Errata Sheet

V1.9, 2005-04-12

Device	**C164CI-8E**
Marking/Step	**Step ES-CA, CA**
Package	**C-MQFP, P-MQFP-80**

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

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

Current Documentation

- **C164CI/CL, SI/SL Data Sheet, V2.0, May 2001**
- **C164CI/CL, SI/SL User's Manual, V3.1, Feb. 2002**
- **Instruction Set Manual, V2.0, Mar. 2001**

Note: Devices marked with EES- or ES are engineering samples which may not be completely tested in all functional and electrical characteristics, therefore they should be used for evaluation only.

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

Contents

Section . . . . . Page
History List/Change Summary 2
Functional Problems 5
Deviations from Electrical- and Timing Specification 26
Application Hints 27
Documentation Update 32

1 History List/Change Summary

(from step ES-BB, previous errata sheet V1.8, step: CA)

Table 1 Functional Deviations

Functional Problem	Short Description	Fixed in step	Change
ADC.11	Modifications of ADM field while bit ADST = 0		
SSC.9	Data Transmission in Slave Mode		new
CAN.7	Unexpected Remote Frame Transmission		
CAN.9	Contents of Message Objects and Mask of Last Message Registers after Reset		
CAPCOM.2	Wakeup from Idle Mode by CAPCOM2 channels CC19 or CC27	CA	
CAPCOM6.3	Write Access to CAPCOM6 Registers	CA	
RTC.2	Read RTC count registers	CA	
RTC.4	Undefined Results after write to T14		
BUS.17	Spikes on $\overline{CS}$ lines after access with $\overline{RDCS}$ and/or $\overline{WRCS}$		
BUS.18	PEC transfers after JMPR		
X9	Read Access to XPERs in Visible Mode		
OTP.7	Verify in CHM when Bootstrap Loader Mode is selected	CA	
OTP.8	Programming Voltage V_{PP} and Supply Voltage V_{DD}		
CPU.16	Data read access with MOVB [Rn], mem instruction to internal ROM/Flash/OTP		
CPU.17	Arithmetic Overflow by DIVLU instruction		
CPU.21	BFLDL/BFLDH Instructions after Write Operation to internal IRAM		
CPU.22	Z Flag after PUSH and PCALL		
PLL.3.1	Increased PLL Jitter caused by external Access		
PWRDN.1	Execution of PWRDN instruction while Pin $\overline{NMI}$ = high		
RST.4	Power on Reset	CA	
RST.13.1	Power Up with missing Clock (former name: RST.13)		name changed

Table 1 Functional Deviations (cont'd)

Functional Problem	Short Description	Fixed in step	Change
ADP.1	Oscillator in Adapt Mode	CA	

Table 2 AC/DC Deviations

AC/DC Deviations	Short Description	Fixed in step	Change
DC.AH.2	ALE active current 650 µA	CA	
DC.HYS.1	Input Hysteresis (Special Threshold) 300 mV		
DC.IOZ2.2	Input Leakage Current IOZ2 +/- 1 µA (Pin $\overline{EA}$ only)		
DC.IPDR.2	see C164 Data Sheet V2.0, May 2001, DC Characteristics (I_{PDR} = 200 + 25 x f_{OSC} [µA])		see Data Sheet
DC.P0L.2	P0 configuration current -120 µA	CA	
DC.P4L.2	P4 active current -650 µA	CA	
DC.RL.2	$\overline{RD}$/$\overline{WR}$ active current -650 µA	CA	
DC.VDD.1	see C164 Data Sheet V2.0, May 2001, Operating Condition Parameters (V_{DD} = 4.75 ... 5.5 V)		see Data Sheet
DC.VILS.1	Input low voltage (Special Threshold) 1.7 V		
DC.VIH.1	Input high voltage VIH_min on P20.5 / $\overline{EA}$		
AC.t48.1	$\overline{RDCS}$/$\overline{WRCS}$ low time (with RW-delay) 2TCL-12ns		
AC.t49.1	$\overline{RDCS}$/$\overline{WRCS}$ low time (no RW-delay) 3TCL-12ns		

Table 3 Application Hints

Hint	Short Description	Change
SSC.H1	Handling of the SSC Busy Flag (SSCBSY)	
SSC.H2	Timing of flag SSCTIR (SSC Transmit Interrupt Request)	
CAN.H1	Note on Interrupt Register behaviour of the CAN module	
EMUL.H1	Adapt Mode Setting	
OTP.H1	OTP Read access in CHM Mode delivers Zeros only	
MainOsc.H1	Oscillator Type_LP2: Negative Resistance and Start-up Reliability	
MainOsc.H2	Maximum (Type_LP2) Oscillator Frequency = 16 MHz	
OWD.H2	Oscillator Watchdog and Prescaler Mode	
ISNC.H1	Maintenance of ISNC Register	

Table 4 Documentation Update

Name	Short Description	Change
	New User's Manual available: C164CI/CL, SI/SL User's Manual, V3.1, Feb. 2002	
	New Data Sheet available: C164CI/CL, SI/SL Data Sheet, V2.0, May 2001	
PORT.D2	Output Driver Control	new
INT.D1	CCxIC Registers	
DOC_OTP.D1	Differences to Data Sheet Version 2.0, May 2001	new
ID-Registers	ID-Registers	new

2 Functional Problems

ADC.11 Modifications of ADM field while bit ADST = 0

The A/D converter may unintentionally start one auto scan single conversion sequence when the following sequence of conditions is true:

1. the A/D converter has finished a fixed channel single conversion of an analog channel n > 0 (i.e. contents of ADCON.ADCH = n during this conversion)

2. the A/D converter is idle (i.e. ADBSY = 0)

3. then the conversion mode in the ADC Mode Selection field ADM is changed to Auto Scan Single (ADM = 10b) or Continuous (ADM = 11b) mode without setting bit ADST = 1 with the same instruction

Under these conditions, the A/D converter will unintentionally start one auto scan single conversion sequence, beginning with channel n-1, down to channel number 0.

When no interrupt or PEC is servicing the A/D Conversion Complete Interrupt, interrupt request flag ADCIR will be set, and for n > 1 also the A/D Overrun Error interrupt request flag will be set, unless the wait for ADDAT read mode had been selected. When ADCON.ADWR = 1 (wait for ADDAT read), the converter will wait after 2 conversions until ADDAT is read.

In case the channel number ADCH has been changed before or with the same instruction which selected the auto scan mode, this channel number has no effect on the unintended auto scan sequence (i.e. it is not used in this auto scan sequence).

Note:

When a conversion is already in progress, and then the configuration in register ADCON is changed,

- the new conversion mode in ADM is evaluated after the current conversion
- the new channel number in ADCH and new status of bit ADST are evaluated after the current conversion when a conversion in fixed channel conversion mode is in progress, and after the current conversion sequence (i.e. after conversion of channel 0) when a conversion in an auto scan mode is in progress.

In this case, it is a specified operational behaviour that channels n-1 .. 0 are converted when ADM is changed to an auto scan mode while a fixed channel conversion of channel n is in progress (see e.g. C164CI User's Manual, V1.0, p.18-4)

Workaround:

When an auto scan conversion is to be performed, always start the A/D converter with the same instruction which sets the configuration in register ADCON.

SSC.9 Data Transmission in Slave Mode

During data reception in slave mode of the SSC module, sporadically the shift clock supplied by the external master on pin SCLK may not be properly recognized due to a synchronization problem when all of the following conditions are true:

1. the latching edge for the serial data is the **falling** edge of SCLK (i.e. both bit SSCPO = 1 and bit SSCPH = 1, or SSCPO = 0 and SSCPH = 0 in register SSCCON), and
2. the transmit buffer **SSCTB** of the slave has **not** been **written** prior to the start of the reception (initiated by the master asserting the shift clock SCLK), and
3. a specific time window (phase delay) is hit by the serial shift clock SCLK in relation to the internal system clock of the slave. Therefore, this synchronization problem will occur in particular when the slave device is clocked (on XTAL1) by an external clock generation circuit which is independent from the clock generation circuit of the master (i.e. slave and master clocks are **asynchronous**).

When the problem occurs, this results in missing bits in the character received in SSCTB, and in duplicated bits in the character transmitted on pin MRST of the slave. As a consequence, interrupt generation in the slave is delayed by the number of missed bits.

Workaround

For systems using the falling edge of SCLK as latching edge (see condition 1. above), always write to the transmit buffer SSCTB prior to any reception in slave mode of the SSC module. For the second and all following characters, e.g. write a (dummy) character to SSCTB in the receive interrupt routine, or use a PEC transfer triggered by the transmit interrupt request to write to SSCTB. In this case, the critical synchronization path is not used, and the problem will not occur.

CAN.7 Unexpected Remote Frame Transmission

Symptom:

The on-chip CAN module may send an unexpected remote frame with the identifier = 0, when a pending transmit request of a message object is disabled by software.

Detailed Description:

There are three possibilities to disable a pending transmit request of a message object (n = 1..14):

- Set CPUUPDn element
- Reset TXRQn element

- Reset MSGVALn element

Either of these actions will prevent further transmissions of message object n.

The symptom described above occurs when the CPU accesses CPUUPD, TXRQ or MSGVAL, while the pending transmit request of the corresponding message object is transferred to the CAN state machine (just before start of frame transmission). At this particular time the transmit request is transferred to the CAN state machine before the CPU prevents transmission. In this case the transmit request is still accepted from the CAN state machine. However the transfer of the identifier, the data length code and the data of the corresponding message object is prevented. Then the pre-charge values of the internal "hidden buffer" are transmitted instead, this causes to a remote frame transmission with identifier = 0 (11 bit) and data length code = 0.

This behavior occurs only when the transmit request of message object n is pending and the transmit requests of other message objects are not active (single transmit request).

If this remote frame loses arbitration (to a data frame with identifier = 0) or if it is disturbed by an error frame, it is not retransmitted.

Effects to other CAN nodes in the network:

The effect leads to delays of other pending messages in the CAN network due to the high priority of the Remote Frame. Furthermore the unexpected remote frame can trigger other data frames depending on the CAN node's configuration.

Workarounds:

- The behavior can be avoided if a message object is not updated by software when a transmission of the corresponding message object is pending (TXRQ element is set) and the CAN module is active (INIT = 0). If a re-transmission of a message (e.g. after lost arbitration or after the occurrence of an error frame) needs to be cancelled, the TXRQ element should be cleared by software as soon as NEWDAT is reset from the CAN module.

- The nodes in the CAN system ignore the remote frame with the identifier = 0 and no data frame is triggered by this remote frame.

CAN.9 Contents of Message Objects and Mask of Last Message Registers after Reset

After any reset, the contents of the CAN Message Objects and the Mask of Last Message Registers may be undefined instead of unaffected (reset value 'X' instead of 'U').

This problem depends on temperature and the length of the reset, and differs from device to device. The problem is more likely to occur at high temperature and for long hardware resets (> 100 ms).

Workaround:

Re-initialize the CAN module after each reset.

RTC.4 Undefined Results after write to T14

A write operation to T14 leads to undefined results depending on contents of T14REL.

Workaround:

Load T14REL (1st) and T14 (2nd) with the desired value.

E.g.

```
MOV   T14REL, #xxxx
MOV   T14, #xxxx
```

BUS.17 Spikes on $\overline{CS}$ Lines after access with $\overline{RDCS}$ and/or $\overline{WRCS}$

Spikes of about 5 ns width (measured at VOH = 0.9 V_{CC}) from V_{CC} to V_{SS} may occur on Port 6 lines configured as $\overline{CS}$ signals. The spikes occur on one $\overline{CSx}$ line at a time for the first external bus access which is performed via a specific BUSCONx/ADDRSELx register pair (x=1..4) or via BUSCON0 (x=0) when the following two conditions are met:

1. the previous bus cycle was performed in a **non-multiplexed** bus mode **without tristate** waitstate via a different BUSCONy/ADDRSELy register pair (y=1..4, y≠x) or BUSCON0 (y=0, y≠x) **and**
2. the previous bus cycle was a read cycle with $\overline{RDCSy}$ (bit BUSCONy.CSRENy = 1) or a write cycle with $\overline{WRCS}$ (bit BUSCONy.CSWENy = 1).

The position of the spikes is at the beginning of the new bus cycle which is performed via $\overline{CSx}$, synchronous with the rising edge of ALE and synchronous with the rising edge of $\overline{RD}$/$\overline{WR}$ of the previous bus cycle.

Potential effects on applications:

- when $\overline{CS}$ lines are used as $\overline{CE}$ signals for external memories, typically no problems are expected, since the spikes occur after the rising edge of the $\overline{RD}$ or $\overline{WR}$ signal.

- when $\overline{CS}$ lines configured as $\overline{RDCS}$ and/or $\overline{WRCS}$ are used e.g. as $\overline{OE}$ signals for external devices or as clock input for shift registers, problems may occur (temporary bus contention for read cycles, unexpected shift operations, etc.). When $\overline{CS}$ lines configured

as $\overline{\text{WRCS}}$ are used as $\overline{\text{WE}}$ signals for external devices, no problems are expected, since a tristate waitstate should be used anyway due to the negative address hold time after $\overline{\text{WRCS}}$ (t55) without tristate WS.

Workarounds:

1. Use a memory tristate WS (i.e. leave bit BUSCONy.5 = 0) in all active BUSCON registers where $\overline{\text{RD}}$/$\overline{\text{WR-CS}}$ is used (i.e. bit BUSCONy.CSRENy = 1 and/or bit BUSCONy.CSWENy = 1), **or**
2. Use Address-$\overline{\text{CS}}$ instead of $\overline{\text{RD}}$/$\overline{\text{WR-CS}}$ (i.e. leave bits BUSCONy[15:14] = 00b) for all BUSCONy registers where a non-multiplexed bus without tristate WS is configured (i.e. bit BUSCONy.5 = 1).

BUS.18 PEC Transfers after JMPR instruction

Problems may occur when a PEC transfer immediately follows a taken JMPR instruction when the following sequence of 4 conditions is met (labels refer to following examples):

1. in an instruction sequence which represents a loop, a jump instruction (Label_B) which is capable of loading the jump cache (JMPR, JMPA, JB/JNB/JBC/JNBS) is taken
2. the target of this jump instruction directly is a JMPR instruction (Label_C) which is also taken and whose target is at address A (Label_A)
3. a PEC transfer occurs immediately after this JMPR instruction (Label_C)
4. in the following program flow, the JMPR instruction (Label_C) is taken a second time, and no other JMPR, JMPA, JB/JNB/JBC/JNBS or instruction which has branched to a different code segment (JMPS/CALLS) or interrupt has been processed in the meantime (i.e. the condition for a jump cache hit for the JMPR instruction (Label_C) is true)

In this case, when the JMPR instruction (Label_C) is taken for the second time (as described in condition 4 above), and the 2 words stored in the jump cache (word address A and A+2) have been processed, the word at address A+2 is erroneously fetched and executed instead of the word at address A+4.

Note: the problem does not occur when

- the jump instruction (Label_C) is a JMPA instruction
- the program sequence is executed from internal Flash/ROM/OTP

Example1:

```
Label_A:  instruction x              ; Begin of Loop
          instruction x+1
```

```
.....
Label_B:  JMP Label_C; JMP may be any of the following jump instructions:
             ; JMPR cc_zz, JMPA cc_zz, JB/JNB/JBC/JNBS
             ; jump must be taken in loop iteration n
             ; jump must not be taken in loop iteration n+1
.....
Label_C:  JMPR cc_xx, Label_A     ; End of Loop
             ; instruction must be JMPR (single word instruction)
             ; jump must be taken in loop iteration n and n+1
             ; PEC transfer must occur in loop iteration n
```

Example2:

```
Label_A:  instruction x               ; Begin of Loop1
          instruction x+1
.....
Label_C:  JMPR cc_xx, Label_A         ; End of Loop1, Begin of Loop2
          ; instruction must be JMPR (single word instruction)
          ; jump not taken in loop iteration n-1, i.e. Loop2 is entered
          ; jump must be taken in loop iteration n and n+1
          ; PEC transfer must occur in loop iteration n
.....
Label_B:  JMP Label_C                 ; End of Loop2
          ; JMP may be any of the following jump instructions:
          ;   JMPR cc_zz, JMPA cc_zz, JB/JNB/JBC/JNBS
          ; jump taken in loop iteration n-1
```

A code sequence with the basic structure of Example1 was generated e.g. by a compiler for comparison of double words (long variables).

Workarounds:

- use a JMPA instruction instead of a JMPR instruction when this instruction can be the direct target of a preceding JMPR, JMPA, JB/JNB/JBC/JNBS instruction, **or**
- insert another instruction (e.g. NOP) as branch target when a JMPR instruction would be the direct target of a preceding JMPR, JMPA, JB/JNB/JBC/JNBS instruction, **or**
- change the loop structure such that instead of jumping from Label_B to Label_C and then to Label_A, the jump from Label_B directly goes to Label_A.

Notes on compilers:

In the **Hightec** compiler beginning with version Gcc 2.7.2.1 for SAB C16x - V3.1 Rel. 1.1, patchlevel 5, a switch -m bus18 is implemented as workaround for this problem. In

addition, optimization has to be set at least to level 1 with -u1.

The **Keil** C compiler versions < V4.0 and run time libraries do not generate or use instruction sequences where a JMPR instruction can be the target of another jump instruction, i.e. the conditions for this problem do not occur.

In V4.0, the problem may occur when optimize (size or speed, 7) is selected. Lower optimization levels than 7 are not affected.

In V4.01, a new directive FIXPEC is implemented which avoids this problem.

In the **TASKING** C166 Software Development Tools, the code sequence related to problem BUS.18 can be generated in Assembly. The problem can also be reproduced in C-language by using a particular sequence of GOTOs.

With V6.0r3, TASKING tested all the Libraries, C-startup code and the extensive set of internal test-suite sources and the BUS.18 related code sequence appeared to be NOT GENERATED.

To prevent introduction of this erroneous code sequence, the TASKING Assembler V6.0r3 has been extended with the CHECKBUS18 control which generates a WARNING in the case the described code sequence appears. When called from within EDE, the Assembler control CHECKBUS18 is automatically 'activated'.

X9 Read Access to XPERs in Visible Mode

The data of a read access to an XBUS-Peripheral (CAN) in Visible Mode (SYSCON.1 = 1) is not driven to the external bus. PORT0 is tristated during such read accesses.

Note that in Visible Mode PORT1 will drive the address for an access to an XBUS-Peripheral, even when only a multiplexed external bus is enabled.

OTP.8 Programming Voltage V_{PP} and Supply Voltage V_{DD}

To guarantee a proper internal OTP programming the tolerances for supply voltage VDD and programming voltage VPP are reduced.

The following values have been successfully tested and can be recommended:

Supply Voltage V_{DD}	Programming Voltage V_{PP}
4.35V - 4.65V	11.35V - 11.65V
4.85V - 5.15V	11.8V - 12.1V

Application Hints:

- Devices formerly programmed with these recommended values can be used without restriction.
- Software for OTP programming tools should be adapted to one of these recommended values
  (V_{DD} = 4.85V - 5.15V and V_{PP} = 11.8 - 12.1V can still be used after the problem is fixed in the future).

CPU.16 Data read access with MOVB [Rn], mem instruction to internal ROM/Flash/ OTP

When the MOVB [Rn], mem instruction (opcode 0A4h) is executed, where

1. mem specifies a direct 16-bit byte operand address in the internal ROM/Flash memory,

AND

2. [Rn] points to an **even** byte address, while the contents of the word which includes the byte addressed by mem is **odd**,

OR

[Rn] points to an **odd** byte address, while the contents of the word which includes the byte addressed by mem is **even**

the following problem occurs:

a) when [Rn] points to **external** memory or to the **X-Peripheral** (XRAM, CAN, etc.) address space, the data value which is written back is always 00h

b) when [Rn] points to the **internal** RAM or SFR/ESFR address space,

- the (correct) data value [mem] is written to [Rn]+1, i.e. to the **odd** byte address of the selected word in case [Rn] points to an **even** byte address,

- the (correct) data value [mem] is written to [Rn]-1, i.e. to the **even** byte address of the selected word in case [Rn] points to an **odd** byte address.

Workaround:

When mem is an address in internal ROM/Flash/OTP memory, substitute instruction

```
MOVB  [Rn], mem e.g. by  MOV    Rm, #mem
                         MOVB   [Rn], [Rm]
```

Notes on compilers:

- the **Keil** C166 Compiler V3.10 has been extended by the directive FIXROM which avoids accesses to 'const' objects via the instruction MOVB [Rn], mem.

- the **Tasking** compiler provides a workaround for this problem from version V6.0r2 on

CPU.17 Arithmetic Overflow by DIVLU instruction

For specific combinations of the values of the dividend (MDH,MDL) and divisor (Rn), the Overflow (V) flag in the PSW may not be set for unsigned divide operations, although an overflow occurred.

Table 5 Example

MDH	MDL	:	Rn	=	MDH MDL	Notes
$F0F0_H$	$0F0F_H$	:	$F0F0_H$	=	$FFFF\ FFFF_H$	but no Overflow indicated ! (result with 32-bit precision: 1 0000h)
The same malfunction appears for the following combinations:						
$n0n0_H$	$0n0n_H$	:	$n0n0_H$	=	$FFFF\ FFFF_H$	
$n00n_H$	$0nn0_H$	:	$n00n_H$	=	$FFFF\ FFFF_H$	
$n000_H$	$000n_H$	:	$n000_H$	=	$FFFF\ FFFF_H$	
$n0nn_H$	$0nnn_H$	:	$n0nn_H$	=	$FFFF\ FFFF_H$	n means any Hex Digit between 8 ... F

i.e. all operand combinations where at least the most significant bit of the dividend (MDH) and the divisor (Rn) is set.

In the cases where an overflow occurred after DIVLU, but the V flag is not set, the result in MDL is equal to FFFFh.

Workaround:

Skip execution of DIVLU in case an overflow would occur, and explicitly set V = 1.

```
;E.g.:   CMP    Rn, MDH
         JMPR   cc_ugt, NoOverflow  ; no overflow if Rn > MDH
         BSET   V                   ; set V = 1 if overflow would occur
         JMPR   cc_uc, NoDivide     ; and skip DIVLU
NoOverflow:
         DIVLU  Rn
NoDivide:
         ...          ; next instruction, may evaluate correct V flag
```

Notes on compilers:

- the **KEIL** C compiler, run time libraries and operating system RTX166 do not generate or use instruction sequences where the V flag in the PSW is tested after a DIVLU instruction.
- with the **TASKING** C166 compiler, for the following intrinsic functions code is generated which uses the overflow flag for minimizing or maximizing the function result after a division with a DIVLU:

  _div_u32u16_u16()
  _div_s32u16_s16()
  _div_s32u16_s32()

  Consequently, an incorrect overflow flag (when clear instead of set) might affect the result of one of the above intrinsic functions but only in a situation where no correct result could be calculated anyway. These intrinsics first appeared in version 5.1r1 of the toolchain.

  Libraries: not affected

CPU.21 BFLDL/BFLDH Instructions after Write Operation to internal IRAM

The result of a BFLDL/BFLDH (= BFLDx) instruction may be incorrect if the following conditions are true at the same time:

1. the previous 'instruction' is a PEC transfer which writes to IRAM, or any instruction with result write back to IRAM (addresses 00'F200$_H$ ... 00'FDFF$_H$ for 3 Kbyte module, 00'F600$_H$ ... 00'FDFF$_H$ for 2 Kbyte module, or 00'FA00$_H$ ... 00'FDFF$_H$ for 1 Kbyte module). For further restrictions on the destination address see case (a) or case (b) below.
2. the BFLDx instruction immediately follows the previous instruction or PEC transfer within the instruction pipeline ('back-to-back' execution), i.e. decode phase of BFLDx and execute phase of the previous instruction or PEC transfer coincide. This situation typically occurs during program execution from internal program memory (ROM/OTP/Flash), or when the instruction queue is full during program execution from external memory
3. the 3rd byte of BFLDx (= ***#mask8*** field of BFLD**L** or ***#data8*** field of BFLD**H**) and the destination address of the previous instruction or PEC transfer match in the following way:
   a. value of #mask8 of BFLDL or #data8 of BFLDH = 0F**y**$_H$ (**y** = 0..F$_H$),
      and the previous instruction or PEC writes to (the low and/or high byte of) GPR R**y** or the memory address of R**y** (determined by the context pointer CP) via any addressing mode.

b. value of #mask8 of BFLDL or #data8 of BFLDH = **00_H ... $0EF_H$**,
and the lower byte **v_L** of the **contents v** of the IRAM location or (E)SFR or GPR which is read by BFLDx is **$00_H \leq v_L \leq 7F_H$**
and the previous instruction or PEC transfer writes to the (low and/or high byte of) the specific bit-addressable IRAM location **$00'FD00_H + 2\ v_L$**
(i.e. the 8-bit offset address of the location in the bit-addressable IRAM area ($00'FD00_H$ ... $00'FDFF_H$) equals **v_L**)

When the problem occurs, the actual result (all 16 bits) of the BFLDx instruction is bitwise ORed with the (byte or word) result of the previous instruction or PEC transfer.

Notes:

Write operations in the sense of the problem description include implicit write accesses caused by

- auto-increment operations of the PEC source or destination pointers (which are located on $00'FCE0_H$ ... $00'FCFE_H$ in IRAM)
- post-increment/pre-decrement operations on GPRs (addressing modes with [R+] or [-R])
- write operations on the system stack (which is located in IRAM).

In case **PEC write operations** to IRAM locations which match the above criteria (bit-addressable or active register bank area, PEC pointers not overlapping with register bank area) can be **excluded**, the problem will **not** occur when the instruction preceding BFLDx in the dynamic flow of the program is one of the following instructions (which do not write to IRAM):

NOP
ATOMIC, EXTx
CALLA/CALLI/JBC/JNBS when branch condition = false
JMPx, JB, JNB
RETx (except RETP)
CMP(B) (except addressing mode with [Rwi+]), BCMP
MULx, DIVx
IDLE, PWRDN, DISWDT, SRVWDT, EINIT, SRST

For implicit IRAM write operations caused by **auto-increment operations of the PEC source or destination pointers**, the problem can only occur if the value of #mask8 of BFLDL or #data8 of BFLDH = $0Fy_H$ (**y** = $0..F_H$), and the range which is covered by the context pointer CP (partially or completely) overlaps the PEC source and destination pointer area ($00'FCE0_H$ ... $00'FCFE_H$), and the address of the source or destination

pointer which is auto-incremented after the PEC transfer is equal to the address of GPR R**y** (included in case 3a).

For **system stack write operations**, the problem can only occur if the system stack is located in the bit-addressable portion of IRAM ($00'FD00_H$ ... $00'FDFF_H$), or if the system stack can overlap the register bank area (i.e. the register bank area is located below the system stack, and the distance between the contents of the context pointer CP and the stack pointer SP is $\leq 20_H$).

Workaround 1:

When a critical instruction combination or PEC transfer to IRAM can occur, then substitute the BFLDx instruction by

- an equivalent sequence of single bit instructions. This sequence may be included in an uninteruptable ATOMIC or EXTEND sequence to ensure completion after a defined time.
- an equivalent byte- or word MOV or logical instruction.

Note that byte operations to SFRs always clear the non-addressed complementary byte.

Note that protected bits in SFRs are overwritten by MOV or logical instructions.

Workaround 2:

When a critical instruction combination occurs, and **PEC write operations** to IRAM locations which match the above criteria (bit-addressable or active register bank area) **can be excluded**, then rearrange the BFLDx instruction within the instruction environment such that a non-critical instruction sequence is generated.

Workaround 3:

When a critical instruction combination or PEC transfer to IRAM can occur, then

- replace the BFLDx instruction by the instruction sequence
  ATOMIC #1
  BFLDx

  This means e.g. when BFLDx was a branch target before, ATOMIC # 1 is now the new branch target.

In case the BFLDx instruction is included at position **n** in an ATOMIC or EXTEND sequence with range operator #**m** (n, m = 2..4, n ≤ m), then

- insert (repeat) the corresponding ATOMIC or EXTEND instruction at position **n** with range operator #**z** where **z** = (**m - n**) + 1

Table 6 Range of original ATOMIC/EXTEND statement

	Range of original ATOMIC/EXTEND statement			
Position of BFLDx within ATOMIC/ EXT... sequence	**1**	**2**	**3**	**4**
1	no problem / no workaround	no problem / no workaround	no problem / no workaround	no problem / no workaround
2	- -	z = 1	z = 2	z = 3
3	- -	- -	z = 1	z = 2
4	- -	- -	- -	z = 1

- - : case can not occur

Note on devices with power management and register RSTCON:

for unlock sequences, which are sequences of four instructions operating on ESFRs SYSCON1/2/3 and RSTCON and which are included in an EXTR #4 sequence, **this workaround must not be implemented when the 4th instruction is a BFLDx instruction**, otherwise the unlock sequence will not work correctly (in fact unlock sequences do not require a workaround).

Tool Support

The **Keil** C166 Compiler V3.xx generates BFLD instructions only in the following cases:

- when using the _bfld_ intrinsic function.
- at the beginning of interrupt service routines, when using #pragma disable
- at the end of interrupt service routines, when using the chip bypass directive FIX166.

The C166 Compiler V4.xx uses the BFLD instruction to optimize bit-field struct accesses. Release C166 V4.10 offers a new directive called FIXBFLD that inserts an ATOMIC #1 instruction before every BFLD instruction that is not enclosed in an EXTR sequence. Detailed information can be found in the C166\HLP\RELEASE.TXT of C166 Version 4.10.

The C166 Run-Time Library for C166 V3.xx and V4.xx uses BFLD instructions only in the START167.A66 file. This part of the code should be not affected by the CPU.21 problem but should be checked by the software designer.

The RTX166 Full Real-Time Operating system (any version) does not use BFLD instructions.

For RTX166 Tiny, you should rebuild the RTX166 Tiny library with the SET FIXBFLD = 1 directive. This directive is enabled in the assembler source file RTX166T.A66. After change of this setting rebuild the RTX166 Tiny library that you are using in your application.

The **Tasking** support organization provides a v7.0r1 A166 Assembler (build 177) including a check for problem CPU.21 with optional pec/no_pec feature. This assembler version can also be used to check code which was generated with previous versions of the Tasking tool chain. A v7.0r1 C166 Compiler (build 368) offering a workaround for problem CPU.21 is also available from Tasking.

The scan tool **aiScan21** analyzes files in hex format plus user-supplied additional information (locator map file, configuration file), checks whether they may be affected by problem CPU.21, and produces diagnostic information about potentially critical instruction sequences. This tool is included in AP1628 'Scanning for Problem CPU.21' on: **www.infineon.com/cgi/ecrm.dll/ecrm/scripts/prod_cat.jsp?oid=-8984**

AP1628 Description: **www.infineon.com/cgi/ecrm.dll/ecrm/scripts/public_download.jsp?oid=18484&parent_oid=-8984**
AP1628 Software: **www.infineon.com/cgi/ecrm.dll/ecrm/scripts/public_download.jsp?oid=18468&parent_oid=-8984**

CPU.22 Z Flag after PUSH and PCALL

The Z flag in the PSW is erroneously set to '1' by ***PUSH reg*** or ***PCALL reg, rel*** instructions when all of the following conditions are true:

a) for ***PUSH reg*** instructions:

- the contents of the high byte of the GPR or (E)SFR which is pushed is 00_H, and
- the contents of the low byte of the GPR or (E)SFR which is pushed is > 00_H, and
- the contents of GPR Rx is odd, where x = 4 msbs of the 8-bit 'reg' address of the pushed GPR or (E)SFR

Examples:

```
PUSH  R1   ;(coding: F1 EC):
           ;  incorrect setting of Z flag if contents of R15 is odd,
           ;  and 00FFh ≥ contents of R1 ≥ 0001h
PUSH  DPP3 ;(coding: 03 EC):
```

```
;  incorrect setting of Z flag if contents of R0 is odd,
;  and 00FFh ≥ contents of DPP3 ≥ 0001h
```

b) for ***PCALL reg, rel*** instructions:

- when the contents of the high byte of the GPR or SFR which is pushed is 00_H, and
- when the contents of the low byte of the GPR or SFR which is pushed is odd

This may lead to wrong results of instructions following PUSH or PCALL if those instructions explicitly (e.g. BMOV .. , Z; JB Z, ..; ..) or implicitly (e.g. JMP cc_Z, ..; JMP cc_NET, ..; ..) evaluate the status of the Z flag before it is newly updated.

Note: Some instructions (e.g. CALL, ..) have no effect on the status flags, such that the status of the Z flag remains incorrect after a PUSH/PCALL instruction until an instruction that correctly updates the Z flag is executed.

Example:

```
   PUSH   R1           ; incorrect setting of Z flag if R15 is odd
   CALL   proc_xyz     ; Z flag remains unchanged
   ...                 ;  (is a parameter for proc_xyz)
   ...
proc_xyz:
   JMP    cc_Z,end_xyz ; Z flag evaluated with incorrect setting
   ...
end_xyz:
```

Effect on Tools:

- The **Hightec** C166 tools (all versions) don't use the combination of PUSH/PCALL and the evaluation of the Z flag. Therefore, these tools are not affected.

- The code generated by the **Keil** C166 Compiler evaluates the Z flag only after MOV, CMP, arithmetic, or logical instructions. It is never evaluated after a PUSH instruction. PCALL instructions are not generated by the C166 Compiler.

  This has been checked with all C166 V3.xx and V4.xx compiler versions. Even the upcoming V5.xx is not affected by the CPU.22 problem.

  The assembler portions of the C166 V3.xx and V4.xx Run-Time Libraries, the RTX166 Full and TX166 Tiny Real Time Operating system do also not contain any evaluation of the Z flag after PUSH or PCALL.

- The **TASKING** compiler V7.5r2 never generates a PCALL instruction, nor is it used in the libraries.

  The PUSH instruction is only used in the entry of an interrupt frame, and sometimes on exit of normal functions. The zero flag is not a parameter or return value, so this does not give any problems.

  Previous versions of TASKING tools:
  V3.x and higher are not affected, versions before 3.x are most likely not affected. Contact TASKING when using versions before V3.x.

  Since code generated by the C166 compiler versions mentioned before is not affected, analysis and workarounds are only required for program parts written in assembler, or instruction sequences inserted via inline assembly.

Workaround (for program parts written in assembler)**:**

Do not evaluate the status of the Z flag generated by a PUSH or PCALL instruction. Instead, insert an instruction that correctly updates the PSW flags, e.g.

```
    PUSH   reg
    CMP    reg, #0      ; note:
                        ;  CMP additionally modifies the C and V flags,
                        ;  while PUSH or MOV leaves them unaffected
    JMPR   cc_Z, label_1; implicitly tests Z flag
```

or

```
    PCALL  reg, procedure_1
    ...
procedure_1:
    MOV    ONES, reg
    JMPR   cc_NET, label_1 ; implicitly tests flags Z and E
```

Hints for Detection of Critical Instruction Combinations

Whether or not an instruction following PUSH reg or PCALL reg, rel actually causes a problem depends on the program context.

In most cases, it will be sufficient to just analyze the instruction following PUSH or PCALL. In case of PCALL, this is the instruction at the call target address.

Support Tool for Analysis of Hex Files

For complex software projects, where a large number of assembler source (or list) files would have to be analyzed, Infineon provides a tool aiScan22 which scans hex files for critical instruction sequences and outputs diagnostic information. This tool is available as part of the Application Note ap1679 'Scanning for Problem CPU.22' on the 16-bit microcontroller internet pages of Infineon Technologies:

www.infineon.com/16-bit-microcontrollers

direct links:

AP1679 Description: **www.infineon.com/cmc_upload/documents/040/841/ap1679_v1.1_2002_05_scanning_cpu22.pdf** and
AP1679 Software: **www.infineon.com/cgi/ecrm.dll/ecrm/scripts/public_download.jsp?oid=40840&parent_oid=-8984**

Individual Analysis of Assembler Source Code

With respect to problem CPU.22, all instructions of the C166 instruction set can be classified into the following groups:

- Arithmetic/logic/data movement instructions as successors of PUSH/PCALL (correctly) modify the condition flags in the PSW according to the result of the operation.

  These instructions may only cause a problem if the PSW is a source or source/ destination operand:
  ADD/B, ADDC/B, CMP/B, CMPD1/2, CMPI1/2, SUB/B, SUBC/B
  AND/B, OR/B, XOR/B
  ASHR
  MOV/B, MOVBZ/MOVBS
  SCXT
  PUSH, PCALL ➔ analysis must be repeated for successor of PUSH/PCALL

- The following instructions (most of them with immediate or register (Rx) addressing modes) can never cause a problem:
  CPL/B, NEG/B
  DIV/U, DIVL/U, MUL/U
  SHL/SHR, ROL/ROR
  PRIOR
  POP
  RETI ➔ updates complete PSW with stacked value
  RETP ➔ updates condition flags
  PWRDN ➔ program restarts after reset

SRST ➔ program restarts

- Conditional branch instructions which may evaluate the Z flag:
  JB/JNB Z, rel ; directly evaluates Z flag
  CALLA/CALLI, JMPA/JMPI/JMPR with the following condition codes
  cc_Z, cc_EQ, cc_NZ, cc_NE
  cc_ULE, cc_UGT, cc_SLE, cc_SGT
  cc_NET
  ➔ For these branch conditions, the branch may be performed in the wrong way.
  ➔ For other branch conditions, the branch target as well as the linear successor of the branch instruction must be analyzed (since these branch instruction don't modify the PSW flags).

- For **instructions that have no effect on the condition flags** and that don't evaluate the Z flag, the instruction that follows this instruction must be analyzed.
  These instructions are:
  NOP
  ATOMIC, EXTxx
  DISWDT, EINIT, IDLE, SRVWDT
  CALLR, CALLS, JMPS ➔ branch target must be analyzed
  RET, RETS ➔ return target must be analyzed
  (value pushed by PUSH/PCALL = return IP, Z flag contains information whether intra-segment target address = 0000_H or not)
  TRAP ➔ both trap target and linear successor must be analyzed, since Z flag may be incorrect in PSW on stack as well as in PSW at entry of trap routine

- For **bit modification instructions**, the problem may only occur if a source bit is the Z flag, and/or the destination bit is in the PSW, but not the Z flag.
  These instructions are:
  BMOV/BMOVN
  BAND/BOR/BXOR
  BCMP
  BFLDH
  BFLDL ➔ problem only if bit 3 of @@ mask = 0, i.e. if Z is not selected
  BCLR/BSET ➔ problem only if operand is not Z flag
  JBC/JNBS ➔ wrong branch if operand is Z flag

PLL.3.1 Increased PLL Jitter caused by external Access

Problem description:

In systems where the PLL is used for generation of the CPU clock frequency, the PLL jitter can increase under certain circumstances and exceed the specified value.

The value of the additional jitter is not a fixed one but it depends on the kind of activity on the external bus or output pins. The additional jitter increases with the number of output/bus pins which are switched at the same time to a new voltage level because the problem is caused by noise on the on-chip power supply.

The capacitive load of the used pins has also an influence to the additional jitter. A high capacitive load can increase the additional jitter.

Effects to the system:

All PLL factors are affected (PLL factor 1.5 / 2 / 2.5 / 3 / 4 / 5). The PLL jitter increases when an access to the external bus or output pins is performed. This phenomenon has no influence to direct drive-, prescaler-, and SDD mode.

The problem does not affect the functionality of the CPU and the on-chip peripherals.

The additional jitter has the maximum effect if only one TCL is considered (period jitter). This can have an influence to the bus timings with one TCL.

The additional jitter decreases with the number of consecutive TCLs (accumulated jitter). The accumulated jitter has an effect on timings with more than one TCL (certain bus timings, CAN bus timing, serial interfaces).

New value of the additional jitter

For an accumulated period of N*TCL the new maximum jitter D_N[ns], which exceeds the specified value in the Data Sheet, is computed using the formula:

$$D_N[ns] = \pm[(50 / f_{CPU} + 164 / f_{CPU}^2 - 0.04) * (N / 2 - 1) + 1180 / f_{CPU}^2 + 0.2]$$

Were N = number of consecutive TCL, f_{CPU} = CPU frequency in MHz.
This approximated formula is valid for $1 \le N \le 12$ and $10 \text{ MHz} \le f_{CPU} \le 25 \text{ MHz}$.

For all accumulated periods longer than 12 TCL the N=12 value can be used; see figure below.

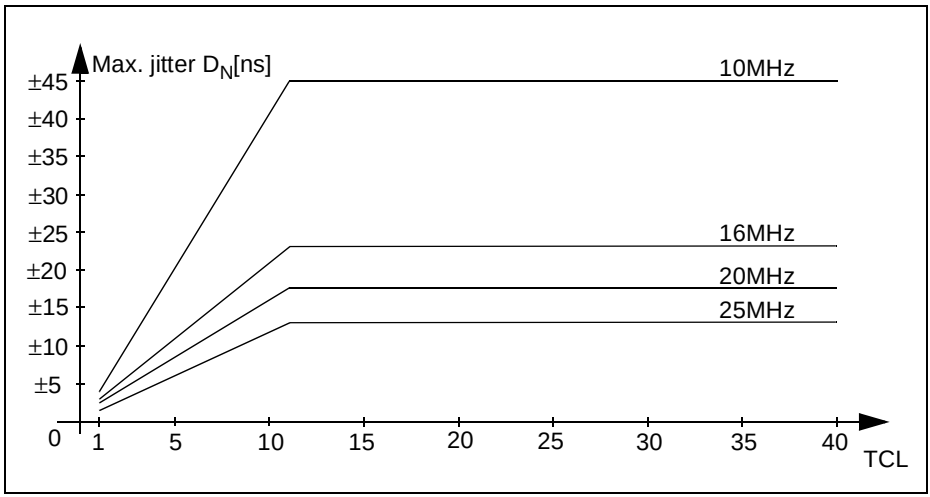


Figure 1 Additional Jitter

Workaround:

In case that the additional jitter causes problems in the system which cannot be relaxed with software adapted timing parameters, then direct drive or prescaler mode can be used. If the maximum frequency of the on-chip oscillator (16 MHz) is not sufficient then an external clock generator can be used.

PWRDN.1 Execution of PWRDN Instruction while pin $\overline{\text{NMI}}$ = high

When instruction PWRDN is executed while pin $\overline{\text{NMI}}$ is at a high level, power down mode should not be entered, and the PWRDN instruction should be ignored. However, under the conditions described below, the PWRDN instruction may not be ignored, and no further instructions are fetched from external memory, i.e. the CPU is in a quasi-idle state. This problem will only occur in the following situations:

a) the instructions following the PWRDN instruction are located in external memory, and a multiplexed bus configuration with memory tristate waitstate (bit MTTCx = 0) is used, or

b) the instruction preceeding the PWRDN instruction writes to external memory or an XPeripheral (e.g. CAN or XRAM), and the instructions following the PWRDN instruction

are located in external memory. In this case, the problem will occur for any bus configuration.

Note: The on-chip peripherals are still working correctly, in particular the Watchdog Timer will reset the device upon an overflow. Interrupts and PEC transfers, however, can not be processed. In case $\overline{NMI}$ is asserted low while the device is in this quasi-idle state, power down mode is entered.

Workaround:

Ensure that no instruction which writes to external memory or an XPeripheral preceeds the PWRDN instruction, otherwise insert e.g. a NOP instruction in front of PWRDN. When a muliplexed bus with memory tristate waitstate is used, the PWRDN instruction should be executed out of internal RAM.

RST.13.1 Power Up with missing Clock

While no clock is provided at XTAL1 **and**
- RSTIN = High (while $V_{DD} \leq 2V$) **or**
- power off/on recovery time is below 1 second (beginning from ≤ 0.3V at all pins)

then following settings can be found:

- The realtime clock registers are not cleared (T14, T14REL, RTCH, RTCL).
- The slow down divider (SDD) can be selected as f_{CPU} clock source (no oscillator watchdog function possible - the device will stop while external clock is missing)

Workaround:

None

3 Deviations from Electrical- and Timing Specification

Problem Short Name	Parameter	Symbol	Limit Values		Unit	Test Condition
			min.	max.		
DC.HYS.1[1]	Input Hysteresis (Special Threshold)	HYS	**300** [1] instead of 400	–	mV	–
DC.IOZ2.2	Input Leakage Current (Pin $\overline{EA}$ only)	I_{OZ2}	–	± **1000** instead of ±500	nA	0.45 V < V_{IN} < V_{DD}
DC.VILS.1[2]	Input low voltage (Special Threshold)	V_{ILS}	- 0.5	**1.7** [2] instead of 2.0	V	–
DC.VIH.1	Input high voltage (Pin $\overline{EA}$ only)	V_{IH}	**0.7 V_{DD}** instead of 0.2 V_{DD} + 0.9	V_{DD} + 0.5	V	–
AC.t48.1	$\overline{RDCS}$/$\overline{WRCS}$ low time (with RW-delay)	t_{48}	**38** + tc instead of 40 + tc	-	ns	–
AC.t49.1	$\overline{RDCS}$/$\overline{WRCS}$ low time (without RW-delay)	t_{49}	**63**+tc instead of 65 + tc	-	ns	–

1) If $f_{CPU} \leq 16$ MHz the specified value is guaranteed by design.

2) For $V_{DD} > 4.4$ V and $f_{CPU} \leq 16$ MHz the specified value is guaranteed by design.

Note: Timing t28: Parameter description and test changed from 'Address hold after $\overline{RD}$/$\overline{WR}$' to 'Address hold after $\overline{WR}$'. It is guaranteed by design that read data are internally latched by the controller before the address changes.

Note: During reset and adapt mode (in external bus mode - pin $\overline{EA}$ = LOW), the internal pull-ups on P4[3:0] are active, independent whether the respective pins are used for $\overline{CS}$ function after reset or not.

4 Application Hints

SSC.H1 Handling of the SSC Busy Flag (SSCBSY)

In master mode of the High-Speed Synchronous Serial Interface (SSC), when register SSCTB has been written, flag SSCBSY is set to '1' when the baud rate generator generates the next internal clock pulse. The maximum delay between the time SSCTB has been written and flag SSCBSY=1 is up to 1/2 bit time. SSCBSY is cleared 1/2 bit time after the last latching edge.

When polling flag SSCBSY after SSCTB has been written, SSCBSY may not yet be set to '1' when it is tested for the first time (in particular at lower baud rates). Therefore, e.g. the following alternative methods are recommended:

- test flag SSCRIR (receive interrupt request) instead of SSCBSY (in case the receive interrupt request is not serviced by CPU interrupt or PEC), e.g.

```
loop: BCLR SSCRIR              ;clear receive interrupt request flag
      MOV SSCTB, #xyz          ;send character
wait_tx_complete:
      JNB SSCRIR, wait_tx_complete  ;test SSCRIR
      JB SSCBSY, wait_tx_complete   ;test SSCBSY to achieve original
                                    ;timing (SSCRIR may be set 1/2 bit

                                    ;time before SSCBSY is cleared)
```

- use a software semaphore bit which is set when SSCTB is written and is cleared in the SSC receive interrupt routine

SSC.H2 Timing of flag SSCTIR (SSC Transmit Interrupt Request)

In master mode, the timing of SSCTIR is as follows:

When SSCTB has been written while the transmit shift register was empty (and the SSC is enabled), flag SSCTIR is set to '1' directly after completion of the write operation, independent of the selected baud rate. When the transmit shift register is not empty when SSCTB was written, SSCTIR is set to '1' after the last latching edge of SCLK (= 1/2 bit time before the first shifting edge of the next character). See also e.g. C167CR User's Manual V3.1, p. 12-5.

The following diagram shows these relations in an example for a data transfer in master mode with SSCPO = 0 and SSCPH = 0. It is assumed that the transmit shift register is empty at the time the first character is written to SSCTB:

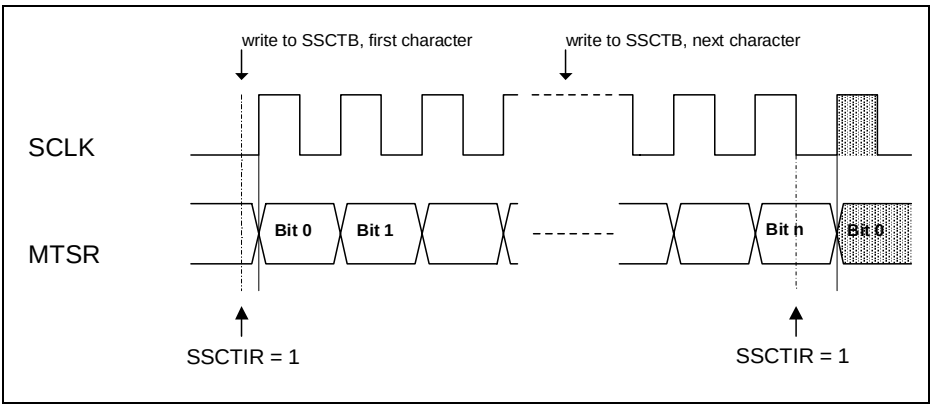


Figure 2 SSCTB Timing

Typically, in interrupt driven systems, no problems are expected from the modified timing of flag SSCTIR. However, when flag SSCTIR is polled by software in combination with other flags which are set/cleared at the end or at the beginning of a transfer (e.g. SSCBSY), the modified timing may have an effect.

Another situation where a different system behaviour may be noticed is the case when only one character is transferred by the PEC into the transmit buffer register SSCTB. In this case, 2 interrupt requests from SSCTIR are expected: the 'PEC COUNT = 0' interrupt, and the 'SSCTB empty' interrupt:

When the PEC transfer is performed with sufficient margin to the next clock tick from the SSC baud rate generator, and no higher priority interrupt request has occurred in the meantime, the 'PEC COUNT = 0' interrupt will be acknowledged before the 'SSCTB empty' interrupt request is generated, i.e. two interrupts will occur based on these events. However, when the PEC transfer takes place relatively close before the next clock tick from the SSC baud rate generator, or a higher priority interrupt request has occurred while the PEC transfer is performed, the 'PEC COUNT = 0' interrupt may not be acknowledged before the 'SSCTB empty' interrupt request is generated, such that effectively only one interrupt request will be generated for two different events.

In order to achieve a defined and systematic behavior with all device steps, the SSC receive interrupt, which is generated at the end of a character transmission, may be used instead of the SSC transmit interrupt.

CAN.H1 Note on Interrupt Register Behaviour of the CAN module

Due to the internal state machine of the CAN module, a specific delay has to be considered between resetting INTPND and reading the updated value of INTID. See Application Note AP2924 "Interrupt Register behaviour of the CAN module ..." on:

Infineon Microcontroller Products: **www.infineon.com/microcontrollers**

AP 2924 V01:

http://www.infineon.com/cgi/ecrm.dll/ecrm/scripts/public_download.jsp?oid=10042&parent_oid=-8984

EMUL.H1 Adapt Mode setting for emulation of devices in Single Chip Mode

In Adapt Mode all target device pins are in high impedance state. This mode is mainly used for deactivation of the soldered device while an emulator probe is connected. In single chip mode ($\overline{EA}$ = 1) Adapt Mode can not be selected via PORT0 (P0L.1) during reset, because PORT0 is not evaluated in this mode.

Recommendation:

- use no target device or a dummy device (package without silicon on the PCB) for emuation **or**
- use the target device on the PCB **and**:
  - provide a possibility to set target device pins $\overline{EA}$ and P0L.1 to "0" during reset (now the adapt mode is enabled on the target device) **and**
  - disconnect pin $\overline{EA}$ from the emulator probe **and**
  - set the $\overline{EA}$ pin on the emulator-probe to "1" to configure the emulator in single chip mode.

OTP.H1 OTP Read access in CHM Mode delivers Zeros only

In CHM mode the ROM-bus interface has no access to OTP (the OTP module is in a special mode for programming). In this mode the OTP contents can be checked by the verify sequence only. See e.g. C164CI User's Manual: Memory Organization - OTP Memory Programming - OTP Programming Example: "ALT_VERIFY".

MainOsc.H1 Main Oscillator Type_LP2: Negative Resistance and Start-up Reliability

Compared to other C16x microcontrollers the gain of the on-chip oscillator (Type_LP2) is slightly different. It is recommended to check the negative resistance and the start-up reliability of the oscillator circuit in the original application. Please refer to the limits specified by the quartz crystal or ceramic resonator supplier.

See also Application Note AP2420 'Crystal Oscillator of the C500 and C166 Microcontroller Families' and Application Note AP2424 'Ceramic Resonator Oscillators of the C500 and C166 Microcontroller Families'.
Infineon Microcontroller Products: **www.infineon.com/microcontrollers**

AP2420: **www.infineon.com/cgi/ecrm.dll/ecrm/scripts/public_download.jsp?oid=9746&parent_oid=-8984**
AP2424: **www.infineon.com/cgi/ecrm.dll/ecrm/scripts/public_download.jsp?oid=9745&parent_oid=-8984**

MainOsc.H2 Maximum Oscillator Frequency = 16 MHz (Main: Type_LP2)

The main oscillator is optimized for oscillation with a crystal within a frequency range of 4...16 MHz. When driven by an external clock signal it will accept the specified frequency range (see Data Sheet, AC Characteristics, tables 'Clock Generation Modes' and 'External Clock Drive Characteristics'). Operation at lower input frequencies is possible but is guaranteed by design only (not 100% tested) (see Data Sheet, AC Characteristics, table 'External Clock Drive Characteristics').

OWD.H2 Oscillator Watchdog and Prescaler Mode

The OWD replaces a missing oscillator clock signal with the PLL clock signal (base frequency).

In direct drive mode the PLL base frequency is used directly (f_{CPU} = 2...5 MHz).

In prescaler mode the PLL base frequency is divided by 2 (f_{CPU} = 1...2.5 MHz).

ISNC.H1 Maintenance of ISNC register

The RTC and PLL interrupts share one interrupt node (XP3IC). If an interrupt request occurs the request bit in the Interrupt Subnode Control register has to be checked and cleared by software. To avoid a collision with the next hardware interrupt request of same source it is recommended to clear the request and the enable bit first and then to set the enable bit again.

Example for an XP3 interrupt service routine (for Tasking C compiler):

```
...
if (PLLIR)
{
  _bfld (ISNC, 0x000C, 0x0000);  // clear PLLIE and PLLIR
  _putbit (1, ISNC, 3);          // set PLLIE
  ...                            // further actions concerning PLL/OWD
}
if (RTCIR)
{
  _bfld (ISNC, 0x0003, 0x0000);  // clear RTCIE and RTCIR
  _putbit (1, ISNC, 1);          // set RTCIE
  ...                            // further actions concerning RTC
}
...
```

Example for an XP3 interrupt service routine (in assembly language):

```
...
  EXTR   #1
  JNB    PLLIR, no_pll_request
  EXTR   #2                    ; no further interruption of this
                               ;   sequence possible
  BFLDL  ISNC, #0Ch, #00h      ; clear PLLIE and PLLIR
  BSET   PLLIE                 ; set PLLIE
...                            ; further actions concerning PLL/OWD
no_pll_request:
  EXTR   #1
  JNB    RTCIR, no_rtc_request
  EXTR   #2  ; no further interruption of this sequence possible
  BFLDL  ISNC, #03h, #00h      ; clear RTCIE and RTCIR
  BSET   RTCIE                 ; set RTCIE
...                            ; further actions concerning RTC
no_rtc_request:
...
```

5 Documentation Update

PORT.D2 Output Driver Control

In contrast to the user's manual in the C164CI-8E the registers POCONx are not available. The output drivers edge characteristic can be configured by register PDCR instead.

Edge Characteristic

This defines the rise/fall time for the respective output, i.e. the output transition time. Slow edges reduce the peak currents that are drawn when changing the voltage level of an external capacitive load. For a bus interface, however, fast edges may still be required. Edge characteristic effects the pre-driver which controls the final output driver stage.

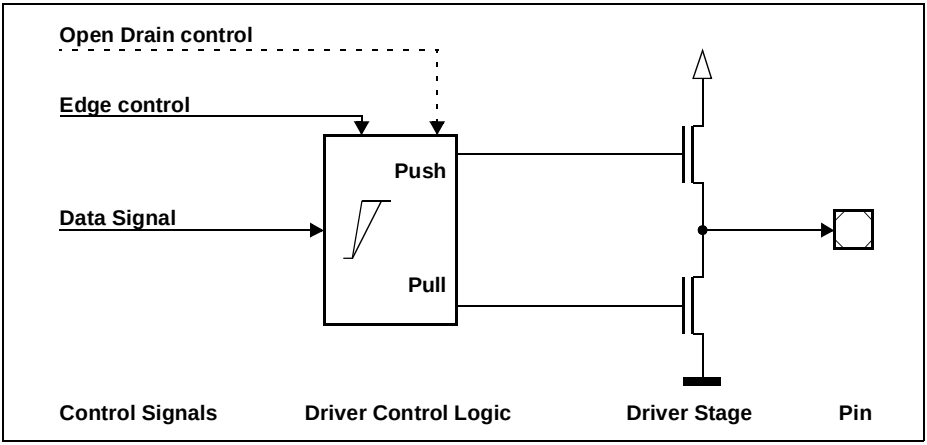


Figure 3 Structure of Output Driver with Edge Control

The **Port Driver Control Register** PDCR provides the corresponding control bits. A separate control bit is provided for bus pins as well as for non-bus pins.

PDCR
Port Driver Control Reg. **ESFR ($F0AA_H/55_H$)** **Reset value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-	-	**NBP EC**	-	-	-	**BIP EC**
-	-	-	-	-	-	-	-	-	-	-	rw	-	-	-	rw

Bit	Function
BIPEC	**Bus Interface Pins Edge Characteristic** (Defines the outp. rise/fall time t_{RF}) 0: Fast edge mode, rise/fall times depend on the driver's dimensioning. 1: Reduced edge mode. BIPEC controls: PORT0, PORT1, Port 4, $\overline{RD}$, $\overline{WR}$, ALE, CLKOUT, $\overline{BHE}$/$\overline{WRH}$, $\overline{RSTOUT}$, $\overline{RSTIN}$ (bidirectional reset mode only).
NBPEC	**Non-Bus Pins Edge Characteristic** (Defines the output rise/fall time t_{RF}) 0: Fast edge mode, rise/fall times depend on the driver's dimensioning. 1: Reduced edge mode. NBPEC controls: Port 3, Port 8

The figure below summarizes the effects of the driver characteristics:

Edge characteristic generally influences the output signal's **shape**.

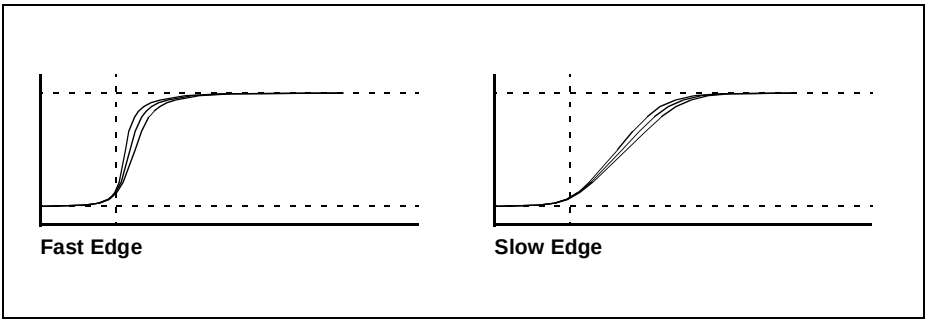


Figure 4 General Output Signal Waveforms

INT.D1 Data Sheet V2.0, 2001-05 - Special Function Registers Overview

In contrast to Table 7 (Data Sheet V2.0, page 33...) the following CCxIC registers are **not** implemented:
CC20IC, CC21IC, CC22IC, CC23IC, CC28IC, CC29IC, CC30IC and CC31IC. See also Table 3 C164CI Interrupt Nodes (page 17).

DOC_OTP.D1 Differences to Data Sheet Version 2.0, May 2001

To merge the documentation of all C164CI derivatives the Data Sheet Version 2.0, May 2001 will be used as the standard description. The following table describes the differences of the C164CI-8E Steps Cx to this document.

Table 7 Documentation Update

Function	Short Description	Data Sheet V2.0, 2001-05	
		affected section	page
Operating Frequency	Standard operating frequency: 20 MHz. All timings, baud rates, resolutions and ADC conversion time have to be recalculated	• overview • Functional Description • Serial Channels • Table 11 • Table 12 • AC Characteristics • CLKOUT	1 11 26 51 54 58 ff 72
Extension RAM	XRAM is not available in step C	overview	1
CAN pin assignment	CAN RxD and TxD are fixed assigned to Port 4 (P4.5 and P4.6), CAN Register IR is implemented instead of PCIR (no bitfield IPC) - see **CAN.D1**	• Figure 2 • Table 2 Pin Definitions and Functions • CAN-Module	4 10 27
CAN clock prescaler	Clock Prescaler Control Bit CPS in register CSR is implented in step D 1st	CAN-Module	27
OTP	Programming voltage see **OTP.8**	Memory Organization	12
External Bus Controller	Demultiplexed Bus Mode available (this mode was not described before)	• Ext. Bus Controller • AC Characteristics	13 65 ff
CAPCOM6	Version 3 implemented	The Capture/ Compare Unit CAPCOM6	21

Table 7 Documentation Update (cont'd)

Function	Short Description	Data Sheet V2.0, 2001-05	
		affected section	page
Watchdog Timer	No input prescaler implemented only f_{CPU} / 2 or f_{CPU} / 128 available time interval 26 µs ... 422 ms @ 20 MHz default interval 6.55 ms @ 20 MHz	Watchdog Timer	27
Frequency output	The programmable frequency output FOUT is not available in step C	• Parallel Ports • Power Management	28 30
Port Driver	Port driver characteristics controlled by register PDCR (for 2 groups: ext. bus ports and others) instead of registers POCONx - see **PORT.D2**	• Parallel Ports • Operating Conditions (footnote 5)	28 42
Sleep Mode	The sleep mode (and register SYSCON1) is not available in step C	• overview • Power Management • Power Consumption	1 30 46
Digital supply voltage	Minimum 4.5 V	Operating Conditions	42
Ports	Input low voltage (XTAL1 ... value ...)	DC Characteristics	43
Ports	Output low voltage $V_{OL\,max}$ = 0.45 V @ I_{OL} = 2.4 mA (PORT0, PORT1, Port 4, ALE, $\overline{RD}$, $\overline{WR}$, $\overline{BHE}$, CLKOUT, $\overline{RSTOUT}$)	DC Characteristics	43
Ports	Output low voltage $V_{OL1\,max}$ = 0.45 V @ I_{OL} = 1.6 mA (all other outputs)	DC Characteristics	43
$\overline{RSTIN}$ inactive / active current	To meet the real test condition the pullup resistor definition is replaced by "RSTIN inactive current" and "RSTIN active current"	DC Characteristics footnote 6	44
Port drivers	The pin drivers have a fixed configuration (control register PDCR is implemented instead of POCONx)	DC Characteristics Table 10 (Current Limits for Port Output Drivers)	45

Table 7 Documentation Update (cont'd)

Function	Short Description	Data Sheet V2.0, 2001-05	
		affected section	page
Power Consumption	Power-down mode supply current with RTC running: I_{PDR} = **200** + 25 * f_{OSC} see DC.IPDR.2	Power Consumption	46
External Clock	Minimum Oscillator Period: • Direct Drive: 50 ns • Prescaler 2:1: 25 ns • PLL 1:N: 75 ns Minimum high time t1 = 18[1] ns Minimum low time t2 = 18[1] ns	External Clock Drive Characteristics	54

[1] The minimum high and low time refers to a duty cycle of 50%. The maximum operating freqency (f_{CPU}) in direct drive mode depends on the duty cycle of the clock input signal.

CAPCOM6 Version 3

- The Block Commutation sequence table for left rotation is not a standard commutation table, but rather is shifted against the standard pattern by 60 degrees.
- Cout63 is switched to P1L.6 while $\overline{\text{CTRAP}}$ = low.
- In case of a trap event (CTRAP = low) CAPCOM6 outputs are not switched to their initial values (CC6MCON, bits 0..5 and bit 7), but to port register P1L instead.
- switches **TT12DIS and TT13DIS are not available**

For more details see application note AP1673: Technical details on CAPCOM6 module

ID-Registers

		Register:	IDMANUF	IDCHIP	IDMEM	IDPROG	IDMEM2
Device	**Step**	**Address:**	**F07E$_H$**	**F07C$_H$**	**F07A$_H$**	**F078$_H$**	**F076$_H$**
C164CI-8E	-CA		1820$_H$	0A03$_H$	4010$_H$	9340$_H$	0000$_H$

Application Engineering Group, Munich