# Errata Sheet

**Rel. 1.7, 28.02.2011**

| | |
|---|---|
| **Device** | **XC886/888CLM Series** |
| **Marking/Step** | **AA** |
| **Package** | **PG-TQFP-48/64** |

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.

This Errata Sheet covers the following devices:

- XC886/888C-6/8RF
- XC886/888CM-6/8RF
- XC886/888CLM-6/8RF

**Table 1 Current Documentation**

| | | |
|---|---|---|
| XC886/888CLM User's Manual | V1.3 | Feb 2010 |
| XC886/888CLM Data Sheet | V1.2 | Jul 2009 |
| SAA-XC886CLM Data Sheet | V1.1 | Aug 2010 |
| SAL-XC886CLM Data Sheet | V1.0 | May 2010 |

Each erratum identifier follows the pattern Module_Arch.TypeNumber:

- **Module**: subsystem or peripheral affected by the erratum
- **Arch**: microcontroller architecture where the erratum was firstly detected.
  - **AI**: Architecture Independent (detected on module level)
  - **CIC**: Companion ICs
  - **TC**: TriCore (32 bit)
  - **X**: XC1xx / XC2000 (16 bit)
  - **XC8**: XC800 (8 bit)

  – **none**: C16x (16 bit)
- **Type**: none - Functional Deviation; '**P**' - Parametric Deviation; '**H**' - Application Hint; '**D**' - Documentation Update
- **Number**: ascending sequencial number within the three previous fields. As this sequence is used over several derivatives, including already solved deviations, gaps inside this enumeration can occur.

*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.

# 1 History List / Change Summary

**Table 2 History List**

| Version | Date | Remark |
|---|---|---|
| 1.0 | 22.09.2006 | |
| 1.1 | 09.10.2006 | |
| 1.2 | 15.02.2007 | |
| 1.3 | 08.06.2007 | |
| 1.4 | 23.11.2007 | |
| 1.5 | 22.02.2008 | |
| 1.6 | 13.02.2009 | |

**Table 3 Errata fixed in this step**

| Errata | Short Description | Chg |
|---|---|---|

**Table 4 Functional Deviations**

| Functional Deviation | Short Description | Chg | Pg |
|---|---|---|---|
| BROM_XC8.006 | IRAM data is corrupted after any warm reset | | 8 |
| BROM_XC8.010 | SYSCON0.RMAP Switching Error | | 8 |
| CD_XC8.001 | Set and Clear of Error Bit in CORDIC Linear Vectoring Mode | | 9 |
| CD_XC8.002 | Data Fetch to CD_STATC Register may capture an incorrect error status | | 9 |

**Table 4 Functional Deviations** (cont'd)

| Functional Deviation | Short Description | Chg | Pg |
|---|---|---|---|
| INT_XC8.004 | Unable to Detect New Interrupt Request if Any One of Timer 2/Timer 21/UART1 Interrupt Flags Is Not Cleared (Unexpectedly) | | 10 |
| INT_XC8.005 | Write to IRCON0 Blocks Interrupt Request of External Interrupt 0,1 | | 13 |
| LIN_XC8.001 | Fast LIN BSL does not support baud rate of up to 115.2 kHz | | 13 |
| MultiCAN_AI.040 | Remote frame transmit acceptance filtering error | | 13 |
| MultiCAN_AI.041 | Dealloc Last Obj | | 14 |
| MultiCAN_AI.042 | Clear MSGVAL during transmit acceptance filtering | | 14 |
| MultiCAN_AI.043 | Dealloc Previous Obj | | 15 |
| MultiCAN_AI.044 | RxFIFO Base SDT | | 16 |
| MultiCAN_AI.045 | OVIE Unexpected Interrupt | | 16 |
| MultiCAN_AI.046 | Transmit FIFO base Object position | | 16 |
| MultiCAN_TC.025 | RXUPD behavior | | 17 |
| MultiCAN_TC.026 | MultiCAN Timestamp Function | | 17 |
| MultiCAN_TC.027 | MultiCAN Tx Filter Data Remote | | 18 |
| MultiCAN_TC.028 | SDT behavior | | 18 |
| MultiCAN_TC.029 | Tx FIFO overflow interrupt not generated | | 19 |
| MultiCAN_TC.030 | Wrong transmit order when CAN error at start of CRC transmission | | 21 |
| MultiCAN_TC.031 | List Object Error wrongly triggered | | 21 |
| MultiCAN_TC.032 | MSGVAL wrongly cleared in SDT mode | | 22 |
| MultiCAN_TC.035 | Different bit timing modes | | 22 |
| MultiCAN_TC.037 | Clear MSGVAL | | 24 |
| MultiCAN_TC.038 | Cancel TXRQ | | 25 |

**Table 4 Functional Deviations** (cont'd)

| Functional Deviation | Short Description | Chg | Pg |
|---|---|---|---|
| OCDS_XC8.008 | Watchdog Timer behavior during Debug with Suspend | | 25 |
| PIN_XC8.005 | Glitches on TCK when switching between clock sources | | 26 |
| PIN_XC8.007 | External pull-up device is required on MBC pin to enter user mode | | 26 |
| SYS_XC8.001 | MOV (direct, direct) instruction might cause a wrong value to be written to the destination register | | 27 |
| T2_XC8.001 | Timer 2 is not suspended during debug in counter mode | | 33 |
| UART_XC8.001 | Bits RB8, TI and RI in UART1_SCON SFR cannot be Written by SETB, CLR and CPL Instructions | | 33 |
| UART_XC8.002 | Bits FDEN and FDM in UART1_FDCON SFR cannot be Written by Read-Modify-Write Instructions | | 33 |

**Table 5 Deviations from Electrical- and Timing Specification**

| AC/DC/ADC Deviation | Short Description | Chg | Pg |
|---|---|---|---|

**Table 6 Application Hints**

| Hint | Short Description | Chg | Pg |
|---|---|---|---|
| ADC_AI.H001 | Arbitration Mode with disabled Arbitration Slots | | 36 |
| ADC_XC8.H001 | Arbitration mode when using external trigger at the selected input line REQTR | | 36 |
| BROM_XC8.H001 | SYSCON0.RMAP handling in ISR | | 37 |
| BROM_XC8.H002 | Obtain Product Derivative Information With Chip Identification Number | | 37 |
| CCU6_XC8.H002 | CCU6 PM event in center-aligned mode | New | 38 |
| FLASH_XC8.H003 | Verify if a Flash program or erase operation is successful | | 39 |
| INT_XC8.H003 | Interrupt Flags of External Interrupt 0 and 1 | | 39 |
| INT_XC8.H004 | NMI Interrupt Request With No NMI Flag Set | | 40 |
| INT_XC8.H005 | Not all Flags are qualified for clearing Pending Interrupt Request | | 41 |
| LIN_XC8.H001 | LIN BRK field detection logic | | 44 |
| LIN_XC8.H002 | LIN Break/Synch field detection | Update | 44 |
| MultiCAN_AI.H005 | TxD Pulse upon short disable request | | 45 |
| MultiCAN_TC.H002 | Double Synchronization of receive input | | 45 |
| MultiCAN_TC.H003 | Message may be discarded before transmission in STT mode | | 45 |
| MultiCAN_TC.H004 | Double remote request | | 46 |
| OCDS_XC8.H002 | Any NMI request is lost on Debug entry and during Debug | | 46 |
| PIN_XC8.H001 | Current over GPIO pin must not source $V_{DDP}$ higher than 0.3V | | 47 |
| PLL_XC8.H001 | Check oscillator run bit 2048 VCO cycles after oscillator run detection is restarted. | | 47 |

**Table 6 Application Hints** (cont'd)

| Hint | Short Description | Chg | Pg |
|---|---|---|---|
| SYS_XC8.H001 | Usage of the Bit Protection Scheme | | 48 |
| SYS_XC8.H003 | Effective write for Read-Modify-Write instructions of two bytes, one machine cycle | New | 48 |
| SYS_XC8.H004 | Switch PLL to prescaler mode before a WDT reset | New | 49 |

# 2 Functional Deviations

### BROM_XC8.006 IRAM data is corrupted after any warm reset

After any warm reset (i.e. reset without powering off the device), boot up via User Mode affects certain IRAM data.

The affected IRAM address ranges are:

(1) $00_H$ - $07_H$

(2) $80_H$ - $C7_H$

**Workaround**

None

### BROM_XC8.010 SYSCON0.RMAP Switching Error

When executing from XRAM, if SYSCON0.RMAP is switched using an one-machine-cycle read-modify-write instruction (e.g. ORL dir,A) and the SFR is accessed immediately by an one-machine-cycle instruction (e.g. MOV A,dir) or a PUSH instruction, the SFR from the previous mapping might be accessed instead.

This RMAP switching error does not occur if code is executed from the Flash memory.

**Workaround**

When executing code from XRAM, use two-machine-cycle instructions to either switch RMAP or access the SFR. Alternatively, add one or more instructions (e.g. NOP) between the one-machine-cycle RMAP switching and SFR accessing instructions.

### CD_XC8.001 Set and Clear of Error Bit in CORDIC Linear Vectoring Mode

In linear vectoring mode, the Error bit of register CD_STATC is set immediately on detecting overflow. When detected between iterations – the Error status is not held internally till the end of the calculation.

As the Error bit is defined such that it is cleared on any read access to the register (e.g. JB BSY), SW checking of the Error bit only at the end of calculation may miss to detect an overflow error condition.

**Workaround**

Especially in linear vectoring mode, if the error condition setting of Error bit must be detected, any read access should be done on the whole CD_STATC register (e.g. MOV) and the Error bit checked in all read instances.

### CD_XC8.002 Data Fetch to CD_STATC Register may capture an incorrect error status

The error bit CD_STATC.ERROR is defined such that the bit is cleared on any read access to the register. Therefore, it is necessary to perform a data fetch on the register and check for the error bit in order not to lose the error status.

However, if the CPU inserts a wait state during the execution of the read instruction from the Flash and extends the read access by another two clock cycles, multiple read accesses will be performed on the CD_STATC register and the error bit will be cleared by the time the CPU performs the final read. As a result, the CPU does not capture the correct error status.

There is no problem if the code to read CD_STATC register is located in the XRAM.

**Workaround**

The following workarounds can be used to avoid incorrect data fetching from the CD_STATC register:

- The PUSH dir and POP dir instructions can be used to read the CD_STATC register in all conditions;

- The following MOV instructions can be used to read the CD_STATC register if the P-Flash parallel read mode is enabled (default) and the MOV instruction is placed on an odd address of the P-Flash[1)]:
  – MOV Rx, dir (where x = 0...7)
  – MOV @Rn, dir (where n = 0 or 1)
  – MOV A, dir
- The following MOV instruction can be used to read the CD_STATC register if the P-Flash parallel read mode is enabled (default) and the MOV instruction is placed on an even address of the P-Flash[1)]:
  – MOV dir, dir

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

*Note: In the current device step, the bug in Timer 2 and Timer 21 has been fixed. Therefore, the errata is applicable only to UART1.*

As illustrated in the simplified figure, the UART1 interrupt flags RI and TI are combined as one interrupt request output. These flags are located within the UART1 kernel, with a single interrupt request line as output from the kernel.

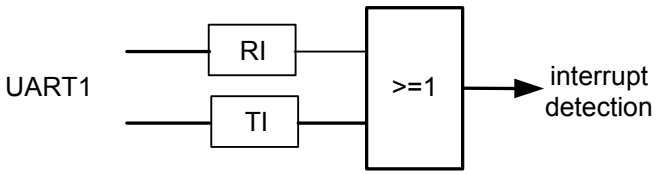



**Figure 1**

1) The workaround does not work if the P-Flash parallel read mode is disabled (through the parallel read disable subroutine in the Boot ROM), or if the code to read the CD_STATC register is located in the D-Flash.

Being of interrupt structure 2, the interrupt request of UART1 is detected on the rising edge of a positive pulse.

The problem is that it may occur at some point in the application, that any new UART1 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. UART1 events are serviced by interrupt (i.e. flags are checked and cleared only in the interrupt routine ISR),
2. Either RI or TI 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 UART1 request line, which is an OR-function of the two flags RI and TI, to remain set throughout the ISR and after reti.

Two example scenerios are illustrated in the following figure:

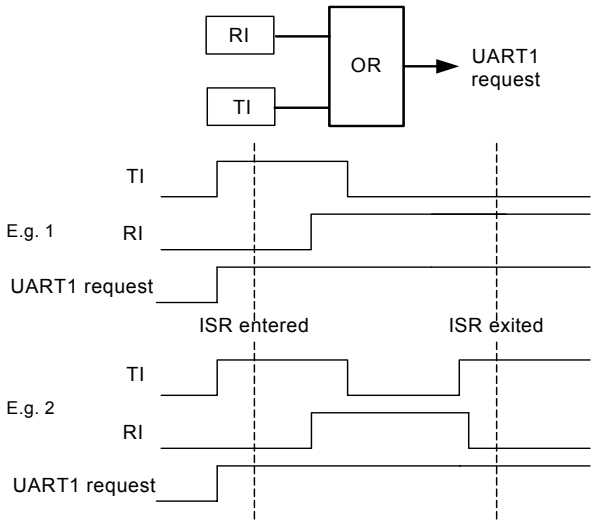



**Figure 2**

This means, any future UART1 RI or TI event is not able to cause a rising edge and therefore not able to trigger an interrupt request to the core – as if the UART1 interrupts had been disabled, until both RI and TI flags are cleared at some time. The clearing of flags would have to be done by user's code

additionally outside of the UART1 interrupt routine, which is however normally not feasible with an interrupt service scheme.

*Note: This condition affects only the detection of UART1 interrupt events RI and TI. It does not block the detection of other interrupt events belonging to the same interrupt node.*

**Workaround**

There are two suggested workaround:

1. If other events of interrupt node XINTR8 (EX2) 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 RI or TI 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 UART1 events are occurring at high rate, the CPU may be 'stuck' in the service routine of the UART1 interrupt for a long time.

```
<entry point of interrupt node service routine>
…..
Start:
check flag RI
…..
clear flag RI
…..
check flag TI
…..
clear flag TI
…..
Finish:
if RI or TI is set, jump to Start
reti (return from interrupt)
```

**Figure 3**

*Note: The Boolean CLR and CPL instructions cannot be used to clear RI and TI bits of UART1. Refer to UART_XC8.001.*

### 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).

### LIN_XC8.001 Fast LIN BSL does not support baud rate of up to 115.2 kHz

Fast LIN BSL supports only a baud rate of up to 57.6 kHz and not up to 115.2 kHz as described in the user's manual.

**Workaround**

None

### 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.

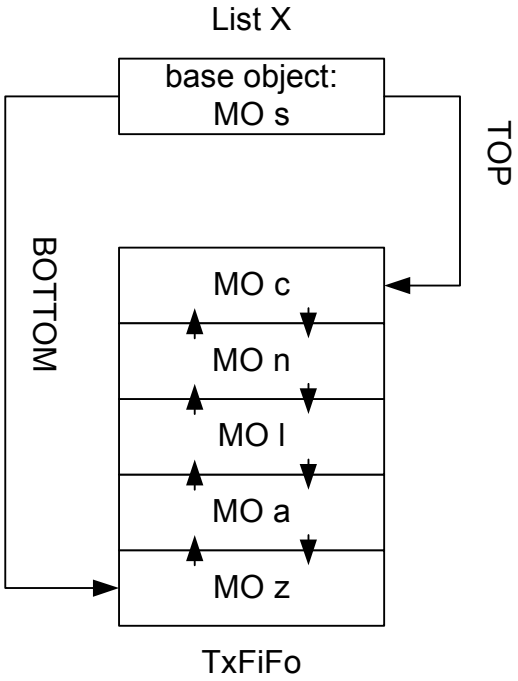



**Figure 4 FIFO structure**

**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.

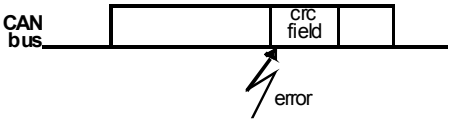



**Figure 5**

**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 7**

| 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_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.

### PIN_XC8.005 Glitches on TCK when switching between clock sources

The JTAG clock input, TCK, has multiple sources; two in XC886 variants (P0.0, P2.0) and three in XC888 variants (P0.0, P2.0, P5.6). Unexpected glitches may occur when the clock source is switched from:

1. P0.0 to P2.0; or
2. P2.0 to P0.0; or
3. P5.6 as a normal GPIO pin to TCK.

These glitches may potentially bring the system into an undefined state.

**Workaround**

To prevent the occurrence of such glitches during the clock switching, the following 2 workarounds are proposed:

1. The user has to follow a specific sequence when carrying out any of the above clock switching. When switching from P0.0 to P2.0 or vice versa, the user must first switch the clock source to P5.6 before proceeding to switch to the desired clock source. Similarly, when switching P5.6 from a normal GPIO pin to TCK, the user must first assigned TCK to either P0.0 or P2.0 before switching from this clock source to P5.6. (Valid only for XC888 variants)
2. The user has to configure all the TCK pins as input and drive them to either all zero or all one before switching the clock source from one pin to another.

### PIN_XC8.007 External pull-up device is required on MBC pin to enter user mode

Although MBC pin is specified to be pull-up in the reset state, an external pull-up device is still required to enter user mode.

**Workaround**

Implement external pull-up (in the range of 4.7 kΩ to 100 kΩ) on the external circuitry for MBC pin. Alternatively, MBC pin could be tied to high.

### SYS_XC8.001 MOV (direct, direct) instruction might cause a wrong value to be written to the destination register

The MOV (direct, direct) instruction (hex code $85_H$) that access registers (direct address ranging from $80_H$ to $FF_H$), does not write the correct value of the source register to the destination register if the destination register is a register listed in the table below.

The source register can be any register from the direct address range $80_H$ to $FF_H$.

**Table 8**

| Module | Register | SFR Address | RMAP | Page | Products Affected |
|---|---|---|---|---|---|
| SCU | IRCON0 | $B4_H$ | 0 | 0 | XC88x, XC878 |
| | IRCON1 | $B5_H$ | 0 | 0 | XC88x, XC878 |
| | IRCON2 | $B6_H$ | 0 | 0 | XC88x, XC878 |
| | IRCON3 | $B4_H$ | 0 | 3 | All |
| | IRCON4 | $B5_H$ | 0 | 3 | All |
| | NMISR | $BC_H$ | 0 | 0 | XC88x, XC878 |
| | FDCON | $E9_H$ | 0 | 0 | XC88x, XC878 |
| | PMCON0 | $B4_H$ | 0 | 1 | XC88x, XC878 |
| | OSC_CON | $B6_H$ | 0 | 1 | XC88x, XC878 |
| | PLL_CON | $B7_H$ | 0 | 1 | XC88x |
| | MISC_CON | $E9_H$ | 0 | 1 | XC88x, XC878 |
| WDT | WDTCON | $BB_H$ | 1 | - | XC88x, XC878 |
| CORDIC | CD_STATC | $A0_H$ | 1 | - | XC88x, XC878 |
| MDU | MDUSTAT | $B0_H$ | 1 | - | XC88x, XC878 |

**Table 8**

| Module | Register | SFR Address | RMAP | Page | Products Affected |
|---|---|---|---|---|---|
| SSC | CONH (Operating Mode) | $AB_H$ | 0 | - | All |
| UART1 | SCON | $C8_H$ | 1 | - | XC88x, XC878 |
| | FDCON | $CC_H$ | 1 | - | XC88x, XC878 |
| T2 | T2CON | $C0_H$ | 0 | - | All |
| T21 | T2CON | $C0_H$ | 1 | - | XC88x, XC878 |
| OCDS | MMCR2 | $E9_H$ | 1 | - | All |
| | MMCR | $F1_H$ | 1 | - | All |
| | MMSR | $F2_H$ | 1 | - | All |
| | MMICR | $F4_H$ | 1 | - | All |
| T2CCU | CCTCON | $C6_H$ | 0 | 1 | XC878 |
| | COSHDW | $C0_H$ | 0 | 2 | XC878 |
| | COCON | $C0_H$ | 0 | 3 | XC878 |
| | CC0L | $C1_H$ | 0 | 2 | XC878 |
| | CC0H | $C2_H$ | 0 | 2 | XC878 |
| | CC1L | $C3_H$ | 0 | 2 | XC878 |
| | CC1H | $C4_H$ | 0 | 2 | XC878 |
| | CC2L | $C5_H$ | 0 | 2 | XC878 |
| | CC2H | $C6_H$ | 0 | 2 | XC878 |
| | CC3L | $C1_H$ | 0 | 3 | XC878 |
| | CC3H | $C2_H$ | 0 | 3 | XC878 |
| | CC4L | $C3_H$ | 0 | 3 | XC878 |
| | CC4H | $C4_H$ | 0 | 3 | XC878 |
| | CC5L | $C5_H$ | 0 | 3 | XC878 |
| | CC5H | $C6_H$ | 0 | 3 | XC878 |

**Table 8**

| Module | Register | SFR Address | RMAP | Page | Products Affected |
|---|---|---|---|---|---|
| CCU6 | CC63SRL | $9A_H$ | 0 | 0 | All |
| | CC63SRH | $9B_H$ | 0 | 0 | All |
| | MCMOUTSL | $9E_H$ | 0 | 0 | All |
| | MCMOUTSH | $9F_H$ | 0 | 0 | All |
| | CC60SRL | $FA_H$ | 0 | 0 | All |
| | CC60SRH | $FB_H$ | 0 | 0 | All |
| | CC61SRL | $FC_H$ | 0 | 0 | All |

**Table 8**

| Module | Register | SFR Address | RMAP | Page | Products Affected |
|---|---|---|---|---|---|
| CCU6 (cont'd) | CC61SRH | $FD_H$ | 0 | 0 | All |
| | CC62SRL | $FE_H$ | 0 | 0 | All |
| | CC62SRH | $FF_H$ | 0 | 0 | All |
| | T12PRL | $9C_H$ | 0 | 1 | All |
| | T12PRH | $9D_H$ | 0 | 1 | All |
| | T13PRL | $9E_H$ | 0 | 1 | All |
| | T13PRH | $9F_H$ | 0 | 1 | All |
| | T12DTCL | $A4_H$ | 0 | 1 | All |
| | T12DTCH | $A5_H$ | 0 | 1 | All |
| | TCTR0L | $A6_H$ | 0 | 1 | All |
| | TCTR0H | $A7_H$ | 0 | 1 | All |
| | T12MSELL | $9A_H$ | 0 | 2 | All |
| | T12MSELH | $9B_H$ | 0 | 2 | All |
| | IENL | $9C_H$ | 0 | 2 | All |
| | IENH | $9D_H$ | 0 | 2 | All |
| | INPL | $9E_H$ | 0 | 2 | All |
| | INPH | $9F_H$ | 0 | 2 | All |
| | PSLR | $A6_H$ | 0 | 2 | All |
| | MCMCTR | $A7_H$ | 0 | 2 | All |
| | TCTR2L | $FA_H$ | 0 | 2 | All |
| | TCTR2H | $FB_H$ | 0 | 2 | All |
| | MODCTRL | $FC_H$ | 0 | 2 | All |
| | MODCTRH | $FD_H$ | 0 | 2 | All |
| | TRPCTRL | $FE_H$ | 0 | 2 | All |
| | TRPCTRH | $FF_H$ | 0 | 2 | All |
| | PISEL0L | $9E_H$ | 0 | 3 | All |
| | PISEL0H | $9F_H$ | 0 | 3 | All |
| | PISEL2 | $A4_H$ | 0 | 3 | All |

**Table 8**

| Module | Register | SFR Address | RMAP | Page | Products Affected |
|---|---|---|---|---|---|
| CCU6 (cont'd) | T13L | $FC_H$ | 0 | 3 | All |
| | T13H | $FD_H$ | 0 | 3 | All |
| | CMPSTATH | $FF_H$ | 0 | 3 | All |
| ADC | GLOBCTR | $CA_H$ | 0 | 0 | All |
| | PRAR | $CC_H$ | 0 | 0 | All |
| | LCBR | $CD_H$ | 0 | 0 | All |
| | INPCR0 | $CE_H$ | 0 | 0 | All |
| | ETRCR | $CF_H$ | 0 | 0 | All |
| | CHCTR0 | $CA_H$ | 0 | 1 | All |
| | CHCTR1 | $CB_H$ | 0 | 1 | All |
| | CHCTR2 | $CC_H$ | 0 | 1 | All |
| | CHCTR3 | $CD_H$ | 0 | 1 | All |
| | CHCTR4 | $CE_H$ | 0 | 1 | All |
| | CHCTR5 | $CF_H$ | 0 | 1 | All |
| | CHCTR6 | $D2_H$ | 0 | 1 | All |
| | CHCTR7 | $D3_H$ | 0 | 1 | All |
| | RCR0 | $CA_H$ | 0 | 4 | All |
| | RCR1 | $CB_H$ | 0 | 4 | All |
| | RCR2 | $CC_H$ | 0 | 4 | All |
| | RCR3 | $CD_H$ | 0 | 4 | All |
| | CHINPR | $CD_H$ | 0 | 5 | All |
| | EVINPR | $D3_H$ | 0 | 5 | All |
| | CRCR1 | $CA_H$ | 0 | 6 | All |
| | CRPR1 | $CB_H$ | 0 | 6 | All |
| | CRMR1 | $CC_H$ | 0 | 6 | All |
| | QMR0 | $CD_H$ | 0 | 6 | All |

For example, in the sample code below, there are two MOV (direct, direct) instructions that write the value of one register into another. All the source and destination registers in these two instructions are from the direct address range $80_H$ to $FF_H$.

The P1_DATA register is not one of the affected registers listed in the table above and therefore, it is written with the correct value of the CC60SRL register. On the other hand, the CC60SRH register is one of the affected registers and therefore, it is written with the wrong value of the B register.

Sample Code:

```
interrupt:
MUL A, B
MOV CC60SRL, A
MOV P1_DATA, CC60SRL
MOV CC60SRH, B
RETI
```

**Workaround**

Instead of using the MOV (direct, direct) instruction, use other instructions or an intermediate variable to write to the targeted register.

For example, the two MOV (direct, direct) instructions in the earlier sample code can be replaced with MOV (direct, A) instructions (hex code $F5_H$). Both the P1_DATA and CC60SRH registers will now be written with the correct source register values.

Sample Code:

```
interrupt:
MUL A, B
MOV CC60SRL, A
MOV P1_DATA, A
XCH A, B
MOV CC60SRH, A
RETI
```

### T2_XC8.001 Timer 2 is not suspended during debug in counter mode

Timer 2 is not suspended (even if Timer 2 suspend control is enabled by setting bit MODSUSP.T2SUSP) during debug if Timer 2 is set to counter mode.

**Workaround**

None.

### UART_XC8.001 Bits RB8, TI and RI in UART1_SCON SFR cannot be Written by SETB, CLR and CPL Instructions

The bits RB8, TI and RI in UART1_SCON SFR, which is a bitaddressable SFR, are not updated when written with SETB, CLR and CPL instructions. As a result, the UART1 module is not fully compatible with standard 8051 code.

**Workaround**

Use MOV bit,C instruction to write to the bits RB8, TI and RI in UART1_SCON SFR or target the write access on the whole register.

### UART_XC8.002 Bits FDEN and FDM in UART1_FDCON SFR cannot be Written by Read-Modify-Write Instructions

The bits FDEN and FDM in UART1_FDCON SFR are not updated when written with the read-modify-write instructions listed in the table below:

**Table 9**

| Affected Read-Modify-Write Instructions | Hex Code |
|---|---|
| INC dir | 05 |
| DEC dir | 15 |
| ANL dir,A | 52 |
| ANL dir,#data | 53 |
| ORL dir,A | 42 |

**Table 9**

| Affected Read-Modify-Write Instructions | Hex Code |
|---|---|
| ORL dir,#data | 43 |
| XRL dir,A | 62 |
| XRL dir,#data | 63 |
| XCH A,dir | C5 |
| DJNZ dir,rel | D5 |

### Workaround

Use MOV instructions, except MOV dir, dir (Hex Code: 85), when writing to the bits `FDEN` and `FDM` in `UART1_FDCON` SFR.

# 3 Deviations from Electrical- and Timing Specification

# 4 Application Hints

**ADC_AI.H001 Arbitration Mode with disabled Arbitration Slots**

In arbitration mode (bit ARBM = $1_B$ in register PRAR), the arbiter only runs while at least one conversion request is pending, otherwise it waits in an idle state for a request to become active. This leads to a constant and reproducible latency from an incoming request to the conversion start.

Each request source x (x = 0, 1) can be individually selected (via the Arbitration Slot Enable bits ASENx in register PRAR) to take part in the arbitration round. However, if a disabled request source (bit ASENx = $0_B$) has a pending request, the related conversion is not started, but the arbiter does not stop and wait. As a result, the latency for requests generated on other arbitration slots is not constant.

To avoid this effect, first disable the request generation of the respective source by setting bit ENGT = $0_B$ in the corresponding register QMR0 or CRMR1 before disabling the arbitration slot via ASENx = $0_B$.

**ADC_XC8.H001 Arbitration mode when using external trigger at the selected input line REQTR**

If an external trigger is expected at the selected input line REQTR to trigger a pending request, the arbitration mode should be set (PRAR.ARBM=1) where the arbitration is started by pending conversion request. This selection will minimize the jitter between asynchronous external trigger with respect to the arbiter and the start of the conversion. The jitter can only be minimized while no other conversion is running and no higher priority conversion can cancel the triggered conversion. In this case, a constant delay (no jitter) has to be taken into account between the trigger event and the start of the conversion.

**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.

**BROM_XC8.H002 Obtain Product Derivative Information With Chip Identification Number**

The chip identification number is an unique 4-byte data that is assigned to each product derivative based on the following differentiations:

- product
- variant type
- device-step

Therefore, to identify a product derivative, the number can be obtained and matched to a list of product derivatives and their corresponding numbers, which can be found in the latest product data sheet.

The number is read using the in-application user subroutine, GET_CHIP_INFO, or BSL mode A. Description on the usage of GET_CHIP_INFO and BSL mode A can be found in the latest user’s manual.

## CCU6_XC8.H002 CCU6 PM event in center-aligned mode

After detecting a period match (PM A) in centre-aligned mode, T12 counts down from PM + 1 as shown below:

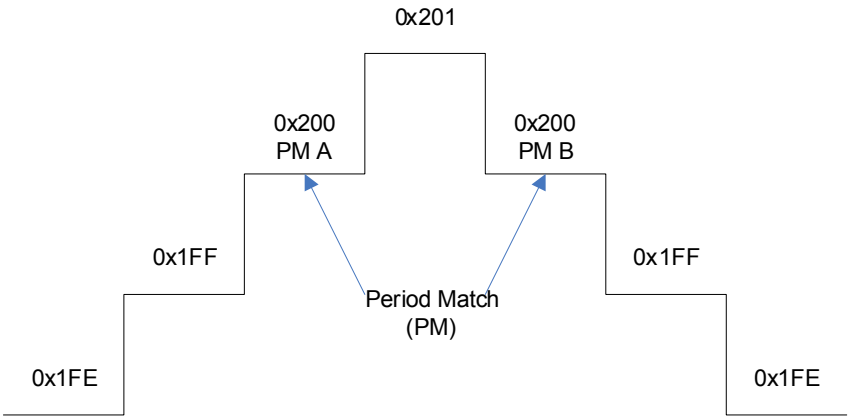



**Figure 6 Counting sequence of T12 in center-aligned mode**

This means a second PM event (PM B) will occur during the counting down. If ADC is triggered externally via ETRx2 (T12PM), it will be triggered twice in succession. Depending on how real-time the application code is running as well as the T12 count rate and ADC conversion rate, the application could observe two ADC interrupts - once at PM A and once at PM B.

To avoid triggering twice the ADC interrupts, it is suggested to use ETRx6 from multi-channel mode instead of ETRx2 as the trigger source for ADC. Additional initialization are as follows:

- Configure MCMCTR.SWSEL = $101_B$ (Transfer on T12 period match)
- Configure MCMCTR.SWSYN = $00_B$ (Direct transfer)
- Write to MCMOUTSTL = $CF_H$ (To enable multi-Channel PWM pattern on CC6x and COUT6x)

*Note: Independent of the external trigger, the CCU6 internal triggers based on T12 PM (e.g. T12 PM interrupt or shadow transfer) are only activiated once while T12 is counting up.*

**FLASH_XC8.H003 Verify if a Flash program or erase operation is successful**

The Flash memory cannot be programed or erased under a PLL loss-of-lock condition. In the Boot ROM program and erase routines, the Boot ROM checks that the PLL NMI flag `NMISR.FNMIPLL` is zero to ensure no PLL loss-of-lock has occurred before starting the program or erase sequence. If the PLL NMI flag is set, the Boot ROM will set the carry flag `PSW.CY` and exit the routine without starting the program or erase sequence.

A manual check on the Flash data or the carry flag is necessary to determine if the program or erase operation is successful.

**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.

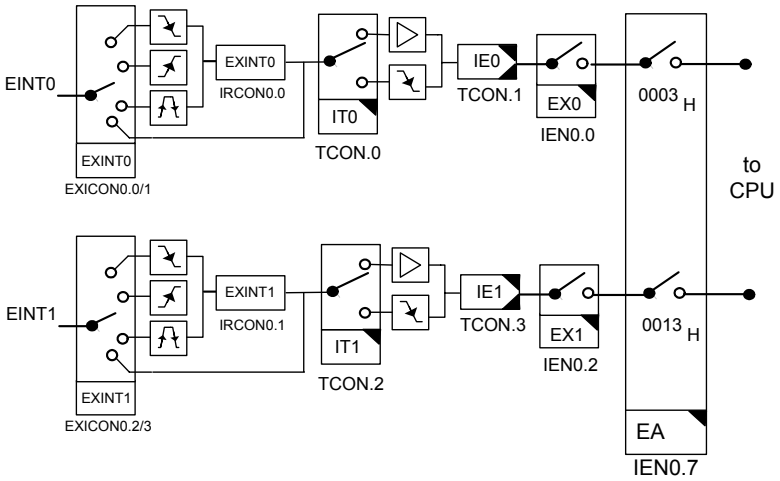



**Figure 7**

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.

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:

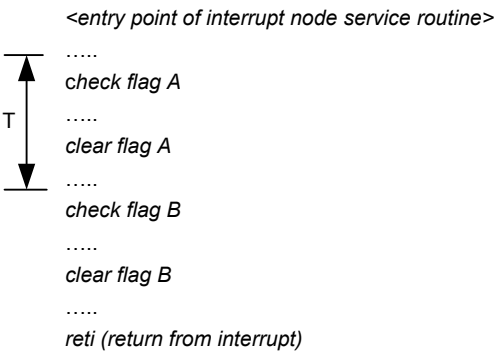



**Figure 8**

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.

**Table 10**

| 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 |
| | | UART1 | RI, TI |
| | | UART1 Fractional Divider (Normal Divider Overflow) | NDOV |
| | | 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:

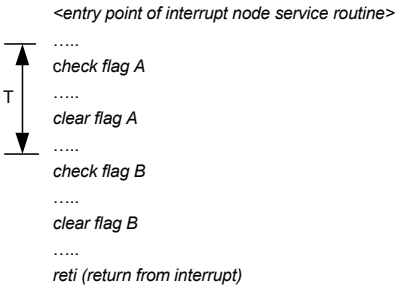



**Figure 9**

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.

## LIN_XC8.H001 LIN BRK field detection logic

Based on the hardware implementation, the maximum number of bits in the BRK field must follow the formula:

$$\text{Maximum number of bits in BRK field} = \text{Baud Rate} \times \frac{4095}{\text{Sample Frequency}}$$

$$\text{Sample Frequency} = \frac{\text{PCLK}}{8 \times 2^{\text{BGSEL}}}$$

For example, if LIN baudrate is 19.2kbps, BGSEL = 0 and CPU frequency is 24MHz, the maximum number of bits in BRK field would be:

19.2k x 4095 / (24M / 8) = ~26.2 bits

If the maximum number of bits in the BRK field exceeded, the internal counter will overflow which results in baudrate detection error. Therefore, the user is advised to choose the appropriate BGSEL value for the required baudrate detection range.

The calculated value above does not consider sample error and transmission error, nevertheless it can be used as a guideline.

## LIN_XC8.H002 LIN Break/Synch field detection

The LIN Break/Synch field detection is default enabled after every reset (`BCON.BRDIS` bit = 0). However, to ensure the first standard LIN frame can always be detected, it is recommended to initialize the detection logic in the user code before receiving the frame. This is through the following two steps:

1. Toggle `BCON.BRDIS` bit (set the bit to 1 before clearing it back to 0).
2. Clear the three status flags `FDCON.BRK`, `FDCON.EOFSYN` and `FDCON.ERRSYN` to 0.

It is also recommended to toggle the BCON.BRDIS bit after the reception of each complete LIN frame to avoid a wrong Break field detection in noisy environments (i.e. spikes on the LIN bus).

### 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:

PMCON1.CAN_DIS = 1 and then PMCON1.CAN_DIS = 0

**Workaround**

Set all INIT bits to 1 before requesting module disable.

### 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 frequency 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 etc.).

### 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.

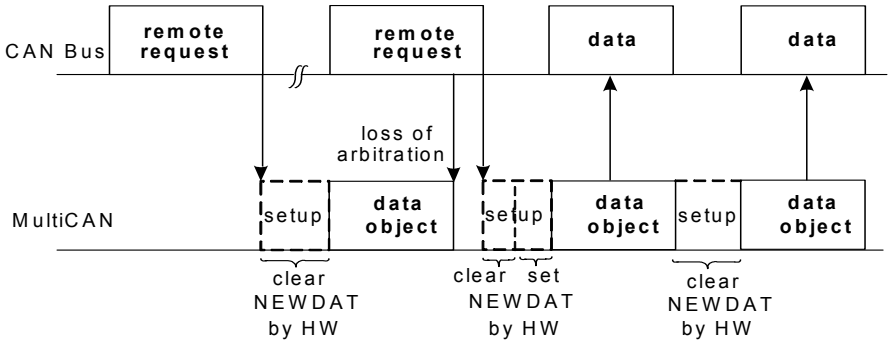



**Figure 10 Loss of Arbitration**

## 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 (event interrupt is lost) 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.

**PIN_XC8.H001 Current over GPIO pin must not source $V_{DDP}$ higher than 0.3V**

When $V_{DDP}$ is not powered on, the current over a GPIO pin has to be limited in such a way that $V_{DDP}$ - $V_{SSP} \leq 0.3V$. This prevents the supply of the device via the ESD diode between the GPIO pin and $V_{DDP}$.

However, for applications with strict low power-down current requirements, it is mandatory that no active voltage source is supplied at any GPIO pin when $V_{DDP}$ is not powered on.

**PLL_XC8.H001 Check oscillator run bit 2048 VCO cycles after oscillator run detection is restarted.**

When performing a loss-of-lock recovery or changing to an external oscillator, one of the steps required is to restart the oscillator run detection logic by setting `OSC_CON.ORDRES` bit to 1.

After the oscillator run detection logic is restarted, the user should wait for minimum 2048 VCO cycles (approximately between 30 µs and 200 µs depending on VCO base frequency) before checking the status of the oscillator run bit `OSC_CON.OSCR`.

### SYS_XC8.H001 Usage of the Bit Protection Scheme

When the bit protection scheme is enabled, bit field `PASSWD.PASS` should always be used to open and close write access to the protected bits. The scheme should be disabled only if it is not required in the application.

In the unlikely event that the scheme is enabled again after disabling it while the write access is still open, the write access will remain open until the count of 32 CCLK cycles is completed.

### SYS_XC8.H003 Effective write for Read-Modify-Write instructions of two bytes, one machine cycle

When read-modify-write instructions requiring 2 bytes and 1 machine cycle (equivalent to 2 CCLK cycles) for execution, such as INC dir, are executed from memories without any wait states[1)], the actual write to the destination is delayed by the internal bus for up to one CCLK cycle. This means that even though the CPU completes the instruction execution after 2 CCLK cycles, the write through the internal bus may take effect only after a further CCLK cycle.

The list of affected read-modify-write instructions is shown below:

**Table 11**

| Mnemonic | Hex Code | Bytes | No. of CCLK cycles (without wait states) |
|---|---|---|---|
| INC dir | 05 | 2 | 2 |
| DEC dir | 15 | 2 | 2 |
| ANL dir, A | 52 | 2 | 2 |
| ORL dir, A | 42 | 2 | 2 |
| XRL dir, A | 62 | 2 | 2 |
| XCH A, dir | C5 | 2 | 2 |
| CLR bit | C2 | 2 | 2 |

1) Applicable also to Flash memory with parallel read feature.

**Table 11**

| Mnemonic | Hex Code | Bytes | No. of CCLK cycles (without wait states) |
|---|---|---|---|
| SETB bit | D2 | 2 | 2 |
| CPL bit | B2 | 2 | 2 |

## SYS_XC8.H004 Switch PLL to prescaler mode before a WDT reset

A WDT reset does not reset the clock system. If a WDT reset occurs while the PLL is in base mode (`PLL_CON.OSCDISC` = 1), the system will be held in reset until the next power-on or hardware reset. This is because the system requires the PLL to be locked (`PLL_CON.LOCK` = 1) or in the prescaler mode (`PLL_CON.OSCDISC` = 0; `PLL_CON.VCOBYP` = 1) before the reset can be released.

If the above behaviour is not desired, it is recommended to switch the PLL to prescaler mode, and with the on-chip oscillator as the input clock source (`OSC_CON.OSCSS` = 0), whenever a WDT prewarning is entered.