Errata Sheet

Rel. 1.6, 2018-07

Device	**XMC4500**
Marking/Step	**ES-AB, AB**
Package	**PG-LQFP-100/144, PG-LFBGA-144**

Overview

Document ID is **082/18**.

This “Errata Sheet” describes product deviations with respect to the user documentation listed below.

Table 1 Current User Documentation

Document	Version	Date
XMC4500 Reference Manual	V1.5	July 2014
XMC4500 Data Sheet	V1.4	January 2016

Make sure that you always use the latest documentation for this device listed in category “Documents” at **http://www.infineon.com/xmc4000**.

Notes

1. *The errata described in this sheet apply to all temperature and frequency versions and to all memory size and configuration variants of affected devices, unless explicitly noted otherwise.*
2. *Devices marked with EES or ES are engineering samples which may not be completely tested in all functional and electrical characteristics, therefore they must be used for evaluation only. Specific test conditions for EES and ES are documented in a separate “Status Sheet”, delivered with the device.*
3. *XMC4000 devices are equipped with an ARM® Cortex®-M4 core. Some of the errata have a workaround which may be supported by some compiler tools. In order to make use of the workaround the corresponding compiler switches may need to be set.*

Conventions used in this Document

Each erratum is identified by **Module_Marker**.**TypeNumber**:

- **Module**: Subsystem, peripheral, or function affected by the erratum.
- **Marker**: Used only by Infineon internal.
- **Type**: type of deviation
  - **(none)**: Functional Deviation
  - **P**: Parametric Deviation
  - **H**: Application Hint
  - **D**: Documentation Update
- **Number**: Ascending sequential number. As this sequence is used over several derivatives, including already solved deviations, gaps inside this enumeration can occur.

1 History List / Change Summary

Table 2 History List

Version	Date	Remark
1.0	2012-11	Initial AB step version. Previous step is AA. Changes wrt. XMC4500 AA Errata Sheet v1.2: Added RESET_CM.H001
1.1	2013-02	Added: ADC_AI.002, CCU8_AI.003, CCU_AI.005, ETH_AI.003, PMU_CM.001 Updated: CCU8_AI.002, PORTS_CM.002 Removed (Fixed): USIC_AI.012, SCU_CM.001, STARTUP_CM.H001
1.2	2013-05	Added: PORTS_CM.005, STARTUP_CM.001
1.3	2014-04	Added: ADC_AI.008, ADC_TC.064, CCU8_AI.004, CPU_CM.004, DSD_AI.001, SCU_CM.015, USB_CM.003, USIC_AI.020, DAC_CM.P001, ADC_TC.H011 Updated: USIC_AI.008, RESET_CM.H001

Table 2 History List (cont'd)

Version	Date	Remark
1.4	2016-05	Added Functional Deviations: ADC_AI.016, ADC_TC.064, CACHE_CM.001, CCU8_AI.006, DSD_AI.002, DTS_CM.001, DTS_CM.001, FCE_CM.001, PARITY_CM.001, PARITY_CM.002, PBA_CM.001, PORTS_CM.007, SCU_CM.002, SCU_CM.015, SCU_CM.021, SDMMC_CM.003, SDMMC_CM.004, SDMMC_CM.005, SDMMC_CM.006, SDMMC_CM.007, SDMMC_CM.008, SDMMC_CM.009, USB_CM.004, WDT_CM.001 Added Deviations from Electrical and Timing Specifications: POWER_CM.P002, POWER_CM.P004 Added Applications Hints: ADC_AI.H003, ADC_AI.H008, ADC_TC.H011, ETH_AI.H001, MultiCAN_AI.H005, RESET_CM.H001, USIC_AI.H004
1.5	2017-08	This Document. Added Functional Deviations: ADC_CM.002 For changes see column "Chg" in the tables below.
1.6	2018-07	This document. Added Functional Deviations: CPU_CM.005 Updated Functional Deviations: PORTS_CM.001 Added Application Hints: PORTS_CM.H002 Added Documentation Updates: MPU_CM.D001, STARTUP_CM.D003 For updates and new issues see column "Chg" in the tables below.

Table 3 Errata fixed in this step

Errata	Short Description	Change
SCU_CM.001	Multiple Power-On resets upon noise on supply voltage	Fixed
STARTUP_CM.H001	ASC Bootstrap Loader Baudrate Limitation	Fixed
USIC_AI.012	USIC2 does not provide FIFO buffer capability	Fixed

Table 4 Functional Deviations

Functional Deviation	Short Description	Chg	Pg
ADC_AI.002	Result of Injected Conversion may be wrong		13
ADC_AI.008	Wait-for-Read condition for register GLOBRES not detected in continuous auto-scan sequence		13
ADC_AI.016	No Channel Interrupt in Fast Compare Mode with GLOBRES		14
ADC_CM.002	Converter diagnostics not functional		14
ADC_TC.064	Effect of conversions in 10-bit fast compare mode on post-calibration		15
CACHE_CM.001	Instruction buffer invalidation control bit needs to be cleared after an invalidation was triggered		15
CCU4_AI.001	CCU4 period interrupt is not generated in capture mode		16
CCU8_AI.001	CCU8 Floating Prescaler function does not work with Capture Trigger 1		18
CCU8_AI.002	CC82 Timer of the CCU8x module cannot use the external shadow transfer trigger connected to the POSIFx module		19

Table 4 Functional Deviations (cont'd)

Functional Deviation	Short Description	Chg	Pg
CCU8_AI.003	CCU8 Parity Checker Interrupt Status is cleared automatically by hardware		21
CCU8_AI.004	CCU8 output PWM glitch when using low side modulation via the Multi Channel Mode		24
CCU8_AI.006	Timer concatenation does not work when using external count signal		27
CCU_AI.001	CCU4 and CCU8 capture full flags do not work when module clock is faster than peripheral bus clock		29
CCU_AI.002	CCU4 and CCU8 Prescaler synchronization clear does not work when Module Clock is faster than Peripheral Bus Clock		31
CCU_AI.003	CCU4 and CCU8 capture full flag is not cleared if a capture event occurs during a bus read phase		32
CCU_AI.004	CCU4 and CCU8 Extended Read Back loss of data		35
CCU_AI.005	CCU4 and CCU8 External IP clock Usage		37
CCU_AI.006	Value update not usable in period dither mode		38
CPU_CM.001	Interrupted loads to SP can cause erroneous behavior		39
CPU_CM.004	VDIV or VSQRT instructions might not complete correctly when very short ISRs are used		40
CPU_CM.005	Store immediate overlapping exception return operation might vector to incorrect interrupt	New	42

Table 4 Functional Deviations (cont'd)

Functional Deviation	Short Description	Chg	Pg
DAC_CM.001	DAC immediate register read following a write issue		43
DAC_CM.002	No error response for write access to read only DAC ID register		43
DEBUG_CM.001	OCDS logic in peripherals affected by TRST		44
DEBUG_CM.002	CoreSight logic only reset after power-on reset		44
DSD_AI.001	Possible Result Overflow with Certain Decimation Factors		45
DSD_AI.002	Timestamp can be calculated wrong		45
DTS_CM.001	DTS offset calibration value limitations		48
ETH_AI.001	Incorrect IP Payload Checksum at incorrect location for IPv6 packets with Authentication extension header		48
ETH_AI.002	Incorrect IP Payload Checksum Error status when IPv6 packet with Authentication extension header is received		49
ETH_AI.003	Overflow Status bits of Missed Frame and Buffer Overflow counters get cleared without a Read operation		50
DTS_CM.001	MAC provides incorrect status and corrupts frames when RxFIFO overflow occurs on penultimate word of Rx frames of specific lengths		51
FCE_CM.001	Result value is wrong if read directly after last write		53
GPDMA_CM.001	Unexpected Block Complete Interrupt During Multi-Block Transfers		53

Table 4 Functional Deviations (cont'd)

Functional Deviation	Short Description	Chg	Pg
GPDMA_CM.002	GPDMA doesn't Accept Transfer During/In 2nd Cycle of 2-Cycle ERROR Response		55
LEDTS_AI.001	Delay in the update of FNCTL.PADT bit field		55
PARITY_CM.001	Parity error signaling can be suppressed in write/read sequence		60
PARITY_CM.002	Clock limitations for ETH and SDMMC modules when using parity check of module SRAMs		61
PBA_CM.001	Bus error request suppressed in sequential write to peripheral bridge		62
PMU_CM.001	Branch from non-cacheable to cacheable address space instruction may corrupt the program execution		63
PORTS_CM.001	P15_PDISC.[4,5] register bits cannot be written	Update	65
PORTS_CM.002	P0.9 Pull-up permanently active		65
PORTS_CM.005	Different PORT register reset values after module reset		66
PORTS_CM.007	P14 and P15 cannot be used in boundary scan test		67
POSIF_AI.001	Input Index signal from Rotary Encoder is not decoded when the length is 1/4 of the tick period		67
RTC_CM.001	RTC event might get lost		69
SCU_CM.002	Missed wake-up event during entering external hibernate mode		70
SCU_CM.003	The state of HDCR.HIB bit of HCU gets updated only once in the register mirror after reset release		70

Table 4 Functional Deviations (cont'd)

Functional Deviation	Short Description	Chg	Pg
SCU_CM.006	Deep sleep entry with PLL power-down option generates SOSCWDGT and SVCOLCKT trap		71
SCU_CM.015	Functionality of parity memory test function limited		71
SCU_CM.021	Registering of service requests in SRRAW register can fail		72
SDMMC_CM.001	Unexpected interrupts after execution of CMD13 during bus test		74
SDMMC_CM.002	Unexpected Tx complete interrupt during R1b response		74
SDMMC_CM.003	SDMMC input pins cannot be released for other usage		75
SDMMC_CM.004	Busy response from card in write resume operation not detected		76
SDMMC_CM.005	Controller sends other command when Auto CMD12 enabled		76
SDMMC_CM.006	Stream write issue due to wrong FIFO handling causes data corruption in eMMC mode		77
SDMMC_CM.007	Consecutive write to the same register in SD clock domain		77
SDMMC_CM.008	Receive state machine hangs if driver programs stop at block gap request using CMD18 when receive buffers are full		78
SDMMC_CM.009	Latching current value in the response register		78
STARTUP_CM.001	CAN Bootstrap Loader		79

Table 4 Functional Deviations (cont'd)

Functional Deviation	Short Description	Chg	Pg
USB_CM.002	GAHBCFG.GlblIntrMsk not cleared with a software reset		79
USB_CM.003	Endpoint NAK not sent in Device Class applications with multiple endpoints enabled		79
USB_CM.004	USB core is not able to detect resume or new session request after PHY clock is stopped		80
USIC_AI.005	Only 7 data bits are generated in IIC mode when TBUF is loaded in SDA hold time		81
USIC_AI.006	Dual SPI format not supported		81
USIC_AI.007	Protocol-related argument and error bits in register RBUFSR contain incorrect values following a received data word		82
USIC_AI.008	SSC delay compensation feature cannot be used		83
USIC_AI.009	Baud rate generator interrupt cannot be used		84
USIC_AI.010	Minimum and maximum supported word and frame length in multi-IO SSC modes		84
USIC_AI.011	Write to TBUF01 has no effect		85
USIC_AI.013	SCTR register bit fields DSM and HPCDIR are not shadowed with start of data word transfer		85
USIC_AI.014	No serial transfer possible while running capture mode timer		85
USIC_AI.015	Wrong generation of FIFO standard transmit/receive buffer events when TBCTR.STBTEN/RBCTR.SRBTEN = 1		86

Table 4 Functional Deviations (cont'd)

Functional Deviation	Short Description	Chg	Pg
USIC_AI.016	Transmit parameters are updated during FIFO buffer bypass		86
USIC_AI.018	Clearing PSR.MSLS bit immediately deasserts the SELOx output signal		87
USIC_AI.020	Handling unused DOUT lines in multi-IO SSC mode		88
WDT_CM.001	No overflow is generated for WUB default value		88

Table 5 Deviations from Electrical- and Timing Specification

AC/DC Deviation	Short Description	Chg	Pg
DAC_CM.P001	INL parameter limits violated by some devices		89
POWER_CM.P002	Minimum limit of external buffer capacitor C_{EXT}		89
POWER_CM.P004	Current consumption while PORST low can exceed specified value		90

Table 6 Application Hints

Hint	Short Description	Chg	Pg
ADC_AI.H003	Injected conversion may be performed with sample time of aborted conversion		91
ADC_AI.H004	Completion of Startup Calibration		92
ADC_AI.H008	Injected conversion with broken wire detection		92
ADC_TC.H011	Bit DCMSB in register GLOBCFG		93
ETH_AI.H001	Sequence for Switching between MII and RMII Modes		94
MultiCAN_AI.H005	TxD Pulse upon short disable request		94

Table 6 Application Hints (cont'd)

Hint	Short Description	Chg	Pg
MultiCAN_AI.H006	Time stamp influenced by resynchronization		94
MultiCAN_AI.H007	Alert Interrupt Behavior in case of Bus-Off		95
MultiCAN_AI.H008	Effect of CANDIS on SUSACK		95
MultiCAN_TC.H003	Message may be discarded before transmission in STT mode		96
MultiCAN_TC.H004	Double remote request		96
PORTS_CM.H002	Class A2 pins GPIO driver strength configuration	New	97
RESET_CM.H001	Power-on reset release		98
USIC_AI.H004	I2C slave transmitter recovery from deadlock situation		99

Table 7 Documentation Updates

Hint	Short Description	Chg	Pg
MPU_CM.D001	No restrictions on using Bit5 to Bit8 of register MPU_RBAR	New	100
STARTUP_CM.D003	Alignment of ABM/PSRAM Header	New	100

2 Functional Deviations

The errata in this section describe deviations from the documented functional behavior.

ADC_AI.002 Result of Injected Conversion may be wrong

In cancel-inject-repeat mode (GxARBPR.CSM* = 1_B), the result of the higher prioritized injected conversion c_H may be wrong if it was requested within a certain time window at the end of a lower prioritized conversion c_L. The width of the critical window depends on the divider factor DIVA for the analog internal clock.

Workaround

Do not use cancel-inject-repeat mode. Instead, use wait-for-start mode (GxARBPR.CSM* = 0_B).

ADC_AI.008 Wait-for-Read condition for register GLOBRES not detected in continuous auto-scan sequence

In the following scenario:

- A continuous auto-scan is performed over several ADC groups and channels by the Background Scan Source, using the global result register (GLOBRES) as result target (GxCHCTRy.RESTBS=1_B), and
- The Wait-for-Read mode for GLOBRES is enabled (GLOBRCR.WFR=1_B),

each conversion of the auto-scan sequence has to wait for its start until the result of the previous conversion has been read out of GLOBRES.

When the last channel of the auto-scan is converted and its result written to GLOBRES, the auto-scan re-starts with the highest channel number of the highest ADC group number. But the start of this channel does not wait until the result of the lowest channel of the previous sequence has been read from register GLOBRES, i.e. the result of the lowest channel may be lost.

Workaround

If either the last or the first channel in the auto-scan sequence does not write its result into GLOBRES, but instead into its group result register (selected via bit GxCHCTRy.RESTBS=0_B), then the Wait-for-Read feature for GLOBRES works correctly for all other channels of the auto-scan sequence.

For this purpose, the auto-scan sequence may be extended by a “dummy” conversion of group x/ channel y, where the Wait-for-Read mode must not be selected (GxRCRy.WFR=0_B) if the result of this “dummy” conversion is not read.

ADC_AI.016 No Channel Interrupt in Fast Compare Mode with GLOBRES

In fast compare mode, the compare value is taken from bitfield RESULT of the selected result register and the result of the comparison is stored in the respective bit FCR.

A channel event can be generated when the input becomes higher or lower than the compare value.

In case the global result register GLOBRES is selected, the comparison is executed correctly, the target bit is stored correctly, source events and result events are generated, but a channel event is not generated.

Workaround

If channel events are required, choose a local result register GxRESy for the operation of the fast compare channel.

ADC_CM.002 Converter diagnostics not functional

The analog converter diagnostics feature of the VADC to test the proper operation is not functional.

Implications

All diagnostic pull devices remain disconnected, also if the converter diagnostics feature is enabled.

No portions of VAref can be selected for diagnostic purpose.

Workaround

None.

ADC_TC.064 Effect of conversions in 10-bit fast compare mode on post-calibration

The calibrated converters Gx (x = 0..3) support post-calibration. Unless disabled by software (via bits GLOBCFG.DPCALx = 0), a calibration step is performed after each conversion, incrementally increasing/decreasing internal calibration values to compensate process, temperature, and voltage variations.

If a conversion in 10-bit fast-compare mode (bit field CMS/E = 101_B in corresponding Input Class register) is performed between two conversions in other (non-fast-compare) modes on a converter Gx, the information gained from the last post-calibration step is disturbed. This will lead to a slightly less accurate result of the next conversion in a non-fast-compare mode.

Depending on the ratio of conversions in fast-compare mode versus conversions in other modes, this effect will be more or less obvious.

In a worst case scenario (fast-compare with a constant result injected between each two normal conversions), all calibration values can drift to their maxima / minima, causing the converter Gx to deliver considerably inaccurate results.

Workaround

Do not mix conversions using 10-bit fast-compare mode and other conversions with enabled postcalibration on the calibrated converters Gx (x = 0..3). Instead, use a dedicated group for fast-compare operations.

CACHE_CM.001 Instruction buffer invalidation control bit needs to be cleared after an invalidation was triggered

The device reference manual describes the PCON.IINV bit of PREF unit as write only. Writing 1_B is initiating the invalidation of the instruction buffer, and writing 0_B has no effect.

However, writing 1_B to PCON.IINV will force sustaining instruction buffer invalidation until the bit is cleared again by software writing 0_B.

Implications

As long as PCON.IINV remains set to 1_B, the system will not benefit from the performance improvement by the instruction buffer. In fact, as the prefetch unit attempts to perform instruction buffer refills in the background, the system may show slower code execution performance as with execution from uncached Flash address range or with bypassed instruction buffer.

Workaround

Execute the following sequence if the PREF instruction cache needs to be invalidated:

1. Branch to RAM or uncached Flash address range.
2. Enable the instruction buffer bypass.
   PCON.IBYP = 1_B
3. Invalidate the instruction buffer.
   PCON.IINV = 1_B
4. Clear the invalidate bit.
   PCON.IINV = 0_B
5. Disable the instruction buffer bypass.
   PCON.IBYP = 0_B
6. Read back PCON to resolve pipelining effects.
7. Return to cacheable Flash address range.

CCU4_AI.001 CCU4 period interrupt is not generated in capture mode

The Capture/Compare Unit 4 (CCU4) has several capture modes. These capture modes are shown in **Figure 1**.

The depth-2 x2 capture mode enables the usage of two different capture triggers (Capture Trigger 0 and Capture Trigger 1). Each capture trigger is linked to two capture registers that work in FIFO mode.

The depth-4 capture mode only has one capture trigger (capture trigger 1). This capture trigger is then linked with the 4 available capture registers that build the FIFO structure.

The period interrupt is not generated when the timer slice is programmed in the depth-4 capture mode or when is programmed in depth-2 capture mode and the capture trigger 1 is used.

These situations occur whenever capture trigger 1 is being used (when the CC4yCMC.CAP1S field is programmed with a value different from 00_B).

Note that the period interrupt is only necessary if the capture trigger periodicity is bigger than the timer period itself.

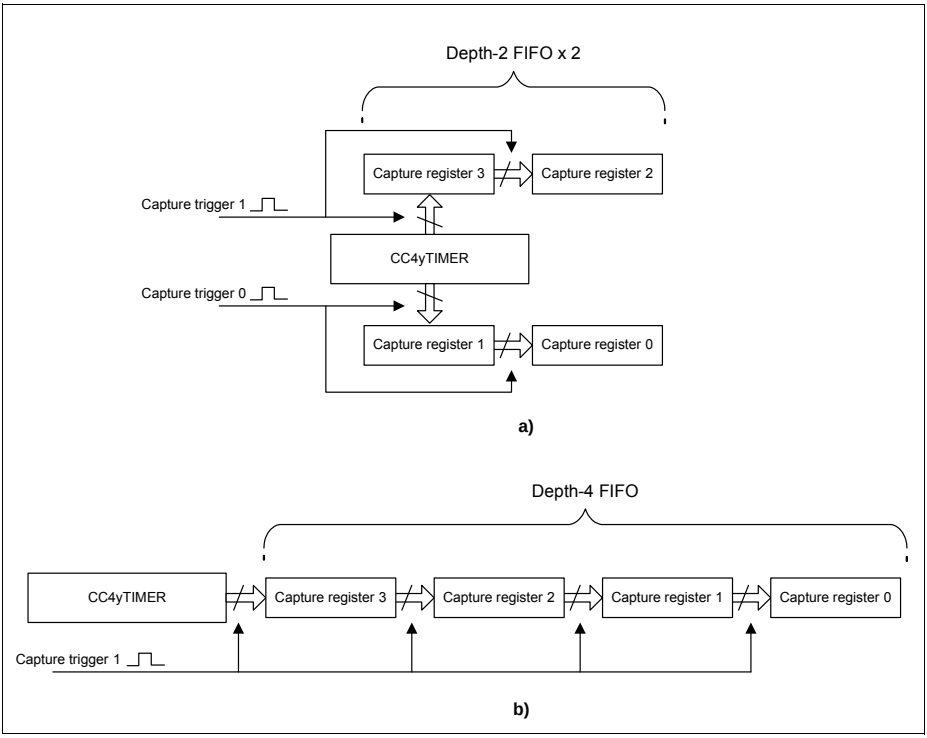

Figure 1 CCU4 capture modes - a) Depth-2 x2 Capture; b) Depth-4 Capture

Workaround 1

A straightforward workaround is to use only 2 capture registers (instead of a maximum of 4). This is done by setting the CC4yCMC.CAP1S to 00_B and program the CC4yCMC.CAP0S with a value different from 00_B.

Workaround 2

By using the floating prescaler function present in each timer slice, the capture routine can ignore the missing period interrupt, for a capture trigger frequency as low as 0.25 Hz (for f_{CCU} equal to 120 MHz).

The floating prescaler function can be enabled by setting the CC4yTC.FPE = 1_B.

CCU8_AI.001 CCU8 Floating Prescaler function does not work with Capture Trigger 1

Each CCU8 Timer Slice contains a Floating Prescaler function that allows capturing the elapsed time between two triggers (with unknown or very high dynamics), with minimum software interaction **Figure 2**.

Referring to **Figure 2**, the time elapsed between the two capture triggers can be calculated as a dependency between the actual Timer Value plus the distance between the two capture triggers dictated by the Current Prescaler Value.

Each CCU8 Timer also has four capture registers: Capture Register 0, Capture Register 1, Capture Register 2 and Capture Register 3. All these registers can be used to capture the Timer Value and the Current Prescaler Value.

The usage of Capture Register 2 and Capture Register 3 is not possible when the Floating Prescaler function is enabled, CC8yTC.FPE = 1_B. This only happens when the Capture Trigger 1 is being used, CC8yCMD.CAP1S != 00_B. Therefore is not possible to use the Floating Prescaler feature with the capture trigger 1. The usage of the capture trigger 0 is not affected, which means that capture register 0 and capture register 1 can be used with the floating prescaler feature.

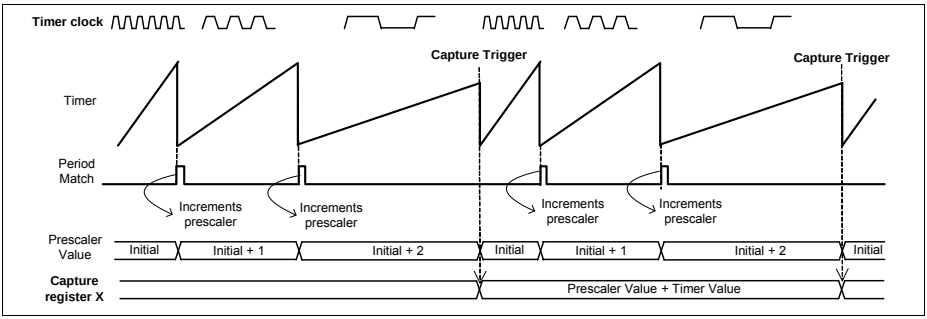

Figure 2 Floating Prescaler for Capturing

Workaround

None.

CCU8_AI.002 CC82 Timer of the CCU8x module cannot use the external shadow transfer trigger connected to the POSIFx module

Each CCU8 Module Slice contains 4 identical timers (CC80, CC81, CC82 and CC83). There is the possiblity of updating the values controlling the duty cycle, period, output passive level, dither and floating prescaler on-the-fly of each and every timer, with a SW request. The update request of these values can also be done via an external trigger that is connected to the POSIFx module Figure 1. An update action of any of these values is named as “shadow transfer”.

The signal between the POSIFx and CCU8x module is used to handshake a concurrent update between several registers, contained in the two modules. The output signal of the POSIFx is named as POSIFx.OUT6 while the input signal on the CCU8x side is named as CCU8x.MCSS.

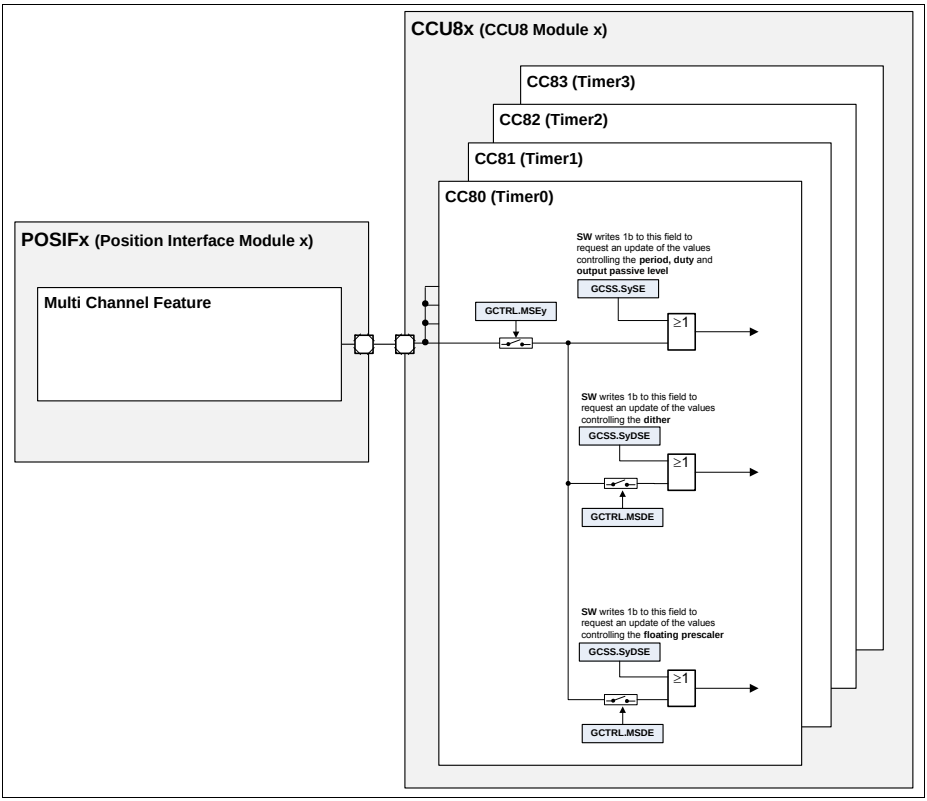

Figure 3 Value update trigger connection between CCU8x and POSIFx

On Figure 2, we see an example how this trigger is used to update at the same time teh duty cycle value of a timer inside the CCU8x and the multi channel pattern inside the POSIFx (the multi channel pattern can affect the CCU8x timer outputs therefore a synchronous update of all the values solves possible output glitches on the generated PWM signals).

On Figure 2, the SW has updated the next values for the duty cycle on a CCU8x Timer (it can be also for the period, output passive level and clock prescaler). After that it updates also the next value of the multi channel pattern inside the POSIFx module. After that, the POSIF reaches an internal state (dictated by specific conditions) where an update of the values is needed. it generates a trigger signal to the CCU8x Timer to signalize that an update of the duty cycle

value needs to be done. After that timeframe, the POSIF waits for the handshake trigger of the CCU8x Timer to indicate that an update is going to be performed. At this specific time, both the values of the CCU8x Timer and POSIF are update completely synchronous.

This feature cannot be used with the Timer2 (defined as CC82 in the documentation) of the CCU8x module(s) (more than one CCU8 module can be contained on a specific device).

All the other 3 Timers (defined as CC80, CC81, CC83) inside the CCU8x moduel are not affected by this issue.

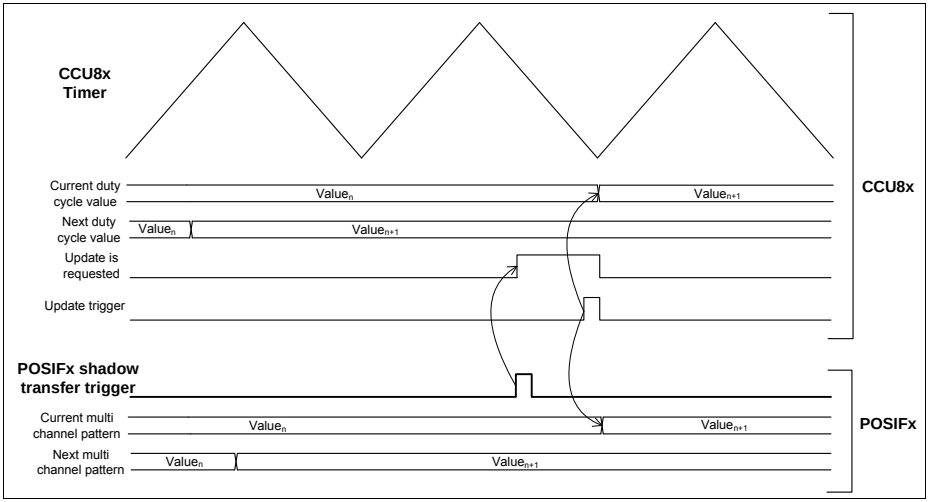

Figure 4 Value update handshake between CCU8x and POSIFx

Workaround

None

CCU8_AI.003 CCU8 Parity Checker Interrupt Status is cleared automatically by hardware

Each CCU8 Module Timer has an associated interrupt status register. This Status register, CC8yINTS, keeps the information about which interrupt source

triggered an interrupt. The status of this interrupt source can only be cleared by software. This is an advantage because the user can configure multiple interrupt sources to the same interrupt line and in each triggered interrupt routine, it reads back the status register to know which was the origin of the interrupt.

Each CCU8 module also contains a function called Parity Checker. This Parity Checker function, crosschecks the output of a XOR structure versus an input signal, as seen in Figure 1.

When using the parity checker function, the associated status bitfield, is cleared automatically by hardware in the next PWM cycle whenever an error is not present.

This means that if in the previous PWM cycle an error was detected and one interrupt was triggered, the software needs to read back the status register before the end of the immediately next PWM cycle.

This is indeed only necessary if multiple interrupt sources are ORed together in the same interrupt line. If this is not the case and the parity checker error source is the only one associated with an interrupt line, then there is no need to read back the status information. This is due to the fact, that only one action can be triggered in the software routine, the one linked with the parity checker error.

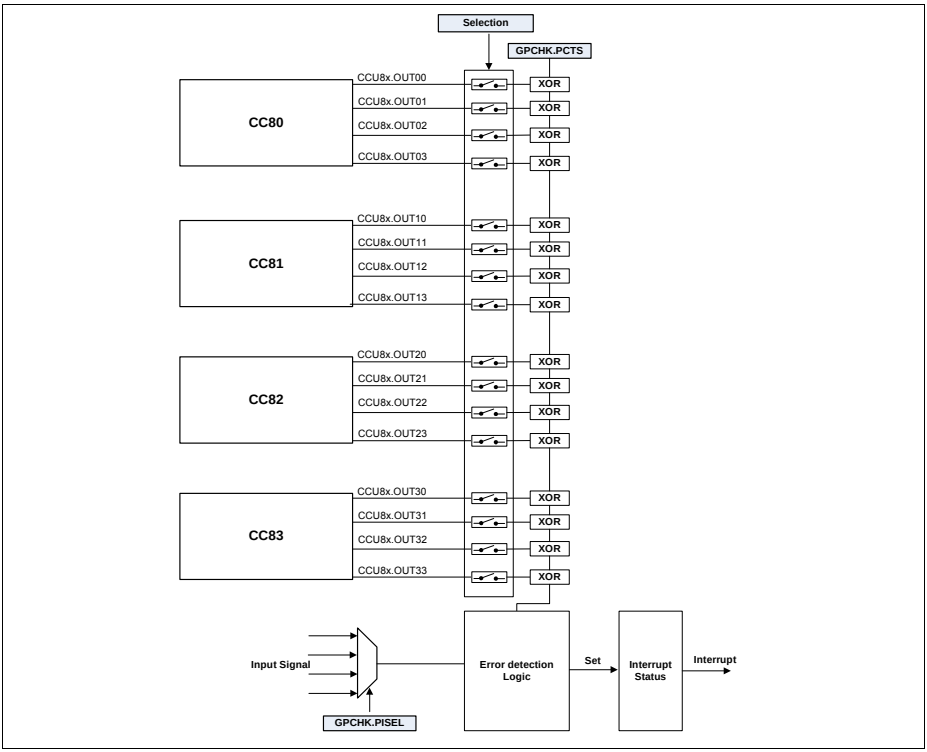

Figure 5 Parity Checker diagram

Workaround

Not ORing the Parity Checker error interrupt with any other interrupt source. With this approach, the software does not need to read back the status information to understand what was the origin of the interrupt - because there is only one source.

CCU8_AI.004 CCU8 output PWM glitch when using low side modulation via the Multi Channel Mode

Each CCU8 Timer Slice can be configured to use the Multi Channel Mode - this is done by setting the CC8yTC.MCME1 and/or CC8yTC.MCME2 bit fields to 1_B. Each bit field enables the multi channel mode for the associated compare channel of the CCU8 Timer Slice (each CCU8 Timer Slice has two compare channels that are able to generate each a complementary pair of PWM outputs).

After enabled, the Multi Channel mode is then controlled by several input signals, one signal per output. Whenever an input is active, the specific PWM output is set to passive level - Figure 1.

The Multi Channel mode is normally used to modulate in parallel several PWM outputs (a complete CCU8 - up to 16 PWM signals can be modulated in parallel).

A normal use case is the parallel control of the PWM output for BLDC motor control. In Figure 2, we can see the Multi Channel Pattern being updated synchronously to the PWM signals. Whenever a multi channel input is active (in this case 0), the specific output is set into passive level (the level in which the external switch is OFF).

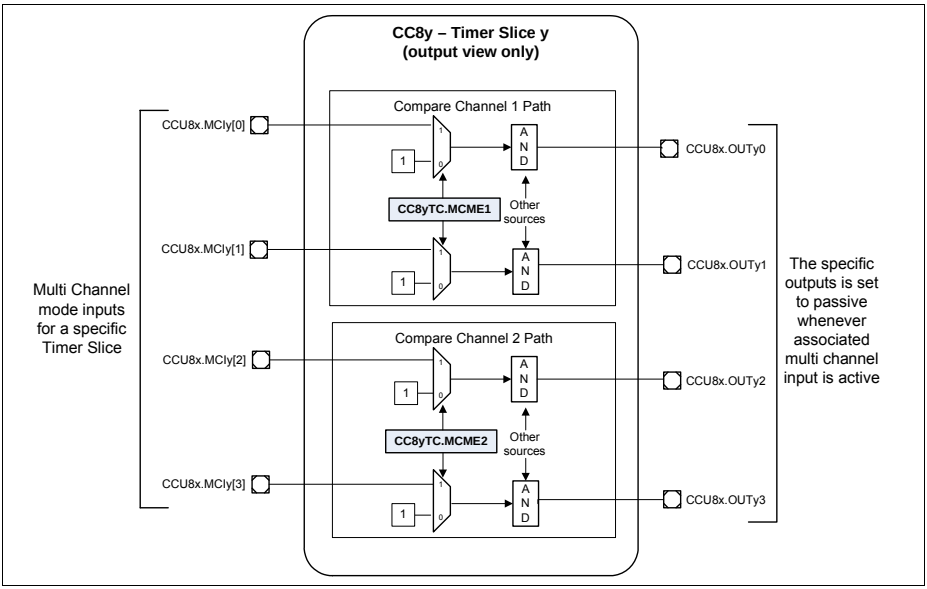

Figure 6 Multi Channel Mode diagram

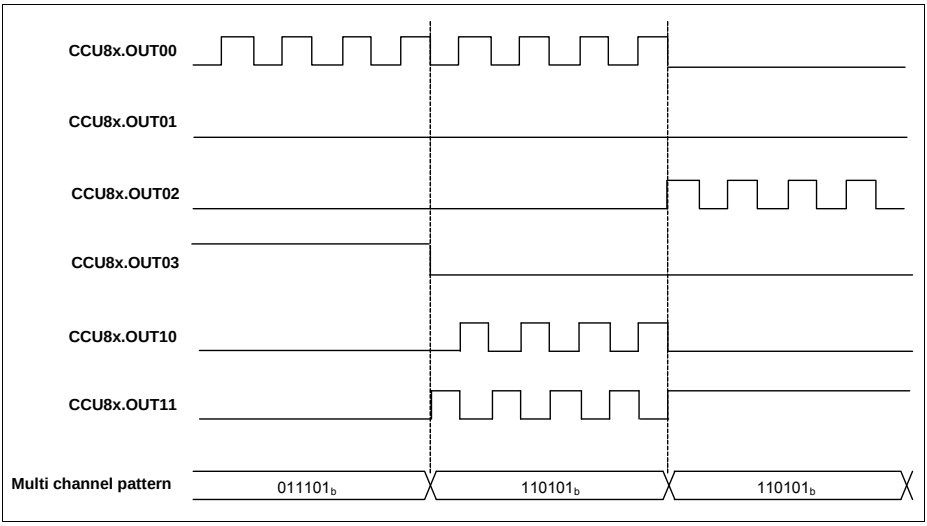

Figure 7 Multi Channel Mode applied to several CCU8 outputs

A glitch is present at the PWM outputs whenever the dead time of the specific compare channel is enabled - CC8yDTC.DTE1 and/or CC8yDTC.DTE2 set to 1_B (each compare channel has a separate dead time function) - and the specific multi channel pattern for the channel is 01_B or 10_B.

This glitch is not present if the specific timer slice is configure in symmetric edge aligned mode - CC8yTC.TCM = 0_B and CC8yCHC.ASE = 0_B.

This glitch only affects the PWM output that is linked to the inverting ST path of each compare channel (non inverting outputs are not affected).

The effect of this glitch can be seen in Figure 3. The duration of the PWM glitch has the same length has the dead time value programmed into the CC8yDC1R.DT1F field (for compare channel 1) or into the CC8yDC1R.DT2F.

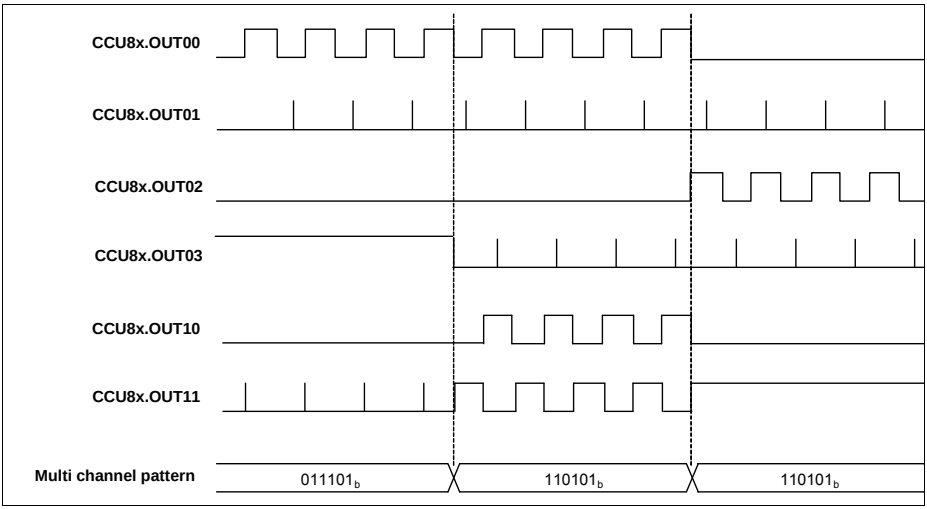

Figure 8 PWM output glitch

Workaround

To avoid the glitch on the inverting path of the PWM output, one can disable the dead time function before the Multi Channel Pattern is set to 01_B or 10_B. Disabling the dead time of the inverting PWM output can be done by setting:

```
CC8yDTC.DCEN2 = 0 //if compare channel 1 is being used
```

```
CC8yDTC.DCEN4 = 0 //if compare channel 2 is being used
```

The dead time needs to be re enabled, before the complementary outputs become modulated at the same time:

```
CC8yDTC.DCEN2 = 1 //if compare channel 1 is being used
CC8yDTC.DCEN4 = 1 //if compare channel 2 is being used
```

CCU8_AI.006 Timer concatenation does not work when using external count signal

Each CCU8 peripheral contains four sixteen bit timers. It is possible nevertheless to concatenate multiple timers to achieve a timer/counter with 32, 48 or 64 bits. To enable the concatenation feature, the CC8yCMC.TCE bitfield needs to be set to 1_B - Figure 1 a), where CCU8x represents a CCU8 peripheral instance x, and CC80 and CC81, represents timer 0 and timer 1 respectively (please notice that CC80 and CC81 are just used for simplicity, meaning that this function can be used also with the other timers inside CCU8x).

It is also possible to use an external signal as a count trigger. This means that when using an external count signal, the LSB timer is incremented each time that a transition on this external signal occurs - Figure 1 b).

When an external count signal is enabled - by programming the CC8yCMC.CNTS with 01_B, 10_B or 11_B - the concatenation function does not work. One cannot use in parallel the timer concatenation and external count signal features.

Note: On Figure 1, the count signal is used in CCU80 because this timer represents the LSBs. While the count signal could be enabled in the MSB timer (CC81), this does not make sense when the timers are concatenate - because the count should be used to increment the LSB timer. The LSB timer will then in each wrap around, increment the MSB timer.

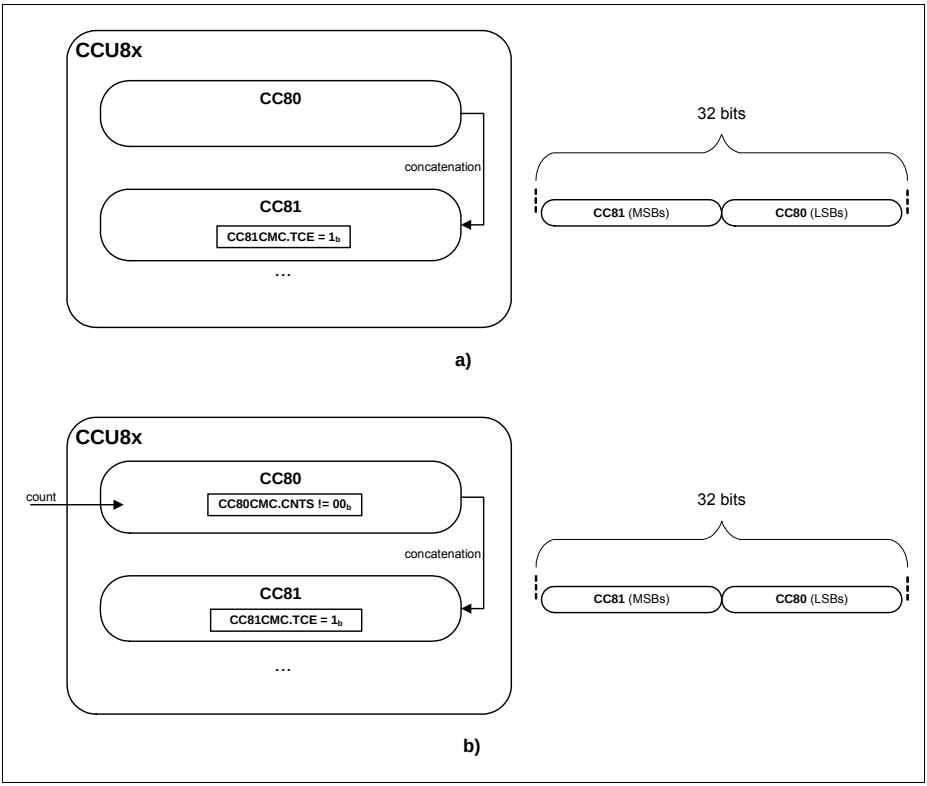

Figure 9 CCU8x concatenation feature resource configuration - a) without external count function; b) with external count function

Workaround

None

CCU_AI.001 CCU4 and CCU8 capture full flags do not work when module clock is faster than peripheral bus clock

Each CCU4/CCU8 timer slice contains a “Full Flag” field in every capture register. The structure of the different capture modes for each timer slice can be seen in **Figure 10**.

The full flag field serves as an indication to the software (when it is reading back the specific capture register), for checking whether a new value has been captured or not into this register, since the previous read back.

When the peripheral bus clock frequency is smaller than the CCU4/CCU8 module clock frequency, $f_{periph} < f_{ccu}$, the read back of the full flag is inconsistent. Sometimes it returns the information that a new value has been captured and sometimes it does not.

Note: The capture interrupt is generated correctly.

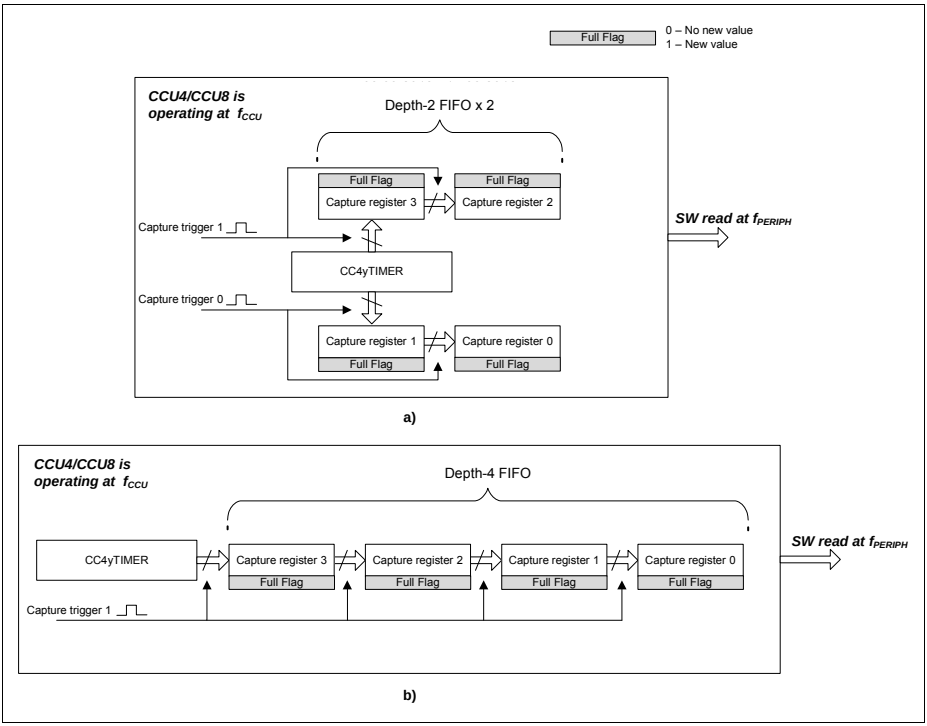

Figure 10 Capture Modes Structure - a) Depth-2 x2 Capture; b) Depth-4 Capture

Workaround

When the usage of the FFL field is needed, the module clock of the CCU4/CCU8 module should be equal to the peripheral bus clock frequency: $f_{periph} = f_{ccu}$.

To do this, the following SCU (System Control Unit) registers should be set with values that force this condition: CCUCLKCR.CCUDIV, CPUCLKCR.CPUDIV and PBCLKCR.PBDIV.

CCU_AI.002 CCU4 and CCU8 Prescaler synchronization clear does not work when Module Clock is faster than Peripheral Bus Clock

Each CCU4/CCU8 module contains a feature that allows to clear the prescaler division counter synchronized with the clear of a run bit of a Timer Slice. This is configure via the GCTRL.PRBC field. The default value of 000_B dictates that only the software can clear the prescaler internal division counter. Programming a value different from 000_B into the PRBC will impose that the prescaler division counter is cleared to 0_D whenever the selected Timer Slice (selected via the PRBC field) run bit is cleared (TRB bit field).

In normal operating conditions, clearing the internal prescaler division counter is not needed. The only situation were a clear of the division may be needed is when several Timer Slices inside one unit (CCU4/CCU8) are using different prescaling factors and a realignment of all the timer clocks is needed. This normally only has a benefit if there is a big difference between the prescaling values, e.g. Timer Slice 0 using a module clock divided by 2_D and Timer Slice 1 using a module clock divided by 1024_D.

When the peripheral bus clock frequency is smaller than the CCU4/CCU8 module clock frequency, $f_{periph} < f_{ccu}$, it is not possible to clear the prescaler division counter, synchronized with the clear of the run bit of one specific Timer Slice.

Workaround 1

The clearing of the prescaler internal division counter needs to be done via software: GCTRL.PRBC programmed with 000_B and whenever a clear is needed, writing 1_B into the GIDLS.CPRB bit field.

Workaround 2

When the usage of the Prescaler internal division clear needs to be synchronized with a timer run bit clear, the module clock of the CCU4/CCU8 should be equal to the peripheral bus clock frequency: $f_{periph} = f_{ccu}$.

To do this, the following SCU (System Control Unit) registers should be set with values that force this condition: CCUCLKCR.CCUDIV, CPUCLKCR.CPUDIV and PBCLKCR.PBDIV.

CCU_AI.003 CCU4 and CCU8 capture full flag is not cleared if a capture event occurs during a bus read phase

Each CCU4/CCU8 Timer Slice contains a Full Flag field in every capture register. The structure of the different capture modes for each Timer Slice can be seen in **Figure 11**.

The full flag field serves as an indication to the software (when it is reading back the specific capture register), if a new value has been captured or not into this register, since the previous read back. (Note that the capture interrupt can still be generated).

When a capture event collides with a read back on the data bus, the proper data is read by the software, but this data is shifted to the immediately capture register and the associated full flag is set, **Figure 12** - for simplification purposes a 2 depth capture scheme is shown.

Referring to **Figure 12**, it can be understood that the proper data is sent to the software when it reads back the capture register 1, nevertheless this same data is shifted to the next capture register (capture register 0) and the full flag of this register is set.

After this if the software reads back capture register 0, a value is going to be returned with a full flag set (indicating that this is a new value - that has not yet been read, which is not true).

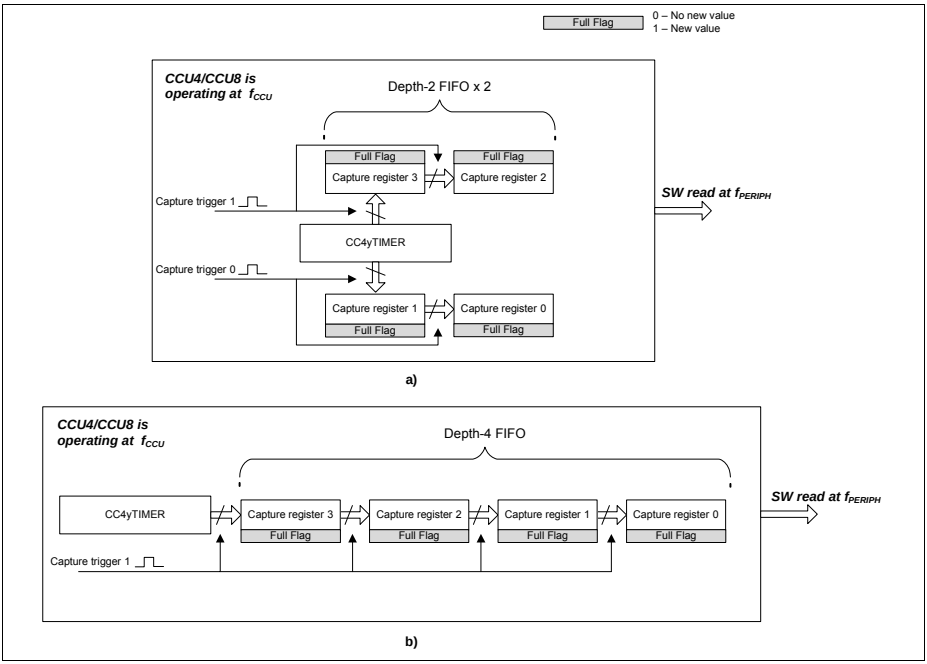

Figure 11 Capture Modes Structure - a) Depth-2 x2 Capture; b) Depth-4 Capture

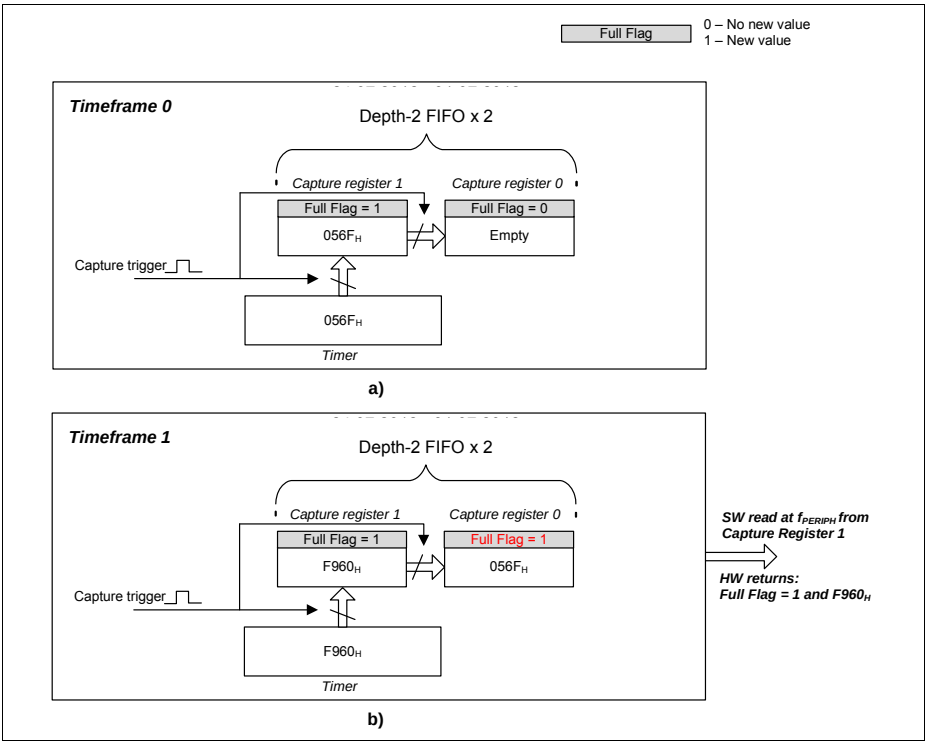

Figure 12 Capture shift during read back phase - a) Capture trigger without collision with read back; b) Capture trigger collision with read back

Workaround 1

If the dynamics of the capture trigger(s) cannot guarantee a safe read back of the captured data without collision, then the software can monitor if the timer has rollover or not between two reads.

This can be done by enabling per example by setting the timer mode in Edge Aligned, TCM = 0_B and enabling the Period Interrupt PME = 1_B. This interrupt should then be routed to one of the service request outputs by setting the POSR field accordingly (e.g. setting POSR = 00_B will output the Period Interrupt at the Service Request Output 0 of CCU4/CCU8).

In every read back where the software finds a value equal to the previous one, it should then poll the interrupt status to understand if the timer has rollover or not. If the timer has not rollover, then this is the same value that was previously read.

Workaround 2

The software reads back in every capture event. This can be done by enabling the capture interrupt, and in each capture interrupt it reads back a capture register.

Enabling the capture interrupt is done in the following way: if the capture trigger is linked to input Event 0, then E0AE needs to be set to 1_B; if the capture trigger is linked to input Event 1, then E1AE needs to be set to 1_B; if the capture trigger is linked to input Event 2, then E2AE needs to be set to 1_B.

Routing the interrupt to one of the four available service request outputs: if Event 0 is being used and the Service Request Output 0 should be used, then E0SR needs to be set with the value 00_B (for Event 1 the field is E1SR and for Event 2 E2SR).

CCU_AI.004 CCU4 and CCU8 Extended Read Back loss of data

Each CCU4/CCU8 Timer Slice contains a bit field that allows the enabling of the Extended Read Back feature. This is done by setting the CC8yTC.ECM/CC4yTC.ECM = 1_B. Setting this bit field to 1_B only has an impact if the specific Timer Slice is working in Capture Mode (CC8yCMC.CAP1S or CC8yCMC.CAP0S different from 00_B - same fields for CCU4).

By setting the bit field to ECM = 1_B, is then possible to read back the capture data of the specific Timer Slice (or multiple Timer Slices, if this bit field is set in more than one Timer Slice) trough a single address. This address is linked to the ECRD register.

Referring to **Figure 13**, the hardware every time that the software reads back from the ECRD address, will return the immediately next capture register that contains new data. This is done in a circular access, that contains all the capture registers from the Timer Slices that are working in capture mode.

When using this feature, there is the possibility of losing captured data within a Timer Slice. The data that is lost is always the last captured data within a timer slice, e.g (with CCU4 nomenclature - same applies to CCU8):

- Timer X has 4 capture registers and is the only Timer set with ECM = 1_B. At the moment that the software starts reading the capture registers via the ECRD address, we have already capture four values. The ECRD read back will output CC4xC0V -> CC4xC1V -> CC4xC2V -> CC4xC2V (CC4xC3V value is lost)
- Timer X has 4 capture registers and is the only Timer set with ECM = 1_B. At the moment that the software starts reading the capture registers via the ECRD address, we have already capture two values. The ECRD read back will output CC4xC2V -> CC4xC2V (CC4xC3V value is lost)
- Timer X and Timer Y have 4 capture registers each and they are both configured with ECM = 1_B. At the moment that the software starts reading the capture registers via the ECRD address, we have already capture two values on Timer X and 4 on Timer Y. The ECRD read back will output CC4xC0V -> CC4xC1V -> CC4xC2V -> CC4xC3V -> CC4yC2V -> CC4yC2V (CC4yC3V value is lost)

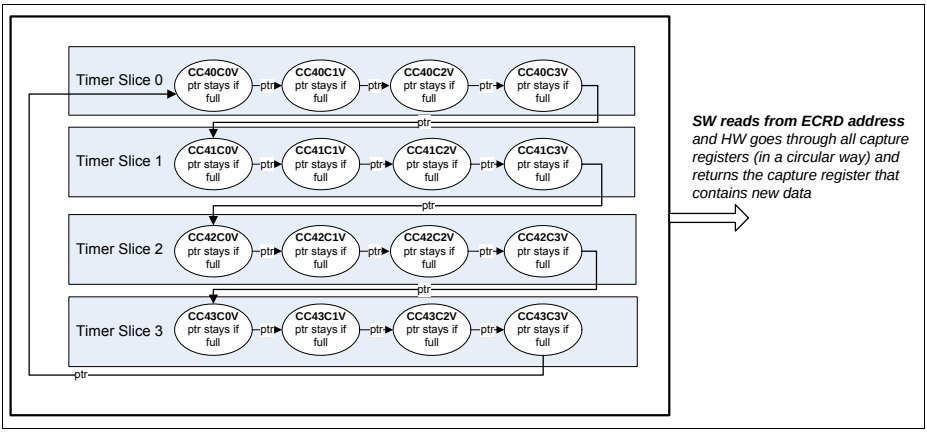

Figure 13 Extended Read Back access - example for CCU4 (CCU8 structure is the same)

Workaround

None.

CCU_AI.005 CCU4 and CCU8 External IP clock Usage

Each CCU4/CCU8 module offers the possibility of selecting an external signal to be used as the master clock for every timer inside the module Figure 1. External signal in this context is understood as a signal connected to other module/IP or connected to the device ports.

The user has the possibility after selecting what is the clock for the module (external signal or the clock provided by the system), to also select if this clock needs to be divided. The division ratios start from 1 (no frequency division) up to 32768 (where the selected timer uses a frequency of the selected clock divided by 32768).

This division is selected by the PSIV field inside of the CC4yPSC/CC8yPSC register. Notice that each Timer Slice (CC4y/CC8y) have a specific PSIV field, which means that each timer can operate in a different frequency.

Currently is only possible to use an external signal as Timer Clock when a division ratio of 2 or higher is selected. When no division is selected (divided by 1), the external signal cannot be used.

The user must program the PSIV field of each Timer Slice with a value different from 0000_B - minimum division value is /2.

This is only applicable if the Module Clock provided by the system (the normal default configuration and use case scenario) is not being used. In the case that the normal clock configured and programmed at system level is being used, there is not any type of constraints.

One should not also confuse the usage of an external signal as clock for the module with the usage of an external signal for counting. These two features are completely unrelated and there are not any dependencies between both.

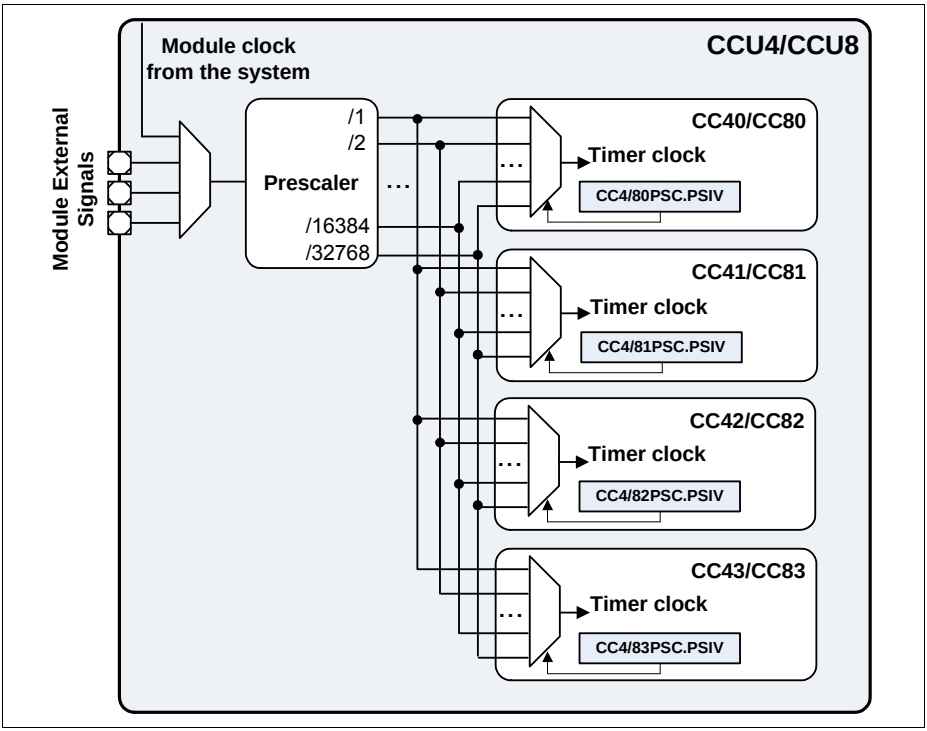

Figure 14 Clock Selection Diagram for CCU4/CCU8

Workaround

None.

CCU_AI.006 Value update not usable in period dither mode

Each CCU4/CCU8 timer gives the possibility of enabling a dither function, that can be applied to the duty cycle and/or period. The duty cycle dither is done to increase the resolution of the PWM duty cycle over time. The period dither is done to increase the resolution of the PWM switching frequency over time.

Each of the dither configurations is set via the DITHE field:

- DITHE = 00_B - dither disabled

- DITHE = 01_B - dither applied to the period (period value)
- DITHE = 10_B - dither applied to the duty-cycle (compare value)
- DITHE = 11_B - dither applied to the duty-cycle and period (compare an period value)

Whenever the dither function is applied to the period (DITHE = 10_B or DITHE = 11_B) and an update of the period value is done via a shadow transfer, the timer can enter a stuck-at condition (stuck at 0).

Implication

Period value update via shadow transfer cannot be used if dither function is applied to the period (DITHE programmed to 10_B or 11_B).

Workaround

None.

CPU_CM.001 Interrupted loads to SP can cause erroneous behavior

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:

1. LDR SP,[Rn],#imm
2. LDR SP,[Rn,#imm]!
3. LDR SP,[Rn,#imm]
4. LDR SP,[Rn]
5. LDR SP,[Rn,Rm]

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

1. LDR SP,[Rn],#imm
2. LDR SP,[Rn,#imm]!

Conditions

1. An LDR is executed, with SP/R13 as the destination
2. The address for the LDR is successfully issued to the memory system
3. An interrupt is taken before the data has been returned and written to the stack-pointer.

Implications

Unless the load is being performed to Device or Strongly-Ordered memory, there should be no implications from the repetition of the load. In the unlikely event that the load is being performed to Device or Strongly-Ordered memory, the repeated read can result in the final stack-pointer value being different than had only a single load been performed.

Interruption of the two write-back forms of the instruction can result in both the base register value and final stack-pointer value being incorrect. This can result in apparent stack corruption and subsequent unintended modification of memory.

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.

If repeated reads are acceptable, then the base-update issue may be worked around by performing the stack pointer load without the base increment followed by a subsequent ADD or SUB instruction to perform the appropriate update to the base register.

CPU_CM.004 VDIV or VSQRT instructions might not complete correctly when very short ISRs are used

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.

Conditions

1. The floating point unit is present and 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. 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 then this erratum cannot be observed.

Implications

The VDIV or VQSRT instruction does not complete correctly and the register bank and FPSCR are not updated, meaning that these registers hold incorrect, out of date, data.

Workaround

A workaround is only required if the floating point unit is present and enabled. A workaround is not required if the memory system inserts one or more wait states to every stack transaction.

There are two workarounds:

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

CPU_CM.005 Store immediate overlapping exception return operation might vector to incorrect interrupt

The Cortex-M4 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.

This erratum only affects systems where writeable memory locations can exhibit more than one wait state. For the XMC4000 Family only devices with external memory controller (EBC) used in Application are affected. All internal memory use zero wait state access.

Implications

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.If interrupt C is level based, then this interrupt will eventually become re-pending and subsequently be handled. If interrupt C is a single pulse interrupt, then there is a possibility that this interrupt will be lost.

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");
}
```

DAC_CM.001 DAC immediate register read following a write issue

In case a read access to a DAC register is done immediately after a write access to the same register, the `old` data value is returned, which was stored before the write access, and not the newly written one.

Workaround

In case of a series of write accesses to DAC registers, repeat the last write access. In case of a single write access, repeat this one. Then no read access can fail.

DAC_CM.002 No error response for write access to read only DAC ID register

The DAC ID register is a read only register. But in case a write access is done to it, no bus error response is returned. The DAC ID register value is kept, as intended.

Workaround

None.

DEBUG_CM.001 OCDS logic in peripherals affected by TRST

The OCDS logic in peripherals is erroneously reset if TRST is activated.

In the device the OCDS logic in the peripherals is kept in reset and therefore not available by default (after reset) because P0.8 is configured as TRST and the internal pull-down is active.

Workaround 1

Connect an external pull-up resistor to P0.8 to drive TRST high (inactive), if P0.8 is not used by the application.

Workaround 2

During software configuration program P0.8 as GPIO input, This configuration drives TRST internally to the inactive state.

Note: With this solution the debug functionality remains unavailable right after reset.

DEBUG_CM.002 CoreSight logic only reset after power-on reset

The CoreSight logic should also be reset with a debug reset (DBGRESET).

Opposed to this specification the debug reset does not have an effect on the CoreSight logic. Therefore CoreSight logic can only be reset by a power-on reset (PORESET).

Workaround

If the user quits the debug session and likes to leave the system clean, without a PORESET, the following steps have to be performed:

- Disable debug functions by disable of DHCSR.C_DEBUGEN bit in debug halting and status register.

- Disable HW breakpoints in FPB unit of each comparator by disable of FP_CTRL.ENABLE bit in flashpatch control register.
- Disable trace functions by disable of DEMCR.TRCENA bit in debug exception and monitor control register. This disables DWT, ITM, ETM and TPIU functions.

DSD_AI.001 Possible Result Overflow with Certain Decimation Factors

Certain combinations of CIC filter grade and oversampling rate (see below) can lead to an overflow within the CIC filter. These combinations must be avoided to ensure proper operation of the digital filter.

Critical combinations:

- CIC2 (CFMC/CFAC = 01_B) with oversampling rate of 182
- CIC3 (CFMC/CFAC = 10_B) with oversampling rate of 33, 41, 51, 65, 81, 102, 129, 162…182, 204
- CICF (CFMC/CFAC = 11_B) with oversampling rate of 129, 182

Note: Filter grade and oversampling rate are defined in register FCFGCx/FCFGAx. The shown oversampling rates are defined as CFMDF+1/CFADF+1.

Workaround

None.

DSD_AI.002 Timestamp can be calculated wrong

Some applications need to determine a result value at points of time in between two regular output values. An interpolation algorithm is then used to determine the point of time in relation to the last regular result.

The cycles consumed since the last regular result value can be calculated from the decimation factor FCFGCx.CFMDF and bit fields NVALCNT and CFMDCNT of TSTMPx register.

In the affected device the value of NVALCNT is calculated wrong upon underflow of CFMDCNT.

Implications

Calculation for cycles consumed (TICKS) since the last regular result value is done according to the following formula.

TICKS = NVALCNT * (CFMDF + 1) + (CFMDF – CFMDCNT) (1)

Upon underflow of CFMDCNT the following actions appear:

- NVALCNT is increased
- CFMDCNT is reloaded from CFMDF

Examples for expected (**Table 8**) and wrong (**Table 9**) behavior are shown below.

Table 8 Expected values using CFMDF = 15

NVALCNT	CFMDCNT	TICKS
2	3	44
2	2	45
2	1	46
2	0	47
3 - correct value	15	48 - correct value
3	14	49
...	...	...

The wrong behavior is that NVALCNT increases only one TICK after the underflow:

Table 9 Measured values using CFMDF = 15

NVALCNT	CFMDCNT	TICKS
2	3	44
2	2	45
2	1	46
2	0	47
2 - wrong value!	15	**32 - wrong value!**

Table 9 Measured values using CFMDF = 15 (cont'd)

NVALCNT	CFMDCNT	TICKS
3	14	49
...	...	...

The cycles consumed are therefore calculated wrong for the later case. Consequently the interpolation is wrong.

For long DSD periods a wrong interpolation can lead to substantial error.

Workaround 1

- Read TSTMPx register containing CFMDCNT and NVALCNT values
- While CFMDCNT is equal to FCFGCx.CFMDF repeat the read of TSTMPx
- Calculate cycles consumed

Workaround 2

- Read TSTMPx register containing CFMDCNT and NVALCNT values
- If CFMDCNT is equal to FCFGCx.CFMDF then increment NVALCNT
- Calculate cycles consumed

Note that this workaround delivers a wrong result for the case that TSTMPx is read at the same time as the last regular output occurred (see **Table 10** below).

Therefore in case of NVALCNT=0 the read of TSTCMPx may be repeated as described in Workaround 1.

Alternatively the wrong result must be regarded by the application.

Table 10 Values calculated with Workaround 2 using CFMDF = 4

NVALCNT	CFMDCNT	TICKS
0 - correct value	4	**5 - wrong value!**
0	3	1
0	2	2
0	1	3
0	0	4
0 - wrong value!	4	5 - this and later values are correct

Table 10 Values calculated with Workaround 2 using CFMDF = 4

NVALCNT	CFMDCNT	TICKS
1	3	6
...	...	...

DTS_CM.001 DTS offset calibration value limitations

When using the value $7F_H$ for offset calibration in DTSCON.OFFSET the Die Temperature Sensor may return invalid results in DTSSTAT.RESULT.

Implication

The value $7F_H$ (equivalent to -1) for DTSCON.OFFSET cannot be used.

Workaround

If the application needs a small negative offset then $7E_H$ (equivalent to -2) could be used.

ETH_AI.001 Incorrect IP Payload Checksum at incorrect location for IPv6 packets with Authentication extension header

When enabled, the Ethernet MAC computes and inserts the IP header checksum (IPv4) or TCP, UDP, or ICMP payload checksum in the transmitted IP datagram (IPv4 or IPv6) on per-packet basis. The Ethernet MAC processes the IPv6 header and the optional extension headers (if present) to identify the start of actual TCP, UDP, or ICMP payload for correct computation and insertion of payload checksum at appropriate location in the packet. The IPv6 header length is fixed (40 bytes) whereas the extension header length is specified in units of N bytes:

Extension Header Length Field Value x N bytes + 8 bytes
where N = 4 for authentication extension header and N = 8 for all other extension headers supported by the Ethernet MAC. If the actual payload bytes are less than the bytes indicated in the Payload Length field of the IP header, the Ethernet MAC indicates the IP Payload Checksum error.

If the payload checksum is enabled for an IPv6 packet containing the authentication extension header, then instead of bypassing the payload checksum insertion, the Ethernet MAC incorrectly processes the packet and inserts a payload checksum at an incorrect location. As a result, the packet gets corrupted, and it is dropped at the destination. The software should not enable the payload checksum insertion for such packets because the Integrity Check Value (ICV) in the authentication extension header is calculated and inserted considering that the payload data is immutable (not modified) in transit. Therefore, even if the payload checksum is correctly calculated and inserted, it results into a failure of the ICV check at the final destination and the packet is eventually dropped.

Workaround

The software should not enable the IP payload checksum insertion by the Ethernet MAC for Tx IPv6 packets with authentication extension headers. The software can compute and insert the IP payload checksum for such packets.

ETH_AI.002 Incorrect IP Payload Checksum Error status when IPv6 packet with Authentication extension header is received

The Ethernet MAC processes a TCP, UDP, or ICMP payload in the received IP datagrams (IPv4 or IPv6) and checks whether the received checksum field matches the computed value. The result of this operation is given as an IP Payload Checksum Error in the receive status word. This status bit is also set if the length of the TCP, UDP, or ICMP payload does not match the expected payload length given in the IP header.

In IPv6 packets, there can be optional extension headers before actual TCP, UDP, or ICMP payload. To compute and compare the payload checksum for such packets, the Ethernet MAC sequentially parses the extension headers, determines the extension header length, and identifies the start of actual TCP, UDP, or ICMP payload. The header length of all extension headers supported by the Ethernet MAC is specified in units of 8 bytes (Extension Header Length Field Value x 8 bytes + 8 bytes) except in the case of authentication extension header. For authentication extension header, the header length is specified in units of 4 bytes (Extension Header Length Field Value x 4 bytes + 8 bytes).

However, because of this defect, the Ethernet MAC incorrectly interprets the size of the authentication extension header in units of 8 bytes, because of which the following happens:

- Incorrect identification of the start of actual TCP, UDP, or ICMP payload
- Computing of incorrect payload checksum
- Comparison with incorrect payload checksum field in the received IPv6 frame that contains the authentication extension header
- Incorrect IP Payload Checksum Error status

As a result, the IP Payload checksum error status is generated for proper IPv6 packets with authentication extension header. If the Ethernet MAC core is programmed to drop such `error` packets, such packets are not forwarded to the host software stack.

Workaround

Disable dropping of TCP/IP Checksum Error Frames by setting Bit 26 (DT) in the Operation Mode Register (OPERATION_MODE). This enables the Ethernet MAC core to forward all packets with IP checksum error to the software driver. The software driver must process all such IPv6 packets that have payload checksum error status and check whether they contain the authentication extension header. If authentication extension header is present, the software driver should either check the payload checksum or inform the upper software stack to check the packet for payload checksum.

ETH_AI.003 Overflow Status bits of Missed Frame and Buffer Overflow counters get cleared without a Read operation

The DMA maintains two counters to track the number of frames missed because of the following:

- Rx Descriptor not being available
- Rx FIFO overflow during reception

The Missed Frame and Buffer Overflow Counter register indicates the current value of the missed frames and FIFO overflow frame counters. This register also has the Overflow status bits (Bit 16 and Bit 28) which indicate whether the

rollover occurred for respective counter. These bits are set when respective counter rolls over. These bits should remain high until this register is read.

However, erroneously, when the counter rollover occurs second time after the status bit is set, the respective status bit is reset to zero.

Effects

The application may incorrectly detect that the rollover did not occur since the last read operation.

Workaround

The application should read the Missed Frame and Buffer Overflow Counter register periodically (or after the Overflow or Rollover status bits are set) such that the counter rollover does not occur twice between read operations.

ETH_CM.002 MAC provides incorrect status and corrupts frames when RxFIFO overflow occurs on penultimate word of Rx frames of specific lengths

When RxFIFO is operating in the threshold mode, it may overflow when the received frame data is written faster than the speed at which the application reads from the RxFIFO. RxFIFO overflow is declared when a non-EOF word is received and the RxFIFO has only two locations available. The RxFIFO write controller (RWC) pushes the dummy end of frame (EOF) data and the Rx status in these two locations indicating overflow error to the application. The RWC drops the remaining bytes of the frame, timestamp status (if available), and corresponding status word received from the MAC receiver. Normally, when no overflow occurs, the timestamp (if available) is written in the RxFIFO after the EOF and with byte enable value as `all-zero` (used as tag). Subsequently, the status word received from the MAC receiver is written in the RxFIFO with `all-one` byte enables.

Conditions

This defect is observed when the following events occur:

- RxFIFO overflow is declared exactly when the RWC receives the penultimate word of Rx frame on MRI.
- EOF word is received in the next clock cycle along with the actual status.
- The EOF word has exactly one valid byte (Byte Enable on MRI = `all-zero`). This is possible only when the length of the packet, after CRC or PAD stripping (if enabled), is a multiple of the data-lane width + 1. For example. for 32-bit data bus, the packet is of length such as 9, 13, 17, and so on, and for 64-bit data bus, the packet is of length such as 17, 25, 33, and so on.

Implications

In the above sequence, the RWC writes the dummy EOF word followed by the RxStatus (with overflow status) with byte enable tag as `all-zero` instead of expected `all-one`. The RWC then performs an additional write of the actual status with byte enable tag as `all-ones` even though the third location may not be available (because previous content is not read by the RxFIFO Read Controller [RRC]). Therefore, the write pointer may cross over the read pointer, falsely indicating that the RxFIFO has enough space to push in the next frame.

When the RRC reads the overflow Rx status with incorrect tag (of byte enable = all zeros), it provides that as timestamp. The Rx status in the next location is read out from the RxFIFO and given out as Rx status word which is incorrect because it neither has the overflow error status nor the correct length of the received frame.

If the next frame is received before the synchronized read pointer increments and crosses over the write pointer, the RWC considers that the RxFIFO has enough space available (FIFO not full). Therefore, it pushes the frame and overwrites the existing unread frames in the RxFIFO. This corrupts the existing frames. The FIFO controllers recovers automatically after transferring a few corrupt or incorrect packets.

Workaround

Operate the RxFIFO in store-and-forward mode.

FCE_CM.001 Result value is wrong if read directly after last write

If a result register RESm is read directly after the last write of input data to the corresponding IRm register then the calculated result is wrong.

Workaround

Insert a wait cycle between last write and result read.

This can be accomplished by:

- reading the result twice or
- inserting a NOP instruction between last write and result read.

GPDMA_CM.001 Unexpected Block Complete Interrupt During Multi-Block Transfers

The GPMDA allows an interrupt to be generated on completion of a DMA block transfer to the destination. This interrupt is generated if the INT_EN (CTLx[0]) bit is set. On a channel enabled for multi-block transfers, the CTLx register is reprogrammed using either of the following methods:

- Block chaining using linked lists
- Auto-reloading

When CTLx is re-programmed using block-chaining of linked lists, interrupts can be enabled or disabled seperately for each block in the transfer. The block interrupt is generated from a combinational logic, which is coded such that, for a particular channel, if the 'RawBlock' register bit is set and the rawblock interrupt is unmasked, an interrupt is triggered soon as the INT_EN (CTLx[0]) bit is written as `1`, as shown in the equation below:

```
block_int = rawblock & (!maskblock) & int_en ;
```

This can cause a false block interrupt to be generated in multi-block transfers.

Conditions

1. Consider a multi-block transfer of three blocks(LLI0, LLI1, LLI2) on channelX, where SARx, DARx, CTLx are all re-programmed using linked lists.

2. For the first block, interrupts are not enabled; that is, LLI0.CTLx[0] = 0. For the second and third blocks, interrupts are enabled; that is, LLI1.CTLx[0] = 1, LLI2.CTLx[0] = 1.
3. Block interrupt for channel x is unmasked by writing to MaskBlock register.
4. After the first block transfer completes, the rawblock bit is set to 1; that is, RawBlock[0] = 1. At this point, no interrupt is generated because, for this block, int_en = 0; that is, LLI0.CTLx[0] = 0).
5. An LLI update occurs for the next block transfer, and SARx, DARx, and CTLx are re-programmed with the contents of LLI1.SARx, LL1.DARx, and LLI1.CTLx, respectively.
6. Since the RawBlock register has not been cleared by software after the first block completion, RawBlock[0] is still set to 1.
7. Because LLI1.CTLx[0] = 1, the int_en bit is set to `1` as soon as CTLx is updated with the contents of LLI1.CTLx[0]. This triggers a false Block Complete Interrupt at this point.

Implications

Unexpected Block Complete Interrupt can occur during Multi-Block Transfers.

Workaround

The software knows which blocks of the multi-block transfer are interrupt-enabled. Based on this, code the Interrupt Service Routine such that it keeps a count of the interrupts. It can then ignore the unwanted interrupts and service only the expected interrupts.In the example above, the software expects block interrupts only for LLI1 and LLI2, but not for LLI0. So the ISR can be coded to ignore the first interrupt and service the next two interrupts, as shown in the psuedo code below:

```
ISR  :
blk_flag++
If(blk_flag==1)
{
  clear_block_interrupt
  exit
} else {
  // do normal operation;
```

```
}
```

GPDMA_CM.002 GPDMA doesn't Accept Transfer During/In 2nd Cycle of 2-Cycle ERROR Response

In the GPDMA, the slave bus interface unit is coded such that, after the second cycle of a two-cycle error response, the logic transitions the state machine to the IDLE state and hence does not accept any transfer issued during the second cycle of the two-cycle error response.

Workaround

- Write software to not perform any actions that cause GPDMA to generate an error response.
- Ensure that any master that communicates to the GPDMA does not issue a transfer in the second cycle of a two-cycle error response.

LEDTS_AI.001 Delay in the update of FNCTL.PADT bit field

The touch-sense pad turn (PADT) value is updated, not at the end of the touch-sense time slice (ColA), but one time slice later (**Figure 15**).

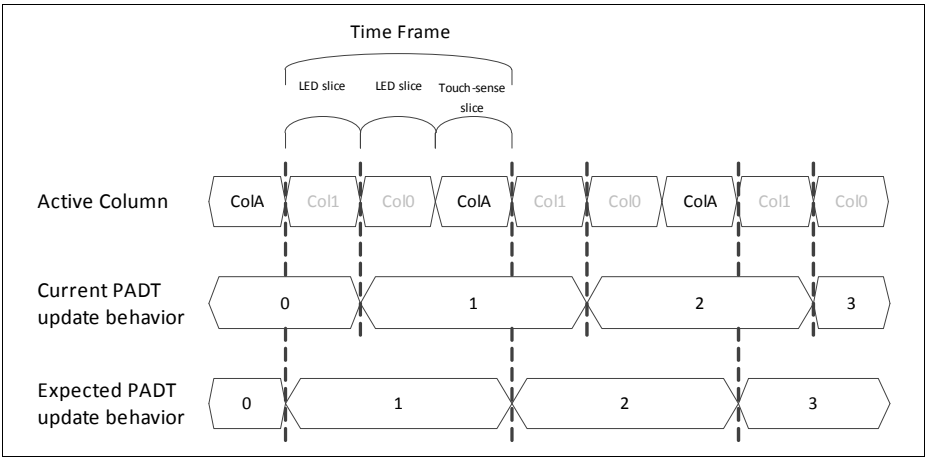

Figure 15 PADT update behavior

If the number of LED columns enabled is smaller than 2, the delay will affect the activation period of the current active pad. At the beginning of every new Col A, the value of the current PADT's compare register is updated to the internal compare register. However, the delay causes the value of the previous PADT's compare register is updated to the internal compare register instead. This means that the current active pad would be activated with the duration of the previous pad's oscillation window (**Figure 16**). In addition to this, when no LEDs are enabled, pad turn 0 will prevail for one time slice longer before it gets updated (**Figure 17**).

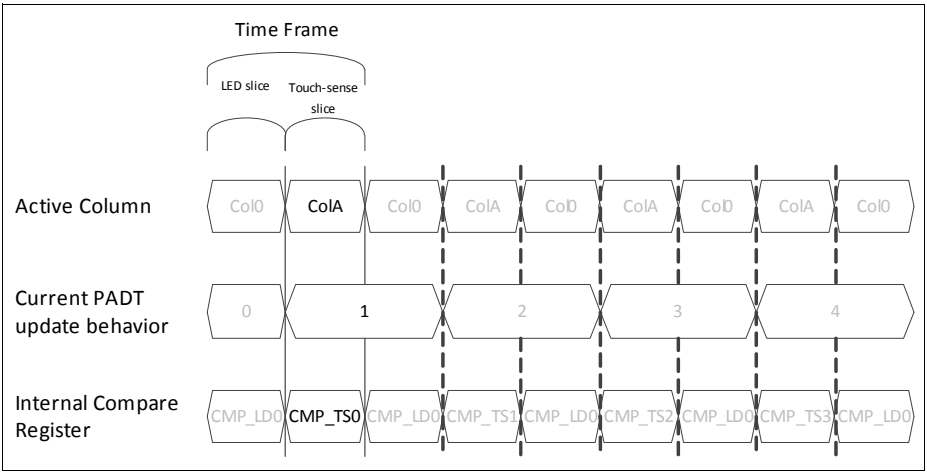

Figure 16 Effect of delay on the update of Internal Compare Register with 1 LED column enabled

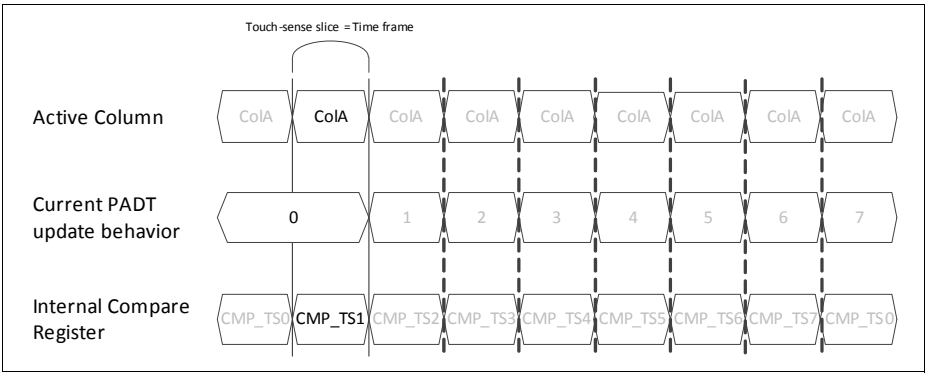

Figure 17 Pad turn 0 prevails for one time slice longer when no LEDs are enabled

If the number of LED columns enabled is 2 or more, the additional LED columns would provide some buffer time for the delay. So, at the start of a new touch-sense time slice, the update of PADT value would have taken place. Hence, the current active PADT compare register value is updated to the internal compare register (**Figure 18**).

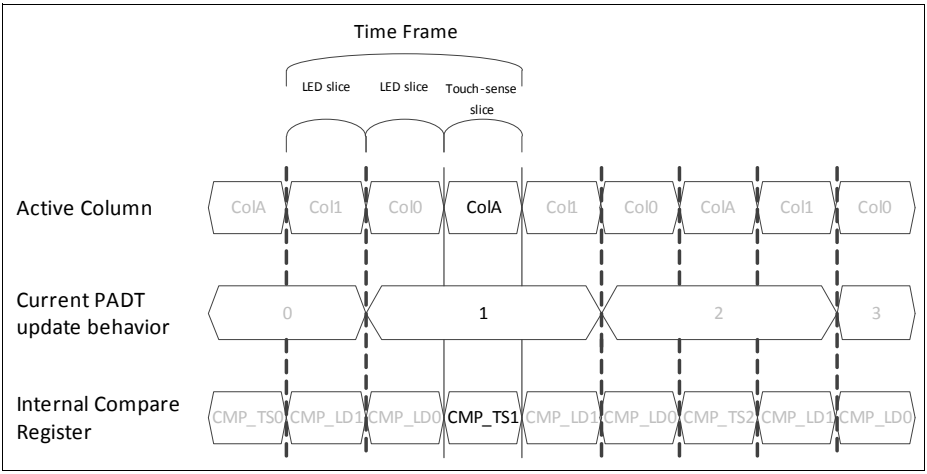

Figure 18 Internal Compare Register updated with correct compare register value with 2 LED columns enabled

Conditions

This delay in PADT update can be seen in cases where hardware pad turn control mode (FNCTL.PADTSW = 0) is selected and the touch-sense function is enabled (GLOBCTL.TS_EN = 1).

Workaround

This section is divided to two parts. The first part will provide a guide on reading the value of the bit field FNCTL.PADT via software. The second part will provide some workarounds for ensuring that the CMP_TS[x] values are aligned to the current active pad turn.

Workaround for reading PADT

Due to the delay in the PADT update, the user would get the current active pad turn when PADT is read in the time frame interrupt. However, this PADT value read differs when read in a time slice interrupt. This depends on the number of LED columns enabled and the active function or LED column in the previous time slice (**Table 11**). The bit field FNCTL.FNCOL provides a way of interpreting the active function or LED column in the previous time slice.

Table 11 PADT value as read in the time slice interrupt

No. of LED Columns Enabled	Previous active function / LED column	FNCTL.FNCOL	PADT value
0-1	Touch-sense or LED Col0	110_B or 111_B	Previous active pad turn
2-7	Touch-sense or first LED column after touch-sense	110_B or 111_B	Previous active pad turn
	Second LED column after touch-sense onwards	101_B to 000_B	Current or next active pad turn

Workaround for aligning CMP_TSx

One workaround is to use the software pad turn control. Then this issue can be avoided entirely because the pad turn update will have to be handled by software.

However, it is still possible to work around this issue when using the hardware pad turn control. In the previous section, it is known that when the number of LED columns enabled is smaller than 2, the current active pad is activated with the oscillation window of the previous active pad. This means that the current active pad is activated with the value programmed in the bit field CMP_TS[x-1] instead of CMP_TS[x]. There are two possible software workarounds for this issue:

1. At the end of the time frame interrupt service routine, software can prepare for the next active pad turn by programming the CMP_TS[x-1] bit field with the intended compare value for TSIN[x]. As an example, if the next active pad is TSIN[2], program CMP_TS[1] with the compare value intended for TSIN[2] (**Figure 19**).

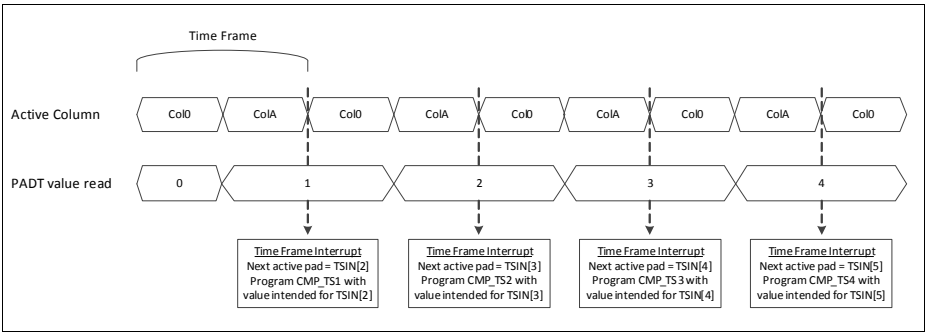

Figure 19 Software workaround demonstration

1. During the initialization phase, program the CMP_TS[x] bit fields with the left-shift factored in. Example: CMP_TS[0] for TSIN[1], CMP_TS[1] for TSIN[2], ... CMP[7] for TSIN[0].

PARITY_CM.001 Parity error signaling can be suppressed in write/read sequence

The device PSRAM and DSRAM offers parity protection. The parity information is stored byte wise. AHB memory access are executed word (32-bit) aligned.

Due to weakness of the memories AHB bus interface the parity error signaling **by AHB bus error** is suppressed if the following usage scenario occurs:

- if a parity error is present in a byte (D1) of a 32-bit aligned word and
- if another byte or half-word (D2) is written to the same word and
- if data which contains both parts (D1) and (D2) is read immediately after the write.

Parity error signaling **by parity error NMI trap** is still available.

The functional problem exists due to buffering of AHB write requests and therefore occurs only in immediate write/read sequence scenarios on the same 32-bit aligned memory cell.

Following types of access sequences are affected.

Table 12 Affected write/read sequences

write	read
byte	half-word
byte	word
half-word	word

Example

If a parity error is present in a byte @200002303$_H$ and if the byte @200002302$_H$ (same 32-bit word) is written and immediately after this the word @200002300$_H$ (or half-word @200002302$_H$) is read then no parity error by AHB bus error is signaled.

Workaround

If the described usage scenario can occur in the application then it is recommended to enable the **parity error trap** for the used PSRAM and/or DSRAM units. By this a **NMI** trap will be signaled to the CPU. Enabling is done by programming the SCU register PETE.

Note: The NMI trap occurs with a delay. Therefore program execution is likely to continue after the read access causing the parity error trap.

PARITY_CM.002 Clock limitations for ETH and SDMMC modules when using parity check of module SRAMs

The SRAM memories used by ETH and SDMMC (XMC4500 devices only) offer error detection by parity bit protection. If a parity error is detected then it is forwarded to SCU and if parity error detection is enabled by settings in SCU register PEEN then a trap request is triggered.

In affected devices the forwarding mechanism does not work with some clock settings.

Workaround

If parity detection shall be enabled then following clock setting limitations must be obeyed:

- For ETH: $f_{CPU} = f_{SYS}$ or CPU clock divider must be disabled (SCU register bit CPUCLKCR.CPUDIV = 0).
- For SDMMC: $f_{CPU} \geq f_{SDMMC}$+25%. For example if SDMMC shall operate with f_{SDMMC} at 48 MHz then f_{CPU} must be set for 60 MHz or higher.

PBA_CM.001 Bus error request suppressed in sequential write to peripheral bridge

The device peripherals DAC, DSD, CAN, CCU4x, CCU8x ERU1, LEDTS0, PORTS, POSIFx, SDMMC, USICx and VADC are accessed thru peripheral bridges PBA0 or PBA1. These bridges buffer writes and always acknowledge the access by HRESP=OKAY. If a buffered write is later forwarded to any of the peripherals connected and is causing a bus error then this error is normally flagged by a service request (NMI) to SCU.

In the erroneous case a bus error occuring in the first of two immediate sequential accesses to the bridge is suppressed and no service request is raised.

Implication

Bus error from peripherals can be unnoticed by application software.

Although the problem is often detected due to secondary effects during debugging there is a risk that it may occur during productive runtime.

Workaround

Application software must avoid immediate sequential writes to peripherals behind the same peripheral bridge.

PBA0 peripherals: DSD, CCU40, CCU41, CCU42, CCU8x, ERU1, POSIFx, USIC0 and VADC.

PBA1 peripherals: DAC, CAN, CCU43, LEDTS0, PORTS, SDMMC, USIC1 and USIC2.

PMU_CM.001 Branch from non-cacheable to cacheable address space instruction may corrupt the program execution

Two consecutive instruction fetch accesses, the first to the non-cacheable and the second to the cacheable address space may cause a corruption of the program flow. If the error occurs, the cached instruction at the target address is replaced with the opcode 0000_0000_H instead of the opcode of the correct instruction.

Conditions

One of the following cases may trigger the erroneous behavior:

1. In the normal program execution, a branch, function call or exception call operation, with the current instruction executed from the non-cacheable Flash address space and the branch target address in the cacheable Flash address space.
2. The CPU generates a speculative fetch access to the ROM address space when executing the BX LR instruction as an exception return to Thread mode and using the Process Stack Pointer (PSP), but not using the extended Floating-Point frame
   (=> EXC_RETURN = $FFFF\_FFFD_H$ stored in the Link Register LR; LR can be any GPR). This specualtive access is followed by an access to the cacheable Flash address space (=> the access to the actual branch target address).

Any of these cases results in two consecutive instruction fetch accesses in the code address space with

- the first an access to the non-cacheable Flash address space ($0C00\_0000_H$ − $0C0F\_FFFF_H$ or the ROM address space (0000_0000_H − 0000_3FFF_H)
- the second access in the cacheable Flash address space (0800_0000_H − $080F\_FFFF_H$), to be serviced from the Instruction Buffer, meaning the instruction is already stored in the Instruction Buffer of the Prefetch Unit (already executed before and not displaced/invalidated later in the program execution)

To see the problem from the normal program execution as described in the first case, the code allocation must be mixed, with some code segments allocated

in the non-cacheable and other code segments in the cacheable address space. In such an environment code branches between the different segments, e.g. a function call from a cached thread to a function in the non-cacheable address space which then returns back to the cached thread, may trigger the problem.

In the second case, even if the complete application code is allocated in the cacheable Flash address space, the CPU may generate a speculative fetch access to the ROM address 0000_0000_H, triggered by the BX LR instruction and with EXC_RETURN = $FFFF\_FFFD_H$, as described above in operation 2.

System considerations

- Only instruction fetches may trigger these accesses, data accesses are not affected.
- Instruction fetches to other address ranges than described above (e.g. PSRAM, DSRAM) are also not subject to the problem.
- The BX LR instruction can be used to return from regular functions and exception handlers alike. When the LR contains a regular address, the CPU will branch to that address. When LR contains a special EXC_RETURN code, the CPU does an exception return instead, reading the target address and restoring the processor status from the selected stack. The problem occurs only with EXC_RETURN = $FFFF\_FFFD_H$. Other system states result in different return codes, e.g. Handler mode instead of Thread mode, Floating Point state, or using the Main Stack Pointer use different EXC_RETURN codes. With any other EXC_RETURN code than $FFFF\_FFFD_H$ the CPU does not generate the speculative access to the address 0000_0000_H, thus not generating the critical access sequence of a non-cacheable access that is followed by a cacheable access.

Workaround

Allocate the complete code either in the cacheable or non-cacheable Flash address space, do not use mixed code allocation. This workaround covers all accesses out of the normal program flow. Equivalent to the allocation in the non-cacheable address space with respect to reduced execution performance, it is also possible to disable the Instruction Buffer with PREF_PCON.IBYP.

If the code is allocated in the cacheable Flash address space, the BX LR instruction must not be executed with the exception return code LR = FFFF_FFFD$_H$.

It is possible to replace the BX LR instruction with the following sequence:

1. PUSH LR
2. POP PC

This sequence does not generate the speculative ROM access, thus it does not generate the critical access sequence of a non-cacheable access that is followed by a cacheable access.

If the application allows, the critical exception return code of the BX LR instruction can be avoided by operating in different CPU state, e.g. if the application does not use the Process Stack Pointer.

PORTS_CM.001 P15_PDISC.[4,5] register bits cannot be written

The bits 4 and 5 of the register P15_PDISC cannot be modified by software and always retain their reset value 0_B. As a result of this, the digital input path of the related shared analog and digital input pins cannot be disabled.

Implications

Software that sets one or both of these bits and later reads P15_PDISC will not see the expected read value, but always reads 0_B for P15_PDISC.[4,5].

Workaround

None.

PORTS_CM.002 P0.9 Pull-up permanently active

A pull-up device on P0.9 is permanently active, disregarding any PORTS or peripheral configurations.

This is not the standard pull device under control of the PORTS, but it is R_{UID_PU}, a part of the USB device detection circuitry of the USB.ID function, mapped to P0.9. Its characteristic resistance is documented in the Data Sheet.

Implications

This pull device may have adverse effects on currents drawn or driven, as well as signal slopes and timings of the connected interfaces.

Workaround

None.

PORTS_CM.005 Different PORT register reset values after module reset

The PORTS registers can be reset independent of the reset of the system with SCU_PRSET1.PPORTSRS. After such a module reset, some PORTS registers have a reset value different to the reset value that is documented in the Reference Manual.

Table 13 PORTS registers reset values

Register	Sytem reset value	Module reset value
Pn_IOCR8	$0000\ 0000_H$	$2020\ 2020_H$ 1)
Pn_PDISC	$XXXX\ XXXX_H$ 2)	$0000\ 0000_H$
Pn_PDR0	$2222\ 2222_H$	$0000\ 0000_H$
Pn_PDR1	$2222\ 2222_H$	$0000\ 0000_H$

1) Only in XMC4500 devices.

2) Device and package dependent

Implications

The different value in Pn_IOCR8 configures the respective port pins Pn.[11:8] as inverted inputs instead of direct inputs. User software in Priviledged Mode can reconfigure them as needed by the application.

With the different value in Pn_PDISC of the digital ports the availability of digital pins in a device can no longer be verified via this register. Note that Pn_PDISC of pure digital ports is read-only; user software can't write to them.

The Pn_PDISC of the shared analog/digital port pins (P14 and P15) enables/disables the digital input path. After a system reset this path is

disabled, after a module reset enabled. User software in Priviledged Mode can reconfigure them as needed by the application.

The different value in the Pn_PDR registers configures output port pins with a “Strong-Sharp” output driver mode, as opposed to “Strong-Soft” driver mode after a system reset. This may result in a higher current consumption and more noise induced to the external system. User software in Priviledged Mode can reconfigure them as needed by the application.

Workaround

None.

PORTS_CM.007 P14 and P15 cannot be used in boundary scan test

P14 and P15 are analog ports with selectable digital input functionality. After reset the digital input functionality is disabled. Due to this the input value present at related pins is not visible inside the device.

Implications

The digital logic values present at package pins related to P14 and P15 cannot be captured in IEEE 1149.1 boundary scan test.

Workaround

None.

POSIF_AI.001 Input Index signal from Rotary Encoder is not decoded when the length is 1/4 of the tick period

Each POSIF module can be used as an input interface for a Rotary Encoder. It is possible to configure the POSIF module to decode 3 different signals: Phase A, Phase B (these two signals are 90° out of phase) and Index. The index signal is normally understood as the marker for the zero position of the motor Figure 1.

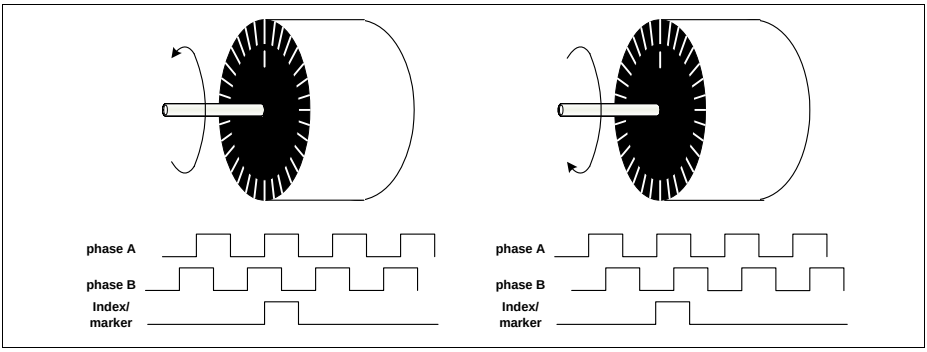

Figure 20 Rotary Encoder outputs - Phase A, Phase B and Index

There are several types of Rotary Encoder when it comes to length of the index signal:

- length equal or bigger than 1 tick period
- length equal or bigger than 1/2 tick period
- length equal or bigger than 1/4 tick period

When the index signal is smaller than 1/2 of the tick period, the POSIF module is not able to decode this signal properly, Figure 2 - notice that the reference edge of the index generation in this figure is the falling of Phase B, nevertheless this is an example and depending on the encoder type, this edge may be one of the other three.

Due to this fact it is not possible to use the POSIF to decode these type of signals (index with duration below 1/2 of the tick period).

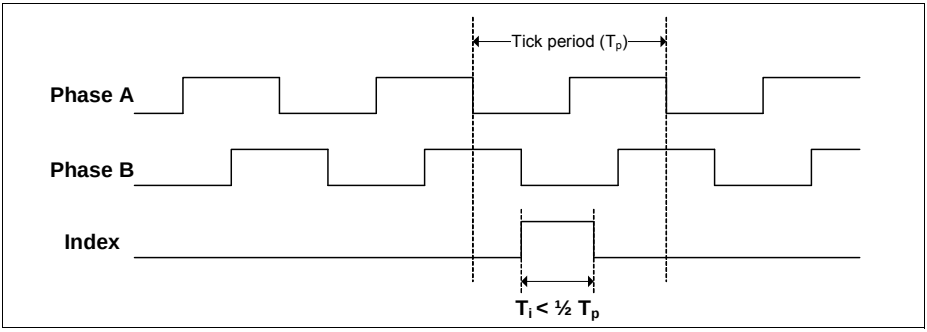

Figure 21 Different index signal types

Workaround

To make usage of the Index signal, when the length of this signal is less than 1/2 of the tick period, one should connect it directly to the specific counter/timer. This connection should be done at port level of the device (e.g. connecting the device port to the specific Timer/Counter(s)), Figure 3.

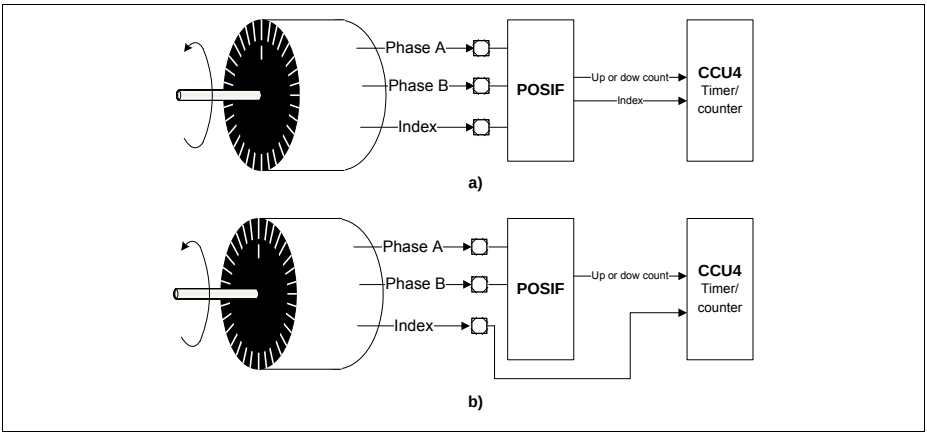

Figure 22 Index usage workaround - a) Non working solution; b) Working solution

RTC_CM.001 RTC event might get lost

RTC interrupt may get cleared in the RTC module before propagated to the CPU interrupt controller.

Single-shot RTC alarm may be missed. Periodic alarm events may be missed at the rate of once in 15 seconds.

Implications

RTC alarm interrupt alone cannot be reliably used for critical control functions.

Note: Wake-up from hibernate mode is not affected and RTC timer value is always correct.

Workaround

While in active mode use alternate timer for periodic event trigger and/or read the RTC timer value periodically.

SCU_CM.002 Missed wake-up event during entering external hibernate mode

Single-shot wake-up event and/or the first occurrence of a periodic wake-up event from hibernate mode may be missed if it occurs within 200 microseconds after hibernate mode request issued in software.

A wake-up event may be missed if it gets triggered during the process of entering hibernate mode i.e. between software access to the hibernate control register and the moment hibernate mode is effectively entered.

Implications

While entering hibernate mode it is required that expected wake-up event will not occur in the next 200 microseconds which not always may be guaranteed if external wake-up source is considered.

Workaround

Use of a backup wake-up event source generated internally with RTC may be applied in order to compensate for the missing trigger after a defined time-out.

SCU_CM.003 The state of HDCR.HIB bit of HCU gets updated only once in the register mirror after reset release

The state of HDCR.HIB bit of HCU gets updated only once in the register mirror in SCU after system reset. Any write access to this register gets propagated to hibernate domain but it will be not propagated back to the register mirror when altered by the hardware inside of the hibernate domain.

Implications

The state of HDCR.HIB cannot be effectively used for the purpose debugging of hibernate mode control software.

Workaround

For debugging of the hibernate mode control software observe the electrical states on the hibernate control pins in order to verify hibernate control circuit behavior.

SCU_CM.006 Deep sleep entry with PLL power-down option generates SOSCWDGT and SVCOLCKT trap

Entering the deep sleep mode with PLL power-down option (selected in DSLEEPCR register of SCU module) may result with system traps triggered by PLL watchdog (the SOSCWDGT trap) and/or loss-of-lock (the SVCOLCKT trap).

Implications

Occurrence of one of the enabled traps will result in an immediate wake-up from the deep sleep state, i.e. the deep sleep is effectively not entered.

Workaround

Disable SOSCWDGT and SVCOLCKT trap generation in TRAPDIS register of SCU before entering deep sleep mode with PLL power-down option selected.

SCU_CM.015 Functionality of parity memory test function limited

The device provides an interface to access the parity bits of the contained SRAM memories for test purpose. This feature is typically used by safety applications which must ensure that the parity mechanism is operational.

The test interface is based on using SCU registers PMTPR, PMTSR and MCHKCON. By those registers it is possible to implant (write) user defined

parity bits to selected memory cells. For checking of the parity value a read function is defined.

Due to synchronization issues wrong results can be produced for the PMTPR.PRD read value.

Implications

The values read back by PMTPR.PRD can be incorrect. Therefore it is not possible to directly check the parity information. Testing for the correct function of the parity logic is still possible by directed activation of parity errors.

Test of parity function

It is possible to test the correct function of the parity logic of PSRAM, DSRAM, USIC, CAN and USB memories using the following scheme:

1. Enable parity error generation and parity error trap generation using registers PEEN and PETE
2. Enable one target for parity test via registers PMTSR and MCHKCON
3. Write parity value for memory test to register bit field PMTPR.PWR
4. Write data value whose parity values conflicts with the parity value written in step 3 to memory location (@address0)
5. For PSRAM and DSRAM only: a 2nd write operation to another memory location (@address1) is required to flush the write buffer from step 4
6. Read back the content from the first memory location (@address0)
7. Parity error and NMI trap occurrence is expected

The NMI handler should check on the right content of the registers PEFLAG and TRAPSTAT and clear the related parity and NMI trap flags before returning.

SCU_CM.021 Registering of service requests in SRRAW register can fail

If a write to the service request clear register (SRCLR) occurs at the same time as one or multiple hardware request(s) then the hardware request(s) normally stored in SRRAW register is (are) lost.

The hardware request(s) and the cleared request(s) must not match to make the error occur.

Implications

If affected hardware requests (see SRRAW column in **Table 14**) are used by the application then these may get lost. The Workaround should be implemented.

Workaround

The interrupt routine assigned to an affected request must

- service the request(s) flagged in the SRSTAT register
- clear the corresponding bit(s) in SRRAW register via SRCLR register
- check the primary request source information of all affected and used service request sources and update the SRRAW via SRSET register accordly.

For checking of the primary request source please use information provided in **Table 14**. Example: if RTC bit RAWSTAT.RAI is set but SCU bit SRRAW.AI is not set then this request was lost. SRRAW should then be updated accordingly.

Table 14 Request source and related SRRAW register bit field

Request Source		SRRAW
Module	**Bit field**	**Bit field**
WDT	TIM counter value	PRWARN
RTC	RAWSTAT.RP*	PI
RTC	RAWSTAT.RAI	AI
DLR	OVRSTAT.LN*	DLROVR
SCU	HDSTAT.ULPWDG	ULP_WDG
SCU	MIRRSTS.HDSET	HDSET
SCU	MIRRSTS.OSCSICTRL	OSCSICTRL
SCU	MIRRSTS.RTC_CTR	RTC_CTR
SCU	MIRRSTS.RTC_ATIM0	RTC_ATIM0
SCU	MIRRSTS.RTC_ATIM1	RTC_ATIM1
SCU	MIRRSTS.RTC_TIM0	RTC_TIM0
SCU	MIRRSTS.RTC_TIM1	RTC_TIM1
SCU	MIRRSTS.RMX	RMX

SDMMC_CM.001 Unexpected interrupts after execution of CMD13 during bus test

This issue affects eMMC cards only.

The conditions for this behavior are as follows (all 2 conditions must be true):

- The host sends CMD19 (bus test pattern to a card), and driver issues CMD13 (SEND_STATUS command) to read the card status
- The transmit FSM is in Tx data state during bus testing procedure

The host controller may assert data timeout error SDMMC_INT_STATUS_ERR.DATA_TIMEOUT_ERR. As a consequence, unexpected interrupts may be generated.

Workaround

User should avoid sending CMD13 when bus testing is in progress.

SDMMC_CM.002 Unexpected Tx complete interrupt during R1b response

This issue affects both SD and eMMC cards.

R1b is a response type with an optional busy indication on the data line DAT[0]. SD and eMMC cards may send a busy response for the following commands:

Table 15 SD Commands with R1b response

CMD INDEX	Response Type	Abbreviation
CMD12	R1b	STOP_TRANSMISSION
CMD28	R1b	SET_WRITE_PROT
CMD29	R1b	CLR_WRITE_PROT
CMD38	R1b	ERASE

Table 16 eMMC Commands with R1b response

CMD INDEX	Response Type	Abbreviation
CMD5	R1b	SLEEP_AWAKE
CMD6	R1b	SWITCH
CMD12	R1b	STOP_TRANSMISSION
CMD28	R1b	SET_WRITE_PROT
CMD29	R1b	CLR_WRITE_PROT
CMD38	R1b	ERASE

When the card is in busy state for R1b, and driver sends the SEND_STATUS command (CMD13) to read the card status. Due to this CMD13, unexpected transfer complete interrupt SDMMC_INT_STATUS_NORM.TX_COMPLETE may be asserted by the host controller even before the busy signal gets released by the card.

Workaround

User should avoid sending CMD13 while the card is in busy state for R1b.

SDMMC_CM.003 SDMMC input pins cannot be released for other usage

SDMMC module inputs are directly connected to ports and cannot be disabled internally.

Implication

If SD card write protect (SDMMC.SDWC) or card detect ($\overline{\text{SDMMC.SDCD}}$) inputs are not required by the application then the related ports (P1.1) and (P1.10) cannot be used for other purpose.

Workaround

If either of the inputs is not required by the application then pull the related port to the desired logic level. This can be done by internal pull devices.

Do not assign other input or output functions to P1.1 or P1.10 when using SDMMC.

SDMMC_CM.004 Busy response from card in write resume operation not detected

The SDIO/SD device may support a suspend/resume mode for multi-function SDIO or a combo card. The host may temporarily halt a data transfer operation to one function or memory (suspend) in order to free the bus for a higher priority transfer to a different function or memory.The host controler can resume the write two cycles after the response end bit from Card. However, the controller fails to detect this and starts driving data to the card resulting in a collision.

Workaround

None.

SDMMC_CM.005 Controller sends other command when Auto CMD12 enabled

As per the SD physical layer specification, the NRC (timing between end bit of response to the start bit of next command) clock cycle should be 8 clocks. This applies for auto commands also. However, the controller sends the second command before 8 clocks which is a violation of the specification. The device may not sample the command correctly. This issue affects both SD and eMMC cards.

Workaround

Set SDMMC_TRANSFER_MODE.ACMD_EN = 00_B to disable auto command mode.

SDMMC_CM.006 Stream write issue due to wrong FIFO handling causes data corruption in eMMC mode

During stream write operation for eMMC cards the controller will stop the SD clock when the write FIFOs are empty. The related logic does not handle the two clocks turnaround cycles between internal finite state machine (FSM) and the top IO. As a consequence the last two bits of the previous set of data are not driven out on the DATA line.

This issue affects eMMC cards only.

Workaround

None.

SDMMC_CM.007 Consecutive write to the same register in SD clock domain

If the software driver programs the same register as back to back write (two consecutive write) before previous one is synchronized, the register does not sync into the destination domain. Any registers that are synchronized to SD clock domain as control information will be affected if written multiple times to set or reset multiple bits in the same register. The following registers are synchronized to the SD clock (f_{SDMMC}).

BLOCK_SIZE

BLOCK_COUNT

ARGUMENT1

TRANSFER_MODE

COMMAND

HOST_CTRL

BLOCK_GAP_CTRL

Workaround

Ensure that the AHB clock frequency (f_{PERIPH}) is not greater than 4 times the SD clock (f_{SDMMC}) or program the software driver in way that two consecutive write to the listed register is not used.

SDMMC_CM.008 Receive state machine hangs if driver programs stop at block gap request using CMD18 when receive buffers are full

In case of CMD18 (read multiple blocks), data blocks are received in the read FIFO which will then be transferred to the data SRAM of the system memory. The host controller will stop the clock supplied to SD/MMC card (SDMMC.CLK_OUT) and waits until one block is transferred to the memory, to delay the next block from the card. During this situation if the host driver sets “stop at block gap request”, the host controller receiver state machine hangs and it never comes of out a particular state. Block gap event is also not generated by the host controller.

This issue affects both SD and eMMC cards.

Workaround

User shall use CMD17 (read single block) or issue a software reset and re-initialize the SDMMC module in case of CMD18 is used.

SDMMC_CM.009 Latching current value in the response register

When the peripheral clock frequency is less than half of SDMMC clock ($f_{PERIPH} < f_{SDMMC}/2$) then the controller fails to store the response contents correctly in to the response register. Hence during resume operation, the driver will not get the expected bits (DF Flag) set in the response register for the resume command and the transaction will fail.

Workaround

Ensure that f_{PERIPH} is greater than $f_{SDMMC}/2$.

STARTUP_CM.001 CAN Bootstrap Loader

The oscillator start up detection by device firmware does not check for a required stable frequency lock. Therefore is not possible to support an entire spectrum of standard XTAL input frequencies in the CAN BSL boot mode. As a result the device may not answer the initial CAN frame.

Workaround

None.

USB_CM.002 GAHBCFG.GlblIntrMsk not cleared with a software reset

When the application issues a software reset to the core through the GRSTCTL.CSftRst bit, the GAHBCFG.GlblIntrMsk bit is not reset to 0.

Therefore, an interrupt will be generated in case any of the individual interrupt mask bit (in GINTMSK) is unmasked after the software reset by the application.

Workaround

The workaround is to clear GAHBCFG.GlblIntrMsk to 0 immediately after GRSTCTL.CSftRst is programmed for software reset.

USB_CM.003 Endpoint NAK not sent in Device Class applications with multiple endpoints enabled

In device descriptor DMA mode, the USB 2.0 OTG core does not send NAK handshake for all OUT endpoints once the transfer is complete.

This can be a problem for an application with high latency if it cannot re-enable the endpoint after the transfer is completed on the OUT endpoint.

If the host sends further OUT tokens when the endpoint is disabled, this packet blocks the RxFIFO till the application re-enables the endpoint to read out the packet. Blocking the RxFIFO results in all the other OUT endpoints not receiving any further data. Eventually, the RxFIFO becomes full.

Implications

The bug affects Communication Device Class (AMC) applications (e.g. Ethernet over USB) where multiple endpoints are enabled. When using the recommended Infineon USB device software stacks, the issue will be handled and no further workaround is needed.

Workaround

The application needs to set MTRF=1 for the OUT endpoints. This ensures that the OUT endpoints do not get disabled and hence the RxFIFO blocking limitation is not seen.

When MTRF=1, in order to ensure that there is no BNA (Buffer Not Available) scenario, the application needs to set a long descriptor chain for the OUT endpoints. When MTRF=1, the OUT EP is not disabled and the application and the core share the same descriptor chain simultaneously.

USB_CM.004 USB core is not able to detect resume or new session request after PHY clock is stopped

The control bit PCGCCTL.StopPclk is intended for the application to stop the PHY clock when USB is suspended, the session is not valid, or the device is disconnected.

However, in the current implementation, it also disables wrongly the logic to detect the USB resume and Session Request Protocol (for USB core with OTG capability) signalling.

Implications

If the PHY clock is stopped by setting the bit StopPclk to 1 following a USB suspend or session end, the USB core is not able to detect resume or new session request. Detection is possible again only after the clock gating is removed by clearing the bit StopPclk to 0.

Workaround

The PHY clock must not be stopped with the bit StopPclk for the cases where the application relies on the detection of resume or new session request to remove the clock gating.

USIC_AI.005 Only 7 data bits are generated in IIC mode when TBUF is loaded in SDA hold time

When the delay time counter is used to delay the data line SDA (HDEL > 0), and the empty transmit buffer TBUF was loaded between the end of the acknowledge bit and the expiration of programmed delay time HDEL, only 7 data bits are transmitted.

With setting HDEL=0 the delay time will be t_{HDEL} = 4 x $1/f_{SYS}$ + delay (approximately 60ns @ 80MHz).

Workaround

- Do not use the delay time counter, i.e use only HDEL=0 (default),
  or
- write TBUF before the end of the last transmission (end of the acknowledge bit) is reached.

USIC_AI.006 Dual SPI format not supported

Dual SPI format is not supported in SSC mode. Therefore, user should always configure either the standard SPI or Quad SPI format in this mode.

Workaround

None.

USIC_AI.007 Protocol-related argument and error bits in register RBUFSR contain incorrect values following a received data word

The protocol-related argument and error bits (PAR and PERR respectively) in register RBUFSR contain incorrect values following a received data word. This leads to the following errors:

Table 17

Protocol	Error due to incorrect PAR and PERR values
ASC	• Received parity bit (RBUFSR.PAR) and result of the parity check (RBUFSR.PERR) are incorrect. • When a data word is received, an alternate receive event (PSR.AIF) may be indicated instead of a receive event (PSR.RIF) even though parity mode is disabled.
SSC	• Received parity bit (RBUFSR.PAR) and result of parity check (PSR.PAERR) are incorrect. • The first data word of a frame may be indicated by a receive event (PSR.RIF) instead of an alternate receive event (PSR.AIF). Similarly, a data word that is not the first word of a frame may be indicated by PSR.AIF instead of PSR.RIF.
IIC	• Received acknowledge bit in RBUFSR.PAR is incorrect. • The first data word of a frame may be indicated by a receive event (PSR.RIF) instead of an alternate receive event (PSR.AIF). Similarly, a data word that is not the first word of a frame may be indicated by PSR.AIF instead of PSR.RIF.
IIS	• Sampling of condition WA = 1 may be indicated by a receive event (PSR.RIF) instead of an alternate receive event (PSR.AIF). Similarly, sampling of condition WA = 0 may be indicated by PSR.AIF instead of PSR.RIF.

Workaround

The workarounds are summarized below:

Table 18

Protocol	Workaround
ASC	• Parity mode cannot be used. • To check if a data word is received, both PSR.RIF and PSR.AIF flags need to be monitored. If interrupts are used, interrupt service handlers need to be set up for both interrupt sources.
SSC	• Parity mode cannot be used. • To check if a data word is received, both PSR.RIF and PSR.AIF flags need to be monitored. If interrupts are used, interrupt service handlers need to be set up for both interrupt sources. • To check if a data word is the first data word of a frame, the bit RBUFSR.SOF can be used.
IIC	• To check for the acknowledge bit, bit 8 of the receive buffer RBUF can be used. • To check if a data word is received, both PSR.RIF and PSR.AIF flags need to be monitored. If interrupts are used, interrupt service handlers need to be set up for both interrupt sources. • To check if a data word is the first data word of a frame, bit 9 of RBUF can be used.
IIS	• To check if a data word is received, both PSR.RIF and PSR.AIF flags need to be monitored. If interrupts are used, interrupt service handlers need to be set up for both interrupt sources. • To check the sampled value of WA, the bit PSR.WA can be used.

USIC_AI.008 SSC delay compensation feature cannot be used

SSC master mode and complete closed loop delay compensation cannot be used. The bit DX1CR.DCEN should always be written with zero to disable the delay compensation.

Workaround

None.

USIC_AI.009 Baud rate generator interrupt cannot be used

The baud rate generator interrupt cannot be used. The bit CCR.BRGIEN must always be written with zero to disable baud rate generator interrupt generation.

Workaround

None.

USIC_AI.010 Minimum and maximum supported word and frame length in multi-IO SSC modes

The minimum and maximum supported word and frame length in multi-IO SSC modes are shown in the table below:

Table 19

Multi-IO SSC Modes	Word Length (bits)		Frame Length (bits)	
	Minimum	Maximum	Minimum	Maximum
Dual-SSC	4	16	4	64
Quad-SSC	8	16	8	64

Workaround

If a frame length greater than 64 data bits is required, the generation of the master slave select signal by SSC should be disabled by PCR.MSLSEN.

To generate the master slave select signal:

- Configure the same pin (containing the SELOx function) to general purpose output function instead by writing 10000_B to the pin's input/output control register (Pn_IOCRx.PCy); and
- Use software to control the output level to emulate the master slave select signal

This way, multiple frames of 64 data bits can be made to appear as a single much larger frame.

USIC_AI.011 Write to TBUF01 has no effect

Writing data to Transmit Buffer Input Location 01 (register TBUF01 at offset address 084_H) does not load the data to the transmit buffer (TBUF).

Workaround

Use registers TBUF00 or TBUFx (x = 02 to 31) to load data to the transmit buffer.

If the Transmit Control Information (TCI) value of 00001_B needs to be generated together with the load to TBUF, use a FIFO setup and Transmit FIFO Buffer Input location 01 (register IN01 at offset address 184_H) instead.

USIC_AI.013 SCTR register bit fields DSM and HPCDIR are not shadowed with start of data word transfer

The bit fields DSM and HPCDIR in register SCTR are not shadowed with the start of a data word transfer.

Workaround

If the transfer parameters controlled by these bit fields need to be changed for the next data word, they should be updated only after the current data word transfer is completed, as indicated by the transmit shift interrupt PSR.TSIF.

USIC_AI.014 No serial transfer possible while running capture mode timer

When the capture mode timer of the baud rate generator is enabled (BRG.TMEN = 1) to perform timing measurements, no serial transmission or reception can take place.

Workaround

None.

USIC_AI.015 Wrong generation of FIFO standard transmit/receive buffer events when TBCTR.STBTEN/RBCTR.SRBTEN = 1

Transmit FIFO buffer modes selected by TBCTR.STBTEN = 1 generates a standard transmit buffer event whenever TBUF is loaded with the FIFO data or there is a write to INxx register, except when TRBSR.TBFLVL = TBCTR.LIMIT. This is independent of TBCTR.LOF setting.

Similarly, receive FIFO buffer modes selected by RBCTR.SRBTEN = 1 generates a standard receive buffer event whenever data is read out from FIFO or received into the FIFO, except when TRBSR.RBFLVL = RBCTR.LIMIT. This is independent of RBCTR.LOF setting.

Both cases result in the wrong generation of the standard transmit and receive buffer events and interrupts, if interrupts are enabled.

Workaround

Use only the modes with TBCTR.STBTEN and RBCTR.SRBTEN = 0.

USIC_AI.016 Transmit parameters are updated during FIFO buffer bypass

Transmit Control Information (TCI) can be transferred from the bypass structure to the USIC channel when a bypass data is loaded into TBUF. Depending on the setting of TCSR register bit fields, different transmit parameters are updated by TCI:

- When SELMD = 1, PCR.CTR[20:16] is updated by BYPCR.SELO (applicable only in SSC mode)
- When WLEMD = 1, SCTR.WLE and TCSR.EOF are updated by BYPCR.BWLE
- When FLEMD = 1, SCTR.FLE[4:0] is updated by BYPCR.BWLE
- When HPCMD = 1, SCTR.HPCDIR and SCTR.DSM are updated by BHPC
- When all of the xxMD bits are 0, no transmit parameters will be updated

However in the current device, independent of the xxMD bits setting, the following are always updated by the TCI generated by the bypass structure, when TBUF is loaded with a bypass data:

- WLE, HPCDIR and DSM bits in SCTR register

- EOF and SOF bits in TCSR register
- PCR.CTR[20:16] (applicable only in SSC mode)

Workaround

The application must take into consideration the above behaviour when using FIFO buffer bypass.

USIC_AI.018 Clearing PSR.MSLS bit immediately deasserts the SELOx output signal

In SSC master mode, the transmission of a data frame can be stopped explicitly by clearing bit PSR.MSLS, which is achieved by writing a 1 to the related bit position in register PSCR.

This write action immediately clears bit PSR.MSLS and will deassert the slave select output signal SELOx after finishing a currently running word transfer and respecting the slave select trailing delay (T_{td}) and next-frame delay (T_{nf}).

However in the current implementation, the running word transfer will also be immediately stopped and the SELOx deasserted following the slave select delays.

If the write to register PSCR occurs during the duration of the slave select leading delay (T_{ld}) before the start of a new word transmission, no data will be transmitted and the SELOx gets deasserted following T_{td} and T_{nf}.

Workaround

There are two possible workarounds:

- Use alternative end-of-frame control mechanisms, for example, end-of-frame indication with TSCR.EOF bit.
- Check that any running word transfer is completed (PSR.TSIF flag = 1) before clearing bit PSR.MSLS.

USIC_AI.020 Handling unused DOUT lines in multi-IO SSC mode

In multi-IO SSC mode, when the number of DOUT lines enabled through the bit field CCR.HPCEN is greater than the number of DOUT lines used as defined in the bit field SCTR.DSM, the unused DOUT lines output incorrect values instead of the passive data level defined by SCTR.PDL.

Implications

Unintended edges on the unused DOUT lines.

Workaround

To avoid unintended edges on the unused DOUT lines, it is recommended to use the exact number of DOUT lines as enabled by the hardware controlled interface during a multi-IO data transfer.

WDT_CM.001 No overflow is generated for WUB default value

The Window Watchdog Timer (WDT) does not generate an overflow event if the default counter value $FFFFFFFF_H$ is used in register WUB.

Implications

Without an timer overflow no reset or pre-warning is requested. For other WUB values the WDT operates correctly and a reset or pre-warning is requested upon WDT overflow.

Workaround

Do not use $FFFFFFFF_H$ as counter value.

3 Deviations from Electrical- and Timing Specification

The errata in this section describe deviations from the documented electrical- and timing specifications.

DAC_CM.P001 INL parameter limits violated by some devices

At some devices the DAC module violates the Integral Nonlinearity (INL) parameter limits of ±4 LSB, especially for very cold temperatures. These devices can show INL values up to ±5.5 LSB, measured with best straight line method.

Note: Integral Nonlinearity (INL) is defined as the max. deviation of the output characteristic against a straight line. Selecting the straight line which gives best INL number, is called best straight line method.

POWER_CM.P002 Minimum limit of external buffer capacitor C_{EXT}

The XMC4500 Data Sheet defines a typical external buffer capacitor C_{EXT} = 10 µF.

A very small buffer capacitor can be insufficient to ensure a stable power supply under high load steps (ΔI_{PLS} ≥ 50 mA). With an insufficient capacitor such a high load step may result in a reset triggered by the integrated supply monitor.

To ensure a stable power supply under high load steps, the actual value (not nominal value!) of the external buffer capacitor must be C_{EXT} ≥ 6.8 µF.

As an example, a capacitor with nominal 10 uF capacitance, and a cumulative tolerance (manufacturing, temperature) of minimum -32% just fulfills this limit.

Note: In addition to C_{EXT}, each V_{DDC} pin must be connected with C = 100 nF capacitors to V_{SS}.

Table 20 Power Sequencing Parameters

Parameter	Symbol	Values			Unit	Note / Test Condition
		Min.	Typ.	Max.		
External Buffer Capacitor on V_{DDC}	C_{EXT} SR	6.8	10	–	µF	In addition C = 100 nF capacitor on each V_{DDC} pin

POWER_CM.P004 Current consumption while $\overline{\text{PORST}}$ low can exceed specified value

The Data Sheet defines a maximum current consumption while the device is held in reset via $\overline{\text{PORST}}$, I_{DDP_PORST}.

Additional measurements have shown that the actual values can exceed the current values defined in the Data Sheet. The table below lists the updated values.

Table 21 Power Supply Parameters

Parameter	Symbol	Values			Unit	Note / Test Condition
		Min	Typ	Max		
I_{DDP} current at $\overline{\text{PORST}}$ Low	I_{DDP_PORST} CC	–	5	9	mA	V_{DDP} = 3.3 V, T_A = 25 °C
		–	16	60	mA	V_{DDP} = 3.6 V, T_J= 150 °C

4 Application Hints

The errata in this section describe application hints which must be regarded to ensure correct operation under specific application conditions.

ADC_AI.H003 Injected conversion may be performed with sample time of aborted conversion

For specific timing conditions and configuration parameters, a higher prioritized conversion c_i (including a synchronized request from another ADC kernel) in cancel-inject-repeat mode may erroneously be performed with the sample time parameters of the lower prioritized cancelled conversion c_c. This can lead to wrong sample results (depending on the source impedance), and may also shift the starting point of following conversions.

The conditions for this behavior are as follows (all 3 conditions must be met):

1. **Sample Time setting**: injected conversion c_i and cancelled conversion c_c use different sample time settings, i.e. bit fields `STC*` in the corresponding Input Class Registers for c_c and for c_i (`GxICLASS0/1, GLOBICLASS0/1`) are programmed to different values.
2. **Timing condition**: conversion c_i starts during the first f_{ADCI} clock cycle of the sample phase of c_c.
3. **Configuration parameters**: the ratio between the analog clock f_{ADCI} and the arbiter speed is as follows:

   $$N_A > N_D*(N_{AR}+3),$$

   with
   a) N_A = ratio f_{ADC}/f_{ADCI} (N_A = 2 .. 32, as defined in bit field `DIVA`),
   b) N_D = ratio f_{ADC}/f_{ADCD} = number of f_{ADC} clock cycles per arbitration slot (N_D = 1 .. 4, as defined in bit field `DIVD`),
   c) N_{AR} = number of arbitration slots per arbitration round (N_{AR} = 4, 8, 16, or 20, as defined in bit field `GxARBCFG.ARBRND`).

Bit fields `DIVA` and `DIVD` mentioned above are located in register `GLOBCFG`.

As can be seen from the formula above, a problem typically only occurs when the arbiter is running at maximum speed, and a divider $N_A > 7$ is selected to obtain f_{ADCI}.

Recommendation 1

Select the same sample time for injected conversions c_i and potentially cancelled conversions c_c, i.e. program all bit fields `STC*` in the corresponding Input Class Registers for c_c and for c_i (`GxICLASS0/1, GLOBICLASS0/1`) to the same value.

Recommendation 2

Select the parameters in register `GLOBCFG` and `GxARBCFG` according to the following relation:

$N_A \leq N_D*(N_{AR}+3)$.

ADC_AI.H004 Completion of Startup Calibration

Before using the VADC the startup calibration must be completed.

The calibration is started by setting GLOBCFG.SUCAL. The active phase of the calibration is indicated by GxARBCFG.CAL = 1. Completion of the calibration is indicated by GxARBCFG.CAL = 0.

When checking for bit CAL = 1 immediately after setting bit SUCAL, bit CAL might not yet be set by hardware. As a consequence the active calibration phase may not be detected by software.The software may use the following sequence for startup calibration:

1. GLOBCFG.SUCAL = 1
2. Wait for GxARBCFG.CAL = 1
3. Check for GxARBCFG.CAL = 0 before starting a conversion

Make sure that steps 1 and 2 of this sequence are not interrupted to avoid a deadlock situation with waiting for GxARBCFG.CAL = 1.

ADC_AI.H008 Injected conversion with broken wire detection

If a higher prioritized injected conversion c_i (in cancel-inject-repeat mode) using the broken wire detection feature (GxCHCTRy.BWDEN = 1_B) interrupts a lower prioritized conversion c_c before start of the conversion phase of c_c, the following

effects will occur for the injected conversion c_i (independent of the recommendations in ADC_AI.H003):

1. The effective sample time is either doubled, or it is equal to the sample time of the lower prioritized cancelled conversion c_c. This will shift the starting point of following conversions, and may lead to wrong sample results if the sample time for c_c is considerably shorter than the programmed sample time for c_i (depending on the source impedance).
2. The preparation phase for c_i may be skipped, i.e. during the effective sample phase (as described above), the selected channel is sampled without precharging the capacitor network to the level selected for the broken wire detection. Depending on the synchronization between c_i and c_c, this may increase the time until a broken connection is detected.

The interrupted conversion c_c will be correctly restarted after completion of the injected conversion c_i.

Recommendation

Perform injected conversions without enabling the broken wire detection feature, and follow the recommendations given in ADC_AI.H003.

Alternatively, configure the trigger source that includes channels using the broken wire detection feature such that it will not cancel other conversions. This can be achieved by setting the priority of the request source s to the lowest priority (GxARBPR.PRIOs = 00_B), or by setting the conversion start mode to “wait-for-start mode” (GxARBPR.CSMs = 0_B).

ADC_TC.H011 Bit DCMSB in register GLOBCFG

The default setting for bit DCMSB (Double Clock for the MSB Conversion) in register GLOBCFG is 0_B, i.e. one clock cycle for the MSB conversion step is selected.

DCMSB = 1_B is reserved in future documentation and must not be used.

ETH_AI.H001 Sequence for Switching between MII and RMII Modes

When switching between MII and RMII modes is required, the ETH module must be in a defined state to avoid unpredictable behavior.

Therefore, it is recommended to use the defined sequence listed below:

1. Finish running transfers and make sure that transmitters and receivers are set to stopped state:
   a) Check the RS and TS status bits in ETH0_STATUS register.
   b) Check that ETH0_DEBUG register content is equal to the reset value.
2. Wait until a currently running interrupt is finished and globally disable interrupts.
3. Apply and release reset to ETH0 module by writing to corresponding bit fields of PRSET2 and PRCLR2 registers.
4. Initialize the new mode (MII or RMII).
5. Apply software reset by writing to ETH0_BUS_MODE.SWR bit.

MultiCAN_AI.H005 TxD Pulse upon short disable request

If a CAN disable request is set and then canceled in a very short time (one bit time or less) then a dominant transmit pulse may be generated by MultiCAN module, even if the CAN bus is in the idle state.

Example for setup of the CAN disable request:

`CAN_CLC.DISR` = 1 and then `CAN_CLC.DISR` = 0

Workaround

Set all INIT bits to 1 before requesting module disable.

MultiCAN_AI.H006 Time stamp influenced by resynchronization

The time stamp measurement feature is not based on an absolute time measurement, but on actual CAN bit times which are subject to the CAN resynchronization during CAN bus operation.The time stamp value merely indicates the number of elapsed actual bit times. Those actual bit times can be

shorter or longer than nominal bit time length due to the CAN resynchronization events.

Workaround

None.

MultiCAN_AI.H007 Alert Interrupt Behavior in case of Bus-Off

The MultiCAN module shows the following behavior in case of a bus-off status:

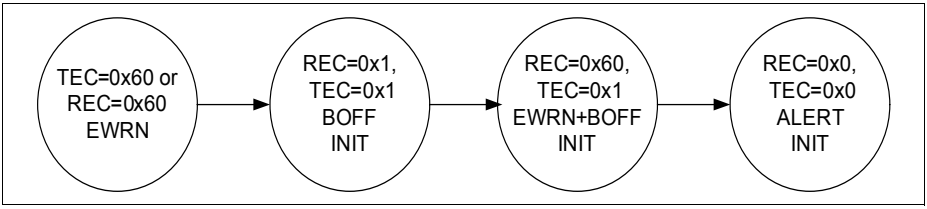

Figure 23 Alert Interrupt Behavior in case of Bus-Off

When the threshold for error warning (EWRN) is reached (default value of Error Warning Level EWRN = 0x60), then the EWRN interrupt is issued. The bus-off (BOFF) status is reached if TEC > 255 according to CAN specification, changing the MultiCAN module with REC and TEC to the same value 0x1, setting the INIT bit to 1_B, and issuing the BOFF interrupt. The bus-off recovery phase starts automatically. Every time an idle time is seen, REC is incremented. If REC = 0x60, a combined status EWRN+BOFF is reached. The corresponding interrupt can also be seen as a pre-warning interrupt, that the bus-off recovery phase will be finished soon. When the bus-off recovery phase has finished (128 times idle time have been seen on the bus), EWRN and BOFF are cleared, the ALERT interrupt bit is set and the INIT bit is still set.

MultiCAN_AI.H008 Effect of CANDIS on SUSACK

When a CAN node is disabled by setting bit NCR.CANDIS = 1_B, the node waits for the bus idle state and then sets bit NSR.SUSACK = 1_B.

However, SUSACK has no effect on applications, as its original intention is to have an indication that the suspend mode of the node is reached during debugging.

MultiCAN_TC.H003 Message may be discarded before transmission in STT mode

If MOFCRn.STT=1 (Single Transmit Trial enabled), bit TXRQ is cleared (TXRQ=0) as soon as the message object has been selected for transmission and, in case of error, no retransmission takes places.

Therefore, if the error occurs between the selection for transmission and the real start of frame transmission, the message is actually never sent.

Workaround

In case the transmission shall be guaranteed, it is not suitable to use the STT mode. In this case, MOFCRn.STT shall be 0.

MultiCAN_TC.H004 Double remote request

Assume the following scenario: A first remote frame (dedicated to a message object) has been received. It performs a transmit setup (TXRQ is set) with clearing NEWDAT. MultiCAN starts to send the receiver message object (data frame), but loses arbitration against a second remote request received by the same message object as the first one (NEWDAT will be set).

When the appropriate message object (data frame) triggered by the first remote frame wins the arbitration, it will be sent out and NEWDAT is not reset. This leads to an additional data frame, that will be sent by this message object (clearing NEWDAT).

There will, however, not be more data frames than there are corresponding remote requests.

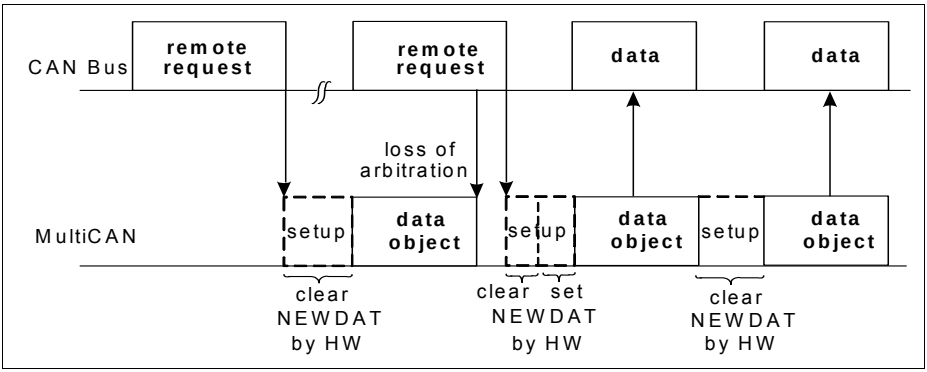

Figure 24 Loss of Arbitration

PORTS_CM.H002 Class A2 pins GPIO driver strength configuration

Before activating the push-pull driver, it is recommended to configure its driver strength and slew rate according to its pad class and the application needs using the Pad Driver Mode register (Pn_PDR).

Selecting the appropriate driver strength allows to optimize the outputs for the needed interface performance, can help to reduce power consumption, and limits noise, crosstalk and electromagnetic emissions (EMI).

There are three classes of GPIO output drivers:

- "Class A1 pads (low speed 3.3V LVTTL outputs)
- "Class A1+ pads (medium speed 3.3V LVTTL outputs)
- "Class A2 pads (high speed 3.3V LVTTL outputs, e.g. for EBU or fast serial interfaces)

Class A1 pins provide the choice between medium and weak output drivers. Speed grade 6MHz.

Class A1+ pins provide the choice between strong/medium/weak output drivers. For the strong driver, the signal transition edge can be additionally selected as soft or slow. Speed grade 25MHz.

Class A2 pins provide the choice between strong/medium/weak output drivers. For the strong driver, the signal transition edge can be additionally selected as sharp/medium/soft. Speed grade 80MHz.

If the output driver strength of Class A2 pins is configured as strong/sharp care need to be taken to avoid overshoots, undershoot and ringing that may lead to high radiated emissions and crosstalk.

The high radiated emissions may lead to Bus Errors exceptions (or Hard Fault exception in case the Bus Error exception is not enabled) caused by a double bit error fail in a flash read access. Flash double bits errors are identified in the FLASH0.FSR register.

Recommendation

In general to avoid the high radiated emissions it is recommended the usage of damping resistors (10 ohms) between the high speed drivers and the transmission lines.

It is also recommended to adapt the driver strength to the needs of the application, i.e. to drive a 25MHz signal strong/medium or strong/soft would be suitable lowering the potential overshoots and undershoots.

RESET_CM.H001 Power-on reset release

The on-chip EVR implements a power validation circuitry which supervises V_{DDP} and V_{DDC}. This circuit releases or asserts the system reset to ensure safe operation. This reset is visible on bidirectional $\overline{\text{PORST}}$ pin. If the $\overline{\text{PORST}}$ release requirement cannot be met due to external circuitry then spikes or toggling on the $\overline{\text{PORST}}$ pin may occur.

Implications

Spikes or repeated PORST assertions may have an effect on the rest of the system if the reset signal is shared with other electronic components on the PCB.

A repeated $\overline{\text{PORST}}$ may also result in loss of information about hibernation status after an interrupted wake-up has been performed.

Recommendation

It is required to ensure a fast rising edge of the $\overline{\text{PORST}}$ signal as specified in section “Power-Up and Supply Monitoring” of the Data Sheet. The recommended approach is to apply a pull-up resistor on the $\overline{\text{PORST}}$ pin.

Typically a 10 - 90 kΩ resistor is sufficient in application cases where the device is in control of the reset generation performed by its internal power validation circuit and no additional load is applied to the $\overline{\text{PORST}}$ pin. The required pull-up resistor value may vary depending on the electrical parameters of the system influencing the signal edges of the $\overline{\text{PORST}}$ signal; for example resistance and capacitance of the PCB and other components connected to the $\overline{\text{PORST}}$ pin.

USIC_AI.H004 I2C slave transmitter recovery from deadlock situation

While operating the USIC channel as an IIC slave transmitter, if the slave runs out of data to transmit before the master receiver issues clock pulses, for example due to an error in the application flow, it ties the SCL infinitely low.

Recommendation

To recover and reinitialize the USIC IIC slave from such a deadlock situation, the following software sequence can be used:

1. Switch the SCL and SDA port functions to be general port inputs for the slave to release the SCL and SDA lines:
   a) Write 0 to the two affected Pn_IOCRx.PCy bit fields.
2. Flush the FIFO buffer:
   a) Write 1_B to both USICx_CHy_TRBSCR.FLUSHTB and FLUSHRB bits.
3. Invalidate the internal transmit buffer TBUF:
   a) Write 10_B to USICx_CHy_FMR.MTDV.
4. Clear all status bits and reinitialize the IIC USIC channel if necessary.
5. Reprogram the Pn_IOCRx.PCy bit fields to select the SCL and SDA port functions again.

At the end of this sequence, the IIC slave is ready to communicate with the IIC master again.

5 Documentation Updates

The errata in this section contain updates to or completions of the user documentation. These updates are subject to be taken over into upcoming user documentation releases.

MPU_CM.D001 No restrictions on using Bit5 to Bit8 of register MPU_RBAR

The XMC4000 reference manuals describe, Bit5 to Bit8 of register MPU_RBAR are read-only and fixed to 0.

The ARM documentation for the Cortex-M4 processors specifies these bits are used to extend the LSBs to set the region base address and size.

Implications

The reference manual is limiting the range of possible base addresses and region sizes for memory protection.

Workaround

Use the ARM documentation set for ARM Cortex-M4 processor for reference.

STARTUP_CM.D003 Alignment of ABM/PSRAM Header

The reference manual is specifying PSRAM/ABM Header of 32 byte size. Only for 20byte of these 32 bytes the functionality is defined. The remaining 12bytes are reserved. Inside the chapter for Startup modes of the reference manual only the functional bytes are specified but not the location of the reserved bytes.

The following figure provides a detailed view on the location of the reserved bytes:

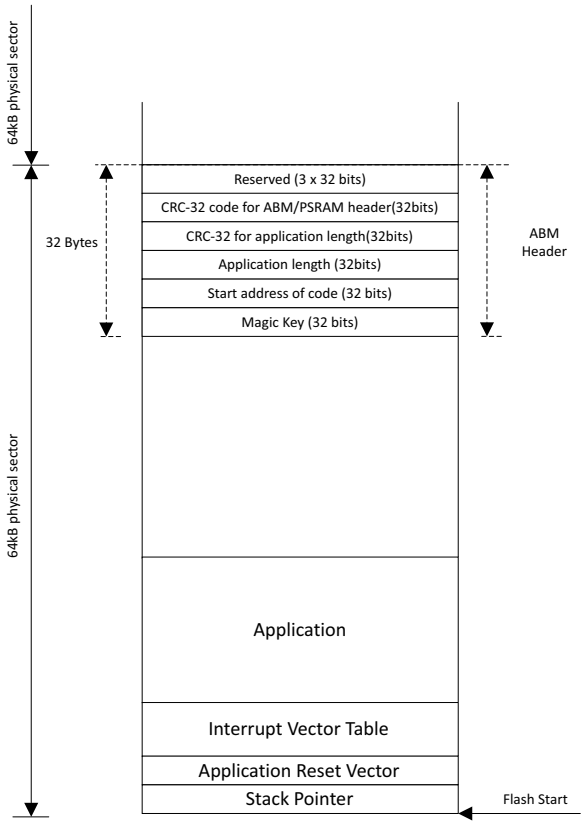

Figure 25 ABM/PSRAM Header - Location of reserved bytes