Errata Sheet

March 18, 1999 / Release 1.0

Device:	**SAF-C163-16F25F**
Stepping Code / Marking:	**ES-BC x, BC x**
Package:	**TQFP-100**

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: C163-16F Data Sheet 11.97,
	User's Manual: C165/C163 User's Manual V2.0 10.96
	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-BB x, BB x:

- Flash Programming and Erase Times (FLASH.5)
- PEC Transfers after JMPR (BUS.18)
- Spikes on CS# lines after access with RDCS# and/or WRCS# (BUS.17)
- Note about workaround for Tasking compiler added for problem CPU.16
- P0H spikes after XPER write access and external 8-bit Non-multiplexed bus (X12)

Functional Problems:

Pin (Vpp/)OWE

This pin is internally not connected (NC). An internal pull-down disables the oscillator watchdog.

FLASH.5: Flash Programming and Erase Times

The internal Flash programming and erase times have been modified as follows:

Programming:	4 ms (burst)
Erase:	16 ms (sector)

Note: with this modification, it is necessary to perform a Flash **erase** sequence **4 times** (i.e. 64 ms) in order to guarantee a proper erased state of the internal Flash.

All processes which depend on this time (e.g. watchdog service intervals, etc.) must be adapted accordingly. The typical values listed above may vary by up to ± 20 % over temperature.

Modified versions of the Flash programming software MEMTOOL will be available on our Internet page http://www.siemens.de/semiconductor/products/ics/34/index3.htm

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 preceeding the PWRDN instruction **writes** to external memory or an XPeripheral (SSP), 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 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.

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’ objectes 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 occured.

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 significat 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

CPU.18: Interrupted Multiply/Divide Instructions in internal Flash

When a multiply (MUL, MULU) or divide (DIV, DIVU, DIVL, DIVLU) instruction which is executed in internal Flash is interrupted, incorrect results may occur under the following conditions:

(1) 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 Flash** according to the following table:

s_{RETI} / s_{MD}	0	1	2
2	c	c	ok
1	ok	ok	c
0	ok	ok	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 Flash 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:

$ca = s_{RETI}:i_{MD}$	always when Flash is mapped to segment 1, or when Flash is mapped to segment 0 and $s_{RETI} \geq 2$
$ca = 1:i_{MD}$	when $s_{RETI} = 0$ and $i_{MD} \geq$ 8000h and Flash mapped to segment 0
$ca = 0:i_{MD}$	when $s_{RETI} = 1$ and $i_{MD} \leq$ 7FFEh and Flash mapped to segment 0

(2) the multipy/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.

Notes on compilers:

- **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.
- in the **Tasking** compiler, option –BM should be used to protect all multiply/divide instructions from being interrupted. Special libraries are included as described in the C compiler manual.

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 Vcc) from Vcc down to Vss (worst case) 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 conditiions 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_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 preceeding 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 preceeding 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 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 (SSP) 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. SSP 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.

Workaround:

- 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
```

Deviations from Electrical- and Timing Specification:

In the following, the deviations of the DC/AC characteristics from the specification in the C163-16F Data Sheet 11.97 are listed:

Test Conditions for IPD (Power Down Mode Current)

The test limit has been set to 300 µA.

Test Condition for IALEH (ALE inactive current)

This parameter is only tested at Vcc = 5.0 V

Notes:

1) Pin **READY**# has an internal pullup (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 pullups 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 BB)

Functional Problems

Functional Problem	Short Description	Fixed in step
	Pin (VPP/)OWE internally not connected	
BUS.18	PEC transfers after JMPR	
BUS.17	Spikes on CS# Lines after access with RDCS# and/or WRCS#	
CPU.16	Data read access with MOVB [Rn], mem instruction to internal ROM/Flash	
CPU.17	Arithmetic Overflow by DIVLU instruction	
CPU.18	Interrupted Multiply/Divide Instructions in internal Flash	
FLASH.5	Flash Programming and Erase Times	
PWRDN.1	Execution of PWRDN Instruction while pin NMI# = high	
X9	Read Access to XPERs in Visible Mode	
X12	P0H spikes after XPER write access and external 8-bit Non-multiplexed bus	

AC/DC Deviations

AC/DC Deviation	Short Description	Fixed in step
	Power Down Mode Current (test limit 300 µA)	
	ALE inactive current (IALEH, test at Vdd = 5.0 V only)	

Application Support Group, Munich