Microcomputer Technical Information

CP(K), O

32-Bit Microcontroller V850E/SV2 Usage Restrictions	Document No.	SBG-DT-04-0045	1/6
	Date issued	February 9, 2004	
	Issued by	Microcomputer Group 2nd Solutions Division Solutions Operations Unit NEC Electronics Corporation	
Related documents V850E/SV2 Hardware User's Manual: U16384E V850E1 Architecture User's Manual: U14559E	Notification classification	√	Usage restriction
			Upgrade
			Document modification
			Other notification

1. Affected products

- V850E/SV2 flash memory version

  μPD70F3116, 70F3116Y (internal flash memory: 512 KB, RAM: 24 KB)

- V850E/SV2 mask ROM version

  μPD703116, 703116Y (internal mask ROM: 512 KB, RAM: 24 KB)

* Restrictions No.1 to No.3 do not apply to the mask ROM version.

2. Details of restrictions

This notification concerns the following restrictions (No.1 to No.9). See attachment 1 for details.

No.1 Restriction on pseudo separate bus output mode

Pseudo separate bus output mode cannot be selected when an external bus is used.

No.2 Restriction on CLKOUT output

The CLKOUT pin outputs an undivided clock (f_{CPU}) regardless of whether the value set to the bus clock division control register (DVC) is 00H ($f_{DV} = f_{CPU}$) or 01H ($f_{DV} = f_{CPU}/2$). Since the base clock for the external clock (f_{DV}) is divided by the ratio set to the DVC register, there is no problem in a system in which the external bus is used asynchronous to CLKOUT.

No.3 Restriction when writing to buffer RAM of clocked serial interface (CSIA) with automatic transmit/receive function

When writing transmit data to the buffer RAM (CBUFnm), data may not be written correctly (e.g. Half_Word write is performed even though the Byte access instruction is used, or vice versa (n = 0 or 1, m = 0 to 15)).

No.4 Restriction on mul/mulu instruction

In the mul and mulu instructions, if an interrupt occurs during execution of an instruction that uses the same register for the 1st and 3rd operands, the operation result (register value of the 3rd operand) may be illegal. The instruction execution is terminated and the subsequent instruction is executed.

No.5 Restriction on DMA transfer forcible termination

When terminating a DMA transfer by setting the INITn bit of the DCHCn register, the transfer may not be terminated, but just suspended, even though the INITn bit is set (1). Or, after the DMA transfer is terminated, the DMA transfer of the target channel to which the INITn bit is set may be performed once with an initialized value.

No.6 Restriction on DMA transfer forcible suspension by NMI

DMA transfer is forcibly suspended by an NMI input during a DMA transfer. When an NMI interrupt is acknowledged, an instruction that has already been fetched is executed.

If this instruction is the one used to manipulate the Enn bit, the DMA transfer suspended by the NMI input cannot be restored as expected regardless of whether DMA transfer has been disabled or enabled.

No.7 Restriction on program execution and DMA transfer in internal RAM

When a DMA transfer for the internal RAM and a bit manipulation instruction (SET1, CLR1, or NOT1) allocated in the internal RAM or a data access instruction for a misaligned address are executed simultaneously, the CPU may deadlock due to conflict between the internal bus operations. At this time, only a reset can be acknowledged. (An NMI or interrupt cannot be acknowledged.)

This bug does not occur if no instruction is executed in the internal RAM, or no DMA transfer is performed on the internal RAM.

No.8 Restriction on automatic clear of TCn bit of DMA

When two or more channels of DMA transfer to the internal RAM (transfer source or transfer destination) are used simultaneously, however, the TCn bit may not be cleared even if the DMA transfer is complete. (n = 0 to 3)

No.9 Restriction on conflict between **sld** instruction and interrupt

If a conflict occurs between the decode operation of the instruction immediately before the **sld** instruction following a special instruction and an interrupt request before execution of the special instruction is complete, the execution result of the special instruction may not be stored in a register. This bug may only occur when the same register is used as the destination register of the special instruction and the **sld** instruction, and when the register value is referenced by the instruction followed by the **sld** instruction.

Special instruction:

- **ld** instruction: ld.b, ld.h, ld.w, ld.bu, ld.hu
- **sld** instruction: sld.b, sld.h, sld.w, sld.bu, sld.hu
- Multiply instruction: mul, mulh, mulhi, mulu

Examples of instruction sequence that may cause the bug:

Example 1:

```
<1> ld.w  [r11],r10
          :
<2> mov   r10  ,r28
<3> sld.w 0x28 ,r10
```

This bug occurs when the decode operation of **mov** (<2>) immediately before **sld** (<3>) and interrupt request servicing conflict before execution of the special instruction **ld** (<1>) is complete.

Example 2:

```
(1) ld.w  [r11],r10
          :
<2> cmp   imm5, r10
<3> sld.w 0x28 ,r10
<4> bz  label
```

This bug occurs when the decode operation of **comp** (<2>) immediately before **sld** (<3>) and interrupt request servicing conflict before execution of the special instruction **ld** (<1>) is complete. As a result, the compare result of **comp** becomes illegal, which may cause illegal operation of the branch instruction **bz** (<4>).

Example 3:

```
<1> ld.w  [r11],r10
          :
<2> add   imm5, r10
<3> sld.w 0x28 ,r10
<4> setf  r16
```

This bug occurs when the decode operation of **add** (<2>) immediately before **sld** (<3>) and interrupt request servicing conflict before execution of the special instruction **ld** (<1>) is complete. As a result, the result of **add** and the flag become illegal, which may cause illegal operation of the **setf** (<4>).

3. Workarounds

The following workarounds are available for these restrictions. See attachment 1 for details.

No.1 Restriction on pseudo separate bus output mode

Use multiplexed bus mode instead of pseudo separate bus output mode when an external bus is used.

No.2 Restriction on CLKOUT output

In a system in which the external bus is used in synchronization with CLKOUT at $f_{DV} = f_{CPU}/2$, divide the CLKOUT output by 2 using an external circuit and connect it to the target device.

No.3 Restriction when writing to buffer RAM of clocked serial interface (CSIA) with automatic transmit/receive function

Implement the workaround shown in attachment 1 by software.

This restriction is not applicable when using CSIA as a normal 3-wire serial interface without using the buffer RAM.

No.4 Restriction on mul/mulu instruction

Implement the workaround shown in attachment 1 by software.

No.5 Restriction on DMA transfer forcible termination

Implement the workaround shown in attachment 1 by software.

No.6 Restriction on DMA transfer forcible suspension by NMI

Initialize the DMA transfer forcibly suspended by the NMI and then execute it again.

No.7 Restriction on program execution and DMA transfer in internal RAM

Implement any of the following workarounds.

- Do not perform a DMA transfer for the internal RAM when an instruction allocated in the internal RAM is being executed.
- Do not execute an instruction allocated in the internal RAM when a DMA transfer for the internal RAM is being performed.

No.8 Restriction on automatic clear of TCn bit of DMA

When reading the TCn bit of the DCHCn register corresponding to the DMA transfer channel that targets the internal RAM, read the TC bit that has already been set (1) and then perform two dummy reads on the DCHCn register in succession. These three successive reads will properly clear the TCn bit (0)

No.9 Restriction on conflict between **sld** instruction and interrupt

Please regard this item as a usage restriction on the CPU function. A compiler that can automatically suppress generation of the instruction sequence that may cause the bug will be provided. Provision of the compiler varies depending on the compiler currently being used. Consult an NEC Electronics sales representative or distributor if you are using a compiler other than one below.

- When NEC Electronics compiler CA850 is used

  CA850 V2.61, which includes the countermeasure function for this bug, has been released via the online delivery service (ODS; user registration is required via mail after purchasing the compiler) on the following website.

  URL: http://www.necel.com/micro/ods/eng/index.html

- When GHS compiler CC850 is used

  We are planning to request the distributor (Advanced Data Controls Corp.) to upgrade Multi 4.0 (Rel. 7.0.0) and Multi 3.5.1 (Rel. 6.5.3) to versions that include the countermeasure function for this bug, so contact the distributor if using these versions. When using versions other than above, separately consult the distributor.

  Inquiry:
  TEL: +81-3-3576-6805
  E-mail: upgv850e@adac.co.jp

4. Action

Circuit modification is planned for No.1 to No.3. Circuit modification is not planned for No.4 to No.9 and these items have been added to the cautions on use.

The schedule for circuit modification is as follows.

Flash memory ES version (ES 1.1): February 2004
Flash memory MP version (rank E): October 2004 or later

* The rank is indicated by the letter appearing as the 5th digit from the left in the lot number marked on each product.

5. Supplement for No.9 Restriction on conflict between **sld** instruction and interrupt

Check whether or not the restriction applies to your system if it is under development or in mass production.

[Support for system already developed]

If your system has already been developed, judge whether or not the bug applies to your system by following the procedure in attachment 2, Bug Check Sheet.

Since there are several factors that prevent this bug, the bug is unlikely to occur even if the instruction sequence that may cause the bug is executed.

Please read this document thoroughly and take appropriate measures.

[Bug check for embedded software products]

The bug check status of NEC Electronics real-time OSs and middleware is shown below.

- Real-time OS

RX850:	Not applicable
RX850 Pro:	Not applicable

- Middleware

File system (RX-FS):	Not applicable
Network library (RX-NET):	Not applicable
High-speed floating-point library (GOFAST):	Not applicable
Speech synthesis library (Text to Speech):	Not applicable

When using products other than the above, separately consult an NEC Electronics sales representative or distributor for whether the bug applies or not. When using products from a third party, separately consult each manufacture.

6. Usage restrictions

The restriction history and detailed information is described in attachment 1.

7. Document revision history

V850E/SV2 Usage Restrictions Revision History

Document Number	Date Issued	Description
SBG-DT-04-0045 (This document)	February 9, 2004	Newly created. Addition of No.1 to No.9

List of Restrictions in V850E/SV2

1. Product Version

μPD70F3166, 70F3166Y:	ES 1.0, rank K
	ES 1.1, rank E
μPD703166, 703166Y:	Rank K

* The rank is indicated by the letter appearing as the 5th digit from the left in the lot number marked on each product.

2. Product History

μPD70F3166, 70F3166Y

No.	Restrictions	Version and Rank	
		ES 1.0, Rank K	ES 1.1, Rank E
1	Restriction on pseudo separate bus output mode	×	√
2	Restriction on CLKOUT output	×	√
3	Restriction when writing to buffer RAM of clocked serial interface (CSIA) with automatic transmit/receive function	×	√
4	Restriction on mul/mulu instruction	Δ	Δ
5	Restriction on DMA transfer forcible termination	Δ	Δ
6	Restriction on DMA transfer forcible suspension by NMI	Δ	Δ
7	Restriction on program execution and DMA transfer in internal RAM	Δ	Δ
8	Restriction on automatic clear of TCn bit of DMA	Δ	Δ
9	Restriction on conflict between sld instruction and interrupt	Δ	Δ

√: Restriction does not apply, Δ: Restriction will also apply in future,
×: Restriction applies (to be corrected in future revision)

μPD703166, 703166Y

No.	Restrictions	Rank
		K
1	Restriction on pseudo separate bus output mode	√
2	Restriction on CLKOUT output	√
3	Restriction when writing to buffer RAM of clocked serial interface (CSIA) with automatic transmit/receive function	√
4	Restriction on mul/mulu instruction	Δ
5	Restriction on DMA transfer forcible termination	Δ
6	Restriction on DMA transfer forcible suspension by NMI	Δ
7	Restriction on program execution and DMA transfer in internal RAM	Δ
8	Restriction on automatic clear of TCn bit of DMA	Δ
9	Restriction on conflict between sld instruction and interrupt	Δ

√: Restriction does not apply, Δ: Restriction will also apply in future,
×: Restriction applies (to be corrected in future revision)

3. Details of Restrictions

No.1 Restriction on pseudo separate bus output mode

[Description]

Pseudo separate bus output mode cannot be selected when an external bus is used.

[Workaround]

Use multiplexed bus mode instead of pseudo separate bus output mode when an external bus is used.

No.2 Restriction on CLKOUT output

[Description]

The CLKOUT pin outputs an undivided clock (f_{CPU}) regardless of whether the value set to the bus clock division control register (DVC) is 00H ($f_{DV} = f_{CPU}$) or 01H ($f_{DV} = f_{CPU}/2$).

Since the base clock for the external clock (f_{DV}) is divided by the ratio set to the DVC register, there is no problem in a system in which the external bus is used asynchronous to CLKOUT.

[Workaround]

In a system in which the external bus is used in synchronization with CLKOUT at $f_{DV} = f_{CPU}/2$, divide the CLKOUT output by 2 using an external circuit and connect it to the target device.

This restriction has been corrected in control code C.

No.3 Restriction when writing to buffer RAM of clocked serial interface (CSIA) with automatic transmit/receive function

[Description]

When writing transmit data to the buffer RAM (CBUFnm), data may not be written correctly (e.g. Half_Word write is performed even though the Byte access instruction is used, or vice versa (n = 0 or 1, m = 0 to 15)).

[Condition under which this restriction does not apply]

This restriction is not applicable when using CSIA as a normal 3-wire serial interface without using the buffer RAM.

[Workaround]

Implement any of the workarounds shown below by software when writing to the buffer RAM.

Workaround (1)

After data is written, read the data using the Half_Word access instruction (ST.H) and verify the values. If the data do not match, continue writing until the data match. At this time, it is not necessary to disable interrupts.

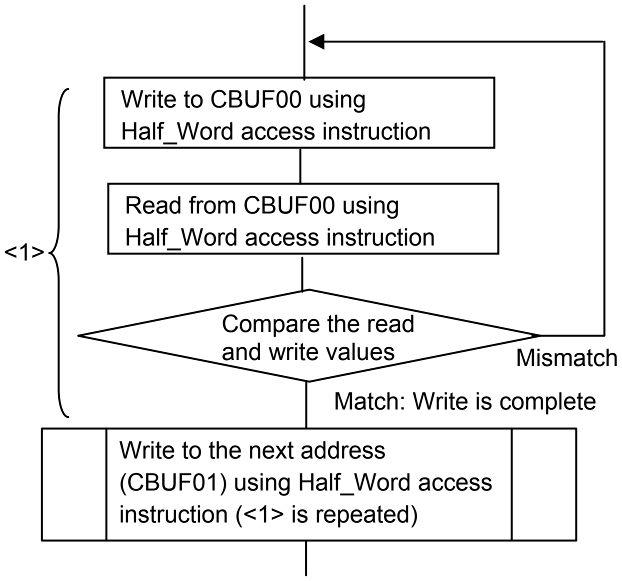
When writing an odd number of bytes, write dummy data to the buffer RAM (CBUFnmH) at the higher address. (n = 0 or 1, m = 0 to 15)

The Half_Word access instruction must be used to access registers until the compare result matches and write is confirmed.

<Example of writing to buffer RAM (CBUF00 to CBUF01)>

Address		
FFFFFE40H	CBUF00	
FFFFFE40H		CBUF00L
FFFFFE41H		CBUF00H
FFFFFE42H	CBUF01	
FFFFFE42H		CBUF01L
FFFFFE43H		CBUF01H

Buffer RAM configuration


Write to CBUF00 using Half_Word access instruction
Read from CBUF00 using Half_Word access instruction
<1>
Compare the read and write values
Mismatch
Match: Write is complete
Write to the next address (CBUF01) using Half_Word access instruction (<1> is repeated)

<Example of writing only 1-byte data (to CBUF00L)>

Perform operation <1> to write data to CBUF00L (the buffer RAM at the lower address), by writing dummy data to CBUF00H (the buffer RAM at the higher address).

Workaround (2)

Disable interrupts using the DI instruction and write data to the buffer RAM using the Half_Word access instruction (ST.H) only. In addition, re-write the last 2 bytes of the written data again. Do not execute instructions other than NOP and operation instructions (arithmetic, saturate, and logical) between when data is written to the buffer RAM and when the last 2 bytes are rewritten.

When writing an odd number of bytes, write dummy data to the buffer RAM (CBUFnmH) at the higher address. (n = 0 or 1, m = 0 to 15)

Workaround (3)

Disable interrupts using the DI instruction and write data to the buffer RAM using the byte access instruction (ST.B) only. In addition, secure a time equivalent to 5 or more input clocks (f_{SCKA}) selected by bits 6 and 7 (CKSAn1 and CKSAn0) of CSISAn by executing a NOP or operation instruction (arithmetic, saturate, or logical) after the last data is written to the buffer RAM. (n = 0 or 1)

No.4 Restriction on mul/mulu instruction

[Description]

In the mul and mulu instructions, if an interrupt occurs during execution of an instruction that uses the same register for the 1st and 3rd operands, the operation result (register value of the 3rd operand) may be illegal. The instruction execution is terminated and the subsequent instruction is executed.

When the NEC Electronics C compiler is used, there is no problem as long as this restriction does not affect the description in the assembly language. The global search function, etc., in the Project Manager can be used to confirm the existence of such a description.

Refer to [Related products] below for information about GHS, Inc., Red Hat Inc. and Wind River Systems, Inc.

Example:

```
  mul     reg1, reg2, reg1
```

And

```
  mulu    reg1, reg2, reg1
```

; Registers reg1 and reg2 are not identical. reg1 ≠ r0 (zero register).

[Condition under which this restriction does not apply]

This restriction does not apply if the register used for the 1st and 3rd operands is not identical.

The NEC Electronics C compiler does not create the instruction format to which this restriction applies. In addition, the real-time OSs (RX850 and RX850 Pro), all middleware products do not use the instruction format to which this restriction applies.

[Workaround]

Describe the program as shown below.

```
  mul     reg1, reg2, reg3
```

And

```
  mulu    reg1, reg2, reg3
```

; **Registers reg1, reg2, and reg3 are not identical.** reg3 ≠ r0.

Or

```
  mov     reg1, rtmp
  mul     rtmp, reg2, reg1
```

And

```
  mov     reg1, rtmp
  mulu    rtmp, reg2, reg1
```

; **Registers reg1, reg2, and rtmp are not identical.** reg1 and rtmp ≠ r0.

[Related products]

- GHS products

In the C compiler up to Ver.1.8.9, the instruction format to which this restriction applies may be selected and created if the customer uses the embedded function __MULSH() or __MULUH(), and depending on the optimization mode setting.

Extract all the mul instructions by using “% gdump a.out | grep mul” by using the “gdump”, which is a disassembler included in the GHS compiler, to check for the existence of the instruction to which this restriction applies.

In MULTI2000 Rel.3.5 and later, this instruction format will not be created in both C description and assembly language description, and the instruction will be detected as an error when assembling the program. MULTI2000 Rel.3.5 is scheduled for release in February 2002 in the US and March 2002 in Japan. Contact Advanced Data Controls, Corp. for the detailed schedule. For the runtime library, the mul/mulu instructions are used but this instruction format is not.

- Products of Red Hat Inc. and Wind River Systems, Inc.

  The GNU compiler from these companies does not create the instruction format to which this restriction applies.

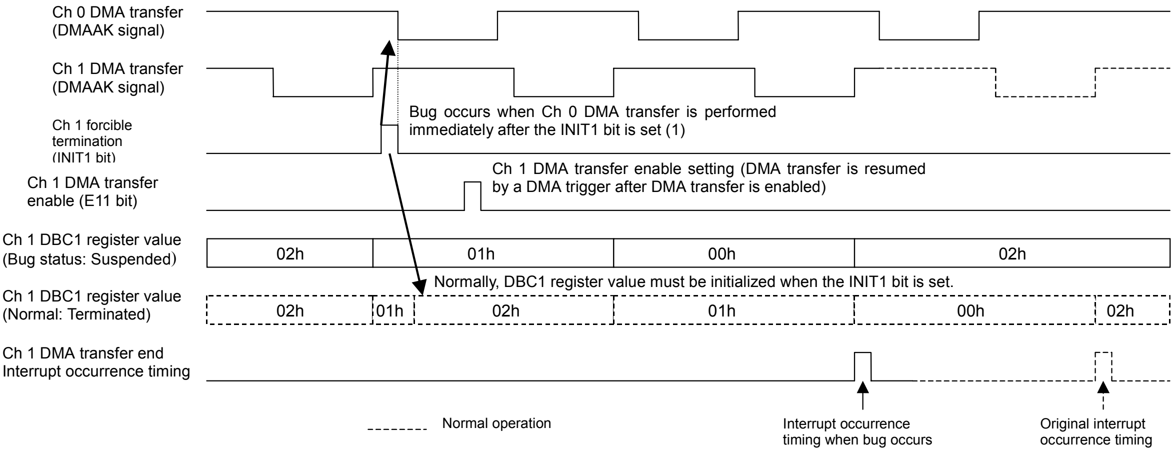
No.5 Restriction on DMA transfer forcible termination

[Description]

When terminating a DMA transfer by setting the INITn bit of the DCHCn register, the transfer may not be terminated, but just suspended, even though the INITn bit is set (1). As a result, when the DMA transfer of a channel that should have been terminated is resumed, the DMA transfer will terminate after an unexpected number of transfers are completed and a DMA transfer completion interrupt may occur (n = 0 to 3). This bug occurs if a DMA transfer is executed immediately after a forcible termination is set (by setting the INITn bit) (see the figure below).

This bug does not depend on the number of transfer channels, transfer type (2-cycle or flyby), transfer target (between memory and memory, memory and I/O; including internal resources), transfer mode (single, single-step, or block), or trigger (external request, interrupt from internal peripheral I/O, or software), and can occur with any combination of the above elements that can be set under the specifications. In addition, another channel may affect the occurrence of this bug.

Both ch 0 and ch 1 are in single transfer mode, ch1 DMA transfer count is 3 (DBC1 register set value = 02h)


Ch 0 DMA transfer (DMAAK signal)
Ch 1 DMA transfer (DMAAK signal)
Ch 1 forcible termination (INIT1 bit)
Ch 1 DMA transfer enable (E11 bit)
Ch 1 DBC1 register value (Bug status: Suspended)
Ch 1 DBC1 register value (Normal: Terminated)
Ch 1 DMA transfer end Interrupt occurrence timing
Bug occurs when Ch 0 DMA transfer is performed immediately after the INIT1 bit is set (1)
Ch 1 DMA transfer enable setting (DMA transfer is resumed by a DMA trigger after DMA transfer is enabled)
02h
01h
00h
02h
Normally, DBC1 register value must be initialized when the INIT1 bit is set.
02h
01h
02h
01h
00h
02h
Normal operation
Interrupt occurrence timing when bug occurs
Original interrupt occurrence timing

The following registers are buffer registers with a 2-stage FIFO configuration of master and slave. If these registers are overwritten during a DMA transfer or in the DMA-suspended status, the value is written to the master register, and reflected in the slave register when the DMA transfer of the overwritten channel is terminated.

The “initialization” in the above figure means that the contents of the master register are reflected in the slave register.

2-stage FIFO configuration registers:

- DMA source address register (DSAnH, DSAnL)
- DMA destination address register (DDAnH, DDAnL)
- DMA transfer count register (DBCn)

[Workaround]

This bug can be avoided by implementing any of the following procedures using the software.

(1) Stop all the transfers from DMA channels temporarily

The following measure is effective if the following condition is satisfied.

- Except for the following workaround processing, the program does not assume that the TCn bit of the DCHCn register is 1. (Since the TCn bit of the DCHCn register is cleared (0) when it is read, execution of the following procedure (b) under <5> clears this bit.)

[Procedure to avoid bug]

<1> Disable interrupts (DI state).

<2> Read the DMA restart register (DRST) and transfer the ENn bit of each channel to a general-purpose register (value A).

<3> Write 00H to the DMA restart register (DRST) twice**Note**.
By executing twice**Note**, the DMA transfer is definitely stopped before proceeding to <4>.

<4> Set (1) the INITn bit of the DCHCn register of the channel that should be terminated forcibly.

<5> Perform the following operations for value A read in <2>. (Value B)

(a) Clear (0) the bit of the channel that should be terminated forcibly

(b) If the TCn and ENn bits of the channel that is not terminated forcibly are 1 (AND makes 1), clear (0) the bit of the channel.

<6> Write value B in <5> to the DRST register.

<7> Enable interrupts (EI state).

Remarks 1. Be sure to execute <5> to prevent the ENn bit from being set illegally for channels that are terminated normally during the period of <2> and<3>.

2. n = 0 to 3

Note Execute three times if the transfer target (transfer source or transfer destination) is the internal RAM.

(2) Repeat setting the INITn bit until the forcible DMA transfer termination is correctly performed (n = 0 to 3)

[Procedure to avoid bug]

<1> Copy the initial transfer count of the channel that should be terminated forcibly to a general-purpose register.

<2> Set (1) the INITn bit of the DCHCn register of the channel that should be terminated forcibly.

<3> Read the value of the DMA transfer count register (DBCn) of the channel that should be terminated forcibly and compare the value with the one copied in <1>. If the values do not match, repeat <2> and <3>.

Remarks **1.** When the DBCn register is read in procedure <3>, the remaining transfer count will be read if the DMA is stopped due to this bug. If the forcible DMA termination is performed correctly, the initial transfer count will be read.

2. Note that it may take some time for forcible termination to take effect if this workaround is implemented in an application in which DMA transfers other than for channels subject to forcible termination are frequently performed.

No.6 Restriction on DMA transfer forcible suspension by NMI

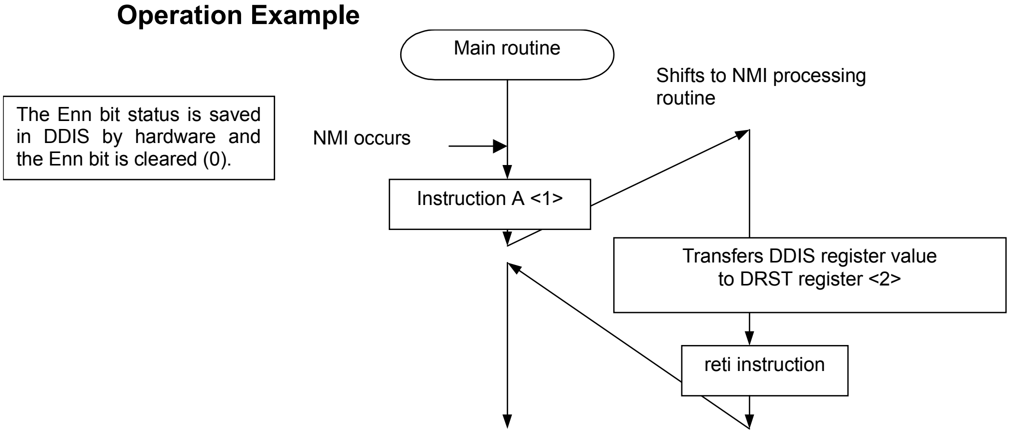
[Description]

DMA transfer is forcibly suspended by an NMI input during a DMA transfer. At this time, the DMA controller saves the status of the Enn bit (bit 0 of the DCHCn register) of all the DMA channels to the DDIS register, clears (0) the Enn bit, and disables the DMA transfer. In addition, when an NMI interrupt is acknowledged in the CPU pipeline operation, an instruction that has already been fetched is executed (1 instruction max.).

If this instruction is the one used to manipulate the Enn bit, the contents of the DDIS register are transferred to the DRST register in the NMI processing routine in order to resume the suspended DMA transfer. Therefore, the Enn bit setting immediately after the NMI input is not reflected.

As a result, the DMA transfer suspended by the NMI input cannot be restored as expected regardless of whether DMA transfer has been disabled or enabled.

Operation Example


Main routine
Shifts to NMI processing routine
The Enn bit status is saved in DDIS by hardware and the Enn bit is cleared (0).
NMI occurs
Instruction A <1>
Transfers DDIS register value to DRST register <2>
reti instruction

When the Enn bit of the DCHCn register is set to disable or enable DMA using instruction A, the disable/enable status of each DMA channel is inconsistent in <1> and <2>. As a result, the normally suspended DMA transfer cannot be restored correctly.

Furthermore, when the Enn bit is set (1) using instruction A while the software trigger bit (STGn) of the DCHCn register is set (1), DMA transfer is performed even in the NMI processing routine.

[Condition under which this bug does not occur]

There is no problem if an NMI is not used.

In addition, there is no problem if the system does not assume to resume the DMA transfer forcibly suspended by the NMI.

[Workaround]

Initialize the DMA transfer forcibly suspended by the NMI and then execute it again.

No.7 Restriction on program execution and DMA transfer in internal RAM

[Description]

When a DMA transfer for the internal RAM and a bit manipulation instruction (SET1, CLR1, or NOT1) allocated in the internal RAM or a data access instruction for a misaligned address are executed simultaneously, the CPU may deadlock due to conflict between the internal bus operations. At this time, only a reset can be acknowledged. (An NMI or interrupt cannot be acknowledged.)

[Condition under which this bug does not occur]

This bug does not occur if no instruction is executed in the internal RAM, or no DMA transfer is performed on the internal RAM.

[Workaround]

Implement any of the following workarounds.

- Do not perform a DMA transfer for the internal RAM when an instruction allocated in the internal RAM is being executed.
- Do not execute an instruction allocated in the internal RAM when a DMA transfer for the internal RAM is being performed.

No.8 Restriction on automatic clear of TCn bit of DMA

[Description]

The TCn bit of the DCHCn register should automatically be cleared when it is read. When two or more channels of DMA transfer to the internal RAM (transfer source or transfer destination) are used simultaneously, however, the TCn bit may not be cleared even if the DMA transfer is complete (n = 0 to 3)

[Condition under which this bug does not occur]

This bug does not occur if any of the following conditions is satisfied.

- Only one channel is used for the DMA transfer.
- The internal RAM (transfer source or transfer destination) is not the target of the DMA transfer.

[Supplement]

This bug occurs only when setting the TC bit (1) is held pending while the DCHCn register is being polled by the CPU program.

If all the following four conditions are satisfied, however, this bug may occur when the TC bit of the DCHCn register is read in the interrupt routine triggered by the DMA transfer end interrupt (INTDMAn).

(1) Multiple channels are used for the DMA transfer.

(2) The DMA transfer is performed from the internal RAM.

(3) The VSWC register setting value is 11H (operating frequency: 4 to less than 33 MHz).

(4) A load/store instruction is not performed on internal RAM, internal I/O area, or external memory before the DCHCn register is read in the DMA transfer end interrupt routine (the DCHCn register is the first register to be accessed).

[Workaround by software]

When reading the TCn bit of the DCHCn register corresponding to the DMA transfer channel that targets the internal RAM, read the TC bit that has already been set (1) and then perform two dummy reads on the DCHCn register in succession. These three successive reads will properly clear the TCn bit (0).

No.9 Restriction on conflict between **sld** instruction and interrupt

[Description]

If a conflict occurs between the decode operation of the instruction (<2> in the examples) immediately before the **sld** instruction (<3> in the examples) following a special instruction (<1> in the examples) and an interrupt request before execution of the special instruction is complete, the execution result of the special instruction may not be stored in a register.

This bug may only occur when the same register is used as the destination register of the special instruction and the **sld** instruction, and when the register value is referenced by the instruction followed by the **sld** instruction.

Special instruction:

• **ld** instruction:	ld.b, ld.h, ld.w, ld.bu, ld.hu
• **sld** instruction:	sld.b, sld.h, sld.w, sld.bu, sld.hu
• Multiply instruction:	mul, mulh, mulhi, mulu

Examples of instruction sequence that may cause the bug:

Example 1:

```
<1> ld.w  [r11],r10
          :
<2> mov   r10  ,r28
<3> sld.w 0x28 ,r10
```

This bug occurs when the decode operation of **mov** (<2>) immediately before **sld** (<3>) and interrupt request servicing conflict before execution of the special instruction **ld** (<1>) is complete.

Example 2:

```
(1) ld.w  [r11],r10
          :
<2> cmp   imm5, r10
<3> sld.w 0x28 ,r10
<4> bz  label
```

This bug occurs when the decode operation of **comp** (<2>) immediately before **sld** (<3>) and interrupt request servicing conflict before execution of the special instruction **ld** (<1>) is complete. As a result, the compare result of **comp** becomes illegal, which may cause illegal operation of the branch instruction **bz** (<4>).

Example 3:

```
<1> ld.w  [r11],r10
          :
<2> add   imm5, r10
<3> sld.w 0x28 ,r10
<4> setf  r16
```

This bug occurs when the decode operation of **add** (<2>) immediately before **sld** (<3>) and interrupt request servicing conflict before execution of the special instruction **ld** (<1>) is complete. As a result, the result of **add** and the flag become illegal, which may cause illegal operation of the **setf** (<4>).

[Conditions under which this bug occurs]

This bug may occur when all the following conditions (1) to (3) are satisfied.

(1) Either condition (I) or (II) is satisfied

Condition (I):

The same register is used as the destination register of a special instruction (see below) and the subsequent **sld** instruction and as the source register (reg1) of an instruction shown below followed by the **sld** instruction. (See Example 1.)

mov **reg1**,reg2   not **reg1**,reg2   satsubr **reg1**,reg2   satsub **reg1**,reg2
satadd **reg1**,reg2   or **reg1**,reg2   xor **reg1**,reg2   and **reg1**,reg2
tst **reg1**,reg2   subr **reg1**,reg2   sub **reg1**,reg2   add **reg1**,reg2
cmp **reg1**,reg2   mulh **reg1**,reg2

Condition (II):

The same register is used as the destination register of a special instruction (see below) and the subsequent **sld** instruction and as the source register (reg2) of an instruction shown below followed by the **sld** instruction. (See Examples 2 and 3.)

not reg1,**reg2**   satsubr reg1,**reg2**   satsub reg1,**reg2**   satadd reg1,**reg2**
satadd imm5,**reg2**   or reg1,**reg2**   xor reg1,**reg2**   and reg1,**reg2**
tst reg1,**reg2**   subr reg1,**reg2**   sub reg1,**reg2**   add reg1,**reg2**
add imm5,**reg2**   cmp reg1,**reg2**   cmp imm5,**reg2**   shr imm5,**reg2**
sar imm5,**reg2**   shl imm5, **reg2**

Special instruction:

• **ld** instruction:	ld.b, ld.h, ld.w, ld.bu, ld.hu
• **sld** instruction:	sld.b, sld.h, sld.w, sld.bu, sld.hu
• Multiply instruction:	mul, mulh, mulhi, mulu

(2) When the execution result of the special instruction (see above) has not been stored in the destination register before execution of the instruction (instruction of condition (I) or (II)) immediately before the **sld** instruction starts in the CPU pipeline.

(3) When the decode operation of the instruction (instruction of condition (I) or (II)) immediately before the **sld** instruction and interrupt request servicing conflict.

[Workaround]

Please regard this item as a usage restriction on the CPU function. A compiler that can automatically suppress generation of the instruction sequence that may cause the bug will be provided. Provision of the compiler varies depending on the compiler currently being used. Consult an NEC Electronics sales representative or distributor if you are using a compiler other than one below.

- **When NEC Electronics compiler CA850 is used**

CA850 V2.61, which includes the countermeasure function for this bug, has been released via the online delivery service (ODS; user registration is required via mail after purchasing the compiler) on the following website.

URL: http://www.necel.com/micro/ods/eng/index.html

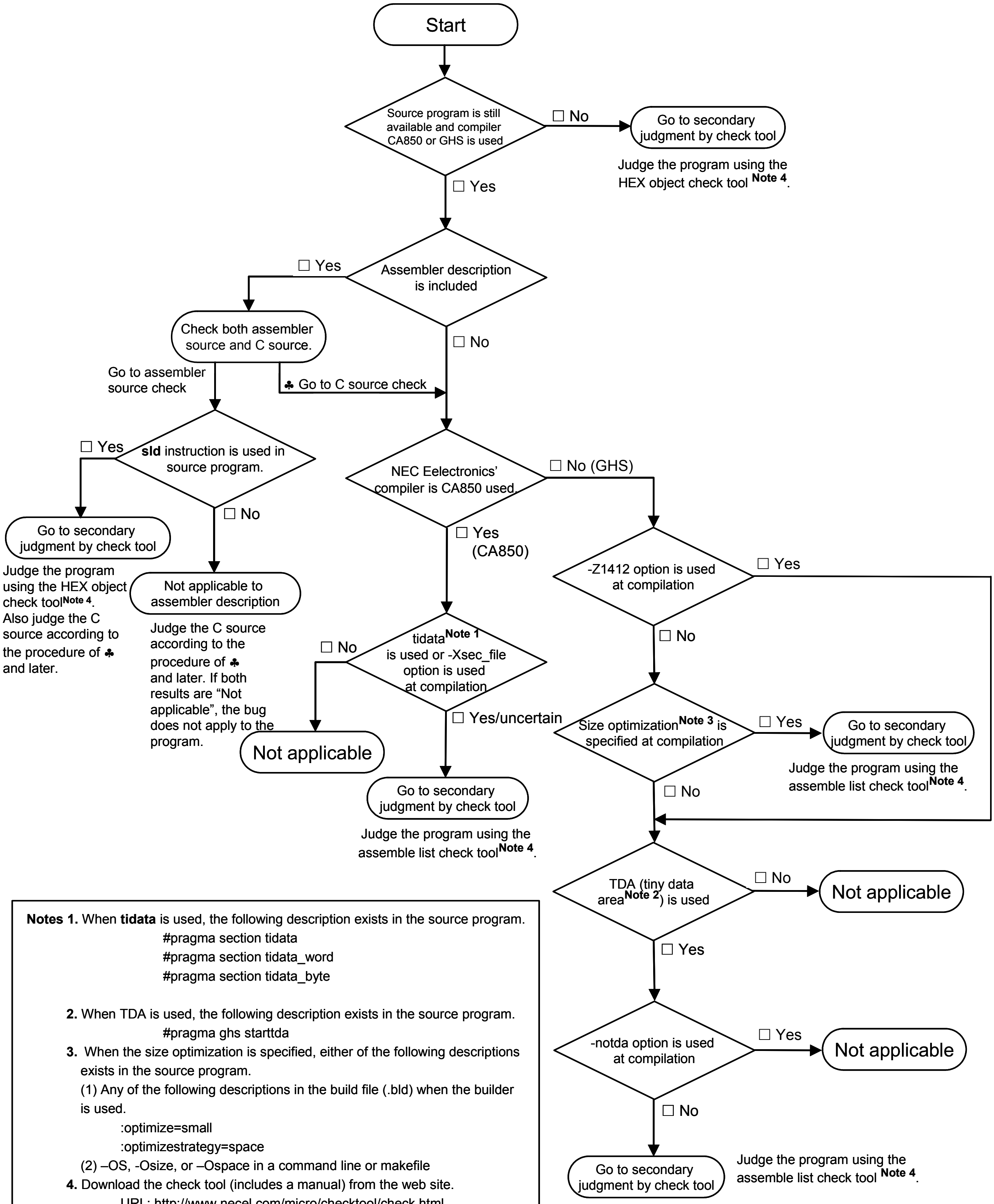
- **When GHS compiler CC850 is used**

We are planning to request the distributor (Advanced Data Controls Corp.) to upgrade Multi 4.0 (Rel. 7.0.0) and Multi 3.5.1 (Rel. 6.5.3) to versions that include the countermeasure function for this bug, so contact the distributor if using these versions. When using versions other than above, separately consult the distributor.

Inquiry:
TEL: +81-3-3576-6805
E-mail: upgv850e@adac.co.jp

Bug Check Sheet (for Primary Judgment)

Perform primary judgment for whether the bug applies or not according to the following flowchart. If the result is not “Not applicable”, then proceed to secondary judgment using the check tool.


Start
Source program is still available and compiler CA850 or GHS is used
□ No
Go to secondary judgment by check tool
Judge the program using the HEX object check tool Note 4.
□ Yes
Assembler description is included
□ Yes
Check both assembler source and C source.
□ No
Go to assembler source check
♣ Go to C source check
□ Yes
sld instruction is used in source program.
□ No
Go to secondary judgment by check tool
Judge the program using the HEX object check toolNote 4. Also judge the C source according to the procedure of ♣ and later.
Not applicable to assembler description
Judge the C source according to the procedure of ♣ and later. If both results are “Not applicable”, the bug does not apply to the program.
NEC Eelectronics’ compiler is CA850 used.
□ No (GHS)
□ Yes (CA850)
-Z1412 option is used at compilation
□ Yes
□ No
tidataNote 1 is used or -Xsec_file option is used at compilation
□ No
Not applicable
□ Yes/uncertain
Go to secondary judgment by check tool
Judge the program using the assemble list check toolNote 4.
Size optimizationNote 3 is specified at compilation
□ Yes
Go to secondary judgment by check tool
Judge the program using the assemble list check toolNote 4.
□ No
TDA (tiny data areaNote 2) is used
□ No
Not applicable
□ Yes
-notda option is used at compilation
□ Yes
Not applicable
□ No
Go to secondary judgment by check tool
Judge the program using the assemble list check tool Note 4.

Notes 1. When **tidata** is used, the following description exists in the source program.
#pragma section tidata
#pragma section tidata_word
#pragma section tidata_byte

2. When TDA is used, the following description exists in the source program.
#pragma ghs starttda

3. When the size optimization is specified, either of the following descriptions exists in the source program.
(1) Any of the following descriptions in the build file (.bld) when the builder is used.
:optimize=small
:optimizestrategy=space
(2) –OS, -Osize, or –Ospace in a command line or makefile

4. Download the check tool (includes a manual) from the web site.
URL: http://www.necel.com/micro/checktool/check.html