Microcomputer Technical Information

CP(K), O

32-Bit Microcontroller V850E/MA2 Usage Restrictions	Document No.	SBG-DT-04-0056	1/3
	Date issued	February 9, 2004	
	Issued by	Microcomputer Group 2nd Solutions Division Solutions Operations Unit NEC Electronics Corporation	
Related documents V850E/MA2 Hardware User's Manual: U14980E V850E1 Architecture User's Manual: U14559E	Notification classification	√ Usage restriction	
		Upgrade	
		Document modification	
		Other notification	

1. Affected product

V850E/MA2

μPD703108

2. Details of restrictions

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

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. Action

[Support for system under development or to be developed]

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

[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.

4. Usage restrictions

Notes on using the V850E/MA2, including the restriction history and detailed information, will be described on the following pages. See the following attachments for details.

5. Document revision history

V850E/MA2 Usage Restrictions Revision History

Document Number	Date Issued	Description
SBG-DT-0057-E	March 11, 2002	Newly created. Addition of No.1
SBG-DT-0077-E	April 11, 2002	Addition of No.2 and No.3
SBG-DT-0116-E	July 29, 2002	Addition of No.4 to No.6
SBG-DT-0134-E	October 31, 2002	Addition of No.7 and No.8
SBG-DT-04-0056 (This document)	February 9, 2004	Addition of No.9

List of Restrictions in V850E/MA2

1. Product Version

μPD703108: Rank K, E

* 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

μPD703108

No.	Bugs	Rank
		K, E
1	Restriction on mul/mulu instruction	Δ
2	Restriction on page ROM access	Δ
3	Restriction on A/D converter	Δ
4	Bug related to DMA transfer forcible termination	Δ
5	Bug that DMA transfer is forcibly suspended by NMI	Δ
6	Bug in program execution and DMA transfer in internal RAM	Δ
7	Bug related to DMA transfer whose transfer count is set to two	Δ
8	Bug that TCn bit of DMA is not cleared automatically	Δ
9	Restriction on conflict between sld instruction and interrupt	Δ

√: Bug does not occur, Δ: Bug will also apply in future, ×: Bug occurs

3. Details of Bugs and Additions to Specifications

No.1 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 an 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 Multi 2000 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. Multi 2000 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.2 Restriction on page ROM access

[Description]

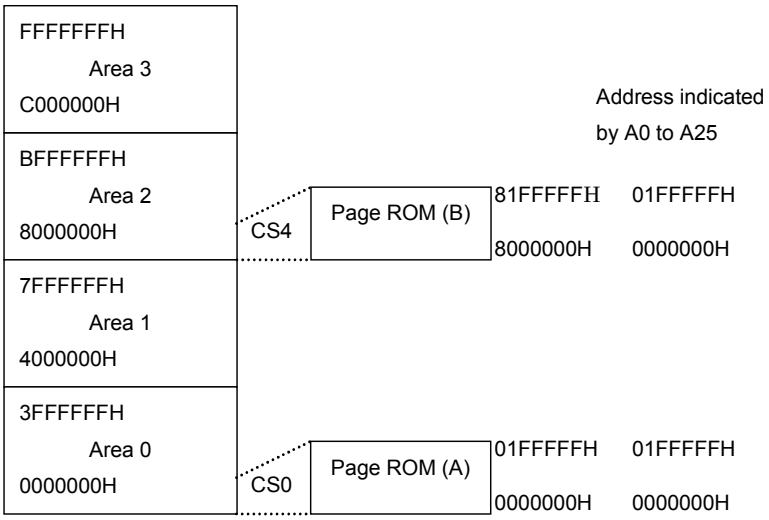
When a CPU data space is used in a system in which multiple page ROMs are connected to multiple CS areas, when a page ROM access and page ROM access to another CS area occur in succession, if the address value of the former address and that of the latter are within the same page length of

the page ROM, it is judged that the same page ROM was accessed, even though the page ROMs with different CS areas were accessed. As a result, an on-page cycle will be issued for the latter access, but because the data access time for the latter access is insufficient, data cannot be read correctly.

Remark Page ROM includes memory that is capable of accessing continuous addresses on a page at high-speed, such as mask ROM with a page access function.

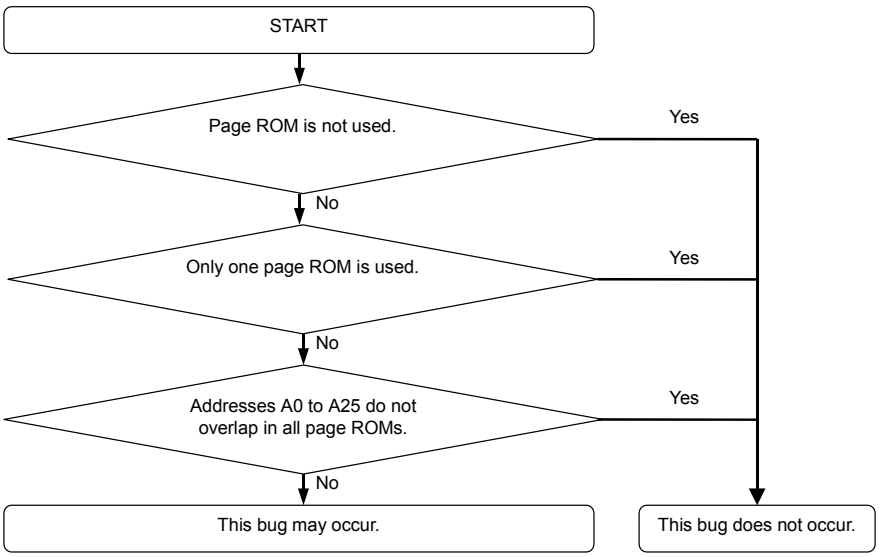
Example:
When accessing address 0xxxxx0H in the CS0 area is followed by accessing address 8xxxxx2H in the CS4 area, an on-page cycle will be issued for address 8xxxxx2H.

Example of configuration in which this bug can occur



Bug check procedure

The following flowchart can be used to check whether this bug occurs or not. Confirm if this bug applies your company's product using this flowchart.

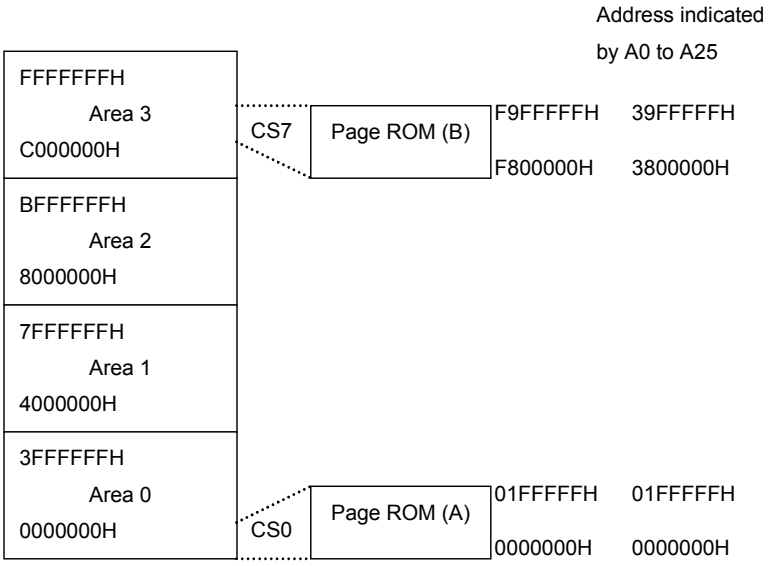


[Workaround]

When multiple page ROMs are used, allocate addresses 25 to 0 so that they do not overlap in each page ROM.

For example, when allocating two 2 MB page ROMs to different CS areas, allocate one page ROM within addresses 0000000H to 01FFFFFH, and the other within addresses F800000H to F9FFFFFH.

Example of configuration in which this bug does not occur



No.3 Restriction on A/D converter

[Description]

This restriction occurs when the timer 1 trigger mode of the A/D converter is used (this bug does not occur when the A/D trigger mode or timer 4 trigger mode is used).

Originally, only INTM000 can be the trigger to start the A/D converter in the timer 1 trigger mode, therefore the interrupt sources listed below should be ignored. However, if one of the interrupt sources listed below occurs immediately before the end of A/D conversion (<1> in the figure below; 2 internal system clocks), it is mistakenly judged as the A/D conversion start trigger. As a result, A/D conversion is started again after the A/D conversion end interrupt (INTAD) is issued. The first A/D conversion ends correctly and the result is stored in the ADCRn register (this value can be read during the second conversion).

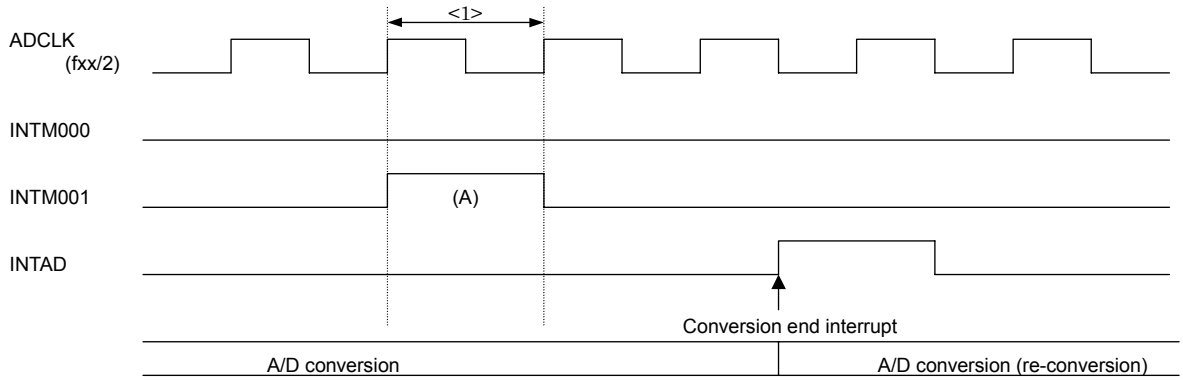
The restarted A/D converter performs the conversion operation correctly, issues the A/D conversion end interrupt (INTAD) and stops. The result is overwritten to the ADCRn register.

[Interrupt source that can trigger A/D conversion]

Timer match interrupt	INTM001
	INTM010
	INTM011
External pin interrupt Note	INTP001
	INTP010
	INTP011

Note The external interrupt signal that functions alternately as the external capture trigger input of timer C (channels 0 and 1) can also be a trigger for re-conversion. In the case of an external interrupt input, this bug occurs when a valid edge is input before the timing of (A) in the figure below by the noise eliminator (analog delay (60 to 220 ns)).

[Example of timing at which this bug occurs]



fxx: Internal system clock

[Condition under which this bug does not occur]

This bug does not occur when the A/D trigger mode or timer 4 trigger mode is used.

The condition under which this bug does not occur in the timer 1 trigger mode is shown below.

- The compare match interrupt (INTM001/010/011) of timer C (channels 0 and 1) does not occur during A/D conversion, and the external interrupt signal (INTP001/010/011) is not input during A/D conversion.

[Workaround]

Since the conversion result is correct even when this bug occurs, the affect of this bug is small when obtaining the latest conversion value. If the re-conversion causes any problems, start A/D conversion in the A/D trigger mode by setting the ADCE bit of the ADM0 register to 1 in the interrupt service routine of the timer match interrupt.

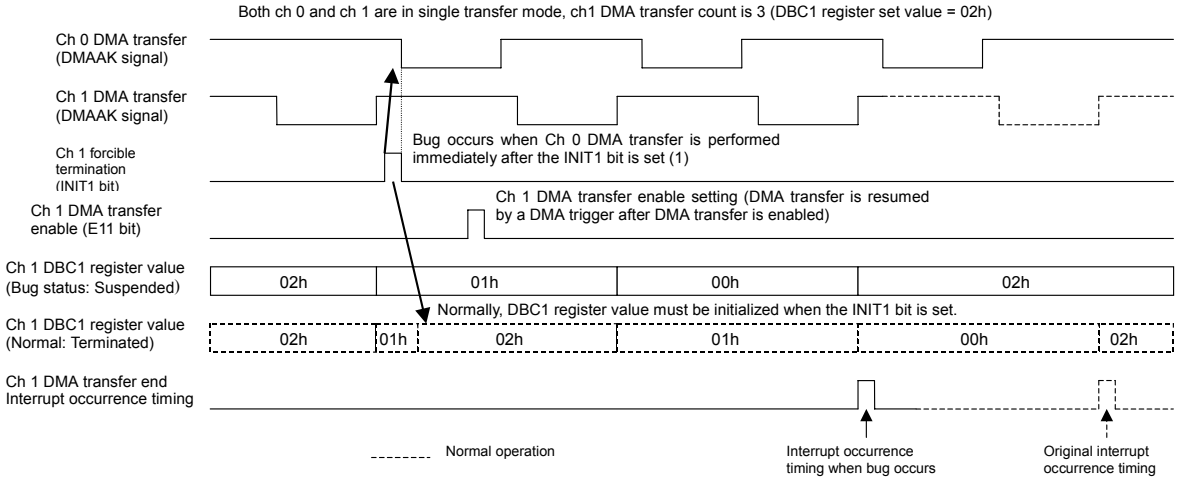
No.4 Bug related to DMA transfer forcible termination

[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.



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.5 Bug that DMA transfer is forcibly suspended by NMI

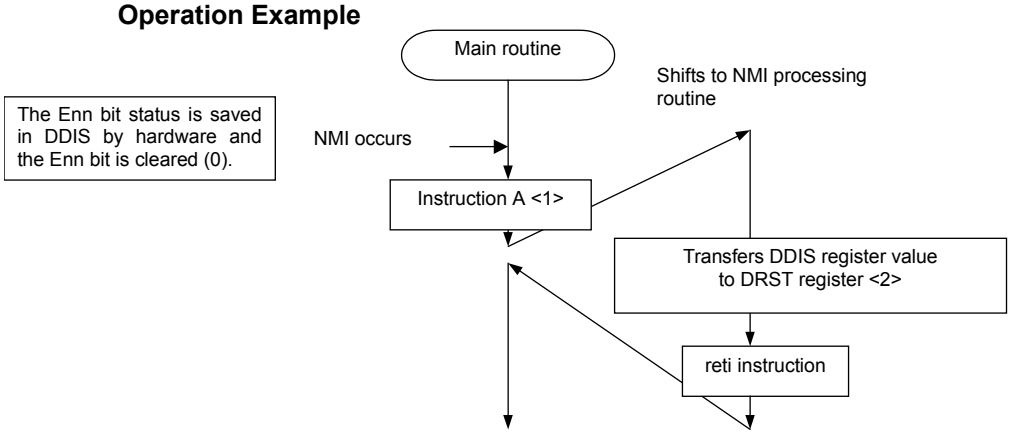
[Description]

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



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.6 Bug in 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.7 Bug related to DMA transfer whose transfer count is set to two

[Description]

With a DMA channel whose remaining transfer count (including the initial setting) is two (DBCn register value = 0001H), if the Enn bit of the DCHCn register is set (1) immediately before a DMA transfer cycle by flyby occurs, or if the Enn bit of the DCHCn register is cleared (0) immediately before the first flyby DMA transfer cycle is activated in block transfer mode in which the total transfer count is set to two, DMA transfer may be performed three times in total until the subsequent series of DMA transfers is complete. (n = 0 to 3)

In the bug operation, the transfer is performed on the address one cycle ahead of the address at which the transfer should have been complete (DMA transfer is performed three times whereas it is only set to two). The settings of other DMA channels and their operating status do not affect the bug occurrence conditions.

Under the above bug occurrence conditions, setting the Enn bit of the DCHCn register (1) means rewriting the Enn bit that has already been set (unnecessary operation). In the same manner, a clear operation (0) means that an enabled DMA transfer in block transfer mode for which the total transfer count is set to two is disabled before it is activated (unnecessary operation).

[Workaround by software]

When the Enn bit of the DCHCn register is set (1), do not set or clear the Enn bit until the transfer count of the DMA transfer set in the DBCn register is complete, or until DMA transfer is forcibly terminated using the INITn bit of the DCHCn register.

No.8 Bug that TCn bit of DMA is not cleared automatically

[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.

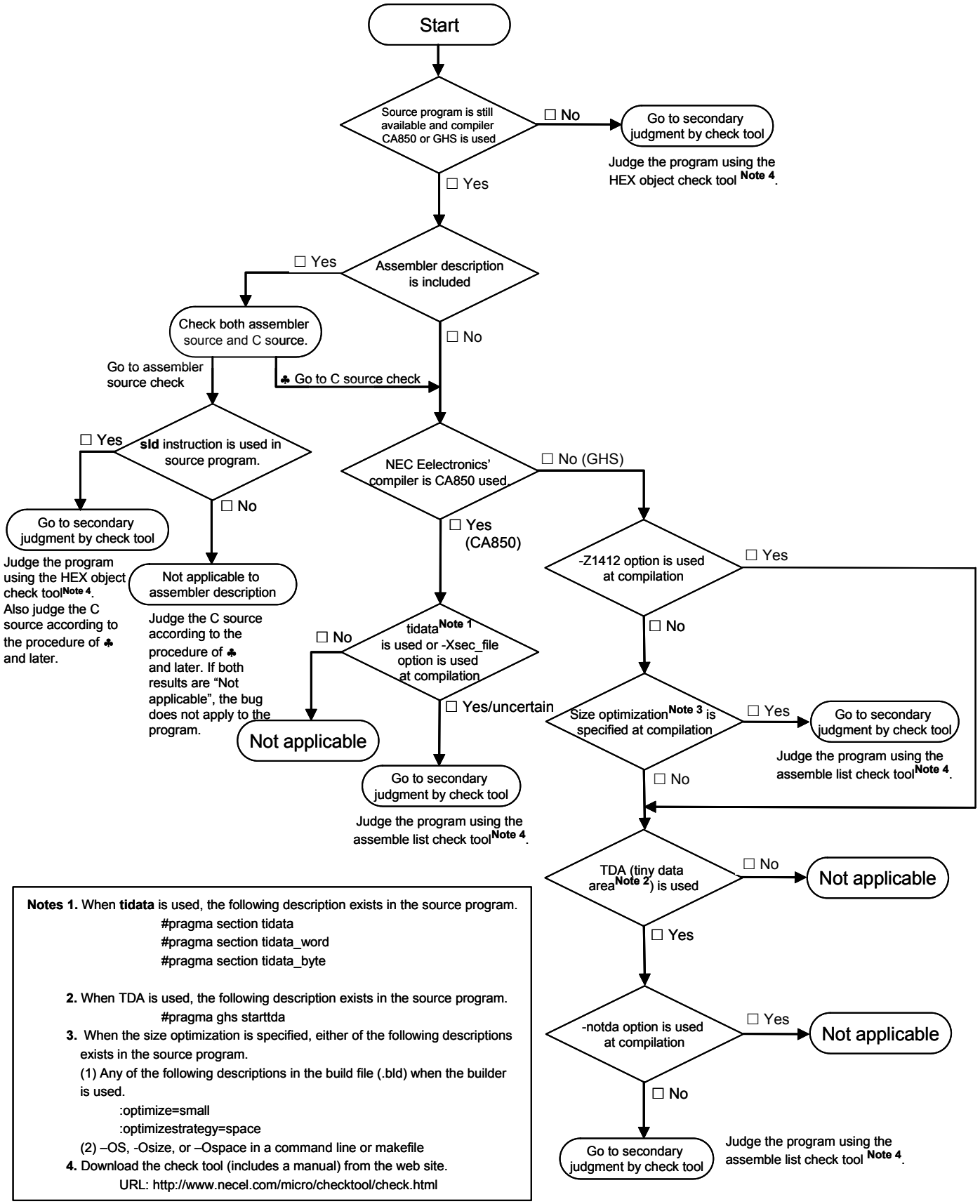
  Inquiry:

  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.



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