SIEMENS

Since April 1, 1999, Siemens Semiconductor is Infineon Technologies. The next revision of this document will be updated accordingly.

Errata Sheet

June 17, 1996 / Release 1.2

Device : **SAK-C167CR-LM**
Stepping Code / Marking : **AB**

The C167**CR** is a version of the C167 with on-chip **CAN** module, 2 Kbyte **XRAM** module, and with an on-chip **PLL** oscillator circuit.

This errata sheet describes the functional problems known in this step. Problem classification and numbering is performed relative to modules, where the C167 AC-step is the reference. Since most problems of earlier steps have already been fixed in the C167CR, problem numbering is not necessarily consecutive.

The C167CR-LM devices are mounted in a 144-pin Plastic Metric Quad Flat Pack (P-MQFP-144-1) package.

Note: The PLL **input clock multiplier factor** for these devices is **4**.

Changes from Errata Sheet **Rel. 1.1** to this Errata Sheet **Rel. 1.2:**

- Test Conditions for A/D Converter changed (**SAK**-C167CR version **only**, see ADCTC.1 in section 'Deviations from DC/AC Specification)

Functional Problems

The following malfunctions are known in this step:

1.) Problems with Peripheral Modules

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 step 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.

<u>CAPCOM.1:</u> 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.

<u>2.) CPU Problems</u>

<u>CPU.7:</u> 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. We are in close contact with the compiler and operating system manufacturers in order to determine eventual problems and workarounds. Information on possible updates will be published as soon as available. Please contact your tool manufacturer or the Siemens Microcontroller Mailbox (Tel. 0049-(0)89-49 84 31). 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 V2.60

CPU.9: 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 C167CR 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 interruptable 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.

Workarounds for C165/C167 in combination with C compilers are currently under evaluation.

<u>RST.1:</u> System Configuration via P0L.0 during Software/Watchdog Timer Reset

The description in the C167 User's Manual V1.0 (chapter 17) concerning software and watchdog timer reset is not correct: 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.

Functional Problem	**Short Description**	**Remarks**
ADC.7	Channel Injection coincident with start of standard conversion	
CAPCOM.1	Software Update of CAPCOM Timers	
CPU.7	Warm hardware reset (pulse length < 1032 TCL)	
CPU.8	Jump instructions in EXTEND sequence	
CPU.9	PEC Transfers during instruction execution from Internal RAM	
CPU.10	Bit Protection for register TFR	
CPU.11	Stack Underflow Trap during Restart of interrupted Multiplication	
RST.1	System Configuration via P0L.0 during Software/ Watchdog Timer Reset	

Table 1: Functional Problems of the C167CR

3.) Specific Problems with X-Peripherals (XPERs)

The following problems with the interface to XPERs, the CAN module, and the XRAM module are currently known:

X9: Read Access to XPER 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.

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, P0H is actively driven with the write data.

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.

Workaround:

Do not use P0H for I/O when (only) an external 8-bit non-multiplexed bus mode is selected.

XRAM.1: XRAM Access in XPER-SHARE Mode

The XPER-SHARE mode (access to the XRAM from an external device) does not work correctly when external bus configurations are configured via the BUSCON1..4 registers. This mode will only operate when only the BUSCON0 register is used to configure an external bus.

Note: The XPER-SHARE mode is not applicable to access the integrated CAN module.

Functional Problem	Short Description	Remarks
X10	P0H I/O conflict during XPER access and external 8-bit Non-multiplexed bus (X10)	
X9	Read Access to XPER in Visible Mode	
XRAM1	XRAM Access in XPER-SHARE Mode	

Table 2: Functional Problems with XPERs on the C167CR

Deviations from DC/AC Specification

ADCTC.1: Test Conditions for A/D Converter (SAK-C167CR only)

The test conditions for the A/D Converter of the SAK-C167CR have been changed as follows:

VAREF = VCC = 5.0 V

Under these test conditions, TUE = +/- 2 LSB is guaranteed for VAIN = 0.0 V ... 1017 * VAREF/ 1024 (4.9658V @ Vcc = VAREF = 5.0V). This means that the minimum value of the digital result which is obtained when VAIN = VAREF is always greater or equal to 3F7h (1015d).

AC.3: Address setup to ALE (t6)

In particular at high temperatures (125 °C), the specified limit for t6 may be exceeded by up to 5 ns:

- t6 (address setup to ALE):

min. TCL-20 + tA	(5 ns + tA @ 20 MHz) instead of
min. TCL-15 + tA	(10 ns + tA @ 20 MHz)

DC.3 Power Down Mode Supply Current I_{PD}

The Power Down Mode supply current I_{PD} may be up to 8 mA (due to CAN module).

DC.4: Input Hysteresis

When the special thresholds V_{ILS}, V_{IHS} are programmed, the hysteresis HYS on some pins may be less than the target value of 400 mV.

DC.5: Port 0 Configuration Current

In particular at low temperatures (-40 °C), the minimum Port 0 configuration current I_{P0L} may exceed the specified limit by up to 20 µA:

I_{P0L} = min. - 120 µA instead of min. - 100 µA

DC.6: RSTIN# Pullup Resistor

The internal pullup-resistor at the RSTIN# pin is too low by a factor of 2 compared to the specified limits:

R_{RST} = 25 KOhm to 125 KOhm instead of 50 KOhm to 250 KOhm

DC.7: Input High Voltage V_{IH} on P2 .. P8

The minimum input high voltage V_{IH} on the pins of P2 .. P8 must be 100 mV higher than the specified limit:

V_{IH} = min. 0.2 Vcc + 1.0 V instead of 0.2 Vcc + 0.9 V

DC/AC Problem	**Short Description**	**Remarks**
ADCTC.1	Test Conditions for ADC (max result ≥ 3F7h, SAK-C167CR)	
AC.3	Address setup to ALE (t6)	
DC.3	Power Down Mode Supply Current I_{PD} (8 mA)	
DC.4	Input Hysteresis	
DC.5	Port 0 Configuration Current	
DC.6	RSTIN# Pullup Resistor	
DC.7	Input High Voltage V_{IH} on P2 .. P8	

Table 3: DC/AC Problems of the C167CR