Microcontrollers

Errata Sheet

March 31, 2003 / Release 1.3

Device:	**SAK-C167CR-L(33)M,** **SAF-C167CR-L(33)M,** **SAB-C167CR-L(33)M**
	SAK-C167CR-4R(33)M, **SAF-C167CR-4R(33)M,** **SAB-C167CR-4R(33)M**
	SAK-C167CR-16R(33)M, **SAF-C167CR-16R(33)M,** **SAB-C167CR-16R(33)M**
	SAK-C167SR-L(33)M
Stepping Code / Marking:	**ES-GA, GA, GA-T, GA-T 6** **ES-JA**
Package:	**P-MQFP-144-1,** **P-MQFP-144-6 (GA-T 6)**

This Errata Sheet describes the deviations from the current user documentation.
The module oriented classification and numbering system uses an ascending sequence over several derivatives, including already solved deviations. So gaps inside this enumeration can occur.

The current documentation is: Data Sheet: C167CR/SR Data Sheet V3.2, 2001-07
User's Manual: C167CR Derivatives User's Manual V3.1, 2000-03
Instruction Set Manual V2.0, 2001-03

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 Sheets Rel. 1.2 for C167CR/SR devices with stepping code/marking (ES-)GA/JA/GA-T:

- New documentation reference: C167CR/SR Data Sheet V3.2, 2001-07
- Z Flag after PUSH and PCALL (CPU.22)
- Note on Early (Unlatched) Chip Select Option (Section Application Hints)
- Note and Link to Oscillator Application Note AP2420 updated (Section Functional Improvements/Documentation Updates)

Functional Problems:

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

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

(1) the A/D converter has finished a fixed channel single conversion of an analog channel n > 0 (i.e. contents of ADCON.ADCH = n during this conversion)
(2) the A/D converter is idle (i.e. ADBSY = 0)
(3) then the conversion mode in the ADC Mode Selection field ADM is changed to Auto Scan Single (ADM = 10b) or Continuous (ADM = 11b) mode without setting bit ADST = 1 with the same instruction

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

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.

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

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

(1) the previous 'instruction' is a PEC transfer which writes to IRAM, or any instruction with result write back to IRAM (addresses 0F200h..0FDFFh for 3 Kbyte module, 0F600h..0FDFFh for 2 Kbyte module, or 0FA00h..0FDFFh for 1 Kbyte module). For further restrictions on the destination address see case (a) or case (b) below.

(2) the BFLDx instruction immediately follows the previous instruction or PEC transfer within the instruction pipeline ('back-to-back' execution), i.e. decode phase of BFLDx and execute phase of the previous instruction or PEC transfer coincide. This situation typically occurs during program execution from internal program memory (ROM/OTP/Flash), or when the instruction queue is full during program execution from external memory

(3) the 3rd byte of BFLDx (= ***#mask8*** field of BFLD**L** or ***#data8*** field of BFLD**H**) and the destination address of the previous instruction or PEC transfer match in the following way:

(a) value of #mask8 of BFLDL or #data8 of BFLDH = 0F**y**h (**y** = 0..Fh),
and the previous instruction or PEC writes to (the low and/or high byte of) GPR R**y** or the memory address of R**y** (determined by the context pointer CP) via any addressing mode.

(b) value of #mask8 of BFLDL or #data8 of BFLDH = **00h..0EFh**,
and the lower byte $\mathbf{v_L}$ of the **contents v** of the IRAM location or (E)SFR or GPR which is read by BFLDx is $\mathbf{00h \le v_L \le 7Fh}$
and the previous instruction or PEC transfer writes to the (low and/or high byte of) the specific bit-addressable IRAM location $\mathbf{0FD00h + 2\ v_L}$ (i.e. the 8-bit offset address of the location in the bit-addressable IRAM area (0FD00h..0FDFFh) equals $\mathbf{v_L}$).

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

Notes:

Write operations in the sense of the problem description include implicit write accesses caused by
- auto-increment operations of the PEC source or destination pointers (which are located on 0FCE0h..0FCFEh in IRAM)
- post-increment/pre-decrement operations on GPRs (addressing modes with [R+] or [-R])
- write operations on the system stack (which is located in IRAM).

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

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

For implicit IRAM write operations caused by **auto-increment operations of the PEC source or destination pointers**, the problem can only occur if the value of #mask8 of BFLDL or #data8 of BFLDH = 0F**y**h (**y** = 0..Fh), and the range which is covered by the context pointer CP (partially or completely) overlaps the PEC source and destination pointer area (0FCE0h..0FCFEh), and the address of the source or destination pointer which is auto-incremented after the PEC transfer is equal to the address of GPR R**y** (included in case 3a).

For **system stack write operations**, the problem can only occur if the system stack is located in the bit-addressable portion of IRAM (0FD00h..0FDFFh), or if the system stack can overlap the register bank area (i.e. the register bank area is located below the system stack, and the distance between the contents of the context pointer CP and the stack pointer SP is ≤ 20h).

Workaround 1:

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

(a) an equivalent sequence of single bit instructions. This sequence may be included in an uninteruptable ATOMIC or EXTEND sequence to ensure completion after a defined time.

(b) an equivalent byte- or word MOV or logical instruction.

Note that byte operations to SFRs always clear the non-addressed complementary byte.
Note that protected bits in SFRs are overwritten by MOV or logical instructions.

Workaround 2:

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

Workaround 3:

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

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

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

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

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

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

- - : case can not occur

Tool Support for Problem CPU.21

The **Keil** C166 Compiler V3.xx generates BFLD instructions only in the following cases:
- when using the _bfld_ intrinsic function.
- at the beginning of interrupt service routines, when using #pragma disable
- at the end of interrupt service routines, when using the chip bypass directive FIX166.

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

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

The RTX166 Full Real-Time Operating system (any version) does not use BFLD instructions.
For RTX166 Tiny, you should rebuild the RTX166 Tiny library with the SET FIXBFLD = 1 directive. This directive is enabled in the assembler source file RTX166T.A66. After change of this setting rebuild the RTX166 Tiny library that you are using in your application.

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

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

CPU.22: Z Flag after PUSH and PCALL

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

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

Examples:

`PUSH R1`	(coding: **F**1 EC):	incorrect setting of Z flag if contents of R**15** is odd, and 00FFh ≥ contents of R1 ≥ 0001h
`PUSH DPP3`	(coding: **0**3 EC):	incorrect setting of Z flag if contents of R**0** is odd, and 00FFh ≥ contents of DPP3 ≥ 0001h

(b) for ***PCALL reg, rel*** instructions:
- when the contents of the high byte of the GPR or (E)SFR which is pushed is 00h, and
- when the contents of the low byte of the GPR or (E)SFR which is pushed is odd

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

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

Example:

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

Effect on Tools:

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

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

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

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

The **TASKING** compiler V7.5r2 never generates a PCALL instruction, nor is it used in the libraries. The PUSH instruction is only used in the entry of an interrupt frame, and sometimes on exit of normal functions. The zero flag is not a parameter or return value, so this does not give any problems.

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

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

Workaround (for program parts written in assembler):

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

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

or

```
   PCALL reg, procedure_1
   ...
```

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

Hints for Detection of Critical Instruction Combinations

Whether or not an instruction following *PUSH reg* or *PCALL reg, rel* actually causes a problem depends on the program context. In most cases, it will be sufficient to just analyze the instruction following PUSH or PCALL. In case of PCALL, this is the instruction at the call target address.

- Support Tool for Analysis of Hex Files

For complex software projects, where a large number of assembler source (or list) files would have to be analyzed, Infineon provides a tool **aiScan22** which scans hex files for critical instruction sequences and outputs diagnostic information. This tool is available as part of the Application Note **ap1679** 'Scanning for Problem CPU.22' on the 16-bit microcontroller internet pages of Infineon Technologies:
http://www.infineon.com/cgi/ecrm.dll/ecrm/scripts/prod_cat.jsp?oid=-8984

direct links:
http://www.infineon.com/cmc_upload/documents/040/841/ap1679_v1.1_2002_05_scanning_cpu22.pdf
http://www.infineon.com/cgi/ecrm.dll/ecrm/scripts/public_download.jsp?oid=40840&parent_oid=-8137

- Individual Analysis of Assembler Source Code

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

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

- These instructions may **only** cause a problem if the **PSW** is a source or source/destination operand:

  ADD/B, ADDC/B, CMP/B, CMPD1/2, CMPI1/2, SUB/B, SUBC/B
  AND/B, OR/B, XOR/B
  ASHR
  MOV/B, MOVBZ/MOVBS
  SCXT
  PUSH, PCALL ; analysis must be repeated for successor of PUSH/PCALL

- The following instructions (most of them with immediate or register (Rx) addressing modes) can **never** cause a problem when they are successors of PUSH/PCALL:

  CPL/B, NEG/B
  DIV/U, DIVL/U, MUL/U
  SHL/SHR, ROL/ROR, PRIOR
  POP
  RETI ; updates complete PSW with stacked value
  RETP ; updates condition flags
  PWRDN ; program restarts after reset
  SRST ; program restarts

- **Conditional branch instructions** which may evaluate the Z flag as successors of PUSH/PCALL:
  JB/JNB Z, *rel* ; directly evaluates Z flag
  CALLA/CALLI, JMPA/JMPI/JMPR with the following condition codes
  cc_Z, cc_EQ, cc_NZ, cc_NE
  cc_ULE, cc_UGT, cc_SLE, cc_SGT
  cc_NET

- For these branch conditions, the branch may be performed in the wrong way.

- For other branch conditions, the branch target as well as the linear successor of the branch instruction must be analyzed (since these branch instruction don't modify the PSW flags).

- For **instructions that have no effect on the condition flags** and that don't evaluate the Z flag, the instruction that follows this instruction must be analyzed. These instructions are

  NOP
  ATOMIC, EXTxx
  DISWDT, EINIT, IDLE, SRVWDT
  CALLR, CALLS, JMPS ; branch target must be analyzed
  RET, RETS ; return target must be analyzed (value pushed by PUSH/PCALL = return IP,
  ; Z flag contains information whether intra-segment target address = 0000h or not)
  TRAP ; both trap target and linear successor must be analyzed, since Z flag may be
  ; incorrect in PSW on stack as well as in PSW at entry of trap routine

- For **bit modification instructions**, the problem may only occur if a source bit is the Z flag, and/or the destination bit is in the PSW, but not the Z flag. These instructions are:

  BMOV/BMOVN
  BAND/BOR/BXOR
  BCMP
  BFLDH
  BFLDL ; problem only if bit 3 of @@ mask = 0, i.e. if Z is not selected
  BCLR/BSET ; problem only if operand is not Z flag
  JBC/JNBS ; wrong branch if operand is Z flag

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.

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, typically about 0.8 Vcc (4.0 V @ Vcc = 5.0V)) may occur on Port 6 lines configured as CS# signals (in default configuration as 'latched chip selects, SYSCON.6/CSCFG = 0). 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_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 (as reported by compiler manufacturers):

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 ≥ V4.02 - in combination with directive FIXPEC when OPTIMIZE(7) is selected -, and version 3.12o, including the associated 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.

With other versions, the problem may occur e.g. in nested for/while loops, when the inner loop looks as follows:

Example i):

```
    while (..)  {
    while (variable == constant)    {

    <last statement is a modification of variable value>
   }
  } ...
```

Example ii):

```
    for (..)  {
    for ( ; variable < 100; variable++)  {
       ..
    }
  }
```

The critical JMPR-JMPR sequence does not occur when a for loop is used with constant initialization, e.g..

```
    while (...)
    for (variable = 0; variable < 100; variable++)  {
     ..
   }
 }
```

Recommendation: use V4.03 (or higher), or V3.12o, or insert *nop* () in nested loops e.g. as follows:

```
    void test(int i, int k)  {
    while (k) {
        nop ();
        while (i) {
        i--;
   };
    k--;
 };
}
```

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'.

BUS.19: Unlatched Chip Selects at Entry into Hold Mode

Unlike in standard (latched) configuration, the chip select lines in unlatched configuration (SYSCON.CSCFG = 1) are not driven high for 1 TCL after HLDA# is driven low, but start to float when HLDA# is driven low.

OWD.1: Function of Bit OWDDIS/SYSCON.4

The status of bit OWDDIS/SSYCON.4 has no effect on the oscillator watchdog, i.e. the oscillator watchdog can not be disabled or enabled by bit OWDDIS. The oscillator watchdog can only be disabled by a low level on pin OWE (84). An internal pull-up holds this pin high in case it is left unconnected, thus enabling the oscillator watchdog in direct drive or prescaler mode.

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.

CAN.7 Unexpected remote frame transmission

Symptom

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

Detailed Description

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

- Set CPUUPDn element
- Reset TXRQn element
- Reset MSGVALn element

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

The symptom described above occurs when the CPU accesses CPUUPD, TXRQ or MSGVAL, while the pending transmit request of the corresponding message object is transferred to the CAN state machine (just before start of frame transmission). At this particular time the transmit request is transferred to the CAN state machine before the CPU prevents transmission. In this case the transmit request is still accepted from the CAN state machine. However the transfer of the identifier, the data length code and the data of the corresponding message object is prevented. Then the pre-charge values of the internal “hidden buffer” are transmitted instead, this causes a remote frame transmission with identifier=0 (11 bit) and data length code=0.
This behavior occurs only when the transmit request of message object n is pending and the transmit requests of other message objects are **not** active (single transmit request).
If this remote frame loses arbitration (to a data frame with identifier=0) or if it is disturbed by an error frame, it is **not** retransmitted.

Effects to other CAN nodes in the network

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

Workarounds

1. The behavior can be avoided if a message object is not updated by software when a transmission of the corresponding message object is pending (TXRQ element is set) **and** the CAN module is active (INIT = 0). If a re-transmission of a message (e.g. after lost arbitration or after the occurrence of an error frame) needs to be cancelled, the TXRQ element should be cleared by software as soon as NEWDAT is reset from the CAN module.
2. The nodes in the CAN system ignore the remote frame with the identifier=0 and **no** data frame is triggered by this remote frame.

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

After any reset, the contents of the CAN Message Objects 1..15 (MCR, UAR, LAR, MCFG, Data[0:7]) and the Mask of Last Message Registers (LMLM, UMLM) may be undefined instead of unchanged (reset value 'X' instead of 'U').

This problem depends on temperature and the length of the reset, and differs from device to device. The problem is more likely (but not restricted) to occur at high temperature and for long hardware resets (> 100 ms).

Workaround:

Re-initialize the CAN module after each reset.

Application Hints

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.infineon.com/cgi/ecrm.dll/ecrm/scripts/prod_cat.jsp?oid=-8984

Handling of the SSC Busy Flag (SSCBSY)

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

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

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

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

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

Oscillator Watchdog and Prescaler Mode

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

- In **direct drive** mode the PLL base frequency is used directly (fcpu = 2...5 MHz).
- In **prescaler** mode the PLL base frequency is divided by 2 (fcpu = 1...2.5 MHz).

PLL lock after temporary clock failure

When the PLL is locked and the input clock at XTAL1 is interrupted then the PLL becomes unlocked, provides the base frequency (2 ... 5 MHz) and the PLL unlock interrupt request flag is set. If the XTAL1 input clock starts oscillation again then the PLL stays in the PLL base frequency. The CPU clock source is only switched back to the XTAL1 oscillator clock after a hardware reset. This can be achieved via a normal hardware reset or via a software reset with enabled bidirectional reset. It is important that the hardware reset is at least active for 1 ms, after that time the PLL is locked in any case.

Note on Early (Unlatched) Chip Select Option

As described in the User's Manuals (e.g. C167CR User's Manual, V3.1, 2000-03, p.9-11), an early (unlatched) address chip select signal (SYSCON.CSCFG = ‘1’) becomes active together with the address and BHE (if enabled) and remains active until the end of the current bus cycle. Early address chip select signals are not latched internally and may toggle intermediately while the address is changing.

These effects may also occur on CSx# lines which are configured as RDCSx# and/or WRCSx# signals (BUSCONx.CSRENx = 1 and/or CSWENx = 1).

The position of these transitions (spikes) is at the beginning of an external bus cycle or an internal XBUS cycle, indicated by the rising edge of signal ALE. The width of these transitions is ~ 5 ns (measured at a reference level of 2.0 V with Vdd = 5.0 V). The falling edge of the spike occurs in the same relation to RD#, WR#/WRH#/WRL# and to other CS# signals as if it was an address chip select signal with early chip select option.

When CS# lines configured as RDCS# and/or WRCS# are used e.g. as output enable (OE#) signals for external devices or as clock input for shift registers, problems might occur (temporary bus contention during data float times (may be solved by tristate wait state), unexpected shift operations, etc.). When CS# lines configured as WRCS# are used as write enable (WE#) signals for external devices or FIFOs, internal locations may be overwritten with undefined data.

When CS# lines are used as chip enable (CE#) signals for external memories, usually no problems are expected, since the falling edge of the spikes has the same characteristics as the falling edge of an access with a regular early (unlatched) address CS# signal. At this time, the memory control signals RD#, WR# (WRH#/WRL#) are on their inactive (high levels).

Deviations from Electrical- and Timing Specification:

The following table lists the deviations of the DC/AC characteristics from the specification in the C167CR/SR Data Sheet V3.2, 2001-07:

Problem short name	Parameter	Symbol	Limit Values		Unit	Test Condition
			min.	max.		
DC.IALEL.1	ALE inactive current	I_{ALEL}	-	**30** instead of 40	µA	$V_{OUT} = V_{OLmax}$
DC.tc8.5	CLKOUT rise time	tc8	-	**5** instead of 4	ns	
DC.tc9.5	CLKOUT fall time	tc9	-	**5** instead of 4	ns	
AC.PLL.1	PLL base frequency	-	2...**6** MHz instead of 2...5 MHz		MHz	(see User's Manual, chapter Clock Generation – PLL Operation)
AC.PLL.2	PLL base frequency	-	2...**8** MHz instead of 2...5 MHz		MHz	devices with date code ≥ 0114 only

- A/D Converter Characteristics:

ADCC.2.3: ADC Overload Current

During exceptional conditions in the application system an overload current I_{OV} can occur on the analog inputs of the A/D converter when $V_{AIN} > V_{dd}$ or $V_{AIN} < V_{ss}$. For this case, the following conditions are specified in the Data Sheet:

$I_{OVmax} = | \pm 5 \text{ mA} |$

The specified total unadjusted error $TUE_{max} = | \pm 2 \text{ LSB} |$ is only guaranteed if overload conditions occur on maximum 2 not selected analog input pins and the absolute sum of input overload currents on all analog input pins does not exceed 10 mA. (It is also allowed to distribute the overload to more than 2 not selected analog input pins).

Due to an internal problem, the specified TUE value is only met for a **positive** overload current $0 \text{ mA} \leq I_{OV} \leq +5 \text{ mA}$ (all currents flowing into the microcontroller are defined as positive and all currents flowing out of it are defined as negative).

If the exceptional conditions in the application system cause a **negative** overload current, then the maximum TUE can be exceeded (depending on value of I_{OV} and R_{AREF}):

Problem Description in Detail:

1. **Overload Current at analog Channel ANn (n ∈ 1 ... 11) and Influence to V_{AREF}**
   If an overload current I_{OV} occurs on analog input channel ANn, then an additional current I_{AREF} (crosstalk current) is caused at pin V_{AREF}.
   Depending on R_{AREF}, the internal resistance of the reference voltage, the crosstalk current I_{AREF} at pin V_{AREF} can cause an additional unadjusted error AUE to all other analog channels.
   In case $R_{AREF} \leq 490$ Ohm [$R_{AREF} \leq$ ((LSB/2) / (I_{OVmax} * ovf-3)] the maximum possible additional

error to all other channels is smaller than 0.5 LSB with the condition of I_{OVmax} = | ±5 mA| at ANn.

Relation between I_{AREF} and I_{OV} at ANn: $I_{AREF} = ovf\text{-}3 * I_{OVn}$ ($n \in 1 \ldots 11$)

Note: The influence to the reference voltage V_{AREF} caused by I_{OVn} (shift of V_{AREF}) is maximum for $V_{AINn} = V_{AREF}$ and the influence is minimum for V_{AINn} = 0V ($n \in 1 \ldots 11$). The condition $R_{AREF} \leq 490$ Ohm and 0.5 LSB is calculated for the worst case at $V_{AINn} = V_{AREF}$.

2. Values of ovf-3

Parameter	Symbol	Min	Max
Overload factor-3	ovf-3	- 0.001	0

History List C167CR-LM (since device step BA)

Functional Problem	Short Description	Fixed in step
ADC.11	Modifications of ADM field while bit ADST = 0	
BUS.17	Spikes on CS# lines after access with RDCS# and/or WRCS# (not in BE and earlier steps)	
BUS.18	PEC transfers after JMPR	
BUS.19	Unlatched Chip Selects at Entry into Hold Mode (not in BE and earlier steps)	
OWD.1	Function of Bit OWDDIS/SYSCON.4 (not in BE and earlier steps)	
PWRDN.1	Execution of PWRDN Instruction while pin NMI# = high	
X9	Read Access to XPERs in Visible Mode	
CAN.7	Unexpected Remote Frame Transmission	
CAN.9	Contents of Message Objects and Mask of Last Message Registers after Reset	
CPU.21	BFLDL/H Instructions after Write Operation to internal IRAM	
CPU.22	Z Flag after PUSH and PCALL	
ADC.8	CC31/ADC Interference	BE
ADC.10	Start of Standard Conversion at End of Injected Conversion	CB
CPU.8	Jump instruction in EXTEND sequence	BE
CPU.9	PEC Transfers during instruction execution from Internal RAM	CB
CPU.11	Stack Underflow during Restart of Interrupted Multiply	BE
CPU.17	Arithmetic Overflow by DIVLU instruction	(EES-)FA
RST.1	System Configuration via P0L.0 during Software/Watchdog Timer Reset	CB
SSC.8	Data Transmission in Slave Mode (Step EES-FA only)	ES-FA
X10	P0H I/O conflict during XPER access and external 8-bit Non-multiplexed bus	BE
X12	P0H spikes after XPER write access and external 8-bit Non-multiplexed bus (Step BE until step DB only)	(EES-)FA
PINS.1	OUTPUT Signal Rise Time (DA-step only)	(EES-)FA

AC/DC Deviation	Short Description	Fixed in step
DC.IALEL.1	ALE inactive current 30 µA (steps ≥ FA only)	
AC.PLL.1	PLL base frequency max 6 MHz (GA-, GA-T-, JA-steps only)	
AC.PLL.2	PLL base frequency 8 MHz (GA-, GA-T-, JA-steps with date code ≥ 0114)	
DC.tc8.5	CLKOUT rise time (GA-, GA-T-, JA-steps only)	
DC.tc9.5	CLKOUT fall time (GA-, GA-T-, JA-steps only)	
ADCC.2.3	ADC Overload Current (steps ≥ FA only)	
DC.IALEH.1	ALE active current 1000µA (DA-step only)	(EES-)FA
DC.IRWL.1	RD#/WR# active current –600µA (DA-step only)	(EES-)FA

DC.IP6L.1	Port 6 active current –600 µA (DA-step only)	(EES-)FA
DC.IP0L.1	Port 0 configuration current –110µA (DA-step only)	(EES-)FA
AC.t5.1	ALE high time TCL-15ns (DA-step only)	(EES-)FA
AC.t12.1	WR#/WRH# low time (with RW-delay) 2TCL-12ns (DA-step only)	(EES-)FA
AC.t13.1	WR#/WRH# low time (no RW-delay) 3TCL-12ns (DA-step only)	(EES-)FA
AC.t15.1	RD# to valid data in 3TCL-25ns (step BE only)	(EES-)FA
AC.t16.1	ALE low to valid data in 3TCL-25ns (step BE only)	(EES-)FA
AC.t34.1	CLKOUT rising edge to ALE falling edge 12ns (step BE only)	(EES-)FA
AC.t38.2	ALE falling edge to CS# -10ns (step BE only)	(EES-)FA
AC.t38.1	ALE falling edge to CS# -7ns (DA-step only)	(EES-)FA
AC.t48.1	RDCS#/WRCS# low time (with RW-delay) 2TCL-12ns (DA-step only)	(EES-)FA
AC.t49.1	RDCS#/WRCS# low time (no RW-delay) 3TCL-12ns (DA-step only)	(EES-)FA
ADCC.2.1	ADC Overload Current (CB-step only)	DA
ADCC.2.2	ADC Overload Current (DA- and DB-step only)	(EES-)FA

History List C167SR-LM (since device step BA)

Functional Problem	Short Description	Fixed in step
ADC.11	Modifications of ADM field while bit ADST = 0	
BUS.17	Spikes on CS# lines after access with RDCS# and/or WRCS# (not in BA and earlier steps)	
BUS.18	PEC transfers after JMPR	
BUS.19	Unlatched Chip Selects at Entry into Hold Mode (not in BA and earlier steps)	
CPU.21	BFLDL/H Instructions after Write Operation to internal IRAM	
CPU.22	Z Flag after PUSH and PCALL	
OWD.1	Function of Bit OWDDIS/SYSCON.4 (not in BA and earlier steps)	
PWRDN.1	Execution of PWRDN Instruction while pin NMI# = high	
X9	Read Access to XPERs in Visible Mode	
ADC.8	CC31/ADC Interference	DA
ADC.10	Start of Standard Conversion at End of Injected Conversion	DA
CPU.8	Jump instruction in EXTEND sequence	DA
CPU.9	PEC Transfers during instruction execution from Internal RAM	DA
CPU.11	Stack Underflow during Restart of Interrupted Multiply	DA
CPU.17	Arithmetic Overflow by DIVLU instruction	(ES-)FA
RST.1	System Configuration via P0L.0 during Software/Watchdog Timer Reset	DA
X10	P0H I/O conflict during XPER access and external 8-bit Non-multiplexed bus	DA
X12	P0H spikes after XPER write access and external 8-bit Non-multiplexed bus (DA-step only)	(ES-)FA

PINS.1	OUTPUT Signal Rise Time (DA-step only)	(ES-)FA
AC/DC Deviation	**Short Description**	**Fixed in step**
DC.IALEL.1	ALE inactive current 30 µA (steps ≥ FA only)	
AC.PLL.1	PLL base frequency (GA-, GA-T-, JA-steps only)	
AC.PLL.2	PLL base frequency 8 MHz (GA-, GA-T-, JA-steps with date code ≥ 0114)	
DC.tc8.5	CLKOUT rise time (GA-, GA-T-, JA-steps only)	
DC.tc9.5	CLKOUT fall time (GA-, GA-T-, JA-steps only)	
ADCC.2.3	ADC Overload Current (steps ≥ FA only)	
DC.IALEH.1	ALE active current 1000µA (DA-step only)	(ES-)FA
DC.IRWL.1	RD#/WR# active current –600µA (DA-step only)	(ES-)FA
DC.IP6L.1	Port 6 active current –600 µA (DA-step only)	(ES-)FA
DC.IP0L.1	Port 0 configuration current –110µA (DA-step only)	(ES-)FA
AC.t5.1	ALE high time TCL-15ns (DA-step only)	(ES-)FA
AC.t12.1	WR#/WRH# low time (with RW-delay) 2TCL-12ns (DA-step only)	(ES-)FA
AC.t13.1	WR#/WRH# low time (no RW-delay) 3TCL-12ns (DA-step only)	(ES-)FA
AC.t38.1	ALE falling edge to CS# -7ns (DA-step only)	(ES-)FA
AC.t48.1	RDCS#/WRCS# low time (with RW-delay) 2TCL-12ns (DA-step only)	(ES-)FA
AC.t49.1	RDCS#/WRCS# low time (no RW-delay) 3TCL-12ns (DA-step only)	(ES-)FA
ADCC.2.2	ADC Overload Current (DA-step only)	(ES-)FA

History List C167CR-4RM (since device step AB)

Functional Problem	**Short Description**	**Fixed in step**
ADC.11	Modifications of ADM field while bit ADST = 0	
BUS.17	Spikes on CS# Lines after access with RDCS# and/or WRCS#	
BUS.18	PEC Transfers after JMPR Instruction	
BUS.19	Unlatched Chip Selects at Entry into Hold Mode (not in DB- and earlier steps)	
CPU.21	BFLDL/H Instructions after Write Operation to internal IRAM	
CPU.22	Z Flag after PUSH and PCALL	
OWD.1	Function of Bit OWDDIS/SYSCON.4	
PWRDN.1	Execution of PWRDN Instruction while pin NMI# = high	
X9	Read Access to XPERs in Visible Mode	
CAN.7	Unexpected Remote Frame Transmission	
CAN.9	Contents of Message Objects and Mask of Last Message Registers after Reset	
CPU.8	Jump instruction in EXTEND sequence	AC

CPU.9	PEC Transfers during instruction execution from Internal RAM	AC
CPU.11	Stack Underflow during Restart of Interrupted Multiply	AC
CPU.16	Data read access with MOVB [Rn], mem instruction to internal ROM	(EES-)FA
CPU.17	Arithmetic Overflow by DIVLU instruction	(EES-)FA
RST.1	System Configuration via P0L.0 during Software/Watchdog Timer Reset	AC
RST.3	Bidirectional Hardware Reset	DA
ADC.8	CC31/ADC Interference	AC
ADC.10	Start of Standard Conversion at End of Injected Conversion	AC
SSC.8	Data Transmission in Slave Mode (Step EES-FA only)	ES-FA
X12	P0H spikes after XPER write access and external 8-bit Non-multiplexed bus	(EES-)FA
PINS.1	OUTPUT Signal Rise Time (DA-step only)	DB

AC/DC Deviation	**Short Description**	**Fixed in step**
DC.IALEL.1	ALE inactive current 30µA (steps ≥ FA only)	
AC.PLL.1	PLL base frequency (GA-, GA-T-, JA-steps only)	
AC.PLL.2	PLL base frequency 8 MHz (GA-, GA-T-, JA-steps with date code ≥ 0114)	
DC.tc8.5	CLKOUT rise time (GA-, GA-T-, JA-steps only)	
DC.tc9.5	CLKOUT fall time (GA-, GA-T-, JA-steps only)	
ADCC.2.3	ADC Overload Current (steps ≥ FA only)	
DC.VOL.1	Output low voltage (Port0/1/4, ALE, RD#, WR#, ...) test condition 1.6mA (AC-step only)	DA
DC.IALEH.1	ALE active current 1000µA	(EES-)FA
DC.IRWL.1	RD#/WR# active current –600µA	(EES-)FA
DC.IP6L.1	Port 6 active current –600 µA	(EES-)FA
DC.IP0L.1	Port 0 configuration current –110µA (problem not in AC step)	(EES-)FA
DC.HYS.1	Input Hysteresis 300mV (restriction not effective in production test)	-
AC.t5.1	ALE high time TCL-15ns	DB
AC.t12.1	WR#/WRH# low time (with RW-delay) 2TCL-12ns	(EES-)FA
AC.t13.1	WR#/WRH# low time (no RW-delay) 3TCL-12ns	(EES-)FA
AC.t38.1	ALE falling edge to CS# -7ns	(EES-)FA
AC.t48.1	RDCS#/WRCS# low time (with RW-delay) 2TCL-12ns	(EES-)FA
AC.t49.1	RDCS#/WRCS# low time (no RW-delay) 3TCL-12ns	(EES-)FA
ADCC.2.2	ADC Overload Current	(EES-)FA

History List C167CR-16RM (since device step AA)

Functional Problem	Short Description	Fixed in step
ADC.10	Start of Standard Conversion at End of Injected Conversion	(EES-)FA
ADC.11	Modifications of ADM field while bit ADST = 0	
BUS.17	Spikes on CS# lines after access with RDCS# and/or WRCS# (FA steps only)	
BUS.18	PEC transfers after JMPR	
BUS.19	Unlatched Chip Selects at Entry into Hold Mode (not in step AA)	
OWD.1	Function of Bit OWDDIS/SYSCON.4 (not in step AA)	
CPU.9	PEC Transfers during instruction execution from Internal RAM	(EES-)FA
CPU.12	Access to internal ROM with EXTS/EXTSR instructions	(EES-)FA
CPU.16	Data read access with MOVB [Rn], mem instruction to internal ROM	(EES-)FA
CPU.17	Arithmetic Overflow by DIVLU instruction	(EES-)FA
CPU.21	BFLDL/H Instructions after Write Operation to internal IRAM	
CPU.22	Z Flag after PUSH and PCALL	
PWRDN.1	Execution of PWRDN Instruction while pin NMI# = high	
ROM.1	Internal ROM access to locations 28000h ... 2FFFFh	(EES-)FA
RST.1	System Configuration via P0L.0 during Software/Watchdog Timer Reset	(EES-)FA
SSC.8	Data Transmission in Slave Mode (Step EES-FA only)	ES-FA
CAN.7	Unexpected Remote Frame Transmission	
CAN.9	Contents of Message Objects and Mask of Last Message Registers after Reset	
X9	Read Access to XPERs in Visible Mode	
X12	P0H spikes after XPER write access and external 8-bit Non-multiplexed bus	(EES-)FA

AC/DC Deviation	Short Description	Fixed in step
DC.IALEL.1	ALE inactive current 30 µA (steps ≥ FA only)	
AC.PLL.1	PLL base frequency (GA-, GA-T-, JA-steps only)	
AC.PLL.2	PLL base frequency 8 MHz (GA-, GA-T-, JA-steps with date code ≥ 0114)	
DC.tc8.5	CLKOUT rise time (GA-, GA-T-, JA-steps only)	
DC.tc9.5	CLKOUT fall time (GA-, GA-T-, JA-steps only)	
ADCC.2.3	ADC Overload Current (steps ≥ FA only)	
AC.t15.1	RD# to valid data in 3TCL-25ns	(EES-)FA
AC.t16.1	ALE low to valid data in 3TCL-25ns	(EES-)FA
AC.t34.1	CLKOUT rising edge to ALE falling edge 12ns	(EES-)FA
AC.t38.2	ALE falling edge to CS# -10ns	(EES-)FA

Functional Improvements/Documentation Updates

Compared to the BA-step, the following feature enhancements have been implemented in the FA- and GA- (and all following) steps of the C167CR/SR. They are described in detail in the respective chapters of the C167CR Derivatives User's Manual V3.1 (2000-03), and are summarized here for easier reference.

Incremental position sensor interface

For each of the timers T2, T3, T4 of the GPT1 unit, an additional operating mode has been implemented which allows to interface to incremental position sensors (A, B, Top0). This mode is selected for a timer Tx via TxM = 110b in register TxCON, x = (2, 3, 4). Optionally, the contents of T5 may be captured into register CAPREL upon an event on T3. This feature is selected via bit CT3 = 1 in register T5CON.10.

Oscillator Watchdog

The Oscillator Watchdog (OWD) monitors the clock at XTAL1 in direct drive and prescaler mode. In case of clock failure, the PLL Unlock/OWD Interrupt Request Flag (XP3IR) is set and the internal CPU clock is supplied with the PLL basic frequency. This feature can be disabled by a low level on pin Vpp/OWE. Bit OWDDIS/SYSCON.4 allows to disable this feature via software on device steps where problem OWD.1 is fixed.

Bidirectional Reset

Optionally, an internal watchdog timer or software reset will be indicated on the RSTIN# pin which will be driven low for the duration of the internal reset sequence. RSTIN# will also be driven low for the duration of the internal reset sequence when this reset was initiated by an external HW reset signal on pin RSTIN#.

This option is selectable by software via bit BDRSTEN/SYSCON.3. After reset, the bidirectional reset option is disabled (BDRSTEN/SYSCON.3 = 0).

Reset Source Indication in Register WDTCON

Besides indication of a watchdog timer reset in bit WDTR in register WDTCON, the FA-/GA-steps additionally allow indication of other reset sources and types (software reset, long/short hardware reset, etc.) in status flags in the low byte of register WDTCON. While in previous steps, only reset values 0000h or 0002h could occur for WDTCON, in the FA-/GA-steps further values may occur in the low byte of WDTCON. Therefore, programs written for previous steps which evaluate the contents of WDTCON after reset and which explicitly test bit WDTR either via bit instructions or via mask operations will work identically on the FA-/GA-steps. However, programs which assume that all other bits in the low byte of WDTCON except bit WDTR are always '0' (which is true for previous steps) and therefore e.g. test WDTCON with byte or word operations may work differently on the FA-/GA-steps.

The following table summarizes the behaviour of the reset source indication flags.

Flag Event	LHWR WDTCON.4	SHWR WDTCON.3	SWR WDTCON.2	WDTR WDTCON.1
Long HW Reset	1	1	1	0
Short HW Reset	- / *	1	1	0
SRST instruction	- / *	- / *	1	-
WDT Reset	- / *	- / *	1	1
EINIT instruction	0	0	0	-
SRVWDT instruction	-	-	-	0

Legend: 1 = flag is set, 0 = flag is cleared, - = flag is not affected,
* = flag is set when bi-directional reset option is enabled

XBUS Peripheral Enable Bit XPEN/SYSCON.2 (does not apply to C167SR)

Bit SYSCON.2 has been modified into a general XBUS Peripheral Enable bit, i.e. it controls both the XRAM **and the CAN module**.

When bit SYSCON.2 = 0 (default after reset), and an access to an address in the range EF00h ... EFFFh is made, either an external bus access is performed (if an external bus is enabled), or the Illegal Bus Trap is entered. In previous versions, the CAN module was accessed in this case. Systems where bit SYSCON.2 was set to '1' before an access to the CAN module in the address range EF00h ... EFFFh was made will work without problems with all steps of the C167CR.

Clock System

- In total 8 different clock configuration options are selectable during reset on P0H.7..5 (direct drive, prescaler 0.5, PLL factors 2, 3, 4, 5, 1.5, 2.5). Some options are configured via settings on P0H.7..5 during reset which would have selected Direct Drive in previous steps (for details, see Appendix in Errata Sheet V1.x of respective device):

Reset Configuration P0H.[7:5]	CPU Frequency fcpu = fxtal * F	Notes
011	fxtal * 1	Direct Drive
010	fxtal * 1.5	1)
001	fxtal / 2	Prescaler Operation, 1)
000	fxtal * 2.5	1)

1) **Note**: previous steps have selected Direct Drive when P0H.[7:5] = 0XX, i.e. the level on P0H.6 and P0H.5 during reset was not evaluated.

- In addition, in each of the steps FA-/GA-/GA-T, the internal oscillator circuit (Type_RE) has been improved and adjusted to the respective technology. The Type_RE oscillator is compatible to the Type_R oscillator with respect to the size of the components for an external crystal oscillator circuit. In any case, it is recommended to check the safety factor of the oscillator circuit in the target system. See Application Note AP2420 'Crystal Oscillator of the C500 and C166 Microcontroller Families' on

  http://www.infineon.com/cgi/ecrm.dll/ecrm/scripts/prod_cat.jsp?oid=-8984

  direct link: http://www.infineon.com/cmc_upload/documents/009/746/ap242005.pdf

External Bus Controller

By default, the CS# signals (when used as address CS# signals) are switched nominally 1 TCL after the address for an external bus access is driven. This ensures a defined transition from active to inactive state without glitches. Optionally, controlled by bit CSCFG/SYSCON.6 = 1, the leading edge of the CS# signals may be generated in an unlatched mode, i.e. the CS# signals are directly derived from the addresses and are switched in the same internal clock phase as the addresses. This allows more time for the 'chip enable access time' tce of external devices, however, glitches may occur on CS# lines while the addresses are changing.

Port Driver Control Register

Beginning with the FA-step, the driving capability of the pad drivers can be selected via software in register PDCR (ESFR address 0F0AAh). Two driving levels (fast edge mode/reduced edge mode) can be selected for two groups of pins. Bit PDCR.0/BIPEC controls the edge characteristic of Bus Interface Pins (PORT0/1, port 4, port 6, RD#, WR#, ALE, WRH#/BHE, CLKOUT), while bit PDCR.4/NBPEC controls Non-Bus Pins (port 2, port 3, port 7, port 8, RSTOUT#, RSTIN# in bidirectional mode). The reset value '0' selects fast edge mode to ensure compatibility with previous versions and steps.

Port 5 Digital Input Control via register P5DIDIS

Beginning with the FA-step, the digital input stages on port 5 may be disconnected from pins used as analog inputs via register P5DIDIS.

A/D Converter

Due to correction of the former problem ADC.7, injected conversions will no longer be aborted by the start of a standard conversion. The following table summarizes the ADC behaviour in this situation for all possible combinations of conversion requests. Note that a conversion request as discussed in this context is activated when the respective control bit (ADST or ADCRQ) is toggled from ’0’ to ’1’, i.e. the bit must have been zero before being set.

Conversion in progress	**New requested conversion**	
	Standard	**Injected**
Standard	Abort running conversion and start requested new conversion	Complete running conversion, start requested conversion after that
Injected	Complete running conversion, start requested conversion after that	Complete running conversion, start requested conversion after that. Bit ADCRQ will be ’0’ for the second conversion.

Due to internal improvements, the internal timing of the A/D converter of the FA-/GA-steps is slightly different from previous versions, which is reflected in a different way of specifying the ADC. When ADCON.[15:12] = 0000b (default), the conversion time tc of the A/D converter of the FA-/GA-steps is identical to previous steps, while the sample time ts is increased by a factor of 1.33. For ADCON.[15:12] $\neq$ 0000b, tc and/or ts may be different from previous steps.

Since the FA-/GA-steps are produced in a different technology than previous steps, it is recommended to check the overall ADC accuracy in the target system with respect to the impedance of the analog signal and the analog reference voltage. This should be done in particular when the FA-/GA-steps are operated at a higher frequency than previous steps.

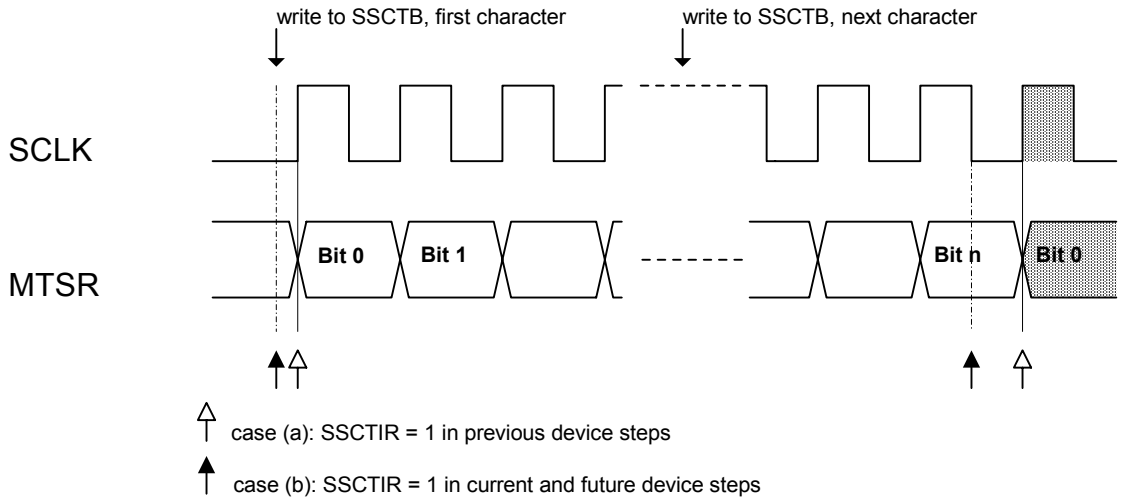
Timing of flag SSCTIR (SSC Transmit Interrupt Request)

In master mode, the timing of SSCTIR depends on the device step as follows:

(a) before step **FA**, flag SSCTIR is set to ‘1’ synchronous to the shift clock SCLK 1/2 bit time before the first latching edge (= first shifting clock edge when SSCPH = 0). When SSCTB is written while the shift register is empty, the maximum delay between the time SSCTB has been written and flag SSCTIR=1 is up to 1/2 bit time.

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

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



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

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

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

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

Product Engineering Group, Munich