SIEMENS

Microcomputer Components
Technical Support Group Munich
HL MCB AT 1

Errata Sheet

October 31, 1996 / Release 1.6

Device :	**SAB 80C166W - M - T4,**
	SAB 83C166W - M - T4
Stepping Code / Marking :	**CB**

The SAB 80C166**W** is the version of the SAB 80C166 **without** oscillator prescaler.

This Errata Sheet describes the functional problems (see part A) and the deviations from the AC/DC specification (see part B) known in this step. Note that specific AC characteristics apply to the SAB 80C166W. For backward reference, a table which lists the problems of previous steps has been included.

From a specific date code on, the following restriction applies:

maximum CPU Clock: 18 MHz @ 50% duty cycle

The timing specifications of the SAB 80C166W Data Sheet have to be recalculated accordingly.

The SAB 80C166W-**M** devices are mounted in a Plastic Metric Rectangular Flat Pack (P-MQFP-100-2) package.

Changes from Errata Sheet **Rel. 1.5** to **Rel. 1.6:**

- Test Conditions for Minimum CPU Clock changed from 1 MHz to **3 MHz** Effective from Date Code **9642** (see Part B, deviations from specification)

Part A: Functional Problems

The following malfunctions are known in this 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 21: Result Forwarding by Bit Field Instructions BFLDL/BFLDH

When a (byte or word) forwarding situation occurs between a BFLDL/H instruction and the instruction immediately following BFLDL/H, i.e. the following instruction uses the result of the BFLDL/H instruction, then the forwarded value may be incorrect.

Example:

```
BFLDH  R0, #0Fh, #05h     ; R0 modified
MOV    R1, R0             ; R0 result forwarded, value incorrect
MOV    R2, R0             ; R0 read, value correct
```

Note: Forwarding of results by the BFLDL/H instruction is only performed for operands in the bit-addressable internal RAM area (0FD00h .. 0FDFFh) and for GPRs (R0 .. R15/RL0 .. RH7), not for SFRs.

Note: With the BSO/Tasking C166 compiler, critical forwarding situations with BFLDL/H instructions may only occur when a value which is different from '0' is assigned to a bitfield whose size is smaller than 8 bits.

Workaround:

Place a NOP or any other instruction which does not use the result of BFLDL/H immediately after the BFLDL/H instruction.

This problem will be fixed in the next step.

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.

<u>Problem 23</u>: 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.

<u>Problem 24:</u> 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 one of the next steps.

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.

<u>Problem 27:</u> 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.

No problems will ocuur when the NMI trap is the only trap used in the system, or when all other traps except the NMI are terminated with a software reset. 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 80C166 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.

Functional Problem	Short Description	Remarks
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 (pulse length < 1032 TCL)	
23	Serial Channel Overrun Error	
22	A/D Converter Overrun Error	
21	Result Forwarding by BFLDL/BFLDH	
20	Restart of A/D Converter	
19	Jump with Cache Hit after branch from internal ROM/Flash	

Table 1: Functional Problems of the SAB 80C166/80C166W

Functional Problem	**Short Description**	**Fixed in Step**
18	Watchdog Timer reset in Idle Mode	**CB** [2)]
17	Interrupted multiplication with interrupt after RETI	**CB**
16	WR# signal	**CA** [1)]
15	Non-sequential instructions in 8-bit bus modes with ALE lengthening	**CA** [1)]
14	External write after first word of double word instruction	**CA**
13	Signed Divisions may produce erroneous results in case of interruption	**BB**
12	Non-interruptability of multiplication **and division**	**CA**
11	ROM mapping and PEC transfers	**CA**
10	Hardware overwrite of software modifications of T3OTL and T6OTL	**CA**
9	Concatenation of timer T6 and the CAPCOM timers T0 and T1	**BA**
8	Incorrect register update by SCXT instruction with SP as operand	**BA**
7	Incorrect multiply or divide results during hold states	**BA**
6	External pre-fetch error when executing a short 'Jump-to-Itself'	**BA**
5	Double external read operations	**BA**
4	Ports 0, 1 and 4 may not be correctly initialized to '0'	**BA**
3	Divisions or multiplications may produce erroneous results in case of interruption	**AB**
2	Erroneous values can be loaded into the MDL and MDH registers	**AB**
1	Power Down Mode deviation	**BA**

Notes: 1) fixed in all devices with stepping code/marking CA except for CA-ES1
2) fixed in all devices with stepping code/marking CB except for CB-ES

Table 2: History of Fixed Problems of the SAB 80C166/80C166W

Part B: Deviations from AC/DC Specification

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

Testcondition for Minimum CPU Clock

The Test Conditions for Minimum CPU Clock have been changed from 1 MHz to **3 MHz**. Therefore, the maximum oscillator period is 333 ns.

This change is effective from Date Code **9642** on.

Electrical/ Timing Problem	**Short Description**	**Remarks**
8	AC Parameters t14/t15 and t16	**see Data Sheet** 2)
7	Conditions for VAREF/VAGND	**see Data Sheet** 2)
6	Maximum Storage Temperature T_{ST}	**fixed** 1)
5	Input Leakage Current I_{OZ}	**fixed** 3)
4	ALE Rising to CLKOUT Falling Edge Time, t_{34}	**see Data Sheet** 2)
3	Address Hold after RD# or WR# Time, t_{28}	**fixed in step CA**
2	Power Down Mode Supply Current, I_{PD}	**fixed in step BB**
1	Idle Mode Supply Current, I_{ID}	**see Data Sheet** 2)

Notes: 1) Problem only relevant for devices with stepping code/marking CA
2) integrated in Data Sheet from Release 2.94 on
3) temporary problem due to test equipment

Table 3: History of AC/DC Spec. Deviations of the SAB 80C166/80C166W

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.

6.) Power Down Mode: When driving the XTAL1 input with an external clock source, XTAL1 must be held at V_{CC} to V_{CC}-0.1V in order to meet the I_{PD} specification of 50 µA. When P2.15 is high, the specified value for I_{PD} will be reached faster than when P2.15 is low.

Functional Improvements/Compatibility Issues

In the following sections, the functional improvements which will be implemented in the 80/83C166/W beginning with the **DA**-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 the flash version 88C166/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 the **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 80C166.

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 80C166 are met.