Errata Sheet, Rel. 1.5, Aug. 2008

XC2300 Derivatives

16/32-Bit Single-Chip Microcontroller with 32-Bit Performance

Microcontrollers

Never stop thinking

Edition 22.08.2008

Published by
Infineon Technologies AG
81726 Munich, Germany

© 2008 Infineon Technologies AG
All Rights Reserved.

Legal Disclaimer

The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics. With respect to any examples or hints given herein, any typical values stated herein and/or any information regarding the application of the device, Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind, including without limitation, warranties of non-infringement of intellectual property rights of any third party.

Information

For further information on technology, delivery terms and conditions and prices, please contact the nearest Infineon Technologies Office (**www.infineon.com**).

Warnings

Due to technical requirements, components may contain dangerous substances. For information on the types in question, please contact the nearest Infineon Technologies Office.

Infineon Technologies components may be used in life-support devices or systems only with the express written approval of Infineon Technologies, if a failure of such components can reasonably be expected to cause the failure of that life-support device or system or to affect the safety or effectiveness of that device or system. Life support devices or systems are intended to be implanted in the human body or to support and/or maintain and sustain and/or protect human life. If they fail, it is reasonable to assume that the health of the user or other persons may be endangered.

XC2300 Derivatives

16/32-Bit Single-Chip Microcontroller with 32-Bit Performance

Microcontrollers

Never stop thinking

Table of Contents

1 History List / Change Summary 6
2 General 7
3 Current Documentation 8
4 Errata Device Overview 9
4.1 Functional Deviations 9
4.2 Deviations from Electrical and Timing Specification 12
4.3 Application Hints 13
5 Short Errata Description 14
5.1 Errata removed in this Errata Sheet 14
5.2 Functional Deviations 15
5.3 Deviations from Electrical and Timing Specification 18
5.4 Application Hints 19
6 Detailed Errata Description 20
6.1 Functional Deviations 20
ADC_AI.001 20
BSL_X.003 20
BSL_X.004 21
CPU_X.004 21
DPRAM_X.001 23
FLASH_X.007 24
FLASH_X.008 24
GPT12E_X.001 25
GSC_X.001 25
INT_X.007 26
INT_X.008 27
INT_X.009 28
INT_X.010 30
MultiCAN_AI.040 31
MultiCAN_AI.041 31
MultiCAN_AI.042 32
MultiCAN_AI.043 32
MultiCAN_AI.044 33
MultiCAN_AI.045 33
MultiCAN_AI.046 34
MultiCAN_TC.025 34
MultiCAN_TC.026 35
MultiCAN_TC.027 35
MultiCAN_TC.028 35

MultiCAN_TC.029 37
MultiCAN_TC.030 38
MultiCAN_TC.031 38
MultiCAN_TC.032 39
MultiCAN_TC.035 39
MultiCAN_TC.037 41
MultiCAN_TC.038 42
OCDS_X.003 42
POWER_X.003 43
POWER_X.005 43
RESET_X.002 44
RESET_X.003 44
RTC_X.003 44
USIC_AI.003 45
USIC_AI.004 45
6.2 Deviations from Electrical and Timing Specification 46
6.3 Application Hints 47
CAPCOM12_X.H001 47
CC6_X.H001 48
INT_X.H002 48
JTAG_X.H001 49
MultiCAN_AI.H005 50
MultiCAN_AI.H006 50
MultiCAN_TC.H002 50
MultiCAN_TC.H003 51
MultiCAN_TC.H004 51
OCDS_X.H002 52
RESET_X.H002 53
RESET_X.H003 54
RTC_X.H003 54
StartUp_X.H002 54
USIC_AI.H001 55
WDT_X.H001 55

1 History List / Change Summary

Table 1 History List

Version	Date	Remark
1.0	13.07.2007	
1.1	27.07.2007	
1.2	19.10.2007	
1.3	11.04.2008	Errata Sheet for all product steps.
1.4	20.08.2008	Cancelled release due missing note.
1.5	22.08.2008	

2 General

This Errata Sheet describes the deviations of the XC2300 Derivatives from the current user documentation.

Each erratum identifier follows the pattern **Module_Arch.TypeNumber**:

- **Module**: subsystem, peripheral, or function affected by the erratum
- **Arch**: microcontroller architecture where the erratum was firstly detected.
  - **AI**: Architecture Independent
  - **CIC**: Companion ICs
  - **TC**: TriCore
  - **X**: XC166 / XE166 / XC2000 Family
  - **XC8**: XC800 Family
  - **[none]**: C166 Family
- **Type**: category of deviation
  - **[none]**: Functional Deviation
  - **P**: Parametric Deviation
  - **H**: Application Hint
  - **D**: Documentation Update
- **Number**: ascending sequential number within the three previous fields. As this sequence is used over several derivatives, including already solved deviations, gaps inside this enumeration can occur.

This Errata Sheet applies to all temperature and frequency versions and to all memory size variants of this device, unless explicitly noted otherwise.

Note: This device is equipped with a C166S V2 core. Some of the errata have a workaround which is possibly supported by the compiler tool vendor. Some corresponding compiler switches need possibly to be set. Please see the respective documentation of your compiler.

Some errata of this Errata Sheet do not refer to all of the XC2300 Derivatives, please look to the overview:

Table 2 for Functional Deviations,

Table 3 for Deviations from Electrical and Timing Specification and

Table 4 for Application Hints.

3 Current Documentation

The Infineon XC2000 Family comprises device types from the XC2200 group, the XC2300 group and the XC2700 group. The XC23xx device types belong to the XC2300 group.

Device	XC23xx
Marking/Step	EES-AA, EES-AB, ES-AB, AB, EES-AC, ES-AC, AC
Package	PG-LQFP-100, PG-LQFP-144

This Errata Sheet refers to the following documentation:

- XC2300 Derivatives User's Manual Volume 1 System Units
- XC2300 Derivatives User's Manual Volume 2 Peripheral Units
- XC2365 Data Sheet
- XC2387 Data Sheet
- Documentation Addendum (if applicable)

Make sure you always use the corresponding documentation for this device available in category 'Documents' at **www.infineon.com/xc2300**.

The specific test conditions for EES and ES are documented in a separate Status Sheet.

Note: Devices marked with EES or ES are engineering samples which may not be completely tested in all functional and electrical characteristics, therefore they should be used for evaluation only.

4 Errata Device Overview

This chapter gives an overview of the dependencies of individual errata to devices and steps. An **X** in the column of the sales codes shows that this erratum is valid.

4.1 Functional Deviations

Table 2 shows the dependencies of functional deviations in the derivatives.

Table 2 Errata Device Overview: Functional Deviations

Functional Deviation	XC23xx			
	AA[1]	AB[2]	AC[3]	
ADC_AI.001	X	X	X	
BSL_X.003	X			
BSL_X.004	X	X	X	
CPU_X.004	X			
DPRAM_X.001	X	X	X	
FLASH_X.007	X			
FLASH_X.008	X	X	X	
GPT12E_X.001	X	X	X	
GSC_X.001	X	X	X	
INT_X.007	X	X	X	
INT_X.008	X	X	X	
INT_X.009	X	X	X	
INT_X.010	X	X	X	
MultiCAN_AI.040	X	X	X	
MultiCAN_AI.041	X	X	X	
MultiCAN_AI.042	X	X	X	
MultiCAN_AI.043	X	X	X	

Table 2 Errata Device Overview: Functional Deviations (cont'd)

Functional Deviation	XC23xx			
	AA[1]	AB[2]	AC[3]	
MultiCAN_AI.044	X	X	X	
MultiCAN_AI.045	X	X	X	
MultiCAN_AI.046	X	X	X	
MultiCAN_TC.025	X	X	X	
MultiCAN_TC.026	X	X	X	
MultiCAN_TC.027	X	X	X	
MultiCAN_TC.028	X	X	X	
MultiCAN_TC.029	X	X	X	
MultiCAN_TC.030	X	X	X	
MultiCAN_TC.031	X	X	X	
MultiCAN_TC.032	X	X	X	
MultiCAN_TC.035	X	X	X	
MultiCAN_TC.037	X	X	X	
MultiCAN_TC.038	X	X	X	
OCDS_X.003	X	X	X	
POWER_X.003	X			
POWER_X.005	X			
RESET_X.002	X	X	X	
RESET_X.003	X	X	X	
RTC_X.003	X	X	X	
USIC_AI.003	X	X	X	
USIC_AI.004	X	X	X	

1) Valid for EES-AA.
2) Valid for EES-AB, ES-AB and AB.

3) Valid for EES-AC, ES-AC and AC. No errata fixing was done for these steps. The only purpose was yield optimization.

4.2 Deviations from Electrical and Timing Specification

Table 3 shows the dependencies of deviations from the electrical and timing specification in the derivatives.

Table 3 Product Overview: Deviations from Electrical and Timing Specification

Functional Deviation	XC23xx		
	AA[1]	AB[2]	AC[3]
- none -			

1) Valid for EES-AA.
2) Valid for EES-AB, ES-AB and AB.
3) Valid for EES-AC, ES-AC and AC. No errata fixing was done for these steps. The only purpose was yield optimization.

4.3 Application Hints

Table 4 shows the dependencies of application hints in the derivatives.

Table 4 Product Overview: Application Hints

Functional Deviation	XC23xx			
	AA[1]	AB[2]	AC[3]	
CAPCOM12_X.H001	X	X	X	
CC6_X.H001	X	X	X	
INT_X.H002	X	X	X	
JTAG_X.H001	X	X	X	
MultiCAN_AI.H005	X	X	X	
MultiCAN_AI.H006	X	X	X	
MultiCAN_TC.H002	X	X	X	
MultiCAN_TC.H003	X	X	X	
MultiCAN_TC.H004	X	X	X	
OCDS_X.H002	X	X	X	
RESET_X.H002	X			
RESET_X.H003	X	X	X	
RTC_X.H003	X	X	X	
StartUp_X.H002	X	X	X	
USIC_AI.H001	X	X	X	
WDT_X.H001	X			

1) Valid for EES-AA.
2) Valid for EES-AB, ES-AB and AB.
3) Valid for EES-AC, ES-AC and AC. No errata fixing was done for these steps. The only purpose was yield optimization.

5 Short Errata Description

This chapter gives an overview over the description and shows changes to the last Errata Sheet.

5.1 Errata removed in this Errata Sheet

Table 5 shows an overview of removed errata in this Errata Sheet. In column ***Change*** was described why it can remove.

Table 5 Errata removed in this Errata Sheet

Errata	Short Description	Change
INT_X.H001	Software Modifications of Interrupt Enable (xx_IE) or Interrupt Request (xx_IR) Flags	User's Manual[1)]
Leakage_X.D002	Pin Leakage Calculation Formula	Data Sheet[1)]
POWER_X.001	Wake-up Trigger during Power Transfers	Programmer's Guide[1)]
RESET_X.001	System Reset	User's Manual[1)]
RESET_X.H001	Register RSTSTAT0 after Power-on Reset for Domain DMP_1	User's Manual[1)]

1) An update in the documentation is done.

5.2 Functional Deviations

Table 6 shows a short description of the functional deviations.

Table 6 Functional Deviations

Functional Deviation	Short Description	Chg	Pg
ADC_AI.001	Conversions requested in Slot 0 started twice		20
BSL_X.003	Single Wire Support for UART Bootstrap Loader		20
BSL_X.004	Evaluation of UART Bootstrap Loader Identification Byte in Single Wire Configuration		21
CPU_X.004	Write requests to PSRAM, to IMB SFRs, or as part of Flash Command Sequences		21
DPRAM_X.001	Parity Error Flag for DPRAM		23
FLASH_X.007	Flash Erase Command “Erase Sector”		24
FLASH_X.008	Flash Read after Flash Erase Command		24
GPT12E_X.001	T5/T6 in Counter Mode with BPS2 = $1X_B$		25
GSC_X.001	Clearing of Request Triggers by the GSC		25
INT_X.007	Interrupt using a Local Register Bank during execution of IDLE		26
INT_X.008	HW Trap during Context Switch in Routine using a Local Bank		27
INT_X.009	Delayed Interrupt Service of Requests using a Global Bank		28
INT_X.010	HW Traps and Interrupts may get postponed		30
MultiCAN_AI.040	Remote frame transmit acceptance filtering error		31
MultiCAN_AI.041	Dealloc Last Obj		31

Table 6 Functional Deviations (cont'd)

Functional Deviation	Short Description	Chg	Pg
MultiCAN_AI.042	Clear MSGVAL during transmit acceptance filtering		32
MultiCAN_AI.043	Dealloc Previous Obj		32
MultiCAN_AI.044	RxFIFO Base SDT		33
MultiCAN_AI.045	OVIE Unexpected Interrupt		33
MultiCAN_AI.046	Transmit FIFO base Object position		34
MultiCAN_TC.025	RXUPD behavior		34
MultiCAN_TC.026	MultiCAN Timestamp Function		35
MultiCAN_TC.027	MultiCAN Tx Filter Data Remote		35
MultiCAN_TC.028	SDT behavior		35
MultiCAN_TC.029	Tx FIFO overflow interrupt not generated		37
MultiCAN_TC.030	Wrong transmit order when CAN error at start of CRC transmission		38
MultiCAN_TC.031	List Object Error wrongly triggered		38
MultiCAN_TC.032	MSGVAL wrongly cleared in SDT mode		39
MultiCAN_TC.035	Different bit timing modes		39
MultiCAN_TC.037	Clear MSGVAL		41
MultiCAN_TC.038	Cancel TXRQ		42
OCDS_X.003	Peripheral Debug Mode Settings cleared by Reset		42
POWER_X.003	External Supply of Digital Core Supply Voltage V_{DDI}		43
POWER_X.005	Start-up of DMP_1 at low temperature		43
RESET_X.002	Startup Mode Selection is not Valid in SCU_STSTAT.HWCFG		44
RESET_X.003	P2.[2:0] and P10.[12:0] Switch to Input	New	44
RTC_X.003	Interrupt Generation in Asynchronous Mode		44

Table 6 Functional Deviations (cont'd)

Functional Deviation	Short Description	Chg	Pg
USIC_AI.003	TCSRL.SOF and TCSRL.EOF not cleared after a transmission is started		45
USIC_AI.004	Receive shifter baudrate limitation		45

5.3 Deviations from Electrical and Timing Specification

Table 7 shows a short description of the electrical- and timing deviations from the specification.

Table 7 Deviations from Electrical and Timing Specification

AC/DC/ADC Deviation	Short Description	Chg	Pg
- none -			

5.4 Application Hints

The Table 8 shows a short description of the application hints.

Table 8 Application Hints

Hint	Short Description	Chg	Pg
CAPCOM12_X.H001	Enabling Single Event Operation	New	47
CC6_X.H001	Modifications of Bit MODEN in Register CCU6x_KSCFG		48
INT_X.H002	Increased Latency for Hardware Traps		48
JTAG_X.H001	JTAG Pin Routing		49
MultiCAN_AI.H005	TxD Pulse upon short disable request	Update	50
MultiCAN_AI.H006	Time stamp influenced by resynchronization	New	50
MultiCAN_TC.H002	Double Synchronization of receive input		50
MultiCAN_TC.H003	Message may be discarded before transmission in STT mode		51
MultiCAN_TC.H004	Double remote request	Update	51
OCDS_X.H002	Suspend Mode Behavior for MultiCAN		52
RESET_X.H002	Oscillator fail detection active by default	New	53
RESET_X.H003	How to Trigger a PORST after an Internal Failure	New	54
RTC_X.H003	Changing the RTC Configuration		54
StartUp_X.H002	FCONCS0..FCONCS4 Registers are Always Configured in External Start-Up Mode	New	54
USIC_AI.H001	FIFO RAM Parity Error Handling		55
WDT_X.H001	Watchdog Timer Reset Functionality not enabled by Default		55

6 Detailed Errata Description

This chapter shows a detailed description of the erratum.

6.1 Functional Deviations

ADC_AI.001 Conversions requested in Slot 0 started twice

A conversion n+1 requested in arbiter slot 0 will be started twice if all configuration and timing conditions of the following sequence are met:

1. A conversion n of a channel is currently running.
2. Slot 0 has won the arbitration while conversion n is in progress.
3. Conversion n ends one f_{ADCD} clock cycle before the end of an arbitration cycle.
4. The conversion n+1 initiated in slot 0 is started exactly in the last f_{ADCD} clock cycle of this arbitration-cycle (see 3.).
5. The conversion time of conversion n+1 is shorter than 2 arbitration cycles.

If all these conditions are met, then the request of slot 0 cannot be cleared in time by the arbiter, and conversion n+1 is requested a second time.

Workaround

The conversion time for channels requested in slot 0 must not be shorter than two arbitration cycles.

BSL_X.003 Single Wire Support for UART Bootstrap Loader

In the current implementation, if the UART bootloader is used in a single wire configuration (RxD/TxD externally connected, e.g. K-line environment), the following restriction must be considered:

The identification byte $D5_H$ that is sent by the microcontroller is also received and interpreted as first (of 32) opcode byte(s). $D5_H$ represents the opcode of the 4-byte instruction ’MOVBS mem, reg’.

Workaround

The host program must only send 31 (instead of 32) bytes. The first 3 bytes should complement $D5_H$ to a useful (or at least harmless) instruction, e.g.

`(D5) 8E 1E FF` results in `MOVBS ONES, ZEROS`.

In case the secondary bootstrap loader described in the 'System Units' Manual is used, the final `JMPS SEG Code_Start, SOF Code_Start` instruction can be omitted to compensate for the initial $D5_H$ + 3 'fill bytes'. In this case the start address of the secondary loader must be 0xE00020.

BSL_X.004 Evaluation of UART Bootstrap Loader Identification Byte in Single Wire Configuration

In the current implementation, transmission of the start bit of the identification byte ($D5_H$) partially overlaps with the stop bit time slot of the zero byte sent by the host. This does not present any problem in a duplex (2-wire) configuration.

If the UART bootstrap loader is used in a single wire configuration (RxD/TxD externally connected, e.g. K-line environment), depending on the baudrate, the start bit of the identification byte may not be correctly recognized by the host. At 9600 Baud, the host typically interprets the identification byte as $F5_H$.

Workaround

The host software either should not evaluate the received identification byte, or should also tolerate values other than $D5_H$.

CPU_X.004 Write requests to PSRAM, to IMB SFRs, or as part of Flash Command Sequences

Under exceptional pipeline conditions, write requests to specific memory areas are not executed if they coincide within one clock cycle with read requests from specific address ranges.

Write requests to the following memory areas can get lost:

- Writes to variables located in the Program SRAM (PSRAM) in segment $E0_H$ (minimum access time) or in segment $E8_H$ (with Flash access timing)

- Writes to IMB SFRs (`IMB_...`, see table in chapter about Program Memory Control in User's Manual), located in the last 256 bytes of segment FF_H
- Writes to the internal Program Flash address range (mapped in segments $C0_H$ and higher), i.e. Flash Command Sequences may either not be executed at all (single cycle commands), or may be executed incorrectly (potentially resulting in a sequence error)

A necessary condition for losing a write request is that the write request is followed by a particular read request that gets cancelled (e.g. operand read after mispredicted conditional jump, etc.).

Read requests that potentially can cause this critical scenario are the following:

- Operand reads from the internal Program Flash
- Operand reads from the PSRAM in segment $E8_H$ (with Flash access timing)
- Operand reads from the External Memory Area via the EBC in the lower 32 Kbytes of segment 0_H, in segments 1_H...$1F_H$, and in segments 40_H...BF_H, **excluding** the I/O area (segments 20_H...$3F_H$)

Instruction fetches from internal Program Flash or PSRAM, or operand reads from the I/O area (segments 20_H...$3F_H$, including USIC and CAN modules) can not trigger the problematic condition.

Example

```
1   MOV [-R1], R12        ; write operand to PSRAM
                          ; assumed: R1 points to PSRAM
2   JMPA+ cc_NN, label1 ; jump predicted taken but not taken
                          ; assumed: R12 is negative
3    ...
...
label1:
n   MOV R3, [R0]          ; read operand from Flash
                          ; assumed: R0 points to Flash
```

When instruction `n` is cancelled due to the mispredicted jump `2` this will erroneously suppress the write to PSRAM by instruction `1`. Note that other functions (besides the write) of this instruction `1` will be performed, e.g. the pre-decrement of R1 due to the addressing mode `[-R1]`.

Workarounds

Because the timing conditions under which this problem occurs are quite complex the proposed workarounds are more restrictive than absolutely necessary. Depending on whether operand write or read accesses are more performance relevant (or easier to control, respectively), one of the following workarounds may be chosen:

Workaround1:

This workaround limits write requests to the critical memory areas:

Do not enable the PSRAM as RAM in the compiler (locator options), use PSRAM as ROM only. The only software routines that write to the critical memory areas are then the driver to perform the Flash Command Sequences, and the routine that copies (constant) data and code from Flash to PSRAM. These pieces of software must be carefully examined for the error conditions and must be verified (including writes to IMB SFRs in the initialization code).

Workaround2:

This workaround allows to use the PSRAM as RAM for variable storage while controlling problematic operand reads:

Perform operand reads from critical memory areas (see list for operand reads above) in a controlled way e.g. in a separate function. This function may read the required operands when no write access to the PSRAM has been performed by the last 4 instructions. Interrupts should be disabled in case PEC transfers to the PSRAM are used.

Note: In case a debugger is used, the 'window refresh' feature for windows covering an internal flash section containing (constant) operands that are also read by the program should be disabled while the program is running

DPRAM_X.001 Parity Error Flag for DPRAM

The parity error flag for the dual port memory (DPRAM) does not work correctly. Under certain conditions bit `PECON.PEFDP` is set, although there is no error in the DPRAM.

Workaround

Do not enable the parity error trap for the dual port memory, i.e. leave bit `PECON.PEENDP` = 0 (default after power-on reset).

FLASH_X.007 Flash Erase Command "Erase Sector"

Do not use the "Erase Sector" command.

Workaround

When programming a device for the first time (after delivery from Infineon) it is recommended to program it without erasing. When later data shall be replaced use only the "Erase Page" command before programming the new data.

FLASH_X.008 Flash Read after Flash Erase Command

Under certain conditions all Flash erase commands do not work correctly. After erasing, all erased bits must be programmed with new data or with all-zero data before reading any data from the addressed sector is allowed.

Workarounds

1. Erase a range of Flash memory and program it completely with new data before reading. This is the fastest solution.
   Additional hint: A Flash driver could implement a programming function that performs first an "Erase Page" and uses directly thereafter "Program Page" to program the data of this page. The Flash driver wouldn't need any separate erase function.
2. Erase a range of Flash memory and program it completely with all-zero data. Only after this the range may be read. Data can be programmed later[1].

1) Please note: only in order to implement this workaround for the noted device steps it is allowed to execute two program commands before erasing it.

<u>GPT12E_X.001</u> T5/T6 in Counter Mode with BPS2 = $1X_B$

When T5 and/or T6 are configured for counter mode (bit field TxM = 001_B in register GPT12E_TxCON, x = 5, 6), **and** bit field **BPS2 = $1X_B$** in register GPT12E_T6CON, then edge detection for the following count input and control signals does not work correctly:

T5IN, T6IN, T5EUD, T6EUD.

Note: The configuration where T5 counts the overflow/underflow events of T6 is not affected by this problem.

Workaround

Do not set bit field BPS2 = $1X_B$ in register GPT12E_T6CON when T5 and/or T6 are configured for counter mode. Use only settings BPS2 = $0X_B$ when T5 and/or T6 are configured for counter mode.

<u>GSC_X.001</u> Clearing of Request Triggers by the GSC

After a request from sources with priority 5..10 ($\overline{\text{ESR0}}$...GPT12E, see table in Chapter “Global State Controller (GSC)” of the current User’s Manual), the following problem will occur:

A trigger for a command request (Wake-up, Clock-off, Suspend Mode) that is enabled in register GSCEN remains pending in the GSC after the arbitration has been finished and the command has been requested. As a consequence, further request triggers with the same or a lower priority will be ignored (14 = lowest priority).

Example

A request from the OCDS to enter Suspend Mode (request source OCDS entry, priority 14) will be ignored if (at any time before) an interrupt request has occurred (request source ITC, priority 9), and the ITC request trigger is enabled in register GSCEN. In this case, modules that are programmed to stop in Suspend Mode (selected in bit field SUMCFG) will continue to run.

Workaround

Disable triggers from request sources that are not used by the application in register GSCEN.
For other sources that shall trigger the GSC, clear and then set again the respective trigger enable bits in register GSCEN each time the GSC logic shall be armed.

INT_X.007 Interrupt using a Local Register Bank during execution of IDLE

During the execution of the IDLE instruction, if an interrupt which uses a local register bank is acknowledged, the CPU may stall, preventing further code execution. Recovery from this condition can only be made through a hardware or watchdog reset.

All of the following conditions must be present for the problem to occur:

- The IDLE instruction is executed while the **global** register bank is selected (bit field BANK = 00_B in register PSW),
- The interrupting routine is using one of the **local** register banks (BANK = 10_B or 11_B), and the local register bank is selected automatically via the bank selection registers BNKSEL0...3, (i.e. the interrupting routine has a priority level $\geq$12),
- The system stack is located in the internal dual-ported RAM (DPRAM, locations $0F600_H$ ... $0FDFF_H$),
- The interrupt is acknowledged during the first 8 clock cycles of the IDLE instruction execution.

Workaround 1

Disable interrupts (either globally, or only interrupts using a local register bank) before execution of IDLE:

```
BCLR IEN       ; Disable interrupts globally
IDLE           ; CPU enters idle mode
BSET IEN       ; After exit from idle mode
               ;   re-enable interrupts
```

If an interrupt request is generated during this sequence, the CPU leaves idle mode and acknowledges the interrupt after BSET IEN.

Workaround 2

Do not use local register banks, use only global register banks.

Workaround 3

Locate the system stack in a memory other than the DPRAM, e.g. in DSRAM.

INT_X.008 HW Trap during Context Switch in Routine using a Local Bank

When a hardware trap occurs under specific conditions in a routine using a local register bank, the CPU may stall, preventing further code execution. Recovery from this condition can only be made through a hardware or watchdog reset.

All of the following conditions must be present for this problem to occur:

- The routine that is interrupted by the hardware trap is using one of the **local** register banks (bit field PSW.BANK = 10_B or 11_B)
- The system stack is located in the internal dual-ported RAM (DPRAM, locations $0F600_H$ ... $0FDFF_H$)
- The hardware trap occurs in the second half (load phase) of a context switch operation triggered by one of the following actions:
  - a) Execution of the IDLE instruction, or
  - b) Execution of an instruction writing to the Context Pointer register CP (untypical case, because this would mean that the routine using one of the local banks modifies the CP contents of a global bank)

Workaround 1

Locate the system stack in a memory other than the DPRAM, e.g. in DSRAM.

Workaround 2

Do not use local register banks, use only global register banks.

Workaround 3

Condition b) (writing to CP while a local register bank context is selected) is not typical for most applications. If the application implementation already eliminates the possibility for condition b), then only a workaround for condition

a) is required.
The workaround for condition a) is to make sure that the IDLE instruction is executed within a code sequence that uses a global register bank context.

INT_X.009 Delayed Interrupt Service of Requests using a Global Bank

Service of an interrupt request using a global register bank is delayed - regardless of its priority - if it would interrupt a routine using one of the local register banks in the following situations:

Case 1:

- The Context Pointer CP is written to (e.g. by POP, MOV, SCXT ... instructions) within a routine that uses one of the **local** register banks (bit field PSW.BANK = 10_B or 11_B),
- Then an interrupt request occurs which is programmed (with GPRSELx = 00_B) to automatically use the **global** bank via the bank selection registers BNKSEL0...3 (i.e. the interrupting routine has a priority level ≥12).

Note that this scenario is regarded as untypical case, because this would mean that the routine using one of the local banks modifies the CP contents of a global bank.

In this case service of the interrupt request is delayed until bit field PSW.BANK becomes 00_B, e.g. by explicitly writing to the PSW, or by an implicit update from the stack when executing the RETI instruction at the end of the routine using the local bank.

Case 2 (see also Figure 1):

- The Context Pointer CP is written to (e.g. by POP, MOV, SCXT ... instructions) within a routine (Task A) that uses a **global** register bank (bit field PSW.BANK = 00_B), i.e. the context for this routine will be modified,
- This context switch procedure (19 cycles) is interrupted by an interrupt request (Task B) which is programmed (with GPRSELx = $1X_B$) to automatically use one of the **local** banks via the bank selection registers BNKSEL0...3 (i.e. the interrupting routine has a priority level ≥12),
- Before the corresponding interrupt service routine is finished, another interrupt request (Task C) occurs which is programmed (with GPRSELx =

00_B) to automatically use the **global** bank via the bank selection registers BNKSEL0...3 (i.e. the interrupting routine has a priority level ≥13)

In this case service of this interrupt request (for Task C) is delayed until bit field PSW.BANK becomes 00_B after executing the RETI instruction at the end of the routine (Task B) using the local bank.

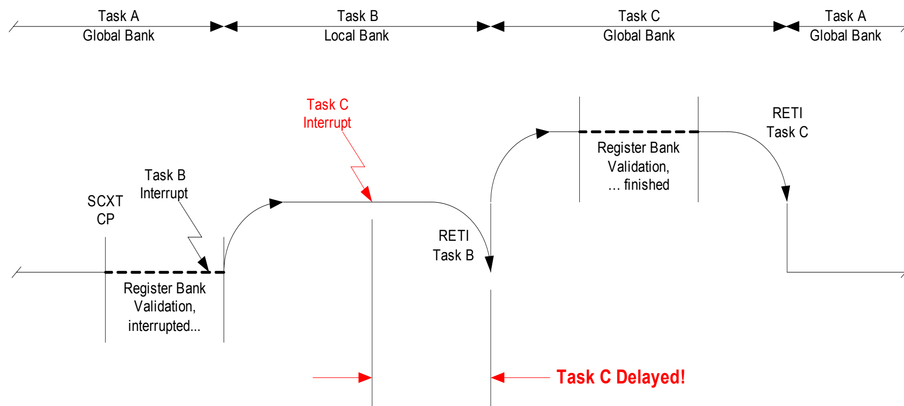

Figure 1 Example for Case 2: Interrupt Service for Task C delayed

Workaround for Case 1

Do not write to the CP register (i.e. modify the context of a global bank) while a local register bank context is selected.

Workaround for Case 2

When using both local and global register banks via the bank selection registers BNKSEL0...3 for interrupts on levels ≥12, ensure that there is no interrupt using a global register bank that has a higher priority than an interrupt using a local register bank.

Example 1:
Local bank interrupts are used on levels 14 and 15, no local bank interrupts on level 12 and 13. In this case, global bank interrupts on level 15 must not be used.

Example 2:
Local bank interrupts are used on level 12. In this case, no global bank interrupts must be used on levels 13, 14, 15.

INT_X.010 HW Traps and Interrupts may get postponed

Under the special conditions described below, a hardware trap (HWTx) and subsequent interrupts, PEC transfers, OCDS service requests (on priority level < 11_H) or class B and class A traps (if HWTx also was class A) may get postponed until the next RETI instruction is executed. If no RETI is executed, these requests may get postponed infinitely.

Both of the following conditions must be fulfilled at the same time when the trigger for the hardware trap HWTx occurs in order to cause the problem:

1. The pipeline is cancelled due to one of the following reasons:
   a) a multiply or divide instruction is followed by a mispredicted conditional (zero-cycle) jump.
   b) a class A hardware trap is triggered quasi-simultaneously with the request for a class B trap (= HWTx), i.e. the trigger for the class A trap arrives before the previously injected TRAP instruction for the class B trap has reached the Execute stage of the pipeline.
   In this case, the class A trap is entered, but when the RETI instruction at the end of the class A trap routine is executed, the pending class B trap (HWTx) is **not** entered, and subsequent interrupts/PECs/class B traps are postponed until the next RETI.
   c) a break is requested by the debugger.
2. The pipeline is stalled in the Execute or Write Back stage due to consecutive writes, or due to a multi-cycle write that is performed to a memory area with wait states (PSRAM, external memory).

Workaround

Disable overrun of pipeline bubbles by setting bit OVRUN (CPUCON2.4) = 0.

MultiCAN_AI.040 Remote frame transmit acceptance filtering error

Correct behaviour:

Assume the MultiCAN message object receives a remote frame that leads to a valid transmit request in the same message object (request of remote answer), then the MultiCAN module prepares for an immediate answer of the remote request. The answer message is arbitrated against the winner of transmit acceptance filtering (without the remote answer) with a respect to the priority class (MOARn.PRI).

Wrong behaviour:

Assume the MultiCAN message object receives a remote frame that leads to a valid transmit request in the same message object (request of remote answer), then the MultiCAN module prepares for an immediate answer of the remote request. The answer message is arbitrated against the winner of transmit acceptance filtering (without the remote answer) with a respect to the CAN arbitration rules and not taking the PRI values into account.

If the remote answer is not sent out immediately, then it is subject to further transmit acceptance filtering runs, which are performed correctly.

Workaround

Set MOFCRn.FRREN=1_B and MOFGPRn.CUR to this message object to disable the immediate remote answering.

MultiCAN_AI.041 Dealloc Last Obj

When the last message object is deallocated from a list, then a false list object error can be indicated.

Workaround

- Ignore the list object error indication that occurs after the deallocation of the last message object.

or

- Avoid deallocating the last message object of a list.

MultiCAN_AI.042 Clear MSGVAL during transmit acceptance filtering

Assume all CAN nodes are idle and no writes to MOCTRn of any other message object are performed. When bit MOCTRn.MSGVAL of a message object with valid transmit request is cleared by software, then MultiCAN may not start transmitting even if there are other message objects with valid request pending in the same list.

Workaround

- Do not clear MOCTRn.MSGVAL of any message object during CAN operation. Use bits MOCTRn.RXEN, MOCTRn.TXEN0 instead to disable/reenable reception and transmission of message objects.

or

- Take a dummy message object, that is not allocated to any CAN node. Whenever a transmit request is cleared, set MOCTRm.TXRQ of the dummy message object thereafter. This retriggers the transmit acceptance filtering process.

MultiCAN_AI.043 Dealloc Previous Obj

Assume two message objects m and n (message object n = MOCTRm.PNEXT, i.e. n is the successor of object m in the list) are allocated. If message m is reallocated to another list or to another position while the transmit or receive acceptance filtering run is performed on the list, then message object n may not be taken into account during this acceptance filtering run. For the frame reception message object n may not receive the message because n is not taken into account for receive acceptance filtering. The message is then received by the second priority message object (in case of any other acceptance filtering match) or is lost when there is no other message object configured for this identifier.For the frame transmission message object n may not be selected for transmission, whereas the second highest priority message object is selected instead (if any). If there is no other message object in the list

with valid transmit request, then no transmission is scheduled in this filtering round. If in addition the CAN bus is idle, then no further transmit acceptance filtering is issued unless another CAN node starts a transfer or one of the bits MSGVAL, TXRQ, TXEN0, TXEN1 is set in the message object control register of any message object.

Workaround

- After reallocating message object m, write the value one to one of the bits MSGVAL, TXRQ, TXEN0, TXEN1 of the message object control register of any message object in order to retrigger transmit acceptance filtering.
- For frame reception, make sure that there is another message object in the list that can receive the message targeted to n in order to avoid data loss (e.g. a message object with an acceptance mask=0_D and PRI=3_D as last object of the list).

MultiCAN_AI.044 RxFIFO Base SDT

If a receive FIFO base object is located in that part of the list, that is used for the FIFO storage container (defined by the top and bottom pointer of this base object) and bit SDT is set in the base object (CUR pointer points to the base object), then MSGVAL of the base object is cleared after storage of a received frame in the base object without taking the setting of MOFGPRn.SEL into account.

Workaround

Take the FIFO base object out of the list segment of the FIFO slave objects, when using Single Data Transfer.

MultiCAN_AI.045 OVIE Unexpected Interrupt

When a gateway source object or a receive FIFO base object with MOFCRn.OVIE set transmits a CAN frame, then after the transmission an unexpected interrupt is generated on the interrupt line as given by MOIPRm.RXINP of the message object referenced by m=MOFGPRn.CUR.

Workaround

Do not transmit any CAN message by receive FIFO base objects or gateway source objects with bit MOFCRn.OVIE set.

MultiCAN_AI.046 Transmit FIFO base Object position

If a message object n is configured as transmit FIFO base object and is located in the list segment that is used for the FIFO storage container (defined by MOFGPRn.BOT and MOFGPRn.TOP) but not at the list position given by MOFGPRn.BOT, then the MultiCAN uses incorrect pointer values for this transmit FIFO.

Workaround

The transmit FIFO works properly when the transmit FIFO base object is either at the bottom position within the list segment of the FIFO (MOFGPRn.BOT=n) or outside of the list segment as described above.

MultiCAN_TC.025 RXUPD behavior

When a CAN frame is stored in a message object, either directly from the CAN node or indirectly via receive FIFO or from a gateway source object, then bit MOCTR.RXUPD is set in the message object before the storage process and is automatically cleared after the storage process.

Problem description

When a standard message object (MOFCR.MMC) receives a CAN frame from a CAN node, then it processes its own RXUPD as described above (correct).

In addition to that, it also sets and clears bit RXUPD in the message object referenced by pointer MOFGPR.CUR (wrong behavior).

Workaround

The “foreign” RXUPD pulse can be avoided by initializing MOFGPR.CUR with the message number of the object itself instead of another object (which would be

message object 0 by default, because MOFGPR.CUR points to message object 0 after reset initialization of MultiCAN).

MultiCAN_TC.026 MultiCAN Timestamp Function

The timestamp functionality does not work correctly.

Workaround

Do not use timestamp.

MultiCAN_TC.027 MultiCAN Tx Filter Data Remote

Message objects of priority class 2 (MOAR.PRI = 2) are transmitted in the order as given by the CAN arbitration rules. This implies that for 2 message objects which have the same CAN identifier, but different DIR bit, the one with DIR = 1 (send data frame) shall be transmitted before the message object with DIR = 0, which sends a remote frame. The transmit filtering logic of the MultiCAN leads to a reverse order, i.e the remote frame is transmitted first. Message objects with different identifiers are handled correctly.

Workaround

None.

MultiCAN_TC.028 SDT behavior

Correct behavior

Standard message objects:

MultiCAN clears bit MOCTR.MSGVAL after the successful reception/transmission of a CAN frame if bit MOFCR.SDT is set.

Transmit Fifo slave object:

MultiCAN clears bit MOCTR.MSGVAL after the successful reception/transmission of a CAN frame if bit MOFCR.SDT is set. After a

transmission, MultiCAN also looks at the respective transmit FIFO base object and clears bit MSGVAL in the base object if bit SDT is set in the base object and pointer MOFGPR.CUR points to MOFGPR.SEL (after the pointer update).

Gateway Destination/Fifo slave object:

MultiCAN clears bit MOCTR.MSGVAL after the storage of a CAN frame into the object (gateway/FIFO action) or after the successful transmission of a CAN frame if bit MOFCR.SDT is set. After a reception, MultiCAN also looks at the respective FIFO base/Gateway source object and clears bit MSGVAL in the base object if bit SDT is set in the base object and pointer MOFGPR.CUR points to MOFGPR.SEL (after the pointer update).

Problem description

Standard message objects:

After the successful transmission/reception of a CAN frame, MultiCAN also looks at message object given by MOFGPR.CUR. If bit SDT is set in the referenced message object, then bit MSGVAL is cleared in the message object CUR is pointing to.

Transmit FIFO slave object:

Same wrong behaviour as for standard message object. As for transmit FIFO slave objects CUR always points to the base object, the whole transmit FIFO is set invalid after the transmission of the first element instead after the base object CUR pointer has reached the predefined SEL limit value.

Gateway Destination/Fifo slave object:

Correct operation of the SDT feature.

Workaround

Standard message object:

Set pointer MOFGPR.CUR to the message number of the object itself.

Transmit FIFO:

Do not set bit MOFCR.SDT in the transmit FIFO base object. Then SDT works correctly with the slaves, but the FIFO deactivation feature by CUR reaching a predefined limit SEL is lost.

MultiCAN_TC.029 Tx FIFO overflow interrupt not generated

Specified behaviour

After the successful transmission of a Tx FIFO element, a Tx overflow interrupt is generated if the FIFO base object fulfils these conditions:

- Bit `MOFCR.OVIE`=1, AND
- `MOFGPR.CUR` becomes equal to `MOFGPR.SEL`

Real behaviour

A Tx FIFO overflow interrupt will not be generated after the transmission of the Tx FIFO base object.

Workaround

If Tx FIFO overflow interrupt needed, take the FIFO base object out of the circular list of the Tx message objects. That is to say, just use the FIFO base object for FIFO control, but not to store a Tx message.

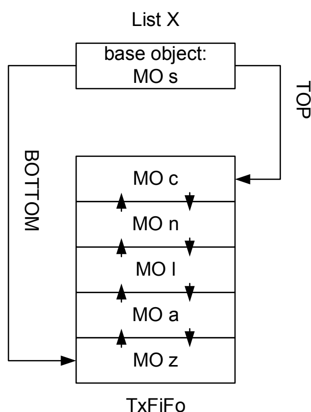

Figure 2 FIFO structure

MultiCAN_TC.030 Wrong transmit order when CAN error at start of CRC transmission

The priority order defined by acceptance filtering, specified in the message objects, define the sequential order in which these messages are sent on the CAN bus. If an error occurs on the CAN bus, the transmissions are delayed due to the destruction of the message on the bus, but the transmission order is kept. However, if a CAN error occurs when starting to transmit the CRC field, the arbitration order for the corresponding CAN node is disturbed, because the faulty message is not retransmitted directly, but after the next transmission of the CAN node.

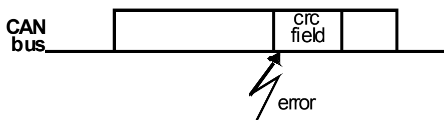

Figure 3

Workaround

None.

MultiCAN_TC.031 List Object Error wrongly triggered

If the first list object in a list belonging to an active CAN node is deallocated from that list position during transmit/receive acceptance filtering (happening during message transfer on the bus), then a "list object" error may occur (`NSRx.LOE`=1_B), which will cause that effectively no acceptance filtering is performed for this message by the affected CAN node.

As a result:

- for the affected CAN node, the CAN message during which the error occurs will not be stored in a message object. This means that although the message is acknowledged on the CAN bus, its content will be ignored.

- the message handling of an ongoing transmission is not disturbed, but the transmission of the subsequent message will be delayed, because transmit acceptance filtering has to be started again.
- message objects with pending transmit request might not be transmitted at all due to failed transmit acceptance filtering.

Workaround

EITHER:

- Avoid deallocation of the first element on active CAN nodes. Dynamic reallocations on message objects behind the first element are allowed, OR
- Avoid list operations on a running node. Only perform list operations, if CAN node is not in use (e.g. when NCRx.INIT=1_B)

MultiCAN_TC.032 MSGVAL wrongly cleared in SDT mode

When Single Data Transfer Mode is enabled (MOFCRn.SDT=1_B), the bit MOCTRn.MSGVAL is cleared after the reception of a CAN frame, no matter if it is a data frame or a remote frame.

In case of a remote frame reception and with MOFCR.FRREN = 0_B, the answer to the remote frame (data frame) is transmitted despite clearing of MOCTRn.MSGVAL (incorrect behaviour). If, however, the answer (data frame) does not win transmit acceptance filtering or fails on the CAN bus, then no further transmission attempt is made due to cleared MSGVAL (correct behaviour).

Workaround

- To avoid a single trial of a remote answer in this case, set MOFCR.FRREN = 1_B and MOFGPR.CUR = this object.

MultiCAN_TC.035 Different bit timing modes

Bit timing modes (NFCRx.CFMOD=10_B) do not conform to the specification.

When the modes 001_B-100_B are set in register `NFCRx.CFSEL`, the actual configured mode and behaviour is different than expected.

Table 9

Bit timing mode (`NFCR.CFSEL`) according to spec	Value to be written to `NFCR.CFSEL` instead	Measurement
001_B	Mode is missing (not implemented) in MultiCAN	Whenever a recessive edge (transition from 0 to 1) is monitored on the receive input the time (measured in clock cycles) between this edge and the most recent dominant edge is stored in CFC.
010_B	011_B	Whenever a dominant edge is received as a result of a transmitted dominant edge the time (clock cycles) between both edges is stored in CFC.
011_B	100_B	Whenever a recessive edge is received as a result of a transmitted recessive edge the time (clock cycles) between both edges is stored in CFC.
100_B	001_B	Whenever a dominant edge that qualifies for synchronization is monitored on the receive input the time (measured in clock cycles) between this edge and the most recent sample point is stored in CFC.

Workaround

None.

MultiCAN_TC.037 Clear MSGVAL

Correct behaviour:

When MSGVAL is cleared for a message object in any list, then this should not affect the other message objects in any way.

Message reception (wrong behaviour):

Assume that a received CAN message is about to be stored in a message object A, which can be a standard message object, FIFO base, FIFO slave, gateway source or gateway destination object.

If during of the storage action the user clears MOCTR.MSGVAL of message object B in any list, then the MultiCAN module may wrongly interpret this temporarily also as a clearing of MSGVAL of message object A. The result of this is that the message is not stored in message object A and is lost. Also no status update is performed on message object A (setting of NEWDAT, MSGLST, RXPND) and no message object receive interrupt is generated. Clearing of MOCTR.MSGVAL of message object B is performed correctly.

Message transmission (wrong behaviour):

Assume that MultiCAN is about to copy the message content of a message object A into the internal transmit buffer of the CAN node for transmission.

If during of the copy action the user clears MOCTR.MSGVAL of message object B in any list, then the MultiCAN module may wrongly interpret this also as a clearing of MSGVAL of message object A. The result of this is that the copy action for message A is not performed, bit NEWDAT is not cleared and no transmission takes place (clearing MOCTR.MSGVAL of message object B is performed correctly). In case of idle CAN bus and the user does not actively set the transmit request of any message object, this may lead to not transmitting any further message object, even if they have a valid transmit request set.

Single data transfer feature:

When the MultiCAN module clears MSGVAL as a result of a single data transfer (MOFCR.SDT = 1 in the message object), then the problem does not occur. The problem only occurs if MSGVAL of a message object is cleared via CPU.

Workaround

Do not clear MOCTR.MSGVAL of any message object during CAN operation. Use bits MOCTR.RXEN, MOCTR.TXEN0 instead to disable/reenable reception and transmission of message objects.

MultiCAN_TC.038 Cancel TXRQ

When the transmit request of a message object that has won transmit acceptance filtering is cancelled (by clearing MSGVAL, TXRQ, TXEN0 or TXEN1), the CAN bus is idle and no writes to MOCTR of any message object are performed, then MultiCAN does not start the transmission even if there are message objects with valid transmit request pending.

Workaround

To avoid that the CAN node ignores the transmission:

- take a dummy message object, that is not allocated to any CAN node. Whenever a transmit request is cleared, set TXRQ of the dummy message object thereafter. This retriggers the transmit acceptance filtering process.

or:

- whenever a transmit request is cleared, set one of the bits TXRQ, TXEN0 or TXEN1, which is already set, again in the message object for which the transmit request is cleared or in any other message object. This retriggers the transmit acceptance filtering process.

OCDS_X.003 Peripheral Debug Mode Settings cleared by Reset

The behavior (run/stop) of the peripheral modules in debug mode is defined in bitfield SUMCFG in the KSCCFG registers. The intended behavior is, that after an application reset has occurred during a debug session, a peripheral re-enters the mode defined for debug mode.

For some peripherals, the debug mode setting in SUMCFG is erroneously set to normal mode upon any reset (instead upon a debug reset only). It remains in this state until SUMCFG is written by software or the debug system.

Some peripherals will **not** re-enter the state defined for debug mode after an application reset:

GPT12, CAPCOM2, and MultiCAN will resume normal operation like after reset, i.e. they are inactive until they are initialized by software.

In case the **RTC** has been running before entry into debug mode, and it was configured in SUMCFG to stop in debug mode, it will resume operation as before entry into debug mode instead.

All other peripheral modules, i.e. ADC0, ADC1, CC60...CC63, USIC0...USIC2**,** will correctly re-enter the state defined for debug mode after an application reset in debug mode.

For **Flash** and **CPU**, bitfield SUMCFG must be configured to normal mode anyway, since they are required for debugging.

Workaround

None.

POWER_X.003 External Supply of Digital Core Supply Voltage V_{DDI}

The digital core supply voltage V_{DDI} must not be supplied externally. The device may only be supplied by the on-chip voltage regulators, which is achieved by connecting pin TRef to V_{DDPB}.

Workaround

None

POWER_X.005 Start-up of DMP_1 at low temperature

The start-up delay of the core power domain DMP_1 upon power-up or a Power-on Reset (via pin $\overline{PORST}$) increases with decreasing temperature. At temperatures below room temperature, the delay may increase by up to ~13 ms compared to temperatures above room temperature.

This increased delay does not occur for transitions between Power States.

Workaround

None.

RESET_X.002 Startup Mode Selection is not Valid in SCU_STSTAT.HWCFG

Reading from SCU_STSTAT.HWCFG-bitfield returns all zeros instead of the information which startup mode has been entered after the last reset.

Workaround

Read the initial value from VECSEG register to evaluate where from the user code is started:

- VECSEG[7:0]=00_H - start from an off-chip memory, external startup mode
- VECSEG[7:0]=$C0_H$ - start from on-chip flash, internal startup mode
- VECSEG[7:0]=$E0_H$ - start from on-chip PSRAM, bootstrap loader mode (UART, CAN or SSC)

RESET_X.003 P2.[2:0] and P10.[12:0] Switch to Input

During a User Reset the port pins P2.[2:0] and P10.[12:0] are switch to input. The user has to check his application to this behaviour.

Workaround

None.

RTC_X.003 Interrupt Generation in Asynchronous Mode

Asynchronous Mode must be selected (bit RTCCM = 1_B in register RTCCLKCON) whenever the system clock is less than 4 times faster than the RTC count input clock ($f_{SYS} < f_{RTC} \times 4$). While in Asynchronous Mode, generation of the RTC interrupt via flag RTCIR in register RTC_IC does not work correctly.

Workaround 1

Select the system clock such that it is at least 4 times faster than the RTC count input clock ($f_{SYS} \geq f_{RTC} \times 4$) and operate the RTC in Synchronous Mode.

Workaround 2

Before switching from Synchronous to Asynchronous Mode, clear the individual interrupt enable bits (CNTxIE, T14IE) in register `RTC_ISNC`. After returning to Synchronous Mode, set the individual interrupt enable bits as required by the application. Then flag RTCIR will get set if at least one of the individual interrupt requests (flags CNTxIR, T14IR) was pending.

USIC_AI.003 TCSRL.SOF and TCSRL.EOF not cleared after a transmission is started

The Start of Frame (SOF) and End of Frame (EOF) bit in the Transmit Control/Status Register (`TCSRL`) will not be cleared by hardware when the data is transfered from the transmit buffers (`TBUFx`) to the transmit shift register, i.e. the transmission of a new word starts.

Workaround

Clear `TCSRL.SOF` and `TCSRL.EOF` by software.

USIC_AI.004 Receive shifter baudrate limitation

If the frame length of `SCTRH.FLE` does not match the frame length of the master, then the baudrate of the SSC slave receiver is limited to $f_{sys}/2$ instead of f_{sys}.

Workaround

None.

6.2 Deviations from Electrical and Timing Specification

- none -

6.3 Application Hints

CAPCOM12_X.H001 Enabling Single Event Operation

The single event operation mode of the CAPCOM1/2 unit eliminates the need for software to react after the first compare match when only one event is required within a certain time frame. The enable bit SEEy for a channel CCy is cleared by hardware after the compare event, thus disabling further events for this channel.

One Channel in Single Event Operation

As the Single Event Enable registers CC1_SEE, CC2_SEE are not located in the bit-addressable SFR address range, they can only be modified by instructions operating on data type WORD. This is no problem when only one channel of a CAPCOM unit is used in single event mode.

Two or more Channels in Single Event Operation

When two or more channels of a CAPCOM unit are independently operating in single event mode, usually an OR instruction is used to enable one or more compare events in register CCx_SEE. In this case, the timing relation of the channels must be considered, otherwise the following typical problem may occur:

- In the Memory stage, software reads register CCx_SEE with bit SEEy = 1_B (event for channel CCy has not yet occurred)
- Meanwhile, event for CCy occurs, and bit SEEy is cleared to 0_B by hardware
- In the Write-Back stage, software writes CCx_SEE with bit SEEx = 1_B (intended event for CCx enabled) **and** bit SEEy = 1_B

In this case, another unintended event for channel CCy is enabled.

To avoid this effect, one of the following solutions - depending on the characteristics of the application - is recommended to enable further compare events for CAPCOM channels concurrently operating in single event mode:

- Modify register CCx_SEE only when it is ensured that no compare event in single event mode can occur, i.e. when CCx_SEE = 0x0000, or

- Modify register CCx_SEE only when it is ensured that there is a sufficient time distance to the events of all channels operating in single event mode, such that none of the bits in CCx_SEE can change in the meantime, or
- Use single event operation for one channel only (i.e. only one bit SEMz may be = 1_B), and/or
- Use one of the standard compare modes, and emulate single event operation for a channel CCs by disabling further compare events in bit field MODs (in register CCx_My) in the corresponding interrupt service routine. Writing to register CCx_My is uncritical, as this register is not modified by hardware.

CC6_X.H001 Modifications of Bit MODEN in Register CCU6x_KSCFG

For each module, setting bit MODEN = 0 immediately switches off the module clock. Care must be taken that the module clock is only switched off when the module is in a defined state (e.g. stop mode) in order to avoid undesired effects in an application.

In addition, for a CCU6 module in particular, if bit MODEN is changed to 0 while the internal functional blocks have not reached their defined stop conditions, and later MODEN is set to 1 and the mode is not set to run mode, this leads to a lock situation where the module clock is not switched on again.

INT_X.H002 Increased Latency for Hardware Traps

When a condition for a HW trap occurs (i.e. one of the bits in register TFR is set to 1_B), the next valid instruction that reaches the Memory stage is replaced with the corresponding TRAP instruction. In some special situations described in the following, a valid instruction may not immediately be available at the Memory stage, resulting in an increased delay in the reaction to the trap request:

1. When the CPU is in break mode, e.g. single-stepping over such instructions as SBRK or BSET TFR.x (where x = one of the trap flags in register TFR)

will have no (immediate) effect until the next instruction enters the Memory stage of the pipeline (i.e. until a further single-step is performed).

2. When the pipeline is running empty due to (mispredicted) branches and a relatively slow program memory (with many wait states), servicing of the trap is delayed by the time for the next access to this program memory, even if vector table and trap handler are located in a faster memory. However, the situation when the pipeline/prefetcher are completely empty is quite rare due to the advanced prefetch mechanism of the C166S V2 core.

JTAG_X.H001 JTAG Pin Routing

In the current device, the pins connected to the JTAG interface can be selected by software (write to register DBGPRR). After a reset, the JTAG interface is connected to position A (see **Table 10**). If connected to these pins, the debugger will work without any restrictions.

Table 10 JTAG Position A

Pin LQFP-100	Pin LQFP-144	Symbol	Signal
23	34	P5.2	TDI_A
6	8	P7.0	TDO_A
57	82	P2.9	TCK_A
28	39	P5.4	TMS_A
5	6	$\overline{\text{TRST}}$	$\overline{\text{TRST}}$

To use other pins for the JTAG interface, the following sequence of steps must be executed:

- $\overline{\text{TRST}}$ must be high at the rising edge of $\overline{\text{PORST}}$. Usually debuggers provide that.
- Debuggers must be set to do a so called `hot attach`. This is connecting the microcontroller without executing a reset.
- Execute a write to DBGPRR register with the desired selection of pins to be used with one of the first instructions out of Flash.

- TCK_A must be stable on a valid low or high level until the change of the JTAG interface was executed.
- A Halt after reset can be achieved by a program loop that is left by control through the debugger.

Verify the correct operation of the sequence within the actual application.

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:

`MCAN_KSCCFG.MODEN` = 0 and then `MCAN_KSCCFG.MODEN` = 1

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_TC.H002 Double Synchronization of receive input

The MultiCAN module has a double synchronization stage on the CAN receive inputs. This double synchronization, delays the receive data by 2 module clock cycles. If the MultiCAN is operating at a low module clock frequencies and high CAN baudrate, this delay may become significant and has to be taken into

account when calculating the overall physical delay on the CAN bus (transceiver delay...).

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.

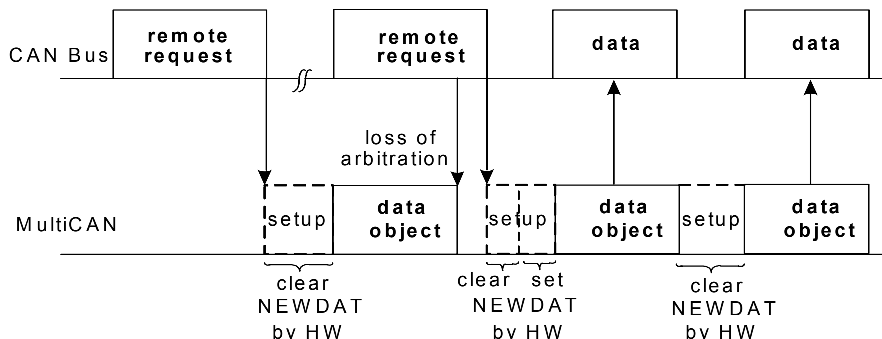

Figure 4 Loss of Arbitration

OCDS_X.H002 Suspend Mode Behavior for MultiCAN

The MultiCAN module basically provides two mechanisms to stop participation in CAN bus communication when a suspend request is issued by the OCDS:

Suspend operation of selected CAN nodes

The sensitivity to a suspend request can be individually enabled/disabled for each CAN node via bit SUSEN in its associated Node Control Register NCRx. With SUSEN = 1_B, upon a suspend request bit INIT is internally forced to 1_B to disable the CAN node as soon as it becomes BUS IDLE or BUS OFF. This way, a CAN node correctly finishes a running CAN frame, but does not start a new one. The network is not blocked due to the suspend state of one communication partner. All CAN registers can be read and written in this state since the module clock is not switched off.

Notes

1. *Depending on CAN activity and bus speed, the contents of some MultiCAN registers may still change if the debugger immediately reads them before the CAN node has reached BUS IDLE or BUS OFF state, i.e. before bit SUSACK = 1_B.*
2. *Bit field SUMCFG in register KSCCFG for the MultiCAN module must be set to 00_B to avoid an immediate stop (see below).*

Immediately stop operation of MultiCAN module

When bit field SUMCFG in register KSCCFG for the MultiCAN module is set to $1X_B$, the clock for the MultiCAN module is switched off as soon as the suspend request from the OCDS becomes active. As a consequence, the module immediately stops all CAN activity (even within a running frame) and sets all transmit outputs to 1_B (recessive state). In this state, write accesses to the module in general, and read accesses to the CAN RAM and most of the MultiCAN registers are no longer supported. A normal continuation when the suspend mode is left may not always be possible and may require a reset (e.g. depending on error counters).

RESET_X.H002 Oscillator fail detection active by default

After a power-on reset in domain DMP_M, the logic of the oscillator watchdog is sensitive to oscillator fail events. This means that if e.g. the amplitude of the starting crystal temporarily falls below the input hysteresis, the OWD will detect an oscillator fail event. As a consequence, the VCO will be automatically disconnected from its input clock.

If an oscillator fail event is detected during the internal reset phase (i.e. before the first instruction of the application is executed), the start-up phase may be elongated significantly or the device will enter and remain in an error mode. Note that this effect will not occur when a crystal is connected to pin XTAL1, since OSC_HP is disabled after reset, such that a crystal connected to XTAL1 is not starting up until enabled by user software in register HPOSCCON.

In case no crystal is connected to pin XTAL1, this pin should be pulled to V_{SS} during the start-up phase.

If an oscillator fail event is detected after the internal reset phase has finished and the system clock is generated using the PLL in normal mode (as configured by the internal start-up mechanism), the system frequency will fall back to f_{VCO_base} / K2.

Software should check the clock configuration after start-up to make sure it matches the timing requirements of the application, i.e. it should check whether a clock failure was detected during start-up.

RESET_X.H003 How to Trigger a PORST after an Internal Failure

There is no internal User Reset which restores the complete device including the power system like a Power-On Reset. In some application it is possible to connected ESR1 or ESR2 with the PORST Pin and set the used ESR Pin as Reset output. With this a WDT or Software Reset can trigger a Power-On Reset.

RTC_X.H003 Changing the RTC Configuration

The count input clock f_{RTC} for the Real Time Clock module (RTC) can be selected via bit field RTCCLKSEL in register RTCCLKCON. Whenever the system clock is less than 4 times faster than the RTC count input clock ($f_{SYS} < f_{RTC} \times 4$), Asynchronous Mode must be selected (bit RTCCM = 1_B in register RTCCLKCON).

To assure data consistency in the count registers T14, RTCL, RTCH, the RTC module must be temporarily switched off by setting bit MODEN = 0_B in register RTC_KSCCFG before register RTCCLKCON is modified, i.e. whenever

- changing the operating mode (Synchronous/Asynchronous) Mode in bit RTCCM, or
- changing the RTC count source in bit field RTCCLKSEL.

In case power domain DMP_1 is switched off, it is not required to switch the RTC to Asynchronous Mode, since it will receive a reset in any case.

StartUp_X.H002 FCONCS0..FCONCS4 Registers are Always Configured in External Start-Up Mode

The Start-Up procedure in External Start Mode writes all the FCONCSx (x=0..4) registers, independently on the number of CS-outputs selected.This has no effect for the user because for unused CS lines the Start-Up procedure configures only the external bus type into the registers but does not enable the output(s) for CS-functionality and they are free available.

USIC_AI.H001 FIFO RAM Parity Error Handling

A false RAM parity error may be signalled by the USIC module, which may optionally lead to a trap request (if enabled) for the USIC RAM, under the following conditions:

- a receive FIFO buffer is configured for the USIC module, and
- after the last power-up, less data elements than configured in bit field SIZE have been received in the FIFO buffer, and
- the last data element is read from the receiver buffer output register OUTRL (i.e. the buffer is empty after this read access).

Once the number of received data elements is greater than or equal to the receive buffer size configured in bit field SIZE, the effect described above can no longer occur.

To avoid false parity errors, it is recommended to initialize the USIC RAM before using the receive buffer FIFO. This can be achieved by configuring a 64-entry transmit FIFO and writing 64 times the value 0x0 to the FIFO input register IN00 to fill the whole FIFO RAM with 0x0.

WDT_X.H001 Watchdog Timer Reset Functionality not enabled by Default

After power-on reset for DMP_M, the default value for bit field RSTCON1.WDT = 00_B, i.e. no reset will be generated when the Watchdog Timer overflows. A Watchdog Timer overflow will only result in a reset if bit field RSTCON1.WDT is initialized differently from 00_B by the application software. E.g. setting RSTCON1.WDT = 11_B will result in an Application Reset when the watchdog timer overflows. Note that register RSTCON1 is protected by the register security mechanism after execution of the EINIT instruction.

w w w . i n f i n e o n . c o m

Published by Infineon Technologies AG