Microcontrollers

Errata Sheet

V0.3 2004-08-19

Device	**XC167CI-16F20F, -16F40F**
Stepping Code/Marking	**ES-AD, AD**
Package	**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.

Current Documentation

- XC167CI Data Sheet - V1.1, 2003-06
- C166S V2 User's Manual (Core, Instruction Set) - V1.7, 2001-01
- XC167 System Units - User's Manual V2.0, 2004-04
- XC167 Peripheral Units - User's Manual V1.1, 2004-04

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.

Contents

Section

1. **History List/Change Summary**
2. **Functional Problems**
3. **Deviations from Electrical- and Timing Specification**
4. **Application Hints**
5. **Documentation Update**

1. History List/Change Summary

from Errata Sheet Rev. 0.2 for **XC167CI-16F20F/-16F40F** devices with marking **ES-AD, AD** to this Errata Sheet Rev. 0.3 for **XC167CI-16F20F/-16F40F** devices with marking **ES-AD, AD**:

Documentation Reference **updated**:
- XC167-16 User's Manual V2.0, Volume 1/2: System/Peripheral Units - 2004-04

Description of the following problems **added**:

- Frequency Limits for Flash Read Accesses (FCPUR_X.162832): see section “Deviations from DC/AC Specification”
- Modification of register PLLCON while CPSYS = 1 (SCU_X.008.1)
- Software or Watchdog Timer Reset while Oscillator is not locked (SCU_X.009.1)
- ASC Autobaud Detection in 8-bit Modes with Parity (ASC_X.001)

The following Application Hint has been **added**:

- Timing of Bit IRQD (IIC_X.H2)

The following Application Hints have been **removed**::

- Handling of Results of Injected Conversions by the A/D Converter (ADC_X.H2): see Manual V2.0 p.16-15

The following text module has been **removed** from section ‘Documentation Update’:

- Minimum Module Clock Frequencies for TwinCAN (TwinCAN_X.D1): see Manual V2.0 p.22-46

1.1 Summary of Fixed Problems

Problem Name	Short Description	Fixed in Step
PORTS_X.009.1	Pins associated with External Bus Controller are driven after Software Reset in Single-Chip Mode	ES-AC
PORTS_X.010	Sleep and Power Down Mode Supply Current influenced by P3.5	ES-AC
SCU_X.007	Reset value of register FOCON after Software/Watchdog Timer reset	ES-AC

1.2 Summary of Open Problems

Problem Name	Short Description	Fixed in Step
EBC_X.003	TwinCAN access with EBC enabled	
CPU_X.002	Branch to wrong target after mispredicted JMPI	
CC6_X.002	T12 Shadow Transfer for Phase Delay Function	
ADC_X.003	Coincidence of Result Read and End of next Conversion or Start of Injected Conversion in Wait for Read Mode	
ADC_X.004	ADC Overrun Error Generation when result is read during last cycle of conversion	
SLEEP_X.001	Wake up trigger during last clock cycle before entry into sleep mode	
TwinCAN2.005	Double Send	
TwinCAN2.006	CPUUPD fast set/reset	
TwinCAN2.007	Transmit after error	
TwinCAN2.008	Double remote request	
ASC_X.001	ASC Autobaud Detection in 8-bit Modes with Parity	
SCU_X.008.1	Modification of register PLLCON while CPSYS = 1	
SCU_X.009.1	Software or Watchdog Timer Reset while Oscillator is not locked	
OCDS_X.001	BRKOUT# pulsing after Instruction Pointer Debug Event	
OCDS_X.002	OCDS indicates incorrect status after break_now requests if PSW.ILVL ≥ CMCTR.LEVEL	
OCE_X.001	Wrong MAC Flags are declared valid at Core - OCE interface	
TAP_X.85	Maximum ambient temperature during flash programming 85°C	
FCPUR_X.162832	Frequency Limits for Flash Read Accesses	

1.3 Summary of Application Hints

Name	Short Description	Remarks
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	steps ≥ AC
FLASH_X.H2.1	Access to Flash Module after Wake-Up from Sleep/Idle Mode or Shut-Down	steps ≥ AC
FLASH_X.H3.1	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.H1	Sleep Mode during PLL Reconfiguration	
SLEEP_X.H3	Clock system after wake-up from Sleep Mode	steps ≥ AD
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	
BREAK_X.H2.1	Behavior of on-chip Peripherals after Break	steps ≤ AC only
IIC_X.H1	Maximum IIC Bus Data Rate at low f_{CPU}	
IIC_X.H2	Timing of Bit IRQD	
POWER_X.H1	Initialization of SYSCON3 for Power Saving Modes	
POWER_X.H2.1	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	
SCU_X.H2.1	Preservation of internal RAM contents after reset	step AD
SCU_X.H3	Effect of PLLODIV on Duty Cycle of CLKOUT	
SCU_X.H4	Changing PLLCON in Emergency Mode	
CC6_X.H1	CAPCOM6 Suspend Mode Entry and Exit	step AD
RTC_X.H1.1	Resetting and Disabling of the Real Time Clock	
FOCON_X.H1	Read Access to register FOCON	
TwinCAN_X.D1	Minimum f_{CAN} for TwinCAN module	see Manual V2.0 p.22-46
ADC_X.H2	Handling of Results of Injected Conversions by the A/D Converter	see Manual V2.0 p.16-15

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 ≠ 00b 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' in all TCONCSx registers which are used for external bus accesses.

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 CPUCON1.ZCJ = 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 (e.g. 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.

CC6_X.002: T12 Shadow Transfer for Phase Delay Function

In Hall mode (T12MSELx = '1001'), T12 never reaches its period value in normal operation because a detected hall event on CCPOSx captures the actual T12 count value to CC60R (speed reference) and resets T12. But this period event is needed to trigger the shadow transfer of T12 registers, i.e. an update of the timer registers (e.g. CC61R as variable phase delay) is not possible. The shadow transfer should be triggered together with the reset event of T12.

Workaround:

The shadow registers of T12 are transparent if the timer is stopped and if the shadow transfer is enabled (STE12=1). Therefore it is possible to write the new value to the shadow registers by enabling the shadow transfer, stopping T12 and starting T12 again. This is equivalent to a hardware triggered shadow transfer event. Inevitably, some clocks for the capture/compare events are lost in between the stop / start operation. It has to be noted that bit STE12 is cleared by hardware after a shadow transfer event.

ADC_X.003 Coincidence of Result Read and End of next Conversion or Start of Injected Conversion in Wait for Read Mode

When the A/D Converter operates in wait for read mode (bit ADWR = 1 in register ADC_CON or ADC_CTR0, respectively), and the result of a previous standard (non-injected) conversion #n is read from the result register (ADC_DAT) **in the same clock cycle** as the ADC

a) completes the next standard conversion #n+1,

or

b) starts an injected conversion after conversion #n+1 has been completed at some earlier time,

the following problem will occur:

- the interrupt request flag ADCIR for standard conversions is **not** set
- the result of conversion #n+1 is **not** transferred to the result register ADC_DAT

This leads to a lock situation where the ADC will not perform further standard conversions, since there is no trigger (interrupt request) for the PEC or interrupt service routines to read the results.

Further injected conversions are performed correctly.

Workaround 1 (for standard conversions with or without injected conversions)

In order to avoid the problem, make sure that the method (interrupt routine or PEC transfer) that is used to read the results of standard conversions can keep track with the conversion rate of the ADC, i.e. make sure that no wait for result read situation occurs. In this context, the following points may be considered:

- assign the highest priority to the PEC channel or interrupt service routine, and use local register banks and the interrupt jump table cache (fast interrupt)
- check for long ATOMIC or EXTEND sequences, or phases where the interrupt system is temporarily disabled (e.g. by an operating system)
- extend the ADC conversion time (bit fields ADCTC, ADSTC)

Workaround 2.1 (for standard conversions if no injected conversions are used)

Use an interrupt service routine to read the results of standard conversions from register DAT. In the interrupt service routine, read the result register DAT twice. In case a lock situation had occurred, the second (dummy) read access will terminate the lock situation: the next conversion result #n+1 will be transferred to DAT, and interrupt request flag ADCIR will be set, such that the result #n+1 will be read correctly in the associated interrupt service routine.

In contrast to Workaround 1, this workaround has less strict timing requirements, because it allows wait for read situations and handles the result management within the interrupt service routine. The interrupt request flag ADCIR is analyzed to identify and store new results.

The sequence in the interrupt service routine for standard conversions is e.g.:

```
     …
     MOV  [Rx+], [Ry]          ; read and store conversion result #n
                               ; [Rx] points to table for results,
                               ; [Ry] points to DAT
     NOP                       ; wait 3 clock cycles before reading ADCIR
     NOP
     NOP
     JB   ADCIR, Label_Done    ; if ADCIR = 1, result of next conversion
                               ; will be read in next invocation of this
                               ; interrupt service routine (wait for read
                               ; protection is effective)
     ATOMIC #4
     MOV  dummy, DAT           ; second read of result #n (workaround)
     NOP                       ; wait 3 clock cycles before reading ADCIR
     NOP
     ATOMIC #3
     JNB  ADCIR, Label_Done    ; if ADCIR = 1, a conversion has just finished
     BCLR ADCIR                ; clear interrupt request flag
     MOV  [Rx+], [Ry]          ; read and store conversion result #n+1
                               ; [Rx] points to table for results,
                               ; [Ry] points to DAT

                               ; if required: emulation of PEC COUNT function
     SUB  s_count, #1          ; decrement counter for standard conversions
Label_Done:
     SUB  s_count, #1          ; decrement counter for standard conversions
     JMPR cc_NZ, Label_End     ; if s_count = 0:
     …                         ; include code for handling of s_count = 0
                               ; (assumes that no further conversion is
                               ; started after s_count = 0, and same
                               ; interrupt as for reading the results is
                               ; used)
Label_End:
     …
     RETI
```

This routine has only the timing requirement that the 7 instructions in the ATOMIC #4 /ATOMIC #3 sequence - from the dummy read until the result is stored in case a new conversion is finished – are executed within tc (ADC conversion time).

Workaround 2.2 (for standard conversions if injected conversions are used in parallel)

Use an interrupt service routine to read the results of standard conversions from register DAT. In the interrupt service routine, read the result register DAT twice. In case a lock situation had occurred, the second (dummy) read access will terminate the lock situation: the next conversion result #n+1 will be transferred to DAT, and interrupt request flag ADCIR will be set, such that the result #n+1 will be read correctly in the associated interrupt service routine.

Interference with injected conversions is avoided by disabling them temporarily.

The sequence in the interrupt service routine for standard conversions is e.g.:

```
     …
     BCLR IEN                  ; disable interrupts (to shorten time where
                               ; injected conversions are disabled)
     BCLR ADCIN                ; temporarily disable injected conversions to
                               ; avoid conflict between read of DAT and start
                               ; of injected conversion for 2nd and 3rd read
                               ; of DAT (1st read covered by workaround)
     MOV  [Rx+], [Ry]          ; read and store conversion result #n
                               ; [Rx] points to table for results,
                               ; [Ry] points to DAT
     NOP                       ; wait 3 clock cycles before reading ADCIR
     NOP
     NOP
     JB   ADCIR, Label_Done    ; if ADCIR = 1, result of next conversion
                               ; will be read in next invocation of this
                               ; interrupt service routine (wait for read
                               ; protection is effective)
     MOV  dummy, DAT           ; second read of result #n (workaround)
     NOP                       ; wait 3 clock cycles before reading ADCIR
     NOP
     NOP
     JNB  ADCIR, Label_Done    ; if ADCIR = 1, a conversion has just finished
     BCLR ADCIR                ; clear interrupt request flag
     MOV  [Rx+], [Ry]          ; read and store conversion result #n+1
                               ; [Rx] points to table for results,
                               ; [Ry] points to DAT

                               ; if required: emulation of PEC COUNT function
     SUB  s_count, #1          ; decrement counter for standard conversions
Label_Done:
     BSET IEN                  ; enable interrupt system again
     BSET ADCIN                ; enable injected conversions again
     SUB  s_count, #1          ; decrement counter for standard conversions
     JMPR cc_NZ, Label_End     ; if s_count = 0:
     …                         ; include code for handling of s_count = 0
                               ; (assumes that no further conversion is
                               ; started after s_count = 0, and same
                               ; interrupt as for reading the results is
                               ; used)
Label_End:
     …
     RETI
```

This routine has only timing requirement that the 7 instructions, from the dummy read (MOV dummy, DAT) until the result is stored in case a new conversion is finished, are executed within tc (ADC

conversion time). Note that this instruction sequence can not be interrupted, because interrupts have been disabled to avoid interference with injected conversions.

Workaround 3

In order to avoid the problem, make sure that only one conversion is started at a time.

a) For standard conversions:

Instead of continuous and auto scan conversion modes, use a sequence of fixed channel single conversions which are started in the ADC conversion complete interrupt service routine (triggered by ADCIR) after the result of the previous conversion has been read from register DAT.

b) For injected conversions:

Instead of starting injected conversions by events of CAPCOM channel CC31, or by a period match of timer T13 (XC164/XC167 only), injected conversions may be started via software in the CC31 or T13 interrupt routine by setting bit ADCRQ = 1. In order to avoid that CC31 or T13 events directly trigger injected conversions while the enable bit ADCIN = 1, the enhanced mode should be used with bit ADCTS = 0 in register ADC_CTR0.

Since the injected conversion is started by software in an interrupt service routine, no coincidence with an instruction that would read register DAT can occur.

ADC_X.004 ADC Overrun Error Generation when result is read during last cycle of conversion

When the A/D Converter operates without wait for read mode (bit ADWR = 0 in register ADC_CON or ADC_CTR0, respectively), and the result of a previous conversion #n is read from the result register (ADC_DAT) **one clock cycle** before the ADC completes the next standard conversion #n+1, the overrun error interrupt request flag ADEIR is already set to ‘1’. This means that ADEIR is set one clock cycle too early, and the result has not been overwritten. Flag ADCIR is set to ‘1’ to indicate that a new result has been written to ADC_DAT.

Workaround:
In the auto scan conversion modes, if an interrupt service routine is used to read the result from register ADC_DAT, the channel number included in the 4 msbs ADC_DAT[15:12] may be compared to the channel number of the previous conversion to identify an overrun situation.

SLEEP_X.001 Wake up trigger during last clock cycle before entry into sleep mode

When the wake up trigger (from the RTC, or from the NMI# or external interrupt input pin) occurs in a specific time window, the device will not wake up from sleep mode until a hardware reset is asserted on pin RSTIN#.

In general, this problem occurs when all of the following conditions are true:

(1) the wake up trigger occurs during the last clock cycle before the device enters sleep mode

(2) the VCO is involved in the clock generation (i.e. PLLCTRL = 11b, 10b, 01b)

The same effect may occur if (due to software malfunction) the device is about to enter sleep mode, and a watchdog timer reset occurs during the last clock cycle before the device enters sleep mode.

This problem does **not** occur in direct drive mode where the VCO is off (PLLCTRL = 00b)

Workaround 1

In order to avoid the problem (when PLLCTRL ≠ 00b), make sure that the wake up trigger only occurs after the device has already entered sleep mode:

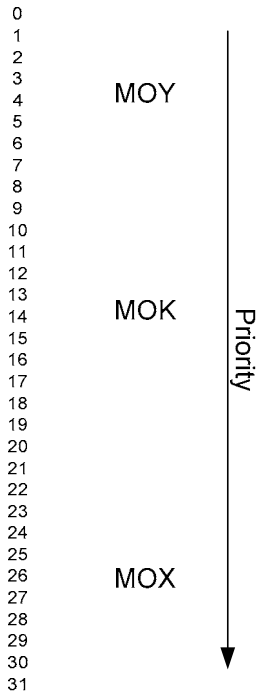
- check the RTC before entering sleep mode. If the wake up trigger will occur soon, either skip entry into sleep mode, or extend the time for the next wake up. If the RTC time interval is reprogrammed, make sure that no interrupt occurs between reprogramming and entry into sleep mode
- synchronize external wake up events such that they do not occur during the last clock cycles before sleep mode is entered. In case the CLKOUT signal is used for synchronization, please note the following:
    - the internal spike filter on the external interrupt input pins may have a maximum delay of 100 ns
    - if the output pins are switched off during sleep mode (SYSCON1.PDCFG = 01b), the last internal clock cycle is no longer visible on pin CLKOUT, i.e. the output drivers are turned off one cycle before the internal clock is turned off

Workaround 2

Before entering sleep mode, set the PLL to a configuration where it can not lock (theoretical output frequency lower than VCO base frequency). After wake-up from sleep mode, reconfigure register PLLCON to the desired clock mode, e.g.:

```
      …                       ; device is running in desired clock mode on
                              ; desired target frequency
      BCLR IEN                ; disable interrupts
Label_enter_sleep:
      EXTR #1
      MOV PLLCON, #618Eh      ; this configuration is expected not to allow
                              ; the PLL to lock
      IDLE                    ; enter sleep mode

Label_reconfigure_clock:
      EXTR #1                 ; first instruction after wake up
      MOV PLLCON, #target_val ; reconfigure target clock mode and frequency

Label_wait_for_target_clk:    ; if defined frequency is required: wait until
      MOV Rx, PLLCON          ; target clock has stabilized (PLLODIV ≠ 0Fh)
      CMP Rx, #target_val
      JMPR cc_NE, Label_wait_for_target_clk

      BSET IEN                ; enable interrupts

      …
```

TwinCAN2.005: Double Send



Under the following conditions, the problem described below will occur:

Message Object X (MO_X, lower priority) is in the process of actually being transmitted, and then the transmit request bit for Message Object Y (MO_Y, higher priority) is set via the user software. Other message objects MO_0 ... MO_n may be pending for transmission, where the priorities are MO_0 > ..> MO_Y > .. > MO_K... > MO_n > MO_X. There is no message object MO_k which has a valid transmit request and a lower message number than MO_Y (priority of MO_K < priority of MO_Y).

Symptom

MO_x is sent twice, MO_y is not sent.

Detailed Description

MO_x is sent, but neither the transmit request is cleared, nor bit INTPND is set. Also no transmit interrupt is requested for MO_x. Instead, TXRQ of MO_Y is cleared, INTPND of MO_Y is set and a transmit interrupt is requested for MO_Y (if enabled), although MO_Y has actually not been transmitted.

If NEWDAT has been set for message object MO_Y, then MO_Y is setup for transmission by the TwinCAN additionally to the INTPND, otherwise MO_K will be transmitted. If NEWDAT has not been set for message object MO_Y, then MO_Y will not be transmitted at all, but the INTPND is set.

As TxRQ has not been cleared for MO_X, MO_X will be transmitted a second time.

The gateway mode is affected as well.

In case the STT bit is used, the error will occur in any case.

Workarounds

a) Remote frames have to be forbidden; otherwise an "uncontrolled" TxRQ will be set. Set Message Object Transmit Requests (TxRQs) in order from 0 to 31, rather than at random. This could be done by implementing a software array for transmit requests TxRQs[32]. When a TxRQ bit for a Message Object has to be set, the software should write to the corresponding location within the software array. On a regular basis, all transmit requests then have to be reset and set according to the settings within the software array, to enable all message objects with TxRQs to be updated in the correct order. Between reset and set the application has to wait for the length of two CAN frames.
In order to use the gateway mode, do not set the GDFS bit in MSGFGCR.

b) Under the same software control as in workaround a) TxRQs can be set, as long as CPUUPD is set and only reset in the natural order. In this case Remote Frames are allowed.

Sample for workaround a)

In a for-loop the following takes place:

```
for (i=0;i<32;i++)
  if (TxRQ[i])   Message_Object[i].MSGCTR=0xe7ff
                 (Send this message [TxRq=Request to Send]
```

Simplified sample code:

```
interrupt (CC1_T0INT) void CC1_viTmr0(void)      // Interrupt which takes place on a regular base
{
  int ubObjNr;                                   // Message Object Number

  for (ubObjNr=0;ubObjNr<32;ubObjNr++)           // for all message objects do
  {
    if (CAN_HWOBJ[ubObjNr].uwMSGCTR & 0x2000)    // if TxRQ is still set for MOX
    {
      CAN_HWOBJ[ubObjNr].uwMSGCTR=0xDFFF;        // reset TxRQ
      if (!(CAN_HWOBJ[ubObjNr].uwMSGCTR & 0x2)) // In case the message has been transmitted before
                                                 //resetting the TxRQ, here the INTPND is detected
         TXRQs[ubObjNr]=1;                       // Set TxRQ in Softwaretable
    }
  }
  Wait the length of two CAN frames
  for (ubObjNr=0;ubObjNr<32;ubObjNr++)           // for all message objects do
  {
    if (TXRQs[ubObjNr])                          // if TxRQ is set in Softwaretable for MOX
    {
      if (CAN_HWOBJ[ubObjNr].uwMSGCTR & 0x2)     // In case the message has been transmitted before
                                                 //resetting the TxRQ, here the INTPND is detected
      {
         TXRQs[ubObjNr]=0;                       // Reset TxRQ in Softwaretable
      }
      else
      {
       CAN_HWOBJ[ubObjNr].uwMSGCTR=0xe7ff;       // Set TxRQ
       TXRQs[ubObjNr]=0;                         // reset TxRQ in Softwaretable
      }
    }
  }

} // End of function CC1_viTmr0
```

TwinCAN2.006: CPUUPD fast set/reset

In case TxRQ is set and CPUUPD in MSGCTR is set and reset within forty CAN cycles, the message data might be corrupted.

Detailed Description

If a transmit request is set for a message object, and this object is currently transmitted, while at the same time the user application set CPUUPD to change the information, then a mixture of the former information and new information may be sent. This is only the case if CPUUPD is set and reset within 40 CAN cycles.

Workarounds

a) Between setting and resetting CPUUPD there has to be a time distance of 40 CAN cycles.

b) Use MSGVAL
   The application will have the following restrictions:
   a) Remote frames will not be received by the message object during the time, where the message object is tagged invalid.
   b) You may lose the transmit interrupt for the already set transmit request.

c) In case no remote frames exist in the system: reset TxRQ and set CPUUPD, change the message object and set TxRQ and reset CPUUPD again.

TwinCAN2.007: 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 there is no other activity on the CAN module (e.g. frame reception / frame transmission on the other CAN node / 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

a) Write periodically 0xFFFF to one of the MSGCTRx registers, as this value is having no effect on the register.

b) 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 0xFFFF, in case the LEC value is unequal to 0.

TwinCAN2.008: 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.

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.

Workaround: None

SCU_X.008.1 Modification of register PLLCON while CPSYS = 1

After a hardware reset, if bit SYSCON1.CPSYS has been set to '1' by software, certain modifications of the clock generation mode in bit field **PLLCON.PLLCTRL** will result in incorrect clock control behavior. The following PLLCTRL transitions will result in incorrect behavior, and must be avoided by the user:

(1) 1xb → 0xb (x = don't care)
(2) 00b → any value other than 00b
(3) 01b → any value other than 01b

These transitions include not only explicit modifications of PLLCON by software, but also implicit modifications of PLLCON by a watchdog timer reset.

A watchdog timer reset will implicitly set PLLCON.PLLCTRL to

a) **01b** (VCO bypass with oscillator watchdog) if pin **EA#** is sampled **high** (start from internal flash)

b1) **01b** (VCO bypass with oscillator watchdog) if pin EA# is sampled low, and P0H.[7:5] = 000b or 011b, and ALE = low and RD# = high
b2) **00b** (VCO bypass without oscillator watchdog) if pin EA# is sampled low, and P0H.[7:5] = 000b or 011b, and both ALE and RD# are low
b3) **11b** (PLL mode) for all other combinations of P0H.[7:5] if pin EA# is sampled low

Note that register SYSCON1 is only cleared after a hardware reset, therefore bit CPSYS remains at '1' after a software or watchdog timer reset once it had been set by software.

If CPSYS = 1, and any of the above mentioned PLLCTRL transitions ((1) ... (3)) are executed due to an explicit or implicit write to PLLCON, the following effect will occur:

- the clock system is reconfigured according to the value written to register PLLCON, but flag PLLWRI remains set at '1'
- no further reconfigurations or modifications of the clock system are possible, neither by writing to PLLCON by software nor by a watchdog timer or software reset.

The locked state of PLLCON can only be resolved by a hardware reset.

Workaround 1

Leave SYSCON1.CPSYS = 0 (default after hardware reset).

Workaround 2

If CPSYS = 1, before performing modifications of PLLCTRL that result in state transitions other than 1Xb ↔ 1Xb, set CPSYS = 0.

Note that a watchdog timer overflow that causes state transitions of PLLCTRL from 1X ↔ 0X or from 00 ↔ 01, and which occurs while CPSYS = 1, still leads to a locked state of PLLCON.

Workaround 3

If CPSYS = 1, in order to avoid a locked state of PLLCON after a watchdog timer or software reset, select the clocking mode in PLLCTRL such that it is either identical to the status after reset (see a) .. b3) above), or such that only state transitions from 1Xb ↔ 1Xb will occur.

Example:
for single chip mode (EA# = high), e.g. the following sequence is recommended:

(1) clear CPSYS = 0
(2) program PLLCON with setting which selects PLLCTRL = 01b (same setting as default setting after reset)
(3) set CPSYS = 1

If a reset occurs at this point, the internal initialization phase will write the same PLLCTRL value that is already in effect, and therefore no problem will occur.

SCU_X.009.1 Software or Watchdog Timer Reset while Oscillator is not locked

If a software reset (instruction SRST) is executed, or a watchdog timer overflow occurs while the oscillator has not yet locked (i.e. 2048 clock cycles that exceed the XTAL1 input hysteresis have been counted), the clock system will remain in the state it had before entry into sleep mode.

This problem occurs **after wake-up from sleep mode**, if the clock at XTAL1 was switched off during sleep mode, i.e. the RTC is not running on the clock derived from XTAL1, and SRST is executed or a watchdog overflow occurs before the oscillator has locked (flag OSCLOCK = 1).

If this failure mode has been entered, write accesses to register PLLCON have no effect. Entry into sleep mode or further internal resets (SRST or WDT reset) will not change the settings of the clock system. Also, if an oscillator fail condition or PLL unlock event occurs in this mode, the clock system will remain unchanged. However, a PLL/OWD interrupt (flag PLLIR) will be generated.

The failure mode can only be resolved by a hardware reset (rising edge at pin RSTIN#).

Workaround 1

Before executing a SRST instruction, wait until the clock system is stable.
- If the PLL is used in locked mode, wait until OSCLOCK = 1, PLLLOCK = 1, and PLLWRI = 0.
- For other clock configurations, wait until OSCLOCK = 1 and PLLWRI = 0.

Make sure that the remaining watchdog time interval will cover the oscillator start-up and lock time after wake-up from sleep mode. This is typically about 6 ms for a 4 MHz crystal, and about 1 ms for a 16 MHz crystal.

Workaround 2

Use an external clock source with permanent oscillation. In this case, only 2048 clock cycles need to be considered for the remaining watchdog time interval

Workaround 3.1 (step ≥ AD only)

Use the clock from XTAL1 as clock source for the real time clock (RTC), and do not switch off the RTC during sleep mode (i.e. set RTC_CON.0 (RUN) = 1, and RTC_CON.4 (REFCLK) = 1).

Items in 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.001: BRKOUT# pulsing after Instruction Pointer Debug Event

When the ACTIVATE_PIN bit of any DxxEVT register is set and a debug event occurs, the BRKOUT# pin is supposed to be asserted as long as the debug event condition is met.

However, when software debug mode is specified as event action in addition to asserting BRKOUT#, and the jump into the monitor program takes place, BRKOUT# will toggle once for each instruction of the monitor program that is executed.

This problem does not occur when software debug mode is not specified as event action, e.g. when halt debug mode is used.

Workaround:

Set DxxEVT.ACTIVATE_PIN = 0 at the beginning of the monitor program, and set DxxEVT.ACTIVATE_PIN = 1 at the end. Note that not all toggles (first/last instruction of monitor program) can be avoided by this method.

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 ≤ 14d, CMCTR.LEVEL = 15d). This means that all generated break_now requests by the OCDS will always be accepted, independent of the CPU / 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 between the two instructions:

```
coshr #00001h
nop
mov  MSW, #00100h  (+ other variations of data16)
```

3. Deviations from Electrical and Timing Specification

Reference: XC167CI Data Sheet - V1.1, 2003-06 (see also Status Sheet)

The following restrictions should be considered:

Symbol	Parameter	Value
TAP_X.85	Maximum ambient temperature during flash programming	T_A < 85°C (**SAK** version only)

FCPUR X.162832 Frequency Limits for Flash Read Accesses

For instruction and data read accesses to the internal flash module (including programming and erase sequences), the frequency limits listed below must be considered, otherwise instructions and operands read from the internal flash may become incorrect.

The problem depends primarily on the internal frequency f_{CPU} and the number of wait states selected for the flash module (bit field WSFLASH in register IMBCTR). It is statistically more likely to occur when the ambient temperature T_A is in the upper region of the specified range, and when the internal supply voltage V_{DDI} is in the lower region of the specified range. When the problem occurs, this typically results in a class B trap (program access error, indicated by flag PACER = 1 in register TFR), while the flash status register FSR signals single or double bit errors.

For applications that exceed a specific frequency limit at a given maximum ambient temperature T_A, an additional wait state for the flash module will avoid the problem as listed in the tables below.

No problem will occur in applications that use the following settings and limits:

a) Ambient Temperature -40 °C ≤ T_A ≤ 85 °C

f_{CPU}	**Number of wait states for flash module**
$f_{CPU} \leq$ 16 MHz	0 WS (WSFLASH = 00b)
16 MHz < $f_{CPU} \leq$ 32 MHz	1 WS (WSFLASH = 01b) default after reset
f_{CPU} > 32 MHz	2 WS (WSFLASH = 10b)

b) Ambient Temperature T_A > 85 °C

f_{CPU}	**Number of wait states for flash module**
$f_{CPU} \leq$ 16 MHz	0 WS (WSFLASH = 00b)
16 MHz < $f_{CPU} \leq$ 28 MHz	1 WS (WSFLASH = 01b) default after reset
f_{CPU} > 28 MHz	2 WS (WSFLASH = 10b)

Note: f_{CPU}(max) = 40 MHz for devices marked .40F, and f_{CPU}(max) = 20 MHz for devices marked .20F.

The performance decrease due to an additional wait state depends on the individual characteristics of the software. Due to the internal instruction prefetch queue, the average performance decrease when using 1 wait state instead of 0 wait states is expected to be approximately 5%, and approximately 15% when using 2 wait states instead of 1 wait state.

4. 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 = 8FBBh ; enables several performance features

CPUCON1: reset/recommended value = 0..0 0XXX X111; 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'E000h..00'EFFFh), including IIC and SDLM module (on XC161)
- the 2 Mbyte external I/O area (20'0000h..3F'FFFFh), including the TwinCAN module (default: from 20'0000h .. 20'07FFh)

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

For further special properties of the I/O areas, see e.g. p.3-8 in the XC167 System Units User's Manual, V1.1, 2002-12.

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.H2.1 Access to Flash Module after Wake-Up from Sleep/Idle Mode or Shut-Down

In order to reduce power consumption, the devices of the XC166 family allow to selectively disable individual modules. The flash module may be disabled upon entry into Idle or Sleep mode (selection in register SYSCON1), and it will be automatically re-enabled after wake-up from idle/sleep. Like other on-chip peripheral modules, the flash module may also be shut down/turned on by software (flexible peripheral management via register SYSCON3) while not required during specific operating periods of an application. Since the transition from active to inactive state (and vice versa) requires several clock cycles, some special situations have to be considered when access to the flash is required directly after wake-up:

If during entry into **sleep** or **idle** mode where the flash module shall be disabled (bit field **PFCFG** = 01b in register **SYSCON1**),

- a wake-up event (PEC, interrupt, NMI#) occurs **before** the flash deactivation process is complete,
- and the corresponding PEC transfer or interrupt/trap routine wants to access operands or instructions in the internal flash

then the flash access is automatically delayed until the flash is ready again.

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.

FLASH_X.H3.1 Read Access to internal Flash Module with modified Margin Level

1. When the internal flash module is read (e.g. for test purposes) with bit field margin = 0001b (low level margin) in register MAR, an additional wait state must be used, i.e.
- bit field WSFLASH in register IMBCTR must be set to 10b for fcpu > 20 MHz,
- bit field WSFLASH in register IMBCTR must be 01b (default after HW reset) for fcpu < 20 MHz

2. When writing to the Margin Control Register MAR (with the Write Margin command), bit MAR.7 must be written as ‘1’.

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 = 01b** 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.
- 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 that waits until the flash is ready 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 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, 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.H1 Sleep Mode during PLL Reconfiguration

The PLL may be reconfigured by software by modifying the settings in register PLLCON. When PLLCON is written to, and the PLL was locked at that time, flag PLLLOCK in register SYSSTAT is cleared to '0' to indicate that a reconfiguration of the clock system is in progress. In addition, in order not to exceed the internal frequency limits, the clock divider K is set to 16, such that $f_{PLL} = f_{VCO}/16$. This is indicated in bit field PLLODIV in register PLLCON, which returns the value Fh when read during this reconfiguration phase. When the reconfiguration is complete, PLLLOCK = 1 and PLLODIV returns the value which was written to PLLCON.

When the PLL is about to be reconfigured (i.e. an instruction which writes to PLLCON has been executed, and no bypass mode has been selected), and sleep mode is entered (instruction IDLE is executed) before the PLL has locked on the new frequency, then, after wake-up from sleep mode, the device will stay in clock system emergency mode (flagged by SYSSTAT.PLLEM=1) with $f_{PLL} = f_{VCO}/16$ (even after the oscillator has locked again).

Therefore, it is recommended not to enter sleep mode before the PLL has locked (i.e. wait until bit OSCLOCK = 1, PLLLOCK = 1 and bit field PLLODIV **<** 0Fh)

SLEEP_X.H3 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 = 00b**), 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.

  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 is locked.

- In bypass mode with VCO on (**PLLCTRL = 01b**), 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.

  If the RTC was not running on the main oscillator, the device will wake-up and run using the internal PLL base frequency from the VCO (f_{base}/16) until the oscillator input XTAL1 is locked, and then switch to the frequency derived from the external oscillator input.

- In PLL mode with input clock from XTAL1 disconnected (**PLLCTRL = 10b**), 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 oscillator input XTAL1 is locked, 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 = 00b) 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 = 11b**), 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 oscillator input XTAL1 is locked. Then the PLL resynchronizes to the target frequency determined by the settings in register PLLCON.

  If the **RTC** is **running** on the main oscillator, the device will wake-up and stay in emergency mode using the internal PLL frequency from the VCO (f_{vco}/16). It is therefore recommended to switch to bypass mode (PLLCTRL = 00b) before entry into sleep mode (check PLLCON for its target value before executing the IDLE instruction to enter sleep 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, 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

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

The extended IIC bus data rate of 400 Kbit/s can not be used for clock frequencies f_{CPU} < 8 MHz.
When a lower CPU frequency is used system limitations (frequency accuracy and duty cycle of serial clock SCL) have to be considered.

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 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).

POWER_X.H2.1 Power Consumption during Clock System Configuration

In the following situations
(1) during and after a hardware reset (from falling edge on RSTIN# until oscillator lock)
(2) after wake-up from sleep mode until oscillator lock
(3) 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 3.75 .. 11.25 MHz.

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 (2), if the RTC is not running on the main oscillator, and case in (3) the device stops until it again receives a clock from the oscillator.

RSTOUT_X.H1 RSTOUT# driven by weak driver during HW Reset

A weak driver (see specification in Data Sheet) has been implemented on pin RSTOUT# which is driven low while RSTIN# is asserted low. After the end of the internal reset sequence, RSTOUT# operates in default mode (strong driver/sharp edge mode, i.e. POCON20.PDM3N[15:12] = 0000b). The software setting POCON20.PDM3N[15:12] = xx11b is not supported and should not be selected by software, otherwise pin 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 8.3.4 in User's Manual, volume System Units). 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 DAT read' mode is selected (bit ADWR = 1), the ADC does not acknowledge the request if the conversion result from register 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) 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 DAT read' mode is selected, register 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.H2.1 Preservation of internal RAM contents after reset

After a power-up hardware reset the RAM contents are undefined.

The contents of the on-chip RAM modules are preserved during a software reset or a watchdog timer reset.

Because a hardware reset can occur asynchronously to internal operation, it may interrupt a current write operation and so inadvertently corrupt the contents of on-chip RAM. RAM contents are preserved if the hardware reset occurs during Power-Down mode, during Sleep mode, or during Idle mode with no PEC transfers enabled.

After any reset, RAM locations in the following area are used by the internal start-up code and will therefore be overwritten:

0FBA0h … 0FC1Fh

Programs that e.g. perform RAM checksum calculations after reset should exclude this memory area.

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 (0Fh) 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 = 0Fh in register PLLCON. Emergency mode is terminated if the internal oscillator lock counter has received 2048 clock ticks from XTAL1.

If PLLCON is written in emergency mode, all settings except bypass modes (PLLCTRL = 0Xb) 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 < 0Fh before PLLCON is written. Use a timeout limit in case a permanent clock failure is present.

CC6_X.H1 CAPCOM6 Suspend Mode Entry and Exit

When a breakpoint is hit, and bit OPSEN.8/CC6SEN = 1, the CAPCOM6 module will suspend its operation while the CPU is in halted state. Depending on how fast other peripherals that are equipped with shutdown handshake will acknowledge the suspend request, the following effect will occur for the CAPCOM6 module:

- if all other peripherals have acknowledged the suspend request **before** the CAPCOM6 timers have reached 0000h, the clock for the CAPCOM6 module is stopped and the module is frozen in the current state. When program execution is resumed, the CAPCOM6 module continues to run from the state where it was stopped.
- if not all other peripherals have acknowledged the suspend request when the CAPCOM6 timers have reached 0000h, the clock for the CAPCOM6 module is stopped **after** the CAPCOM6 timers have reached 0000h. The Run Bits T12R and T13R in register TCTR0 are cleared to ‘0’, and the module is frozen in a defined state. When program execution is resumed, the Run Bits T12R and T13R must be set to ‘1’ by the application software (write accesses to registers are ignored while the module is in suspend mode).

It is therefore recommended to place the breakpoint at an appropriate position that ensures a safe stop/restart of the CAPCOM6 module, or not to suspend the CAPCOM6 module after a breakpoint (i.e. leave OPSEN.8/CC6SEN = 0).

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 (bit position 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 8003h (RTC runs, prescaler by 8 enabled, bit REFCLK = 0).

The RTC clocking mode (synchronous, asynchronous) is determined by bit SYSCON0.14/RTCCM. 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 SYSCON0.14/RTCCM = 1), it will return to asynchronous mode after a RTC reset triggered by setting bit SYSCON0.15/RTCRST = 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 SYSCON0.15/RTCRST = 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 SYSCON0.15/RTCRST = 1.

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

5. Documentation Update

- XC167 System Units – User's Manual V2.0, 2004-04:

- p. 6-56:

WDTCON X.D1 **Write access to register WDTCON**

A write access to register WDTCON in **unprotected** or **secured** mode (e.g. before execution of EINIT, or by using a special command sequence) directly

- copies bit field WDTREL of register WDTCON to the high byte of the watchdog timer register WDT
- clears the low byte of register WDT,
- and selects the WDT clock prescaler factor according to bit field WDTIN.

This means that an effective write to WDTCON has the same effect as execution of instruction SRVWDT.

Modifications in step AC (and following)

The following modifications have been performed on devices with stepping code/marking **AC** and higher:

PORTS X.D1 **Port Output Control Registers (POCONz)**

The encoding of the port driver modes in bit field PDMyN of the Port Output Control Registers POCONz (y= 0..3, z=0L, 0H, 1L, 1H, 2..4, 6, 7, 9) has been modified for weak and medium driver mode as listed below. The settings for strong driver mode remain unchanged.

Bit field POCONz.PDMyN	Mode	Notes
0011	weak driver	
0100	medium driver	1)
0101	medium driver	1)
0110	medium driver	1)
0111	reserved	2)

Notes:
1) There is no difference in the port output characteristics (driver strength, edge shape) between the 3 selections for medium driver mode (PDMyN = 010Xb, 0110b). The electrical characteristics are comparable to medium driver/sharp edge mode of the XC16x-16F step AB devices.

2) Currently, weak driver mode is selected for setting 0111b. However there will be approximately 30 µA (low level) to 80 µA (high level) additional current on Vddp per pin which is configured as output in weak driver mode. To avoid this, **use only setting 0011b for weak driver mode**.

Modifications in step AD (and future devices)

The following modifications have been performed on devices with stepping code/marking **AD** and higher:

ADC_X.D2 A/D Converter Characteristics, Conversion Time

In order to increase accuracy, the conversion time with post calibration (default after reset) is extended by $4t_{BC}$ as follows:

resolution	with post calibration	without post calibration
8-bit	t_{C8} = **44**t_{BC} + t_S+ 6t_{SYS} (2.75µs @ f_{BC} = 20MHz)	t_{C8} = 32t_{BC} + t_S+ 6t_{SYS} (2.15µs @ f_{BC} = 20MHz)
10-bit	t_{C10} = **52**t_{BC} + t_S+ 6t_{SYS} (3.15µs @ f_{BC} = 20MHz)	t_{C10} = 40t_{BC} + t_S+ 6t_{SYS} (2.55µs @ f_{BC} = 20MHz)

Note:

(1) For time critical applications, post calibration may be disabled by software (set bit CALOFF = 1 in register CTR0) after the reset calibration is finished (indicated by bit CAL = 0 in register CON1), and no conversion is in progress (indicated by bit ADBSY = 0 in register CON). This is possible in enhanced mode (bit MD = 1 in register CTR0) as well as in standard mode (MD = 0).

(2) The accuracy of the ADC depends to a great extent on the stability of the reference voltage V_{AREF}, in particular during the reset calibration phase.

(3) The ADC is currently tested with post calibration enabled. The accuracy with post calibration disabled is guaranteed by design.

RTC_X.D2 Real Time Clock Source Selection

The clock source for the RTC can be selected by software via bit REFCLK (bit position 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 8003h (RTC runs, prescaler by 8 enabled, synchronous mode, bit REFCLK = 0). As a general recommendation, bit REFCLK should be set to the desired value after any start-up (hardware, software, watchdog reset), and whenever the RTC module is reset by setting bit SYSCON0.15/RTCRST = 1.

SCU_X.D1 Oscillator gain reduction

The gain of the main oscillator (pins XTAL1/2) may be reduced via software by setting bit SYSCON.12 (OSCGRED) = 1. After reset, high gain is selected by default to guarantee safe start-up.

The gain may be reduced when the oscillations have reached their full amplitude. For standard crystals or ceramic resonators, this is typically 2^{15} (32768) cycles of f_{XTAL1} after bit OSCLOCK = 1 (and, if the PLL is used, PLLLOCK = 1) in register SYSSTAT. After reset, the internal reset sequence ensures that OSCLOCK = 1 and PLLLOCK = 1 before the first instruction of the application software is executed, unless a problem with the oscillator circuit is detected (in this case, flag PLLIR = 1).

If the RTC is not running on the main oscillator during sleep mode (i.e. the main oscillator is off during sleep mode), bit SYSCON.12 (OSCGRED) should be set to ‘0’ (high gain) before entry into sleep mode to guarantee safe oscillator start-up after wake-up from sleep.

SCU_X.D2 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.

Application hint BREAK_X.H2 (Behavior of on-chip peripherals after Break) which was valid for step AC and earlier steps is therefore removed for the Errata Sheet of step AD (and newer steps).

Note also the following documentation updates for some versions of the User’s Manual, volume System Units:

Register Bit	Function	Device
SYSCON3.12	SDLMSEN	XC161CJ-16F
SYSCON3.10	ASC1DIS	all XC16x
SYSCON3.8	CC6DIS	XC164, XC167
OPSEN.13	CANSEN	all XC16x
OPSEN.12	SDLMSEN	XC161CJ-16F
OPSEN.11	I2CSEN	XC161, XC167
OPSEN.10	ASC1SEN	all XC16x
OPSEN.8	CC6SEN	XC164, XC167
OPSEN.5	PFLSEN	all devices with program flash
OPSEN.0	ADCSEN	all XC16x

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.D4 VCO band after hardware/watchdog reset in single chip mode

From the falling edge of RSTIN# until 2048 stable oscillator clocks after the rising edge of RSTIN#, always the highest VCO band is selected (see POWER_X.H2). 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 = 2710h).