

Device **XC886CM-8FF**
Marking/Step **EES AA**
Package **PG-TQFP-48**

This Errata Sheet describes the deviations from the current user documentation.

The module oriented classification and numbering system uses an ascending sequence over several derivatives, including already solved deviations. So gaps inside this enumeration can occur.

Current Documentation

- XC886/888CLM User's Manual V0.1, Jan 2006
- XC886/888CLM Data Sheet V0.1, Feb 2006

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.

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

Contents

Section	Page
History List/Change Summary	2
Functional Deviations	4
Deviations from Electrical- and Timing Specification	18
Application Hints	19

1 History List/Change Summary

(no previous device step)

Table 1 Functional Deviations

Functional Deviation	Short Description	Fixed in Step	Change
ADC_XC8.003	Limitation during power-down mode		
BROM_XC8.006	IRAM data corrupted after hardware reset		
BROM_XC8.008	MCAN Bootstrap Loader functionality not available.		
BROM_XC8.009	LIN BSL Mode 4 (Flash Mass Erase) does not work correctly.		
INT_XC8.004	Unable to Detect New Interrupt Request if Any One of Timer 2/Timer 21/UART1 Interrupt Flags Is Not Cleared (Unexpectedly)		
INT_XC8.005	Write to IRCON0 Blocks Interrupt Request of External Interrupt 0,1		
MDU_XC8.001	Overflow during Left Shift may lead to a wrong shift result		
OCDS_XC8.008	Watchdog Timer behavior during Debug with Suspend		
OCDS_XC8.009	Break while CPU is in NMI service mode		
PIN_XC8.004	Reset values of PUDSEL and PUDEN registers in Port 4 and Port 5 are wrong.		
WDT_XC8.001	WDT is not default suspended in Monitor Mode		
WDT_XC8.002	WDT should not be refreshed during Prewarning period		
OSC_XC8.002	OSC_CON register is not reset following a wake-up reset		
PLL_XC8.001	PLL N-Divider is reset to default value after a WDT reset		

Table 1 Functional Deviations

Functional Deviation	Short Description	Fixed in Step	Change
MultiCAN_XC8.001	RXUPD behavior		
MultiCAN_XC8.002	MultiCAN Timestamp Function		
MultiCAN_XC8.003	MultiCAN Tx Filter Data Remote		
MultiCAN_XC8.004	SDT behavior		
MultiCAN_XC8.005	Tx FIFO overflow interrupt not generated		
MultiCAN_XC8.006	Wrong transmit order when CAN error at start of CRC transmission		
MultiCAN_XC8.007	List Object Error wrongly triggered		
MultiCAN_XC8.008	MSGVAL wrongly cleared in SDT mode		
MultiCAN_XC8.009	Different bit timing modes		

Table 2 Application Hints

Application Hints	Short Description	Fixed in Step	Change
BROM_XC8.H001	SYSCON0.RMAP handling in ISR		
FLASH_XC8.H001	Limitation to Flash operations under PLL Loss-of-Lock condition		
INT_XC8.H003	Interrupt Flags of External Interrupt 0 and 1		
INT_XC8.H004	NMI Interrupt Request With No NMI Flag Set		
INT_XC8.H005	Not all Flags are qualified for clearing Pending Interrupt Request		
OCDS_XC8.H001	Write to OCDS registers can disturb the Debug Function		
OCDS_XC8.H002	Any NMI request is lost on Debug entry and during Debug		

2 Functional Deviations

ADC_XC8.003 Limitation during power-down mode

When the device enters power-down mode, the ADC power-down control level shifter will experience an overstress condition. This overstress condition may cause the ADC power-down control level shifter to fail, resulting the ADC module in entering power-down mode permanently. Therefore, it is advisable to avoid using power-down mode.

Workaround:

None.

BROM_XC8.006 IRAM data corrupted after hardware reset

After the hardware reset, boot up via User Mode (MBC=1) affects certain IRAM data.

The affected IRAM address range:

(1) 00_H - 07_H

(2) 61_H - 64_H

(3) 80_H - BF_H

Workaround:

None

BROM_XC8.008 MCAN Bootstrap Loader functionality not available.

The functionality of the MCAN Bootstrap loader is not implemented.

Workaround:

None

BROM_XC8.009 LIN BSL Mode 4 (Flash Mass Erase) does not work correctly.

For full erase of the flash memories, user selects Mode 4 (Mass Erase). Completion of this operation is indicated by an acknowledgement code of 0x55.

In the current implementation, though acknowledgement code is received, the Flash memories are not erased.

Workaround:

To ensure a full erase of the Flash memories, users need to execute an erase of all the P-Flash and D-Flash banks separately.

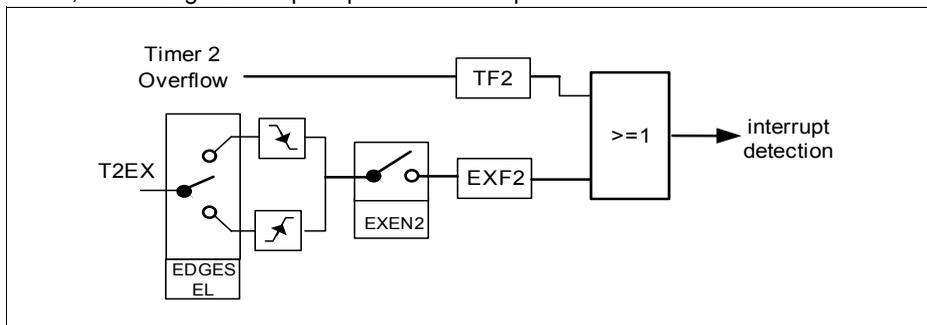
INT_XC8.004 Unable to Detect New Interrupt Request if Any One of Timer 2/Timer 21/UART1 Interrupt Flags Is Not Cleared (Unexpectedly)

The situation leading to unable to detect new interrupt request is similar for

1. Timer 2 (TF2 and EXF2),
2. Timer 21 (TF21 and EXF21), and
3. UART1 (RI and TI)

independently, as described in the following for Timer 2.

As illustrated in the simplified figure, the Timer 2 interrupt flags TF2 and EXF2 are combined as one interrupt request output. These flags are located within the Timer 2 kernel, with a single interrupt request line as output from the kernel.



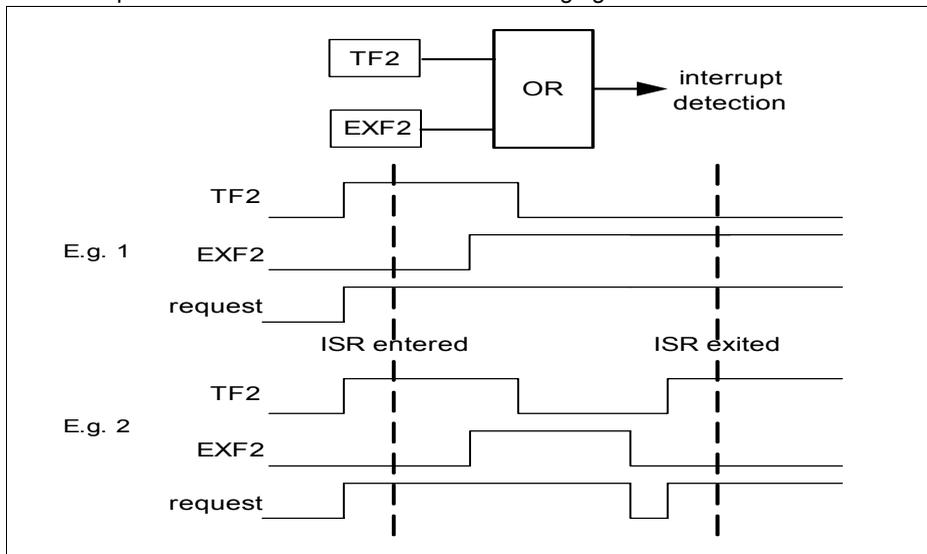
Being of interrupt structure 2, the interrupt request of Timer 2 is detected on the rising edge of a positive pulse.

Functional Deviations

The problem is that it may occur at some point in the application, that any new Timer 2 interrupt request can no longer be detected after a return-from-interrupt (reti) due to service of earlier event(s). This happens when the following conditions are true:

1. Timer 2 events are serviced by interrupt (i.e. flags are checked and cleared only in the interrupt routine ISR),
2. Either TF2 or EXF2 is set at any one time throughout the ISR (even while the other is cleared) such that at least one of the flags is still set after reti, causing the Timer 2 request line, which is an OR-function of the two flags TF2 and EXF2, to remain set throughout the ISR and after reti.

Two example scenerios are illustrated in the following figure:



This means, any future Timer 2 TF2 or EXF2 event is not able to cause a rising edge and therefore not able to trigger an interrupt request to the core – as if the Timer 2 interrupts had been disabled, until both EXF2 and TF2 flags are cleared at some time. The clearing of flags would have to be done by user’s code additionally outside of the Timer 2 interrupt routine, which is however normally not feasible with an interrupt service scheme.

Workaround:

Timer 2 and Timer 21

There is no problem if EXF2 is not enabled by setting bit EXEN2 = 0 i.e. external events are disabled, or if bit DCEN = 1 in auto-reload mode.

Otherwise if EXF2 is enabled, while TF2 is always enabled when Timer 2 is running, there are two suggested workaround:

Functional Deviations

1. If other events of Timer 2 interrupt node (ET2) need not be enabled for interrupt, disable this interrupt node and use software polling of the flags instead.
2. Before return from interrupt, check again if TF2 or EXF2 is (still) set (due to new request since the last check). If so, jump and execute the ISR routine from start. Exit only when all flags are checked to be cleared. However, dummy interrupt of the node may occur after return from interrupt, and should be ignored. Another drawback is if Timer 2 events are occurring at high rate, the CPU may be 'stuck' in the service routine of the Timer 2 interrupt for a long time.

<entry point of interrupt node service routine>

.....

Start:

check flag TF2

.....

clear flag TF2

.....

check flag EXF2

.....

clear flag EXF2

.....

Finish:

if TF2 or EXF2 is set, jump to Start

reti (return from interrupt)

UART1

The two suggested workaround for Timer 2 above applies. In this case, replace with UART1 interrupt node and the UART1 flags: RI and TI.

INT_XC8.005 Write to IRCON0 Blocks Interrupt Request of External Interrupt 0,1

Any write (read-modify-write or direct MOV) to the SFR IRCON0 will block an incoming interrupt request from external interrupt 0 or 1, even though the respective flag (EXINT0 or EXINT1) is set.

Workaround:

After any write to the IRCON0, check (read IRCON0) the flags EXINT0 and EXINT1. If any flag is set, run the service routine; otherwise proceed.

In case of enabled for interrupt, the service routine should be duplicated: one copy as the interrupt service routine (with *reti* executed); another copy in main code memory for software call (with *ret* executed).

MDU_XC8.001 Overflow during Left Shift may lead to a wrong shift result

There is no overflow or saturation defined for MDU shift operations. However, in the implementation of left shift, when a '1' is shifted out of the 32-bit range for the first time, the error flag MDUSTAT.IERR will be wrongly set to indicate an overflow situation. Additionally, the shift operand becomes saturated, leading to a wrong shift result.

An example is shown in the following table where the value 40_H is left shifted twice.

Operation	Operand	Result	Status
First left shift	0100 0000 _B	1000 0000 _B	Correct
Second left shift	1000 0000 _B	1111 1111 _B	Wrong

Workaround:

The user should limit the shift count during left shift operations so that no '1' is shifted out of the 32-bit range.

OCDS_XC8.008 Watchdog Timer behavior during Debug with Suspend

The WDT may be enabled for suspend (stops counting) in debug mode.

In this suspended state, when the WDT is refreshed by writing WDTCON.WDTRS, the timer base counter which provide the clock to WDT is not refreshed to zero.

The effect is that on exiting Monitor Mode in user mode, the WDT may count a little shorter to overflow. This shortened time is less than one WDT count.

Workaround:

None. This WDT behavior occurs only during debug where WDT is enabled for suspend.

OCDS XC8.009 Break while CPU is in NMI service mode

While the CPU is in NMI service mode, the debug function has the following limitations:

1. External break via JTAG or MBC pin occurring while in NMI service mode, results in debug mode entry being postponed until NMI mode is ended.
2. Hardware or software breakpoints inside the NMI service routine will trigger the debug function. However, it is not possible to return to and execute user code (run or step) at the point of break as the debug cannot be exit.

Workaround:

The workaround for the respective scenerios are

1. None. In fact, since NMI is usually a critical situation, any asynchronous break event postponed till after return from NMI mode has insignificant effects.
2. Do not set breakpoints or put the TRAP instruction inside the NMI service routine.

PIN XC8.004 Reset values of PUDSEL and PUDEN registers in Port 4 and Port 5 are wrong.

The implemented reset values of PUDSEL and PUDEN registers in Port 4 and Port 5 differ from those that are defined in the specification. This leads to wrong pull-up/pull-down status at some of the Port 4 and Port 5 pins after reset.

Register Name	Reset Value	
	Implementation	Specification
P4_PUDSEL	BF _H	FF _H
P4_PUDEN	40 _H	04 _H
P5_PUDSEL	BF _H	FF _H
P5_PUDEN	40 _H	FF _H

Note: Pins P4.2, P4.4-P4.7 and P5.0-P5.7 are not available in TQFP-48 package.

Workaround:

All four registers should be re-initialized to the reset value defined in the specification.

WDT_XC8.001 WDT is not default suspended in Monitor Mode

The WDT is not default suspended when OCDS enters monitor mode. This requires the WDT to be always refreshed to prevent a WDT reset from taking place.

Workaround:

The user have to disable the WDT first before entering monitor mode, and enable again on exiting monitor mode.

WDT_XC8.002 WDT should not be refreshed during Prewarning period

If the WDT is not serviced before the timer overflows, a WDT NMI request is asserted and Prewarning is entered. During the Prewarning period, which last for 30_H counts, refreshing of the Watchdog Timer should not be possible and the microcontroller is expected to reset at the end of this period.

However in the current implementation, the WDT reset can be prevented by constantly writing to WDTCON.WDTRS during Prewarning.

Workaround:

The user should not refresh the WDT upon entering the Prewarning period.

OSC_XC8.002 OSC_CON register is not reset following a wake-up reset

On a wake-up from power-down with reset, the register OSC_CON is not reset to its default value as specified. If an external oscillator is used, it will remain as the selected oscillator source on wake-up reset. On the other hand, the PLL settings including the NDIV value will be reset to the default state.

Depending on the external oscillator's frequency (f_{OSC}), there are three possible scenarios:

1. If $f_{OSC} > 10$ MHz, the system might freeze due to a system frequency that is too high.

Functional Deviations

2. If $7.5 \text{ MHz} \leq f_{\text{OSC}} \leq 10 \text{ MHz}$, the system starts executing the user code correctly, even though it might run at a reduced frequency.
3. If $f_{\text{OSC}} < 7.5 \text{ MHz}$, the system might freeze due to the inability of the PLL to lock.

Workaround:

The wake-up from power-down with reset feature should not be used with an external oscillator outside the range of 7.5 to 10MHz (scenarios 1 and 3). There is no impact on the user if the system is using the on-chip oscillator.

PLL_XC8.001 PLL N-Divider is reset to default value after a WDT reset

The user configured PLL N-divider value should be retained following a WDT reset. However in the current implementation, it is reset to its default value instead.

A problem arises when an external oscillator is used to generate the system clock. Depending on the external oscillator's frequency (f_{OSC}), there are three possible scenarios:

1. If $f_{\text{OSC}} > 10 \text{ MHz}$, the system might freeze due to a system frequency that is too high.
2. If $7.5 \text{ MHz} \leq f_{\text{OSC}} \leq 10 \text{ MHz}$, the system starts executing the user code correctly, even though it might run at a reduced frequency.
3. If $f_{\text{OSC}} < 7.5 \text{ MHz}$, the system might freeze due to the inability of the PLL to lock.

Workaround:

The WDT should not be used with an external oscillator outside the range of 7.5 to 10MHz (scenarios 1 and 3). If the external oscillator falls within 7.5 to 10MHz (scenario 2), the PLL N-divider must always be re-initialized in the user code with the required value after a WDT reset. There is no impact on the user if the system is using the on-chip oscillator.

MultiCAN_XC8.001 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 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_XC8.002 MultiCAN Timestamp Function

The timestamp functionality does not work correctly.

Workaround:

Do not use timestamp.

MultiCAN_XC8.003 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_XC8.004 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 reaches 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_XC8.005 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:

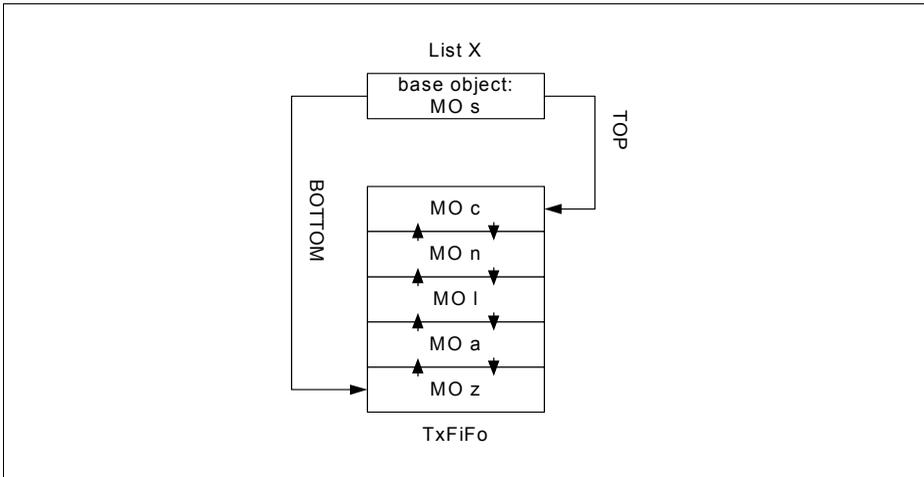
- Bit MOFCR.OVIE=1, AND
- MOFGPR.CUR becomes equal to MOFGPR.SEL

Real behaviour

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

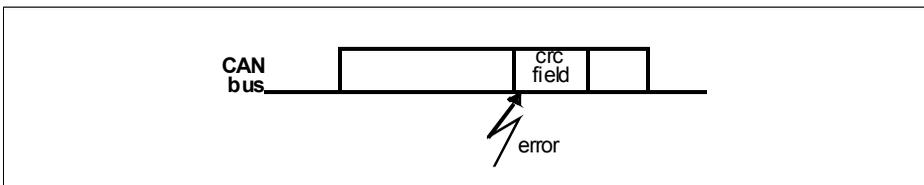
Workaround

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.



MultiCAN_XC8.006 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.



Workaround

None.

MultiCAN_XC8.007 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

1. Avoid deallocation of the first element on active CAN nodes. Dynamic reallocations on message objects behind the first element are allowed.

Or

2. 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_XC8.008 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_XC8.009 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.

Bit timing mode (NFCRx.CFSEL) according to spec	Value to be written to NFCRx.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.

Deviations from Electrical- and Timing Specification

Bit timing mode (NFCR.CFSEL) according to spec	Value to be written to NFCR.CFSEL instead	Measurement
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.

3 **Deviations from Electrical- and Timing Specification**

No deviation from Electrical- and Timing Specification are known for this step.

4 Application Hints

BROM_XC8.H001 SYSCON0.RMAP handling in ISR

The ISR has to handle SYSCON0.RMAP correctly when Flash user routines provided in the Boot ROM are used together with the interrupt system. Any ISR with the possibility of interrupting these user routines has to do the following in the interrupt routine:

- save the value of the RMAP bit at the beginning
- restore the value before the exit

This is to prevent access of the wrong address map upon return to the Flash user routine since the RMAP bit may be changed within the interrupt routine. The critical point is when

Flash user routines sets RMAP to '1' and the interrupt occurs that needs RMAP at '0' in the ISR.

Please note that NMI is an interrupt as well.

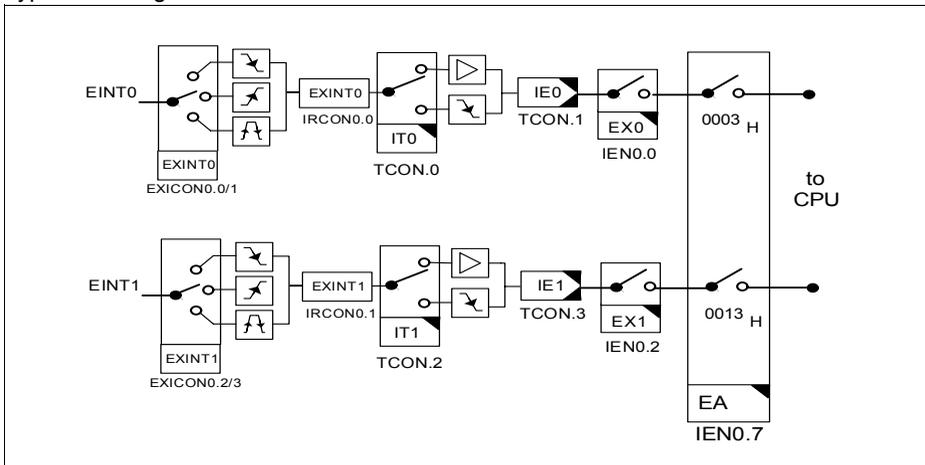
FLASH_XC8.H001 Limitation to Flash operations under PLL Loss-of-Lock condition

When the PLL Loss-of-Lock occurs, the PLL will run at its VCO base frequency. Under this condition, reading from Flash operation is possible. However, any ongoing Flash program or erase operation may fail and therefore should be aborted.

Before the recovery of Loss-of-Lock, no program or erase operation shall be started on the Flash.

INT_XC8.H003 Interrupt Flags of External Interrupt 0 and 1

External interrupt 0 and 1 may individually be selected via respective bits (EXINTx) in EXICON0 register, to request interrupt on falling edge, rising edge, both edges or to bypass the edge detection.



Edge detection is done in the system unit. If enabled, an active event will set the EXINTx flag and correspondingly set the IEx flag in TCON. It should be noted that after any external interrupt x event, flag EXINTx must be cleared. In case of falling edge as active event, this allows any future active event to be able to set the flag IEx as interrupt request. In case of low level as active event, this prevents unintended recurring triggering of interrupt request.

In case of bypass edge detection, the input is connected directly to the core (bypass the system unit). The flag IEx in TCON will be set on the selected falling edge or low level (at least 2 clock cycles) event. In case of falling edge selected as active event by TCON.ITx, the flag EXINTx is set in addition to the flag IEx.

Besides the above notes, the following should be noted on the behavior regarding setting and clearing of the external interrupt x (x = 0 or 1) flags, applicable to both edge and bypass edge detection modes:

Setting of External Interrupt x Flags

1. The flag TCON.IEx will be set in all modes selectable via EXICON0 register.
2. Flag IRCON0.EXINTx will be set in all modes as long as an active edge is detected; flag EXINTx will not be set for low level as active event.

Clearing of External Interrupt x Flags

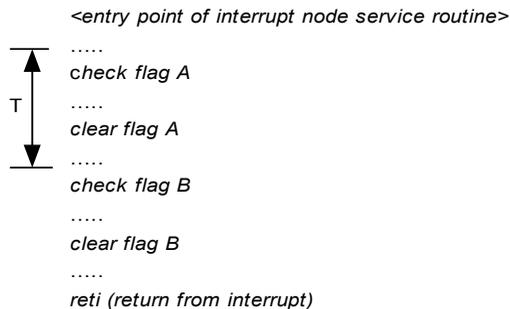
1. Flag IEx is cleared automatically by hardware when the interrupt is being vectored to.

2. Flag EXINTx has to be cleared by software.
3. Clearing one external interrupt x flag will not clear the other. Especially, clearing flag EXINTx will not clear the flag IEx. Being of interrupt structure 1, the flag IEx is the request polled by the CPU for interrupt servicing. Therefore user has to take care to clear the flag IEx before switching from SW polling method to enabling the external interrupt x node, to prevent potential dummy interrupt request.
4. Always clear both EXINTx and IEx flags before (if) changing the trigger select in EXICON0 register.

INT_XC8.H004 NMI Interrupt Request With No NMI Flag Set

It might occur in the application, that sometimes NMI interrupt requests are serviced, but no active NMI interrupt flags are found.

Consider the following NMI interrupt service routine pseudo-code, and scenerio where a NMI interrupt source A event leads to interrupt request of the NMI node and CPU vectors to the NMI interrupt routine. Meanwhile, NMI interrupt source B and its flag becomes active any time in the duration indicated by T:



In this case, NMI flag B will be cleared as a standard procedure by the NMI interrupt routine in the current service. However, the pending interrupt request for the NMI node remains activated after RETI, as it is only cleared by hardware when CPU acknowledge the NMI interrupt and vectors to the NMI service routine.

This leads to following servicing of the NMI interrupt node again, but potentially no active NMI flag is found, i.e. dummy NMI interrupt service. The point to note is that the NMI interrupt source B is not lost, as it was actually serviced in the current service of the NMI interrupt node.

The recommendation is to ignore these dummy NMI interrupt vectoring.

INT_XC8.H005 Not all Flags are qualified for clearing Pending Interrupt Request

For interrupts of structure 2, qualified event (status) flags are used for clearing any pending interrupt request to the core. An event flag is qualified as long as the event is enabled for interrupt. By this means, as long as all qualified flags of the node are cleared, any still active pending interrupt request to the core will be cleared by hardware.

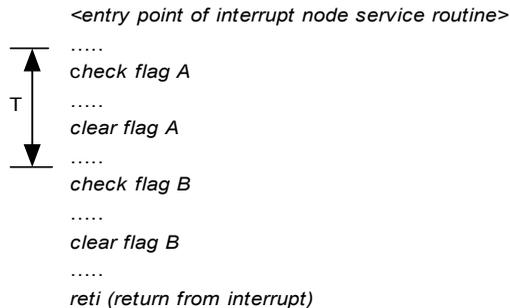
However with existing implementation, not all event flags are qualified for clearing the pending interrupt request to core. These flags are listed in the following table, corresponding to the interrupt node the flag belongs to.

Interrupt Node	Vector Address	Assignment	Unqualified Flags Belonging to Node
NMI	0073 _H	Watchdog Timer NMI	NMIWDT
		PLL NMI	NMIPLL
		Flash NMI	NMIFLASH
		OCDS NMI	NMIOCDS
		VDDC Prewarning NMI	NMIVDD
		VDDP Prewarning NMI	NMIVDDP
		Flash ECC NMI	NMIECC
XINTR5	002B _H	LIN	EOFSYN, ERRSYN
XINTR8	0043 _H	External Interrupt 2	EXINT2
		CORDIC	EOC
		MDU	IRDY, IERR
XINTR9	004B _H	External Interrupt 3	EXINT3
		External Interrupt 4	EXINT4
		External Interrupt 5	EXINT5
		External Interrupt 6	EXINT6

Note: Some events e.g. TF2, EXF2 of Timer 2, Timer21; NDOV of UART; NDOV, RI and TI of UART1 do not have separate interrupt enable apart from its interrupt node enable. These event flags are therefore always qualified, if the interrupt node is enabled.

Consider the case where an enabled interrupt node is shared by more than 2 events, and where at least two of the events (A and B) are enabled for interrupt while at least one event (C) is not enabled for interrupt. In this case, flag C is one of the flags listed in above table.

While the interrupt routine is already running due to event A having occurred, event B and C occurs any time in the duration indicated by T:



This sets the pending interrupt request while flag B is set.

Although event B is serviced and flag B is cleared in the following service routine, on return from interrupt, the pending interrupt request remains activated. This is because event C is not enabled for interrupt (and therefore flag C is neither checked nor cleared in the interrupt routine) while flag C is not qualified.

This leads to following servicing of the interrupt node again, but potentially no active flag is found, i.e. dummy interrupt service.

To prevent such dummy interrupt vectoring on the above listed interrupt nodes, do not use mixed interrupt servicing and polling scheme, i.e, enable all events of the node for interrupt if interrupt node is enabled.

Otherwise if mixed interrupt servicing and polling scheme is to be used, ignore these dummy interrupt vectoring.

OCDS_XC8.H001 Write to OCDS registers can disturb the Debug Function

The User’s Manual states that the OCDS registers are in general, protected and cannot be written by user software. However, unintentional access to OCDS registers by the user software can disturb the normal debug function. Therefore, users should take care that the application code does not write to the OCDS registers.

OCDS_XC8.H002 Any NMI request is lost on Debug entry and during Debug

All NMI events are disabled while in debug mode. This has two main effects:

1. On debug entry, any pending NMI request will be lost, although the status flag remains set. The probability of losing an NMI request in this way is very low, since NMI always has the highest priority to be serviced.
2. Any NMI event that occurs during debugging is not able to generate an NMI request although the status flag will be set. It is normally not critical that on exit from debug mode, the CPU must service NMI requests that had occurred while in debug mode.

The fact that the debug system is not specified to support NMI interrupt while in debug mode makes the above trivial. As precaution, avoid starting any debug session while expecting an NMI event.

Application Support Group, Singapore