AN3324
Application note

Implementing power-on self tests for SPC56EL60 in locked step

Introduction

SPC56EL60 is a 32-bit system-on-chip (SoC) automotive microcontroller designed for safety applications with a focus to minimize software measures within the CPU core subsystem.

In order to reach this state, several software measures are required during the MCU power-on start-up procedure. This application note describes the software measures that user must perform after the boot in order to detect and manage latent faults.

This document is valid only under the assumption that the MCU is used in locked step for automotive applications with fail-silent or fail-indicate micros.

This application note is based on *AN3077* rev. 2 (see *B.1: Reference documents*).

All the topics covered in this document also refer to *RM0032* rev. 5, *SPC56EL60L3, SPC56EL60L5* datasheet rev. 5 and *AN3121* rev. 1 (see *B.1: Reference documents* in *Appendix B*).

This application note applies to SPC56EL60 devices according to *Table 1*.

Table 1. Device summary

Part number	Package
SPC56EL60L3	LQFP100 (3.3 V)
SPC56EL60L5	LQFP144 (3.3 V)

Contents

1 Document hierarchy 6

2 How to implement power-on self test features 7

2.1 MCU initialization 7

2.1.1 Safety initialization 8

2.2 Safety verification and faults checking 10

3 Module software requirements for non applicative peripherals 12

3.1 System Status and Configuration Module (SSCM) 12

3.2 Self Test Control Unit (STCU) 12

3.3 Redundancy Control Checker Unit (RCCU) 13

3.4 Reset Generation Module (MC_RGM) 13

3.5 Fault Collection and Control Unit (FCCU) 13

3.6 Clock configuration 15

3.7 Clock Monitor Unit (CMU) 18

3.8 Frequency-Modulated Phase-Locked Loop (FMPLL) 18

3.9 Internal RC Oscillator (IRCOSC) 19

3.10 Flash memory 19

3.10.1 Array integrity self check procedure 20

3.11 Temperature sensors 21

3.12 Software Watchdog Timer (SWT) 21

3.13 Power Management Unit (PMU) 22

4 Module software requirements for applicative peripherals 24

4.1 Analog to Digital Converter (ADC) 24

4.1.1 Self test algorithms 24

4.1.2 Analog watchdog 26

5 Summary 27

Appendix A CPU core initialization 28

A.1 CPU register initiliazation 28

A.2 Example of SPC56EL60 boot file for Flash 29

Appendix B Additional information. 34

B.1 Reference documents 34

B.2 Acronyms 34

Revision history 36

List of tables

Table 1. Device summary . . . 1
Table 2. Fault assertion conditions . . . 22
Table 3. SPC56EL60 registers to initialize . . . 29
Table 4. Acronyms . . . 34
Table 5. Document revision history . . . 36

List of figures

Figure 1. Initialization flow . . . 8
Figure 2. Safety initialization flow . . . 9
Figure 3. Faults check flow . . . 11
Figure 4. FCCU configuration flow . . . 14
Figure 5. SPC56EL60 system clock generation . . . 16
Figure 6. Clock configuration flow . . . 17
Figure 7. Built-in self test flow . . . 22
Figure 8. PMU power-on self test flow . . . 23
Figure 9. ADC self test in CPU mode using one shot sequence . . . 25
Figure 10. SPC56EL60: checking flow . . . 28

1 Document hierarchy

The Safety Application Guide (SAG) (please refer to *AN3077*, see *B.1: Reference documents* in *Appendix B*) is the reference document to use.

This application note is focused to describe the individual software measures.

The SAG describes which measure to apply according to the application and peripheral usage.

The hints that are described in this document should be considered as proposals to implement the requirements described in SPC56EL60 SAG. Based on their applications and the SAG, user can decide to use different implementations.

2 How to implement power-on self test features

The goal of this application note is to show how users can implement properly the safety initialization and the self tests to allow to detect latent fault[a] and to manage them.

2.1 MCU initialization

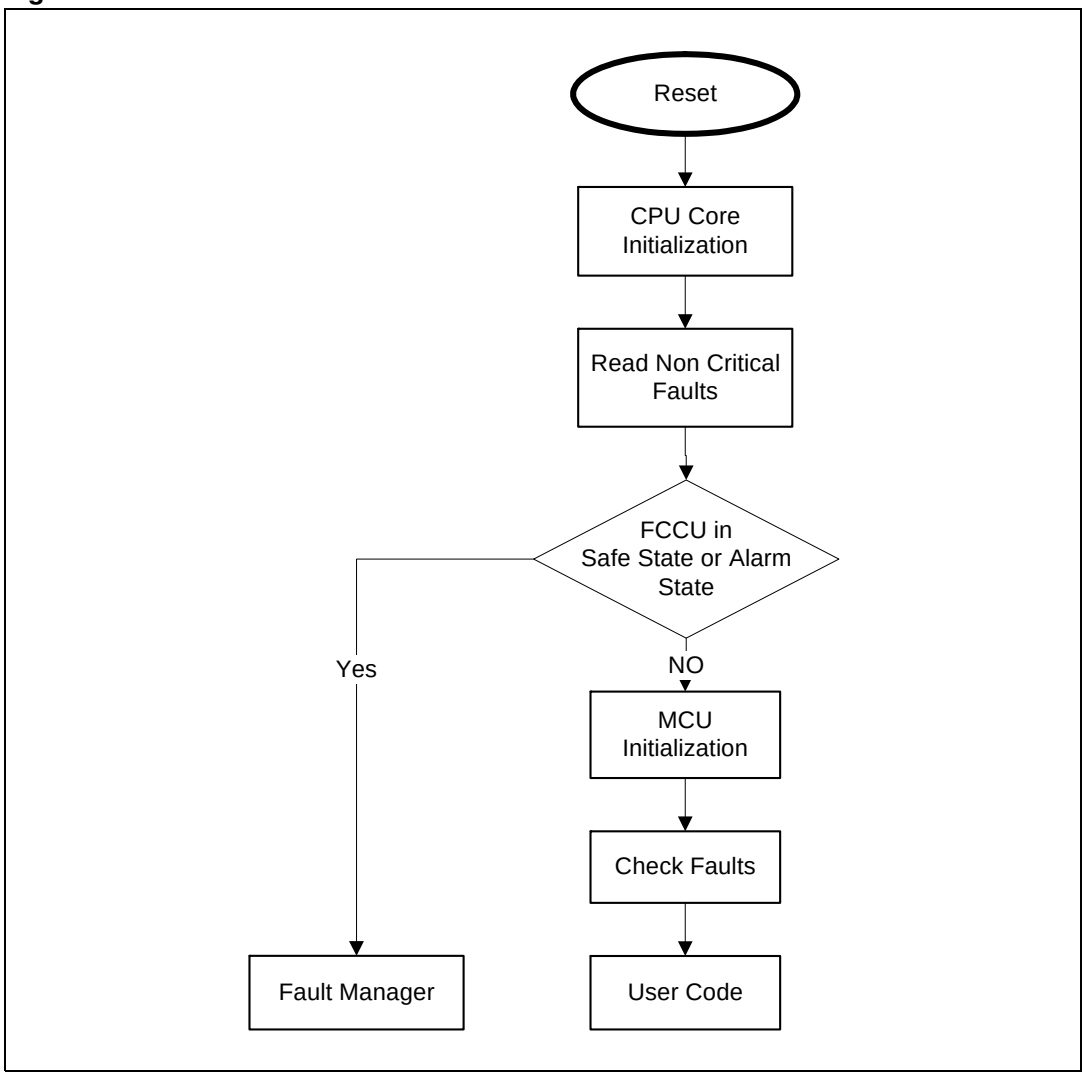
At power-on, after register initialization (see *Section 3.3: Redundancy Control Checker Unit (RCCU)*) and other basic initializations (MMU configuration, stack initialization, etc.) (see *Appendix A: CPU core initialization*) user software has to verify if MCU is in alarm state or in safe mode (coming from a Reset Condition) (see *Section B.1: Reference documents* in *Appendix B*) and in that case must manage fault causes.

If current mode in Mode Entry module is default run mode (DRUN), software can proceed with the default safety MCU initialization with self test features (see *Figure 1: Initialization flow*).

Note: User can verify alarm state by reading Non Critical Fault on FCCU while he can verify safe mode by reading Current Mode field (GS register) on Mode Entry module.

a. Latent fault: multiple point fault whose presence is not detected by a safety mechanism nor perceived by the driver within the multiple point fault detection interval.

Figure 1. Initialization flow



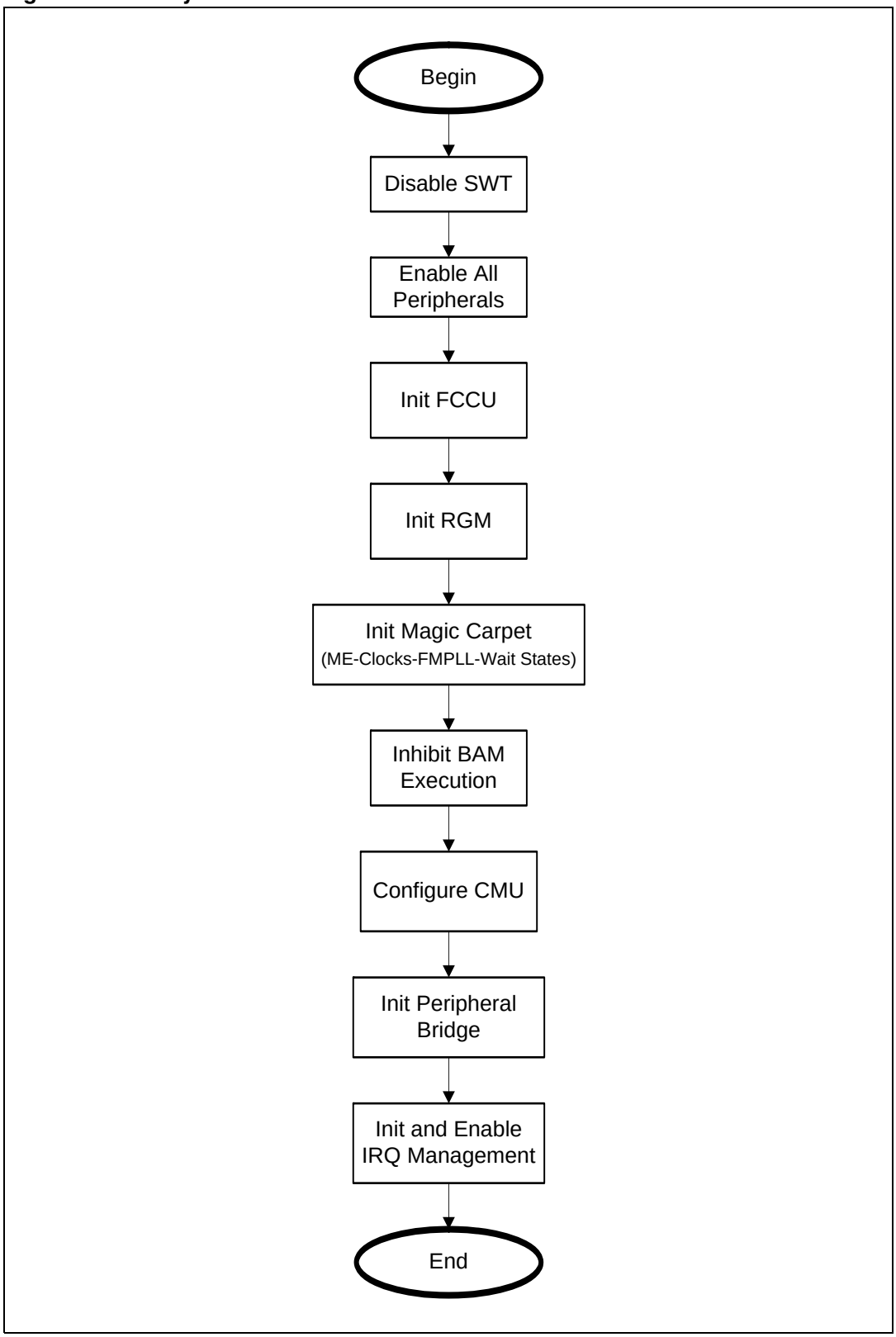
2.1.1 Safety initialization

Figure 2 shows an example of how to implement a safety initialization (see *Section 3: Module software requirements for non applicative peripherals*).

User should take care that:

1. Execution order is not mandatory but it is strongly recommended (see *Figure 2: Safety initialization flow*).
2. SWT (Software Watchdog Timer) is enabled.
3. RGM (Re set Generation Module) and FCCU (Fault Collection and Control Unit) must be configured before all monitors or detectors are initialized.

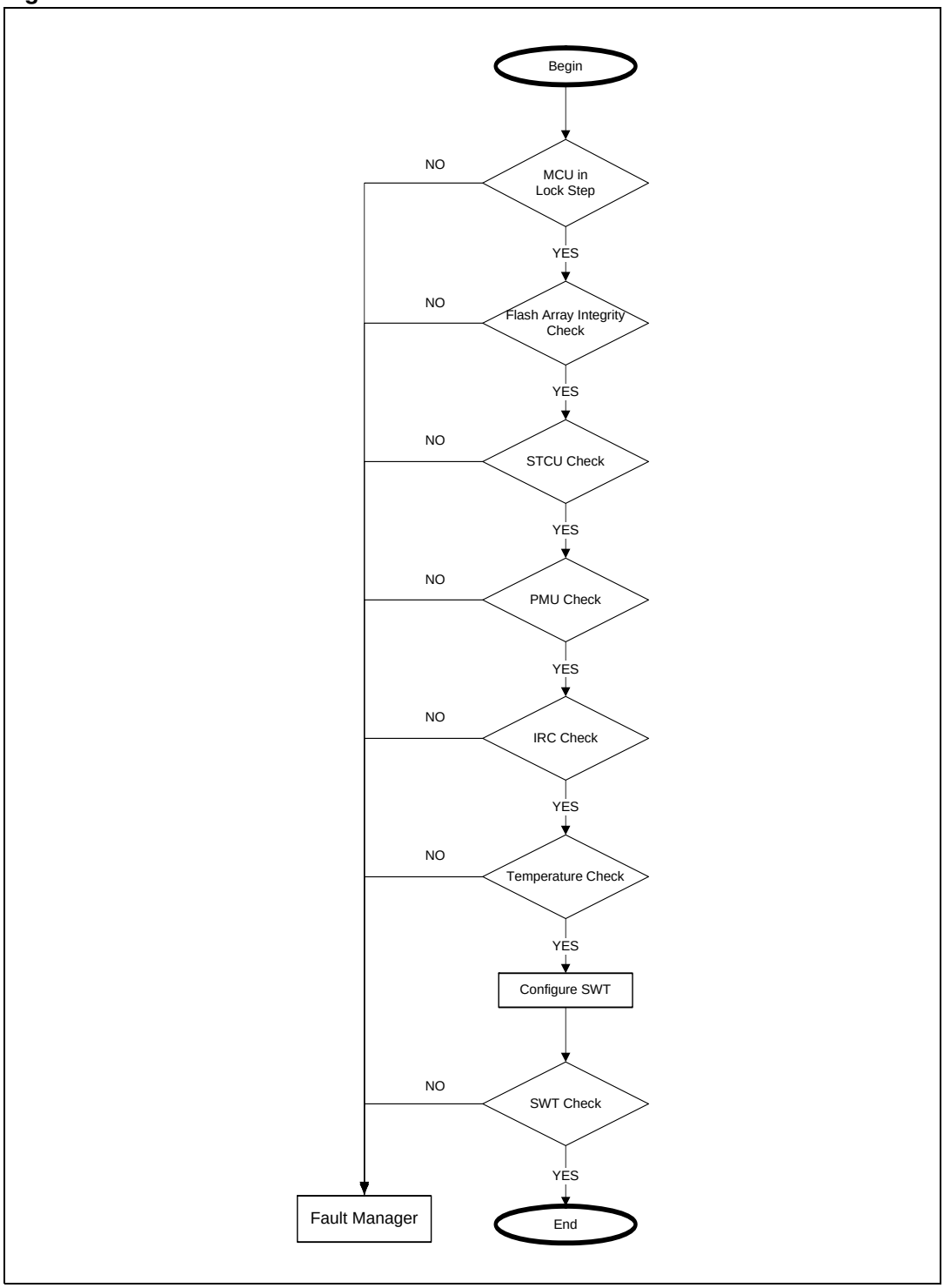
Figure 2. Safety initialization flow



2.2 Safety verification and faults checking

At the end of safety initialization, user software has to verify some basic safety requirements and verify if there is any fault (see *Section 3: Module software requirements for non applicative peripherals*). *Figure 3* shows an example of how to implement the faults check flow.

Figure 3. Faults check flow



3 Module software requirements for non applicative peripherals

This chapter describes the requirements of the software modules that should check the system peripherals and the Flash. The checks are required for any application.

The peripherals treated in this chapter are accounted as non applicable peripherals because they are not involved directly in any application Safety Integrity Function (SIF) please refer to *AN3077* (see *B.1: Reference documents* in *Appendix B*).

3.1 System Status and Configuration Module (SSCM)

Before executing safety functions, user must perform two actions:

1. Configure the SSCM to inhibit unintentional execution of the BAM code.

Note: This requirement is satisfied by asserting the flag PAE in the ERROR register of the SSCM. Each access to the BAM memory area produces a Prefetch or Abort exception.

2. Verify that the device operates in Lock-Step Mode (LSM).

Note: Software needs to check this condition by reading the LSM flag in the System Status Register (SSCM_STATUS) and verifying that the device is operated in the intended mode of operation.

3.2 Self Test Control Unit (STCU)

After boot, user software must check the STCU to ensure its reliability. The software must perform several operations based on the STCU status conditions after the power-on self test. Even if no errors are reported, user software should confirm that the expected and actual values within the CRC (Cyclic Redundancy Check) and LBIST MISR registers do not indicate an error.
This software confirmation prevents a fault within the STCU itself incorrectly indicating that the self test passed.

In the case of no reported errors, user software should confirm that:

1. The internal CRC computation result matches the expected value.

Note: Read the CRCE and CRCR registers to check the coherency with the STCU_ERR[CRCS] flag.

2. The signature registers of each of the LBIST results match their corresponding expected values.

Note: For each LBIST, read the STCU_LBMISREL/H and STCU_LBIST_NMISRRL/H registers to check the coherency with the STCU_LBS bits.

3. Read the registers used for Reported Errors and verify that their values are as expected. Refer to the “Integrity SW operations” section in RM0032 *(see B.1: Reference documents in Appendix B)*.

Note: Verify that STCU_LBS, STCU_LBE, STCU_MBSL, STCU_MBEL flag registers values are as expected. (LBIST and MBIST finished with success).

This is an additional Safety Layer since the STCU propagates the L/MBIST and internal faults using the CF/NCF lines toward FCCU. So reading the above registers helps in increasing the STCU auto-test coverage.

3.3 Redundancy Control Checker Unit (RCCU)

The RCCU unit performs a cycle-by-cycle comparison of the outputs of the modules included in the sphere of replication (SoR)[b].

The RCCU is able to detect any mismatch between the outputs of two replicated modules. The error information is forwarded to the Reset Generation Module (RGM) (see *Section 3.4: Reset Generation Module (MC_RGM)*) and to the Fault Collection and Control Unit (FCCU) (see *Section 3.5: Fault Collection and Control Unit (FCCU)*).

Fault injection for the RCCUs controlled by the FCCU is provided primarily for software development and validation purposes.
The RCCU's are only enabled when SPC56EL60 is in the LSM mode.

Note: Application software must assert that the LSM mode is activated (see Section 3.1: System Status and Configuration Module (SSCM)).

RCCU is automatically managed by the SPC56EL60 device and user cannot disable it.

3.4 Reset Generation Module (MC_RGM)

User has to configure MC_RGM and FCCU (Fault Collection and Control Unit) (see *Section 3.5: Fault Collection and Control Unit (FCCU)*) to react to critical application faults.

User has to trigger reset sequence when one of the following events happen:

- a CMU1/2 clock freq. too high/low event
- a PLL1 fail event
- a system clock freq. too high/low event
- a oscillator freq. too low event
- a PLL0 fail event
- a core watchdog reset event

Note: RGM_FERD (Functional Event Reset Disable Register) set to zero so that all events trigger a reset sequence. User should set to zero RGM_FEAR (Functional Event Alternate Request Register) to allow to generate a SAFE mode request on events if the reset is disabled.

3.5 Fault Collection and Control Unit (FCCU)

User has to configure the FCCU so that it reacts to generate a functional reset or so that it forces the MCU to switch in fail safe.

It is possible to configure the reaction for each fault source and ensuring the rule described above is valid for each individual source.

b. The SoR is the logical part of the device that contains all the modules that are replicated for functional safety reasons.

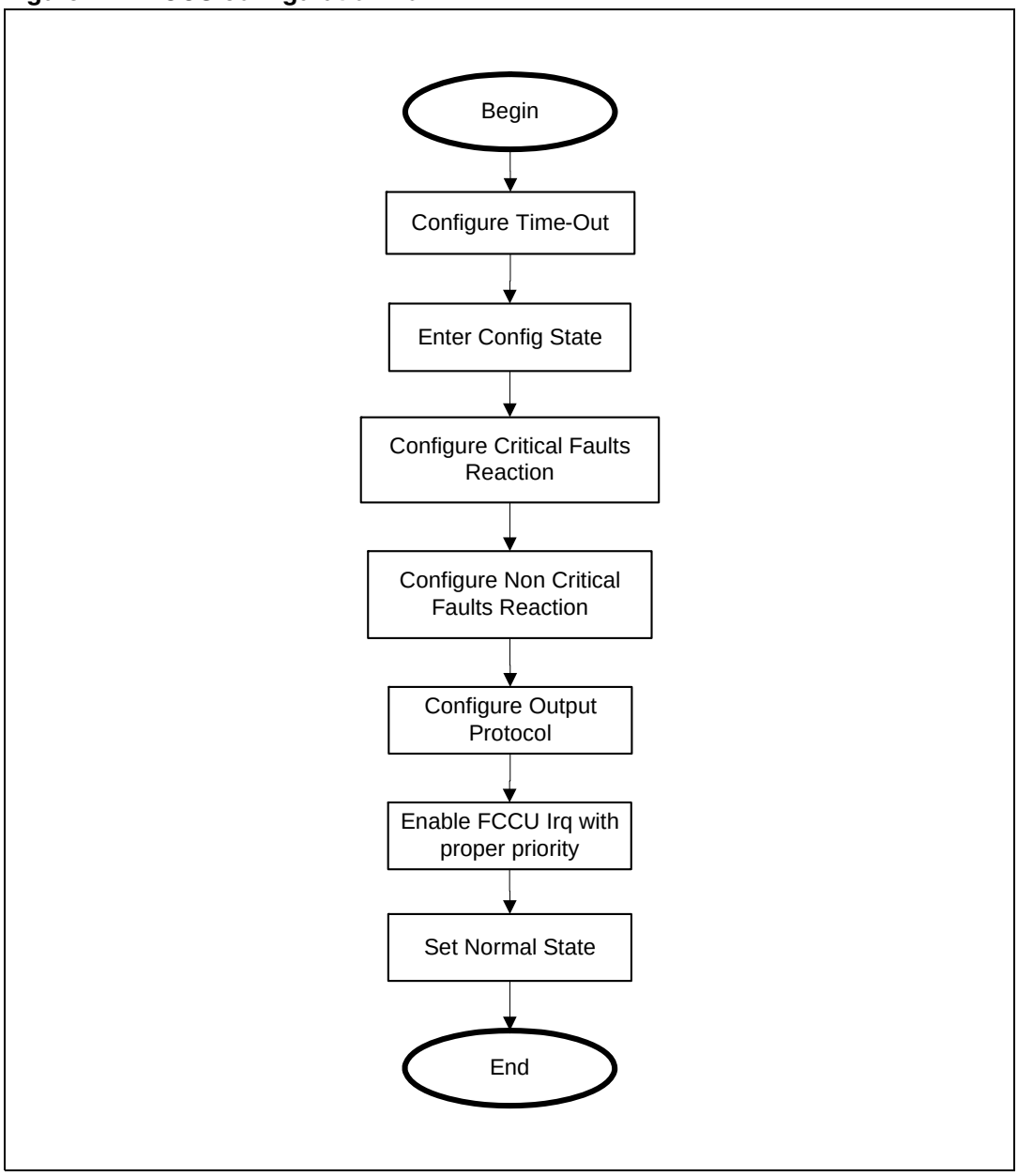
The only exception to this rule is when the CMU monitors a PLL that is not used or is used for non safety critical modules only.

In this case, error masking and limited internal reaction can be tolerated. External reaction of the FCCU is always enabled and it can not be disabled.

User has to verify that no Critical Fault (CF) or Non Critical Fault (NCF) are present after the boot.

Note: The application has to configure the FCCU to enable all reactions related to faults of peripherals used by the application safety function (see Figure 4: FCCU configuration flow).

Figure 4. FCCU configuration flow



3.6 Clock configuration

The system starts up using the internal RC oscillator clock as clock source. Refer to the “IRCOSC 16 MHz Internal RC Oscillator” section in RM0032 *(see B.1: Reference documents in Appendix B)* for detailed informations about the internal oscillator.

User has to configure the FMPLLs to use the clock signal from the external oscillator (XOSC) as a clock source before any safety functions are executed.

Note: SELECT field in the CGM_AC3_SC and CGM_AC4_SC have to be set to 1. See Figure 5: SPC56EL60 system clock generation.

All safety-relevant IP modules are clocked with a FMPLL-generated clock signal to reduce the impact of glitches stemming from the external quartz crystal and its connection to the MCU.

Note: This requirement is fulfilled by appropriately programming the MC_CGM and MC_ME modules.

See *Figure 6: Clock configuration flow* to see what is the correct way to configure Clock and FMPLLs.

Figure 5. SPC56EL60 system clock generation

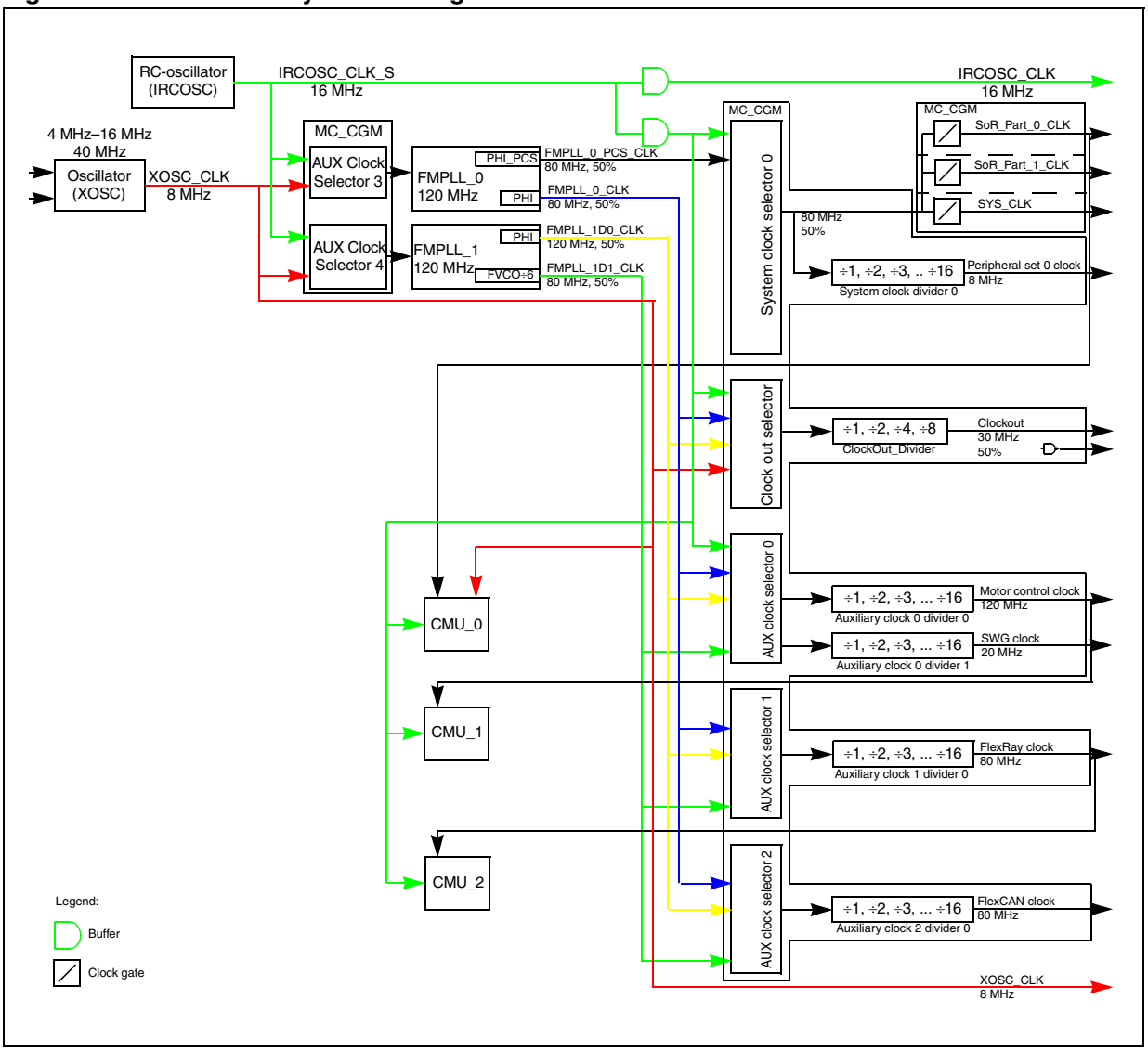
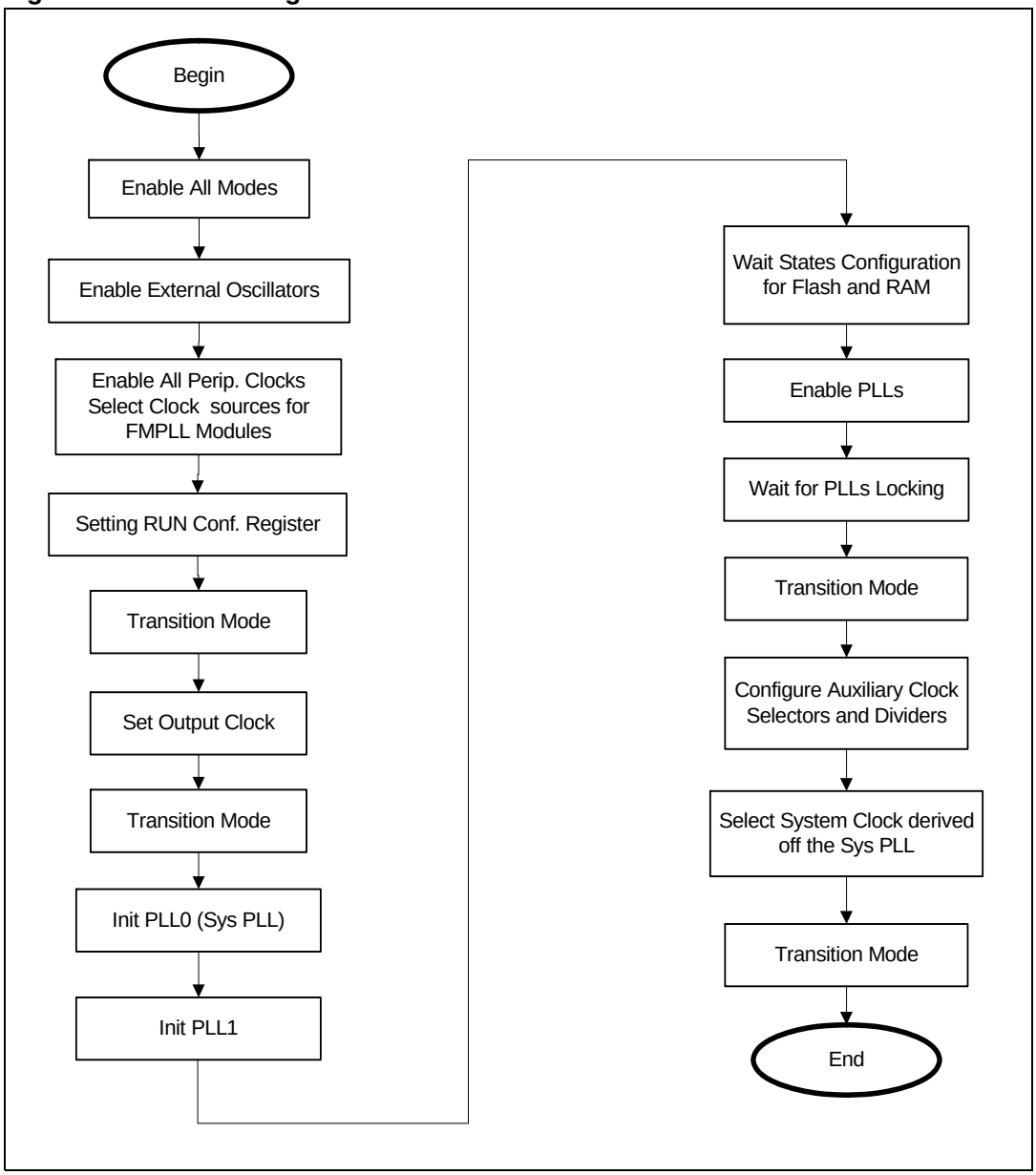


Figure 6. Clock configuration flow(c)



c. Flash and SRAM wait states depend on MCU work frequency. For this reason it's a good choice to configure these together with clocks.

3.7 Clock Monitor Unit (CMU)

The main task of the Clock Monitor Unit (CMU) is to supervise the integrity of various clock sources.

User software has to manage the following conditions:

- Loss of external crystal oscillator clock
- FMPLL frequency higher than a (programmable) value set as high reference
- FMPLL frequency lower than a (programmable) value set as low reference

SPC56EL60 includes three CMUs:

a) CMU_0 to monitor the clock signal of the Sphere of Replication (SoR) and the clock from the crystal oscillator

b) CMU_1 to monitor the clock signal used by the Motor Control related peripherals (eTimer, FlexPWM, CTU, and ADC)

c) CMU_2 to monitor the clock signal for the protocol engine of the FlexRay module

Use of the CMU is mandatory: if the related modules are used by the application safety function, the user must verify that the CMUs are not disabled and their faults managed by the FCCU.

Note: In general, the following two application-dependent configurations must be executed before CMU monitoring can be enabled.

1 *Crystal oscillator clock monitor (only CMU_0): the software must configure the RCDIV field of the Control Status register (CMU_0_CSR) with a value related to the external oscillator frequency.*

2 *Clock signal monitors (CMU_1, CMU_2): the CMU_x_HFREFR and CMU_x_LFREFR registers must be configured depending on the SoR, Motor Control, and FlexRay clock frequencies.*

To later enable the CMUs, the flag CME in the respective Control Status Register (CMU_x_CSR) must be asserted.

3.8 Frequency-Modulated Phase-Locked Loop (FMPLL)

Application software has to check that the system uses the system FMPLL clock as system clock before running any safety element function.

Note: Application software can verify the current system clock by checking the S_SYSCLK flag of the ME_GS register. S_SYSCLK equal to 0x4 indicates that the system FMPLL clock is used as system clock.

Each FMPLL provides a loss of lock error indication that is routed to the RGM (see *Section 3.4: Reset Generation Module (MC_RGM)*) and the FCCU (see *Section 3.5: Fault Collection and Control Unit (FCCU)*).

The application software has to enable the respective fault and configure the FCCU to manage it. Since in case of fault the system clock can be driven by the internal RC oscillator (see *Section 3.9: Internal RC Oscillator (IRCOSC)*), the fault of the FMPLL is considered as Non-Critical Fault.

Note: The pll_fail output is not masked (pll_fail_mask flag in the FMPLL_x Control Register (CR) deasserted). To enable the RGM input related to FMPLL loss of clock, the registers RGM_FERD and RGM_FEAR must be configured. To enable the FCCU fault path some

registers have to be configure (NCF_CFG0, NCFS_CFG0, NCF_TOE0, etc.). Loss of lock signals from FMPLL_0 and FMPLL_1 provide the FCCU "Non-Critical Fault" inputs 2 and 3.

Note: The RGM and FCCU configuration includes the reaction in case of FMPLL loss of lock. This reaction is application-dependent.

3.9 Internal RC Oscillator (IRCOSC)

User has to verify the availability and frequency of the internal RC oscillator and for this reason the frequency meter of the CMU0 (see *Section 3.7: Clock Monitor Unit (CMU)*) must be exploited.

User has to use this feature to measure the RC oscillator frequency using the external oscillator clock as known one and compare it to the expected one (16MHz[d]).

Note: The reference clock is always the XOSC. The measure starts when CMU_CSR[SFM] is set. The measurement duration is given by the CMU_MDR register in terms of IRC clock cycles with a width of 20 bits. The SFM bit is cleared by the hardware after the frequency measurement is done and the count is loaded in the CMU_FDR.
The frequency FRC can be derived from the value loaded in the CMU_FDR register (FRC = (FOSC × MD) / n) where n is the value in CMU_FDR register and MD is the value in CMU_MDR.

Note: Safety related modules which work on the RC clock are: FCCU and SWT. In case of RC clock failure, these modules stop working.

3.10 Flash memory

To support the detection of dormant faults in the entire memory array and addressing logic, user must execute array integrity self check (see *Section 3.10.1: Array integrity self check procedure*).

This BIST is based on functionality built into the Flash control logic. It calculates a MISR signature over the array content and thus validates the content of the array as well as the decoder logic. The calculated MISR value is dependent on the array content and must be validated by application software.

For self check during boot, user has to do the array integrity check over the entire area except over the sectors used for EEPROM (Electrically Erasable Programmable Read Only Memory) emulation as only one of the sectors used for EEPROM contains valid data and that the data in this sector varies during ECU (Electronic Control Unit) life time.

Note: This BIST must be started by application software; its result must be validated by reading the corresponding registers in the Flash Controller after it has been finished. Refer to the "Array integrity self check" section in the "Flash memory" chapter of RM0032 (see B.1: Reference documents in Appendix B) for detailed informations about this BIST.

d. The internal RC oscillator nominal frequency is 16 MHz, but a post trim accuracy of ±6% overvoltage and temperature must be taken into account.

3.10.1 Array integrity self check procedure

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 operation is completed, the results of the reads can be checked by reading the MISR value, to determine if an incorrect read or ECC (Error Correction Code) detection was noted. Array integrity is controlled by the system clock, and it is required that the Read Wait States and Address Pipelined control registers in the BIU (Bus Interface Unit) be set to match the user defined frequency being used.

Caution: While array integrity is being executed, Flash memory array access 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.

Caution: Locked blocks can be tested with array integrity if selected in LMS and HBS. 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.

Caution: 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) completely, it is recommended that AIS be left at 0, to use the address sequence that checks the read path fully, and examine read transitions. This sequence takes more time.

4. Seed the MISR UM0 through UM4 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 with the next step. It should be noted that in the event of an aborted array integrity check the MISR registers contains a signature for the portion of the operation that was completed prior to the abort, and it is not deterministic. Prior to doing another array integrity operation, the UM0, UM1, UM2 and UM3 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 (UM0 through UM4) to ensure correct signature.
8. Write a logic 0 to the UT0[AIE] bit.

If the Flash contents change, the MISRs are different: for this reason user should store MISR value (used by self check at boot) in a separate Flash block that is unselected during MISR calculation.

Note: This test does have some coverage on unselected sectors like sector_eeprom, since the array integrity check always goes through the entire Flash array (but only accumulate MISRs for selected sectors). Therefore, the array integrity check signals any ECC error in unselected sectors.

3.11 Temperature sensors

There are two temperature sensors: temperature sensor 0 mapped to ADC_0 channel 15 and temperature sensor 1 mapped to ADC_1 channel 15.

During power up, user software has to read temperature sensors and verify that the values are similar as means of assessing the functionality of the sensors.

Note: In case of a fault, software must trigger the appropriate reaction. To set a proper threshold the customer must consider that the maximum operating junction temperature is 150 °C, for details please refer to SPC56EL60 datasheet (see B.1: Reference documents in Appendix B) and the temperature sensor accuracy is 10 °C.
It is mandatory to read the 2 sensors synchronously or with a reduced time interval.

Note: It is important to note that the ADC is part of the temperature measuring safety integrity function, and it is therefore required that the BIST of the ADC must be executed once after the boot even if the ADC is not in application use.

3.12 Software Watchdog Timer (SWT)

The SWT has to be enabled and configuration registers have to be hard-locked against manipulation.

The time window settings of the SWT have to be set to a value less than the PST[e] (process safety time). Detection latency is smaller than process safety time.

Before the safety function is executed, the software must verify that the SWT is enabled by reading the SWT control register (SWT_CR).

Note: To enable the SWT and to hard-lock the configuration register, the WEN and HLK flags of the SWT control register (SWT_CR) must be asserted. The timeout register (SWT_TO) must contain a 32-bit value that represents a timeout less than the process safety time.

Caution: SWT must be refreshed with a timeline that depends by setted timeout (SWT_TO value). For this reason it should be configured properly when MCU execute uninterruptible tasks (see *Section 3.10.1: Array integrity self check procedure*) to avoid application reset.

e. PST is strictly application dependant.

3.13 Power Management Unit (PMU)

Software has to run core voltage LVD and HVD hardware-assisted self-test after the boot.

The PMU provides (to the user) the software capability to check the run of the BIST procedure, generating non-critical faults (NCFs) or critical faults (CF) conditions for the FCCU module (see *Table 2: Fault assertion conditions*).

Table 2. Fault assertion conditions

Fault number	Signal
NCF[13]	LVD BIST ok in test mode
NCF[14]	HVD BIST ok in test mode
CF[21]	LVD/HVD BIST failure result in test mode

At each power-on, the self-test circuitry is able to detect a failure of one of the two voltage detectors and to provide a non critical fault (NCF) to the FCCU (see *Section 3.5: Fault Collection and Control Unit (FCCU)*).

Note: The hardware-assisted self-tests are initiated through the SIL fields in the PMU Control Register (PMUCTLR_CTRL). If the self-test passes, a Non-Critical Fault is triggered (see Figure 8: PMU power-on self test flow). If the self-test fails, a PMUCTRL_IRQS and Critical Fault are asserted.
The BIST execution is controlled by the PMUCTRL_CTRL[SILHT] field (see Figure 7: Built-in self test flow).

Figure 7. Built-in self test flow

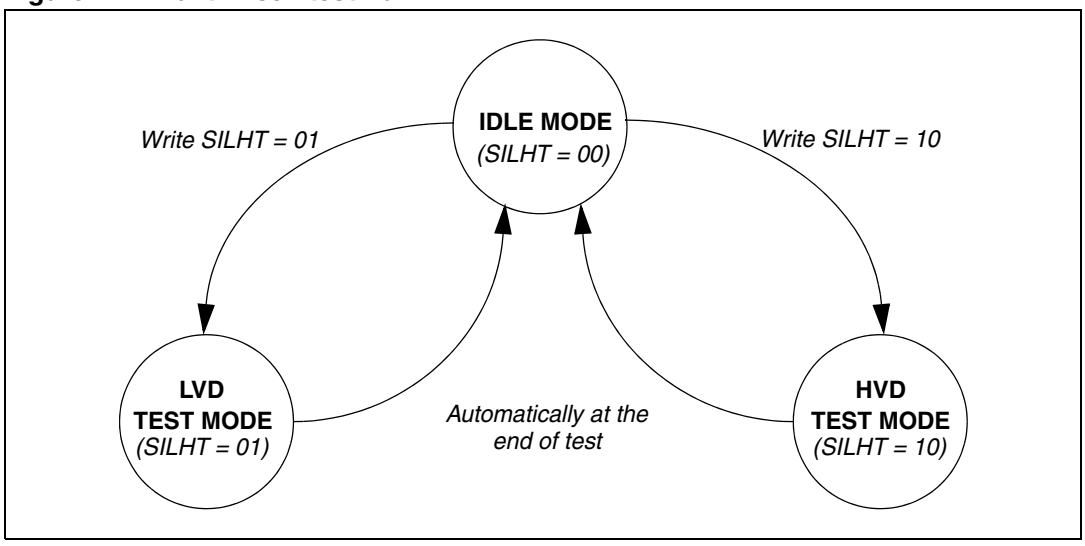
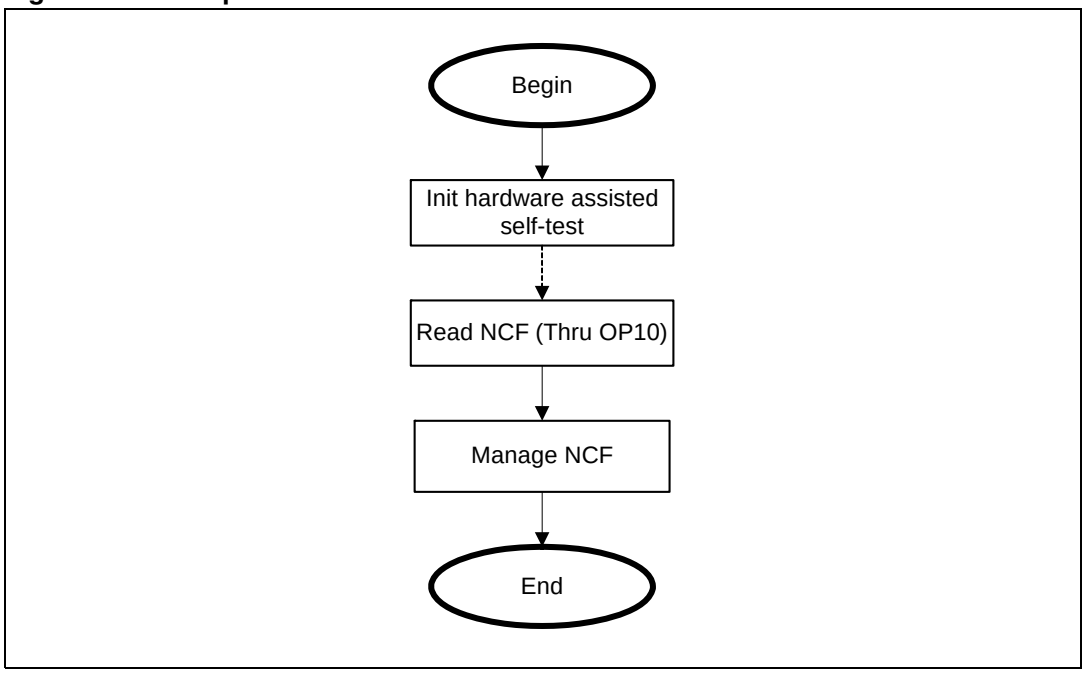


Figure 8. PMU power-on self test flow



4 Module software requirements for applicative peripherals

The ADC, due to its analog part, is the only application peripheral that requires additional hardware bist after the boot.

This module is part of the temperature measuring safety integrity function, and it is therefore required that the HWBIST functions are executed once after the boot (please refer to *AN3077*, see *B.1: Reference documents* in *Appendix B*).

4.1 Analog to Digital Converter (ADC)

After boot and before starting execution of any safety function, user has to execute ADC HWBIST functions to check if the ADC is functioning correctly and to increase the diagnostic coverage.

4.1.1 Self test algorithms

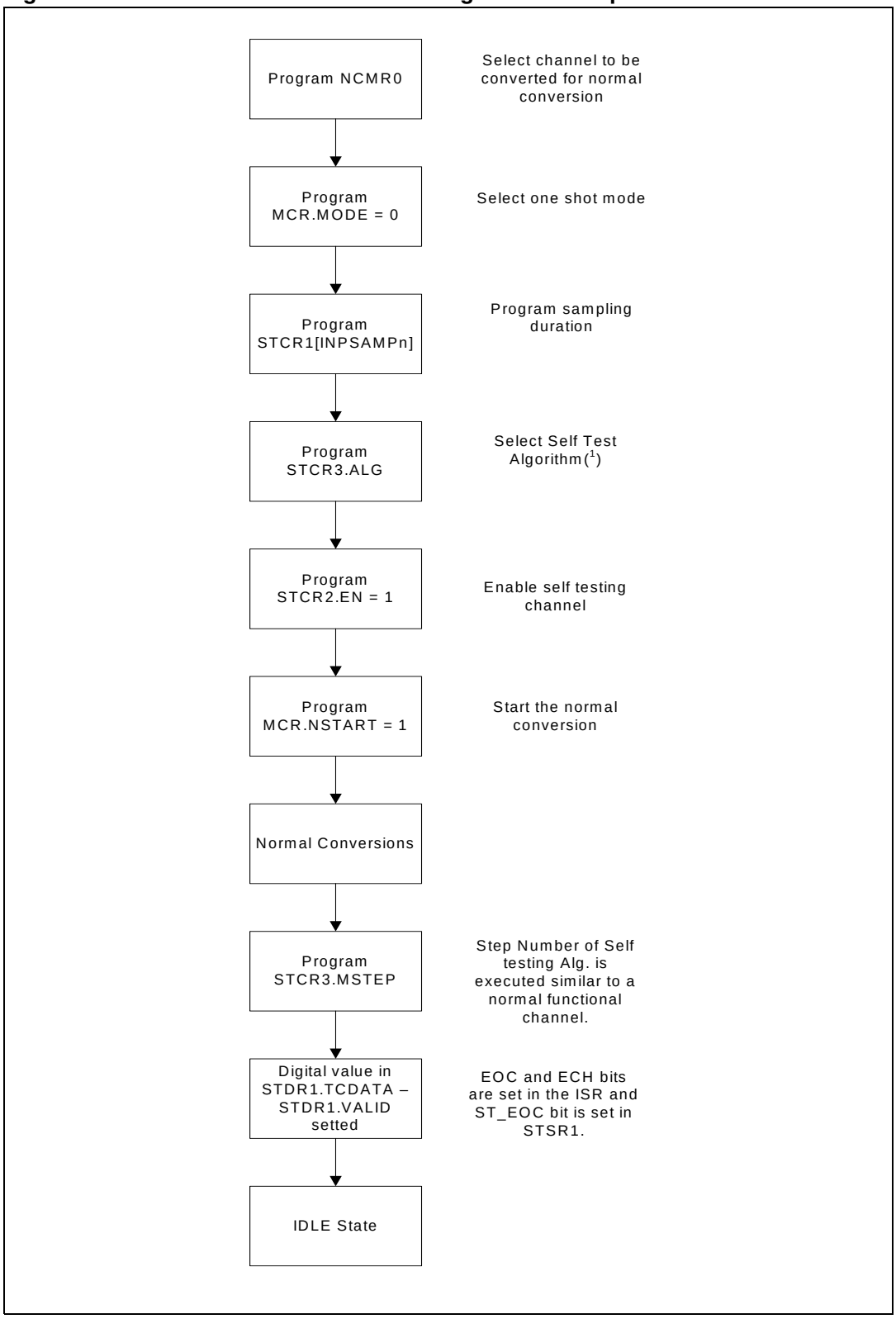
Three types of self testing algorithms have been implemented inside ADC analog.

- Supply self test: Algorithm S.
  It includes the conversion of the ADC internal bandgap voltage, ADC supply voltage, and ADC reference voltage. It includes a sequence of 3 test conversions (steps). The supply test conversions must be an atomic operation (no functional conversions interleaved).
- Resistive-Capacitive self test: Algorithm RC.
  It includes a sequence of 19 test conversions (steps) by setting the ADC internal resistive digital-to-analog converter (DAC).
- Capacitive self test: Algorithm C.
  It includes a sequence of 17 test conversions (steps) by setting the capacitive elements comprising the sampling capacitor/ capacitive DAC.

The ADC implements an additional test channel dedicated for self testing. It also provides signals to schedule self testing algorithms using configuration registers, monitors the converted data using analog watchdog registers, flags the error to FCCU in case some failure occurs in any of the algorithms.

Note: User can execute these tests in CPU Mode (One shot or Scan Mode) and in CTU Mode (see Figure 9: ADC self test in CPU mode using one shot sequence).

Figure 9. ADC self test in CPU mode using one shot sequence



1. In one shot mode, if test channel is enabled, only one step of selected self testing algorithm is executed at the end of the chain. The step number and algorithm to be executed is programmed in STCR3 register.

4.1.2 Analog watchdog

The ADC provides a monitor (watchdog) for the values returned by its analog portion for Self Test algorithms. The analog watchdogs are used to determine whether the result of conversion for self test algorithms lie in a particular guard area. For this purpose, separate Self test analog watchdog registers have been provided for each algorithm.

Note: After the conversion of each step of an algorithm, a comparison is performed between the converted value and the threshold values if Analog watchdog feature is enabled by setting STAWxR.AWDE bit. If the converted value does not lie between the upper and lower threshold values specified by Analog Watchdog Register of the particular algorithm, corresponding error bit STSR1.ERR_x is set and Step Number in which error occurred is updated in STSR1.STEP_x (in case of C or RC algorithm). Also, erroneous data is written in STSR4.DATAx field. The STSR1.ERR_x bits generates an interrupt if enabled by corresponding Mask bit in STCR2 register. The fault indication is also given to FCCU via CF and NCF, so that necessary action can be taken.

Note: Before running the hardware self test, the customer must copy the threshold values of the analog watchdogs from Test Flash into the watchdog registers (STAWxR). Refer to the "Analog-to-Digital Converter (ADC)" section in RM0032 (see B.1: Reference documents in Appendix B) for detailed informations about watchdog registers and threshold values.

5 Summary

The described Power On Self Test procedure allow application software to covers all requirements indicated by Safety Application Guide (please refer to *AN3077*, see *B.1: Reference documents* in *Appendix B*) as mandatory for power on boot.

User should take care that these check are valid only one time and not cover all safety measures that user must implement to have SIL3 applications.

Furthermore, a reference example has been implemented.

Appendix A CPU core initialization

User, before MCU initialization must initialize CPU core and prepare the right environment on what user code is executed.

The booting phase include (see *Section A.2: Example of SPC56EL60 boot file for Flash*):

- Define reset vector
- Initialize CPU registers (see *Section A.1: CPU register initiliazation*)
- MMU programming
- Initialize stack and small data section pointers

A.1 CPU register initiliazation

After SPC56ELx reset, core registers must be initialized to a known value to avoid unexpected fault triggered.

SPC56ELx cores (in Lock Step configuration) are linked to a checker that verify their alignment: output of Core_0 is compared to output of Core_1 through RCCU (see *Section 3.3: Redundancy Control Checker Unit (RCCU)*): if a difference is found the error is forwarded (see *Figure 10: SPC56EL60: checking flow*) to FCCU (see *Section 3.5: Fault Collection and Control Unit (FCCU)*) and to RGM (see *Section 3.4: Reset Generation Module (MC_RGM)*).

Figure 10. SPC56EL60: checking flow

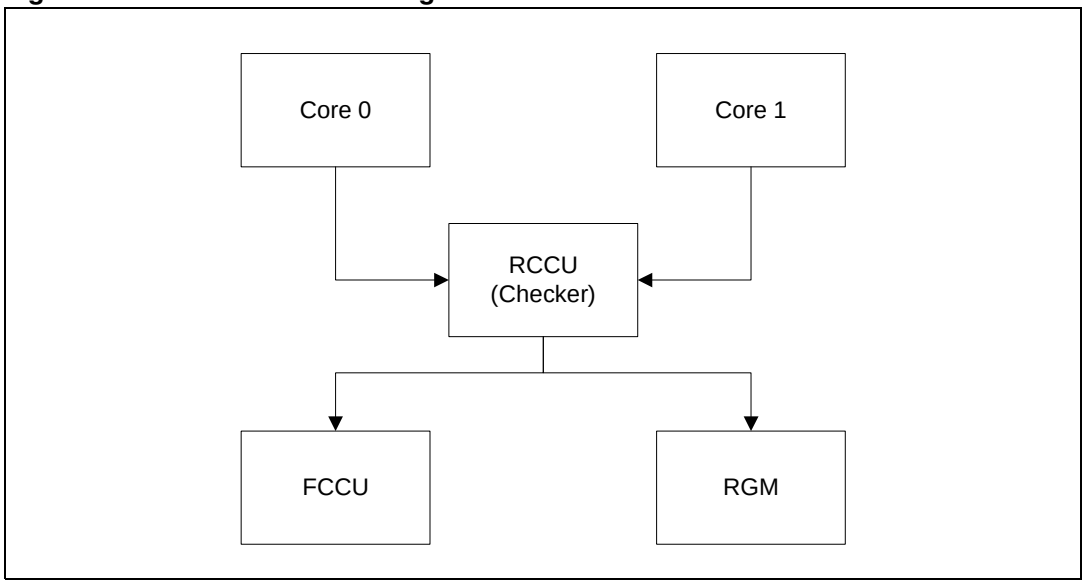


Table 3. SPC56EL60 registers to initialize

Register ranges
r0 - r31
spr1
spr8-spr9
spr22
spr26 - spr27
spr54
spr58 - spr59
spr61-spr63
spr256
spr272 - spr279
spr284 - spr285
spr318 - spr319
spr340
spr400-spr415
spr512
spr528-spr530
spr562
spr570 - spr571
spr573 - spr575
spr604 - spr605
spr628
spr630
ACC
CR

A.2 Example of SPC56EL60 boot file for Flash

In boot file user has to provide a reset vector in order to run the application from Flash memory. The standard startup process of SPC56ELX processor consists in execution of the special boot code which reads specific addresses in a Flash memory where reset configuration half word is stored together with boot reset vector pointing to first valid instruction of the code. Refer to “Boot Assist Module (BAM)” section in *RM0032 (see B.1: Reference documents in Appendix B)*.

In this boot sequence user must initialize SRAM and copy data section from Flash to SRAM (this operation is made only when user runs code from Flash). As it may see in scratch code below stated, this copy is split into two steps and between them it is inserted software watchdog refresh to avoid device reset.

Finally user must set stack and small data area pointers.

```
;------- Reset Configuration Half Word ----------------------------------

   .if __ghs__
   .section .rcw
   .else
   .section        .rcw,,c
   .endif

reset_vector:
    .LONG 0x015A0000       # SPC56ELX - e200 Core Watchdog OFF, External Boot OFF,
VLE ON

    .LONG _start            # Code starts at _start
;--------------------------------------------------------------------------
   .if __ghs__
   ; GHS declarations
   .section .vletext_init, va
   .vle
   .global__ghs_board_memory_init

   .else
   ; WindRiver declarations
   .section .vletext_init,4,c

   .endif

   .global_start
   .align2
   e_add2i.r0,0# Debuggers may object to starting at 0.

_start:

##--------- SET UP MMU (BEGIN)-----------------------------------------
                    #table 0
   e_lis r5, 0x10000000@ha
   e_add16i r5, r5, 0x10000000@l
    mtspr mas0,r5            # mtspr MAS0,r5
   e_lis r5, 0xC0000500@ha
   e_add16i r5, r5, 0xC0000500@l
    mtspr mas1,r5# mtspr MAS1,r5
   e_lis r5, 0x00000028@ha
   e_add16i r5, r5, 0x00000028@l
    mtspr mas2,r5            # mtspr MAS2,r5
   e_lis r5, 0x0000003F@ha
   e_add16i r5, r5, 0x0000003F@l
    mtspr mas3,r5            # mtspr MAS3,r5
    tlbwe                    # Write the entry to the TLB

                    #table 1
....
    mtspr mas3,r5            # mtspr MAS3,r5
    tlbwe                    # Write the entry to the TLB

                    #table 2
....
    tlbwe                   # Write the entry to the TLB

                    #table 3
....
    mtspr mas3,r5            # mtspr MAS3,r5
```

```
    tlbwe                    # Write the entry to the TLB

                    #table 4
....
    mtspr mas3,r5            # mtspr MAS3,r5
    tlbwe                    # Write the entry to the TLB

                    #table 5
....
    tlbwe                    # Write the entry to the TLB

    se_isync

##--------- SET UP MMU (END)-----------------------------------------

##---------  Initialise Registers --------------------

   e_li r1, 5
....
   e_li r31, 5

# reset of selected registers
    mtcrf 0xFF,r31
    mtspr 285,r31 #TBU
    .... mtspr 603,r31
##---------  end Initialise Registers ---------------------


##------- Initialise SRAM ECC ---------------------------------------------
# Doing this in halves for 128k SRAM to allow for WDOG service at
# the half-way point.

# Base Address of the internal SRAM
    e_lis       r5, _SRAM_BASE_ADDR@h
    e_or2i      r5, _SRAM_BASE_ADDR@l

# Store number of 128Byte (32GPRs) segments in Counter
    e_lis       r6, _SRAM_SIZE@h    # Initialize r6 to size of SRAM (Bytes)
    e_or2i      r6, _SRAM_SIZE@l

    e_srwi      r6, r6, 0x3         # Divide SRAM size by 8 (half SRAM size in words)
    mtctr       r6                  # Move to counter for use with "bdnz"

# Fill SRAM with known values not registers ################
# Never write content of uninitialised registers to SRAM ####
sram_loop1:
    e_lis       r0,0x0
    e_stw       r0,0x0(r5)          # Write all 32 registers to SRAM
    e_addi      r5,r5,4             # Increment the RAM pointer to next 128byte
    e_bdnz      sram_loop1          # Loop for all of SRAM

# Service the watchdog now (doing the entire SRAM init is too long)
   e_lis       r1,0xfff3
   e_or2i      r1,0x8010
   e_li        r2,0xA602# SR sequence value 1
   se_stw      r2,0x0(r1)
   e_li        r2,0xB480# SR sequence value 2
   se_stw      r2,0x0(r1)

# Finish initializing SRAM
    mtctr       r6                  # r6 still contains half the SRAM size in words
sram_loop2:
    e_lis     r0,0x0
```

```
    e_stw       r0,0x0(r5)          # Write all 32 registers to SRAM
    e_addi      r5,r5,4             # Increment the RAM pointer to next 128byte
    e_bdnz      sram_loop2           # Loop for all of SRAM

##-------------------------------------------------------------------------

    e_lis       r1, __SP_INIT@h     # Initialize stack pointer r1 to
    e_or2i      r1, __SP_INIT@l     # value in linker command file.

    e_lis       r13, _SDA_BASE_@h   # Initialize r13 to sdata base
    e_or2i      r13, _SDA_BASE_@l  # (provided by linker).

    e_lis       r2, _SDA2_BASE_@h   # Initialize r2 to sdata2 base
    e_or2i      r2, _SDA2_BASE_@l   # (provided by linker).

    e_stwu      r0,-64(r1)          # Terminate stack.

##--------- Load Initialised Data Values from Flash into RAM ----------------
#--------- Load ".data" section value into RAM (un-initialised) -------------
# Set GPR9 to the count of the SRAM load size

   e_lis r9, __DATA_SIZE@h  # Load upper SRAM load size (# of bytes) into R9
   e_or2ir9, __DATA_SIZE@l # Load lower SRAM load size into R9

   e_cmp16ir9,0         # Compare to see if equal to 0
    e_beq ROMSDATACOPY            # Exit cfg_ROMCPY if size is zero

    mtctr  r9                     # Store # of bytes to be moved in spr CTR

    e_lis   r10, __DATA_ROM_ADDR@h  # Load address of first SRAM load into R10
    e_or2i r10, __DATA_ROM_ADDR@l  # Load lower address of SRAM load into R10
    e_subi  r10,r10, 1       # Decrement address to prepare for ROMCPYLOOP

# Load SRAM base address (__DATA_SRAM_ADDR) for loading instructions into R5
#    (__SRAM_CPY_START = ADDR(.data))
    e_lis   r5, __DATA_SRAM_ADDR@h  # Load upper SRAM address into R5
    e_or2i   r5, __DATA_SRAM_ADDR@l # Load lower SRAM address into R5
    e_subi  r5, r5, 1          # Decrement address to prepare for ROMCPYLOOP

ROMDATACPYLOOP:
    e_lbzu   r4, 1(r10) # Load data byte at R10 into R4,incrementing (update) ROM
address
    e_stbu   r4, 1(r5)  # Store R4 data byte into SRAM at R5 and update SRAM address
    e_bdnz   ROMDATACPYLOOP          # Branch if more bytes to load from ROM

##--------- Load ".sdata" section value into RAM (un-initialised) ------------------

ROMSDATACOPY:
   e_lis r9, __SDATA_SIZE@h  # Load upper SRAM load size (# of bytes) into R9
   e_or2ir9, __SDATA_SIZE@l # Load lower SRAM load size into R9

   e_cmp16ir9,0         # Compare to see if equal to 0
    e_beq ROMCPYEND               # Exit cfg_ROMCPY if size is zero

    mtctr  r9                     # Store # of bytes to be moved in spr CTR

    e_lis   r10, __SDATA_ROM_ADDR@h # Load address of first SRAM load into R10
    e_or2i r10, __SDATA_ROM_ADDR@l  # Load lower address of SRAM load into R10
    e_subi  r10,r10, 1# Decrement address to prepare for ROMCPYLOOP

# Load SRAM base address (__SDATA_SRAM_ADDR) for loading instructions into R5
#    (__SDATA_SRAM_ADDR = ADDR(.data))
    e_lis   r5, __SDATA_SRAM_ADDR@h  # Load upper SRAM address into R5
```

```
    e_or2i   r5, __SDATA_SRAM_ADDR@l  # Load lower SRAM address into R5
    e_subi  r5, r5, 1          # Decrement address to prepare for ROMCPYLOOP

ROMSDATACPYLOOP:
    e_lbzu   r4, 1(r10) # Load data byte at R10 into R4,incrementing (update) ROM
address
    e_stbu   r4, 1(r5)  # Store R4 data byte into SRAM at R5 and update SRAM address
    e_bdnz   ROMSDATACPYLOOP          # Branch if more bytes to load from ROM

ROMCPYEND:
##--------- end Load un-initialised values into RAM ---------------------
   .if __ghs__
   e_bl  main
   .else
   bl          __init_main
   .endif
```

Appendix B Additional information

B.1 Reference documents

- *Safety application guide for SPC56EL60* (AN3077, Doc ID 16384).
- *SPC56EL60 32-bit MCU family built on the embedded Power Architecture®* (RM0032, Doc ID 15265).
- *Getting started tutorial for SPC56EL60* (AN3121, Doc ID 16853).
- *32-bit Power Architecture® microcontroller for automotive SIL3/ASILD chassis and safety applications (*SPC56EL60L3, SPC56EL60L5 datasheet, Doc ID 15457).

B.2 Acronyms

Table 4. Acronyms

Acronym	Name
ADC	Analog to Digital Converter
BAM	Boot Assist Module
BIST	Built In Self Test
BIU	Bus Interface Unit
CF	Critical Fault
CMU	Clock Monitor Unit
CRC	Cyclic Redundancy Check
DMA	Direct memory access
ECC	Error Correction Code
ECU	Electronic COntrol Unit
EEPROM	Electrically Erasable Programmable Read Only Memory
FCCU	Fault Collection and Control Unit
FMPLL	Frequency Modulated Phase-Locked Loop
HVD	High Voltage Detector
INTC	Interrupt controller
IRC	Internal RC Oscillator
LVD	Low Voltage Detector
MCU	Microcontroller unit
NCF	Non Critical Fault
PMU	Power Management Unit
RCCU	Redundancy Control Checking Unit
RGM	Reset Generation Module
SAG	Safety Application Guide

Table 4. Acronyms

Acronym	Name
SIF	Safety Integrity Function
SSCM	System Status and Configuration Module
STCU	Self Test Control Unit
SWT	Software Watchdog Timer
XOSC	External Oscillator

Revision history

Table 5. Document revision history

Date	Revision	Changes
03-Jan-2011	1	Initial release.
25-Sep-2013	2	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