Errata Sheet

V1.0, 2003-11-07

Device	**C161S-L25M, -LM3V**
Marking/Step	**(E)ES-AA, AA**
Package	**P-MQFP-80**

This Errata Sheet describes the deviations from the current user documentation.

The module oriented classification and numbering system uses an ascending sequence over several derivatives, including already solved deviations. So gaps inside this enumeration can occur.

Current Documentation

- C161S Data Sheet, V1.0, Nov. 2003
- C161S User's Manual - currently not available, see documentation update
- Instruction Set Manual, V2.0, Mar. 2001

Note: Devices marked with EES- or ES are engineering samples which may not be completely tested in all functional and electrical characteristics, therefore they should be used for evaluation only.

The specific test conditions for EES and ES are documented in a separate Status Sheet.

Contents

1 History List/Change Summary

(from step AA, no previous errata sheet)

Table 1 Functional Deviations

Functional Problem	Short Description	Fixed in step	Change
RTC.4	Undefined Results after write to T14 (C161S-L25M version only)		
BUS.17	Spikes on CS Lines after access with $\overline{\text{RDCS}}$ and/or $\overline{\text{WRCS}}$		
BUS.18	PEC Transfers after JMPR instruction		
CPU.17	Arithmetic Overflow by DIVLU instruction		
CPU.21	BFLDL/BFLDH Instructions after Write Operation to internal IRAM		
CPU.22	Z Flag after PUSH and PCALL		
PLL.3.x	Increased PLL Jitter caused by external Access		
PWRDN.1	Execution of PWRDN instruction while Pin $\overline{\text{NMI}}$ = high		
POWER.1	Internal Reset generation when PLL is switched off		

Table 2 AC/DC Deviations

AC/DC Deviations	Short Description	Fixed in step	Change
	no deviations found		

Table 3 Application Hints

Hint	Short Description	Change
SSC.H1	Handling of the SSC Busy Flag (SSCBSY)	
SSC.H2	Timing of flag SSCTIR (SSC Transmit Interrupt Request)	
PLL.H1	PLL holds Reset Mode while missing clock	
MainOsc.H1	Oscillator Type_LP2: Negative Resistance and Start-up Reliability	
MainOsc.H2	Maximum (Type_LP2) Oscillator Frequency = 16 MHz	
OWD.H2	Oscillator Watchdog and Prescaler Mode	
ISNC.H1	Maintenance of ISNC Register	

Table 4 Documentation Update

Update	Short Description	Change
DOC_C161S.D1	Differences to C161PI User's Manual	

2 Functional Problems

RTC.4 Undefined Results after write to T14

A write operation to T14 leads to undefined results depending on contents of T14REL.

Workaround:

Load T14REL (1st) and T14 (2nd) with the desired value.

E.g.

```
MOV   T14REL, #xxxx
MOV   T14, #xxxx
```

Note: The problem does not exist in C161S-LM3V version - "Reduced digital supply voltage" (see Data Sheet "4.2 Operating Conditions").

BUS.17 Spikes on $\overline{\text{CS}}$ Lines after access with $\overline{\text{RDCS}}$ and/or $\overline{\text{WRCS}}$

Spikes of about 5 ns width (measured at VOH = 0.9 V_{CC}) from V_{CC} to V_{SS} may occur on Port 6 lines configured as $\overline{\text{CS}}$ signals. The spikes occur on one $\overline{\text{CS}}$x line at a time for the first external bus access which is performed via a specific BUSCONx/ADDRSELx register pair (x=1..4) or via BUSCON0 (x=0) when the following two conditions are met:

1. the previous bus cycle was performed in a **non-multiplexed** bus mode **without tristate** waitstate via a different BUSCONy/ADDRSELy register pair (y=1..4, y≠x) or BUSCON0 (y=0, y≠x) **and**
2. the previous bus cycle was a read cycle with $\overline{\text{RDCS}}$y (bit BUSCONy.CSRENy = 1) or a write cycle with $\overline{\text{WRCS}}$ (bit BUSCONy.CSWENy = 1).

The position of the spikes is at the beginning of the new bus cycle which is performed via $\overline{\text{CS}}$x, synchronous with the rising edge of ALE and synchronous with the rising edge of RD#/WR# of the previous bus cycle.

Potential effects on applications:

- when $\overline{\text{CS}}$ lines are used as $\overline{\text{CE}}$ signals for external memories, typically no problems are expected, since the spikes occur after the rising edge of the $\overline{\text{RD}}$ or $\overline{\text{WR}}$ signal.

- when $\overline{\text{CS}}$ lines configured as $\overline{\text{RDCS}}$ and/or $\overline{\text{WRCS}}$ are used e.g. as $\overline{\text{OE}}$ signals for external devices or as clock input for shift registers, problems may occur (temporary bus contention for read cycles, unexpected shift operations, etc.). When $\overline{\text{CS}}$ lines configured as $\overline{\text{WRCS}}$ are used as $\overline{\text{WE}}$ signals for external devices, no problems are expected, since

a tristate waitstate should be used anyway due to the negative address hold time after $\overline{\text{WRCS}}$ (t55) without tristate WS.

Workarounds:

1. Use a memory tristate WS (i.e. leave bit BUSCONy.5 = 0) in all active BUSCON registers where $\overline{\text{RD}}/\overline{\text{WR}}$-$\overline{\text{CS}}$ is used (i.e. bit BUSCONy.CSRENy = 1 and/or bit BUSCONy.CSWENy = 1), **or**
2. Use Address-$\overline{\text{CS}}$ instead of $\overline{\text{RD}}/\overline{\text{WR}}$-$\overline{\text{CS}}$ (i.e. leave bits BUSCONy[15:14] = 00b) for all BUSCONy registers where a non-multiplexed bus without tristate WS is configured (i.e. bit BUSCONy.5 = 1).

BUS.18 PEC Transfers after JMPR instruction

Problems may occur when a PEC transfer immediately follows a taken JMPR instruction when the following sequence of 4 conditions is met (labels refer to following examples):

1. in an instruction sequence which represents a loop, a jump instruction (Label_B) which is capable of loading the jump cache (JMPR, JMPA, JB/JNB/JBC/JNBS) is taken
2. the target of this jump instruction directly is a JMPR instruction (Label_C) which is also taken and whose target is at address A (Label_A)
3. a PEC transfer occurs immediately after this JMPR instruction (Label_C)
4. in the following program flow, the JMPR instruction (Label_C) is taken a second time, and no other JMPR, JMPA, JB/JNB/JBC/JNBS or instruction which has branched to a different code segment (JMPS/CALLS) or interrupt has been processed in the meantime (i.e. the condition for a jump cache hit for the JMPR instruction (Label_C) is true)

In this case, when the JMPR instruction (Label_C) is taken for the second time (as described in condition 4 above), and the 2 words stored in the jump cache (word address A and A+2) have been processed, the word at address A+2 is erroneously fetched and executed instead of the word at address A+4.

Note: the problem does not occur when

- the jump instruction (Label_C) is a JMPA instruction
- the program sequence is executed from internal Flash/ROM/OTP

Example1:

```
Label_A:  instruction x                ; Begin of Loop
          instruction x+1
.....
Label_B:  JMP Label_C; JMP may be any of the following jump instructions:
```

```
            ; JMPR cc_zz, JMPA cc_zz, JB/JNB/JBC/JNBS
            ; jump must be taken in loop iteration n
            ; jump must not be taken in loop iteration n+1
.....
Label_C:  JMPR cc_xx, Label_A     ; End of Loop
            ; instruction must be JMPR (single word instruction)
            ; jump must be taken in loop iteration n and n+1
            ; PEC transfer must occur in loop iteration n
```

Example2:

```
Label_A:  instruction x               ; Begin of Loop1
          instruction x+1
.....
Label_C:  JMPR cc_xx, Label_A         ; End of Loop1, Begin of Loop2
          ; instruction must be JMPR (single word instruction)
          ; jump not taken in loop iteration n-1, i.e. Loop2 is entered
          ; jump must be taken in loop iteration n and n+1
          ; PEC transfer must occur in loop iteration n
.....
Label_B:  JMP Label_C                 ; End of Loop2
          ; JMP may be any of the following jump instructions:
          ;   JMPR cc_zz, JMPA cc_zz, JB/JNB/JBC/JNBS
          ; jump taken in loop iteration n-1
```

A code sequence with the basic structure of Example1 was generated e.g. by a compiler for comparison of double words (long variables).

Workarounds:

- use a JMPA instruction instead of a JMPR instruction when this instruction can be the direct target of a preceding JMPR, JMPA, JB/JNB/JBC/JNBS instruction, **or**
- insert another instruction (e.g. NOP) as branch target when a JMPR instruction would be the direct target of a preceding JMPR, JMPA, JB/JNB/JBC/JNBS instruction, **or**
- change the loop structure such that instead of jumping from Label_B to Label_C and then to Label_A, the jump from Label_B directly goes to Label_A.

Notes on compilers:

In the **Hightec** compiler beginning with version Gcc 2.7.2.1 for SAB C16x - V3.1 Rel. 1.1, patchlevel 5, a switch -m bus18 is implemented as workaround for this problem. In addition, optimization has to be set at least to level 1 with -u1.

The **Keil** C compiler versions < V4.0 and run time libraries do not generate or use instruction sequences where a JMPR instruction can be the target of another jump instruction, i.e. the conditions for this problem do not occur.

In V4.0, the problem may occur when optimize (size or speed, 7) is selected. Lower optimization levels than 7 are not affected.

In V4.01, a new directive FIXPEC is implemented which avoids this problem.

In the **TASKING** C166 Software Development Tools, the code sequence related to problem BUS.18 can be generated in Assembly. The problem can also be reproduced in C-language by using a particular sequence of GOTOs.

With V6.0r3, TASKING tested all the Libraries, C-startup code and the extensive set of internal test-suite sources and the BUS.18 related code sequence appeared to be NOT GENERATED.

To prevent introduction of this erroneous code sequence, the TASKING Assembler V6.0r3 has been extended with the CHECKBUS18 control which generates a WARNING in the case the described code sequence appears. When called from within EDE, the Assembler control CHECKBUS18 is automatically 'activated'.

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.

Table 5 Example

MDH	MDL	:	Rn	=	MDH MDL	Notes
$F0F0_H$	$0F0F_H$	:	$F0F0_H$	=	$FFFF\ FFFF_H$	but no Overflow indicated ! (result with 32-bit precision: 1 0000h)
The same malfunction appears for the following combinations:						
$n0n0_H$	$0n0n_H$	:	$n0n0_H$	=	$FFFF\ FFFF_H$	
$n00n_H$	$0nn0_H$	:	$n00n_H$	=	$FFFF\ FFFF_H$	
$n000_H$	$000n_H$	:	$n000_H$	=	$FFFF\ FFFF_H$	
$n0nn_H$	$0nnn_H$	:	$n0nn_H$	=	$FFFF\ FFFF_H$	n means any Hex Digit between 8 ... F

i.e. all operand combinations where at least the most significant 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
```

Notes on compilers:

- 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

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

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

1. the previous 'instruction' is a PEC transfer which writes to IRAM, or any instruction with result write back to IRAM (addresses $00'F200_H$ ... $00'FDFF_H$ for 3 Kbyte module, $00'F600_H$ ... $00'FDFF_H$ for 2 Kbyte module, or $00'FA00_H$ ... $00'FDFF_H$ for 1 Kbyte

module). For further restrictions on the destination address see case (a) or case (b) below.

2. the BFLDx instruction immediately follows the previous instruction or PEC transfer within the instruction pipeline ('back-to-back' execution), i.e. decode phase of BFLDx and execute phase of the previous instruction or PEC transfer coincide. This situation typically occurs during program execution from internal program memory (ROM/OTP/Flash), or when the instruction queue is full during program execution from external memory
3. the 3rd byte of BFLDx (= ***#mask8*** field of BFLD**L** or ***#data8*** field of BFLD**H**) and the destination address of the previous instruction or PEC transfer match in the following way:
   a. value of #mask8 of BFLDL or #data8 of BFLDH = 0F**y**$_H$ (**y** = 0..F$_H$),
      and the previous instruction or PEC writes to (the low and/or high byte of) GPR R**y** or the memory address of R**y** (determined by the context pointer CP) via any addressing mode.
   b. value of #mask8 of BFLDL or #data8 of BFLDH = **00$_H$ ... 0EF$_H$**,
      and the lower byte **v**$_L$ of the **contents v** of the IRAM location or (E)SFR or GPR which is read by BFLDx is **00$_H$** ≤ **v**$_L$ ≤ **7F$_H$**
      and the previous instruction or PEC transfer writes to the (low and/or high byte of) the specific bit-addressable IRAM location **00'FD00$_H$ + 2 v**$_L$
      (i.e. the 8-bit offset address of the location in the bit-addressable IRAM area (00'FD00$_H$ ... 00'FDFF$_H$) equals **v**$_L$)

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

Notes:

Write operations in the sense of the problem description include implicit write accesses caused by

- auto-increment operations of the PEC source or destination pointers (which are located on 00'FCE0$_H$ ... 00'FCFE$_H$ in IRAM)
- post-increment/pre-decrement operations on GPRs (addressing modes with [R+] or [-R])
- write operations on the system stack (which is located in IRAM).

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

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

For implicit IRAM write operations caused by **auto-increment operations of the PEC source or destination pointers**, the problem can only occur if the value of #mask8 of BFLDL or #data8 of BFLDH = $0Fy_H$ (**y** = $0..F_H$), and the range which is covered by the context pointer CP (partially or completely) overlaps the PEC source and destination pointer area ($00'FCE0_H$ ... $00'FCFE_H$), and the address of the source or destination pointer which is auto-incremented after the PEC transfer is equal to the address of GPR R**y** (included in case 3a).

For **system stack write operations**, the problem can only occur if the system stack is located in the bit-addressable portion of IRAM ($00'FD00_H$ ... $00'FDFF_H$), or if the system stack can overlap the register bank area (i.e. the register bank area is located below the system stack, and the distance between the contents of the context pointer CP and the stack pointer SP is $\leq 20_H$).

Workaround 1:

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

- an equivalent sequence of single bit instructions. This sequence may be included in an uninteruptable ATOMIC or EXTEND sequence to ensure completion after a defined time.
- an equivalent byte- or word MOV or logical instruction.

Note that byte operations to SFRs always clear the non-addressed complementary byte.

Note that protected bits in SFRs are overwritten by MOV or logical instructions.

Workaround 2:

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

Workaround 3:

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

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

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

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

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

Table 6 Range of original ATOMIC/EXTEND statement

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

- - : case can not occur

Note on devices with power management and register RSTCON:

for unlock sequences, which are sequences of four instructions operating on ESFRs SYSCON1/2/3 and RSTCON and which are included in an EXTR #4 sequence, **this workaround must not be implemented when the 4th instruction is a BFLDx instruction**, otherwise the unlock sequence will not work correctly (in fact unlock sequences do not require a workaround).

Tool Support

The **Keil** C166 Compiler V3.xx generates BFLD instructions only in the following cases:

- when using the _bfld_ intrinsic function.
- at the beginning of interrupt service routines, when using #pragma disable
- at the end of interrupt service routines, when using the chip bypass directive FIX166.

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

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

The RTX166 Full Real-Time Operating system (any version) does not use BFLD instructions.

For RTX166 Tiny, you should rebuild the RTX166 Tiny library with the SET FIXBFLD = 1 directive. This directive is enabled in the assembler source file RTX166T.A66. After change of this setting rebuild the RTX166 Tiny library that you are using in your application.

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

The scan tool **aiScan21** analyzes files in hex format plus user-supplied additional information (locator map file, configuration file), checks whether they may be affected by problem CPU.21, and produces diagnostic information about potentially critical instruction sequences. This tool is included in AP1628 'Scanning for Problem CPU.21' on: www.infineon.com/cgi/ecrm.dll/ecrm/scripts/prod_cat.jsp?oid=-8137

AP1628 Description: www.infineon.com/cgi/ecrm.dll/ecrm/scripts/public_download.jsp?oid=18484&parent_oid=-8137
AP1628 Software: www.infineon.com/cgi/ecrm.dll/ecrm/scripts/public_download.jsp?oid=18468&parent_oid=-8137

CPU.22 Z Flag after PUSH and PCALL

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

a) for ***PUSH reg*** instructions:

- the contents of the high byte of the GPR or (E)SFR which is pushed is 00_H, and

- the contents of the low byte of the GPR or (E)SFR which is pushed is > 00_H, and
- the contents of GPR Rx is odd, where x = 4 msbs of the 8-bit 'reg' address of the pushed GPR or (E)SFR

Examples:

```
PUSH  R1   ;(coding: F1 EC):
           ;  incorrect setting of Z flag if contents of R15 is odd,
           ;  and 00FFh ≥ contents of R1 ≥ 0001h
PUSH  DPP3 ;(coding: 03 EC):
           ;  incorrect setting of Z flag if contents of R0 is odd,
           ;  and 00FFh ≥ contents of DPP3 ≥ 0001h
```

b) for ***PCALL reg, rel*** instructions:

- when the contents of the high byte of the GPR or SFR which is pushed is 00_H, and
- when the contents of the low byte of the GPR or SFR which is pushed is odd

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

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

Example:

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

Effect on Tools:

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

- The code generated by the **Keil** C166 Compiler evaluates the Z flag only after MOV, CMP, arithmetic, or logical instructions. It is never evaluated after a PUSH instruction.

PCALL instructions are not generated by the C166 Compiler.

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

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

- The **TASKING** compiler V7.5r2 never generates a PCALL instruction, nor is it used in the libraries.

  The PUSH instruction is only used in the entry of an interrupt frame, and sometimes on exit of normal functions. The zero flag is not a parameter or return value, so this does not give any problems.

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

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

Workaround (for program parts written in assembler)**:**

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

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

or

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

Hints for Detection of Critical Instruction Combinations

Whether or not an instruction following PUSH reg or PCALL reg, rel actually causes a problem depends on the program context.

In most cases, it will be sufficient to just analyze the instruction following PUSH or PCALL. In case of PCALL, this is the instruction at the call target address.

Support Tool for Analysis of Hex Files

For complex software projects, where a large number of assembler source (or list) files would have to be analyzed, Infineon provides a tool aiScan22 which scans hex files for critical instruction sequences and outputs diagnostic information. This tool is available as part of the Application Note ap1679 'Scanning for Problem CPU.22' on the 16-bit microcontroller internet pages of Infineon Technologies:

www.infineon.com/cgi/ecrm.dll/ecrm/scripts/prod_cat.jsp?oid=-8137

direct links:

AP1679 Description: www.infineon.com/cmc_upload/documents/040/841/ap1679_v1.1_2002_05_scanning_cpu22.pdf and
AP1679 Software: www.infineon.com/cgi/ecrm.dll/ecrm/scripts/public_download.jsp?oid=40840&parent_oid=-8137

Individual Analysis of Assembler Source Code

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

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

  These instructions may only cause a problem if the PSW is a source or source/destination operand:
  ADD/B, ADDC/B, CMP/B, CMPD1/2, CMPI1/2, SUB/B, SUBC/B
  AND/B, OR/B, XOR/B
  ASHR
  MOV/B, MOVBZ/MOVBS
  SCXT
  PUSH, PCALL ➔ analysis must be repeated for successor of PUSH/PCALL

- The following instructions (most of them with immediate or register (Rx) addressing modes) can never cause a problem:
  CPL/B, NEG/B
  DIV/U, DIVL/U, MUL/U

SHL/SHR, ROL/ROR
PRIOR
POP
RETI ➔ updates complete PSW with stacked value
RETP ➔ updates condition flags
PWRDN ➔ program restarts after reset
SRST ➔ program restarts

- Conditional branch instructions which may evaluate the Z flag:
  JB/JNB Z, rel ; directly evaluates Z flag
  CALLA/CALLI, JMPA/JMPI/JMPR with the following condition codes
  cc_Z, cc_EQ, cc_NZ, cc_NE
  cc_ULE, cc_UGT, cc_SLE, cc_SGT
  cc_NET
  ➔ For these branch conditions, the branch may be performed in the wrong way.
  ➔ For other branch conditions, the branch target as well as the linear successor of the branch instruction must be analyzed (since these branch instruction don't modify the PSW flags).

- For **instructions that have no effect on the condition flags** and that don't evaluate the Z flag, the instruction that follows this instruction must be analyzed.
  These instructions are:
  NOP
  ATOMIC, EXTxx
  DISWDT, EINIT, IDLE, SRVWDT
  CALLR, CALLS, JMPS ➔ branch target must be analyzed
  RET, RETS ➔ return target must be analyzed
  (value pushed by PUSH/PCALL = return IP, Z flag contains information whether intra-segment target address = 0000_H or not)
  TRAP ➔ both trap target and linear successor must be analyzed, since Z flag may be incorrect in PSW on stack as well as in PSW at entry of trap routine

- For **bit modification instructions**, the problem may only occur if a source bit is the Z flag, and/or the destination bit is in the PSW, but not the Z flag.
  These instructions are:
  BMOV/BMOVN
  BAND/BOR/BXOR
  BCMP
  BFLDH
  BFLDL ➔ problem only if bit 3 of @@ mask = 0, i.e. if Z is not selected
  BCLR/BSET ➔ problem only if operand is not Z flag
  JBC/JNBS ➔ wrong branch if operand is Z flag

PLL.3.x Increased PLL Jitter caused by external Access

When external bus mode is used and the CPU frequency is in the range from 10 to 15 MHz the PLL jitter is higher than specified.

Details are under evaluation.

PWRDN.1 Execution of PWRDN Instruction while pin $\overline{\text{NMI}}$ = high

When instruction PWRDN is executed while pin $\overline{\text{NMI}}$ 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 preceeding the PWRDN instruction writes to external memory or an XPeripheral (e.g. CAN or XRAM), 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 $\overline{\text{NMI}}$ 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 muliplexed bus with memory tristate waitstate is used, the PWRDN instruction should be executed out of internal RAM.

POWER.1 Internal Reset generation when PLL is switched off

Unintended internal reset may be generated under the following conditions:

- PLL mode is selected (reset configuration P0H.7-5 is set to f_{XTAL} * x; x = 1.5, 2, 2.5, 3, 4 or 5) and
- The PLL is switched off in power reduction mode (SDD) and
- Slow down divider clock is used (SYSCON2).

Corresponding SYSCON2 setting:

SYSCON2 = xxxx xx10 xxxx $xxxx_B$ (SDD enabled, PLL **off**)

Workaround:

No workaround available for the described mode.

3 Deviations from Electrical- and Timing Specification

No deviations found.

4 Application Hints

SSC.H1 Handling of the SSC Busy Flag (SSCBSY)

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

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

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

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

                                    ;time before SSCBSY is cleared)
```

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

SSC.H2 Timing of flag SSCTIR (SSC Transmit Interrupt Request)

In master mode, the timing of SSCTIR is as follows:

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

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

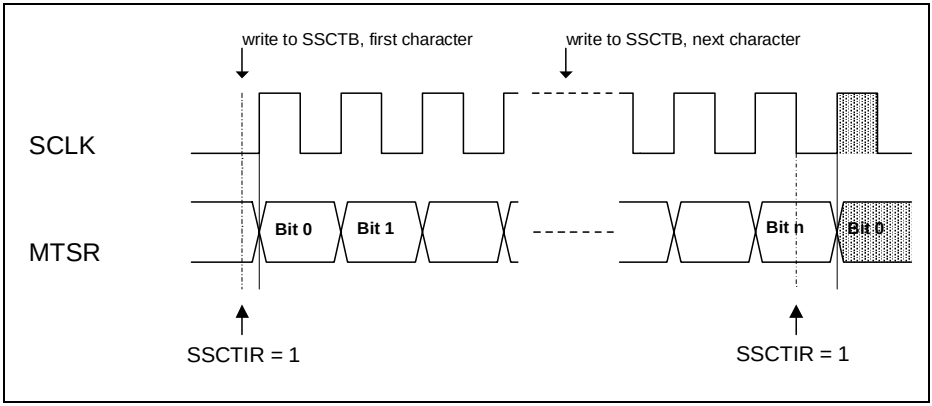

Figure 1 SSCTB Timing

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

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

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

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

PLL.H1 PLL holds Reset Mode while missing Clock

In case of a missing clock signal at input XTAL1 during reset in PLL modes (reset configuration P0H.7-5 is set to $f_{CPU} = f_{XTAL} * x$; x = 1.5, 2, 2.5, 3, 4 or 5) the PLL circuit holds the device in reset mode instead of providing a PLL base frequency clock signal.

MainOsc.H1 Main Oscillator Type_LP2: Negative Resistance and Start-up Reliability

Compared to other C16x microcontrollers the gain of the on-chip oscillator (Type_LP2) is slightly different. It is recommended to check the negative resistance and the start-up reliability of the oscillator circuit in the original application. Please refer to the limits specified by the quartz crystal or ceramic resonator supplier.

See also Application Note AP2420 'Crystal Oscillator of the C500 and C166 Microcontroller Families' and Application Note AP2424 'Ceramic Resonator Oscillators of the C500 and C166 Microcontroller Families'.
www.infineon.com/cgi/ecrm.dll/ecrm/scripts/prod_cat.jsp?oid=-8137

AP2420: www.infineon.com/cgi/ecrm.dll/ecrm/scripts/public_download.jsp?oid=9746&parent_oid=-8137
AP2424: www.infineon.com/cgi/ecrm.dll/ecrm/scripts/public_download.jsp?oid=9745&parent_oid=-8137

MainOsc.H2 Maximum Oscillator Frequency = 16 MHz (Main: Type_LP2)

The main oscillator is optimized for oscillation with a crystal within a frequency range of 4...16 MHz. When driven by an external clock signal it will accept the specified frequency range (see Data Sheet, AC Characteristics, tables 'Clock Generation Modes' and 'External Clock Drive Characteristics'). Operation at lower input frequencies is possible but is guaranteed by design only (not 100% tested) (see Data Sheet, AC Characteristics, table 'External Clock Drive Characteristics').

ISNC.H1 Maintenance of ISNC register

The RTC and PLL interrupts share one interrupt node (XP3IC). If an interrupt request occurs the request bit in the Interrupt Subnode Control register has to be checked and cleared by software. To avoid a collision with the next hardware interrupt request of same source it is recommended to clear the request and the enable bit first and then to set the enable bit again.

Example for an XP3 interrupt service routine (for Tasking C compiler):

```
...
if (PLLIR)
{
  _bfld (ISNC, 0x000C, 0x0000);  // clear PLLIE and PLLIR
  _putbit (1, ISNC, 3);          // set PLLIE
  ...                            // further actions concerning PLL/OWD
}
if (RTCIR)
{
  _bfld (ISNC, 0x0003, 0x0000);  // clear RTCIE and RTCIR
  _putbit (1, ISNC, 1);          // set RTCIE
  ...                            // further actions concerning RTC
}
...
```

Example for an XP3 interrupt service routine (in assembly language):

```
...
  EXTR   #1
  JNB    PLLIR, no_pll_request
  EXTR   #2                    ; no further interruption of this
                               ;   sequence possible
  BFLDL  ISNC, #0Ch, #00h      ; clear PLLIE and PLLIR
  BSET   PLLIE                 ; set PLLIE
...                            ; further actions concerning PLL/OWD
no_pll_request:
  EXTR   #1
  JNB    RTCIR, no_rtc_request
  EXTR   #2  ; no further interruption of this sequence possible
  BFLDL  ISNC, #03h, #00h      ; clear RTCIE and RTCIR
  BSET   RTCIE                 ; set RTCIE
...                            ; further actions concerning RTC
no_rtc_request:
...
```

OWD.H2 Oscillator Watchdog and Prescaler Mode

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

In direct drive mode the PLL base frequency is used directly (f_{CPU} = 2...5 MHz).

In prescaler mode the PLL base frequency is divided by 2 (f_{CPU} = 1...2.5 MHz).

5 Documentation Update

DOC_C161S.D1 Differences to C161PI User's Manual

Since no dedicated C161S User's Manual is available we recommend to use the C161PI User's Manual V1.0 1999-08 instead. The following table describes the differences of the C161S functionality to this document.

Table 7 Documentation Update

Function	Short Description	C161PI User's Manual V1.0 1999-08	
		affected section	page
Features	see C161S Data Sheet "1 Summary of Features"	1.2 Summary of Basic Features	1-4
I^2C Bus Module	this module is not implemented	• 2 Architectural Overview • 17 The I2C Bus Module • 5.1 Interrupt System Structure • 21 The Register Set	2-1 ff 17-1 ff 5.3 21-1 ff
ADC	this module is not implemented	• 2.3 The On-chip Peripheral Blocks • 2.5 Protected Bits • 5.1 Interrupt System Structure • 16 The Analog / Digital Converter • 21 The Register Set	2-11 ff 2-20 5-3 16-1 ff 21-1 ff
Port and dedicated Pins	different package and pin assignment - see Data Sheet "2.2 Pin Configuration and Definition"	• 2.3 The On-chip Peripheral Blocks • 7 Parallel Ports • 8 Dedicated Pins • 9.4 READY Controlled Bus Cycles • 23 Device Specification	2-11 ff 7-1 ff 8-1 ff 9-17 23-1 ff

Table 7 Documentation Update

Function	Short Description	C161PI User's Manual V1.0 1999-08	
		affected section	**page**
Missing Pins	The following pins and functions are **not** implemented:		
	• $\overline{\text{READY}}$, V_{AREF}, V_{AGND}, V_{DD}, V_{SS} (2 pins each)	• 8 Dedicated Pins	8-1 ff
	• P2.8/EX0IN	• 7.6 Port 2	7-16 ff
	• P3.0, P3.1, P3.15/CLKOUT	• 7.7 Port 3	7-19 ff
	• P4.6/A22	• 7.8 Port 4	7-24 ff
	• P5.0 ... P5.3 (A/D converter)	• 7.9 Port 5	7-27 ff
	• P6.4 ... P6.7 (I^2C Bus interface)	• 7.10 Port 6	7-30 ff
Internal memory	• 2 Kbytes of IRAM • XRAM is not implemented	3 Memory Organization	3-1 ff
Sleep Mode	this mode is not implemented	• 19.2 Sleep Mode • External Interrupts During Sleep Mode • 21 The Register Set	19-5 ff 5-28 21-1 ff
Frequency Output Signal	The CLKOUT pin and therefore the programmable Frequency Output Signal (and register FOCON) is not implemented	19.6 Programmable Frequency Output Signal	19-16 ff
XPER-SHARE	Bit XPER-SHARE in register SYSCON is not available (no XBUS peripheral and no HOLD function implemented)	• 4.5 CPU Special Function Registers • 9.5 Controlling the External Bus Controller	4-14 9-19
IDCHIP	IDCHIP of the C161S is 0513_H	Table 21-3 and 21-4 C161PI Registers	21-5 21-9

Product Engineering Group, Munich