Errata Sheet

V1.1, 2007-06-21

Device	**XC167CI-16F**
Marking/Step	**(E)ES-BB, BB**
Package	**PG-TQFP-144-7, P-TQFP-144-19**

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 applies to all temperature (SAB-/SAF-/SAK-.....) and frequency versions (.20./.40.), unless explicitly noted otherwise.

Versions beginning from V1.0 apply to the current documentation rules.

Current Documentation

- **XC167CI-16F Data Sheet V1.3, 2006-08**
- **XC167-16 User's Manual V2.0, Volume 1: System Units, 2004-04**
- **XC167-16 User's Manual V2.0, Volume 2: Peripheral Units, 2004-04**
- **C166S V2 User's Manual (Core, Instruction Set) V1.7, 2001-01**

(for viewing, click on weblinks above and follow link to “Documents”)

Note: Devices additionally marked with EES- or ES- or E followed by a 3-digit date code 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.

Note: For simplicity all versions are referred to by the term XC167 throughout this document.

Contents

Section . . . . . Page
History List/Change Summary 2
Functional Problems 7
OCDS and OCE Modules 29
Deviations from Electrical- and Timing Specification 31
Application Hints 32
Documentation Update 49

1 History List/Change Summary

(from step AD, previous errata sheet: V1.0, step: ES-BB, BB)

Table 1 Functional Deviations

Functional Problem	Short Description	Fixed in step	Change
EBC_X.003	TwinCAN access with EBC enabled		
EBC_X.006	Visibility of internal LXBus cycles on external address bus		new
CPU_X.002	Branch to wrong target after mispredicted JMPI		
CPU_X.003	Software Modification of Return Address on System Stack in combination with RET/RETP/RETS instructions		new
INT_X.007	Interrupt using a Local Register Bank during execution of IDLE		new
INT_X.008	HW Trap during Context Switch in Routine using a Local Bank		new
INT_X.009	Delayed Interrupt Service of Requests using a Global Bank		new
PORTS_X.012.1	Interference of Input Signals on P9.2 and P9.4 with internal Flash		new
SCU_X.008.1	Modification of register PLLCON while CPSYS = 1	(E)ES-BA	
SCU_X.009.1	Software or Watchdog Timer Reset while Oscillator is not locked	(E)ES-BA	
SCU_X.010	Reset while PLL is not locked - step (E)ES-BA only	ES+BA BA	
SCU_X.011	Register Security Mechanism after Write Access in Secured Mode		
TwinCAN2.005	Double Send	(E)ES-BA	
TwinCAN2.006	CPUUPD fast set/reset	(E)ES-BA	
TwinCAN_AI.007	Transmit after error (former name: TwinCAN_X.007)		re-named

Table 1 Functional Deviations (cont'd)

Functional Problem	Short Description	Fixed in step	Change
TwinCAN_AI.008	Double remote request (former name: TwinCAN_X.008)		re-named
TwinCAN_AI.009	CPUUPD remote (former name: TwinCAN_X.009		re-named
TwinCAN_AI.010	Reserved Bits in Register MSGARHn[15:13]		new
FLASH_X.004	PACER trap after wake-up from Sleep/Idle mode		
FLASH_X.005.1	Erase of Logical Sector 1, 2, 3, 4		new
ASC_X.001	ASC Autobaud Detection in 8-bit Modes with Parity		
ADC_X.003	Coincidence of Result Read and End of next Conversion or Start of Injected Conversion in Wait for Read Mode	(E)ES-BA	
ADC_X.004	ADC Overrun Error Generation when result is read during last cycle of conversion	(E)ES-BA	
CC6_X.002	T12 Shadow Transfer for Phase Delay Function	(E)ES-BA	
SLEEP_X.001	Wake up trigger during last clock cycle before entry into sleep mode	(E)ES-BA	
GPT12E_X.001	T5/T6 in Counter Mode with BPS2 = $1X_B$		new

Table 2 OCDS and OCE Modules

Functional Problem	Short Description	Fixed in step	Change
OCDS_X.001	$\overline{\text{BRKOUT}}$ pulsing after Instruction Pointer Debug Event	(E)ES-BA	
OCDS_X.002	OCDS indicates incorrect status after break_now requests if PSW.ILVL $\geq$ CMCTR.LEVEL		
OCE_X.001	Wrong MAC Flags are declared valid at Core - OCE interface		

Table 3 AC/DC Deviations

AC/DC Deviations	Short Description	Fixed in step	Change
TAP_X.85	Maximum ambient temperature during flash programming 85°C	(E)ES-BA	
FCPUR_X.1632	Frequency Limits for Flash Read Accesses - SAK temperature version only !	BB	
M40_WLE_X.001	Minimum Ambient Temperature for Flash Wordline Erase Command	BB	

Table 4 Application Hints

Hint	Short Description	Change
CPU_X.H1	Configuration of Registers CPUCON1 and CPUCON2	
CPU_X.H2	Special Characteristics of I/O Areas	
FLASH_X.H1.1	Access to Flash Module after Program/Erase	
FLASH_X.H2.2	Access to Flash Module after Shut-Down	
FLASH_X.H3.2	Read Access to internal Flash Module with modified Margin Level	
FLASH_X.H4	Minimum active time after wake-up from sleep or idle mode	
SLEEP_X.H3.2	Clock system after wake-up from Sleep Mode	
IDLE_X.H1	Entering Idle Mode after Flash Program/Erase	
ADC_X.H1	Polling of Bit ADBSY	
BREAK_X.H1	Break on MUL/DIV followed by zero-cycle jump	
OCDS_X.H001	Effect of Bit OCDSIOEN in Register EMUCON	new
IIC_X.H1	Maximum IIC Bus Data Rate at low fCPU	
IIC_X.H2	Timing of Bit IRQD	
POWER_X.H1.1	Initialization of SYSCON3 for Power Saving Modes	
POWER_X.H2.2	Power Consumption during Clock System Configuration	
RSTOUT_X.H1	RSTOUT# driven by weak driver during HW Reset	
SCU_X.H1	Shutdown handshake by software reset (SRST) instruction	

Table 4 Application Hints (cont'd)

Hint	Short Description	Change
SCU_X.H3	Effect of PLLODIV on Duty Cycle of CLKOUT	
SCU_X.H4	Changing PLLCON in Emergency Mode	
SCU_X.H5	Sleep/Idle/Power Down Mode not entered while PLLODIV = 0Fh	
SCU_X.H6	Interrupt request during entry into sleep mode	
SCU_X.H7	VCO Configuration with Input Clock disconnected	new
SCU_X.H8	PLL Bypass Mode with VCO on	new
SCU_X.H10	Register Security Mechanism usage with CPUCON1 and CPUCON2 registers	new
RTC_X.H1.1	Resetting and Disabling of the Real Time Clock	
FOCON_X.H1	Read Access to register FOCON	
OSC_X.H001	Signal Amplitude on Pin XTAL1	new
TwinCAN_AI.H2	Reading Bitfield INTID	
INT_X.H1	Software Modifications of Interrupt Enable (xx_IE) or Interrupt Request (xx_IR) Flags	new
INT_X.H002	Increased Latency for Hardware Traps	new

Table 5 Documentation Update

Name	Short Description	Change
INT_X.D1	Interrupt Vector Location of CAPCOM Register 28	
SCU_X.D1.1	Oscillator Gain Reduction	
SCU_X.D2.2	Functionality of register OPSEN	
SCU_X.D3	Register PLLCON after software reset	
SCU_X.D5	VCO band after hardware/watchdog reset in single chip mode	
SCU_X.D6	Register Security Mechanism - Unprotected Mode active until execution of EINIT instruction	
FLASH_X.D1	Interaction between Program Flash and Security Sector Programming	
PORTS_X.D2_144	Internal Pull-up Device active on P3.12 during Reset	

Table 5 Documentation Update (cont'd)

Name	Short Description	Change
TwinCAN_AI.D1	Reserved Bits in Registers of the TwinCAN Module	new
CPU_X.D1	Write Operations to Control Registers	new
ADC_X.D3	ADC Sample Time with Improved/Enhanced Timing Control	new
ID-Registers	ID-Registers	

2 Functional Problems

EBC_X.003 TwinCAN Access with EBC enabled

If the External Bus Controller (EBC) is enabled, a read or write access to the TwinCAN module fails when an external bus access with TCONCSx.PHA ≠ 00_B precedes the TwinCAN access.

Workaround:

Since it is hard to predict the order of external bus and TwinCAN accesses (in particular when PEC transfers are involved), it is recommended to set bitfield PHA to 00_B in all TCONCSx registers which are used for external bus accesses.

EBC_X.006 Visibility of internal LXBus cycles on external address bus

When an external bus is enabled, accesses to internal LXBus modules are to some extent reflected on those parts of the external address bus that is not multiplexed, i.e.

- on those pins of Port 4 that are configured as segment address lines,
- and on the pins of PORT1 (P1L, P1H) if a non-multiplexed external bus mode is enabled in one of the FCONCSx registers.

The effect is as follows

a) After an LXBus access (e.g. to the TwinCAN module), the address of the location accessed last on the LXBus remains visible on the non-multiplexed part of the external address bus, unless an external bus cycle immediately follows the LXBus cycle.

b) During an internal LXBus access, the external address bus is tristated. Due to an internal race condition, the pads on the non-multiplexed part of the external address bus may already start to drive the LXBus address (for ~ 3 ns) before they are tristated. This may result in undefined logic levels on the non-multiplexed part of the external address bus during an internal LXBus cycle.

However, no external chip select signal $\overline{CS}$x or ALE/$\overline{RD}$/$\overline{WR}$ strobe is active during an internal LXBus cycle.

Workaround:

Not needed if external circuitry does not evaluate levels on external address bus during internal LXBus cycles.

CPU_X.002 Branch to wrong target after mispredicted JMPI

After a JMPI is initially mispredicted according to the static branch prediction scheme of the C166S V2, code execution may continue at a wrong target address in the following situations:

Situation I:

- a memory write operation is processed by the DMU
- followed by a MUL(U)
- followed by the mispredicted JMPI

```
Example_1:
MOV mem, [Rwn]
MUL R13, R14
JMPI cc_NV, [R6]
```

Situation II:

- MUL(U) or DIV(L/U/LU)
- followed by not-mispredicted zero-cycle jump
  (e.g. JMPA, JMPR, JMPS; bit ZCJ (CPUCON1.0) = 1)
- followed by the mispredicted JMPI

```
Example_2a:
MULU R13, R14
JMPA- cc_V, _some_target  ; predicated not taken => correct
JMPI cc_NV, [R6]          ; taken, but predicted not taken
```

It could be possible that the JMPI is at the jump target of the JMPA, if it is taken:

```
Example_2b:
MULU R13, R14
JMPA+ cc_NZ, _jmpi_addr   ; predicted taken => correct
..... other code ....
_jmpi_addr: JMPI cc_NV, [R6] ; taken but predicted not taken
```

Effect on tools:

In the **Altium/Tasking** compiler (V7.0 and above) the problem is not present. The result of a MUL/DIV instruction is available through the MDL/MDH SFRs. These SFRs are not

allocatable by the register allocator. Therefore, the compiler always needs a MOV instruction to transfer MDL/H to a GPR. This avoids the problem.

In the RT- and FP-libraries (v7.0 and above) the problem was not found. Versions lower than v7.0 do not explicitly support the C166S V2 core.

In case optimizations are implemented in future versions which could cause this problem to occur, also a workaround will be included.

All **Keil** C166 tool Versions (compiler and libraries) since V3.xx do not generate a MUL(U) or a DIV(L/U/LU) followed by either of the jump instructions JMPR, JMPS, JMPA, JMPI. Basically the support of the C166S V2 core requires anyway V4.21 or higher.

Workarounds:

Examples for program parts written in assembly language:

- generally disable overrun of pipeline bubbles by clearing bit CPUCON2.OVRUN (CPUCON2.4 = 0). This will result only in a negligible performance decrease, and will prohibit corruption of the target IP of the JMPI.

or:

- provide a NOP (or any other suitable instruction) between the MUL/DIV instructions and the succeeding jump in the above cases. To simplify, place a NOP between any MUL/DIV and a JMPR, JMPS, JMPA, JMPI that might follow it. Other branches (CALLs, jump-on-bit instructions) do not need to be taken into account.

CPU_X.003 Software Modification of Return Address on System Stack in combination with RET/RETP/RETS instructions

Introduction:

Upon subroutine calls (instructions CALLA/R/I/S, PCALL) or interrupts/traps, the return address and status information is saved on the system stack, including:

- IP: intra-segment address of the next, not executed instruction
- CSP: segment number of the next, not executed instruction (for inter-segment CALLS instructions and interrupt/traps only)
- PSW: processor status word for next, not executed instruction (for interrupt/traps only)

When the corresponding return instruction (RET, RETS, RETP, RETI) is executed, program execution normally continues - at the return address popped from the system stack - with the next instruction, i.e. the instruction linearly following the subroutine call, or the instruction following the one after which the interrupt/trap occurred.

In case the return address on the system stack has been modified by software, program execution will continue at this address - instead of the address that was originally pushed on the system stack - when the next return instruction is executed. This method might be used e.g. by operating systems, or by compilers in conjunction with specific options (e.g. stack models), to perform branches within or between code segments. Two methods basically exist for manipulations of the return address:

- **M1**: the return address contained in the last stack frame is directly modified:
  [SP]+4: CSP_o --> CSP_m ; CSP value originally pushed is modified to CSP_m[SP]+2: IP_o --> IP_m ; IP value originally pushed is modified to IP_m [SP]: <Top of System Stack>
- **M2**: an additional frame with a new return (branch target) address is pushed onto the system stack. The original return address is still available on the system stack:
  [SP]+4: CSP_o ; CSP value originally pushed on system stack by last CALLS
  [SP]+2: IP_o ; IP value originally pushed on system stack by last CALL(S)
  [SP]: CSP_m ; CSP value additionally pushed on system stack
  [SP]-2: IP_o ; IP value additionally pushed on system stack
  [SP]: <Top of System Stack>

In order to optimize the execution speed of return instructions, the C166S V2 core additionally stores (mirrors) the 24-bit return address information (CSP_r:IP_r, r=1..6) of the last up to 6 subroutine calls (invoked by CALLA/R/I/S, PCALL) in a 'last in/first out' Return Stack (see User's Manual, IFU Block Diagram). Whenever one of the return instructions RET, RETS, RETP is executed, the information from the top of the Return Stack is used to already start fetching the return target instruction. When the actual return address from the system stack is available, it is compared with the information from the Return Stack. In case of a mismatch, the instruction that was 'speculatively' read with the address information obtained from the Return Stack is cancelled, and a new fetch with the address from the system stack is started.

The Return Stack (including the potential performance increase) is not used in combination with interrupts/traps and the corresponding RETI instruction.

The Return Stack can only provide a performance increase when identical return addresses are delivered by the Return Stack and by the system stack. This is only the case when the system stack is not manipulated by software, i.e.

- the return address on the system stack is not directly modified by software (method M1)
- no additional 'return' addresses are pushed on the system stack by software to perform special branches (method M2), or RETI is used (after pushing PSW) to return from a subroutine call. Otherwise, Return Stack and system stack are no longer aligned, and the comparison of the return addresses will also fail for all remaining Return Stack entries (r=1..5).

Problem Description:

When the return address on the system stack, composed of CSP_m and/or IP_m, is modified by software, and the Return Stack is not empty, care must be taken that a valid instruction is also located at

1. address CSP_{rtop}:IP_{rtop} when the next return instruction is RETS
2. address CSP_C:IP_{rtop} when the next return instruction is RET or RETP

where:

- CSP_{rtop} = CSP value on top of Return Stack
  = CSP value originally pushed on system stack by last call instruction if no other return instruction has been executed, else
  = CSP value originally pushed on system stack by `last-nth` call instruction if `n` other return instructions have been executed, and no other call instruction has been executed
- IP_{rtop} = IP value on top of Return Stack
  = IP value originally pushed on system stack by last call instruction if no other return instruction has been executed, else
  = IP value originally pushed on system stack by `last-nth` call instruction if `n` other return instructions have been executed, and no other call instruction has been executed
- CSP_C = current contents of CSP
- valid instruction = code, or data decoded as instruction, that does not access memory areas that will generate a PACER trap (e.g. reserved memory areas, disabled flash, etc.)
- Return Stack empty = e.g. number of executed return instructions ≥ number of call instructions, or 6 consecutive return instructions are executed.

Otherwise, when no valid instruction is located at the address described above, a PACER trap could be generated (due to a 'speculative' access) instead of branching to the modified return address from the system stack.

Examples:

Case 1 (see 1. in Problem Description above):

Usually, a valid instruction will be located at address CSP_{rtop}:IP_{rtop}, i.e. at the location after the original CALL instruction. An exception is the case where the CALL instruction is the last instruction in a code section that is terminated with RETV (virtual return, no code generated), and a data section directly follows this code section, e.g.:

```
  ...
  CALLS f_no_ret ; function modifying system stack, does
                 ; not return to following instruction
  RETV           ; virtual return (no code generated)
_proc_xy ENDP          ; end of procedure xy
```

```
_sec_c_xy ENDS          ; end of code section xy
_sec_d_ab SECTION DATA ; begin of data section ab
_ab_array LABEL WORD ; next location used by locator
                       ; this address is pushed on system
                       ; stack by CALLS
  DW 0FFFAh ; data located here, decoded as instruction,
  DW 0000h  ; must result in a valid instruction
            ; FA FF 00 00 is decoded as JMPS 0FFh, 0000h
            ; ==> access to FF:0000 will cause PACER trap
            ; when function f_no_ret executes RETS
  ...
```

In the above example, a PACER trap will be generated after the RETS instruction (corresponding to the CALLS instruction) at the end of function `f_no_ret` is executed: the speculatively prefetched instruction JMPS 0FFh, 0000h is trying to access a reserved memory area.

Case 2 (see 2. in Problem Description above):

When a RET or RETP instruction is executed after a modification of the system stack, an intra-segment branch is performed. If the contents of the CSP register has not changed since the last CALL instruction (e.g. no JMPS/CALLS or interrupt has occurred), then $CSP_C = CSP_{rtop}$, which is the same situation as already discussed in Case 1 before.

In case the last function was e.g. called with a CALLS instruction (inter-segment call), the current contents CSP_C of the Code Segment Pointer CSP is different from the CSP value that has been originally pushed on the stack by the CALLS instruction, e.g.:

```
Location: Instruction
C1'9876:  CALLS C0, 3456h ; inter-segment call,
                          ; return addr CSP=C1, IP=987A
                          ; pushed on system stack by HW
...
C0'3456:  MOV Rx, #5678h  ; prepare intra-segment branch
C0'345A:  PUSH Rx         ; push 'return' addr IPm=5678
                          ; on system stack by SW
C0'345C:  RET             ; branch (=return) to 'target'
...                       ; addr IPm=5678 with CSPc=C0
C0'5678:  <target_instr>  ; continue at branch target
...
C0'987A:  <valid_instr>   ; code or data located here must
                          ; result in a valid instruction
...
```

In the above example, if the instruction (fragment) or data located at address CSP_C:IP_m (here: C0'987A) - when decoded as instruction - does not result in a valid instruction, a PACER trap will be generated after the RET instruction is executed.

Workaround 1

Do not perform modifications of the return address on the system stack.

Workaround 2 (when stack modification method M1 is used)

To avoid generation of the (unjustified) PACER trap,

- use RETS to branch to the modified return address CSP_m:IP_m after manipulation of the system stack. In case the last subroutine call was only an intra-segment CALL, extend the last stack frame with CSP_m, **and**
- use a real return instruction (e.g. RETS) instead of the virtual RETV when the last executed call instruction was the last instruction of a code section followed by a data section. This ensures that a valid instruction is located after the original CALL/S instruction.

Workaround 3 (when stack modification method M2 is used)

To avoid generation of the (unjustified) PACER trap,

- use RETI to branch to the modified return address CSP_m:IP_m after manipulation of the system stack. RETI does not affect the Return Stack. In this case, the PSW must be pushed on the system stack in addition to the components of the modified return address CSP_m and IP_m, e.g.:

```
PUSH PSW ; push current contents of PSW
PUSH Rx  ; push return segment address CSPm
PUSH Ry  ; push return address IPm
RETI     ; branch (=return) to CSPm:IPm
```

Workaround 4

After manipulation of the return address on the system stack, perform a sequence of 6 CALL instructions, followed by 6 return instructions. This will clear the Return Stack. Then, perform the return instruction that branches to the modified return address.

Workaround 5

If the modifications of the return instructions discussed in Workaround 2 .. 4 can not be performed (e.g. because the code is part of library functions or an operating system), the PACER trap service routine should be modified. To distinguish the 'unjustified' PACER trap from other defined conditions for a PACER trap, e.g.

- first, check bit DBER in register FSR for flash double bit errors

- else, perform a sequence of six CALL instructions, followed by 6 return instructions to clear the Return Stack, and then return from the PACER trap handler, e.g.

```
BCLR PACER              ; clear PACER trap flag
CALLS clear_ret_stack1 ; clear Return Stack
RETI                    ; return from PACER trap handler
...
clear_ret_stack1:
    CALLS clear_ret_stack2
    RETS
...
clear_ret_stack5:
    CALLS clear_ret_stack6
    RETS
clear_ret_stack6:
    RETS
```

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

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

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

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

Workaround 1

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

```
BCLR IEN   ; Disable interrupts globally
IDLE       ; Enter idle or sleep mode
BSET IEN   ; After wake-up: re-enable interrupts
```

If an interrupt request is generated during this sequence, idle/sleep mode is terminated and the interrupt is acknowledged after BSET IEN.

Workaround 2

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

Workaround 3

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

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

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

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

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

Workaround 1

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

Workaround 2

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

Workaround 3

Condition b) (writing to CP while a local register bank context is selected) is not typical for most applications. If the application implementation already eliminates the possibility for condition b), then only a workaround for condition a) is required.
The workaround for condition a) is to make sure that the IDLE instruction is executed within a code sequence that uses a global register bank context.

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

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

Case 1:

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

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

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

Case 2 (see also Figure 1):

- The Context Pointer CP is written to (e.g. by POP, MOV, SCXT .. instructions) within a routine (Task A) that uses a **global** register bank (bit field PSW.BANK = 00_B), i.e. the context for this routine will be modified,
- This context switch procedure (19 cycles) is interrupted by an interrupt request (Task B) which is programmed (with GPRSELx = $1X_B$) to automatically use one of the **local** banks via the bank selection registers BNKSEL0..3 (i.e. the interrupting routine has a priority level ≥12),
- Before the corresponding interrupt service routine is finished, another interrupt request (Task C) occurs which is programmed (with GPRSELx = 00_B) to automatically use the **global** bank via the bank selection registers BNKSEL0..3 (i.e. the interrupting routine has a priority level ≥13)

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

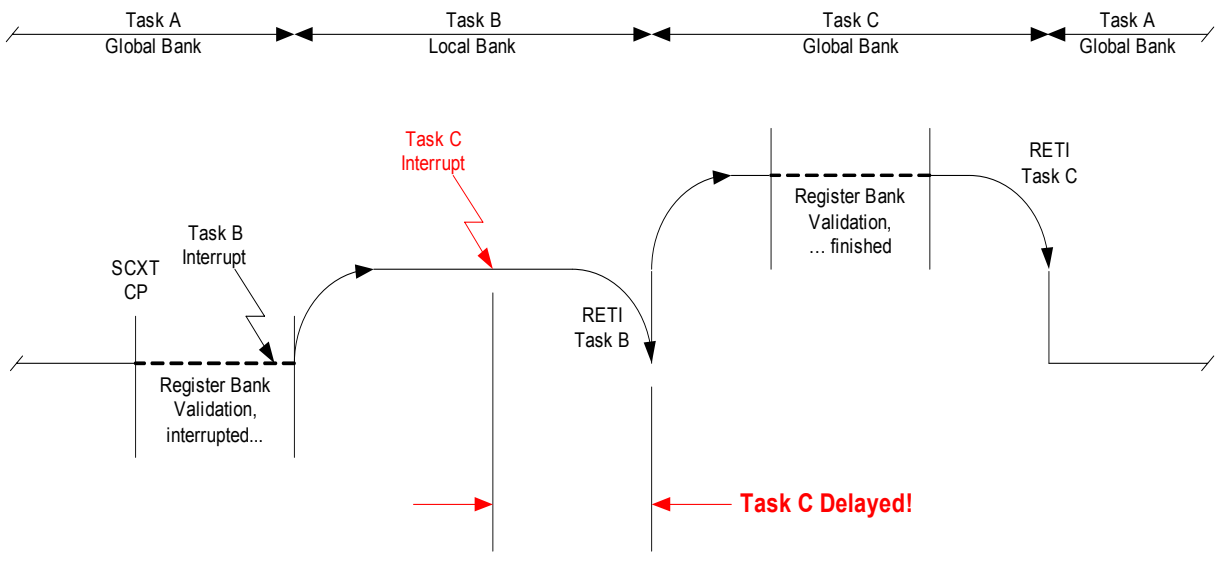


Figure 1 Example for Case 2: Interrupt Service for Task C delayed

Workaround for Case 1

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

Workaround for Case 2

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

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

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

PORTS_X.012.1 Interference of Input Signals on P9.2 and P9.4 with internal Flash

As specified in table "DC Characteristics" of the Data Sheet, input signals may exceed the positive or negative supply voltages V_{SSP} or V_{DDP} by up to 0.5V, while during Overload Conditions (Vin > V_{DDP}+0.5V or Vin < V_{SSP}-0.5V), the (injected) overload current must be limited to 5mA per pin/50 mA per device (see section "Operating Conditions").

However, two pins (**P9.2/CC18IO/CAN1_RxD** and **P9.4/CC20IO**) have been identified that under exceptional conditions (e.g. massive low frequency noise) may cause side effects on internal flash (read or erase) operations before these overload limits are reached or exceeded, depending on the amplitude and width of the respective input signal:

No problem will occur

- when the input levels on pin **P9.2/CC18IO/CAN1_RxD** do not exceed the positive or negative supply voltages V_{SSP} or V_{DDP} as listed in the table below.
  Otherwise, incorrect data may be read from the internal flash module, or program or erase operations may fail if this condition is present while the respective operation is in progress.

- when the input low level on pin **P9.4/CC20IO** does not exceed the negative supply voltage V_{SSP} as listed in the table below.
  Otherwise, an erase operation of the internal flash module may fail if this condition is present while an erase operation is in progress.

Conditions for Amplitude and Width of Input Signal:

No problem will occur for signals on pin **P9.2/CC18IO/CAN1_RxD** with the following characteristics:

- for low or high pulses with an amplitude Vin < V_{DDP}+0.5V or Vin > V_{SSP}-0.5V **and** pulse width < 100 ns **and** duty cycle DC < 0.33 (e.g. high pulses < 100 ns with this amplitude are tolerated if the following low phase is > 200 ns)
- for steady state signals or pulses > 100 ns with an amplitude Vin < V_{DDP}+0.2V or Vin > V_{SSP}-0.2V

No problem will occur for signals on pin **P9.4/CC20IO** with the following characteristics:

- for low pulses with an amplitude Vin > V_{SSP}-0.5V **and** pulse width < 1 µs **and** duty cycle DC < 0.33 (e.g. low pulses < 1 µs with this amplitude are tolerated if the following high phase is > 2 µs)
- for steady state signals or low pulses > 1 µs with an amplitude Vin > V_{SSP}-0.3V.

Typical overshoot/undershoot after signal transitions is normally uncritical (covered by duty cycle specification, see notes [1)] and [2)] below).

Table 6 DC Characteristics P9.2 and P9.4 (Operating Conditions apply)

Parameter	Symbol		Limit Values		Unit	Notes
			Min.	Max.		
Input low voltage on P9.2	V_{IL}, V_{ILS}	SR	-0.2 instead of -0.5	not affected	V	
Input low voltage on P9.2	V_{IL}, V_{ILS}	SR	-0.5	not affected	V	for pulses with T_{low} < 100 ns and DC < 0.33[1)]
Input high voltage on P9.2	V_{IH}, V_{IHS}	SR	not affected	V_{DDP} + 0.2	V	
Input high voltage on P9.2	V_{IH}, V_{IHS}	SR	not affected	V_{DDP} + 0.5	V	for pulses with T_{high} < 100 ns and DC < 0.33[2)]
Input low voltage on P9.4	V_{IL}, V_{ILS}	SR	-0.3 instead of -0.5	not affected	V	
Input low voltage on P9.4	V_{IL}, V_{ILS}	SR	-0.5	not affected	V	for pulses with T_{low} < 1 µs and DC < 0.33[1)]

1) For low pulses, DC = $T_{low}/(T_{high}+T_{low})$ must be < 0.33. This means e.g. low pulses < 100 ns are tolerated if the following high phase is > 200 ns.

2) For high pulses, DC = $T_{high}/(T_{high}+T_{low})$ must be < 0.33. This means e.g. high pulses < 100 ns are tolerated if the following low phase is > 200 ns.

Using these pins as outputs is considered uncritical because the output signal is tightly coupled to the internal supply voltages V_{SSP} or V_{DDP}. If unused, these pins should be switched to output by software. If possible, these pins should be left floating after reset instead of being pulled high or low by an external pull device.

If used as inputs, a series resistor will further reduce (but can not under all circumstances fully exclude) the risk in case the input levels described above are exceeded. Furthermore, the risk resulting from input signals exceeding these levels is reduced if the device is not operated at the upper limits of supply voltage, temperature, and frequency at the same time.

SCU_X.011 Register Security Mechanism after Write Access in Secured Mode

To modify an SFR that is protected by the register security mechanism, a certain security level has to be selected and/or a command sequence has to be executed prior to the write access to one of these registers. Table 6-15 in the User's Manual, volume System Units, lists all registers protected by the security mechanism (see copy of Table 6-15 below).

After selecting Secured Mode (bitfield SL = 01_B in register SCUSLS), a single command (command4) enables one single write access to a protected register. After this write access the protected registers are locked again automatically.

Exception:

After modification of registers CPUCON1, CPUCON2, EBCMOD0, EBCMOD1, TCONCSx, FCONCSx, ADDRSELx (which are not part of the SCU), all registers listed in Table 6-15 are **not locked** until the next write access to an SCU register (i.e. a register which is different from the group CPUCON1 .. ADDRSELx).

Workaround:

In order to lock all registers again, after a write access to the non-SCU registers CPUCON1 .. ADDRSELx a "dummy" write access to an SCU register should be executed. It is therefore proposed to use e.g. the read-only register IDCHIP for this purpose. The registers of the identification control block also belong to the SCU, and a write access to these read-only registers re-enables secured mode:

Example:

```
MOV  R4, #2000H       ; value to be stored in register EBCMOD0
EXTR #1               ; Access sequence in secured mode
MOV  SCUSLC, #8E12H   ; Command4: current password = EDH
OR   EBCMOD0, R4      ; Access to EBCMOD enabled
                      ;  by preceding Command4
MOV  IDCHIP, ZEROS    ; dummy write to a read-only SCU register
                      ;  re-enables secured mode
```

Table 7 Registers Protected by the Security Mechanism

Register Name	Function	Loc.
RSTCON	Reset control	SCU
SYSCON0	General system control	SCU
SYSCON1	Power management	SCU
PLLCON	Clock generation control	SCU

Table 7 Registers Protected by the Security Mechanism (cont'd)

Register Name	Function	Loc.
SYSCON3	Peripheral management	SCU
FOCON	Peripheral management (CLKOUT/FOUT)	SCU
IMBCTR	Control of internal instruction memory block	SCU
OPSEN	Emulation control	SCU
EMUCON	Emulation control	SCU
WDTCON	Watchdog timer properties	SCU
EXICON	Ext. interrupt control	SCU
EXISEL0, EXISEL1	Ext. interrupt control	SCU
CPUCON1, CPUCON2	CPU configuration	CPU
EBCMOD0, EBCMOD1	EBC mode selection	EBC
TCONCSx	EBC timing configuration	EBC
FCONCSx	EBC function configuration	EBC
ADDRSELx	EBC address window configuration	EBC

TwinCAN_AI.007 Transmit after Error

During a CAN error, transmission may stop (after EOF or an error frame), until a successful reception or a write access to the TwinCAN module.

Detailed Description

In case of a CAN error and when there is no other activity on the CAN module (e.g. frame reception or frame transmission on the other CAN node or write access to any CAN register), the transmission of messages may stop, even if some transmit requests are still set.

The CAN module will start transmitting immediately, after a reception or a write access to the module.

Workarounds:

- Write periodically $FFFF_H$ to one of the MSGCTRx registers, as this value is having no effect on the register.
- In case writing to a CAN register shall be the exception, use the last error code (LEC) interrupt. This shall start writing to one of the MSGCTRx register $FFFF_H$, in case the LEC value is unequal to 0.

TwinCAN_AI.008 Double remote request

After the transmission of the remote request, TXRQ is not cleared in the receive object, if NEWDAT is set. As a consequence the remote request is transmitted once again.

Workaround:

Clear NEWDAT after the reception of a data frame.

TwinCAN_AI.009 CPUUPD remote

In case of a remote request to a standard message object which is chosen for transmission, a transmit of the data frame takes place, even if CPUUPD is currently set.

Detailed Description

If a transmit message object gets a remote request and there is no other message object with higher transmit priority pending for transmission, then the transmit object sends the data frame to answer the remote request, even if CPUUPD is set.

Workaround:

This workaround is only required in systems where remote requests are used.

To answer remote requests, the MMC bitfield in MSGFGCR has to be configured to a FIFO slave object instead of a standard message object for transmission. To reach this goal, the following settings for the corresponding message object are needed:

- bitfield MMC (MSGFGCRHn.10-8) = 011_B (FIFO functionality enabled (slave object))
- bitfield CANPTR (MSGFGCRHn.4-0) = n (the CAN Pointer shall reference itself, by referring to the message object number of this object)
- bit FD (MSGFGCRLn.13) = 0 (the CANPTR is updated after a correct reception)

TwinCAN_AI.010 Reserved Bits in Register MSGARHn[15:13]

The 3 reserved bits MSGARHn[15:13] in the Message Object n Arbitration Register may be erroneously loaded with non-zero values (i.e. different from their reset values) under an arbitration loss condition.

Workaround:

If the received identifier is checked by software, the software should be written in a way that these bits have no impact on the decision (e.g. by masking off the upper 3 bits).

FLASH_X.004 PACER Trap after Wake-Up from Sleep/Idle Mode

An unexpected Program Access Error Trap (flag PACER = 1 in register TFR) occurs after a wake-up event from sleep or idle mode under the following conditions:

- bit field PFCFG = 01_B in register SYSCON1, i.e. the flash module is switched off during sleep or idle mode
- a wake-up event (interrupt, PEC transfer, $\overline{\text{NMI}}$) occurs in a specific time window (few clock cycles) after execution of the IDLE instruction during the flash deactivation process
- and the corresponding interrupt/trap routine or the instruction following IDLE (in case interrupts are disabled) or the PEC source data are located in the internal flash.

Workaround 1:

Do not switch off the flash module in sleep or idle mode, i.e. leave bit field PFCFG = 00_B in register SYSCON1 (default after reset). This increases power consumption to a certain extent while reducing the time overhead for sleep/idle mode entry and exit (in clocking modes where the clock is derived from the VCO).

Workaround 2

In order to avoid the problem (when PFCFG = 01_B), make sure that the wake-up trigger only occurs after the device has completely entered sleep or idle mode.

If the RTC is used as wake-up source, check e.g. the RTC before entering sleep mode. If the wake-up trigger will occur soon, either skip entry into sleep mode, or extend the time for the next wake-up. If the RTC time interval is reprogrammed, make sure that no interrupt occurs between reprogramming and entry into sleep mode.

Workaround 3

In order to differentiate an unexpected PACER trap due to a wake-up trigger in the critical time window from other events that can lead to a PACER trap, set a semaphore bit before executing the IDLE instruction., e.g.

```
ATOMIC #2
BSET   sema_idle
IDLE
```

In the trap handler for the PACER trap, if the semaphore bit is set and no other indications for the PACER trap are found (e.g. error flags in register FSR are set), clear the semaphore bit and the PACER flag and return from the trap handler with RETI. The stack contains a valid return address in this case (e.g. address of instruction following IDLE in case interrupts were disabled during wake-up).

For **PEC transfers** during sleep or idle mode, which will cause a PACER trap if they read data from flash, automatic return to sleep/idle mode is not accomplished with the concept described so far.

- In order to support return to idle/sleep mode after a PEC transfer (which is performed after the RETI instruction from the PACER trap routine is executed), e.g. a semaphore bit may be used. This bit may be set to '1' before the IDLE instruction is executed. All trap (except PACER) and interrupt service routines invoked after wake up from idle/sleep should clear this bit to '0'. After having returned from the PACER trap routine to the program in the internal flash, this bit should be tested (allow a sufficient time of e.g. 12 cycles for interrupt arbitration), and if it is still at '1' (i.e. no interrupts/traps have occurred), repeat the IDLE instruction for re-entry into idle/sleep mode.

Workaround 4

Use an auxiliary sequence in internal PRAM that bridges the time until the flash is ready after wake-up from sleep/idle mode, e.g.:

- Disable interrupts, and execute the IDLE instruction to enter sleep mode from the internal PRAM. After wake-up, the instruction following IDLE will be executed (if no hardware trap or $\overline{\text{NMI}}$ has occurred).
- Wait until the internal flash is ready after wake-up (check register FSR) before reading from the internal flash). If the sequence in internal PRAM that includes the IDLE instruction is not CALLed from internal flash (i.e. it is not terminated with a RETx instruction), at least 8 instructions that do not read from the internal flash should be inserted after the IDLE instruction to avoid speculative prefetches
- Enable the interrupt system again.

The following details should be considered:

- If hardware traps (including $\overline{\text{NMI}}$) can occur, add the corresponding interrupt vectors to PRAM and modify register VECSEG to point to the PRAM space.
- In order to support return to idle/sleep mode after a PEC transfer, e.g. a semaphore bit may be used. This bit may be set to '1' before the IDLE instruction is executed. All trap and interrupt service routines invoked after wake up from idle/sleep should clear this bit to '0'. After having returned to the program in the internal flash and having enabled the interrupt system, this bit should be tested (allow a sufficient time of e.g. 12 cycles for interrupt arbitration), and if it is still at '1' (i.e. no interrupts/traps have occurred), repeat the auxiliary routine that prepares for re-entry into idle/sleep mode.

FLASH_X.005.1 Erase of Logical Sector 1, 2, 3, 4

When the last erase or program operation was performed on logical sector 5, a successive erase of logical sector 1, 2, 3, 4 may fail.

Typically, the problem is more likely to occur under the following external conditions:

- The voltage V_{DDI} is above its nominal value (i.e. > 2.5 V).
- The voltage V_{DDP} is below its nominal value (i.e. < 5.0 V).
- The ambient temperature is above room temperature (i.e. > 25°C).

However, it can **not** be excluded that this problem does not occur outside the temperature and voltage ranges listed above.

An erase operation of a wordline in a logical sector and an erase operation of logical sector 0 does always work. Thus these operations can be done under all conditions.

This also implies that an erase or erase-and-program sequence starting with logical sector 0 and increasing the logical sector number will always work.

Workaround 1:

Do the erase logical sector operation twice:

```
Example code (for sector 4 starting at C0’8000h):

MOV  R4 , #0080h        ; data for 1st erase command
MOV  R5 , #00AAh        ; data for 2nd erase command
MOV  R6 , #0033h        ; data for 3rd erase command

CALL waitBusy    ; before erase operation check if flash is busy
EXTS #0C0h  , #3        ; use segment 0C0h for next 3 operations
MOV  000AAh , R4        ; 1st erase command
MOV  00054h , R5        ; 2nd erase command
MOV  08000h , R6        ; 3rd erase command
                        ; for logical sector 4 at 08000h
CALL waitBusy           ; wait until erase operation is done

EXTS #0C0h  , #3        ; use segment 0C0h for next 3 operations
MOV  000AAh , R4        ; 1st erase command
MOV  00054h , R5        ; 2nd erase command
MOV  08000h , R6        ; 3rd erase command
                        ; for logical sector 4 at 08000h
CALL waitBusy
```

```
... further code

; subroutine to wait for busy signal
waitBusy:                ; check busy flag in FSR
     EXTS #0FFh, #1      ; use segment 0FFh for next operation
     MOV  R7   , 0F000h  ; read FSR
     JB   R7.0 , waitBusy; if flash still busy loop again
     RET
```

Workaround 2:

Program one page of the logical sector first and then erase the logical sector:

```
Example code (for sector 4 starting at C0’8000h):

CALL waitBusy                ; wait for flash ready

CALL enterPageMode           ; enter Page mode for programming
CALL writePage               ; program content of page buffer


MOV  R4 , #0080h             ; data for 1st erase command
MOV  R5 , #00AAh             ; data for 2nd erase command
MOV  R6 , #0033h             ; data for 3rd erase command

CALL waitBusy    ; before erase operation check if flash is busy

EXTS #0C0h  , #3        ; use segment 0C0h for next 3 operations
MOV  000AAh , R4        ; 1st erase command
MOV  00054h , R5        ; 2nd erase command
MOV  08000h , R6        ; 3rd erase command to...
                        ; ...logical sector 4 at 08000h
CALL waitBusy           ; wait until erase operation is done

waitBusy:               ; check busy flag in FSR
     EXTS #0FFh, #1     ; use segment 0FFh for next operation
     MOV  R7   , 0F000h ; read FSR
     JB   R7.0 , waitBusy     ; if flash still busy loop again
     RET

enterPageMode:
  MOV  R4   , #00AAh ; first command address for enter page mode
  MOV  R5   , #0050h ; first command data for enter page mode
```

```
  MOV  R6   , #08000h; page address of logical sector 4
  EXTS #0C0h, #2
  MOV  [R4] , R5     ; first command  of enter page mode sequence
  MOV  [R6] , R4     ; second command of enter page mode sequence
  RET

writePage:
     MOV  R4   , #00AAh  ; first write page sequence address
     MOV  R5   , #00A0h  ; first write page sequence data
     MOV  R6   , #005Ah  ; second write page sequence address
     MOV  R7   , #00AAh  ; second write page sequence data
     EXTS #0C0h, #2
     MOV  [R4] , R5      ; first command of write page sequence
     MOV  [R6] , R7      ; second command of write page sequence
     RET
```

The second workaround is faster than the first workaround. While the first approach takes about 200 ms, the second approach only takes about 100 ms.

Note: Usually it is not allowed to program a page twice without an erase operation inbetween. This may lead to flash read problems in user applications. In this case double programming is not critical, because the program operation is followed by an erase operation.

After a reset the default configuration of the flash module allows an erase of logical sector 1, 2, 3, 4 without using one of the workarounds listed above.

ASC_X.001 ASC Autobaud Detection in 8-bit Modes with Parity

The Autobaud Detection feature of the Asynchronous/Synchronous Serial Interface (ASC) does **not** work correctly for **8-bit** modes **with** even or odd **parity**.

The Autobaud Detection feature works correctly for 7-bit modes with even or odd parity, and for 8-bit modes without parity.

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

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

T5IN, T6IN, T5EUD, T6EUD.

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

Workaround

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

3 OCDS and OCE Modules

The following issues have been found in the OCDS and OCE modules. Please see the debugger or emulator manufacturer's documentation whether or not these issues actually cause a problem or restriction when the respective tool is used.

OCDS_X.002 OCDS indicates incorrect status after break_now requests if PSW.ILVL ≥ CMCTR.LEVEL

When the OCDS processes a break_now request while the CPU priority level (in PSW.ILVL) is not lower than the OCDS break level (in CMCTR.LEVEL), the actual break is delayed until either PSW.ILVL or CMCTR.LEVEL is reprogrammed such that CMCTR.LEVEL > PSW.ILVL. If in the meantime further debug events have occurred, register DBGSR will still indicate the status of the first break_now request. If e.g. a software break is executed, the OCDS will accept this, but register DBGSR will indicate the wrong cause of break.

Workarounds:

1. If the application uses tasks with different levels and debugging is to take place using the OCDS break level feature (e.g. only tasks up to a maximum level are halted, higher-level tasks aren't halted, and the OCDS level is programmed in between), there is no problem if:

- only classic hardware breakpoints (IP address) or software breakpoints are used (i.e. no trigger on address, data, TASKID)
- no external pin assertions are used to trigger breaks
- no direct writes to DBGSR.DEBUG_STATE are used to force breaks

2. If break_now request sources are to be used, the maximum level of the application (PSW.ILVL) should always be lower than the programmed OCDS break level (e.g. PSW.ILVL ≤ 14_D and CMCTR.LEVEL = 15_D). This means that all generated break_now requests by the OCDS will always be accepted, independent of the CPU or interrupt priority.

OCE_X.001 Wrong MAC Flags are declared valid at Core - OCE interface

In case a MAC instruction (Co...) is directly followed by a MOV MSW, #data16 instruction, the upper byte of data16 is output instead of the flags corresponding to the MAC instruction. The bug was found with code:

```
COSHR   #00001h
MOV     MSW, #00100h  ;(+ other variations of data16)
```

Workaround

Add a NOP instruction between the two instructions:

```
COSHR   #00001h
NOP
MOV     MSW, #00100h  ;(+ other variations of data16)
```

4 Deviations from Electrical- and Timing Specification

No deviations from the Electrical- and Timing specification are known for this step.

5 Application Hints

CPU_X.H1 Configuration of Registers CPUCON1 and CPUCON2

The default values of registers CPUCON1 and CPUCON2 have been chosen to provide optimized performance directly after reset. It is recommended

- not to modify the performance related parts of register CPUCON1
- not to modify register CPUCON2, except for test purposes or for enabling specific workarounds under special conditions (see e.g. problem CPU_X.002 or application hint BREAK_X.H1).

CPUCON2: reset/recommended value = $8FBB_H$; enables several performance features

CPUCON1: reset/recommended value = 0..0 0XXX $X111_B$; only the 3 LSBs are performance related

Bit Position	Field Name	Value	Description
CPUCON1.[15:7]	0	0	reserved
CPUCON1.[6:5]	VECSC	00	scaling factor for vector table, value depends on application, '00' is compatible to C166 systems
CPUCON1.4	WDTCTL	0	configuration for scope and function of DISWDT/ENWDT instructions, value depends on application, '0' is compatible to C166 systems
CPUCON1.3	SGTDIS	0	segmentation enable/disable control, value depends on application
CPUCON1.2	INTSCXT	1	enable interruptibility of switch context
CPUCON1.1	BP	1	enable branch prediction unit
CPUCON1.0	ZCJ	1	enable zero cycle jump function

CPU_X.H2 Special Characteristics of I/O Areas

As an element of performance optimization, the pipeline of the C166S V2 core may perform speculative read accesses under specific conditions. In case the prediction for the speculative read was wrong, the read to the actually required location is restarted. While this method is uncritical e.g. for accesses to non-volatile memories or SRAMs, it may cause problems on devices which do not tolerate speculative reads (e.g. FIFOs which are advanced on every read access).

No speculative reads are performed in memory areas which are marked as I/O area. This memory area includes

- the SFR and ESFR space (e.g. with buffers for received data from serial interfaces or A/D results)
- the 4-Kbyte internal I/O area ($00'E000_H$ ..$00'EFFF_H$), including IIC[1)] and SDLM[1)] module
- the 2-Mbyte external I/O area ($20'0000_H$ ..$3F'FFFF_H$), including the TwinCAN[1)] module (default: from $20'0000_H$ .. $20'07FF_H$)

It is therefore recommended to map devices which do not tolerate speculative reads into the 2-Mbyte external I/O area ($20'0000_H$ ..$3F'FFFF_H$).

For further special properties of the I/O areas, see section IO Areas (3.6) in chapter Memory Organization in the User's Manual.

FLASH_X.H1.1 Access to Flash Module after Program/Erase

After the last instruction of a program or erase command, the BUSY bit in register FSR is set to '1' (status = busy) after a delay of one instruction cycle. When polling the BUSY flag, one NOP or other instruction which is not evaluating the BUSY flag must be inserted after the last instruction of a program or erase command.

No additional delay is required when performing the first operand read or instruction fetch access from the flash module after the BUSY bit has returned to '0' (status = not busy).

FLASH_X.H2.2 Access to Flash Module after Shut-Down

When the flash is disabled by software (shut-down) by writing bit PFMDIS = 1 in register SYSCON3,

- and it is (at some later time) enabled again by writing PFMDIS = 0
- and the instruction immediately following the instruction which sets PFMDIS = 0 is fetched or reads operands from internal flash

then the PACER flag in register TFR is set and the BTRAP routine is entered.

Therefore, it is recommended to insert 4 NOPs before the internal flash is accessed again after PFMDIS has been set to 0.

1) this module is implemented in specific derivatives of the XC166 family

FLASH_X.H3.2 Read Access to internal Flash Module with modified Margin Level

When the internal flash module is read (e.g. for test purposes) with modified margin level (i.e. bitfield MARLEVSEL = 0001_B or 0100_B) in register MAR, an additional wait state must be selected in bitfield WSFLASH in register IMBCTR. This waitstate must be added to the number of flash waitstates that are required to match the flash access time to the CPU operating frequency.

FLASH_X.H4 Minimum active time after wake-up from sleep or idle mode

If the flash module is automatically disabled upon entry into sleep or idle mode (bit field PFCFG = 01_B in register SYSCON1), sleep or idle mode should not be re-entered before a minimum active ("awake") time has elapsed. Otherwise, the current consumption during this sleep/idle phase will be ~ 1 mA above the specified limits of the Data Sheet. Therefore,

- If code is executed from the **internal flash** after wake-up, at least 16 instructions should be executed from the internal flash before re-entering sleep/idle mode. This ensures that the flash module is actually accessed after wake-up, since more instructions are required than can be stored in the prefetch queue.
- If code is executed from **external memory or PRAM**, wait until the flash BUSY bit returns to '0' before re-entering sleep/idle mode.
- If **PEC transfers** with automatic return to sleep/idle mode shall be triggered by the wake-up event, use e.g. the following procedure:

  Use an auxiliary routine in internal flash with the required minimum active time after wake-up from sleep or idle mode, e.g.
  - define a semaphore bit that is set to '1' before the IDLE instruction is executed. All trap and interrupt service routines invoked after wake up from idle/sleep should clear this bit to '0'
  - disable interrupts
  - execute the IDLE instruction
  - if idle or sleep mode is terminated by an interrupt request, the instructions following the IDLE instruction will be executed (the interrupt request flags remain set)
  - if idle or sleep mode was terminated by an $\overline{\text{NMI}}$, the trap handler will be invoked
  - enable interrupts to allow prioritization of requests for interrupt or PEC service
  - the instructions following the IDLE instruction should test the flash BUSY bit in register FSR; when the flash is ready (BUSY = 0), and at least 12 instructions have been executed after the interrupt system has been enabled, and if the semaphore bit is still at '1' (i.e. no interrupts/traps have occurred), disable interrupts and return to the IDLE instruction.

SLEEP_X.H3.2 Clock system after wake-up from Sleep Mode

There are different wake-up behaviors, depending on the PLL control setting used in register PLLCON during entry into sleep mode, and depending on whether the RTC is running on the main oscillator. Note that in either case, the VCO is turned off during sleep mode, and does not contribute to any additional power consumption.

- In bypass mode with VCO off (**PLLCTRL = 00_B**), the device will directly continue to run on the frequency derived from the external oscillator input after wake-up from sleep. If the **RTC** is running on the **main** oscillator, the device is immediately clocked, since the oscillator (input XTAL1) is not turned off during sleep mode.

  If the **RTC** was **not** running on the main oscillator, the system will not be clocked until the amplitude on the external oscillator input XTAL1 exceeds the input hysteresis. This requires typ. a few ms, depending on external crystal/oscillator circuit.

  With this mode, there is **no oscillator watchdog function**, and the system will not be clocked until the external oscillator input XTAL1 receives a clock that exceeds the input hysteresis.

- In bypass mode with VCO on (**PLLCTRL = 01_B**), the device will directly continue to run on the frequency derived from the external oscillator input after wake-up from sleep if the **RTC** continues to run on the **main** oscillator in sleep mode.

  In case the PLL was **locked** before entry into sleep mode, **emergency mode** is entered. This results in PLLODIV = $0F_H$ and bit SYSSTAT.EM = 1. This change of configuration will **not** be notified by the PLL Unlock/OWD interrupt (flag PLLIR). This condition will remain until an external HW reset is applied, or a wake-up event from sleep mode with main oscillator off (i.e. RTC not running on main oscillator) occurs.

  If the **RTC** was **not** running on the main oscillator, (i.e. the main oscillator was off during sleep mode), the device will wake-up using the internal PLL base frequency from the VCO (f_{base}/16) and will temporarily stay in emergency mode (i.e. run on the frequency derived from the VCO) until bit OSCLOCK in register SYSSTAT gets set to 1.

  It is not possible to switch to direct drive (VCO bypass) mode within this timeframe. If bypass mode (PLLCTRL = 00_B, i.e. no oscillator watchdog support) is required by an application after wake-up from sleep, it is therefore recommended to switch to bypass mode already before entry into sleep mode (check PLLCON for its target value before executing the IDLE instruction to enter sleep mode). See also SCU_X.H5.

- In PLL mode with input clock from XTAL1 disconnected (**PLLCTRL = 10_B**), the device will **only** wake up from sleep if the RTC was **not** running on the main oscillator (i.e. when the main oscillator is off during sleep mode). In this case, the device will run using the internal PLL base frequency from the VCO (f_{base}/16) until the amplitude on the external oscillator input XTAL1 exceeds the input hysteresis, and then switch to f_{base}/k with the output divider selected by PLLODIV.

  If the **RTC** is running on the **main** oscillator, the device will **not wake-up** from sleep mode with this PLLCTRL setting. It is therefore recommended to switch to bypass mode (PLLCTRL = 00_B) before entry into sleep mode (check PLLCON for its target value before executing the IDLE instruction to enter sleep mode).

- In PLL mode with input clock from XTAL1 connected to the VCO (**PLLCTRL = 11_B**), if the **RTC** was **not** running on the main oscillator, the device will wake-up in emergency mode and run using the internal PLL base frequency from the VCO (f_{base}/16) until the amplitude on the external oscillator input XTAL1 exceeds the input hysteresis. Then the PLL resynchronizes to the target frequency determined by the settings in register PLLCON. When bit OSCLOCK gets set in register SYSSTAT, the output divider PLLODIV will be set to the target value.

  If the **RTC** is running on the main oscillator, the device will wake-up and resynchronize to the target frequency determined by the settings in register PLLCON.

  In case the PLL was **locked** before entry into sleep mode, **emergency mode** is entered. This results in PLLODIV = $0F_H$ and bit SYSSTAT.EM = 1. This change of configuration will not be notified by the PLL Unlock/OWD interrupt (flag PLLIR). This condition will remain until an external HW reset is applied, or a wake-up event from sleep mode with main oscillator off (i.e. RTC not running on main oscillator) occurs.

  As an alternative, switch to bypass mode with VCO on and PLL unlocked before entering sleep mode (e.g. PLLCON = 2000_H). After wake-up, PLLCON may be reconfigured to the desired PLL operating mode.

IDLE_X.H1 Entering Idle Mode after Flash Program/Erase

After a program/erase operation, idle mode should not be entered before the BUSY bit in register FSR has returned to '0' (status = not busy).

ADC_X.H1 Polling of Bit ADBSY

After an A/D conversion is started (standard conversion by setting bit ADST = 1, injected conversion by setting ADCRQ = 1), flag ADBSY is set 5 clock cycles later. When polling for the end of a conversion, it is therefore recommended to check e.g. the interrupt request flags ADC_CIC_IR (for standard conversions) or ADC_EIC_IR (for injected conversions) instead of ADBSY.

BREAK_X.H1 Break on MUL/DIV followed by zero-cycle jump

When a MUL or DIV instruction is immediately followed by a falsely predicted conditional zero-cycle jump (JMPR or JMPA on any condition other than cc_UC),

and

- either a 'break now' request is set at the time the MUL / DIV instruction is being executed (i.e. a break request on operand address, data, task ID, $\overline{\text{BRKIN}}$ pin etc. is generated by one of the instructions (may be up to four) preceding MUL/DIV)
- or a 'break-before-make' request (break on IP address) is derived from the instruction immediately following the jump (jump target or linear following address, depending whether the jump is taken or not)

then the internal program counter will be corrupted (equal to last value before jump), which will lead to a false update of the IP with the next instruction modifying the IP.

This problem occurs for debugging with OCDS as well as with OCE.

Note: The Tasking and Keil compilers (including libraries) do not generate this type of critical instruction sequence.

Workarounds (choices)

For assembler programmers, one of the following workarounds may be used

1. disable zero-cycle operation for jumps when debugging code (set CPUCON1.ZCJ to '0'), or
2. include a NOP after any MUL/DIV instruction followed by a conditional jump (JMPR, JMPA), or
3. do not set any 'break-before-make'-type breakpoints on the instruction following the jump, or 'break now'-type breakpoints shortly before or on the MUL / DIV instructions

OCDS_X.H001 Effect of Bit OCDSIOEN in Register EMUCON

XC16x devices in a 144-pin package have dedicated pins $\overline{\text{BRKIN}}$ and $\overline{\text{BRKOUT}}$ (pin 144 and 143). Therefore, it is not required to set bit OCDSIOEN = 1 in register EMUCON to select the $\overline{\text{BRKIN}}$/$\overline{\text{BRKOUT}}$ functionality on these pins.

On the contrary, it is recommended to leave bit **OCDSIOEN = 0** (default after reset). Otherwise, the I/O or alternate input functionality on P3.5 and P3.7 is affected (P3.5 switched to output high, P3.7 switched to input).

IIC_X.H1.1 Maximum IIC Bus Data Rate at low f_{CPU}

The IIC bus module requires a minimum of 32 CPU clock cycles per bit. Therefore, the prescaler value BRP for the baudrate generator circuitry must fulfil the following criteria:

Mode 0: if PREDIV = 00_B then BRPmin = 7_H, else BRPmin = 0_H
Mode 1: if PREDIV = 00_B then BRPmax = 20_H, else BRPmax = 0_H

This means that e.g. the extended IIC bus data rate of 400 kbit/s can not be used for clock frequencies f_{CPU} < 12.8 MHz.

IIC_X.H2 Timing of Bit IRQD

The Data Transfer Event Interrupt Request Flag IRQD in register IIC_ST is set to '1' after the acknowledge bit of the last byte has been received or transmitted.

If bit IIC_CON.INT = 0, IRQD is automatically cleared to '0' by HW with a delay of 1 module clock cycle upon a complete read or write accesses to the buffers RTB0...3 (according to the buffer size defined via bit field CI in register IIC_CON). Therefore, after reading/writing RTB0...3, either

- add 5 NOPS, or
- read status register IIC_ST twice

before evaluating the status of flag IRQD.

POWER_X.H1.1 Initialization of SYSCON3 for Power Saving Modes

For minimum power consumption during power saving modes, all modules which are not required should be disabled in register SYSCON3, i.e. the corresponding disable bits should be set to '1', including bits which are marked as 'reserved' (this provides compatibility with future devices, since all SYSCON3 bits are disable bits). Reading these bits will return the written value, as for peripherals without shut-down handshake.

For peripherals equipped with peripheral shut-down handshake, reading allows to check their shut-down status.

POWER_X.H2.2 Power Consumption during Clock System Configuration

In the following situations

1. after wake-up from sleep mode until oscillator lock in case the main oscillator was turned off during sleep mode
2. after a clock failure (PLL unlock or oscillator fail) until clock reconfiguration by software

the device is internally clocked by the VCO running on the base frequency of the currently selected VCO band divided by 16. This results in an operating frequency range of 1.25 .. 11.25 MHz, depending on the currently selected VCO band.

Systems designed for lower target frequencies should consider the increased power consumption due to the potential frequency increase during these phases of operation.

Exception in **bypass mode with VCO off**: in case (1), if the RTC is not running on the main oscillator, and case in (2) the device stops until it again receives a clock from the oscillator.

RSTOUT_X.H1 $\overline{\text{RSTOUT}}$ driven by weak driver during HW Reset

A weak driver (see specification in Data Sheet) has been implemented on pin $\overline{\text{RSTOUT}}$ which is driven low while $\overline{\text{RSTIN}}$ is asserted low. After the end of the internal reset sequence, $\overline{\text{RSTOUT}}$ operates in default mode (strong driver/sharp edge mode, i.e. POCON20.PDM3N[15:12] = 0000_B).

The software setting POCON20.PDM3N[15:12] = $xx11_B$ is not supported and should not be selected by software, otherwise pin $\overline{\text{RSTOUT}}$ floats.

SCU_X.H1 Shutdown handshake by software reset (SRST) instruction

In the pre-reset phase of the software reset instruction, the SCU requests a shutdown from the active modules equipped with shutdown handshake (see section Peripheral Shutdown Handshake (6.3.3) in chapter Central System Control Functions in the User's Manual). The pre-reset phase is complete as soon as all modules acknowledge the shutdown state.

As a consequence, e.g. the A/D converter will only acknowledge the request after the current conversion is finished (fixed channel single conversion mode), or after conversion of channel 0 (auto scan single conversion mode). If the 'Wait for Read Mode' mode is selected (bit ADWR = 1), the ADC does not acknowledge the request if the conversion result from register ADC_DAT has not been read.

Therefore, before the SRST instruction is executed, it is recommended e.g. in the continuous (fixed or auto scan) conversion modes to switch to fixed channel single conversion mode (ADM = 00_B) and perform one last conversion in order to stop the ADC in a defined way. In the auto scan conversion modes, this switch is performed after conversion of channel 0. If a 0-to-1 transition is forced in the start bit ADST by software, a new conversion is immediately started. If the 'Wait for Read Mode' is selected, register ADC_DAT must be read after the last conversion is finished.

The external bus controller e.g. may not acknowledge a shutdown request if bus arbitration is enabled and the HOLD input is asserted low.

SCU_X.H3 Effect of PLLODIV on Duty Cycle of CLKOUT

When using even values (0..14) for the output divider PLLODIV in register PLLCON, the duty cycle for signal CLKOUT may be below its nominal value of 50%. This should only be a problem for applications that use both the rising and the falling edge of signal CLKOUT.

When using odd values (1..15) for PLLODIV, where PLLODIV = 15 ($0F_H$) is selected by hardware only during clock system emergency mode or reconfiguration, the duty cycle for signal CLKOUT is on its nominal value of 50%

PLLODIV	0	2	4	6	8	10	12	14
Duty Cycle [%]	45	33.33	40	42.86	44.44	45.45	46.13	46.67

SCU_X.H4 Changing PLLCON in emergency mode

While the clock system is in emergency mode (e.g. after wake-up from sleep, or due to an external clock failure), the clock output divider is set to 16, i.e. PLLODIV = $0F_H$ in register PLLCON. Emergency mode is only terminated if the internal oscillator lock counter has received 2048 clock ticks from XTAL1 after wake-up from sleep mode (when the oscillator was off during sleep).

If PLLCON is written in emergency mode, all settings except bypass modes (PLLCTRL = $0X_B$) become effective immediately within a few clock cycles. As long as

the system clock is still derived from the VCO, and if a relatively small value k is written to PLLODIV, this results in the system running on an internal frequency of f_{VCO}/k that may exceed the specified frequency limit for the device.

In general, it is recommended to wait until PLLODIV < $0F_H$ before PLLCON is written. Use a timeout limit in case a permanent clock failure is present.

SCU_X.H5 Sleep/Idle/Power Down Mode not entered while PLLODIV = $0F_H$

While the clock system is in reconfiguration (e.g. after write to PLLCON, or after wake-up from sleep when an oscillator lock event occurs, or during transition to emergency mode after clock failure), entry into power saving modes is delayed. If e.g. the IDLE instruction to enter sleep mode is executed in this state, the peripherals are already stopped, and the CPU goes into hold state, but the internal clock system will not be switched off until the reconfiguration is complete.

Unless it is guaranteed that the clock system will become stable after a reconfiguration, it is recommended to wait until the clock system is stable (i.e. check for PLLODIV < $0F_H$, use a timeout limit in case a permanent clock failure is present) before executing the IDLE or PWRDN instruction to enter the respective power saving mode.

SCU_X.H6 Interrupt request during entry into sleep mode

After the IDLE instruction has been executed in order to enter sleep mode (SLEEPCON (SYSCON1.1-0) = 01_B), clock system emergency mode (with $f_{VCObase}$ / 16) will become active during the shut down phase before the clock is finally switched off under the following conditions:

- the clock system is not running in bypass mode (PLLCTRL (PLLCON.14-13) = 00_B), **and**
- the RTC is not running on the clock derived from XTAL1 during sleep mode.

If an interrupt request (from an internal or external peripheral module) is generated during this time period, sleep mode is not entered, but instead the associated interrupt service routine is entered. The internal system frequency at that time might not yet be the intended target frequency, since the clock system requires some time to return to its previous state.

To avoid operation on a frequency that is different from the target frequency, either

- do not enter sleep mode (execute the IDLE instruction) while interrupt requests can still occur, **or**
- wait at the beginning of the interrupt service routine until the contents of register PLLCON has returned to the intended target configuration, **or**

- if the interrupt service is not time critical, disable interrupts (IEN (PSW.11) = 0, or select CPU priority level ILVL (PSW.15-12) = $0F_H$) before executing the IDLE instruction. The interrupt service will then be performed after the wake-up from sleep mode has occurred and the interrupt system has been re-enabled (see also SLEEP_X.H3.2).

SCU_X.H7 VCO Configuration with Input Clock disconnected

The clock configuration where the input clock is disconnected from XTAL1 (PLLCON.PLLCTRL=10_B) was primarily implemented for test purposes. Nevertheless, it may be used in an application if certain limitations are observed:

- Because the clock frequency is derived from the VCO running in unlocked mode on its base frequency, it will vary to some extent with technology, temperature, and supply voltage.
- When switching from a configuration where the VCO was running on a high frequency (> 100 MHz in locked mode), i.e. from PLLCON.PLLCTRL=11_B or 01_B, to the clock configuration with input clock disconnected from XTAL1, i.e. to PLLCON.PLLCTRL=10_B, and a relatively small value is selected for the output divider PLLCON.PLLODIV (e.g. PLLODIV = 0_H), this may temporarily lead to a master clock overdrive (f_{MC} > 40 MHz or 20 MHz, respectively).
  In this case, it is recommended to use an intermediate PLLCON setting (e.g. PLLCON=2000_H, i.e. bypass mode with VCO running in unlocked mode on lowest VCO band), and then wait ~ 200 µs until the VCO is definitely running on its base frequency before finally switching to the desired target configuration with input clock disconnected from XTAL1 (PLLCON.PLLCTRL=10_B) and setting PLLVB and PLLODIV to the desired target values.

SCU_X.H8 PLL Bypass Mode with VCO on

In bypass mode with VCO running as oscillator watchdog OWD (PLLCON.PLLCTRL=01_B) it is recommended to configure f_{in} to a value that does not allow the PLL to lock with f_{VCO}/(PLLCON.PLLMUL+1). This can be achieved by using an appropriate VCO band in combination with PLLCON.PLLMUL=0_H, e.g. PLLCON=2080_H, such that $f_{in}=f_{OSC}$/(PLLCON.PLLIDIV+1) is lower than f_{VCO_base} (see table below).

Otherwise, when f_{in} is at the transition between a lock and an unlock operation of the PLL, the internal master clock f_{MC} may not reach the intended target frequency, but is set to f_{OSC}/((PLLCTRL.PLLIDIV+1)*16), i.e. PLLODIV is set to the maximum value of $0F_H$. When a clock failure (e.g. crystal break) occurs, the PLL may not switch to emergency

mode and the device stops.

Table 8 Relation between maximum f_{in} and VCO Band Selection

PLLCON.PLLVB	VCO base frequency f_{VCO_base}	Maximum f_{in}
0	20 MHz	< 20 MHz
1	40 MHz	< 40 MHz
2	60 MHz	< 60 MHz

SCU_X.H10 Register Security Mechanism usage with CPUCON1 and CPUCON2 registers

When using the register security mechanism in conjunction with protected registers CPUCON1 or CPUCON2, CPU pipeline effects must be taken into account. When using Secured Mode (bitfield SL = 01_B in register SCUSLS), command 4 of the unlock sequence will update the SCUSLS register in the write back stage of the pipeline. But, the registers CPUCON1 and CPUCON2 belong to the class of registers known as CPU core SFRs (CSFRs), and updates to these registers occur one cycle earlier, in the execute stage (see User's Manual, chapter 4.3 'Instruction Processing Pipeline'). Attempting to write one of the CPUCONx registers immediately after command 4 of the unlock sequence will cause the write operation to the CPUCONx register to occur before the unlock sequence is in effect. Therefore, the CPUCONx register will not be updated as expected (see also CPU_X.D1).

To avoid this effect, it is generally recommended to initialize the CPUCON1 and CPUCON2 registers during software initialization, before the EINIT instruction is executed. Until EINIT, the Unsecured Mode is active and no unlock sequence is required.

If, instead, the CPUCON1 or CPUCON2 register must be modified using Secured Mode, the following sequence is recommended:

```
Example:
   MOV R4, #8FBBH      ; Value to be stored in register CPUCON2
   EXTR #2             ; Access sequence in secured mode
                       ; (use EXTR #4 to avoid interrupts
                       ; if interrupts are enabled at this time)
   MOV SCUSLC, #8E12H  ; Command4: current password = EDH
   MOV ZEROS, SCUSLC   ; Dummy read of SCUSLC control register,
                       ; to ensure Command 4 is in effect
                       ; before continuing[1)]
   MOV CPUCON2, R4     ; Access to CPUCON2 enabled by
                       ; preceding Command4
```

```
MOV IDCHIP, ZEROS  ; Dummy write to a read-only SCU register
                   ; re-enables secured mode[1)]
```

RTC_X.H1.1 Resetting and Disabling of the Real Time Clock

The clock source for the RTC can be selected by software via bit REFCLK (RTC_CON.4)

- REFCLK = 0: RTC count clock signal is f_{OSCaux} (input XTAL3)
- REFCLK = 1: RTC count clock signal is $f_{OSCmain}$ / 32 (input XTAL1)

Register RTC_CON is not affected by a hardware/software/watchdog reset. After power-up, it is undefined. A reset of the RTC module is achieved by setting bit SYSCON0.15/RTCRST = 1. This way, register RTC_CON is set to 8003_H (RTC runs, prescaler by 8 enabled, bit REFCLK = 0).

The RTC clocking mode (synchronous, asynchronous) is determined by bit RTCCM (SYSCON0.14). Note that register SYSCON0 is not affected by a software or watchdog reset. This means that when a software or watchdog reset occurred while the RTC module was in asynchronous mode (selected by bit RTCCM (SYSCON0.14) = 1), it will return to asynchronous mode after a RTC reset triggered by setting bit RTCRST (SYSCON0.15) = 1 with a bit instruction.

For a software or watchdog reset that is followed by an initialization of the RTC module, it is recommended to

- select synchronous RTC clocking mode, i.e. clear bit SYSCON0.14/RTCCM = 0
- reset the RTC module, i.e. set bit RTCRST (SYSCON0.15) = 1.

This may be achieved with one word or bit field instruction, e.g.

```
             EXTR  #1
             BFLDH SYSCON0, #0C0h, #80h ; RTCRST = 1, RTCCM = 0
wait_accpos: EXTR  #1
             JNB   ACCPOS, wait_accpos ; wait until bit ACCPOS = 1
```

As a general recommendation, bit REFCLK should be set to the desired value after any start-up (hardware, software, watchdog reset), and in particular whenever the RTC module is reset by setting bit RTCRST (SYSCON0.15) = 1.

1) see CPU_X.D1

1) see SCU_X.011

When the RTC module is not used and shall be disabled after a (power-on) hardware reset, the following steps are recommended:

1. reset the RTC by setting bit RTCRST (SYSCON0.15) = 1
2. clear the RTC run bit by setting bit RUN (RTC_CON.0) = 0
3. disable the RTC module by setting bit RTCDIS (SYSCON3.14) = 1.

FOCON_X.H1 Read access to register FOCON

Bit FOTL and bit field FOCNT in register FOCON are marked as 'rh' in the User's Manual, i.e. they can not be modified by software. If register FOCON is read directly after it was written, the value read back from the positions of FOTL and FOCNT represents the value that was written by the preceding instruction, but not the actual contents of FOTL and FOCNT. In order to obtain correct values for FOTL and FOCNT, either insert one NOP or other instruction that does not write to FOCON, or read FOCON twice and discard the first result.

OSC_X.H001 Signal Amplitude on Pin XTAL1

When XTAL1 is driven by a crystal or a ceramic resonator, reaching an amplitude (peak to peak) of 0.4 x V_{DDI} is sufficient.

When XTAL1 is driven by an external clock signal, the levels V_{ILC} (max. 0.3 x V_{DDI}) and V_{IHC} (min. 0.7 x V_{DDI}) must be respected (see Data Sheet, section DC Parameters).

TwinCAN_AI.H2 Reading Bitfield INTID

It is not recommended to use the information stored in bitfield INTID in register AIR/BIR, as it is updated with low priority within the CAN controller. Instead, similar information can be obtained from registers RXIPND and TXIPND.

INT_X.H1 Software Modifications of Interrupt Enable (xx_IE) or Interrupt Request (xx_IR) Flags

In microcontroller architectures that are optimized for real time applications, the CPU operates in parallel to the interrupt system in order to minimize interdependencies resulting in delayed processing. However, in certain exceptional situations, it is sometimes required by software to temporarily disable interrupts.

Application Hints

In the microcontrollers of the C166 and XC166 microcontroller families, the interrupt system can be **globally** disabled with a single instruction e.g. via

```
BCLR IEN  ; clear IEN flag (causes pipeline restart in C166S V2 core)
```

In the microcontrollers of the XC166 family with the C166S V2 core, the pipeline side effects of the classic C166 core when globally disabling interrupts[1] no longer need to be considered. Nevertheless, the special instruction sequences recommended to avoid these side effects of the C166 core will work without problems with the C166S V2 core.

Instead of globally disabling the interrupt system, it might be appropriate in some cases to **selectively** clear an individual interrupt enable flag (xx_IE) e.g. via

```
BCLR GPT12E_T2IE  ; clear Timer 2 interrupt enable flag
```

Theoretically, in the C166S V2 core, an interrupt (in this example: from Timer 2) may still be accepted by the CPU for a few cycles after the `BCLR GPT12E_T2IE` instruction has been executed. This is typically the case if the interrupt request occurs asynchronously to program execution, i.e. if the software does not know when an event occurs, and it is just about to clear flag xx_IE when the interrupt request from source xx occurs.

Note: In practice, the normal case to disable an interrupt channel xx via `BCLR xx_IE` would be at the beginning of the interrupt service routine when the associated peripheral is finished (e.g. has sent serial data stream). Then, either no more requests from this source occur for a certain period of time, and/or several instructions to restore the system state (inserted by the compiler) are executed after `BCLR xx_IE` at the end of the interrupt service routine before the terminating RETI. In this case, interrupt requests from source xx are safely disabled.

The same effect will occur when an interrupt request flag (xx_IR) is cleared by software. Therefore, in the following examples, both cases (clearing xx_IE or xx_IR) are discussed together.

With the following modified sequence, no more Timer 2 interrupts can occur after `BCLR GPT12E_T2xx` has been executed:

```
  BCLR IEN          ; globally disable interrupts
  BCLR GPT12E_T2IE  ; clear Timer 2 interrupt enable
                    ;  (or request) flag
  ...               ; any number of other instructions (incl. zero)
  JNB GPT12E_T2IE, Next ; this or any other instruction
                        ;  reading T2IC (assures T2IC is written
                        ;  by BCLR before being read by JNB)
```

1) see Application Note ap16009 'How to make instruction sequences uninterruptable' on the Microcontroller internet pages of Infineon Technologies www.infineon.com/c166-family (follow link to Application Notes - 16-bit Microcontrollers)

```
Next:
  BSET IEN            ; globally enable interrupts again
```

In case the above sequence is included in some sort of macro, it may be desirable to prevent interrupts from inadvertently being globally re-enabled e.g. by the following sequence:

```
  JNB  IEN, int_dis
  BCLR IEN
  BCLR GPT12E_T2IE ; clear Timer 2 interrupt enable
                   ;  (or request) flag
  ...              ; any number of other instructions (incl. zero)
  JNB GPT12E_T2IE, Next ; this or any other instruction
                        ;  reading T2IC (assures T2IC is written
                        ;  by BCLR before being read by JNB)
Next:
  BSET IEN              ; globally enable interrupts again
  JMPR cc_uc, skip_over
int_dis:
  BCLR GPT12E_T2IE  ; clear Timer 2 interrupt enable
                    ;  (or request) flag
  ...              ; any number of other instructions (incl. zero)
  JNB GPT12E_T2IE, skip_over ; read T2IC (not required
                             ;  unless interrupts are globally
                             ;  re-enabled within the next
                             ;  few instructions)
skip_over:
...
```

This is also easy to implement as a macro in C:

```
#define Disable_One_Interrupt(IE_bit) {if(IEN) {IEN=0; IE_bit=0;
while (IE_bit); IEN=1;} else {IE_bit=0; while (IE_bit);}}
```

Note: Due to optimization of the interrupt response time in conjunction with ATOMIC/EXTEND sequences, an interrupt request that has won interrupt priorization at the beginning of an ATOMIC/EXTEND sequence is processed after the ATOMIC/EXTEND sequence. When an xx_IR or xx_IE flag is cleared within an ATOMIC/EXTEND sequence, the associated interrupt might still be acknowledged after the end of the ATOMIC/EXTEND sequence.

*Therefore, e.g. the following sequence is **not recommended** if the intention is that no more Timer 2 interrupts can occur after the BCLR GPT12E_T2Ix instruction:*

```
ATOMIC #3
BCLR GPT12E_T2IE ; clear Timer 2 interrupt enable flag
NOP
NOP ; Timer 2 interrupt might still occur
    ;  after this instruction!
```

INT_X.H002 Increased Latency for Hardware Traps

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

1. When the CPU is in break mode, e.g. single-stepping over such instructions as `SBRK` or `BSET TFR.x` (where `x` = one of the trap flags in register TFR) will have no (immediate) effect until the next instruction enters the Memory stage of the pipeline (i.e. until a further single-step is performed).
2. When the pipeline is running empty due to (mispredicted) branches and a relatively slow program memory (with many wait states), servicing of the trap is delayed by the time for the next access to this program memory, even if vector table and trap handler are located in a faster memory. However, the situation when the pipeline/prefetcher are completely empty is quite rare due to the advanced prefetch mechanism of the C166S V2 core.
3. When the CPU is in a power saving state while the $\overline{\text{NMI}}$ trap request occurs, the next instruction <n+1> that would normally follow the IDLE instruction has to be fetched first and processed up to the Memory stage (where it is cancelled). If the internal flash was off during the power saving state, and the IDLE instruction was executed from flash, and/or the vector table and trap handler are located in internal flash, the ramp-up time for the flash must be considered in addition.
   To achieve the fastest possible response to hardware traps, the instruction to enter the power saving state as well as the vector table and trap handler should be located in the internal PSRAM.

6 Documentation Update

INT_X.D1 Interrupt Vector Location of CAPCOM Register 28

The vector location for requests from CAPCOM Register 28 is xx'00F0$_H$ (not xx'00E0$_H$ as documented in User's Manual)

SCU_X.D1.1 Oscillator gain reduction

The gain of the main oscillator (pins XTAL1/2) may be reduced via software by setting bit OSCGRED (SYSCON0.12) = 1. After HW reset, and after wake-up from sleep mode where the main oscillator was switched off, high gain is selected by default to guarantee safe start-up. When OSCGRED has been set to ‘1’ by software, the gain is reduced automatically when bit OSCSTAB (SYSSTAT.11) = 1. Bit OSCSTAB (and OSCLOCK) is cleared after HW reset or after wake-up from sleep when the main oscillator was switched off during sleep (i.e. RTC not running on frequency derived from XTAL1). Bit OSCSTAB is set to ‘1’ after 2^{15} (32768) cycles of f_{XTAL1} that exceed the input hysteresis have been detected.

SCU_X.D2.2 Functionality of register OPSEN

When a breakpoint is hit, the on-chip peripherals selected in register OPSEN are stopped and placed in power-down mode the same way as if disabled via register SYSCON3. Registers of peripherals which are stopped this way can be read, but not written. A read access will not trigger any actions within a disabled peripheral.

The SYSCON3 bits return the shutdown status independently of the reason for the shutdown (static shutdown via SYSCON3 or intermediate shutdown via OPSEN), i.e. when SYSCON3 is read via the debugger after a breakpoint has been hit, it returns the contents of SYSCON3 ORed bitwise with the contents of OPSEN.

It is recommended to leave bit OPSEN.5 (PFLSEN) at its default value '0'. Otherwise, the program flash is deactivated when a breakpoint is hit (i.e. it can not be read), and it has to ramp up when program execution is resumed (i.e. synchronization between software and peripherals is lost).

SCU_X.D3 Register PLLCON after software reset

Register PLLCON is not affected by a software reset, i.e. the current clock configuration remains unchanged. PLLCON may be reconfigured by software, or an endless loop

terminated by a watchdog timer reset may be used to force PLLCON to its hardware reset value.

SCU_X.D5 VCO band after hardware/watchdog reset in single chip mode

From the falling edge of $\overline{\text{RSTIN}}$ until 2048 stable oscillator clocks after the rising edge of $\overline{\text{RSTIN}}$, always the lowest VCO band is selected. After that, for a hardware or watchdog timer reset in single-chip mode, PLL bypass mode with the lowest VCO band is selected during the internal reset phase (PLLCON = 2710_H). See also application hint POWER_X.H2.2.

SCU_X.D6 Register Security Mechanism - Unprotected Mode active until execution of EINIT instruction

After reset, the Unprotected Mode is selected by default (bitfield SL = 00_B in register SCUSLS). This allows the initialization software to handle all SFRs (with or without register security mechanism) identically. Unprotected Mode remains effective until execution of the EINIT instruction. After execution of the EINIT instruction, Write Protected Mode becomes active (bitfield SL = 11_B in register SCUSLS). Only after execution of EINIT the security level can be changed to Secured Mode or Write Protected Mode by software.

FLASH_X.D1 Interaction between Program Flash and Security Sector Programming

The on-chip Flash module of the XC16x uses specific internal status information for the Program Flash area and for the Security Sector. This internal status information is updated with an erase operation or a programming operation (write page command). After reset, always the status information for the program flash is selected.

As documented in the User's Manual, an area in the program flash or security sector must be erased before it is reprogrammed:

"**Caution**: Writing to a flash page (space for the 128-byte buffer) more than once before erasing may destroy data stored in neighbor cells! This is especially important for programming algorithms that do not write to sequential locations."

For the interaction of erase/programming operations to the program or security areas, the following additional rule must be considered:

To ensure correct programming, make sure that a programming operation to either area (program/security) is always preceded by a programming or erase operation to the same area, otherwise wrong data may be stored.

An erase operation to either area (program/security) is always executed correctly, independent of the preceding operation.

In other words:

- make sure that no erase/program operation to the program flash or reset occurs between erasing and programming of an area in the security sector.
- make sure that no erase/program operation to the security sector is executed between erasing and programming of an area in the program flash.

PORTS_X.D2.144 Internal Pull-up Devices active on pins P4.3-0 and P3.12 during Reset

In addition to the pins mentioned in chapter 6.1.4. (System Startup Configuration), internal pull-up devices are active

- on pins P4.3-0 during a hardware reset (low level on pin RSTIN#). The pull-ups are turned off after the rising edge on pin RSTIN#.

- on pin P3.12 during the internal reset phase of each reset (including hardware, software and watchdog timer reset).

The electrical characteristics of these internal pull-up devices are described in section 'DC Characteristics' of the Data Sheet by the parameters 'Level inactive hold current' (symbol I_{LHI}) and 'Level active hold current' (symbol I_{LHA}).

If any of these pins needs to be on a low level during reset, external pull-down devices that meet this specification must be used.

TwinCAN_AI.D1 Reserved Bits in Registers of the TwinCAN Module

Reserved bits in registers of the TwinCAN module are marked as '0, r' or '1, r' (read only) in the User's Manual. Actually, these bits are implemented as RAM (read-write) bits that are initialized with the reset value documented in the User's Manual. This means that these bits can be modified by software. This will not present a problem as long as these bits, when written by software, are written with their specified reset values ('0' or '1', respectively). In this case, they will always return their specified reset value when they

are read by software. There is only one exceptional case which is described as TwinCAN_AI.010 Reserved Bits in Register MSGARHn[15:13].

No problem can occur in principle if these bits are ignored (e.g. masked off) when they are read.

CPU_X.D1 Write Operations to Control Registers

After write operations to CPU core SFRs (CSFRs) that control important system parameters (e.g. PSW, DPP0..3, CPUCON1/2, SP, etc., see User's Manual chapter 4 for a complete summary of CSFRs), internal control mechanisms (e.g. pipeline stall) ensure that the effect of the write operation has taken place when the following instruction is executed.

However, for other SFRs not belonging to the CSFRs, the effect of write operations may not yet be seen immediately within the next instruction. This is because in general (i.e. except for CSFRs) the CPU is not able to recognize the need of holding the pipeline till the write is effective (unless the write is followed by a read from the same address). To ensure that in this case a write operation has been completed before making use of it, it is recommended to make a read operation on the same memory location.

In addition, there are other registers where a write operation triggers a state machine that may take several cycles to complete (e.g. write to PLLCON, SYSCON3). Reading these registers immediately after a write usually returns the current status which may differ from the written value.

ADC_X.D3 ADC Sample Time with Improved/Enhanced Timing Control

– In enhanced mode (bit MD = 1 in register ADC_CTR0),
– or in compatibility mode (bit MD = 0) with improved conversion and sample timing control (bit ICST = 1 in register ADC_CON1),

the ADC sample time controlled via the 6-bit field ADSTC is defined as follows:

$t_S = t_{BC}$ x 4 x (<ADSTC> + **2**) [instead of $t_S = t_{BC}$ x 4 x (<ADSTC> + 1)]

This applies to the register description of the following registers:

- ADC_CON1
- ADC_CTR2
- ADC_CTR2IN

The values and the formula for the sample time listed in chapter 'Conversion Timing Control' and in particular in Table 'Improved Conversion and Sample Timing Control' are correct.

ID-Registers

		Register:	IDMANUF	IDCHIP	IDMEM	IDPROG
Device	**Step**	**Address:**	**$F07E_H$**	**$F07C_H$**	**$F07A_H$**	**$F078_H$**
XC167CI-16F	BA		1820_H	2004_H	3020_H	4040_H
XC167CI-16F	BB		1820_H	2004_H	3020_H	4040_H

Product and Test Engineering Group, Munich