Errata Sheet

Rel. 1.4, 2015-07

Device	**XMC4100/XMC4200**
Marking/Step	**EES-AA, ES-AA**
Package	**PG-VQFN-48, PG-LQFP-64**

Overview

Document ID is **02530AERRA**.

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

Table 1 Current User Documentation

Document	Version	Date
XMC4100/XMC4200 Reference Manual	V1.5	April 2014
XMC4100/XMC4200 Data Sheet	V1.1	March 2014

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	2013-02	Initial Version
1.1	2013-05	Added: PORTS_CM.005, STARTUP_CM.001
1.2	2013-09	Added: ADC_AI.008, CCU8_AI.002, CCU8_AI.004, CPU_CM.004, HRPWM_AI.003. Updated: RESET_CM.H001
1.3	2014-04	Added: ADC_TC.064, HRPWM_AI.004, SCU_CM.015, USIC_AI.008, USIC_AI.020 ADC_TC.H011.
1.4	2015-07	This Document. For changes see column "Chg" in the tables below.

Table 3 Functional Deviations

Functional Deviation	Short Description	Chg	Pg
ADC_AI.002	Result of Injected Conversion may be wrong		8
ADC_AI.008	Wait-for-Read condition for register GLOBRES not detected in continuous auto-scan sequence		8
ADC_AI.016	No Channel Interrupt in Fast Compare Mode with GLOBRES	New	9
ADC_TC.064	Effect of conversions in 10-bit fast compare mode on post-calibration	Update	9
CCU8_AI.002	CC82 Timer of the CCU8x module cannot use the external shadow transfer trigger connected to the POSIFx module		10
CCU8_AI.003	CCU8 Parity Checker Interrupt Status is cleared automatically by hardware		12

Table 3 Functional Deviations (cont'd)

Functional Deviation	Short Description	Chg	Pg
CCU8_AI.004	CCU8 output PWM glitch when using low side modulation via the Multi Channel Mode		15
CCU_AI.002	CCU4 and CCU8 Prescaler synchronization clear does not work when Module Clock is faster than Peripheral Bus Clock		18
CCU_AI.004	CCU4 and CCU8 Extended Read Back loss of data		19
CCU_AI.005	CCU4 and CCU8 External IP clock Usage		20
CCU_AI.006	Value update not usable in period dither mode	New	22
CPU_CM.001	Interrupted loads to SP can cause erroneous behavior		23
CPU_CM.004	VDIV or VSQRT instructions might not complete correctly when very short ISRs are used		24
DTS_CM.001	DTS offset calibration value limitations	New	26
HRPWM_AI.001	HRPWM output signal interference while using two control sources		26
HRPWM_AI.002	HRPWM CSG missing DAC conversion trigger in static mode	New	29
HRPWM_AI.003	HRPWM Usage Limitations		31
HRPWM_AI.004	HRPWM Peripheral Bus Clock Limitation		33
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		38
PORTS_CM.003	P14_PDISC register can't be written		40

Table 3 Functional Deviations (cont'd)

Functional Deviation	Short Description	Chg	Pg
PORTS_CM.005	Different PORT register reset values after module reset		41
POSIF_AI.001	Input Index signal from Rotary Encoder is not decoded when the length is 1/4 of the tick period		42
SCU_CM.006	Deep sleep entry with PLL power-down option generates SOSCWDGT and SVCOLCKT trap		44
SCU_CM.009	Out of Range Comparator Control		45
SCU_CM.015	Functionality of parity memory test function limited	Update	45
STARTUP_CM.001	CAN Bootstrap Loader		46
USB_CM.004	USB core is not able to detect resume or new session request after PHY clock is stopped	New	47
USIC_AI.008	SSC delay compensation feature cannot be used		47
USIC_AI.010	Minimum and maximum supported word and frame length in multi-IO SSC modes		48
USIC_AI.013	SCTR register bit fields DSM and HPCDIR are not shadowed with start of data word transfer		48
USIC_AI.014	No serial transfer possible while running capture mode timer		49
USIC_AI.015	Wrong generation of FIFO standard transmit/receive buffer events when TBCTR.STBTEN/RBCTR.SRBTEN = 1		49
USIC_AI.016	Transmit parameters are updated during FIFO buffer bypass		49

Table 3 Functional Deviations (cont'd)

Functional Deviation	Short Description	Chg	Pg
USIC_AI.017	Clock phase of data shift in SSC slave cannot be changed		50
USIC_AI.018	Clearing PSR.MSLS bit immediately deasserts the SELOx output signal		50
USIC_AI.019	First data word received by IIC receiver triggers RIF instead of AIF		51
USIC_AI.020	Handling unused DOUT lines in multi-IO SSC mode		52

Table 4 Deviations from Electrical- and Timing Specification

AC/DC Deviation	Short Description	Chg	Pg
POWER_CM.P001	Risk of increased current consumption in internally controlled hibernate mode	New	53

Table 5 Application Hints

Hint	Short Description	Chg	Pg
ADC_AI.H003	Injected conversion may be performed with sample time of aborted conversion	New	54
ADC_AI.H004	Completion of Startup Calibration		55
ADC_AI.H008	Injected conversion with broken wire detection	New	55
ADC_TC.H011	Bit DCMSB in register GLOBCFG		56
LEDTS_AI.H001	Column A is unavailable	New	56
MultiCAN_AI.H005	TxD Pulse upon short disable request	Update	57
MultiCAN_AI.H006	Time stamp influenced by resynchronization		57
MultiCAN_AI.H007	Alert Interrupt Behavior in case of Bus-Off		58

Table 5 Application Hints (cont'd)

Hint	Short Description	Chg	Pg
MultiCAN_AI.H008	Effect of CANDIS on SUSACK		58
MultiCAN_TC.H003	Message may be discarded before transmission in STT mode		59
MultiCAN_TC.H004	Double remote request		59
RESET_CM.H001	Power-on reset release	Update	60
USIC_AI.H004	I2C slave transmitter recovery from deadlock situation	New	61

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_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.

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.

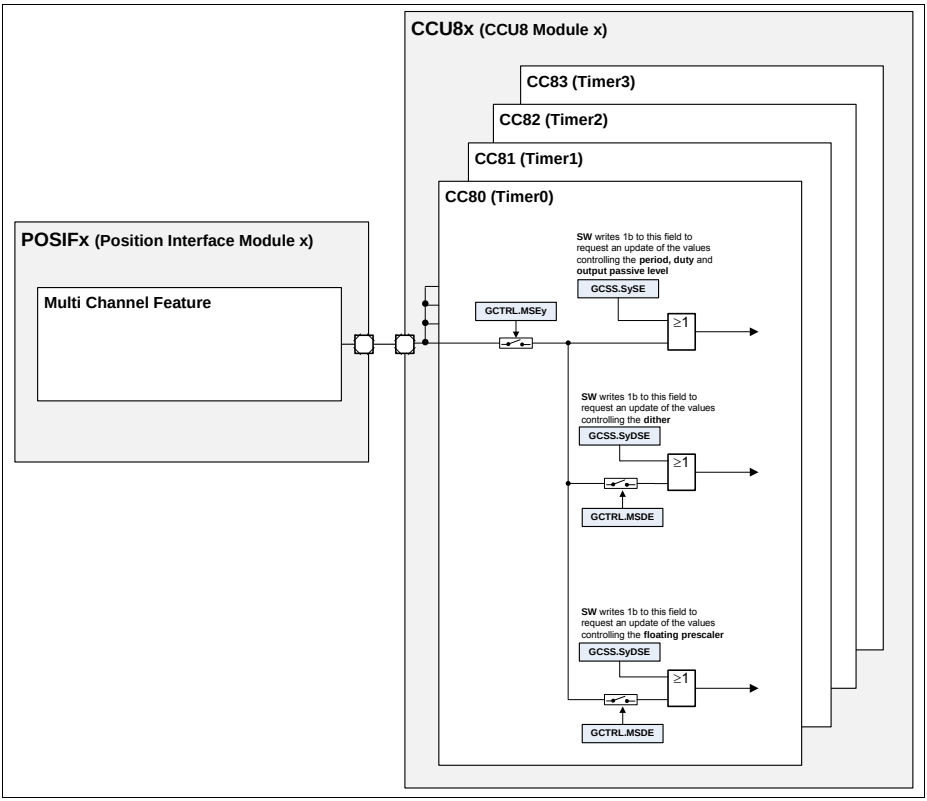

Figure 1 Value update trigger connection between CCU8x and POSIFx

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.

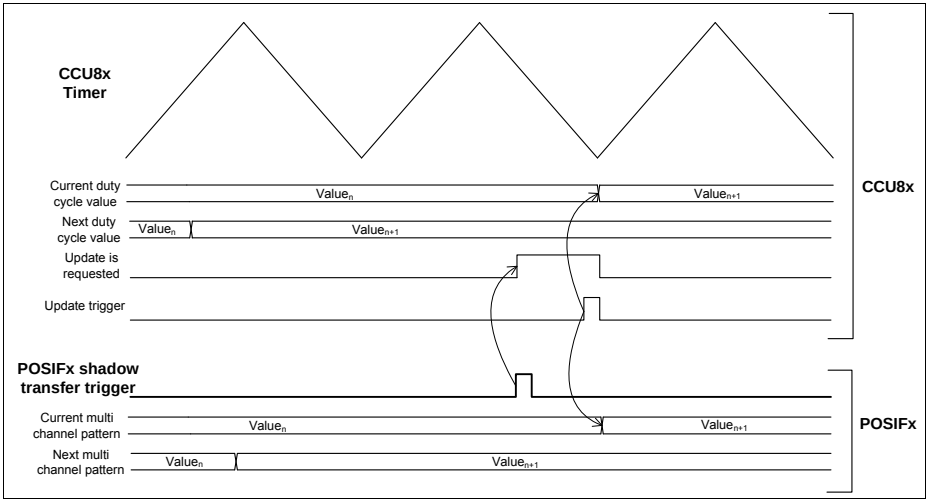

Figure 2 Value update handshake between CCU8x and POSIFx

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.

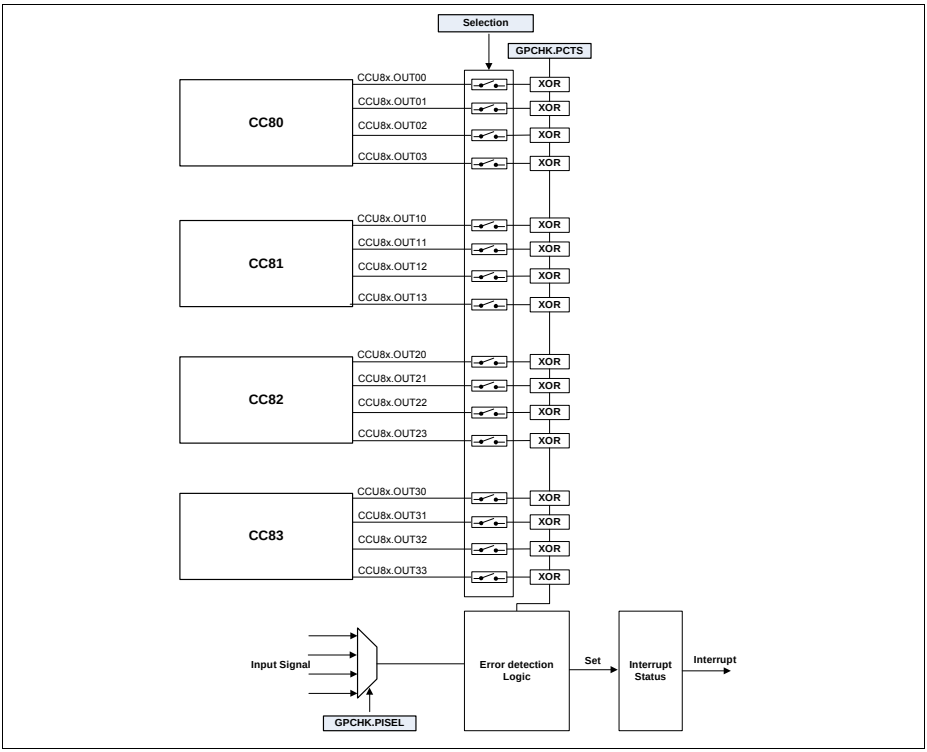

Figure 3 Parity Checker diagram

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).

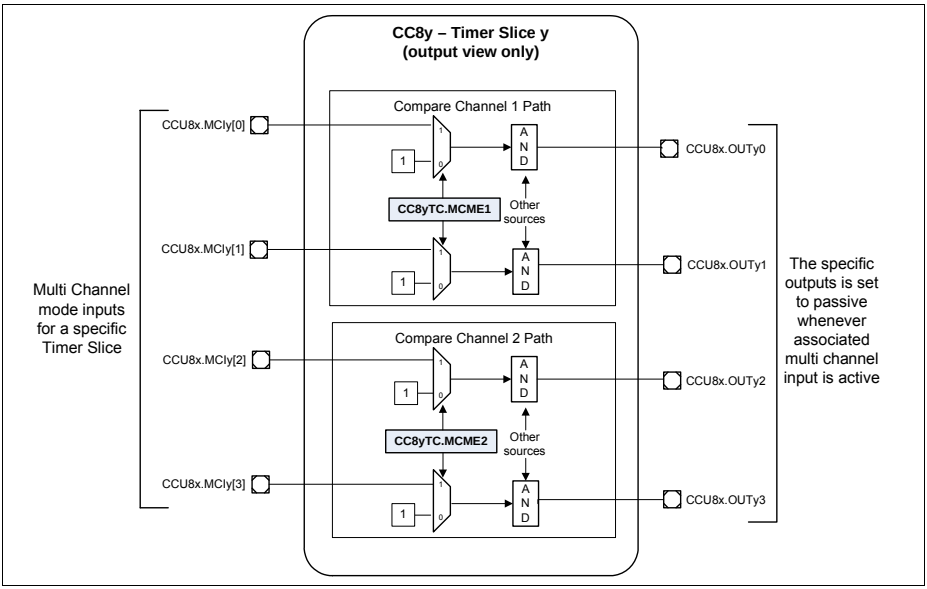

Figure 4 Multi Channel Mode diagram

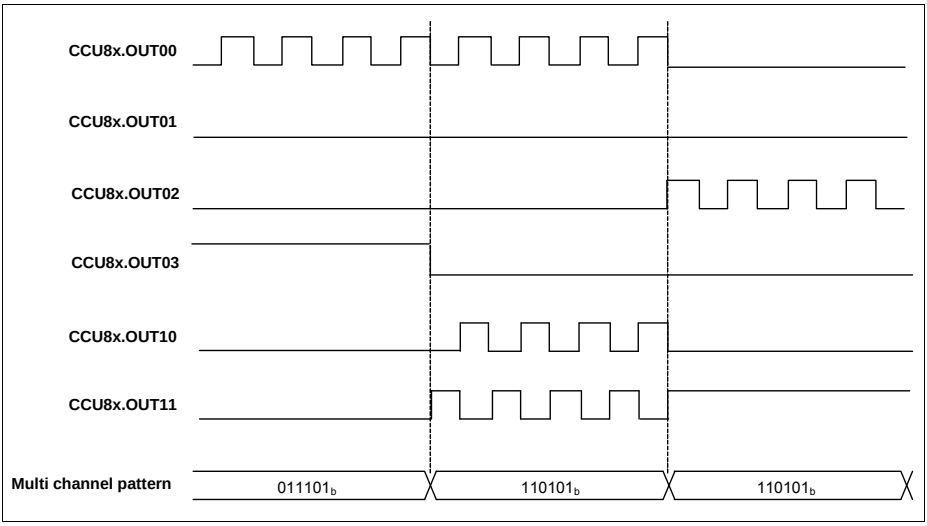

Figure 5 Multi Channel Mode applied to several CCU8 outputs

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.

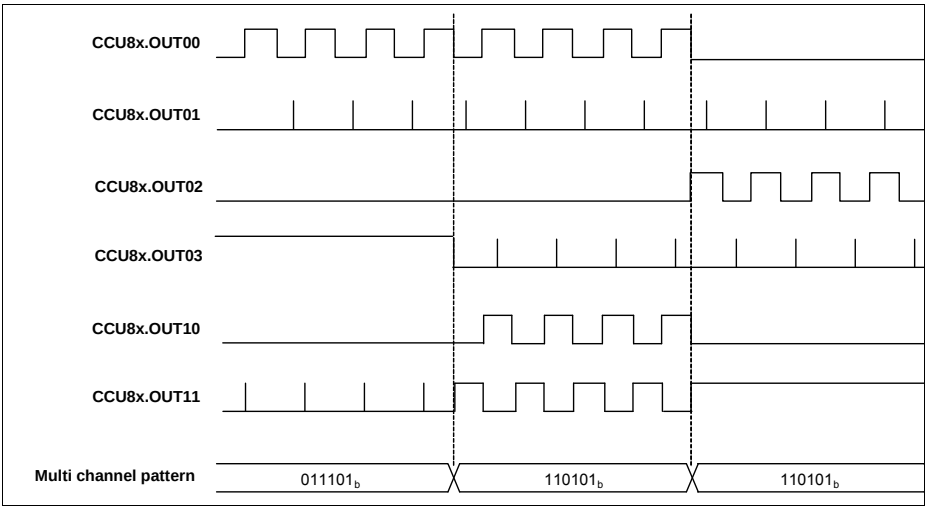

Figure 6 PWM output glitch

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
```

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.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 7**, 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)

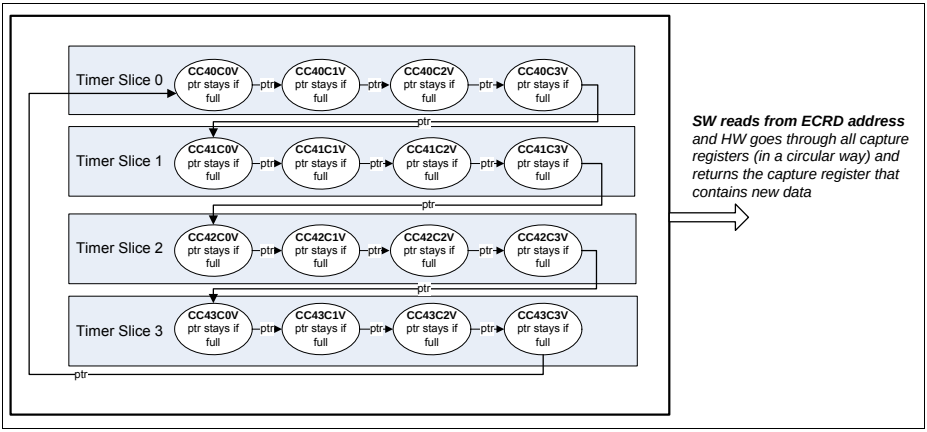

Figure 7 Extended Read Back access - example for CCU4 (CCU8 structure is the same)

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.

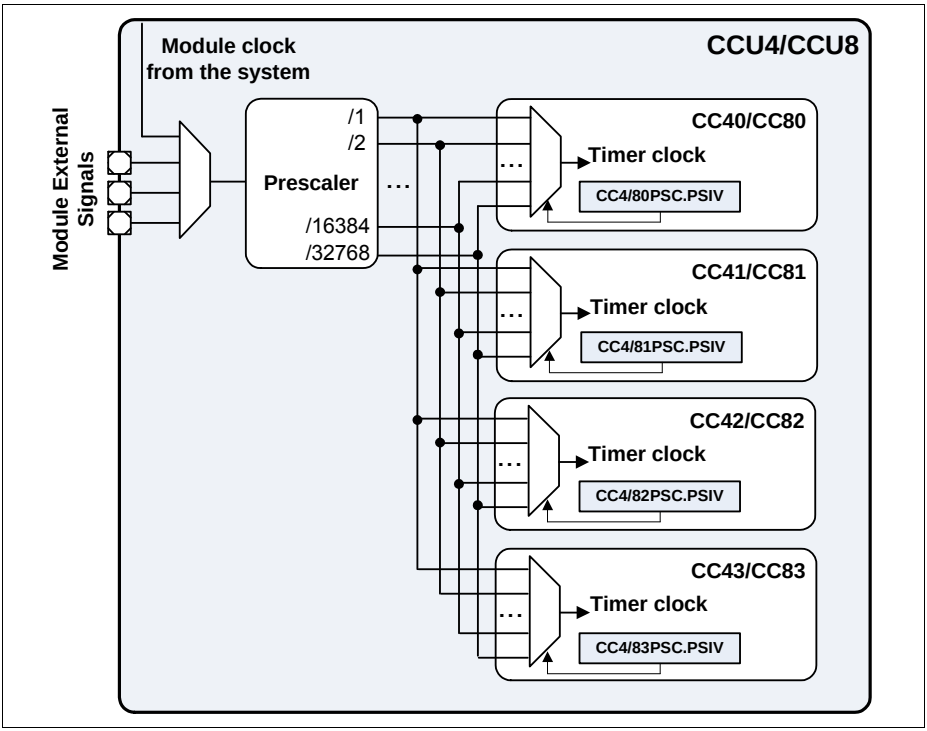

Figure 8 Clock Selection Diagram for CCU4/CCU8

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 duty-cycle (compare value)
- DITHE = 10_B - dither applied to the period (period 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.

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.

HRPWM_AI.001 HRPWM output signal interference while using two control sources

The High Resolution PWM (HRPWM) unit has 4 High Resolution Channel (HRC) sub modules. These are the sub modules that are used to extend the normal PWM resolution up to 150 ps. The structure of each of these sub modules is depicted in **Figure 9**.

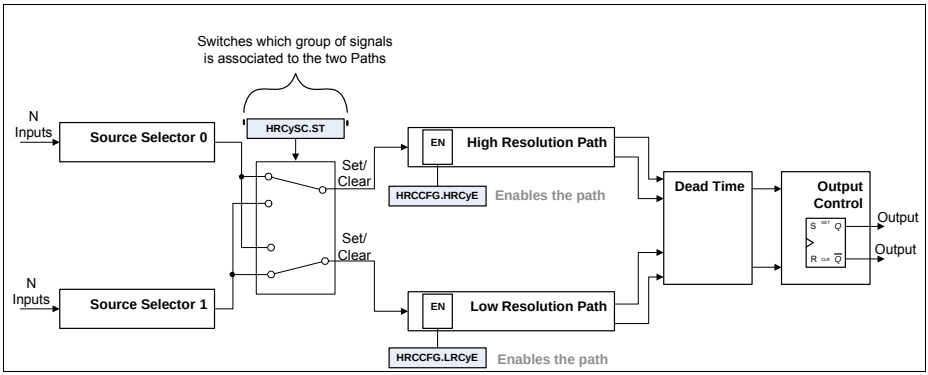

Figure 9 High Resolution Channel (HRC) simplified block diagram

From the scheme of each HRC, it is possible to control the output PWM signal via two Source Selectors (Source Selector 0 and Source Selector 1). Each of these Source Selectors can generate a pair of PWM set and clear signals (that are propagated to the output control stage).

When both paths are being used (High Resolution Path and Low Resolution Path), the Source Selector 0 output signals always have priority over the signals generated via Source Selector 1.

At any given instant a bitfield that controls which Source Selector is connected to which path can be updated by Software, via the HRCyST.ST bitfield.

When only one path is used (only High Resolution Path or only High Resolution Path) the software can still update at any given time which is the Source Selector controlling this specific path (via the HRCyST.ST bitfield).

This scenario where only one path is enabled but both Source Selectors are being used, and by being used is understood that signals are being actively decoded from the Source Selectors (input signals for the Source Selectors are being updated) is depicted on **Figure 10** - in this example the High Resolution Path is the one being used.

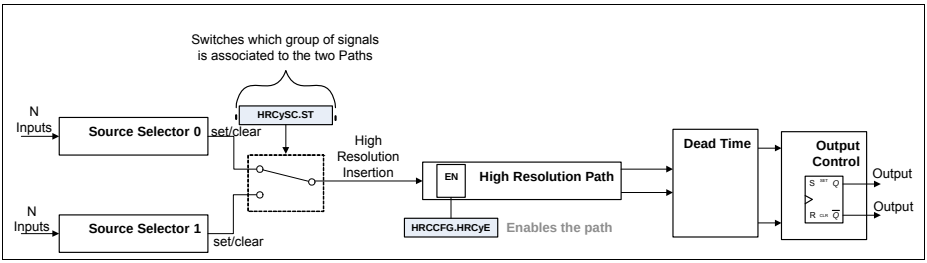

Figure 10 HRC with only one path enabled and two Source Selectors

In the above scenario, every time that the HRCyST.ST bitfield is 1_B, the Source Selector 1 is the one that has control over the PWM signal. Whenever the HRCyST.ST bitfield is 0_B is the Source Selector 0 that has control over the PWM signal.

An issue exists whenever the Source Selector 1 is selected (HRCyST.ST = 1_B) and there is a collision between signals from both Source Selectors. A collision is understood has:

- at the same module clock time frame Source Selector 0 and Source Selector 1 both decode a signal (a PWM set or PWM clear).

If the described condition occurs, then the signal generated by Source Selector 1 is lost and the PWM signal is not handled properly - **Figure 11**.

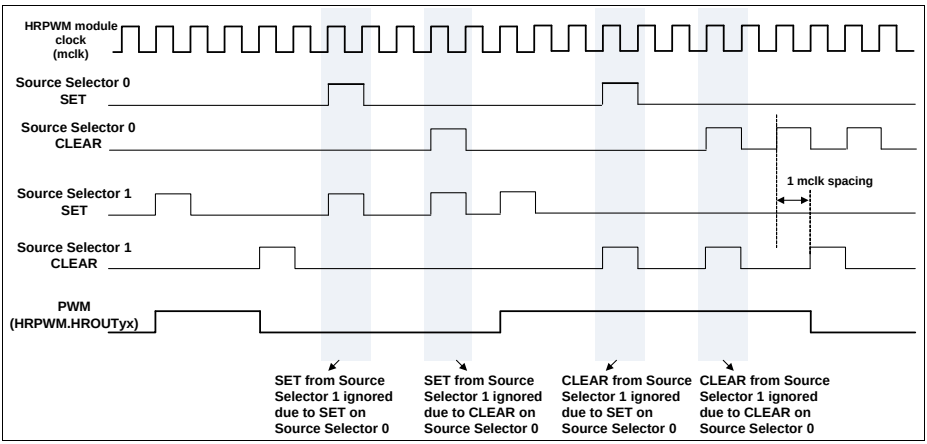

Figure 11 Collision between signals of the two Source Selectors - valid

only when HRCyST.ST = 1_B

Workaround 1

Ensuring that the signals controlling both Source Selectors are spaced by a minimum of 1 HRPWM module clock cycle. This is indeed only necessary if is currently Source Selector 1 the one selected to control the PWM path (HRCyST.ST = 1_B).

Workaround 2

Ensuring that the signals controlling Source Selector 0 are stopped whenever Source Selector 1 the one selected to control the PWM path (HRCyST.ST = 1_B).

HRPWM_AI.002 HRPWM CSG missing DAC conversion trigger in static mode

The High Resolution PWM (HRPWM) unit has 3 Comparator and Slope Generation (CSG) sub modules. These sub modules are comprised of a DAC an analog Comparator and additional control logic. The structure of each of these sub modules is depicted in **Figure 9**.

Each CSG has two major modes that can be used to control the DAC values:

- Static Mode
- Slope Generation Mode

The selection between these two modes can be done via the CSGySC.SCM field. When the SCM field is set to 0_H, the static mode is selected.

When the static mode is selected, the DAC is only converting a single value and is not generating any type of automatic pattern. Only the software can update the value that is converted by the DAC. This value needs to be written into the CSGyDSV1 or CSGyDSV2 registers.

When the static mode is selected (SCM = 0_H), the DAC conversion trigger is not generated whenever the CSGyDSV1 or CSGyDSV2 values are updated via software.

This issue imposes that during run time, it is not possible to update on-the-fly the DAC value when the static mode is selected (SCM = 0_H), without additional software routines or additional hardware resources.

When a pattern is being generated automatically by the hardware (SCM != 0_H), this issue does not exist.

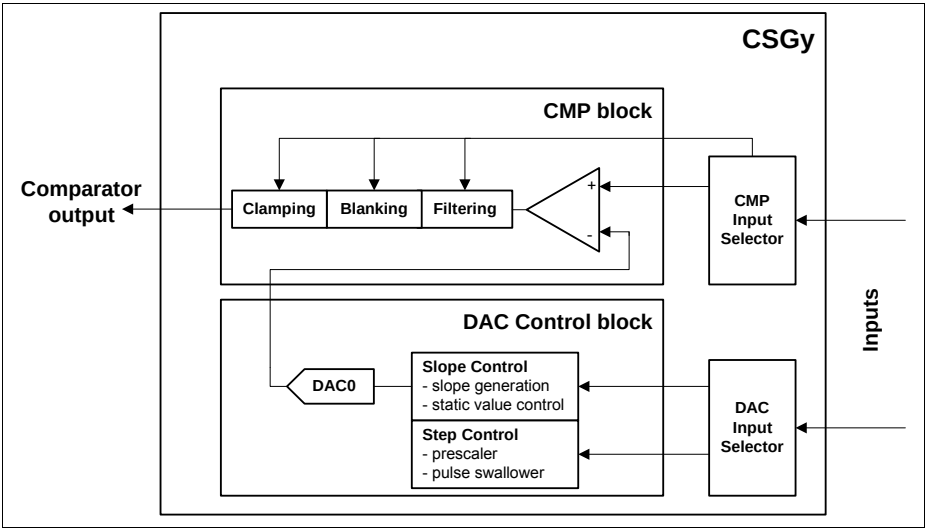

Figure 12 High Resolution Channel (HRC) simplified block diagram

Workaround

Every time that the DAC value needs to be updated (with SCM = 0_H), the DAC control logic run bit should be cleared and then re set.

When the CSGyDSV2 register is being used, then the following code should be implemented:

```
//configures the CSGyDSV2 value has the one used for the
//input register (y is the sub module instance)- this is
//one time configuration at startup
CSGySWSM = 0;

//During the run time the loop to update the DSV2 value is
//as follows (y is the sub module instance)
CSGyDSV2 = [NEW_VALUE];
CSGCLRG.CDyR = 1; //clears the DAC control logic run bit
CSGSETG.SDyR = 1; //sets the DAC control logic run bit
```

When the CSGyDSV1 register is being used, then the following code should be implemented:

```
//configures the CSGyDSV1 value has the one used for the
//input register (y is the sub module instance)- this is
//one time configuration at startup
CSGySWSM = 1;
CSGySC.IST = 1;

//During the run time the loop to update the DSV2 value is
//as follows (y is the sub module instance)
CSGySDSV1 = [NEW_VALUE];
CSGTRG = 0x1 //in case the CSG0 is being used. If other CSG
is being used then the specific bit position should be
written
CSGCLRG.CDyR = 1; //clears the DAC control logic run bit
CSGSETG.SDyR = 1; //sets the DAC control logic run bit
```

HRPWM_AI.003 HRPWM Usage Limitations

The High Resolution PWM (HRPWM) is comprised of 4 High Resolution Channels (HRC) and 3 Comparator and Slope Generation sub modules. The general structure of the peripheral can be seen in **Figure 9** (HRPWM = 3xCSGs + 4xHRCs).

Each HRC sub module is able to generate a complementary PWM output signal with a resolution of 150 ps.

The CSG sub modules contain a high speed comparator and a 10 bit high speed DAC for reference generation. It is also possible to use the CSG to generate sawtooth and triangular waveforms via the 10 bit DAC.

Due to power connection issues, in this current device step, the CSG sub modules are not functional.

Also due to power connection issues, the HRC sub modules can have the following problems:

- High Resolution Channel 3 may stop working due to internal voltage drop.

- Generation of the 150 ps resolution of all the HRCs may not ramp up properly during module initialization (temperature dependency). If this happens, then all the HRC channels are not able to generate a 150 ps resolution.The proper ramp can be check by polling the HRGHRS.HRGR bitfield.

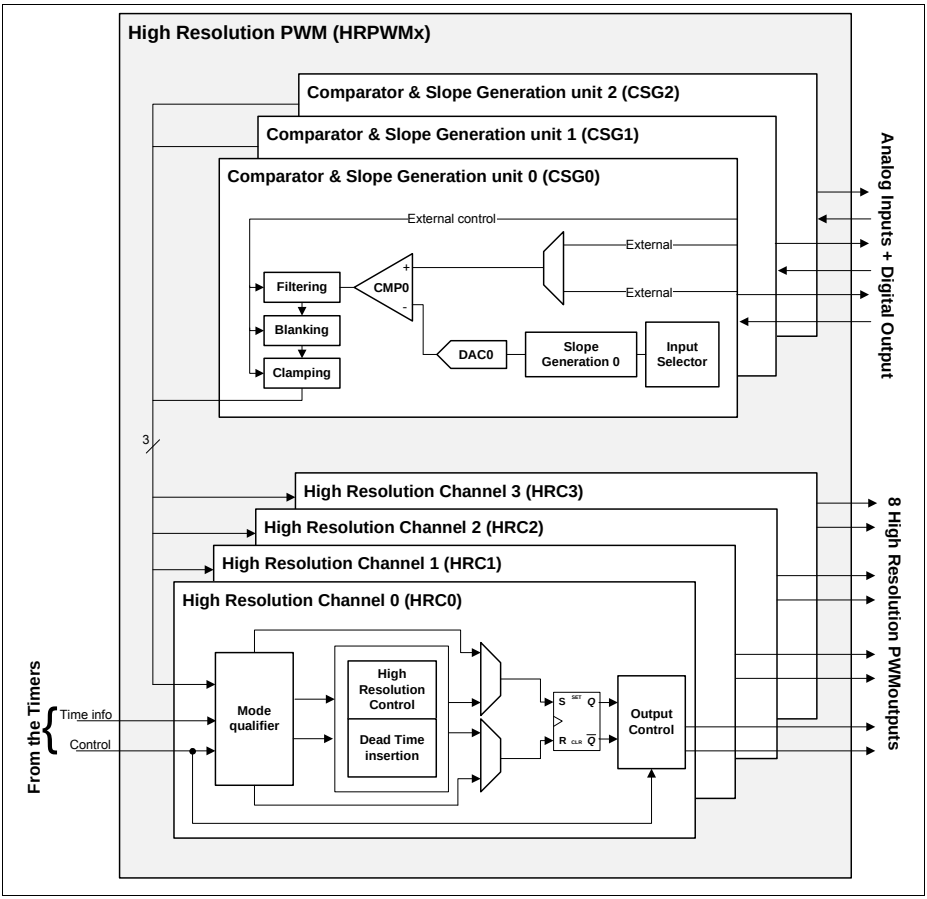

Figure 13 High Resolution Channel (HRC) simplified block diagram

Workaround

None.

HRPWM_AI.004 HRPWM Peripheral Bus Clock Limitation

The High Resolution PWM (HRPWM) peripheral uses two clocks generated by the SCU, the module clock (defined as f_{ccu} in the System Control Unit) and the peripheral bus clock (defined as f_{periph} in the System Control Unit).

The module clock is the one being used by the HRPWM kernel. The peripheral bus clock is the one being used by the interface between the CPU and the HRPWM to write and read back the module registers.

Due to synchronization issues, the HRPWM module clock frequency needs to be the same as the peripheral bus clock frequency ($f_{periph} < f_{ccu}$).

The peripheral bus clock frequency is controlled via the PBCLKCR.PBDIV field. The HRPWM module clock frequency is controlled via the CCUCLKCR.CCUDIV field. Both register are located in the SCU (System Control Unit).

Workaround

The peripheral bus clock and the module clock of the HRPWM need to have the same frequency ($f_{periph} = f_{ccu}$).

This is achieved by setting accordingly the following three fields:

- the CCUCLKCR.CCUDIV (field controlling the HRPWM and CCU4, CCU8 and POSIF peripherals module clock ratio);
- the CPUCLKCR.CPUDIV (field controlling the CPU clock division ratio);
- the PBCLKCR.PBDIV (field controlling the peripheral bus clock division ratio)

Table 6 shows the valid combinations for each of the previously mentioned fields. Notice that the frequency values given in the table are just an example (the real frequency would depend on the specific device and the PLL configuration).

Table 6 Valid Clock Combinations

Valid	CCUCLKCR.CCUDIV (frequency example)	CPUCLKCR.CPUDIV (frequency example)	PBCLKCR.PBDIV (frequency example)
Yes	0 (120 MHz)	0 (120 MHz)	0 (120 MHz)
No	0 (120 MHz)	0 (120 MHz)	1 (60 MHz)

Table 6 Valid Clock Combinations (cont'd)

Valid	CCUCLKCR.CCUDIV (frequency example)	CPUCLKCR.CPUDIV (frequency example)	PBCLKCR.PBDIV (frequency example)
No	0 (120 MHz)	1 (60 MHz)	0 (60 MHz)
Yes	1 (60 MHz)	0 (120 MHz)	1 (60 MHz)
Yes	1 (60 MHz)	1 (60 MHz)	0 (60 MHz)

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 14**).

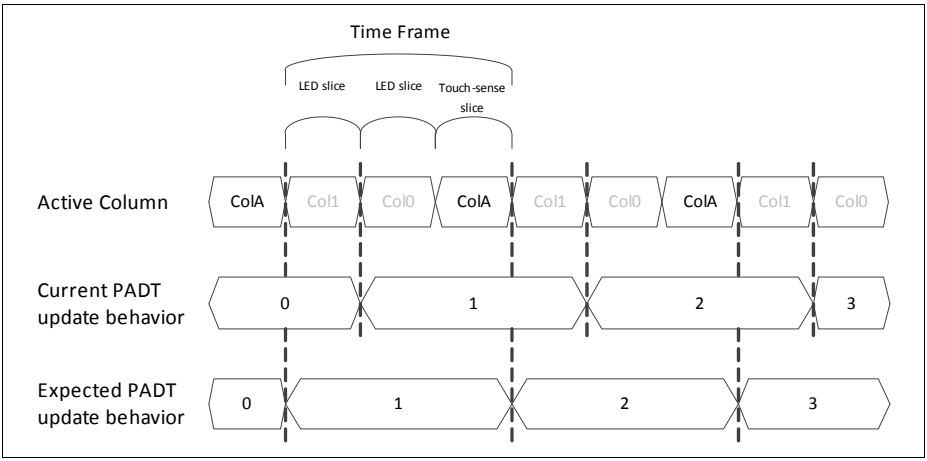

Figure 14 PADT update behavior

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 15**). In addition to this, when no LEDs

are enabled, pad turn 0 will prevail for one time slice longer before it gets updated (**Figure 16**).

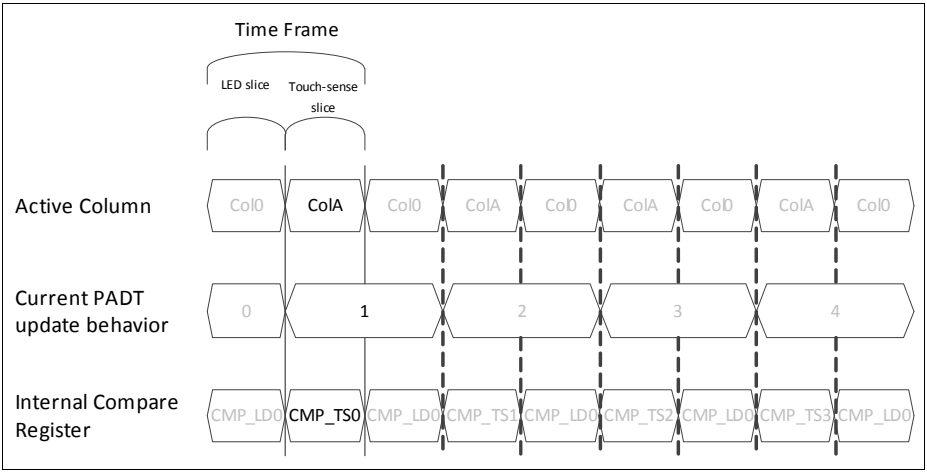

Figure 15 Effect of delay on the update of Internal Compare Register with 1 LED column enabled

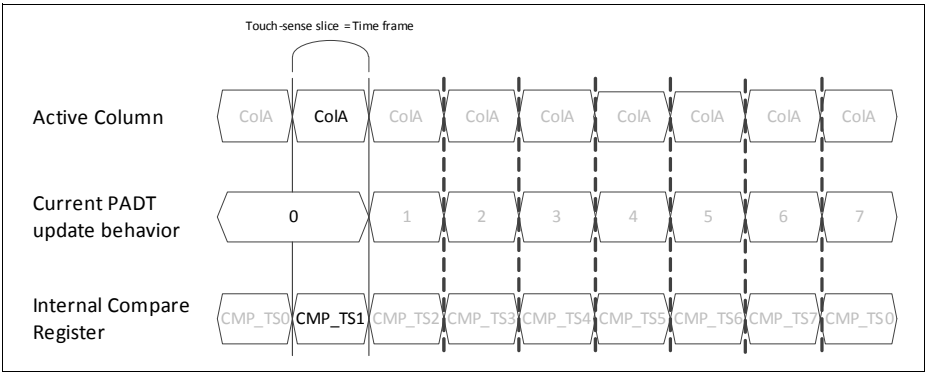

Figure 16 Pad turn 0 prevails for one time slice longer when no LEDs are enabled

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 17**).

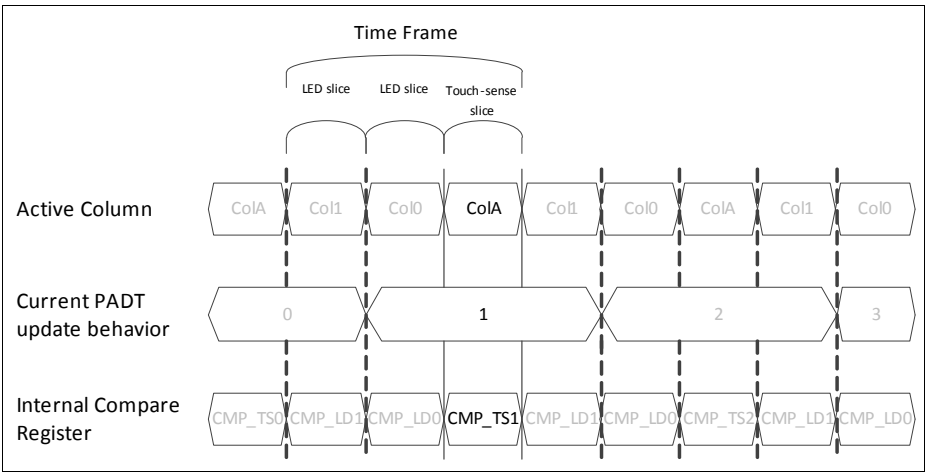

Figure 17 Internal Compare Register updated with correct compare register value with 2 LED columns enabled

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

Table 7 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 18**).

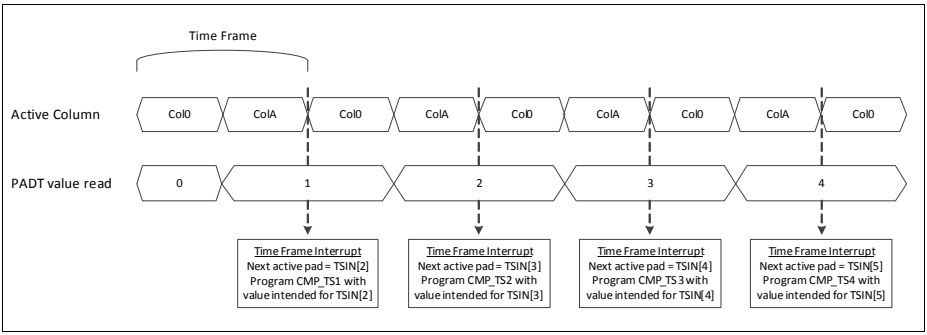

Figure 18 Software workaround demonstration

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.003 P14_PDISC register can’t be written

The Ports register P14_PDISC can’t be written by software. Depending on the device and package variant, the devices are configured with

1. the digital input path of the analog/digital input pins disabled, limiting the pins to the analog function
   a) PG-LQFP-64: P14_PDISC = 0000_43F9$_H$

b) PG-VQFN-48: P14_PDISC = 0000_03F9$_H$

2. the digital input path of the analog/digital input pins enabled, allowing the operation as analog and/or digital input (P14_PDISC = 0000_0000$_H$)

Either of the above configurations is static, software can't change the register value.

Implications

Software that programs and reads back the P14_PDISC register will not see the expected read value, but always reads the pre-configured values.

Workaround

In devices shipped in the first configuration, P14 pins can't be used as digital inputs. There is no workaround to use these pins as digital inputs.

Devices shipped in the second configuration support both, analog and digital functiontality. As such the device operation is not affected, except for the fact, that the digital input path can't be disabled.

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 8 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.

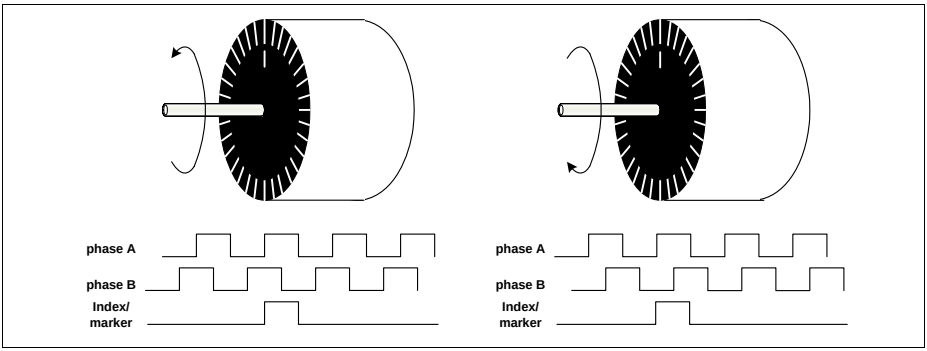

Figure 19 Rotary Encoder outputs - Phase A, Phase B and Index

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).

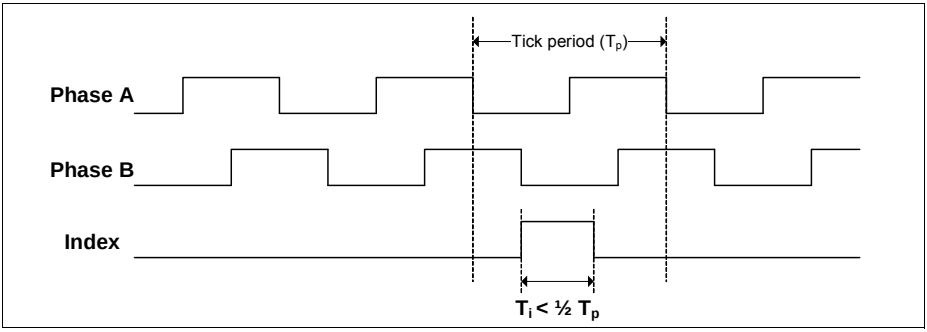

Figure 20 Different index signal types

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.

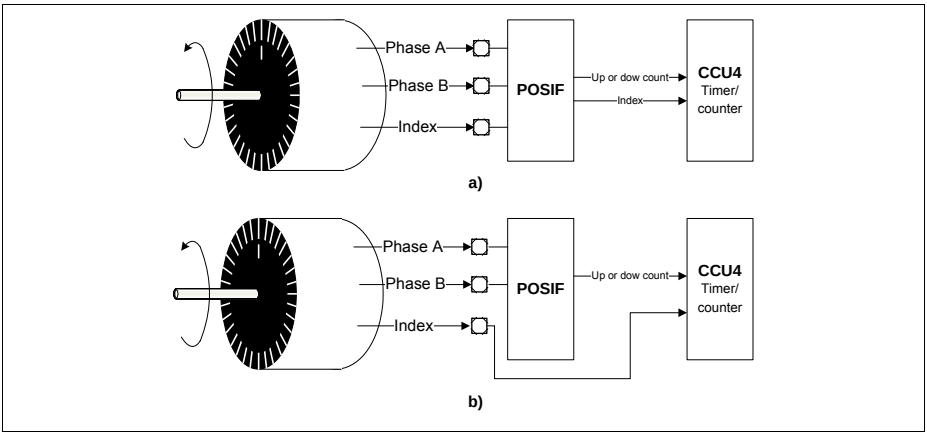

Figure 21 Index usage workaround - a) Non working solution; b) Working solution

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.009 Out of Range Comparator Control

The device provides the following Out of Range Comparator Inputs:

- G0ORC6
- G1ORC6

Control of G1ORC6 input is erroneously assigned to the G0ORC7 control logic.

Implications

Bit ENORC6 of SCU register G1ORCEN has no function and the related SCU output G1ORCOUT6 is not connected to ERU0.

Workaround

To enable G1ORC6 input, use bit ENORC7 of SCU register G0ORCEN. This will enable the path from G1ORC6 device input to SCU output G0ORCOUT7. Note that consequently you must use the ERU0 input 1A3 for further processing of the signal.

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.

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.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.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.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 9

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.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.017 Clock phase of data shift in SSC slave cannot be changed

Setting PCR.SLPHSEL bit to 1 in SSC slave mode is intended to change the clock phase of the data shift such that reception of data bits is done on the leading SCLKIN clock edge and transmission on the other (trailing) edge.

However, in the current implementation, the feature is not working.

Workaround

None.

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.019 First data word received by IIC receiver triggers RIF instead of AIF

When operating in IIC mode as a master or slave receiver, the first data word received following a start condition and address match triggers a receive event (indicated by PSR.RIF flag) instead of an alternate receive event (indicated by PSR.AIF flag).

Workaround

To determine if a received data word is the first word of a new frame, bit 9 of RBUF needs to be read:

- When RBUF[9] is 1, the first data word of a new frame is indicated;
- When RBUF[9] is 0, subsequent data words of the frame are indicated.

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.

3 Deviations from Electrical- and Timing Specification

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

POWER_CM.P001 Risk of increased current consumption in internally controlled hibernate mode

When

- the device is in internally controlled hibernate mode and
- if a voltage difference of (V_{DDP} - V_{BAT}) > 1 V is present

then I_{DDPH} may be up to 50 µA higher than specified.

Recommendation

When the device is in internally controlled hibernate mode it must be ensured that V_{BAT} level is maintained close to the level of V_{DDP} .

For details please refer to the “System Level Integration” examples of the “Hibernate Control” section in the Reference Manual.

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.

LEDTS_AI.H001 Column A is unavailable

Column A is not connected to any port pin in this device.

Implication

Related module use cases are restricted.

Recommendation

When using touch-sense function, the option to use external pull resistor is unavailable. Register bit FNCTL.EPULL should always be configured with the value of 0.

If only LED function is used, one time slice will have to be used as dummy (active column with no LEDs or touch pads connected) because Column A is unavailable. A minimum of 2 LED columns will have to be configured (FNCTL.NR_LEDCOL = 001_B).

If both LED and touch-sense functions are enabled, there is no impact on LED columns and it is possible to configure 1 LED column (FNCTL.NR_LEDCOL = 000_B) only.

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:

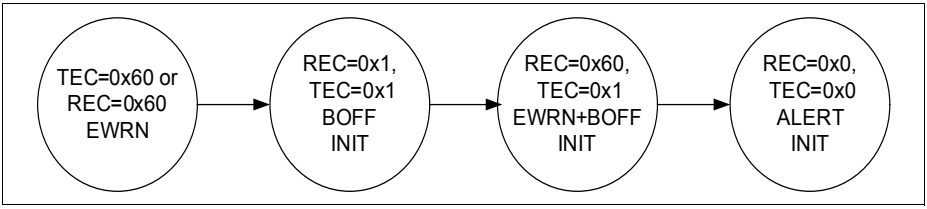

Figure 22 Alert Interrupt Behavior in case of Bus-Off

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.

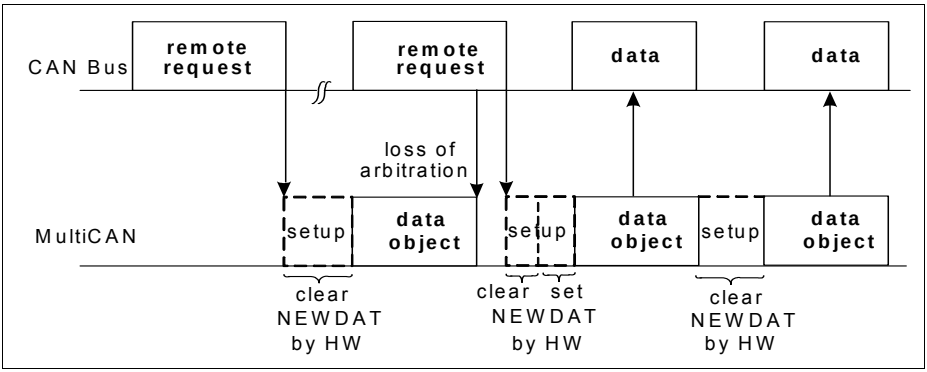

Figure 23 Loss of Arbitration

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.