SIEMENS

Microcontroller Components

# Errata Sheet

October 9, 1998 / Release 1.2

| | |
|---|---|
| **Device:** | **SAB-C167CR-16FM** |
| **Stepping Code / Marking:** | **ES-AC, ES1-AC, ES1*-AC** |
| **Package:** | **MQFP-144** |

This Errata Sheet describes the deviations from the current user documentation. The classification and numbering system is module oriented in a continual ascending sequence over several derivatives, as well already solved deviations are included. So gaps inside this enumeration could occur.

The current documentation is: Data Sheet: C167CR-16FM Data Sheet 03.97,
User's Manual: C167 Derivatives User's Manual V2.0 03.96
Instruction Set Manual 12.97 Version 1.2

**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.

**Change summary to Errata Sheet Rel.1.1 for devices with stepping code/marking ES-AC, ES1-AC, ES1*-AC:**

- PEC Transfers after JMPR (BUS.18)
- Modifications of ADM field while bit ADST = 0 (ADC.11)
- P0H I/O conflict during XPER access and external 8-bit non-multiplexed bus (X10): description modified
- Note about workaround for Tasking compiler added for problem CPU.16
- Note on Interrupt Register Behaviour of CAN module added
- new naming convention for DC/AC specification deviations used
- Note about overload conditions modified (see note 1.) in section Deviations from DC Characteristics)
- Note about Address Window Arbitration added (see end of document)

## Functional Problems:

### PWRDN.1: Execution of PWRDN Instruction while pin NMI# = high

When instruction PWRDN is executed while pin 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 preceding the PWRDN instruction **writes** to external memory or an XPeripheral (XRAM, CAN), 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 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 precedes 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 or XRAM.

### CPU.7: Warm HW Reset (Pulse Length < 1032 TCL)

In case a HW reset signal with a length < 1032 TCL (25.8 µs @ 20 MHz) is applied to pin RSTIN#, the internal reset sequence may be terminated before the specified time of 1032 TCL, and **not all SFRs and ESFRs may be correctly reset to their default state**. Instead, they maintain the state which they had before the falling edge of RSTIN#. The problem occurs when the falling edge of the (asynchronous) external RSTIN# signal is coincident with a specific internal state of the controller. The problem will statistically occur more frequently when waitstates are used on the external bus.

### Workaround:

Extend the HW reset signal at pin RSTIN# (e.g. with an external capacitor) such that it stays below $V_{IL}$ (0.2 Vcc - 0.1 V) for at least 1032 TCL.

## CPU.8: Jump instructions in EXTEND sequence

When a jump or call is taken in an EXTS, EXTSR, EXTP, or EXTPR sequence, a following data access included in the EXTEND sequence might be performed to a wrong segment or page number.

**Note**: ATOMIC or EXTR sequences are **not** affected by this problem.

Example: Accessing double-word data with a check on segment overflow between the two accesses (R5 contains 8-bit segment number, R4 contains 16-bit intra-segment offset address):

```
      EXTS  R5,#4         ; start EXTEND sequence
      MOV   R10,[R4+]     ; get first word
      CMP   R4,#0         ; check for segment overflow
      JMPR  cc_NZ,Next    ; jump if no segment overflow
      ADD   R5,#1         ; increment to next segment
      EXTS  R5,#1         ; continue EXTEND sequence
Next: MOV   R11,[R4]      ; get second word
```

With this sequence, the problem can occur when the jump is taken to label Next; the data access here might use a wrong segment number.

### Workaround:

Do not use jumps or calls in EXTS, EXTSR, EXTP, or EXTPR sequences. This can be done very easily since only an actual data access must be included in an EXTEND sequence. All other instructions, such as comparisons and jumps, do not necessarily have to be in the EXTEND sequence.

For the example shown above, there are several possibilities to get around the problem:

a) with a jump, but EXTEND sequence only for the data accesses

```
      EXTS  R5,#1         ; EXTEND sequence only for data access
      MOV   R10,[R4+]     ; get first word
      CMP   R4,#0         ; check for segment overflow
      JMPR  cc_NZ,Next    ; jump if no segment overflow
      ADD   R5,#1         ; increment to next segment
Next: EXTS  R5,#1         ; second EXTEND sequence for data access
      MOV   R11,[R4]      ; get second word
```

b) without a jump

```
      EXTS  R5,#4         ; EXTEND sequence
      MOV   R10,[R4]      ; get first word
      ADD   R4,#2         ; increment pointer here
      ADDC  R5,#0         ; add possible overflow from pointer inc.
      EXTS  R5,#1         ; continue EXTEND sequence
      MOV   R11,[R4]      ; get second word
```

The first EXTEND instruction of example b) can also be modified such that only the following data access is included in the EXTEND sequence (EXTS R5,#1). This additionally has the effect of a reduced interrupt latency.

**Notes on Compilers and Operating Systems**

Such critical sequences might be produced within library functions of C-Compilers when accessing huge double-word data, or in operating systems.

From the following compiler versions, we currently know that they are **not** affected by this problem:

BSO/Tasking V4.0r3
HighTec C16x-GNU-C V3.1
Keil C166 (from V2.60)

## <u>CPU.9:</u> PEC Transfers during instruction execution from Internal RAM

When a PEC transfer occurs after a jump with cache hit during instruction execution from internal RAM (locations 0F600h - 0FDFFh), the instruction following the jump target instruction may not be (correctly) executed. This problem occurs when the following sequence of conditions is true:

i) a loop terminated with a jump which can load the jump target cache (possible for JMPR, JMPA, JB, JNB, JBC, JNBS) is executed in the internal RAM

ii) at least two loop iterations are performed, and no JMPS, CALLS, RETS, TRAP, RETI instruction or interrupt is processed between the last and the current iteration through the loop (i.e. the condition for a jump cache hit is true)

iii) a PEC transfer is performed after the jump at the end of the loop has been executed

iv) the jump target instruction is a double word instruction

**Note**: **No problem** will occur during instruction execution from the internal **XRAM** (locations 0E000h - 0E7FFh).

### Workaround 1:

Place a single word instruction (e.g. NOP) at the jump target address in the internal RAM.

### Workaround 2:

Use JMPS (unconditional) or JMPI (conditional) instructions at the end of the loop in the internal RAM. These instructions will not use the jump cache.

## CPU.10: Bit Protection for register TFR

The bit protection for the Trap Flag Register (TFR) does not operate correctly: when bits (trap flags) in this register are modified via bit or bit-field instructions, other trap flags which could have been set after the read phase and before the write phase of these instructions, and which are not explicitly selected by the bit instruction itself, may erroneously be cleared. This way, a trap event may be lost.

Typically, bit accesses to register TFR are only performed in trap service routines in order to clear the trap flag which has caused the trap. In practice, the malfunction of the bit protection may only cause problems in systems where the NMI trap (asynchronous event) is used. All other situations where the malfunction could have an effect are under software control: the occurrence of a class A stack underflow/overflow trap in a class B trap service routine, or the intentional use of (illegal) instructions which may cause a class B trap condition in a class A trap routine.

### Workaround:

When the NMI trap is used, connect the NMI# pin to a pin of the C167 which is capable of generating an interrupt request on a falling signal edge. In each trap routine, test the respective interrupt request flag xxIR after modifications of trap flags in register TFR have been performed, e.g. as follows:

```
TrapEntry:
        BCLR   STKOF              ; clear (stack overflow) trap flag
        ...                       ; service (stack overflow) trap
        ...
        JNB    xxIR, Trap Exit    ; test for lost NMI
        BSET   NMI                ;
TrapExit:
        RETI
```

In the NMI trap routine, both the actual NMI flag and the auxiliary interrupt request flag xxIR must be cleared.

## CPU.11: Stack Underflow Trap during Restart of interrupted Multiply

Wrong multiply results may be generated when a STUTRAP (stack underflow) is caused by the last implicit stack access (= pop PSW) of a RETI instruction which restarts an interrupted MUL/MULU instruction.

No problem will occur in systems where the stack overflow/underflow detection is not used, or where an overflow/underflow will result in a system reset.

### Workaround 1:

Avoid a stack overflow/underflow e.g. by
- allocating a larger internal system stack (via bitfield STKSZ in register SYSCON), or
- reducing the required stack space by reducing the number of interrupt levels, or
- testing in each task procedure whether a stack underflow is imminent, and anticipating the stack refill procedure before executing the RETI instruction.

### Workaround 2:

Disable MULx instructions from being interrupted e.g. with the following instruction sequence:

```
        ATOMIC #1
        MULx Rm, Rn
```

**Workaround 3** (may be selected if **no divide** operations are used in an interruptible program section):

In each interrupt service routine (task procedure), always clear bit MULIP in the PSW and set register MDC to 0000h. This will cause an interrupted multiplication to be completely restarted from the first cycle after return to the priority level on which it was interrupted.

In case that an interrupt service routine is also using multiplication, only registers MDH and MDL must be saved/restored when using this workaround, while bit MULIP and register MDC must be set to zero.

## CPU.12: Access to internal Flash with EXTS/EXTSR instructions

EXTS instructions (EXTS, EXTSR) do not work correctly for data accesses to internal Flash when the 2 msbs of the intra-segment address are different from the 2 lsbs of the DPP which is selected by these 2 msbs of the intra-segment address. In this case, the 2 msbs of the intra-segmented address are substituted by the 2 lsbs of the selected DPP.

Example:

```
EXTS #1, #1         ; Flash access to be performed in segment 1
MOV R0, 8000h       ; intra-segment address to be accessed = 8000h
                    ; 2 msbs = 10b select DPP2
```

In case DPP2 contains 000h, location 1: 0000h is accessed instead of 1:8000h.

### Workarounds:

Use e.g. EXTP/EXTPR instructions instead of EXTS/EXTSR.

On C level, workarounds depend on the memory model and data types.

For the KEIL compiler, the following specific workaround is suggested:

a) do not use the memory models HLARGE or HCOMPACT and do not use the memory types huge or xhuge, only in these combinations the EXTS/EXTSR instruction is generated.

b) when memory types huge or xhuge are required, use the C166xxx.LIB run-time libraries instead of the C167xxx.LIB run-time library files and do not compile with the MOD167 directive. ESFRs may be referenced with addresses casted to pointers, e.g. #define DP0L (*((unsigned int volatile sdata*) 0xF100))

## CPU.16: Data read access with MOVB [Rn], mem instruction to internal Flash

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

1. *mem* specifies a direct 16-bit byte operand address in the internal 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) 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, 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.

E.g.:

```
MDH    MDL       Rn          MDH  MDL
F0F0   0F0Fh : F0F0h  = FFFF FFFFh, but no Overflow indicated !
                        (result with 32-bit precision: 1 0000h)
```

The same malfunction appears for the following combinations:

```
n0n0 0n0n : n0n0
n00n 0nn0 : n00n
n000 000n : n000
n0nn 0nnn : n0nn         where 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
```

**Note**:
- 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

## BUS.14: Spikes on CS# Lines when using RD/WR-CS#

Spikes of >= 5 ns width from Vcc to Vss may occur on Port 6 lines configured as CS#:

1. The spikes occur on all lines defined as CS# when at least one of them is configured as RD/WR-CS#, and when the low-to-high transition of ALE, RD# or WR#, and RD-CS# or WR-CS# occur simultaneously (i.e. when a non-muliplexed bus without memory tristate WS is used).

2. Spikes may also occur when in all BUSCONs where RD/WR-CS# was configured a memory tristate WS was programmed, and read/write accesses to the internal XRAM were performed.

## Workaround:

Use Address-CS# instead of RD/WR-CS# for all BUSCONx registers, i.e. leave bits BUSCONx[15:14] = 00b.

Note: using memory tristate WS in combination with RD/WR-CS# does not generally solve the problem (see above, item 2).

## 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

**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:

1. 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
2. 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
3. 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 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 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'.

## ADC.7: Channel Injection coincident with start of Standard Conversion

When a request for an injected conversion is triggered simultaneously with the start of a standard conversion (single channel, auto scan, continuous modes), only the standard conversion will be performed. The channel injection request and **all further** requests for injected conversions are **not** processed (i.e. they are blocked). This problem may only occur

a) within a time window of 2 TCL after the instruction which starts a standard conversion has been executed, in case the channel injection is triggered by hardware (CC31), or
b) when the channel injection is triggered via software by the same instruction which writes to ADCON in order to start a standard conversion.

## Workaround:

When a channel injection is to be triggered by software, use separate instructions and first start the standard conversion, and then trigger the channel injection.

When a channel injection is to be triggered by hardware, interrupt request flag CC31IR may be used to detect whether the request was generated in the critical time window. The blocking of injected conversion requests can be removed by forcing a 0-1 transition of bit ADCRQ (due to internal timing, it is not sufficient to clear ADCRQ within the MOV ADCON instruction which starts the standard conversion; ADCRQ must explicitly be cleared after this instruction):

```
        BCLR   CC31IR              ; clear CC31IR for later evaluation
        MOV    ADCON, #XYXXh       ; start standard conversion,
                                   ; set ADCRQ = 0, ADCIN = 1
        BCLR   ADCRQ               ; clear ADCRQ to avoid blocking of
                                   ; further injection requests
        EXTR   #1
        JNB    CC31IR, Done        ; check for injection request
        BSET   ADCRQ               ; request channel injection if required
Done:   ...
```

**Note:** an injected conversion in progress will be aborted when a standard conversion is started (0-1 transition of bit ADST) by software. However, bit ADCRQ is not automatically cleared to 0, thus blocking further channel injection requests. Therefore, a standard conversion should only be started when no injected conversion is in progress (e.g. in the interrupt routine after completion of a channel injection), or, if abortion of an injected conversion is tolerable, bit ADCRQ should be set to 0 by the instruction which starts a standard conversion in order not to block further injections.

### This problem will be fixed in the next steps as follows:

- when an injected conversion is in progress and a standard conversion is started by software, the injected conversion is **completed** and then the standard conversion is performed.
- when an injected conversion is triggered simultaneously with the start of a standard conversion, the **injected** conversion is performed **first**, and then the standard conversion is started.
- when a standard conversion is in progress and an injected conversion is requested, the conversion of the currently selected channel of the standard conversion is completed and then the injected conversion is performed.

## 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.

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. C167 User’s Manual, V2.0, p16-4)

### 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.

## CAPCOM.1: Software Update of CAPCOM Timers

When a CAPCOM Timer is updated via software with a value which exactly matches the contents of an associated CAPCOM register (configured for compare mode), it may happen that the programmed compare event is **not** generated (as expected) within the next 16 TCL, i.e. it is lost in the current timer period. This behaviour statistically occurs in 50% of the cases where the timer update value is equal to a compare value.

Correct compare events are only assured if the timer is incremented/reloaded by hardware.

### Workaround:

Do not perform software updates of a CAPCOM timer if compare events are programmed for the associated CAPCOM registers.

## RST.1: System Configuration via P0L.0 during Software/Watchdog Timer Reset

Unlike P0L.5 .. P0L.1, **P0L.0 is not disregarded during software or watchdog timer reset**. This means that when P0L.0 is (erroneously) externally pulled low at the end of the internal software or watchdog timer reset sequence, the device will enter emulation mode.
Therefore, ensure that the level at P0L.0 is above the minimum input high voltage $V_{IHmin}$= 0.2 Vcc + 0.9 V (1.9 V @ Vcc = 5.0 V) at the end of the internal reset sequence.

## X9: Read Access to XPERs in Visible Mode

The data of a read access to an XBUS-Peripheral (XRAM, CAN) in Visible Mode 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.

## X10: P0H I/O conflict during XPER access and external 8-bit Non-multiplexed bus

When an external 8-bit non-multiplexed bus mode is selected,
**and** P0H is used for general purpose I/O,
**and** an internal (byte or word) write access to the XRAM or CAN module is performed,
**and** an **external** bus cycle is directly following the internal XBUS write cycle,
then P0H is actively driven with the write data for 1TCL (pulse on P0H).

Note that if any of the other bus modes is selected in addition to the 8-bit non-multiplexed mode, P0H can not be used for I/O per default.

The pulses will occur after the rising edge of ALE of the first external bus cycle that directly follows the internal XBUS write cycle.

### Workarounds:

- use a different port instead of P0H for I/O when (only) an external 8-bit non-multiplexed bus mode is selected
- or use a different bus type (e.g. 8-bit multiplexed, where P1H may be used for I/O instead of P0H)
- or do not perform an external bus access directly after an XBUS write access:
  this may be achieved by an instruction sequence which is executed in internal ROM/Flash/OTP, or internal RAM, or internal XRAM

```
e.g.    ATOMIC #3     ; to prevent PEC transfers which may access external memory
        instruction which writes to XBUS peripheral
        NOP
        NOP
```

## X11: Illegal Bus Trap after XPER access in Single Chip Mode

When single chip mode has been selected during reset (pin EA# = high), and no external bus has been enabled in addition in any of the BUSCONx registers, the following problem will occur:

an illegal external bus access trap is generated when an access to an XBUS peripheral (XRAM, CAN) is performed

### Workaround:

Enable an external bus in one of the BUSCONx registers before the first XPER (XRAM, CAN) access is performed.
E.g.:

```
MOV BUSCON1, # 04C0h    ; Multiplexed bus, port 0 must not be used for I/O
                        ; ADDRSEL1 = 0000h (default):
                        ; --> resulting dummy window size: 4K @ 00:0000h
```

P0H.4 should be pulled low during HW-reset
    --> no segment address output at port 4, port 4 free for I/O and CAN
P0H.1 should be pulled low during HW-reset
    --> no CS# output at port 6[4:0], pins free for I/O

## Deviations from Electrical- and Timing Specification:

The following table lists the deviations of the DC/AC characteristics from the specification in the C167CR-16FM Data Sheet 3.97:

### - DC Characteristics:

Supply Voltage Vcc = 5 V ± **5 %** instead of 5 V ± 10 %

| **Problem** | **Parameter** | **Symbol** | **Limit Values** | | **Unit** | **Test** |
|---|---|---|---|---|---|---|
| **short name** | | | min. | max. | | **Condition** |
| **DC.IID.1** | Idle mode supply current | $I_{ID}$ | - | **40**+2*$f_{cpu}$ instead of 30+2*$f_{cpu}$ | mA | RSTIN# = $V_{IH1}$, $f_{cpu}$ in MHz |
| **DC.IPD.2** | Power down mode supply current | $I_{PD}$ | - | not tested$^{1)}$ | mA | |
| **DC.IOZ1.1** | Input leakage current (Port 5) | $I_{OZ1}$ | - | ±**1000** instead of ±200 | nA | 0.45 V < $V_{IN}$ < Vcc |
| **DC.IOZ2.1** | Input leakage current (all other inputs) | $I_{OZ2}$ | - | ±**1000** instead of ±500 | nA | 0.45 V < $V_{IN}$ < Vcc |

1) typically several mA

Notes:

1.) The protection mechanism for **overload conditions** (Vin > Vcc+0.5V or Vin < Vss-0.5V) on the C167CR-16FM is different from ROM and ROMless devices. In particular on pins NMI#, RSTIN#, XTAL1, and Port 5, the input voltage must not exceed the absolute maximum ratings (-0.5V, Vcc+0.5V). Otherwise, external clamp diodes must be used. When an overload condition occurs on Port 5 pins, the specified TUE for the A/D converter is no longer guaranteed.

2.) Pin **READY#** has an internal pull-up (all C167xx derivatives). This will be documented in the next revision of the Data Sheet.

3.) During **Reset**, the internal **pull-ups on P6.[4:0]** are active, independent whether the respective pins are used for CS# function after reset or not.

### - A/D Converter Characteristics

**ADCC.1:** Total unadjusted error TUE = ± **3** LSB instead of ± 2 LSB

| **Problem short name** | Parameter | Symbol | Max. CPU Clock = 20 MHz | | Variable CPU Clock 1/2TCL = 1 to 20 MHz | | Unit |
|---|---|---|---|---|---|---|---|
| | | | min. | max. | min. | max. | |
| **AC.t6.3** | Address setup to ALE | t6 | **0**+ta instead of 10+ta | - | TCL-**25**+ta instead of TCL-15+ta | - | ns |
| **AC.t12.2** | RD#/WR# low time (with R/W Delay) | t12 | **25**+tc instead of 40+tc | - | 2TCL-**25**+tc instead of 2TCL -10+tc | - | ns |
| **AC.t14.2** | RD# to valid data in (with RW-delay) | t14 | - | **5**+tc instead of 30+tc | - | 2TCL-**45**+tc instead of 2TCL -20+tc | ns |
| **AC.t16.3** | ALE low to valid data in (no RW-delay) | t16 | - | **40**+ta+tc instead of 55+ta+tc | - | 3TCL-**35**+ta+tc instead of 3TCL-20+ta+tc | ns |
| **AC.t17.1** | Address to valid data in | t17 | - | **60**+ta+tc instead of 70+2ta+tc | - | 4TCL-**40**+2ta+tc instead of 4TCL-30+2ta+tc | ns |
| **AC.t22.3** | Data valid to WR# | t22 | **15**+tc instead of 25+tc | - | 2TCL-**35**+tc instead of 2TCL-25+tc | - | ns |
| **AC.t28.1** | Address hold after RD#/WR# | t28 | **-2.5**+tf instead of 0+tf | - | **-2.5**+tf instead of 0+tf | - | ns |
| **AC.t33.1** | CLKOUT fall time | t33 | - | **10** instead of 5 | - | **10** instead of 5 | ns |
| **AC.t34.2** | CLKOUT rising edge to ALE falling edge | t34 | **-5**+ta instead of 0+ta | 10+ta | **-5**+ta instead of 0+ta | 10+ta | ns |
| **AC.t35.1** | Synchronous READY# setup time to CLKOUT | t35 | **30** instead of 15 | - | **30** instead of 15 | - | ns |
| **AC.t39.1** | CS# low to valid data in | t39 | - | **45**+2ta+tc instead of 55+2ta+tc | - | 3TCL-**30**+2ta+tc instead of 3TCL-20+2ta+tc | ns |
| **AC.t46.2** | RdCS# low to valid data in (with R/W Delay) | t46 | - | **15**+tc instead of 25+tc | - | 2TCL-**35**+tc instead of 2TCL-25+tc | ns |
| **AC.t61.1** | HOLD# input setup time to CLKOUT | t61 | **35** instead of 20 | - | **35** instead of 20 | - | ns |

## History List (since device step ES-AC)

### Functional Problems

| Functional Problem | Short Description | Fixed in step |
|---|---|---|
| ADC.7 | Channel Injection coincident with start of standard conversion | |
| ADC.11 | Modifications of ADM field while bit ADST = 0 | |
| CAPCOM.1 | Software Update of CAPCOM Timers | |
| CPU.7 | Warm hardware reset (pulse length < 1032 TCL) | |
| CPU.8 | Jump instruction in EXTEND sequence | |
| CPU.9 | PEC Transfers during instruction execution from Internal RAM | |
| CPU.10 | Bit protection for register TFR | |
| CPU.11 | Stack Underflow during Restart of Interrupted Multiply | |
| CPU.12 | Access to internal Flash with EXTS/EXTSR instructions | |
| CPU.16 | Data read access with MOVB [Rn], mem instruction to internal Flash | |
| CPU.17 | Arithmetic Overflow by DIVLU instruction | |
| BUS.14 | Spikes on CS# Lines when using RD/WRCS# | |
| BUS.18 | PEC Transfers after JMPR Instruction | |
| RST.1 | System Configuration via P0L.0 during Software/Watchdog Timer Reset | |
| PWRDN.1 | Execution of PWRDN Instruction while pin NMI# = high | |
| X9 | Read Access to XPERs in Visible Mode | |
| X10 | P0H I/O conflict during XPER access and external 8-bit Non-multiplexed bus | |
| X11 | Illegal Bus Trap after XPER access in Single Chip Mode | |

## AC/DC Deviations

| **AC/DC Deviation** | **Short Description** | **Fixed in step** |
|---|---|---|
| | Supply Voltage Vcc = 5 V ± 5 % | |
| DC.IID.1 | Idle Mode Supply Current 40+2*fcpu mA | |
| DC.IPD.2 | Power down mode supply current: not tested | |
| DC.IOZ1.1 | Input leakage current (Port 5): 1000nA | |
| DC.IOZ2.1 | Input leakage current (all other): 1000nA | |
| AC.t6.3 | Address setup to ALE TCL-25ns | |
| AC.t12.1 | RD#WR# low time (with RW-delay) 2TCL-25ns | |
| AC.t14.2 | RD# to valid data in (with RW-delay) 2TCL-45ns | |
| AC.t16.3 | ALE low to valid data in (no RW-delay) 3TCL-35ns | |
| AC.t17.1 | Address to valid data in 4TCL-40ns | |
| AC.t22.3 | Data valid to WR# 2TCL-35ns | |
| AC.t28.1 | Address hold after RD#/WR# -2.5ns | |
| AC.t33.1 | CLKOUT fall time 10ns | |
| AC.t34.2 | CLKOUT rising edge to ALE falling edge min –5ns | |
| AC.t35.1 | Synchronous READY# setup time to CLKOUT 30ns | |
| AC.t39.1 | CS# low to valid data in 3TCL-30ns | |
| AC.t46.2 | RdCS# low to valid data in (with R/W Delay) 2TCL-35ns | |
| AC.t61.1 | HOLD# input setup time to CLKOUT 35ns | |
| ADCC.1 | TUE = ± 3 LSB | |

## Notes

1. The **Address Window Arbitration** feature as described in the C167 Derivatives User's Manual V2.0, p.8-22 is not yet implemented in C167CR-16FM devices up to and including step AC. For these devices, the description in C167 User's Manual V1.0 still applies in this context, i.e. **the address windows defined by ADDRSEL1 through ADDRSEL4 must not overlap each other.**

2. **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 in Siemens 16-bit Microcontrollers" on
   http://www.siemens.de/semiconductor/products/ics/34/pdf/ap292401.pdf

Application Support Group, Munich