To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: http://www.renesas.com

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)
Send any inquiries to http://www.renesas.com/inquiry.
HITACHI SEMICONDUCTOR TECHNICAL UPDATE

<table>
<thead>
<tr>
<th>Classification of Production</th>
<th>Development Environment</th>
<th>No</th>
<th>TN-CSX-040A/E</th>
<th>Rev</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>THEME</td>
<td>SuperH RISC engine C/C++ Compiler Ver.7.0 bug report (4)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Classification of Information</td>
<td></td>
<td>1. Spec change</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. Supplement of Documents</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3. Limitation of Use</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4. Change of Mask</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>5. Change of Production Line</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PRODUCT NAME</th>
<th>Lot No.</th>
<th>Reference Documents</th>
<th>Effective Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0700CAS7-MWR</td>
<td>All</td>
<td>SuperH RISC engine C/C++ Compiler, Assembler, Optimizing Linkage Editor User’s Manual ADE-702-246A Rev.2.0</td>
<td>Eternity</td>
</tr>
<tr>
<td>P0700CAS7-SLR</td>
<td>All</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P0700CAS7-H7R</td>
<td>All</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Attached is the description of the known bugs in Ver. 7.0 series of the SuperH RISC engine C/C++ compiler. Inform the customers who have the package version in the table below of the bugs.

<table>
<thead>
<tr>
<th>Package version</th>
<th>Compiler version</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.0B</td>
<td>7.0B</td>
</tr>
<tr>
<td>7.0.01</td>
<td>7.0.03</td>
</tr>
<tr>
<td>7.0.02</td>
<td>7.0.04</td>
</tr>
<tr>
<td>7.0.03</td>
<td>7.0.06</td>
</tr>
<tr>
<td>7.0B</td>
<td>7.0B</td>
</tr>
<tr>
<td>7.0.02</td>
<td>7.0.03</td>
</tr>
<tr>
<td>7.0.03</td>
<td>7.0.04</td>
</tr>
<tr>
<td>7.0.04</td>
<td>7.0.06</td>
</tr>
<tr>
<td>7.0B</td>
<td>7.0B</td>
</tr>
<tr>
<td>7.0.02</td>
<td>7.0.03</td>
</tr>
<tr>
<td>7.0.03</td>
<td>7.0.04</td>
</tr>
<tr>
<td>7.0.04</td>
<td>7.0.06</td>
</tr>
</tbody>
</table>

The checker of the bugs is on the URL below for downloading.


Attached: P0700CAS7-020715E
SuperH RISC engine C/C++ Compiler Ver. 7
Known bugs in this release (4)
SuperH RISC engine C/C++ Compiler Ver. 7
Known Bugs in This Release (4)

The known bugs in the ver.7.0 series of the SuperH RISC engine C/C++ compiler are listed below.

1. Deleting an assignment to the register for which #pragma global_register has been specified

[Contents]
An assignment to the register for which #pragma global_register has been specified may be illegally deleted.

[Conditions]
This problem may occur when all of the following conditions are satisfied;
(1) #pragma global_register is specified.
(2) The optimize=1 option is specified.
(3) The variable specified in (1) is defined or used in a single expression such as a compound assignment expression.

<Example>
#pragma global_register(a=R14)
:
a += b; // or a=a+b;

[How to avoid the bug]
The bug can be avoided with either method of the following;
(1) Cancel the specification of #pragma global_register.
(2) Specify the optimize=0 option.

2. Illegal comparison of a 32-bit bitfield

[Contents]
Before comparing a 32-bit bitfield to 0, an AND operation with 0 may be illegally generated. The result will always be true (or false).

[Example]
struct ST {
    unsigned int b: 32;
};

void f(struct ST *x) {
    if (x->b) {
        :
    }
}
: MOV.L @R4,R0
    AND #0,R0 ; ANDed with 0.
    TST R0,R0 ; Always true.
    BF L12 ; A branch to L12 will not occur.
:

[Conditions]
This problem may occur when all of the following conditions are satisfied;
(1) There is a 32-bit field member in a structure.
(2) The relevant member is compared to 0 (== or !=).

[How to avoid the bug]
The bug can be avoided with the following method;
(1) Set the 32-bit bitfield to be of the integer type.
    <Example>
    struct ST {
        unsigned int b;
    }

3. Illegal instruction to move stacks while the trapa_svc function is in use
[Contents]
If pic=1 is specified for compilation, an illegal instruction to move stacks may be generated when loading the address of the function that uses the intrinsic function trapa_svc.

[Example]
#include <machine.h>
extern char *b(void (*yyy)(char));

void y(char c) {
    trapa_svc(160, 10, c);
}

char *a(void) {
    return b(y);
}
_a:
    MOV.L       L14,R4  ; _y-L12
    MOVA        L12,R0
    ADD         R0,R4

L12:
    MOV.L       @R15+,R0  ; <- Unnecessary instruction to move stacks
    MOV.L       L14+4,R2  ; b-L13
    MOVA        L13,R0
    ADD         R0,R2

L13:
    JMP         @R2
    MOV.L       @R15+,R0  ; <- Unnecessary instruction to move stacks

[Conditions]
This problem may occur when all of the following conditions are satisfied;
(1) An option other than cpu=sh1 is specified.
(2) The pic=1 option is specified.
(3) The intrinsic function trapa_svc is in use.
(4) The location of loading the address that uses the trapa_svc function comes later than the location
    where the trapa_svc function is called.

[How to avoid the bug]
This problem can be prevented by the following method;
(1) Change the order of the definitions so that the location of loading the address of the function
    that uses the trapa_svc function within comes before the location where the trapa_svc function is
called.
<Example>
#include <machine.h>
extern char *b(void (*yyy)(char));
void y(char c);    // Prototype declaration

char *a(void) {
    return b(y);    // Loading of the address of the y function comes before calling of
                     the trapa_svc function
}
void y(char c) {
    trapa_svc(160, 10, c);
}
4. Illegal movement of a copy instruction for R0-R7

[Contents]
Due to an optimization, a copy instruction or an extended instruction for R0-R7 may be illegally moved beyond the range of calling functions. The result of CMP/EQ (TST) may be incorrect.

[Example]
```
: MOV.L @R4,R2
MOV R5,R14
MOV #1,R5      ; H'00000001
ADD #24,R2
MOV.L @R2,R6
EXTU.W R14,R1  ; The definition of R1 moves beyond JSR
MOV.L @(8,R2),R7
JSR @R7        ; R1 may be damaged at the callee
ADD R6,R4
MOV #4,R2      ; H'00000004
CMP/EQ R2,R1   ; Compares by using the value of damaged R1
EXTU.W R0,R0   
BF L16
:
```

[Conditions]
This problem may occur when all of the following conditions are satisfied;
(1) The optimize=1 option is specified.
(2) The program includes conditional branches and calling of functions.
(3) An optimization has been performed as follows;
   <Before an optimization>
   MOV   R0, Rn   ; or EXTU   R0,Rn
   :
   MOV   Rx, R0   ; or EXTU   Rx,R0
   TST   #imm, R0 ; or CMP/EQ #imm,R0
   MOV   Rn, R0   ; or EXTU   Rn,R0
   <After an optimization>
   MOV   Rx, Rm   ; or EXTU   Rx,Rm
   MOV   #imm, Ry
   TST   Ry, Rm   ; or CMP/EQ   Ry,Rm
(4) In the optimization mentioned in (3), the Rm register is R0-R7. No other instruction in this function uses this register.
(5) Due to another optimization, the MOV Rx,Rm (or EXTU) function, which has been generated by the optimization mentioned in (3), is moved beyond the range of calling functions.
[How to avoid the bug]

We distribute a checker to check if the this bug exists. If found, problems can be avoided with the following method;

(1) Specify the optimize=0 option.

The tool can be downloaded from the following URL;


This checker also checks if the optimization mentioned in the third condition has been performed. Thus even the functions that are not concerned with this bug may be detected.