SIEMENS

Microcomputer Components
Technical Support Group Munich
HL MCB AT 1

Errata Sheet

March 7, 1996 / Release 1.0

Device : SAB 88C166 - 5M

Stepping Code / Marking : ES-CA, ES-CA2, CA2, V59

The SAB 88C166 is a version of the SAB 80C166/83C166 with **32 Kbyte** on-chip **Flash** memory.

This errata sheet describes the functional problems known in this step. Problems which are also present in the basis SAB 80C166 have identical numbers in the respective errata sheets.

Note: devices which are marked as **ES-CA** or **ES-CA2** are engineering samples which may not be completely tested in all functional and electrical characteristics. They should be used for functional evaluation only.

Note: devices which are marked as **V59** are functionally identical to devices which are markes as **ES-CA2** or **CA2**. Note that some specific restrictions apply to these devices (see pages 15/16).

The SAB 88C166-5**M** is mounted in a 100-pin Plastic Metric Rectangular Flat Pack (P-MQFP-100) package.

Changes from Errata Sheet **Rel. 1.1** for devices with stepping code/marking **ES-CA** to this Errata Sheet **Rel. 1.0** for devices with stepping code/marking **ES-CA, ES-CA2, CA2, or V59**:

- specific restrictions for devices with date code 414 and/or stepping code/marking ES-CA2 or CA2 added (see pages 15/16)
- Stack Underflow Trap during Restart of interrupted Multiplication (problem 28)
- Bit Protection for register TFR (problem 27)
- PEC Transfers during instruction execution from internal RAM (problem 26)
- Warm Hardware Reset (problem 24)
- package: references to Bumpered Quad Flat Pack (P-BQFP-100) package eliminated

Functional Problems

The following malfunctions are known in this step:

Problem 17: Interrupted Multiplication in Combination with higher Priority Interrupt after RETI

When a multiply instruction has been interrupted, it may be completed incorrectly after return from interrupt if a higher priority interrupt or hardware trap is generated while the RETI instruction is executed. This problem does not occur with PEC transfers.

Workaround #1 (may be selected if **no divide operations** are used in an interruptable program section):

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

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

Workaround #2 (may be selected if **no class A or class B traps** can occur while the RETI instruction is processed):

Place a BCLR IEN or BFLDH PSW, #0F0h, #0F0h instruction before the RETI instruction in each interrupt service routine. This will cause an interrupted multiplication (or division) to be correctly completed after RETI before a higher priority interrupt will be acknowledged.

Note that an interrupt may be acknowledged after BCLR IEN or BFLDH PSW, #0F0h, #0F0h and before RETI. When BCLR IEN (size 1 word) is used, the interrupt routine which is entered in this case will not be interruptable. When BFLDH PSW, #0F0h, #0F0h (size 2 words) is used, the normal priority structure and therefore the real-time behaviour of the application is not affected.

On the C level, these instructions may be inserted via inline assembly with #pragma asm/#pragma endasm or by using the built-in function _bfld (PSW, 0x0F000, 0x0F000).

Workaround #3 (may be selected when a program is generated with C compilers):

Select the compiler option which prevents MUL/DIV instructions to be interrupted. Note that this compiler option does not protect MUL/DIV instructions from being interrupted by class A or Class B traps.

In the BSO Tasking C166 V2.1, this is the default (option -OM). For more details, please refer to the newest C166 documentation.

In the Siemens CC166 V2.0, this option is controlled via the variable __DISABLE in file gc.a66. For more details, refer to chapter 8.5 (Disabled interrupts during MUL/DIV instructions) in the CC166 manual.

Workaround #4 (may be selected when none of the previous workarounds can be used):

Execute all multiply operations on the class A trap level. This way, multiplications can not be interrupted, neither by interrupts nor by hardware traps. One possible implementation of this workaround is described in the following.

A class A trap can be forced by software by setting one of the trap flags NMI, STKOF, STKUF in register TFR. In case that all class A traps are already used in an application, an additional flag must be defined which indicates to the trap service routine whether a multiply operation or an exception trap is to be processed. If both signed and unsigned multiplication is used in an application, an additional flag must be defined which indicates what type of operation is to be performed.

Main Program:

```
BitSection  SECTION BIT

            Multi DBIT          ; flag for multiply operation
            GLOBAL Multi        ; may be used by different task procedures

BitSection  ENDS

MultiRB     REGDEF R3-R5 COMMON = Para, R0-R2 PRIVATE

CodeSection SECTION CODE
MainProc    PROC TASK INTNO = 0     ; task which is executed after reset

  ...
                                    ; example: MUL R4, R5
  BCLR      IEN                     ; disable interrupts
  NOP
  NOP
                                    ; parameter for trap routine
  SCXT      CP, MultiRB             ; passed via overlapping registerbanks
  BCLR      R3.0                    ; type indicator, 0 = unsigned, 1 = signed
  MOV       R4, multiplicand
  MOV       R5, multiplier
  BSET      STKUF                   ; set STKUF flag to force trap
  BSET      Multi                   ; flag multiply operation
  NOP
  POP       CP                      ; restore context
  BSET      IEN                     ; enable interrupt system
  ...
```

Note: When trap flags are set by software, 1 (at least) or 2 (at most) instructions following the instruction which set the trap flag may be executed before the trap is entered.

Workaround #4 (cont'd):

Trap Service Routine:

```
EXTERN     Multi:BIT

StackUnRB REGDEF R0-R2 COMMON = Para, R2-R3 PRIVATE

CodeSec    SECTION CODE
StackUnTrap PROC Task INTNO = 6        ; stack underflow task

     SCXT   CP, #StackUnRB             ; switch registerbank
     BCLR   STKUF                      ; clear trap flag
     JBC    Multi, SHORT MultiService  ; test and clear Multi flag

ExceptionTrapService:
     ...                               ; code for exception trap service
     ...                               ;
     JMP    Continue                   ; end of exception trap service

MultiService:
     JB     R0.0, SignedMult           ; test for type of multiplication

UnsignedMult:
     MULU   R1, R2
     JMPR cc_UC, Continue

SignedMult:
     MUL    R1, R2

Continue:
     POP    CP                         ; restore previous registerbank
     RETI                              ; return from trap service routine

StackUnTrap ENDP
```

This problem will be fixed in the next step.

Problem 19: Jump with Cache Hit after Branch from internal ROM/Flash

Note: This problem does NOT occur for the following configurations:

- for **ROMless** applications,
- for **Single Chip** applications where no external bus is used,
- for applications where the **internal ROM/Flash** is used and only data but no code is accessed over the external bus,
- for applications where the **internal ROM/Flash** is used and the external bus configuration is one of the following:
  - 16-bit multiplexed with zero waitstates,
  - 16-bit non-multiplexed with not more than one waitstate,
    where the term 'waitstate' is used as defined below.

For other configurations, this problem should be considered, especially when designing programs for a ROM mask or Flash, or when testing these programs with the Flash version or the emulator.

Problem Description:

When the internal ROM/Flash is enabled and external memory is accessed in one of the following bus configurations

- **8-bit multiplexed** or **8-bit non-multiplexed,**
- **16-bit multiplexed** with at least one waitstate,
- **16-bit non-multiplexed** with at least two waitstates,
  where the term 'waitstate' may refer to any of the following types or combinations of it:
  - Memory Cycle Time Waitstate,
  - programmed Memory Tristate Waitstate,
  - ALE Lengthening option,

a problem may occur when **all** of the following conditions are true:

1.) a jump which loads the jump target cache (possible for JMPR, JMPA, JB, JNB, JBC, JNBS) is taken from the internal ROM/Flash to external memory, **or** a branch (call/return/RETI) is performed from the internal ROM/Flash to external memory and a previous jump (JMPR, JMPA, JB, JNB, JBC, JNBS) taken within the internal ROM/Flash has loaded the jump target cache,

2.) the first instruction fetched from external memory is a JMPR instruction in a loop for which the branch condition is true, i.e. the cache is loaded with the two words at the target address of this jump,

3.) a PEC transfer (internal or external source or destination) is performed immediately after the JMPR instruction,

4.) in the flow of the program, the JMPR instruction is executed a second time and a cache hit occurs, i.e. the branch condition is also true after the second iteration through the loop, and no JMPS, CALLS, RETS, TRAP, RETI instruction or interrupt has been processed between the first and the second iteration through the loop.

In this case, the second word which was stored in the jump cache, and which was already processed as instruction or second word of an instruction, is again fetched via the external bus and interpreted as (first word of) an instruction. The target address of the next relative branch which will be taken in the following will be offset by 2 bytes.

Workaround #1 (only program analysis in external memory required):

Use JMPA instructions instead of JMPR instructions in all progamm sections which are located in external memory.

Workaround #2 (optimized):

Since only RETI instructions at the end of interrupt service routines may return to instructions which are not known in advance, substitute all RETI instructions in the internal ROM/Flash by jump instructions to a common RETI instruction in external memory. Or, since most assemblers and compilers will not allow to omit a RETI instruction at the end of an interrupt procedure, place the jump to an external RETI in front of the original RETI in the internal ROM/Flash:

Internal ROM/Flash:

```
        JMPA cc_UC, CommonRETI
        RETI                    ; never executed
or      RETV                    ; virtual return/BSO Tasking
```

External Memory:

```
CommonRETI:  RETI
```

In addition (although this constellation seems very unlikely), make sure that for all other non-sequential instructions (jump/jump on bit/call/return) which may branch from internal ROM/Flash to external memory the instruction at the target address is not a JMPR instruction in a loop.

Workaround #3 (only program analysis in internal ROM/Flash required):

Place a dummy call instruction in front of any instruction in the internal ROM which may branch to external memory. The destination of this call instruction must be a corresponding return instruction which is located in external memory.

This problem will be fixed in the next step.

Problem 20: Restart of A/D Converter

If a running A/D conversion is stopped through clearing the start bit ADST, and then a new conversion is initiated, it might occur under the following conditions that the conversion of the newly specified channel is omitted, and conversion of the next lower channel is performed. This can only happen within a small time window (2 TCL), if the new conversion is started exactly at the time point at which the previous conversion is terminated, and if the new channel number is > 0, and if the new conversion mode is either the single channel continuous, the auto scan, or the auto scan continuous mode.

Workaround #1:
Perform the restart of the A/D converter directly after the current conversion has finished, i.e. wait for the conversion complete interrupt request. In this way, the small time window will not be met (provided the restart instruction sequence is not interrupted).

Workaround #2:
Perform a double restart through first starting a dummy single channel conversion, and then restarting with the actual desired conversion mode. This could be done via the following instruction sequence:

```
BCLR ADST                ; reset the start bit
MOV  ADCON,#80h          ; start dummy single channel conversion
BCLR ADST                ; reset start bit again
MOV  ADCON,#NEW_MODE     ; start with the actual desired conversion
                         ; mode
```

This problem will be fixed in one of the next steps.

Problem 22: A/D Converter Overrun Error Generation

When the result of an A/D conversion has not been read from register ADDAT by the time the next conversion is complete (e.g. because reading of ADDAT was blocked by a higher priority interrupt), the following problem may occur for read operations from ADDAT within a time window of 6 TCL after completion of an A/D conversion:

A read operation which was triggered by the last ADC conversion complete interrupt (flag ADCIR) may already read the result of the just finished conversion, however the ADC overrun error interrupt (flag ADEIR) is **not** generated. Note that flag ADCIR is set again, so that another conversion complete interrupt or PEC transfer (which will deliver the same result) will be generated.

Workaround:

Assign an interrupt priority to the ADC conversion complete interrupt which is high enough to avoid an overrun error situation. Or, in the autoscan modes, check the channel number information in the 4 MSBs of the converted result to see whether the results of one channel are missing in the autoscan sequence.

This problem will be fixed in one of the next steps.

Problem 23: Serial Channel Overrun Error Generation

When the last character received has not been read from register SxRBUF by the time reception of the next character is complete (e.g. because reading of SxRBUF was blocked by a higher priority interrupt), the following problem may occur for read operations from SxRBUF within a time window of 4 TCL after reception of a character:

A read operation which was triggered by the last receive interrupt (flag SxRIR) may already read the next character received, however the overrun error flag (SxOE) is **not** set and the overrun interrupt (flag SxEIR) is **not** generated. Note that flag SxRIR is set again, so that another receive interrupt or PEC transfer (which will read the same character) will be generated.

Workaround:

Assign an interrupt priority to the receive interrupt which is high enough to avoid an overrun error situation.

This problem will be fixed in one of the next steps.

Problem 24: Warm HW Reset (Pulse Length < 1032 TCL)

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

Workaround:

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

This problem will be fixed in the next step.

Problem 26: PEC Transfers during instruction execution from Internal RAM

When a PEC transfer occurs after a jump with cache hit during instruction execution from internal RAM (locations 0FA00h - 0FDFFh), the instruction following the jump target instruction may not be (correctly) executed. This problem occurs when the following sequence of conditions is true:

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

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

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

iv) the jump target instruction is a double word instruction

Workaround 1:

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

Workaround 2:

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

This problem will be fixed in one of the next steps.

Problem 27: Bit Protection for register TFR

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

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

Workaround:

When the NMI trap is used **and** also other traps (which are not terminated with a software reset) may occur, connect the NMI# pin to a pin of the 88C166 which is capable of generating an interrupt request on a falling signal edge. In each trap routine, test the respective interrupt request flag xxIR after modifications of trap flags in register TFR have been performed, e.g. as follows:

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

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

This problem will be fixed in one of the next steps.

Problem 28: Stack Underflow Trap during Restart of interrupted Multiplication

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

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

Workaround 1:

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

Workaround 2 (may be selected if **no divide** operations are used in an interruptable program section):

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

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

Workaround 3 (may be selected when a program is generated with C compilers):

Select the compiler option which prevents MULx instructions to be interrupted.

In the BSO Tasking C166, the -BM option in combination with the 'protected' libraries (lib\p directory) should be used. Note that in this case
- both MUL and DIV will be protected against interrupts
- BCLR IEN is used to temporarily disable interrupts; however, a (low priority) interrupt may still be acknowledged before the protection is in effect. This may affect the real time behaviour of the system, since this interrupt routine is now uninterruptable.
- the NMI# (class A trap) may still interrupt a multiplication, so this method should not be used when the NMI# is used.

Workaround 4 (may be selected when none of the previous workarounds can be used):

a) if **no NMI#** is used:

```
SCXT PSW, #0Fxy0h      ; x = 8h if HLDEN = 0, x = 0Ch if HLDEN = 1
                       ; y = 0h if USR0 = 0, y = 4h if USR0 = 1
NOP
NOP
MULx Rm, Rn
POP PSW
```

Note: It is not recommended to use BCLR IEN to disable interrupts due to pipeline effects which may lead to an unexpected order of interrupt nesting (see also Application Note Interrupt System #2).

b) if **NMI#** is used: the multiply operation must be executed on the class A trap level. This may be achieved e.g. with the following sequence, where the stack overflow trap routine is shared with a multiply routine:

Main Program:

```
DataSection   SECTION BIT
              Atomic DBIT          ; flag for uninterruptable sequence
              GLOBAL Atomic        ; may be used by different task procedures
DataSection   ENDS

MainRB        REGDEF R2-R4 COMMON = Para, R0-R1 PRIVATE

CodeSection   SECTION CODE
MainProc      PROC TASK INTNO = 0      ; e.g. task which is executed after
reset
              ...
              MOV  R2, multiplicand    ; parameters for multiply routine
              MOV  R3, multiplier      ;
              MOV  R4, #SOF AtomicMUL  ; segment offset of procedure
                                       ; AtomicMUL required as
                                       ; parameter if both MUL and
                                       ; MULU are used
              BSET STKOF               ; set STKOF flag to force trap
              BSET Atomic              ; flag uninterruptable sequence
              ...
```

Note: When trap flags are set by software, 1 (at least) or 2 (at most) instructions following the instruction which set the trap flag may be executed before the trap is entered.

Trap Service Routine:

```
EXTERN      Atomic:BIT
StackOvRB   REGDEF R0-R2 COMMON = Para, R2-R3 PRIVATE

CodeSec     SECTION CODE
StackOvTrap PROC Task INTNO = 4          ; stack overflow task
            SCXT CP, #StackOvRB          ; switch registerbank
            BCLR STKOF                   ; clear trap flag
            JBC  Atomic, SHORT MultiplyService      ; test and clear
                                                    ; Atomic flag
ExceptionTrapService:
            ...                          ; code for exception trap service
            ...                          ;
            JMP  Continue                ; end of exception trap service

MultiplyService:
            CALLI cc_UC, [R2]            ; execute AtomicMUL or AtomicMULU,
                                         ; specified by R2

Continue:
            POP  CP                      ; restore previous registerbank
            RETI                         ; return from trap service routine
StackOvTrap ENDP
```

This problem will be fixed in one of the next steps.

Problem S1: Erroneous Byte Forwarding for internal RAM locations

When a **byte write** operation to the internal RAM is performed and a byte forwarding condition occurs, i.e the next instruction or the instruction after the next
a) **reads** the **same** internal RAM location as a **word**, or
b) **reads** the **complementary byte** within the same word,
the result of the corresponding word-based read operation could be wrong.

Note: Bit or **Bitfield** instructions also perform word-based read operations and must be considered accordingly in this context.

Note: Forwarding of results is only performed for operands in the internal RAM (0FA00h ... 0FDFFh), including the GPRs, but not for SFRs.

Examples:

```
I1:   MOVB  RL3, #data8     ; BYTE write to low byte of R3 in internal RAM
I2:   MOV   R1, R3          ; explicit WORD read of R3 (could be wrong)
I3:   BSET  R3.1            ; implicit WORD read of R3 (could be wrong)
```

```
I4:   MOVB   0FD01h, RH1        ; BYTE write to internal RAM location
I5:   BFLDL  0FD00h, #m8, #d8   ; implicit WORD read of same word location (could be wrong)
I6:   ADD    0FD00h, T0         ; explicit WORD read of same word location (could be wrong)

I7:   XORB   RL2, #data8        ; BYTE write to low byte of R2 in internal RAM
I8:   MOV    RH1, RH2           ; explicit read of high (complementary) byte of R2 (could be wrong)
```

Note that **no** relevant read operation is performed on the **destination** address of **MOV/MOVB** instructions, therefore no problems will be caused e.g. by the following instruction sequence:

```
I9:   XORB   RL2, #data8        ; BYTE write to low byte of R2 in internal RAM
IA:   MOVB   RH2, RH1           ; no read of high (complementary) byte of R2 (correct result)
IB:   MOV    R2,  R3            ; no read of same word location (correct result)
```

Workaround:

The next **two** instructions after an instruction writing a byte to an internal RAM location must not have word, bitword, bit or complementary byte operand **reads** at the same word address in the internal RAM. See also note 3 on page 10.

Note that **MOVBS** and **MOVBZ** instructions perform a **byte read** operation and a **word write** operation and will therefore not cause a byte forwarding problem:

```
MOVBZ   R3, mem      ; byte read, but word write
ADDB    RH0, RL3     ; byte read/write, correct operation
CMPB    RH0, #0FFh   ; byte read, correct operation
```

This problem will be fixed in the next step.

Problem S4: Flash memory read protection is not yet working

In this step, the Flash memory read protection can not yet be activated or enabled.

Workaround: None

This problem will be fixed in one of the next steps.

Functional Problem	Short Description	Fixed in Step
S4	Flash memory read protection is not yet working	
S3	Deviating internal Flash timer clock control bits coding	**ES-CA**
S2	Bootstrap Loader is not working	**ES1-BA**
S1	Erroneous Byte Forwarding for internal RAM locations	
28	Stack Underflow Trap during restart of interrupted Multiplication	
27	Bit Protection for register TFR	
26	PEC Transfers during instruction execution from internal RAM	
24	Warm Hardware Reset	
23	Serial Channel Overrun Error	
22	A/D Converter Overrun Error	
20	Restart of A/D Converter	
19	Jump with Cache Hit after branch from internal ROM/Flash	
17	Interrupted multiplication with interrupt after RETI	
15	Non-sequential instructions in 8-bit bus modes with ALE lengthening	**ES-CA**
14	External write after first word of double word instruction	**ES-CA**
13	Signed Divisions may produce erroneous results in case of Interruption	**ES-CA**
12	Non-interruptability of multiplication	**ES-CA**
10	Hardware overwrite of software modifications of T3OTL and T6OTL	**ES-CA**

Table 1: History of Functional Problems of the SAB 88C166

Specific Restrictions for SAB 88C166 Devices with
- Stepping Code/Marking ES-CA and Date Code ≥ 414 or
- Stepping Code/Marking ES-CA2, CA2, V59

1. Application: Bootstrap Loader/Flash Mapping to Code Segment 1

When pin BUSACT# = low and pin ALE is pulled high by external devices during a HW reset (RSTIN# = low), the following problems will occur:

1. The **Bootstrap Loader** is **not activated** after the end of reset when a falling edge on pin NMI# is generated. Instead, the NMI# trap is entered.

2. Program execution immediately starts over the external bus after the end of reset. When later on the internal Flash is enabled and mapped to code segment 1, it can be correctly programmed (written) and verified (read), but **the programmed code can not be executed correctly**. A branch to the internal Flash will result in unpredictable behaviour.

No problems will occur for applications where BUSACT# = high or ALE = low during a HW reset.

BUSACT#	ALE	88C166 ES-CA2/CA2/V59 and ES-CA with Date Code > 414	Standard (all other 8xC166)
L	L	external access after HW reset	external access after HW reset
L	H	external program execution NMI: BSL **not** activated no NMI: **no** internal SRST code execution from Flash **not** possible	NMI: BSL activated no NMI: internal SRST after approx. 22 ms @ 20 MHz, internal RAM overwritten then external execution
H	L	internal access after reset	internal access after reset
H	H	NMI: BSL activated no NMI: internal SRST after approx. 22 ms @ 20 MHz, internal RAM overwritten then internal execution	NMI: BSL activated no NMI: internal SRST after approx. 22 ms @ 20 MHz, internal RAM overwritten then internal execution

Workarounds in case BUSACT# = L and ALE = H in an application during hardware reset:

Workaround 1 (Bootstrap Loader operation required):

Tie pin BUSACT# to a logic high level, i.e. select the Single Chip Mode during a HW reset. After the boot sequence is terminated via a SRST instruction, program execution will start from address 0000h in the internal Flash in code segment 0. With an instruction sequence which is then executed e.g. from the internal RAM, the internal Flash may be mapped to code segment 1 and the external bus may be enabled.

Workaround 2 (no Bootstrap Loader operation required):

Insure that during a HW reset pin ALE is not pulled above VILmax = 0.2 Vcc - 0.1 V if no bootstrap loader invocation is intended. If necessary, either an external pull down or devices from a different logic family may be used on pin ALE.

Workaround 3 (no Bootstrap Loader operation required):

In case it is impossible to keep ALE below VILmax = 0.2 Vcc - 0.1 V during reset, but bootstrap loader operation is not intended, execute a SRST before executing code in the internal Flash in segment 1. In this case, a HW reset must be differentiated from a software reset. This can be done e.g. by using predefined test patterns which are compared with the contents of (internal or external) RAM locations which are typically undefined after power up. Depending on whether there are several sources for a hardware and/or software reset, several different patterns may have to be used. The basic principle is shown in the following:

```
   ...
   MOV     Rn, #const          ; load constant test pattern
   CMP     Rn, mem             ; test pattern equal to SRST indication pattern
                               ; in RAM ?
   JMPR    cc_EQ, Continue     ; YES: software reset has been performed
   MOV     mem, Rn             ; NO: store SRST indication pattern to RAM
   SRST                        ; perform software reset
Continue:
   ADD     mem, ONES           ; complement pattern before next HW reset
   ...
```

2. Programming Boards

Depending on the hardware revision of the board, it may happen that the devices can not be programmed successfully in some programming boards. In case of problems, please contact your local Siemens office or the manufacturer of the programming board.

Hints for Flash Programming and Erasing

1.) EBC1 pin reset state: The Flash programming voltage Vpp applied to the chip must not exceed the specified value (+13.5V). Otherwise, permanent damage could be caused to the chip or to parts of it . One negative effect of such an specification violation could be, for example, that the state internally latched from the EBC1 pin is always '0'. This meant that the chip could not be started in an 16-bit external bus mode any longer.

2.) VPP appliance to the device: Note that Vpp must not be applied to the device before Vcc, and Vcc must not be switched off from the device before Vpp.

3.) Programming the device with software causing problem S1 to occur: If it is inevitable to program the device by running software causing the byte forwarding problem to occur, the following measure could be tried. While using the fault causing software, select a non-multiplexed bus mode, lower the power supply voltage Vcc to +4.5 V and do eventually chose a system clock frequency below 16 MHz. Note that this suggestion is not a generally appropriate workaround for any application because other problems could occur for these ES-CA step engineering samples consequently.

Running the EVA166 monitor program on this step does cause problem S1 to occur for some operations. The byte forwarding problem can become evident, for example, if the fill memory command of the EVA166 monitor is executed. In the failure case, the memory will not be filled with the correct values. If the fill memory operation works properly, it can most likely be taken as an indication that the measure introduced above workarounds the byte forwarding problem for this application.

Appendix

In this appendix, some architectural items are described which are not yet documented in a very detailled manner in the current release of the SAB 80C166 User's Manual and/or Data Sheet. In the next revisions, these items will be integrated.

1.) Synchronous READY#: When the 'Synchronous Ready' mode has been selected for external memory accesses, for example by the following instruction sequence,

```
BCLR DP3.14    ;    Pin direction = INPUT
BCLR SYSCON.3  ;    Select 'Synchronous Ready' Mode
BSET RDYEN          ;    Enable READY function
```

this mode will only work properly if the system clock output pin CLKOUT is enabled in addition (P3.15 = 1, DP3.15 = 1, SYSCON.CLKEN = 1). The reason for this is that the system clock is normally required externally as synchronous reference signal.

2.) MDL Register: From the **CA-step** on, the MDL register may also be read via a short 'reg' addressing mode immediately after a divide instruction. This means that the note concerning the pipeline side effect in the Appendix of previous Errata Sheets must no longer be taken into account.

E.g., both of the following instruction sequences will always work correctly:

```
a)   DIV  Rx
     MOV  Ry, MDL   ;    'reg, mem' addressing mode

b)   DIV  Rx
     MOV  ext_mem, MDL;  'mem,reg' addressing mode
```

3.) Uninterruptable Instruction Sequences: To be absolutely sure that an instruction (sequence) can not be interrupted, at least 2 instructions (e.g. NOPs) must be programmed after the instruction disabling the interrupts, as shown in the following example (see also the application note *Interrupt System #1* in the SAB 80C166 Family ApNotes collection):

```
BCLR IEN
NOP            ;    or other appropriate instructions
NOP
...            ;    Start of the uninterruptable range
```

4.) JBC/JNBS Instructions: From the **CA-step** on, the operation of the semaphore instructions JBC/JNBS has been changed such that these instructions only write back to the bit to be tested when the branch condition is true. This modification has no effect on bits in the internal RAM or on SFR bits which are not modified by hardware. However, the use of these instructions on SFR bits which may be changed both by hardware and software is now directly possible without any special considerations (see also the application note on *JBC/JNBS* in the SAB 80C166 Family ApNotes collection).

The modified operation of the JBC/JNBS instructions is as follows:

JBC:

```
IF (bit) = 1 THEN
        (bit) := 0
        (IP) := target
ELSE
        next instruction
END IF
```

JNBS:

```
IF (bit) = 0 THEN
        (bit) := 1
        (IP) := target
ELSE
        next instruction
END IF
```

5.) P3.12 (BHE#) in Single Chip Mode: If in a single chip application pin P3.12/BHE# is intended to be used as a general purpose I/O pin, and not as the high byte enable signal for external memories, the following behaviour must be taken into account.

When the Single Chip mode is selected during reset (pin BUSACT# = 1, pins EBC0 = EBC1 = 0), bit BYTDIS in register SYSCON is automatically initialized to 0. This means that unlike all other port pins, P3.12 is not floating but is switched to **output** after reset. During reset, however, P3.12 is floating as all other port pins. After reset, P3.12 will output a **low** level as long as instructions, word operands, or byte operands on an odd address, are fetched from the internal ROM space, and it will show a high level when a byte operand is read from an even address. P3.12 will remain active as output until bit BYTDIS is set to '1' by software. Modifications of the direction bit DP3.12 or the port register bit P3.12 by software have no effect on the level at P3.12 after reset until BYTDIS = '1'.

Due to this reset behaviour, it is recommended to use P3.12 as a general purpose output pin with a default low level after reset. If P3.12 must be used as an input, external glue logic is required (e.g input buffer controlled by RSTOUT# signal) to avoid possible conflicts.