the FIFO.
$ENTRY\_SIZE_A$ $ENTRY\_SIZE_B$	**Entry Size** — This field defines the size of the frame data sections for the selected FIFO in 2 byte entities.

Receive FIFO A Read Index Register (FR_RFARIR)

Figure 762. Receive FIFO A Read Index Register (FR_RFARIR)

Base + 0x008C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	RDIDX									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register provides the message buffer header index of the next available FIFO A entry that the application can read.

Table 767. FR_RFARIR field description

Field	Description
RDIDX	**Read Index** — This field provides the message buffer header index of the next available FIFO message buffer that the application can read. If the old style FIFO mode is configured (FR_MCR[FIMD] = 0), the CC updates this index by 1 entry, when the application writes to the FAFAIF flag in the *Global Interrupt Flag and Enable Register (FR_GIFER)*. If the new style FIFO mode is configured (FR_MCR[FIMD] = 1), the CC updates this index by PCA entries, when the application writes to the *Receive FIFO Fill Level and POP Count Register (FR_RFFLPCR)*.

Note: If the FIFO is empty, the RDIDX field points to an physical message buffer with invalid content.

Receive FIFO B Read Index Register (FR_RFBRIR)

Figure 763. Receive FIFO B Read Index Register (FR_RFBRIR)

Base + 0x008E

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	RDIDX									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register provides the message buffer header index of the next available FIFO B entry that the application can read.

Table 768. FR_RFBRIR field description

Field	Description
RDIDX	**Read Index** — This field provides the message buffer header index of the next available FIFO message buffer that the application can read.

Note: If the FIFO is empty, the RDIDX field points to an physical message buffer with invalid content.

Receive FIFO Fill Level and POP Count Register (FR_RFFLPCR)

Figure 764. Receive FIFO Fill Level and POP Count Register (FR_RFFLPCR)

Base + 0x00EE

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FLB								FLA							
W	PCB								PCA							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register provides the current fill level of the two receiver FIFOs and is used to pop a number of entries from the FIFOs.

Table 769. FR_RFFLPCR field description

Field	Description
FLB	**Fill Level FIFO B** — This field provides the current number of entries in the FIFO B.
FLA	**Fill Level FIFO A** — This field provides the current number of entries in the FIFO A.
PCB	**Pop Count FIFO B** — This field defines the number of entries to be removed from FIFO B.
PCA	**Pop Count FIFO A**— This field defines the number of entries to be removed from FIFO A.

Note: If the pop count value PCA/PCB is greater than the current FIFO fill level FLB/FLA, than the FIFO is empty after the update. No notification is given that not the required number of entries was removed.

Receive FIFO Message ID Acceptance Filter Value Register (FR_RFMIDAFVR)

Figure 765. Receive FIFO Message ID Acceptance Filter Value Register (FR_RFMIDAFVR)

Base + 0x0090 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MIDAFVAL$_A$/MIDAFVAL$_B$															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register defines the filter value for the message ID acceptance filter of the selected FIFO. For details on message ID filtering see *Section , FIFO filtering*.

Table 770. FR_RFMIDAFVR field description

Field	Description
MIDAFVAL$_A$ MIDAFVAL$_B$	**Message ID Acceptance Filter Value** — Filter value for the message ID acceptance filter.

Receive FIFO Message ID Acceptance Filter Mask Register (FR_RFMIDAFMR)

Figure 766. Receive FIFO Message ID Acceptance Filter Mask Register (FR_RFMIDAFMR)

Base + 0x0092 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MIDAFMSK$_A$/MIDAFMSK$_B$															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register defines the filter mask for the message ID acceptance filter of the selected FIFO. For details on message ID filtering see *Section , FIFO filtering*.

Table 771. FR_RFMIDAFMR field description

Field	Description
MIDAFMSK$_A$ MIDAFMSK$_B$	**Message ID Acceptance Filter Mask** — Filter mask for the message ID acceptance filter.

Receive FIFO Frame ID Rejection Filter Value Register (FR_RFFIDRFVR)

Figure 767. Receive FIFO Frame ID Rejection Filter Value Register (FR_RFFIDRFVR)

Base + 0x0094 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	FIDRFVAL$_A$/FIDRFVAL$_B$										
W	▨	▨	▨	▨	▨											
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register defines the filter value for the frame ID rejection filter of the selected FIFO. For details on frame ID filtering see *Section , FIFO filtering*.

Table 772. FR_RFFIDRFVR field description

Field	Description
FIDRFVAL$_A$ FIDRFVAL$_B$	**Frame ID Rejection Filter Value** — Filter value for the frame ID rejection filter.

Receive FIFO Frame ID Rejection Filter Mask Register (FR_RFFIDRFMR)

Figure 768. Receive FIFO Frame ID Rejection Filter Mask Register (FR_RFFIDRFMR)

Base + 0x0096 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	$FIDRFMSK_A/FIDRFMSK_B$										
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register defines the filter mask for the frame ID rejection filter of the selected FIFO. For details on frame ID filtering see *Section , FIFO filtering*.

Table 773. FR_RFFIDRFMR field description

Field	Description
FIDRFMSK	**Frame ID Rejection Filter Mask** — Filter mask for the frame ID rejection filter.

Receive FIFO Range Filter Configuration Register (FR_RFRFCFR)

Figure 769. Receive FIFO Range Filter Configuration Register (FR_RFRFCFR)

Base + 0x0098 16-bit write access required Write: WMD, IBD, SEL: Any Time

SID: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	IBD	SEL		0	SID_A/SID_B										
W	WMD															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register provides access to the four internal frame ID range filter boundary registers of the selected FIFO. For details on frame ID range filter see *Section , FIFO filtering*.

Table 774. FR_RFRFCFR field description

Field	Description
WMD	**Write Mode** — This control bit defines the write mode of this register. 0 Write to all fields in this register on write access. 1 Write to SEL and IBD field only on write access.
IBD	**Interval Boundary** — This control bit selects the interval boundary to be programmed with the SID value. 0 program lower interval boundary 1 program upper interval boundary
SEL	**Filter Selector** — This control field selects the frame ID range filter to be accessed. 00 select frame ID range filter 0. 01 select frame ID range filter 1. 10 select frame ID range filter 2. 11 select frame ID range filter 3.
SID_A SID_B	**Slot ID —** Defines the IBD-selected frame ID boundary value for the SEL-selected range filter.

Receive FIFO Range Filter Control Register (FR_RFRFCTR)

Figure 770. Receive FIFO Range Filter Control Register (FR_RFRFCTR)

Base + 0x009A Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	F3MD	F2MD	F1MD	F0MD	0	0	0	0	F3EN	F2EN	F1EN	F0EN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register is used to enable and disable each frame ID range filter and to define whether it is running as acceptance or rejection filter.

Table 775. FR_RFRFCTR field description

Field	Description
F3MD	**Range Filter 3 Mode** — This control bit defines the filter mode of the frame ID range filter 3. 0 range filter 3 runs as acceptance filter 1 range filter 3 runs as rejection filter
F2MD	**Range Filter 2 Mode** — This control bit defines the filter mode of the frame ID range filter 2. 0 range filter 2 runs as acceptance filter 1 range filter 2 runs as rejection filter
F1MD	**Range Filter 1 Mode** — This control bit defines the filter mode of the frame ID range filter 1. 0 range filter 1 runs as acceptance filter 1 range filter 1 runs as rejection filter
F0MD	**Range Filter 0 Mode** — This control bit defines the filter mode of the frame ID range filter 0. 0 range filter 0 runs as acceptance filter 1 range filter 0 runs as rejection filter
F3EN	**Range Filter 3 Enable** — This control bit is used to enable and disable the frame ID range filter 3. 0 range filter 3 disabled 1 range filter 3 enabled
F2EN	**Range Filter 2 Enable** — This control bit is used to enable and disable the frame ID range filter 2. 0 range filter 2 disabled 1 range filter 2 enabled
F1EN	**Range Filter 1 Enable** — This control bit is used to enable and disable the frame ID range filter 1. 0 range filter 1 disabled 1 range filter 1 enabled
F0EN	**Range Filter 0 Enable** — This control bit is used to enable and disable the frame ID range filter 0. 0 range filter 0 disabled 1 range filter 0 enabled

Last Dynamic Transmit Slot Channel A Register (FR_LDTXSLAR)

Figure 771. Last Dynamic Transmit Slot Channel A Register (FR_LDTXSLAR)

Base + 0x009C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	LASTDYNTXSLOTA										
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register provides the number of the last transmission slot in the dynamic segment for channel A. This register is updated after the end of the dynamic segment and before the start of the next communication cycle.

Table 776. FR_LDTXSLAR field description

Field	Description
LASTDYNTX SLOTA	**Last Dynamic Transmission Slot Channel A** — protocol related variable: *zLastDynTxSlot* channel A Number of the last transmission slot in the dynamic segment for channel A. If no frame was transmitted during the dynamic segment on channel A, the value of this field is set to 0.

Last Dynamic Transmit Slot Channel B Register (FR_LDTXSLBR)

Figure 772. Last Dynamic Transmit Slot Channel B Register (FR_LDTXSLBR)

Base + 0x009E

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	LASTDYNTXSLOTB										
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register provides the number of the last transmission slot in the dynamic segment for channel B. This register is updated after the end of the dynamic segment and before the start of the next communication cycle.

Table 777. FR_LDTXSLBR field description

Field	Description
LASTDYNTX SLOTB	**Last Dynamic Transmission Slot Channel B** — protocol related variable: *zLastDynTxSlot* channel B Number of the last transmission slot in the dynamic segment for channel B. If no frame was transmitted during the dynamic segment on channel B the value of this field is set to 0.

Protocol configuration registers

The following configuration registers provide the necessary configuration information to the protocol engine. The individual values in the registers are described in *Table 778*. For more details about the FlexRay related configuration parameters and the allowed parameter ranges, see *FlexRay Communications System Protocol Specification, Version 2.1 Rev A*.

Table 778. Protocol configuration register fields (Sheet 1 of 2)

Name	Description(1)	Min	Max	Unit	FR_PCR
coldstart_attempts	*gColdstartAttempts*			number	3
action_point_offset	*gdActionPointOffset* - 1			MT	0
cas_rx_low_max	*gdCASRxLowMax* - 1			*gdBit*	4
dynamic_slot_idle_phase	*gdDynamicSlotIdlePhase*			minislot	28
minislot_action_point_offset	*gdMinislotActionPointOffset* - 1			MT	3
minislot_after_action_point	*gdMinislot* - *gdMinislotActionPointOffset* - 1			MT	2
static_slot_length	*gdStaticSlot*			MT	0
static_slot_after_action_point	*gdStaticSlot* - *gdActionPointOffset* - 1			MT	13
symbol_window_exists	*gdSymbolWindow* != 0	0	1	bool	9
symbol_window_after_action_point	*gdSymbolWindow* - *gdActionPointOffset* - 1			MT	6
tss_transmitter	*gdTSSTransmitter*			*gdBit*	5
wakeup_symbol_rx_idle	*gdWakeupSymbolRxIdle*			*gdBit*	5
wakeup_symbol_rx_low	*gdWakeupSymbolRxLow*			*gdBit*	3
wakeup_symbol_rx_window	*gdWakeupSymbolRxWindow*			*gdBit*	4
wakeup_symbol_tx_idle	*gdWakeupSymbolTxIdle*			*gdBit*	8
wakeup_symbol_tx_low	*gdWakeupSymbolTxLow*			*gdBit*	5
noise_listen_timeout	(*gListenNoise* * *pdListenTimeout*) - 1			µT	16/17
macro_initial_offset_a	*pMacroInitialOffset[A]*			MT	6
macro_initial_offset_b	*pMacroInitialOffset[B]*			MT	16
macro_per_cycle	*gMacroPerCycle*			MT	10
macro_after_first_static_slot	*gMacroPerCycle* - *gdStaticSlot*			MT	1
macro_after_offset_correction	*gMacroPerCycle* - *gOffsetCorrectionStart*			MT	28
max_without_clock_correction_fatal	*gMaxWithoutClockCorrectionFatal*			cyclepairs	8
max_without_clock_correction_passive	*gMaxWithoutClockCorrectionPassive*			cyclepairs	8
minislot_exists	*gNumberOfMinislots* != 0	0	1	bool	9
minislots_max	*gNumberOfMinislots* - 1			minislot	29
number_of_static_slots	*gNumberOfStaticSlots*			static slot	2
offset_correction_start	*gOffsetCorrectionStart*			MT	11
payload_length_static	*gPayloadLengthStatic*			2-bytes	19
max_payload_length_dynamic	*pPayloadLengthDynMax*			2-bytes	24

Table 778. Protocol configuration register fields (Sheet 1 of 2)

Name	Description(1)	Min	Max	Unit	FR_PCR
first_minislot_action_point_offset	max(*gdActionPointOffset*, *gdMinislotActionPointOffset*) - 1			MT	13
allow_halt_due_to_clock	*pAllowHaltDueToClock*			bool	26
allow_passive_to_active	*pAllowPassiveToActive*			cyclepairs	12
cluster_drift_damping	*pClusterDriftDamping*			µT	24
comp_accepted_startup_range_a	*pdAcceptedStartupRange* - *pDelayCompensation[A]*			µT	22
comp_accepted_startup_range_b	*pdAcceptedStartupRange* - *pDelayCompensation[B]*			µT	26
listen_timeout	*pdListenTimeout* - 1			µT	14/15
key_slot_id	*pKeySlotId*			number	18
key_slot_used_for_startup	*pKeySlotUsedForStartup*			bool	11
key_slot_used_for_sync	*pKeySlotUsedForSync*			bool	11
latest_tx	*gNumberOfMinislots* - *pLatestTx*			minislot	21
sync_node_max	*gSyncNodeMax*			number	30
micro_initial_offset_a	*pMicroInitialOffset[A]*			µT	20
micro_initial_offset_b	*pMicroInitialOffset[B]*			µT	20
micro_per_cycle	*pMicroPerCycle*			µT	22/23
micro_per_cycle_min	*pMicroPerCycle - pdMaxDrift*			µT	24/25
micro_per_cycle_max	*pMicroPerCycle + pdMaxDrift*			µT	26/27
micro_per_macro_nom_half	round(*pMicroPerMacroNom* / 2)			µT	7
offset_correction_out	*pOffsetCorrectionOut*			µT	9
rate_correction_out	*pRateCorrectionOut*			µT	14
single_slot_enabled	*pSingleSlotEnabled*			bool	10
wakeup_channel	*pWakeupChannel*	see *Table 779*			10
wakeup_pattern	*pWakeupPattern*			number	18
decoding_correction_a	*pDecodingCorrection* + *pDelayCompensation[A]* + 2			µT	19
decoding_correction_b	*pDecodingCorrection* + *pDelayCompensation[B]* + 2			µT	7
key_slot_header_crc	header CRC for key slot	0x000	0x7FF	number	12
extern_offset_correction	*pExternOffsetCorrection*			µT	29
extern_rate_correction	*pExternRateCorrection*			µT	21

1. See *FlexRay Communications System Protocol Specification, Version 2.1 Rev A* for detailed protocol parameter definitions

Table 779. Wakeup channel selection

wakeup_channel	Wakeup channel
0	A
1	B

Protocol Configuration Register 0 (FR_PCR0)

Figure 773. Protocol Configuration Register 0 (FR_PCR0)

Base + 0x00A0 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	action_point_offset						static_slot_length									
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Protocol Configuration Register 1 (FR_PCR1)

Figure 774. Protocol Configuration Register 1 (FR_PCR1)

Base + 0x00A2 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	macro_after_first_static_slot													
W	▨	▨														
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Protocol Configuration Register 2 (FR_PCR2)

Figure 775. Protocol Configuration Register 2 (FR_PCR2)

Base + 0x00A4 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	minislot_after_action_point						number_of_static_slots									
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Protocol Configuration Register 3 (FR_PCR3)

Figure 776. Protocol Configuration Register 3 (FR_PCR3)

Base + 0x00A6 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	wakeup_symbol_rx_low						minislot_action_point_offset[4:0]					coldstart_attempts				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Protocol Configuration Register 4 (FR_PCR4)

Figure 777. Protocol Configuration Register 4 (FR_PCR4)

Base + 0x00A8 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	cas_rx_low_max							wakeup_symbol_rx_window								
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Protocol Configuration Register 5 (FR_PCR5)

Figure 778. Protocol Configuration Register 5 (FR_PCR5)

Base + 0x00AA Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	tss_transmitter				wakeup_symbol_tx_low						wakeup_symbol_rx_idle					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Protocol Configuration Register 6 (FR_PCR6)

Figure 779. Protocol Configuration Register 6 (FR_PCR6)

Base + 0x00AC Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	symbol_window_after_action_point								macro_initial_offset_a						
W	▨															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Protocol Configuration Register 7 (FR_PCR7)

Figure 780. Protocol Configuration Register 7 (FR_PCR7)

Base + 0x00AE Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	decoding_correction_b									micro_per_macro_nom_half						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Protocol Configuration Register 8 (FR_PCR8)

Figure 781. Protocol Configuration Register 8 (FR_PCR8)

Base + 0x00B0 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	max_without_clock_correction_fatal				max_without_clock_correction_passive				wakeup_symbol_tx_idle							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Protocol Configuration Register 9 (FR_PCR9)

Figure 782. Protocol Configuration Register 9 (FR_PCR9)

Base + 0x00B2 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	minislot_exists	symbol_window_exists	offset_correction_out													
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Protocol Configuration Register 10 (FR_PCR10)

Figure 783. Protocol Configuration Register 10 (FR_PCR10)

Base + 0x00B4 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	single_slot_enabled	wakeup_channel	macro_per_cycle													
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Protocol Configuration Register 11 (FR_PCR11)

Figure 784. Protocol Configuration Register 11 (FR_PCR11)

Base + 0x00B6 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	key_slot_used_for_startup	key_slot_used_for_sync	offset_correction_start													
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Protocol Configuration Register 12 (FR_PCR12)

Figure 785. Protocol Configuration Register 12 (FR_PCR12)

Base + 0x00B8 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	allow_passive_to_active					key_slot_header_crc										
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Protocol Configuration Register 13 (FR_PCR13)

Figure 786. Protocol Configuration Register 13 (FR_PCR13)

Base + 0x00BA Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	first_minislot_action_point_offset						static_slot_after_action_point									
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Protocol Configuration Register 14 (FR_PCR14)

Figure 787. Protocol Configuration Register 14 (FR_PCR14)

Base + 0x00BC Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	rate_correction_out											listen_timeout[20:16]				
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Protocol Configuration Register 15 (FR_PCR15)

Figure 788. Protocol Configuration Register 15 (FR_PCR15)

Base + 0x00BE Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	listen_timeout[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Protocol Configuration Register 16 (FR_PCR16)

Figure 789. Protocol Configuration Register 16 (FR_PCR16)

Base + 0x00C0 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	macro_initial_offset_b							noise_listen_timeout[24:16]								
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Protocol Configuration Register 17 (FR_PCR17)

Figure 790. Protocol Configuration Register 17 (FR_PCR17)

Base + 0x00C2 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	noise_listen_timeout[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Protocol Configuration Register 18 (FR_PCR18)

Figure 791. Protocol Configuration Register 18 (FR_PCR18)

Base + 0x00C4 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	wakeup_pattern						key_slot_id									
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Protocol Configuration Register 19 (FR_PCR19)

Figure 792. Protocol Configuration Register 19 (FR_PCR19)

Base + 0x00C6 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	decoding_correction_a									payload_length_static						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Protocol Configuration Register 20 (FR_PCR20)

Figure 793. Protocol Configuration Register 20 (FR_PCR20)

Base + 0x00C8 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	micro_initial_offset_b								micro_initial_offset_a							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Protocol Configuration Register 21 (FR_PCR21)

Figure 794. Protocol Configuration Register 21 (FR_PCR21)

Base + 0x00CA Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	extern_rate_correction			latest_tx												
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Protocol Configuration Register 22 (FR_PCR22)

Figure 795. Protocol Configuration Register 22 (FR_PCR22)

Base + 0x00CC Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	R*	comp_accepted_startup_range_a											micro_per_cycle[19:16]			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Protocol Configuration Register 23 (FR_PCR23)

Figure 796. Protocol Configuration Register 23 (FR_PCR23)

Base + 0x00CE Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	micro_per_cycle[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Protocol Configuration Register 24 (FR_PCR24)

Figure 797. Protocol Configuration Register 24 (FR_PCR24)

Base + 0x00D0 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	cluster_drift_damping					max_payload_length_dynamic							micro_per_cycle_min [19:16]			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Protocol Configuration Register 25 (FR_PCR25)

Figure 798. Protocol Configuration Register 25 (FR_PCR25)

Base + 0x00D2 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	micro_per_cycle_min[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Protocol Configuration Register 26 (FR_PCR26)

Figure 799. Protocol Configuration Register 26 (FR_PCR26)

Base + 0x00D4 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	allow_halt_due_to_clock	comp_accepted_startup_range_b											micro_per_cycle_max [19:16]			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Protocol Configuration Register 27 (FR_PCR27)

Figure 800. Protocol Configuration Register 27 (FR_PCR27)

Base + 0x00D6 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	micro_per_cycle_max[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Protocol Configuration Register 28 (FR_PCR28)

Figure 801. Protocol Configuration Register 28 (FR_PCR28)

Base + 0x00D8 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	dynamic_slot_idle_phase		macro_after_offset_correction													
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Protocol Configuration Register 29 (FR_PCR29)

Figure 802. Protocol Configuration Register 29 (FR_PCR29)

Base + 0x00DA Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	extern_offset_correction			minislots_max												
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Protocol Configuration Register 30 (FR_PCR30)

Figure 803. Protocol Configuration Register 30 (FR_PCR30)

Base + 0x00DC Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	sync_node_max			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

ECC Error Interrupt Flag and Enable Register (FR_EEIFER)

Figure 804. ECC Error Interrupt Flag and Enable Register (FR_EEIFER)

Base + 0x00F0 Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	LRNE_OF	LRCE_OF	DRNE_OF	DRCE_OF	LRNE_IF	LRCE_IF	DRNE_IF	DRCE_IF	0	0	0	0	LRNE_IE	LRCE_IE	DRNE_IE	DRCE_IE
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c								
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register provides the means to control the ECC related interrupt request lines and provides the corresponding interrupt flags. The interrupt flags are cleared by writing 1, which resets the corresponding report registers. For a detailed description see *Section , Memory error reporting*.

Table 780. FR_EEIFER field description

Field	Description
Error Overflow Flags	
LRNE_OF	**LRAM Non-Corrected Error Overflow Flag** — This flag is set to 1 when at least one of the following events appears: a) memory errors are detected **but not corrected** on CHI LRAM and interrupt flag LRNE_IF is already 1. b) memory errors are detected **but not corrected** on at least two banks of CHI LRAM 0 no such event 1 Non-Corrected Error overflow detected on CHI LRAM
LRCE_OF	**LRAM Corrected Error Overflow Flag** — This flag is set to 1 when at least one of the following events appears: a) memory errors are detected **and corrected** on CHI LRAM and interrupt flag LRCE_IF is already 1. b) memory errors are detected **and corrected** on at least two banks of CHI LRAM 0 no such event 1 Corrected Error overflow detected on CHI LRAM Note: Error Correction not implemented on CHI LRAM, flag will never be asserted.
DRNE_OF	**DRAM Non-Corrected Error Overflow Flag** — This flag is set to 1 when at least one of the following events appears: a) memory errors are detected **but not corrected** on PE DRAM and interrupt flag DRNE_IF is already 1. b) memory errors are detected **but not corrected** on at least two banks of the PE DRAM 0 no such event 1 Non-Corrected Error overflow detected on PE DRAM
DRCE_OF	**DRAM Corrected Error Overflow Flag** — This flag is set to 1 when at least one of the following events appears: a) memory errors are detected **and corrected** on PE DRAM and interrupt flag DRCE_IF is already 1. b) memory errors are detected **and corrected** on at least two banks of PE DRAM 0no such event 1 Corrected Error overflow detected on PE DRAM

Table 780. FR_EEIFER field description

Field	Description
Error Interrupt Flags	
LRNE_IF	**LRAM Non-Corrected Error Interrupt Flag** — This interrupt flag is set to 1 when a memory error is **detected but not corrected** on the CHI LRAM. 0 no such event 1 Non-Corrected Error detected on CHI LRAM
LRCE_IF	**LRAM Corrected Error Interrupt Flag** — This interrupt flag is set to 1 when a memory error is **detected and corrected** on the CHI LRAM. 0 no such event 1 Corrected Error detected on CHI LRAM Note: Error Correction not implemented on CHI LRAM, flag will never be asserted.
DRNE_IF	**DRAM Non-Corrected Error Interrupt Flag** — This interrupt flag is set to 1 when a memory error is **detected but not corrected** on PE DRAM. 0 no such event 1 Non-Corrected Error detected on PE DRAM
DRCE_IF	**DRAM Corrected Error Interrupt Flag** — This interrupt flag is set to 1 when a memory error is **detected and corrected** on PE DRAM. 0 no such event 1 Corrected Error detected on PE DRAM
Error Interrupt Enables	
LRNE_IE	**LRAM Non-Corrected Error Interrupt Enable** — This flag controls if the LRAM Non-Corrected Error Interrupt line is asserted when the LRNE_IF flag is set. 0 Disable interrupt line 1 Enable interrupt line
LRCE_IE	**LRAM Corrected Error Interrupt Enable** — This flag controls if the LRAM Corrected Error Interrupt line is asserted when the LRCE_IF flag is set. 0 Disable interrupt line 1 Enable interrupt line
DRNE_IE	**DRAM Non-Corrected Error Interrupt Enable** — This flag controls if the DRAM Non-Corrected Error Interrupt line is asserted when the DRNE_IF flag is set. 0 Disable interrupt line 1 Enable interrupt line
DRCE_IE	**DRAM Corrected Error Interrupt Enable** — This flag controls if the DRAM Corrected Error Interrupt line is asserted when the DRCE_IF flag is set. 0 Disable interrupt line 1 Enable interrupt line

ECC Error Report and Injection Control Register (FR_EERICR)

Figure 805. ECC Error Report and Injection Control Register (FR_EERICR)

Base + 0x00F2

Write: ERS: Anytime
ERM, EIM, EIE: IDL

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	BSY	0	0	0	0	0	ERS		0	0	0	ERM	0	0	EIM	EIE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register configures the error injection and error reporting and provides the selector for the content of the report registers.

Table 781. FR_EERICR field description

Field	Description
BSY	**Register Update Busy**— This field indicates the current state of the ECC configuration update and controls the register write access condition IDL specified in *Section , Register write access* 0 ECC configuration is idle 1 ECC configuration is running
ERS	**Error Report Select** — This field selects the content of the ECC Error reporting registers. 00 show PE DRAM non-corrected error information 01 show PE DRAM corrected error information 10 show CHI LRAM non-corrected error information 11 show CHI LRAM corrected error information
ERM	**Error Report Mode** — This bit configures the type of data written into the internal error report registers on the detection of a memory error. 0 store data and code as delivered by ECC decoding logic. 1 store data and code as read from the memory.
EIM	**Error Injection Mode** — This bit configures the ECC error injection mode. 0 use FR_EEIDR[DATA] and FR_EEICR[CODE] as XOR distortion pattern for error injection. 1 use FR_EEIDR[DATA] and FR_EEICR[CODE] as write value for error injection.
EIE	**Error Injection Enable** — This bit configures the ECC error injection on the memories. 0 Error injection disabled 1 Error injection enabled When the ECC functionality is required to be disabled (i.e.value of the FR_MCR[ECCE] is 0), Error Injection Enable bit FR_EERICR[EIE] should not be set to 1.

ECC Error Report Address Register (FR_EERAR)

Figure 806. ECC Error Report Address Register (FR_EERAR)

Base + 0x00F4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MID	BANK			ADDR											
W																
Reset	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0

This register provides the memory identifier, bank, and address for which the memory error is reported.

Table 782. FR_EERAR field description

Field	Description
MID	**Memory Identifier** — This flag provides the memory instance for which the memory error is reported. 0 PE DRAM 1 CHI LRAM
BANK	**Memory Bank** — This field provides the BANK for which the memory error is reported. 111 reset value, indicates no error found after reset. For MID = 0: 000 BANK0: PE DRAM [7:0] 001 BANK1: PE DRAM [15:8] Others – not used For MID = 1: 000 BANK0: FR_MBCCFR(2n) 001 BANK1: FR_MBFIDR(2n) 010 BANK2: FR_MBIDXR(2n) 011 BANK3: FR_MBCCFR(2n+1) 100 BANK4: FR_MBFIDR(2n+1) 101 BANK5: FR_MBIDXR(2n+1) Others – not used
ADDR	**Memory Address** — This field provides the address of the failing memory location.

ECC Error Report Data Register (FR_EERDR)

Figure 807. ECC Error Report Data Register (FR_EERDR)

Base + 0x00F6

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	DATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register provides the data related information of the reported memory read access. The assignment of the bits depends on the selected memory and memory bank as shown in

Table 784.

Table 783. FR_EERDR field description

Field	Description
DATA	**Data** — The content of this field depends on the report mode selected by FR_EERICR[ERM] ERM = 0: ECC Data, shows data as generated by the ECC decoding logic ERM = 1: Memory Data, shows data as read from the memory

Table 784. Valid Bits in FR_EERDR[DATA] / FR_EEIDR[DATA] field

MEM	BANK	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PE DRAM	0	▨	▨	▨	▨	▨	▨	▨	▨	PE DRAM[7:0]							
PE DRAM	1	▨	▨	▨	▨	▨	▨	▨	▨	PE DRAM[15:8]							
CHI LRAM	0	FR_MBCCFR(2n)															
CHI LRAM	1	▨	▨	▨	▨	▨	FR_MBFIDR(2n)										
CHI LRAM	2	▨	▨	▨	▨	▨	▨	▨	▨	FR_MBIDXR(2n)							
CHI LRAM	3	FR_MBCCFR(2n+1)															
CHI LRAM	4	▨	▨	▨	▨	▨	FR_MBFIDR(2n+1)										
CHI LRAM	5	▨	▨	▨	▨	▨	▨	▨	▨	FR_MBIDXR(2n+1)							

ECC Error Report Code Register (FR_EERCR)

Figure 808. ECC Error Report Code Register (FR_EERCR)

Base + 0x00F8

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	CODE				
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register provides the ECC related information of the reported memory read access.

Table 785. FR_EERSR field description

Field	Description
CODE	**Code** — The content of this field depends on the report mode selected by FR_EERICR[ERM] ERM = 0: Syndrome. Shows the ECC syndrome generated by the ECC decoding logic. The coding of the PE DRAM syndrome is shown in *Section , PE DRAM syndrome* The coding of the CHI LRAM syndrome is shown in *Section , CHI LRAM syndrome*. ERM = 1: Checkbits. Shows the ECC checkbits read from the memory.

ECC Error Injection Address Register (FR_EEIAR)

Figure 809. ECC Error Injection Address Register (FR_EEIAR)

Base + 0x00FA Write: IDL

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MID	BANK			ADDR											
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register defines the memory module, bank, and address where the ECC error has to be injected.

Table 786. FR_EEIAR field description

Field	Description
MID	**Memory Identifier** — This flag defines the memory instance for ECC error injection. 0 PE DRAM 1 CHI LRAM
BANK	**Memory Bank** — This field defines the memory bank for ECC error injection. For MID = 0: 000 BANK0: PE DRAM [7:0] 001 BANK1: PE DRAM [15:8] Others – Reserved For MID = 1: 000 BANK0: FR_MBCCFR(2n) 001 BANK1: FR_MBFIDR(2n) 010 BANK2: FR_MBIDXR(2n) 011 BANK3: FR_MBCCFR(2n+1) 100 BANK4: FR_MBFIDR(2n+1) 101 BANK5: FR_MBIDXR(2n+1) Others – Reserved
ADDR	**Memory Address** — This flag defines the memory address for ECC error injection.

ECC Error Injection Data Register (FR_EEIDR)

Figure 810. ECC Error Injection Data Register (FR_EEIDR)

Base + 0x00FC Write: IDL

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	DATA															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register defines the data distortion pattern for the error injection write. The number of valid bits depends on the selected memory and memory bank as shown in *Table 784*.

Table 787. FR_EEIDR field description

Field	Description
DATA	**Data** — The content of this field depends on the error injection mode selected by FR_EERICR[EIM]. EIM = 0: This field defines the XOR distortion pattern for the data written into the memory. EIM = 1: This field defines the data to be written into the memory.

ECC Error Injection Code Register (FR_EEICR)

Figure 811. ECC Error Injection Code Register (FR_EEICR)

Base + 0x00FE

Write: IDL

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	CODE				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register defines the ECC code distortion pattern for the error injection write.

Table 788. FR_EEICR field description

Field	Description
CODE	**Code** — The content of this field depends on the error injection mode selected by FR_EERICR[EIM]. EIM = 0: This field defines the XOR distortion pattern for the ECC checkbits written into the memory. EIM = 1: This field defines the ECC checkbits written into the memory.

Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn)

Figure 812. Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn)

Base + 0x0100 (FR_MBCCSR0)
Base + 0x0108 (FR_MBCCSR1)
...
Base + 0x04F8 (FR_MBCCSR127)

Write: MCM, MBT, MTD: *POC:config* or MB_DIS
CMT: MB_LCK or MB_DIS
EDT, LCKT, MBIE, MBIF: Normal Mode

Additional Reset: CMT, DUP, DVAL, MBIF: Message Buffer Disable

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	MCM	MBT	MTD	CMT	0	0	MBIE	0	0	0	DUP	DVAL	EDS	LCKS	MBIF
W					rwm	EDT	LCKT									w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

The content of these registers comprises message buffer configuration data, message buffer control data, message buffer status information, and message buffer interrupt flags. A detailed description of all flags can be found in *Section 31.6.6, Individual message buffer* functional description.

If the application writes 1 to the EDT bit, no write access to the other register bits is performed.

If the application writes 0 to the EDT bit and 1 to the LCKT bit, no write access to the other bits is performed.

Table 789. FR_MBCCSRn field description

Field	Description
Message Buffer Configuration	
MCM	**Message Buffer Commit Mode** — This bit configures the commit mode of a double buffered message buffer. 0 Streaming commit mode 1 Immediate commit mode
MBT	**Message Buffer Type** — This bit configures the buffering type of a transmit message buffer. 0 Single buffered message buffer 1 Double buffered message buffer
MTD	**Message Buffer Transfer Direction** — This bit configures the transfer direction of a message buffer. 0 Receive message buffer 1 Transmit message buffer
Message Buffer Control	
CMT	**Commit for Transmission** — This bit indicates if the transmit message buffer data are ready for transmission. 0 Message buffer data not ready for transmission 1 Message buffer data ready for transmission
EDT	**Enable/Disable Trigger** — If the application writes 1 to this bit, a message buffer enable or disable is triggered, depending on the current value EDS status bit is 0. 0 No effect 1 Message buffer enable or disable is triggered
LCKT	**Lock/Unlock Trigger** — If the application writes 1 to this bit, a message buffer lock or unlock is triggered, depending on the current value of the LCKS status bit. 0 No effect 1 Message buffer lock or unlock is triggered
MBIE	**Message Buffer Interrupt Enable** — This control bit defines whether the message buffer will generate an interrupt request when its MBIF flag is set. 0 Interrupt request generation disabled 1 Interrupt request generation enabled

Table 789. FR_MBCCSRn field description (continued)

Field	Description
Message Buffer Status	
DUP	**Data Updated** — This status bit indicates whether the frame header in the message buffer header field and the data in the message buffer data field were updated after a frame reception. 0 Frame Header and Message buffer data field not updated 1 Frame Header and Message buffer data field updated
DVAL	**Data Valid** — For receive message buffers this status bit indicates whether the message buffer data field contains valid frame data. For transmit message buffers the status bit indicates if a message is transferred again due to the state transmission mode of the message buffer. 0 receive message buffer contains no valid frame data / message is transmitted for the first time 1 receive message buffer contains valid frame data / message will be transferred again
EDS	**Enable/Disable Status** — This status bit indicates whether the message buffer is enabled or disabled. 0 Message buffer is disabled. 1 Message buffer is enabled.
LCKS	**Lock Status** — This status bit indicates the current lock status of the message buffer. 0 Message buffer is not locked by the application. 1 Message buffer is locked by the application.
MBIF	**Message Buffer Interrupt Flag** — This flag is set when the slot status field of the message buffer was updated after frame transmission or reception, or when a transmit message buffer was just enabled by the application. 0 No such event 1 Slot status field updated or transmit message buffer just enabled

Message Buffer Cycle Counter Filter Registers (FR_MBCCFRn)

Figure 813. Message Buffer Cycle Counter Filter Registers (FR_MBCCFRn)

Base + 0x0102 (FR_MBCCFR0)
Base + 0x010A (FR_MBCCFR1)
...
Base + 0x04FA (FR_MBCCFR127)

16-bit write access required

Write: *POC:config* or MB_DIS

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MTM	CHA	CHB	CCFE	CCFMSK						CCFVAL					
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

This register contains message buffer configuration data for the transmission mode, the channel assignment, and for the cycle counter filtering. For detailed information on cycle counter filtering, refer to *Section , Message buffer cycle counter filtering*.

Table 790. FR_MBCCFRn field description

Field	Description
MTM	**Message Buffer Transmission Mode** — This control bit applies only to transmit message buffers and defines the transmission mode. 0 Event transmission mode 1 State transmission mode
CHA CHB	**Channel Assignment** — These control bits define the channel assignment and control the receive and transmit behavior of the message buffer according to *Table 791*.
CCFE	**Cycle Counter Filtering Enable** — This control bit is used to enable and disable the cycle counter filtering. 0 Cycle counter filtering disabled 1 Cycle counter filtering enabled
CCFMSK	**Cycle Counter Filtering Mask** — This field defines the filter mask for the cycle counter filtering.
CCFVAL	**Cycle Counter Filtering Value** — This field defines the filter value for the cycle counter filtering.

Table 791. Channel assignment description

CHA	CHB	Transmit message buffer		Receive message buffer	
		Static segment	Dynamic segment	Static segment	Dynamic segment
1	1	transmit on both channel A and channel B	Reserved (function not available)	store first valid frame received on either channel A or channel B	Reserved (function not available)
0	1	transmit on channel B	transmit on channel B	store first valid frame received on channel B	store first valid frame received on channel B
1	0	transmit on channel A	transmit on channel A	store first valid frame received on channel A	store first valid frame received on channel A
0	0	no frame transmission	no frame transmission	no frame stored	no frame stored

Note: If at least one message buffer assigned to a certain slot is assigned to both channels, then all message buffers assigned to this slot have to be assigned to both channels. Otherwise, the message buffer configuration is illegal and the result of the message buffer search is not defined.

Message Buffer Frame ID Registers (FR_MBFIDRn)

Figure 814. Message Buffer Frame ID Registers (FR_MBFIDRn)

Base + 0x0104 (FR_MBFIDR0)
Base + 0x010C (FR_MBFIDR1)
...
Base + 0x04FC (FR_MBFIDR127)

16-bit write access required

Write: *POC:config* or MB_DIS

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	FID										
W																
Reset	0	0	0	0	0	–	–	–	–	–	–	–	–	–	–	–

Table 792. FR_MBFIDRn field description

Field	Description
FID	**Frame ID** — The semantic of this field depends on the message buffer transfer type. – *Receive Message Buffer:* This field is used as a filter value to determine if the message buffer is used for reception of a message received in a slot with the slot ID equal to FID. – *Transmit Message Buffer:* This field is used to determine the slot in which the message in this message buffer should be transmitted.

Message Buffer Index Registers (FR_MBIDXRn)

Figure 815. Message Buffer Index Registers (FR_MBIDXRn)

Base + 0x0106 (FR_MBIDXR0)
Base + 0x010E (FR_MBIDXR1)
...
Base + 0x04FE (FR_MBIDXR127)

16-bit write access required

Write: *POC:config* or MB_DIS

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	MBIDX							
W																
Reset	0	0	0	0	0	0	0	0	–	–	–	–	–	–	–	–

Table 793. FR_MBIDXRn field description

Field	Description
MBIDX	**Message Buffer Index** — This field provides the index of the message buffer header field of the physical message buffer that is currently associated with this message buffer. The application writes the index of the initially associated message buffer header field into this register. The CC updates this register after frame reception or transmission.

31.6 Functional description

This section provides a detailed description of the functionality implemented in the CC.

31.6.1 Message buffer concept

The CC uses a data structure called *message buffer* to store frame data, configuration, control, and status data. Each message buffer consists of two parts, the *message buffer control data* and the *physical message buffer.* The message buffer control data are located in dedicated registers. The structure of the message buffer control data depends on the message buffer type and is described in *Section 31.6.3, Message buffer types*. The physical message buffer is located in the FlexRay memory area and is described in *Section 31.6.2, Physical message buffer*.

31.6.2 Physical message buffer

All FlexRay messages and related frame and slot status information of received frames and of frames to be transmitted to the FlexRay bus are stored in data structures called *physical message buffers*. The physical message buffers are located in the FlexRay memory area.The structure of a physical message buffer is depicted in *Figure 816*.

A physical message buffer consists of two fields, the *message buffer header field* and the *message buffer data field*. The message buffer header field contains the *frame header*, the *data field offset*, and the *slot status*.The message buffer data field contains the *frame data*.

The connection between the two fields is established by the *data field offset*.

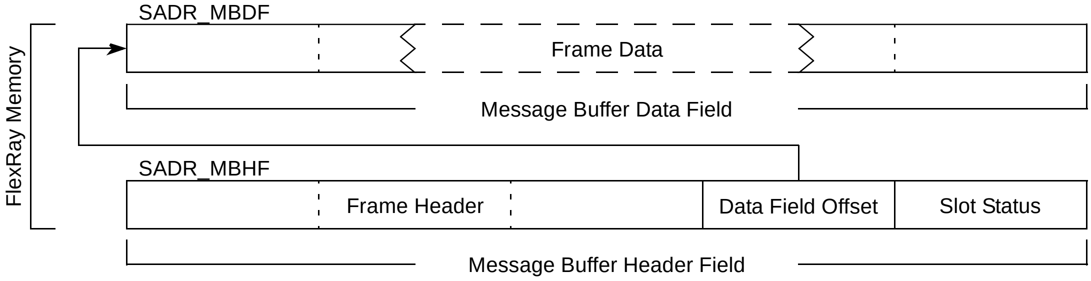


Figure 816. Physical message buffer structure

Message buffer header field

The message buffer header field is a contiguous region in the FlexRay memory area and occupies ten bytes. It contains the frame header, the data field offset, and the slot status. Its structure is shown in *Figure 816*. The physical start address *SADR_MBHF* of the message buffer header field must be 16-bit aligned.

Frame header

The frame header occupies the first six bytes in the message buffer header field. It contains all FlexRay frame header related information according to the *FlexRay Communications System Protocol Specification, Version 2.1 Rev A*. A detailed description of the usage and the content of the frame header is provided in *Section , Frame header description*.

Data field offset

The data field offset follows the frame header in the message buffer data field and occupies two bytes. It contains the offset of the corresponding message buffer data field with respect to the CC FlexRay memory area base address as provided by SMBA field in the *System*

Memory Base Address Register (FR_SYMBADR). The data field offset is used to determine the start address *SADR_MBDF* of the corresponding message buffer data field in the FlexRay memory area according to *Equation 44*.

Equation 44 SADR_MBDF = [Data Field Offset] + SMBA

Slot status

The slot status occupies the last two bytes of the message buffer header field. It provides the slot and frame status related information according to the *FlexRay Communications System Protocol Specification, Version 2.1 Rev A*. A detailed description of the content and usage of the slot status is provided in *Section , Slot status description*.

Message buffer data field

The message buffer data field is a contiguous area of 2-byte entities. This field contains the frame payload data, or a part of it, of the frame to be transmitted to or received from the FlexRay bus. The minimum length of this field depends on the specific message buffer configuration and is specified in the message buffer descriptions given in *Section 31.6.3, Message buffer types*.

31.6.3 Message buffer types

The CC provides three different types of message buffers.

- Individual Message Buffers
- Receive Shadow Buffers
- Receive FIFO Buffers

For each message buffer type the structure of the physical message buffer is identical. The message buffer types differ only in the structure and content of message buffer control data, which control the related physical message buffer. The message buffer control data are described in the following sections.

Individual message buffers

The individual message buffers are used for all types of frame transmission and for dedicated frame reception based on individual filter settings for each message buffer. The CC supports three types of individual message buffers, which are described in *Section 31.6.6, Individual message buffer functional description*.

Each individual message buffer consists of two parts, the physical message buffer, which is located in the FlexRay memory area, and the message buffer control data, which are located in dedicated registers. The structure of an individual message buffer is given in *Figure 817*.

Each individual message buffer has a message buffer number n assigned, which determines the set of message buffer control registers associated to this individual message buffer. The individual message buffer with message buffer number *n* is controlled by the registers FR_MBCCSRn, FR_MBCCFRn, FR_MBFIDRn, and FR_MBIDXRn.

The connection between the message buffer control registers and the physical message buffer is established by the message buffer index field MBIDX in the *Message Buffer Index Registers (FR_MBIDXRn)*. The start address SADR_MBHF of the related message buffer header field in the FlexRay memory area is determined according to *Equation 45*.

Equation 45 SADR_MBHF = (FR_MBIDXRn[MBIDX] * 10) + SMBA

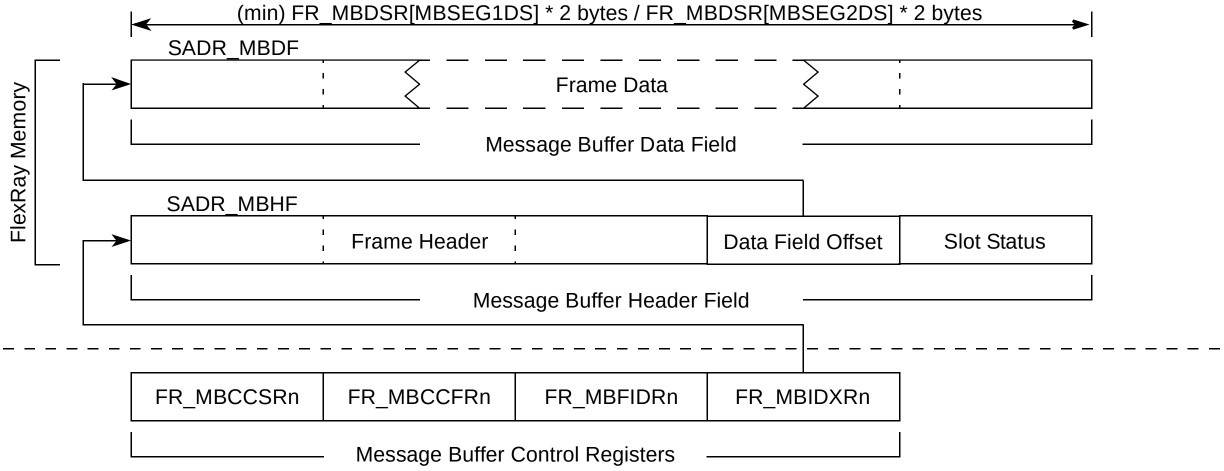


Figure 817. Individual message buffer structure

Individual message buffer segments

The set of the individual message buffers can be split up into two message buffer segments using the *Message Buffer Segment Size and Utilization Register (FR_MBSSUTR)*. All individual message buffers with a message buffer number $n \leq$ FR_MBSSUTR[LAST_MB_SEG1] belong to the first message buffer segment. All individual message buffers with a message buffer number $n >$ FR_MBSSUTR[LAST_MB_SEG1] belong to the second message buffer segment. The following rules apply to the length of the message buffer data field:

- all physical message buffers associated to individual message buffers that belong to the same message buffer segment must have message buffer data fields of the same length
- the minimum length of the message buffer data field for individual message buffers in the first message buffer segment is 2 * FR_MBDSR[MBSEG1DS] bytes
- the minimum length of the message buffer data field for individual message buffers assigned to the second segment is 2 * FR_MBDSR[MBSEG2DS] bytes.

Receive shadow buffers

The receive shadow buffers are required for the frame reception process for individual message buffers. The CC provides four receive shadow buffers, one receive shadow buffer per channel and per message buffer segment.

Each receive shadow buffer consists of two parts, the physical message buffer located in the FlexRay memory area and the receive shadow buffer control registers located in dedicated registers. The structure of a receive shadow buffer is shown in *Figure 818*. The four internal shadow buffer control registers can be accessed by the *Receive Shadow Buffer Index Register (FR_RSBIR)*.

The connection between the receive shadow buffer control register and the physical message buffer for the selected receive shadow buffer is established by the receive shadow buffer index field RSBIDX in the *Receive Shadow Buffer Index Register (FR_RSBIR)*. The start address SADR_MBHF of the related message buffer header field in the FlexRay memory area is determined according to *Equation 46*.

Equation 46 SADR_MBHF = (FR_RSBIR[RSBIDX] * 10) + SMBA

The length required for the message buffer data field depends on the message buffer segment that the receive shadow buffer is assigned to. For the receive shadow buffers assigned to the first message buffer segment, the length must be the same as for the individual message buffers assigned to the first message buffer segment. For the receive shadow buffers assigned to the second message buffer segment, the length must be the same as for the individual message buffers assigned to the second message buffer segment. The receive shadow buffer assignment is described in *Receive Shadow Buffer Index Register (FR_RSBIR)*.

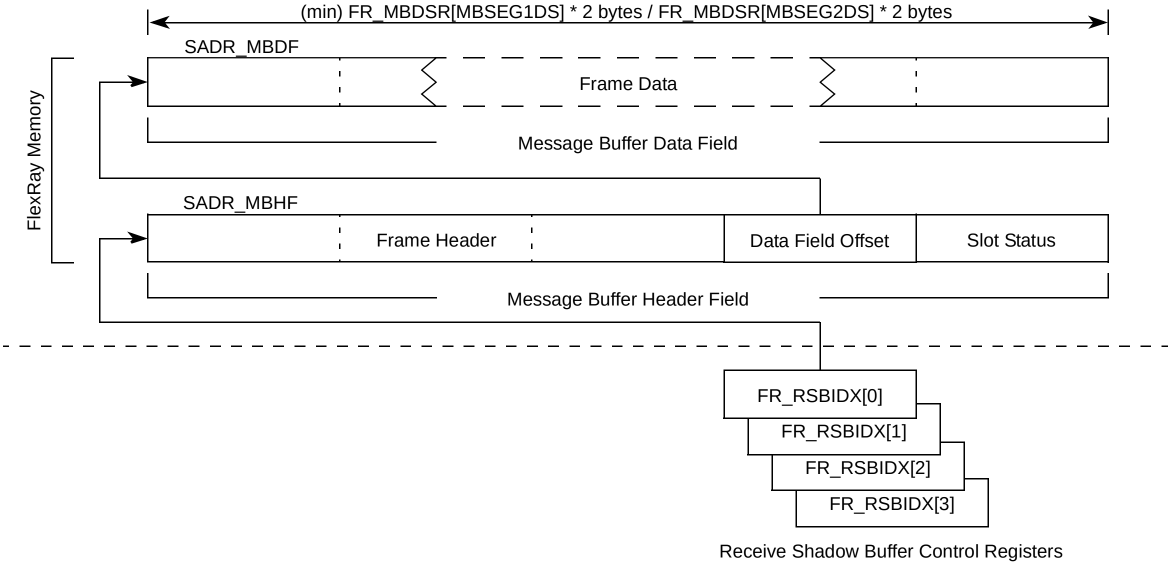


Figure 818. Receive shadow buffer structure

Receive FIFO

The receive FIFO implements a frame reception system based on the FIFO concept. The CC provides two independent receive FIFOs, one per channel.

A receive FIFO consists of a set of physical message buffers in the FlexRay memory area and a set of receive FIFO control registers located in dedicated registers. The structure of a receive FIFO is given in *Figure 819*.

The connection between the receive FIFO control registers and the set of physical message buffers is established by the *Receive FIFO Start Index Register (FR_RFSIR)*, the *Receive FIFO Depth and Size Register (RFDSR)*, and the *Receive FIFO A Read Index Register (FR_RFARIR)* / *Receive FIFO B Read Index Register (FR_RFBRIR)*. The system memory base address SMBA is defined by the system memory base address register selected by the FIFO address mode bit FR_MCR[FAM].

The start byte address SADR_MBHF[1] of the first message buffer header field that belongs to the receive FIFO in the FlexRay memory area is determined according to *Equation 47*.

Equation 47 SADR_MBHF[1] = (10 * FR_RFSIR[SIDX]) + SMBA

The start byte address SADR_MBHF[n] of the last message buffer header field that belongs to the receive FIFO in the FlexRay memory area is determined according to *Equation 48*.

Equation 48 SADR_MBHF[n] = (10 * (FR_RFSIR[SIDX] + RFDSR[FIFO_DEPTH])) + SMBA

Note: All message buffer header fields assigned to a receive FIFO must be a contiguous region.

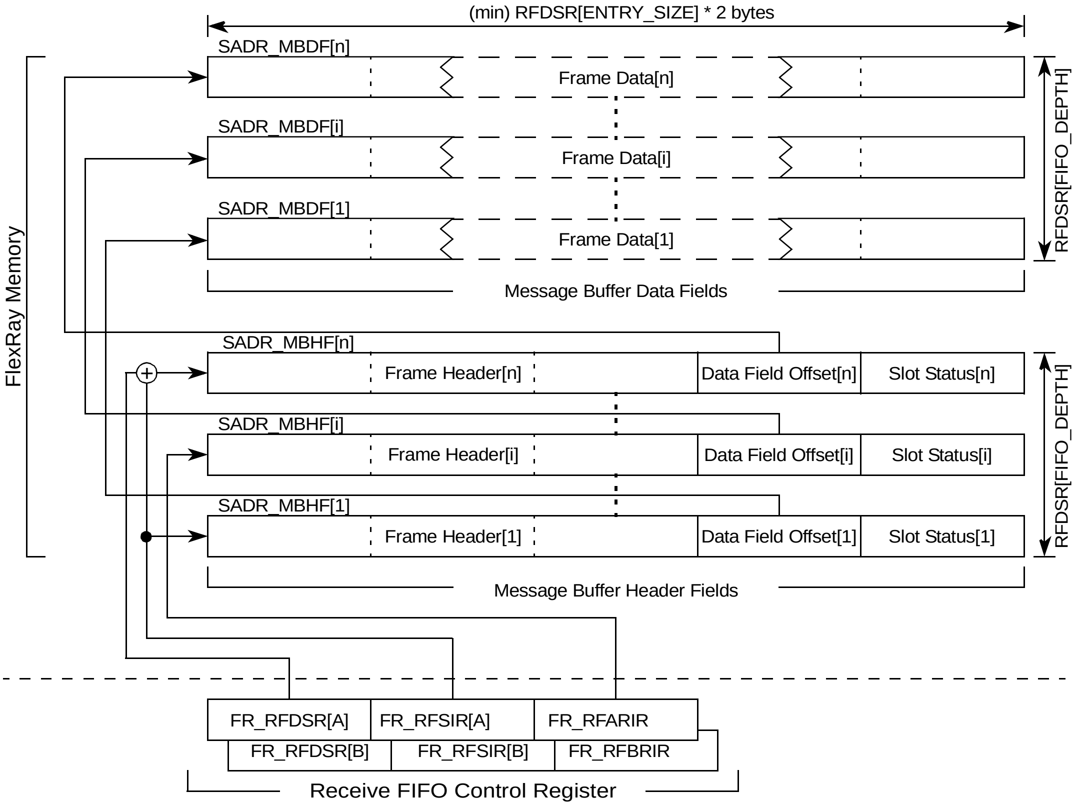


Figure 819. Receive FIFO structure

Message buffer configuration and control data

This section describes the configuration and control data for each message buffer type.

Individual message buffer configuration data

Before an individual message buffer can be used for transmission or reception, it must be configured. There is a set of common configuration parameters that applies to all individual message buffers and a set of configuration parameters that applies to each message buffer individually.

Common configuration data

The set of common configuration data for individual message buffers is located in the following registers.

- *Message Buffer Data Size Register (FR_MBDSR)*
  The MBSEG2DS and MBSEG1DS fields define the minimum length of the message buffer data field with respect to the message buffer segment.
- *Message Buffer Segment Size and Utilization Register (FR_MBSSUTR)*
  The LAST_MB_SEG1 and LAST_MB_UTIL fields define the segmentation of the individual message buffers and the number of individual message buffers that are used. For more details, see *Section , Individual message buffer segments*.

Specific configuration data

The set of message buffer specific configuration data for individual message buffers is located in the following registers.

- *Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn)*
  The MCM, MBT, MTD bits configure the message buffer type.
- *Message Buffer Cycle Counter Filter Registers (FR_MBCCFRn)*
  The MTM, CHA, CHB bits configure the transmission mode and the channel assignment. The CCFE, CCFMSK, and CCFVAL bits and fields configure the cycle counter filter.
- *Message Buffer Frame ID Registers (FR_MBFIDRn)*
  For a transmit message buffer, the FID field is used to determine the slot in which the message in this message buffer will be transmitted.
- *Message Buffer Index Registers (FR_MBIDXRn)*
  This MBIDX field provides the index of the message buffer header field of the physical message buffer that is currently associated with this message buffer.

Individual message buffer control data

During normal operation, each individual message buffer can be controlled by the control and trigger bits CMT, LCKT, EDT, and MBIE in the *Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn)*.

Receive shadow buffer configuration data

Before frame reception into the individual message buffers can be performed, the receive shadow buffers must be configured. The configuration data are provided by the *Receive Shadow Buffer Index Register (FR_RSBIR)*. For each receive shadow buffer, the application provides the message buffer header index. When the protocol is in the *POC:normal active* or *POC:normal passive* state, the receive shadow buffers are under full CC control.

Receive FIFO control and configuration data

This section describes the configuration and control data for the two receive FIFOs.

Receive FIFO configuration data

The CC provides two functional independent receive FIFOs, one per channel. The FIFOs have a common subset of configuration data:

- *Receive FIFO System Memory Base Address Register (FR_RFSYMBADR)*
- *Receive FIFO Periodic Timer Register (FR_RFPTR)*

Each FIFO has its own set of configuration data. The configuration data are located in the following registers:

- *Receive FIFO Watermark and Selection Register (FR_RFWMSR)*
- *Receive FIFO Start Index Register (FR_RFSIR)*
- *Receive FIFO Depth and Size Register (RFDSR)*
- *Receive FIFO Message ID Acceptance Filter Value Register (FR_RFMIDAFVR)*
- *Receive FIFO Message ID Acceptance Filter Mask Register (FR_RFMIDAFMR)*
- *Receive FIFO Frame ID Rejection Filter Value Register (FR_RFFIDRFVR)*
- *Receive FIFO Frame ID Rejection Filter Mask Register (FR_RFFIDRFMR)*
- *Receive FIFO Range Filter Configuration Register (FR_RFRFCFR)*

Receive FIFO control data

The application can access the FIFOs at any time using the control bits in the following registers:

- *Global Interrupt Flag and Enable Register (FR_GIFER)*
- *Receive FIFO Fill Level and POP Count Register (FR_RFFLPCR)*

Receive FIFO status data

The current status of the receive FIFO is provided in the following register:

- *Global Interrupt Flag and Enable Register (FR_GIFER)*
- *Receive FIFO A Read Index Register (FR_RFARIR)*
- *Receive FIFO B Read Index Register (FR_RFBRIR)*
- *Receive FIFO Fill Level and POP Count Register (FR_RFFLPCR)*

31.6.4 FlexRay memory area layout

The CC supports a wide range of possible layouts for the FlexRay memory area. Two basic layout modes can be selected by the FIFO address mode bit FR_MCR[FAM].

FlexRay memory area layout (FR_MCR[FAM] = 0)

Figure 820 shows an example layout for the FIFO address mode FR_MCR[FAM] = 0. In this mode, the following set of rules applies to the layout of the FlexRay memory area:

- The FlexRay memory area is one contiguous region.
- The FlexRay memory area size is maximum 64 Kbytes.
- The FlexRay memory area starts at a 16 byte boundary

The FlexRay memory area contains three areas: the *message buffer header area*, the *message buffer data area*, and the *sync frame table area*.

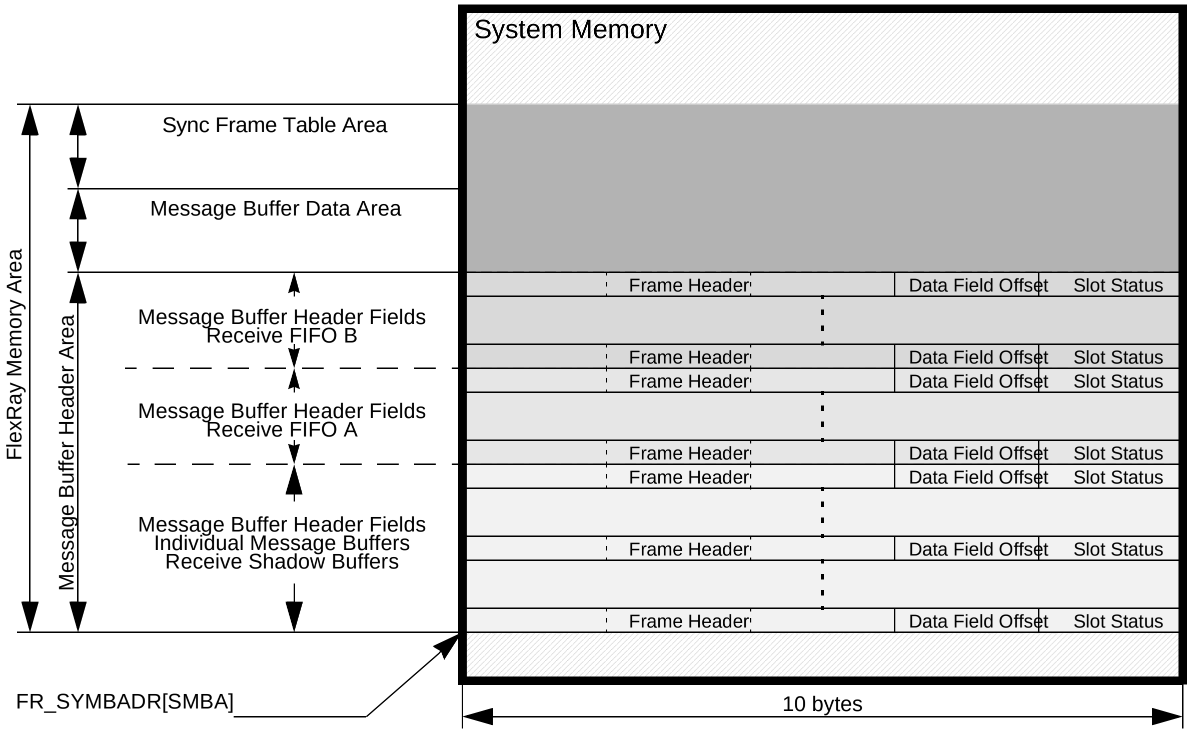


Figure 820. Example of FlexRay memory area layout (FR_MCR[FAM] = 0)

FlexRay memory area layout (FR_MCR[FAM] = 1)

Figure 821 shows an example layout for the FIFO address mode FR_MCR[FAM] = 1. The following set of rules applies to the layout of the FlexRay memory area:

- The FlexRay memory area consists of two contiguous regions.
- The size of each region is maximum 64 Kbytes.
- Each region start at a 16 byte boundary.

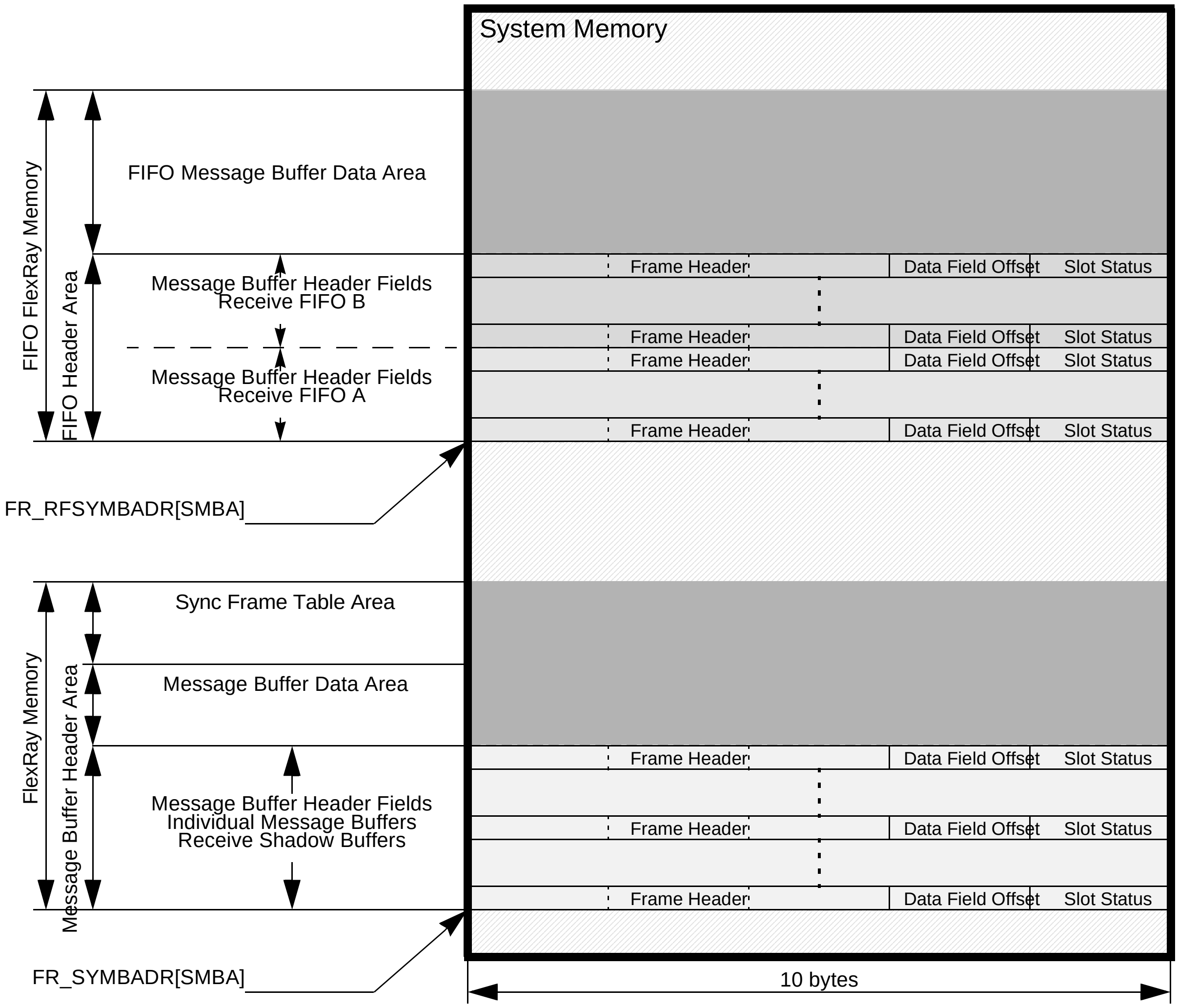


Figure 821. Example of FlexRay memory area layout (FR_MCR[FAM] = 1)

Message buffer header area (FR_MCR[FAM] = 0)

The message buffer header area contains all message buffer header fields of the physical message buffers for all message buffer types. The following rules apply to the message buffer header fields for the three type of message buffers.

1. The start byte address SADR_MBHF of each message buffer header field for *individual message buffers* and *receive shadow buffers* must fulfill *Equation 49*.

Equation 49 SADR_MBHF = (i * 10) + FR_SYMBADR[SMBA]; (0 ≤ i < 256)

2. The start byte address SADR_MBHF of each message buffer header field for the *FIFO* must fulfill *Equation 50*.

Equation 50 SADR_MBHF = (i * 10) + FR_SYMBADR[SMBA]; (0 < i < 1024)

Equation 51 SADR_MBHF = (i * 10) + FR_SYMBADR[SMBA]; $(0 \leq i < 1024)$

3. The message buffer header fields for each FIFO have to be a contiguous area.

Message buffer header area (FR_MCR[FAM] = 1)

The message buffer header area contains all message buffer header fields of the physical message buffers for the individual message buffers and receiver shadow buffers. The following rules apply to the message buffer header fields for the two type of message buffers.

1. The start address SADR_MBHF of each message buffer header field for *individual message buffers* and *receive shadow buffers* must fulfill *Equation 52*.

Equation 52 SADR_MBHF = (i * 10) + FR_SYMBADR[SMBA]; $(0 \leq i < 256)$

FIFO message buffer header area (FR_MCR[FAM] = 1)

The FIFO message buffer header area contains all message buffer header fields of the physical message buffers for the FIFO. The following rules apply to the FIFO message buffer header fields.

1. The start byte address SADR_MBHF of each message buffer header field for the *FIFO* must fulfill *Equation 53*.

Equation 53 SADR_MBHF = (i * 10) + FR_RFSYMBADR[SMBA]; $(0 \leq i < 1024)$

2. The message buffer header fields for each FIFO have to be a contiguous area.

Message buffer data area

The message buffer data area contains all the message buffer data fields of the physical message buffers. Each message buffer data field must start at a 16-bit boundary.

Sync frame table area

The sync frame table area is used to provide a copy of the internal sync frame tables for application access. Refer to *Section 31.6.12, Sync frame ID and sync frame deviation tables* for the description of the sync frame table area.

31.6.5 Physical message buffer description

This section provides a detailed description of the usage and the content of the two parts of a physical message buffer, the message buffer header field and the message buffer data field.

Message buffer protection and data consistency

The physical message buffers are located in the FlexRay memory area. The CC provides no means to protect the FlexRay memory area from uncontrolled or illegal host or other client write access. To ensure data consistency of the physical message buffers, the application must follow the write access scheme that is given in the description of each of the physical message buffer fields.

Message buffer header field description

This section provides a detailed description of the usage and content of the message buffer header field. A description of the structure of the message buffer header fields is given in

Section , Message buffer header field. Each message buffer header field consists of three sections: the frame header section, the data field offset, and the slot status section. For a detailed description of the Data Field Offset, see *Section , Data field offset*.

Frame header description

Frame header content

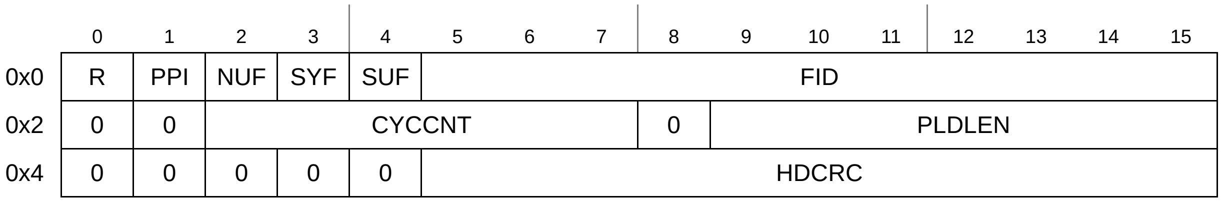
The semantic and content of the frame header section depends on the message buffer type.

For individual receive message buffers and receive FIFOs, the frame header receives the frame header data of the **first valid frame** received on the assigned channels.

For receive shadow buffers, the frame header receives the frame header data of the current frame received regardless of whether the frame is valid or not.

For transmit message buffers, the application writes the frame header of the frame to be transmitted into this location. The frame header will be read out when the frame is transferred to the FlexRay bus.

The structure of the frame header in the message buffer header field for receive message buffers and the receive FIFO is given in *Figure 822*. A detailed description is given in *Table 795*.



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0x0	R	PPI	NUF	SYF	SUF	FID										
0x2	0	0	CYCCNT						0	PLDLEN						
0x4	0	0	0	0	0	HDCRC										

Figure 822. Frame header structure (receive message buffer and receive FIFO)

The structure of the frame header in the message buffer header field for transmit message buffers is given in *Figure 823*. A detailed description is given in *Table 796*. The checks that will be performed are described in *Frame header checks*.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0x0	R	PPI	NUF	SYF	SUF	FID										
0x2			CYCCNT							PLDLEN						
0x4						HDCRC										

Figure 823. Frame header structure (transmit message buffer)

The structure of the frame header in the message buffer header field for transmit message buffers assigned to key slot is given in *Figure 824*.

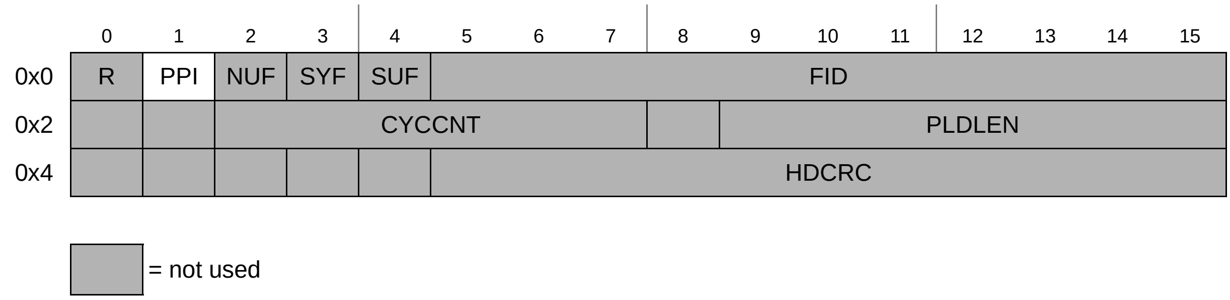


Figure 824. Frame header structure (transmit message buffer for key slot)

Frame header access

The frame header is located in the FlexRay memory area. To ensure data consistency, the application must follow the write access scheme described below.

For receive message buffers, receive shadow buffers, and receive FIFOs, the application must not write to the frame header field.

For transmit message buffers, the application must follow the write access restrictions given in *Table 794*. This table shows the condition under which the application can write to the frame header entries without corrupting the FlexRay message transmission.

Table 794. Frame header write access constraints (transmit message buffer)

Field	Single buffered segments		Double buffered segments			
	Static	Dynamic	Static		Dynamic	
			Commit side	Transmit side	Commit side	Transmit side
FID	*POC:config* or MB_DIS					
PPI, PLDLEN, HDCRC	*POC:config* or MB_DIS or					
		MB_LCK			MB_LCK	

Frame header checks

As shown in *Figure 823* and *Figure 824* not all fields in the message buffer frame header are used for transmission. Some fields in the message buffer frame header are ignored, some are used for transmission, and some of them are checked for correct values. All checks that will be performed are described below.

For message buffers assigned to the key slot, no checks will be performed.

The value of the FID field must be equal to the value of the corresponding *Message Buffer Frame ID Registers (FR_MBFIDRn)*. If the CC detects a mismatch while transmitting the frame header, it will set the frame ID error flag FID_EF in the *CHI Error Flag Register (FR_CHIERFR)*. The value of the FID field will be ignored and replaced by the value provided in the *Message Buffer Frame ID Registers (FR_MBFIDRn)*.

For transmit message buffers assigned to the *static* segment, the PLDLEN value must be equal to the value of the payload_length_static field in the *Protocol Configuration Register 19 (FR_PCR19)*. If this is not fulfilled, the static payload length error flag SPL_EF in the *CHI*

Error Flag Register (FR_CHIERFR) is set when the message buffer is under transmission. A syntactically and semantically correct frame is generated with payload_length_static payload words and the payload length field in the transmitted frame header set to payload_length_static.

For transmit message buffers assigned to the *dynamic* segment, the PLDLEN value must be less than or equal to the value of the max_payload_length_dynamic field in the *Protocol Configuration Register 24 (FR_PCR24)*. If this is not fulfilled, the dynamic payload length error flag DPL_EF in the *CHI Error Flag Register (FR_CHIERFR)* is set when the message buffer is under transmission. A syntactically and semantically correct dynamic frame is generated with PLDLEN payload words and the payload length field in the frame header set to PLDLEN.

Table 795. Frame header field description (receive message buffer and receive FFO)

Field	Description
R	**Reserved Bit** — This is the value of the *Reserved bit* of the received frame stored in the message buffer
PPI	**Payload Preamble Indicator** — This is the value of the *Payload Preamble Indicator* of the received frame stored in the message buffer.
NUF	**Null Frame Indicator** — This is the value of the *Null Frame Indicator* of the received frame stored in the message buffer.
SYF	**Sync Frame Indicator** — This is the value of the *Sync Frame Indicator* of the received frame stored in the message buffer.
SUF	**Startup Frame Indicator** — This is the value of the *Startup Frame Indicator* of the received frame stored in the message buffer.
FID	**Frame ID** — This is the value of the *Frame ID* field of the received frame stored in the message buffer.
CYCCNT	**Cycle Count** — This is the number of the communication cycle in which the frame stored in the message buffer was received.
PLDLEN	**Payload Length** — This is the value of the *Payload Length* field of the received frame stored in the message buffer.
HDCRC	**Header CRC** — This is the value of the *Header CRC* field of the received frame stored in the message buffer.

Table 796. Frame header field description (transmit message buffer)

Field	Description
R	**Reserved Bit** — This bit is not used, the value of the *Reserved bit* is generated internally according to *FlexRay Communications System Protocol Specification, Version 2.1 Rev A*.
PPI	**Payload Preamble Indicator** — This bit provides the value of the *Payload Preamble Indicator* for the frame transmitted from the message buffer.
NUF	**Null Frame Indicator** — This bit is not used, the value of the *Null Frame Indicator* is generated internally according to *FlexRay Communications System Protocol Specification, Version 2.1 Rev A*.
SYF	**Sync Frame Indicator** — This bit is not used, the value of the *Sync Frame Indicator* is generated internally according to *FlexRay Communications System Protocol Specification, Version 2.1 Rev A*.
SUF	**Startup Frame Indicator** — This bit is not used, the value of the *Startup Frame Indicator* is generated internally according to *FlexRay Communications System Protocol Specification, Version 2.1 Rev A*.
FID	**Frame ID** — This field is checked as described in *Frame header checks*.
CYCCNT	**Cycle Count** — This field is not used, the value of the transmitted *Cycle Count* field is taken from the internal communication cycle counter.
PLDLEN	**Payload Length** — This field is checked and used as described in *Frame header checks*.
HDCRC	**Header CRC** — This field provides the value of the *Header CRC* field for the frame transmitted from the message buffer.

Data field offset description

Data field offset content

For a detailed description of the Data Field Offset, see *Section , Data field offset*.

Data field offset access

The application shall program the Data Field Offset when configuring the message buffers either in the *POC:config* state or when the message buffer is disabled.

Slot status description

The slot status is a read-only structure for the application and a write-only structure for the CC. The meaning and content of the slot status in the message buffer header field depends on the message buffer type.

Receive message buffer and receive FIFO slot status description

This section describes the slot status structure for the individual receive message buffers and receive FIFOs. The content of the slot status structure for receive message buffers depends on the message buffer type and on the channel assignment for individual receive message buffers as given by *Table 797*.

Table 797. Receive message buffer slot status content

Receive message buffer type	Slot status content
Individual Receive Message Buffer assigned to both channels FR_MBCCFRn[CHA] = 1 and FR_MBCCFRn[CHB] = 1	see *Figure 825*
Individual Receive Message Buffer assigned to channel A FR_MBCCFRn[CHA] = 1 and FR_MBCCFRn[CHB] = 0	see *Figure 826*
Individual Receive Message Buffer assigned to channel B FR_MBCCFRn[CHA] = 0 and FR_MBCCFRn[CHB] = 1	see *Figure 827*
Receive FIFO Channel A Message Buffer	see *Figure 826*
Receive FIFO Channel B Message Buffer	see *Figure 827*

The meaning of the bits in the slot status structure is explained in *Table 798*.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VFB	SYB	NFB	SUB	SEB	CEB	BVB	CH	VFA	SYA	NFA	SUA	SEA	CEA	BVA	0
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 825. Receive message buffer slot status structure (ChAB)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	VFA	SYA	NFA	SUA	SEA	CEA	BVA	0
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 826. Receive message buffer slot status structure (ChA)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VFB	SYB	NFB	SUB	SEB	CEB	BVB	1	0	0	0	0	0	0	0	0
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 827. Receive message buffer slot status structure (ChB)

Table 798. Receive Message Buffer Slot Status field description)

Field	Description
Common Message Buffer Status Bits	

Table 798. Receive Message Buffer Slot Status field description (continued))

Field	Description
VFB	**Valid Frame on Channel B** — protocol related variable: *vSS!ValidFrame* channel B 0 *vSS!ValidFrame* = 0 1 *vSS!ValidFrame* = 1
SYB	**Sync Frame Indicator Channel B** — protocol related variable: *vRF!Header!SyFIndicator* channel B 0 *vRF!Header!SyFIndicator* = 0 1 *vRF!Header!SyFIndicator* = 1
NFB	**Null Frame Indicator Channel B** — protocol related variable: *vRF!Header!NFIndicator* channel B 0 *vRF!Header!NFIndicator* = 0 1 *vRF!Header!NFIndicator* = 1
SUB	**Startup Frame Indicator Channel B** — protocol related variable: *vRF!Header!SuFIndicator* channel B 0 *vRF!Header!SuFIndicator* = 0 1 *vRF!Header!SuFIndicator* = 1
SEB	**Syntax Error on Channel B** — protocol related variable: *vSS!SyntaxError* channel B 0 *vSS!SyntaxError* = 0 1 *vSS!SyntaxError* = 1
CEB	**Content Error on Channel B** — protocol related variable: *vSS!ContentError* channel B 0 *vSS!ContentError* = 0 1 *vSS!ContentError* = 1
BVB	**Boundary Violation on Channel B** — protocol related variable: *vSS!BViolation* channel B 0 *vSS!BViolation* = 0 1 *vSS!BViolation* = 1
CH	**Channel First Valid Received** — This status bit applies only to receive message buffers assigned to the static segment and to both channels. It indicates the channel that has received the *first valid* frame in the slot. This flag is set to 0 if no valid frame was received at all in the subscribed slot. 0 first valid frame received on channel A, or no valid frame received at all 0 first valid frame received on channel B
VFA	**Valid Frame on Channel A** — protocol related variable: *vSS!ValidFrame* channel A 0 *vSS!ValidFrame* = 0 1 *vSS!ValidFrame* = 1
SYA	**Sync Frame Indicator Channel A** — protocol related variable: *vRF!Header!SyFIndicator* channel A 0 *vRF!Header!SyFIndicator* = 0 1 *vRF!Header!SyFIndicator* = 1
NFA	**Null Frame Indicator Channel A** — protocol related variable: *vRF!Header!NFIndicator* channel A 0 *vRF!Header!NFIndicator* = 0 1 *vRF!Header!NFIndicator* = 1
SUA	**Startup Frame Indicator Channel A** — protocol related variable: *vRF!Header!SuFIndicator* channel A 0 *vRF!Header!SuFIndicator* = 0 1 *vRF!Header!SuFIndicator* = 1
SEA	**Syntax Error on Channel A** — protocol related variable: *vSS!SyntaxError* channel A 0 *vSS!SyntaxError* = 0 1 *vSS!SyntaxError* = 1

Table 798. Receive Message Buffer Slot Status field description (continued))

Field	Description
CEA	**Content Error on Channel A** — protocol related variable: *vSS!ContentError* channel A 0 *vSS!ContentError* = 0 1 *vSS!ContentError* = 1
BVA	**Boundary Violation on Channel A** — protocol related variable: *vSS!BViolation* channel A 0 *vSS!BViolation* = 0 1 *vSS!BViolation* = 1

Transmit message buffer slot status description

This section describes the slot status structure for transmit message buffers. Only the TCA and TCB status bits are directly related to the transmission process. All other status bits in this structure are related to a receive process that may have occurred. The content of the slot status structure for transmit message buffers depends on the channel assignment as given by *Table 799*.

Table 799. Transmit message buffer slot status content

Transmit message buffer type	Slot status content
Individual Transmit Message Buffer assigned to both channels FR_MBCCFRn[CHA] = 1 and FR_MBCCFRn[CHB] = 1	see *Figure 828*
Individual Transmit Message Buffer assigned to channel A FR_MBCCFRn[CHA] = 1 and FR_MBCCFRn[CHB] = 0	see *Figure 829*
Individual Transmit Message Buffer assigned to channel B FR_MBCCFRn[CHA] = 0 and FR_MBCCFRn[CHB] = 1	see *Figure 830*

The meaning of the bits in the slot status structure is described in *Table 798*.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VFB	SYB	NFB	SUB	SEB	CEB	BVB	TCB	VFA	SYA	NFA	SUA	SEA	CEA	BVA	TCA
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 828. Transmit message buffer slot status structure (ChAB)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	VFA	SYA	NFA	SUA	SEA	CEA	BVA	TCA
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 829. Transmit message buffer slot status structure (ChA)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VFB	SYB	NFB	SUB	SEB	CEB	BVB	TCB	0	0	0	0	0	0	0	0
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 830. Transmit message buffer slot status structure (ChB)

Table 800. Transmit Message Buffer Slot Status Structure field description

Field	Description
VFB	**Valid Frame on Channel B** — protocol related variable: *vSS!ValidFrame* channel B 0 *vSS!ValidFrame* = 0 1 *vSS!ValidFrame* = 1
SYB	**Sync Frame Indicator Channel B** — protocol related variable: *vRF!Header!SyFIndicator* channel B 0 *vRF!Header!SyFIndicator* = 0 1 *vRF!Header!SyFIndicator* = 1
NFB	**Null Frame Indicator Channel B** — protocol related variable: *vRF!Header!NFIndicator* channel B 0 *vRF!Header!NFIndicator* = 0 1 *vRF!Header!NFIndicator* = 1
SUB	**Startup Frame Indicator Channel B** — protocol related variable: *vRF!Header!SuFIndicator* channel B 0 *vRF!Header!SuFIndicator* = 0 1 *vRF!Header!SuFIndicator* = 1
SEB	**Syntax Error on Channel B** — protocol related variable: *vSS!SyntaxError* channel B 0 *vSS!SyntaxError* = 0 1 *vSS!SyntaxError* = 1
CEB	**Content Error on Channel B** — protocol related variable: *vSS!ContentError* channel B 0 *vSS!ContentError* = 0 1 *vSS!ContentError* = 1
BVB	**Boundary Violation on Channel B** — protocol related variable: *vSS!BViolation* channel B 0 *vSS!BViolation* = 0 1 *vSS!BViolation* = 1
TCB	**Transmission Conflict on Channel B** — protocol related variable: *vSS!TxConflict* channel B 0 *vSS!TxConflict* = 0 1 *vSS!TxConflict* = 1

Table 800. Transmit Message Buffer Slot Status Structure field description

Field	Description
VFA	**Valid Frame on Channel A** — protocol related variable: *vSS!ValidFrame* channel A 0 *vSS!ValidFrame* = 0 1 *vSS!ValidFrame* = 1
SYA	**Sync Frame Indicator Channel A** — protocol related variable: *vRF!Header!SyFIndicator* channel A 0 *vRF!Header!SyFIndicator* = 0 1 *vRF!Header!SyFIndicator* = 1
NFA	**Null Frame Indicator Channel A** — protocol related variable: *vRF!Header!NFIndicator* channel A 0 *vRF!Header!NFIndicator* = 0 1 *vRF!Header!NFIndicator* = 1
SUA	**Startup Frame Indicator Channel A** — protocol related variable: *vRF!Header!SuFIndicator* channel A 0 *vRF!Header!SuFIndicator* = 0 1 *vRF!Header!SuFIndicator* = 1
SEA	**Syntax Error on Channel A** — protocol related variable: *vSS!SyntaxError* channel A 0 *vSS!SyntaxError* = 0 1 *vSS!SyntaxError* = 1
CEA	**Content Error on Channel A** — protocol related variable: *vSS!ContentError* channel A 0 *vSS!ContentError* = 0 1 *vSS!ContentError* = 1
BVA	**Boundary Violation on Channel A** — protocol related variable: *vSS!BViolation* channel A 0 *vSS!BViolation* = 0 1 *vSS!BViolation* = 1
TCA	**Transmission Conflict on Channel A** — protocol related variable: *vSS!TxConflict* channel A 0 *vSS!TxConflict* = 0 1 *vSS!TxConflict* = 1

Message buffer data field description

The message buffer data field is used to store the frame payload data, or a part of it, of the frame to be transmitted to or received from the FlexRay bus. The minimum required length of this field depends on the message buffer type that the physical message buffer is assigned to and is given in *Table 801*. The structure of the message buffer data field is given in *Figure 831*.

Table 801. Message buffer data field minimum length

Physical message buffer assigned to	Minimum length defined by
Individual Message Buffer in Segment 1	FR_MBDSR[MBSEG1DS]
Receive Shadow Buffer in Segment 1	FR_MBDSR[MBSEG1DS]
Individual Message Buffer in Segment 2	FR_MBDSR[MBSEG2DS]
Receive Shadow Buffer in Segment 2	FR_MBDSR[MBSEG2DS]
Receive FIFO for channel A	FR_RFDSR[ENTRY_SIZE] (FR_RFWMSR[SEL] = 0)
Receive FIFO for channel B	FR_RFDSR[ENTRY_SIZE] (FR_RFWMSR[SEL] = 1)

Note: The CC will not access any locations outside the message buffer data field boundaries given by Table 801.

	0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15
0x0	DATA0 / MID0 / NMV0	DATA1 / MID1 / NMV1
0x2	DATA2 / NMV2	DATA3 / NMV3
...	...	...
0xN-2	DATA N-2	DATA N-1

Figure 831. Message buffer data field structure

The message buffer data field is located in the FlexRay memory area; thus, the CC has no means to control application write access to the field. To ensure data consistency, the application must follow a write and read access scheme.

Message buffer data field read access

For transmit message buffers, the CC will not modify the content of the Message Buffer Data Field. Thus the application can read back the data at any time without any impact on data consistency.

For receive message buffers the application must lock the related receive message buffer and retrieve the message buffer header index from the *Message Buffer Index Registers (FR_MBIDXRn)*. While the message buffer is locked, the CC will not update the Message Buffer Data Field.

For receive FIFOs, the application can read the message buffer indicated by the *Receive FIFO A Read Index Register (FR_RFARIR)* or the *Receive FIFO B Read Index Register (FR_RFBRIR)* when the related fill levels in the *Receive FIFO Fill Level and POP Count Register (FR_RFFLPCR)* indicate an non-empty FIFO.

Message buffer data field write access

For receive message buffers, receive shadow buffers, and receive FIFOs, the application must not write to the message buffer data field.

For transmit message buffers, the application must follow the write access restrictions given in *Table 802*.

Table 802. Frame data write access constraints

Field	Single buffered	Double buffered	
		Commit side	Transmit side
DATA, MID, NMV	*POC:config* or MB_DIS or MB_LCK	*POC:config* or MB_DIS or MB_LCK	*POC:config* or MB_DIS

Table 803. Frame Data field description

Field	Description
DATA 0, DATA 1, ... DATA N-1	**Message Data** — Provides the message data received or to be transmitted. For receive message buffer and receive FIFOs, this field provides the message data received for this message buffer. For transmit message buffers, the field provides the message data to be transmitted.
MID 0, MID 1	**Message Identifier** — If the payload preamble bit PPI is set in the message buffer frame header, the MID field holds the message ID of a dynamic frame located in the message buffer. The receive FIFO filter uses the received message ID for message ID filtering.
NMV 0, NMV 1, ... NMV 11	**Network Management Vector** — If the payload preamble bit PPI is set in the message buffer frame header, the network management vector field holds the network management vector of a static frame located in the message buffer. The MID and NMV bytes replace the corresponding DATA bytes.

31.6.6 Individual message buffer functional description

The CC provides three basic types of individual message buffers:

1. Single transmit message buffers
2. Double transmit message buffers
3. Receive message buffers

Before an individual message buffer can be used, it must be configured by the application. After the initial configuration, the message buffer can be reconfigured later. The set of the configuration data for individual message buffers is given in *Section , Individual message buffer configuration data*.

Individual message buffer configuration

The individual message buffer configuration consists of two steps.

1. The first step is the allocation of the required amount of memory for the FlexRay memory area.
2. The second step is the programming of the message buffer configuration registers, which is described in this section.

Common configuration data

One part of the message buffer configuration data is common to all individual message buffers and the receive shadow buffers. These data can only be set when the protocol is in the *POC:config* state.

The application configures the number of utilized individual message buffers by writing the message buffer number of the last utilized message buffer into the LAST_MB_UTIL field in the *Message Buffer Segment Size and Utilization Register (FR_MBSSUTR)*.

The application configures the size of the two segments of individual message buffers by writing the message buffer number of the last message buffer in the first segment into the LAST_MB_SEG1 field in the *Message Buffer Segment Size and Utilization Register (FR_MBSSUTR)*.

The application configures the length of the message buffer data fields for both of the message buffer segments by writing to the MBSEG2DS and MBSEG1DS fields in the *Message Buffer Data Size Register (FR_MBDSR)*.

Depending on the current receive functionality of the CC, the application must configure the receive shadow buffers. For each segment and for each channel with at least one individual receive message buffer assigned, the application must configure the related receive shadow buffer using the *Receive Shadow Buffer Index Register (FR_RSBIR)*.

Specific configuration data

The second part of the message buffer configuration data is specific for each message buffer.

These data can be changed only when either

- the protocol is in the *POC:config* state or
- the message buffer is disabled, that is, FR_MBCCSRn[EDS] = 0

The individual message buffer type is defined by the MTD and MBT bits in the *Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn)* as given in *Table 804*.

Table 804. Individual message buffer types

FR_MBCCSRn		Individual message buffer description
MTD	**MBT**	
0	0	Receive Message Buffer
0	1	Reserved
1	0	Single Transmit Message Buffer
1	1	Double Transmit Message Buffer

The message buffer specific configuration data are

1. MCM, MBT, MTD bits in *Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn)*
2. all fields and bits in *Message Buffer Cycle Counter Filter Registers (FR_MBCCFRn)*
3. all fields and bits in *Message Buffer Frame ID Registers (FR_MBFIDRn)*
4. all fields and bits in *Message Buffer Index Registers (FR_MBIDXRn)*

The meaning of the specific configuration data depends on the message buffer type, as given in the detailed message buffer type descriptions *Section , Single transmit message buffers*, *Section , Receive message buffers* and *Section , Double transmit message buffer*.

Single transmit message buffers

The section provides a detailed description of the functionality of single buffered transmit message buffers.

A single transmit message buffer is used by the application to provide message data to the CC that will be transmitted over the FlexRay Bus. The CC uses the transmit message buffers to provide information about the transmission process and status information about the slot in which message was transmitted.

The individual message buffer with message buffer number *n* is configured to be a single transmit message buffer by the following settings:

- FR_MBCCSRn[MBT] = 0 (single buffered message buffer)
- FR_MBCCSRn[MTD] = 1 (transmit message buffer)

Access regions

To certain message buffer fields, both the application and the CC have access. To ensure data consistency, a message buffer locking scheme is implemented, which is used to control the access to the data, control, and status bits of a message buffer. The access regions for single transmit message buffers are depicted in *Figure 832*. A description of the regions is given in *Table 805*. If an region is active as indicated in *Table 806*, the access scheme given for that region applies to the message buffer.

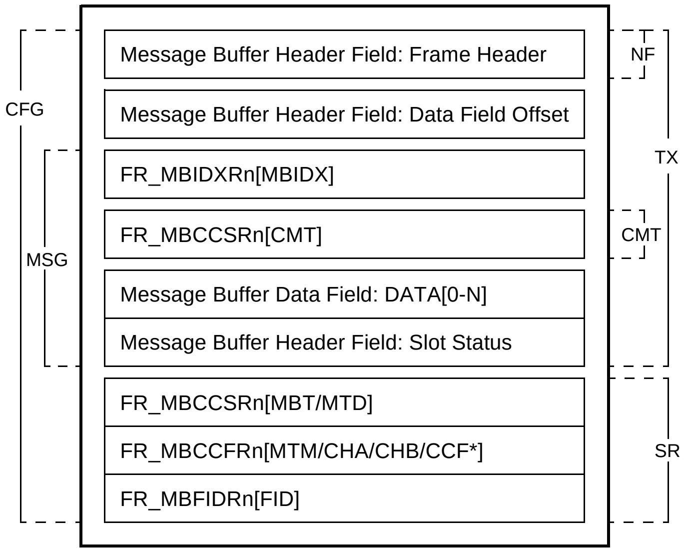


Figure 832. Single transmit message buffer access regions

Table 805. Single transmit message buffer access regions description

Region	Access from		Region used for
	Application	Module	
CFG	read/write	—	Message Buffer Configuration
MSG	read/write	—	Message Data and Slot Status Access
NF	—	read-only	Message Header Access for Null Frame Transmission
TX	—	read/write	Message Transmission and Slot Status Update
CM	—	read-only	Message Buffer Validation
SR	—	read-only	Message Buffer Search

The trigger bits FR_MBCCSRn[EDT] and FR_MBCCSRn[LCKT], and the interrupt enable bit FR_MBCCSRn[MBIE] are not under access control and can be accessed from the application at any time. The status bits FR_MBCCSRn[EDS] and FR_MBCCSRn[LCKS] are not under access control and can be accessed from the CC at any time.

The interrupt flag FR_MBCCSRn[MBIF] is not under access control and can be accessed from the application and the CC at any time. CC clear access has higher priority.

The CC restricts its access to the regions depending on the current state of the message buffer. The application must adhere to these restrictions in order to ensure data consistency. The transmit message buffer states are given in *Figure 833*. A description of the states is given in *Table 806*, which also provides the access scheme for the access regions.

The status bits FR_MBCCSRn[EDS] and FR_MBCCSRn[LCKS] provide the application with the required message buffer status information. The internal status information is not visible to the application.

Message buffer states

This section describes the transmit message buffer states and provides a state diagram.

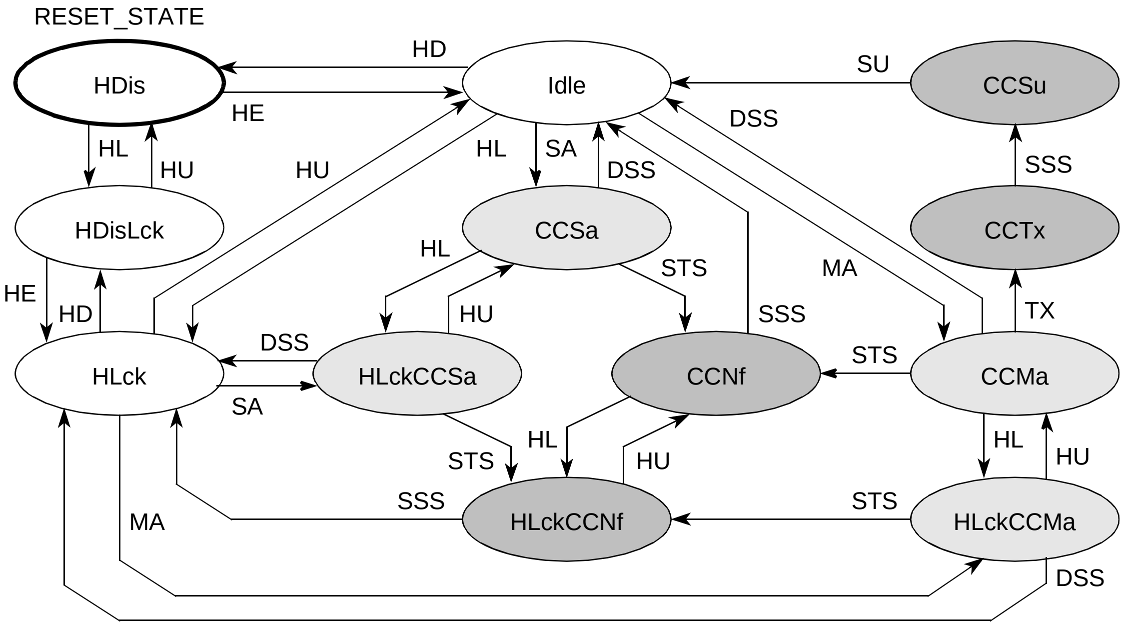


Figure 833. Single transmit message buffer states

Table 806. Single transmit message buffer state description (Sheet 1 of 2)

State	FR_MBCCSRn		Access region		Description
	EDS	LCKS	Appl.	Module	
Idle	1	0	—	CM, SR	**Idle** – Message Buffer is idle. Included in message buffer search.
HDis	0	0	CFG	—	**Disabled** – Message Buffer under configuration. Excluded from message buffer search.
HDisLck	0	1	CFG	—	**Disabled and Locked** – Message Buffer under configuration. Excluded from message buffer search.
HLck	1	1	MSG	SR	**Locked** – Applications access to data, control, and status. Included in message buffer search.
CCSa	1	0	—	—	**Slot Assigned** – Message buffer assigned to next static slot. Ready for Null Frame transmission.
HLckCCSa	1	1	MSG	—	**Locked and Slot Assigned** – Applications access to data, control, and status.Message buffer assigned to next static slot
CCNf	1	0	—	NF	**Null Frame Transmission** – Header is used for null frame transmission
HLckCCNf	1	1	MSG	NF	**Locked and Null Frame Transmission** – Applications access to data, control, and status. Header is used for null frame transmission.
CCMa	1	0	—	CM	**Message Available** – Message buffer is assigned to next slot and cycle counter filter matches.
HLckCCMa	1	1	MSG	—	**Locked and Message Available** – Applications access to data, control, and status. Message buffer is assigned to next slot and cycle counter filter matches.
CCTx	1	0	—	TX	**Message Transmission** – Message buffer data transmit. Payload data from buffer transmitted
CCSu	1	0	—	TX	**Status Update** – Message buffer status update. Update of status flags, the slot status field, and the header index.

Message buffer transitions

Application transitions

The application transitions can be triggered by the application using the commands described in *Table 807*. The application issues the commands by writing to the *Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn)*. Only one command can be issued with one write access. Each command is executed immediately. If the command is ignored, it must be issued again.

Message Buffer Enable and Disable

The enable and disable commands issued by writing 1 to the trigger bit FR_MBCCSRn[EDT]. The transition that will be triggered by each of these command depends on the current value of the status bit FR_MBCCSRn[EDS]. If the command triggers the disable transition HD and the message buffer is in one of the states CCSa, HLckCCSa, CCMa, HLckCCMa, CCNf, HLckCCNf, or CCTx, the disable transition has no effect (command is ignored) and the message buffer state is not changed. No notification is given to the application.

If the communication controller is started as a non-coldstart node, and the message buffers are configured and enabled in the POC config state for Slot 1, then the message buffer cannot be disabled in the INTEGRATION_LISTEN state by directly writing 1 to the EDT bit. To facilitate this, a FREEZE command needs to be issued just before running the message buffer disable for slot 1. Executing this command enables the message buffer disable during the LISTEN states.

Message Buffer Lock and Unlock

The lock and unlock commands issued by writing 1 to the trigger bit FR_MBCCSRn[LCKT]. The transition that will be triggered by each of these commands depends on the current value of the status bit FR_MBCCSRn[LCKS]. If the command triggers the lock transition HL and the message buffer is in the state CCTx, the lock transition has no effect (command is ignored) and message buffer state is not changed. In this case, the message buffer lock error flag LCK_EF in the *CHI Error Flag Register (FR_CHIERFR)* is set.

Table 807. Single transmit message buffer application transitions

Transition	Command	Condition	Description
HE	FR_MBCCSRn[EDT] = 1	FR_MBCCSRn[EDS] = 0	Application triggers message buffer enable
HD		FR_MBCCSRn[EDS] = 1	Application triggers message buffer disable
HL	FR_MBCCSRn[LCKT] = 1	FR_MBCCSRn[LCKS] = 0	Application triggers message buffer lock
HU		FR_MBCCSRn[LCKS] = 1	Application triggers message buffer unlock

Module transitions

The module transitions that can be triggered by the CC are described in *Table 808*. Each transition will be triggered for certain message buffers when the related condition is fulfilled.

Table 808. Single transmit message buffer module transitions

Transition	Condition	Description
SA	Slot match and static slot	**Slot Assigned** – Message buffer is assigned to next static slot.
MA	Slot match and CycleCounter match	**Message Available** – Message buffer is assigned to next slot and cycle counter filter matches.
TX	Slot start and FR_MBCCSRn[CMT] = 1	**Transmission Slot Start** – Slot Start and commit bit CMT is set. In case of a dynamic slot, pLatestTx is not exceeded.
SU	Status updated	**Status Updated** – Slot Status field and message buffer status flags updated. Interrupt flag set.
STS	Static slot start	**Static Slot Start** – Start of static slot.
DSS	Dynamic slot start or symbol window start or NIT start	**Dynamic Slot or Segment Start** – Start of dynamic slot or symbol window or NIT.
SSS	Slot start or symbol window start or NIT start	**Slot or Segment Start** – Start of static slot or dynamic slot or symbol window or NIT.

Transition priorities

The application can trigger only one transition at a time. There is no need to specify priorities among them.

As shown in the first part of *Table 809*, the module transitions have a higher priority than the application transitions. For all states except the CCMa state, both a lock/unlock transition HL/HD and a module transition can be executed at the same time. The result state is reached by first applying the application transition and subsequently the module transition to the intermediately reached state. For example, if the message buffer is in the HLck state and the application unlocks the message buffer by the HU transition and the module triggers the slot assigned transition SA, the intermediate state is Idle and the resulting state is CCSa.

The priorities among the module transitions is given in the second part of *Table 809*.

Table 809. Single transmit message buffer transition priorities

State	Priorities	Description
Module versus application		
Idle, HLck	SA > HD MA > HD	Slot Assigned > Message Buffer Disable Message Available > Message Buffer Disable
CCMa	TX > HL	Transmission Start > Message Buffer Lock
Module internal		
Idle, HLck	MA > SA	Message Available > Slot Assigned
CCMa	TX > STS TX > DSS	Transmission Slot Start > Static Slot Start Transmission Slot Start > Dynamic Slot Start

Transmit message setup

To transmit a message over the FlexRay bus, the application writes the message data into the message buffer data field and sets the commit bit CMT in the *Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn)*. The physical access to the message buffer data field is described in *Section , Individual message buffers*.

As indicated by *Table 806*, the application shall write to the message buffer data field and change the commit bit CMT only if the transmit message buffer is in one of the states HDis, HDisLck, HLck, HLckCCSa, HLckCCMa, or HLckCCMa. The application can change the state of a message buffer if it issues the appropriate commands shown in *Table 807*. The state change is indicated through the FR_MBCCSRn[EDS] and FR_MBCCSRn[LCKS] status bits.

If the transmit message buffer enters one of the states HDis, HDisLck, HLck, HLckCCSa, HLckCCMa, or HLckCCMa the FR_MBCCSRn[DVAL] flag is negated.

Message transmission

As a result of the message buffer search described in *Section 31.6.7, Individual message buffer search*, the CC triggers the message available transition MA for up to two transmit message buffers. This changes the message buffer state from Idle to CCMa and the message buffers can be used for message transmission in the next slot.

The CC transmits a message from a message buffer if both of the following two conditions are fulfilled at the start of the transmission slot:

1. the message buffer is in the message available state CCMa
2. the message data are still valid, that is, FR_MBCCSRn[CMT] = 1

In this case, the CC triggers the TX transition and changes the message buffer state to CCTx. A transmit message buffer timing and state change diagram for message transmission is given in *Figure 834*. In this example, the message buffer with message buffer number n is Idle at the start of the search slot, matches the slot and cycle number of the next slot, and message buffer data are valid, that is, FR_MBCCSRn[CMT] = 1.

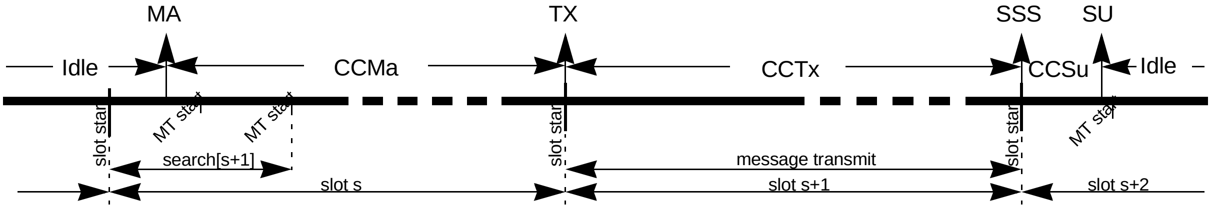


Figure 834. Message transmission timing

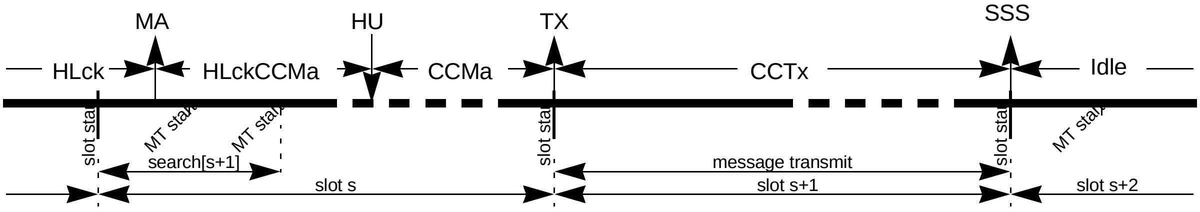


Figure 835. Message transmission from HLck state with unlock

The amount of message data read from the FlexRay memory area and transferred to the FlexRay bus is determined by the following three items

1. the message buffer segment that the message buffer is assigned to, as defined by the *Message Buffer Segment Size and Utilization Register (FR_MBSSUTR)*.
2. the message buffer data field size, as defined by the related field of the *Message Buffer Data Size Register (FR_MBDSR)*
3. the value of the PLDLEN field in the message buffer header field, as described in *Section , Frame header description*

If a message buffer is assigned to message buffer segment 1, and PLDLEN > MBSEG1DS, then 2 * MBSEG1DS bytes will be read from the message buffer data field and zero padding is used for the remaining bytes for the FlexRay bus transfer. If PLDLEN ≤ MBSEG1DS, the CC reads and transfers 2*PLDLEN bytes. The same holds for segment 2 and MBSEG2DS.

Null frame transmission

A static slot with slot number S is assigned to the CC for channel A, if at least one transmit message buffer is configured with the FR_MBFIDRn[FID] set to S and FR_MBCCFRn[CHA] set to 1. A Null Frame is transmitted in the static slot S on channel A, if this slot is assigned to the CC for channel A, and all transmit message buffers with FR_MBFIDRn[FID] = s and FR_MBCCFRn[CHA] = 1 are either not committed, that is, FR_MBCCSRn[CMT] = 0, or locked by the application, that is, FR_MBCCSRn[LCKS] = 1, or the cycle counter filter is enabled and does not match.

Additionally, the application can clear the commit bit of a message buffer that is in the CCMa state, which is called *uncommit* or *transmit abort*. This message buffer will be used for null frame transmission.

As a result of the message buffer search described in *Section 31.6.7, Individual message buffer search*, the CC triggers the slot assigned transition SA for up to two transmit message buffers if at least one of the conditions mentioned above is fulfilled for these message buffers. The transition SA changes the message buffer states from either Idle to CCSa or from HLck to HLckCCSa. In each case, these message buffers will be used for null frame transmission in the next slot. A message buffer timing and state change diagram for null frame transmission from Idle state is given in *Figure 836*.

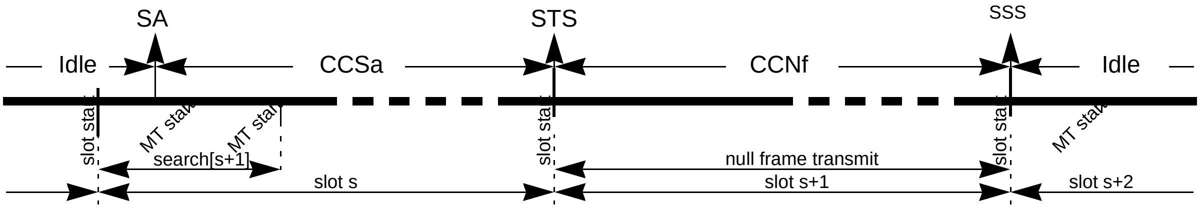


Figure 836. Null frame transmission from idle state

A message buffer timing and state change diagram for null frame transmission from HLck state is given in *Figure 837*.

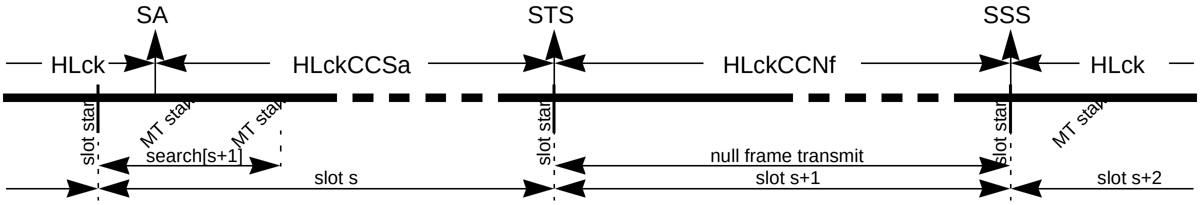


Figure 837. Null frame transmission from HLck state

If a transmit message buffer is in the CCSa or HLckCCSa state at the start of the transmission slot, a null frame is transmitted in any case, even if the message buffer is unlocked or committed before the transmission slot starts. A transmit message buffer timing and state change diagram for null frame transmission for this case is given in *Figure 838*.

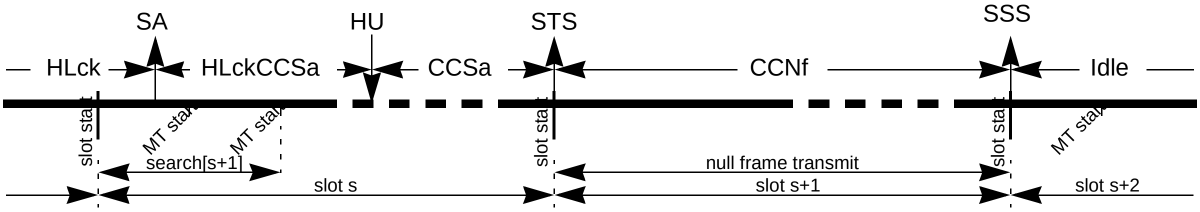


Figure 838. Null frame transmission from HLck state with unlock

Since the null frame transmission will not use the message buffer data, the application can lock/unlock the message buffer during null frame transmission. A transmit message buffer timing and state change diagram for null frame transmission for this case is given in *Figure 839*.

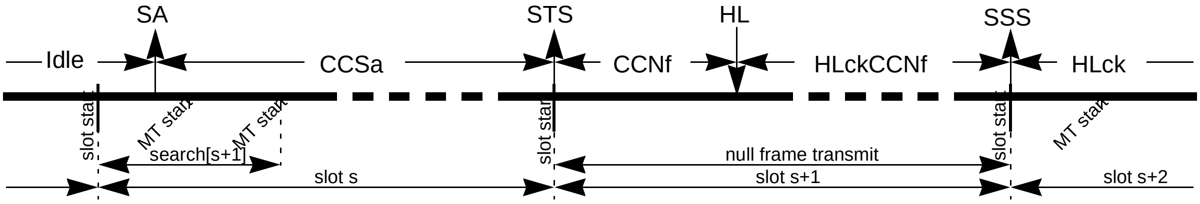


Figure 839. Null frame transmission from idle state with locking

Message buffer status update

After the end of each slot, the PE generates the slot status vector. Depending on the this status, the transmitted frame type, and the amount of transmitted data, the message buffer status is updated.

Message buffer status update after complete message transmission

The term complete message transmission refers to the fact that all payload data stored in the message buffer were send to FlexRay bus. In this case, the CC updates the slot status field of the message buffer and triggers the status updated transition SU. With the SU transition, the CC sets the message buffer interrupt flag FR_MBCCSRn[MBIF] to indicate the successful message transmission.

Depending on the transmission mode flag FR_MBCCFRn[MTM], the CC changes the commit flag FR_MBCCSRn[CMT] and the valid flag FR_MBCCSRn[DVAL]. If the FR_MBCCFRn[MTM] flag is negated, the message buffer is in the *event transmission mode.* In this case, each committed message is transmitted only once. The commit flag FR_MBCCSRn[CMT] is cleared with the SU transition. If the FR_MBCCFRn[MTM] flag is asserted, the message buffer is in the *state transmission mode*. In this case, each committed message is transmitted as long as the application provides new data or locks the message buffers. The CC will not clear the FR_MBCCSRn[CMT] flag at the end of transmission and will set the valid flag FR_MBCCSRn[DVAL] to indicate that the message will be transmitted again.

Message buffer status update after incomplete message transmission

The term incomplete message transmission refers to the fact that not all payload data that should be transmitted were send to FlexRay bus. This may be caused by the following regular conditions in the dynamic segment:

1. The transmission slot starts in a minislot with a minislot number greater than *pLatestTx*.
2. The transmission slot did not exist in the dynamic segment at all.

Additionally, an incomplete message transmission can be caused by internal communication errors. If those error occur, the Protocol Engine Communication Failure Interrupt Flag PECF_IF is set in the *Protocol Interrupt Flag Register 1 (FR_PIFR1)*.

In any of these two cases, the status of the message buffer is not changed at all with the SU transition. The slot status field is not updated, the status and control flags are not changed, and the interrupt flag is not set.

Message buffer status update after null frame transmission

After the transmission of a null frame, the status of the message buffer that was used for the null frame transmission is not changed at all. The slot status field is not updated, the status and control flags are not changed, and the interrupt flag is not set.

Receive message buffers

The section provides a detailed description of the functionality of the receive message buffers. If receive message buffers are used it is required to configure the related receive shadow buffer as described in *Section , Receive shadow buffers*.

A receive message buffer is used to receive a message from the FlexRay Bus based on individual filter criteria. The CC uses the receive message buffer to provide the following data to the application

1. message data received
2. information about the reception process
3. status information about the slot in which the message was received

A individual message buffer with message buffer number *n* is configured as a receive message buffer by the following configuration settings

- FR_MBCCSRn[MBT] = 0 (single buffered message buffer)
- FR_MBCCSRn[MTD] = 0 (receive message buffer)

To certain message buffer fields, both the application and the CC have access. To ensure data consistency, a message buffer locking scheme is implemented that is used to control the access to the data, control, and status bits of a message buffer. The access regions for receive message buffers are depicted in *Figure 840*. A description of the regions is given in *Table 810*. If an region is active as indicated in *Table 811*, the access scheme given for that region applies to the message buffer.

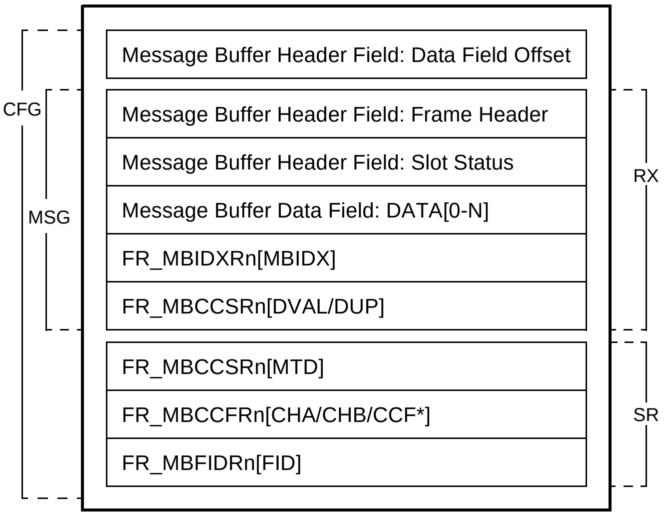


Figure 840. Receive message buffer access regions

Table 810. Receive message buffer access region description

Region	Access from		Region used for
	Application	Module	
CFG	read/write	—	Message Buffer Configuration, Message Data and Status Access
MSG	read/write	—	Message Data, Header, and Status Access
RX	—	write-only	Message Reception and Status Update
SR	—	read-only	Message Buffer Search Data

The trigger bits FR_MBCCSRn[EDT] and FR_MBCCSRn[LCKT] and the interrupt enable bit FR_MBCCSRn[MBIE] are not under access control and can be accessed from the application at any time. The status bits FR_MBCCSRn[EDS] and FR_MBCCSRn[LCKS] are not under access control and can be accessed from the CC at any time.

The interrupt flag FR_MBCCSRn[MBIF] is not under access control and can be accessed from the application and the CC at any time. CC set access has higher priority.

The CC restricts its access to the regions depending on the current state of the message buffer. The application must adhere to these restrictions in order to ensure data consistency. The receive message buffer states are given in *Figure 841*. A description of the message buffer states is given in *Table 806*, which also provides the access scheme for the access regions.

The status bits FR_MBCCSRn[EDS] and FR_MBCCSRn[LCKS] provide the application with the required status information. The internal status information is not visible to the application.

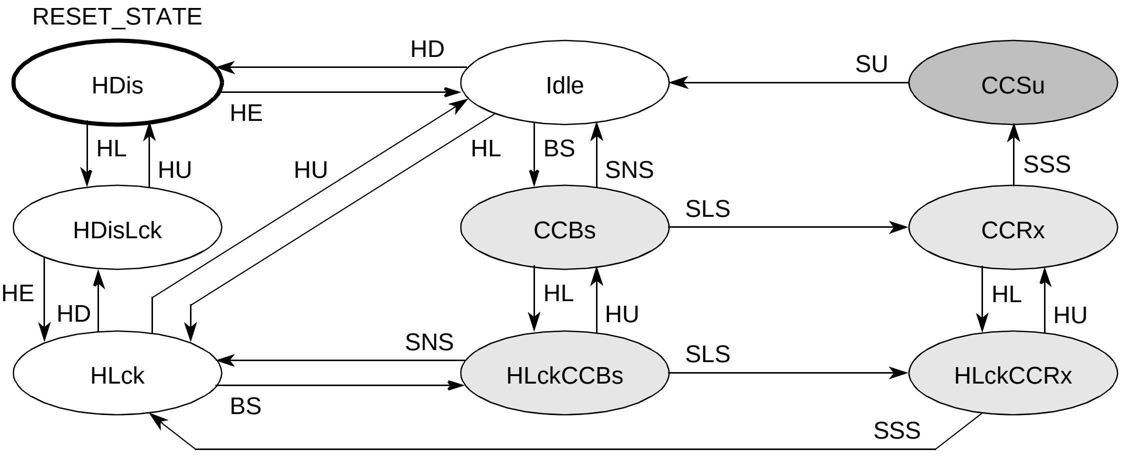


Figure 841. Receive message buffer states

Table 811. Receive message buffer states and access

State	FR_MBCCSRn		Access from		Description
	EDS	LCKS	Appl.	Module	
Idle	1	0	—	SR	**Idle** – Message Buffer is idle. Included in message buffer search.
HDis	0	0	CFG	—	**Disabled** – Message Buffer under configuration. Excluded from message buffer search.
HDisLck	0	1	CFG	—	**Disabled and Locked** – Message Buffer under configuration. Excluded from message buffer search.
HLck	1	1	MSG	—	**Locked** – Applications access to data, control, and status. Included in message buffer search.
CCBs	1	0	—	—	**Buffer Subscribed** – Message buffer subscribed for reception. Filter matches next (slot, cycle, channel) tuple.
HLckCCBs	1	1	MSG	—	**Locked and Buffer Subscribed** – Applications access to data, control, and status. Message buffer subscribed for reception.
CCRx	1	0	—	—	**Message Receive** – Message data received into related shadow buffer.
HLckCCRx	1	1	MSG	—	**Locked and Message Receive** – Applications access to data, control, and status. Message data received into related shadow buffer.
CCSu	1	0	—	RX	**Status Update** – Message buffer status update. Update of status flags, the slot status field, and the header index.

Message buffer transitions

Application transitions

The application transitions that can be triggered by the application using the commands described in *Table 812*. The application issues the commands by writing to the *Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn)*. Only one command can be issued with one write access. Each command is executed immediately. If the command is ignored, it must be issued again.

Message Buffer Enable and Disable

The enable and disable commands issued by writing 1 to the trigger bit FR_MBCCSRn[EDT]. The transition that will be triggered by each of these command depends on the current value of the status bit FR_MBCCSRn[EDS]. If the command triggers the disable transition HD and the message buffer is in one of the states CCBs, HLckCCBs, or CCRx, the disable transition has no effect (command is ignored) and the message buffer state is not changed. No notification is given to the application.

If the communication controller is started as a non-coldstart node, and the message buffers are configured and enabled in the POC config state for Slot 1, then the message buffer cannot be disabled in the INTEGRATION_LISTEN state by directly writing 1 to the EDT bit. To facilitate this, a FREEZE command needs to be issued just before running the message buffer disable for slot 1. Executing this command enables the message buffer disable during the LISTEN states.

Message Buffer Lock and Unlock

The lock and unlock commands issued by writing ‘1’ to the trigger bit FR_MBCCSRn[LCKT]. The transition that will be triggered by each of these commands depends on the current value of the status bit FR_MBCCSRn[LCKS]. If the command triggers the lock transition HL while the message buffer is in the state CCRx, the lock transition has no effect (command is ignored) and message buffer state is not changed. In this case, the message buffer lock error flag LCK_EF in the *CHI Error Flag Register (FR_CHIERFR)* is set.

Table 812. Receive message buffer application transitions

Transition	Host command	Condition	Description
HE	FR_MBCCSRn[EDT] = 1	FR_MBCCSRn[EDS] = 0	Application triggers message buffer enable
HD		FR_MBCCSRn[EDS] = 1	Application triggers message buffer disable
HL	FR_MBCCSRn[LCKT] = 1	FR_MBCCSRn[LCKS] = 0	Application triggers message buffer lock
HU		FR_MBCCSRn[LCKS] = 1	Application triggers message buffer unlock

Module transitions

The module transitions that can be triggered by the CC are described in *Table 813*. Each transition will be triggered for certain message buffers when the related condition is fulfilled.

Table 813. Receive message buffer module transitions

Transition	Condition	Description
BS	Slot match and CycleCounter match	**Buffer Subscribed** – The message buffer filter matches next slot and cycle
SLS	Slot start	**Slot Start** – Start of either Static Slot or Dynamic Slot
SNS	Symbol window start or NIT start	**Symbol Window or NIT Start** – Start of either Symbol Window or NIT
SSS	Slot start or symbol window start or NIT start	**Slot or Segment Start** – Start of either Static Slot, Dynamic Slot, Symbol Window, or NIT
SU	Status updated	**Status Updated** – Slot Status field, message buffer status flags, header index updated; interrupt flag set

Transition priorities

The application can trigger only one transition at a time. There is no need to specify priorities among them.

As shown in *Table 814*, the module transitions have a higher priority than the application transitions. For all states except the CCRx state, a module transition and the application lock/unlock transition HL/HU and can be executed at the same time. The result state is reached by first applying the module transition and subsequently the application transition to the intermediately reached state. For example, if the message buffer is in the buffer subscribed state CCBs and the module triggers the slot start transition SLS at the same time as the application locks the message buffer by the HL transition, the intermediate state is CCRx and the resulting state is locked buffer subscribed state HLckCCRx.

Table 814. Receive message buffer transition priorities

State	Priorities	Description
Module versus application		
Idle	BS > HD	Buffer Subscribed > Message Buffer Disable
HLck	BS > HD	Buffer Subscribed > Message Buffer Disable
CCRx	SSS > HL	Slot or Segment Start > Message Buffer Lock

Message Reception

As a result of the message buffer search, the CC changes the state of up to two enabled receive message buffers from either idle state Idle or locked state HLck to the either subscribed state CCBs or locked buffer subscribed state HLckCCBs by triggering the buffer subscribed transition BS.

If the receive message buffers for the next slot are assigned to both channels, then at most one receive message buffer is changed to a buffer subscribed state.

If more than one matching message buffers assigned to a certain channel, then only the message buffer with the lowest message buffer number is in one of the states mentioned above.

With the start of the next static or dynamic slot the module trigger the slot start transition SLS. This changes the state of the subscribed receive message buffers from either CCBs to CCRx or from HLckCCBs to HLckCCRx, respectively.

During the reception slot, the received frame data are written into the shadow buffers. For details on receive shadow buffers, see *Section , Receive shadow buffers concept*. The data and status of the receive message buffers that are the CCRx or HLckCCRx are not modified in the reception slot.

Message buffer update

With the start of the next static or dynamic slot or with the start of the symbol window or NIT, the module triggers the slot or segment start transition SSS. This transition changes the state of the receiving receive message buffers from either CCRx to CCSu or from HLckCCRx to HLck, respectively.

If a message buffer was in the locked state HLckCCRx, no update will be performed. The received data are lost. This is indicated by setting the Frame Lost Channel A/B Error Flag FRLA_EF/FRLB_EF in the *CHI Error Flag Register (FR_CHIERFR)*.

If a message buffer was in the CCRx state it is now in the CCSu state. After the evaluation of the slot status provided by the PE the message buffer is updated. The message buffer update depends on the slot status bits and the segment the message buffer is assigned to. This is described in *Table 815*.

Table 815. Receive message buffer update

vSS!ValidFrame	*vRF!Header!NFIndicator*	**Update description**
1	1	**Valid non-null frame received** - Message Buffer Data Field updated - Frame Header Field updated - Slot Status Field updated - DUP = 1 - DVAL = 1 - MBIF = 1
1	0	**Valid null frame received** - Message Buffer Data Field **not** updated - Frame Header Field **not** updated - Slot Status Field updated - DUP = 0 - DVAL **not** changed - MBIF = 1
0	x	**No valid frame received** - Message Buffer Data Field not updated - Frame Header Field not updated - Slot Status Field updated - DUP = 0 - DVAL **not** changed - MBIF = 1, if the slot was not an empty dynamic slot An empty dynamic slot is indicated by the following frame and slot status bit values: *vSS!ValidFrame* = 0 and *vSS!SyntaxError* = 0 and *vSS!ContentError* = 0 and *vSS!BViolation* = 0.

Note: *If the number of the last slot in the current communication cycle on a given channel is n, then all receive message buffers assigned to this channel with FR_MBFIDRn[FID] > n will not be updated at all.*

When the receive message buffer update has finished the status updated transition SU is triggered, which changes the buffer state from CCSu to Idle. An example receive message buffer timing and state change diagram for a normal frame reception is given in *Figure 842*.

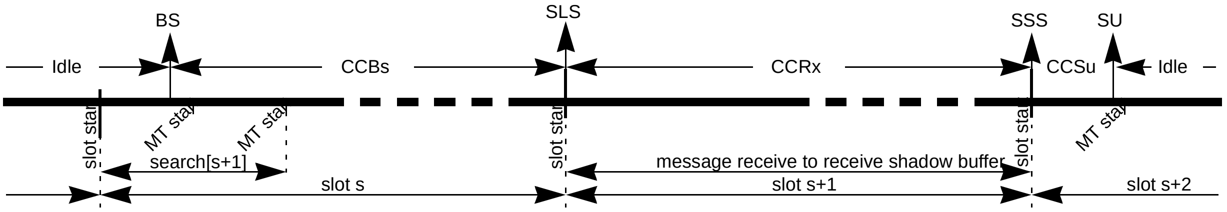


Figure 842. Message reception timing

The amount of message data written into the message buffer data field of the receive shadow buffer is determined by the following two items:

1. the message buffer segment that the message buffer is assigned to, as defined by the *Message Buffer Segment Size and Utilization Register (FR_MBSSUTR)*.
2. the message buffer data field size, as defined by the related field of the *Message Buffer Data Size Register (FR_MBDSR)*
3. the number of bytes received over the FlexRay bus

If the message buffer is assigned to the message buffer segment 1, and the number of received bytes is greater than 2*FR_MBDSR.MBSEG1DS, the CC writes only 2*FR_MBDSR.MBSEG1DS bytes into the message buffer data field of the receive shadow buffer. If the number of received bytes is less than 2*FR_MBDSR.MBSEG1DS, the CC writes only the received number of bytes and will not change the trailing bytes in the message buffer data field of the receive shadow buffer. The same holds for the message buffer segment 2 with FR_MBDSR.MBSEG2DS.

Received message access

To access the message data received over the FlexRay bus, the application reads the message data stored in the message buffer data field of the corresponding receive message buffer. The access to the message buffer data field is described in *Section , Individual message buffers*.

The application can read the message buffer data field if the receive message buffer is one of the states HDis, HDisLck, or HLck. If the message buffer is in one of these states, the CC will not change the content of the message buffer.

Receive shadow buffers concept

The receive shadow buffer concept applies only to individual receive message buffers. The intention of this concept is to ensure that only syntactically and semantically valid received non-null frames are presented to the application in a receive message buffer. The basic structure of a receive shadow buffer is described in *Section , Receive shadow buffers*.

The receive shadow buffers temporarily store the received frame header and message data. After the slot boundary the slot status information is generated. If the slot status information indicates the reception of the valid non-null frame (see *Table 815*), the CC writes the slot status into the slot status field of the receive shadow buffer and exchanges the content of the *Message Buffer Index Registers (FR_MBIDXRn)* with the content of the corresponding internal shadow buffer index register. In all other cases, the CC writes the slot status into the identified receive message buffer, depending on the slot status and the FlexRay segment the message buffer is assigned to.

The shadow buffer concept, with its index exchange, results in the fact that the FlexRay memory area located message buffer associated to an individual receive message buffer changes after successful reception of a valid frame. This means that the message buffer area in the FlexRay memory area accessed by the application for reading the received message is different from the initial setting of the message buffer. Therefore, the application must not rely on the index information written initially into the *Message Buffer Index Registers (FR_MBIDXRn)*. Instead, the index of the message buffer header field must be fetched from the *Message Buffer Index Registers (FR_MBIDXRn)*.

Double transmit message buffer

The section provides a detailed description of the functionality of the double transmit message buffers.

Double transmit message buffers are used by the application to provide the CC with the message data to be transmitted over the FlexRay Bus. The CC uses this message buffer to provide information to the application about the transmission process, and status information about the slot in which message data was transmitted.

In contrast to the single transmit message buffers, the application can provide new transmission data while the transmission of the previously provided message data is running. This scheme is called double buffering and can be considered as a FIFO of depth 2.

Double transmit message buffers are implemented by combining two individual message buffers that form the two sides of an double transmit message buffer. One side is called the *commit side* and will be accessed by the application to provide the message data. The other side is called the *transmit side* and is used by the CC to transmit the message data to the FlexRay bus. The two sides are located in adjacent individual message buffers. The message buffer that implements the commit side has an even message buffer number 2n. The transmit side message buffer follows the commit side message buffer and has the message buffer number 2n+1. The basic structure and data flow of a double transmit message buffer is given in *Figure 843*.

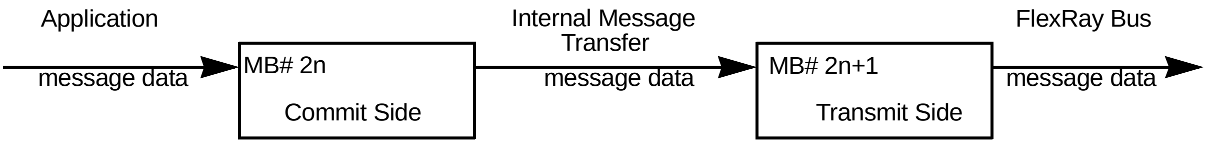


Figure 843. Double transmit buffer structure and data flow

Note: Both the commit and the transmit side must be configured with identical values except for the Message Buffer Index Registers (FR_MBIDXRn).

Access regions

To certain message buffer fields, both the application and the CC have access. To ensure data consistency, a message buffer locking scheme is implemented, which controls the exclusive access to the data, control, and status bits of the message buffer.

The access scheme for double transmit message buffers is depicted in *Figure 844*. The given regions represent fields that can be accessed from both the application and the CC and, thus, require access restrictions. A description of the regions is given in *Table 816*.

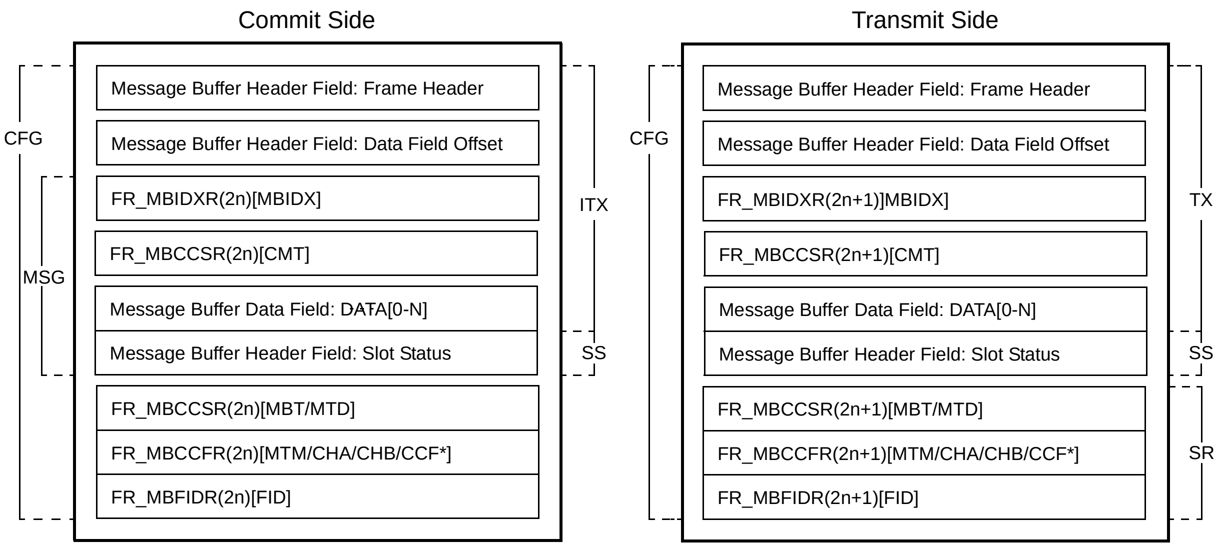


Figure 844. Double transmit message buffer access regions layout

Table 816. Double transmit message buffer access regions description

Access			Description
Region	Type		
	Application	Module	
Commit side			
CFG	read/write	—	Message Buffer Configuration
MSG	read/write	—	Message Buffer Data and Control access
ITX	—	read/write	Internal Message Transfer
SS	—	write-only	Slot Status Update
Transmit side			
CFG	read/write	—	Message Buffer Configuration
SR	—	read-only	Message Buffer Search
TX	—	read-only	Internal Message Transfer, Message Transmission
SS	—	write-only	Slot Status Update

The trigger bits FR_MBCCSRn[EDT] and FR_MBCCSRn[LCKT], and the interrupt enable bit FR_MBCCSRn[MBIE] are not under access control and can be accessed from the application at any time. The status bits FR_MBCCSRn[EDS] and FR_MBCCSRn[LCKS] are not under access control and can be accessed from the CC at any time.

The interrupt flag FR_MBCCSRn[MBIF] is not under access control and can be accessed from the application and the CC at any time. CC set access has higher priority.

The CC restricts its access to the regions, depending on the current state of the corresponding part of the double transmit message buffer. The application must adhere to these restrictions in order to ensure data consistency. The states for the commit side of a double transmit message buffer are given in *Figure 845*. A description of the states is given in *Table 818*. The states for the transmit side of a double transmit message buffer are given in *Figure 846*. A description of the states is given in *Table 818*. The description tables also provide the access scheme for the access regions.

The status bits FR_MBCCSRn[EDS] and FR_MBCCSRn[LCKS] provide the application with the required message buffer status information. The internal status information is not visible to the application.

Message buffer states

This section describes the transmit message buffer states and provides a state diagram.

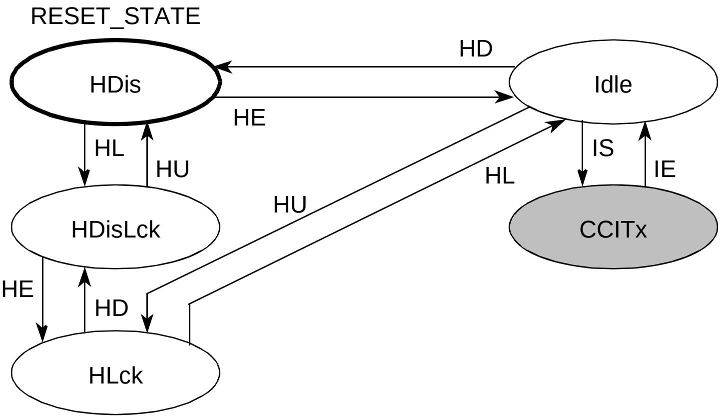


Figure 845. Double transmit message buffer state diagram (commit side)

A description of the states of the commit side of a double transmit message buffer is given in *Table 817*.

Table 817. Double transmit message buffer state description (commit side)

State	FR_MBCCSR(2n)		Access region		Description
	EDS	LCKS	Appl.	Module	
Common states					
HDis	0	0	CFG	—	**Disabled** – Message Buffer under configuration. Commit Side can **not** be used for internal message transfer.
CCITx	1	0	—	ITX	**Internal Message Transfer** – Message Buffer Data transferred from commit side to transmit side.
Commit side specific states					
Idle	1	0	—	ITX, SS	**Idle** – Message Buffer Commit Side is idle. Commit Side can be used for internal message transfer.
HDisLck	0	1	CFG	SS	**Disabled and Locked** – Message Buffer under configuration. Commit Side can **not** be used for internal message transfer.
HLck	1	1	MSG	SS	**Locked** – Applications access to data, control, and status. Commit Side can **not** be used for internal message transfer.

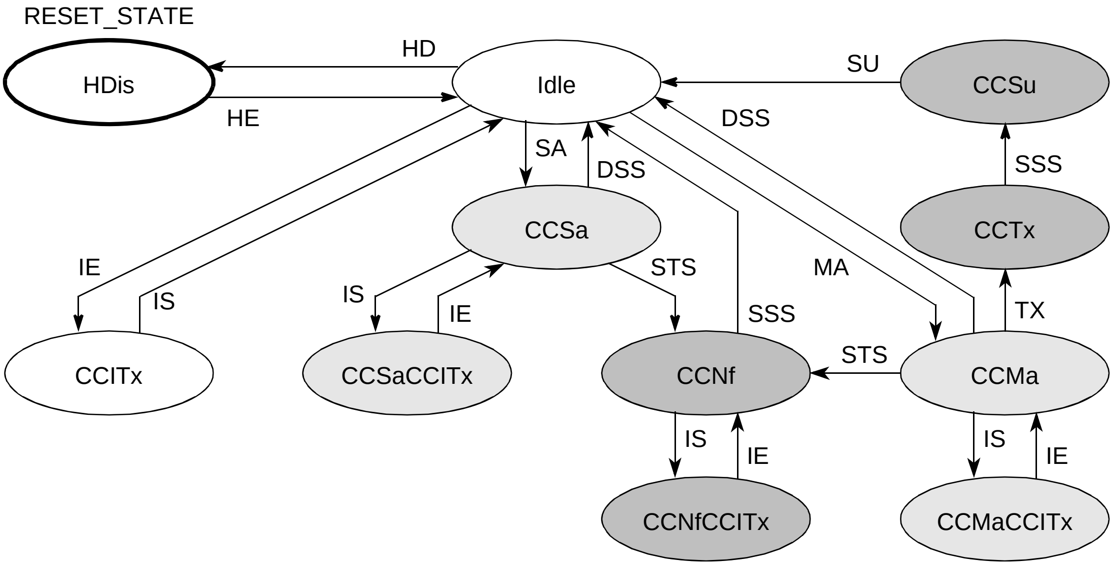


Figure 846. Double transmit message buffer state diagram (transmit side)

A description of the states of the transmit side of a double transmit message buffer is given in *Table 818*.

Table 818. Double transmit message buffer state description (transmit side) (sheet 2 of 2)

State	FR_MBCCSRn		Access region		Description
	EDS	LCKS	Appl.	Module	
Common states					
HDis	0	0	CFG	—	**Disabled** – Message Buffer under configuration. Excluded from message buffer search.
CCITx	1	0	—	TX	**Internal Message Transfer** – Message Buffer Data transferred from commit side to transmit side
Transmit side specific states					
Idle	1	0	—	SR	**Idle** – Message Buffer Transmit Side is idle. Transmit Side is included in message buffer search.
CCSa	1	0	—	—	**Slot Assigned** – Message buffer assigned to next static slot. Ready for Null Frame transmission.
CCSaCCITx	1	0	—	TX	**Slot Assigned and Internal Message Transfer** – Message buffer assigned to next static slot and Message Buffer Data transferred from commit side to transmit side
CCNf	1	0	—	TX	**Null Frame Transmission** – Header is used for null frame transmission
CCNfCCITx	1	0	—	TX	**Null Frame Transmission and Internal Message Transfer** – Header is used for null frame transmission and Message Buffer Data transferred from commit side to transmit side
CCMa	1	0	—	—	**Message Available** – Message buffer is assigned to next slot and cycle counter filter matches
CCMaCCITx	1	0	—	—	**Message Available and Internal Message Transfer** – Message buffer is assigned to next slot and cycle counter filter matches and Message Buffer Data transferred from commit side to transmit side.
CCTx	1	0	—	TX	**Message Transmission** – Message buffer data transmit. Payload data from buffer transmitted
CCSu	1	0	—	SS	**Status Update** – Message buffer status update. Update of status flags, the slot status field, and the header index. **Note:** The slot status field of the commit side is updated too, even if the application has locked the commit side.

Message buffer transitions

Application transitions

The application transitions that can be triggered by the application using the commands described in *Table 819*. The application issues the commands by writing to the *Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn)*. Only one command can be issued with one write access. Each command is executed immediately. If the command is ignored, it must be issued again.

Message Buffer Enable and Disable

The enable and disable commands can be issued on the transmit side only. Any enable or disable command issued on the commit side will be ignored without notification. The transitions that will be triggered depends on the value of the EDS bit. The enable and disable commands will affect both the commit side and the transmit side at the same time. If the application triggers the disable transition HD while the transmit side is in one of the states CCSa, CCSaCCITx, CCNf, CCNfCCITx, CCMa, CCMaCCITx, CCTx, or CCSu, the disable transition has no effect (command is ignored) and the message buffer state is not changed. No notification is given to the application.

Message Buffer Lock and Unlock

The lock and unlock commands can be issued on the commit side only. Any lock or unlock command issued on the transmit side will be ignored and the double transmit buffer lock error flag DBL_EF in the *CHI Error Flag Register (FR_CHIERFR)* will be set. The transitions that will be triggered depends on the current value of the LCKS bit. The lock and unlock commands will only affect the commit side. If the application triggers the lock transition HL while the commit side is in the state CCITx, the message buffer state will not be changed and the message buffer lock error flag LCK_EF in the *CHI Error Flag Register (FR_CHIERFR)* will be set.

Table 819. Double transmit message buffer host transitions

Transition	Host command	Condition	Description
HE	FR_MBCCSR(2n+1)[EDT] = 1	FR_MBCCSR(2n+1)[EDS] = 0	Application triggers message buffer enable
HD		FR_MBCCSR(2n+1)[EDS] = 1	Application triggers message buffer disable
HL	FR_MBCCSR(2n)[LCKT] = 1	FR_MBCCSR(2n)[LCKS] = 0	Application triggers message buffer lock
HU		FR_MBCCSR(2n)[LCKS] = 1	Application triggers message buffer unlock

Module transitions

The module transitions that can be triggered by the CC are described in *Table 820*. The transitions C1 and C2 apply to both sides of the message buffer and are applied at the same time. All other CC transitions apply to the transmit side only.

Table 820. Double transmit message buffer module transitions

Transition	Condition	Description
Common transitions		
IS	See *Section , Internal message transfer*	**Internal Message Transfer Start** – Start transfer of message data from commit side to transmit side
IE		**Internal Message Transfer End** – Stop transfer of message data from commit side to transmit side **Note:** The internal message transfer is stopped before the slot or segment start.
Transmit side specific transitions		
SA	Slot match and static slot	**Slot Assigned** – Message buffer is assigned to next static slot
MA	Slot match and CycleCounter match	**Message Available** – Message buffer is assigned to next slot and cycle counter filter matches
TX	Slot start and FR_MBCCSR(2n + 1) [CMT] = 1	**Transmission Slot Start** – Slot Start and commit bit CMT is set. In case of a dynamic slot, pLatestTx is not exceeded.
SU	Status updated	**Status Updated** – Slot Status field and message buffer status flags updated. Interrupt flag set.
STS	Static slot start	**Static Slot Start** – Start of static slot.
DSS	Dynamic slot start or symbol window start or NIT start	**Dynamic Slot or Segment Start** – Start of dynamic slot or symbol window or NIT
SSS	Slot start or symbol window start or NIT start	**Slot or Segment Start** – Start of static slot or dynamic slot or symbol window or NIT

Transition priorities

The application can trigger only one transition at a time. There is no need to specify priorities among them.

As shown in the first part of *Table 821*, the module transitions have a higher priority than the application transitions. The priorities among the CC transitions and the related states are given in the second part of *Table 821*. These priorities apply only to the transmit side. The internal message transmit start transition IS has tho lowest priority.

Table 821. Double transmit message buffer transition priorities

State	Priority	Description
Module versus application		
Idle	IS > HD IS > HL	Internal Message Transfer Start > Message Buffer Disable Internal Message Transfer Start > Message Buffer Lock
Module internal		
Idle	MA > SA	Message Available > Slot Assigned
CCMa	TX > STS TX > DSS	Transmission Slot Start > Static Slot Start Transmission Slot Start > Dynamic Slot Start

Message preparation

The application provides the message data through the commit side. The transmission itself is executed from the transmit side. The transfer of the message data from the commit side to the transmit side is done by the *Internal Message Transfer*, which is described in *Section , Internal message transfer*

To transmit a message over the FlexRay bus, the application writes the message data into the message buffer data field of the commit side and sets the commit bit CMT in the *Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn)*. The physical access to the message buffer data field is described in *Section , Individual message buffers*.

As indicated by *Table 817*, the application shall write to the message buffer data field and change the commit bit CMT only if the transmit message buffer is in one of the states HDis, HDisLck, or HLck. The application can change the state of a message buffer if it issues the appropriate commands shown in *Table 819*. The state change is indicated through the FR_MBCCSRn[EDS] and FR_MBCCSRn[LCKS] status bits.

Internal message transfer

The internal message transfer transfers the message data from the commit side to the transmit side. The internal message transfer is implemented as the swapping of the content of the *Message Buffer Index Registers (FR_MBIDXRn)* of the commit side and the transmit side. After the swapping, the commit side CMT bit is cleared, the commit side interrupt flag MBIF is set, the transmit side CMT bit is set, and the transmit side DVAL bit is cleared.

The conditions and the point in time when the internal message transfer is started are controlled by the message buffer commit mode bit MCM in the *Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn)*. The MCM bit configures the message buffer for either the streaming commit mode or the immediate commit mode. A detailed description is given in *Streaming commit mode* and *Immediate commit mode*. The Internal Message Transfer is triggered with the transition IS. Both sides of the message buffer enter one of the CCITx states. The internal message transfer is finished with the transition IE.

Streaming commit mode

The intention of the streaming commit mode is to ensure that each committed message is transmitted **at least once**. The CC will not start the Internal Message Transfer for a message buffer as long as the message data on the transmit side is not transmitted at least once.

The streaming commit mode is configured by clearing the message buffer commit mode bit MCM in the *Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn)*.

In this mode, the internal message transfer from the commit side to the transmit side is started for a double transmit message buffer when all of the following conditions are fulfilled

1. the commit side is in the Idle state
2. the commit site message data are valid, that is, FR_MBCCSR(2n)[CMT] = 1
3. the transmit side is in one of the states Idle, CCSa, or CCMa
4. the transmit side contains either no valid message data, that is, FR_MBCCSR(2n+1)[CMT] = 0 or
   the message data were transmitted at least once, that is, FR_MBCCSR(2n+1)[DVAL] = 1

An example of a streaming commit mode state change diagram is given in *Figure 847*. In this example, both the commit and the transmit side do not contain valid message data and the application provides two messages. The message buffer does not match the next slot.

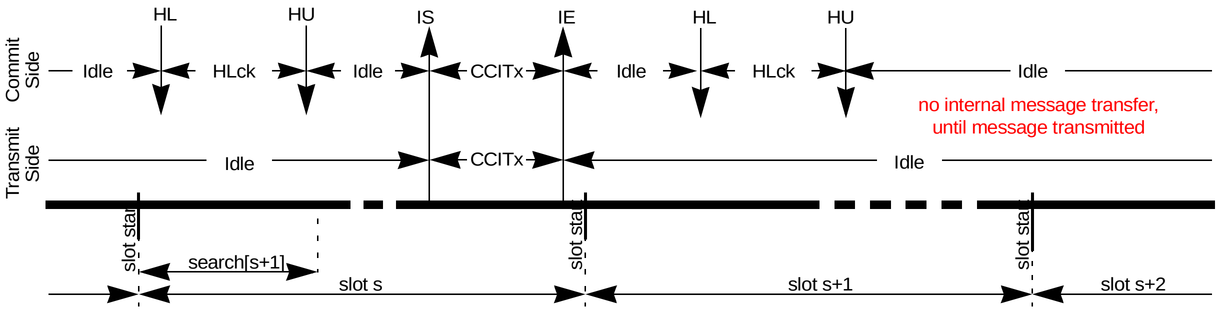


Figure 847. Internal message transfer in streaming commit mode

Immediate commit mode

The intention of the immediate commit mode is to transmit the **latest** data provided by the application. This implies that it is not guaranteed that each provided message will be transmitted at least once.

The immediate commit mode is configured by setting the message buffer commit mode bit MCM in the *Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn)*.

In this mode, the internal message transfer from the commit side to the transmit side is started for one double transmit message buffer when all of the following conditions are fulfilled

1. the commit side is in the Idle state
2. the commit site message data are valid, that is, FR_MBCCSR(2n)[CMT] = 1
3. the transmit side is in one of the states Idle, CCSa, or CCMa

It is not checked whether the transmit side contains no valid message data or valid message data were transmitted at least once. If message data are valid and not transmitted, they may be overwritten.

An example of a streaming commit mode state change diagram is given in *Figure 848*. In this example, both the commit and the transmit side do not contain valid message data, and the application provides two messages and the first message is gets overwritten. The message buffer does not match the next slot.

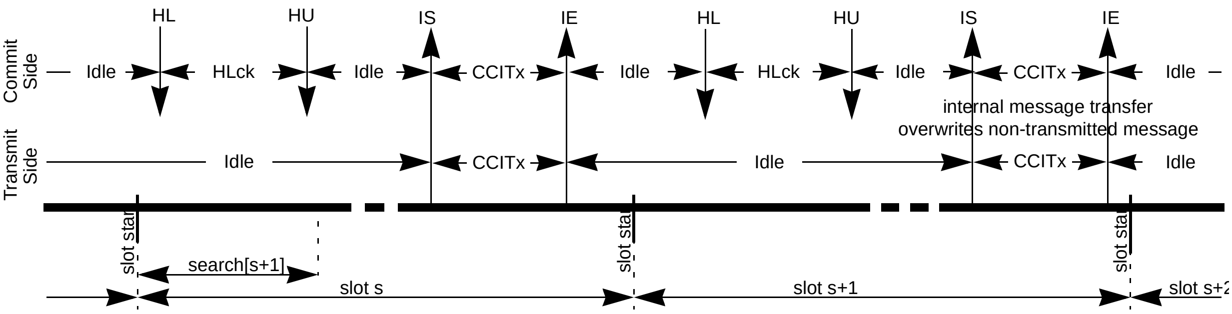


Figure 848. Internal message transfer in immediate commit mode

Message transmission

For double transmit message buffers, the message buffer search checks only the transmit side part. The internal scheduling ensures, that the internal message transfer is stopped on the message buffer search start. Thus, the transmit side of message buffer, that is not in its transmission or status update slot, is always in the Idle state.

The message transmit behavior and transmission state changes of the transmit side of a double transmit message buffer are the same as for single buffered transmit buffers, except that the transmit side of double buffers can not be locked by the application, that is, the HU and HL transition do not exist. Therefore, refer to *Section , Message transmission*.

Message buffer status update

The message buffer status update behavior of the transmit side of a double transmit message buffer is the same as for single transmit message buffers which is described in *Section , Message buffer status update*.

Additionally, the slot status field of the commit side is update after the update of the slot status field of the transmit side, even if the commit side is locked by the application. This is implemented to provide the slot status of the most recent transmission slot.

31.6.7 Individual message buffer search

This section provides a detailed description of the message buffer search algorithm.

The message buffer search determines for each enabled channel if a slot *s* in a communication cycle *c* is assigned for frame or null frame transmission or if it is subscribed for frame reception on that channel.

The message buffer search is a sequential algorithm which is invoked at the following protocol related events:

1. NIT start
2. slot start in the static segment
3. minislot start in the dynamic segment

The message buffer search within the NIT searches for message buffers assigned or subscribed to slot 1. The message buffer search within slot *n* searches for message buffers assigned or subscribed to slot *n+1*.

In general, the message buffer search for the next slot *n* considers only message buffers which are

1. enabled, that is, FR_MBCCSRn[EDS] = 1, and
2. matches the next slot *n*, that is, FR_MBFIDRn[FID] = *n*, and
3. are the transmit side buffer in case of a double transmit message buffer.

On top of that, for the static segment only those message buffers are considered, that match the condition of at least one row of *Table 822*. For the dynamic segment only those message buffers are considered, that match the condition of at least one row of *Table 823*. These message buffers are called *matching* message buffers.

For each enabled channel the message buffer search may identify multiple *matching* message buffers. Among all matching message buffers the message buffers with highest priority according to *Table 822* for the static segment and according to *Table 823* for the dynamic segment are selected.

Table 822. Message buffer search priority (static segment)

Priority	MTD	LCKS	CMT	CCFM (1)	Description	Transition
(highest) 0	1	0	1	1	Transmit buffer, matches cycle count, not locked and committed	MA
1	1	—	0	1	Transmit buffer, matches cycle count, not committed	SA
	1	1	—	1	Transmit buffer, matches cycle count, locked	SA
2	1	—	—	—	Transmit buffer	SA
3	0	0	n/a	1	Receive buffer, matches cycle count, not locked	SB
(lowest) 4	0	1	n/a	1	Receive buffer, matches cycle count, locked	SB

1. Cycle Counter Filter Match, see *Section , Message buffer cycle counter filtering*

Table 823. Message buffer search priority (dynamic segment)

Priority	MTD	LCKS	CMT	CCFM (1)	Description	Transition
(highest) 0	1	0	1	1	Transmit buffer, matches cycle count, not locked and committed	MA
1	0	0	n/a	1	Receive buffer, matches cycle count, not locked	SB
(lowest) 2	0	1	n/a	1	Receive buffer, matches cycle count, locked	SB

1. Cycle Counter Filter Match, see *Section , Message buffer cycle counter filtering*

If there are multiple message buffer with highest priority, the message buffer with the lowest message buffer number is selected. All message buffer which have the highest priority must have a consistent channel assignment as specified in *Section , Message buffer channel assignment consistency*.

Depending on the message buffer channel assignment the same message buffer can be found for both channel A and channel B. In this case, this message buffer is used as described in *Section , Individual message buffers*.

Message buffer cycle counter filtering

The message buffer cycle counter filter is a value-mask filter defined by the CCFE, CCFMSK, and CCFVAL fields in the *Message Buffer Cycle Counter Filter Registers (FR_MBCCFRn)*. This filter determines a set of communication cycles in which the message buffer is considered for message reception or message transmission. If the cycle counter filter is disabled, that is, CCFE = 0, this set of cycles consists of all communication cycles.

If the cycle counter filter of a message buffer does not match a certain communication cycle number, this message buffer is not considered for message transmission or reception in that communication cycle. In case of a transmit message buffer assigned to a slot in the static segment, though, this buffer is added to the matching message buffers to indicate the slot assignment and to trigger the null frame transmission.

The cycle counter filter of a message buffer matches the communication cycle with the number CYCCNT if at least one of the following conditions evaluates to true:

Equation 54

$$\text{MBCCFRn[CCFE]} = 0$$

Equation 55

$$\text{CYCCNT \& MBCCFRn[CCFMSK]} = \text{MBCCFRn[CCFVAL] \& MBCCFRn[CCFMSK]}$$

Message buffer channel assignment consistency

The message buffer channel assignment given by the CHA and CHB bits in the *Message Buffer Cycle Counter Filter Registers (FR_MBCCFRn)* defines the channels on which the message buffer will receive or transmit. The message buffer with number *n* transmits or receives on channel A if FR_MBCCFRn[CHA] = 1 and transmits or receives on channel B if FR_MBCCFRn[CHB] = 1.

To ensure correct message buffer operation, all message buffers assigned to the same slot and with the same priority must have a **consistent** channel assignment. That means they must be either assigned to one channel only, or must be assigned to **both** channels. The behavior of the message buffer search is not defined, if both types of channel assignments occur for one slot and priority. An inconsistent channel assignment for message buffer 0 and message buffer 1 is depicted in *Figure 849*.

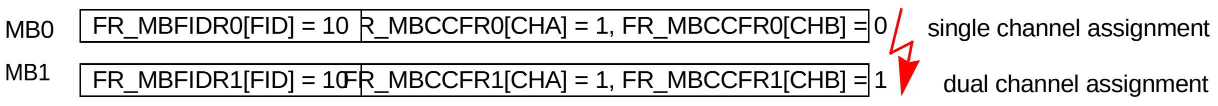


Figure 849. Inconsistent channel assignment

Node related slot multiplexing

The term *Node Related Slot Multiplexing* applies to the dynamic segment only and refers to the functionality if there are transmit as well as receive message buffers are configured for the same slot.

According to *Table 823* the transmit buffer is only found if the cycle counter filter matches, and the buffer is not locked and committed. In all other cases, the receive buffer will be found. Thus, if the block has no data to transmit in a dynamic slot, it is able to receive frames on that slot.

Message buffer search error

If the message buffer search is running while the next message buffer search start event appears, the message buffer search is stopped and the Message Buffer Search Error Flag MSB_EF is set in the *CHI Error Flag Register (FR_CHIERFR)*. This appears only if the CHI frequency is too low to search through all message buffers within the NIT or a minislot. The message buffer result is not defined in this case. For more details see *Section 31.7.6, Number of usable message buffers*.

31.6.8 Individual message buffer reconfiguration

The initial configuration of each individual message buffer can be changed even when the protocol is not in the *POC:config* state. This is referred to as individual message buffer *reconfiguration*. The configuration bits and fields that can be changed are given in the section on *Specific configuration data*. The common configuration data given in the section on *Specific configuration data* can not be reconfigured when the protocol is out of the *POC:config* state.

Reconfiguration schemes

Depending on the target and destination basic state of the message buffer that is to be reconfigured, there are three reconfiguration schemes.

Basic type not changed (RC1)

A reconfiguration will not change the basic type of the individual message buffer, if both the message buffer transfer direction bit FR_MBCCSRn[MTD] and the message buffer type bit FR_MBCCSRn[MBT] are not changed. This type of reconfiguration is denoted by RC1 in *Figure 850*. Single transmit and receive message buffers can be RC1-reconfigured when in the HDis or HDisLck state. Double transmit message buffers can be RC1-reconfigured if both the transmit side and the commit side are in the HDis state.

Buffer type not changed (RC2)

A reconfiguration will not change the buffer type of the individual message buffer if the message buffer buffer type bit FR_MBCCSRn[MBT] is not changed. This type of reconfiguration is denoted by RC2 in *Figure 850*. It applies only to single transmit and receive message buffers. Single transmit and receive message buffers can be RC2-reconfigured when in the HDis or HDisLck state.

Buffer type changed (RC3)

A reconfiguration will change the buffer type of the individual message buffer if the message buffer type bit FR_MBCCSRn[MBT] is changed. This type of reconfiguration is denoted by RC3 in *Figure 850*. The RC3 reconfiguration splits one double buffer into two single buffers or combines two single buffer into one double buffer. In the later case, the two single message buffers must have consecutive message buffer numbers and the smaller one must be even. Message Buffers can be RC3 reconfigured if they are in the HDis state.

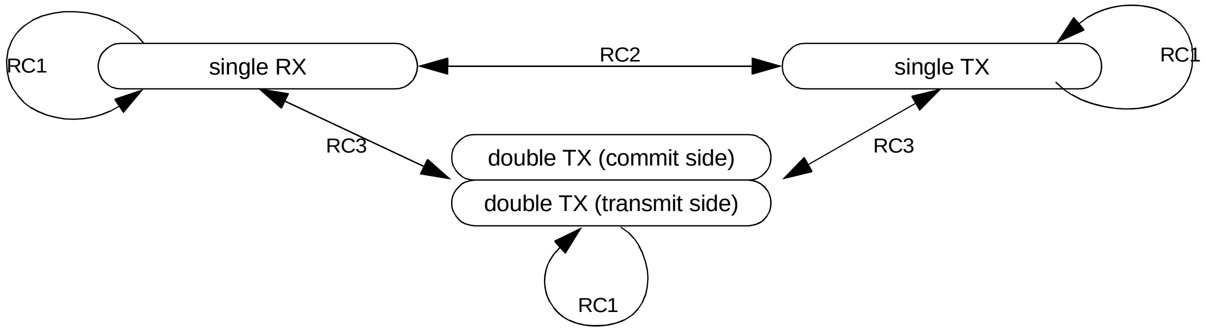


Figure 850. Message buffer reconfiguration scheme

31.6.9 Receive FIFOs

This section provides the functional description of the two receive FIFOs.

Overview

The two receive FIFOs implement the queued message buffer concept defined by the *FlexRay Communications System Protocol Specification, Version 2.1 Rev A.* One FIFO is assigned to channel A, the other FIFO is assigned to channel B. Both FIFOs work completely independent from each other.

The message buffer structure of each FIFO is described in *Section , Receive FIFO.* The area in the FlexRay memory area for each of the two FIFOs is characterized by:

- The FIFO system memory base address
- The index of the first FIFO entry given by *Receive FIFO Start Index Register (FR_RFSIR)*
- The number of FIFO entries and the length of each FIFO entry as given by *Receive FIFO Depth and Size Register (RFDSR)*

FIFO configuration

The FIFOs can be configured for two different locations of the system memory base address via the FIFO address mode bit FAM in the *Module Configuration Register (FR_MCR)*.

Single system memory base address mode

This mode is configured, when the FIFO address mode flag FR_MCR[FAM] is set to 0. In this mode, the location of the system memory base address for the FIFO buffers is *System Memory Base Address Register (FR_SYMBADR)*.

Dual system memory base address mode

This mode is configured, when the FIFO address mode flag FR_MCR[FAM] is set to 1. In this mode, the location of the system memory base address for the FIFO buffers is *Receive FIFO System Memory Base Address Register (FR_RFSYMBADR)*.

The FIFO control and configuration data are given in *Section , Receive FIFO control and configuration data.* The configuration of the FIFOs consists of two steps.

1. The first step is the allocation of the required amount of memory for the FlexRay memory area. This includes the allocation of the message buffer header area and the

allocation of the message buffer data fields. For more details see *Section 31.6.4, FlexRay memory area layout*.

2. The second step is the programming of the configuration data register while the PE is in *POC:config*.

The following steps configure the layout of the FIFO:

- Configure the FIFO update and address modes in *Module Configuration Register (FR_MCR)*
- Configure the FIFO system memory base address
- Configure the *Receive FIFO Start Index Register (FR_RFSIR)* with the first message buffer header index that belongs to the FIFO
- Configure the *Receive FIFO Depth and Size Register (RFDSR)* with FIFO entry size
- Configure the *Receive FIFO Depth and Size Register (RFDSR)* with FIFO depth
- Configure the FIFO filters

FIFO periodic timer

The FIFO periodic timer is used to generate an FIFO almost-full interrupt at certain point in time, if the almost-full watermark is not reached, but the FIFO is not empty. This can be used to prevent frames from get stuck in the FIFO for a long time.

The FIFO periodic timer is configured via the *Receive FIFO Periodic Timer Register (FR_RFPTR)*. If the periodic timer duration FR_RFPTR[PTD] is configured to 0x0000, the periodic timer is continuously expired. If the periodic timer duration FR_RFPTR[PTD] is configured to 0x3FFF, the periodic timer never expires. If the periodic timer is configured to a value *ptd,* greater than 0x0000 and smaller 0x3FFF, the periodic timer expires and is restarted at the start of every communication cycle, and expires and is restarted after *ptd* macroticks have been elapsed.

FIFO reception

The FIFO reception is a CC internal operation.

A message frame reception is directed into the FIFO, if no individual message buffer is assigned for transmission or subscribed for reception for the current slot. In this case the FIFO filter path shown in *Figure 851* is activated.

If the FIFO filter path indicates that the received frame has to be appended to the FIFO and the FIFO is not full, the CC writes the received frame header into the message buffer header field indicated by the CC internal FIFO write index. The frame payload data are written into the corresponding message buffer data field. If the status of the received frame indicates a valid non-null frame, the slot status information is written into the message buffer header field and the CC internal FIFO write index is updated by 1 and the FIFO fill level FLA (FLB) in the *Receive FIFO Fill Level and POP Count Register (FR_RFFLPCR)* is incremented.If the status of the received frame indicates an invalid or null frame, the frame is not appended to the FIFO.

FIFO almost-full interrupt generation

If the FIFO fill level FLA (FLB) is updated after a frame reception and exceeds the FIFO watermark level WM, that is, $FLA > WM_A$ ($FLB > WM_B$), then the FIFO almost-full interrupt flag FR_GIFER[FAFAIF] (FR_GIFER[FAFBIF]) is asserted.If the periodic timer expires, and FIFOA (FIFOB) is not empty, that is, $FLA > 0$ ($FLB > 0$), then the FIFO almost-full interrupt flag FR_GIFER[FAFAIF] (FR_GIFER[FAFBIF]) is asserted.

FIFO overflow error generation

If the FIFOA (FIFOB) is full, that is, FLA = $FIFO\_DEPTH_A$ (FLB = $FIFO\_DEPTH_B$) and the conditions for a FIFO reception as described in *Section , FIFO reception* are fulfilled, then the FIFO overflow error flag FR_CHIERFR[FOVA_EF] (FR_CHIERFR[FOVB_EF]) is asserted.

FIFO message access

The FIFOA (FIFOB) contains valid messages if the FIFO fill level FLA (FLB) is greater than 0. The *Receive FIFO A Read Index Register (FR_RFARIR)* (*Receive FIFO B Read Index Register (FR_RFBRIR)*) pointing to a message buffer with valid content and the oldest frames stored in the FIFO.

If the FIFO fill level FLA (FLB) is 0, than the FIFOA (FIFOB) contains no valid messages and the *Receive FIFO A Read Index Register (FR_RFARIR)* (*Receive FIFO B Read Index Register (FR_RFBRIR)*) pointing to a message buffer with invalid content. In this case the application must not read data from the FIFO.

To access the oldest message in the FIFOA (FIFOB), the application first reads the read index RDIDX out of the *Receive FIFO A Read Index Register (FR_RFARIR)* (*Receive FIFO B Read Index Register (FR_RFBRIR)*). This read index points to the message buffer header field of the oldest message buffer that contains valid received message data. The application can access the message data as described in *Section , Receive FIFO.* When the application has read the message buffer data and status information, it can update the FIFO as described in *Section , FIFO update*.

FIFO update

The application updates the FIFOA (FIFOB) by writing a pop count value *pc* different from 0 to the PCA (PCB) field in the *Receive FIFO Fill Level and POP Count Register (FR_RFFLPCR)*.

As a result of the this operation, the CC removes the oldest *pc* entries from FIFOA (FIFOB).

If the specified pop count value *pc* is greater than the current fill level *fl* provided in FLA (FAB) field, then only *fl* entries are removed from the FIFOA (FIFOB), the remaining *fl-pc* requested pop operations are discarded without any notification. In this case FIFOA (FIFOB) is empty after the update operation.

The read index in the *Receive FIFO A Read Index Register (FR_RFARIR)* (*Receive FIFO B Read Index Register (FR_RFBRIR)*) is incremented by the number of removed items. If the read index reaches the top of the FIFO, it wraps around to the FIFO start index defined in *Receive FIFO Start Index Register (FR_RFSIR)* automatically.

FIFO interrupt flag update

Th FIFO Interrupt Flag Update mode is configured when the FIFO update mode flag FR_MCR[FUM] is set to ‘0’. In this mode FIFOA (FIFOB) will be updated by one entry, when the interrupt flag FR_GIFER[FAFAIF] (FR_GIFER[FAFBIF]) is written with ‘1’ by the application.

If the FIFO is empty, the update request is ignored without any notification.

The read index in the *Receive FIFO A Read Index Register (FR_RFARIR)* (*Receive FIFO B Read Index Register (FR_RFBRIR)*) is incremented by 1 if the FIFO was not empty. If the read index reaches the top of the FIFO, it wraps around to the FIFO start index automatically.

FIFO filtering

The FIFO filtering is activated after all enabled individual receive message buffers have been searched without success for a message buffer to receive the current frame.

The CC provides three sets of FIFO filters. The FIFO filters are applied to valid non-null frames only. The FIFO will not receive invalid or null-frames. For each FIFO filter, the pass criteria is specified in the related section given below. Only frames that have passed all filters will be appended to the FIFO. The FIFO filter path is depicted in *Figure 851*.

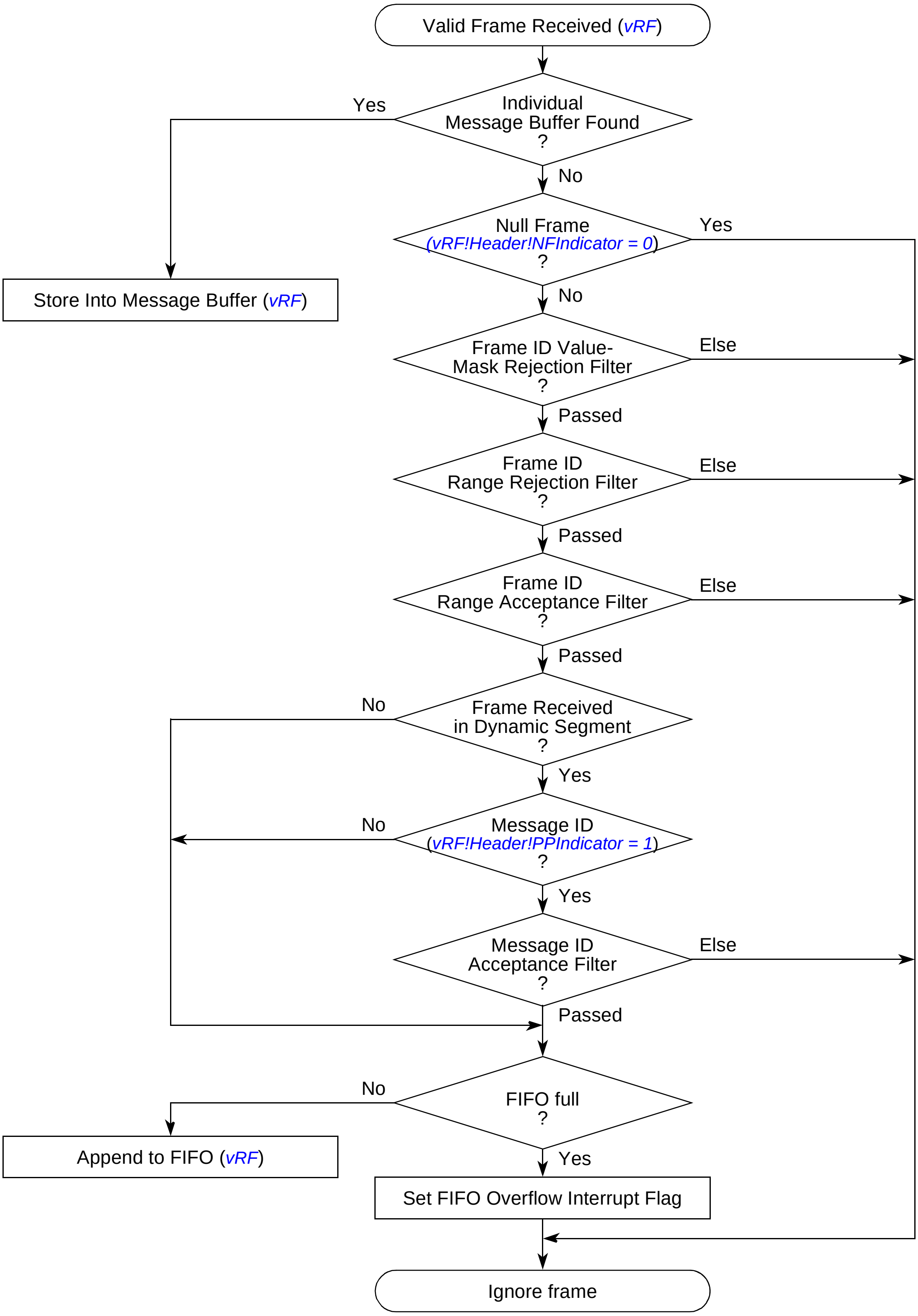


Figure 851. Received frame FIFO filter path

A received frame passes the FIFO filtering if it has passed all three type of filter.

RX FIFO frame ID value-mask rejection filter

The frame ID value-mask rejection filter is a value-mask filter and is defined by the fields in the *Receive FIFO Frame ID Rejection Filter Value Register (FR_RFFIDRFVR)* and the *Receive FIFO Frame ID Rejection Filter Mask Register (FR_RFFIDRFMR)*. Each received frame with a frame ID FID that does not match the value-mask filter value passes the filter, that is, is not rejected.

Consequently, a received valid frame with the frame ID FID passes the RX FIFO Frame ID Value-Mask Rejection Filter if *Equation 56* is fulfilled.

Equation 56

$$\text{FID} \mathbin{\&} \text{FR\_RFFIDRFMR}[\text{FIDRFMSK}] \neq \text{FR\_RFFIDRFVR}[\text{FIDRFVAL}] \mathbin{\&} \text{FR\_RFFIDRFMR}[\text{FIDRFMSK}]$$

The RX FIFO Frame ID Value-Mask Rejection Filter can be configured to pass all frames by the following settings.

- FR_RFFIDRFVR[FIDRFVAL] = 0x000 and FR_RFFIDRFMR[FIDRFMSK] = 0x7FF

Using the settings above, only the frame with frame ID 0 will be rejected, which is an invalid frame. All other frames will pass.

The RX FIFO Frame ID Value-Mask Rejection Filter can be configured to reject all frames by the following settings.

- FR_RFFIDRFMR[FIDRFMSK] = 0x000

Using the settings above, *Equation 56* can never be fulfilled (0 != 0) and thus all frames are rejected; no frame will pass. This is the reset value for the RX FIFO.

RX FIFO frame ID range rejection filter

Each of the four RX FIFO Frame ID Range filters can be configured as a rejection filter. The filters are configured by the *Receive FIFO Range Filter Configuration Register (FR_RFRFCFR)* and controlled by the *Receive FIFO Range Filter Control Register (FR_RFRFCTR)*. The RX FIFO Frame ID range filters apply to all received valid frames. A received frame with the frame ID FID passes the RX FIFO Frame ID Range rejection filters if either no rejection filter is enabled, or, for all of the enabled RX FIFO Frame ID Range rejection filters, that is, FR_RFRFCTR[FiMD] = 1 and FR_RFRFCTR[FiEN] = 1, *Equation 57* is fulfilled.

Equation 57

$$(\text{FID} < \text{FR\_RFRFCFR}_{\text{SEL}}[\text{SID}_{\text{IBD}\,=\,0}]) \text{ or } (\text{FR\_RFRFCFR}_{\text{SEL}}[\text{SID}_{\text{IBD}\,=\,1}] < \text{FID})$$

Consequently, all frames with a frame ID that fulfills *Equation 58* for at least one of the enabled rejection filters will be rejected and thus not pass.

Equation 58

$$\text{FR\_RFRFCFR}_{\text{SEL}}[\text{SID}_{\text{IBD}\,=\,0}] \leq \text{FID} \leq \text{FR\_RFRFCFR}_{\text{SEL}}[\text{SID}_{\text{IBD}\,=\,1}]$$

RX FIFO frame ID range acceptance filter

Each of the four RX FIFO Frame ID Range filters can be configured as an acceptance filter. The filters are configured by the *Receive FIFO Range Filter Configuration Register (FR_RFRFCFR)* and controlled by the *Receive FIFO Range Filter Control Register (FR_RFRFCTR)*. The RX FIFO Frame ID range filters apply to all received valid frames. A received frame with the frame ID FID passes the RX FIFO Frame ID Range acceptance filters if either no acceptance filter is enabled, or, for at least one of the enabled RX FIFO Frame ID Range acceptance filters, that is, FR_RFRFCTR[FiMD] = 0 and FR_RFRFCTR[FiEN] = 1, *Equation 59* is fulfilled.

Equation 59

$$\mathrm{FR\_RFRFCFR}_{\mathrm{SEL}}[\mathrm{SID}_{\mathrm{IBD}\,=\,0}] \le \mathrm{FID} \le \mathrm{FR\_RFRFCFR}_{\mathrm{SEL}}[\mathrm{SID}_{\mathrm{IBD}\,=\,1}]$$

RX FIFO message ID acceptance filter

The RX FIFO Message ID Acceptance Filter is a value-mask filter and is defined by the *Receive FIFO Message ID Acceptance Filter Value Register (FR_RFMIDAFVR)* and the *Receive FIFO Message ID Acceptance Filter Mask Register (FR_RFMIDAFMR)*. This filter applies only to valid frames received in the dynamic segment with the payload preamble indicator bit PPI set to 1. All other frames will pass this filter.

A received valid frame in the dynamic segment with the payload preamble indicator bit PPI set to 1 and with the message ID MID (the first two bytes of the payload) will pass the RX FIFO Message ID Acceptance Filter if *Equation 60* is fulfilled.

Equation 60

$$\mathrm{MID}\ \&\ \mathrm{FR\_RFMIDAFMR}[\mathrm{MIDAFMSK}] = \mathrm{FR\_RFMIDAFMR}[\mathrm{MIDAFVAL}]\ \&\ \mathrm{FR\_RFMIDAFMR}[\mathrm{MIDAFMSK}]$$

The RX FIFO Message ID Acceptance Filter can be configured to accept all frames by setting

- FR_RFMIDAFMR[MIDAFMSK] = 0x000

Using the settings above, *Equation 60* is always fulfilled and all frames will pass.

31.6.10 Channel device modes

This section describes the two FlexRay channel device modes that are supported by the CC.

Dual channel device mode

In the dual channel device mode, both FlexRay ports are connected to physical FlexRay bus lines. The FlexRay port consisting of FR_A_RX, FR_A_TX, and FR_A_TX_EN is connected to the physical bus channel A and the FlexRay port consisting of FR_B_RX, FR_B_TX, and FR_B_TX_EN is connected to the physical bus channel B. The dual channel system is shown in *Figure 852*.

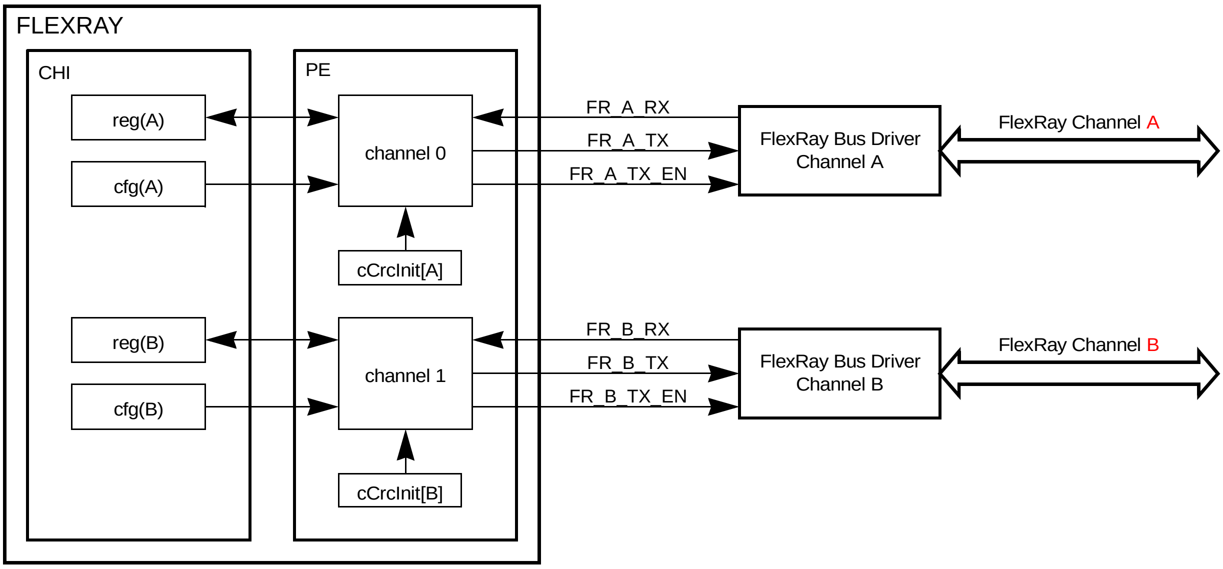


Figure 852. Dual channel device mode

Single channel device mode

The single channel device mode supports devices that have only one FlexRay port available. This FlexRay port consists of the signals FR_A_RX, FR_A_TX, and FR_A_TX_EN and can be connected to either the physical bus channel A (shown in *Figure 853*) or the physical bus channel B (shown in *Figure 854*).

If the device is configured as a single channel device by setting FR_MCR.SCD to 1, only the internal channel A and the FlexRay Port A is used. Depending on the setting of FR_MCR.CHA and FR_MCR.CHB, the internal channel A behaves either as a FlexRay Channel A or FlexRay Channel B. The bit FR_MCR.CHA must be set, if the FlexRay Port A is connected to a FlexRay Channel A. The bit FR_MCR.CHB must be set if the FlexRay Port A is connected to a FlexRay Channel B. The two FlexRay channels differ only in the initial value for the frame CRC *cCrcInit*. For a single channel device, the application can access and configure only the registers related to internal channel A.

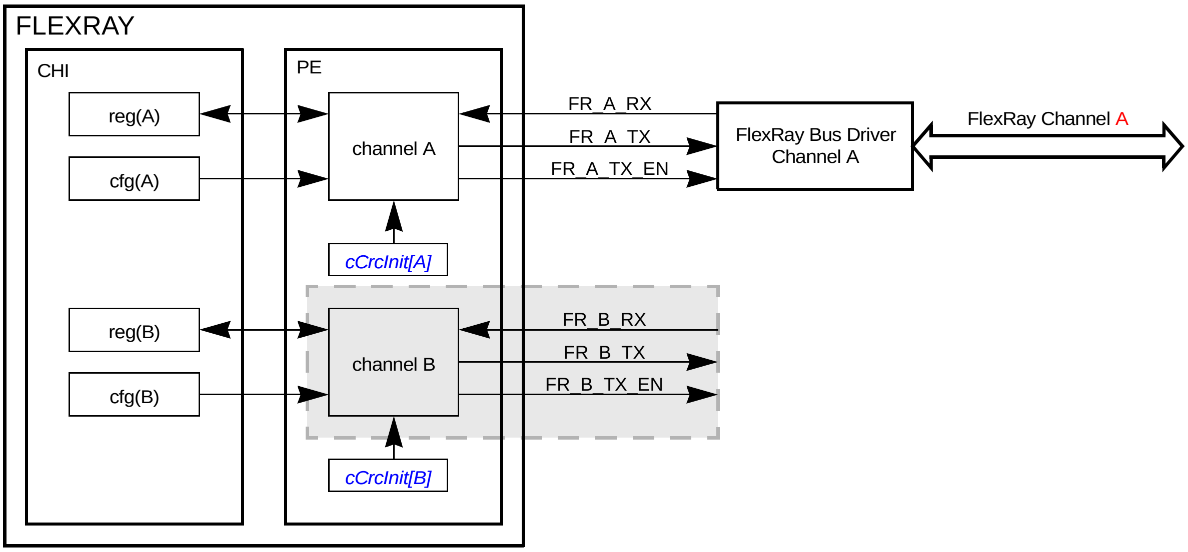


Figure 853. Single channel device mode (channel A)

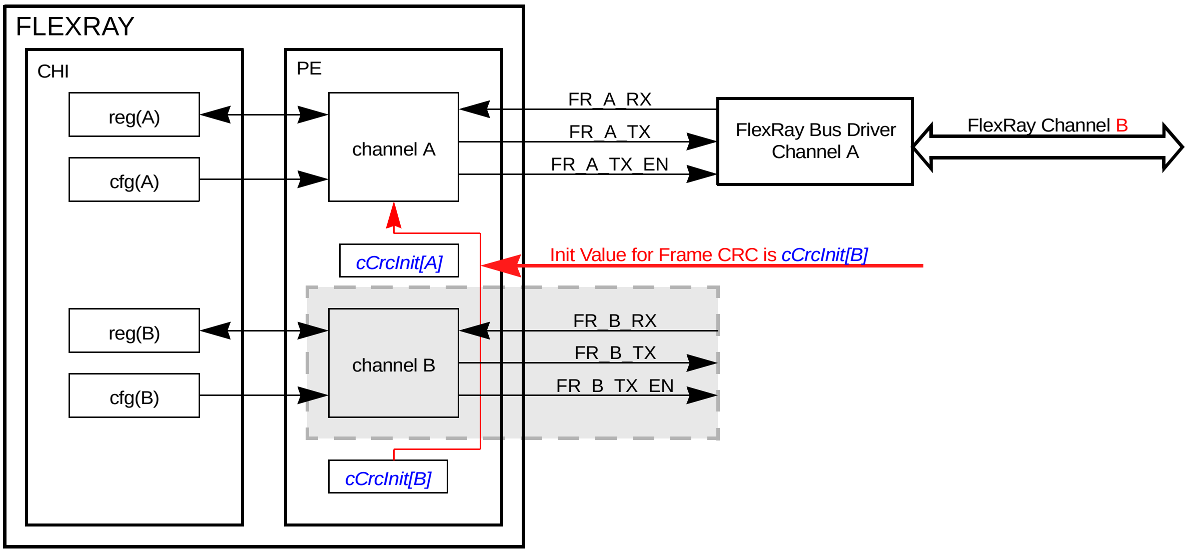


Figure 854. Single channel device mode (channel B)

31.6.11 External clock synchronization

The application of the external rate and offset correction is triggered when the application writes to the EOC_AP and ERC_AP fields in the *Protocol Operation Control Register (FR_POCR)*. The PE applies the external correction values in the next even-odd cycle pair as shown in *Figure 855* and *Figure 856*.

Note: The values provided in the EOC_AP and ERC_AP fields are the values that were written from the application most recently. If these value were already applied, they will not be applied in the current cycle pair again.

If the offset correction applied in the NIT of cycle 2n+1 shall be affect by the external offset correction, the EOC_AP field must be written to after the start of cycle 2n and before the end of the static segment of cycle 2n+1. If this field is written to after the end of the static segment of cycle 2n+1, it is not guaranteed that the external correction value is applied in cycle 2n+1. If the value is not applied in cycle 2n+1, then the value will be applied in the cycle 2n+3. Refer to *Figure 855* for timing details.

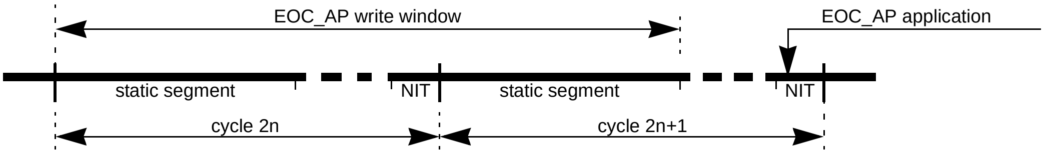


Figure 855. External offset correction write and application timing

If the rate correction for the cycle pair [2n+2, 2n+3] shall be affect by the external offset correction, the ERC_AP field must be written to after the start of cycle 2n and before the end of the static segment start of cycle 2n+1. If this field is written to after the end of the static segment of cycle 2n+1, it is not guaranteed that the external correction value is applied in cycle pair [2n+2, 2n+3]. If the value is not applied for cycle pair [2n+2, 2n+3], then the value will be applied for cycle pair [2n+4, 2n+5]. Refer to *Figure 856* for details.

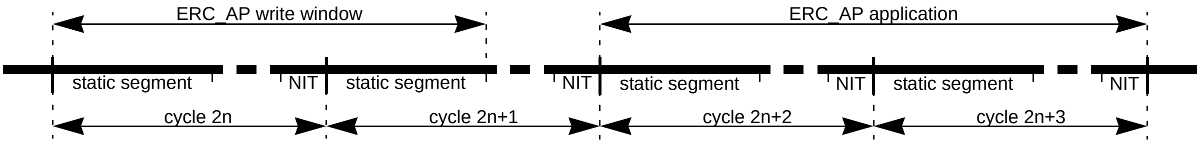


Figure 856. External rate correction write and application timing

31.6.12 Sync frame ID and sync frame deviation tables

The FlexRay protocol requires the provision of a snapshot of the Synchronization Frame ID tables for the even and odd communication cycle for both channels. The CC provides the means to write a copy of these internal tables into the FlexRay memory area and ensures application access to consistent tables by means of table locking. Once the application has locked the table successfully, the CC will not overwrite these tables and the application can read a consistent snapshot.

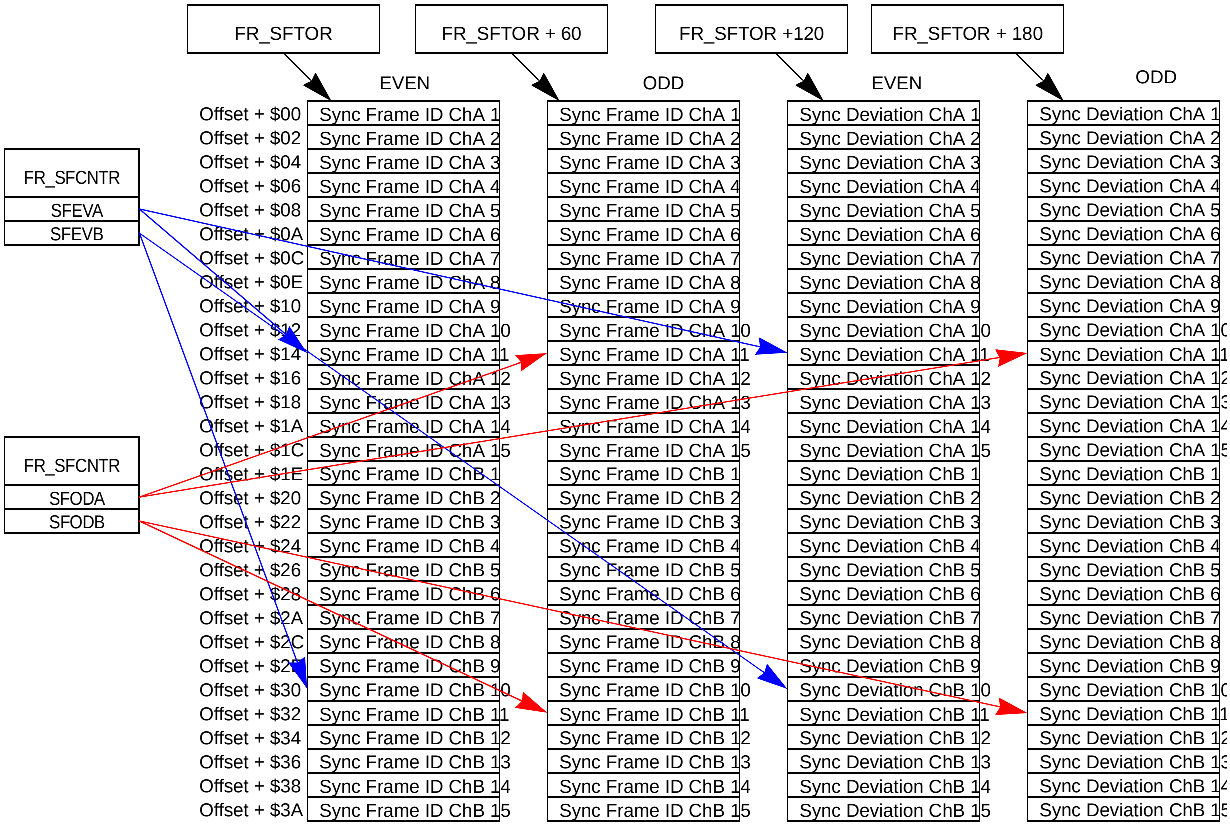
Note: Only synchronization frames that have passed the synchronization frame filters are considered for clock synchronization and appear in the sync frame tables.

Sync frame ID table content

The Sync Frame ID Table is a snapshot of the protocol related variables *vsSyncIdListA* and *vsSyncIdListB* for each even and odd communication cycle. This table provides a list of the frame IDs of the synchronization frames received on the corresponding channel and cycle that are used for the clock synchronization.

Sync frame deviation table content

The Sync Frame Deviation Table is a snapshot of the protocol related variable zsDev(id)(oe)(ch)!Value. Each Sync Frame Deviation Table entry provides the deviation value for the sync frame, with the frame ID presented in the corresponding entry in the Sync Frame ID Table.



	FR_SFTOR EVEN	FR_SFTOR + 60 ODD	FR_SFTOR +120 EVEN	FR_SFTOR + 180 ODD
Offset + $00	Sync Frame ID ChA 1	Sync Frame ID ChA 1	Sync Deviation ChA 1	Sync Deviation ChA 1
Offset + $02	Sync Frame ID ChA 2	Sync Frame ID ChA 2	Sync Deviation ChA 2	Sync Deviation ChA 2
Offset + $04	Sync Frame ID ChA 3	Sync Frame ID ChA 3	Sync Deviation ChA 3	Sync Deviation ChA 3
Offset + $06	Sync Frame ID ChA 4	Sync Frame ID ChA 4	Sync Deviation ChA 4	Sync Deviation ChA 4
Offset + $08	Sync Frame ID ChA 5	Sync Frame ID ChA 5	Sync Deviation ChA 5	Sync Deviation ChA 5
Offset + $0A	Sync Frame ID ChA 6	Sync Frame ID ChA 6	Sync Deviation ChA 6	Sync Deviation ChA 6
Offset + $0C	Sync Frame ID ChA 7	Sync Frame ID ChA 7	Sync Deviation ChA 7	Sync Deviation ChA 7
Offset + $0E	Sync Frame ID ChA 8	Sync Frame ID ChA 8	Sync Deviation ChA 8	Sync Deviation ChA 8
Offset + $10	Sync Frame ID ChA 9	Sync Frame ID ChA 9	Sync Deviation ChA 9	Sync Deviation ChA 9
Offset + $12	Sync Frame ID ChA 10	Sync Frame ID ChA 10	Sync Deviation ChA 10	Sync Deviation ChA 1
Offset + $14	Sync Frame ID ChA 11	Sync Frame ID ChA 11	Sync Deviation ChA 11	Sync Deviation ChA 11
Offset + $16	Sync Frame ID ChA 12	Sync Frame ID ChA 12	Sync Deviation ChA 12	Sync Deviation ChA 1
Offset + $18	Sync Frame ID ChA 13	Sync Frame ID ChA 13	Sync Deviation ChA 13	Sync Deviation ChA 1
Offset + $1A	Sync Frame ID ChA 14	Sync Frame ID ChA 14	Sync Deviation ChA 14	Sync Deviation ChA 1
Offset + $1C	Sync Frame ID ChA 15	Sync Frame ID ChA 15	Sync Deviation ChA 15	Sync Deviation ChA 1
Offset + $1E	Sync Frame ID ChB 1	Sync Frame ID ChB 1	Sync Deviation ChB 1	Sync Deviation ChB 1
Offset + $20	Sync Frame ID ChB 2	Sync Frame ID ChB 2	Sync Deviation ChB 2	Sync Deviation ChB 2
Offset + $22	Sync Frame ID ChB 3	Sync Frame ID ChB 3	Sync Deviation ChB 3	Sync Deviation ChB 3
Offset + $24	Sync Frame ID ChB 4	Sync Frame ID ChB 4	Sync Deviation ChB 4	Sync Deviation ChB 4
Offset + $26	Sync Frame ID ChB 5	Sync Frame ID ChB 5	Sync Deviation ChB 5	Sync Deviation ChB 5
Offset + $28	Sync Frame ID ChB 6	Sync Frame ID ChB 6	Sync Deviation ChB 6	Sync Deviation ChB 6
Offset + $2A	Sync Frame ID ChB 7	Sync Frame ID ChB 7	Sync Deviation ChB 7	Sync Deviation ChB 7
Offset + $2C	Sync Frame ID ChB 8	Sync Frame ID ChB 8	Sync Deviation ChB 8	Sync Deviation ChB 8
Offset + $2E	Sync Frame ID ChB 9	Sync Frame ID ChB 9	Sync Deviation ChB 9	Sync Deviation ChB 9
Offset + $30	Sync Frame ID ChB 10	Sync Frame ID ChB 10	Sync Deviation ChB 10	Sync Deviation ChB 1
Offset + $32	Sync Frame ID ChB 11	Sync Frame ID ChB 11	Sync Deviation ChB 11	Sync Deviation ChB 11
Offset + $34	Sync Frame ID ChB 12	Sync Frame ID ChB 12	Sync Deviation ChB 12	Sync Deviation ChB 1
Offset + $36	Sync Frame ID ChB 13	Sync Frame ID ChB 13	Sync Deviation ChB 13	Sync Deviation ChB 1
Offset + $38	Sync Frame ID ChB 14	Sync Frame ID ChB 14	Sync Deviation ChB 14	Sync Deviation ChB 1
Offset + $3A	Sync Frame ID ChB 15	Sync Frame ID ChB 15	Sync Deviation ChB 15	Sync Deviation ChB 1

Figure 857. Sync table memory layout

Sync frame ID and sync frame deviation table setup

The CC writes a copy of the internal synchronization frame ID and deviation tables into the FlexRay memory area if requested by the application. The application must provide the appropriate amount of FlexRay memory area for the tables. The memory layout of the tables is given in *Figure 857*. Each table occupies 120 16-bit entries.

While the protocol is in *POC:config* state, the application must program the offsets for the tables into the *Sync Frame Table Offset Register (FR_SFTOR)*.

Sync frame ID and sync frame deviation table generation

The application controls the generation process of the Sync Frame ID and Sync Frame Deviation Tables into the FlexRay memory area using the *Sync Frame Table Configuration, Control, Status Register (FR_SFTCCSR)*. A summary of the copy modes is given in *Table 824*.

Table 824. Sync frame table generation modes

FR_SFTCCSR			Description
OPT	SDVEN	SIDEN	
0	0	0	No Sync Frame Table copy
0	0	1	Sync Frame ID Tables will be copied continuously
0	1	0	Reserved
0	1	1	Sync Frame ID Tables and Sync Frame Deviation Tables will be copied continuously
1	0	0	No Sync Frame Table copy
1	0	1	Sync Frame ID Tables for next even-odd-cycle pair will be copied
1	1	0	Reserved
1	1	1	Sync Frame ID Tables and Sync Frame Deviation Tables for next even-odd-cycle pair will be copied

The Sync Frame Table generation process is described in the following for the even cycle. The same sequence applies to the odd cycle.

If the application has enabled the sync frame table generation by setting FR_SFTCCSR[SIDEN] to 1, the CC starts the update of the even cycle related tables after the start of the NIT of the next even cycle. The CC checks if the application has locked the tables by reading the FR_SFTCCSR[ELKS] lock status bit. If this bit is set, the CC will not update the table in this cycle. If this bit is cleared, the CC locks this table and starts the table update. To indicate that these tables are currently updated and may contain inconsistent data, the CC clears the even table valid status bit FR_SFTCCSR[EVAL]. Once all table entries related to the even cycle have been transferred into the FlexRay memory area, the CC sets the even table valid bit FR_SFTCCSR[EVAL] and the Even Cycle Table Written Interrupt Flag EVT_IF in the *Protocol Interrupt Flag Register 1 (FR_PIFR1)*. If the interrupt enable flag EVT_IE is set, an interrupt request is generated.

To read the generated tables, the application must lock the tables to prevent the CC from updating these tables. The locking is initiated by writing a 1 to the even table lock trigger FR_SFTCCSR[ELKT]. When the even table is not currently updated by the CC, the lock is granted and the even table lock status bit FR_SFTCCSR[ELKS] is set. This indicates that the application has successfully locked the even sync tables and the corresponding status information fields SFRA, SFRB in the *Sync Frame Counter Register (FR_SFCNTR)*. The value in the FR_SFTCCSR[CYCNUM] field provides the number of the cycle that this table is related to.

The number of available table entries per channel is provided in the FR_SFCNTR[SFEVA] and FR_SFCNTR[SFEVB] fields. The application can now start to read the sync table data from the locations given in *Figure 857*.

After reading all the data from the locked tables, the application must unlock the table by writing to the even table lock trigger FR_SFTCCSR[ELKT] again. The even table lock status bit FR_SFTCCSR[ELKS] is reset immediately.

If the sync frame table generation is disabled, the table valid bits FR_SFTCCSR[EVAL] and FR_SFTCCSR[EVAL] are reset when the counter values in the *Sync Frame Counter Register (FR_SFCNTR)* are updated. This is done because the tables stored in the FlexRay memory area are no longer related to the values in the *Sync Frame Counter Register (FR_SFCNTR)*.

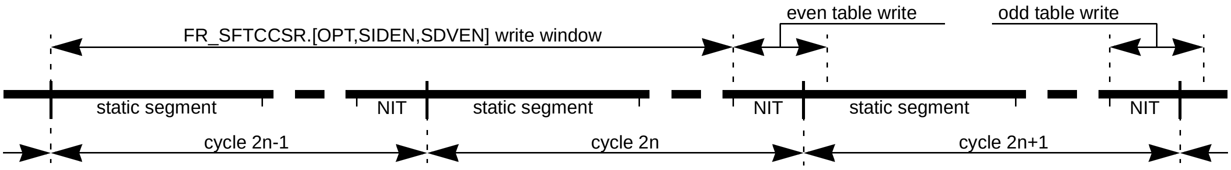


Figure 858. Sync frame table trigger and generation timing

Sync frame table access

The sync frame tables will be transferred into the FlexRay memory area during the table write windows shown in *Figure 858*. During the table write, the application can not lock the table that is currently written. If the application locks the table outside of the table write window, the lock is granted immediately.

Sync frame table locking and unlocking

The application locks the even/odd sync frame table by writing 1 to the lock trigger bit ELKT/OLKT in the *Sync Frame Table Configuration, Control, Status Register (FR_SFTCCSR)*. If the affected table is not currently written to the FlexRay memory area, the lock is granted immediately, and the lock status bit ELKS/OLKS is set. If the affected table is currently written to the FlexRay memory area, the lock is not granted. In this case, the application must issue the lock request again until the lock is granted.

The application unlocks the even/odd sync frame table by writing 1 to the lock trigger bit ELKT/OLKT. The lock status bit ELKS/OLKS is cleared immediately.

31.6.13 MTS generation

The CC provides a flexible means to request the transmission of the Media Access Test Symbol MTS in the symbol window on channel A or channel B.

The application can configure the set of communication cycles in which the MTS will be transmitted over the FlexRay bus by programming the CYCCNTMSK and CYCCNTVAL fields in the *MTS A Configuration Register (FR_MTSACFR)* and *MTS B Configuration Register (MTSBCFR)*.

The application enables or disables the generation of the MTS on either channel by setting or clearing the MTE control bit in the *MTS A Configuration Register (FR_MTSACFR)* or *MTS B Configuration Register (MTSBCFR)*. If an MTS is to be transmitted in a certain communication cycle, the application must set the MTE control bit during the static segment of the preceding communication cycle.

The MTS is transmitted over channel A in the communication cycle with number CYCCNT, if *Equation 62*, *Equation 63*, and *Equation 63* are fulfilled.

Equation 61 $\mathrm{FR\_PSR0[PROTSTATE]} = \mathit{POC{:}normal\ active}$

Equation 62 $\mathrm{FR\_MTSACRF[MTE]} = 1$

Equation 63

```
CYCCNT & FR_MTSACFR[CYCCNTMSK]
 = FR_MTSACFR[CYCCNTVAL] & FR_MTSACFR[CYCCNTMSK]
```

The MTS is transmitted over channel B in the communication cycle with number CYCCNT, if *Equation 61*, *Equation 64*, and *Equation 65* are fulfilled.

Equation 64 FR_MTSBCRF[MTE] = 1

Equation 65

```
CYCCNT & FR_MTSBCFR[CYCCNTMSK]
 = FR_MTSBCFR[CYCCNTVAL] & FR_MTSBCFR[CYCCNTMSK]
```

31.6.14 Key slot transmission

Key slot assignment

A key slot is assigned to the CC if the key_slot_id field in the *Protocol Configuration Register 18 (FR_PCR18)* is configured with a value greater than 0 and less or equal to number_of_static_slots in *Protocol Configuration Register 2 (FR_PCR2)*, otherwise no key slot is assigned.

Key slot transmission in *POC:startup*

If a key slot is assigned and the CC is in the *POC:startup* state, startup null frames will be transmitted as specified by *FlexRay Communications System Protocol Specification, Version 2.1 Rev A*.

Key slot transmission in *POC:normal active*

If a key slot is assigned and the CC is in *POC:normal active*, a frame of the type as shown in *Table 825* is transmitted. If a transmit message buffer is configured for the key slot and a valid message is available, a message frame is transmitted (see *Section , Message transmission*). If no transmit message buffer is configured for the key slot or no valid message is available, a null frame is transmitted (see *Section , Null frame transmission*).

Table 825. Key slot frame type

FR_PCR11[key_slot_used_for_sync]	FR_PCR11[key_slot_used_for_startup]	Key slot frame type
0	0	normal frame
0	1	normal frame[1]
1	0	sync frame
1	1	startup frame

1. The frame transmitted has an semantically incorrect header and will be detected as an invalid frame at the receiver.

31.6.15 Sync frame filtering

Each received synchronization frame must pass the Sync Frame Acceptance Filter and the Sync Frame Rejection Filter before it is considered for clock synchronization. If the synchronization frame filtering is globally disabled, that is, the SFFE control bit in the *Module Configuration Register (FR_MCR)* is cleared, all received synchronization frames are considered for clock synchronization. If a received synchronization frame did not pass at least one of the two filters, this frame is processed as a normal frame and is not considered for clock synchronization.

Sync frame acceptance filtering

The synchronization frame acceptance filter is implemented as a value-mask filter. The value is configured in the *Sync Frame ID Acceptance Filter Value Register (FR_SFIDAFVR)* and the mask is configured in the *Sync Frame ID Acceptance Filter Mask Register (FR_SFIDAFMR)*. A received synchronization frame with the frame ID FID passes the sync frame acceptance filter, if *Equation 66* or *Equation 67*evaluates to true.

Equation 66

$$\mathrm{FR\_MCR[SFFE]} = 0$$

Equation 67

$$\mathrm{FID}\ \&\ \mathrm{FR\_SFIDAFMR[FMSK]} = \mathrm{FR\_SFIDAFVR[FVAL]}\ \&\ \mathrm{FR\_SFIDAFMR[FMSK]}$$

Note: Sync frames are transmitted in the static segment only. Thus FID ≤ 1023.

Sync frame rejection filtering

The synchronization frame rejection filter is a comparator. The compare value is defined by the *Sync Frame ID Rejection Filter Register (FR_SFIDRFR)*. A received synchronization frame with the frame ID FID passes the sync frame rejection filter if *Equation 68* or *Equation 69* evaluates to true.

Equation 68

$$\mathrm{FR\_MCR[SFFE]} = 0$$

Equation 69

$$\mathrm{FID} \neq \mathrm{FR\_SFIDRFR[SYNFRID]}$$

Note: Sync frames are transmitted in the static segment only. Thus FID ≤ 1023.

31.6.16 Strobe signal support

The CC provides a number of strobe signals for observing internal protocol timing related signals in the protocol engine. The signals are listed and described in *Table 712*.

Strobe signal assignment

Each of the strobe signals listed in *Table 712* can be assigned to one of the four strobe ports using the *Strobe Signal Control Register (FR_STBSCR)*. To assign multiple strobe signals, the application must write multiple times to the *Strobe Signal Control Register (FR_STBSCR)* with appropriate settings.

To read out the current settings for a strobe signal with number N, the application must execute the following sequence.

1. Write to FR_STBSCR with WMD = 1 and SEL = N. (updates SEL field only)
2. Read STBCSR.
   The SEL field provides N and the ENB and STBPSEL fields provides the settings for signal N.

Strobe signal timing

This section provides detailed timing information of the strobe signals with respect to the protocol engine clock.

The strobe signals display internal PE signals. Due to the internal architecture of the PE, some signals are generated several PE clock cycles before the actual action is performed on the FlexRay Bus. These signals are listed in *Table 712* with a negative clock offset. An example waveform is given in *Figure 859*.

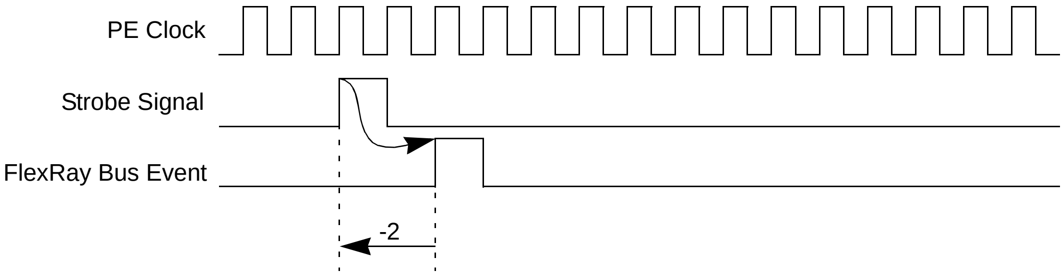


Figure 859. Strobe signal timing (type = pulse, clk_offset = -2)

Other signals refer to events that occurred on the FlexRay Bus some cycles before the strobe signal is changed. These signals are listed in *Table 712* with a positive clock offset. An example waveform is given in *Figure 860*.

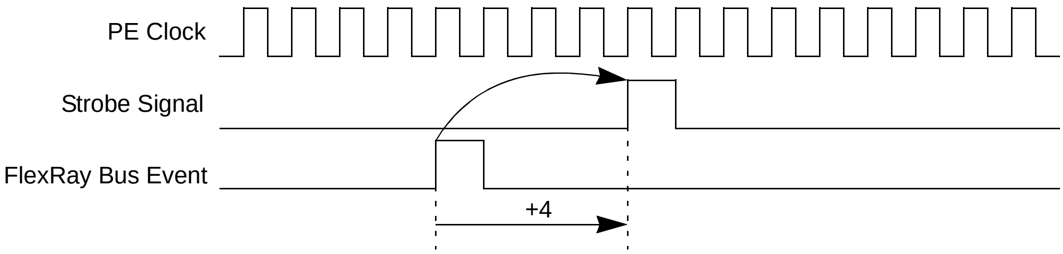


Figure 860. Strobe signal timing (type = pulse, clk_offset = +4)

31.6.17 Timer support

The CC provides two timers, which run on the FlexRay time base. Each timer generates a maskable interrupt when it reaches a configured point in time. Timer T1 is an absolute timer. Timer T2 can be configured to be an absolute or a relative timer. Both timers can be configured to be repetitive. In the non-repetitive mode, timer stops if it expires. In repetitive mode, timer is restarted when it expires.

Both timers are active only when the protocol is in *POC:normal active* or *POC:normal passive* state. If the protocol is not in one of these modes, the timers are stopped. The application must restart the timers when the protocol has reached the *POC:normal active* or *POC:normal passive* state.

Absolute timer T1

The absolute timer T1 has the protocol cycle count and the macrotick count as the time base. The timer 1 interrupt flag TI1_IF in the *Protocol Interrupt Flag Register 0 (FR_PIFR0)* is set at the macrotick start event, if *Equation* and *Equation 71* are fulfilled

Equation 70

$$\text{CYCTR[CTCCNT] \& FR\_TI1CYSR[T1\_CYC\_MSK]} \\ = \text{FR\_TI1CYSR[T1\_CYC\_VAL] \& FR\_TI1CYSR[T1\_CYC\_MSK]}$$

Equation 71

$$\text{FR\_MTCTR[MTCT]} = \text{FR\_TI1MTOR[T1\_MTOFFSET]}$$

If the timer 1 interrupt enable bit TI1_IE in the *Protocol Interrupt Enable Register 0 (FR_PIER0)* is asserted, an interrupt request is generated.

The status bit T1ST is set when the timer is triggered, and is cleared when the timer expires and is non-repetitive. If the timer expires but is repetitive, the T1ST bit is not cleared and the timer is restarted immediately. The T1ST is cleared when the timer is stopped.

Absolute / Relative timer T2

The timer T2 can be configured to be an absolute or relative timer by setting the T2_CFG control bit in the *Timer Configuration and Control Register (FR_TICCR)*. The status bit T2ST is set when the timer is triggered, and is cleared when the timer expires and is non-repetitive. If the timer expires but is repetitive, the T2ST bit is not cleared and the timer is restarted immediately. The T2ST is cleared when the timer is stopped.

Absolute timer T2

If timer T2 is configured as an absolute timer, it has the same functionality timer T1 but the configuration from *Timer 2 Configuration Register 0 (FR_TI2CR0)* and *Timer 2 Configuration Register 1 (FR_TI2CR1)* is used. On expiration of timer T2, the interrupt flag TI2_IF in the *Protocol Interrupt Flag Register 0 (FR_PIFR0)* is set. If the timer 1 interrupt enable bit TI1_IE in the *Protocol Interrupt Enable Register 0 (FR_PIER0)* is asserted, an interrupt request is generated.

Relative timer T2

If the timer T2 is configured as a relative timer, the interrupt flag TI2_IF in the *Protocol Interrupt Flag Register 0 (FR_PIFR0)* is set, when the programmed amount of macroticks MT[31:0], defined by *Timer 2 Configuration Register 0 (FR_TI2CR0)* and *Timer 2 Configuration Register 1 (FR_TI2CR1)*, has expired since the trigger or restart of timer 2. The relative timer is implemented as a down counter and expires when it has reached 0. At the macrotick start event, the value of MT[31:0] is checked and then decremented. Thus, if the timer is started with MT[31:0] == 0, it expires at the next macrotick start.

31.6.18 Slot status monitoring

The CC provides several means for slot status monitoring. All slot status monitors use the same slot status vector provided by the PE. The PE provides a slot status vector for each static slot, for each dynamic slot, for the symbol window, and for the NIT, on a per channel base. The content of the slot status vector is described in *Table 826*. The PE provides the slot status vector within the first macrotick after the end of the related slot/window/NIT, as shown in *Figure 861*.

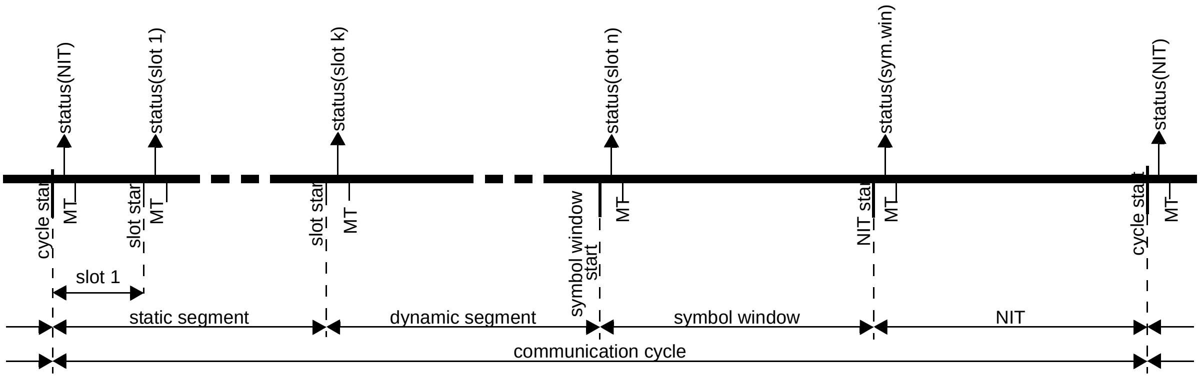


Figure 861. Slot status vector update

Note: The slot status for the NIT of cycle n is provided after the start of cycle n+1.

Table 826. Slot status content

Slot	Status content
static / dynamic Slot	**Slot related status** *vSS!ValidFrame* - valid frame received *vSS!SyntaxError* - syntax error occurred while receiving *vSS!ContentError* - content error occurred while receiving *vSS!BViolation* - boundary violation while receiving *for slots in which the module transmits:* *vSS!TxConflict* - reception ongoing while transmission starts *for slots in which the module does not transmit:* *vSS!TxConflict* - reception ongoing while transmission starts first valid - channel that has received the first valid frame **Received frame related status** *extracted from* *a) header of valid frame, if vSS!ValidFrame = 1* *b) last received header, if vSS!ValidFrame = 0* *c) set to 0, if nothing was received* *vRF!Header!NFIndicator* - Null Frame Indicator (0 for null frame) *vRF!Header!SuFIndicator* - Startup Frame Indicator *vRF!Header!SyFIndicator* - Sync Frame Indicator
Symbol Window	**Window related status** *vSS!ValidFrame* - always 0 *vSS!ContentError* - content error occurred while receiving *vSS!SyntaxError* - syntax error occurred while receiving *vSS!BViolation* - boundary violation while receiving *vSS!TxConflict* - reception ongoing while transmission starts **Received symbol related status** *vSS!ValidMTS* - valid Media Test Access Symbol received **Received frame related status** *see static/dynamic slot*
NIT	**NIT related status** *vSS!ValidFrame* - always 0 *vSS!ContentError* - content error occurred while receiving *vSS!SyntaxError* - syntax error occurred while receiving *vSS!BViolation* - boundary violation while receiving *vSS!TxConflict* - always 0 **Received frame related status** *see static/dynamic slot*

Channel status error counter registers

The two channel status error counter registers, *Channel A Status Error Counter Register (FR_CASERCR)* and *Channel B Status Error Counter Register (FR_CBSERCR)*, incremented by one, if at least one of four slot status error bits, *vSS!SyntaxError*, *vSS!ContentError*, *vSS!BViolation*, or *vSS!TxConflict* is set to 1. The status vectors for all slots in the static and dynamic segment, in the symbol window, and in the NIT are taken into account. The counters wrap round after they have reached the maximum value.

Protocol status registers

The *Protocol Status Register 2 (FR_PSR2)* provides slot status information about the Network Idle Time NIT and the Symbol Window. The *Protocol Status Register 3 (FR_PSR3)* provides aggregated slot status information.

Slot status registers

The eight slot status registers, *Slot Status Registers (FR_SSR0–FR_SSR7)*, can be used to observe the status of static slots, dynamic slots, the symbol window, or the NIT without individual message buffers. These registers provide all slot status related and received frame / symbol related status information, as given in *Table 826*, except of the *first valid* indicator for non-transmission slots.

Slot status counter registers

The CC provides four slot status error counter registers, *Slot Status Counter Registers (FR_SSCR0–FR_SSCR3)*. Each of these slot status counter registers is updated with the value of an internal slot status counter at the start of a communication cycle. The internal slot status counter is incremented if its increment condition, defined by the *Slot Status Counter Condition Register (FR_SSCCR)*, matches the status vector provided by the PE. All static slots, the symbol window, and the NIT status are taken into account. *Dynamic* slots are **excluded**. The internal slot status counting and update timing is shown in *Figure 862*.

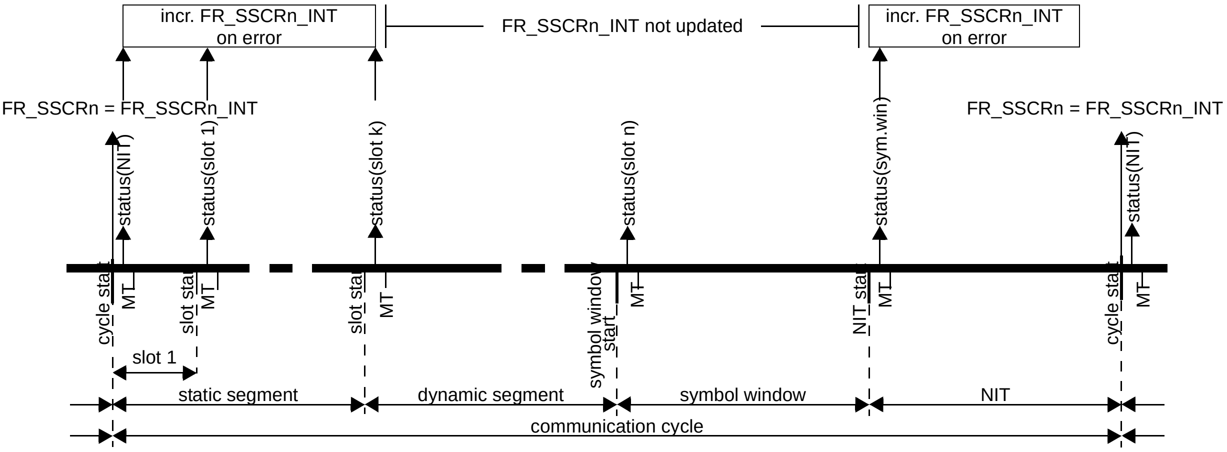


Figure 862. Slot status counting and FR_SSCRn update

The PE provides the status of the NIT in the first slot of the next cycle. Due to these facts, the FR_SSCRn register reflects, in cycle n, the status of the NIT of cycle n-2, and the status of all static slots and the symbol window of cycle n-1.

The increment condition for each slot status counter consists of two parts, the frame related condition part and the slot related condition part. The internal slot status counter FR_SSCRn_INT is incremented if at least one of the conditions is fulfilled:

1. frame related condition:

- (FR_SSCCRn[VFR] | FR_SSCCRn[SYF] | FR_SSCCRn[NUF] | FR_SSCCRn[SUF]) // count on frame condition
  = 1;

and

- ((~FR_SSCCRn[VFR] | *vSS!ValidFrame*) & // valid frame restriction
  (~FR_SSCCRn[SYF] | *vRF!Header!SyFIndicator*) & // sync frame indicator restriction
  (~FR_SSCCRn[NUF] | ~*vRF!Header!NFIndicator*) & // null frame indicator restriction
  (~FR_SSCCRn[SUF] | *vRF!Header!SuFIndicator*)) // startup frame indicator restriction
  = 1;

Note: The indicator bits SYF, NUF, and SUF are valid only when a valid frame was received. Thus it is required to set the VFR always, whenever count on frame condition is used.

2. slot related condition:

- ((FR_SSCCRn[STATUSMASK[3]] & *vSS!ContentError*) | // increment on content error
  (FR_SSCCRn[STATUSMASK[2]] & *vSS!SyntaxError*) | // increment on syntax error
  (FR_SSCCRn[STATUSMASK[1]] & *vSS!BViolation*) | // increment on boundary violation
  (FR_SSCCRn[STATUSMASK[0]] & *vSS!TxConflict*)) // increment on transmission conflict
  = 1;

If the slot status counter is in single cycle mode, that is, FR_SSCCRn[MCY] = 0, the internal slot status counter FR_SSCRn_INT is reset at each cycle start. If the slot status counter is in the multicycle mode, that is, FR_SSCCRn[MCY] = 1, the counter is not reset and incremented, until the maximum value is reached.

Message buffer slot status field

Each individual message buffer and each FIFO message buffer provides a slot status field, which provides the information shown in *Table 826* for the static/dynamic slot. The update conditions for the slot status field depend on the message buffer type. Refer to the Message Buffer Update Sections in *Section 31.6.6, Individual message buffer functional description*.

31.6.19 System bus access

This section provides a description of the system bus accesses failures and the related CC behavior. System bus access failures may occur when the CC transfers data to or from the flexray memory area.

The system bus access failure types are described in *Section , System bus access failure types*.

The behavior of the CC after the occurrence of a system bus access failure is described in *Section , System bus access failure response*.

System bus access failure types

This section describes the two types of system bus access failures.

The behavior of the CC after the occurrence of a system bus failure is defined by the SBFF bit in the *Module Configuration Register (FR_MCR)*.

System bus illegal address access

If the system bus detects an CC access to an illegal address, the CC receives a notification from the system bus about this event and sets the ILSA_EF flag in the *CHI Error Flag Register (FR_CHIERFR)*.

System bus access timeout

A system bus access timeout is detected if an access to the flexray memory area is not finished in time. The timeout value is derived from the SYMATOR[TIMEOUT] setting (see *Section , Configure System Memory Access Time-Out Register (FR_SYMATOR)*.

If a system bus access timeout is detected, the CC sets the SBCF_EF flag in the *CHI Error Flag Register (FR_CHIERFR)*.

System bus access failure response

This section describes the two types of behavior of the CC after the occurrence of a system bus access failure. The actual behavior is defined by the SBFF bit in the *Module Configuration Register (FR_MCR)*.

Continue after system bus access failure

If the SBFF bit in the *Module Configuration Register (FR_MCR)* is 0, the CC will continue its operation after the occurrence of the system bus access failure, but will not generate any system bus accesses until the start of the next communication cycle.Since no data are read from or written to the flexray memory area, no messages are received or transmitted. Consequently, none of the individual message buffers or receive FIFOs will be updated until the next communication cycle starts.

If a frame is under transmission when the system bus failure occurs, a correct frame is generated with the remaining header and frame data are replaced by all zeros. Depending on the point in time this can affect the PPI bit, the Header CRC, the Payload Length in case of an dynamic slot, and the payload data. Starting from the next slot in the current cycle, no frames will be transmitted and received, except for the key slot, where a sync or startup null-frame is transmitted, if the key slot is assigned.

If a frame is received when the system bus failure occurs, the reception is aborted and the related receive message buffer is not updated.

Normal operation is resumed after the start of next communication cycle.

Freeze after system bus access failure

If the SBFF bit in the *Module Configuration Register (FR_MCR)* is set to 1, the CC will go into the freeze mode immediately after the occurrence of one of the system bus access failures.

31.6.20 Interrupt support

The CC provides 172 individual interrupt sources and five combined interrupt sources.

Individual interrupt sources

Message buffer interrupts

The CC provides 128 message buffer interrupt sources.

Each individual message buffer provides an interrupt flag FR_MBCCSRn[MBIF] and an interrupt enable bit FR_MBCCSRn[MBIE]. The CC sets the interrupt flag when the slot status of the message buffer was updated. If the interrupt enable bit is asserted, an interrupt request is generated.

FIFO interrupts

The CC provides two FIFO interrupt sources.

Each of the two FIFOs provides a Receive FIFO Almost Full Interrupt Flag. The CC sets the Receive FIFO Almost Full Interrupt Flags (FR_GIFER[FAFBIF], FR_GIFER[FAFAIF]) in the *Global Interrupt Flag and Enable Register (FR_GIFER)* if the corresponding Receive FIFO fill level exceeds the defined watermark.

Wakeup interrupt

The CC provides one interrupt source related to the wakeup.

The CC sets the Wakeup Interrupt Flag FR_GIFER[WUPIF] when it has received a wakeup symbol on the FlexRay bus. The CC generates an interrupt request if the interrupt enable bit FR_GIFER[WUPIE] is asserted.

Protocol interrupts

The CC provides 25 interrupt sources for protocol related events. For details, see *Protocol Interrupt Flag Register 0 (FR_PIFR0)* and *Protocol Interrupt Flag Register 1 (FR_PIFR1)*. Each interrupt source has its own interrupt enable bit.

CHI interrupts

The CC provides 16 interrupt sources for CHI related error events. For details, see *CHI Error Flag Register (FR_CHIERFR)*. There is one common interrupt enable bit FR_GIFER[CHIE] for all CHI error interrupt sources.

Combined interrupt sources

Each combined interrupt source generates an interrupt request only when at least one of the interrupt sources that is combined generates an interrupt request.

Receive message buffer interrupt

The Receive Message Buffer Interrupt request is generated when at least one of the individual receive message buffers generates an interrupt request MBXIRQ[n] and the interrupt enable bit FR_GIFER[RBIE] is set.

Transmit message buffer interrupt

The Transmit Message Buffer Interrupt request is generated when at least one of the individual transmit message buffers generates an interrupt request MBXIRQ[n] and the interrupt enable bit FR_GIFER[TBIE] is asserted.

Protocol interrupt

The Protocol Interrupt request is generated when at least one of the individual protocol interrupt sources generates an interrupt request and the interrupt enable bit FR_GIFER[PRIE] is set.

CHI interrupt

The CHI Interrupt request is generated when at least one of the individual CHI error interrupt sources generates an interrupt request and the interrupt enable bit FR_GIFER[CHIE] is set.

Module interrupt

The Module Interrupt request is generated if at least one of the combined interrupt sources generates an interrupt request and the interrupt enable bit FR_GIFER[MIE] is set.

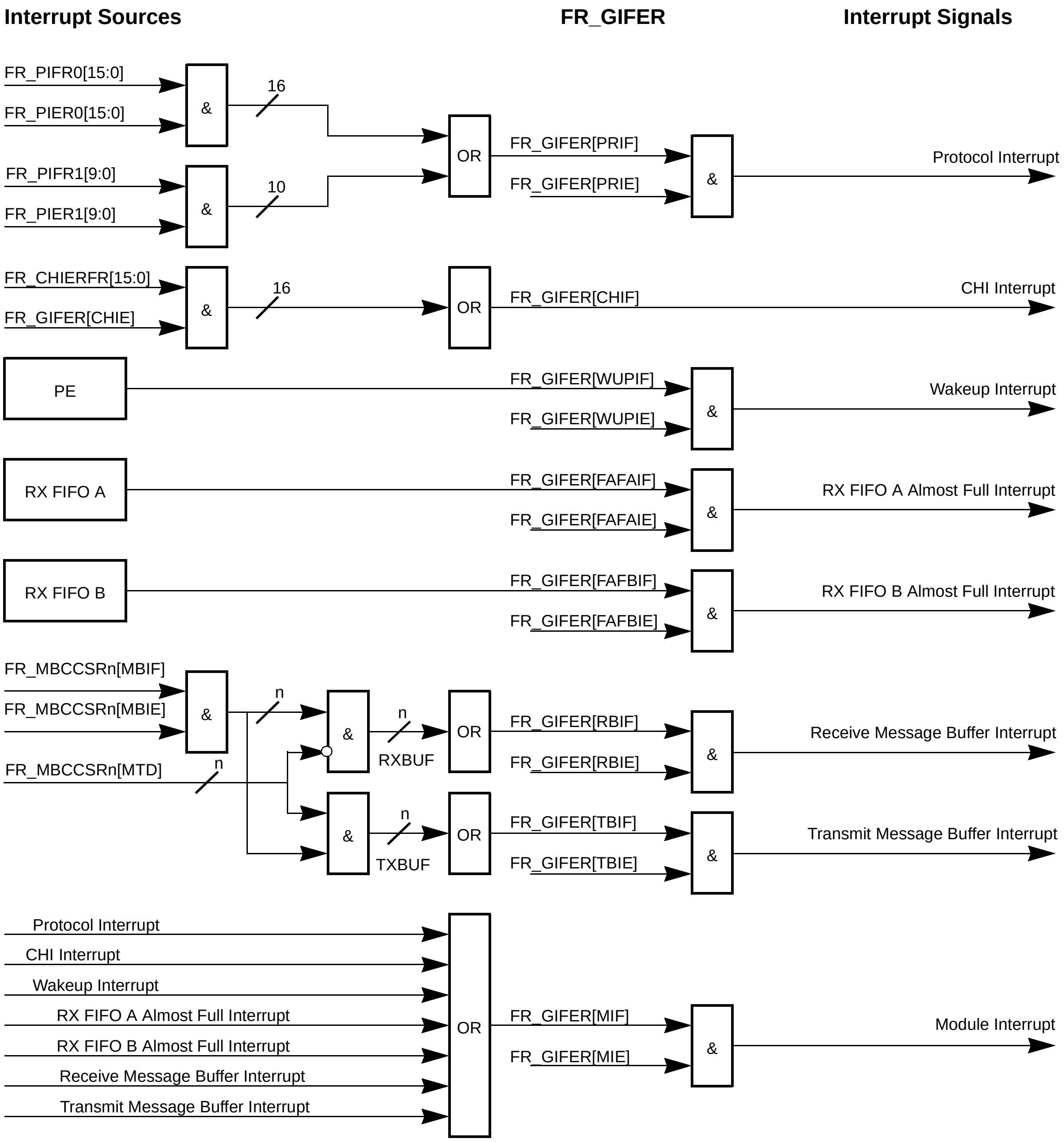


Figure 863. Scheme of FR_GIFER interrupt signal generation

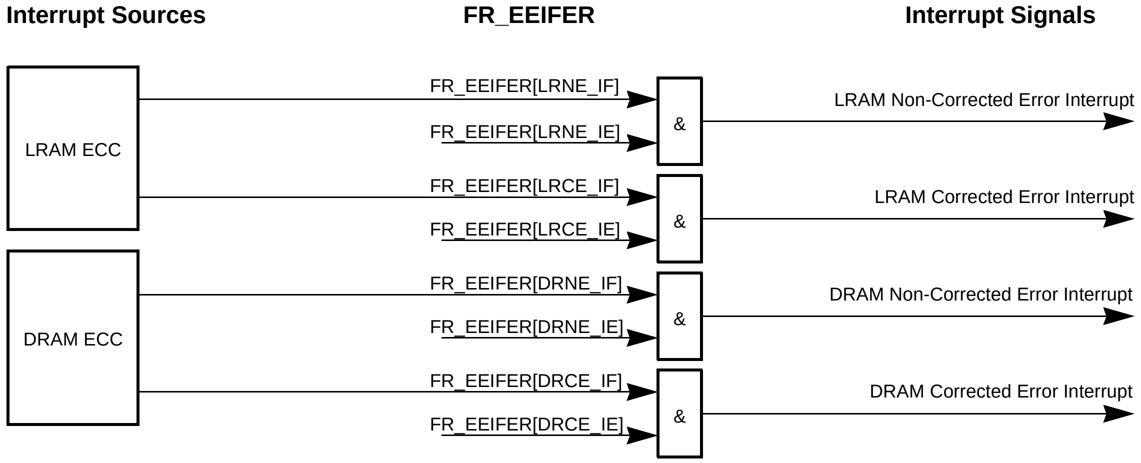


Figure 864. Scheme of FR_EEIFER interrupt signal generation

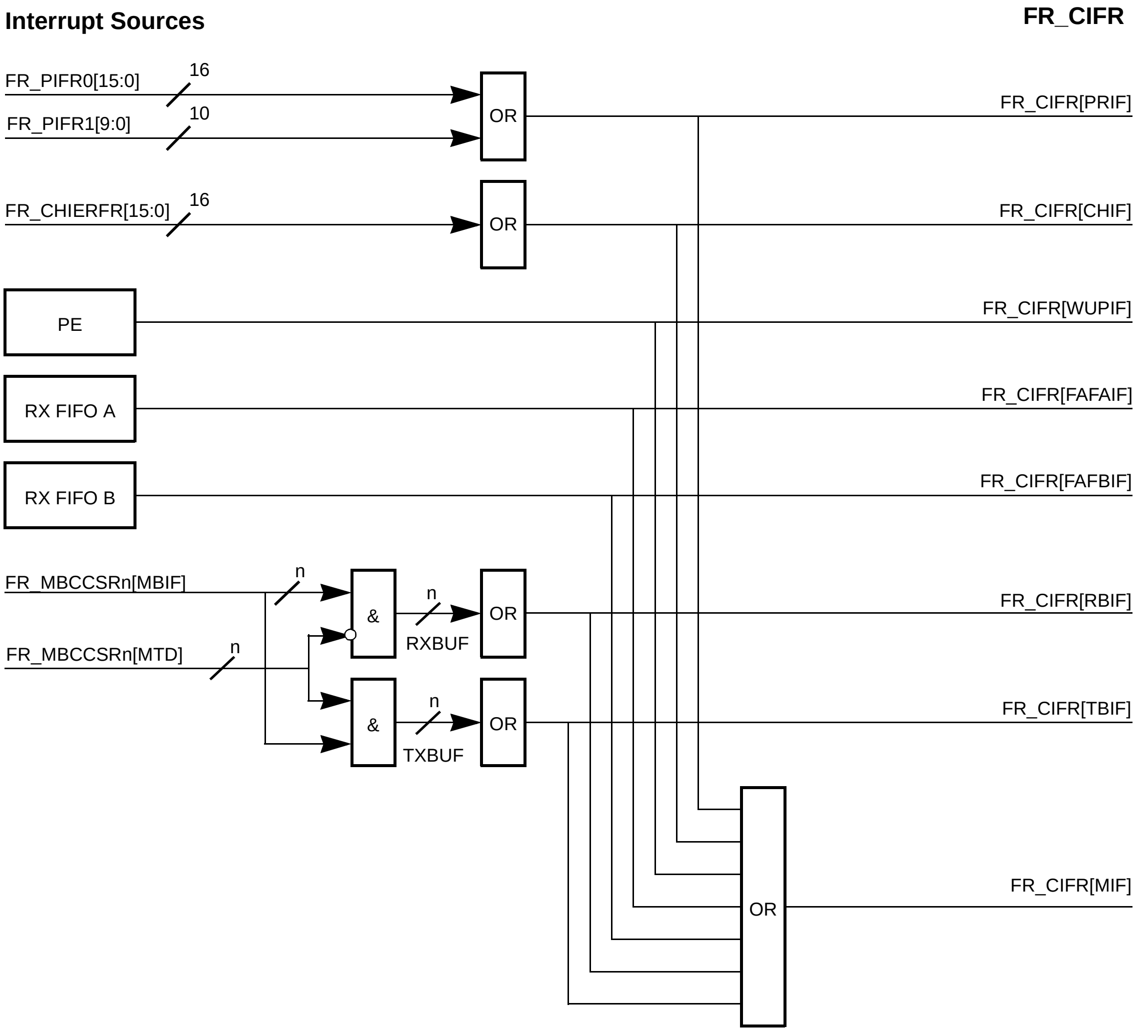


Figure 865. Scheme of FR_CIFR flags generation

31.6.21 Lower bit rate support

The CC supports a number of lower bit rates on the FlexRay bus channels. The lower bit rates are implemented by modifying the duration of the microtick *pdMicrotick*, the number of samples per microtick *pSamplesPerMicrotick*, the number of samples per bit *cSamplesPerBit*, and the strobe offset *cStrobeOffset.* The application configures the FlexRay channel bit rate by setting the BITRATE field in the *Module Configuration Register (FR_MCR)*. The protocol values are set internally. The available bit rates, the related BITRATE field configuration settings and related protocol parameter values are shown in *Table 827*.

Table 827. FlexRay channel bit rate control

FlexRay channel bit rate [Mbit/s]	FR_MCR.BITRATE	*pdMicrotick* [ns]	*gdSampleClockPeriod* [ns]	*pSamplesPerMicrotick*	*cSamplesPerBit*	*cStrobeOffset*
10.0	000	25.0	12.5	2	8	5
8.0	011	25.0	12.5	2	10	6
5.0	001	25.0	25.0	1	8	5
2.5	010	50.0	50.0	1	8	5

Note: The bit rate of 8 Mbit/s is not defined by the FlexRay Communications System Protocol Specification, Version 2.1 Rev A.

31.6.22 PE data memory (PE DRAM)

The PE Data Memory (PE DRAM) is 128 word, 16-bit wide memory with byte access, which contains the program data of the PE internal CPU. The PE DRAM is divided into two banks, 8-bit each. The memory data [7:0] are assigned to BANK0, the memory data [15:8] are assigned to BANK1.

Table 828. PE DRAM layout

ADDR	BANK1	BANK0
0x00	byte1	byte0
0x01	byte3	byte2
	...	
0x7F	byte255	byte254

The FlexRay module provides means to access the PE DRAM from the application. The PE DRAM application access is initiated and controlled via *PE DRAM Access Register (FR_PEDRAR)* and *PE DRAM Data Register (FR_PEDRDR)*. This functionality is used to check the memory error detection.

PE DRAM read access

A read access from the PE DRAM can be initiated in any protocol state. The following sequence describes a read access from the PE DRAM address 0x70.

1. FR_PEDRAR = 0x00E0; // INST = 0x0; ADDR = 070
2. wait until FR_PEDRAR[DAD] == 1; // wait for end of PE DRAM access
3. val = FR_PEDRDR[DATA]; // get read PE DRAM data

The read access is handled by the PE internal CPU with the lowest execution priority. This may cause an response delay with a maximum of 1000 PE clock cycle (25µs).

PE DRAM write access

A write access into the PE DRAM can be initiated in any protocol state. The following sequence describes a write access to the PE DRAM address 0x70.

1. FR_PEDRAR = 0x30E0; // INST = 0x3; ADDR = 0x70
2. wait until FR_PEDRAR[DAD] == 1; // wait for end of PE DRAM access
3. val = FR_PEDRDR[DATA]; // get read back PE DRAM data

The write access is handled by the PE internal CPU with the lowest execution priority. This may causes an response delay with a maximum of 1000 PE clock cycle (25µs).

If the conditions given in *Section , PE DRAM write access limitations* are fulfilled, the data provided in *PE DRAM Data Register (FR_PEDRDR)* are written into the PE DRAM, read back in the next clock cycle and stored into the *PE DRAM Data Register (FR_PEDRDR)*. Otherwise, data are not written into the PE DRAM and 0x0000 is stored into the *PE DRAM Data Register (FR_PEDRDR)*.

PE DRAM write access limitations

The PE DRAM is used by the protocol engine if the module is not in *POC:default config* state. The only address not used by the protocol engine is 0x70. To prevent the corruption of protocol engine data the following PE DRAM write access limitations apply for application writes.

1. When the module is in *POC:default config* state, all PE DRAM addresses are writable.
2. When the module is not in *POC:default config* state, only PE DRAM address 0x70 is writable.

31.6.23 CHI lookup-table memory (CHI LRAM)

The CHI Lookup-Table Memory (CHI LRAM) is an CHI internal memory which contains the message buffer configuration data. The configuration data for two message buffers are contained in one memory row. The CHI LRAM is divided into six memory banks.

Table 829. CHI LRAM layout

ADR	BANK5	BANK4	BANK3	BANK2	BANK1	BANK0
0x00	FR_MBIDXR1	FR_MBFIDR1	FR_MBCCFR1	FR_MBIDXR0	FR_MBFIDR0	FR_MBCCFR0
0x01	FR_MBIDXR3	FR_MBFIDR3	FR_MBCCFR3	FR_MBIDXR2	FR_MBFIDR2	FR_MBCCFR2
...						
0x1F	FR_MBIDXR127	FR_MBFIDR127	FR_MBCCFR127	FR_MBIDXR126	FR_MBFIDR126	FR_MBCCFR126

The CHI LRAM is accessed by the application via regular register read and write accesses.

31.6.24 Memory content error detection

The FlexRay module provides integrated memory content error detection for both the CHI LRAM and PE DRAM, and memory content error correction for the PE DRAM. The memory error detection for the CHI LRAM uses an standard Hamming code with a Hamming distance of 3 and detects all single-bit and double-bit errors (SEDDED). The memory error detection and correction for the PE DRAM uses an enhanced Hamming code with a

Hamming distance of 4 and detects and corrects all single-bit errors and detects all double-bit errors (SECDED).

This section describes the reporting of the occurrence of memory content errors, the reaction of the module on the occurrence, and how the application can inject memory errors in order to trigger the report and response behavior.

Memory error types

A memory error is the distortion of one or more bits read out of the memory. The reading of the values of all zeros and all ones is considered as an special case. The FlexRay module detects and indicates the memory errors as shown in *Table 830*. The entries on the top have higher priority.

Each memory read access reads out **all** banks of the addressed row, and runs error detection on **all** banks, even in the case that the application has triggered a read from only one bank. This may lead to the reporting of an memory error if at least one bank contains a memory error, even if an error free bank has been read.

Table 830. Detected memory error types

Memory	Priority	Memory data	Indication
CHI LRAM	0 (highest)	All zero's	No Error – Valid Data
PE DRAM			Non-Corrected Error
CHI LRAM		All one's	Non-Corrected Error
PE DRAM			
CHI LRAM	1 (lowest)	One bit flipped	Non-Corrected Error
PE DRAM			Corrected Error
CHI LRAM		Two bits flipped	Non-Corrected Error
PE DRAM			
CHI LRAM		Three or more bits flipped	One out of {No error, Non-Corrected Error}, defined by coding given in *Section , CHI LRAM checkbits* and *Section , CHI LRAM checkbits*
PE DRAM			One out of {No error, Corrected Error, Non-Corrected Error}, defined by coding given in *Section , PE DRAM checkbits* and *Section , PE DRAM syndrome*

Memory error reporting

The memory error reporting is enabled only if the ECC functionality enable bit ECCE in the *Module Configuration Register (FR_MCR)* is set.

For each of the two memories exists two sets of internal registers to store the detection of one corrected and one non-corrected memory error.

If a memory error is detected, the module checks whether the related error interrupt flag in the *ECC Error Interrupt Flag and Enable Register (FR_EEIFER)* is set.

- If the error interrupt flag is set, the related internal error reporting register is not updated and the related error overflow flag is set to 1 to indicate a loss of error condition.
- If the error interrupt flag is not set, the internal reporting register is updated and the error interrupt flag is set to 1. If two or more memory errors of the same type are detected, the error for the bank with the lower bank number will be reported, and the error overflow flag will be set to 1.

If a memory error is detected for at least two banks of one memory, the related error overflow flag is set to 1 to indicate a loss of error condition.

PE DRAM checkbits

The coding of the checkbits reported in *ECC Error Report Code Register (FR_EERCR)* for PE DRAM memory errors is shown in *Table 832*. This table shows the implemented enhanced Hamming code. If the error injection was applied to distort the checkbits, then the distorted checkbits are reported.

Table 831. PE DRAM checkbits coding

CODE	**CODE**				**DATA**							
	3	**2**	**1**	**0**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
4(1)	X	X	X	X	X	X	X	X	X	X	X	X
3(2)	—	—	—	—	X	X	X	X	—	—	—	—
2	—	—	—	—	X	—	—	—	X	X	X	—
1	—	—	—	—	—	X	X	—	X	X	—	X
0	—	—	—	—	—	X	—	X	X	—	X	X

1. The checkbit CODE[4] is set to 1 if and only if there is a even number of 1’s in columns with X.
2. The checkbits CODE[3]... CODE[0] are set to 1 if and only if there is a odd number of 1’s in all columns with X.

This coding of the checkbit ensures that neither 0x000 nor 0xFFF are valid code words and written into the memory

PE DRAM syndrome

The coding of the syndrome reported in *ECC Error Report Code Register (FR_EERCR)* for PE DRAM memory errors is shown in *Table 832*.

Table 832. FR_EERCR[CODE] PE DRAM syndrome coding

FR_EERCR[CODE]		Description
[4]	[3:0]	
0x1	0x0	No Error (Never appears in error report registers)
0x0	0x0	If data == 0: Non-Corrected Error (Dedicated Handling of All Zero Code Word) If data != 0: Corrected Error (Parity Bit 4)
0x0	0x1	Corrected Error (Parity Bit 0)
0x0	0x2	Corrected Error (Parity Bit 1)
0x0	0x3	Corrected Error (Data Bit 0)
0x0	0x4	Corrected Error (Parity Bit 2)
0x0	0x5	Corrected Error (Data Bit 1)
0x0	0x6	Corrected Error (Data Bit 2)
0x0	0x7	Corrected Error (Data Bit 3)
0x0	0x8	Corrected Error (Parity Bit 3)
0x0	0x9	Corrected Error (Data Bit 4)
0x0	0xA	Corrected Error (Data Bit 5)
0x0	0xB	Corrected Error (Data Bit 6)
0x0	0xC	Corrected Error (Data Bit 7)
0x0	0xD – 0xF	Non-Corrected Error
0x1	0x1 – 0xF	Non-Corrected Error

CHI LRAM checkbits

The coding of the checkbits reported in *ECC Error Report Code Register (FR_EERCR)* for CHI LRAM memory errors is shown in *Table 833*. This table shows the implemented Hamming code. If the error injection was applied to distort the checkbits, then the distorted checkbits are reported.

Table 833. CHI LRAM checkbits coding

CODE(1)	DATA															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
4	X	X	X	X	X	—	—	—	—	—	—	—	—	—	—	—
3	—	—	—	—	—	X	X	X	X	X	X	X	—	—	—	—
2	X	X	—	—	—	X	X	X	X	—	—	—	X	X	X	—
1	—	—	X	X	—	X	X	—	—	X	X	—	X	X	—	X
0	X	—	X	—	X	X	—	X	—	X	—	X	X	—	X	X

1. The checkbit CODE[n] is set to 1 if and only if there is a odd number of 1's in all columns with X.

CHI LRAM syndrome

The coding of the syndrome reported in *ECC Error Report Code Register (FR_EERCR)* for CHI LRAM memory errors is shown in *Table 834*.

Table 834. FR_EERCR[CODE] CHI LRAM syndrome coding

FR_EERCR[CODE]	Description
0x00	No Error (Never appears in error report registers)
0x01 – 0x1F	Non-Corrected Error

Memory error response

The memory error response is enabled only when the ECC functionality enable bit ECCE in the *Module Configuration Register (FR_MCR)* is set.

In case of the detection of a *corrected* memory error, the FlexRay module continues its normal operation using the corrected data word. This section describes the behavior of the FlexRay module after the detection of a *non-corrected* memory error.

CHI LRAM memory error response after module read

The FlexRay module reads the message buffer configuration buffer data located in the CHI LRAM for each message buffer one time in each slot and in the NIT.

If a non-corrected memory error is detected during this module read access, the FlexRay module will consider the affected message buffer as disabled for the current search and will exclude this buffer from the search. The configuration of the affected message buffer is not changed.

If the affected message buffer is a tx message buffer, no frame will be transmitted from this message buffer in the next slot. If the affected message buffer is a rx message buffer, no frame will be received to this message buffer in the next slot.

CHI LRAM memory error response after application read

The application can read the content of the CHI LRAM via reading the FR_MBCCFRn, FR_MBFIDRn, and FR_MBIDXRn registers. If a non-corrected memory error is detected during this kind of read access, the module indicates the detected memory error, delivers the non-corrected data read and continues its normal operation.

PE DRAM error response after module read

If the module detects an non-corrected memory error during read of program data which is contained in PE DRAM, this is considered as an fatal protocol error and the module enters the protocol freeze state immediately.

PE DRAM error response after application read in *POC:default config* state

If the module detects an non-corrected memory error during an application triggered read from any PE DRAM address and the protocol is in the *POC:default config* state, this is considered as an fatal protocol error and the module enters the protocol freeze state. This behavior allows for checking the freeze functionality in case of the detection of non-corrected errors.

PE DRAM error response after application read out of *POC:default config*

If the module detects an non-corrected memory error during an application triggered read from any PE DRAM address, and the protocol is not in the *POC:default config* state, this error is not considered as an fatal error and the protocol state is not changed. This prevents any interference of the running protocol by PE DRAM error injection reads.

31.6.25 Memory error injection

The error injection functionality is used by the application to inject data errors into the memories to trigger and check the memory error detection functionality.

The error injection is enabled only if the ECC functionality enable bit ECCE in the *Module Configuration Register (FR_MCR)* and the error injection enable control bit EIE in the *ECC Error Report and Injection Control Register (FR_EERICR)* are set.

The error injection mode is configured by the EIM configuration bit in the *ECC Error Report and Injection Control Register (FR_EERICR)*.When the error injection is enabled, each write access to the configured memory location will be distorted.

The injector has the same behavior for FlexRay module memory writes and application memory writes.

CHI LRAM error injection

The following sequence describes an error injection sequence for the CHI LRAM. This sequence includes the setup of the error injector followed by an application triggered write access to provoke an distortion of the memory content. When the FlexRay module is in *POC:default config*, there are no limitations and impacts of error injection for the application. For error injection out of *POC:default config* see *Section 31.7.3, CHI LRAM error injection out of POC:default config*.

Injector setup:

1. FR_MCR[ECCE] = 1;
   enable ECC functionality
2. FR_EERICE[EIE] = I_MODE;
   configure error injection mode
3. FR_EEIAR[MID] = 1;
   select CHI LRAM for error injection
4. FR_EEIAR[BANK] = I_BANK;
   define the bank for error injection; I_BANK = {0,1,2,3,4,5}
5. FR_EEIAR[ADDR] = I_ADDR;
   define the address for error injection; 0 <= I_ADDR <= 0x1F
6. FR_EEIDR[DATA] = D_DIST;
   define the data distortion pattern
7. FR_EEICR[CODE] = C_DIST;
   define the checkbit distortion pattern
8. FR_EERICE[EIE] = 1;
   enable error injection

Application write access:

1. If (I_BANK==0) ⇒ FR_MBCCFR(2*I_ADDR) = DATA;
   If (I_BANK==1) ⇒ FR_MBFIDR(2*I_ADDR) = DATA;
   If (I_BANK==2) ⇒ FR_MBIDXR(2*I_ADDR) = DATA;
   If (I_BANK==3) ⇒ FR_MBCCFR(2*I_ADDR+1) = DATA;
   If (I_BANK==4) ⇒ FR_MBFIDR(2*I_ADDR+1) = DATA;
   If (I_BANK==5) ⇒ FR_MBIDXR(2*I_ADDR+1) = DATA;

   write DATA to the defined injection bank and injection address

PE DRAM error injection

The following sequence describes an error injection sequence for the PE DRAM. This sequence includes the setup of error injector followed by an application triggered write access to provoke an distortion of the memory content. When the FlexRay module is in *POC:default config*, there are no limitations and impacts of error injection for the application. For error injection out of *POC:default config* see *Section 31.7.4, PE DRAM error injection out of POC:default config*.

Injector Setup:

1. FR_MCR[ECCE] = 1;
   - enable ECC functionality
2. FR_EERICE[EIE] = I_MODE;
   - configure error injection mode
3. FR_EEIAR[MID] = 0;
   - select PE DRAM for error injection
4. FR_EEIAR[BANK] = I_BANK;
   - define the bank for error injection; I_BANK = {0,1}
5. FR_EEIAR[ADDR] = I_ADDR;
   - define the address for error injection; 0<= I_ADDR <= 0x7F
6. FR_EEIDR[DATA] = D_DIST;
   - define the data distortion pattern
7. FR_EEICR[CODE] = C_DIST;
   - define the checkbit distortion pattern
8. FR_EERICE[EIE] = 1;
   - enable error injection

Application Write Access (I_ADDR = 0x70):

1. FR_PEDRAR:= 0x30E0; // INST = 0x3; ADDR = 0x70
2. wait until FR_PEDRAR[DAD] == 1; // wait for end of PE DRAM access
3. val = FR_PEDRDR[DATA]; // get read back PE DRAM data

Note: *The write access to the PE DRAM triggers a read from PE DRAM in the next cycle, which triggers the detection of the distorted data.*

31.7 Application information

31.7.1 Module configuration

This section describes essential parts of the module configuration.

Configure *System Memory Access Time-Out Register (FR_SYMATOR)*

To ensure reliable operation of the CC, the application has to ensure that the TIMEOUT value in *System Memory Access Time-Out Register (FR_SYMATOR)* and the CHI clock frequency f_{CHI} in MHz fulfill *Equation 72*[bf].

Equation 72

$$0 \leq \mathrm{SYMATOR[TIMEOUT]} \leq \lfloor 0.45 \cdot f_{CHI} - 8 \rfloor$$

If the SYMATOR[TIMEOUT] value and f_{CHI} violates *Equation 72*, the behavior of the CC becomes unreliable and undefined. It may happen that frames are not transmitted at all, including key slot frames.

For a given SYMATOR[TIMEOUT] value f_{CHI} can be increased without causing unreliable operation of the CC. The same holds for reducing the SYMATOR[TIMEOUT] value for a given f_{CHI}.

Some examples for maximum values of the SYMATOR[TIMEOUT] for a minimum CHI frequency are given in *Table 835*.

Table 835. Maximum SYMATOR[TIMEOUT] examples

f_{CHI}	SYMATOR[TIMEOUT]	f_{CHI}	SYMATOR[TIMEOUT]
≥ 18 MHz	0	≥ 100 MHz	≤ 37
≥ 23 MHz	≤ 2	≥ 120 MHz	≤ 46
≥ 27 MHz	≤ 4	≥ 140 MHz	≤ 55
≥ 32 MHz	≤ 6	≥ 160 MHz	≤ 64
≥ 60 MHz	≤ 19	≥ 180 MHz	≤ 73
≥ 80 MHz	≤ 28	≥ 200 MHz	≤ 82

System bus wait state constraints

The SYMATOR[TIMEOUT] value corresponds directly to a certain acceptable number of wait states on the system bus.

For single channel configurations and if the sync frame table generation functionality is **not** used (FR_SFTCCSR[SDVEN,SIDEN] = 0) no timeout will be detected if less than 2*SYMATOR[TIMEOUT] + 1 wait states will be seen on the system bus for each system bus access.

For dual channel configurations, or if the sync frame table generation functionality is used, no timeout will be detected if less than SYMATOR[TIMEOUT] - 1 wait states will be seen on the system bus for each system bus access.

[bf] See *Section 31.3, Controller host interface clocking*” for all constraints of minimum CHI clock frequency.

31.7.2 Initialization Sequence

This section describes the required steps to initialize the CC. The first subsection describes the steps required after a system reset, the second section describes the steps required after preceding shutdown of the CC.

Module Initialization

This section describes the module related initialization steps after a system reset.

1. Configure CC.
   a) configure the control bits in the *Module Configuration Register (FR_MCR)*
   b) configure the system memory base address in *System Memory Base Address Register (FR_SYMBADR)*
2. Enable the CC.
   a) write 1 to the module enable bit MEN in the *Module Configuration Register (FR_MCR)*

The CC now enters the Normal Mode. The application can commence with the protocol initialization described in *Section , Protocol Initialization*.

Protocol Initialization

This section describes the protocol related initialization steps.

1. Configure the Protocol Engine.
   a) issue CONFIG command via *Protocol Operation Control Register (FR_POCR)*
   b) wait for *POC:config* in *Protocol Status Register 0 (FR_PSR0)*
   c) configure the FR_PCR0,..., FR_PCR30 registers to set all protocol parameters
2. Configure the Message Buffers and FIFOs.
   a) set the number of message buffers used and the message buffer segmentation in the *Message Buffer Segment Size and Utilization Register (FR_MBSSUTR)*
   b) define the message buffer data size in the *Message Buffer Data Size Register (FR_MBDSR)*
   c) configure each message buffer by setting the configuration values in the *Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn)*, *Message Buffer Cycle Counter Filter Registers (FR_MBCCFRn)*, *Message Buffer Frame ID Registers (FR_MBFIDRn)*, *Message Buffer Index Registers (FR_MBIDXRn)*
   d) configure the FIFOs
   e) issue CONFIG_COMPLETE command via *Protocol Operation Control Register (FR_POCR)*
   f) wait for *POC:ready* in *Protocol Status Register 0 (FR_PSR0)*

After this sequence, the CC is configured as a FlexRay node and is ready to integrate into the FlexRay cluster.

CHI LRAM initialization

The module will start reading CHI LRAM data if it has entered the start up state, thus, all ECC bits have to set correctly. To fulfill this requirement, the application must write initial values into all message buffer configuration registers FR_MBCCFRn, FR_MBFIDRn, and FR_MBIDXRn during the protocol config state, even if the message buffers are not used.

PE DRAM initialization

The PE DRAM initialization is performed by the module in the *POC:default config* state. This initialization runs for 4.8 µs, and will delay the state transition from *POC:default config* into *POC:config*.

31.7.3 CHI LRAM error injection out of *POC:default config*

When the FlexRay module is out of the *POC:default config* state, it reads the configuration data of all utilized message buffers in every slot. If the module reads the CHI LRAM address that was used for error injection, an memory error is detected and the message buffer is not used for transmission or reception. This section describes how to inject errors on the CHI LRAM without disturbing the running application.

- Set injection address to FR_EEIDR[ADDR] = 0x1F
  - only the last two message buffers are affected by error injection
- Utilize less than 63 message buffers; FR_MBSSUTR[LAST_MB_UTIL] ≤ 62
  - the last two message buffers are not used and configuration data are not read by the module

31.7.4 PE DRAM error injection out of *POC:default config*

When the FlexRay module is out of the *POC:default config* state, only the PE DRAM address 0x70 is writable by the application. This location is not used by the FlexRay module.

31.7.5 Shut down sequence

This section describes a secure shut down sequence to stop the CC. The main targets of this sequence are

- Finish all ongoing reception and transmission
- Do not corrupt FlexRay bus and do not disturb ongoing FlexRay bus communication

For a shutdown the application shall perform the following tasks:

1. Disable all enabled message buffers.
   a) Repeatedly write ‘1’ to FR_MBCCSRn[EDT] until FR_MBCCSRn[EDS] == 0.
2. Stop Protocol Engine.
   a) Issue HALT command via *Protocol Operation Control Register (FR_POCR)*
   b) Wait for *POC:halt* in *Protocol Status Register 0 (FR_PSR0)*

31.7.6 Number of usable message buffers

This section describes the required minimum CHI clock frequency for a specified number of utilized message buffers configured in the *Message Buffer Segment Size and Utilization Register (FR_MBSSUTR)*, a configured minislot length *gdMinislot*, and a configured nominal macrotick length *gdMacrotick*(bg).

Additional constraints for the minimum CHI clock frequency are given in *Section 31.3, Controller host interface clocking*.

[bg] See *Section 31.3, Controller host interface clocking*” for all constraints of minimum CHI clock frequency.

The CC uses a sequential search algorithm to determine the individual message buffer assigned or subscribed to the next slot. This search is started at the start of slot and must be finished before the start of the next slot.

The shortest FlexRay slot is an corrected empty dynamic slot. An corrected empty dynamic slot is a minislot and consists of *gdMinislot* corrected macroticks with a duration of *gdMacrotick*. The minimum duration of an corrected macrotick is $gdMacrotick_{min}$ = 39 µT. This results in a minimum length of an correct slot

Equation 73

$$\Delta_{slotmin} = 39 \cdot pdMicrotick \cdot gdMinislot$$

The message buffer search engine runs on the CHI clock and evaluates one individual message buffer per CHI clock cycle. For internal status update operations and to account for clock domain crossing jitter, an additional amount of 10 CHI clock cycles is required to ensure correct search engine operation.

For a given number of utilized message buffers FR_MBSSUTR[LAST_MB_UTIL] + 1 and for a given CHI clock frequency f_{chi}, this results in a search duration of

Equation 74

$$\Delta_{search} = \frac{1}{f_{chi}} \cdot (\text{FR\_MBSSUTR[LAST\_MB\_UTIL]+10})$$

The message buffer search must be finished within one slot which requires that *Equation 75* must be fulfilled:

Equation 75

$$\Delta_{search} \leq \Delta_{slotmin}$$

This results in the formula given in *Equation 76* which determines the required minimum CHI frequency for a given number of message buffers that are utilized.

Equation 76

$$f_{chi} \geq \frac{(\text{FR\_MBSSUTR[LAST\_MB\_UTIL]+10})}{39 \cdot pdMicrotick \cdot gdMinislot}$$

The required minimum CHI Clock frequency for a selected set of relevant protocol parameters and for the LAST_MB_UTIL field in the *Message Buffer Segment Size and Utilization Register (FR_MBSSUTR)* set to 127 is given in *Table 836*.

Table 836. Minimum f_{chi} [MHz] examples (128 message buffers)

pdMicrotick **[ns]**	***gdMinislot***					
	2	**3**	**4**	**5**	**6**	**7**
25.0	70.77	47.18	35.39	28.31	23.59	20.22
50.0	35.39	23.59	17.70	14.16	11.80	10.11

31.7.7 Protocol control command execution

This section considers the issues of the protocol control command execution.

The application issues any of the protocol control commands listed in the POCCMD field of *Table 716* by writing the command to the POCCMD field of the *Protocol Operation Control Register (FR_POCR)*. As a result the CC sets the BSY bit while the command is transferred to the PE. When the PE has accepted the command, the BSY flag is cleared. All commands are accepted by the PE.

The PE maintains a protocol command vector. For each command that was accepted by the PE, the PE sets the corresponding command bit in the protocol command vector. If a command is issued while the corresponding command bit is set, the command is not queued and is lost.

If the command execution block of the PE is idle, it selects the next accepted protocol command with the highest priority from the current protocol command vector according to the protocol control command priorities given in *Table 837*. If the current protocol state does not allow the execution of this protocol command (see POC state changes in *FlexRay Communications System Protocol Specification, Version 2.1 Rev A*) the CC asserts the illegal protocol command interrupt flag IPC_IF in the *Protocol Interrupt Flag Register 1 (FR_PIFR1)*. The protocol command is not executed in this case.

Some protocol commands may be interrupted by other commands or the detection of a fatal protocol error as indicated by *Table 837*. If the application issues the FREEZE or READY command, or if the PE detects a fatal protocol error, some commands already stored in the command vector will be removed from this vector.

Table 837. Protocol control command priorities

Protocol command	Priority	Interrupted by	Cleared and terminated by
FREEZE	(highest) 1	none	
READY	2		
CONFIG_COMPLETE	3		
ALL_SLOTS	4	FREEZE, READY, CONFIG_COMPLETE, fatal protocol error	FREEZE, READY, CONFIG_COMPLETE, fatal protocol error
ALLOW_COLDSTART	5		
RUN	6		FREEZE, fatal protocol error
WAKEUP	7		FREEZE, fatal protocol error
DEFAULT_CONFIG	8		FREEZE, fatal protocol error
CONFIG	9		
HALT	(lowest) 10		FREEZE, READY, CONFIG_COMPLETE, fatal protocol error

31.7.8 Message buffer search on simple message buffer configuration

This sections describes the message buffer search behavior for a simplified message buffer configuration. The FIFO behavior is not considered in this section.

Simple message buffer configuration

A simple message buffer configuration is a configuration that has at most one transmit message buffer and at most one receive message buffer assigned to a slot *S*. The simple configuration used in this section utilizes two message buffers, one single buffered transmit message buffer and one receive message buffer.

The transmit message buffer has the message buffer number *t* and is configured as shown in *Table 838*.

Table 838. Transmit buffer configuration

Register	Field	Value	Description
FR_MBCCSR*t*	MCM	—	used only for double buffers
	MBT	0	single transmit buffer
	MTD	1	transmit buffer
FR_MBCCFR*t*	MTM	0	event transition mode
	CHA	1	assigned to channel A
	CHB	0	not assigned to channel B
	CCFE	1	cycle counter filter enabled
	CCFMSK	000011	cycle set = {4n} = {0,4,8,12,...}
	CCFVAL	000000	
FR_MBFIDR*t*	FID	*S*	assigned to slot S

The availability of data in the transmit buffer is indicated by the commit bit FR_MBCCSRt[CMT] and the lock bit FR_MBCCSRt[LCKS].

The receive message buffer has the message buffer number *r* and is configured as shown in *Table 839*.

Table 839. Receive buffer configuration

Register	Field	Value	Description
FR_MBCCSR*r*	MCM	—	n/a
	MBT	—	n/a
	MTD	0	receive buffer
FR_MBCCFR*r*	MTM	—	n/a
	CHA	1	assigned to channel A
	CHB	0	not assigned to channel B
	CCFE	1	cycle counter filter enabled
	CCFMSK	000001	cycle set = {2n} = {0,2,4,6,...}
	CCFVAL	000000	
FR_MBFIDR*r*	FID	*S*	subscribed slot

Furthermore the assumption is that both message buffers are enabled (FR_MBCCSR*t*[EDS] = 1 and FR_MBCCSR*r*[EDS] = 1).

Note: The cycle set {4n+2} = {2,6,10,...} is assigned to the receive buffer only.

The cycle set {4n} = {0,4,8,12,...} is assigned to both buffers.

Behavior in static segment

In this case, both message buffers are assigned to a slot *S* in the *static* segment.

The configuration of a transmit buffer for a static slot *S* assigns this slot to the node as a transmit slot. The FlexRay protocol requires:

- When a slot occurs, if the slot is assigned to a node on a channel that node must transmit either a normal frame or a null frame on that channel. Specifically, a null frame will be sent if there is no data ready, or if there is no match on a transmit filter (cycle counter filtering, for example).

Regardless of the availability of data and the cycle counter filter, the node will transmit a frame in the static slot *S.* In any case, the result of the message buffer search will be the transmit message buffer *t.* The receive message buffer *r* will not be found, no reception is possible.

Behavior in dynamic segment

In this case, both message buffers are assigned to a slot *S* in the *dynamic* segment. The FlexRay protocol requires:

- When a slot occurs, if a slot is assigned to a node on a channel that node only transmits a frame on that channel if there is data ready and there is a match on relevant transmit filters (no null frames are sent).

The transmission of a frame in the dynamic segment is determined by the availability of data and the match of the cycle counter filter of the transmit message buffer.

Transmit data not available

If transmit data are **not available**, that is, the transmit buffer is not committed FR_MBCCSR*t*[CMT] = 0 and/or locked FR_MBCCSR*t*[LCKS] = 1,

a) for the cycles in the set {4n}, which is assigned to both buffers, the receive buffer will be found and the node can receive data, and

b) for the cycles in the set {4n + 2}, which is assigned to the receive buffer only, the receive buffer will be found and the node can receive data.

The receive cycles are shown in *Figure 866*

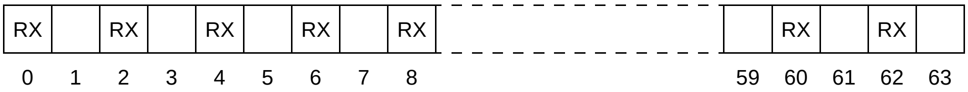


Figure 866. Transmit data not available

Transmit data available

If transmit data are **available**, that is, the transmit buffer is committed FR_MBCCSR*t*[CMT] = 1 and not locked FR_MBCCSR*t*[LCKS] = 0,

a) for the cycles in the set {4n}, which is assigned to both buffers, the transmit buffer will be found and the node transmits data.

b) for the cycles in the set {4n + 2}, which is assigned to the receive buffer only, the receive buffer will be found and the node can receive data.

The receive and transmit cycles are shown in *Figure 866*.

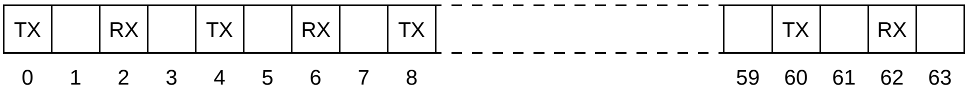


Figure 867. Transmit data not available

32 Power Management Controller (PMC)

32.1 Introduction

Internally, SPC564A70 devices have four supply voltages, nominally 5 V, 3.3 V, 1.2 V and 1.0 V V_{STBY}. Externally only a 5 V supply is required. The other voltages are supplied by internal regulators. All supply voltages have voltage monitors and both the V_{DD} regulator and all monitors are adjustable. The PMC controls the internal voltage supplies, with the exception of V_{STBY}, which has its own regulator. Additionally the PMC controls the low voltage inhibit (LVI) circuits and power-on reset (POR) functions.

Note that although SPC564A70 devices have features intended for use in low-power applications, reduced power modes are achieved by clock gating. It is not possible to switch off the power to any on-chip module. Reduced power consumption is achieved by turning off the system clock to the modules. See *Chapter 5: Operating Modes and Clocking* for details.

The power management controller contains circuitry to generate the internal 3.3 V supply and to control the regulation of 1.2 V supply with external npn ballast transistor. It also contains low voltage inhibit (LVI) and power-on reset (POR) circuits for the 1.2 V supply, the 3.3 V supply, the 3.3 V/5 V supply of the closest I/O segment (VDDEH1) and the 5 V supply of the regulators (VDDREG). There is no requirement for special power up or down sequencing. VDDREG can be tied to VSS to bypass the 3.3 V and 1.2 V internal regulators.

Note: Although it is possible to bypass the internal regulators by tying the VDDREG to ground (VSS), doing so also disables most PMC functions, e.g., the user-programmable LVIs.

Regulators and power supply LVI/POR control blocks use a precision bandgap voltage reference.

A low impedance buffered version of the absolute and curvature corrected bandgap voltage reference is available to be measured using an ADC dedicated channel. The PMC block has the following operating modes:

- Normal mode
- Low power RAM test
- VDDREG supply grounded (See *Section 32.4, Functional description* for details)

32.1.1 Block diagram

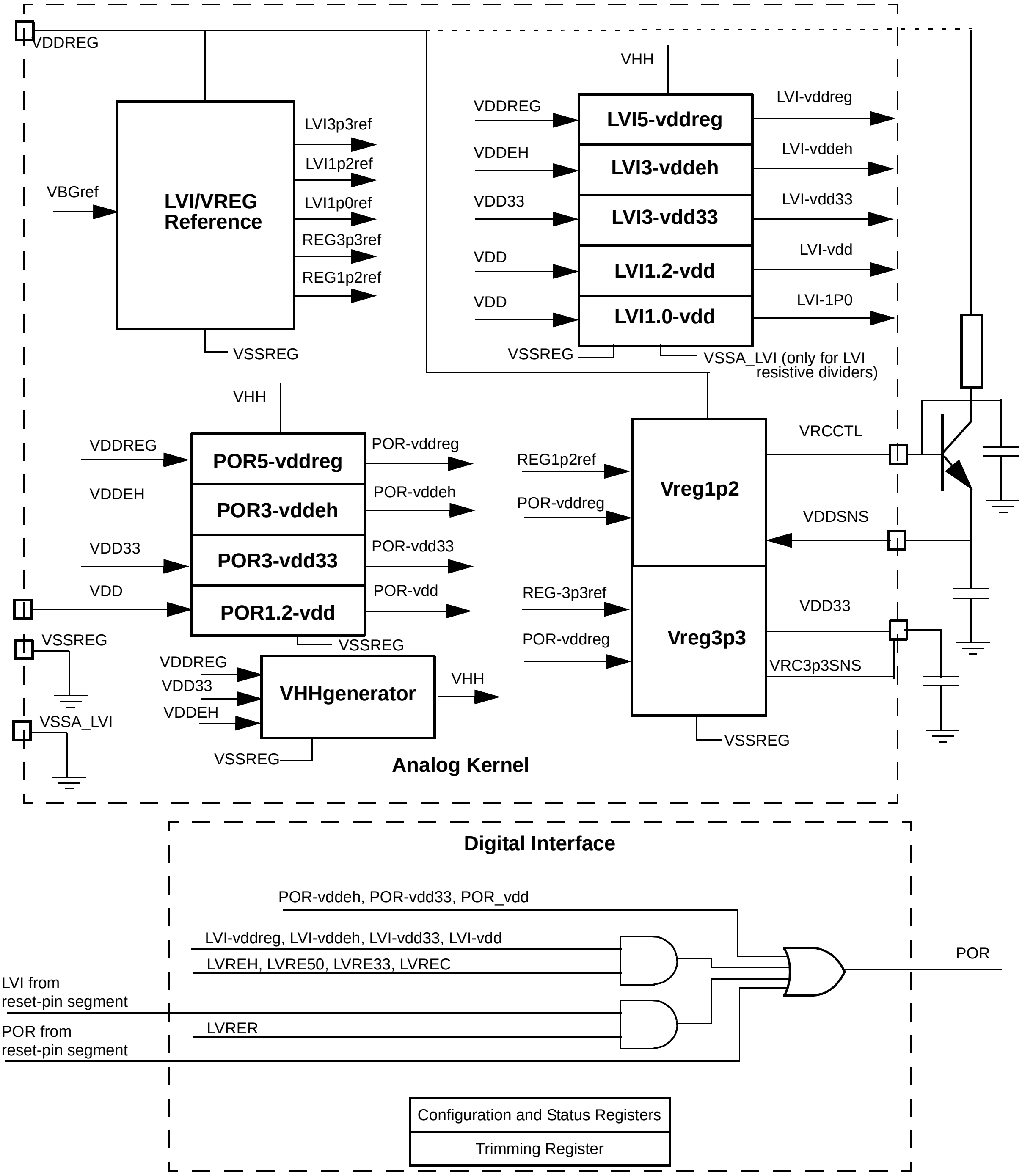


Figure 868. Power management controller diagram

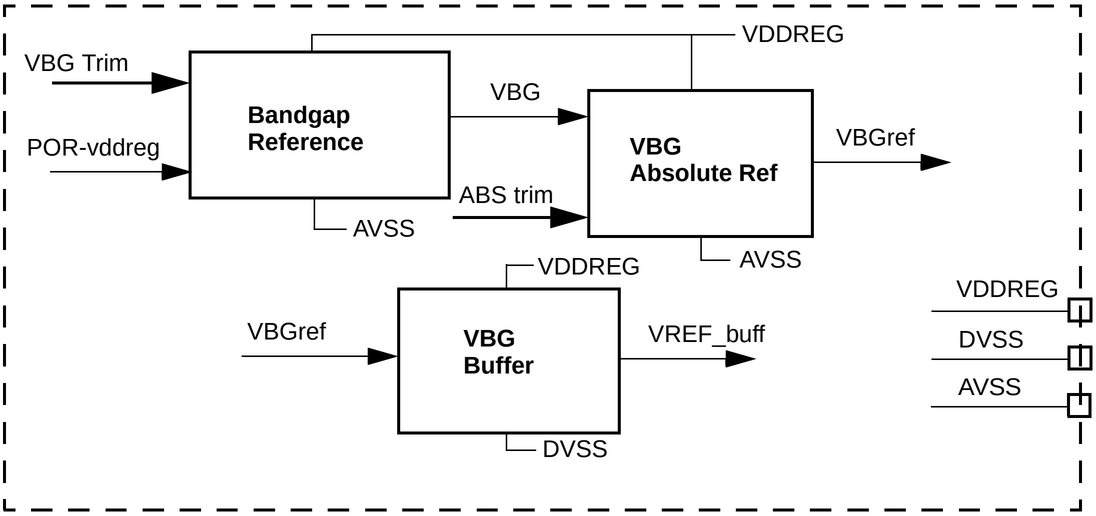


Figure 869. Bandgap reference block diagram

32.2 External signal description

Table 840 provides an overview of the PMC supply signals.

Table 840. Power management controller external signals (maximum ratings)

Name	Type	Voltage	Description
VDDREG	Supply	4.5 – 5.5 V	Power supply for the voltage regulator
VDDEH1	Supply	2.7 – 5.5 V	Power supply of the closest I/O segment
VRC33[1], [2]	Supply	3.3 – 3.6 V	3.3 V bypass capacitor or 3.3 V external power supply
VDD	Supply	1.2 – 1.32 V	1.2 V supply from external ballast transistor
VSSREG	Ground	—	Ground supply for digital core and PMC
VRCCTL	Output	—	Regulator drive for external npn base

1. This table represents the maximum variation of the supply when internal regulators are used. For external power supply requirements check the PMC electrical specifications in the “PMC Operating Conditions and External Regulators Voltage” table in the device datasheet.

2. Within the PMC block, the signal VDD33 is the output of the internal 3.3 V voltage regulator and the signal VRC33 is used as the feedback of the internal 3.3 V voltage regulator. These signals are shorted together on the production package. Throughout this document, references to VRC33 and VDD33 refer to the production package signal VRC33.

32.2.1 Detailed signal descriptions

This section provides the descriptions of external signals coming into and going out of the power management control unit.

VDDREG

Quiet 5 V supply for the voltage regulator and LVI block. It must have an external decoupling capacitor of the order of 4.7 µF – 20 µF. Regulators and LVI can be turned off by grounding VDDREG. In this case external regulation and low voltage control must be supplied.

VDDEH1

Power supply input (5 V or 3.3 V nominal), taken from one of the pad ring I/O segment which is near the voltage regulator.

VRC33

When the internal 3.3 V voltage regulator is enabled, this pin must be connected to an external bypass capacitor of 600 nF – 2 µF with low ESR (max 50 mΩ). If the voltage regulator is not powered or the regulator is disabled (pin VDDREG to ground or shutdown bit PMC_SR[V33DIS] set to ‘1’), this pin must be connected to an external 3.3 V supply.

VDD

This is the 1.2 V supply coming from the emitter of an external NPN ballast transistor, whose base current is supplied by VRCCTL. If the internal voltage regulator controller is not powered (pin VDDREG tied to ground) or the external ballast transistor is not present, the VDD pin must be connected to an external 1.2 V power supply.

For maximum transient performance, the recommended bypass capacitor for each pin that supplies the digital core is 2.2 µF – 6 µF with very low ESR (max 50 mΩ). A ceramic capacitor is also desirable, with 100 nF capacitance. Moreover, a 1 µF to 2 µF cap might be connected to the base of the external bipolar.

VRCCTL

1.2 V regulator output that drives the base of the external NPN transistor.

32.3 Memory map/register definition

Table 841 shows the PMC memory map. The PMC memory map has three registers for configuring, monitoring, and trimming the LVI monitors.

Note: The PMC base address (PMC_BASE) is 0xC3FB_C000. Register addresses in this chapter are given as offsets to PMC_BASE.

Table 841. Power management controller memory map

Address	Register	Location
PMC_BASE (0XC3FB_C000) + 0x0000	MCR — Module Configuration Register	*on page 32-1488*
PMC_BASE (0XC3FB_C000) + 0x0004	TRIMR — Trimming Register	*on page 32-1490*
PMC_BASE (0XC3FB_C000) + 0x0008	SR — Status Register	*on page 32-1493*

32.3.1 Module Configuration Register (MCR)

The configuration register contains configuration and reset and interrupt enable bits for the LVI monitors.

Please note that in the SPC564A70 devices the LVI reset is equivalent to a POR. After an LVI reset, bit SIU_RSR[PORS] is set and the PMC_SR is reset—no LVI event information is retained. To enable application to report an LVI event, disable the reset using the appropriate field, i.e., LVRER or LVREH, and enable the interrupt for the LVI using its interrupt enable field, e.g., LVIER. You must make sure that the ISR will be finished before the voltage sinks below its functional specification and this may require an increase in the LVI level.

The software system reset and the POR/LVI reset are handled differently by the device. See *Section 4.5, Reset source descriptions* for more details. After a software system reset different status flags are set in the SIU_RSR register.

Note: LVI resets and interrupts are only enabled when the voltage regulator is enabled (VDDREG = 5 V). If the user grounds VDDREG (VDDREG = 0 V) and supplies the voltages externally (1.2 V and 3.3 V), it is also necessary to provide the LVI monitoring externally.

Figure 870. Module Configuration Register (MCR)

Offset: PMC_BASE + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	LVRER	LVREH	LVRE50	LVRE33	LVREC	0	0	0	LVIER	LVIEH	LVIE50	LVIE33	LVIC	0	0	TLK
W																
Reset	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 842. MCR field descriptions

Field	Description
0 LVRER	Reset-pin-supply LVI (VDDEH6) LVI reset enable This bit defines whether an LVI assertion on the VDDEH6 supply will generate system reset or not. 0 No reset—LVI assertion on the supply of the I/O segment that contains the reset pin does not cause system reset. 1 Reset—LVI assertion on the supply of the I/O segment that contains the reset pin causes system reset.
1 LVREH	VDDEH1 LVI reset enable This bit defines whether an LVI assertion on the VDDEH1 supply will generate system reset or not. 0 No reset—LVI assertion on the VDDEH supply does not cause system reset. 1 Reset—LVI assertion on the VDDEH supply causes system reset.

Table 842. MCR field descriptions (continued)

Field	Description
2 LVRE50	5 V LVI reset enable This bit defines whether an LVI assertion on the 5 V supply of the voltage regulator (VDDREG) will generate system reset or not. 0 No reset—LVI assertion on the 5 V supply of the voltage regulator does not cause system reset. 1 Reset—LVI assertion on the 5 V supply of the voltage regulator causes system reset.
3 LVRE33	3.3 V LVI reset enable This bit defines whether an LVI assertion on the 3.3 V supply will generate system reset or not. 0 No reset—LVI assertion on the 3.3 V supply does not cause system reset. 1 Reset—LVI assertion on the 3.3 V supply causes system reset.
4 LVREC	1.2 V LVI reset enable This bit defines whether an LVI assertion on the 1.2 V supply will generate system reset or not. 0 No reset—LVI assertion on the 1.2 V supply does not cause system reset. 1 Reset—LVI assertion on the 1.2 V supply causes system reset.
5:7	Reserved
8 LVIER	Reset-pin-supply (VDDEH6) LVI enable This bit enables the generation of the LVI interrupt request when the VDDEH6 LVI falls below the corresponding LVI threshold. The LVI interrupt is independent from LVI reset. If both, interrupt and reset, are enabled, then reset and interrupt will be generated, but reset will then clear the interrupt. 0 Disabled—LVI interrupt request is disabled. 1 Enabled—LVI interrupt request is enabled.
9 LVIEH	VDDEH1 LVI enable This bit enables the generation of the LVI interrupt request when the VDDEH1 supply falls below the corresponding LVI threshold. The LVI interrupt is independent from LVI reset. If both, interrupt and reset, are enabled, then reset and interrupt will be generated, but reset will then clear the interrupt. 0 Disabled—LVI interrupt request is disabled. 1 Enabled—LVI interrupt request is enabled.
10 LVIE50	5 V LVI enable This bit enables the generation of the LVI interrupt request when the 5 V supply of the voltage regulator (VDDREG) falls below the corresponding LVI threshold. The LVI interrupt is independent from LVI reset. If both, interrupt and reset, are enabled, then reset and interrupt will be generated, but reset will then clear the interrupt. 0 Disabled—LVI interrupt request is disabled. 1 Enabled—LVI interrupt request is enabled.
11 LVIE33	3.3 V LVI enable This bit enables the generation of the LVI interrupt request when the 3.3 V power supply goes below the corresponding LVI threshold. The LVI interrupt is independent from LVI reset. If both, interrupt and reset, are enabled, then reset and interrupt will be generated, but reset will then clear the interrupt. 0 Disabled—LVI interrupt request is disabled. 1 Enabled—LVI interrupt request is enabled.
12 LVIC	1.2 V LVI enable This bit enables the generation of the LVI interrupt request when the 1.2 V power supply goes below the corresponding LVI threshold. The LVI interrupt is independent from LVI reset. If both, interrupt and reset, are enabled, then reset and interrupt will be generated, but reset will then clear the interrupt. 0 Disabled—LVI interrupt request is disabled. 1 Enabled—LVI interrupt request is enabled.

Table 842. MCR field descriptions (continued)

Field	Description
13–14	Reserved, should be cleared
15 TLK	Trimming lock This is a set-only bit that comes out of reset negated, and can be asserted one time after reset to lock the trimming register. Once asserted, it cannot be negated anymore. When TLK is asserted, the Trimming Register becomes read-only and cannot be changed until the next reset. 0 Trimming register can be written. 1 Trimming register is read-only.
16:31	Reserved

32.3.2 Trimming Register (TRIMR)

The trimming register enables the user to fine tune the voltage of the regulators and the LVI thresholds. It can only be written when bit PMC_MCR[TLK] is negated. Once PMC_MCR[TLK] has been asserted, this register becomes read-only until the next system reset.

Figure 871. Trimming Register (TRIMR)

Offset: PMC_BASE + 0x0004 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	LVDREGTRIM			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	VDD33TRIM				LVD33TRIM				VDDCTRIM				LVDCTRIM			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 843. TRIMR field descriptions

Field	Description
0–11	Reserved, should be cleared

Table 843. TRIMR field descriptions (continued)

Field	Description
12–15 LVDREGTRIM	LVI 5 V trimming This field is used to fine tune the voltage threshold of the 5 V rising LVI, which is used to monitor the VDDREG supply. Nominal configuration: 0111 4.43 V 0110 4.41 V 0101 4.39 V 0100 4.37 V 0011 4.35 V 0010 4.33 V 0001 4.31 V 0000 4.29 V Default. 1111 4.27 V 1110 4.25 V 1101 4.23 V 1100 4.21 V 1011 4.19 V 1010 4.17 V 1001 4.15 V 1000 4.13 V
16–19 VDD33TRIM	VREG 3.3 V trimming This field is used to fine tune the voltage of the 3.3 V regulator. Nominal configuration: 0111 3.60 V 0110 3.57 V 0101 3.54 V 0100 3.51 V 0011 3.48 V 0010 3.45 V 0001 3.42 V 0000 3.39 V Default. 1111 3.36 V 1110 3.33 V 1101 3.30 V 1100 3.27 V 1011 3.24 V 1010 3.21 V 1001 3.18 V 1000 3.15 V

Table 843. TRIMR field descriptions (continued)

Field	Description
20–23 LVD33TRIM	LVI 3.3 V trimming This field is used to fine tune the voltage threshold of the 3.3 V rising LVI, which is used to monitor the 3.3 V regulated output and the VDDEH supply. Nominal configuration: 0111 3.23 V 0110 3.21 V 0101 3.19 V 0100 3.17 V 0011 3.15 V 0010 3.13 V 0001 3.11 V 0000 3.09 V Default. 1111 3.07 V 1110 3.05 V 1101 3.03 V 1100 3.01 V 1011 2.99 V 1010 2.97 V 1001 2.95 V 1000 2.93 V The recommended value for the LVD33TRIM is 0b0011. In the cut 1 device, the register should be written by software.
24–27 VDDCTRIM	VREG 1.2 V trimming This field is used to fine tune the voltage of the 1.2 V regulator. Nominal configuration: 0111 1.42 V 0110 1.40 V 0101 1.38 V 0100 1.36 V 0011 1.34 V 0010 1.32 V 0001 1.30 V 0000 1.28 V Default. 1111 1.26 V 1110 1.24 V 1101 1.22 V 1100 1.20 V 1011 1.18 V 1010 1.16 V 1001 1.14 V 1000 1.12 V

Table 843. TRIMR field descriptions (continued)

Field	Description
28–31 LVDCTRIM	LVI 1.2 V trimming This field is used to fine tune the voltage threshold of the 1.2 V rising LVI. Nominal configuration: 0111 1.30 V 0110 1.28 V 0101 1.26 V 0100 1.24 V 0011 1.22 V 0010 1.20 V 0001 1.18 V 0000 1.16 V Default 1111 1.14 V 1110 1.12 V 1101 1.10 V 1100 1.08 V 1011 1.06 V 1010 1.04 V 1001 1.02 V 1000 1.0 V

32.3.3 Status Register (SR)

The status register contains interrupt flag bits for the LVD monitors.

Figure 872. Status Register (SR)

Offset: PMC_BASE + 0x0008 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	LVFVSTBY	BGRDY	BGTS	0	0	0	0	0	0	0	V33DIS
W														LVFCSTBY		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	LVFR	LVFH	LVF50	LVF33	LVFC	0	0	0
W	LVFCR	LVFCH	LVFC50	LVFC33	LVFCC											
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 844. SR field descriptions

Field	Description
0–4	Reserved
5 LVFVSTBY	Brown out flag This bit indicates that a brown out condition was detected on the RAM standby regulator switch. 0 No brown out detected 1 Brown out detected
6 BGRDY	Bandgap Status1 This read-only bit is asserted when the bandgap circuit has finished its startup procedure during power-up. The LVIs are disabled (output negated) while BGRDY is negated. 0 Bandgap not ready—LVIs disabled 1 Bandgap ready—LVIs enabled
7 BGTS	Bandgap Status 2 This read-only bit stores bandgap temperature status information. 0 Temperature out of range—above 160 °C 1 Temperature in range—below 160 °C
8:12	Reserved
13 LVFCSTBY	Standby-RAM-supply LVF clear This write-only bit is used to clear the low-voltage flag reported by the RAM standby regulator switch. Writing 1 to this bit informs the RAM standby regulator switch to clear LVFVSTBY. Writing 0 has no effect. Reading this bit always returns 0. 0 No effect 1 Clears LVFVSTBY
14	Reserved
15 V33DIS	3.3 V Internal Regulator Shutdown status bit 0 Enabled—Vreg3p3 ON 1 Disabled—Vreg3p3 OFF
16 LVFCR	Reset-pin-supply (VDDEH6) LVI clear This write-only bit is used to clear the LVI interrupt flag associated with the VDDEH6 supply. Writing 1 to this bit clears the LVFR flag. Writing 0 has no effect. Reading this bit always return 0. 0 No effect 1 Clears the LVFR flag
17 LVFCH	VDDEH1 LVI clear This write-only bit is used to clear the LVI interrupt flag associated with the VDDEH1 supply. Writing 1 to this bit clears the LVFH flag. Writing 0 has no effect. Reading this bit always return 0. 0 No effect 1 Clears the LVFH flag
18 LVFC50	5 V LVI clear This write-only bit is used to clear the LVI interrupt flag associated with the 5 V voltage regulator supply (VDDREG). Writing 1 to this bit clears the LVF50 flag. Writing 0 has no effect. Reading this bit always return 0. 0 No effect 1 Clears the LVF50 flag

Table 844. SR field descriptions (continued)

Field	Description
19 LVFC33	3.3 V LVI clear This write-only bit is used to clear the LVI interrupt flag associated with the 3.3 V supply. Writing 1 to this bit clears the LVF33 flag. Writing 0 has no effect. Reading this bit always return 0. 0 No effect 1 Clears the LVF33 flag
20 LVFCC	1.2 V LVI clear This write-only bit is used to clear the LVI interrupt flag associated with the 1.2 V supply. Writing 1 to this bit clears the LVFC flag. Writing 0 has no effect. Reading this bit always return 0. 0 No effect 1 Clears the LVFC flag
21:23	Reserved
24 LVFR	Reset-pin-supply (VDDEH6) LVI flag This read-only bit is the LVI interrupt flag associated with the VDDEH6 supply. It is asserted when the supply falls below the corresponding LVI threshold, and can be cleared by the CPU by writing 1 to the LVFCR bit. If the LVIER bit is also asserted, an LVI interrupt is sent to the CPU. If LVRER is also asserted, a system reset will be generated, which will clear the LVFR flag and negate the interrupt request. 0 No occurrence 1 LVI occurrence detected on the VDDEH6 supply
25 LVFH	VDDEH1 LVI flag This read-only bit is the LVI interrupt flag associated with the VDDEH1 supply. It is asserted when the supply falls below the corresponding LVI threshold, and can be cleared by the CPU by writing 1 to the LVFCH bit. If the LVIEH bit is also asserted, an LVI interrupt is sent to the CPU. If LVREH is also asserted, a system reset will be generated, which will clear the LVFH flag and negate the interrupt request. 0 No occurrence 1 LVI occurrence detected on the VDDEH1 supply
26 LVF50	5 V LVI flag This read-only bit is the LVI interrupt flag associated with the 5 V supply of the voltage regulator. It can be cleared by the CPU by writing 1 to the LVFC50 bit. If the LVIE50 bit is also asserted, an LVI interrupt is sent to the CPU. If LVRE50 is also asserted, a system reset will be generated, which will clear the LVF50 flag and negate the interrupt request. 0 No occurrence 1 LVI occurrence detected on the 5 V supply of the voltage regulator
27 LVF33	3.3 V LVI flag This read-only bit is the LVI interrupt flag associated with the 3.3 V supply. It is asserted when the 3.3 V supply falls below the corresponding LVI threshold, and can be cleared by the CPU by writing 1 to the LVFC33 bit. If the LVIE33 bit is also asserted, an LVI interrupt is sent to the CPU. If LVRE33 is also asserted, a system reset will be generated, which will clear the LVF33 flag and negate the interrupt request. 0 No occurrence 1 LVI occurrence detected on the 3.3 V supply

Table 844. SR field descriptions (continued)

Field	Description
28 LVFC	1.2 V LVI flag This read-only bit is the LVI interrupt flag associated with the 1.2 V supply. It is asserted when the 1.2 V supply falls below the corresponding LVI threshold, and can be cleared by the CPU by writing 1 to the LVFCC bit. If the LVIC bit is also asserted, an LVI interrupt is sent to the CPU. If LVREC is also asserted, a system reset will be generated, which will clear the LVFC flag and negate the interrupt request. 0 No occurrence 1 LVI occurrence detected on the 1.2 V supply
29:31	Reserved

32.4 Functional description

The power management controller provides the base current for an external ballast transistor to generate the 1.2 V supply. It also contains a 3.3 V regulator and several POR / LVI voltage monitors for system supervision.

A 5 V loose tolerance detection circuit (POR 5 V) is used to determine if the supply voltage VDDREG is high enough to guarantee the startup of bandgap voltage reference. As a consequence 5 V and 3 V LVIs can safely startup with reliable output value. The device is kept in reset until 5 V LVI has been cleared, allowing for correct 3.3 V regulator, 1.2 V regulator and 1.2 V LVI startup. Internal 3.3 V and 1.2 V POR are included to provide minimum low voltage reset capability.

Internal 3.3 V and 1.2 V POR are included to provide minimum low voltage reset capability.

The voltage regulators can be disabled to support the connection of external power supplies to the 3.3 V and 1.2 V pins. It can be done in two ways:

1. Grounding VDDREG

   Internal regulators are automatically disabled, external bipolar transistor is omitted. In this case, all LVI monitors are also disabled. Therefore, when using external power supplies and grounding VDDREG, the user has to provide external LVI monitoring.

2. Keeping VDDREG powered

   External bipolar transistor is omitted, 3.3 V internal regulator can be disabled after startup by clearing the V33DIS bit in the NVUSRO register (Non Volatile User Option register in the flash memory. See *Section 32.4.3, 3.3 V internal voltage regulator* for more details.). In this case there is no need of external LVI monitoring.

As PORs are powered by VHH their value is reliable also when VDDREG is grounded, as long as VDDEH1 or VDD33 are at the correct voltage.

32.4.1 Bandgap

The bandgap voltage of the PMC is capable of generating a reference voltage of 1.219 V. It is used as reference to generate all supply voltages, for this reason it is powered directly out of the 5 V domain to avoid startup racing conditions. Supply voltage range is from 4.5 V to 5.5 V. The bandgap shall work with decreased performance down to 4.0 V supply.

The bandgap is calibrated during factory test—see the “PMC Electrical Characteristics” table in the device datasheet for accuracy details. The calibration data is typically stored in flash memory and is automatically read from the flash every time the part is initiated through reset. The default LVI thresholds are set to a safe value that accommodates the full uncalibrated bandgap range, so that external voltages can be applied in the specified ranges—see the “PMC Electrical Characteristics” table in the device datasheet for details.

The bandgap has two monitor signals: bandgap_OK and temperature_OK. The former is low during startup and rises when bandgap reference has settled (uses an internal POR on the bandgap supply); the latter is a low accuracy temperature sensor, which goes low when an over temperature condition has occurred, i.e., when local temperature is higher than 160 – 200 °C. Both signals and POR value are available in the status register (PMC_SR) as bandgap status bits.

Bandgap is powered by the same supply of the voltage regulator. When voltage regulator supply is removed (to allow for external powering of 3.3 V and 1.2 V domains), bandgap voltage reference will also be disabled.

32.4.2 5 V LVI

A programmable low voltage monitor is connected internally to the 5 V supply that feeds the voltage regulator (VDDREG). The output of the LVI goes to logical 1 when the monitored voltage rises above the LVI rising trip point. In case the monitored voltage falls below the falling trip point, the LVI output goes to logical 0.

The assertion and negation voltages are adjustable via software by writing to field PCM_TRIMR[LVDREGTRIM], which selects one of the 16 voltages available. The reset value of the 4-bit register is “0000”, corresponding to the rising trip point voltage of 4.29 V. This is the typical default value, the real default value may vary from sample to sample according to process, temperature and voltage supply conditions, as detailed in the “PMC Electrical Characteristics” table in the device datasheet.

32.4.3 3.3 V internal voltage regulator

The 3.3 V internal voltage regulator supplies a total DC current of up to 80 mA or a maximum transient current peak up to 300 mA (if the external decoupling capacitor has the recommended value of 600 nF – 2 µF and ESR < 50 mΩ).

The regulator is powered by VDDREG and works in the range 4.5 V to 5.5 V, down to 4.0 V with lower current drive capabilities, and uses the bandgap voltage as absolute reference. The 3.3 V regulator output is connected to an I/O pad (VRC33) for external capacitance decoupling, then all blocks (except the flash) that need a 3.3 V supply are connected to the VRC33 pad. The 3.3 V flash power supply is taken at a pad next to the regulator I/O, which is double bonded together with the VRC33 pad.

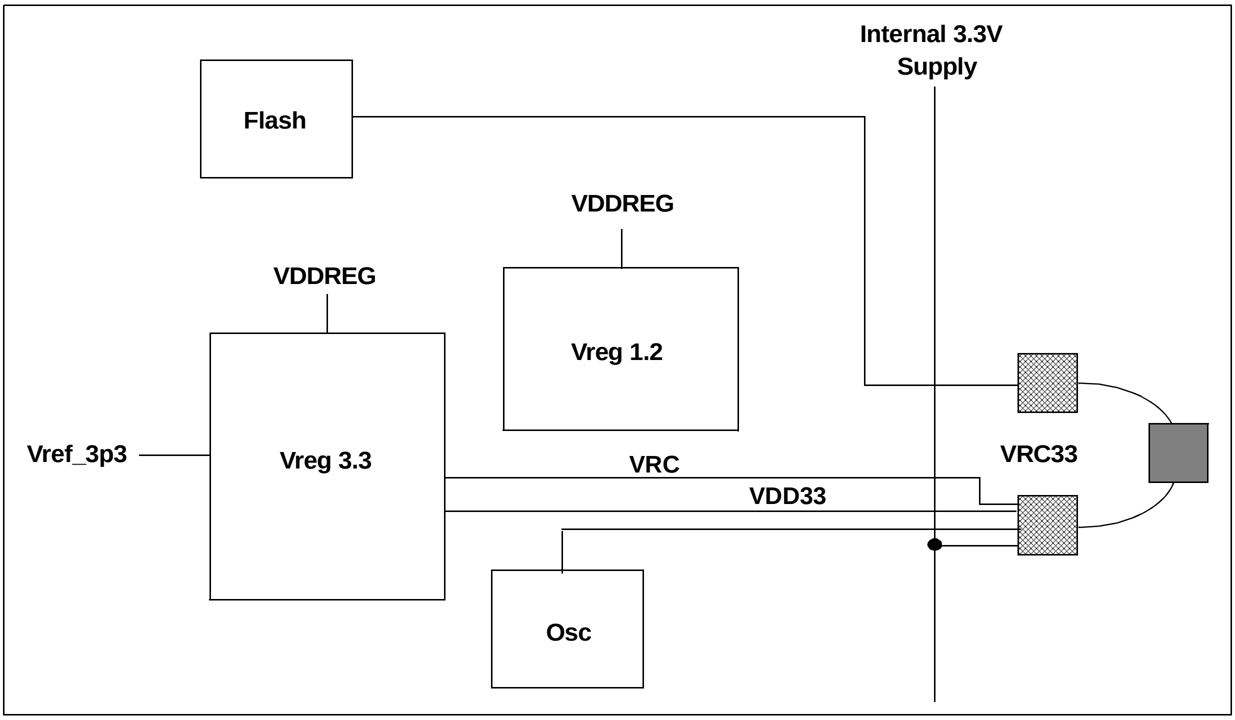


Figure 873. Vreg 3.3 V power connection

The regulator output voltage is adjustable via software by writing to field PCM_TRIMR[VDD33TRIM], which selects one of the 16 voltages available. The reset value of the 4-bit register is “0000”, corresponding to nominal voltage of 3.39 V. This is the typical default value, the real default value may vary from sample to sample according to process, temperature and voltage supply conditions, as detailed in the “PMC Electrical Characteristics” table in the device datasheet.

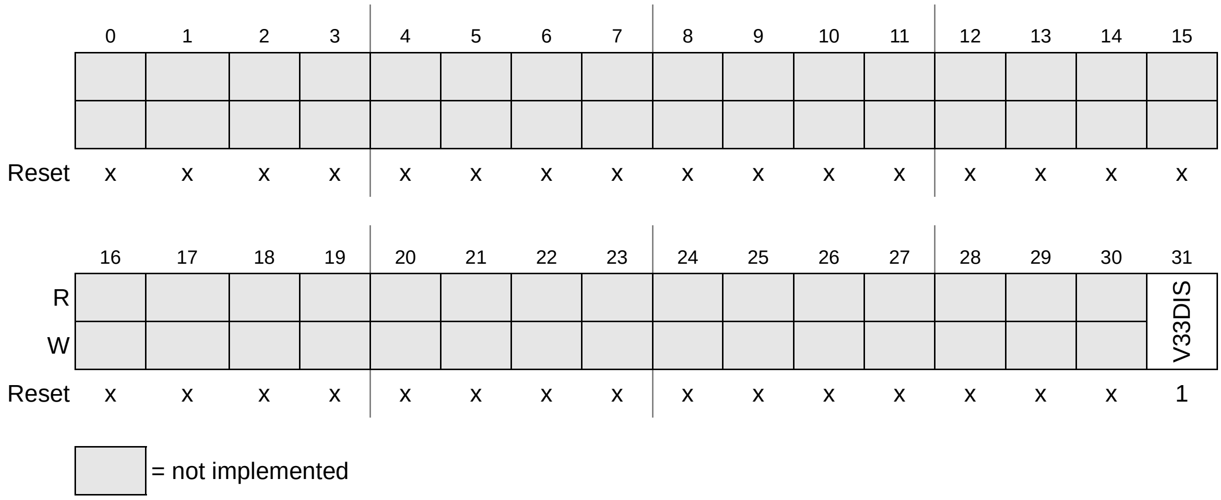
The 3.3 V supply is internally connected to a 5 V ADC channel such that the actual voltage may be read. Moreover, if an external 3.3 V voltage source will be used, the user can disable this regulator by clearing the shutdown bit NVUSRO[V33DIS]. The user might have both internal 3.3 V regulator and external 3.3 V voltage sources at the same time at the cost of additional power consumption on the external voltage source. It is recommended to disable the internal 3.3 V regulator in this case.

Additional features of this regulator include a current limiter that protects a PMOS pass device from overload condition and soft start-up to avoid overshoot during power-on.

If an external 3.3 V supply will be used, this regulator can be disabled by setting vdd33shutdown bit to logical 1. The bit resides in the NVUSRO register in the flash memory shadow row. See *Figure 874* and *Table 845* for details.

Figure 874. Non-Volatile User Options Register (NVUSRO) - Array0

Address: 0x00FF_FE10



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Reset	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R																V33DIS
W																
Reset	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	1

▨ = not implemented

Table 845. Non-Volatile User Options Register (NVUSRO) field description

Field	Description
0:30	Reserved
31 V33DIS	VREG33 shutdown bit 1: 3.3 V regulator is enabled. 0: 3.3 V regulator is disabled. The shutdown bit is masked until the 1.2 V achieves its POR trip point, keeping the 3p3 regulator on. After a POR the reset value reflects the state of the shadow flash bit at 0x00FF_FE10.

32.4.4 3.3 V LVI

The PMC contains two 3.3 V low voltage monitors. One is connected to the internal 3.3 V supply and the other is connected to VDDEH1.

The output of the LVI goes to logical 1 when the monitored voltage rises above the rising trip point. In case the monitored voltage falls below the falling trip point, the LVI output goes to logical 0. The assertion and negation voltages are adjustable via software by writing to field PCM_TRIMR[LVD33TRIM], which selects one of the 16 voltages available. The reset value of the 4-bit register is “0000”, corresponding to rising trip point voltage of 3.09 V. This is the typical default value, the real default value may vary from sample to sample according to process, temperature and voltage supply conditions, as detailed in the “PMC Electrical Characteristics” table in the device datasheet.

The LVIs can be programmed to trigger power-on reset (enabled by default). If programmed to generate reset, the monitors are able to hold reset from 3.3 V POR until greater or equal to LVI threshold.

32.4.5 1.2 V voltage regulator controller

A voltage regulator controller is used to source current to the base of an external NPN transistor which operates as an emitter follower. The 1.2 V supply is produced at the emitter of this transistor with a maximum DC current of 400 mA (or maximum transient current of 1.2 A).

The regulator output voltage is adjustable using software, to permit the device to center the supply for maximum transient margin. This adjustment is achieved by writing to field PCM_TRIMR[VDDCTRIM], which selects one of the 16 voltages available. The reset value of the 4-bit register is “0000”, corresponding to a typical default voltage of 1.28 V.

The 1.2 V supply can be internally connected internally to a 5 V ADC channel such that the actual voltage may be read.

32.4.6 1.2 V LVI

A programmable low voltage monitor is connected to the 1.2 V supply. The output of the LVI goes to logical 1 when the monitored voltage rises above the rising trip point. In case the monitored voltage falls below the falling trip point, the LVI output goes to logical 0.

The assertion and negation voltages are adjustable via software by writing to field PCM_TRIMR[LVDCTRIM], which selects one of the 16 voltages available. The reset value of the 4-bit register is “0000”, corresponding to a rising trip point voltage of 1.16 V. This is the typical default value, the real default value may vary from sample to sample according to process, temperature and voltage supply conditions, as detailed in the “PMC Electrical Characteristics” table in the device datasheet.

The LVI 1.2 V can be programmed to trigger power-on reset (enabled by default). If programmed to generate reset, the monitor is able to hold reset from 1.2 V POR until greater or equal to LVI threshold.

32.4.7 Resets and interrupts

Power-on reset

Power-on reset (POR) circuits are present at the following power supplies:

- 3.3 V / 5 V supply of the closest I/O segment (VDDEH1);
- 5 V supply of the PMC block and bandgap (VDDREG);
- 3.3 V regulated supply VDD33;
- 1.2 V regulated supply.

Power-on reset will assert when the voltage levels of the POR power supplies begin to rise. Each POR will negate when its power supply rises into the specified range. The behavior for each POR during power supply ramping is shown in *Figure 875*.

The dependence between POR and LVI is summarized in *Figure 876*. As shown, the LVI will reach a consistent state before the POR actually releases the reset, avoiding false startup condition. This is valid for each voltage supply monitored by POR/LVI.

The PORs for each power supply are not intended to indicate that the power supply has dropped below the specified voltage range for the device. The 1.2 V and the 3.3 V supplies are monitored, respectively, by the LVI 1.2 V and LVI 3.3 V circuits for this purpose.

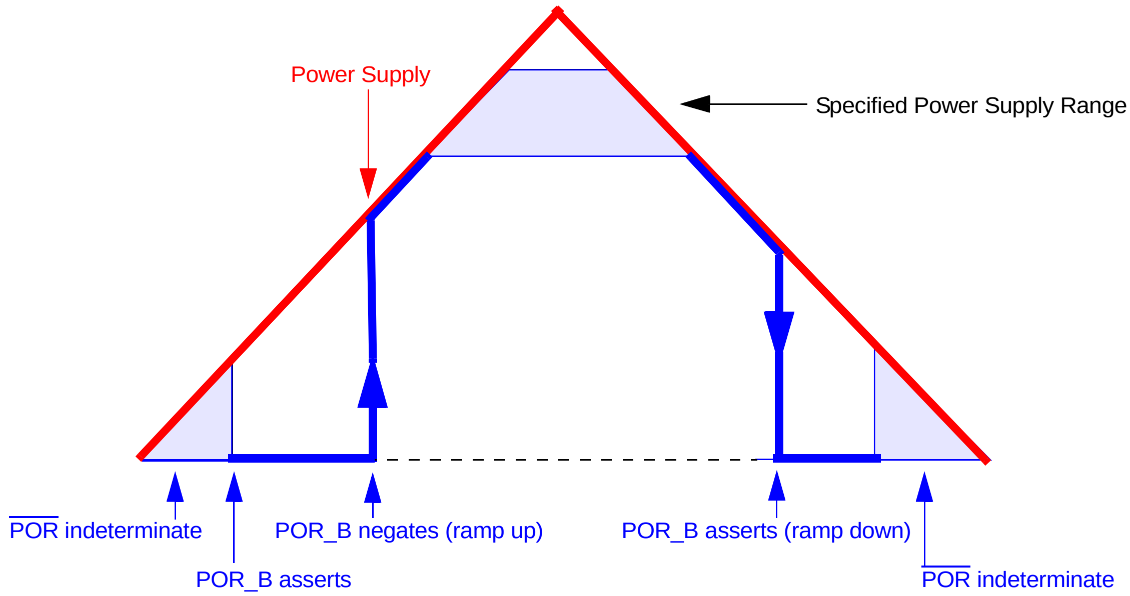


Figure 875. POR rising and falling edges

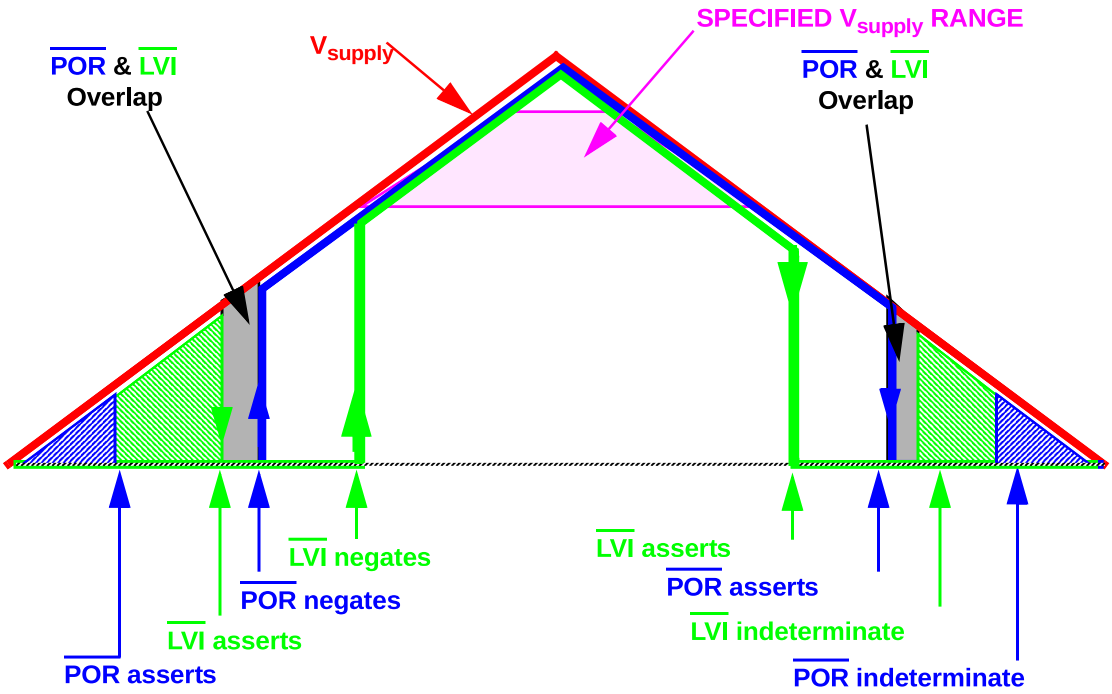


Figure 876. POR - LVI relative rising and falling edges

The LVI monitors can be configured to generate power-on reset by programming the LVRER, LVREH, LVRE50, LVRE33 and LVREC bits in the MCR register.

The combination of POR and LVI sources within the PMC generates a single power-on reset output signal which can be distributed throughout the device.

The following sections discuss the various modules and functions that use the POR.

Clock control

The clock control divides the system clock to generate CLKOUT. Because CLKOUT toggles during system reset, one of the sources of reset for the dividers is POR.

SIU

The SIU uses POR in its reset controller state machine, synchronizers and $\overline{\text{RESET}}$ filter, SIU_RSR, SIU_RCSR, SIU_CCR and an internal register.

Reset controller state machine

Because the reset controller state machine is active during system reset, it is reset with POR.

Synchronizers and $\overline{\text{Reset}}$ filter

Signals affecting the reset controller’s ability to negate system reset need to be synchronized with synchronizers reset by POR if they are asynchronous or come from clock domains other than the system clock.

A synchronized and filtered assertion of the $\overline{\text{RESET}}$ pin will hold the device in system reset. The synchronizers and filter are reset with POR so that $\overline{\text{RESET}}$ appears to be asserted while POR is asserted. Synchronizers for the watchdog, Nexus, checkstop, and JTAG sources of reset use POR to reset the synchronizers, but system reset can suffice because those sources are not analyzed until after system reset negates. The loss of clock source synchronizers also are reset with POR, but since loss of clock results in loss of lock, which needs and uses POR as reset, system reset can suffice for loss of clock.

The synchronization of WKPCFG and BOOTCFG uses POR. In the case of WKPCFG, the pin is applied during POR. In the case of BOOTCFG, the pin value is latched before the negation of system reset. Changes in BOOTCFG have no effect after the negation of system reset.

SIU_RSR

Because POR sets the PORS bit of the SIU_RSR to indicate that POR was the source of reset, the other source indicators, ERS, LLRS, LCRS, WDRS, CRS, and SSRS, are reset with POR. The WKPCFG and BOOTCFG bits are reset with POR so that the WKPCFG and BOOTCFG values can be latched at the negation of system reset. The RGF bit also is reset with POR. The SERF bit is reset with POR, but also is cleared during system reset. System reset can suffice to reset the SERF bit.

SIU_SRCR

The CRE bit of the SIU_SRCR is reset with POR. The SSR and SER bits are reset with POR, but system reset can suffice for both of them.

SIU_CCR

The MATCH and DISNEX bits of the SIU_CCR use POR.

Flash memory

The flash state machine is reset during POR. VFLASH must be at a high enough voltage to read the shadow row before system reset negates. $\overline{\text{RESET}}$ must be asserted when

VFLASH is below the minimum specification. Since the synchronized and filtered RESET appears as asserted during POR and then must remain asserted until VFLASH is within specification, it is used to reset the flash state machine.

When the system reset is caused by other sources besides POR or RESET assertion, the flash state machine uses the system reset indication as an input, but not as a reset, to indicate that the state machine is to read the shadow row.

Note: the field LVD33TRIM of the register TRIMR must be programmed with “0011” at least, in order to enhance the LVI33 threshold by 60 mV and monitor the VDD33 Voltage in all the corners (voltage, process and temperature). This is to ensure that the voltage never becomes lower than 3.0 V in order to guarantee the operations on the flash.

FMPLL

The FMPLL analog hard block is held in power down state during POR to guarantee that it starts operating only when voltages are high enough to allow its operation.

The FMPLL programmer’s model registers are reset with POR so that the FMPLL does not lose lock every time a system reset is asserted.

The Clock Quality Monitor (CQM) inside the FMPLL uses POR to initialize its registers and counters because it operates during system reset to detect when the crystal oscillator has stabilized.

NPC

The NPC uses POR because it can be configured during system reset.

JTAGC

The assertion of POR is equivalent to the negation of the JCOMP pin.

e200z4

The e200z4 is reset with POR or system reset except in some portions of its Nexus interface, which only are reset with POR.

Padring

The padring 3-states all of the output pins, including CLKOUT, MCKO, and RSTOUT when the 1.2 V supply is too low to propagate internal signals, including the POR indication. When the POR indication can be propagated, the output pins, including CLKOUT, MCKO, and RSTOUT, also are 3-stated during POR.

During POR, the actual value of the pin cannot be read. Instead, the padring drives an input value. The actual value during POR is important for only two pins: WKPCFG and PLLREF. The values driven during POR for all other pins are irrelevant. For those pins, the input value is a 0.

WKPCFG

During POR, the direction of the weak pull is determined by the reset state of the WPS bit in the SIU_PCRs. For those pins whose WPS reset state is determined by the WKPCFG pin, the value of WKPCFG is treated as a 1 during POR to be consistent with the default pull up for the WKPCFG pin. Therefore, those pins whose WPS reset state is determined by WKPCFG will have a pull up during POR.

PLLREF

The PLLREF pins selects whether crystal or external clock is used as clock source in bypass mode, which is the default mode out of POR. Furthermore, PLLREF selects whether the clock reference is monitored or not by the Clock Quality Monitor. If the reference is the crystal oscillator, it is monitored. If the reference is an external clock, it is not monitored.

The PLLREF value during POR is kept at logic level 0 to minimize the probability of a clock glitch in the more stringent case when PLLREF = 0, therefore the CQM will not monitor the reference clock and the internal POR will be released as soon as the voltages achieve the LVI thresholds. The clock glitch may occur when the POR is released near the external clock falling edge. Even if such a glitch happens, it will be still inside the POR pulse because all synchronous logic that use POR are supposed to synchronize the POR negation with a double-register.

Interrupts

The PMC generates one interrupt request signal for each LVI source: reset-pin-supply (VDDEH6) LVI, VDDEH1 LVI, 5 V LVI, 3.3 V LVI and 1.2 V LVI. The module also generates combined interrupt request signal which is asserted whenever any of the three individual interrupt request signals becomes asserted.

32.4.8 Soft-Start (for 1.2 V and 3.3 V regulators)

A soft-start circuit has been implemented for 1.2 V and 3.3 V regulators. This circuit controls the reference voltage rise time to avoid abrupt ramp-up of these regulators.

32.4.9 ADC test mux

During PMC functional mode it is possible to perform direct measurements through the ADC. PMC internal voltages are routed to the ADC. Each signal can be measured with ADC running at full speed.

Table 846. eQADC test mux channel for internal PMC signals

eQADC channel	ADC	Description
45	ADC0/ADC1	Buffered Band Gap
128	ADC0/ADC1	Temp Sensor
129	ADC0/ADC1	VSSA
144	ADC0	Buffered Band Gap
145	ADC0	Reference Voltage for 1.2 V LVD
146	ADC0	Reference for 1.2 V Regulator
147	ADC0	Reference Voltage for 3.3 V LVD
162	ADC0	VDDA
163	ADC0	50% VDDEH6
164	ADC0	VSSA
165	ADC0	50% VDDEH7
166	ADC0	50% VDDEH4

Table 846. eQADC test mux channel for internal PMC signals (continued)

eQADC channel	ADC	Description
167	ADC0	VSSA
180	ADC0	Reference Voltage for 5.0 V LVD
181	ADC0	Reference Voltage for 3.3 V LVD
182	ADC0	Reference for 3.3 V Regulator
194	ADC1	Test output from Stby PMC
195	ADC1	50% VDDEH1B
196	ADC1	VRC33
197	ADC1	VRC33
198	ADC1	50% VDDEH4
199	ADC1	50% VDDEH1A

For any measurements it is strongly recommended to disable all LVI outputs to the logic via software as the multiplexer toggling could induce false detection.

32.5 Electrical characteristics

For electrical characteristics, please refer to the device datasheet.

33 JTAG Controller (JTAGC)

33.1 Information specific to this device

This section presents device-specific parameterization, customization, and feature availability information not specifically referenced in the remainder of this chapter.

33.1.1 Device-specific parameters

Table 847 shows the parameters and associated values for this device.

Table 847. Device-specific parameters

Parameter	Value
Number of JCOMP bits used	1
Length of the boundary scan chain path for the device	248
Number of auxiliary TAP controllers that share the TAP with the JTAGC via an ACCESS_AUX_TAP_x instruction (not including the JTAGC)	5
Size of the CENSOR_CTRL register (bits)	64

33.1.2 Device identification register parameters

Table 848 shows the fields and values for the Device Identification Register on this device.

Table 848. Device identification register parameters

Field	Value
Part revision number (PRN)	Changes in each revision
Design center code (DC)	0x2B
Part identification number (PIN)	0x203
Manufacturer identity code (MIC)	0x020

33.1.3 Auxiliary TAP controller instructions

Table 849 lists the auxiliary TAP controller instructions (discussed in *Section 33.5.4, JTAGC block instructions*) available on this device.

Table 849. Device-specific auxiliary TAP controller Instructions

Instruction	Code[4:0]	Instruction summary
ACCESS_AUX_TAP_NPC	10000	Enables access to the NPC TAP controller
ACCESS_AUX_TAP_ONCE	10001	Enables access to the e200z4 OnCE TAP controller
ACCESS_AUX_TAP_eTPU	10010	Enables access to the eTPU Nexus TAP controller

33.2 Introduction

Figure 877 is a block diagram of the JTAG Controller (JTAGC) block.

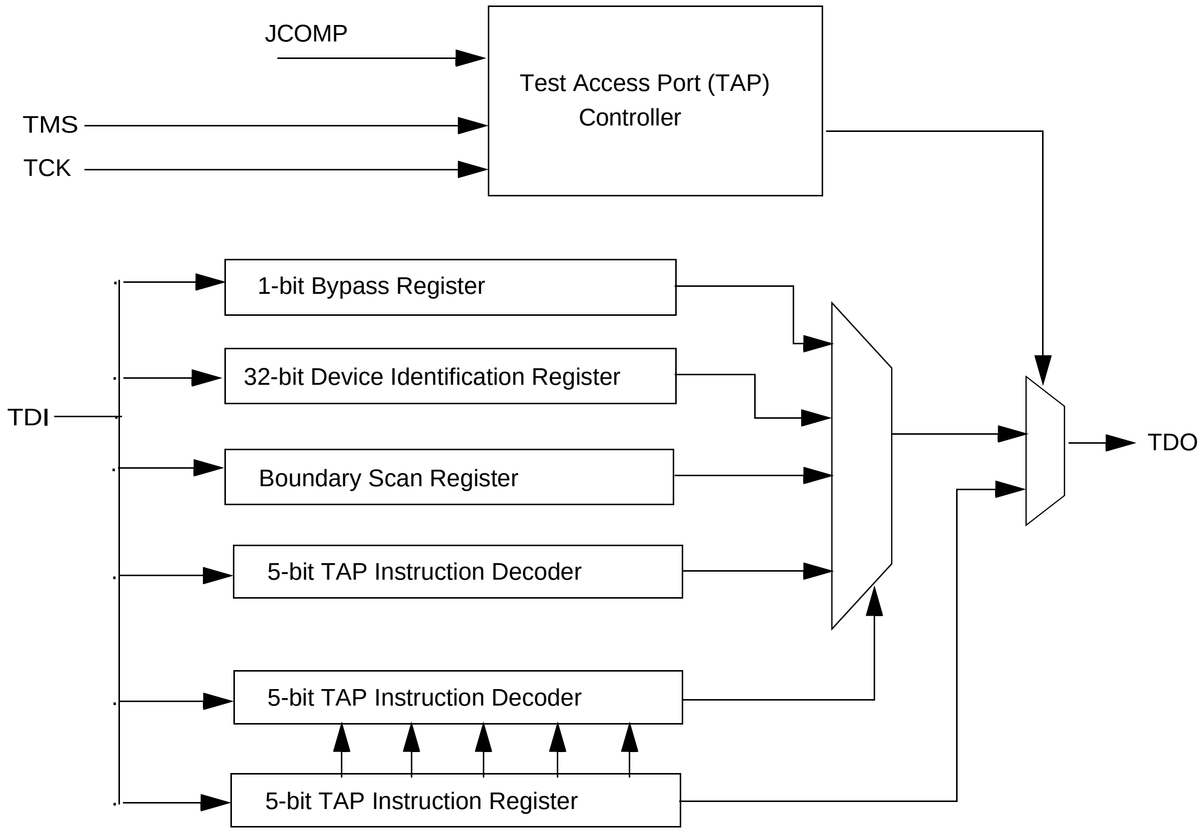


Figure 877. JTAG STL (IEEE 1149.1) block diagram

33.2.1 Overview

The JTAGC block provides the means to test chip functionality and connectivity while remaining transparent to system logic when not in test mode. Testing is performed via a boundary scan technique, as defined in the IEEE 1149.1-2001 standard. All data input to and output from the JTAGC block is communicated in serial format.

33.2.2 Features

The JTAGC block is compliant with the IEEE 1149.1-2001 standard, and supports the following features:

- IEEE 1149.1-2001 Test Access Port (TAP) interface
  - 4 pins (TDI, TMS, TCK, and TDO)
- JCOMP input that provides reset control and the ability to share the TAP
- 5-bit instruction register that supports several IEEE 1149.1-2001 defined instructions as well as several public and private device-specific instructions (Refer to *Table 853* for a list of supported instructions.)
- Sharing of the TAP with other TAP controllers via ACCESS_AUX_TAP_x instructions
- Test data registers: a bypass register, a boundary scan register, and a device identification register
- TAP controller state machine that controls the operation of the data registers, instruction register and associated circuitry

33.2.3 Modes of operation

The JTAGC block uses JCOMP and a power-on reset indication as its primary reset signals. Several IEEE 1149.1-2001 defined test modes are supported, as well as a bypass mode.

Reset

The JTAGC block is placed in reset when either power-on reset is asserted, JCOMP is negated, or the TMS input is held high for enough consecutive rising edges of TCK to sequence the TAP controller state machine into the Test-Logic-Reset state. Holding TMS high for five consecutive rising edges of TCK guarantees entry into the Test-Logic-Reset state regardless of the current TAP controller state. Asserting power-on reset or setting JCOMP to a value other than the value required to enable the JTAGC block results in asynchronous entry into the reset state. While in reset, the following actions occur:

- The TAP controller is forced into the Test-Logic-Reset state, thereby disabling the test logic and allowing normal operation of the on-chip system logic to continue unhindered.
- The instruction register is loaded with the IDCODE instruction.

IEEE 1149.1-2001 defined test modes

The JTAGC block supports several IEEE 1149.1-2001 defined test modes. A test mode is selected by loading the appropriate instruction into the instruction register while the JTAGC is enabled. Supported test instructions include EXTEST, HIGHZ, CLAMP, SAMPLE and SAMPLE/PRELOAD. Each instruction defines the set of data register(s) that may operate and interact with the on-chip system logic while the instruction is current. Only one test data register path is enabled to shift data between TDI and TDO for each instruction.

The boundary scan register is enabled for serial access between TDI and TDO when the EXTEST, SAMPLE or SAMPLE/PRELOAD instructions are active. The single-bit bypass register shift stage is enabled for serial access between TDI and TDO when the HIGHZ, CLAMP or reserved instructions are active. The functionality of each test mode is explained in more detail in *Section 33.5.4, JTAGC block instructions*.

Bypass Mode

When no test operation is required, the BYPASS instruction can be loaded to place the JTAGC block into bypass mode. While in bypass mode, the single-bit bypass shift register is used to provide a minimum-length serial path to shift data between TDI and TDO.

33.3 External signal description

33.3.1 Overview

The JTAGC consists of five signals that connect to off chip development tools and allow access to test support functions. The JTAGC signals are outlined in *Table 850*.

Table 850. JTAG signal properties

Name	I/O	Function	Reset state	Pull[1]
TCK	Input	Test Clock	—	Down
TDI	Input	Test Data In	—	Up
TDO	Output	Test Data Out	High Z[2]	—
TMS	Input	Test Mode Select	—	Up
JCOMP	Input	JTAG Compliancy	—	Down

1. The pull is not implemented in this block. Pullup/pulldown devices are implemented in the pads.
2. TDO output buffer enable is negated when the JTAGC is not in the Shift-IR or Shift-DR states. A weak pull may be implemented at the TDO pad for use when JTAGC is inactive.

33.3.2 Detailed signal descriptions

This section describes each of the signals listed in *Table 850* in more detail.

TCK—Test Clock Input

Test Clock Input (TCK) is an input pin used to synchronize the test logic and control register access through the TAP.

TDI—Test Data Input

Test Data Input (TDI) is an input pin that receives serial test instructions and data. TDI is sampled on the rising edge of TCK.

TDO—Test Data Output

Test Data Output (TDO) is an output pin that transmits serial output for test instructions and data. TDO is three-stateable and is actively driven only in the Shift-IR and Shift-DR states of the TAP controller state machine, which is described in *Section 33.5.3, TAP controller state machine*. The TDO output of this block is clocked on the falling edge of TCK and sampled by the development tool on the rising edge of TCK.

TMS—Test Mode Select

Test Mode Select (TMS) is an input pin used to sequence the IEEE 1149.1-2001 test control state machine. TMS is sampled on the rising edge of TCK.

JCOMP—JTAG compliancy

The JCOMP signal provides IEEE 1149.1-2001 compatibility and provides the ability to share the TAP. The JTAGC TAP controller is enabled when JCOMP is set to the JTAGC enable encoding, otherwise the JTAGC TAP controller remains in reset.

33.4 Register definition

This section provides a detailed description of the JTAGC block registers accessible through the TAP interface, including data registers and the instruction register. Individual bit-level descriptions and reset states of each register are included. These registers are not memory-mapped and can only be accessed through the TAP.

33.4.1 Register descriptions

The JTAGC block registers are described in this section.

Instruction Register

The JTAGC block uses a 5-bit instruction register as shown in *Table 878*. The instruction register allows instructions to be loaded into the block to select the test to be performed or the test data register to be accessed or both. Instructions are shifted in through TDI while the TAP controller is in the Shift-IR state, and latched on the falling edge of TCK in the Update-IR state. The latched instruction value can only be changed in the Update-IR and Test-Logic-Reset TAP controller states. Synchronous entry into the Test-Logic-Reset state results in the IDCODE instruction being loaded on the falling edge of TCK. Asynchronous entry into the Test-Logic-Reset state results in asynchronous loading of the IDCODE instruction. During the Capture-IR TAP controller state, the instruction shift register is loaded with the value 0b10101, making this value the register's read value when the TAP controller is sequenced into the Shift-IR state.

	4	3	2	1	0
R	1	0	1	0	1
W	Instruction Code				
Reset:	0	0	0	0	1

Figure 878. 5-bit Instruction Register

Bypass Register

The bypass register is a single-bit shift register path selected for serial data transfer between TDI and TDO when the BYPASS, CLAMP, HIGHZ or reserve instructions are active. After entry into the Capture-DR state, the single-bit shift register is set to a logic 0. Therefore, the first bit shifted out after selecting the bypass register is always a logic 0.

Device Identification Register

The device identification register, shown in *Figure 879*, allows the revision number, part number, manufacturer, and design center responsible for the design of the part to be determined through the TAP. The device identification register is selected for serial data transfer between TDI and TDO when the IDCODE instruction is active. Entry into the

Capture-DR state while the device identification register is selected loads the IDCODE into the shift register to be shifted out on TDO in the Shift-DR state. No action occurs in the Update-DR state. The part revision number (PRN) and part identification number (PIN) fields are system plugs, and the manufacturer identity code (MIC) is a constant value assigned to the manufacturer by the JEDEC.

The shift register LSB is forced to logic 1 on the rising edge of TCK following entry into the Capture-DR state. Therefore, the first bit to be shifted out after selecting the IDCODE register is always a logic 1. The remaining 31 bits are forced to the value of the device identification register on the rising edge of TCK following entry into the Capture-DR state.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Part Revision Number				Design Center						Part Identification Number					
W	▨				▨						▨					
Reset	PRN				DC						PIN					

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Part Identification Number				Manufacturer Identity Code											1
W	▨				▨											▨
Reset	PIN (cont'd)				—[1]	—	—	—	—	—	—	—	—	—	—	1

▨ = Reserved

1. See *Table 848* for value.

Figure 879. Device Identification Register

Table 851. Device identification register field descriptions

Field	Description
31–28 PRN	Part Revision Number Bits [31:28] contain the revision number of the part.
27–22 DC	Design Center Bits [27:22] indicate the design center.
21–12 PIN	Part Identification Number Bits [21:12] contain the part number of the device.
11–1 MIC	Manufacturer's Identification Code Bits [11:1] contain the JEDEC (Joint Electron Device Engineering Council) manufacturer's identification code.
Bit [0]	IDCODE Register ID Bit [0] identifies this register as the device identification register and not the bypass register

CENSOR_CTRL Register

The CENSOR_CTRL register is a 64-bit shift register path from TDI to TDO selected when the ENABLE_CENSOR_CTRL instruction is active. The default reset value of the CENSOR_CTRL register is 64’b0. The CENSOR_CTRL register transfers its value to a parallel hold register on the rising edge of TCK when the TAP controller state machine is in the Update-DR state. Once the ENABLE_CENSOR_CTRL instruction is executed, the register value will remain valid until a JTAG reset occurs.

	*(1)	...	2	1	0
R W	CENSOR_CTRL				
Reset:	*(2)	*	*	*	*

1. The size of CENSOR_CTRL is 64 bits.
2. The reset value of CENSOR_CTRL is 64’b0.

Figure 880. CENSOR_CTRL Register

Table 852. CENSOR_CTRL register field descriptions

Field	Description
63–0 CENSOR_CTRL [63:0]	Censorship Control The CENSOR_CTRL bits are used to control chiptop censorship functions.

Boundary Scan Register

The boundary scan register is connected between TDI and TDO when the EXTEST, SAMPLE or SAMPLE/PRELOAD instructions are active. It is used to capture input pin data, force fixed values on output pins, and select a logic value and direction for bidirectional pins. Each bit of the boundary scan register represents a separate boundary scan register cell, as described in the IEEE 1149.1-2001 standard and discussed in *Section 33.5.5, Boundary scan*. The size of the boundary scan register and bit ordering is device-dependent.

33.5 Functional description

33.5.1 JTAGC reset configuration

While in reset, the TAP controller is forced into the Test-Logic-Reset state, thus disabling the test logic and allowing normal operation of the on-chip system logic. In addition, the instruction register is loaded with the IDCODE instruction.

33.5.2 IEEE 1149.1-2001 (JTAG) test access port

The JTAGC block uses the IEEE 1149.1-2001 TAP for accessing registers. This port can be shared with other TAP controllers on the MCU. Ownership of the port is determined by the value of the JCOMP signal and the currently loaded instruction. For more detail on TAP sharing via JTAGC instructions refer to *Section , ACCESS_AUX_TAP_x instructions*.

Data is shifted between TDI and TDO through the selected register starting with the least significant bit, as illustrated in *Figure 881*. This applies for the instruction register, test data registers, and the bypass register.

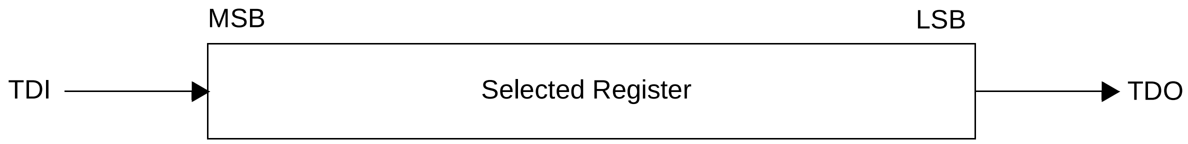
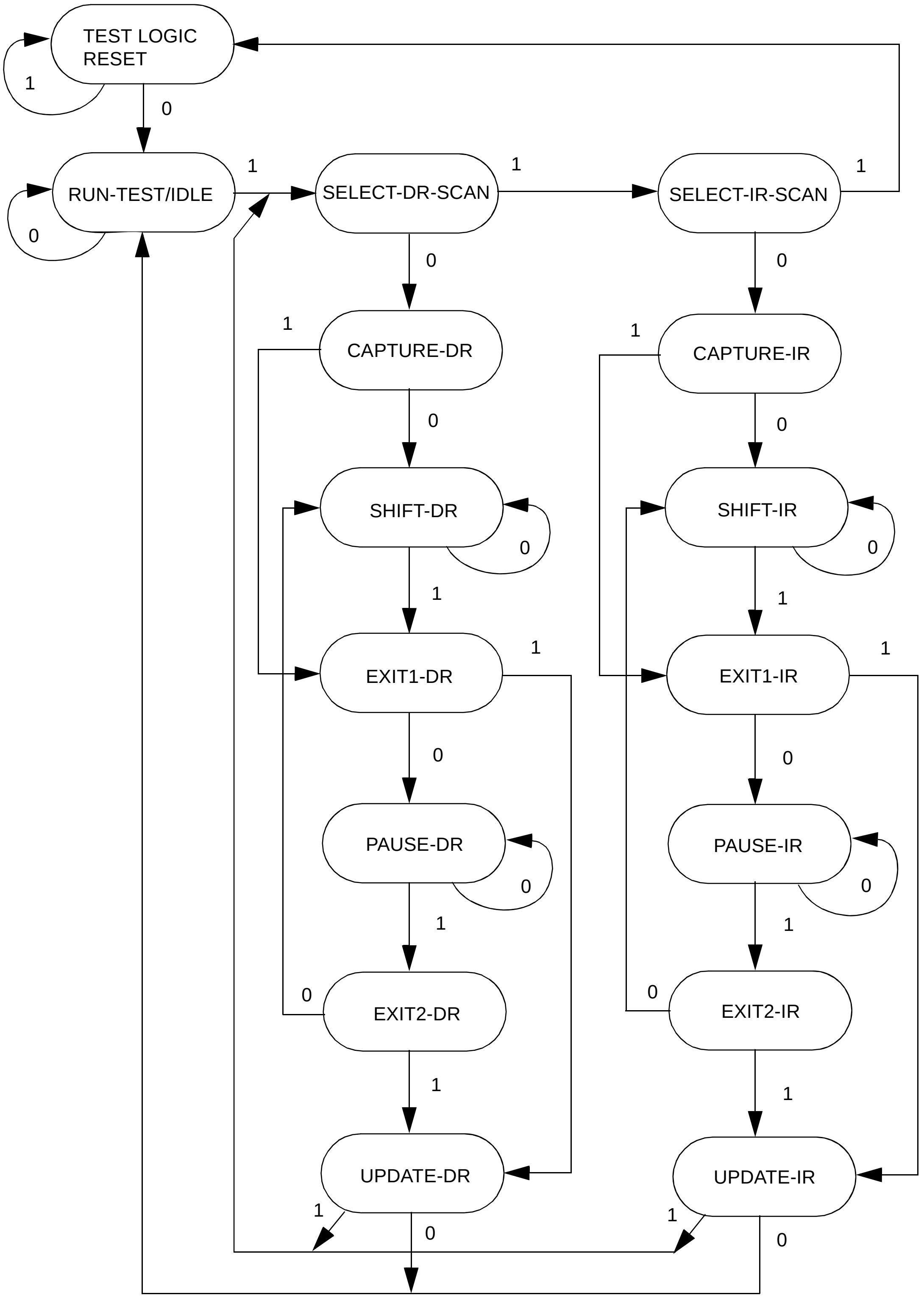


Figure 881. Shifting data through a register

33.5.3 TAP controller state machine

The TAP controller is a synchronous state machine that interprets the sequence of logical values on the TMS pin. *Figure 882* shows the machine's states. The value shown next to each state is the value of the TMS signal sampled on the rising edge of the TCK signal. As *Figure 882* shows, holding TMS at logic 1 while clocking TCK through a sufficient number of rising edges also causes the state machine to enter the Test-Logic-Reset state.



NOTE: The value shown adjacent to each state transition in this figure represents the value of TMS at the time of a rising edge of TCK.

Figure 882. IEEE 1149.1-2001 TAP controller finite state machine

Enabling the TAP controller

The JTAGC TAP controller is enabled by setting JCOMP to the JTAGC enable value. The JTAGC TAP controller is enabled by setting JCOMP to a logic 1 value.

Selecting an IEEE 1149.1-2001 register

Access to the JTAGC data registers is achieved by loading the instruction register with any of the JTAGC block instructions while the JTAGC is enabled. Instructions are shifted in via the Select-IR-Scan path and loaded in the Update-IR state. At this point, all data register access is performed via the Select-DR-Scan path.

The Select-DR-Scan path is used to read or write the register data by shifting in the data (LSB first) during the Shift-DR state. When reading a register, the register value is loaded into the IEEE 1149.1-2001 shifter during the Capture-DR state. When writing a register, the value is loaded from the IEEE 1149.1-2001 shifter to the register during the Update-DR state. When reading a register, there is no requirement to shift out the entire register contents. Shifting may be terminated once the required number of bits have been acquired.

33.5.4 JTAGC block instructions

The JTAGC block implements the IEEE 1149.1-2001 defined instructions listed in *Table 853*. This section gives an overview of each instruction; refer to the IEEE 1149.1-2001 standard for more details. All undefined opcodes are reserved.

Table 853. JTAG Instructions

Instruction	Code[4:0]	Instruction summary
IDCODE	00001	Selects device identification register for shift
SAMPLE/PRELOAD	00010	Selects boundary scan register for shifting, sampling, and preloading without disturbing functional operation
SAMPLE	00011	Selects boundary scan register for shifting and sampling without disturbing functional operation
EXTEST	00100	Selects boundary scan register while applying preloaded values to output pins and asserting functional reset
HIGHZ	01001	Selects bypass register while three-stating all output pins and asserting functional reset
CLAMP	01100	Selects bypass register while applying preloaded values to output pins and asserting functional reset
ACCESS_AUX_TAP_x	10000–11110	Grants one of the auxiliary TAP controllers ownership of the TAP as shown in the cells below. The number of auxiliary TAP controllers sharing the port is 4.
ACCESS_AUX_TAP_NPC	10000	Enables access to the NPC TAP controller
ACCESS_AUX_TAP_ONCE	10001	Enables access to the primary OnCE TAP controller (Primary CPU)
ACCESS_AUX_TAP_eTPU	10010	Enables access to the eTPU Nexus TAP controller
BYPASS	11111	Selects bypass register for data operations
Factory debug reserved	00101, 00110, 01010, 00111	Intended for factory debug only

Table 853. JTAG Instructions (continued)

Instruction	Code[4:0]	Instruction summary
Reserved[1]	All other opcodes	Decoded to select bypass register

1. The manufacturer reserves the right to change the decoding of reserved instruction codes in the future.

IDCODE instruction

IDCODE selects the 32-bit device identification register as the shift path between TDI and TDO. This instruction allows interrogation of the MCU to determine its version number and other part identification data. IDCODE is the instruction placed into the instruction register when the JTAGC block is reset.

SAMPLE/PRELOAD instruction

The SAMPLE/PRELOAD instruction has two functions:

- First, the SAMPLE portion of the instruction obtains a sample of the system data and control signals present at the MCU input pins and just before the boundary scan register cells at the output pins. This sampling occurs on the rising edge of TCK in the Capture-DR state when the SAMPLE/PRELOAD instruction is active. The sampled data is viewed by shifting it through the boundary scan register to the TDO output during the Shift-DR state. Both the data capture and the shift operation are transparent to system operation.
- Secondly, the PRELOAD portion of the instruction initializes the boundary scan register cells before selecting the EXTEST or CLAMP instructions to perform boundary scan tests. This is achieved by shifting in initialization data to the boundary scan register during the Shift-DR state. The initialization data is transferred to the parallel outputs of the boundary scan register cells on the falling edge of TCK in the Update-DR state. The data is applied to the external output pins by the EXTEST or CLAMP instruction. System operation is not affected.

SAMPLE instruction

The SAMPLE instruction obtains a sample of the system data and control signals present at the MCU input pins and just before the boundary scan register cells at the output pins. This sampling occurs on the rising edge of TCK in the Capture-DR state when the SAMPLE instruction is active. The sampled data is viewed by shifting it through the boundary scan register to the TDO output during the Shift-DR state. There is no defined action in the Update-DR state. Both the data capture and the shift operation are transparent to system operation.

EXTEST—external test instruction

EXTEST selects the boundary scan register as the shift path between TDI and TDO. It allows testing of off-chip circuitry and board-level interconnections by driving preloaded data contained in the boundary scan register onto the system output pins. Typically, the preloaded data is loaded into the boundary scan register using the SAMPLE/PRELOAD instruction before the selection of EXTEST. EXTEST asserts the internal system reset for the MCU to force a predictable internal state while performing external boundary scan operations.

HIGHZ instruction

HIGHZ selects the bypass register as the shift path between TDI and TDO. While HIGHZ is active all output drivers are placed in an inactive drive state (e.g., high impedance). HIGHZ also asserts the internal system reset for the MCU to force a predictable internal state.

CLAMP instruction

CLAMP allows the state of signals driven from MCU pins to be determined from the boundary scan register while the bypass register is selected as the serial path between TDI and TDO. CLAMP enhances test efficiency by reducing the overall shift path to a single bit (the bypass register) while conducting an EXTEST type of instruction through the boundary scan register. CLAMP also asserts the internal system reset for the MCU to force a predictable internal state.

ACCESS_AUX_TAP_x instructions

The JTAGC is configurable to allow up to fifteen other TAP controllers on the device to share the port with it. This is done by providing ACCESS_AUX_TAP_x instructions for each of these TAP controllers. When this instruction is loaded, control of the JTAG pins are transferred to the selected TAP controller. Any data input via TDI and TMS is passed to the selected TAP controller, and any TDO output from the selected TAP controller is sent back to the JTAGC to be output on the pins. The JTAGC regains control of the JTAG port during the UPDATE-DR state if the PAUSE-DR state was entered. Auxiliary TAP controllers are held in RUN-TEST/IDLE while they are inactive. Instructions not used to access an auxiliary TAP controller on a device are treated like the BYPASS instruction.

BYPASS instruction

BYPASS selects the bypass register, creating a single-bit shift register path between TDI and TDO. BYPASS enhances test efficiency by reducing the overall shift path when no test operation of the MCU is required. This allows more rapid movement of test data to and from other components on a board that are required to perform test functions. While the BYPASS instruction is active the system logic operates normally.

33.5.5 Boundary scan

The boundary scan technique allows signals at component boundaries to be controlled and observed through the shift-register stage associated with each pad. Each stage is part of a larger boundary scan register cell, and cells for each pad are interconnected serially to form a shift-register chain around the border of the design. The boundary scan register consists of this shift-register chain, and is connected between TDI and TDO when the EXTEST, SAMPLE, or SAMPLE/PRELOAD instructions are loaded. The shift-register chain contains a serial input and serial output, as well as clock and control signals.

33.6 Initialization/application information

The test logic is a static logic design, and TCK can be stopped in either a high or low state without loss of data. However, the system clock is not synchronized to TCK internally. Any mixed operation using both the test logic and the system functional logic requires external synchronization.

To initialize the JTAGC block and enable access to registers, the following sequence is required:

1. Set the JCOMP signal to the JTAGC enable value, thereby enabling the JTAGC TAP controller.
2. Load the appropriate instruction for the test or action to be performed.

34 Nexus Port Controller (NPC)

34.1 Information specific to this device

This section presents device-specific parameterization, customization, and feature availability information not specifically referenced in the remainder of this chapter.

34.1.1 Device-specific features

- Nexus Class 3+
- 14-bit full duplex pin interface for high visibility throughput
  - Reduced port mode (RPM) comprises 4 pins
  - Auxiliary Output Port
    1 MCKO (Message Clock Out) pin
    4 MDO (Message Data Out) pins
    2 $\overline{\text{MSEO}}$ (Message Start/End Out) pins
    1 $\overline{\text{RDY}}$ (Ready) pin
    1 $\overline{\text{EVTO}}$ (Event Out) pin
  - Auxiliary Input Port
    1 $\overline{\text{EVTI}}$ (Event In) pin
  - 5-pin JTAG port (JCOMP, TDI, TDO, TMS, and TCK)
- Host Processor (e200z4) Development Support
  - IEEE-ISTO 5001-2010 standard Class 2 compliant
  - Program Trace via Branch Trace Messaging (BTM). Branch trace messaging displays program flow discontinuities (direct branches, indirect branches, exceptions, etc.), allowing the development tool to interpolate what transpires between the discontinuities. Thus, static code can be traced
  - Watchpoint Messaging (WPM) via the auxiliary port
  - Watchpoint Trigger enable of Program Trace Messaging
  - Subset of Power Architecture embedded category software debug facilities with OnCE block (Nexus Class 1 features)
- eTPU Development Support
  - IEEE-ISTO 5001™ - 2003 standard Class 1 compliant for the eTPU engine
  - Nexus based breakpoint/watchpoint configuration and single step support
  - Run-time access to the on-chip memory map via the Nexus Read/Write Access protocol. This feature supports accesses for run-time internal visibility, calibration

variable acquisition, calibration constant tuning, and external rapid prototyping for powertrain automotive development systems

  - All features are independently configurable and controllable via the IEEE 1149.1 I/O port
  - The Nexus block reset is controlled with JCOMP, power-on reset, and the TAP state machine. All theses sources are independent of system reset
  - Power-on-reset status indication during reset via MDO[0] in disabled and reset modes

- As some of the pads used for the Nexus interface are multi-voltage pads, the maximum supported interface frequency is limited to 40 MHz in some modes. This corresponds to a MCKO clock divider of 1/2 for system clocks up to 80 MHz, and 1/4 for system clocks > 80 MHz.
- Nexus Double Data Rate (DDR) mode is not available in the SPC564A70 family of devices.
- A control bit in the Nexus port controller, NP_PCR[NEXCFG], is used to control whether the added signals for full width trace port are routed to the MDO[4:11] signals, or the CAL_MDO[4:11] signals. Although the CAL_MDO[4:11] signals are only available in the 496 CSP package, this control bit still needs to be programmed when the device is assembled in any other package.

  Table 854 shows the maximum trace port frequencies supported in different configurations, as well as the operation of the NPC_PCR[FPM] and [NEXCFG] control bits.

Table 854. Nexus trace port routing and speed

Package	Max. trace port MHz	Port width NPC_PCR[FPM]	Port routing bit NPC_PCR[NEXCFG]	MDO[0:3] usage	MDO[4:11] usage	CAL_MDO[4:11] usage
Production (176QFP, 208BGA, 324BGA)	80	4 MDO (Narrow, FPM=0)	Don’t care	Trace port use	I/O use	Inactive
	40	12 MDO (Wide, FPM=1)	NEXCFG = 1		Trace port use	
Debug/Cal package (496 CSP)	80	4 MDO (Narrow, FPM=0)	Don't care		I/O use	
	40	12 MDO (Wide, FPM=1)	NEXCFG = 1 Use MDO		Trace port use	Trace port use
	80		NEXCFG = 0 (Default setting) Use CAL_MDO		I/O use	Trace port use

34.1.2 Parameter values

Table 855. Parameter values

Parameter	Description	Value
NPC parameters		
RPM_MDO (R)	Number of MDO pins available in reduced-port mode	4
NUM_AUX (X)	Number of Nexus devices on the device sharing the Nexus auxiliary port (not including the NPC)	1
NUM_BKPT (B)	Number of breakpoint requests coming into the block from Nexus and non-Nexus sources on the device	1
NUM_EVTO (E)	Number of sources that output an EVTO	2
NUM_JCOMP (J)	Number of JCOMP bits used. Depends on the number of non-Nexus blocks sharing the TAP	1
DC	Design Code for the design center responsible for the design of the device (TECD)	0x2B
Part Identification Number (npc_did_pn_plug[9:0])	Part number of the device	0x203
Part Revision Number (npc_did_rev_plug[3:0])	Revision number of the device	0x1
NPC JCOMP Plug (npc_jcomp_plug)	JCOMP value required to grant Nexus control of the JTAG port	0b1
NSEDI parameters		
Part Identification Number (nex_did_pn_plug[9:0])	Part number of the eTPU	0x125
Part Revision Number (nex_did_rev_plug[3:0])	Revision number of the eTPU	0x0

34.2 Introduction

Figure 883 is a block diagram of the Nexus Port Controller (NPC) block.

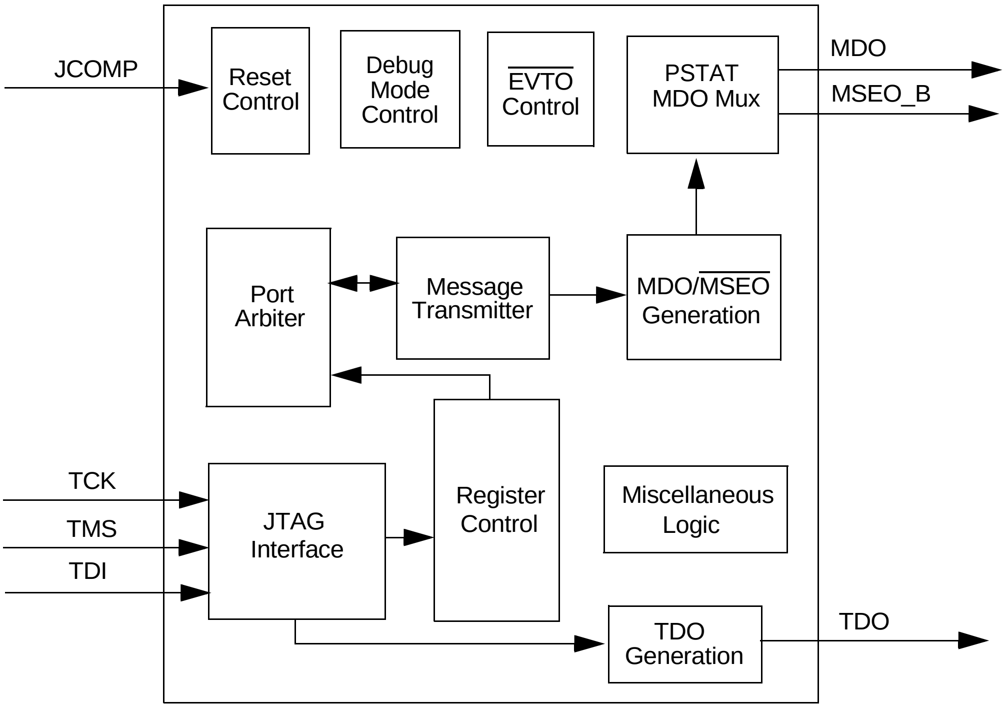


Figure 883. Nexus Port Controller block diagram

34.2.1 Overview

On a system-on-chip device, there are often multiple blocks that require development support. Each of these blocks implements a development interface based on the IEEE-ISTO 5001-2001 standard. The blocks share input and output ports that interface with the development tool. The NPC controls the usage of the input and output port in a manner that allows all the individual development interface blocks to share the port, and appear to the development tool to be a single block.

34.2.2 Features

The NPC block performs the following functions:

- Controls arbitration for ownership of the Nexus Auxiliary Output Port
- Nexus Device Identification Register and Messaging
- Generates MCKO enable and frequency division control signals
- Controls sharing of $\overline{\text{EVTO}}$
- Generates an MCKO clock gating control signal to enable gating of MCKO when the auxiliary output port is idle
- Control of the device-wide debug mode
- Generates asynchronous reset signal for Nexus blocks based on JCOMP input and power-on reset status
- System clock locked status indication via MDO[0] following power-on reset

34.2.3 Modes of operation

The NPC block uses the JCOMP input and an internal power-on reset indication as its primary reset signals. Upon exit of reset, the mode of operation is determined by the Port Configuration Register (PCR) settings.

Reset

The NPC block is asynchronously placed in reset when either power-on reset is asserted, JCOMP is not set for Nexus access or the TAP controller state machine is in the Test-Logic-Reset state. Holding TMS high for five consecutive rising edges of TCK guarantees entry into the Test-Logic-Reset state regardless of the current TAP controller state. Following negation of power-on reset, the NPC remains in reset until the system clock achieves lock. The NPC is unaffected by other sources of reset. While in reset, the following actions occur:

- The TAP controller is forced into the Test-Logic-Reset state
- The auxiliary output port pins are negated
- The TDI, TMS, and TCK TAP inputs are ignored (when in power-on reset or JCOMP not set for NPC operation only)
- Registers default back to their reset values

Disabled-Port Mode

In disabled-port mode, auxiliary output pin port enable signals are negated, thereby disabling message transmission. Any debug feature that generates messages can not be used. The primary features available are Class 1 features and read/write access to the registers. Class 1 features include the ability to trigger a breakpoint event indication through $\overline{\text{EVTO}}$.

Full-Port Mode

Full-port mode (FPM) is entered by asserting the MCKO_EN and FPM bits in the PCR. All trace features are enabled or can be enabled by writing the configuration registers via the TAP. The number of MDO pins available is device-specific.

Reduced-Port Mode

Reduced-port mode (RPM) is entered by asserting the MCKO_EN bit and negating the FPM bit in the PCR. All trace features are enabled or can be enabled by writing the configuration registers via the TAP. The number of MDO pins available is device-specific.

34.3 External signal description

34.3.1 Overview

The NPC pin interface provides for the transmission of messages from Nexus blocks to the external development tools and for access to Nexus client registers. The NPC pin definition is outlined in *Table 856*.

Table 856. NPC signal properties

Name	Port	Function	Reset State	Pull(1)
EVTO_B	Auxiliary	Event Out pin	0b1	—
JCOMP	JTAG	JTAG Compliancy and TAP Sharing Control	—	Down
MDO	Auxiliary	Message Data Out pins	0(2)	—
MSEO	Auxiliary	Message Start/End Out pins	0b11	—
TCK	JTAG	Test Clock Input	—	Down
TDI	JTAG	Test Data Input	—	Up
TDO	JTAG	Test Data Output	High Z(3)	—
TMS	JTAG	Test Mode Select Input	—	Up

1. The pull is not implemented in this block. Pullup/pulldown devices are implemented in the pads.
2. Following a power-on reset, MDO[0] remains asserted until power-on reset is exited and the system clock achieves lock.
3. TDO output buffer enable is negated when the NPC is not in the Shift-IR or Shift-DR states. A weak pull may be implemented on TDO at the SoC level.

34.3.2 Detailed signal descriptions

This section describes each of the signals listed in *Table 856* in more detail.Note that the JTAG test clock (TCK) input from the pin is not a direct input to the NPC. The NPC requires two separate input clocks for TCK clocked logic, one for posedge (rising edge TCK) logic and one for negedge (falling edge TCK) logic. Both clocks are derived from the pin TCK, and generated external to the NPC.

EVTO_B — Event Out

Event Out ($\overline{\text{EVTO}}$) is an output pin that is asserted upon breakpoint occurrence to provide breakpoint status indication. The EVTO output of the NPC is generated based on the values of the individual EVTO signals from all Nexus blocks that implement the signal.

JCOMP - JTAG Compliancy

The JCOMP signal provides the ability to share the TAP. The NPC TAP controller is enabled when JCOMP is set to the NPC enable encoding, otherwise the NPC TAP controller remains in reset.

MDO - Message Data Out

Message Data Out (MDO) are output pins used for uploading OTM, BTM, DTM, and other messages to the development tool. The development tool should sample MDO on the rising edge of MCKO. The width of the MDO bus used is determined by reset configuration.

MSEO - Message Start/End Out

Message Start/End Out ($\overline{\text{MSEO}}$) is an output pin that indicates when a message on the MDO pins has started, when a variable length packet has ended, or when the message has ended. The development tool should sample $\overline{\text{MSEO}}$ on the rising edge of MCKO.

TCK - Test Clock Input

Test Clock Input (TCK) pin is used to synchronize the test logic and control register access through the JTAG port.

TDI - Test Data Input

Test Data Input (TDI) pin receives serial test instruction and data. TDI is sampled on the rising edge of TCK.

TDO - Nexus Test Data Output

Test Data Output (TDO) pin transmits serial output for instructions and data. TDO is three-stateable and is actively driven in the SHIFT-IR and SHIFT-DR controller states. TDO is updated on the falling edge of TCK and sampled by the development tool on the rising edge of TCK.

TMS - Test Mode Select

Test Mode Select Input (TMS) pin is used to sequence the IEEE 1149.1-2001 TAP controller state machine. TMS is sampled on the rising edge of TCK.

34.4 Register definition

This section provides a detailed description of the NPC registers accessible to the end user. Individual bit-level descriptions and reset states of the registers are included.

Table 857 shows the NPC registers by index values. The registers are not memory-mapped and can only be accessed via the TAP. The NPC block does not implement the client select control register because the value does not matter when accessing the registers. Note that the bypass and instruction registers have no index values. These registers are not accessed in the same manner as Nexus client registers. Refer to the individual register descriptions for more detail.

Table 857. NPC registers

Index	Register
0	Device ID Register (DID)
127	Port Configuration Register (PCR)

34.4.1 Register descriptions

This section consists of NPC register descriptions.

Bypass Register

The bypass register is a single-bit shift register path selected for serial data transfer between TDI and TDO when the BYPASS instruction or any unimplemented instructions are active. After entry into the Capture-DR state, the single-bit shift register is set to a logic 0. Therefore, the first bit shifted out after selecting the bypass register is always a logic 0.

Instruction Register

The NPC block uses a 4-bit instruction register as shown in *Figure 884*. The instruction register is accessed via the SELECT_IR_SCAN path of the tap controller state machine, and allows instructions to be loaded into the block to enable the NPC for register access (NEXUS_ENABLE) or select the bypass register as the shift path from TDI to TDO (BYPASS or unimplemented instructions).

Instructions are shifted in through TDI while the TAP controller is in the Shift-IR state, and latched on the falling edge of TCK in the Update-IR state. The latched instruction value can only be changed in the Update-IR and Test-Logic-Reset TAP controller states. Synchronous entry into the Test-Logic-Reset state results in synchronous loading of the BYPASS instruction. Asynchronous entry into the Test-Logic-Reset state results in asynchronous loading of the BYPASS instruction. During the Capture-IR TAP controller state, the instruction register is loaded with the value of the previously executed instruction, making this value the register’s read value when the TAP controller is sequenced into the Shift-IR state.

	3	2	1	0
R	Previous Instruction Opcode			
W	Instruction Opcode			
Reset:	BYPASS Instruction Opcode (0xF)			

Figure 884. 4-bit Instruction Register

Nexus Device ID Register (DID)

The device identification register, shown in *Figure 885*, allows the part revision number, design center, part identification number, and manufacturer identity code of the part to be determined through the auxiliary output port.

Register index: 0

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Part Revision Number				Design Center						Part Identification Number					
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	PRN				DC						PIN					

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Part Identification Number				Manufacturer Identity Code											1
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	PIN (cont'd.)				0	0	0	0	0	0	0	1	1	1	0	1

▨ = Reserved

Figure 885. Nexus Device ID Register

Table 858. DID field descriptions

Bit	Name	Description
31:28	**PRN**	**Part Revision Number** These bits contain the revision number of the part.
27:22	**DC**	**Design Center** These bits indicate the device design center.
21:12	**PIN**	**Part Identification Number** These bits contain the part number of the device.
11:1	**MIC**	**Manufacturer Identity Code** These bits contain the reduced Joint Electron Device Engineering Council (JEDEC) ID.
0	**Bit [0]**	**IDCODE Register ID** Bit [0] identifies this register as the device identification register and not the bypass register.

Port Configuration Register (PCR)

Register index: 127

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	FPM	MCKO_GT	MCKO_EN	MCKO_DIV			EVT_EN	0	NEXCFG	0	0	0	0	0	0	0
W								▨		▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	LP_DBG	0	0	0	0	0	LP2_SYN	LP1_SYN	0	0	0	0	0	0	0	PSTAT_EN
W		▨	▨	▨	▨	▨			▨	▨	▨	▨	▨	▨	▨	
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Reserved

Figure 886. Port Configuration Register (PCR)

The PCR, shown in *Figure 886*, is used to select the NPC mode of operation, enable MCKO and select the MCKO frequency, and enable or disable MCKO gating. This register should be configured as soon as the NPC is enabled.

The PCR register may be rewritten by the debug tool subsequent to the enabling of the NPC for low power debug support. In this case, the debug tool may set and clear the LP_DBG and LPn_SYN bits, but must preserve the original state of the remaining bits in the register.

Note: The mode or clock division must not be modified after MCKO has been enabled. Changing the mode or clock division while MCKO is enabled can produce unpredictable results.

Table 859. PCR field descriptions

Bit	Name	Description
31	FPM	**Full Port Mode** The value of the FPM bit determines if the auxiliary output port uses the full MDO port or a reduced MDO port to transmit messages. 1 = All MDO pins are used to transmit messages 0 = A subset of MDO pins are used to transmit messages
30	MCKO_GT	**MCKO Clock Gating Control** This bit is used to enable or disable MCKO clock gating. If clock gating is enabled, the MCKO clock is gated when the NPC is in enabled mode but not actively transmitting messages on the auxiliary output port. When clock gating is disabled, MCKO is allowed to run even if no auxiliary output port messages are being transmitted. 1 = MCKO gating is enabled 0 = MCKO gating is disabled
29	MCKO_EN	**MCKO Enable** This bit enables the MCKO clock to run. When enabled, the frequency of MCKO is determined by the MCKO_DIV field. 1 = MCKO clock is enabled 0 = MCKO clock is driven to zero
28:26	MCKO_DIV	**MCKO Division Factor** The value of this signal determines the frequency of MCKO relative to the system clock frequency when MCKO_EN is asserted. *Table 860* shows the meaning of MCKO_DIV Values. In this table, SYS_CLK represents the system clock frequency.
25	EVT_EN	**EVTO/EVTI Enable** This bit enables the EVTO/EVTI port functions. 1 = EVTO/EVTI port enabled 0 = EVTO/EVTI port disabled
24	—	**Reserved**
23	NEXCFG	**Nexus Configuration Select** Generic Nexus control bit. Function is device-specific. 1 = NEXCFG set 0 = NEXCFG cleared
22:16	—	**Reserved**

Table 859. PCR field descriptions (continued)

Bit	Name	Description
15	LP_DBG_EN	**Low Power Debug Enable** This bit enables debug functionality on exit from low power modes on supported devices. 1 = Low power debug enabled 0 = Low power debug disabled
14:10	—	**Reserved**
9:8	LPn_SYN	**Low Power Mode n Synchronization** These bits are used to synchronize the entry into low power modes between the device and debug tool. Supported devices set these bits before a pending entry into low power mode. After reading the bit as set, the debug tool then clears the bit to acknowledge to the device that it may enter the low power mode. 1 = Low power mode entry pending 0 = Low power mode entry acknowledged
7:1	—	**Reserved**
0	PSTAT_EN	**Processor Status Mode Enable(1)** This bit enables processor status (PSTAT) mode. In PSTAT mode, all auxiliary output port MDO pins are used to transmit processor status information, and Nexus messaging is unavailable. 1 = PSTAT mode enabled 0 = PSTAT mode disabled

1. PSTAT Mode is intended for factory processor debug only. The PSTAT_EN bit should be written to disable PSTAT mode if Nexus messaging is desired. No Nexus messages are transmitted under any circumstances when PSTAT mode is enabled.

Table 860. MCKO_DIV values

MCKO_DIV[2:0]	MCKO frequency
0	SYS_CLK(1)
1	SYS_CLK/2
2	SYS_CLK/3
3	SYS_CLK/4
4	Reserved
5	Reserved
6	Reserved
7	SYS_CLK/8

1. The SYS_CLK setting for MCKO frequency should only be used if this setting does not violate the maximum operating frequency of the auxiliary port pins.

34.5 Functional description

34.5.1 NPC reset configuration

The NPC is placed in disabled mode upon exit of reset. If message transmission via the auxiliary port is desired, a write to the PCR is then required to enable the NPC and select the mode of operation. Asserting MCKO_EN places the NPC in enabled mode and enables MCKO. The frequency of MCKO is selected by writing the MCKO_DIV field. Asserting or negating the FPM bit selects full-port or reduced-port mode, respectively.

Table 861 describes the NPC reset configuration options.

Table 861. NPC reset configuration options

JCOMP equal to npc_jcomp_plug?	MCKO_EN bit of the Port Configuration Register	FPM bit of the Port Configuration Register	Configuration
no	X	X	Reset
yes	0	X	Disabled
yes	1	1	Full-Port Mode
yes	1	0	Reduced-Port Mode

34.5.2 Auxiliary output port

The auxiliary output port is shared by each of the Nexus modules on the device. The NPC communicates with each of the Nexus modules and arbitrates for access to the port.

Output message protocol

The protocol for transmitting messages via the auxiliary port is accomplished with the $\overline{\text{MSEO}}$ functions. The $\overline{\text{MSEO}}$ pins are used to signal the end of variable-length packets and the end of messages. They are not required to indicate the end of fixed-length packets. MDO and $\overline{\text{MSEO}}$ are sampled on the rising edge of MCKO.

Figure 887 illustrates the state diagram for $\overline{\text{MSEO}}$ transfers. All transitions not included in the figure are reserved, and must not be used.

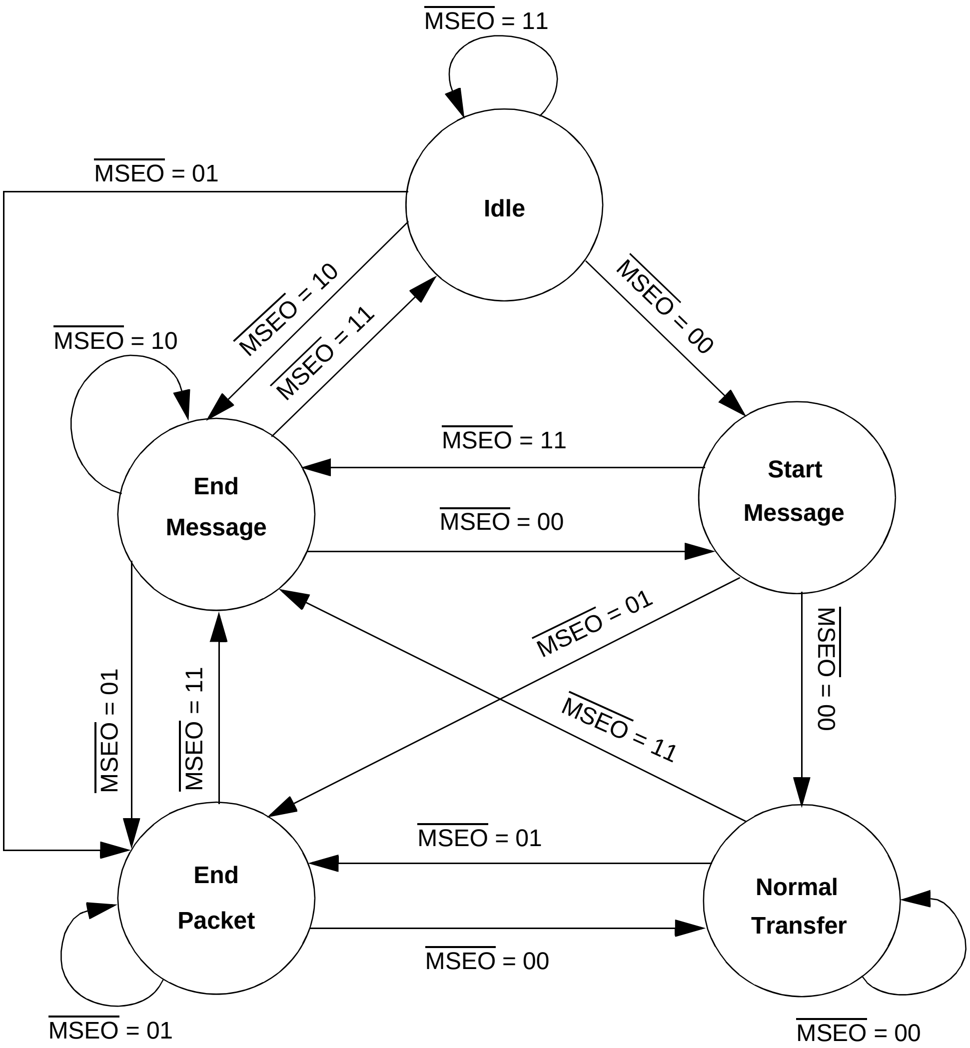


Figure 887. $\overline{\text{MSEO}}$ **transfers (for 2-bit** $\overline{\text{MSEO}}$**)**

Output messages

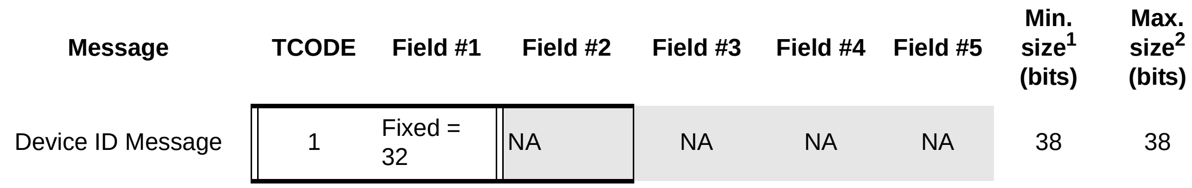
In addition to sending out messages generated in other Nexus blocks, the NPC can also output the device ID message contained in the device ID register and the port replacement output message on the MDO pins. The device ID message can also be sent out serially through TDO.

Table 862 describes the device ID and port replacement output messages that the NPC can transmit on the auxiliary port. The TCODE is the first packet transmitted.

Table 862. NPC output messages

Message name	Min. packet size (bits)	Max. packet size (bits)	Packet type	Packet name	Packet description
Device ID Message	6	6	fixed	TCODE	Value = 1
	32	32	fixed	ID	DID register contents

Figure 888 shows the various message formats that the pin interface formatter has to encounter. Note that for variable-length fields, the transmitted size of the field is determined from the range of the least significant bit to the most significant non-zero-valued bit (i.e. most significant zero-valued bits are not transmitted).



Message	TCODE	Field #1	Field #2	Field #3	Field #4	Field #5	Min. size[1] (bits)	Max. size[2] (bits)
Device ID Message	1	Fixed = 32	NA	NA	NA	NA	38	38

NOTES:

1. Minimum information size. The actual number of bits transmitted depends on the number of MDO pins

2. Maximum information size. The actual number of bits transmitted depends on the number of MDO pins

Figure 888. Message Field Sizes

The double edges in *Figure 888* indicate the starts and ends of messages. Fields without shaded areas between them are grouped into super-fields and can be transmitted together without end-of-packet indications between them.

Rules of message

- A variable-sized field within a message must end on a port boundary. (Port boundaries depend on the number of MDO pins active with the current reset configuration.)
- A variable-sized field may start within a port boundary only when following a fixed-length field.
- Super-fields must end on a port boundary.
- When a variable-length field is sized such that it does not end on a port boundary, it is necessary to extend and zero fill the remaining bits after the highest order bit so that it can end on a port boundary.
- Multiple fixed-length packets may start and/or end on a single clock.
- When any packet follows a variable-length packet, it must start on a port boundary.
- The field containing the TCODE number is always transferred out first, followed by subsequent fields of information.
- Within a field, the lowest significant bits are shifted out first. *Figure 889* shows the transmission sequence of a message that is made up of a TCODE followed by two fields.

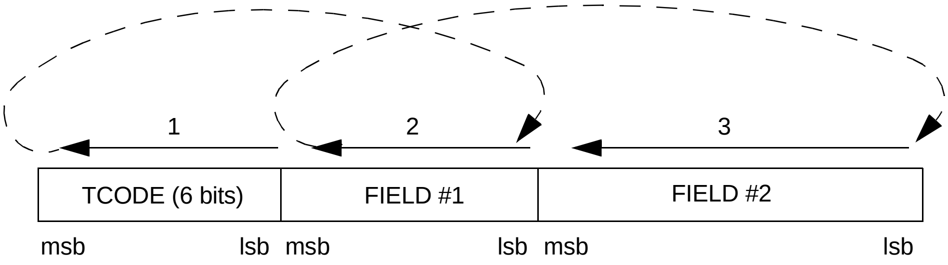


Figure 889. Transmission sequence of messages

34.5.3 IEEE 1149.1-2001 (JTAG) TAP

The NPC block uses the IEEE 1149.1-2001 TAP for accessing registers. Each of the individual Nexus blocks on the device implements a TAP controller for accessing its registers as well. TAP signals include TCK, TDI, TMS, and TDO. There may also be other blocks on the MCU that use the TAP and implement a TAP controller. The value of the JCOMP input controls ownership of the port between Nexus and non-Nexus blocks sharing the TAP.

Refer to the IEEE 1149.1-2001 specification for further detail on electrical and pin protocol compliance requirements.

The NPC implements a Nexus controller state machine that transitions based on the state of the IEEE 1149.1-2001 state machine shown in *Figure 891*. The Nexus controller state machine is defined by the IEEE-ISTO 5001-2010 standard. It is shown in *Figure 892*.

The instructions implemented by the NPC TAP controller are listed in *Table 863*. The value of the NEXUS-ENABLE instruction is 0b0000. Each unimplemented instruction acts like the BYPASS instruction. The size of the NPC instruction register is 4-bits.

Table 863. Implemented instructions

Instruction name	Private/Public	Opcode	Description
NEXUS-ENABLE	public	0x0	Activate Nexus controller state machine to read and write NPC registers.
BYPASS	private	0xF	NPC BYPASS instruction. Also the value loaded into the NPC IR upon exit of reset.

Data is shifted between TDI and TDO starting with the least significant bit as illustrated in *Figure 890*. This applies for the instruction register and all Nexus tool-mapped registers.

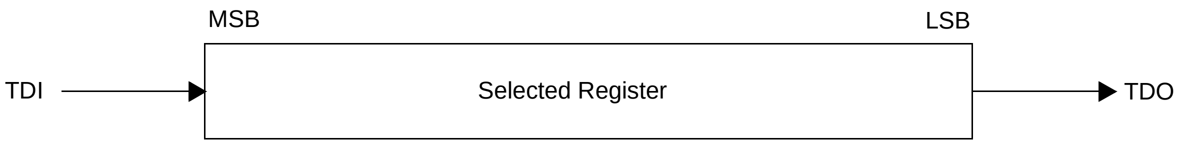
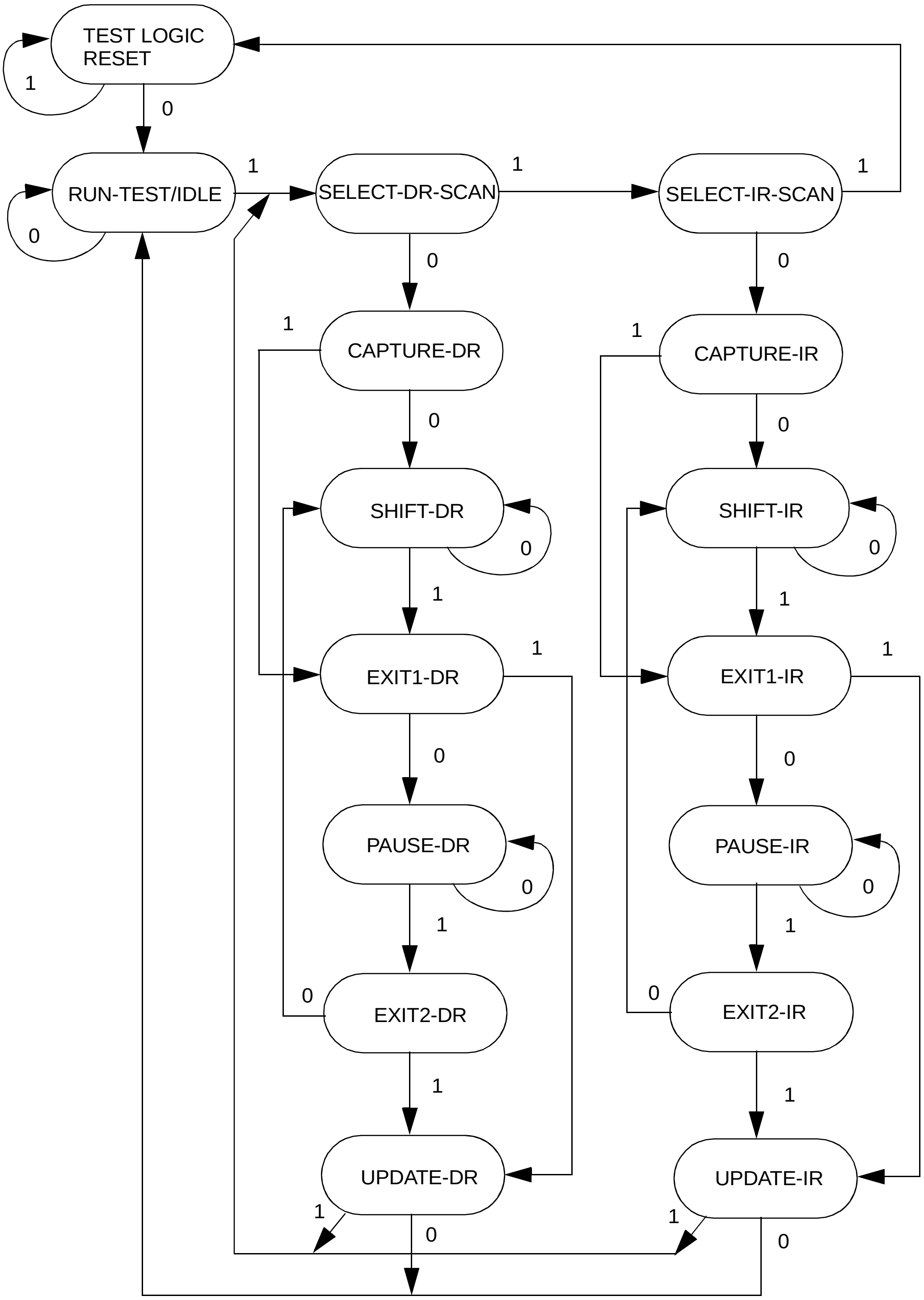


Figure 890. Shifting data into register

Enabling the NPC TAP controller

Assertion of the power-on reset signal or setting JCOMP to a value other than the NPC enable encoding resets the NPC TAP controller. When not in power-on reset, the NPC TAP controller is enabled by driving JCOMP with the NPC enable value and exiting the Test-Logic-Reset state. Loading the NEXUS-ENABLE instruction then grants access to Nexus debug.



NOTE: The value shown adjacent to each state transition in this figure represents the value of TMS at the time of a rising edge of TCK.

Figure 891. IEEE 1149.1-2001 TAP controller state machine

Retrieving device IDCODE

The Nexus TAP controller does not implement the IDCODE instruction. However, the device identification message can be output by the NPC through the auxiliary output port or shifted out serially by accessing the Nexus Device ID register through the TAP. Transmission of the device identification message on the auxiliary output port MDO pins occurs immediately after a write to the PCR, if the NPC is enabled. Transmission of the device identification message serially via TDO is achieved by performing a read of the register contents as described in *Section , Selecting a Nexus client register*.

Loading NEXUS-ENABLE instruction

Access to the NPC registers is enabled when the TAP controller instruction register is loaded with the NEXUS-ENABLE instruction. This instruction is shifted in via the SELECT-IR-SCAN path and loaded in the UPDATE-IR state. At this point, the Nexus controller state machine, shown in *Figure 892*, transitions to the REG_SELECT state. The Nexus controller has three states: idle, register select, and data access. *Table 864* illustrates the IEEE 1149.1 sequence to load the NEXUS-ENABLE instruction.

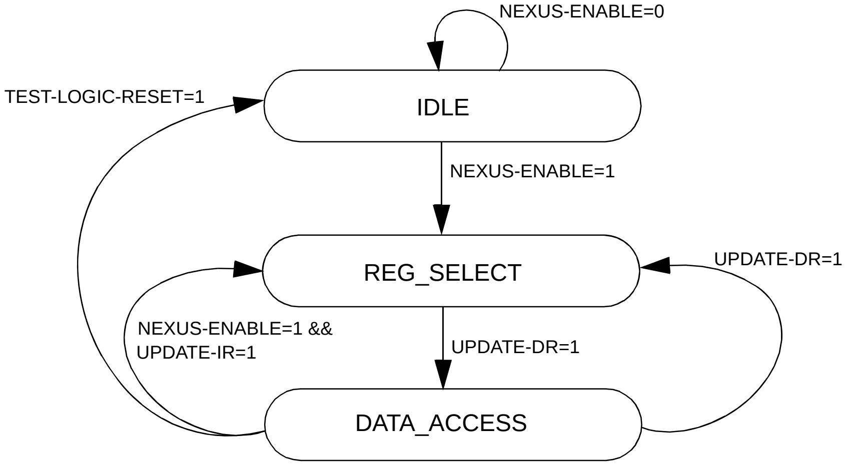


Figure 892. NEXUS controller state machine

Table 864. Loading NEXUS-ENABLE instruction

Clock	TMS	IEEE 1149.1 state	Nexus state	Description
0	0	RUN-TEST/IDLE	IDLE	IEEE 1149.1-2001 TAP controller in idle state
1	1	SELECT-DR-SCAN	IDLE	Transitional state
2	1	SELECT-IR-SCAN	IDLE	Transitional state
3	0	CAPTURE-IR	IDLE	Internal shifter loaded with current instruction
4	0	SHIFT-IR	IDLE	TDO becomes active, and the IEEE 1149.1-2001 shifter is ready. Shift in all but the last bit of the NEXUS_ENABLE instruction.
3 TCKS				
12	1	EXIT1-IR	IDLE	Last bit of instruction shifted in

Table 864. Loading NEXUS-ENABLE instruction (continued)

Clock	TMS	IEEE 1149.1 state	Nexus state	Description
13	1	UPDATE-IR	IDLE	NEXUS-ENABLE loaded into instruction register
14	0	RUN-TEST/IDLE	REG_SELECT	Ready to be read/write Nexus registers

Selecting a Nexus client register

When the NEXUS-ENABLE instruction is decoded by the TAP controller, the input port allows development tool access to all Nexus registers. Each register has a 7-bit address index.

All register access is performed via the SELECT-DR-SCAN path. The Nexus Controller defaults to the REG_SELECT state when enabled. Accessing a register requires two passes through the SELECT-DR-SCAN path: one pass to select the register and the second pass to read/write the register.

The first pass through the SELECT-DR-SCAN path is used to enter an 8-bit Nexus command consisting of a read/write control bit in the LSB followed by a 7-bit register address index, as illustrated in *Figure 893*. The read/write control bit is set to 1 for writes and 0 for reads.

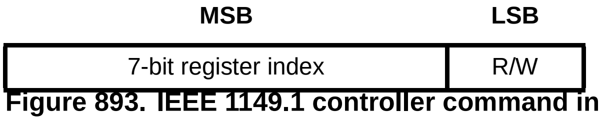


Figure 893. IEEE 1149.1 controller command input

The second pass through the SELECT-DR-SCAN path is used to read or write the register data by shifting in the data (LSB first) during the SHIFT-DR state. When reading a register, the register value is loaded into the IEEE 1149.1-2001 shifter during the CAPTURE-DR state. When writing a register, the value is loaded from the IEEE 1149.1-2001 shifter to the register during the UPDATE-DR state. When reading a register, there is no requirement to shift out the entire register contents. Shifting may be terminated once the required number of bits have been acquired.

Table 865 illustrates a sequence which writes a 32-bit value to a register

Table 865. Write to a 32-bit Nexus client register

Clock	TMS	IEEE 1149.1 state	Nexus state	Description
0	0	RUN-TEST/IDLE	REG_SELECT	IEEE 1149.1-2001 TAP controller in idle state
1	1	SELECT-DR-SCAN	REG_SELECT	First pass through SELECT-DR-SCAN path
2	0	CAPTURE-DR	REG_SELECT	Internal shifter loaded with current value of controller command input.
3	0	SHIFT-DR	REG_SELECT	TDO becomes active, and write bit and 6 bits of register index shifted in.
7 TCKs				
12	1	EXIT1-DR	REG_SELECT	Last bit of register index shifted into TDI
13	1	UPDATE-DR	REG_SELECT	Controller decodes and selects register
14	1	SELECT-DR-SCAN	DATA_ACCESS	Second pass through SELECT-DR-SCAN path

Table 865. Write to a 32-bit Nexus client register (continued)

Clock	TMS	IEEE 1149.1 state	Nexus state	Description
15	0	CAPTURE-DR	DATA_ACCESS	Internal shifter loaded with current value of register
16	0	SHIFT-DR	DATA_ACCESS	TDO becomes active, and outputs current value of register while new value is shifted in through TDI
31 TCKs				
48	1	EXIT1-DR	DATA_ACCESS	Last bit of current value shifted out TDO. Last bit of new value shifted in TDI.
49	1	UPDATE-DR	DATA_ACCESS	Value written to register
50	0	RUN-TEST/IDLE	REG_SELECT	Controller returned to idle state. It could also return to SELECT-DR-SCAN to write another register.

34.5.4 Nexus JTAG port sharing

Each of the individual Nexus blocks on the device implements a TAP controller for accessing its registers. When Nexus has ownership of the TAP, only the block whose NEXUS-ENABLE instruction is loaded has control of the TAP. This allows the interface to all of these individual TAP controllers to appear to be a single port from outside the device. If no register is selected as the shift path for a Nexus block, that block acts like a single-bit shift register, or bypass register.

34.5.5 MCKO and ipg_sync_mcko

MCKO is an output clock to the development tools used for the timing of MSEO and MDO pin functions. MCKO is derived from the system clock and its frequency is determined by the value of the MCKO_DIV field in the PCR. Possible operating frequencies include system clock, one-half system clock, one-quarter system clock, and one-eighth system clock speed.

The NPC also generates an MCKO clock gating control output signal. This output can be used by the MCKO generation logic to gate the transmission of MCKO when the auxiliary port is enabled but not transmitting messages. The setting of the MCKO_GT bit inside the PCR determines whether or not MCKO gating control is active. The MCKO_GT bit resets to a logic 0. In this state gating of MCKO is disabled. To enable gating of MCKO, the MCKO_GT bit in the PCR is written to a logic 1.

34.5.6 EVTO sharing

The NPC block controls sharing of the EVTO output between all Nexus clients that produce an EVTO signal. The NPC assumes incoming EVTO signals will be asserted for one system clock period. After receiving a single clock period of asserted EVTO from any Nexus client, the NPC latches the result, and drives EVTO for one MCKO period on the following clock. When there is no active MCKO, such as in disabled mode, the NPC drives EVTO for two system clock periods. EVTO sharing is active as long as the NPC is not in reset.

34.5.7 Nexus reset control

The JCOMP input that is used as the primary reset signal for the NPC is also used by the NPC to generate a single-bit reset signal for other Nexus blocks. If JCOMP is negated, an internal reset is asserted, indicating that all Nexus modules should be held in reset.

34.5.8 System clock locked indication

Following a power-on reset, MDO[0] can be monitored to provide the lock status of the system clock. MDO[0] is driven to a logic 1 until the system clock achieves lock after exiting power-on reset. Once the system clock is locked, MDO[0] is negated and tools may begin Nexus configuration. Loss of lock conditions that occur subsequent to the exit of power-on reset and the initial lock of the system clock do not cause a Nexus reset, and therefore do not result in MDO[0] driven high.

34.6 Initialization/Application information

34.6.1 Accessing NPC tool-mapped registers

To initialize the TAP for Nexus register accesses, the following sequence is required:

1. Enable the Nexus TAP controller
2. Load the TAP controller with the NEXUS-ENABLE instruction

To write control data to NPC tool-mapped registers, the following sequence is required:

1. Write the 7-bit register index and set the write bit to select the register with a pass through the SELECT-DR-SCAN path in the TAP controller state machine.
2. Write the register value with a second pass through the SELECT-DR-SCAN path. Note that the prior value of this register is shifted out during the write.

To read status and control data from NPC tool-mapped registers, the following sequence is required:

1. Write the 7-bit register index and clear the write bit to select register with a pass through SELECT-DR-SCAN path in the TAP controller state machine.
2. Read the register value with a second pass through the SELECT-DR-SCAN path. Data shifted in is ignored.

See the IEEE-ISTO 5001-2001 standard for more detail.

35 Development Trigger Semaphore (DTS)

35.1 Introduction

Devices in the SPC564A70 family include a system development feature, the Development Trigger Semaphore (DTS) module, that enables software to signal an external tool by driving a persistent (affected only by reset or an external tool) signal on an external device pin. There are a variety of ways this module can be used, including as a component of an external real-time data acquisition system(bh).

35.2 Overview

The Development Trigger Semaphore (DTS) module consists of three registers and a small amount of combinational logic to generate an output signal—DTS Trigger Output (DTO). The registers are as follows.

- DTS_SEMAPHORE register—Any bit in this 32-bit register, when set to a value of logic '1', causes the DTS module output signal to be asserted, enabling an external tool to detect up to 32 signals from the application software. In an application, each bit is generally associated with a specific data set.

  Only the processor core and DMA module can set bits in this register. The bits can only be cleared by a tool access via Nexus Read/Write Access over the JTAG port.
- DTS_STARTUP register—This register provides a mechanism for the external tool to notify software running on the CPU that the tool is connected and can provide information about either the type of tool or options that can be used by the software.
- DTS_ENABLE register—This register provides an enable/disable capability for the DTS feature.

The architecture is shown in *Figure 894*.

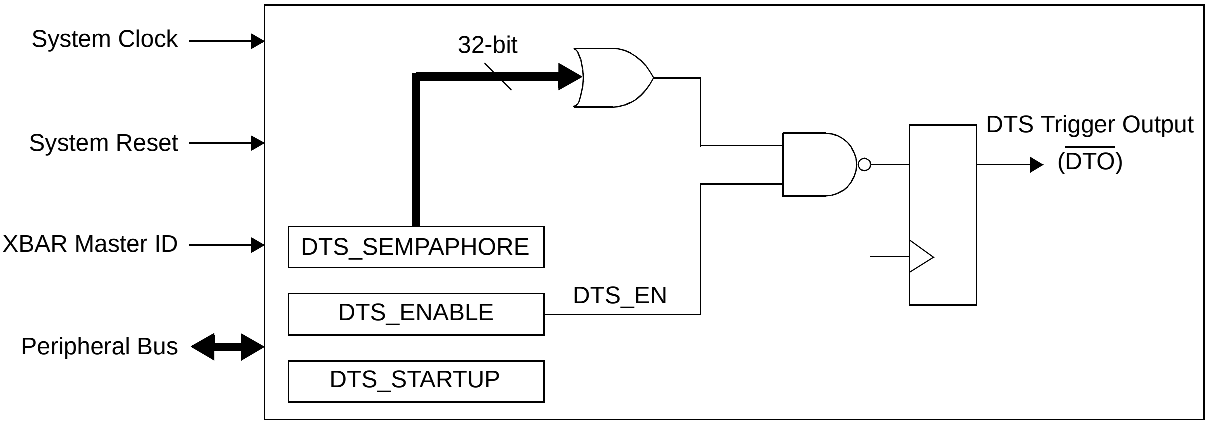


Figure 894. DTS block diagram

---

bh. When used as a component of a triggered data acquisition system, Nexus read/write access is via the JTAG interface of the Nexus debug port and is different than the data acquisition protocol defined in the IEEE-ISTO 5001-2003 or IEEE-ISTO 5001-2010 Nexus standards, which use the Nexus Auxiliary port.

The DTS Trigger Output (DTO) signal is connected to one of the EVTO inputs of the Nexus Port Controller (NPC). The other EVTO inputs to the NPC are connected to the other Nexus modules in the device. DTO is asserted when any bit in the DTS_SEMAPHORE register is set.

Note: When the DTS module is enabled (DTS_ENABLE[DTS_EN] = 0b1), the Nexus EVTO function of the EVTO pin is disabled and EVTO becomes the DTO. Unlike the EVTO function that only asserts for one clock, the DTO function remains asserted until the tool reads the DTS_SEMAPHORE register, clearing the register's contents.

Figure 895 shows the chain of events that begins with setting of any bit in the DTS_SEMAPHORE register and the clearing of the register caused by a Nexus read.

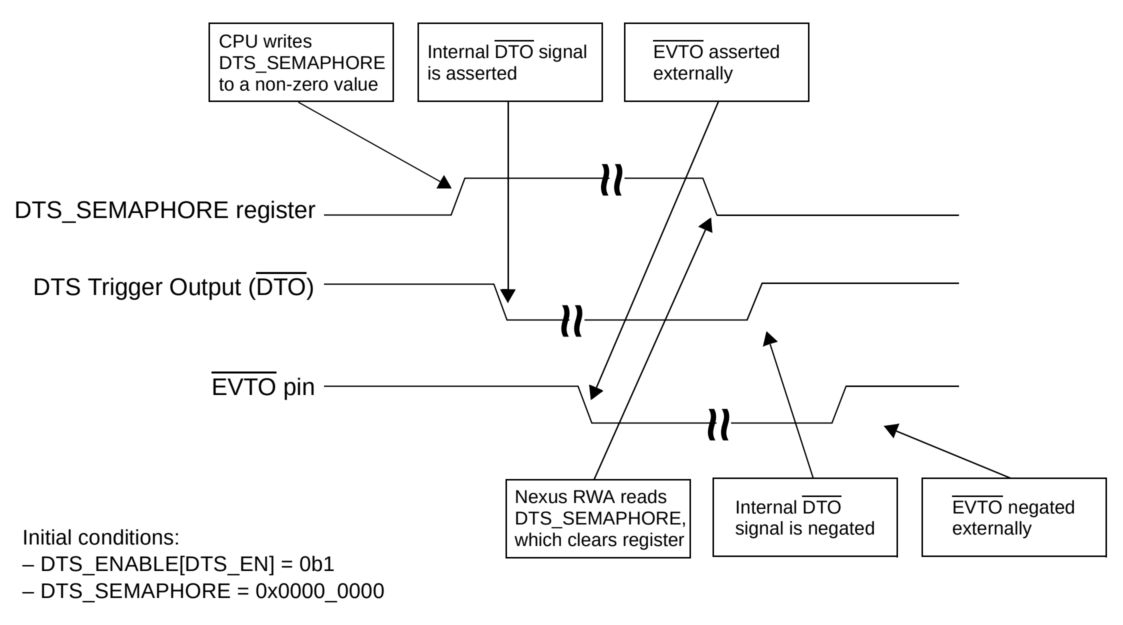


Figure 895. DTO event sequence

35.3 DTS device connections

The DTS module connects to the Peripheral Bridge (PBRIDGE) for access to the registers. The PBRIDGE is connected to a slave port of the Crossbar bus interface (XBAR). Connected to the XBAR master ports are the core (e200z4, with one master port for the Instruction and another for the Load/Store bus), the eDMA module, and the FlexRay module.

The registers have limited access as described in *Section 35.3.1, DTS register access*. Access is based on the XBAR Master ID of the accessing module. Access to the DTS_SEMAPHORE register is limited to the e200z4 core and the eDMA module and is restricted to only setting bits. Only an access via a Nexus Read/Write Access from an external tool through the Nexus/JTAG port of the device can clear bits in the DTS_SEMAPHORE register(bi). Similarly, the DTS_ENABLE and DTS_STARTUP registers can only be written via a Nexus Read/Write Access.

Note: Nexus Read/Write Accesses use the load/store bus of the core to perform accesses, but Nexus accesses have a different Master ID than normal core load/stores.

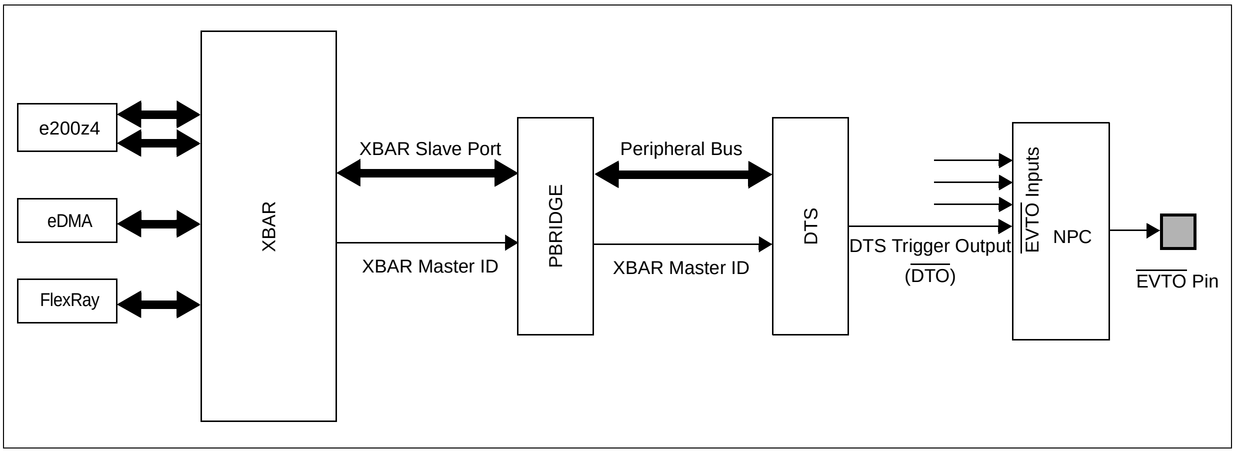


Figure 896. DTS device connections

35.3.1 DTS register access

A summary of accesses to all DTS registers by bus masters is provided in *Table 866*. Note that only proper 32-bit accesses are valid. The effect of write accesses, that are not 32 bits, is not defined.

Table 866. DTS register access effects

Register	32-bit Read				32-bit Write			
	RWA(1)	e200z4	eDMA	FlexRay	RWA(1)	e200z4	eDMA	FlexRay
DTS_ENABLE	Data	Data	Data	Data	Data	No effect	No effect	No effect
DTS_STARTUP	Data	Data	Data	Data	Data	No effect	No effect	No effect
DTS_SEMAPHORE	Data and Clear(2)	Data	Data	Data	No effect	Bit OR	Bit OR	No effect

1. Nexus Read/Write access via an external tool.
2. A read of the DTS_SEMAPHORE register by either Nexus Read/Write Access module is destructive and clears all bits in the register.

Access to DTS module registers is controlled based on the XBAR Master ID of the accessing module. The table below shows the XBAR Master IDs for each of port.

Note: The XBAR Master ID should not be confused with the Master Port number of the XBAR. See Chapter 9: Multi-Layer AHB Crossbar Switch (XBAR), for details.

Tools must access the DTS registers (DTS_ENABLE, DTS_STARTUP, and DTS_SEMAPHORE) through the Nexus Read/Write Access mechanism of the e200z4

---

bi. DTS_SEMAPHORE bits are cleared automatically when read through the Nexus/JTAG port.

core. JTAG accesses through either core appear as if the access is via the core and therefore will not have the same level of access as a Nexus R/W access.

35.4 Memory map

Table 867 shows the memory map of the Development Trigger Semaphore module registers. Three 32-bit registers are implemented. The rest of the memory map (0xC3F9_C00C through 0xC3F9_FFFF) is reserved.

Table 867. DTS Module

Address	Register	Description	Size (bits)	Access
DTS_BASE (0xC3F9_C000)	DTS_ENABLE	DTS output enable register	32	Restricted R/W(1)
DTS_BASE + 0x0004	DTS_STARTUP	DTS startup register	32	Restricted R/W(1)
DTS_BASE + 0x0008	DTS_SEMAPHORE	DTS semaphore register	32	Restricted R/W(1)
DTS_BASE + 0x000C – DTS_BASE + 0xFFFF	Reserved			

1. Only certain types of accesses are allowed. See separate description.

35.5 Register descriptions

35.5.1 DTS Output Enable Register (DTS_ENABLE)

This DTS_ENABLE register controls the DTS Trigger Output ($\overline{\text{DTO}}$) and whether $\overline{\text{DTO}}$ is active on the $\overline{\text{EVTO}}$ output pin of the device. *Figure 897* shows the format of the DTS_ENABLE register.

Note: *Access to the DTS_SEMAPHORE and DTS_STARTUP registers are unaffected by the state of this register.*

Figure 897. DTS_ENABLE register

Address: DTS_BASE+0x0000 Access: Restricted R/W(1)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	DTS_EN
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

1. The DTS_ENABLE register can be read by the e200z4 core, but can only be written by a Nexus Read Write Access (RWA).

Table 868. DTS_ENABLE field descriptions

Name	Description
31 DTS_EN	DTS Enable Controls whether the $\overline{\text{DTO}}$ signal is routed to the $\overline{\text{EVTO}}$ pin. 0: DTS output is disabled. 1: DTS output is enabled. Any bit set in the DTS_SEMAPHORE register will assert the DTS Trigger Output signal ($\overline{\text{DTO}}$). The DTS Enable bit is cleared by a device reset (either the assertion of the external $\overline{\text{RESET}}$ or by an internally generated reset). A JTAG reset does not change the state of this register.

35.5.2 DTS Startup Register (DTS_STARTUP)

The DTS_STARTUP register is used for tool detection and startup information exchange between the tool and software running on the microcontroller.

Figure 898. DTS_STARTUP register

Address: DTS_BASE+0x0004 Access: Restricted R/W(1)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	AD31	AD30	AD29	AD28	AD27	AD26	AD25	AD24	AD23	AD22	AD21	AD20	AD19	AD18	AD17	AD16
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	AD15	AD14	AD13	AD12	AD11	AD10	AD9	AD8	AD7	AD6	AD5	AD4	AD3	AD2	AD1	AD0
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1. The DTS_STARTUP register can be read by the e200z4 core, the eDMA module and Nexus but can only be updated by a Nexus Read Write Access (RWA).

Table 869. DTS_STARTUP field descriptions

Name	Description
AD[31:0]	Application Dependent register bits The bits have no defined meaning to the microcontroller. They are used to by an external tool to pass information, e.g., application options and status, to application software running on target microcontroller at startup time. Use a Nexus RWA 32-bit write access to update the contents of this register. A device reset (either from the $\overline{\text{RESET}}$ pin or an internally generated reset) clears all bits in the register. A JTAG reset does not change the contents of the register.

35.5.3 DTS Semaphore Register (DTS_SEMAPHORE)

The DTS_SEMAPHORE register is used by software to assert the DTO signal on the device EVTO pin. A 0b1 in any bit of this register causes the DTO signal on the EVTO pin to be driven low. The intended use of this register is for the DTO signal to notify tools that data is available. Individual bits are used to identify the specific data.

Figure 899. DTS_SEMAPHORE register

Address: DTS_BASE+0x0008 Access: Restricted R/W(1)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	ST31	ST30	ST29	ST28	ST27	ST26	ST25	ST24	ST23	ST22	ST21	ST20	ST19	ST18	ST17	ST16
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	ST15	ST14	ST13	ST12	ST11	ST10	ST9	ST8	ST7	ST06	ST5	ST04	ST3	ST02	ST1	ST0
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1. The e200z4 core and eDMA modules can set bits in the DTS_SEMAPHORE register but cannot clear them—writes by the core and eDMA are bitwise ORed to the contents of the register. Nexus can only read this register but all bits are cleared after the read operation.

Table 870. DTS_SEMAPHORE field descriptions

Name	Description
ST[31:0]	Semaphore Trigger When a core or eDMA writes a logical ‘1’ to a bit, the bit is set. A write of ‘0’ by the core or DMA does not change the state of the bit. – All register bits are set to ‘1’ by a device reset. – A JTAG reset does not change the state of this register. – The register can be accessed, with restrictions, by any core, DMA or any Nexus RWA. – For the core or DMA, only 32-bit write or read accesses are valid. – A core or DMA valid read access returns the current value of the register and leaves the register unchanged. 0: No flag 1: Flag is set

35.6 Example application

The calibration process of a new engine requires real-time access to calibration tables and the ability to update the tables in real-time(bj). The DTS module enables this capability by enabling software to assert a signal to an external device pin to notify an external tool that data is available. The tool can then retrieve the data.

In this type of application the DTS_SEMAPHORE register and DTS Trigger Output ($\overline{\text{DTO}}$) signal provide a mechanism to notify the calibration tool that the calibration variable or variables (or sets of measurements), up to 32, have been updated with new values and are available for the tool to access.

Note: It is the user’s responsibility to ensure that the tool has time to retrieve the data prior to that particular trigger being set a second time. It is also permissible to have multiple triggers active at the same time or for a second trigger to be set before a previous trigger has been serviced, as long as it is not the same trigger (unless it is acceptable to the tool to not receive every data set).

Figure 900 shows an example DTS startup sequence for an external real-time data acquisition system. The startup and synchronization sequence can be as simple or as complicated as the need requires. However, a typical startup sequence is as follows:

1. The DTS_STARTUP register is cleared by a power on reset or any CPU reset.
2. The tool writes a non-zero value to the DTS_STARTUP register.
3. The CPU (user application software) then reads the value of the DTS_STARTUP register. Based on this value, different initialization options can be selected. The bits can be used for any application specific definitions.
4. Since the DTS_SEMAPHORE register is cleared when the tool reads the current value. The tool should perform all necessary initialization before reading this register. The application software can then check that the DTS_SEMAPHORE register was cleared by the tool, to determine that it is safe to start using it for its intended raster trigger semaphore function.
5. An optional hand shake from the CPU can be used to inform the tool that the user software has detected that the tool is attached and the CPU has performed the proper initialization for the tool by writing a predefined value to the DTS_SEMAPHORE register (the example shown in the figure above uses 0xAAAA_AAAA—all A’s was used since it is unrealistic that 16 channels could be enabled very quickly after start up after a reset).

[bj] SPC564A70 devices also include an MMU modification feature, which enables real-time switching of calibration tables.

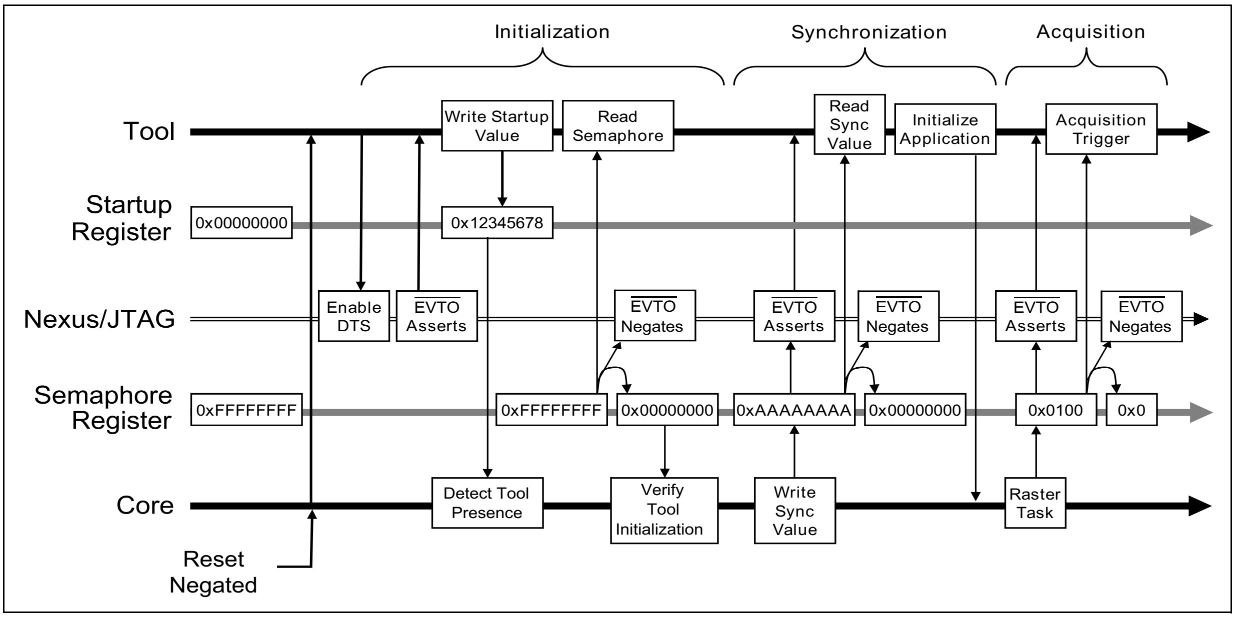


Figure 900. DTS startup sequence example

36 Reaction Module (REACM)

36.1 Introduction

The Reaction Module (REACM) is composed of 6 channels. Each channel contains three outputs. The primary application of this module is in the area of solenoid control for direct injection systems, valve control in automatic transmissions and others. It is connected to the on-chip ADC which monitors the current on the solenoid or valve. Based on that the reaction channel generates a PWM signal that modulates the current circulating in the solenoid or valve. It is a cost effective solution due to extensive sharing of several resources among channels and parameterized register banks for adequate dimensioning of resources and functionality.

36.1.1 Features

The REACM features include:

- Per-channel architecture for independent output control
- Interface with on-chip ADC for fast response times
- Hardware connection with on-chip timer channels with channel routing capability
- Innovative concept of Shared Modulation Control
- Innovative concept of dynamic timer allocation
- 3 outputs per channel to support different driver architectures
- Flexibility to operate based on timing and threshold
- On-the-fly capture of ADC result reference for fast calibration
- Open and short circuit monitoring capability

Note: DMA is not supported in Andorra devices.

36.1.2 Modes of operation

Programing Mode

After a reset is applied, the reaction module is in programming mode. In this mode all channels are disabled and outputs are at logic zero. Note that this state does not necessarily mean that zero is the neutral state for the channel load, so care must be taken in order to disconnect the channel from the load in this case.

In the programming mode the host CPU writes all module parameters including:

1. Modulation word data (see *Section 36.4.2, Modulation control words bank*)
2. Channel control data (see *Section 36.3.7, REACM Channel n Configuration Register (REACM_CHCRn)*)
3. Threshold data (see *Section 36.3.12, REACM Threshold Bank Register (REACM_THBK)*)
4. Timer bank data (see *Section 36.3.10, REACM Shared Timer Bank Registers (REACM_STBK)*)
5. Hold-off timer bank data (see *Section 36.3.11, REACM Hold-off Timer Bank Registers (REACM_HOTBK)*)
6. Timer router data (see *Section 36.3.9, REACM Channel n Router Register (REACM_CHRRn)*)
7. ADC router data (see *Section 36.3.4, REACM Threshold Router Register (REACM_THRR)*)

The last action to perform is to *enable* the channel, after which the channel is able to respond to timer signals and ADC data, thus able to perform modulation on the output pins. It is recommended to keep the timer signals inactive until all data to all reaction channel modules are programmed and all channels have been put in the *enabled* mode.

Low power mode

Coming out of reset all channels are in the disabled state. The channel may also be in low power mode depending on a parameter that configures the initial state of the MDIS in the REACM module configuration register (REACM_MCR) (see *Figure 904*). If REACM_MCR[MDIS] = 1 the module clock may be disabled allowing for a low power state. The low power mode is controlled either by REACM_MCR[MDIS] or by a global stop signal. There is no explicit clock gating implemented in hardware within the reaction module.

Note: Low power mode must be entered only when all channels are disabled by REACM_CHCRn[CHEN] = 00.

Channel modes

After a channel is in *enabled* mode that channel is also said to be in the *normal* mode of operation, which means it responds to timer signals from the timer inputs connected to the reaction module and also to ADC results received from the on-chip ADC module. Channel outputs are controlled in accordance with those inputs in order to perform an output modulation process. When performing a modulation the reaction channel is said to be in the *active state*. The modes a reaction channel can be in and the ability to execute a modulation related to the modes are:

- Disabled: The channel cannot execute modulation.
- Enabled: The channel is able to execute a modulation. It may be in the *Active* or *Inactive* state.
  - Inactive state: The channel is not executing a modulation.
  - Active state: The channel is executing a modulation.

Debug mode

The Reaction Module Debug operation is defined by bits FRZ and FREN in the REACM module configuration register (REACM_MCR) (see *Figure 904*). In debug mode all timers are halted, including the timers in the Shared Time Bank and Hold-off Timers.

The module can enter debug mode either by software control or by the hardware debug input signal controlled by the chip logic. In both cases the reaction module only enters debug mode if enabled by bit REACM_MCR[FREN]:

- If the FREN bit and the FRZ bit are both is asserted the module enters debug mode.
- If the FREN bit is asserted and a global debug signal is issued the module enters debug mode.

In debug mode, the channel outputs are held at HOD (High Output Drive), LOD (Low Output Drive), or Drive Off (DOFF) state, as determined by the current channel state. When resuming normal operation after exiting debug mode the channel output is set to DOFF until the next timer control rising edge occurs.

The ADC Maximum Limit Detection (REACM_CHSRn[MAXL]) flag is the only flag that operates in debug mode. All other flags keep the state present when the module entered debug mode. Note that the corresponding error flag in the REACM Global Error Flag Register (REACM_GEFR) (*Figure 908*) is also set.

The REACM ADC Sensor Input Register (REACM_SINR) allows direct access for write to the TAG and ADC result values input to the reaction module. This software control may be used for module debug purposes. Please see *Figure 907*.

36.1.3 Block diagram

Figure 901 shows the on-chip connections of the reaction module. The ADC module is the source of data monitored from external sensors, usually voltage information, which is used by the reaction channel modulation to generate an PWM signal control signal in order to maintain the load current within a certain predefined boundary. The on-chip eTPU module is commonly used as the source for controlling the activation of a reaction channel. Alternatively an eMIOS or PIT module can be used to provide that control signal.

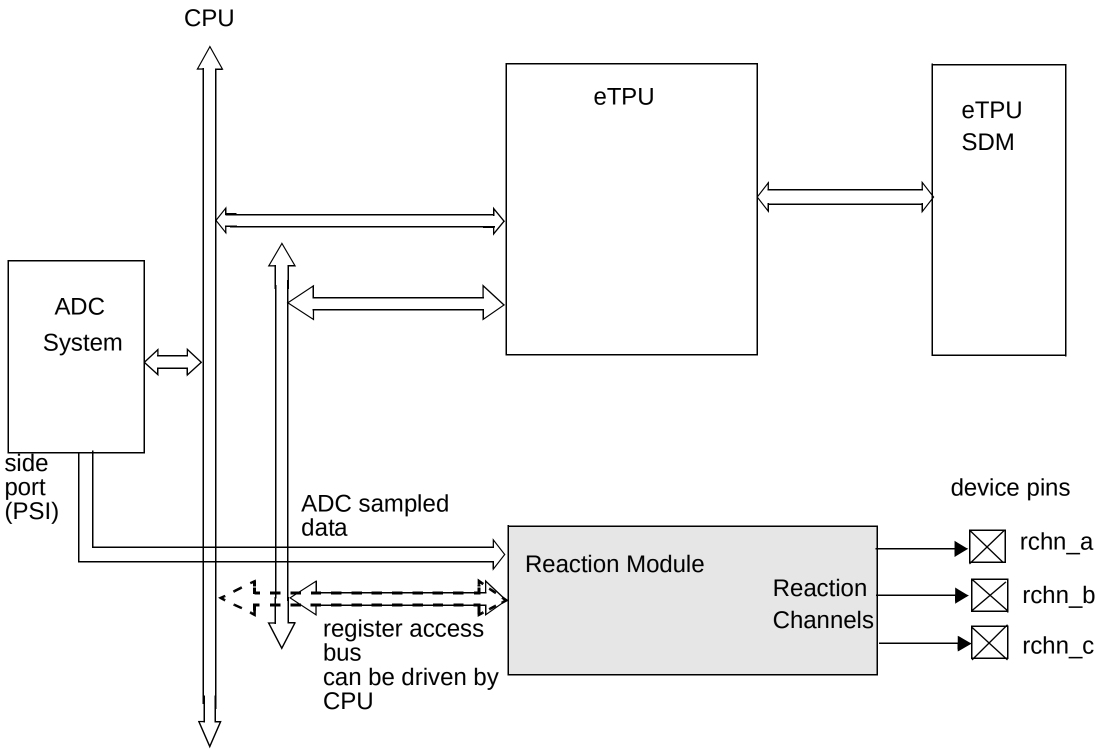


Figure 901. Reaction module system interconnection

Each reaction channel has three independently programmable outputs. The outputs do not have a predefined ON or OFF state meaning that the definition of ON and OFF depends upon the application.

Figure 902 illustrates the interconnection between a reaction channel and the blocks in the reaction module. The indexes next to the blocks indicate the order in which each submodule executes an action during the modulation process. A simplified sequence of these steps is described as follows:

1. A new ADC result from the eQADC module is received, triggering one reaction channel.
2. The triggered reaction channel generates the Modulation Word address based on the data programmed in the internal configuration register.
3. A Modulation word is selected by the reaction channel.
4. The selected Modulation Word generates the address for the Threshold Bank.
5. The Threshold bank selected data is compared against the incoming ADC result.
6. The comparison result is sent back to the channel.
7. The reaction channel executes modulation action and updates the channel output.

In order to execute steps 1 through 7 the reaction module takes five module clock cycles.

The reaction channel sets the outputs according to fields HOD or LOD provided by the selected modulation word.

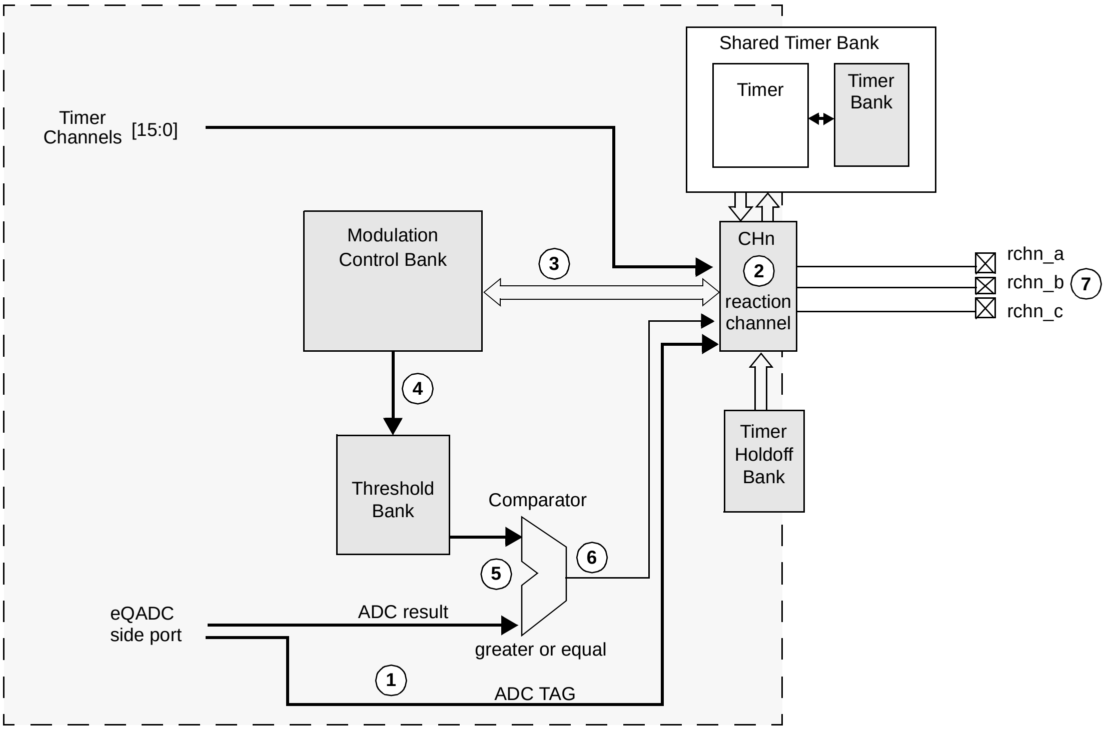


Figure 902. Channel interaction with internal submodules

Figure 903 shows the reaction module internal architecture. The channels are controlled by a Modulation Control Word provided by the Modulation Control Bank submodule. The Modulation Word has information about the type of modulation to be executed as well as the addresses for the Threshold and Timer banks. The channel stores information such as the address of the modulation word and initial values for the output pins after the channel is *enabled*.

Note that the modulation process only initiates after the channel is moved to the *active* state. The following sections describe this process in more detail.

After the activation, the channel sets the outputs with a predefined value stored in the REACM Channel n Configuration Register (REACM_CHCRn). Only after this initialization process is the channel ready to start a modulation process. Note that the channel only considers an ADC result as valid if it is within a window defined by the timer input, usually connected to the on-chip eTPU. The reaction channel can access more than one modulation word by incrementing the internal MODULATION_ADDR field in the REACM_CHCRn. The original value of this field is preserved, thus when a new modulation cycle is initiated the same modulation address value may be used.

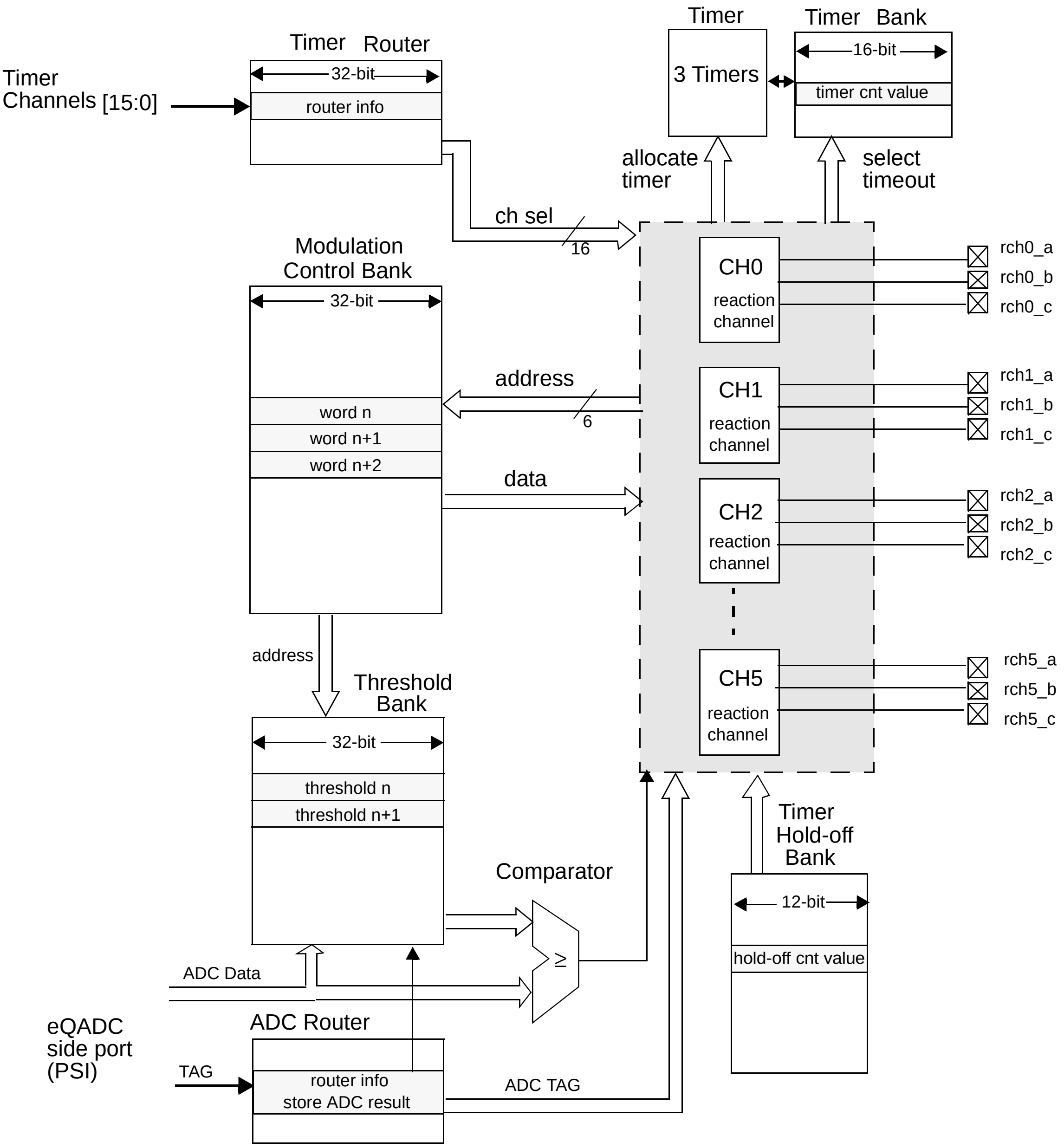


Figure 903. Reaction module block diagram

Table 871 lists the SPC564A70 reaction module outputs.

Table 871. Reaction module outputs

Reaction channel	Output pin
rch0_a	eTPU14

Table 871. Reaction module outputs (continued)

Reaction channel	Output pin
rch0_b	eTPU20
rch0_c	eTPU21
rch1_a	eTPU15
rch1_b	eTPU9
rch1_c	eTPU10
rch2_a	eTPU16
rch2_b	eMIOS2
rch2_c	eMIOS4
rch3_a	eTPU17
rch3_b	eMIOS10
rch3_c	eMIOS11
rch4_a	eTPU18
rch4_b	eTPU11
rch4_c	eTPU12
rch5_a	eTPU19
rch5_b	eTPU28
rch5_c	eTPU29

Timer channels, such as eTPU channel outputs, are connected to the Channel Router. This submodule routes each timer channel to a reaction channel. Note that one timer channel can be routed to more than one reaction channel.

The modulation process starts when an ADC result arrives and the time window is active. The ADC Router sends a trigger signal to all channels indicating that an ADC result is available. The channel address resolution is based on the TAG field received with the incoming ADC result. After decoding the TAG field the channel accesses the modulation word using the information stored in the REACM_CHCRn.

The Threshold Bank submodule stores values to be used on the comparison with incoming ADC results. The address for the stored values is generated by the Modulation Control Word Bank. This address generation is actually executed in a two-step process since the modulation word is first addressed by the channel which then generates the address for the Threshold Bank. After having both inputs defined the Comparator generates the comparison result back to the channels.

The Hold-off Timer Bank address is also stored in the modulation word. The reaction channel uses that information for the modulation process which requires the output to remain OFF during a certain amount of time. The hold-off counter itself is located inside each one of the channels.

36.2 Signal description

Table 872 shows the chip-level signals for the Reaction Module.

Table 872. Signal properties

Name	Function
RCHn_a	Output pin 'a' of reaction channel 'n'
RCHn_b	Output pin 'b' of reaction channel 'n'
RCHn_c	Output pin 'c' of reaction channel 'n'

36.2.1 REACM_RCHn — REACM Channel (n) Output Pin a, b and c

Each reaction channel provides three outputs to be connected to device primary output PADs. These outputs can be configured as necessary in order to meet application requirements. Thus the active state as well as overall functionality are defined by the registers within the Reaction Module and the defined sequence of operation. The reset state for these outputs is zero.

36.3 Memory map and register definition

This section provides a detailed description of all Reaction Module registers accessible to the end user.

36.3.1 Module memory map

Table 873 presents the reaction module memory map.

Table 873. Reaction module memory map

Offset from REACM base address (0xC3FC_7000)	Register name	Size in words	Location
0x0000	REACM module configuration register (REACM_MCR)	1	*on page 36-1557*
0x0004	REACM Timer Configuration Register (REACM_TCR)	1	*on page 36-1558*
0x0008	REACM Threshold Router Register (REACM_THRR)	1	*on page 36-1559*
0x000C	Reserved		
0x0010	REACM ADC Sensor Input Register (REACM_SINR)	1	*on page 36-1560*
0x0014 – 0x001F	Reserved		
0x0020	REACM Global Error Flag Register (REACM_GEFR)	1	*on page 36-1561*
0x0024 – 0x00FF	Reserved		

Table 873. Reaction module memory map (continued)

Offset from REACM base address (0xC3FC_7000)	Register name	Size in words	Location
0x0100 + (n*0x10)	REACM Channel n Configuration Register (REACM_CHCRn) (n = 0–5)	6	*on page 36-1562*
0x0100 + (n*0x10 + 0x4)	REACM Channel n Status Register (REACM_CHSRn) (n = 0–5)	6	*on page 36-1565*
0x0100 + (n*0x10 + 0x8)	REACM Channel n Router Register (REACM_CHRRn) (n = 0–5)	6	*on page 36-1568*
0x0100 + (n*0x10 + 0xC)	Reserved (n = 0–5)		
0x0160 – 0x02FF	Reserved		
0x0300 – 0x0308	REACM Shared Timer Bank Registers (REACM_STBK)	3	*on page 36-1569*
0x030C – 0x037F	Reserved		
0x0380 – 0x0388	REACM Hold-off Timer Bank Registers (REACM_HOTBK)	3	*on page 36-1570*
0x038C – 0x03FF	Reserved		
0x0400 – 0x045C	REACM Threshold Bank Register (REACM_THBK)	24	*on page 36-1570*
0x0460 – 0x05FF	Reserved		
0x0600	REACM ADC result maximum limit check register (REACM_ADCMAX)	1	*on page 36-1571*
0x0604 – 0x067F	Reserved		
0x0680	REACM Modulation Range Pulse Width Register (REACM_RANGEPWD)	1	*on page 36-1572*
0x0684 – 0x06BF	Reserved		
0x06C0	REACM Modulation Minimum Pulse Width Register (REACM_MINPWD)	1	*on page 36-1573*
0x06C4 – 0x06FF	Reserved		
0x0700 – 0x072C	REACM Modulation Control Word Bank Registers (REACM_MWBK)	12	*on page 36-1574*
0x0730 – 0x0FFF	Reserved		

36.3.2 REACM module configuration register (REACM_MCR)

The REACM module configuration register (REACM_MCR) contains the control bits to configure the general operation of the Reaction Module.

Address: REACM_BASE (0xC3FC_7000) + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	MDIS	FRZ	0	FREN	TPREN	HPREN	GIEN	OVREN	0	0	0	0	0	0	0
W	OVRC															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 904. REACM module configuration register (REACM_MCR)

Table 874. REACM_MCR field descriptions

Field	Description
0 OVRC	Overrun Detection Flag Clear The OVRC clears the OVR flag if write 0x1. This bit reads always as 0x0. If a set event occurs at the same time a flag clear is done, the set event has precedence over the clear thus the flag remains set. 0 No action 1 Clears OVR bit
1 MDIS	Module Disable The MDIS bit puts the Reaction Module in low power mode. Communication through the slave-bus Interface is ignored in this mode except writes to the REACM_MCR which are allowed, except for the FRZ and FREN bits. The global debug signal state is not changed internally while in low power mode. 0 Normal Mode 1 Low Power Mode
2 FRZ	Freeze Control The FRZ bit controls the state of the Reaction Module regarding debug operation. If FREN bit is asserted and FRZ bit is also asserted the module enters debug mode. In this mode all time bases are halted and the channels outputs are controlled solely by software. See *Section , Debug mode*. This bit cannot be written if MDIS bit is asserted or when the Reaction Module is in stopped by a device stop request. 0 Normal Mode 1 Debug Mode
3	Reserved, should be cleared.
4 FREN	Freeze Enable The FREN bit enables the Reaction Module to enter debug mode. The debug mode is controlled either by the FRZ bit or by a global debug signal. This bit cannot be written if MDIS bit is asserted or when the Reaction Module is in stopped by a device stop request. 0 Debug Mode disabled 1 Debug Mode enable
5 TPREN	Timer Prescaler Enable The TPREN bit enables the Shared Timer Prescaler in the Reaction Module. 0 Prescaler Disabled 1 Prescaler Enabled

Table 874. REACM_MCR field descriptions (continued)

Field	Description
6 HPREN	Hold-off Prescaler Enable The HPREN bit enables the Hold-off Prescaler in the Reaction Module. 0 Prescaler Disabled 1 Prescaler Enabled
7 GIEN	Global Interrupt Enable The GIEN bit enables the assertion of the interrupt request to the CPU when any of the channel flags or the OVR flag are set. The channel error flag bits are: MAXL, OCDF, SCDF and TAER. Note that for the interrupt to be asserted these flag bits need also to be enabled by the corresponding enable bit defined in *Section 36.3.7, REACM Channel n Configuration Register (REACM_CHCRn)*. 0 Interrupt disabled 1 Interrupt enabled The GIEN bit only affects the general interrupt signal, and not the individual channel interrupts. GIEN=0 does not inhibit the channel interrupts.
8 OVREN	Overrun Detection Interrupt Enable The OVREN enables the OVR flag, when set, to generate a global interrupt request for the CPU. 0 Interrupt disabled 1 Interrupt enabled
9–31	Reserved, should be cleared.

36.3.3 REACM Timer Configuration Register (REACM_TCR)

The REACM Timer Configuration Register (REACM_TCR) contains the prescaler settings to configure the operation of the Reaction Module Holdoff Timer and Shared Timers. It is recommended to change the value of the prescalers either when the prescalers are disabled by its control enable bits HPREN and TPREN in the REACM_MCR or when the counters are not being used by any channel. Note that the prescalers are completely independent, thus modifying HPRE does not affect TPRE and modifying TPRE does not affect HPRE.

Address: REACM_BASE (0xC3FC_7000) + 0x0004 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	HPRE[11:0]											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	TPRE[7:0]							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 905. REACM Timer Configuration Register (REACM_TCR)

Table 875. REACM_TCR field descriptions

Field	Description
0–3	Reserved, should be cleared.
4–15 HPRE [11:0]	Hold-off Timer Prescaler The HPRE[11:0] field defines the rate of the Hold-off Timers on each reaction channel. If its value is zero the prescaler is bypassed thus the Hold-off timer operates at the module clock rate. If HPRE = 0x01 the module operates at module clock divide by two and so forth up to HPRE = 0xFFF which defines system clock divided by 4096.
16–23	Reserved, should be cleared.
24–31 TPRE [7:0]	Timer Prescaler The TPRE[7:0] field defines the rate of the Timers on the Timer bank. If its value is zero the prescaler is bypassed thus the Timer operates at the module clock rate. If TPRE = 0x01 the Timer operates at module clock divide by two and so forth up to TPRE = 0xFF which defines module clock divided by 256 as the frequency of operation for the Timer.

36.3.4 REACM Threshold Router Register (REACM_THRR)

The REACM Threshold Router Register (REACM_THRR) routes the ADC result to the Threshold Bank. Since a TAG is assigned for each ADC result coming from the on-chip ADC module, this TAG is specified in this register and used for the routing process. These ADC result is written to the Threshold Value Bank as soon as it is received by the Reaction Module. Note that THRADC0 and THRADC1 TAG values may also be used by the reaction channels. In this case the results are routed to both, the channel and the Threshold Bank.

Note: *Due to the timing in which these parallel events occur, if the channel uses the same Threshold Bank address in which the incoming ADC result was written to, this ADC result is used for the channel modulation being executed.*

Address: REACM_BASE (0xC3FC_7000) + 0x0008 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	WREN1	WREN0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	THRADC1[3:0]				0	0	0	0	THRADC0[3:0]			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 906. REACM Threshold Router Register (REACM_THRR)

Table 876. REACM_THRR field descriptions

Field	Description
0–5	Reserved, should be cleared.
6 WREN1	Write Enable Bit for THRADC1 The WREN1 write enable bit 1 controls if the ADC result having a TAG matching to THRADC1 field will be written into address one of the Threshold Bank. 1 Write received ADC result to Threshold bank address one 0 Do not write received ADC result to Threshold bank
7 WREN0	Write Enable Bit for THRADC0 The WREN0 write enable bit 0 controls if the ADC result having a TAG matching to THRADC0 field will be written into address zero of the Threshold Bank. 1 Write received ADC result to Threshold bank address zero 0 Do not write received ADC result to Threshold bank
8–19	Reserved, should be cleared.
20–23 THRADC1 [3:0]	ADC result Router value for Threshold Bank address one The THRADC1[3:0] field controls the routing from the received ADC result to the Threshold Bank address one. Any ADC result which TAG matching THRADC1 will be routed to Threshold Bank.
24–27	Reserved, should be cleared.
28–31 THRADC0 [3:0]	ADC result Router for Threshold Bank address zero The THRADC0[3:0] field controls the routing from the received ADC result to the Threshold Bank address zero. Any ADC result which TAG matching THRADC0 will be routed to Threshold Bank.

36.3.5 REACM ADC Sensor Input Register (REACM_SINR)

The REACM ADC Sensor Input Register (REACM_SINR) is used to monitor the ADC Interface (see *Section 36.4.6, ADC interface*) and allows the software to define the ADC result and TAG values for the Reaction Module. Also, ADC values captured after filtering can be transferred to the Reaction Module by the CPU. Writes to this register overwrite any value coming from the ADC interface, thus it should be used when no ADC conversion has the Reaction Module as the target. Writing to this register triggers the reaction channel selected by the TAG value to execute a comparison and to evaluate an eventually new value for the channel outputs.

Address: REACM_BASE (0xC3FC_7000) + 0x0010 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0			0	0	0	0	ADC_TAG[3:0]			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	ADC_RESULT[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 907. REACM ADC Sensor Input Register (REACM_SINR)

Table 877. REACM_SINR field descriptions

Field	Description
0–11	Reserved, should be cleared.
12–15 ADC_TAG [3:0]	TAG value The ADC_TAG[3:0] represents the TAG for the ADC conversion which is used by the Reaction Module to select the reaction channel to execute the modulation.
16–31 ADC_RESULT [15:0]	ADC conversion result value The ADC_RESULT[15:0] represents the value resulting from an ADC conversion. This value is used for the Reaction Channel Modulation process or for capturing by the Threshold Bank.

36.3.6 REACM Global Error Flag Register (REACM_GEFR)

The REACM Global Error Flag Register (REACM_GEFR) helps the software in the resolution of error conditions signaled by the reaction channels. This allows a faster service in error handling. This register mirrors the Error Flag in the REACM Channel n Status Register (REACM_CHSRn).

Address: REACM_BASE (0xC3FC_7000) + 0x0020 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	OVR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	EF5	EF4	EF3	EF2	EF1	EF0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 908. REACM Global Error Flag Register (REACM_GEFR)

Table 878. REACM_GEFR field descriptions

Field	Description
0 OVR	Overrun Detection Flag The OVR flag is used to indicate that an overrun condition was detected at the ADC Interface. See *Section 36.4.6, ADC interface*. 1 Overrun detected 0 Overrun not detected
1–25	Reserved, should be cleared.
26–31 EFn	Error Flag The EFn error flag bit indicates an error condition occurred in channel n. This bit is provided for a fast channel error resolution in the Reaction module. The condition could be any error indicated by MAXL, OCDF, SCDF, SQER, RAER, or TAER error flags as described in *Section 36.3.8, REACM Channel n Status Register (REACM_CHSRn)*. The EFn bit is automatically cleared if the corresponding channel flags are all cleared.

36.3.7 REACM Channel n Configuration Register (REACM_CHCRn)

The REACM Channel n Configuration Register (REACM_CHCRn) controls the channel behavior.

Address: REACM_BASE (0xC3FC_7000) + 0x0100 + (n* 0x0010 + 0x0000) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CHEN[1:0]		SWMC	MAXLEN	OCDFEN	SCDFEN	TAEREN	SQEREN	RAEREN	DMAEN	CHOFF	0	0	DOFF[2:0]		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	BSB[2:0]			0	0	MODULATION _ADDR[5:0]					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 909. REACM Channel n Configuration Register (REACM_CHCRn)

Table 879. REACM_CHCRn field descriptions

Field	Description
0–1 CHEN[1:0]	Channel Enable Bits The CHEN[1:0] bits control the activation of the reaction channel. Once changed from *disabled*, CHEN = 00, to any *enabled* state, the channel reads the first Modulation Word and initializes itself in order to be ready to execute a modulation as soon as a timer control window is detected or SWMC = 1, if software modulation control is selected by CHEN = 11. 00 Channel disabled, meaning that it does not execute any modulation even if a timer window is detected or SWMC is made active. In this mode DOFF field in the REACM_CHCR defines the state of the channel outputs. 01 Channel is enabled for timer control only, meaning that as soon as a timer window is detected a modulation sequence starts. 10 Reserved 11 Channel enabled for software control only, meaning that as soon as SWMC bit is set a modulation sequence starts. If CHEN is programed with 01 or 11 enabling a channel, and the eTPU time window is already active (or SWMC = 1), the reaction channel disregards this window and waits until the next window activation in order to start the modulation process by moving to the active state. In order to start a modulation controlled by software, it is necessary to first write CHEN = 1 and only after that write SWMC = 1.
2 SWMC	Software Modulation Control bit If this bit is set, the channel initiates a modulation. It is equivalent to the assertion of an eTPU channel connected to the Reaction module. In order for this functionality to be used it is required that CHEN[1:0] = 11. 1 Channel executes modulation 0 Channel does not perform modulation

Table 879. REACM_CHCRn field descriptions (continued)

Field	Description
3 MAXLEN	ADC result Maximum Limit Interrupt Enable bit The MAXLEN enables the MAXL flag, when set, to generate an interrupt request for the CPU. 1 Interrupt enabled 0 Interrupt disabled
4 OCDFEN	OCDF Interrupt Enable bit The OCDFEN bit enables the OCDF bit to issue an interrupt request. 1 OCDF interrupt enabled 0 OCDF interrupt disabled
5 SCDFEN	SCDF Interrupt Enable bit The SCDFEN bit enables the SCDF bit to issue an interrupt request. 1 SCDF interrupt enabled 0 SCDF interrupt disabled
6 TAEREN	TAER Interrupt Enable bit The TAEREN bit enables the TAER bit to issue an interrupt request. 1 TAER interrupt enabled 0 TAER interrupt disabled
7 SQEREN	Modulation Word Sequence Error Interrupt Enable The SQEREN bit enables the SQER flag to generate an interrupt request. 1 SQER interrupt enabled 0 SQER interrupt disabled
8 RAEREN	Resource Allocation Error Interrupt Enable The RAEREN bit enables the RAER flag to generate an interrupt request. 1 RAER interrupt enabled 0 RAER interrupt disabled
9 DMAEN	Direct Memory Access Enable bit The DMAEN bit enables a DMA request by the channel when an Modulation Word sequence advance occurs. 1 Enables channel DMA request 0 Disables channel DMA request
10 CHOFF	Output Disable bit The CHOFF bit allows disabling the output pin. DOFF[2:0] field is used as disable state for the channel outputs. The output pins will set to DOFF value if CHOFF is asserted. Note that disabling the channel through CHOFF does not disable the channel operation, only the channel outputs are forced to DOFF state. 1 Output Disable Enabled 0 Output Disable Disabled
11–12	Reserved, should be cleared.

Table 879. REACM_CHCRn field descriptions (continued)

Field	Description
13–15 DOFF[2:0]	Drive Off Control field The DOFF[2:0] field defines the reaction channel output disabled state. This condition is achieved either when CHEN = 00, or after CHEN activation, or when Output is disabled by CHOFF in the Channel Configuration register. It is possible to control the channel outputs directly through the software by writing to the DOFF bits. Refer to *Table 880*. In this case the channel should be in the disabled state, CHEN = 00. If the DOFF value is changed just after the channel is enabled, it is not assured the new DOFF value is immediately used in the channel output.
16–20	Reserved, should be cleared.
21–23 BSB[2:0]	Bank Support Bits The BSB[2:0] provides control for a banked mode operation of the Reaction Module. Each bit in this field controls the channel outputs ch*n*_c, ch*n*_b and ch*n*_a respectively. When asserted the channel output implements an OR with the corresponding output of the subsequent channel.
24–25	Reserved, should be cleared.
26–31 MODULATION_ADDR [5:0]	Address for Modulation Control Bank The MODULATION_ADDR[5:0] field has the address of the Modulation Word in the Modulation Control Bank. This address is used as a base address for the first modulation word. The reaction channel can access subsequent words in the Modulation Control Bank by incrementing the Modulation Address field. Note that this field is not modified by the reaction channel in this process.

Table 880. Output assignment through DOFF

DOFF	Channel output	Comments
DOFF[0]	CHn_a	DOFF[0] bit defines CHn_a output pin value
DOFF[1]	CHn_b	DOFF[1] bit defines CHn_b output pin value
DOFF[2]	CHn_c	DOFF[2] bit defines CHn_c output pin value

36.3.8 REACM Channel n Status Register (REACM_CHSRn)

The REACM Channel n Status Register (REACM_CHSRn) provides access to the channel flags and flag clear bits. It also provides access to the current values of the channel output being driven and the Modulation Word being accessed by the channel.

Address: REACM_BASE (0xC3FC_7000) + 0x0100 + (n* 0x0010 + 0x0004) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	MODACT	MAXL	OCDF	SCDF	TAER	SQER	RAER	CHOUT[2:0]			0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	MODULATION_POINTER[5:0]					
W				MAXLC	OCDFC	SCDFC	TAERC	SQERC	RAERC							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Figure 910. REACM Channel n Status Register (REACM_CHSRn)

Table 881. REACM_CHSRn field descriptions

Field	Description
0–1	Reserved, should be cleared.
2 MODACT	Modulation Active Flag The MODACT flag indicates that the channel is *enabled* and *active* thus executing a modulation cycle. 1 Channel is *active* 0 Channel is not *active*
3 MAXL	ADC Maximum Limit Detection Flag The MAXL flag indicates that the ADC result which TAG is addressing this channel achieved or passed the maximum allowed limit specified in the ADCMAX register. This flag is set only if the channel is in the *active state* (see *Section , Channel modes*). 1 ADC Maximum Limit Detected 0 Normal operation
4 OCDF	Open Circuit Detection Flag The OCDF flag indicates that an open circuit was detected in the channel load using the PWM monitored modulation, as described in *Section 36.6, Monitored modulation*. This Flag is set only of the channel is in the *active* state. 1 Open Circuit Detected 0 Normal operation
5 SCDF	Short Circuit Detection Flag The SCDF flag indicates that a short circuit was detected in the channel load using the PWM monitored modulation, as described in *Section 36.6, Monitored modulation*. This Flag is set if the channel activation window signal is set (eTPU channel or SWMC bit). 1 Short Circuit Detected 0 Normal Operation

Table 881. REACM_CHSRn field descriptions (continued)

Field	Description
6 TAER	Timer Allocation Error The Timer Allocation Error bit indicates that the channel tried to allocate a timer counter in the Shared Timer Bank without success. This situation is an indication that the Timer resources available in the module are not sufficient to execute the required functionality. This error indication is used during software development and should not occur during normal use of the module since it may result in incorrect operation. This Flag is set only of the channel is in the *active* state. 1 Error occurred during timer allocation 0 No error occurred during timer allocation.
7 SQER	Modulation Word Sequence Error Flag The SQER bit indicates a Modulation Sequence error occurred, meaning that the time window which defines a modulation cycle ended in a premature modulation phase. The correct modulation phase for the time window to close is when SM field in the Modulation Word indicates *no advance*, or SM = 00. This Flag is set only of the channel is in the *active* state. 1 Modulation Sequence Error occurred 0 Modulation Sequence Error did not occur.
8 RAER	Resource Allocation Error Flag The RAER bit indicates that a resource allocation error occurred. The possible allocation errors are: Modulation Control Word address is out of available range (including if the MODULATION_POINTER increments to an inexistent MCW address or wraps to 0x0), Threshold Value Bank address is out of available range, Hold-off Timer Bank address is out of available range, Shared Timer Bank address is out of available range, Channel Input Router points to an inexistent eTPU channel, and when the Hold-off timer is select for both modulation and sequence advance at the same time (i.e, SM = 10 and MM = 01). Note that the amount of hardware resources is configuration dependent thus may vary according to module instantiations in the SoC. This flag is intended to help on the debug of the Reaction Module during software development and can be set only if the channel is in the *enabled* state. 1 Allocation error occurred 0 No allocation error occurred The condition that sets the RAER bit must be resolved prior to clear the bit, otherwise the bit can be set again.
9–11 CHOUT[2:0]	Channel Output Monitoring bits The CHOUT[2:0] Channel Output Monitoring bits provides for the software the ability to monitor the output provided by the channel. This data is not buffered thus represents the channel output at the time the CPU read access is done. CHOUT[0] corresponds to the chn_a output, CHOUT[1] corresponds to the chn_b output and CHOUT[2] corresponds to the ch*n*_c output pin. These bits are available independent of the channel mode or state.
12–18	Reserved, should be cleared.
19 MAXLC	ADC Maximum Limit Flag Clear The MAXLC clears the MAXL flag if write 0x1. This bit reads always as 0x0. If a set event occurs at the same time a flag clear is done, the set event has precedence over the clear thus the flag remains set. 1 Clears MAXL bit 0 No action

Table 881. REACM_CHSRn field descriptions (continued)

Field	Description
20 OCDFC	Open Circuit Detection Flag Clear The OCDFC clears the OCDF flag if write 0x1. This bit reads always as 0x0 1 Clears OCDF bit 0 No action
21 SCDFC	Short Circuit Detection Flag clear The SCDFC bit clears the SCDF flag if write ‘1’. This bit is self negated thus read always as ‘0’. If a set event occurs at the same time a flag clear is done, the set event has precedence over the clear thus the flag remains set. 1 Clears SCDF 0 No action
22 TAERC	TAER Clear bit The TAERC bit clears the TAER bit if write ‘1’. This bit is self-negated thus reads always as ‘0’. If a set event occurs at the same time a flag clear is done, the set event has precedence over the clear thus the flag remains set. 1 Clears TAER 0 No action
23 SQERC	Modulation Word Sequence Error Flag clear The SQERC bit clears the SQER flag in the Channel Status register. 1 SQER flag is cleared 0 SQER flag is not cleared
24 RAERC	Resource Allocation Error Flag clear The RAERC bit clears the RAER flag in the Channel Status register. 1 RAER flag is cleared 0 RAER flag is not cleared
25	Reserved, should be cleared.
26–31 MODULATION_ POINTER[5:0]	Modulation bank address generated by the channel The MODULATION_POINTER[5:0] gives to the software the information of the current address being generated by the channel to access the Modulation Word. Note that the address increments from a base address stored in the channel, MODULATION_ADDR[5:0]. This register is not buffered thus represents the current address being generated.

36.3.9 REACM Channel n Router Register (REACM_CHRRn)

The REACM Channel n Router Register (REACM_CHRRn) controls the channel interconnection to external timers, such as eTPU and eMIOS and ADC result data. The channels have access to any 16 timer inputs and any ADC sample tag from 0 to 15. Note that this architecture allows several reaction channels to point to the same timer channel or to the same ADC result.

Address: REACM_BASE (0xC3FC_7000) + 0x0100 + (n* 0x0010 + 0x0008) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	ADCR[3:0]			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	CHIR[3:0]			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 911. REACM Channel n Router Register (REACM_CHRRn)

Table 882. REACM_CHRRn field descriptions

Field	Description
0–11	Reserved, should be cleared.
12–15 ADCR[3:0]	ADC result router field The ADCR[3:0] field selects which ADC result is used by the reaction channel for the modulation. The TAG[3:0] received along with the ADC result is used for comparison with the ADCR[3:0] field in order to define if the received result is used by the reaction channel.
16–27	Reserved, should be cleared.
28–31 CHIR[3:0]	Channel Input router field The CHIR[3:0] field selects which eTPU channel is used by the reaction channel for the modulation. See *Table 883* for valid values.

Table 883. REACM_CHRRn[CHIR] values

CHIR[3:0]	eTPU A channel
0b0000	10
0b0001	11
0b0010	12
0b0011	13
0b0100	14
0b0101	15
0b0110	16
0b0111	17
0b1000	18
0b1001	19
0b1010	20
0b1011	21

Table 883. REACM_CHRRn[CHIR] values (continued)

CHIR[3:0]	eTPU A channel
0b1100 to 0b1111	Reserved

36.3.10 REACM Shared Timer Bank Registers (REACM_STBK)

The REACM Shared Timer Bank Registers (REACM_STBK) is a set of registers which define the values used by the Reaction Module Timer. The timer values are programmed by the host CPU during the configuration of the Reaction Module. Modulation Word accessed by the reaction channel contains the address of a specific Timer value stored in the Shared Timer bank. By selecting a Timer value the reaction channel also selects and enables a counter. When this counter reaches the selected Timer value a timeout indication is generated for the reaction channel that initiated the counter. This event is used, for example, to indicate that a next Modulation Word should be used for the modulation.

Address: REACM_BASE (0xC3FC_7000) + (from 0x0300 to 0x0308) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SHARED_TIMER[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 912. REACM Shared Timer Bank Registers (REACM_STBK)

Table 884. REACM_STBK field descriptions

Field	Description
0–15	Reserved, should be cleared.
16–31 SHARED_TIMER [15:0]	Timer Value The SHARED_TIMER[15:0] value is one element of the Timer Register Bank. Up to three timer values can be stored within the Timer Bank. When using the shared timer for sequence advance, the counted time (considering prescaler) must be greater than 64 clock cycles.

36.3.11 REACM Hold-off Timer Bank Registers (REACM_HOTBK)

The REACM Hold-off Timer Bank Registers (REACM_HOTBK) is a set of registers that defines the values used by the reaction channels to measure hold-off time on certain modulation schemes. The timer values are programmed by software and addressed by the reaction channel based on the data read from a Modulation Word.

Address: REACM_BASE (0xC3FC_7000) + (from 0x0380 to 0x0388) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R					HOLD_OFF[11:0]											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 913. REACM Hold-off Timer Bank Registers (REACM_HOTBK)

Table 885. REACM_HOTBK field descriptions

Field	Description
0–19	Reserved, should be cleared.
20–31 HOLD_OFF[11:0]	Hold-off Timer Value The HOLD_OFFT[11:0] value is one element of the Hold-off Timer Register Bank. Up to three values can be stored within the Hold-off Timer Bank. When using the hold-off timer for sequence advance, the counted time (considering prescaler) must be greater than 64 clock cycles.

36.3.12 REACM Threshold Bank Register (REACM_THBK)

The REACM Threshold Bank Register (REACM_THBK) holds the value to be used for comparison against results received from the ADC. Based on that comparison the reaction channel decides the Channel output value to be either HOD or LOD.

Address: REACM_BASE (0xC3FC_7000) + (from 0x0400 to 0x045C) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	THRESHOLD_VALUE[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 914. REACM Threshold Bank Register (REACM_THBK)

Table 886. REACM_THBK field descriptions

Field	Description
0–15	Reserved, should be cleared.
16–31 THRESHOLD_VALUE [15:0]	Threshold Value The THRESHOLD_VALUE[15:0] unsigned value is one element of the Threshold Register register used for a threshold modulation.

36.3.13 REACM ADC result maximum limit check register (REACM_ADCMAX)

The REACM ADC result maximum limit check register (REACM_ADCMAX) holds the maximum expected value of the ADC result. If a value greater than ADC_MAX_LIMIT is received the MAXL bit in the corresponding Channel Status Register Error is asserted.

Address: REACM_BASE (0xC3FC_7000) + (from 0x0600) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R																
W																
Reset																

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	ADC_MAX_LIMIT[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 915. REACM ADC result maximum limit check register (REACM_ADCMAX)

Table 887. REACM_ADCMAX field descriptions

Field	Description
0–15	Reserved, should be cleared.
16–31 ADC_MAX_LIMIT[15:0]	Maximum Limit allowed for the ADC result The ADC_MAX_LIMIT[15:0] value indicates the maximum expected value for the ADC result. The MAXL bit in the corresponding channel Status register is set in case a greater or equal value is received from the ADC. If set to zero no limit checking is performed, thus MAXL bit will not be set anyway. ADC results are always considered unsigned unless specific note states the contrary. Since ADC received results are 14-bit values a two bit sign extension must be performed before any comparison is executed.

36.3.14 REACM Modulation Range Pulse Width Register (REACM_RANGEPWD)

The REACM Modulation Range Pulse Width Register (REACM_RANGEPWD) provides the value used to check if the PWM pulse width generated during the modulation process is larger than a maximum pulse width specified by (MIN_PWD + RANGE_PWD). The checking is performed by the channel logic during appropriate time intervals, see

Section 36.6, Monitored modulation. This function is implemented by sharing the use of reaction channel Hold-off counter, thus if the Hold-off timer is used by the channel sequence mode (SM in the REACM Modulation Control Word Bank Registers (REACM_MWBK)), this checking function is not active.

Address: REACM_BASE (0xC3FC_7000) + (from 0x0680) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R																
W																
Reset																

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R					RANGE_PWD[11:0]											
W																
Reset					0	0	0	0	0	0	0	0	0	0	0	0

Figure 916. REACM Modulation Range Pulse Width Register (REACM_RANGEPWD)

Table 888. REACM_RANGEPWD field descriptions

Field	Description
0–19	Reserved, should be cleared.
20–31 RANGE_PWD[11:0]	Range Pulse Width The RANGE_PWD[11:0] field defines the maximum pulse width allowed by the channel PWM generation. The checking is performed by the channel internal Hold-off Timer at appropriate times in the PWM modulation process. The maximum pulse is defined as (RANGE_PWD + MIN_PWD) (see *Section 36.3.15, REACM Modulation Minimum Pulse Width Register (REACM_MINPWD)* for MIN_PWD description). If RANGE_PWD = 0x00 then no maximum pulse width checking is performed. The RANGE_PWD value should be calculated considering the prescaler settings used for the HOLD_OFF counter. For a programmed (RANGE_PWD + MIN_PWD) value, a pulse narrower than or equal to (RANGE_PWD + MIN_PWD + 1) does not set the OCDF flag.

36.3.15 REACM Modulation Minimum Pulse Width Register (REACM_MINPWD)

The REACM Modulation Minimum Pulse Width Register (REACM_MINPWD) provides the value used to check if the pulse width generated during the PWM modulation process on the channel outputs is shorter than a minimum pulse width specified by this register. The checking is performed by using the channel internal Hold-off timer during appropriate times on the channel operation when this counter is not being used for the hold-off modulation cycle.

Address: REACM_BASE (0xC3FC_7000) + (from 0x06C0) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R																
W																
Reset																

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R					MIN_PWD[11:0]											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 917. REACM Modulation Minimum Pulse Width Register (REACM_MINPWD)

Table 889. REACM_MINPWD field descriptions

Field	Description
0–19	Reserved, should be cleared.
20–31 MIN_PWD[11:0]	Minimum Pulse Width The MIN_PWD[11:0] field defines the minimum pulse width allowed by the channel PWM generation. The checking is performed by the channel hold-off timer. If MIN_PWD[11:0] = 0x00 then no checking is done even for maximum pulse width or minimum pulse width. The MIN_PWD value should be calculated considering the prescaler settings used for the HOLD_OFF counter.For a programmed MIN_PWD value, a pulse wider than (MIN_PWD + 1) does not set the SCDF flag.

36.3.16 REACM Modulation Control Word Bank Registers (REACM_MWBK)

The REACM Modulation Control Word Bank Registers (REACM_MWBK) are a set of registers that controls the Reaction Channel Modulation scheme. These registers are programmed by software and read by the reaction channels. All the information required by the reaction channels to perform a modulation is stored in these words. All channels have access to the same words thus sharing of Modulation Words among channels is possible in this architecture.

Address: REACM_BASE (0xC3FC_7000) + (from 0x0700 to 0x072C) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	LOOP	IOSS	0	MM[1:0]		0	SM[1:0]		0	HOD[2:0]			0	LOD[2:0]		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	THRESPT[5:0]						STPT[3:0]				0	HDOFFTPT[3:0]			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 918. REACM Modulation Control Word Bank Registers (REACM_MWBK)

Table 890. REACM_MWBK field descriptions

Field	Description
0 LOOP	Loop Control Bit The LOOP Control bit indicates that the next Modulation Control Word accessed by the reaction channel when a Modulation Word address increment event occurs should be the one indicated by the Channel MODULATION_ADDR[5:0] register with no offset. 1 Loop back to initial channel modulation word 0 Increment to the next modulation word
1 IOSS	Initial Output State Selection The IOSS bit indicates the state of the channel output pin when the Modulation Word is just accessed by the reaction channel. This access occurs when the channel is activated by a Timer Input Channel being asserted, by software using SWMC bit in the REACM_CHCR or when the modulation phase advances, e.g. due to a timer event. 1 HOD[2:0] is initially used for chn_c, chn_b and chn_a respectively 0 LOD[2:0] is initially used for chn_c, chn_b and chn_a respectively IOSS must not be 0b0 when MM is 0b01(threshold-holdoff).
2	Reserved, should be cleared.
3–4 MM[1:0]	Modulation Mode The MM[1:0] Modulation Mode field indicates the type of the modulation that is executed by the channel. *Table 891* defines the modulation modes. IOSS must not be 0b0 when MM is 0b01(threshold-holdoff).
5	Reserved, should be cleared.
6–7 SM[1:0]	Sequencer Mode The SM[1:0] field defines the event that makes the channel address the next modulation word. This event can be a timer event or a threshold level event. Note that the channel does not necessarily increment the Modulation Word address thus the same Modulation Word is executed again in the new modulation sequence. *Table 892* defines the Sequencer modes. For time-out event selections, it is required the related time be greater than 64 clock cycles

Table 890. REACM_MWBK field descriptions (continued)

Field	Description
8	Reserved, should be cleared.
9–11 HOD[2:0]	High Output Drive The HOD[2:0] field defines the values driven on the chn_c, chn_b and chn_a channel output pins, respectively, when the channel is at ON state. Set HOD[2:0]=LOD[2:0] when using threshold-threshold modulation together with sequence advance, in order to avoid fast glitches during the sequence advance
12	Reserved, should be cleared.
13–15 LOD[2:0]	Low Output Drive The LOD[2:0] field defines the values driven on the chn_c, chn_b and chn_a channel output pins, respectively, when the channel is at OFF state. Set HOD[2:0]=LOD[2:0] when using threshold-threshold modulation together with sequence advance, in order to avoid fast glitches during the sequence advance
16	Reserved, should be cleared.
17–22 THRESPT[5:0]	Threshold Pointer The THRESPT[5:0] Threshold Pointer is the address of the Threshold Bank that holds values to be used for ADC result comparisons in the modulation process. This pointer has the resolution for a 16-bit data stored in the register described in *Section 36.3.12, REACM Threshold Bank Register (REACM_THBK)*.
23–26 STPT[3:0]	Shared Timer Pointer The STPT[3:0] Shared Timer Pointer field is the pointer for the Shared Timer Bank. The Shared Timer is used as timer sequencer defining the advance of Modulation Words.
27	Reserved, should be cleared.
28–31 HDOFFPT[3:0]	Hold-off Pointer The HDOFFPT[3:0] field is the address of the Hold-off timer value in the Hold-off Timer Bank that is used in the modulation cycle. Note that there are modulation sequences that do not require hold-off measurement such as threshold/threshold modulations.

Table 891. MM[2:0] configuration: Modulation modes

MM[2:0]	Modulation type	Modulation description
00	Threshold/Threshold	Modulation occurs between two threshold levels defined by THRESPT upper level threshold pointer and THRESPT +1 which corresponds to a lower level threshold pointer. Output ON state is defined by HOD and Output OFF state is defined by LOD.
01	Threshold/Hold-off	Modulation occurs between an upper level threshold defined by THRESPT and a Hold-off time is initiated after that level is achieved which set channel outputs to an off state.
10	Reserved	—
11	Reserved	—

Table 892. SM[1:0] configuration: Sequencer modes

SM[1:0]	Event	Event description
00	No advance	Current modulation word is used, no advance is performed on any event. No timer is activated by this modulation word.
01	Timer time-out	Advance to the next modulation word when a Timer time-out event is detected. The Timer is activated when the modulation word is accessed by the channel.
10	Hold-off timer time-out	Advance to the next modulation word when a Hold-off Timer time-out event is detected. The Hold-off timer is activated when the modulation word is accessed by the channel.
11	Threshold level achieved	Advance to the next modulation word when a threshold level is achieved. The threshold level is defined by the THRESPT pointer. See section *Section 36.10.1, Advancing modulation phase on a threshold level*.

36.4 Functional description

The following sections describe the Reaction Module functionality.

The Reaction Module is designed to allow a closed feedback loop control over the driver load currents. The load can be an injector for a direct injection system or an electromagnetic actuator for a robotized transmission. In both cases it is expected that a solenoid will actually be the load. The module architecture is based on shared resources submodules that can be used by all six reaction channels. Please refer to *Figure 903* for details about the module architecture. The Reaction Module comprises the following internal submodules:

- Reaction Channel
- Modulation Control Word Bank
- Shared Timer Bank
- Hold-off Bank
- Threshold Bank

The following sections describe each one of these submodules and their combined operation.

36.4.1 Reaction channel

The Reaction Channel is the core of the Reaction Module. Each channel controls three output pins and is controlled by a Control finite state machine (FSM) that receives parameters to generate a modulated waveform as well as the timer control window, usually provided by eTPU. The ADC interface indicates to the channel that a new ADC result is available. If *activated* by the eTPU channel, the reaction channel reads the Modulation Word which provides addresses for the Threshold Value bank, Shared Timer Bank and Hold-off Timer Bank. The date provided by the Threshold Bank is compared with the incoming ADC result. Based on that comparison the reaction channel state machine selects the values for the output pin registers. The modulation only occurs if *activated* by the timer control signal, which can be generated by eTPU, or can be controlled by software by writing to the SWMC, software modulation control bit, in the Channel Configuration register. The MM and SM fields in the Modulation Word provide the Modulation Mode and Sequencer Mode control, respectively. *Figure 919* describes the internal architecture of the reaction channel and its interconnection with other submodules.

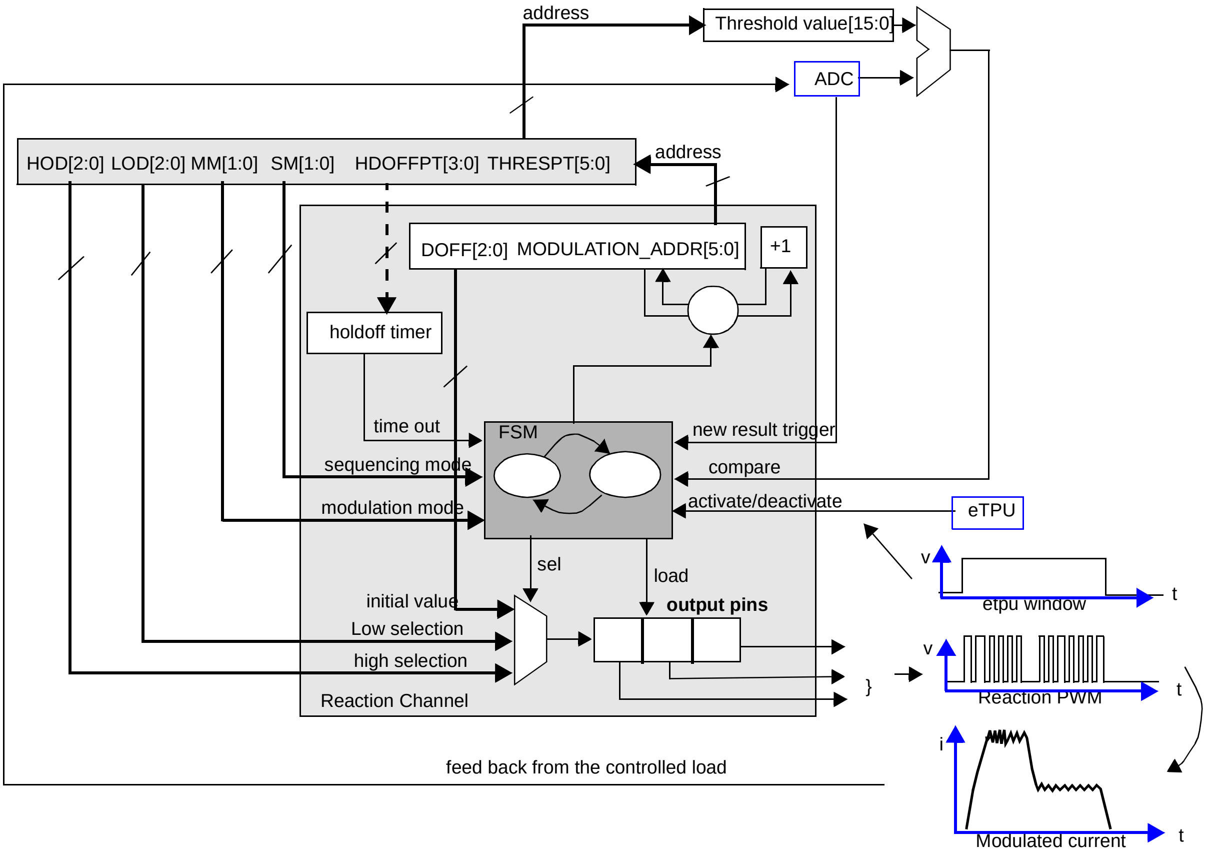


Figure 919. Reaction channel architecture simplified diagram

The addresses stored in the Modulation Word are pointers to timers and threshold banks. HOD and LOD provides the 3-bit field which controls the states, HIGH and LOW, respectively for the reaction channel output pins. The reaction channel FSM selects the values for the output pins as well as the load signal used to update those values. The modulation executed by the reaction channel is based on the feedback loop provided by the ADC results. These results are routed to one side of a comparator while the other comparator input receives the selected Threshold value from the threshold bank. The comparison result indicates for the reaction channel FSM if the output pin value should be defined either by HOD or by LOD. As a result a PWM is generated on the output pins. Note that since there are three independent outputs the PWM signal can be generated on any output or even on several outputs at the same time. Usually a PWM signal is generated in one of the outputs only, while the others outputs are used to control power supply switching, for example.

Three wave forms are shown at the right side of *Figure 919*. On the top there is control signal from eTPU which defines the boundaries of the modulated waveform. In the middle there is the PWM waveform which is used to drive the power on and off at the load, thus generating the third waveform which represents the current passing through the load.

36.4.2 Modulation control words bank

The Modulation Control Word provides information about the modulation type to be performed and the addresses for several address banks. *Figure 918* describes the REACM Modulation Control Word Bank Registers (REACM_MWBK) and *Figure 920* describes the interfaces of this submodule with some other submodules in the Reaction Module. The informations stored in the Modulation Word are:

- Modulation control parameters for the reaction channel
- Threshold Value Register Bank address
- Hold-off bank address
- Shared Timer bank address
- DMA support

All channels share the information stored in the Modulation Control Word Bank, which provides a size-effective implementation avoiding the duplication of information and allowing flexible implementation. The sharing of modulation control words allows several channels to execute the same modulation sequence.

The Modulation Control is designed to be used by all reaction channels as a centralized resource. However, only one channel is able to access the Modulation bank at a given time. Therefore, there is a priority in the selection of the channel that will have the access granted, but note that this condition does not occur too often since ADC results are provided for one channel at a time.

An arbiter in the Modulation Control Word bank selects one of the channels which are requesting access to a modulation word. The priority criteria is fixed and based on the channel number, considering Channel 0 the highest priority channel. The channel selected by the arbiter receives an acknowledge signal which indicates that channel was selected and therefore can access the modulation word.

Note: In order to avoid an initial delay when processing a timer window start event, the channel performs a speculative read operation of the first modulation word when it is enabled (CHEN is configured). Therefore, when the modulation cycle is triggered by the timer window start event, all needed information for the modulation is already stored inside the channel.

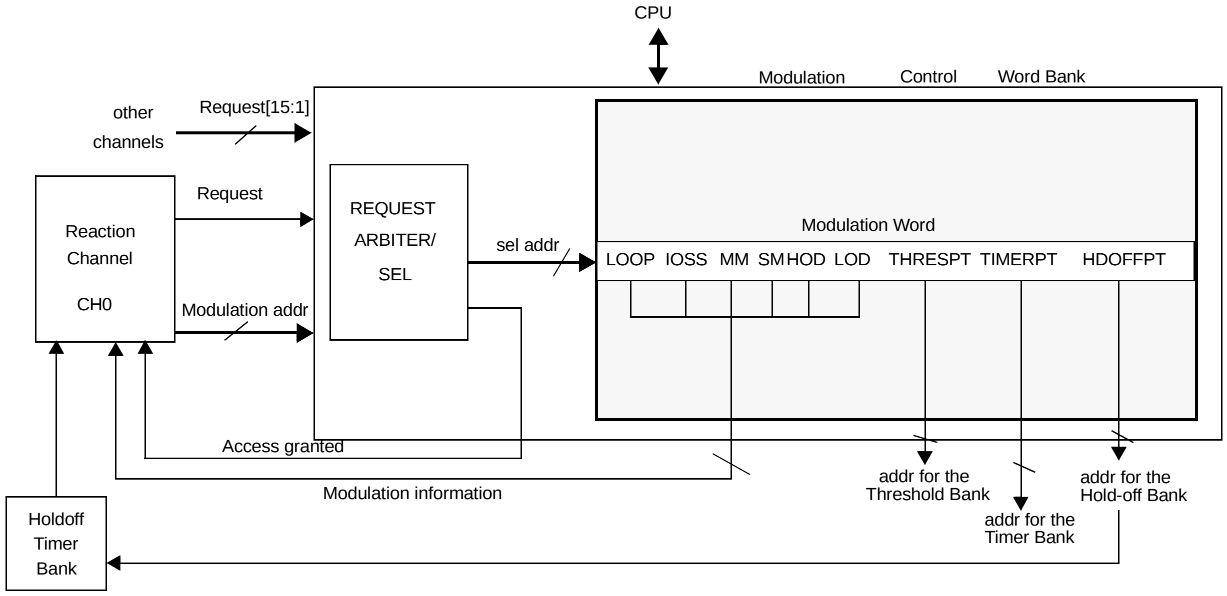


Figure 920. Modulation control word bank interfaces

36.4.3 Shared timer bank

The Shared Timer bank is an innovative concept of dynamic timer allocation. Since the number of timers can be smaller than the number of channels in the Reaction Module, there is a possibility that all timers are allocated at a certain time. This architecture is based on the low probability of such scenario since the timer allocation is a sporadic event. The timers in the Shared Timer Bank are usually in IDLE state until they are allocated by a channel. Any timer can be allocated as soon as it is IDLE. The shared timers work in conjunction with a timer bank which stores values to be used by the timing measurement.

The Shared Timer Bank block is composed of two submodules:

- Three 16-bit counters
- A bank with maximum of 16 selectable 16-bit time values

If there is an attempt to allocate more than three timers then an error flag TAER (see *Figure 910*) is set and no timer is allocated by the requesting channel. As a general guideline the system should be dimensioned in such a way that the timer allocation is always possible.

Note: *In case of an allocation error the channel forces DOFF to its output pin preventing any damage to occur to the actuator being controlled. This state will not change until bit TAER is cleared.*

During the timer allocation the channel also provides the TIMERPT pointer which selects a timing value. The valued pointed by TIMERPT is loaded into one of the three counters which counts down until reaching zero. At this time a timeout indication is sent to the requesting channel and the timer is deallocate, moving back to IDLE state.

In case of several timer activation requests being issued at the same time, the logic in the Timer Bank will prioritize giving higher priority for the channel with lower number, thus channel zero has higher priority than the others. No flags are set in the case several requests are issued simultaneously unless there are more requests than the number of available timers. In this case the TAER error flag is set in the requesting channel status register. *Figure 921* presents a block diagram of the Timer Bank.

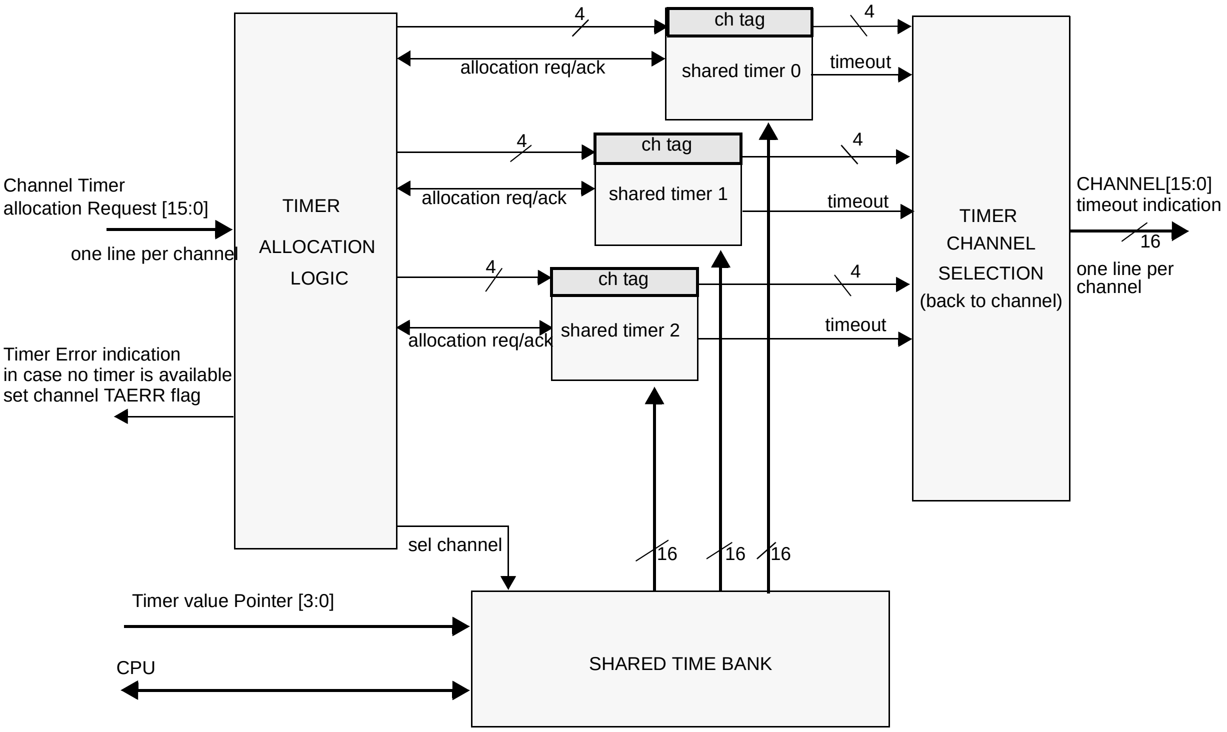


Figure 921. Shared timer bank block diagram

36.4.4 Hold-off timer bank

The Hold-off Timer Bank stores time values that are used by the hold-off periods during the modulation process. This bank is shared among all channels and is addressed based on the HDOFFPT pointer in the Modulation Control Word. *Figure 922* shows a block diagram of this bank and its interconnections.

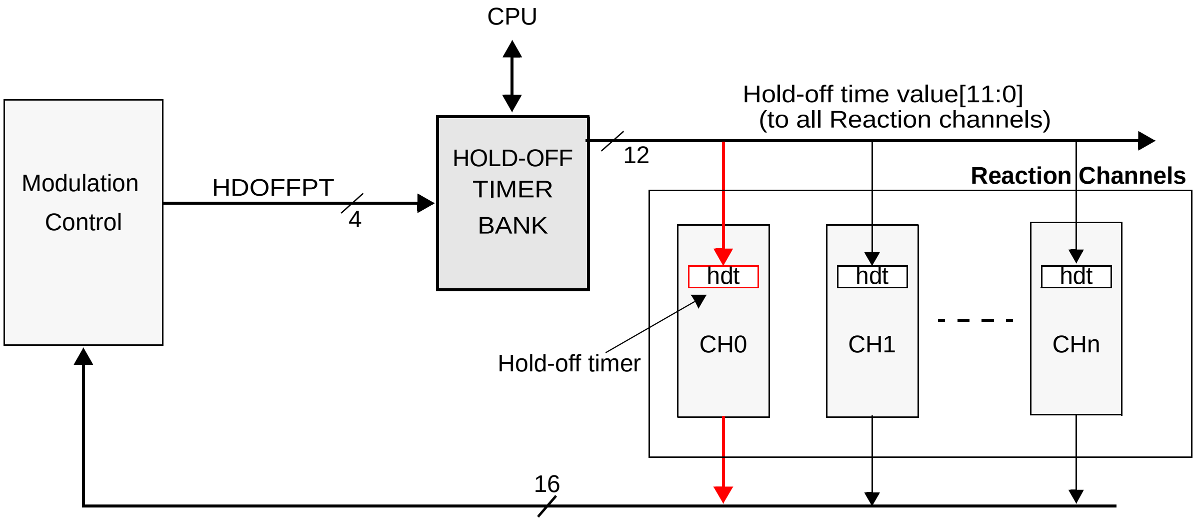


Figure 922. Hold-off timer bank block diagram

The Modulation Control Word generates the address for the Hold-off Timer bank which then generates the hold-off value for the channels. The channel that addressed the Modulation Word captures the 12-bit value storing it in the channel Hold-off timer. In *Figure 922* channel CH0 is requesting access to the Modulation Word thus receiving the hold-off value from the Hold-off Timer Bank. Each reaction channel has its own internal hold-off timer.

36.4.5 Threshold bank and comparator

The Threshold bank and comparator contains the threshold levels to be compared against the ADC results. This submodule also performs the comparison between the ADC result and the selected threshold value.

The Threshold bank can be programmed with up to 64 threshold values depending on the configuration defined during the module integration.

An ADC result received from the on-chip ADC is connected to the ADC router and then to the comparator. See *Figure 923*. This result stays at the comparator input until the Reaction Channel and Modulation Control Word selects the address of the Threshold value. The THRESPT field in the Modulation Control Word submodule is used as the address for the Threshold Bank. Note that if Threshold/Threshold modulation mode is used two comparisons need to be executed with two different Threshold values. After the first comparison is made the THRESPT pointer is incremented and a new comparison is done:

- First comparison: COMP = ADC_DATA ≥ THRESHOLD_VALUE[THRESPT]
- Second comparison: COMP = ADC_DATA < THRESHOLD_VALUE[THRESPT + 1]

The COMP (comparison result) is routed to the channel selected by the received TAG. Once having received the comparison result the channel takes the appropriate actions in order to execute the modulation mode as defined by the modulation control word.

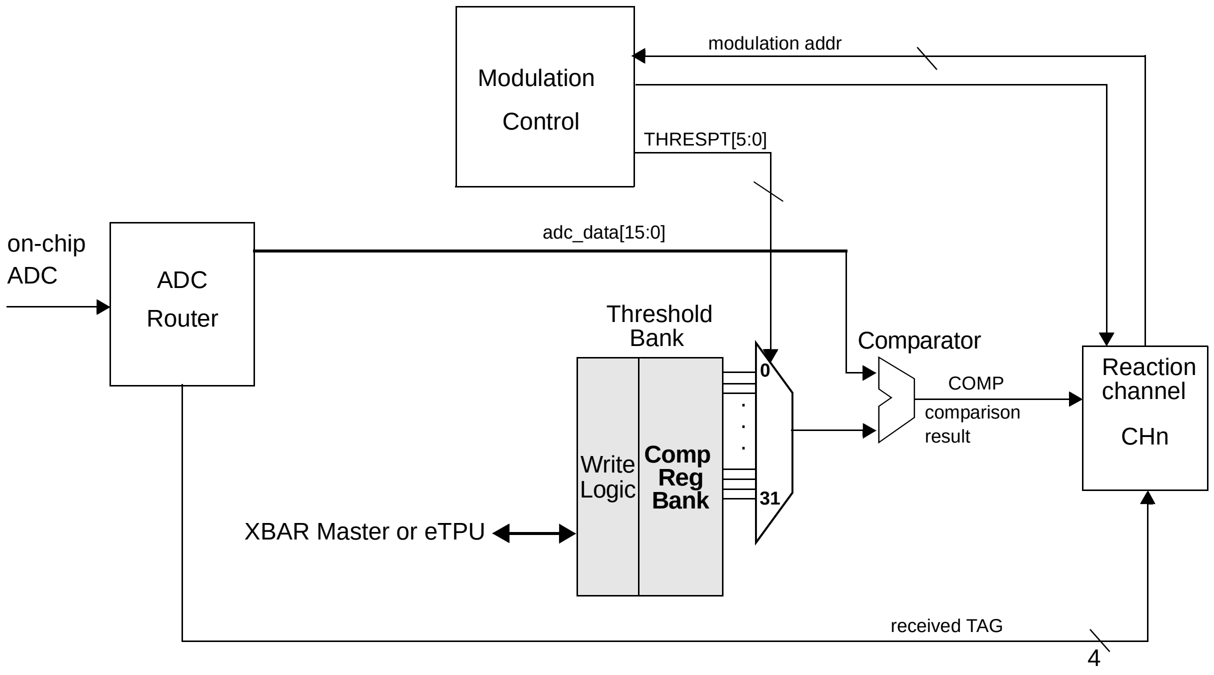


Figure 923. Threshold bank and comparator block diagram

36.4.6 ADC interface

The ADC Interface connects to the eQADC module through a Parallel Side Port also called PSI. Please refer to *Chapter 23: Enhanced Queued Analog-to-Digital Converter (EQADC)* for more information about this interface. The eQADC sends conversion results to the reaction module which are received by the ADC interface submodule. The ADC interface is capable of distinguishing between ADC results, Time stamps and *Prefill* information from eQADC. Only ADC results with no *Prefill* are considered by the reaction module as valid ADC samples. The TAG value received with the conversion result indicates which channel the result is addressed to. The ADC interface is also responsible to indicate for the selected reaction channel that a new ADC result is available. It is also possible to access the ADC interface data through the REACM_SINR register.

Note: The ADC interface data (ADC_TAG and ADC_RESULT) should not be updated until all reaction channels (with ADCR = ADC_TAG) process the received data. The OVR flag is set in the case of an overrun condition occurs, indicating that at least one ADC data was lost.

Input buffer overrun

The OVR flag indicates than an input buffer overrun occurred. This buffer is normally written by the ADC interface (PSI) but can be written also by the CPU using the REACM_SINR register.

The ADC Interface data (ADC_TAG and ADC_RESULT) should not be updated until all Reaction Channels (with ADCR = ADC_TAG) process the received data. The OVR flag is set in the case of an overrun condition occurs, indicating that at least one ADC data was lost (the input data that caused the overrun is lost).

The following situations can generate an input overrun:

- Two consecutive sample data are received from ADC or from CPU.
- ADC and CPU sending sample data at the same time (asynchronous events).
- Input buffer is holding sample data that are being processed by some channel and a new sample is received from ADC or CPU (normal mode of operation).

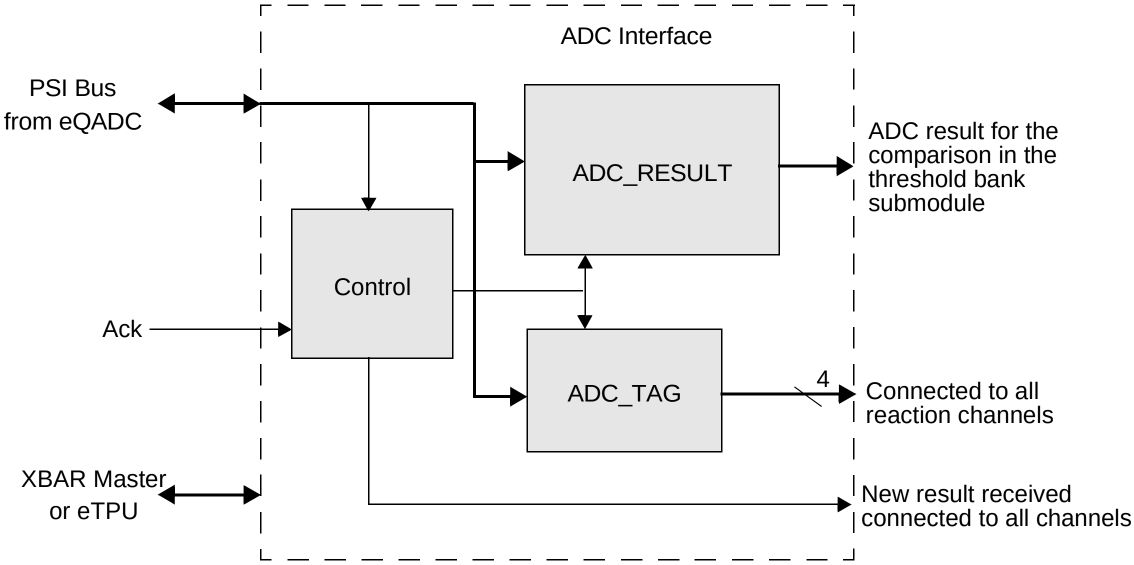


Figure 924. ADC interface block diagram

The maximum throughput supported by the reaction module depends upon the TAG of the incoming ADC data. If only one reaction channel is addressed by the ADC TAG then the maximum supported ADC data rate is one sample each five clock cycles. If two reaction channels are addressed by the same ADC TAG, thus having the same CHRRn ADCR field, and are active at the same time, the maximum supported rate is one ADC data on each 10 clock cycles. In general if (n) channels share the same TAG and are active at the same time the maximum supported ADC data rate is 5×(n) clock cycles.

These limitations are related to the sharing of internal reaction module resources such as the Modulation Word bank. The ADC conversion data should remain stable in the ADC interface until it is used by all channels which matching TAG and CHRRn ADCR fields.

Note that if all active channels have different CHRRn ADCR fields, that is, are assigned to different TAGs, the maximum supported ADC rate is five clock cycles. If multiple channels have the same CHRRn ADCR field but only one is active at a time, then the maximum supported ADC data rate is also five clock cycles.

On-the-fly ADC data acquisition

The ADC interface and the threshold bank can operate in a learn mode, meaning that the received ADC result can be stored in the threshold bank and used for comparisons. This functionality allows the user to interactively calibrate the modulation levels or capture on-the-fly levels from the external application board for debug purposes. The REACM_THRR is used for that purpose. See *Figure 906* for more details about the register architecture.

Figure 925 describes the main connections between the ADC interface and threshold bank submodules.

The routing of received ADC results to be stored in the threshold bank is independent from the routing of the same result to the reaction channels (see *Figure 911*). Thus the same ADC result can be used in a comparison and stored in the threshold bank.

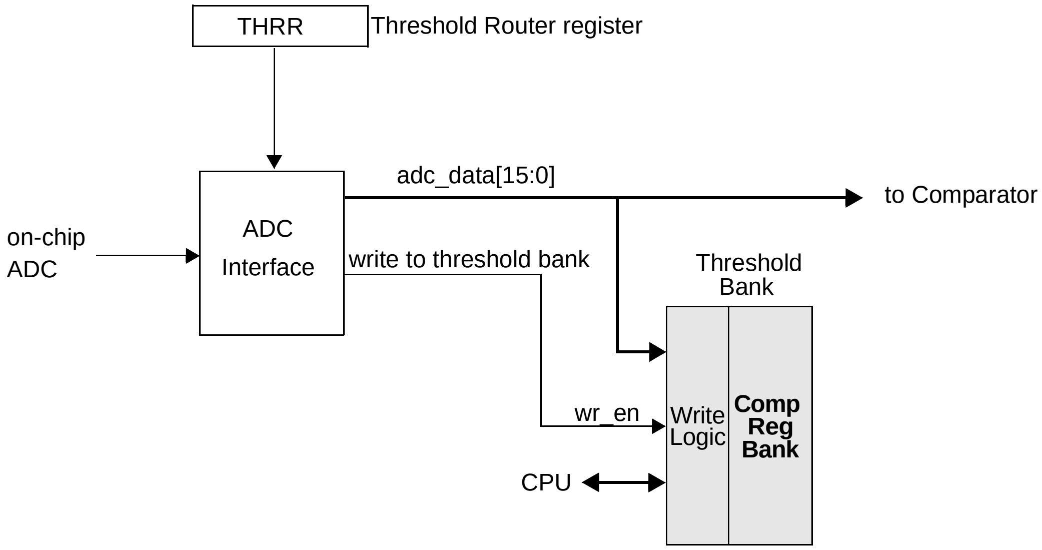


Figure 925. ADC interface and threshold bank interconnections

36.4.7 Prescalers

The prescalers provide internal system clock divided signals to be used by internal timers. The reaction module contains two prescalers: a 12-bit prescaler HPRE[11:0] and an 8-bit prescaler TPRE[7:0]. Both are defined in the REACM Timer Configuration Register (REACM_TCR) (see *Figure 905* for details). Prescaler HPRE[11:0] is dedicated to the Hold-off timers within the reaction channels. Prescaler TPRE[7:0] is used by the Shared Timer Bank counters. The HPRE[11:0] and TPRE[7:0] prescalers are enabled by HPREN and TPREN bits, respectively, in the REACM module configuration register (REACM_MCR) (see *Figure 904*). Note that prescalers operate in a similar way regarding their activation. Once the prescaler is enabled by HPREN or TPREN bits in the REACM_MCR, it starts a new count sequence meaning that it is put in reset state and will generate the first prescaler tick after it reaches the programmed value defined by the HPRE or TPRE fields.

36.4.8 Banked mode support

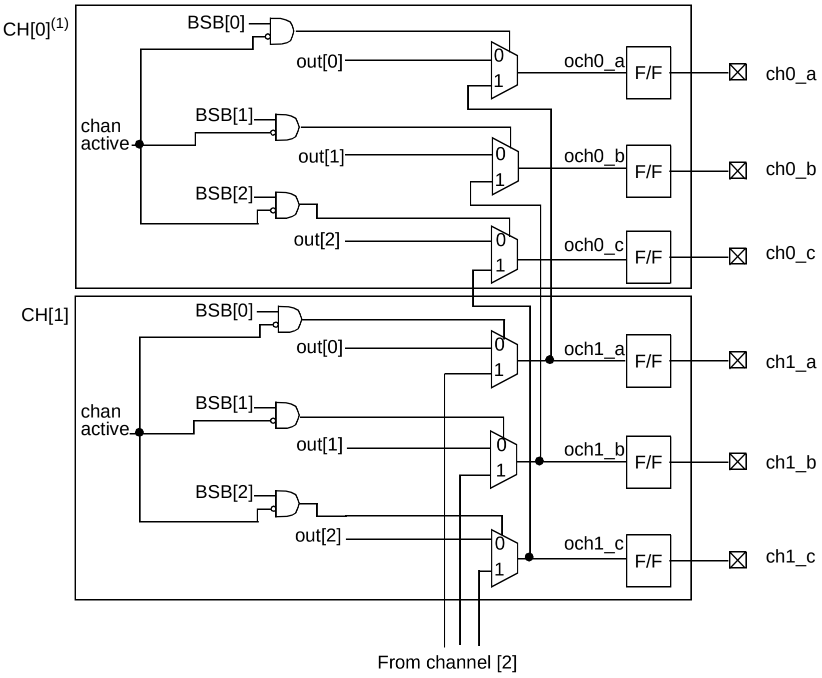
Banked Mode is a reaction module hardware configuration which allows the sharing of reaction channel output pins at the device I/O level. The banked mode architecture allows the stacking of up to four reaction channels. *Figure 926* shows the connection between two adjacent channels, CH0 and CH1. The REACM_CHCRn BSB bits are used to control the configuration of channel output logic. Thus if BSB[0] in reaction channel [0] is asserted to 1 and the channel is not in the active state, ch0_a output is switched from CH[0] OUT[0] to CH[1] OUT[0]. If a channel is active, that is, executing a modulation, it takes control over its outputs independent of the BSB bits setting. The banked mode logic is extended to reaction channels CH[2] and CH[3] thus defining a group of four channels. For simplification,

Figure 926 shows only channels CH[0] and CH[1] logic.

Note: When CHEN = 00 for CH0 (channel disabled), the BSB bits do not influence the channel output, which is driven CH0's DOFF. Therefore, a banked injector driven by CH1 will have part of its controls off, even if CH1 is enabled.

To use BSB of CH0 in this case, an option is to program channel CH0 with CHEN = 11 (channel enabled) and with SWMC = 0 (modulation OFF).

In case of using CH2 in banked mode with CH0, the intermediate channel CH1 should also be configured with CHEN different from 00.



NOTES:
1. CH[0] should be enabled by CHEN to use BSB.

Figure 926. Banked mode showing stacking of channels [0] and [1]

The Banked Mode support hardware is implemented on groups of four channels. The groups are defined as CH[3:0] and CH[5:4]. Thus CH[3] and CH[5] do not connect to the subsequent channel which are CH[4] and CH[0] respectively.

36.5 Modulation Modes

This section describes the modulation modes provided by the reaction module.

36.5.1 Threshold/Threshold mode

This mode is programmed by using REACM_MCWBx[MM] = 00. In this modulation mode the reaction channel tries to maintain the ADC results between two threshold limits. The modulation actions are as follows:

- ADC result ≥ [THRESPT]: The reaction channel turns the outputs off by loading LOD[2:0] to the channel outputs.
- ADC result < [THRESPT + 1]: The reaction channel turns the outputs on by loading HOD[2:0] to the channel outputs.
- [THRESPT + 1] ≤ ADC result < [THRESPT]: The reaction channel keeps the outputs unchanged.

Figure 927 indicates the threshold/threshold modulation mode.

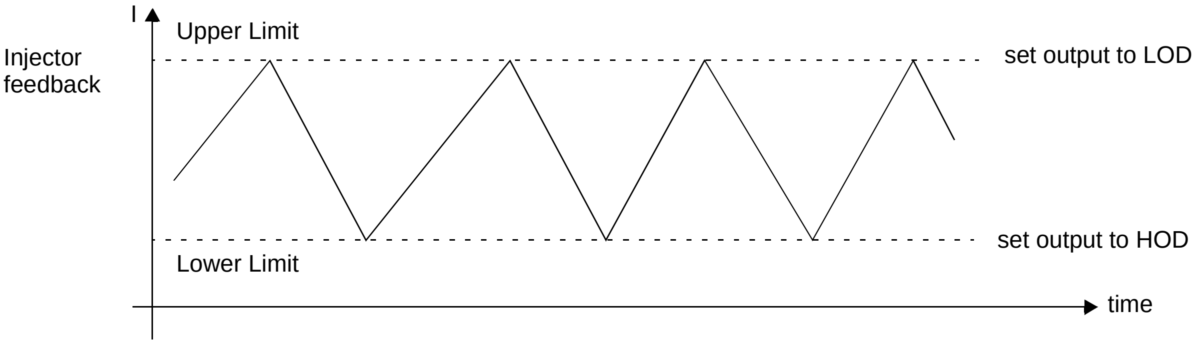


Figure 927. Threshold/threshold modulation mode

There can be overshoots or undershoots in the real application related to the threshold limits. This occurs due to following:

1. Feedback values are periodically sampled thus can present gaps on the measured values, for example when the sample occurs the value already passed the range defined by the limits.
2. There is a delay between the sampling of feedback value and the reaction of the reaction channel. This is mainly caused by the ADC conversion time and its maximum sampling frequency.

36.5.2 Threshold/Hold-off mode

This modulation mode is programmed by using REACM_MCWBx[MM] = 01. In this mode the output pins are driven with HOD[2:0] until the ADC results matches or is greater than the programmed Threshold. The output controls are then driven with LOD[2:0] for a fixed amount of time defined by the hold-off timer. Note that the upper threshold limit and the hold-off timer pointer are indicated by the modulation word in the REACM_MCR register.

Note: ADC samples are not considered reliable in hold-off time, therefore no comparison for MAXL or for output definition are performed. If these comparisons are necessary, the Threshold / Threshold modulation mode can be used.

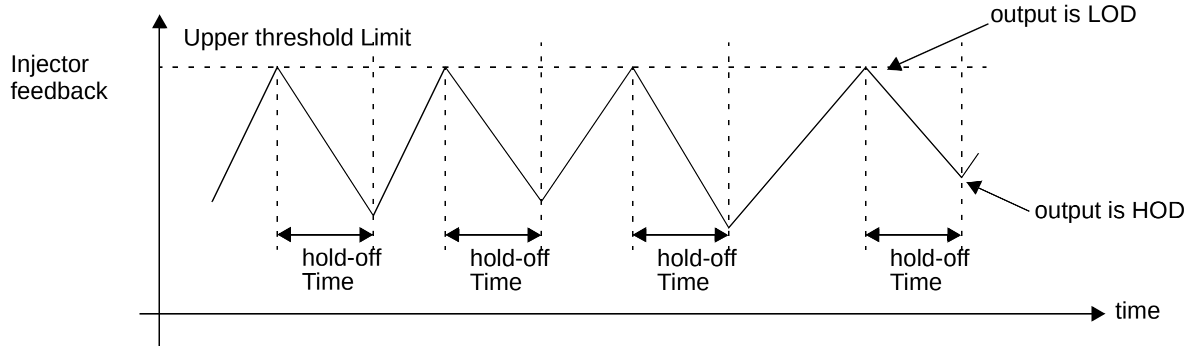


Figure 928. Threshold/hold-off modulation mode

36.5.3 Limitations on the modulation process

This section describes the Channel limitations on the modulation process such as the width and distance between consecutive modulation pulses.

Minimum distance between consecutive timer control pulses

The control signal generated by the external timer that controls the modulation process in a worst case scenario has a minimum distance of 64 system clock periods. This is required for the reaction channel state machine to proper re-start the modulation process and address the correct modulation word. The worst case scenario considers a Reaction Module with 16 channels and all channels being activated at the same time. If this minimum distance is violated the modulation on the second pulse will not be executed, meaning that the channel output will be defined by the DOFF field.

Note: No error flag is set if this violation occurs.

Figure 929 describes the channel behavior if the minimum time between two consecutive timer control pulses is violated.

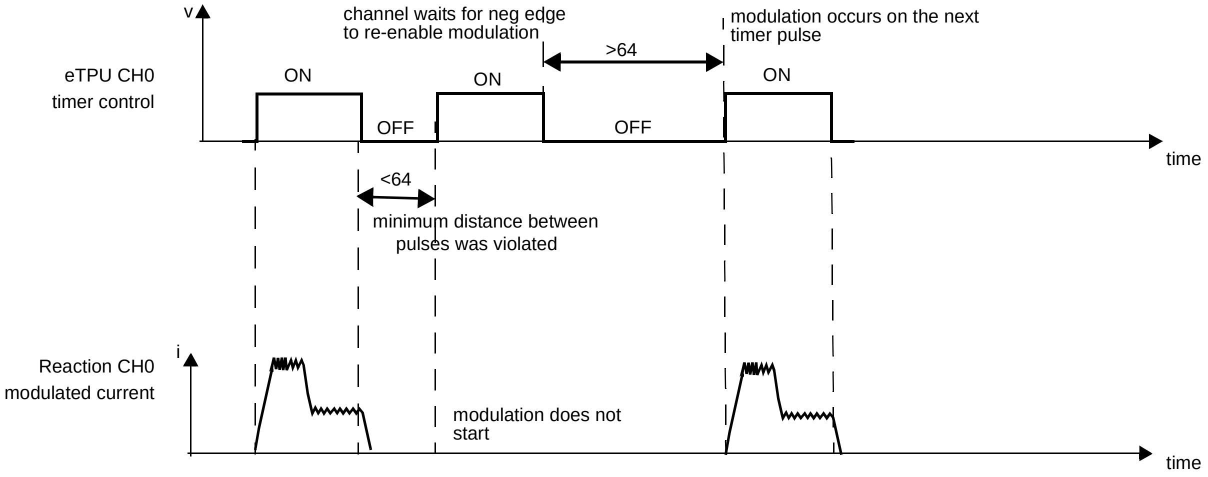


Figure 929. Limitation on the OFF modulation timing

Minimum timer control pulse width

There is no limitation on the Timer Control pulse minimum width. The channel just finishes the modulation process when the pulse ends even if the Modulation Words did not complete the expected sequence. Thus if a modulation phase is designed to have four Modulation Words starting from word 0, and the Timer Pulse ends at the second word (word 1) then the modulation process finishes at this point in time. The channel output is set to DOFF one clock cycle after the Timer Pulse ends and the flag SQER is set because the SM bit field for the second modulation word is not 00. This flag occurs to indicate that the modulation was ended before the last phase of the sequence that uses the modulation word 3 and has SM = 00.

Figure 930 describes the channel behavior when an early end of the Timer Control pulse occurs. Note that the channel output is driven to DOFF which causes the modulation to end. Note that an early end of pulse only affects the current modulation cycle. Meaning that on the next modulation cycle the modulation word 0 is executed first and all subsequent words are executed in the appropriate sequence.

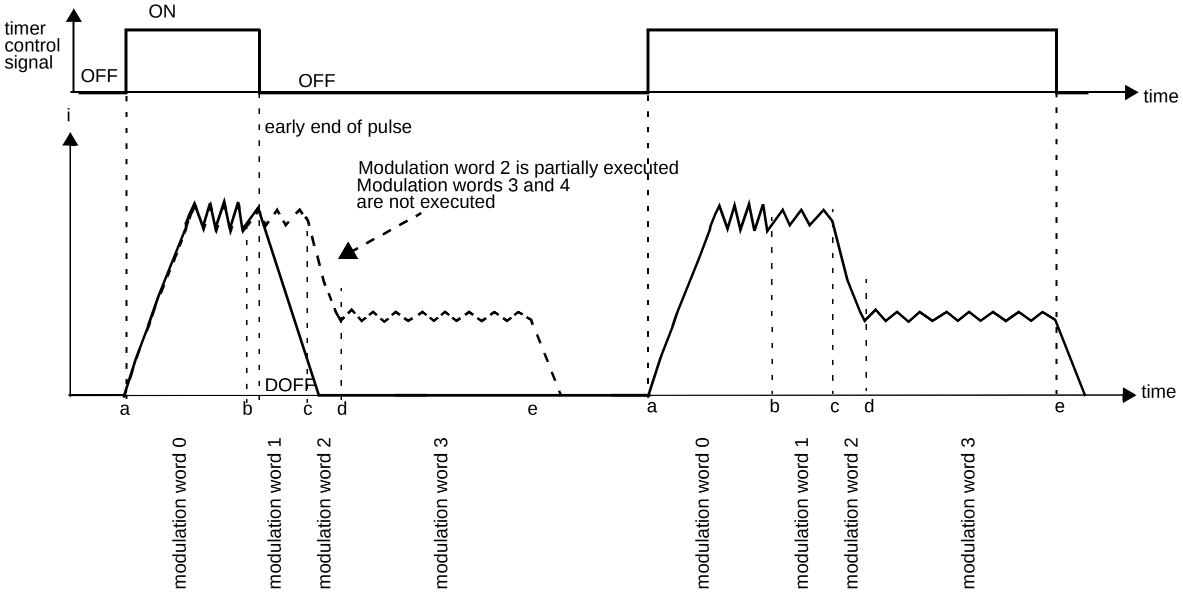


Figure 930. Early end of Timer Control pulse

Note: In some applications the modulation runs continuously and the input timer control signal is not used as a modulation pulse control but only as an enabling signal. Therefore, if the modulation is turned off, the REACM can issue a Modulation Word Sequence Error by setting SQER flag. In this case, this SQER flag can be ignored (masked) without prejudice.

An option, if the application permits, is to disable the REACM channel (CHEN = 00), thus avoiding unwanted SQER error.

CHOFF behavior during modulation

The CHOFF—channel output disable bit (Figure 909) is intended to disable the channel immediately before a modulation cycle had ended. This is important on error conditions detected by the software, thus setting the channel outputs to a safe state defined by DOFF.

It is important to notice that in order to reactivate the channel by setting CHOFF = 0 does not implies that the channel outputs immediately returns to the state defined by HOD and LOD on the modulation word being executed. Instead, the HOD and LOD are driven to the channel outputs only when one of the following conditions occur:

- a sequence advance event (timeout or threshold, depending on SM).
- a new sample is received (no matter if the comparison matches or not).
- a hold-off timeout.

Module initialization

To execute the modulation process the Reaction Module must be initialized with a correct sequence. One method is described as follows:

1. Make sure eTPU channels are at zero, inactive state.
2. Set up eQADC module.
3. Program the Modulation Words.
4. Program the Threshold Bank.
5. Program the Shared Timer Bank.
6. Program the Holdoff Timer Bank.
7. Program Timer Configuration register.
8. Program the Module Configuration register.
9. Program the Channel Router register.
10. Program the Channel Configuration register.
11. Start eTPU channels.

It is important to notice that the channel activation, by setting CHEN = 01, should be done after all other registers have been configured and before the input timer control signals are active. Violating this order may lead to errors when the modulation cycle is executed by the channel.

Note: If a glitch is introduced in the input timer control pulse, the channel stops to modulate and does not operate during the pulse just after the glitch. The glitch value for wrong operation ranges from 1 system clock to about 5 times the number of channels.

36.6 Monitored modulation

The modulation executed by the reaction channel can be monitored by measuring the width of the PWM pulses provided by the channel. If the pulse becomes too narrow it means that the load impedance is probably too low, thus indicating a possible short circuit. If the PWM pulses become too wide it may indicate an open circuit on the solenoid.

The limits for narrow and wide pulses are defined by RANGE_PWD and MIN_PWD registers (see *Figure 916* and *Figure 917*). These values apply to all reaction channels. The PWM pulses are measured by the Hold-off timers within the reaction channels. This is possible only during idle periods of this timer, for example from the moment a hold-off timeout occurs until the maximum threshold is reached when HOD is being used for the channel output pins. During this period the hold-off counter is not used and thus it can measure pulse widths and compare them against predefined limits defined by RANGE_PWD and MIN_PWD registers, as shown in *Figure 931*.

Note: *Consider an uncertainty of (+1) in the value MIN_PWD and (MIN_PWD + RANGE_PWD) when calculating the pulse width limits. The Hold-off prescaler contributes to this uncertainty.*

For a programmed MIN_PWD value, a pulse wider than (MIN_PWD+1) does not set the SCDF flag.

For a programmed (RANGE_PWD + MIN_PWD) value, a pulse narrower than or equal to (RANGE_PWD + MIN_PWD + 1) does not set the OCDF flag.

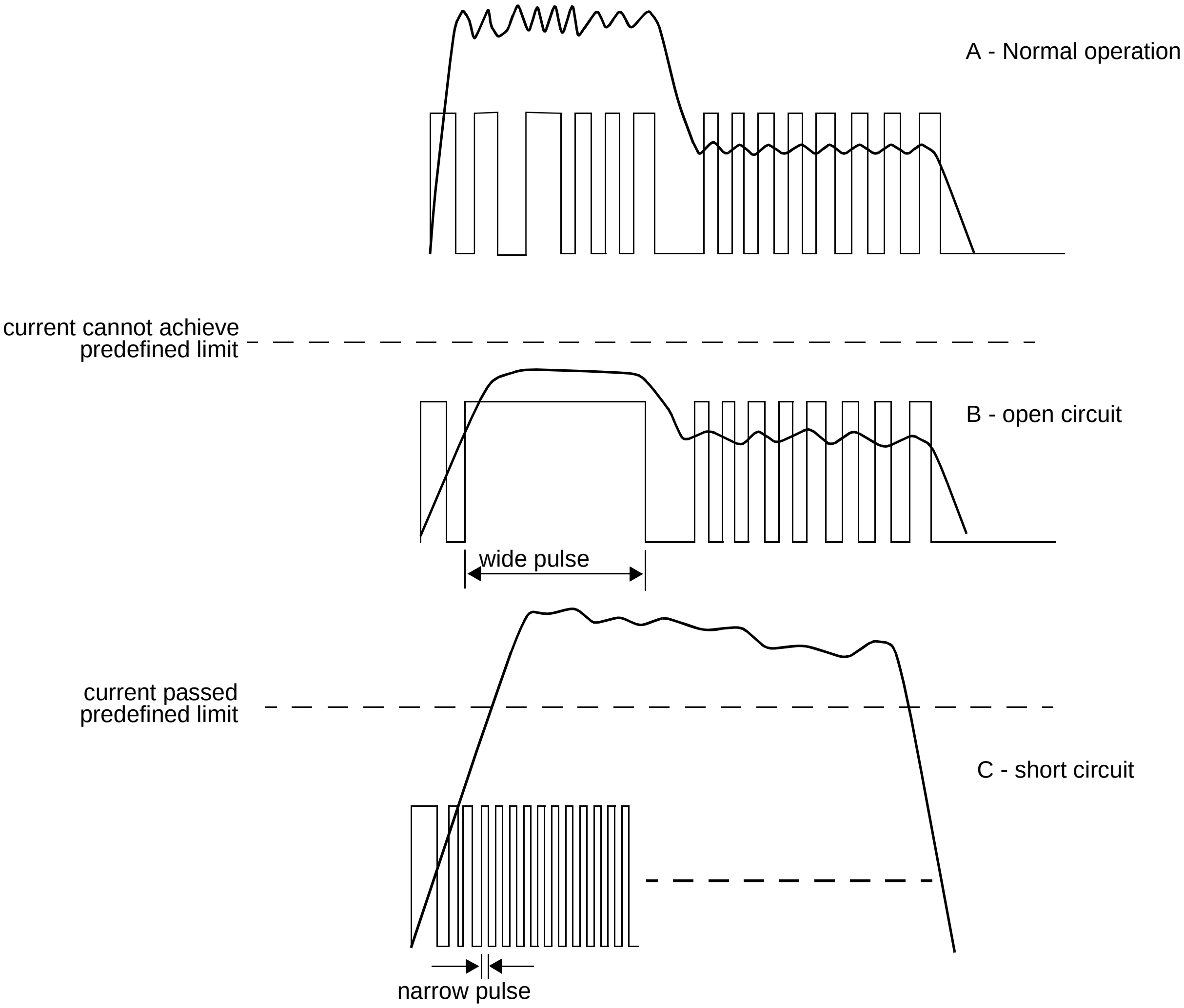


Figure 931. Fails detected by the modulation monitoring

Figure 932 describes in more detail an open circuit detection using the Hold-off timer. Note that the Hold-off timer is used when the output is at the ON state, which means HOD is used to drive the outputs, thus hold-off functionally is not required at this moment. The Hold-off timer is loaded with MIN_PWD and starts counting. After a time-out occurs RANGE_PWD value is loaded and the Hold-off counter starts counting again. An open circuit is detected if the timer times-out on the second RANGE_PWD counting and the PWM pulse is still high, meaning that the outputs did not switched to LOD. In this case the OCDF flag is set in the CHSR, Channel Status register, of the corresponding channel. See *Figure 910*.

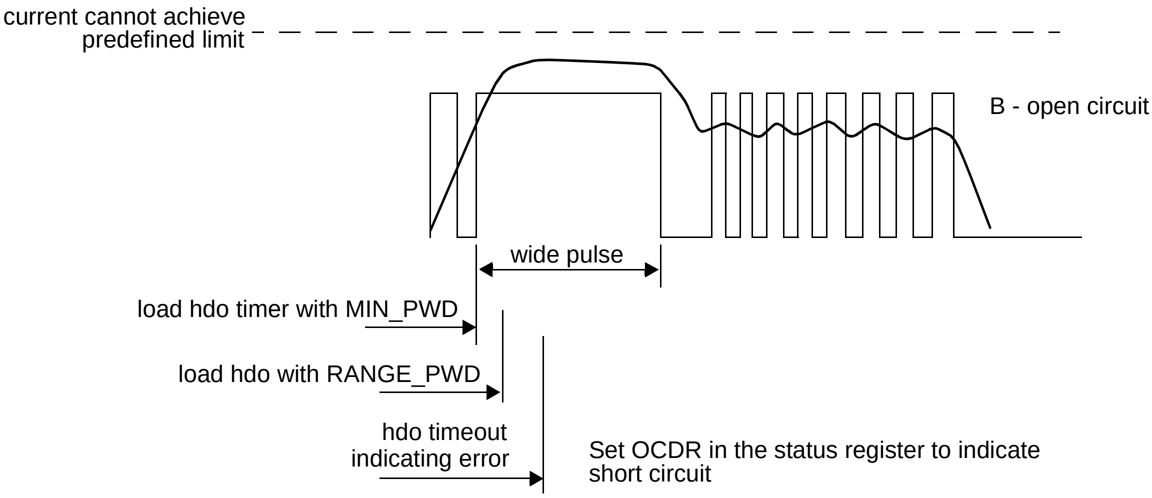


Figure 932. Open circuit detection using hold-off timer

Figure 933 describes how the Hold-off timer detects a short circuit. The Hold-off timer is loaded with MIN_PWD value and enabled. The condition that indicates that a short circuit occurred is the Hold-off timer still running and LOD is loaded to the channel output pins. That means the PWM pulse are too narrow and a short circuit have occurred. In this case the SCDF flag is set in the CHSR, Channel Status register, of the corresponding channel. *Figure 910*.

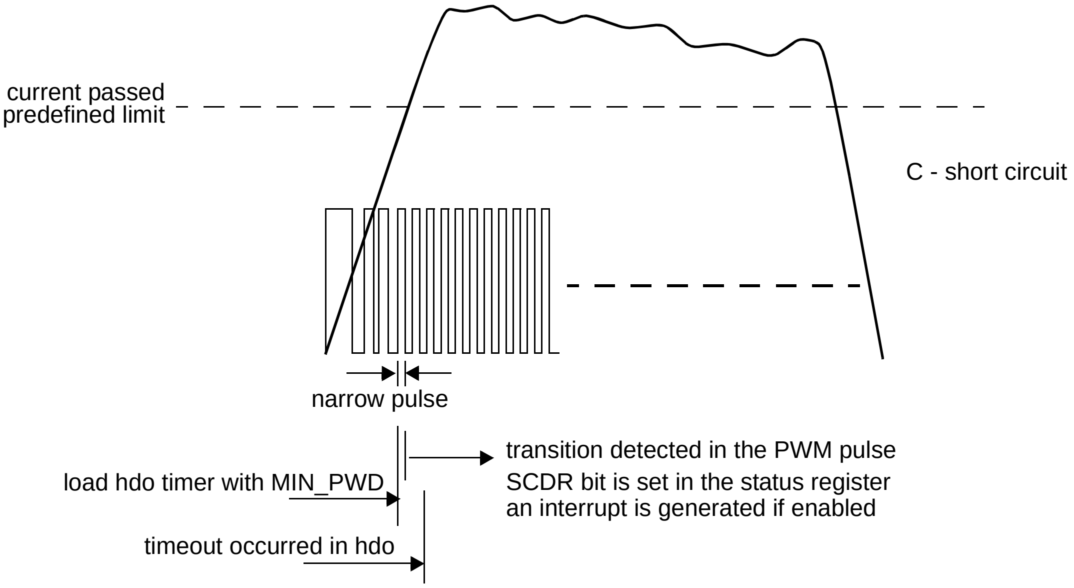


Figure 933. Short circuit detection using hold-off timer

Note: *In order to define RANGE_PWD value it is required to consider that the Hold-off timer already measured MIN_PWD, thus actually the maximum allowed pulse width = (MIN_PWD + RANGE_PWD). In other words, RANGE_PWD = (maximum allowed pulse width − MIN_PWD).*

IF MIN_PWD = 0x00 or RANGE_PWD = 0x00 no pulse width is performed.

The CHSR SCDF flag does not set if the pulse was finished by disabling the modulation (i.e., eTPU channel = 0 or SWMC = 0) or by disabling the channel, CHEN = 00, even if it ended shorter than MIN_PWD. However this flag can set in some situations that really indicates a short pulse detection but it is the result of some internal condition of the reaction module. The known situations are listed below:

- when the shared timer error occurs (TAER flag is set), a narrow pulse can be generated and SCDF flag is set.
- when the CHCR CHOFF bit is set, a narrow pulse can be generated and SCDF flag is set.

The CHSR OCDF flag only sets when the channel is enabled (CHEN not null) and the eTPU channel signal or SWMC is active too. However, the OCDF flag can set in some cases when the CHOFF bit is set. In this case, this OCDF flag should be disregarded because it is a false indication of the detector.

There can be a conflict of resource allocation if the Hold-off timer is used as the timer for the sequencer mode SM = 10. In this case it is not possible to detected minimum or maximum pulse widths thus the monitored modulation is deactivated. Which means the use of the Hold-off timer in the sequence mode has precedence over the monitored modulation. This configuration is not considered an error though, since it may occur during one of the phases of a modulation cycle and return to a sequence mode where the monitored modulation is possible. Thus no flags will be set to signal this conflict condition.

36.7 DMA support

The Reaction Module provides supports for one DMA channel per Reaction Channel. The DMA request signal is controlled by the DMAEN bit in the Channel Configuration Register *Figure 909* and by the DMA bit in the Modulation Control Word, *Figure 918*. If the DMAEN = 1 and the DMA = 1 then a DMA request is issued by the Reaction Channel. Note that the DMA request is deaserted if the DMA done signal is asserted even though the channel is still pointing to the same Modulation Control Word that generated the DMA request. In order for a new DMA request be issued after the DMA done is issued, the Reaction Channel must access a new Modulation Control Word or execute a new modulation cycle controlled by the timer input signal. *Figure 934* shows the DMA protocol executed by the Reaction Channel. The DMA request signal is asserted when the Modulation Word 1 is executed by the channel. This signal remains asserted until a DMA done signal is issued by the DMA controller.

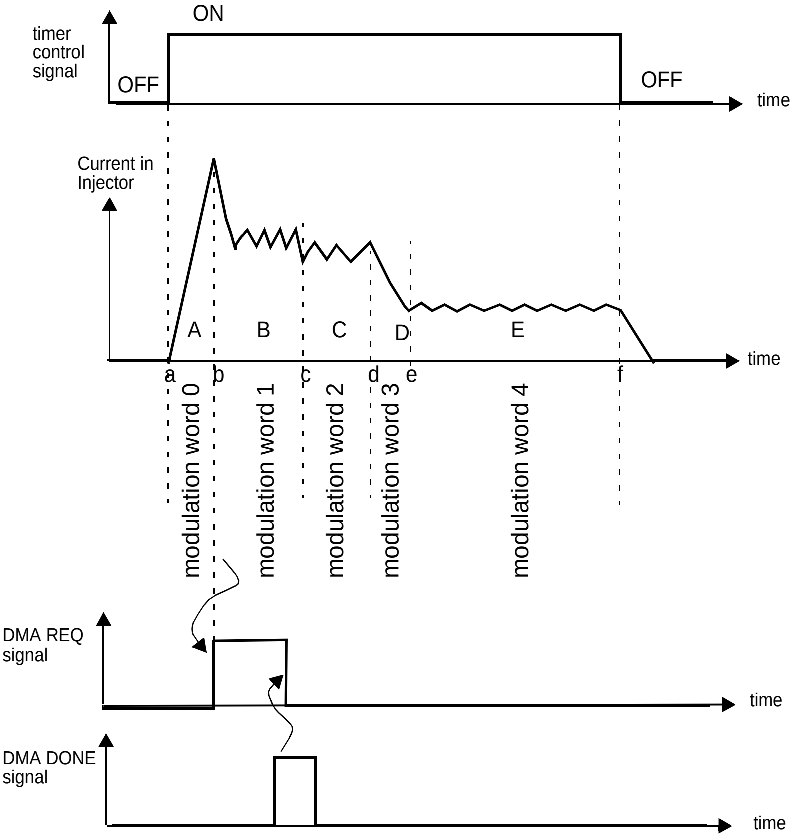


Figure 934. DMA Req/Done protocol

36.8 Reset overview

The Reaction Module is reset whenever any MCU reset occurs. In order to re-initialize the reaction channels the CHEN, channel enable register should be used. Once disabled the channel output is set to DOFF state and the channel configuration can be changed safely. The disable/enable operation does not change the channel setup, thus the configuration registers remain at the state as before the channel was disabled. The Modulation Control word addressed by the channel after the enable bit is asserted is defined by the MODULATION ADDR field in the channel configuration register.

36.9 Reaction module interrupts

The Reaction Module issues one global interrupt signal and one interrupt signal per channel. If using the Global interrupt signal the resolution of the interrupt source need to be performed by reading the Global Error Flag register to evaluate which channel issued the interrupt. After that the Channel Status register need to be read to distinguish between the several interrupt sources by evaluating the flags MAXL, OCDF, SCDF, TAER, and SQER.

36.9.1 Interrupt sources

There are several sources of interrupts that indicates a faulty condition:

- MAXL: maximum ADC result value was reached
- OCDF: open circuit detected which indicates an open circuit and thus a potential malfunction in the circuitry controlled by the Reaction Module. Note that differently from the TAER, this flag does not indicate a faulty condition in the channel but in the circuit outside the device.
- SCDF: short circuit detected which indicates a short circuit was detected on the off-chip logic controlled by the reaction channel.
- TAER: timer allocation error which indicates a required Timer resource was not allocated properly thus leading to faulty operation of the Reaction module.
- SQER: sequencer error occurred meaning that the timer input signal was deasserted in a modulation phase with SM != 00.

36.10 Use cases

Figure 935 shows an example of the Reaction Module used to control an Injector solenoid. Note that this is a dual injector which is also called banked injector. Two Reaction channels are used to control this injector.

The injector Boost transistor on the top applies a higher voltage in order to minimize the time necessary for the injector to start injecting fuel. Transistors A and B control which injector is active in the injector bank. A sensor resistor is used to feed the current flowing through the solenoid back to the on-chip ADC. The current is sampled by the ADC and the result is sent to the Reaction Module, allowing closed loop control.

Note: This injector bank architecture does not allow both injectors to operate at the same time since the sensor in the feedback loop is shared by both injectors.

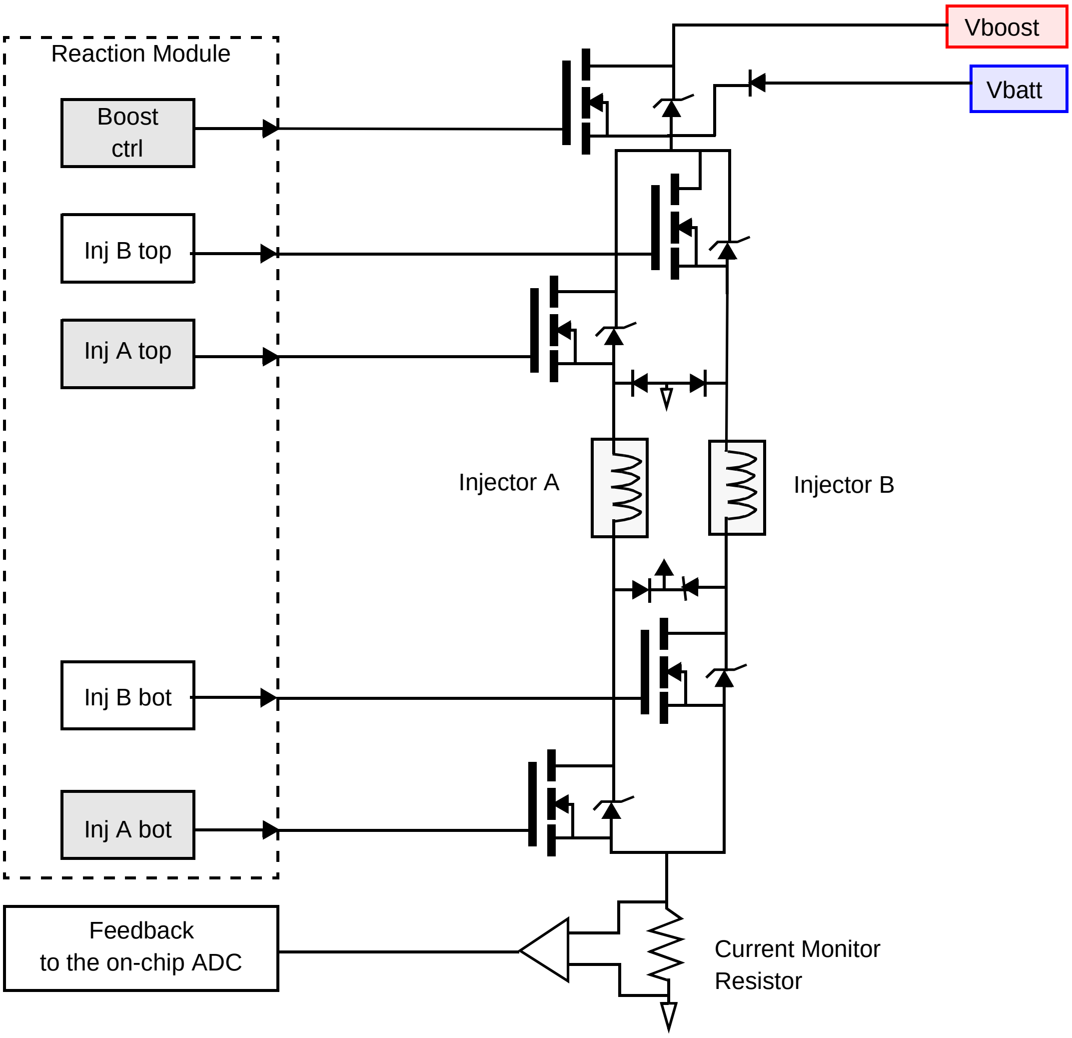


Figure 935. Boosted Banked Direct Injection with Passive Recirculation

Two eTPU channels are used to provide timing control signals, one for CH0 and one for CH1. The on-chip ADC monitors the sensor current periodically and send the digitalized results to the Reaction Module. In a banked configuration as shown in this example one ADC channel is used to monitor both injectors current. Note that only one channel is *active* at a given time since the eTPU time windows are not *active* at the same time for both reaction channels. Please see *Figure 936* for more details.

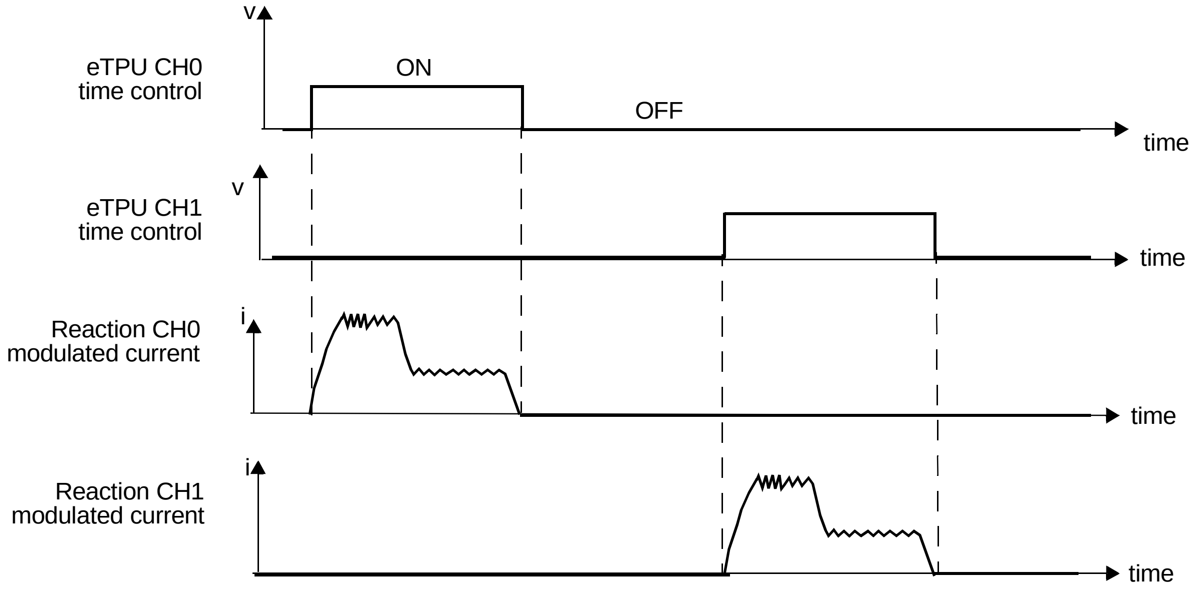


Figure 936. eTPU CH10/1 controlling reaction CH0/1

There are several possible configurations for a banked mode application. The objective on those configurations is to share hardware resources such as Reaction Module channels, ADC monitor inputs and MCU pins, among others related to the Injector driver which is not covered in detail in this document.

Figure 937 shows a more detailed diagram of the interconnection between the injector bank and the Reaction Module. Two reaction channels are used in this application. CH0 is used to control Injector A and CH1 is used to control Injector B. However, the Vboost/Vbatt selection is controlled by CH0 ch0_a output only since when Vboost driver is switched off Vbatt power source is applied to the injectors by the direct bias of the diode connecting Vbatt to the injector bank.

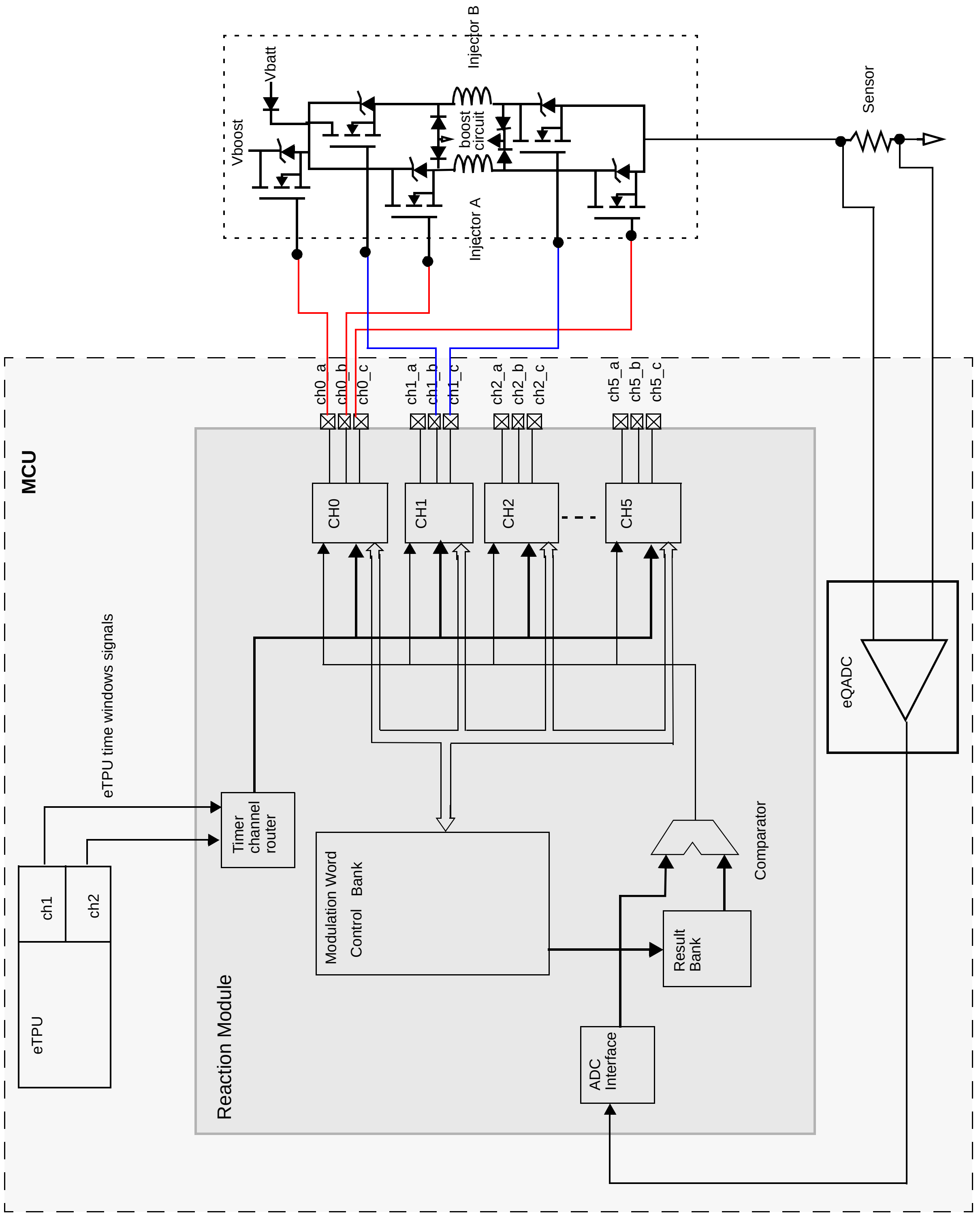


Figure 937. System level connection in a banked configuration

It is important to notice that even if two reaction channels control different injectors they can share the data stored in the Modulation Word Control. In this case both channels should execute the same type of modulation and use the same threshold values. Note also that the data stored in the Threshold bank in this case is also shared between these channels. This is an important feature of the Reaction Module architecture since it allows the sharing of resources and therefore provides savings in size without compromising the module functionality.

Figure 938 shows an example of the required current levels through Injector A and B. In order to generate this waveform, the Reaction Module uses one Modulation Control Word for each one of the five phases of the waveform from A through F. In this example the Module should be configured in the following way:

1. Set the REACM_CHRR0 CHIR[3:0] = 0x0, thus routing eTPU channel 0 to reaction channel 0
2. Set the REACM_CHRR1 CHIR[3:0] = 0x1, thus routing eTPU channel 1 to reaction channel 1
3. Set the REACM_CHRR0 ADCR[3:0] = 0x0, thus routing ADC TAG 0 to reaction channel 0
4. Set the REACM_CHRR1 ADCR[3:0] = 0x0, thus routing ADC TAG 0 to reaction channel 1
5. Program Modulation Word Control bank according to *Figure 939*
6. Program Shared Timer Bank REACM_STBK for addresses from 0 through 3 with timing intervals related to the duration of phases A,B,C and D respectively.
7. Program appropriate values in the Threshold Bank. Since threshold-threshold modulation is to be used in this example, four pairs of values should be provided for phases A,B,C and D respectively. Each pair corresponds to one address of the REACM_THBK starting at address 0x0400.
8. Program configuration registers for both channels, REACM_CHCR0/1. The parameters are DOFF[2:0] which defines the OFF state of the channel outputs and the MODULATION_ADDR = 0x0, which defines the address of the Modulation Control word. It is assumed that the Modulation Word zero is the first word to be accessed by both channels. Since four Modulation words will be used the addresses will be incremented by the reaction channel as needed, thus only the address for the first word is required. Note that MODULATON_ADDR = 0x0 points to the first Modulation Word in the Modulation Word Bank.
9. Program the prescalers HPRE and TPRE in the REACM_TCR register. Also enable the prescalers by setting the TPREN and HPREN bits in the REACM_MCR register.
10. Enable channels CH0 and CH1 to start the modulation sequence by programing field CHEN = 01 on REACM_CHCR1/0 registers. At this time the Reaction channel CH0 accesses the Modulation Control word zero and switches to ON state as defined by the data stored HOD[2:0] field. Up to this point any activity in the eTPU channel or income ADC result is ignored by the Reaction module. After CHEN field is programed, the reaction channels wait until a timer window is initiated by eTPU for the modulation process to start.

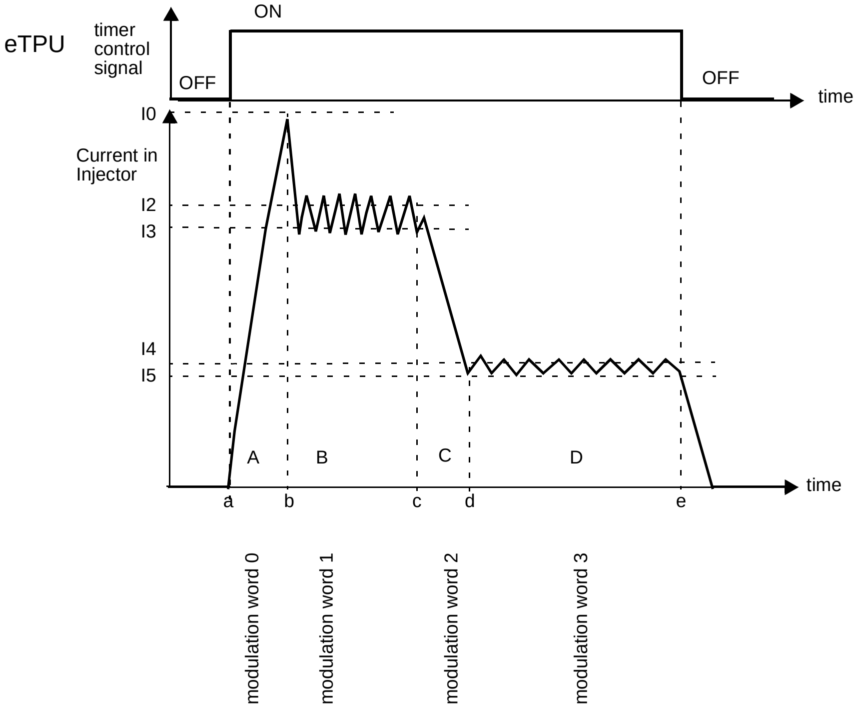


Figure 938. Modulation phases

The current through the injectors are initially zero since the reaction channel output is in the OFF state as configured by the DOFF[2:0] field. The eTPU timer window is at the OFF state as well. The modulation starts when the eTPU channel time window switches to ON state. Modulation *phase A* starts at this time as described in *Figure 938*. The modulation word 0 is used. In this application this phase corresponds to the *Boosted Peak phase*. This process allows the fuel injection to start sooner because of the sharp edge of the current which is important for a precise control of the fuel to be injected. *Phase A* is the setup to execute a Threshold-Threshold modulation but note that only *I0* value is used since the phase advances when a certain current is achieved. The following is a description of the bit fields in the Modulation word in order to execute the modulation described in *phase A*:

- LOOP = 0
- The initial value should be HOD (IOSS = 1)
- Threshold-Threshold modulation mode is obtained with MM = 00
- This phase ends when the threshold value *I0* is achieved (SM = 11)
- Necessary to have HOD = 111 for boosted operation and sensor active, LOD setting is not important in this case since it is not used
- *I0* is read from Threshold Bank by using THRESPT = 0x0 that points to address 0 of this bank
- *I1* is read from Threshold Bank by using (THRESPT + 1) = 0x1
- Hold-off timer is not used, therefore HDOFFPT can have any value (X)

At point b, *phase B* is initiated by the threshold being achieved from *phase A*. CH0 increments the Modulation Word address to MODULATION_ADDR = 0x1 and the second

Modulation Word is read by the channel. As a result of the second Modulation Word decoding the *Vboost* voltage is disabled causing a *peak* modulation with *Vbatt*. The phase is called the *Peak Vbatt phase*. For *phase B* a Threshold-Threshold modulation is used with levels *I2* and *I3* during a period defined by *TB*. Please see *Figure 939*.

A timeout event is received from the Shared Timer submodule and a new Modulation Word is read. *Phase C* corresponds to the *recirculation phase*. Energy from the injector is transferred back to the boost circuitry. In this phase the current can not be measured because there is no current flowing through the sensor resistor. A Threshold-Threshold modulation mode is used. The Shared Timer is started at the beginning of this phase and *TD* delay is measured. The channel outputs are kept in the OFF state.

When the Shared Timer times out after *TD* delay the Modulation Word address is incremented, MODULATION_ADDR = 0x4 and the *Hold phase* is initiated. This phase is typically longer compared to the other phases and defines the amount of fuel that will be injected. Threshold-Threshold modulation mode is used between levels *I4* and *I5* and *Vbatt* is selected as the power supply. The phase ends based on the eTPU time window switching to *off* at point *e*. At this time the channel outputs are set to OFF and the channel points to MODULATION_ADDR = 0x0 which is the address of the first Modulation Word. Note that this address is not necessarily 0x0.

This modulation process is executed by a sequence of Modulation Words as described in *Figure 939*.

Figure 939. Modulation words for injector application

Modulation Control Words REACM_MCW, see *Figure 920*

	0	1	2	3 4	5	6 7	8	9 10 11	12	13 14 15	16 17 18 19 20 21	22	23 24 25 26	27	28 29 30 31
Fields	**L**	**I**		**MM**		**SM**		**HOD**		**LOD**	**THRESHPT**		**STPT**		**HDOFFPT**
PhaseA	0	1		00		11		111		101	000010 (I0) 000011 (I1)		0001 (TB)		X
Phase B	0	0		00		01		011		001	000010 (I2) 000011 (I3)		0010 (TC)		X
Phase C	0	0		00		01		000		000	X (1)		0011 (TD)		X
Phase D	0	0		00		00		011		001	000100 (I4) 000101 (I5)		X		X

1. Any value

36.10.1 Advancing modulation phase on a threshold level

The Modulation Phase may be set to advance when a specific threshold value is reached. The use case for this scenario is described in *Figure 940*. This functionality is used to assure a specific current level was achieved before the reaction channel advances to the next modulation phase, thus making sure that the solenoid had the fastest opening speed.

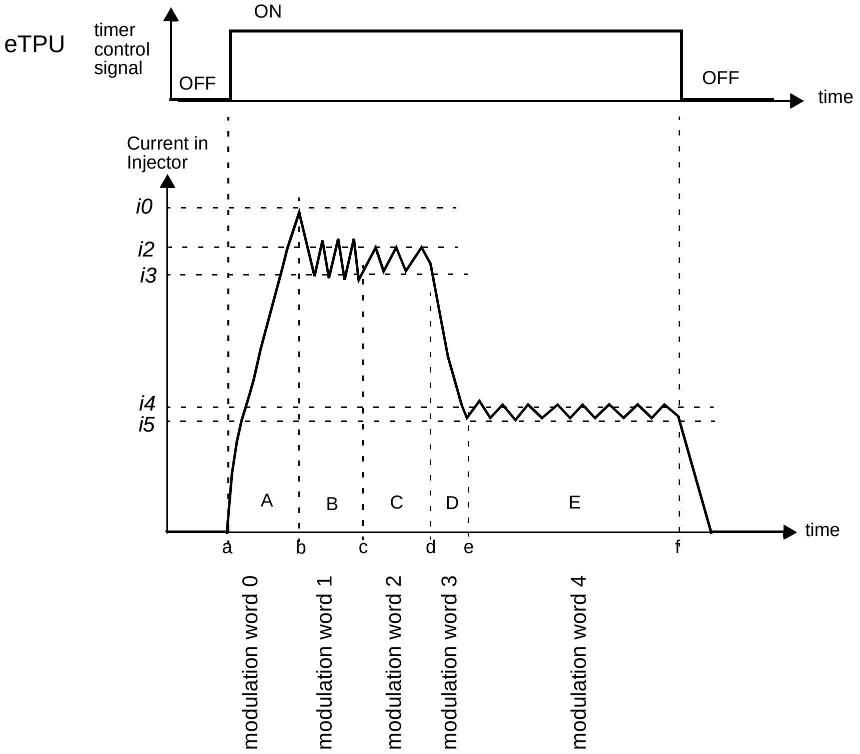


Figure 940. Advancing modulation phase on a threshold level

In the example shown in *Figure 940* the Modulation Word 0 has the field SM = 11 meaning that it should advance if the ADC comparator, which compares ADC results and threshold values, is true. This means that the ADC result is greater or equal to the threshold value pointed by THRESPT Modulation Word 0 field. The advance from phase A to phase B in *Figure 940* is described as follows:

- Initially at time *a* the reaction channel output is loaded with HOD[2:0] as indicated by IOSS value, both fields of modulation word 0.
- If ADC result ≥ [THRESPT] the reaction channel turns the outputs off by loading LOD[2:0] to the channel outputs and advance to the next modulation cycle at time *b*.

Note: The advance on threshold, SM = 11, is intended to be used with IOSS = 1, thus the advance occurs when the level from the ADC is greater or equal to the value pointed by THRESPT. IOSS = 0 is a reserved value for this bit in this configuration and should not be used.

36.10.2 Controlling the loop function

The LOOP field in the Modulation Word controls the sequencing of Modulation Words to be executed by the channel. If LOOP = 1 and a phase is ended the next Modulation Word address returns to the initial modulation address programmed in the channel when the cycle was initiated. In this case a loop in the Modulation Bank is created. This loop ends when the cycle ends.

Figure 941 shows the sequence of Modulation Words within a Modulation Cycle if LOOP function is used. Note that an alternate modulated waveform is generated using only two Modulation Words, in this case Modulation Word 0 and Modulation Word 1.

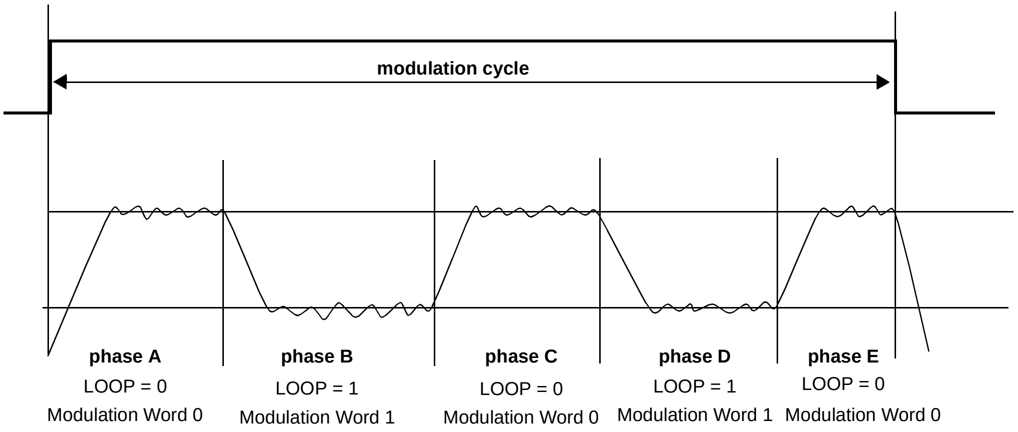


Figure 941. LOOP function used within a modulation cycle

36.10.3 Banked mode

Figure 942 describes the interconnection of four channels controlling two injector banks. Note that two output pins are not connected since the boost control is shared between channels [0] and [1] and channels [2] and [3].

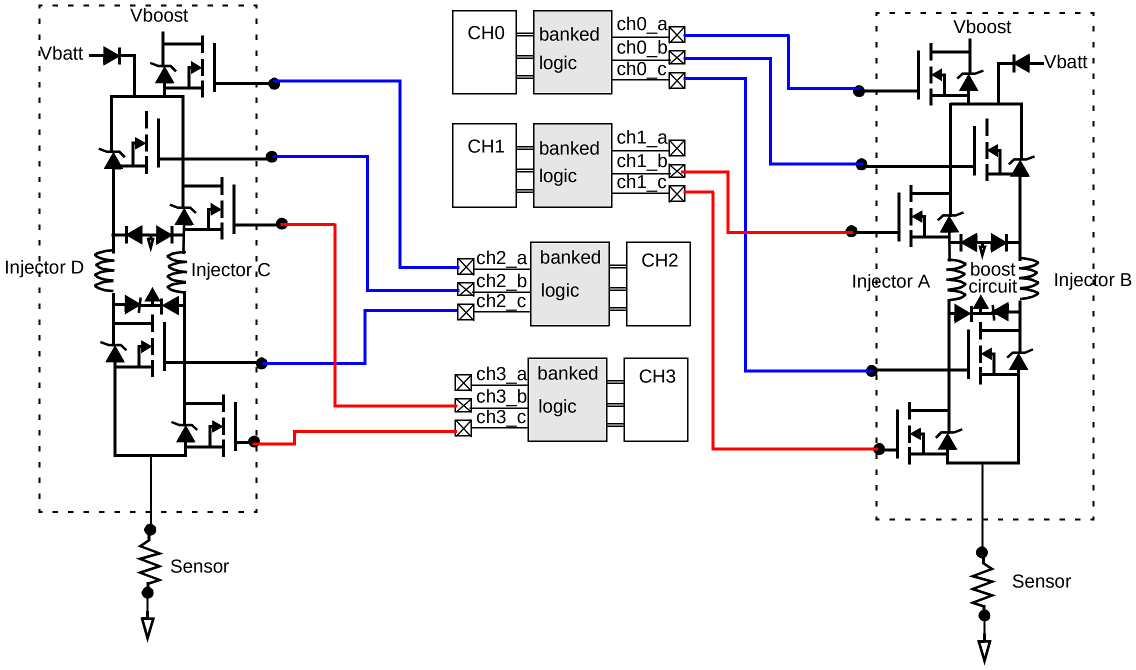


Figure 942. Four channels controlling two injector banks in banked mode

37 Temperature Sensor

37.1 Overview

SPC564A70 MCUs include an onboard temperature sensor that monitors device temperature and produces a voltage directly proportional to the internal junction temperature. Internal junction temperature must be calculated by software based on the sampled temperature sensor voltage, sampled bandgap voltage and calibration parameter values stored in internal flash memory.

37.2 Detailed description

The temperature sensor generates a voltage that increases linearly with temperature. Since the voltage is an amplified version of a ΔV_{BE} voltage it is proportional to absolute temperature. This voltage, $V_{TSENS}(T)$, is read by software using the onboard eQADC module and used with the bandgap voltage and constants stored in flash memory during factory test to calculate device junction temperature.

Five calibration parameters are stored in flash memory during factory test:

- T_{LOW} is the low temperature factory calibration temperature value.
- T_{HIGH} is the hot factory calibration temperature value.
- $V_{BG\_CODE}(T_{LOW})$ is the bandgap voltage at low calibration temperature (T_{LOW}) sampled by the eQADC and converted to a 14-bit value.
- $T_{TSENS\_CODE}(T_{LOW})$ is the temperature sensor voltage at low calibration temperature (T_{LOW}) sampled by the eQADC and converted to a 14-bit value.
- $T_{TSENS\_CODE}(T_{HIGH})$ is the temperature sensor voltage at high calibration temperature (T_{HIGH}) sampled by the eQADC and converted to a 14-bit value.

The calibration points are illustrated in *Figure 943*.

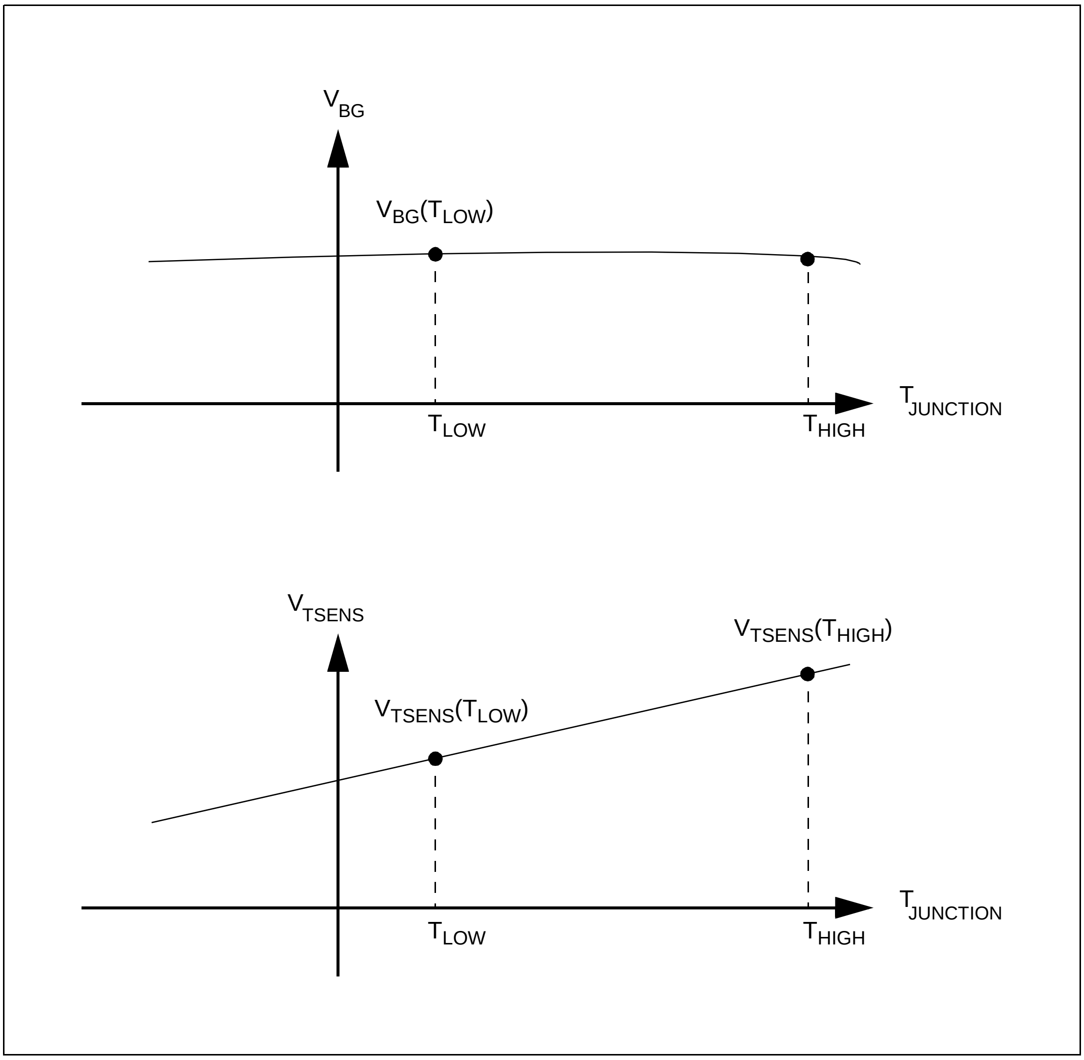


Figure 943. Calibration points

37.3 Temperature formula

The temperature formula is shown in *Figure 944*.

$$T = T_{LOW} + \frac{T_{TSENS\_CODE}(T) \times \beta - T_{TSENS\_CODE}(T_{LOW})}{T_{TSENS\_CODE}(T_{HIGH}) - T_{TSENS\_CODE}(T_{LOW})} \times (T_{HIGH} - T_{LOW})$$

where:

$$T_{TSENS\_CODE}(T_{LOW}) = \frac{V_{TSENS}(T_{LOW})}{V_{ref0}} \times 2^{14}$$ (Stored in device flash during factory calibration)

$$T_{TSENS\_CODE}(T_{HIGH}) = \frac{V_{TSENS}(T_{HIGH})}{V_{ref0}} \times 2^{14}$$ (Stored in device flash during factory calibration)

$$V_{BG\_CODE}(T_{LOW}) = \frac{V_{BG}(T_{LOW})}{V_{ref0}} \times 2^{14}$$ (Stored in device flash during factory calibration)

$$V_{BG\_CODE}(T) = \frac{V_{BG}(T)}{V_{ref}} \times 2^{14}$$

$$T_{TSENS\_CODE}(T) = \frac{V_{TSENS}(T)}{V_{ref}} \times 2^{14}$$

$$\beta = \frac{V_{BG\_CODE}(T_{LOW})}{V_{BG\_CODE}(T)}$$

Notes:

1. $V_{TSENS}(T)$ is the temperature sensor output sampled by the ADC.
2. $V_{BG}(T)$ is the bandgap voltage sampled by the ADC.
3. V_{ref} is the ADC reference voltage.
4. V_{ref0} is the ADC reference voltage during factory calibration.
5. T_{LOW} is the low temperature factory calibration temperature (stored in device flash).
6. T_{HIGH} is the hot factory calibration temperature (stored in device flash).

Figure 944. Temperature formula

The following sections detail the values required and where to obtain them.

37.3.1 T_{LOW} and T_{HIGH}

T_{LOW} is the factory low calibration temperature; T_{HIGH} is the hot factory calibration temperature. These values are stored in shadow flash memory during factory calibration. See *Section , Temperature Calculation Constants Register 0 (TSENS_TCCR0)*for details.

37.3.2 $T_{TSENS\_CODE}(T_{LOW})$ and $T_{TSENS\_CODE}(T_{HIGH})$

$T_{TSENS\_CODE}(T_{LOW})$ is the sampled output voltage of the temperature sensor during low temperature factory calibration. $T_{TSENS\_CODE}(T_{HIGH})$ is the sampled output voltage of the temperature sensor during hot temperature factory calibration. These values are stored in shadow flash memory during factory calibration. See *Section , Temperature Calculation Constants Register 0 (TSENS_TCCR0)* for details.

37.3.3 $V_{BG\_CODE}(T_{LOW})$

$V_{BG\_CODE}(T_{LOW})$ is the value of the bandgap voltage sampled during low temperature factory calibration. This value is stored in shadow flash memory during factory calibration. See *Section , Temperature Calculation Constants Register 1 (TSENS_TCCR1)* for details.

37.3.4 Temperature sensor voltage ($V_{TENS}(T)$)

$V_{TENS}(T)$ is the output voltage of the device temperature sensor. Software must sample the voltage from eQADC_A channel 128 (ADC0 and ADC1).

37.3.5 Bandgap reference voltage ($V_{BG\_CODE}(T)$)

V_{BG} is the bandgap reference voltage. Software must sample the voltage from eQADC_A channel 144 (ADC0).

37.3.6 Registers

The calibration constants described previously, that is, T_{LOW}, T_{HIGH}, $T_{SENS\_CODE}(T_{LOW})$, $T_{SENS\_CODE}(T_{HIGH})$, and $V_{BG\_CODE}(T_{LOW})$, are stored in device shadow flash memory during factory test. This section details the registers where the values reside.

Temperature Calculation Constants Register 0 (TSENS_TCCR0)

This register contains the calibration temperatures and temperature sensor outputs measured during factory calibration:

- T_{HIGH}
- $T_{SENS\_CODE}(T_{HIGH})$
- T_{LOW}
- $T_{SENS\_CODE}(T_{LOW})$

Address: 0xFFFE_C000

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TSCV2															
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	TSCV1															
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

▨ = Unimplemented or Reserved

Figure 945. Temperature Calculation Constants Register 0 (TSENS_TCCR0)

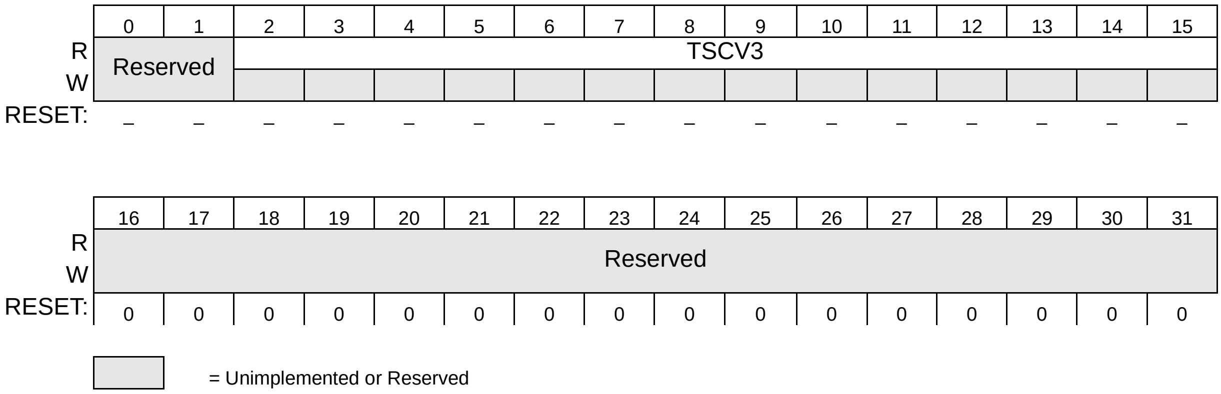
Table 893. Temperature Calculation Constants Register 0 (TSENS_TCCR0) field descriptions

Field	Bits	Description
TSCV2	0–15	Combination of encoded hot factory calibration temperature (T_{HIGH}) and the temperature sensor output at that temperature (TSENS_CODE(T_{HIGH})). Bits 0–1 contain a value representing the hot factory calibration temperature (T_{HIGH}). The values are as follows: – 00: T_{HIGH} = Reserved – 01: T_{HIGH} = Reserved – 10: T_{HIGH} = 145 °C – 11: T_{HIGH} = 150 °C Bits 2–15 are the temperature sensor voltage sampled and converted by the eQADC during factory test with device at hot temperature (T_{HIGH}). This is the $T_{SENS\_CODE}(T_{HIGH})$ parameter value referenced in the temperature calculation formula (see *Figure 944*). The reset value of this register is device-dependent. The value is set during factory test.
TSCV1	16–31	Combination of encoded low factory calibration temperature (T_{LOW}) and the temperature sensor output at that temperature (TSENS_CODE(T_{LOW})). Bits 16–17 contain a code indicating the value of T_{LOW}. The values are as follows: – 00: T_{LOW} = 25 °C – 01: T_{LOW} = -40 °C – 10: T_{LOW} = Reserved – 11: T_{LOW} = Reserved Bits 18–31 are the temperature sensor voltage sampled and converted by the eQADC during factory test with device at the low calibration temperature. This is the $T_{SENS\_CODE}(T_{LOW})$ parameter value referenced in the temperature calculation formula (see *Figure 944*). The reset value of this register is device-dependent. The value is set during factory test.

Temperature Calculation Constants Register 1 (TSENS_TCCR1)

This register contains the $V_{BG\_CODE}(T_{LOW})$ parameter value used in the temperature calculation.

Address: 0xFFFE_C004



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	Reserved		TSCV3													
W			▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	Reserved															
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 946. Temperature Calculation Constants Register 1 (TSENS_TCCR1)

Table 894. Temperature Calculation Constants Register 1 (TSENS_TCCR1) field descriptions

Field	Bits	Description
Reserved	0–1	Reserved
TSCV3	2–15	Bandgap voltage sampled and converted by ADC during factory test. This is the $V_{BG\_CODE}(T_{LOW})$ parameter value referenced in the temperature calculation formula (see *Figure 944*). The reset value of this register is device-dependent. The value is set during factory test.
Reserved	16–31	Reserved

38 Revision history

Table 895. Document revision history

Date	Revision	Changes
25-Oct-2010	1	Initial release.
01-Feb-2012	2	*Chapter 2: Memory map* *Table 2 (SPC564A70 Memory Map)*: deleted the ‘Used Size’ entry for ‘Start Address’ of 0xC3FB_C000. *Chapter 4: Resets* *Figure 5 (Reset configuration timing)* - replaced “WKPCFG, PLLREF, BOOTCFG[0:1] and PLLCFG1 are latched.” with “WKPCFG, PLLREF, BOOTCFG[0:1]” *Chapter 5: Operating Modes and Clocking* *Figure 6 (System clock diagram)* updated. Changed the Note of *Section , External Bus Clock (CLKOUT)* from “The CLKOUT pin is only available in the 208- and 324-pin packages” to “The CLKOUT pin is only available in the 324-pin package”. Updated text of *Section , Support for 150 MHz system clock generation* to “A possible PLL configuration is shown below: • Input clock (crystal frequency): 40 MHz • EPREDIV/IDF divider = /8 (1–15 range supported) • EMFD/NDIV loop divider = 60 (32–96 supported) • VCO clock out = 300 MHz (256–512 MHz range supported) • ERFD/ODF output divider = /2 (/2, /4, /8, /16 supported)”. Updated the following texts of *Section : Support for 100 MHz system clock generation* to • EPREDIV/IDF divider = /8 (1–15 range supported) • ERFD/ODF output divider = /4 (/2, /4, /8, /16 supported). *Chapter 6: Device Performance Optimization* *Section , Recommended configuration*: • Changed bit 27 in *Figure 14 (L1 Cache Control and Status Register 1 (L1CSR1))* from “0” to “ICORG”; • Removed instance of z7 Added Description for “ICORG” bit. *Chapter 8: Enhanced Direct Memory Access Controller (eDMA)* *Table 43 (DMA request summary for eDMA)*: • Removed the mention of eTPUB in the RM. • Changed the ‘Description’ and ‘Source’ for the channels 32, 33 and 52 - 63. • Changed “DMA Request” from “Reserved” to “No request” . • Changed “Source” and “Description” from “Reserved” to “—”. Updated *Section 8.5.8: Dynamic programming*

Table 895. Document revision history (continued)

Date	Revision	Changes
01-Feb-2012	2 (continued)	*Chapter 9: Multi-Layer AHB Crossbar Switch (XBAR)* *Figure 49 (Master Priority Register (XBAR_MPRn))*: changed the reset value of bit 1, from 1 to 0. *Figure 50 (Slave General Purpose Control Register (XBAR_SGPCRn))*: • Removed HPE7 and HPE2 bits • Added HPE6, HPE4 and HPE1 bits • Bit “14” from “0” to “HPE1” and access to Read/Write. • Bit “13” from “HPE2” to “0” and access to “Read only zero” • Bit “11” from “0” to “HPE4” and access to Read/Write. Bit “9” from “0” to “HPE6” and access to Read/Write. *Chapter 10: Peripheral Bridge (PBRIDGE)* *Table 59 (Off-platform Peripheral Access Control Register (OPACR) fields)*: removed OPACR 65 and OPACR 67 rows. *Chapter 11: Flash memory* Changed BIUCR reset and BIUAPR reset bits. Added Section 12.5.4, UTest Mode. Replaced UMx with UMISRx, throughout. *Table 56 (Module Configuration Register (MCR))*: • Replaced reset value of SIZE bits with 0b101, was 0b011 • Replaced reset value of LAS bits with 0b100, was 0b110 *Table 60 (Flash memory map)*: • removed references to Flash A • Added a new row “0x00F0_0000 Reserved” *Table 61 (Flash Shadow block mapping)* • Replaced 0x0010_0000 with 0x00FF_FFF for LMLR reset configuration • Added NVUSR0 row *Table 62 (Flash configuration register memory map)*: • removed access and reset columns • in the footnote removed FLASH_B_REGS_BASE = 0xC3F8_C000. *Section 11.2, External signal description*: changed V_{PP} with V_{FLASH} *Section , Address Register (AR)*: changed ADDR[15:0] to ADDR[14:0] *Section , User Test 0 (UT0) Register*: added EA bit filed. *Figure 55 (Flash system block diagram)*: • changed Flash_A with Flash. • changed V_{FLASH_A} with V_{FLASH} • changed V_{PP} with V_{FLASH} • changed “Flash bus interface unit (FBIU)”with “Platform Flash bus interface unit (PFBIU)”. *Section 11.4.2, Flash Read and Write*: added sentence concerning return data. *Section 11.4.3, Read While Write (RWW)*: replaced 128 KB with 256 KB. *Section 11.4.5, Flash Programming*: added a warning concerning over programming. *Section 11.4.6, Flash Erase*: updated

Table 895. Document revision history (continued)

Date	Revision	Changes
01-Feb-2012	2 (continued)	*Chapter 15: System Integration Unit (SIU)* *Figure 230 (Pad Configuration Register (SIU_PCR219))*: added two footnotes about OBE and IBE bit fields. *Table 227 (SIU_PCR215 PA values)*, changed the “I/O” value from “I/O” to “O”. *Table 273 (SIU_ECCR field description)* bit “ENGDICV” changed the equation *Table 255 (SIU_PCR350 – SIU_PCR381 DSPI muxing)*: —Removed column “DSPI deserialize destination”. —Updated column “PA value” by changing “0b01” to “0b001” “0b11” to “0b100” “0b10” to “0b010” “0b00” to “0b000”. *Table 256 (SIU_PCR382 – SIU_PCR389 DSPI muxing)*: —Removed column “DSPI deserialize destination”. —Updated column “PA value” by changing “0b011” to “0b100” *Table 257 (SIU_PCR390 – SIU_PCR413 DSPI muxing)*: —Removed column “DSPI deserialize destination”. —Updated column “PA value” by changing “0b01” to “0b001” “0b11” to “0b100” “0b10” to “0b010” “0b00” to “0b000”. *Section , Pad Configuration Registers 75–82 (SIU_PCR75–SIU_PCR82)*: Modified the note. *Chapter 16: Frequency-modulated phase locked loop (FMPLL)* Replaced instances of f_{ref} with f_{fbk} throughout the chapter. *Section 16.5.2, Clock configuration*: • Added a paragraph. • Added note “Maximum system clock frequency is 150 MHz and max/min VCO frequency is 256 MHz to 512 MHz”. *Chapter 20: Boot Assist Module (BAM)* *Table 329 (Possible RCHW locations in the internal flash)* updated *Chapter 22: Enhanced Time Processing Unit (eTPU2)* *Table 467 (Output disable channel groups)*: added a footnote.

Table 895. Document revision history (continued)

Date	Revision	Changes
01-Feb-2012	2 (continued)	*Chapter 23: Enhanced Queued Analog-to-Digital Converter (EQADC)* Placed register field desriptions into a table for all registers. *Section 23.2.2, Block diagram*: • Added foot note “Decimation filters A and B and Reaction module”. • Added information about Decimation filters A and B and about reaction module. • In *Section , Distributing Result Data into RFIFOs* added information about Decimation filters A and B and about reaction module. In *Table 542 (Non-multiplexed Channel Assignments)*and *Table 543 (Multiplexed Channel Assignments)* for channel 45 updated “Bandgap” to “Buffered bandgap”. *Chapter 25: Decimation Filter* *Table 566 (DECFILTER_MXCR Register Field Descriptions)*: • For bits SZROSEL[1:0], SRQSEL[2:0] and SENSEL[1:0] • The hardware input signals are ZSELA for Decimation filter A and ZSELB fractionation filter B • For bit SHLTSEL[1:0] • The hardware input signals are HSELA for Decimation filter A and HSEB for Decimation filter B. *Figure 546 (Decimation filter block diagram)*: replaced “By-passk” with “Bypass”. Section 26.6.3, Input buffer description, restored section title. Updated *Section 25.5.10, Soft-reset command description* *Section , Block interrupt request* and *Section 25.5.13, Freeze mode description*: replaced “DEC_FILTER” with “DECFILTER” *Table 557 (Decimation filter device memory map)*: removed access and reset columns. *Table 584 (Decimation filter cascade mode data bus field description)*: created usign plain text from the same section. *Chapter 30, Deserial Serial Peripheral Interface (DSPI)* *Section , Stop mode (External Stop mode)*, removed figure. *Figure 726 (DSPI DSI Configuration Register 1 (DSPI_DSICR1))*, added DSE[1:0] bits. *Table 698 (DSPI_DSICR1 field description)*, added DSE[1:0] bits. In *Section , DSPI DSI Configuration Register (DSPI_DSICR)*. Added bits DMS,PES,PE,PP in DSPI_DSICR register. Added the following registers —DSPI Hardware Configuration Register (DSPI_HCR) —DSPI DSI Serialization Source Select Register (DSPI_SSR) —DSPI DSI Parallel Input Select Register 0 (DPSI_PISR0) —DSPI DSI Parallel Input Select Register 1 (DPSI_PISR1) —DSPI DSI Parallel Input Select Register 2 (DPSI_PISR2) —DSPI DSI Parallel Input Select Register 3 (DPSI_PISR3) —DSPI DSI Deserialized Data Interrupt Mask Register (DSPI_DIMR) —DSPI DSI Deserialized Data Polarity Interrupt Register (DSPI_DPIR)

Table 895. Document revision history (continued)

Date	Revision	Changes
01-Feb-2012	2 (continued)	*Chapter 31: Enhanced Serial Communication Interface (eSCI)* *Section 31.1, Introduction* - Removed reference to LIN 2.1 *Section 31.1.1, Bibliography*: - added reference to SAE J2602 *Section 31.3.2.2, Control register 1 (eSCI_CR1)* - enhancement: add description for upper/lower byte access handling - Changed access of Bits “0 to15” to “Read/Write”. *Section 31.3.2.3, Control register 2 (eSCI_CR2)* - fixed typo: reset value of BSTP control bit to 1 - refined description of TXDIR - SBSTP: updated description, removed stop of internal operation and tx stopped as long as BERR is asserted. - TXDMA, RXDMA: DMA request deasserted on bit clear. - changing bit: “BRK13” to “BRCL” “BESM13” to “BESM” “SBSTP” to “BESTP”. *Section 31.3.2.4, SCI data register (eSCI_DR)* - changed read data in LIN mode to undefined - changing bit: “R8” to “RN” “R” to “RD[11:8]”. *Section 31.3.2.4, Status register (eSCI_SR)* - updated description of TXRDY set condition (content processed) - fixed reset value for TDRE to 1 - fixed reset value for TXRDY to 1 - TXRDY: added wakeup transmitted condition - NF: noise flag is set whenever noise has been detected - RAF: updated description, reference to receiver section - RDRF asserted only when receive DMA mode disabled - RXRDY asserted only when receive DMA mode disabled - Updated reset value from “0” to “1” for bits “TDRE-bit 31” and “TXRDY-bit 14” *Section 31.3.2.5, LIN control register (eSCI_LCR)* - removed LDBG control bit, removed automatic LIN protocol engine reset on bit/bus error - removed DSF control bit, no extra stop bit transmitted on bit error

Table 895. Document revision history (continued)

Date	Revision	Changes
01-Feb-2012	2 (continued)	*Chapter 31: Enhanced Serial Communication Interface (eSCI)* *Section 31.3.2.9, LIN transmit register (eSCI_LTR)* - added register read value - added condition of non-successful write - added write mode: LIN Mode Added *Section 31.3.2.12, Control register 3 (eSCI_CR3)* *Table 31-10 (Transmitter Module Transitions)* - fixed description of TC generation in done *Table 31-11 (Transmit Source Priority)* - fixed transmit priorities (Preamble > Break > Data) *Section 31.1.6, Modes of operation* Added *Section 31.1.6.2, SCI mode* and *Section 31.1.6.3, LIN mode* Deleted “RUN mode” section *Section 31.1.6.1, Module idle condition* - added description of module idle *Section 31.1.6.4, Disabled mode* - added idle condition Added the string interrupt to all flags listed in *Table 31-19 (eSCI Interrupt Flags and Interrupt Enable Bits)*. *Section 31.4.5.3.4, Single wire mode* removed the text “The TXDIR bit (eSCI_CR2[1]) determines whether the TXD pin is going to be used as an input (TXDIR= 0) or an output (TXDIR = 1) in this mode of operation”. *Section 31.4.5.3.8, Idle character detection* - removed start of idle detection after low data bit option *Section 31.4.5.3.15, Start Bit Sampling* - refined description of no ongoing reception, related to preceding stop bit. *Section 31.4.6.2.1, LIN byte field reception* - added section *Section 31.4.6.5.3, Standard bit error detection* - fixed compare point to end of byte field reception Added *Section 31.4.6.5.5, Slave-not-responding-error detection* instead of “Slave timeout detection” section *Section 31.4.6.6, LIN wake up* - Updated it and its subsections - added *Section 31.4.6.7, LIN protocol engine stop and reset*

Table 895. Document revision history (continued)

Date	Revision	Changes
01-Feb-2012	2 (continued)	*Chapter 32, FlexCAN Module* *Section 32.2.3, Modes of operation*, replaced the text with “This low power mode...sub-modules” under bullet “Module Disable Mode”. *Section , Module Disable Mode*, replaced the text with “This low power mode...and negates the FRZ_ACK bit”. *Section 29.5.2, Transmit process*, deleted text and added “The deactivated message... CODE field”. *Section , Transmission abort mechanism*, added note “An Abort request to a TxMB... CAN bus arbitration”. *Table 683 (ESR Register field descriptions)*, updated the “Description” for “Field” TXWRN to “TX Error Warning” RXWRN to “RX Error Warning”. *Section , Error and Status Register (ESR)*, changed the text from “The CPU read action clears bits 16–23” to “The CPU read action clears bits 16–21”. *Chapter 33: JTAG Controller (JTAGC)* *Table 848 (Device identification register parameters)*, corrected the description for “MIC” field. *Figure 878 (5-bit Instruction Register)*, changed the Reset value for bit 0 to “0”. *Chapter 34, Nexus Port Controller (NPC)* *Table 854 (Nexus trace port routing and speed)*, row “Debug/Cal package” changes done are as follows: - In column “Port routing bit NPC_PCR[NEXCFG]” changed the value to “Don’t care”. - In column “CAL_MDO[4:11] usage changed the value to “Trace port use”. *Table 855 (Parameter values)*, added values of DC, PIN, and PRN. *Chapter 36: Reaction Module (REACM)* *Table 874 (REACM_MCR field descriptions)* Note added to REACM_MCR[GIEN]: The GIEN bit only affects the general interrupt signal, and not the individual channel interrupts. GIEN=0 does not inhibit the channel interrupts. *Table 424 (REACM_MWBK field descriptions)* Note added to REACM_MWBK[IOSS] and REACM_MWBK[MM]: IOSS must not be 0b0 when MM is 0b01(threshold-holdoff). Note added to REACM_MWBK[HOD] and REACM_MWBK[LOD]: Set HOD[2:0]=LOD[2:0] when using threshold-threshold modulation together with sequence advance, in order to avoid fast glitches during the sequence advance. *Figure 901 (Reaction module system interconnection)*: changed “register access bus can be driven by CPU or eTPU” with “register access bus can be driven by CPU” deleted arrow between eTPU and Reaction module blocks. In *Section 36.1.1, Features* added a note “DMA is not supported in Andorra devices”.
18-Sep-2013	3	Updated Disclaimer.

Please Read Carefully:

Information in this document is provided solely in connection with ST products. STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, modifications or improvements, to this document, and the products and services described herein at any time, without notice.

All ST products are sold pursuant to ST's terms and conditions of sale.

Purchasers are solely responsible for the choice, selection and use of the ST products and services described herein, and ST assumes no liability whatsoever relating to the choice, selection or use of the ST products and services described herein.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted under this document. If any part of this document refers to any third party products or services it shall not be deemed a license grant by ST for the use of such third party products or services, or any intellectual property contained therein or considered as a warranty covering the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

UNLESS OTHERWISE SET FORTH IN ST'S TERMS AND CONDITIONS OF SALE ST DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE USE AND/OR SALE OF ST PRODUCTS INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION), OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

ST PRODUCTS ARE NOT DESIGNED OR AUTHORIZED FOR USE IN: (A) SAFETY CRITICAL APPLICATIONS SUCH AS LIFE SUPPORTING, ACTIVE IMPLANTED DEVICES OR SYSTEMS WITH PRODUCT FUNCTIONAL SAFETY REQUIREMENTS; (B) AERONAUTIC APPLICATIONS; (C) AUTOMOTIVE APPLICATIONS OR ENVIRONMENTS, AND/OR (D) AEROSPACE APPLICATIONS OR ENVIRONMENTS. WHERE ST PRODUCTS ARE NOT DESIGNED FOR SUCH USE, THE PURCHASER SHALL USE PRODUCTS AT PURCHASER'S SOLE RISK, EVEN IF ST HAS BEEN INFORMED IN WRITING OF SUCH USAGE, UNLESS A PRODUCT IS EXPRESSLY DESIGNATED BY ST AS BEING INTENDED FOR "AUTOMOTIVE, AUTOMOTIVE SAFETY OR MEDICAL" INDUSTRY DOMAINS ACCORDING TO ST PRODUCT DESIGN SPECIFICATIONS. PRODUCTS FORMALLY ESCC, QML OR JAN QUALIFIED ARE DEEMED SUITABLE FOR USE IN AEROSPACE BY THE CORRESPONDING GOVERNMENTAL AGENCY.

Resale of ST products with provisions different from the statements and/or technical features set forth in this document shall immediately void any warranty granted by ST for the ST product or service described herein and shall not create or extend in any manner whatsoever, any liability of ST.

ST and the ST logo are trademarks or registered trademarks of ST in various countries.
Information in this document supersedes and replaces all information previously supplied.
The ST logo is a registered trademark of STMicroelectronics. All other names are the property of their respective owners.

© 2013 STMicroelectronics - All rights reserved

STMicroelectronics group of companies

Australia - Belgium - Brazil - Canada - China - Czech Republic - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Philippines - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States of America

www.st.com