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 : **CC, ES-CC**

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-CC** are engineering samples which may not be completely tested in all functional and electrical characteristics. They should be used for functional evaluation only.

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. 1.2** for devices with stepping code/marking **ES-CC** to this Errata Sheet **Rel. 1.0** for devices with stepping code/marking **ES-CC** or **CC:**

- Stack Underflow Trap during Restart of interrupted Multiplication (problem 28)
- ADC test conditions: TUE ± 3 LSB for devices with date code > 6xx (Part B, problem 9)

Part A: Functional Problems

The following malfunctions are known in this step:

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 25: Bit Instructions on Interrupt Control Registers

An interrupt to vector address 100h may erroneously be generated by a bit instruction which writes to an interrupt control register xxIC under the following condition:

- the bit instruction explicitly writes to one or more bits of xxIC **except bit xxIE** (interrupt enable bit),
- xxIE = 0 by the time the bit instruction is executed,
- the request flag xxIR = 1 by the time the bit instruction writes to xxIC, or xxIR is set to 1 by the bit instruction itself.

Bit instructions in this context include BCLR/SET, BFLDL/H, JBC/JNBS, BMOV/N, BAND/OR/XOR.

The problem does **not** occur

- when xxIE = 1 by the time the bit instruction is executed (independent of status/changes of xxIR),
- or when xxIE = 0 and xxIR is explicitly cleared to 0 by the bit instruction.

Workarounds:

1. Use instructions with data type BYTE or WORD to access interrupt control registers xxIC.
2. Explicitly clear or set xxIE with the same bit instruction which accesses other bits in xxIC (e.g.: BFLDL xxIC, #0C0h, #80h: xxIR = 1, xxIE = 0).
3. Place a RETI instruction at address 100h. This way, the erroneous interrupt will occur, but an immediate return to normal program execution is performed.

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.

<u>Problem 28:</u> 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.

Functional Problem	Short Description	Fixed in Step
S4	Flash memory read protection is not yet working	**ES-CC**
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	**ES-CC**
28	Stack Underflow Trap during restart of interrupted Multiplication	
27	Bit Protection for register TFR	
26	PEC Transfers during instruction execution from internal RAM	
25	Bit Instructions on Interrupt Control Registers	
24	Warm Hardware Reset	
23	Serial Channel Overrun Error	**ES-CC**
22	A/D Converter Overrun Error	**ES-CC**
20	Restart of A/D Converter	**ES-CC**
19	Jump with Cache Hit after branch from internal ROM/Flash	**ES-CC**
17	Interrupted multiplication with interrupt after RETI	**ES-CC**
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: Functional Problems of the SAB 88C166

Part B: Deviations from AC/DC Specification

The following deviations of electrical and timing parameters from the specification are known in this step:

Problem 9: A/D Converter Offset Error

The offset error of the A/D converter may vary up to +/- 3 LSB within the operating temperature range of 0 to 70 °C. Therefore, the test conditions for the A/D converter had to be changed as follows:

TUE = +/- 4 LSB,	VAREF = Vcc = 5.0 V	devices with date code < 6xx
TUE = +/- 3 LSB,	**VAREF = Vcc = 5.0 V**	**devices with date code > 6xx**

Characterization results have shown that besides the offset error all other factors which contribute to the total unadjusted error (TUE) are within the original limits of +/- 2 LSB.

Problem 10: Supply Current Icc

The supply current Icc may be up to 180 mA.

Electrical/ Timing Problem	Short Description	Remarks
10	Supply Current Icc	
9	A/D Converter Offset Error	

Table 3: AC/DC Spec. Deviations of the SAB 88C166

Hints for Flash Programming and Erasing

1.) EBC1 pin reset state: The Flash programming voltage Vpp applied to the chip on pin EBC1/Vpp must not exceed the specified maximum value (+13.5V). Otherwise, permanent damage could be caused to the chip or to parts of it. This may result in improper recognition of the EBC1 reset state, i.e. an undesired bus configuration may be selected.

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

Functional Improvements/Compatibility Issues

In the following sections, the functional improvements which have been implemented in the 88C166 beginning with the **CC**-step are described, and compatibility issues related with these improvements are discussed. These improvements are already included in the **CC**-step of the bondout version 80C166E, and they will be included in the **DA**-step of the 80C166/83C166/W. In general, all of these improvements are hardware and software upward compatible.

1. Serial Channels ASCx: Start Bit detection (async mode)

Reception of data in the asynchronous mode of ASCx is initiated whenever a high to low transition (start bit) is detected at the respective RxD pin. When RxD stays at a low level during and after the bit time of the stop bit (BREAK situation), a receive interrupt and a framing error is generated, and no further data frames are received until the next high to low transition at pin RxD signals the start of a new data frame. This means that only one incorrect data frame is received at the beginning of such a BREAK situation.

In **previous steps**, reception was already initiated when a low level was detected at RxD. In case RxD stayed low during and after the stop bit time slot, a receive interrupt and a framing error was generated, but reception of further data frames proceeded, including generation of receive interrupts and framing errors, until RxD went to a high level (end of BREAK situation). This means that several incorrect data frames could be received during such a BREAK situation.

In systems where the RxD line correctly returns to a high level while no data are transmitted (i.e. no BREAK situations occur), there will be no differences between the current step and previous steps of the 88C166.

In systems where RxD may go to a low level during and after the stop bit time slot, the functionality improvement of the start bit detection in the current step may lead to simpler software implementations of detecting the end of such a BREAK situation.

2. Serial Channels ASCx: Receive data hold time (sync mode)

In the synchronous mode of ASCx, the timing for the receive data on the respective RxD line in relation to the shift clock which is output on the corresponding TxD line is as follows:

Input data setup time to shift clock rising edge:	4 TCL (100 ns @ 20 MHz)
Input data hold time after shift clock rising edge:	0.0 us

In **previous steps**, this timing was as follows:

Input data setup time to shift clock rising edge:	not relevant
Input data hold time after shift clock rising edge:	1/4 of shift clock cycle time

This functionality improvement is upward compatible, since a transmitter which is connected to the RxD line will normally change its data with the falling edge of the shift clock, such that the timing requirements of all steps of the 88C166 are met.