Concerned Products:	**Customer Notification**	Date: Feb 21th 01
		NEC-Electronics (Europe) GmbH EAD –Technical Product Support
μPD703100 ***μPD703101*** ***μPD703102*** ***μPD70F3102*** ***μPD703100A*** ***μPD703101A*** ***μPD703102A*** ***μPD70F3102A***	**Bug Report**	Source Doc: HW 140598 HW 120898 SUD-T-4087-4 SUD-T-4115-2 SUD-T-3385-5 SBG-T-1659 SBG-T-1845
		Author: H.-W. Hoeft
Date of initial issue: Jun 4th 98		Doc. No.: TPS-HE-B-4105

(A) BUG LIST

Bug No.		Outline	µPD70F3102					µPD703100			µPD703101/ µPD703102		
							M						
			ES 4.0	ES 5.0	ES 5.1	ES 6.0	(7.2)	ES 1.0	ES 2.0	ES 3.0	ES 1.0	ES 2.0	ES 3.0
1		iRAM fetch by branch fram external memory to iRAM	✋	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
2		Read for register PCS3	✋	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
3		Read of port register at output port	✋	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
4		DMA function bug	💣	💣	✋	✓	✓	💣	✋	✓	💣	✋	✓
5		Standby current	✓	✓	✓	✓	✓	✋	✓	✓	✋	✓	✓
6		Forced shut-down during DMA transfer	💣	💣	✋	✓	✓	✋	✓	✓	✋	✓	✓
7		External bus lock with DMA cycle	💣	💣	✋	✓	✓	💣	✋	✓	💣	✋	✓
8		Multiplication instruction	💣	💣	✋	✓	✓	💣	✋	✓	💣	✋	✓
9		A/D converter function	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
10		PMx and PMCx read	✓	✓	✓	✓	✓	💣	💣	💣	💣	💣	💣
11		Interrupt on sld	💣	💣	💣	💣	💣	💣	💣	💣	💣	💣	💣

✓: No problem

✋: Bug (will be corrected by next version upgrade)

💣: Bug (restriction, not corrected by version upgrade)

Bug No.		Outline	µPD703100A			µPD703101A /µPD703102 A			µPD70F3102A			
												M
			ES 1.0			ES 1.0			ES 1.0	ES 2.0	ES 3.0	(4.1)
1		iRAM fetch by branch fram external memory to iRAM	✓			✓			✓	✓	✓	✓
2		Read for register PCS3	✓			✓			✓	✓	✓	✓
3		Read of port register at output port	✓			✓			✓	✓	✓	✓
4		DMA function bug	✓			✓			💣	✋	✓	✓
5		Standby current	✓			✓			✓	✓	✓	✓
6		Forced shut-down during DMA transfer	✓			✓			✋	✓	✓	✓
7		External bus lock with DMA cycle	✓			✓			💣	✋	✓	✓
8		Multiplication instruction	✓			✓			💣	✋	✓	✓
9		A/D converter function	✓			✓			✋	✓	✓	✓
10		PMx and PMCx read	💣			💣			✓	✓	✓	✓
11		Repeated execution of sld instruction	💣			💣			💣	💣	💣	💣

✓: No problem

✋: Bug (will be corrected by next version upgrade)

💣: Bug (restriction, not corrected by version upgrade)

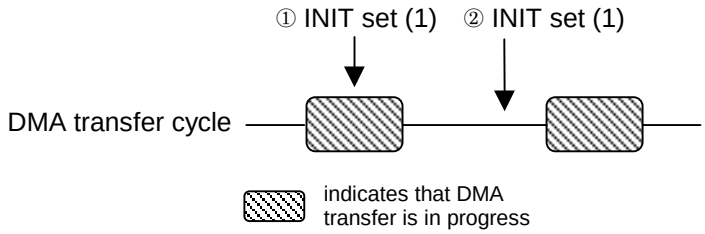
V850E/MS1 ROMLESS	µPD703100
V850E/MS1 Mask ROM 96k	µPD703101
V850E/MS1 Mask ROM 128k	µPD703101
V850E/MS1 Flash	µPD70F3102
V850E/MS1 ROMLESS(3,3V)	µPD703100A
V850E/MS1 Mask ROM 96k (3,3V)	µPD703101A
V850E/MS1 Mask ROM 96k(3,3V)	µPD703102A
V850E/MS1 Flash (3,3V)	µPD70F3102A

(B) BUG DESCRIPTION

Bug No.	Outline	Description
1	iRAM fetch by branch from external memory to iRAM	Details When a branch from external memory to iRAM was accessed, the next instruction fetch from iRAM doesn't work correct. Workaround Use an unconditional branch (jp instruction) to enable a branch to the next instruction within the iRAM.
2	Read of register PCS3	Details When bit 5 (PCS35) of port/control selection register 3 (PCS3) is read, always value "0" is read regardless of the existing conditions. WRITE is working correctly and the operation mode selection of the external interrupt/CSI2, is in line with the status of register PCS3. Workaround No available
3	Read of the port register at output port	Details When the output port mode is selected, READ of port register yields status of the port pin as its value, instead of port register value. concerned port registers; P0, P1, P2, P4, P5, P6, P8, P9, P10, P11, P12, PA, PB, PX Workaround No available

4	DMA function bug	In some cases, DMAAKn signal is kept active until the next DMA transfer. This occurs, when DMA transfer is interrupted with an NMI input and all of the following conditions are met. (n=0-3) There is no other effect than for DMAAKn under the same condition. (conditions) – 2 cycles transfer – block transfer – transfer from external memory (or external I/O) to external memory (or external I/O) – on-page access at no-wait when destination is EDO DRAM Workaround When the above mentioned phenomenon appears, the following routine (for example within the handler of NMI interrupt) will recover the original status (inactive) of DMAAKn signal. (n=0-3) ld.b DDIS[0], reg ; check DMA channel which is disconnected by NMI st.b reg, DRST[r0] ; resume transfer through the disconnected channel st.b r0, DRST[r0] ; interrupt again Note This limitation will remain in future as well and treated as special remark for the usage of this device. This phenomenon may happen especially in the application where DMA transfer is triggered by DMARQ signal and DMARQ signal is reset by falling signal of DMAAK. However, the above countermeasure to execute the routine is effective to evade the problem in any cases.
5	Standby Current	Details Using the standby mode (IDLE/STOP), a supply current of about 5mA flows. (direct mode or PLL mode) Workaround No available
6	Forced shut-down during DMA transfer	Details Normal operation: If the registers ① are re-written during DMA transfer, the new information is validated by issuing a terminal count or setting the DCHCn register INIT bit to 1. Abnormal operation: It is impossible to reset the registers ①, even if the INIT bit is set to 1. (Only in a case of single transfer mode) This problem invariably occurs during transfer from internal I/O to internal RAM. This problem does not occur, if the INIT bit set (1) operation overlaps with the DMA transfer cycle when transfer is performed between two external memory areas (or external I/O). (See figure.) ① registers DMA source address register (DSAnH, DSAnl) DMA destination address register (DDAHnH, DDAHnL) DMA byte count register (DBCn) (n=0-3)

Transfer between two external memory areas (or external I/O)



Problem does not occur when INIT bit is set to 1 with timing ①

Problem occurs when INIT bit is set to 1 with timing ②

Countermeasure

Because this problem only occurs in single transfer mode, desired operation is possible by re-initialising once you have switched to single transfer mode.

After forced DMA shut-down by setting the INIT bit, set an address that does not negatively influence the system and start single step transfer mode to that address with the soft-trigger. Even if DMA transfer to the applicable address is executed in the DMA related peripheral I/O register. With the instruction that immediately follows (executed when the CPU uses the bus interchangeable with DMA), reset the INIT-bit. When this is done normal initialization can be performed.

```
DMA_INIT    mov     0x04,r28
            st.b    r28,DCHCn       - Sets INIT bit and prohibits DMA
                                      transfer.

            movea   0xFFFFFFFF,r0,r28
            movea   0xFFFFFFFF,r0,r29
            st.h    r28,DSAnH       - Sets the source and destination
                                      addresses to reserved area of
                                      the internal peripheral I/O.

            st.h    r29,DSAnL
            st.h    r28,DDAnL
            st.h    r29,DDAnL

            movea   0x00A4,r0,r28   - change to single step transfer.
            st.h    r28,DADCn
            mov     0x03,r28
            st.b    r28,DCHCn       - Restarts dummy DMA transfer
                                      with soft-trigger.

            mov     0x04,r28
            st.b    r28,DCHCn       - Resets INIT bit and prohibits
                                      DMA transfer.
```

7	External bus lock with DMA cycle	Details This bug appears, during a conflict of an instruction fetch request from external memory and DMA request that use the external bus. This instruction fetch is created during a misalign-access to external memory or when SET1, NOT1 and CLR1 instruction are executed. The external bus cycle will lock and CPU instruction execution will stop. However, internal peripheral I/O itself will continue operating. Conditions Any of the following conditions will render this problem non applicable whether the DMA function is used or not. – Instruction fetch is not performed to external memory (the instruction code exists in the internal memory only). – DMA transfer is performed between internal RAM and internal I/O only. – There is no misalign-access and SET1, NOT1 and CLR1 instructions are not used. A number of bus-cycles occur when misalign-access is performed to external-memory, instruction fetch cycles occur between these bus cycles as a result of pre-fetch queue status. Similarly, an instruction fetch occurs between the read cycle and write cycle because a bit operation instruction induces a read modify write operation. The external bus locks due to a conflict between the fetch request for this instruction fetch and the DMA request that uses the external bus. Currently, the instruction fetch request that causes the lock is cleared, if an interrupt request occurs while the external bus is locked and the bus conflict status is eliminated. Workaround Do not use SET1, NOT1 and CLR1 instructions or instructions accompanying misalign-access to the external bus when a program is stored in external memory and DMA transfer is performed using external memory.
8	Multiplication instruction	Details After a multiplication command is executed, a problem occurs when commands that contain a write-back instruction (writing back to a register) follow. Before the result of the first multiplication command is stored in a register, the next command is fetched. If the execution of this command is cancelled by an interrupt, then the first multiplication will be stored in the second multiplication command's destination register. This applies to programs that repeatedly execute multiplication instructions (mul, mulh, mulhi, mulu). The result of the multiplication process is not written back to the first multiplication command's destination register, and the previous data remains in the register. This problem does not occur: - if the instruction not accompanying write back operation. - if the following instructions are enclosed: Division instructions (DIVH, DIV, DIVU), MOVimm32, TRAP, PREPARE, DISPOSE. Although the instructions used the write back operations, they do not result in problem conditions. - if the destination register of instructions accompanying a write-back operation is r0.

<table>
<tr><td></td><td></td><td>● Instructions accompanying write-back operations that result in problem conditions.
Load instructions, arithmetic operation, logic operation, saturation operation, LDSR, STSR, JARL
(Refer to the chapter on pipelining in the V850E architecture manual about the different instruction type's and write-back behaviour)

Note: This problem exerts no actual influence in the cases given below.
- When the destination registers of the preceding and following multiplication instructions are the same.
- When the result of the preceding multiplication instruction is not used after the following multiplication instruction is executed.

<u>Example</u>

mulh r0, r2
mov r0, r5
mov r0, r6
mulh r0, r3 - this instruction is cancelled by an interrupt.

<u>Workaround</u>

In principal, this problem can be avoided by inserting a NOP instruction between two multiplication instructions.</td></tr>
<tr><td>9</td><td>A/D converter function</td><td><u>Details</u>

Precision is limited to a 3-bit segment because the lower 7-bits of A/D conversion are not converted normally</td></tr>
<tr><td>10</td><td>PMx and PMCx read</td><td><u>Details</u>

The value of the Port X Mode Register (PMX) and the Port X Mode Control Register (PCMx) cannot b eread correctly. However, the write operation is performed correctly, and the operation mode setting conforms to the specified value.

Thte low-order 5 bits of the PMx register are fixed to 1 in hardware, and the low order 5-bits of the PMCx register are fixed to 0. These bits are read correctly. The PMx and PMCx registers should be able to read and written in 8-bit and 1-bit units.

Operation mode PMx Register – Value after Reset:
Value in User's Manual Actual read value
All modes FFh 1Fh

Operation mode PMCx Register – Value after Reset:
Value in User's Manual Actual read value
Singlechip 0 00h E0h
Singlechip 1 E0h 00h
Romless 0,1 E0h 00h

<u>Workaround:</u>

When Modiffying the contents of the PMx and PMCx registers, write data to the registers in byte-unit. Do not use bit operation instructions (clr1, not1, set1, tst1) to change the contents of the register.

PMx and PMCx registers are normally only written to once at system initialization, so the problem is not expected to significantly affected system operation.</td></tr>
<tr><td>11</td><td>Repeated execution of sld instruction</td><td>Data that should be loaded may not be transferred correctly to the register when sld instructions accessing external memory are repeatedly executed and interrupt processing is executed.

<u>Details</u></td></tr>
</table>

The malfunction occurs when the following conditions are met:

The instruction sequence <1> to <4> is executed and at <4> the sld instruction is executed repeatedly. The malfunction occurs when an interrupt occurs during the execution of the second sld instruction. When the malfunction occurs the data loaded by the sld instruction immediately before the interrupt is written to the register incorrectly.

<1> ld instruction or sld instruction

<2> One or more of any instruction other than the ld or mul instruction

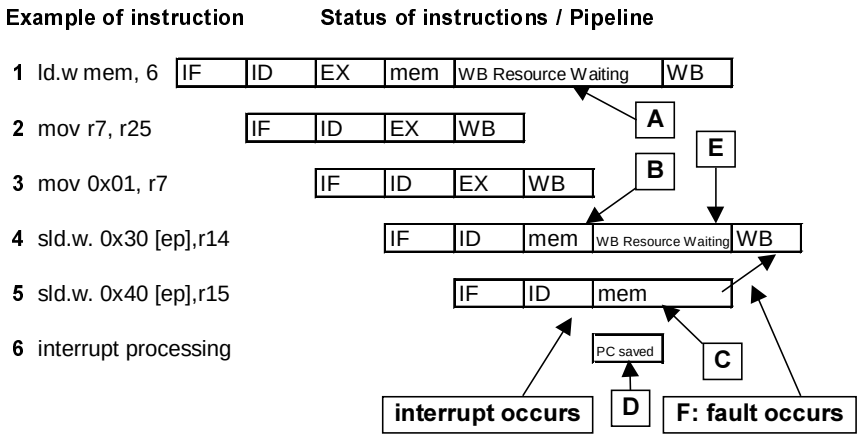
<3> One or more of any instructions that write to a register (applicable instructions are shown in table 1 below)

<4> sld instructions (repeated execution of sld instructions where data is loaded from external memory)
`sld` instruction ← !!interrupt occurs!!
`sld` instruction
…

This malfunction does not occur unless the sequence <1> to <4> is executed. Cases where this malfunction does not occur are shown below:

– The sld instruction in <4> loads from internal memory
– The instruction in <4> are load instructions other than sld
– The repeated sld instructions are separated by a branch instruction (such as a jr or a reti instruction)
– An ep (element pointer) setting occurs immediately before the repeated sld instructions
– A nop instruction is executed immediately before the repeated sld instructions
– Interrupts are disabled before executing the sequence <1> to <4>

This malfunction occurs when the following pipeline status occurs:



Explanation of the above pipeline operation:

A. To improve performance with V850E/MS1, the pipelining of other instructions goes ahead regardless of the dst. waiting for rsources in instruction 1; consequently the WB of instructions 2 and 3 are executed before the WB of instruction 1.

B. The EX stage of instruction 4 is omitted and mem access is executed because address calculation resources are available in the ID stage

C. Execution of mem access cannot be stopped even if an interrupt is

received in the ID stage of instruction 5, so a dummy access is performed

D. The PC is saved and so on due to the interrupt processing. Since this uses WB resources, the WB of instruction 1 is delayed even more

E. The WB of instruction 4 is made to wait one clock cycle because otherwise it would conflict with the WB of instruction 1

F. The data written in the WB stage of instruction 4 is the result of the dummy cycle of the mem stage of instruction 5 because reading cannot be masked due to the dummy cycle in instruction 5 and an incorrect write to r14 is performed causing the malfunction

Table 1: Instructions that write to a register immediately before the sld instructions

Mnemonic	Operand	Mnemonic	Operand
mov	R, r	movea	Imm16, R, r
not	R, r	movhi	imm16, R, r
divh	R, r	satsubi	imm16, R, r
satsubr	R, r	mov	imm16, R, r
satsub	R, r	ori	imm16, R, r
satadd	R, r	xori	imm16, R, r
zxb	R	andi	imm16, R, r
sxb	R	setf	cccc, r
zxh	R	ldsr	R, sr
sxh	R	stsr	SR, r
or	R, r	shr	R, r
xor	R, r	sar	R, r
and	R, r	shl	R, r
subr	R, r	sasf	cccc, r
sub	R, r	divh	R, r, w
add	R, r	divhu	R, r, w
mov	imm5, r	div	R, r, w
satadd	imm5, r	divu	R, r, w
add	imm5, r	cmov	imm5, r, m
shr	imm5, r	cmov	R, r, w
sar	imm5, r	bsw	r, w
shl	imm5, r	bsh	r, w
addi	imm16, R, r	hsw	r, w

Workaround

<Assembler>

This malfunction can be avoided by using one of the following countermeasures <a> to <c> below:

a. Change the first sld instruction to an ld instruction where repeated sld instructions are used. This malfunction will not occur if all the sld instructions are changed to ld instructions

b. Set ep immediately before the first sld instruction when repeated sld instructions are used

c. insert a nop instruction immediately before the firs sld instruction when repeated sld instructions are used

		<GHS compiler> This malfunction can be avoided using one of the following countermeasures so that the compiler does not output repeated sld instructions: a. Specify the following option at compilation: -Z1412 b. or, avoid using a TDA (Tiny Data Area) function pragma <IAR compiler> Using the IAR compiler this malfunction can be avoided by not using sld instructions. This can be achieved as follows: a. first enable SADDR; to do so activate 'short address mode' (Menu: Options – General) b. and do not use __saddr variables

(C) Flash Limitations

Limitations	µPD70F3102
	I / K,P / M
W/E cycles	5 / 10 / 20 ability (100 cannot be achieved)
Blocks	Sub Blocks 0 - 31 4kB write/erase is not possible

(D) Flash Programming Information

Programmer settings	µPD70F3102		µPD70F3102-A	
	K/P	M	I/P	M
	ES6.0	7.2	ES3.0	4.1
Programmer	Flash Master			
Vpp/V	7.8			
Erase/s (max)	20	20	20	20
Write/µs	50	20	50	20
Prewrite	yes			
W/E cycles	5/10	20	5/10	20
Block/Chip	all Blocks / Chip	all Blocks / Chip	all Blocks / Chip	all Blocks / Chip
Interface	UART 0(4800-76800 Bd)			
	CSI 0 (100Hz-1MHz)			

(E) Revision History

1st issue:

Author: H.-W. Hoeft

Date: Jun 4th 1998

Changes: initial issue

2nd issue:

Author: H.-W. Hoeft

Date:

3rd issue:

Author: H.-W. Hoeft

Date:

4st issue:

Author: H.-W. Hoeft

Date: Mar 3th 2000

Changes:

IE-703102-MC: added new control code version D
IE-703102-MC-EM1: added new control code version B
Added restrictions 16 to

5st issue:

Author: H.-W. Hoeft

Date: Feb 20th 2001

Changes:

Add new bug description for ‘repeate execution of sld instruction’ restriction no.11.
Change Flash information.
Change Flash programming information.