Errata Sheet

Rel. 1.4, 2013-05

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

Overview

Document ID is **02462AERRA**.

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	V1.2	December 2012
XMC4500 Data Sheet	V1.0	January 2013

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	2012-03	Initial Version
1.1	2012-08	Added: CCU8_AI.001, CCU_AI.002-003, CCU_AI.004, ETH_AI.001-002, LEDTS_AI.001, PORTS_CM.001-002, USIC_AI.010-016, STARTUP_CM.H001
1.2	2012-11	Added: CCU8_AI.002, GPDMA_CM.001, GPDMA_CM.002, POSIF_AI.001, USIC_AI.018, ADC_AI.H004 Updated: USIC_AI.016
1.3	2013-02	Added: ADC_AI.002, CCU8_AI.003, CCU_AI.005, PMU_CM.001, RESET_CM.H001 Updated: CCU8_AI.002, PORTS_CM.002

Table 3 Errata fixed in this step

Errata	Short Description	Change
- none -		

Table 4 Functional Deviations

Functional Deviation	Short Description	Chg	Pg
ADC_AI.002	Result of Injected Conversion may be wrong		9
CCU4_AI.001	CCU4 period interrupt is not generated in capture mode		9
CCU8_AI.001	CCU8 Floating Prescaler function does not work with Capture Trigger 1		11

Table 4 Functional Deviations (cont'd)

Functional Deviation	Short Description	Chg	Pg
CCU8_AI.002	CC82 Timer of the CCU8x module cannot use the external shadow transfer trigger connected to the POSIFx module		12
CCU8_AI.003	CCU8 Parity Checker Interrupt Status is cleared automatically by hardware		14
CCU_AI.001	CCU4 and CCU8 capture full flags do not work when module clock is faster than peripheral bus clock		17
CCU_AI.002	CCU4 and CCU8 Prescaler synchronization clear does not work when Module Clock is faster than Peripheral Bus Clock		19
CCU_AI.003	CCU4 and CCU8 capture full flag is not cleared if a capture event occurs during a bus read phase		20
CCU_AI.004	CCU4 and CCU8 Extended Read Back loss of data		23
CCU_AI.005	CCU4 and CCU8 External IP clock Usage		25
CPU_CM.001	Interrupted loads to SP can cause erroneous behavior		26
DAC_CM.001	DAC immediate register read following a write issue		28
DAC_CM.002	No error response for write access to read only DAC ID register		28
DEBUG_CM.001	OCDS logic in peripherals affected by TRST		28
DEBUG_CM.002	CoreSight logic only reset after power-on reset		29

Table 4 Functional Deviations (cont'd)

Functional Deviation	Short Description	Chg	Pg
ETH_AI.001	Incorrect IP Payload Checksum at incorrect location for IPv6 packets with Authentication extension header		29
ETH_AI.002	Incorrect IP Payload Checksum Error status when IPv6 packet with Authentication extension header is received		30
ETH_AI.003	Overflow Status bits of Missed Frame and Buffer Overflow counters get cleared without a Read operation		32
GPDMA_CM.001	Unexpected Block Complete Interrupt During Multi-Block Transfers		32
GPDMA_CM.002	GPDMA doesn't Accept Transfer During/In 2nd Cycle of 2-Cycle ERROR Response		34
LEDTS_AI.001	Delay in the update of FNCTL.PADT bit field		34
PMU_CM.001	Branch from non-cacheable to cacheable address space instruction may corrupt the program execution		39
PORTS_CM.001	P15_PDISC.[4,5] register bits cannot be written		41
PORTS_CM.002	P0.9 Pull-up permanently active		42
PORTS_CM.005	Different PORT register reset values after module reset	New	42
POSIF_AI.001	Input Index signal from Rotary Encoder is not decoded when the length is 1/4 of the tick period		44
RTC_CM.001	RTC event might get lost		46
SCU_CM.001	Multiple Power-On resets upon noise on supply voltage		46

Table 4 Functional Deviations (cont'd)

Functional Deviation	Short Description	Chg	Pg
SCU_CM.002	Missed wake-up event during entering external hibernate mode		47
SCU_CM.003	The state of HDCR.HIB bit of HCU gets updated only once in the register mirror after reset release		47
SCU_CM.006	Deep sleep entry with PLL power-down option generates SOSCWDGT and SVCOLCKT trap		48
SDMMC_CM.001	Unexpected interrupts after execution of CMD13 during bus test		48
SDMMC_CM.002	Unexpected Tx complete interrupt during R1b response		49
STARTUP_CM.001	CAN Bootstrap Loader	New	50
USB_CM.002	GAHBCFG.GlblIntrMsk not cleared with a software reset		50
USIC_AI.005	Only 7 data bits are generated in IIC mode when TBUF is loaded in SDA hold time		50
USIC_AI.006	Dual SPI format not supported		51
USIC_AI.007	Protocol-related argument and error bits in register RBUFSR contain incorrect values following a received data word		51
USIC_AI.008	Bit SCLKOSEL in register BRG not implemented		53
USIC_AI.009	Baud rate generator interrupt cannot be used		54
USIC_AI.010	Minimum and maximum supported word and frame length in multi-IO SSC modes		54
USIC_AI.011	Write to TBUF01 has no effect		55

Table 4 Functional Deviations (cont'd)

Functional Deviation	Short Description	Chg	Pg
USIC_AI.012	USIC2 does not provide FIFO buffer capability		55
USIC_AI.013	SCTR register bit fields DSM and HPCDIR are not shadowed with start of data word transfer		55
USIC_AI.014	No serial transfer possible while running capture mode timer		56
USIC_AI.015	Wrong generation of FIFO standard transmit/receive buffer events when TBCTR.STBTEN/RBCTR.SRBTEN = 1		56
USIC_AI.016	Transmit parameters are updated during FIFO buffer bypass		57
USIC_AI.018	Clearing PSR.MSLS bit immediately deasserts the SELOx output signal		57

Table 5 Application Hints

Hint	Short Description	Chg	Pg
ADC_AI.H004	Completion of Startup Calibration		59
MultiCAN_AI.H005	TxD Pulse upon short disable request		59
MultiCAN_AI.H006	Time stamp influenced by resynchronization		60
MultiCAN_AI.H007	Alert Interrupt Behavior in case of Bus-Off		60
MultiCAN_AI.H008	Effect of CANDIS on SUSACK		61
MultiCAN_TC.H003	Message may be discarded before transmission in STT mode		61
MultiCAN_TC.H004	Double remote request		61

Table 5 Application Hints (cont'd)

Hint	Short Description	Chg	Pg
RESET_CM.H001	Power-On Reset Release		62
STARTUP_CM.H001	ASC Bootstrap Loader Baudrate Limitation		63

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).

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.

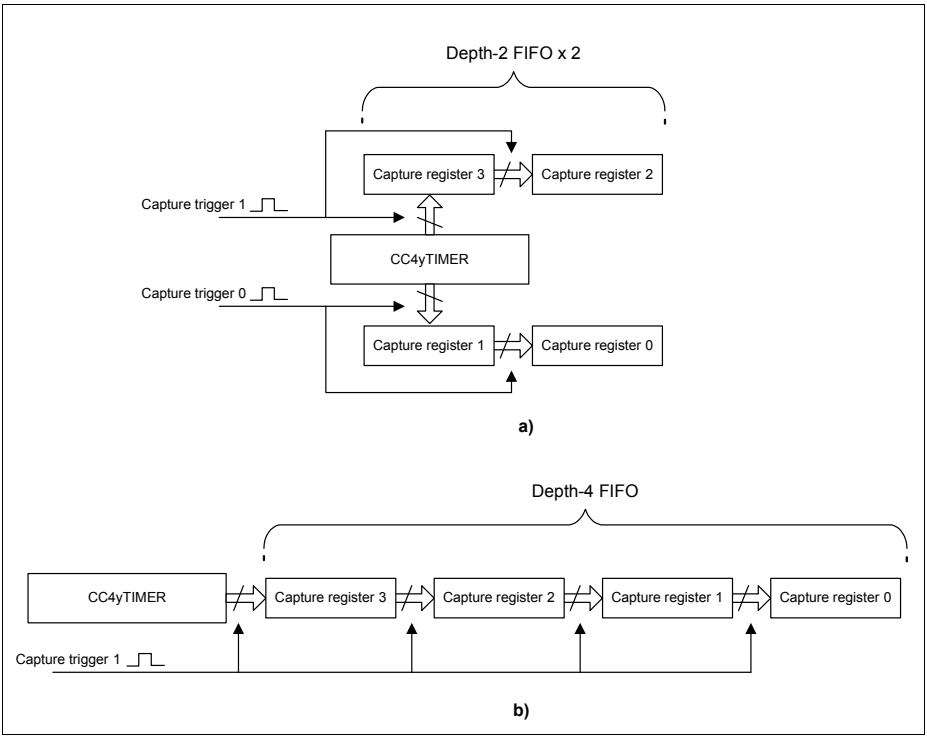

Figure 1 CCU4 capture modes - a) Depth-2 x2 Capture; b) Depth-4 Capture

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.

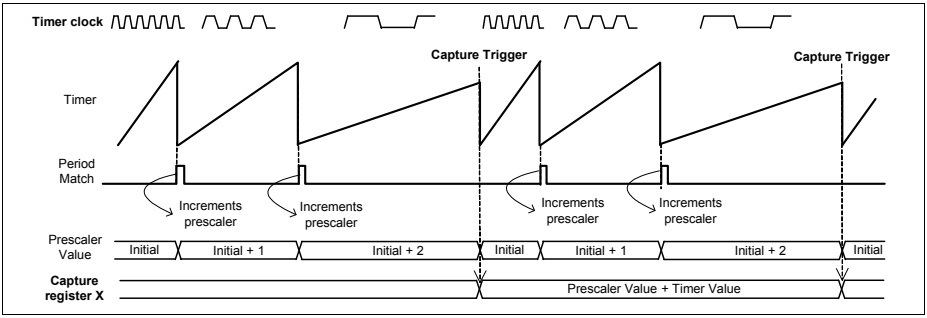

Figure 2 Floating Prescaler for Capturing

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.

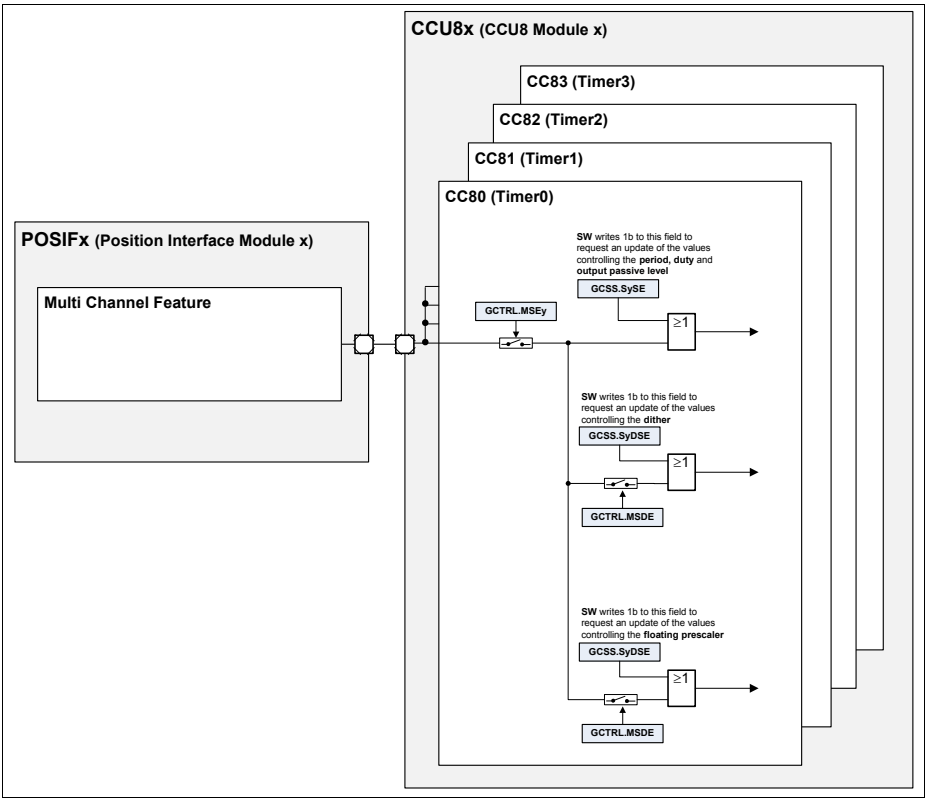

Figure 3 Value update trigger connection between CCU8x and POSIFx

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.

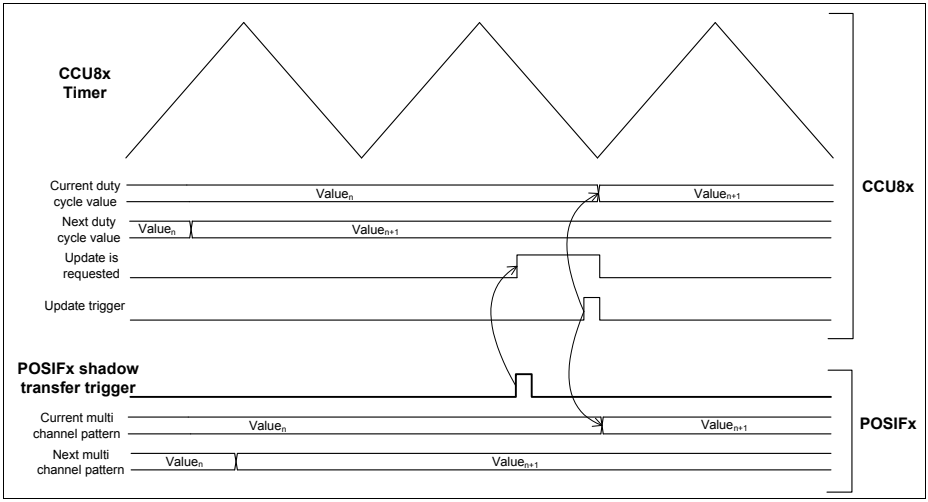

Figure 4 Value update handshake between CCU8x and POSIFx

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.

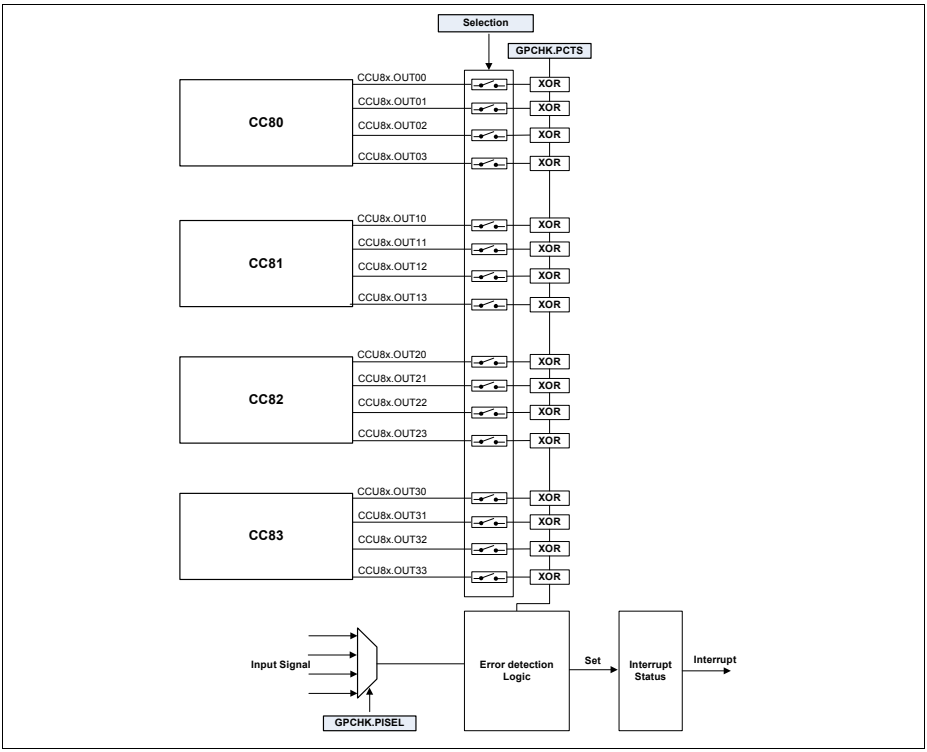

Figure 5 Parity Checker diagram

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.

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 6**.

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.

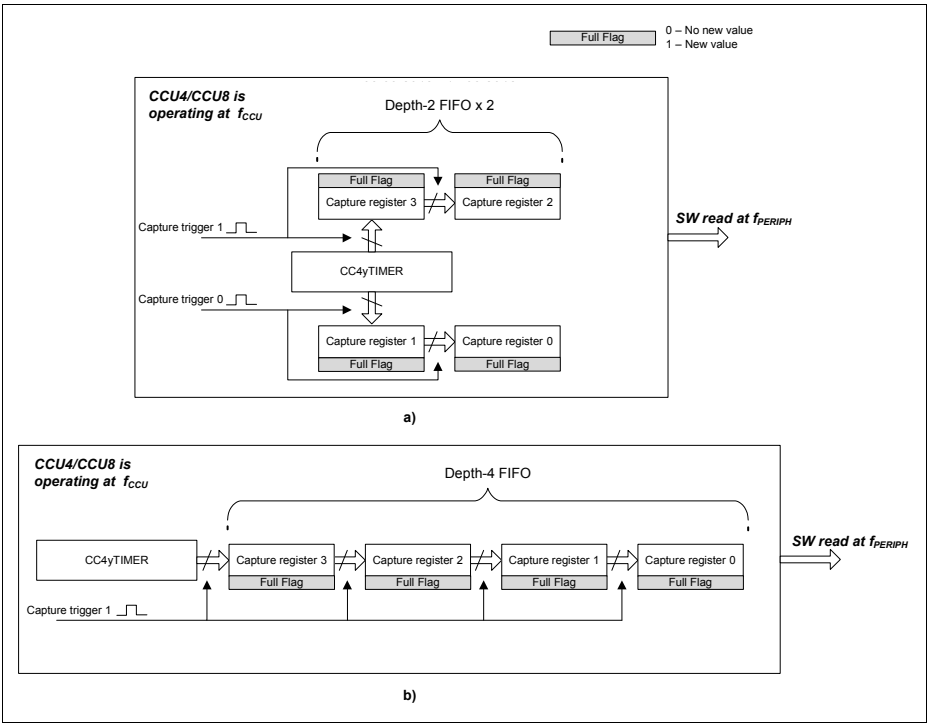

Figure 6 Capture Modes Structure - a) Depth-2 x2 Capture; b) Depth-4 Capture

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 7**.

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 8** - for simplification purposes a 2 depth capture scheme is shown.

Referring to **Figure 8**, 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).

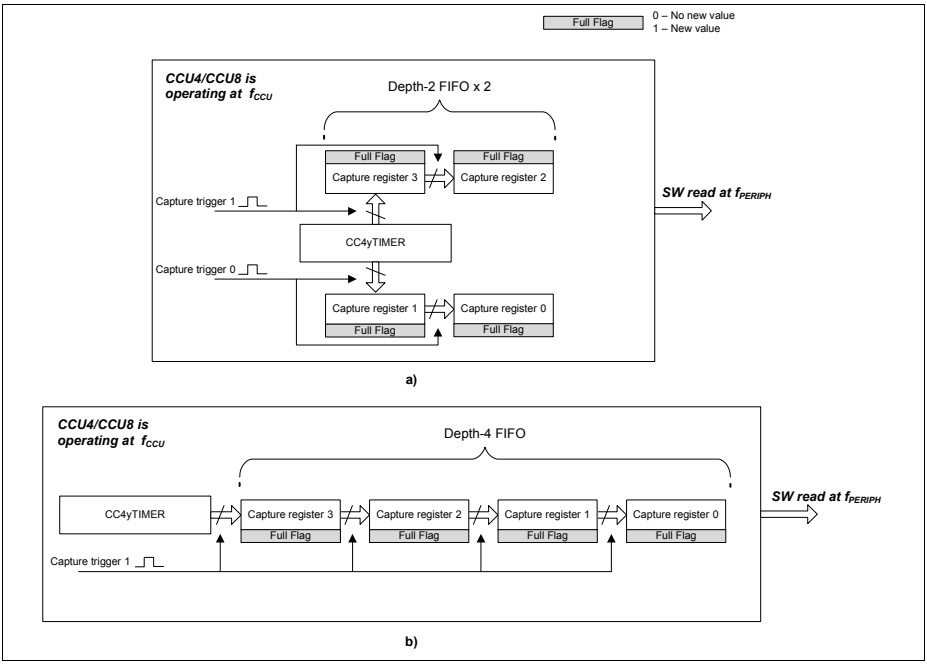

Figure 7 Capture Modes Structure - a) Depth-2 x2 Capture; b) Depth-4 Capture

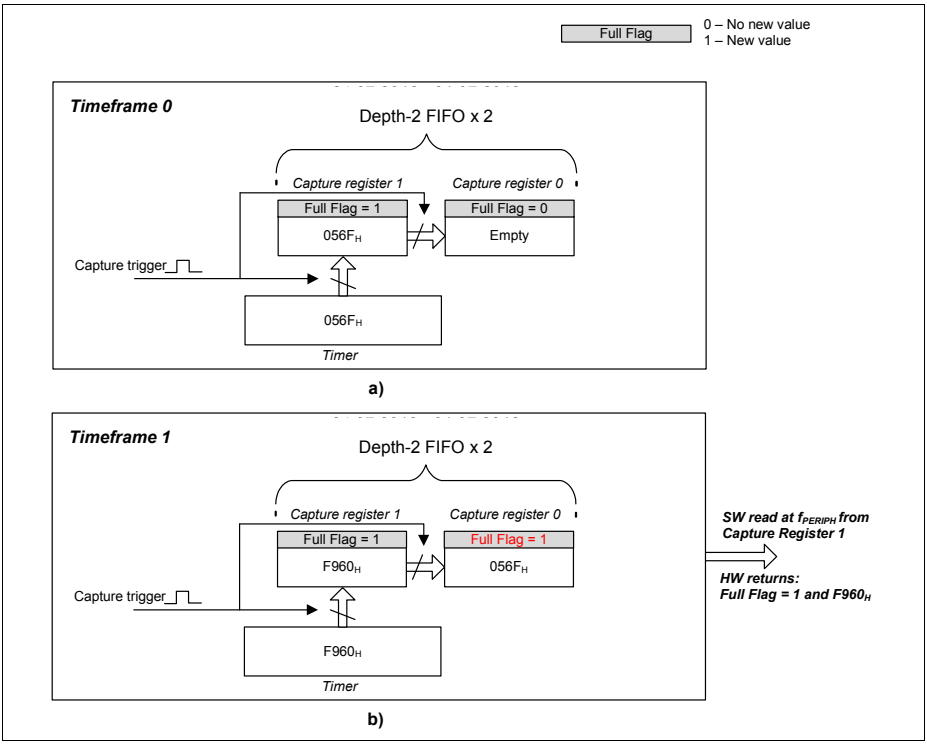

Figure 8 Capture shift during read back phase - a) Capture trigger without collision with read back; b) Capture trigger collision with read back

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 9**, 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)

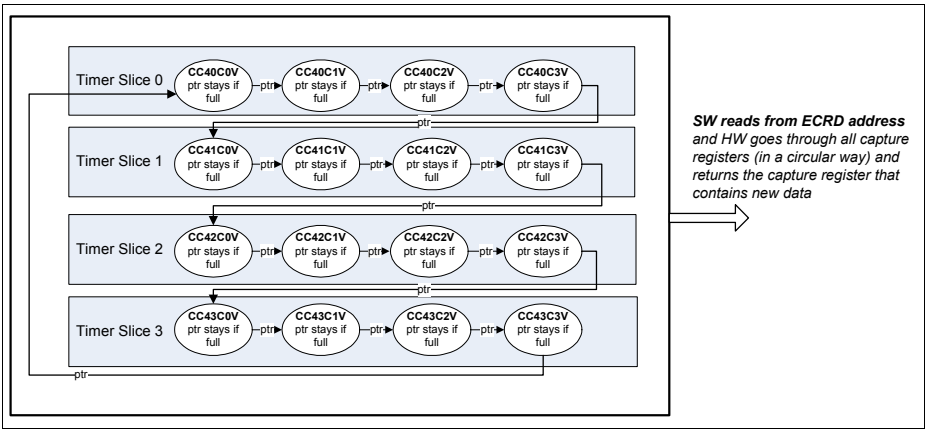

Figure 9 Extended Read Back access - example for CCU4 (CCU8 structure is the same)

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.

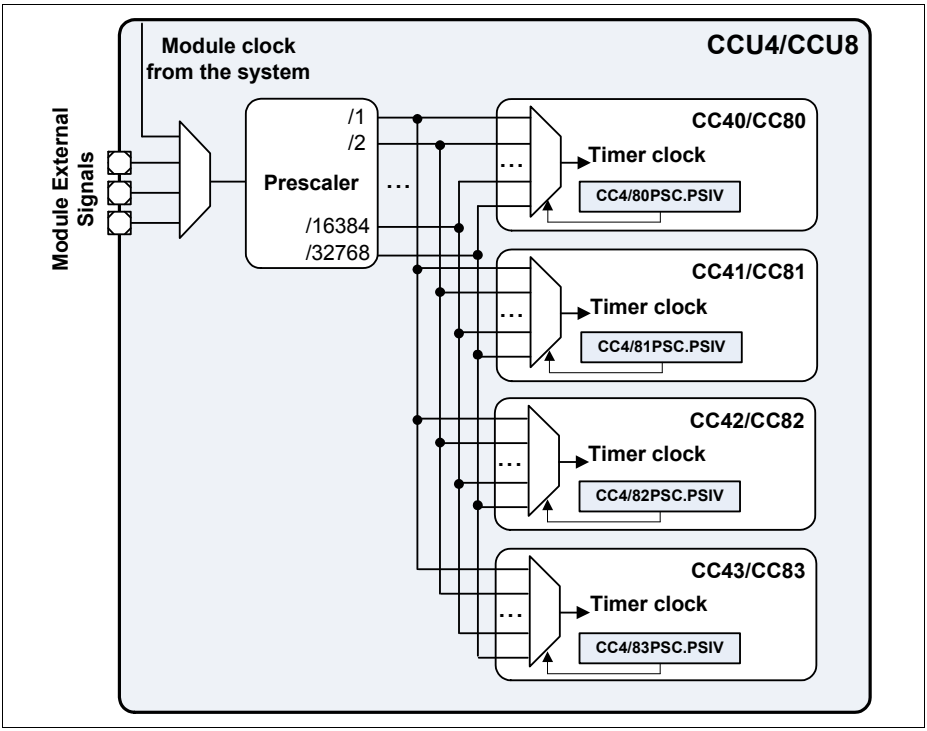

Figure 10 Clock Selection Diagram for CCU4/CCU8

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.

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.

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.

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 11**).

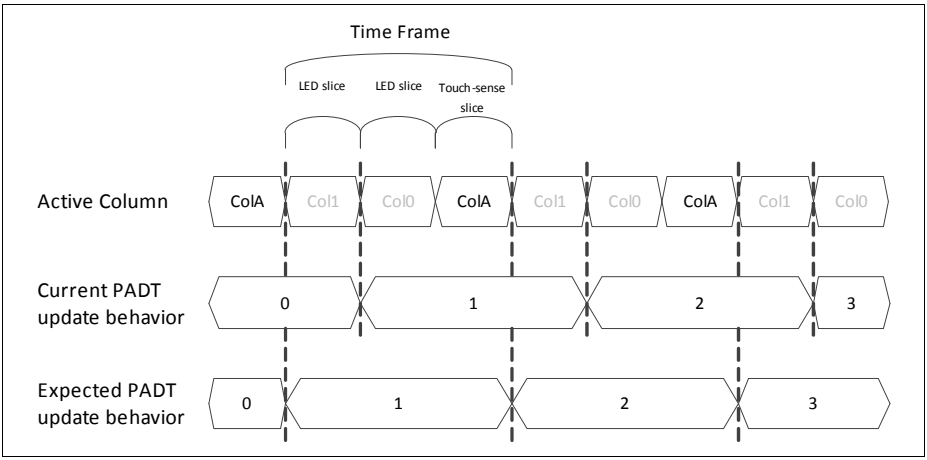

Figure 11 PADT update behavior

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 12**). In addition to this, when no LEDs are enabled, pad turn 0 will prevail for one time slice longer before it gets updated (**Figure 13**).

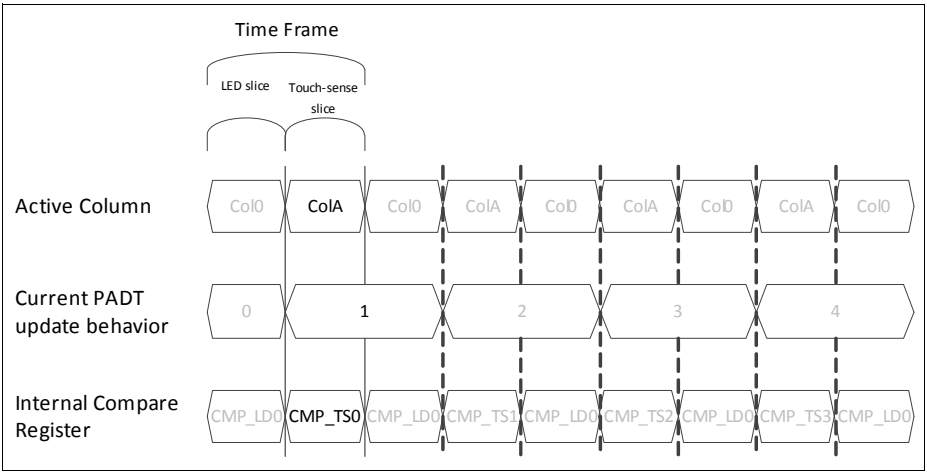

Figure 12 Effect of delay on the update of Internal Compare Register with 1 LED column enabled

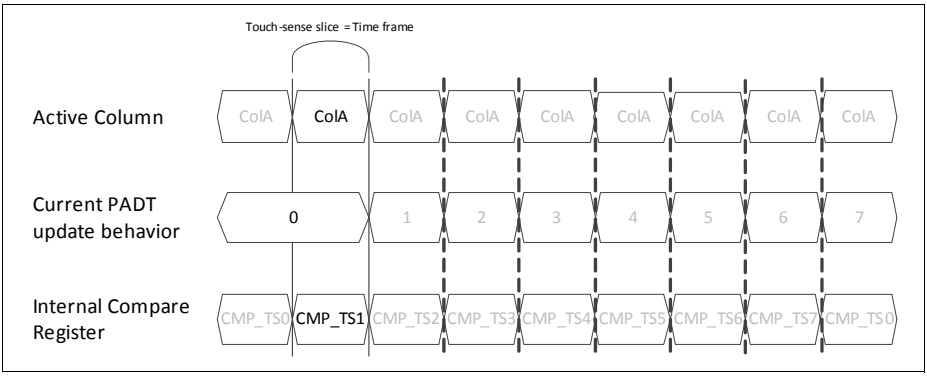

Figure 13 Pad turn 0 prevails for one time slice longer when no LEDs are enabled

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 14**).

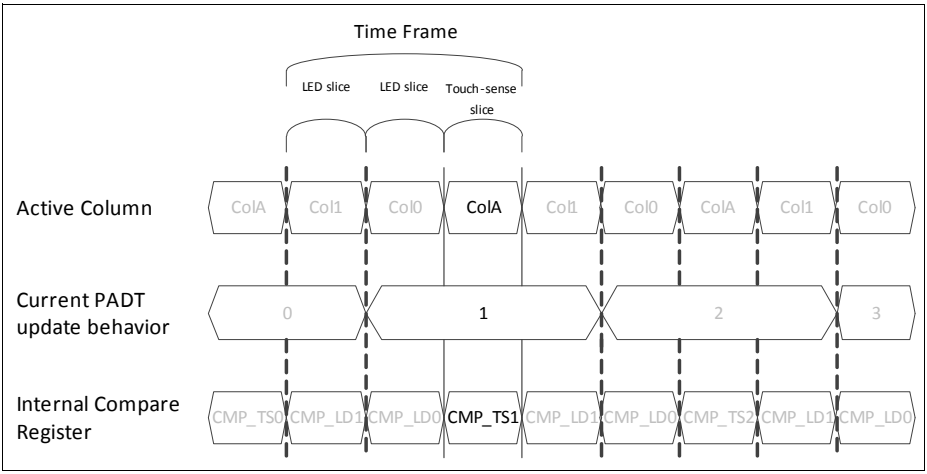

Figure 14 Internal Compare Register updated with correct compare register value with 2 LED columns enabled

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 6**). The bit field FNCTL.FNCOL provides a way of interpreting the active function or LED column in the previous time slice.

Table 6 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 15**).

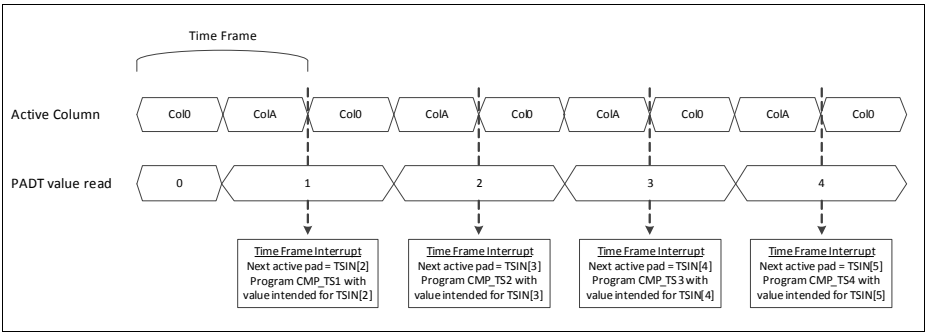

Figure 15 Software workaround demonstration

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].

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].

Software that reads P15_IN will read undefined values for P15_IN[4,5]. The read values depend on the analog input level of the respective pin.

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 7 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.

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.

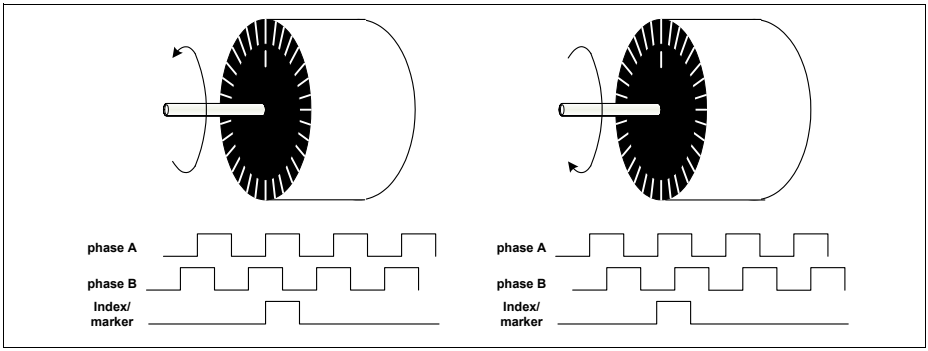

Figure 16 Rotary Encoder outputs - Phase A, Phase B and Index

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).

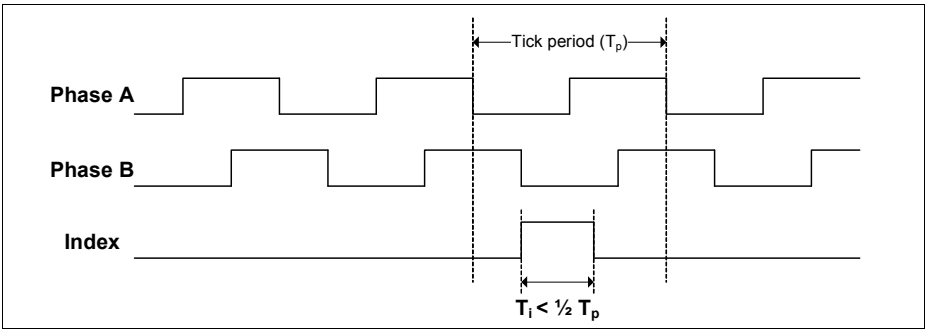

Figure 17 Different index signal types

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.

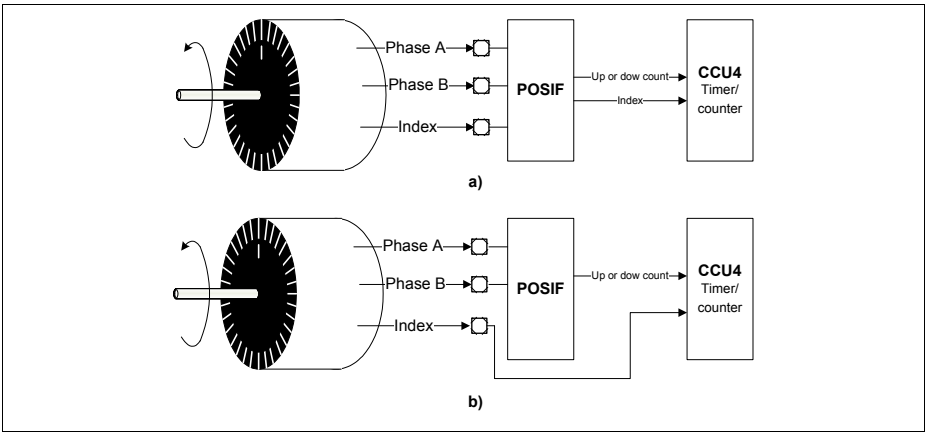

Figure 18 Index usage workaround - a) Non working solution; b) Working solution

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.001 Multiple Power-On resets upon noise on supply voltage

The on-chip EVR has a power validation circuitry which supervises VDDP and VDDC. This circuit releases or asserts the system reset to ensure safe operation. This reset is visible on bidirectional PORST pin.

Upon a short drop of supply voltage caused by noise, for a period of time longer than around 2 microseconds, unintended multiple assertions and releases of Power-On reset may occur before stable operation is restored.

Implications

Toggling of the internal power-on reset gets propagated to external components via PORST pin potentially causing additional reset of the external components.

Workaround

Externally controlled reset released when supply voltage is stable and at a valid level will solve the issue. The reset shall be applied to the PORST pin and overcome its driving strength in order to suppress on-chip generated power-on reset.

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.

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 8 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 9 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.

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.

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 10

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 11

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 Bit SCLKOSEL in register BRG not implemented

In SSC slave mode, setting the bit SCLKOSEL in register BRG to one selects the transmit shift clock as the signal SCLKOUT input source. This is required for a complete closed loop delay compensation.

However, in the current device version, the bit SCLKOSEL is not implemented and setting the bit to one has no effect. The bit remains always zero. Therefore, the complete closed loop delay compensation feature cannot be supported.

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 12

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.012 USIC2 does not provide FIFO buffer capability

USIC module 2 (USIC2) does not provide FIFO buffer capability.

Workaround

If FIFO buffer capability is required, use USIC modules 0 and 1 (USIC0 and USIC1) instead.

If USIC2 is required for serial communications, use only the standard transmit and receive data buffers for data transmission and reception.

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.

3 Application Hints

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

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.

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:

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:

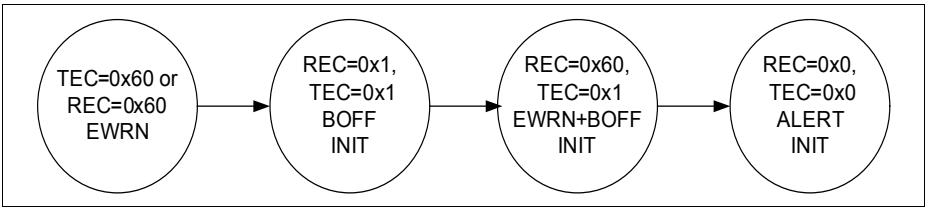

Figure 19 Alert Interrupt Behavior in case of Bus-Off

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.

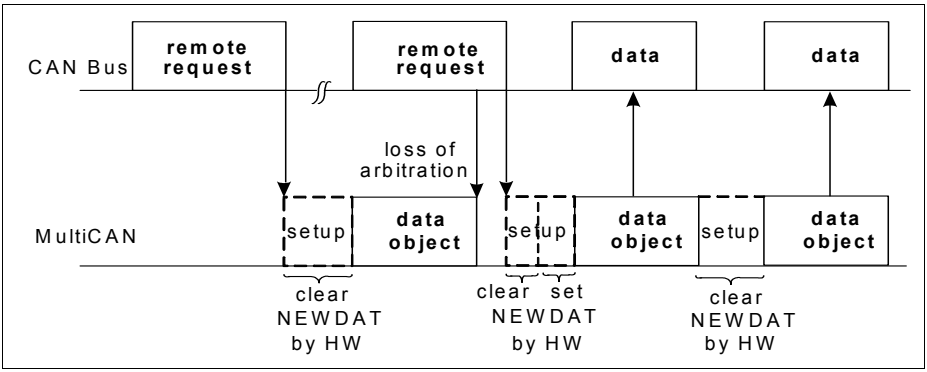

Figure 20 Loss of Arbitration

RESET_CM.H001 Power-On Reset Release

The on-chip EVR implements a power validation circuitry which supervises VDDP and VDDC. This circuit releases or asserts the system reset to ensure safe operation. This reset is visible on bidirectional $\overline{\text{PORST}}$ pin.

Implications

Potential effects if the $\overline{\text{PORST}}$ release requirement is not met (please refer to the Data Sheet for details) is presence of spikes or toggling on the $\overline{\text{PORST}}$ pin which 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. Potential presence of the spikes on $\overline{\text{PORST}}$, however, will not lead to a fatal system startup failure or deadlock.

Recommendation

It is required to ensure fast $\overline{\text{PORST}}$ release, as specified in Data Sheet. The recommended approach is to apply a pull-up resistor on the $\overline{\text{PORST}}$ pin.

Typically a 10 - 90 k? 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,

like parasitic wire resistance and capacitance of the PCB, driving strength of other electronic components connected to the $\overline{\text{PORST}}$ pin and other side conditions. The pull-up resistance may need to be adapted accordingly.

STARTUP_CM.H001 ASC Bootstrap Loader Baudrate Limitation

Given the tolerance of the internal oscillator frequency f_{OFI} it is not possible to support an entire spectrum of standard baud rates in the ASC BSL boot mode.

Solution

Only the baud rates in the range 9600 bps to 115200 bps are supported. With frequency scaling enabled, supported baud rates are in the range 19200 bps to 115200 bps.It is highly recommended that a baud rate centered around 38400 bps is chosen for the ASC BSL mode to achieve consistent results across devices.