STM32G491xx/4A1xx device errata

Applicability

This document applies to the part numbers of STM32G491xx/4A1xx devices and the device variants as stated in this page.

It gives a summary and a description of the device errata, with respect to the device datasheet and reference manual RM0440.

Deviation of the real device behavior from the intended device behavior is considered to be a device limitation. Deviation of the description in the reference manual or the datasheet from the intended device behavior is considered to be a documentation erratum. The term *"errata"* applies both to limitations and documentation errata.

Table 1. Device summary

Reference	Part numbers
STM32G491xx	STM32G491CC, STM32G491CE, STM32G491KC, STM32G491KE, STM32G491MC, STM32G491ME, STM32G491RC, STM32G491RE, STM32G491VC, STM32G491VE
STM32G4A1xx	STM32G4A1CE, STM32G4A1KE, STM32G4A1ME, STM32G4A1RE, STM32G4A1VE

Table 2. Device variants

Reference	Silicon revision codes	
	Device marking(1)	REV_ID(2)
STM32G491xx/4A1xx	A	0x1000
STM32G491xx/4A1xx	Z	0x1001

1. *Refer to the device datasheet for how to identify this code on different types of package.*
2. *REV_ID[15:0] bitfield of DBGMCU_IDCODE register.*

1 Summary of device errata

The following table gives a quick reference to the STM32G491xx/4A1xx device limitations and their status:

A = limitation present, workaround available

N = limitation present, no workaround available

P = limitation present, partial workaround available

"-" = limitation absent

Applicability of a workaround may depend on specific conditions of target application. Adoption of a workaround may cause restrictions to target application. Workaround for a limitation is deemed partial if it only reduces the rate of occurrence and/or consequences of the limitation, or if it is fully effective for only a subset of instances on the device or in only a subset of operating modes, of the function concerned.

Table 3. Summary of device limitations

Function	Section	Limitation	Status	
			Rev. A	Rev. Z
Core	2.1.1	Interrupted loads to SP can cause erroneous behavior	A	A
	2.1.2	VDIV or VSQRT instructions might not complete correctly when very short ISRs are used	A	A
	2.1.3	Store immediate overlapping exception return operation might vector to incorrect interrupt	A	A
System	2.2.1	Full JTAG configuration without NJTRST pin cannot be used	A	A
	2.2.2	FLASH_ECCR corrupted upon reset or power-down occurring during flash memory program or erase operation	A	A
	2.2.3	MCU cannot enter in Standby mode when HSE bypass used	A	A
	2.2.4	Corrupted content of the backup domain due to a missed power-on reset after this domain supply voltage drop	A	A
	2.2.5	SRAM write error	A	A
	2.2.6	Wrong instruction fetches from flash memory upon wakeup from Sleep or Stop mode when debug in low-power mode is enabled	A	A
	2.2.7	LSE crystal oscillator may be disturbed by transitions on PC13	N	N
	2.2.8	Device cannot enter Standby mode when HSE bypass used as system clock	A	A
DMA	2.3.1	DMA disable failure and error flag omission upon simultaneous transfer error and global flag clear	A	A
DMAMUX	2.4.1	SOFx not asserted when writing into DMAMUX_CFR register	N	N
	2.4.2	OFx not asserted for trigger event coinciding with last DMAMUX request	N	N
	2.4.3	OFx not asserted when writing into DMAMUX_RGCFR register	N	N
	2.4.4	Wrong input DMA request routed upon specific DMAMUX_CxCR register write coinciding with synchronization event	A	A
QUADSPI	2.5.1	Memory-mapped read of last memory byte fails	P	P
ADC	2.6.1	New context conversion initiated without waiting for trigger when writing new context in ADC_JSQR with JQDIS = 0 and JQM = 0	A	A
	2.6.2	Two consecutive context conversions fail when writing new context in ADC_JSQR just after previous context completion with JQDIS = 0 and JQM = 0	A	A
	2.6.3	Unexpected regular conversion when two consecutive injected conversions are performed in Dual interleaved mode	A	A
	2.6.4	ADC_AWDy_OUT reset by non-guarded channels	A	A

Function	Section	Limitation	Status	
			Rev. A	Rev. Z
ADC	2.6.5	Injected data stored in the wrong ADC_JDRx registers	A	A
	2.6.6	ADC slave data may be shifted in Dual regular simultaneous mode	A	A
	2.6.7	Wrong ADC result if conversion done late after calibration or previous conversion	A	A
	2.6.8	ADC channel 0 converted instead of the required ADC channel	A	-
	2.6.9	An ADC instance may impact the accuracy of another ADC instance at specific conditions	A	A
OPAMP	2.7.1	OPAMP disturbed by fast-edge toggle on GPIO pin corresponding to VINM0	P	P
TIM	2.8.1	One-pulse mode trigger not detected in master-slave reset + trigger configuration	P	P
	2.8.2	Consecutive compare event missed in specific conditions	N	N
	2.8.3	Compare event missed in center-aligned mode 1 and 2 with dithering enabled	N	N
	2.8.4	Output compare clear not working with external counter reset	P	P
	2.8.5	Bidirectional break mode not working with short pulses	N	N
LPTIM	2.9.1	Device may remain stuck in LPTIM interrupt when entering Stop mode	A	A
	2.9.2	Device may remain stuck in LPTIM interrupt when clearing event flag	P	P
	2.9.3	LPTIM events and PWM output are delayed by one kernel clock cycle	P	P
RTC and TAMP	2.10.1	Calendar initialization may fail in case of consecutive INIT mode entry	A	A
	2.10.2	Alarm flag may be repeatedly set when the core is stopped in debug	N	N
	2.10.3	A tamper event fails to trigger timestamp or timestamp overflow events during a few cycles after clearing TSF	N	N
	2.10.4	REFCKON write protection associated to INIT KEY instead of CAL KEY	A	A
	2.10.5	Tamper flag not set on LSE failure detection	N	N
I2C	2.11.1	Wrong data sampling when data setup time ($t_{SU;DAT}$) is shorter than one I2C kernel clock period	P	P
	2.11.2	Spurious bus error detection in master mode	A	A
	2.11.3	OVR flag not set in underrun condition	N	N
	2.11.4	Transmission stalled after first byte transfer	A	A
	2.11.5	SDA held low upon SMBus timeout expiry in slave mode	A	A
USART	2.12.1	Anticipated end-of-transmission signaling in SPI slave mode	A	A
	2.12.2	Data corruption due to noisy receive line	A	A
	2.12.3	Received data may be corrupted upon clearing the ABREN bit	A	A
	2.12.4	Noise error flag set while ONEBIT is set	N	N
LPUART	2.13.1	Possible LPUART transmitter issue when using low BRR[15:0] value	P	P
SPI	2.14.1	BSY bit may stay high when SPI is disabled	A	A
	2.14.2	BSY bit may stay high at the end of data transfer in slave mode	A	A
FDCAN	2.15.1	Desynchronization under specific condition with edge filtering enabled	A	A
	2.15.2	Tx FIFO messages inverted under specific buffer usage and priority setting	A	A
USB	2.16.1	ESOF interrupt timing desynchronized after resume signaling	A	A
	2.16.2	Incorrect CRC16 in the memory buffer	N	N

Function	Section	Limitation	Status	
			Rev. A	Rev. Z
UCPD	2.17.1	TXHRST upon write data underflow corrupting the CRC of the next packet	A	A
	2.17.2	Ordered set with multiple errors in a single K-code is reported as invalid	N	N

2 Description of device errata

The following sections describe the errata of the applicable devices with Arm® core and provide workarounds if available. They are grouped by device functions.

Note: Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

2.1 Core

Reference manual and errata notice for the Arm® Cortex®-M4 core revision r0p1 is available from http://infocenter.arm.com.

2.1.1 Interrupted loads to SP can cause erroneous behavior

This limitation is registered under Arm ID number 752770 and classified into “Category B”. Its impact to the device is minor.

Description

If an interrupt occurs during the data-phase of a single word load to the stack-pointer (SP/R13), erroneous behavior can occur. In all cases, returning from the interrupt will result in the load instruction being executed an additional time. For all instructions performing an update to the base register, the base register will be erroneously updated on each execution, resulting in the stack-pointer being loaded from an incorrect memory location.

The affected instructions that can result in the load transaction being repeated are:

- LDR SP, [Rn],#imm
- LDR SP, [Rn,#imm]!
- LDR SP, [Rn,#imm]
- LDR SP, [Rn]
- LDR SP, [Rn,Rm]

The affected instructions that can result in the stack-pointer being loaded from an incorrect memory address are:

- LDR SP,[Rn],#imm
- LDR SP,[Rn,#imm]!

As compilers do not generate these particular instructions, the limitation is only likely to occur with hand-written assembly code.

Workaround

Both issues may be worked around by replacing the direct load to the stack-pointer, with an intermediate load to a general-purpose register followed by a move to the stack-pointer.

2.1.2 VDIV or VSQRT instructions might not complete correctly when very short ISRs are used

This limitation is registered under Arm ID number 776924 and classified into “Category B”. Its impact to the device is limited.

Description

The VDIV and VSQRT instructions take 14 cycles to execute. When an interrupt is taken a VDIV or VSQRT instruction is not terminated, and completes its execution while the interrupt stacking occurs. If lazy context save of floating point state is enabled then the automatic stacking of the floating point context does not occur until a floating point instruction is executed inside the interrupt service routine.

Lazy context save is enabled by default. When it is enabled, the minimum time for the first instruction in the interrupt service routine to start executing is 12 cycles. In certain timing conditions, and if there is only one or two instructions inside the interrupt service routine, then the VDIV or VSQRT instruction might not write its result to the register bank or to the FPSCR.

The failure occurs when the following condition is met:

1. The floating point unit is enabled
2. Lazy context saving is not disabled
3. A VDIV or VSQRT is executed
4. The destination register for the VDIV or VSQRT is one of s0 - s15
5. An interrupt occurs and is taken
6. The interrupt service routine being executed does not contain a floating point instruction
7. Within 14 cycles after the VDIV or VSQRT is executed, an interrupt return is executed

A minimum of 12 of these 14 cycles are utilized for the context state stacking, which leaves 2 cycles for instructions inside the interrupt service routine, or 2 wait states applied to the entire stacking sequence (which means that it is not a constant wait state for every access).

In general, this means that if the memory system inserts wait states for stack transactions (that is, external memory is used for stack data), then this erratum cannot be observed.

The effect of this erratum is that the VDIV or VQSRT instruction does not complete correctly and the register bank and FPSCR are not updated, which means that these registers hold incorrect, out of date, data.

Workaround

A workaround is only required if the floating point unit is enabled. A workaround is not required if the stack is in external memory.

There are two possible workarounds:

- Disable lazy context save of floating point state by clearing LSPEN to 0 (bit 30 of the FPCCR at address 0xE000EF34).
- Ensure that every interrupt service routine contains more than 2 instructions in addition to the exception return instruction.

2.1.3 Store immediate overlapping exception return operation might vector to incorrect interrupt

This limitation is registered under Arm ID number 838869 and classified into “Category B (rare)”. Its impact to the device is minor.

Description

The core includes a write buffer that permits execution to continue while a store is waiting on the bus. Under specific timing conditions, during an exception return while this buffer is still in use by a store instruction, a late change in selection of the next interrupt to be taken might result in there being a mismatch between the interrupt acknowledged by the interrupt controller and the vector fetched by the processor.

The failure occurs when the following condition is met:

1. The handler for interrupt A is being executed.
2. Interrupt B, of the same or lower priority than interrupt A, is pending.
3. A store with immediate offset instruction is executed to a bufferable location.
   - STR/STRH/STRB <Rt>, [<Rn>,#imm]
   - STR/STRH/STRB <Rt>, [<Rn>,#imm]!
   - STR/STRH/STRB <Rt>, [<Rn>],#imm
4. Any number of additional data-processing instructions can be executed.
5. A BX instruction is executed that causes an exception return.
6. The store data has wait states applied to it such that the data is accepted at least two cycles after the BX is executed.
   - Minimally, this is two cycles if the store and the BX instruction have no additional instructions between them.
   - The number of wait states required to observe this erratum needs to be increased by the number of cycles between the store and the interrupt service routine exit instruction.
7. Before the bus accepts the buffered store data, another interrupt C is asserted which has the same or lower priority as A, but a greater priority than B.

Example:

The processor should execute interrupt handler C, and on completion of handler C should execute the handler for B. If the conditions above are met, then this erratum results in the processor erroneously clearing the pending state of interrupt C, and then executing the handler for B twice. The first time the handler for B is executed it will be at interrupt C's priority level. If interrupt C is pended by a level-based interrupt which is cleared by C's handler then interrupt C will be pended again once the handler for B has completed and the handler for C will be executed.

As the STM32 interrupt C is level based, it eventually becomes pending again and is subsequently handled.

Workaround

For software not using the memory protection unit, this erratum can be worked around by setting DISDEFWBUF in the Auxiliary Control Register.

In all other cases, the erratum can be avoided by ensuring a DSB occurs between the store and the BX instruction. For exception handlers written in C, this can be achieved by inserting the appropriate set of intrinsics or inline assembly just before the end of the interrupt function, for example:

ARMCC:

```
...
__schedule_barrier();
__asm{DSB};
__schedule_barrier();
}
```

GCC:

```
...
__asm volatile ("dsb 0xf":::"memory");
}
```

2.2 System

2.2.1 Full JTAG configuration without NJTRST pin cannot be used

Description

When using the JTAG debug port in Debug mode, the connection with the debugger is lost if the NJTRST pin (PB4) is used as a GPIO or for an alternate function other than NJTRST. Only the 4-wire JTAG port configuration is impacted.

Workaround

Use the SWD debug port instead of the full 4-wire JTAG port.

2.2.2 FLASH_ECCR corrupted upon reset or power-down occurring during flash memory program or erase operation

Description

Reset or power-down occurring during a flash memory location program or erase operation, followed by a read of the same memory location, may lead to a corruption of the FLASH_ECCR register content.

Workaround

Under such condition, erase the page(s) corresponding to the flash memory location.

2.2.3 MCU cannot enter in Standby mode when HSE bypass used

Description

When the system clock is using HSE in bypass mode, the MCU cannot enter into Standby mode.

Workaround

While using HSE clock in bypass mode, switch the system clock to HSI clock before entering into Standby mode.

2.2.4 Corrupted content of the backup domain due to a missed power-on reset after this domain supply voltage drop

Description

The backup domain reset may be missed upon a power-on following a power-off, if its supply voltage drops during the power-off phase hitting a window, which is few mV wide before it starts to rise again. In this critical window, the flip-flops are no longer able to safely retain the information and the backup domain reset has not yet been triggered. This window is located in the range between 100 mV and 700 mV, with the exact position depending mainly on the device and on the temperature.

This missed reset results in unpredictable values of the backup domain registers. This may cause a spurious behavior (such as driving the LSCO output pin on PA2 or influencing backup functions).

Workaround

Apply one of the following measures:

- In the application, let the V_{DD} and V_{BAT} supply voltages fall to a level below 100 mV for more than 200 ms before a new power-on.
- If the above workaround cannot be applied, and the boot follows a power-on reset, erase the backup domain by software.
  When the application is using shutdown mode, user needs to discriminate between the power-on reset or an exit from a shutdown mode.
  For this purpose, at least one backup register must have been previously programmed with a BKP_REG_VAL value with 16 bits set and 16 bits cleared.
  Robustness of this workaround can be significantly improved by using a CRC rather than registers. The registers are subject to backup domain reset.
  The workaround consists of calculating the CRC of the backup registers: RCC_BDCR and RTC registers, excluding bits modified by HW.
  The CRC result can be stored in the backup register instead of a fixed value. This value needs to be updated for each modification of values covered by CRC, such as by using CRC peripheral.
  At the very beginning of the boot code, insert the following software sequence:
  1. Check the BORRSTF flag of the RCC_CSR register. If set, the reset is caused by a power on, or is exiting from shutdown mode.
  2. If BORRSTF flag is true, and the shutdown mode is used in the application, check that the backup register value is different from BKP_REG_VAL. When tamper detection is enabled,check that no tamper flag is set. If both conditions are met then the reset is caused by a power-on.
  3. If the reset is caused by a power-on, apply the following sequence:
     a. Enable the PWR clock in the RCC, by setting the PWREN bit.
     b. Enable the backup domain access in the PWR, by setting the DBP bit.
     c. Reset the backup domain, by:
        i. Writing 0x0001 0000 in the RCC_BDCR register, which sets the BDRST bit and clears other register bits that might not be reset.
        ii. reading the RCC_BDCR register, to make the reset time long enough
        iii. writing 0x0000 0000 in the RCC_BDCR register, to clear the BDRST bit
     d. Clear the BORRSTF flag by setting the RMVF bit of the RCC_CSR register.

2.2.5 SRAM write error

Description

In rare cases, system reset occurring in a critical instant may upset the SRAM state machine. The first SRAM read or write access after the system reset then restores the normal operation of the SRAM for any subsequent accesses. However, if it is a write access, it fails to write data.

Workaround

Upon system reset, make a dummy read access to each 32-Kbyte SRAM instance used by the application, through one of the following methods:

1. Place a dummy variable initialization, which allows the compiler to create the assembly code.
2. Use this initialization assembly code:

```
MOV32 R0, #0x20000000 //SRAM address
LDR R1, [R0, #+0] //read access
MOV32 R0, #0x20008000 //next SRAM instance
LDR R1, [R0, #+0] //dummy read, no consequence on R1 value
MOV32 R0, #0x20010000
LDR R1, [R0, #+0]
MOV32 R0, #0x20014000 //the SRAM2 cut
LDR R1, [R0, #+0]
MOV32 R0, #0x10000000 // CCM SRAM
LDR R1, [R0, #+0]
```

Follow the first method for SRAM instances with the parity check enabled. Follow the first or the second method for SRAM instances with the parity check disabled.

2.2.6 Wrong instruction fetches from flash memory upon wakeup from Sleep or Stop mode when debug in low-power mode is enabled

Description

When debug in low-power mode is enabled, wrong instructions can be fetched from flash memory and executed after waking up from Sleep or Stop mode, causing an unpredictable behavior of the device or a CPU exception.

This issue occurs when the device exits Sleep or Stop mode in the following conditions:

- At least one of DBG_SLEEP or DBG_STOP bit of DBGMCU_CR register is set.
- Interrupts with wakeup capability are disabled at Sleep or Stop entry.
- The flash memory interface gating is enabled by clearing FLASHSMEN bit of RCC_AHB1SMENR (only for Sleep mode).

Note: The issue affects only debug mode and has no effect on real applications.

Workaround

Apply one of the following measures:

- Add an ISB just after WFI (or WFE) instruction.
- Disable the flash memory interface gating, by setting FLASHSMEN bit (valid only for Sleep mode).

2.2.7 LSE crystal oscillator may be disturbed by transitions on PC13

Description

On LQFP and UFQFPN packages, the LSE crystal oscillator clock frequency can be incorrect when PC13 is toggling in input or output (for example when used for RTC_OUT1).

The external clock input (LSE bypass) is not impacted by this limitation.

The WLCSP and UFBGA packages are not impacted by this limitation.

Workaround

None.

Avoid toggling PC13 when LSE is used on LQFP and UFQFPN packages.

2.2.8 Device cannot enter Standby mode when HSE bypass used as system clock

Description

The device using HSE clock in bypass mode as system clock cannot enter Standby mode.

Workaround

Select HSI as system clock before entering Standby mode.

2.3 DMA

2.3.1 DMA disable failure and error flag omission upon simultaneous transfer error and global flag clear

Description

Upon a data transfer error in a DMA channel x, both the specific TEIFx and the global GIFx flags are raised and the channel x is normally automatically disabled. However, if in the same clock cycle the software clears the GIFx flag (by setting the CGIFx bit of the DMA_IFCR register), the automatic channel disable fails and the TEIFx flag is not raised.

This issue does not occur with ST's HAL software that does not use and clear the GIFx flag when the channel is active.

Workaround

Do not clear GIFx flags when the channel is active. Instead, use HTIFx, TCIFx, and TEIFx specific event flags and their corresponding clear bits.

2.4 DMAMUX

2.4.1 SOFx not asserted when writing into DMAMUX_CFR register

Description

The SOFx flag of the DMAMUX_CSR status register is not asserted if overrun from another DMAMUX channel occurs when the software writes into the DMAMUX_CFR register.

This can happen when multiple DMA channels operate in synchronization mode, and when overrun can occur from more than one channel. As the SOFx flag clear requires a write into the DMAMUX_CFR register (to set the corresponding CSOFx bit), overrun occurring from another DMAMUX channel operating during that write operation fails to raise its corresponding SOFx flag.

Workaround

None. Avoid the use of synchronization mode for concurrent DMAMUX channels, if at least two of them potentially generate synchronization overrun.

2.4.2 OFx not asserted for trigger event coinciding with last DMAMUX request

Description

In the DMAMUX request generator, a trigger event detected in a critical instant of the last-generated DMAMUX request being served by the DMA controller does not assert the corresponding trigger overrun flag OFx. The critical instant is the clock cycle at the very end of the trigger overrun condition.

Additionally, upon the following trigger event, one single DMA request is issued by the DMAMUX request generator, regardless of the programmed number of DMA requests to generate.

The failure only occurs if the number of requests to generate is set to more than two (GNBREQ[4:0] > 00001).

Workaround

Make the trigger period longer than the duration required for serving the programmed number of DMA requests, so as to avoid the trigger overrun condition from occurring on the very last DMA data transfer.

2.4.3 OFx not asserted when writing into DMAMUX_RGCFR register

Description

The OFx flag of the DMAMUX_RGSR status register is not asserted if an overrun from another DMAMUX request generator channel occurs when the software writes into the DMAMUX_RGCFR register. This can happen when multiple DMA channels operate with the DMAMUX request generator, and when an overrun can occur from more than one request generator channel. As the OFx flag clear requires a write into the DMAMUX_RGCFR register (to set the corresponding COFx bit), an overrun occurring in another DMAMUX channel operating with another request generator channel during that write operation fails to raise the corresponding OFx flag.

Workaround

None. Avoid the use of request generator mode for concurrent DMAMUX channels, if at least two channels are potentially generating a request generator overrun.

2.4.4 Wrong input DMA request routed upon specific DMAMUX_CxCR register write coinciding with synchronization event

Description

If a write access into the DMAMUX_CxCR register having the SE bit at zero and SPOL[1:0] bitfield at a value other than 00:

- sets the SE bit (enables synchronization),
- modifies the values of the DMAREQ_ID[5:0] and SYNC_ID[4:0] bitfields, and
- does not modify the SPOL[1:0] bitfield,

and if a synchronization event occurs on the previously selected synchronization input exactly two AHB clock cycles before this DMAMUX_CxCR write, then the input DMA request selected by the DMAREQ_ID[5:0] value before that write is routed.

Workaround

Ensure that the SPOL[1:0] bitfield is at 00 whenever the SE bit is 0. When enabling synchronization by setting the SE bit, always set the SPOL[1:0] bitfield to a value other than 00 with the same write operation into the DMAMUX_CxCR register.

2.5 QUADSPI

2.5.1 Memory-mapped read of last memory byte fails

Description

Regardless of the number of I/O lines used (1, 2 or 4), a memory-mapped read of the last byte of the memory region defined through the FSIZE[4:0] bitfield of the QUADSPI_DCR register always yields 0x00, whatever the memory byte content is. A repeated attempt to read that last byte causes the AXI bus to stall.

Workaround

Apply one of the following measures:

- Avoid reading the last byte of the memory region defined through FSIZE, for example by taking margin in FSIZE bitfield setting.
- If the last byte is read, ignore its value and abort the ongoing process so as to prevent the AXI bus from stalling.
- For reading the last byte of the memory region defined through FSIZE, use indirect read.

2.6 ADC

2.6.1 New context conversion initiated without waiting for trigger when writing new context in ADC_JSQR with JQDIS = 0 and JQM = 0

Description

Once an injected conversion sequence is complete, the queue is consumed and the context changes according to the new ADC_JSQR parameters stored in the queue. This new context is applied for the next injected sequence of conversions.

However, the programming of the new context in ADC_JSQR (change of injected trigger selection and/or trigger polarity) may launch the execution of this context without waiting for the trigger if:

- the queue of context is enabled (JQDIS cleared to 0 in ADC_CFGR), and
- the queue is never empty (JQM cleared to 0 in ADC_CFGR), and
- the injected conversion sequence is complete and no conversion from previous context is ongoing

Workaround

Apply one of the following measures:

- Ignore the first conversion.
- Use a queue of context with JQM = 1.
- Use a queue of context with JQM = 0, only change the conversion sequence but never the trigger selection and the polarity.

2.6.2 Two consecutive context conversions fail when writing new context in ADC_JSQR just after previous context completion with JQDIS = 0 and JQM = 0

Description

When an injected conversion sequence is complete and the queue is consumed, writing a new context in ADC_JSQR just after the completion of the previous context and with a length longer that the previous context, may cause both contexts to fail. The two contexts are considered as one single context. As an example, if the first context contains element 1 and the second context elements 2 and 3, the first context is consumed followed by elements 2 and 3 and element 1 is not executed.

This issue may happen if:

- the queue of context is enabled (JQDIS cleared to 0 in ADC_CFGR), and
- the queue is never empty (JQM cleared to 0 in ADC_CFGR), and
- the length of the new context is longer than the previous one

Workaround

If possible, synchronize the writing of the new context with the reception of the new trigger.

2.6.3 Unexpected regular conversion when two consecutive injected conversions are performed in Dual interleaved mode

Description

In Dual ADC mode, an unexpected regular conversion may start at the end of the second injected conversion without a regular trigger being received, if the second injected conversion starts exactly at the same time than the end of the first injected conversion. This issue may happen in the following conditions:

- two consecutive injected conversions performed in Interleaved simultaneous mode (DUAL[4:0] of ADC_CCR = 0b00011), or
- two consecutive injected conversions from master or slave ADC performed in Interleaved mode (DUAL[4:0]of ADC_CCR = 0b00111)

Workaround

- In Interleaved simultaneous injected mode: make sure the time between two injected conversion triggers is longer than the injected conversion time.
- In Interleaved only mode: perform injected conversions from one single ADC (master or slave), making sure the time between two injected triggers is longer than the injected conversion time.

2.6.4 ADC_AWDy_OUT reset by non-guarded channels

Description

ADC_AWDy_OUT is set when a guarded conversion of a regular or injected channel is outside the programmed thresholds. It is reset after the end of the next guarded conversion that is inside the programmed thresholds.

However, the ADC_AWDy_OUT signal is also reset at the end of conversion of non-guarded channels, both regular and injected.

Workaround

When ADC_AWDy_OUT is enabled, it is recommended to use only the ADC channels that are guarded by a watchdog.

If ADC_AWDy_OUT is used with ADC channels that are not guarded by a watchdog, take only ADC_AWDy_OUT rising edge into account.

2.6.5 Injected data stored in the wrong ADC_JDRx registers

Description

When the AHB clock frequency is higher than the ADC clock frequency after the prescaler is applied (ratio > 10), if a JADSTP command is issued to stop the injected conversion (JADSTP bit set to 1 in ADC_CR register) at the end of an injected conversion, exactly when the data are available, then the injected data are stored in ADC_JDR1 register instead of ADC_JDR2/3/4 registers.

Workaround

Before setting JADSTP bit, check that the JEOS flag is set in ADC_ISR register (end of injected channel sequence).

2.6.6 ADC slave data may be shifted in Dual regular simultaneous mode

Description

In Dual regular simultaneous mode, ADC slave data may be shifted when all the following conditions are met:

- A read operation is performed by one DMA channel,
- OVRMOD = 0 in ADC_CFGR register (Overrrun mode enabled).

Workaround

Apply one of the following measures:

- Set OVRMOD = 1 in ADC_CFGR. This disables ADC_DR register FIFO.
- Use two DMA channels to read data: one for slave and one for master.

2.6.7 Wrong ADC result if conversion done late after calibration or previous conversion

Description

The result of an ADC conversion done more than 1 ms later than the previous ADC conversion or ADC calibration might be incorrect.

Workaround

Perform two consecutive ADC conversions in single, scan or continuous mode. Reject the result of the first conversion and only keep the result of the second.

2.6.8 ADC channel 0 converted instead of the required ADC channel

Description

The following ADC conversions unduly convert the ADC internal channel 0 instead of the required channel, and as a consequence, they return zero as conversion result:

1. an injected conversion triggered while regular conversion is on-going
2. the master ADC first injected conversion and the slave ADC first regular conversion (after resume) when dual simultaneous or interleaved dual ADC is used
3. the first regular/injected conversion after a regular or injected software STOP (setting ADSTP or JADSTP bit respectively)
4. the first regular or injected conversion following the DMA end of transfer, after the completion of a regular sequence read by the DMA in one-shot mode

Workaround

Apply one of the following measures:

- Depending on the case, insert a dummy conversion:
  - at the beginning of the injected sequence (not applicable in injected discontinuous mode, JDISCEN = 1) in the case 1
  - after the ADC slave resumes regular conversions in dual simultaneous or interleaved dual ADC mode (not applicable in injected discontinuous mode, JDISCEN = 1) in the case 2
  - after stopping the ADC by software in the case 3
  - after DMA end-of-transfer in the case 4
- Avoid collisions between injected and regular conversions by using triggered regular conversions or by launching regular conversion at the end of injected sequence in the cases 1 and 2.
- After a regular or injected software STOP, disable the ADC with ADDIS = 1 and enable it again with ADEN = 1. This introduces a hardware dummy conversion (the cases 3 and 4).

2.6.9 An ADC instance may impact the accuracy of another ADC instance at specific conditions

Description

An ADC instance may impact the accuracy of another ADC instance if their respective conversions are concurrent. That is, when they occur, partly or entirely, at the same time.

Note: The term ADC conversion comprises the sampling phase and the successive approximation phase.

The instances ADC1 and ADC2 operating with asynchronous clock do not mutually impact one another. The instances ADC3, ADC4, and ADC5 operating with asynchronous clock do not mutually impact each other, either. Furthermore, the instances ADC1 and ADC2 operating in a dual mode other than *alternate trigger mode only* do not impact one another whatever the clock source. The same holds for the instances ADC3 and ADC4 operating in a dual mode other than *alternate trigger mode only*.

For the other combinations of ADC instances running concurrent conversions, one instance may impact the accuracy of another instance when:

- they use different clock sources, and/or
- the clock prescaler division ratio is equal to or greater than two.

Note: The settings for the prescaler division ratio equal to or greater than two are:

- *synchronous clock: CKMODE[1:0] = 10 (division by 2) or CKMODE[1:0] = 11 (division by 4)*
- *asynchronous clock: CKMODE[1:0] = 00 and PRESC[3:0] ≠ 0000 (division by 2 and more)*

The settings for the prescaler division ratio equal to one (no division) are:

- *synchronous clock: CKMODE[1:0] = 01*
- *asynchronous clock: CKMODE[1:0] = 00 and PRESC[3:0] = 0000*

Workaround

Apply one of the following measures:

1. Ensure that conversions by different ADC instances do not occur concurrently. This can be achieved through trigger sequencing by the application software.
2. Use the same clock source for all ADC instances running concurrent conversions and set the clock prescaler division ratio to one (no division).
3. With synchronous clock and the clock prescaler division ratio equal to or greater than two, use the same clock configuration for the concurrently converting ADC instances. Furthermore, trigger them with the same timer using the same clock prescaler division ratio (or its integer multiple) as the ADC prescaler.

Note: The timer can use different outputs (such as timN_trgo, timN_oc1) for different ADC instances, to trigger regular and injected conversions.

2.7 OPAMP

2.7.1 OPAMP disturbed by fast-edge toggle on GPIO pin corresponding to VINM0

Description

In non-inverting PGA OPAMP mode, fast-edge toggle on the GPIO pin corresponding to VINM0 OPAMP input disturbs the operational amplifier inverting input. The disturbance on the OPAMP output is then multiplied depending on the gain set.

Note: The disturbance is not observed in the OPAMP external gain and follower modes.

Note: The VINM0 input corresponds to the GPIO pin PA3 for OPAMP1, PA5 for OPAMP2, PB2 for OPAMP3, and PA1 for OPAMP6.

Workaround

Apply one of the following measures:

- Prevent fast-edge toggle on VINM0, by using the corresponding GPIO as slow-edge analog signal input, as a static output, or by grounding it.
- If the OPAMP output is sampled by an ADC, synchronize that sampling with the fast-edge VINM0 signal such as to move the ADC sampling away from the VINM0 signal edges.

2.8 TIM

2.8.1 One-pulse mode trigger not detected in master-slave reset + trigger configuration

Description

The failure occurs when several timers configured in one-pulse mode are cascaded, and the master timer is configured in combined reset + trigger mode with the MSM bit set:

OPM = 1 in TIMx_CR1, SMS[3:0] = 1000 and MSM = 1 in TIMx_SMCR.

The MSM delays the reaction of the master timer to the trigger event, so as to have the slave timers cycle-accurately synchronized.

If the trigger arrives when the counter value is equal to the period value set in the TIMx_ARR register, the one-pulse mode of the master timer does not work and no pulse is generated on the output.

Workaround

None. However, unless a cycle-level synchronization is mandatory, it is advised to keep the MSM bit reset, in which case the problem is not present. The MSM = 0 configuration also allows decreasing the timer latency to external trigger events.

2.8.2 Consecutive compare event missed in specific conditions

Description

Every match of the counter (CNT) value with the compare register (CCR) value is expected to trigger a compare event. However, if such matches occur in two consecutive counter clock cycles (as consequence of the CCR value change between the two cycles), the second compare event is missed for the following CCR value changes:

- in edge-aligned mode, from ARR to 0:
  - first compare event: CNT = CCR = ARR
  - second (missed) compare event: CNT = CCR = 0
- in center-aligned mode while up-counting, from ARR-1 to ARR (possibly a new ARR value if the period is also changed) at the crest (that is, when TIMx_RCR = 0):
  - first compare event: CNT = CCR = (ARR-1)
  - second (missed) compare event: CNT = CCR = ARR
- in center-aligned mode while down-counting, from 1 to 0 at the valley (that is, when TIMx_RCR = 0):
  - first compare event: CNT = CCR = 1
  - second (missed) compare event: CNT = CCR = 0

This typically corresponds to an abrupt change of compare value aiming at creating a timer clock single-cycle-wide pulse in toggle mode.

As a consequence:

- In toggle mode, the output only toggles once per counter period (squared waveform), whereas it is expected to toggle twice within two consecutive counter cycles (and so exhibit a short pulse per counter period).
- In center mode, the compare interrupt flag does note rise and the interrupt is not generated.

Note: The timer output operates as expected in modes other than the toggle mode.

Workaround

None.

2.8.3 Compare event missed in center-aligned mode 1 and 2 with dithering enabled

Description

With dithering enabled, the compare event is not generated and the output in toggle mode is incorrect in:

- center-aligned mode 1, with the CCR value between ARR-16 and ARR
- center-aligned mode 2, with the CCR value lower than 16

Note: These CCR values correspond to a waveform with extreme duty cycles: close to 0%, with the pulse width being one timer clock cycle, or close to 100%, with the pulse width being the PWM period minus one timer clock cycle.

The timer output operates as expected if configured in other than toggle mode.

The center-aligned mode 3 is exempt of this failure.

Workaround

None.

2.8.4 Output compare clear not working with external counter reset

Description

The output compare clear event (ocref_clr) is not correctly generated when the timer is configured in the following slave modes: Reset mode, Combined reset + trigger mode, and Combined gated + reset mode.

The PWM output remains inactive during one extra PWM cycle if the following sequence occurs:

1. The output is cleared by the ocref_clr event.
2. The timer reset occurs before the programmed compare event.

Workaround

Apply one of the following measures:

- Use BKIN (or BKIN2 if available) input for clearing the output, selecting the Automatic output enable mode (AOE = 1).
- Mask the timer reset during the PWM ON time to prevent it from occurring before the compare event (for example with a spare timer compare channel open-drain output connected with the reset signal, pulling the timer reset line down).

2.8.5 Bidirectional break mode not working with short pulses

Description

The TIM_BKIN and TIM_BKIN2 I/Os can be configured in bidirectional mode using the BKBID and BK2BID bits in the TIMx_BDTR register, to be forced to 0 when a break/break2 event occurs. The bidirectional break/break2 mode is not functional when the pulse width on break/break2 input is lower than two tim_ker_clk periods.

This limitation is also valid when software break events are generated (the break event is correctly generated internally but not reflected on break inputs).

Workaround

None.

For applications that can afford some latency in bidirectional break mode, the break interrupt can eventually be enabled, for the CPU to verify the break input state and force it to zero when a break/break2 event occurred.

2.9 LPTIM

2.9.1 Device may remain stuck in LPTIM interrupt when entering Stop mode

Description

This limitation occurs when disabling the low-power timer (LPTIM).

When the user application clears the ENABLE bit in the LPTIM_CR register within a small time window around one LPTIM interrupt occurrence, then the LPTIM interrupt signal used to wake up the device from Stop mode may be frozen in active state. Consequently, when trying to enter Stop mode, this limitation prevents the device from entering low-power mode and the firmware remains stuck in the LPTIM interrupt routine.

This limitation applies to all Stop modes and to all instances of the LPTIM. Note that the occurrence of this issue is very low.

Workaround

In order to disable a low power timer (LPTIMx) peripheral, do not clear its ENABLE bit in its respective LPTIM_CR register. Instead, reset the whole LPTIMx peripheral via the RCC controller by setting and resetting its respective LPTIMxRST bit in the relevant RCC register.

2.9.2 Device may remain stuck in LPTIM interrupt when clearing event flag

Description

This limitation occurs when the LPTIM is configured in interrupt mode (at least one interrupt is enabled) and the software clears any flag in LPTIM_ISR register by writing its corresponding bit in LPTIM_ICR register. If the interrupt status flag corresponding to a disabled interrupt is cleared simultaneously with a new event detection, the set and clear commands might reach the APB domain at the same time, leading to an asynchronous interrupt signal permanently stuck high.

This issue can occur either during an interrupt subroutine execution (where the flag clearing is usually done), or outside an interrupt subroutine.

Consequently, the firmware remains stuck in the LPTIM interrupt routine, and the device cannot enter Stop mode.

Workaround

To avoid this issue, it is strongly advised to follow the recommendations listed below:

- Clear the flag only when its corresponding interrupt is enabled in the interrupt enable register.
- If for specific reasons, it is required to clear some flags that have corresponding interrupt lines disabled in the interrupt enable register, it is recommended to clear them during the current subroutine prior to those which have corresponding interrupt line enabled in the interrupt enable register.
- Flags must not be cleared outside the interrupt subroutine.

Note: The standard clear sequence implemented in the HAL_LPTIM_IRQHandler in the STM32Cube is considered as the proper clear sequence.

2.9.3 LPTIM events and PWM output are delayed by one kernel clock cycle

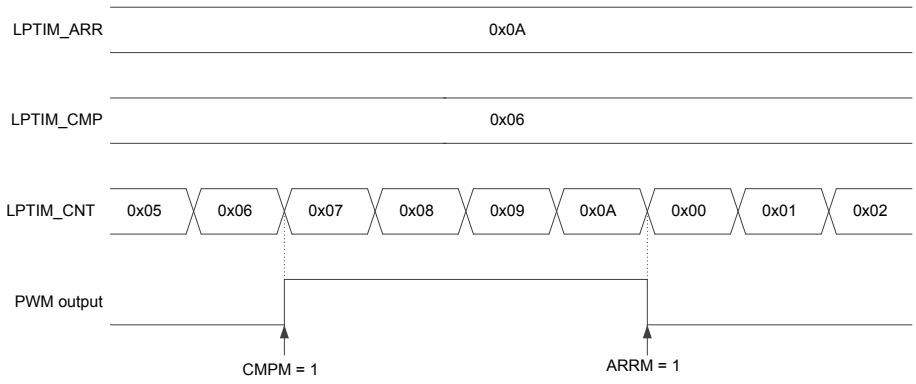
Description

The compare match event (CMPM), auto reload match event (ARRM), PWM output level and interrupts are updated with a delay of one kernel clock cycle.

Consequently, it is not possible to generate PWM with a duty cycle of 0% or 100%.

The following waveform gives the example of PWM output mode and the effect of the delay:

Figure 1. Example of PWM output mode



Workaround

Set the compare value to the desired value minus 1. For instance in order to generate a compare match when LPTM_CNT = 0x08, set the compare value to 0x07.

2.10 RTC and TAMP

2.10.1 Calendar initialization may fail in case of consecutive INIT mode entry

Description

If the INIT bit of the RTC_ICSR register is set between one and two RTCCLK cycles after being cleared, the INITF flag is set immediately instead of waiting for synchronization delay (which should be between one and two RTCCLK cycles), and the initialization of registers may fail. Depending on the INIT bit clearing and setting instants versus the RTCCLK edges, it can happen that, after being immediately set, the INITF flag is cleared during one RTCCLK period then set again. As writes to calendar registers are ignored when INITF is low, a write occurring during this critical period might result in the corruption of one or more calendar registers.

Workaround

After existing the initialization mode, clear the BYPSHAD bit (if set) then wait for RSF to rise, before entering the initialization mode again.

Note: It is recommended to write all registers in a single initialization session to avoid accumulating synchronization delays.

2.10.2 Alarm flag may be repeatedly set when the core is stopped in debug

Description

When the core is stopped in debug mode, the clock is supplied to subsecond RTC alarm downcounter even when the device is configured to stop the RTC in debug.

As a consequence, when the subsecond counter is used for alarm condition (the MASKSS[3:0] bitfield of the RTC_ALRMASSR and/or RTC_ALRMBSSR register set to a non-zero value) and the alarm condition is met just before entering a breakpoint or printf, the ALRAF and/or ALRBF flag of the RTC_SR register is repeatedly set by hardware during the breakpoint or printf, which makes any attempt to clear the flag(s) ineffective.

Workaround

None.

2.10.3 A tamper event fails to trigger timestamp or timestamp overflow events during a few cycles after clearing TSF

Description

With the timestamp on tamper event enabled (TAMPTS bit of the RTC_CR register set), a tamper event is ignored if it occurs:

- within four APB clock cycles after setting the CTSF bit of the RTC_SCR register to clear the TSF flag, while the TSF flag is not yet effectively cleared (it fails to set the TSOVF flag)
- within two ck_apre cycles after setting the CTSF bit of the RTC_SCR register to clear the TSF flag, when the TSF flag is effectively cleared (it fails to set the TSF flag and timestamp the calendar registers)

Workaround

None.

2.10.4 REFCKON write protection associated to INIT KEY instead of CAL KEY

Description

The write protection of the REFCKON bit is unlocked if the key sequence is written in RTC_WPR with the privilege and security rights set by the INITPRIV and INITSEC bits, instead of being set by the CALPRIV and CALSEC bits.

Workaround

Unlock the INIT KEY before writing REFCKON.

2.10.5 Tamper flag not set on LSE failure detection

Description

With the timestamp on tamper event enabled (the TAMPTS bit of the RTC_CR register set), the LSE failure detection (LSE clock stopped) event connected to the internal tamper 3 fails to raise the ITAMP3F and ITAMP3MF flags, although it duly erases or blocks (depending on the internal tamper 3 configuration) the backup registers and other device secrets, and the RTC and TAMP peripherals resume normally upon the LSE restart.

Note: As expected in this particular case, the TSF and TSMF flags remain low as long as LSE is stopped as they require running RTCCLK clock to operate.

Workaround

None.

2.11 I2C

2.11.1 Wrong data sampling when data setup time ($t_{SU;DAT}$) is shorter than one I2C kernel clock period

Description

The I^2C-bus specification and user manual specify a minimum data setup time ($t_{SU;DAT}$) as:

- 250 ns in Standard mode
- 100 ns in Fast mode
- 50 ns in Fast mode Plus

The device does not correctly sample the I^2C-bus SDA line when $t_{SU;DAT}$ is smaller than one I2C kernel clock (I^2C-bus peripheral clock) period: the previous SDA value is sampled instead of the current one. This can result in a wrong receipt of slave address, data byte, or acknowledge bit.

Workaround

Increase the I2C kernel clock frequency to get I2C kernel clock period within the transmitter minimum data setup time. Alternatively, increase transmitter's minimum data setup time. If the transmitter setup time minimum value corresponds to the minimum value provided in the I^2C-bus standard, the minimum I2CCLK frequencies are as follows:

- In Standard mode, if the transmitter minimum setup time is 250 ns, the I2CCLK frequency must be at least 4 MHz.
- In Fast mode, if the transmitter minimum setup time is 100 ns, the I2CCLK frequency must be at least 10 MHz.
- In Fast-mode Plus, if the transmitter minimum setup time is 50 ns, the I2CCLK frequency must be at least 20 MHz.

2.11.2 Spurious bus error detection in master mode

Description

In master mode, a bus error can be detected spuriously, with the consequence of setting the BERR flag of the I2C_SR register and generating bus error interrupt if such interrupt is enabled. Detection of bus error has no effect on the I^2C-bus transfer in master mode and any such transfer continues normally.

Workaround

If a bus error interrupt is generated in master mode, the BERR flag must be cleared by software. No other action is required and the ongoing transfer can be handled normally.

2.11.3 OVR flag not set in underrun condition

Description

In slave transmission with clock stretching disabled (NOSTRETCH = 1 in the I2C_CR1 register), an underrun condition occurs if the current byte transmission is completed on the I^2C bus, and the next data is not yet written in the TXDATA[7:0] bitfield. In this condition, the device is expected to set the OVR flag of the I2C_ISR register and send 0xFF on the bus.

However, if the I2C_TXDR is written within the interval between two I2C kernel clock cycles before and three APB clock cycles after the start of the next data transmission, the OVR flag is not set, although the transmitted value is 0xFF.

Workaround

None.

2.11.4 Transmission stalled after first byte transfer

Description

When the first byte to transmit is not prepared in the TXDATA register, two bytes are required successively, through TXIS status flag setting or through a DMA request. If the first of the two bytes is written in the I2C_TXDR register in less than two I2C kernel clock cycles after the TXIS/DMA request, and the ratio between APB clock and I2C kernel clock frequencies is between 1.5 and 3, the second byte written in the I2C_TXDR is not internally detected. This causes a state in which the I2C peripheral is stalled in master mode or in slave mode, with clock stretching enabled (NOSTRETCH = 0). This state can only be released by disabling the peripheral (PE = 0) or by resetting it.

Workaround

Apply one of the following measures:

- Write the first data in I2C_TXDR before the transmission starts.
- Set the APB clock frequency so that its ratio with respect to the I2C kernel clock frequency is lower than 1.5 or higher than 3.

2.11.5 SDA held low upon SMBus timeout expiry in slave mode

Description

For the slave mode, the SMBus specification defines $t_{TIMEOUT}$ (detect clock low timeout) and $t_{LOW:SEXT}$ (cumulative clock low extend time) timeouts. When one of them expires while the I2C peripheral in slave mode drives SDA low to acknowledge either its address or a data transmitted by the master, the device is expected to report such an expiry and release the SDA line.

However, although the device duly reports the timeout expiry, it fails to release SDA. This stalls the I^2C bus and prevents the master from generating RESTART or STOP condition.

Workaround

When a timeout is reported in slave mode (TIMEOUT bit of the I2C_ISR register is set), apply this sequence:

1. Wait until the frame is expected to end.
2. Read the STOPF bit of the I2C_ISR register. If it is low, reset the I2C kernel by clearing the PE bit of the I2C_CR1 register.
3. Wait for at least three APB clock cycles before enabling again the I2C peripheral.

2.12 USART

2.12.1 Anticipated end-of-transmission signaling in SPI slave mode

Description

In SPI slave mode, at low USART baud rate with respect to the USART kernel and APB clock frequencies, the *transmission complete* flag TC of the USARTx_ISR register may unduly be set before the last bit is shifted on the transmit line.

This leads to data corruption if, based on this anticipated end-of-transmission signaling, the application disables the peripheral before the last bit is transmitted.

Workaround

Upon the TC flag rise, wait until the clock line remains idle for more than the half of the communication clock cycle. Then only consider the transmission as ended.

2.12.2 Data corruption due to noisy receive line

Description

In all modes, except synchronous slave mode, the received data may be corrupted if a glitch to zero shorter than the half-bit occurs on the receive line within the second half of the stop bit.

Workaround

Apply one of the following measures:

- Either use a noiseless receive line, or
- add a filter to remove the glitches if the receive line is noisy.

2.12.3 Received data may be corrupted upon clearing the ABREN bit

Description

The USART receiver may miss data or receive corrupted data when the auto baud rate feature is disabled by software (ABREN bit cleared in the USART_CR2 register) after an auto baud rate detection, while a reception is ongoing.

Workaround

Do not clear the ABREN bit.

2.12.4 Noise error flag set while ONEBIT is set

Description

When the ONEBIT bit is set in the USART_CR3 register (one sample bit method is used), the noise error (NE) flag must remain cleared. Instead, this flag is set upon noise detection on the START bit.

Workaround

None.

Note: Having noise on the START bit is contradictory with the fact that the one sample bit method is used in a noise free environment.

2.13 LPUART

2.13.1 Possible LPUART transmitter issue when using low BRR[15:0] value

Description

The LPUART transmitter bit length sequence is not reset between consecutive bytes, which could result in a jitter that cannot be handled by the receiver device. As a result, depending on the receiver device bit sampling sequence, a desynchronization between the LPUART transmitter and the receiver device may occur resulting in data corruption on the receiver side.

This happens when the ratio between the LPUART kernel clock and the baud rate programmed in the LPUART_BRR register (BRR[15:0]) is not an integer, and is in the three to four range. A typical example is when the 32.768 kHz clock is used as kernel clock and the baud rate is equal to 9600 baud, resulting in a ratio of 3.41.

Workaround

Apply one of the following measures:

- On the transmitter side, increase the ratio between the LPUART kernel clock and the baud rate. To do so:
  - Increase the LPUART kernel clock frequency, or
  - Decrease the baud rate.
- On the receiver side, generate the baud rate by using a higher frequency and applying oversampling techniques if supported.

2.14 SPI

2.14.1 BSY bit may stay high when SPI is disabled

Description

The BSY flag may remain high upon disabling the SPI while operating in:

- master transmit mode and the TXE flag is low (data register full).
- master receive-only mode (simplex receive or half-duplex bidirectional receive phase) and an SCK strobing edge has not occurred since the transition of the RXNE flag from low to high.
- slave mode and NSS signal is removed during the communication.

Workaround

When the SPI operates in:

- master transmit mode, disable the SPI when TXE = 1 and BSY = 0.
- master receive-only mode, ignore the BSY flag.
- slave mode, do not remove the NSS signal during the communication.

2.14.2 BSY bit may stay high at the end of data transfer in slave mode

Description

BSY flag may sporadically remain high at the end of a data transfer in slave mode. This occurs upon coincidence of internal CPU clock and external SCK clock provided by master.

In such an event, if the software only relies on BSY flag to detect the end of SPI slave data transaction (for example to enter low-power mode or to change data line direction in half-duplex bidirectional mode), the detection fails.

As a conclusion, the BSY flag is unreliable for detecting the end of data transactions.

Workaround

Depending on SPI operating mode, use the following means for detecting the end of transaction:

- When NSS hardware management is applied and NSS signal is provided by master, use NSS flag.
- In SPI receiving mode, use the corresponding RXNE event flag.

- In SPI transmit-only mode, use the BSY flag in conjunction with a timeout expiry event. Set the timeout such as to exceed the expected duration of the last data frame and start it upon TXE event that occurs with the second bit of the last data frame. The end of the transaction corresponds to either the BSY flag becoming low or the timeout expiry, whichever happens first.

Prefer one of the first two measures to the third as they are simpler and less constraining.

Alternatively, apply the following sequence to ensure reliable operation of the BSY flag in SPI transmit mode:

1. Write last data to data register.
2. Poll the TXE flag until it becomes high, which occurs with the second bit of the data frame transfer.
3. Disable SPI by clearing the SPE bit mandatorily before the end of the frame transfer.
4. Poll the BSY bit until it becomes low, which signals the end of transfer.

Note: The alternative method can only be used with relatively fast CPU speeds versus relatively slow SPI clocks or/and long last data frames. The faster is the software execution, the shorter can be the duration of the last data frame.

2.15 FDCAN

2.15.1 Desynchronization under specific condition with edge filtering enabled

Description

FDCAN may desynchronize and incorrectly receive the first bit of the frame if:

- the edge filtering is enabled (the EFBI bit of the FDCAN_CCCR register is set), and
- the end of the integration phase coincides with a falling edge detected on the FDCAN_Rx input pin

If this occurs, the CRC detects that the first bit of the received frame is incorrect, flags the received frame as faulty and responds with an error frame.

Note: This issue does not affect the reception of standard frames.

Workaround

Disable edge filtering or wait for frame retransmission.

2.15.2 Tx FIFO messages inverted under specific buffer usage and priority setting

Description

Two consecutive messages from the Tx FIFO may be inverted in the transmit sequence if:

- FDCAN uses both a dedicated Tx buffer and a Tx FIFO (the TFQM bit of the FDCAN_TXBC register is cleared), and
- the messages contained in the Tx buffer have a higher internal CAN priority than the messages in the Tx FIFO.

Workaround

Apply one of the following measures:

- Ensure that only one Tx FIFO element is pending for transmission at any time:
  The Tx FIFO elements may be filled at any time with messages to be transmitted, but their transmission requests are handled separately. Each time a Tx FIFO transmission has completed and the Tx FIFO gets empty (TFE bit of FDACN_IR set to 1) the next Tx FIFO element is requested.
- Use only a Tx FIFO:
  Send both messages from a Tx FIFO, including the message with the higher priority. This message has to wait until the preceding messages in the Tx FIFO have been sent.
- Use two dedicated Tx buffers (for example, use Tx buffer 4 and 5 instead of the Tx FIFO). The following pseudo-code replaces the function in charge of filling the Tx FIFO:

```
Write message to Tx Buffer 4
Transmit Loop:
    Request Tx Buffer 4 - write AR4 bit in FDCAN_TXBAR
    Write message to Tx Buffer 5
    Wait until transmission of Tx Buffer 4 complete (IR bit in FDCAN_IR),
    read TO4 bit in FDCAN_TXBTO
    Request Tx Buffer 5 - write AR5 bit of FDCAN_TXBAR
    Write message to Tx Buffer 4
    Wait until transmission of Tx Buffer 5 complete (IR bit in FDCAN_IR),
    read TO5 bit in FDCAN_TXBTO
```

2.16 USB

2.16.1 ESOF interrupt timing desynchronized after resume signaling

Description

Upon signaling resume, the device is expected to allow full 3 ms of time to the host or hub for sending the initial SOF (start of frame) packet, without triggering SUSP interrupt. However, the device only allows two full milliseconds and unduly triggers SUSP interrupt if it receives the initial packet within the third millisecond.

Workaround

When the device initiates resume (remote wakeup), mask the SUSP interrupt by setting the SUSPM bit for 3 ms, then unmask it by clearing SUSPM.

2.16.2 Incorrect CRC16 in the memory buffer

Description

Memory buffer locations are written starting from the address contained in the ADDRn_RX for a number of bytes corresponding to the received data packet length, CRC16 inclusive (that is, data payload length plus two bytes), or up to the last allocated memory location defined by BL_SIZE and NUM_BLOCK, whichever comes first. In the former case, the CRC16 checksum is written wrongly, with its least significant byte going to both memory buffer byte locations expected to receive the least and the most significant bytes of the checksum.

Although the checksum written in the memory buffer is wrong, the underlying CRC checking mechanism in the USB peripheral is fully functional.

Workaround

Ignore the CRC16 data in the memory buffer.

2.17 UCPD

2.17.1 TXHRST upon write data underflow corrupting the CRC of the next packet

Description

TXHRST command issued at the instant of detecting write data underflow during a packet transmission can cause a corrupt CRC of the following packet.

Workaround

Use DMA (TXDMAEN) rather than software writing to UCPD_TXDR. Normally, this prevents write data underflow. Should a corrupt CRC event still occur, the DMA transfer method retransmits the packet until the CRC is correct and the packet acknowledged by the receiver.

2.17.2 Ordered set with multiple errors in a single K-code is reported as invalid

Description

The Power Delivery standard allows considering a received ordered set as valid even if it contains errors, provided that they only affect a single K-code of the ordered set.

In the reference manual, the RXSOP3OF4 flag is specified to signal errors affecting a single K-code, the RXERR flag to signal errors in multiple K-codes.

However, the behaviour does not conform with the reference manual. The RXSOP3OF4 flag is only raised in the case of a single error. The RXERR flag is raised in the case of multiple errors, regardless of whether they affect a single K-code or multiple K-codes. As a consequence, ordered sets with multiple errors in a single K-code are reported by the device as invalid although the Power Delivery standard allows considering them as valid.

Despite this non-conformity versus its reference manual, the device remains compliant with the Power Delivery standard.

Workaround

None.

Important security notice

The STMicroelectronics group of companies (ST) places a high value on product security, which is why the ST product(s) identified in this documentation may be certified by various security certification bodies and/or may implement our own security measures as set forth herein. However, no level of security certification and/or built-in security measures can guarantee that ST products are resistant to all forms of attacks. As such, it is the responsibility of each of ST's customers to determine if the level of security provided in an ST product meets the customer needs both in relation to the ST product alone, as well as when combined with other components and/or software for the customer end product or application. In particular, take note that:

- ST products may have been certified by one or more security certification bodies, such as Platform Security Architecture (www.psacertified.org) and/or Security Evaluation standard for IoT Platforms (www.trustcb.com). For details concerning whether the ST product(s) referenced herein have received security certification along with the level and current status of such certification, either visit the relevant certification standards website or go to the relevant product page on www.st.com for the most up to date information. As the status and/or level of security certification for an ST product can change from time to time, customers should re-check security certification status/level as needed. If an ST product is not shown to be certified under a particular security standard, customers should not assume it is certified.
- Certification bodies have the right to evaluate, grant and revoke security certification in relation to ST products. These certification bodies are therefore independently responsible for granting or revoking security certification for an ST product, and ST does not take any responsibility for mistakes, evaluations, assessments, testing, or other activity carried out by the certification body with respect to any ST product.
- Industry-based cryptographic algorithms (such as AES, DES, or MD5) and other open standard technologies which may be used in conjunction with an ST product are based on standards which were not developed by ST. ST does not take responsibility for any flaws in such cryptographic algorithms or open technologies or for any methods which have been or may be developed to bypass, decrypt or crack such algorithms or technologies.
- While robust security testing may be done, no level of certification can absolutely guarantee protections against all attacks, including, for example, against advanced attacks which have not been tested for, against new or unidentified forms of attack, or against any form of attack when using an ST product outside of its specification or intended use, or in conjunction with other components or software which are used by customer to create their end product or application. ST is not responsible for resistance against such attacks. As such, regardless of the incorporated security features and/or any information or support that may be provided by ST, each customer is solely responsible for determining if the level of attacks tested for meets their needs, both in relation to the ST product alone and when incorporated into a customer end product or application.
- All security features of ST products (inclusive of any hardware, software, documentation, and the like), including but not limited to any enhanced security features added by ST, are provided on an "AS IS" BASIS. AS SUCH, TO THE EXTENT PERMITTED BY APPLICABLE LAW, ST DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, unless the applicable written and signed contract terms specifically provide otherwise.

Revision history

Table 4. **Document revision history**

Date	Version	Changes
28-Jul-2020	1	Initial release.
01-Sept-2020	2	Removed erratum *VREF+ output voltage disturbed by some GPIOs toggling*.
05-Nov-2020	3	Updated document scope to include device revision Z. All tables in the document updated accordingly. Added errata: • New context conversion initiated without waiting for trigger when writing new context in ADC_JSQR with JQDIS = 0 and JQM = 0 • Two consecutive context conversions fail when writing new context in ADC_JSQR just after previous context completion with JQDIS = 0 and JQM = 0 • Unexpected regular conversion when two consecutive injected conversions are performed in Dual interleaved mode • ADC_AWDy_OUT reset by non-guarded channels • LPTIM events and PWM output are delayed by one kernel clock cycle Modified erratum DMA disable failure and error flag omission upon simultaneous transfer error and global flag clear.
16-Apr-2021	4	Added erratum OPAMP disturbed by fast-edge toggle on GPIO pin corresponding to VINM0
06-Mar-2023	5	Added errata: • MCU cannot enter in Standby mode when HSE bypass used • Corrupted content of the backup domain due to a missed power-on reset after this domain supply voltage drop • SRAM write error • Wrong instruction fetches from flash memory upon wakeup from Sleep or Stop mode when debug in low-power mode is enabled • An ADC instance may impact the accuracy of another ADC instance at specific conditions Modified errata: • Full JTAG configuration without NJTRST pin cannot be used • Device may remain stuck in LPTIM interrupt when clearing event flag ADC errata re-ordered. Added Section Important security notice.
24-Jun-2024	6	Added errata: • LSE crystal oscillator may be disturbed by transitions on PC13 • Device cannot enter Standby mode when HSE bypass used as system clock • Bidirectional break mode not working with short pulses • SDA held low upon SMBus timeout expiry in slave mode • Received data may be corrupted upon clearing the ABREN bit • Noise error flag set while ONEBIT is set • Possible LPUART transmitter issue when using low BRR[15:0] value • TXHRST upon write data underflow corrupting the CRC of the next packet • Ordered set with multiple errors in a single K-code is reported as invalid Modified errata: • Device may remain stuck in LPTIM interrupt when entering Stop mode • Data corruption due to noisy receive line

Contents