The C167CR-16RM is the 128 Kbyte ROM version of the C167CR, including an on-chip CAN module, 2 Kbyte XRAM module, and a PLL oscillator circuit.

This errata sheet describes the functional problems known in this step. Problem classification and numbering is performed relative to modules, where the C167 AC-step is the reference. Since most problems of earlier steps have already been fixed in this step of the C167CR, problem numbering is not necessarily consecutive.

The C167CR-16RM devices are mounted in a 144-pin Plastic Metric Quad Flat Pack (P-MQFP-144-1) package.

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

Changes from Errata Sheet Rel. 1.0 to this Errata Sheet Rel. 1.1:
- Execution of PWRDN Instruction while pin NMI# = high (PWRDN.1)
- Arithmetic Overflow by DIVLU instruction (CPU.17)
- Data read access with MOV [Rn], mem instruction to internal ROM (CPU.16)
- Access to internal ROM with EXTS/EXTSR instructions (CPU.12)
Functional Problems

The following malfunctions are known in this step:

**ADC.10: Start of Standard Conversion at End of Injected Conversion**

When an A/D conversion in any of the standard modes (single channel, auto scan, continuous modes) is started by software within a time window of 2 TCL (50 ns @ 20 MHz) before the end of an injected conversion, the following problem will occur:
- the result of the injected conversion is not transferred to ADDAT2, and interrupt request flag ADEIR is not set
- no further requests for injected conversions are processed
- the standard conversion is not started (i.e. the A/D converter is blocked)

**Workaround 1:**

When using the continuous conversion modes, enable channel injection after conversion in the continuous mode has been started. Due to problem ADC.8, this should be done before the first capture/compare event on CC31 occurs.

**Workaround 2:**

Do not start a standard conversion while an injected conversion is in progress. In this case, start the standard conversion in the ADEINT interrupt service routine which is invoked after the injected conversion is finished. As an indication whether an injected conversion is in progress, bits ADCRQ or ADBSY may be tested.

- Main Program: ...

  ATOMIC #3 ; see Note 1)
  BMOV ADStart, ADCRQ ; copy current status of ADCRQ
  JB ADStart, Done ; injected conversion in progress?
  BSET ADST ; no: start standard conversion here,
  ; see Note 2)
Done: ... ; yes: start standard conversion in ADEINT ISR

- ADEINT Interrupt Service Routine:

  ... 
  EXTR #1
  MOV ICResult, ADDAT2 ; read ADDAT2 to avoid wait-for-read situation
  JNB ADStart, Done2 ; start of standard conversion required?
  ATOMIC #2 ; yes, see Note 1)
  JB ADCRQ, Done2 ; - another injected conversion in progress?
  ; if yes: start standard conversion in next
  ; invocation of ADEINT ISR, see Note 3)
  ; no other injected conversion in progress:
  BSET ADST ; start standard conversion here, see Note 2)
  BCLR ADStart ; clear flag
Done2: ... 
  RETI
Note 1): ATOMIC should be used here to ensure correct flag handling and avoid side effects which may be caused by interrupts.

Note 2): In case ADCRQ has been set by a channel injection request from CC31 in the time between the second half of the decode phase and the second half of the execute phase of instruction BSET ADST, flag ADCRQ may be unintentionally cleared by instruction BCLR ADST due to missing hardware bit protection of bit ADCRQ. This does not effect the actual start and processing of an injected conversion, since an internal (not user accessible) latch controls its correct operation. However, bit ADCRQ does not reflect the correct status for this particular injected conversion.

Note 3): This extra test of ADCRQ may be omitted if it can be guaranteed that the processing of the ADEINT interrupt service routine which was triggered by the last injected conversion is not delayed that long by higher priority tasks that already the next injected conversion is finished when the standard conversion shall be started in the ADEINT routine.

Compatibility with previous steps of the C167CR

In previous steps (e.g. C167CR-LM AB-step), an injected conversion in progress was aborted when a standard conversion was started by software (see C167 User's Manual V1.0 or V2.0, p. 16-8, third Note). If this effect was not desired, it was suggested to check whether no injected conversion was in progress before starting a standard conversion. This means that all systems which have considered this proposal implicitly already have implemented the workaround described above and will work without problems also with the C167CR-16RM AA-step.

In the C167CR-16RM AA-step, as a correction of problem ADC.7 (Channel Injection request coincident with start of standard conversion), start of standard conversions will no longer abort injected conversions. If this 'new' feature is used, the above software workaround must be used.

If the abortion of an injected conversion by the start of a standard conversion was tolerated in systems with previous steps (e.g. C167CR-LM AB-step), the workaround must be implemented when switching to the C167CR-16RM AA-step.
**CPU.9: 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 0F600h - 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

*Note: No problem will occur during instruction execution from the internal XRAM (locations 0E000h - 0E7FFh).*

**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.

---

**CPU.12: Access to internal ROM with EXTS/EXTSR instructions**

EXTS instructions (EXTS, EXTSR) do not work correctly for data accesses to internal ROM when the 2 msbs of the intra-segment address are different from the 2 lsbs of the DPP which is selected by these 2 msbs of the intra-segment address. In this case, the 2 msbs of the intra-segmented address are substituted by the 2 lsbs of the selected DPP.

Example:

```
EXTS #1, #1 ; ROM access to be performed in segment 1
MOV R0, 8000h ; intra-segment address to be accessed = 8000h
              ; 2 msbs = 10b select DPP2
```

In case DPP2 contains 000h, location 1: 0000h is accessed instead of 1:8000h.

**Workarounds:**

Use e.g. EXTP/EXTPR instructions instead of EXTS/EXTSR.

On C level, workarounds depend on the memory model and data types.

For the KEIL compiler, the following specific workaround is suggested:

a) do not use the memory models HLARGE or HCOMPACT and do not use the memory types huge or xhuge, only in this combinations the EXTS/EXTSR instruction is generated.
b) when memory types huge or xhuge are required, use the C166xxx.LIB run-time libraries instead of the C167xxx.LIB run-time library files and do not compile with the MOD167 directive. ESFRs may be referenced with addresses casted to pointers, e.g. #define DP0L (*((unsigned int volatile sdata*) 0xF100))

**CPU.16: Data read access with MOVB [Rn], mem instruction to internal ROM**

When the MOVB [Rn], mem instruction (opcode 0A4h) is executed, where
1. mem specifies a direct 16-bit byte operand address in the internal ROM/Flash memory,
   AND
2. [Rn] points to an **even** byte address, while the contents of the word which includes the byte addressed by mem is **odd**,  
   OR
   [Rn] points to an **odd** byte address, while the contents of the word which includes the byte addressed by mem is **even**

the following problem occurs:

a) when [Rn] points to **external** memory or to the **X-Peripheral** (XRAM, CAN) address space, the data value which is written back is always 00h
b) when [Rn] points to the **internal** RAM or SFR/ESFR address space,
   - the (correct) data value [mem] is written to [Rn]+1, i.e. to the **odd** byte address of the selected word in case [Rn] points to an **even** byte address,  
   - the (correct) data value [mem] is written to [Rn]-1, i.e. to the **even** byte address of the selected word in case [Rn] points to an **odd** byte address.

**Workaround:**

When mem is an address in internal ROM, substitute instruction

\[
\text{MOV}B \ [Rn], \ mem \quad \text{e.g. by} \quad \text{MOV} \ Rm, \ #\text{mem} \\
\text{MOV}B[Rn], \ [Rm]
\]

**Note:** the Keil C166 Compiler V3.10 has been extended by the directive FIXROM which avoids accesses to 'const' objectes via the instruction MOVB [Rn], mem.
**CPU.17: Arithmetic Overflow by DIVLU instruction**

For specific combinations of the values of the dividend (MDH,MDL) and divisor (Rn), the Overflow (V) flag in the PSW may not be set for unsigned divide operations, although an overflow occurred.

E.g.:

```
MDH  MDL  Rn  MDH  MDL
F0F0 0F0Fh : F0F0h  = FFFF FFFFh, but no Overflow indicated !
```

(result with 32-bit precision: 1 0000h)

The same malfunction appears for the following combinations:

- n0n0 0n0n : n0n0
- n00n 0nn0 : n00n
- n000 000n : n000
- n0nn 0nnn : n0nn

where n means any Hex Digit between 8 ... F

i.e. all operand combinations where at least the most significat bit of the dividend (MDH) and the divisor (Rn) is set.

In the cases where an overflow occurred after DIVLU, but the V flag is not set, the result in MDL is equal to FFFFh.

**Workaround:**

Skip execution of DIVLU in case an overflow would occur, and explicitly set V = 1.

E.g.:

```
CMP Rn, MDH
JMPR cc_ugt, NoOverflow ; no overflow if Rn > MDH
BSET V ; set V = 1 if overflow would occur
JMPR cc_uc, NoDivide ; and skip DIVLU
```

```
NoOverflow: DIVLU Rn
NoDivide: ... ; next instruction, may evaluate correct V flag
```

**Note:**

- the KEIL C compiler, run time libraries and operating system RTX166 do not generate or use instruction sequences where the V flag in the PSW is tested after a DIVLU instruction.

- with the TASKING C166 compiler, for the following intrinsic functions code is generated which uses the overflow flag for minimizing or maximizing the function result after a division with a DIVLU:

  ```
  _div_u32u16_u16()
  _div_s32u16_s16()
  _div_s32u16_s32()
  ```

Consequently, an incorrect overflow flag (when clear instead of set) might affect the result of one of the above intrinsic functions but only in a situation where no correct result could be calculated anyway. These intrinsics first appeared in version 5.1r1 of the toolchain.

Libraries: not affected

---

**PWRDN.1: Execution of PWRDN Instruction while pin NMI# = high**

Errata Sheet C167CR-16RM, ES-AA, AA, 1.1
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 preceeds 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.

**ROM.1:** Internal ROM access to locations 28000h ... 2FFFFh

When accesses to internal ROM locations 28000h ... 2FFFFh are performed, these accesses are mapped to locations 20000h ... 27FFFh. This means that actually only 96 Kbyte of internal ROM are available.

**RST.1:** System Configuration via P0L.0 during Software/Watchdog Timer Reset

Unlike P0L.5 .. P0L.1, **P0L.0 is not disregarded during software or watchdog timer reset**. This means that when P0L.0 is (erroneously) externally pulled low at the end of the internal software or watchdog timer reset sequence, the device will enter emulation mode.

Therefore, ensure that the level at P0L.0 is above the minimum input high voltage $V_{IH_{min}} = 0.2 \ V_{cc} + 0.9 \ V$ ($1.9 \ V @ \ V_{cc} = 5.0 \ V$) at the end of the internal reset sequence.
Table 1: Functional Problems of the C167CR-16RM

<table>
<thead>
<tr>
<th>Functional Problem</th>
<th>Short Description</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC.10</td>
<td>Start of Standard Conversion at end of Injected Conversion</td>
<td></td>
</tr>
<tr>
<td>CPU.9</td>
<td>PEC Transfers during instruction execution from Internal RAM</td>
<td></td>
</tr>
<tr>
<td>CPU.12</td>
<td>Access to internal ROM with EXTS/EXTSR instructions</td>
<td></td>
</tr>
<tr>
<td>CPU.16</td>
<td>Data read access with MOVB [Rn], mem instruction to internal ROM</td>
<td></td>
</tr>
<tr>
<td>CPU.17</td>
<td>Arithmetic Overflow by DIVL Instructions</td>
<td></td>
</tr>
<tr>
<td>PWRDN.1</td>
<td>Execution of PWRDN Instruction while pin NMI# = high</td>
<td></td>
</tr>
<tr>
<td>ROM.1</td>
<td>Internal ROM access to locations 28000h ... 2FFFFh</td>
<td></td>
</tr>
<tr>
<td>RST.1</td>
<td>System Configuration via P0L.0 during Software/Watchdog Timer Reset</td>
<td></td>
</tr>
</tbody>
</table>

Specific Problems with X-Peripherals (XPERs)

The following problems with the interface to XPERs, the CAN module, and the XRAM module are currently known:

**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.

Table 2: Functional Problems with XPERs on the C167CR-16RM

| Functional Problem | Short Description | fixed in Step |"
Deviations from DC/AC Specification

The following table lists the deviations of the DC/AC characteristics from the specification in the C167CR Data Sheet 6.95.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>Max. CPU Clock $= 20$ MHz</th>
<th>Variable $1/2TCL = 1$ to $20$ MHz</th>
<th>CPU Clock $= 1$ to $20$ MHz</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>RD# to valid data in (no RW-delay)</td>
<td>t15</td>
<td>-</td>
<td>$50+tc$ instead of $55+tc$</td>
<td>-</td>
<td>$3TCL-25+tc$ instead of $3TCL-20+tc$</td>
</tr>
<tr>
<td>ALE low to valid data in</td>
<td>t16</td>
<td>-</td>
<td>$50+tc$ instead of $55+tc$</td>
<td>-</td>
<td>$3TCL-25+tc$ instead of $3TCL-20+tc$</td>
</tr>
<tr>
<td>CLKOUT rising edge to ALE falling edge</td>
<td>t34</td>
<td>$0+ta$</td>
<td>$12+ta$ instead of $10+ta$</td>
<td>$0+ta$</td>
<td>$12+ta$ instead of $10+ta$</td>
</tr>
<tr>
<td>ALE falling edge to CS#</td>
<td>t38</td>
<td>$-10-ta$ instead of $-5-ta$</td>
<td>$10-ta$</td>
<td>$-10-ta$ instead of $-5-ta$</td>
<td>$10-ta$</td>
</tr>
</tbody>
</table>

Table 3: Deviations from DC/AC Specification of the C167CR-16RM

Notes:

1) Pin READY# has an internal pullup (all C167xx derivatives). This will be documented in the next revision of the Data Sheet.

2) Timing $t28$: Parameter description and test changed from 'Address hold after RD#/WR#' to 'Address hold after WR#'. It is guaranteed by design that read data are internally latched by the controller before the address changes.

3) During reset, the internal pullups on P6.[4:0] are active, independent whether the respective pins are used for CS# function after reset or not.

In addition to the description in the C167 Derivatives User's Manual V2.0, the following feature enhancement has been implemented in the C167CR-16RM:

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 $QCAP = 1$ in register $T5CON.10$

Compatibility with previous versions:

In previous versions (e.g. C167CR-LM), both of the settings ($TxM = 110b, T5CON.10 = 1$) were reserved and should not be used. Therefore, systems designed for previous versions will also work without problems with the C167CR-16RM.