Infineon

Errata Sheet

Device	**XC886LM-6FF**
Marking/Step	**ES AB**
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.

Table 1   Current Documentation

XC886/888CLM User's Manual	V0.3	Aug 2006
XC886/888CLM Data Sheet	V0.2	Oct 2006

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 / XC2xx (16 bit)
  - **XC8**: XC8xx (8 bit)
  - **none**: C16x (16 bit)
- **Type**: none - Functional Deviation; '**P**' - Parametric Deviation; '**H**' - Application Hint; '**D**' - Documentation Update
- **Number**: ascending sequential number within the three previous fields. As this sequence is used over several derivatives, including already solved deviations, gaps inside this enumeration can occur.

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	29.09.2006	

Table 3   Errata fixed in this step

Errata	Short Description	Chg
ADC_XC8.003	Limitation during power-down mode	Fixed
BROM_XC8.009	LIN BSL Mode 4 (Flash Mass Erase) does not work correctly.	Fixed
MDU_XC8.001	Overflow during Left Shift may lead to a wrong shift result	Fixed
OCDS_XC8.H001	Write to OCDS registers can disturb the Debug Function	Fixed
OSC_XC8.002	OSC_CON register is not reset following a wake-up reset	Fixed
PIN_XC8.004	Reset values of `PUDSEL` and `PUDEN` registers in Port 4 and Port 5 are wrong.	Fixed
PLL_XC8.001	PLL N-Divider is reset to default value after a WDT reset	Fixed
WDT_XC8.001	WDT is not default suspended in Monitor Mode	Fixed
WDT_XC8.002	WDT should not be refreshed during Prewarning period	Fixed

Table 4   Functional Deviations

Functional Deviation	Short Description	Chg	Pg
BROM_XC8.006	**IRAM data is corrupted after any warm reset**	Update	**6**
BROM_XC8.010	**SYSCON0.RMAP Switching Error**		**6**
CD_XC8.001	**Set and Clear of Error Bit in CORDIC Linear Vectoring Mode**		**7**
EVR_XC8.005	**Reset toggling issue for repeated power up**	New	**7**
FLASH_XC8.004	**Wrong data fetched during backward read-access in P-Flash with Parallel Read Mode enabled**		**9**
INT_XC8.004	**Unable to Detect New Interrupt Request if Any One of Timer 2/UART1 Interrupt Flags Is Not Cleared (Unexpectedly)**	Update	**14**
INT_XC8.005	**Write to IRCON0 Blocks Interrupt Request of External Interrupt 0,1**		**17**
OCDS_XC8.008	**Watchdog Timer behavior during Debug with Suspend**		**18**
PIN_XC8.005	**Glitches on TCK when switching between clock sources**		**18**
PIN_XC8.006	**Port 4 pads (excluding P4.2) toggle once upon a power-on or hardware reset**		**19**
UART_XC8.001	**Bits RB8, TI and RI in UART1_SCON SFR cannot be Written by SETB, CLR and CPL Instructions**		**19**

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_XC8.H001	**Arbitration mode when using external trigger at the selected input line REQTR**		**21**
BROM_XC8.H001	**SYSCON0.RMAP handling in ISR**		**21**
BROM_XC8.H002	**Obtain Product Derivative Information With Chip Identification Number**		**22**
FLASH_XC8.H002	**Programming to invalid P-Flash address range 5000$_H$ to 5FFF$_H$ may corrupt existing P-Flash data**	New	**22**
INT_XC8.H003	**Interrupt Flags of External Interrupt 0 and 1**		**22**
INT_XC8.H004	**NMI Interrupt Request With No NMI Flag Set**		**24**
INT_XC8.H005	**Not all Flags are qualified for clearing Pending Interrupt Request**		**25**
LIN_XC8.H001	**LIN BRK field detection logic**		**27**
LIN_XC8.H002	**EOFSYN and ERRSYN bits status after exit from Boot ROM**		**28**
OCDS_XC8.H002	**Any NMI request is lost on Debug entry and during Debug**		**29**

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.

EVR_XC8.005 Reset toggling issue for repeated power up

Due to the limitation of the EVR, the reset toggling may occur during the power up. To prevent the reset toggling issue for power up, the following conditions must be met :

1. For power up, the rising time must be less than 2ms/V.
2. The power down time must be more than 0.2ms/V.
3. A permanent V_{DDP} residual voltage is less than 500mV.

In case of V_{DDP} residual voltage is more than 500mV, the holding time of residal voltage affects the reset toggling issue. **Figure 1** shows the critical cases for power up/power down when the residual voltages is held for some time. **Figure 2** shows the uncritical case for power up and power down when the power up is triggered immediately after power down. In case of a residual voltage due to immediate power up after power down, the users have to contact IFX to ensure no power up failure.
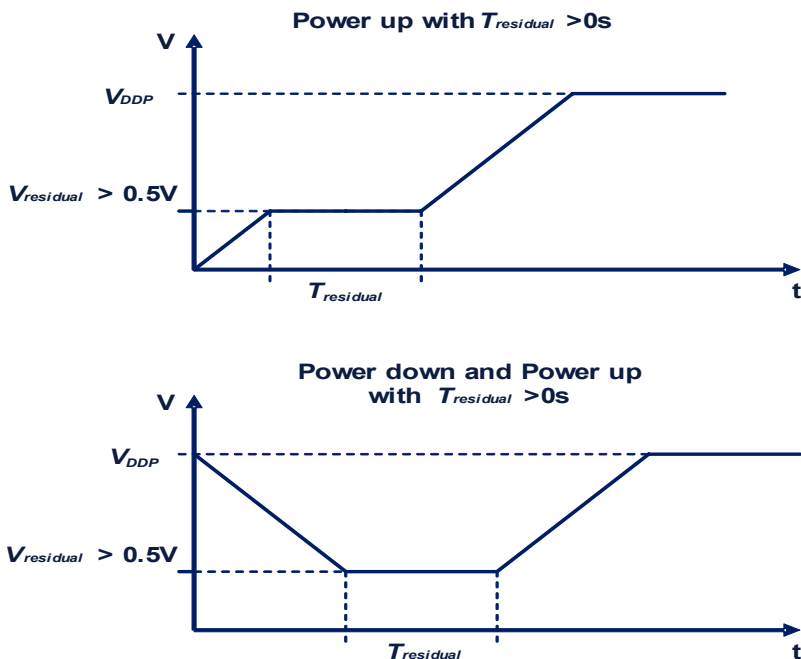
Power up with $T_{residual}$ >0s

Power down and Power up with $T_{residual}$ >0s

Figure 1   Critical Cases for Power up/ Power down when $V_{residual}$ >500mV

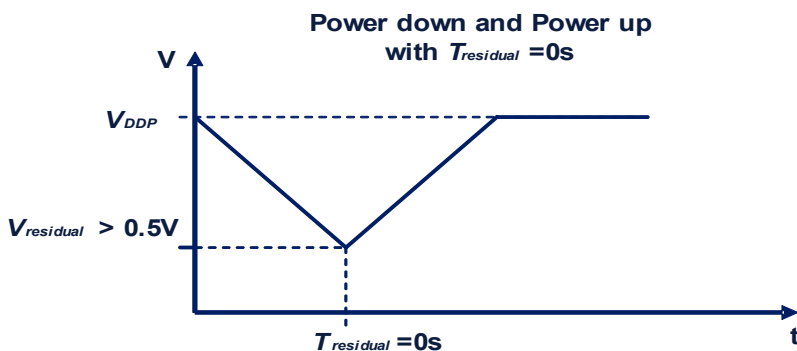**Power down and Power up with $T_{residual}$ =0s**

Figure 2   Uncritical Case for Power up/Power down when $V_{residual}$ >500mV

Workaround

None.

FLASH_XC8.004 Wrong data fetched during backward read-access in P-Flash with Parallel Read Mode enabled

Program Flash Banks 0 to 5 (address range 0000_H to $4FFF_H$) implement the parallel read feature for an increased performance during Flash read-access. For that, the Flash contains a 2-Byte wide cache (even and odd), and an address comparison circuitry that decides if access is served from the Flash Array or from this cache. Note that this mechanism applies, regardless of whether the Flash read-access is a code fetch or a data access. For the purpose of simplicity, the term "data byte" is used to mean either code or data.

If a read-access is requested on an even address, and the requested data byte is not available in the cache, the Flash reads from the Flash Array the even data byte. In parallel, it also prefetches another data byte from the following odd address in the Flash Array. This 2-byte even-odd pair is stored respectively in the even and odd cache, and the data byte in the even cache is also sent to the CPU.

If a read-access is requested on an odd address, and the requested data byte is not available in the cache, the Flash reads from the Flash Array the odd data byte. However, it does not perform an additional prefetch of the preceeding even address. While the odd data byte is stored in the odd cache and sent to the CPU, the even cache remains unchanged and is therefore not up-to-date.

In the current implementation, when parallel read access mode is enabled, performing a read-access to an odd address in the Program Flash, followed by a subsequent read access to its preceeding even address causes the access to the even address to return wrong data. This is because the data byte is not read from the Flash Array but is read from the even cache which contains an out-of-date data.

The problem is caused by the following read sequence to P-Flash:

1. Read access to X (where X is located at an odd address)

2. Read access to X-1

The error occurs during the read-access to even address X-1. The read-access will not read out the data byte at address X-1, but instead will return the out-of-date data byte in the even cache.

Read-access can be generated with code fetch or a data access. The problem is encountered only when there is backward flowing read-access to Program Flash.

In the case of data access, MOVC A,@A+DPTR and MOVC A,@A+PC instructions are the only possible instructions that can create the problem. In view of this, there are only 3 possible scenarios that can create the problem.

Scenario 1

1. perform data access at odd address in P-Flash. (via MOVC instruction).
2. perform code fetch at odd-1 address P-Flash.

For example:

Table 7

Address	Code		
1280_H	04	inc a	
1281_H	93	movc a, @a+dptr	;; a+dptr = 1283_H
1282_H (X-1)	14	dec a	
1283_H (X)	E4	clr a	

After the MOVC instruction (located at 1281_H) performs a read access to address 1283_H, which reads out $E4_H$, the program counter increments to the next read access at address 1282_H. The latter read access should fetch code 14_H from the Flash Array, but instead it fetches 04_H from the even cache which has not been updated with new code but rather still contains the code (04_H inc a) from the last read access on an even address. The CPU receives the wrong codes in the sequence 04_H 93_H $E4_H$ 04_H $E4_H$. The correct code sequence should be 04_H 93_H $E4_H$ 14_H $E4_H$. The code at 1282_H is skipped. Hence, this is an example of a wrong code fetch.

Scenario 2

1. perform branch to odd address P-Flash.
2. perform data access odd-1 address P-Flash. (via MOVC instruction).

For example:

Table 8

Address	Code		
1280_H	80	sjmp label	
1281_H	01		
1282_H (X-1)	E4	clr a	
1283_H (X)	93	label: movc a, @a+dptr	;; a+dptr = 1282_H

After the SJMP instruction (located at $1280/81_H$) to the MOVC instruction, the latter performs a read access to address 1282_H. This read access should fetch code $E4_H$ from the Flash Array, but instead it fetches 80_H from the even cache which has not been updated with new byte code but rather still contains the data byte from the last read access on an even address. The CPU receives the wrong codes in the sequence 80_H 93_H 80_H. The correct code sequence should be 80_H 93_H $E4_H$. The MOVC data access returns wrong readout. Hence, this is an example of a wrong data readout.

Scenario 3

1. From anywhere outside P-Flash (eg. XRAM or D-Flash), perform data access to odd address in P-Flash.
2. From anywhere outside P-Flash (eg. XRAM or D-Flash), perform data access to odd - 1 address in P-Flash.

For example:

(the following code is executing out of D-Flash or XRAM)

Table 9

movc a,@a+dptr	;; a+dptr = any odd address (X) in P-Flash
mov r1,a	

Table 9

```
dec a
movc a,@a+dptr          ;; a+dptr = even address (X-1) in P-Flash
```

Workaround #1 for scenarios 1, 2 and 3

The parallel read feature can be disabled before performing the data access on P-Flash described in the above three scenarios. The feature can then be enabled again after the data access. Disabling the parallel access to P-Flash does not have any adverse impact on the code execution other than a longer execution time.

Parallel access can be disabled and enabled via the Flash In-Application Programming routines, PARALLEL_READ_DISABLE and PARALLEL_READ_ENABLE, which are located at 0xDFFC and 0xDFFF respectively.

For example, the workaround for scenario 3 will be:

(the following code is executing out of D-Flash or XRAM)

Table 10

```
LCALL 0xDFFC            ;; disable parallel access
mov a,#1
movc a,@a+dptr          ;; a+dptr = any odd address (X) in P-Flash
mov r1,a
mov a,#0
movc a,@a+dptr          ;; a+dptr = even address (X-1) in P-Flash
LCALL 0xDFFF            ;; enable parallel access
```

Workaround #2 for scenarios 1 and 2

A dummy NOP instruction can be added before and after any MOVC instruction (applies to both movc a,@a+dptr, movc a,@a+pc). The NOP instructions act as dummy instructions and when executed or skipped, does not have any adverse impact on the code execution other than a longer execution time.

For example, the workaround for scenario 1 will be:

Table 11

Address	Code		
$127F_H$	04	inc a	
1280_H	00	nop	;; dummy
1281_H	93	movc a, @a+dptr	;; a+dptr = 1283_H
1282_H	00	nop	;; dummy
1282_H	14	dec a	
1283_H	E4	clr a	

which results in the execution sequence 04_H **00_H** 93_H $E4_H$ **00_H** 14_H $E4_H$.

For example, the workaround for scenario 2 will be:

Table 12

Address	Code		
1280_H	80	sjmp label	
1281_H	01		
1282_H	E4	clr a	
1283_H	00	nop	;; dummy
1284_H	93	label: movc a, @a+dptr	;; a+dptr = 1282_H
1285_H	00	nop	;; dummy

which results in the execution sequence 80_H 93_H $E4_H$ **00_H**.

Workaround #3 for scenarios 1 and 2

Any MOVC instruction (applies to both movc a,@a+dptr, movc a,@a+pc) must be located in the even byte address. This can be achieved by forcing the alignment in the compiler, or by adding one or more instructions (e.g. NOP) to push the MOVC instruction into an even byte address.

For example, the workaround for scenario 1 with a NOP instruction inserted will be:

Table 13

Address	Code		
1280$_H$	04	inc a	
1281$_H$	00	nop	;; dummy
1282$_H$	93	movc a, @a+dptr	;; a+dptr = (1283+1)$_H$
1283$_H$	14	dec a	
1284$_H$	E4	clr a	

which results in the execution sequence 04$_H$ **00$_H$** 93$_H$ E4$_H$ **14$_H$ E4$_H$**.

For example, the workaround for scenario 2 with a NOP instruction inserted will be:

Table 14

Address	Code		
1280$_H$	80	sjmp label	
1281$_H$	01		
1282$_H$	E4	clr a	
1283$_H$	00	nop	;; dummy
1284$_H$	93	label: movc a, @a+dptr	;; a+dptr = 1282$_H$

which results in the execution sequence 80$_H$ 93$_H$ E4$_H$.

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

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.
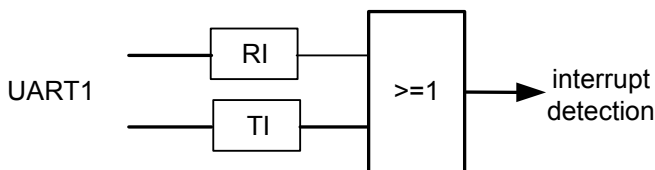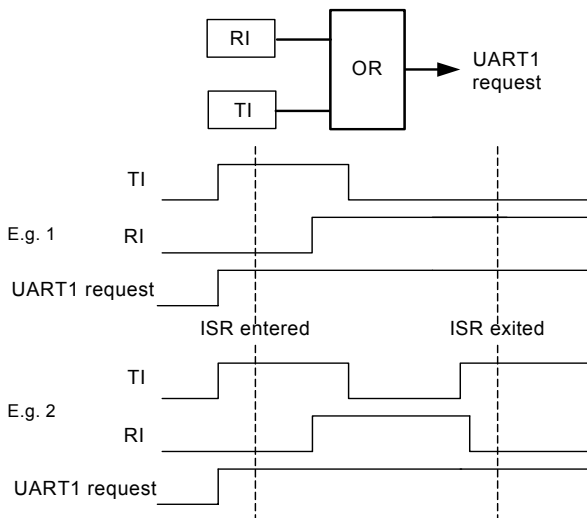
Figure 3

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 4

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 5

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).

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 2 possible sources: P0.0 and P2.0. Unexpected glitches may occur when the clock source is switched from one source to the other.

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

Workaround

To prevent the occurrence of such glitches during the clock switching, the user has to configure both pins as input and drive them to either all zero or all one before switching the clock source from one pin to another.

PIN_XC8.006 Port 4 pads (excluding P4.2) toggle once upon a power-on or hardware reset

All Port 4 pads, excluding P4.2, are stated to have Hi-Z as the reset state. However, these pads are pulled up internally when a power-on or hardware reset is asserted. They are returned to the Hi-Z state with internal pull-ups disabled only after the reset is deasserted. The result is that these pads toggle once upon a power-on or hardware reset.

Workaround

There are two proposed workarounds:

1. Implement external pulls on the external circuitry to prevent the toggle of the Port 4 pins. In case low level is required, strong external pull-downs (in the range of 2 kΩ to 3 kΩ) must be used.
2. Use alternate pins of the same function from the other ports. For example, for motor control applications which make use of the CCU6 output pins, Port 3 pins can be used instead.

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.

3 Deviations from Electrical- and Timing Specification

4 Application Hints

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.

FLASH_XC8.H002 Programming to invalid P-Flash address range 5000_H to $5FFF_H$ may corrupt existing P-Flash data

Writing to the unavailable program memory addresses from 5000_H to $5FFF_H$ can lead to corruption of existing data in the valid P-Flash addresses from 4000_H to $4FFF_H$. The user has to make sure not to write to the addresses from 5000_H to $5FFF_H$.

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.
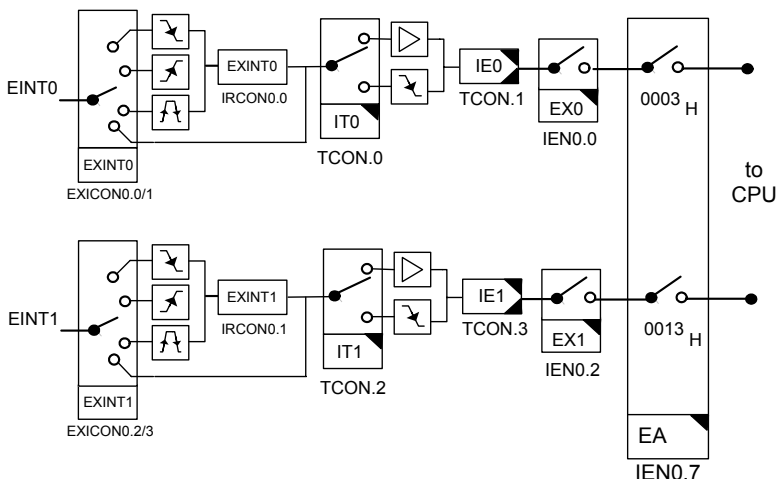
Figure 6

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:
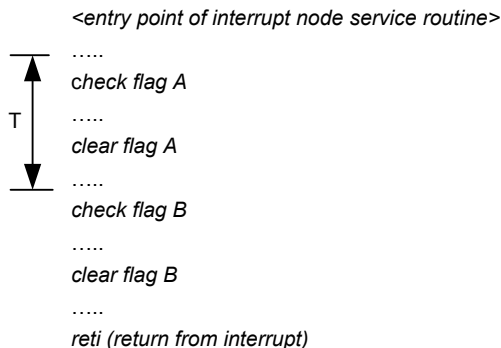
<entry point of interrupt node service routine>

…..
check flag A
…..
clear flag A
…..
check flag B
…..
clear flag B
…..
reti (return from interrupt)

T

Figure 7

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 15

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

Table 15

Interrupt Node	Vector Address	Assignment	Unqualified Flags Belonging to Node
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:

<entry point of interrupt node service routine>

.....

check flag A

.....

clear flag A

.....

check flag B

.....

clear flag B

.....

reti (return from interrupt)

T
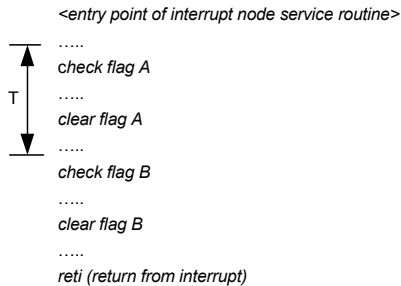
Figure 8

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 EOFSYN and ERRSYN bits status after exit from Boot ROM

When the device exits from Boot ROM and starts user code execution (either in XRAM or flash memory), the bitfields EOFSYN and ERRSYN in SFR FDCON remain in their last accessed states. Therefore these bitfields must be initialized according to the application requirement by the user code.

The following are the possible application scenarios that could lead to the above:

1) When any LIN BSL supported modes is activated before the device starts to execute user code (either in XRAM or flash memory).

2) After the device is reset and remain in the LIN BSL, if there is any error frame detected in the LIN bus, the user code in the flash memory will be executed.

User code should initialize the register properly according to the application requirement.

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.