STM32MP131x STM32MP133x STM32MP135x

Errata sheet

STM32MP131x/3x/5x device errata

Applicability

This document applies to the part numbers of STM32MP131x/3x/5x 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 RM0475.

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
STM32MP131x	STM32MP131A, STM32MP131C, STM32MP131D, STM32MP131F
STM32MP133x	STM32MP133A, STM32MP133C, STM32MP133D, STM32MP133F
STM32MP135x	STM32MP135A, STM32MP135C, STM32MP135D, STM32MP135F

Table 2. Device variants

Reference	Silicon revision codes	
	Device marking(1)	REV_ID(2)
STM32MP13xx	Y	0x1003

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_IDC register.*

1 Summary of device errata

The following table gives a quick reference to the STM32MP131x/3x/5x 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. Z	Rev. Y
Core	2.1.1	Memory locations might be accessed speculatively due to instruction fetches when HCR.VM is set	A	A
	2.1.2	Cache maintenance by set/way operations can execute out of order	A	A
	2.1.3	PMU events 0x07, 0x0C, and 0x0E do not increment correctly	N	N
	2.1.4	PMU event counter 0x14 does not increment correctly	A	A
	2.1.5	Exception mask bits are cleared when an exception is taken in Hyp mode	N	N
System	2.2.1	TPIU fails to output sync after the pattern generator is disabled in Normal mode	A	A
	2.2.2	Incorrect reset of glitch-free kernel clock switch	P	P
	2.2.3	SAES, RNG, PKA stuck after first stage bootloader (FSBL) decryption	A	A
DMAMUX	2.3.1	SOFx not asserted when writing into DMAMUX_CFR register	N	N
	2.3.2	OFx not asserted for trigger event coinciding with last DMAMUX request	N	N
	2.3.3	OFx not asserted when writing into DMAMUX_RGCFR register	N	N
	2.3.4	Wrong input DMA request routed upon specific DMAMUX_CxCR register write coinciding with synchronization event	A	A
FMC	2.4.1	NOR Flash memory/PSRAM incorrect bus turnaround timing	A	A
	2.4.2	Incorrect FMC_CLK clock period when CLKDIV value is changed on-the-fly in Continuous clock mode	A	A
	2.4.3	NAND Flash memory IREF/IFEF flags wrongly asserted just after enabling in FMC_IER	A	A
	2.4.4	Command sequencer accesses NAND Flash memory device while PBKEN bit is cleared in FMC_PCR	A	A
	2.4.5	NAND Flash memory IREF flag wrongly asserted after reset	A	A
	2.4.6	NAND ECC corrupted due to insufficient ECCEN low period in between sectors	A	A
QUADSPI	2.5.1	Memory-mapped read of last memory byte fails	P	P
ADC	2.6.1	Injected queue of context is not available if JQM = 0	N	N
	2.6.2	Sampling time shortened in JAUTO auto delayed mode	A	A
	2.6.3	Load multiple not supported by ADC interface	A	A
	2.6.4	Overrun flag might not be set when converted data have not been read before new data are written	A	A

Function	Section	Limitation	Status	
			Rev. Z	Rev. Y
ADC	2.6.5	New context conversion initiated without waiting for trigger when writing new context in ADC_JSQR with JQDIS = 0 and JQM = 0	A	A
	2.6.6	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
	2.6.7	Unexpected regular conversion when two consecutive injected conversions are performed in Dual interleaved mode	A	A
	2.6.8	ADC_AWDy_OUT reset by non-guarded channels	A	A
	2.6.9	Injected data stored in the wrong ADC_JDRx registers	A	A
	2.6.10	ADC slave data may be shifted in Dual regular simultaneous mode	A	A
DTS	2.7.1	DTS incorrect operation with LSE as reference clock and PCLK enabled	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	Output compare clear not working with external counter reset	P	P
	2.8.4	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 1 kernel clock cycle	P	P
RTC and TAMP	2.10.1	Alarm flag may be repeatedly set when the core is stopped in debug	N	N
	2.10.2	Binary mode: SSR is not reloaded with 0xFFFF FFFF when SSCLR = 1	A	A
I2C	2.11.1	Wakeup frame may not wake up the MCU from Stop mode if tHD;STA is close to I2C kernel clock startup time	P	P
	2.11.2	Wrong data sampling when data setup time (tSU;DAT) is shorter than one I2C kernel clock period	P	P
	2.11.3	Spurious bus error detection in master mode	A	A
	2.11.4	OVR flag not set in underrun condition	N	N
	2.11.5	Transmission stalled after first byte transfer	A	A
	2.11.6	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	N	N
	2.12.3	USART does not generate DMA requests after setting/clearing DMAT bit	A	-
	2.12.4	Noise error flag set while ONEBIT is set	N	N
SPI	2.13.1	Master data transfer stall at system clock much faster than SCK	A	A
	2.13.2	Corrupted CRC return at non-zero UDRDET setting	P	P
	2.13.3	TXP interrupt occurring while SPI disabled	A	A
	2.13.4	Possible corruption of last-received data depending on CRCSIZE setting	A	A
	2.13.5	Truncation of SPI output signals after EOT event	A	A
FDCAN	2.14.1	Desynchronization under specific condition with edge filtering enabled	A	A
	2.14.2	Tx FIFO messages inverted under specific buffer usage and priority setting	A	A
	2.14.3	DAR mode transmission failure due to lost arbitration	A	A

Function	Section	Limitation	Status	
			Rev. Z	Rev. Y
OTG_HS	2.15.1	Host packet transmission may hang when connecting the full speed interface through a hub to a low-speed device	N	N
ETH	2.16.1	Incorrect L4 inverse filtering results for corrupted packets	N	N
	2.16.2	Rx DMA may fail to recover upon DMA restart following a bus error, with Rx timestamping enabled	A	A
	2.16.3	Spurious receive watchdog timeout interrupt	A	A
	2.16.4	Incorrect flexible PPS output interval under specific conditions	A	A
	2.16.5	Packets dropped in RMII 10Mbps mode due to fake dribble and CRC error	A	A
	2.16.6	ARP offload function not effective	A	A
	2.16.7	Spurious checksum error upon MTL pending Tx queue flush	N	N
	2.16.8	Bus error coinciding with start-of-packet corrupts MAC-generated packet transmission	N	N
	2.16.9	DMA spurious state upon AXI DMA slave bus error	P	P

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®-A7 core revision r0p5-REVIDR=0x01 is available from http://infocenter.arm.com.

2.1.1 Memory locations might be accessed speculatively due to instruction fetches when HCR.VM is set

This limitation is registered under Arm ID number 844169 and classified into “Category B”. Its impact to the device is minor.

Description

The ARMv7 architecture requires that when all associated stages of translation are disabled for the current privilege level, memory locations are only accessed due to instruction fetches within the same or next 4 KB region as an instruction which has been or is going to be fetched due to sequential execution. In the conditions detailed below, the Cortex-A7 processor might access other locations speculatively due to instruction fetches.

The unwanted speculative instruction fetches may occur when the following conditions are all met:

1. The processor must be executing at PL2 or Secure PL1.
2. Address translation is disabled for the current exception level (by clearing the appropriate SCTLR.M or HSCTLR.M bit).
3. The HCR.VM bit is set.

As the HCR.VM is reset low, this issue does not manifest during boot code.

Workaround

This issue is most likely to arise in powerdown code, if PL2 or secure PL1 software disables address translation before the core is powered down.

To work around this erratum, software should ensure that HCR.VM is cleared before disabling address translation at PL2 or Secure PL1.

2.1.2 Cache maintenance by set/way operations can execute out of order

This limitation is registered under Arm ID number 814220 and classified into “Category B”. Its impact to the device is limited.

Description

The ARM v7 architecture states that all cache and branch predictor maintenance operations that do not specify an address execute in program order, relative to each other. However, because of this erratum, an L2 set/way cache maintenance operation can overtake an L1 set/way cache maintenance operation.

Code that intends to clean dirty data from L1 to L2 and then from L2 to L3 using set/way operations might not behave as expected. The L2 to L3 operation might happen first and result in dirty data remaining in L2 after the L1 to L2 operation has completed.

If dirty data remains in L2 then an external agent, such as a DMA agent, might observe stale data.

If the processor is reset or powered-down while dirty data remains in L2 then the dirty data are lost.

The failure occurs when the following conditions are all met:

1. A CPU performs an L1 DCCSW or DCCISW operation.
2. The targeted L1 set/way contains dirty data.
3. Before the next DSB, the same CPU executes an L2 DCCSW or DCCISW operation while the L1 set/way operation is in progress.
4. The targeted L2 set/way is within the group of L2 set/ways that the dirty data from L1 can be allocated to.

If the above conditions are met then the L2 set/way operation can take effect before the dirty data from L1 has been written to L2.

Note: Conditions (3) and (4) are not likely to be met concurrently when performing set/way operations on the entire L1 and L2 caches. This is because cache maintenance code is likely to iterate through sets and ways in a consistent ascending or consistent descending manner across cache levels, and to perform all operations on one cache level before moving on to the next cache level. This means that, for example, cache maintenance operations on L1 set 0 and L2 set 0 are separated by cache maintenance operations for all other sets in the L1 cache. This creates a large window for the cache maintenance operations on L1 set 0 to complete.

Workaround

Correct ordering between set/way cache maintenance operations can be forced by executing a DSB before changing cache levels.

2.1.3 PMU events 0x07, 0x0C, and 0x0E do not increment correctly

This limitation is registered under Arm ID number 809719 and classified into “Category C”. Its impact to the device is minor.

Description

The Cortex-A7 processor implements version 2 of the Performance Monitor Unit architecture (PMUv2). The PMU can gather statistics on the operation of the processor and memory system during runtime. This event information can be used when debugging or profiling code.

The PMU can be programmed to count architecturally executed stores (event 0x07), software changes of the PC (event 0x0C), and procedure returns (event 0x0E). However, because of this erratum, these events do not fully adhere to the descriptions in the PMUv2 architecture.

As a result of this limitation, the information returned by PMU counters that are programmed to count events 0x07, 0x0C, or 0x0E might be misleading when debugging or profiling code executed on the processor.

The error occurs when the following conditions are met:

Either:

1. A PMU counter is enabled and programmed to count event 0x07. That is: instruction architecturally executed, condition code check pass, store.
2. A PLDW instruction is executed.

If the above conditions are met, the PMUv2 architecture specifies that the counter for event 0x07 does not increment. However, the counter does increment.

Or:

1. A PMU counter is enabled and programmed to count event 0x0C. That is: instruction architecturally executed, condition code check pass, software change of the PC.
2. An SVC, HVC, or SMC instruction is executed.

If the above conditions are met, the PMUv2 architecture specifies that the counter for event 0x0C increments. However, the counter does not increment.

Or:

1. A PMU counter is enabled and programmed to count event 0x0E. That is: instruction architecturally executed, condition code check pass, procedure return.
2. One of the following instructions is executed:
   a. `MOV PC, LR`
   b. `ThumbEE LDMIA R9!, {?, PC}`
   c. `ThumbEE LDR PC, [R9], #offset`
   d. `BX Rm, where Rm != R14`
   e. `LDM SP, {?, PC}`

If the above conditions are met, the PMUv2 architecture specifies that the counter for event 0x0E increments for (a), (b), (c) and does not increment for (d) and (e). However, the counter does not increment for (a), (b), (c) and increments for (d) and (e).

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 is 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 is pended again once the handler for B has completed and the handler for C is executed.

As the STM32 interrupt C is level based, then this interrupt eventually becomes re-pending and subsequently be handled.

Workaround

None.

2.1.4 PMU event counter 0x14 does not increment correctly

This limitation is registered under Arm ID number 805420 and classified into “Category C”. Its impact to the device is minor.

Description

The Cortex-A7 MPCore processor implements version 2 of the Performance Monitor Unit architecture (PMUv2). The PMU can gather statistics on the operation of the processor and memory system during runtime. This event information can be used when debugging or profiling code. When a PMU counter is programmed to count L1 instruction cache accesses (event 0x14), the counter should increment on all L1 instruction cache accesses.

This limitation has the following implications:

1. If SCR.{AW, FW} is set to 0 then the clearing of corresponding bit CPSR.{A, F} to 0 has no effect. The value of CPSR.{A, F} is ignored.
2. A PMU counter is enabled and programmed to count L1 instruction cache accesses (event 0x14).
3. Cacheable instruction fetches miss in the L1 instruction cache.

If the above conditions are met, the event counter does not increment.

Workaround

To obtain a better approximation for the number of L1 instruction cache accesses, enable a second PMU counter and program it to count instruction fetches that cause linefills (event 0x01). Add the value returned by this counter to the value returned by the L1 instruction access counter (event 0x14). The result of the addition is a better indication of the number of L1 instruction cache accesses.

2.1.5 Exception mask bits are cleared when an exception is taken in Hyp mode

This limitation is registered under Arm ID number 804069 and classified into “Category C”. Its impact to the device is minor.

Description

The Cortex-A7 processor implements the ARM Virtualization Extensions and the ARM Security Extensions. Exceptions can be routed to Monitor mode by setting SCR.{EA, FIQ, IRQ} to 1. Exceptions can be masked by setting corresponding bit CPSR.{A, I, F} to 1.

The ARMv7-A architecture states that an exception taken in Hyp mode does not change the value of the mask bits for exceptions routed to Monitor mode. However, because of this erratum, the corresponding mask bits are cleared to 0.

The error occurs when the following conditions are met:

1. One or more exception types are routed to Monitor mode by setting one or more of SCR.{EA, FIQ, IRQ} to 1.
2. The corresponding exception types are masked by setting the corresponding CPSR.{A, F, I} bits to 1.
3. Any exception is taken in Hyp mode.

If the above conditions are met then the exception mask bit CPSR.{A, F, I} is cleared to 0 for each exception type that meets conditions (1) and (2). The affected mask bits are cleared together regardless of the exception type in condition (3).

The implications of this erratum are:

- If SCR.{AW, FW} is set to 0 then the clearing of corresponding bit CPSR.{A, F} to 0 has no effect. The value of CPSR.{A, F} is ignored.
- Otherwise, when CPSR.{A, F, I} is set to 1, Secure code cannot rely on CPSR.{A, F, I} remaining set to 1. An exception that should be masked might be routed to Monitor mode.

The impact of this limitation is considered to be minor as it is expected that the users:

1. set SCR.{AW, FW} to 0 when SCR.{EA, FIQ} is set to 1.
2. set SCR.IRQ to 0.

Workaround

None.

2.2 System

2.2.1 TPIU fails to output sync after the pattern generator is disabled in Normal mode

Description

The TPIU includes a pattern generator that can be used by external tool to determine the operating behavior of the trace port and timing characteristics. This pattern generator includes a mode that transmits the test pattern for a specified number of cycles, and then reverts to transmitting normal trace data.

When the TPIU is configured to operate in Normal mode (FFCR.EnFCont=0), the synchronization sequence that is required between the test pattern and the trace data is not generated. Synchronization is generated at a later time, as determined by the synchronization counter.

Conditions:

The following conditions must all occur:

- The TPIU is configured in normal mode, `FFCR.EnFCont==0`
- The TPIU is configured with the formatter enabled, `FFCR.EnFTC==1`
- The pattern generator is enabled in timed mode, `Current_test_pattern_mode.PTIMEEN==1`

Implications:

The timed mode of the TPIU is intended to permit the TPIU to transition between an initial synchronization sequence using the pattern generator and functional mode without any further programming intervention. If the synchronization sequence is not generated at the end of the test pattern, the trace port analyzer is unlikely to be able to capture the start of the trace stream correctly. Synchronization is correctly inserted based on the value configured in the FSCR, once the specified number of frames of trace data have been output.

Workaround

Workaround requires software interaction to detect the completion of the test pattern sequence. In addition, any trace data present at the input to the TPIU is lost whilst the pattern generator is active. Any trace data present in the input to the TPIU before the formatter is re-enabled (and synchronization generated) is not de-compressible.

1. After enabling the pattern generator, set `FFCR.StopOnFl==1` and `FFCR.FOnMan==1`.
2. Poll `FFSR.FtStopped` until 1 is read
3. Set `FFCR.EnFTC==1`

2.2.2 Incorrect reset of glitch-free kernel clock switch

Description

The activation of NRST (by external signal, internal watchdog or software) may not properly reset the glitch-free clock switches inside the RCC.

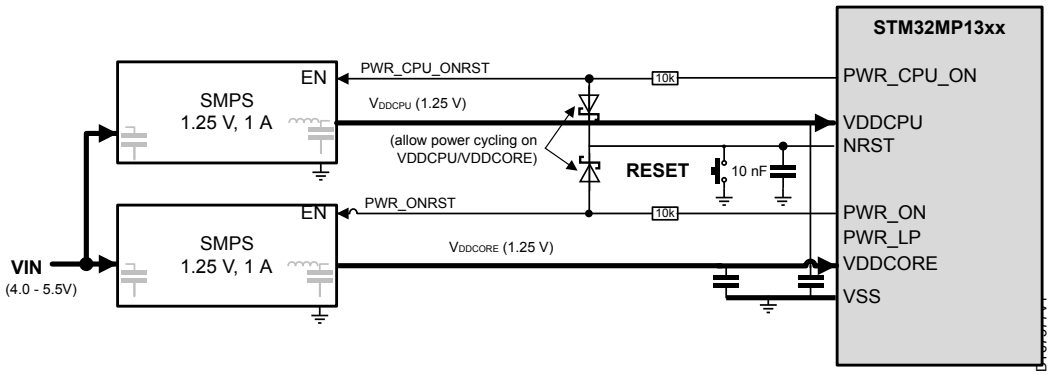
As a consequence, the peripheral kernel clock previously used by the application remains selected. If not enabled again (by default or by software), there is no kernel clock present on the related peripheral and the BootROM hangs.

Note: The USB boot is usually not affected as the application always uses the same clocking scheme depending on the HSE crystal frequency choice. For example, there is no issue for HSE at 24 MHz as hse_ker_ck is used for clocking OTG. Other peripherals, such as LPTIM, USART, and I2C, work without care if the previous application clock is enabled again to ensure that the clock switch is not stuck.

Workaround

Apply one of the following measures:

1. **By hardware,** ensure that the V_{DDCORE} and V_{DDCPU} logic is reset on NRST activation:
   - With STPMIC1
     By default, a power cycle on V_{DDCORE} and V_{DDCPU} (and V_{DD}) occurs upon activating NRST.
   - Without STPMIC1
     Connect the POWER_CPU_ON and PWR_ON signals to the external SMPS enable inputs as illustrated next.



   Activating NRST then causes the V_{DDCPU} and V_{DDCORE} supply voltages to cycle.

   The drawback is that, the debug logic also being reset, it is not possible to debug the boot chain (TF-A and U-Boot) as any breakpoints set are lost during the power cycle.

2. **By software:**
   - For interfaces required during boot, whether Flash memory peripherals (SDMMC1/2, QUADSPI, or FMC) or USART/UART (USART3/6 or UART4/5/7/8), use the same clock as the one used during the BootROM execution:
     - If BOOT[2:0] = 000 or 110 (UART boot), set UARTxxSRC[2:0] to 0 (pclk) or 2 (hsi_ker_ck), for all UART instances not disabled via uart_intances_disabled bitfield of the BSEC OTP WORD 3.
     - If BOOT[2:0] = 001 or 111 (QUADSPI boot), set QSPISRC[2:0] to 0 (aclk) or 3 (per_ck).
     - If BOOT[2:0] = 010 or 101 (SDMMC boot), set SDMMC1SRC[2:0] or SDMMC2SRC[2:0] to 0 (hclk6) or 3 (hsi_ker_ck).
   - For SD card, the use of HSI (64 MHz) causes raw bandwidth performance penalty as only 32 or 64 MHz could be used instead of respectively 50 MHz (SDR25/DDR50) and 100 MHz (SDR50)
   - For eMMC, the use of HSI (64 MHz) causes raw bandwidth performance penalty as only 32 or 64 MHz could be used instead of respectively 52 MHz (SDR/DDR) or over 100 MHz (HS200). Note that hclk6/hclk5/aclk are limited to 200 MHz when hclk6 is used as SDMMC1/SDMMC2 kernel clock
     - If BOOT[2:0] = 011 (FMC boot), set FMCSRC[2:0] to 0 (aclk) or 3 (per_ck)

2.2.3 SAES, RNG, PKA stuck after first stage bootloader (FSBL) decryption

Description

When an encrypted first stage bootloader (FSBL) is used, after decryption of the FSBL, the SAES is deactivated which prevents the usage of SAES, RNG or PKA.

Workaround

When OpenSTLinux TF-A is used, the enabling of SAES is handled within TF-A.

When an other FSBL is used the SAES clock should be enabled in the customer software, even if SAES is not used, in order to be able to use RNG or PKA.

2.3 DMAMUX

2.3.1 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.

2.3.2 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.

2.3.3 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.

2.3.4 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.

2.4 FMC

2.4.1 NOR Flash memory/PSRAM incorrect bus turnaround timing

Description

The delays between consecutive device accesses, programmed through the BUSTURN[3:0] bitfield of the FMC_BTRx and FMC_BWTRx registers, have no effect. Instead systematic delays are applied:

- t_{IDLE2}: 2 * fmc_ker_ck cycles between accesses to two NOR/PSRAM devices
- t_{IDLE1}: 1 * fmc_ker_ck cycle between accesses to a NOR/PSRAM and a NAND Flash memory

Workaround

Extend the bus turnaround delays to satisfy bus turnaround constraints. Three cases need to be considered:

1. Two consecutive accesses to non-multiplexed NOR/PSRAM devices:
   Program t_{ADDSET} (NOR/PSRAM address setup phase) as needed (see Figure 1).
2. Access to a non-multiplexed NOR/PSRAM followed by an access either to a NOR/PSRAM device with multiplexed A/DQ signals or to a synchronous device:
   Decrease the FMC kernel clock frequency in order to meet the timing constraints (see Figure 2).
3. Access to a non-multiplexed NOR/PSRAM followed by an access to a NAND Flash memory device
   Program t_{MEMSET} (NOR/PSRAM address setup phase) as needed (see Figure 3).

Figure 1. **Bus turn timing recovery - asynchronous accesses to NOR/PSRAM devices (case 1)**

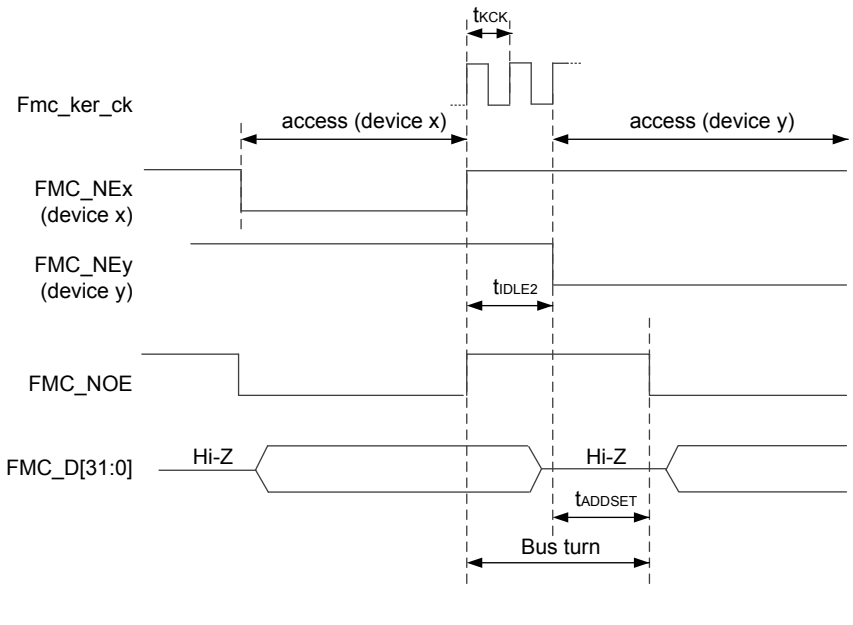


Figure 2. **Bus turn timing recovery - access to NOR/PSRAM followed by access to multiplexed A/DQ synchronous device (case 2)**

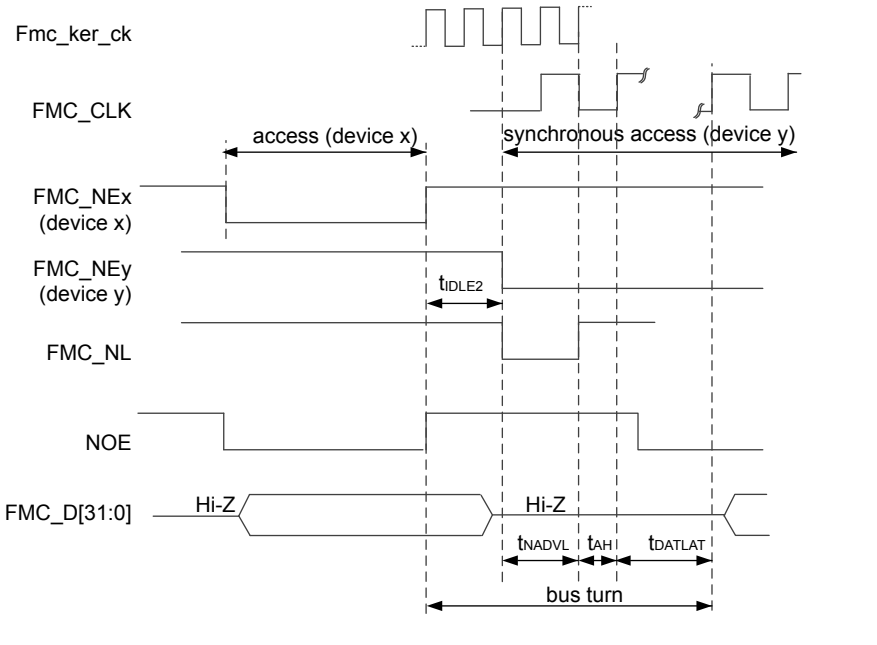


Figure 3. Bus turn timing recovery - access to NOR/PSRAM followed by access to NAND Flash device (case 3)

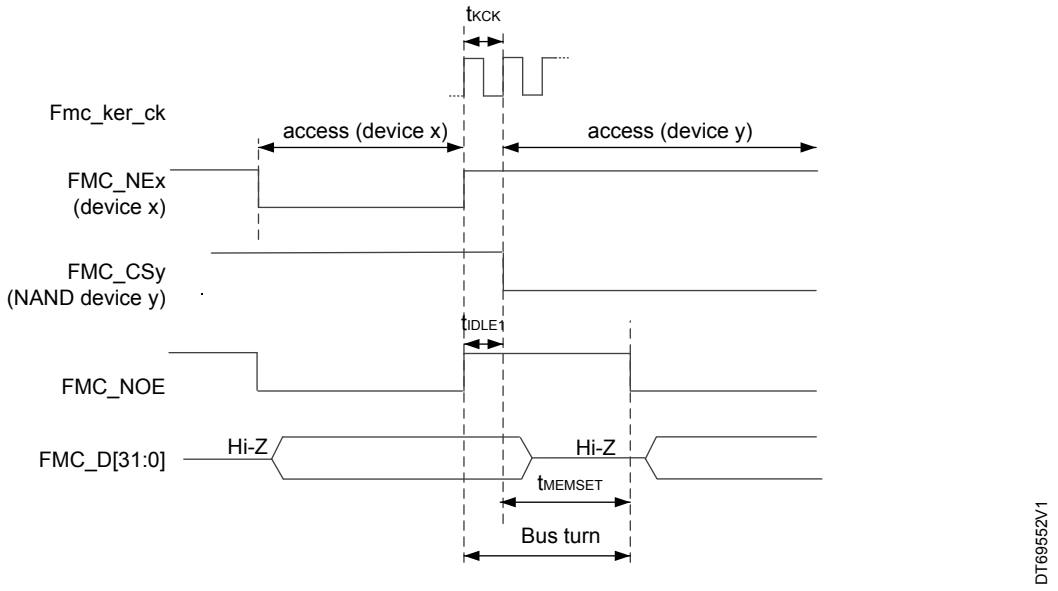


Table 4 gives the internal latencies that are not mentioned in the product datasheets. Refer to the datasheets for more details about the others timing values.

Table 4. Timing parameter description

Parameter	Description	Minimum value
t_{KCK}	FMC kernel clock period	See product datasheet
t_{IDLE1}	NEx high to CSy low (switching to a NAND Flash memory device)	1 * t_{KCK}
t_{IDLE2}	NEx high to NEy low (NOR/PSRAM devices)	2 * t_{KCK}
t_{ADDSET}	NOR/PSRAM address setup phase	See Ref. Man.
t_{NADVL}	Address valid low pulse duration in synchronous mode	(CLKDIV+1) * t_{KCK}
t_{AH}	Address hold in synchronous mode	((CLKDIV+1) * t_{KCK}) / 2
t_{DATLAT}	NOR/PSRAM data latency in synchronous mode	See product datasheet
$t_{MEMxSET}$	NAND Flash memory address setup phase	See product datasheet

2.4.2 Incorrect FMC_CLK clock period when CLKDIV value is changed on-the-fly in Continuous clock mode

Description

When the FMC operates in Continuous clock mode (CCLKEN is set in FMC_BCRx register), a new clock division factor is applied by changing the value of CLKDIV[3:0] in FMC_CFGR while the FMC is disabled (FMCEN cleared in FMC_BCRx), there is one FMC_CLK clock cycle during which the FMC_CLK period is not as expected: for example the clock low pulse duration matches the previous CLKDIV[3:0] value whereas the clock high pulse duration matches the new CLKDIV[3:0] value.

Workaround

Use the following sequence:

1. Stop the memory traffic for all devices.
2. Disable the FMC (refer to the disabling sequence described in the product reference manual).
3. Change CCLKEN for 1 to 0 in the FMC_BCRx register to stop the clock generation.
4. Program the desired CLKDIV[3:0] value in the FMC_CFGR register.
5. Change back CCLKEN from 0 to 1.
6. Enable the FMC.

2.4.3 NAND Flash memory IREF/IFEF flags wrongly asserted just after enabling in FMC_IER

Description

When the application enables interrupt rising edge/falling edge detection (IREE/IFEE set in FMC_IER register) while the corresponding IREF/IFEF status flag is already set in FMC_ISR register due to a previous event, then a spurious interrupt is immediately generated, corresponding to an out-of-date event.

Workaround

Clear the FMC_ISR flags before programming FMC_IER register.

2.4.4 Command sequencer accesses NAND Flash memory device while PBKEN bit is cleared in FMC_PCR

Description

When the PBKEN bit is cleared in FMC_PCR register, the FMC should discard all accesses from the system to an external NAND Flash memory device. However, the command sequencer can access NAND memory devices if it is accidentally enabled by setting CSQSTART bit in FMC_CSQCR register.

Please note that AXI commands and data are discarded as expected (returning a bus error)

Workaround

Deactivate the NAND Flash command sequencer to avoid unwanted accesses to NAND Flash memory devices when PBKEN is dynamically cleared during application execution.

2.4.5 NAND Flash memory IREF flag wrongly asserted after reset

Description

The FMC NAND Flash memory Ready/Busy input signal (RNB) is asserted when the FMC is under reset. If RNB remains high when the FMC reset is released., the IREF status flag is set in the FMC status register (FMC_SR) thus triggering a spurious interrupt.

Workaround

Clear the FMC status register (FMC_SR) after the FMC reset release.

2.4.6 NAND ECC corrupted due to insufficient ECCEN low period in between sectors

Description

When the FMC processes multiple NAND Flash memory sectors with Hamming ECC computation, the ECCEN bit of the FMC_PCR register is cleared after the current sector and set back before the next sector. The computed ECC may be corrupted if the ECCEN bit low period in between sectors is too short to be sampled correctly.

Workaround

By software, ensure that between sectors, the ECCEN bit remains low for at least 1.5 kernel clock periods.

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 Injected queue of context is not available if JQM = 0

Description

The queue mechanism is not functional when JQM bit of ADC_CFGR is cleared to 0. The effective queue length is equal to 1 stage: a new context written before the previous context's consumption leads to a queue overflow and is ignored. Consequently, the ADC must be stopped before programming the ADC_JSQR register.

Workaround

None.

2.6.2 Sampling time shortened in JAUTO auto delayed mode

Description

When the ADC is configured in JAUTO single conversion mode (CONT=0), with auto delayed mode enabled (AUTDLY = 1), if the last regular conversion is read and a new regular trigger arrives before the JEOS bit is cleared, the first regular conversion sampling time is shortened by 1 cycle.

This does not apply for configuration where SMP = 000 (1.5 cycle sampling time), or if the interval between triggers is always above the auto-injected sequence conversion period.

Workaround

The sampling time can be increased by 1 clock cycle if the situation is foreseen.

2.6.3 Load multiple not supported by ADC interface

Description

The ADC interface does not support LDM, STM, LDRD and STRD instructions for successive multiple-data read and write accesses to a contiguous address block.

Workaround

The workaround consists in preventing compilers from generating LDM, STM, LDRD and STRD instructions.

2.6.4 Overrun flag might not be set when converted data have not been read before new data are written

Description

When converted data are read from the ADC_DR register during the very same APB cycle used to write data from a new conversion, the previously written data or the new data are lost but the overrun flag (OVR) may not be set to ‘1’.

Workaround

Read the converted data before data from a new conversion are available, to avoid overrun errors.

2.6.5 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.

2.6.6 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.

2.6.7 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.

2.6.8 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.

2.6.9 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).

2.6.10 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.

2.7 DTS

2.7.1 DTS incorrect operation with LSE as reference clock and PCLK enabled

Description

The DTS does not operate correctly when LSE is selected as reference clock (the REFCLK_SEL bit of the DTS_CFGR1 register set) and the PCLK is enabled.

Workaround

Only use the DTS with the PCLK selected as reference clock (REFCLK_SEL cleared).

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 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).

2.8.4 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 RCC_APByRSTRz 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.

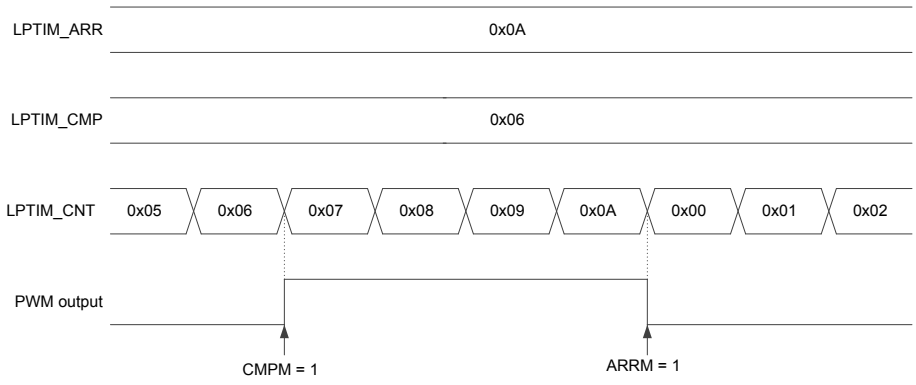
2.9.3 LPTIM events and PWM output are delayed by 1 kernel clock cycle

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:



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 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.

2.10.2 Binary mode: SSR is not reloaded with 0xFFFF FFFF when SSCLR = 1

Description

When SSCLR bit of the RTC_ALRMxSSR register is set when in binary mode, SSR is reloaded with 0xFFFF FFFF at the end of the ck_apre cycle when RTC_SSR is set to RTC_ALRxBINR (*x* stands for either A or B)

RTC_SSR is not reloaded with 0xFFFF FFFF if RTC_ALRxBINR is modified while RTC_SSR is set to RTC_ALRxBINR. Rather, SSR continues to decrement.

Workaround

The workarounds are described for alarm A, and can be applied in the same manner for alarm B. Two workarounds are proposed, the second one requires to use the second alarm.

- Wait for one ck_apre cycle after an alarm A event before changing the RTC_ALRABINR register value.
- Do not reprogram RTC_ALRABINR following the alarm A event itself. Instead, use alarm B configured with RTC_ALRBBINR set to 0xFFFF FFFF, and reprogram RTC_ALRABINR after the alarm B event. This ensures that one ck_apre cycle elapses following the alarm A event.

2.11 I2C

2.11.1 Wakeup frame may not wake up the MCU from Stop mode if $t_{HD;STA}$ is close to I2C kernel clock startup time

Description

Under specific conditions and if the START condition hold time $t_{HD;STA}$ is very close to the startup time of the internal oscillator selected for I2C kernel clock, I2C is not able to detect the address match and, as a consequence, to wake up the MCU from Stop mode.

The failure described occurs when one of the following conditions is met:

1. Timeout detection is enabled (TIMOUTEN = 1 or TEXTEN = 1) and the frame before the wakeup frame is finished abnormally due to I2C timeout detection (TIMOUT = 1).
2. Slave arbitration is lost during the frame preceding the wakeup frame (ARLO = 1).
3. The MCU enters Stop mode while another slave is addressed, after the address phase and before STOP condition (BUSY = 1).
4. The MCU is in Stop mode and another slave is addressed before the MCU itself is addressed.

Note: The conditions 2, 3 and 4 can only occur in a multi-slave network.

In Stop mode, the internal oscillator selected for I2C kernel clock is switched on by I2C when START condition is detected. The I2C kernel clock is then used to receive the address. The internal oscillator is switched off upon the address receipt if the address received does not match the own slave address. If one of the conditions listed is met and if the SCL falling edge following the START condition occurs within the first cycle of the I2C kernel clock, the address is received incorrectly and the address match wakeup interrupt is not generated.

Workaround

None at MCU level.

Upon non-acknowledge by the MCU of a wakeup frame, the I^{2}C-bus master with programmable START condition hold time can set that hold time such that it exceeds one MCU internal oscillator period, then resend the wakeup frame.

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

Description

The I^{2}C-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^{2}C-bus SDA line when $t_{SU;DAT}$ is smaller than one I2C kernel clock (I^{2}C-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^{2}C-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.

2.11.3 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^{2}C-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.

2.11.4 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 I2C 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.

2.11.5 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.

2.11.6 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 UART mode with oversampling by 8 or 16 and with 1 or 2 stop bits, 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

None.

2.12.3 USART does not generate DMA requests after setting/clearing DMAT bit

Description

If the DMA is used for data transmission (DMAT = 1 in USART_CR3 register), and the software clears DMAT bit and sets it again to prepare the next transmission, then the peripheral does not generate DMA requests anymore. As a result, data are not transmitted.

Workaround

- Avoid clearing DMAT.
- If clearing DMAT is needed after the end of DMA transfers, once DMAT is cleared, disable and reenable the peripheral through UE bit of USART_CR1 register. This workaround is acceptable only if the peripheral is not used in receiver mode.
- DMAT can be cleared if the next transmission is based on polling/interrupt.

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 SPI

2.13.1 Master data transfer stall at system clock much faster than SCK

Description

With the system clock (spi_pclk) substantially faster than SCK (spi_ker_ck divided by a prescaler), SPI master data transfer can stall upon setting the CSTART bit within one SCK cycle after the EOT event (EOT flag raise) signaling the end of the previous transfer.

Workaround

Apply one of the following measures:

- Disable then enable SPI after each EOT event.
- Upon EOT event, wait for at least one SCK cycle before setting CSTART.
- Prevent EOT events from occurring, by setting transfer size to undefined (TSIZE = 0) and by triggering transmission exclusively by TXFIFO writes.

2.13.2 Corrupted CRC return at non-zero UDRDET setting

Description

With non-zero setting of UDRDET[1:0] bitfield, the SPI slave can transmit the first bit of CRC pattern corrupted, coming wrongly from the UDRCFG register instead of SPI_TXCRC. All other CRC bits come from the SPI_TXCRC register, as expected.

Workaround

Keep TXFIFO non-empty at the end of transfer.

2.13.3 TXP interrupt occurring while SPI disabled

Description

SPI peripheral is set to its default state when disabled (SPE = 0). This flushes the FIFO buffers and resets their occupancy flags. TXP and TXC flags become set (the latter if the TSIZE field contains zero value), triggering interrupt if enabled with TXPIE or EOTIE bit, respectively. The resulting interrupt service can be spurious if it tries to write data into TXFIFO to clear the TXP and TXC flags, while both FIFO buffers are inaccessible (as the peripheral is disabled).

Workaround

Keep TXP and TXC (the latter if the TSIZE field contains zero value) interrupt disabled whenever the SPI peripheral is disabled.

2.13.4 Possible corruption of last-received data depending on CRCSIZE setting

Description

With the CRC calculation disabled (CRCEN = 0), the transfer size bitfield set to a value greater than zero (TSIZE[15:0] > 0), and the length of CRC frame set to less than 8 bits (CRCSIZE[4:0] < 00111), the last data received in the RxFIFO may be corrupted.

Workaround

Keep the CRCSIZE[4:0] bitfield at its default setting (00111) during the data reception if CRCEN = 0 and TSIZE[15:0] > 0.

2.13.5 Truncation of SPI output signals after EOT event

Description

After an EOT event signaling the end of a non-zero transfer size transaction (TSIZE > 0) upon sampling the last data bit, the software may disable the SPI peripheral. As expected, disabling SPI deactivates the SPI outputs (SCK, MOSI and SS when the SPI operates as a master, MISO when as a slave), by making them float or statically output their by-default levels, according to the AFCNTR bit of the SPI_CFG2 register.

With fast software execution (high PCLK frequency) and slow SPI (low SCK frequency), the SPI disable occurring too fast may result in truncating the SPI output signals. For example, the device operating as a master then generates an asymmetric last SCK pulse (with CPHA = 0), which may prevent the correct last data bit reception by the other node involved in the communication.

Workaround

Apply one of the following measures or their combination:

- Add a delay between the EOT event and SPI disable action.
- Decrease the ratio between PCLK and SCK frequencies.

2.14 FDCAN

2.14.1 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.

2.14.2 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
```

2.14.3 DAR mode transmission failure due to lost arbitration

Description

In DAR mode, the transmission may fail due to lost arbitration at the first two identifier bits.

Workaround

Upon failure, clear the corresponding Tx buffer transmission request bit TRPx of the FDCAN_TXBRP register and set the corresponding cancellation finished bit CFx of the FDCAN_TXBCF register, then restart the transmission.

2.15 OTG_HS

2.15.1 Host packet transmission may hang when connecting the full speed interface through a hub to a low-speed device

Description

When the USB on-the-go high-speed peripheral is used with the full speed interface (DM and DP pins, N.B. not available on all devices), and connects to a low-speed device via a hub, the transmitter internal state machine may hang. This leads, after a timeout expiry, to a port disconnect interrupt.

Workaround

None. However, increasing the capacitance on the data lines may reduce the occurrence.

2.16 ETH

2.16.1 Incorrect L4 inverse filtering results for corrupted packets

Description

Received corrupted IP packets with payload (for IPv4) or total (IPv6) length of less than two bytes for L4 source port (SP) filtering or less than four bytes for L4 destination port (DP) filtering are expected to cause a mismatch. However, the inverse filtering unduly flags a match and the corrupted packets are forwarded to the software application. The L4 stack gets incomplete packet and drops it.

Note: The perfect filtering correctly reports a mismatch.

Workaround

None.

2.16.2 Rx DMA may fail to recover upon DMA restart following a bus error, with Rx timestamping enabled

Description

When the timestamping of Rx packets is enabled, some or all of the received packets can have Rx timestamp which is written into a descriptor upon the completion of the Rx packet/status transfer.

However, due to a defect, when bus error occurs during the descriptor read (that is subsequently used as context descriptor to update the Rx timestamp), the context descriptor write is skipped by DMA. Also, Rx DMA does not flush the Rx timestamp stored in the intermediate buffers during the error recovery process and enters Stop state. Due to this residual timestamp in the intermediate buffer, Rx DMA, after being restarted, does not transfer packets.

Workaround

Issue a soft reset to drop all Tx packets and Rx packets present inside the controller at the time of bus error. After the soft reset, reconfigure the controller and re-create the descriptors.

Note: The workaround introduces additional latency.

2.16.3 Spurious receive watchdog timeout interrupt

Description

Setting the RWTU[1:0] bitfield of the ETH_DMAC0RXIWTR register to a non-zero value while the RWT[7:0] bitfield is at zero leads to a spurious receive watchdog timeout interrupt (if enabled) and, as a consequence, to executing an unnecessary interrupt service routine with no packets to process.

Workaround

Ensure that the RWTU[1:0] bitfield is not set to a non-zero value while the RWT[7:0] bitfield is at zero. For setting RWT[7:0] and RWTU[1:0] bitfields each to a non-zero value, perform two successive writes. The first is either a byte-wide write to the byte containing the RWT[7:0] bitfield, or a 32-bit write that only sets the RWT[7:0] bitfield and keeps the RWTU[1:0] bitfield at zero. The second is either a byte-wide write to the RWTU[1:0] bitfield or a 32-bit write that sets the RWTU[1:0] bitfield while keeping the RWT[7:0] bitfield unchanged.

2.16.4 Incorrect flexible PPS output interval under specific conditions

Description

The use of the fine correction method for correcting the IEEE 1588 internal time reference, combined with a large frequency drift of the driving clock from the grandmaster source clock, leads to an incorrect interval of the flexible PPS output used in Pulse train mode. As a consequence, external devices synchronized with the flexible PPS output of the device can go out of synchronization.

Workaround

Use the coarse method for correcting the IEEE 1588 internal time reference.

2.16.5 Packets dropped in RMII 10Mbps mode due to fake dribble and CRC error

Description

When operating with the RMII interface at 10 Mbps, the Ethernet peripheral may generate a fake extra nibble of data repeating the last packet (nibble) of the data received from the PHY interface. This results in an odd number of nibbles and is flagged as a dribble error. As the RMII only forwards to the system completed bytes of data, the fake nibble would be ignored and the issue would have no consequence. However, as the CRC error is also flagged when this occurs, the error-packet drop mechanism (if enabled) discards the packets.

Note: Real dribble errors are rare. They may result from synchronization issues due to faulty clock recovery.

Workaround

When using the RMII 10 MHz mode, disable the error-packet drop mechanism by setting the FEP bit of the ETH_MTLRXQ0OMR or ETH_MTLRXQ1OMR register. Accept packets of transactions flagging both dribble and CRC errors.

2.16.6 ARP offload function not effective

Description

When the Target Protocol Address of a received ARP request packet matches the device IP address set in the ETH_MACARPAR register, the source MAC address in the SHA field of the ARP request packet is compared with the device MAC address in ETH_MACA0LR and ETH_MACA0HR registers (Address0), to filter out ARP packets that are looping back.

Instead, a byte-swapped comparison is performed by the device. As a consequence, the packet is forwarded to the application as a normal packet with no ARP indication in the packet status, and the device does not generate an ARP response.

For example, with the Address0 set to 0x665544332211:

- If the SHA field of the received ARP packet is 0x665544332211, the ARP response is generated while it should not.
- If the SHA field of the received ARP packet is 0x112233445566, the ARP response not is generated while it should.

Workaround

Parse the received frame by software and send the ARP response if the source MAC address matches the byte-swapped Address0.

2.16.7 Spurious checksum error upon MTL pending Tx queue flush

Description

Transmit checksum engine signals packet transmission errors via IHE (IP header error) and PCE (payload checksum error) flags.

When a flush of a pending (not currently served) MTL Tx FIFO queue 0 or 1 is initiated (by setting the FTQ bit of the ETH_MTLTXQ0OMR or ETH_MTLTXQ1OMR register, respectively), the MTL sends a dummy Tx status indicating the flushed packets. The checksum engine is expected to ignore this dummy Tx status.

However, when multiple transmit queues with checksum offload engines are enabled and the drop Tx status disabled (DTXSTS = 0 in the ETH_MTLOMR register), the checksum engine unduly takes into account the dummy Tx status. The MAC Tx status of the packet under transmission then returns zeros in its checksum field, instead of the due value.

As a consequence, the checksum engine may spuriously signal transmission error for the packet under transmission and for the following packet.

Note: The defect is expected to have no impact to the user application as the checksum error flags are intended for debug purposes only.

Workaround

None.

2.16.8 Bus error coinciding with start-of-packet corrupts MAC-generated packet transmission

Description

A bus error coinciding with the start of a new packet unduly aborts the transmission of any internally MAC-generated packet (ARP, PTO, Pause). As a consequence, the packet is sent on the line as a runt frame with corrupted FCS.

The aborted packet is not retransmitted and can cause:

- flow control failure in case of Pause/PFC packet corruption
- delay in ARP handshake from ARP offload engine (the ARP stack recovers because it sends ARP requests periodically)
- delay in PTP response/SYNC packets generated by the PTP offload engine (the PTP stack recovers because it sends request packets periodically)

The occurrence rate of this failure is very low.

Workaround

None.

2.16.9 DMA spurious state upon AXI DMA slave bus error

Description

Normally, upon an AXI DMA slave bus error, the ETH controller triggers a fatal bus error interrupt (by setting the FBE bit of the ETH_DMACSR register) and stops the corresponding DMA channel by clearing the ST bit of the ETH_DMAC0TXCR or ETH_DMAC1TXCR register. The software can then recreate the list of descriptors and restart the Tx DMA channel (set the ST bit).

However, in the following cases, the DMA controller fails to recover from the bus error or it corrupts the TSO/USO header data:

- when the bus error occurs for a packet transfer initiated by a Tx DMA channel 1 and the OSF bit of the ETH_DMAC0TXCR or ETH_DMAC1TXCR register is set
- when the TSO/USO segmentation is enabled in the Tx descriptor

Workaround

Reset and reconfigure the ETH controller by software, then re-create the descriptors. This restores the normal DMA controller operation, although it causes the loss of all latent Tx and Rx packets in the ETH controller and adds processing latency.

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 5. **Document revision history**

Date	Version	Changes
28-Feb-2023	1	Initial release.

Contents

1 Summary of device errata 2
2 Description of device errata 5
2.1 Core 5
2.1.1 Memory locations might be accessed speculatively due to instruction fetches when HCR.VM is set 5
2.1.2 Cache maintenance by set/way operations can execute out of order 5
2.1.3 PMU events 0x07, 0x0C, and 0x0E do not increment correctly 6
2.1.4 PMU event counter 0x14 does not increment correctly 7
2.1.5 Exception mask bits are cleared when an exception is taken in Hyp mode 7
2.2 System 8
2.2.1 TPIU fails to output sync after the pattern generator is disabled in Normal mode 8
2.2.2 Incorrect reset of glitch-free kernel clock switch 9
2.2.3 SAES, RNG, PKA stuck after first stage bootloader (FSBL) decryption 10
2.3 DMAMUX 10
2.3.1 SOFx not asserted when writing into DMAMUX_CFR register 10
2.3.2 OFx not asserted for trigger event coinciding with last DMAMUX request 10
2.3.3 OFx not asserted when writing into DMAMUX_RGCFR register 10
2.3.4 Wrong input DMA request routed upon specific DMAMUX_CxCR register write coinciding with synchronization event 11
2.4 FMC 11
2.4.1 NOR Flash memory/PSRAM incorrect bus turnaround timing 11
2.4.2 Incorrect FMC_CLK clock period when CLKDIV value is changed on-the-fly in Continuous clock mode 13
2.4.3 NAND Flash memory IREF/IFEF flags wrongly asserted just after enabling in FMC_IER . 14
2.4.4 Command sequencer accesses NAND Flash memory device while PBKEN bit is cleared in FMC_PCR 14
2.4.5 NAND Flash memory IREF flag wrongly asserted after reset 14
2.4.6 NAND ECC corrupted due to insufficient ECCEN low period in between sectors 14
2.5 QUADSPI 15
2.5.1 Memory-mapped read of last memory byte fails 15
2.6 ADC 15
2.6.1 Injected queue of context is not available if JQM = 0 15
2.6.2 Sampling time shortened in JAUTO auto delayed mode 15
2.6.3 Load multiple not supported by ADC interface 15
2.6.4 Overrun flag might not be set when converted data have not been read before new data are written 16
2.6.5 New context conversion initiated without waiting for trigger when writing new context in ADC_JSQR with JQDIS = 0 and JQM = 0 16

2.6.6 Two consecutive context conversions fail when writing new context in ADC_JSQR just after previous context completion with JQDIS = 0 and JQM = 0 . . . 16
2.6.7 Unexpected regular conversion when two consecutive injected conversions are performed in Dual interleaved mode . . . 17
2.6.8 ADC_AWDy_OUT reset by non-guarded channels . . . 17
2.6.9 Injected data stored in the wrong ADC_JDRx registers . . . 17
2.6.10 ADC slave data may be shifted in Dual regular simultaneous mode . . . 17
2.7 DTS . . . 18
2.7.1 DTS incorrect operation with LSE as reference clock and PCLK enabled . . . 18
2.8 TIM . . . 18
2.8.1 One-pulse mode trigger not detected in master-slave reset + trigger configuration . . . 18
2.8.2 Consecutive compare event missed in specific conditions . . . 18
2.8.3 Output compare clear not working with external counter reset . . . 19
2.8.4 Bidirectional break mode not working with short pulses . . . 19
2.9 LPTIM . . . 19
2.9.1 Device may remain stuck in LPTIM interrupt when entering Stop mode . . . 19
2.9.2 Device may remain stuck in LPTIM interrupt when clearing event flag . . . 20
2.9.3 LPTIM events and PWM output are delayed by 1 kernel clock cycle . . . 20
2.10 RTC and TAMP . . . 21
2.10.1 Alarm flag may be repeatedly set when the core is stopped in debug . . . 21
2.10.2 Binary mode: SSR is not reloaded with 0xFFFF FFFF when SSCLR = 1 . . . 21
2.11 I2C . . . 21
2.11.1 Wakeup frame may not wake up the MCU from Stop mode if $t_{HD;STA}$ is close to I2C kernel clock startup time . . . 21
2.11.2 Wrong data sampling when data setup time ($t_{SU;DAT}$) is shorter than one I2C kernel clock period . . . 22
2.11.3 Spurious bus error detection in master mode . . . 22
2.11.4 OVR flag not set in underrun condition . . . 23
2.11.5 Transmission stalled after first byte transfer . . . 23
2.11.6 SDA held low upon SMBus timeout expiry in slave mode . . . 23
2.12 USART . . . 24
2.12.1 Anticipated end-of-transmission signaling in SPI slave mode . . . 24
2.12.2 Data corruption due to noisy receive line . . . 24
2.12.3 USART does not generate DMA requests after setting/clearing DMAT bit . . . 24
2.12.4 Noise error flag set while ONEBIT is set . . . 24
2.13 SPI . . . 25
2.13.1 Master data transfer stall at system clock much faster than SCK . . . 25
2.13.2 Corrupted CRC return at non-zero UDRDET setting . . . 25
2.13.3 TXP interrupt occurring while SPI disabled . . . 25

2.13.4 Possible corruption of last-received data depending on CRCSIZE setting . . . 25
2.13.5 Truncation of SPI output signals after EOT event . . . 26
2.14 FDCAN . . . 26
2.14.1 Desynchronization under specific condition with edge filtering enabled . . . 26
2.14.2 Tx FIFO messages inverted under specific buffer usage and priority setting . . . 26
2.14.3 DAR mode transmission failure due to lost arbitration . . . 27
2.15 OTG_HS . . . 27
2.15.1 Host packet transmission may hang when connecting the full speed interface through a hub to a low-speed device . . . 27
2.16 ETH . . . 27
2.16.1 Incorrect L4 inverse filtering results for corrupted packets . . . 27
2.16.2 Rx DMA may fail to recover upon DMA restart following a bus error, with Rx timestamping enabled . . . 28
2.16.3 Spurious receive watchdog timeout interrupt . . . 28
2.16.4 Incorrect flexible PPS output interval under specific conditions . . . 28
2.16.5 Packets dropped in RMII 10Mbps mode due to fake dribble and CRC error . . . 29
2.16.6 ARP offload function not effective . . . 29
2.16.7 Spurious checksum error upon MTL pending Tx queue flush . . . 29
2.16.8 Bus error coinciding with start-of-packet corrupts MAC-generated packet transmission . . . 30
2.16.9 DMA spurious state upon AXI DMA slave bus error . . . 30
Important security notice . . . 31
Revision history . . . 32