

Errata Sheet

November 17, 1998 / Release 1.2

Device: **SAB-C167E2**
Stepping Code / Marking: **ES-AC**
Package: **CBGA-444**

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 Data Sheet 06.95
 C167SR/CR-L25M Data Sheet Addendum 1998-03
 User's Manual: C167 Derivatives User's Manual V2.0 03.96
 or User's Manual of emulated target device
 Instruction Set Manual 12.97 Version 1.2
For emulator manufacturers: C167E2 specification, V1.4

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

- PEC Transfers after JMPR (BUS.18)
- Spikes on CS# lines after access with RDCS# and/or WRCS# (BUS.17)
- Write to external memory followed by XRAM access (X15)
- P4.6 and P4.7: no Open Drain Functionality (PRT.2)
- Modifications of ADM field while bit ADST = 0 (ADC.11): reference to auto scan mode eliminated
- Read Access to XPERs in Visible Mode (X9): description modified
- Note on Interrupt Register Behaviour of CAN module added

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 multiplexed bus with memory tristate waitstate is used, the PWRDN instruction should be executed out of internal RAM or XRAM.

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

<i>MOVB [Rn], mem</i>	e.g. by	<i>MOV Rm, #mem</i>
		<i>MOVB [Rn], [Rm]</i>

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
```

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.18: Interrupted Multiply/Divide Instructions in internal program memory

When a multiply (MUL, MULU) or divide (DIV, DIVU, DIVL, DIVLU) instruction which is executed in internal program memory (simulated ROM or Flash) is interrupted, incorrect results may occur under the following conditions **when an internal program memory size of > 64 Kbytes is emulated**:

the multiply/divide instruction and the RETI instruction of the interrupt service routine which has interrupted the multiply/divide operation are **both** located in different code segments in **internal program memory** according to the following table (example shown for 256 Kbytes internal program memory):

s_{RETI} s_{MD}	0	1	2	3	4
4	c	c	c	c	ok
3	c	c	c	ok	c
2	c	c	ok	c	c
1	ok	ok	c	c	c
0	ok	ok	c	c	c

combinations marked as **ok** will not lead to problem

combinations marked as **c** ('critical') will lead to a problem when the word at a specific location ca ('critical address') in internal program memory represents the opcode of an instruction which operates on data type **BYTE** (typically the 4LSBs of these opcodes are odd hex numbers from 1 .. 9). The critical address ca depends on the 16-bit intra-segment address i_{MD} of the multiply/divide instruction and the code segment i_{RETI} in which the RETI instruction is executed:

$ca = s_{RETI} \cdot i_{MD}$ always when internal program memory is mapped to segment 1,

or when internal program memory is mapped to segment 0 and $s_{RETI} \geq 2$

$ca = 1 : i_{MD}$ when $s_{RETI} = 0$ and $i_{MD} \geq 8000h$ and internal program memory mapped to segment 0

$ca = 0 : i_{MD}$ when $s_{RETI} = 1$ and $i_{MD} \leq 7FFEh$ and internal program memory mapped to segment 0

(2) the multiply/divide instruction is interrupted by a PEC **byte** data transfer

Workaround:

Avoid interrupts or PEC transfers during execution of multiply/divide instructions e.g. by placing ATOMIC #1 in front of every MULx/DIVx instruction.

KEIL offers a special version of its C compiler V3.12 with a directive FIXMDU which automatically inserts ATOMIC #1 in front of every MULx/DIVx instruction.

BUS.16: Branches to Code Segments 3 and 4 when internal program memory is disabled

When the internal program memory is disabled,
and a simulated program memory size of > 128 Kbytes is selected,
and a program which is executed over the external bus performs a branch (CALLS, RETS, JMPS, ..) to code segments 3 or 4, the internal program memory will be accessed instead of the target address in external memory.

Workarounds:

- Locate code which is contained in segments 3 and 4 to other segments in external memory, e.g. to segments 13h and 14h, or
- Do not select a ROM size > 128 Kbytes in register EMUCON when the internal ROM/Flash/OTP shall not be emulated in an application.

BUS.17: Spikes on CS# Lines after access with RDCS# and/or WRCS#

Spikes of about 5 ns width (measured at $V_{OH} = 0.9 V_{CC}$) from V_{CC} to V_{SS} may occur on Port 6 lines configured as CS# signals. The spikes occur on one 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 RDCSy# (bit BUSCONy.CSRENy = 1) or a write cycle with WRCS# (bit BUSCONy.CSWENy = 1).

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

Potential effects on applications:

- when CS# lines are used as CE# signals for external memories, typically no problems are expected, since the spikes occur after the rising edge of the RD# or WR# signal.
- when CS# lines configured as RDCS# and/or WRCS# are used e.g. as 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 CS# lines configured as WRCS# are used as 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 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 RD/WR-CS# is used (i.e. bit BUSCONy.CSRENy = 1 and/or bit BUSCONy.CSWENy = 1), or
2. Use Address-CS# instead of RD/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 ROM/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_??, JMPA cc_??, 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_??, 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_??, 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_??, JMPA cc_??, 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.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 $n-1 \dots 0$ 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.

PRT.2: P4.6 and P4.7: no Open Drain Functionality

When the open drain feature is selected for P4.6 and/or P4.7, i.e. bits DP4.6 = 1 and/or DP4.7 = 1, the corresponding pins P4.6 and/or P4.7 will go into high impedance state instead of open drain mode.

Workaround:

Use standard push/pull mode for P4.6 and/or P4.7 in combination with appropriate external components (e.g. devices with open drain output functionality). In order to connect the CAN modules CAN1 and CAN2 to one CAN bus without bus transceivers, external diodes are necessary in combination with the standard push/pull mode for P4.6 and P4.7. See also application note AP2921 'On-Board Communication via CAN without Transceiver' on

<http://www.siemens.de/semiconductor/products/ics/34/pdf/ap292101.pdf>

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.

X12: P0H spikes after XPER write 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 an XBUS peripheral (e.g. XRAM, CAN, or I²C 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 approx. 7ns (spikes on P0H).

The spikes also occur if P0H is configured as input. However, read operations from P0H are not affected and will always return the correct logical state.

The spikes have the following position and shape in a typical application:

spikes occur after the rising edge of CLKOUT which follows the rising edge of ALE for the external bus cycle

P0H.x = low --> output low voltage rises to approx. 2.5V, spike width approx. 7ns (@ 0.2 Vcc)

P0H.x = high --> output high voltage drops to approx. 2.0V, spike width approx. 7ns (@ 0.8 Vcc)

Referring to a worst case simulation the maximum width of the spikes may be 15ns with full amplitude (Vcc/Vss). But this might not be seen on application level.

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.

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 the spikes on P0H may be filtered with an application specific RC element,
- 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

X15: Write to external memory followed by XRAM access

When a write access to external memory is performed in a **non-multiplexed** bus mode **without Memory Tristate Waitstate** (BUSCONx.5 = 1) and **without Early Write** (BUSCONx.8 = 0), and an XRAM access (operand read/write, instruction fetch) immediately follows this external write cycle, specific locations the internal XRAM may be overwritten. This may happen under the following conditions:

- in the 2 Kbyte XRAM (0E000h .. 0E7FFh), when a write access to a location s:p.yyyy (s: 8-bit segment number 0..FFh, p: 2-bit page number 0..3, yyyy: 14-bit page offset 2000h .. 27FFh) is performed, then location 0Eyyyh in the XRAM (yyy = 12 LSBs of page offset) is overwritten with the last value read from or written to any location in the XRAM
- in the 6 Kbyte XRAM (0C000h .. 0D7FFh), when a write access to a location s:p.zzzz (s: 8-bit segment number 0..FFh, p: 2-bit page number 0..3, zzzz: 14-bit page offset 0000h .. 17FFh), then location 0Czzzh or 0Dzzzh in the XRAM (zzz = 12 LSBs of page offset) is overwritten with the last value read from or written to any location in the XRAM.

Workaround:

use a Memory Tristate Waitstate (BUSCONx.5 = 0) when writing to external memory in a non-multiplexed bus mode without Early Write (BUSCONx.8 = 0), or use Early Write (BUSCONx.8 = 1) when writing to external memory in a non-multiplexed bus mode without Memory Tristate Waitstate (BUSCONx.5 = 1).

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 in Siemens 16-bit Microcontrollers" on

<http://www.siemens.de/semiconductor/products/ics/34/pdf/ap292401.pdf>

Bondout Specific Functional Problems:

BROM.1: Access to external memory while emulated internal ROM/Flash/OTP is disabled

When the emulated internal ROM/Flash/OTP is disabled (pin EA# = low during reset and/or bit SYSCON.ROMEN = 0),
and in register EMUCON a ROM size > 128 Kbytes is selected,
and an access to a location **s:FxXXh** (**x** = 200h .. DFFh, **s**: see table below) is performed,
then the emulated internal ROM/Flash/OTP will be accessed instead of external memory under the following conditions:

Selected ROM size	Selected segment s
512 Kbytes	3, 4, 5, 6, 7, 8
384 Kbytes	3, 4, 5, 6
256 Kbytes	3, 4
196 Kbytes	3

Workaround:

Do not select a ROM size > 128 Kbytes in register EMUCON when the internal ROM/Flash/OTP shall not be emulated in an application.

BX12: Disabling of CAN2 via XPERCON.1

Disabling of the internal CAN2 module via XPERCON.1 does not work: CAN2 is always enabled when bit SYSCON.XPEN = 1.

Deviations from Electrical- and Timing Specification:

under evaluation

Notes:

- 1) Pin **READY#** has an internal pull-up (all C167xx derivatives). This will be documented in the next revision of the Data Sheet.
- 2) Timing **t28**: Parameter description and test changed from 'Address hold after RD#/WR#' to 'Address hold after WR#'. It is guaranteed by design that read data are internally latched by the controller before the address changes.
- 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.

History List (since device step ES-AB)

Functional Problems

Functional Problem	Short Description	Fixed in step
PWRDN.1	Execution of PWRDN Instruction while pin NMI# = high	
CPU.16	Data read access with MOVB [Rn], mem instruction to internal ROM/Flash/OTP	
CPU.17	Arithmetic Overflow by DIVLU instruction	
CPU.18	Interrupted Multiply/Divide Instructions in internal Flash	
BUS.16	Branches to Code Segments 3 and 4 when internal Flash is disabled	
BUS.17	Spikes on CS# lines after access with RDCS# and/or WRCS#	
BUS.18	PEC transfers after JMPR	
RST.4	Power On Reset	ES-AC
ADC.11	Modifications of ADM field while bit ADST = 0	
PRT.2	P4.6 and P4.7: no Open Drain Functionality	
X9	Read Access to XPERs in Visible Mode	
X12	P0H spikes after XPER write access and external 8-bit Non-multiplexed bus	
X15	Write to external memory followed by XRAM access	
BROM.1	Access to external memory while emulated internal ROM/Flash/OTP is disabled	
BX12	Disabling of CAN2 via XPERCON.1	
BP0.1	Clock Configuration during reset	ES-AC
BINT.9	CAPCOM6 Emergency Interrupt Vector Location	ES-AC

AC/DC Deviations

AC/DC Deviation	Short Description	Fixed in step

Application Support Group, Munich