RENESAS Tool News

RENESAS TOOL NEWS on February 20, 2009: 090220/tn1

Notes on Using the C/C++ Compiler Package V.9 for the SuperH RISC engine Family

Please take note of the following problems in using the C/C++ compiler package for the SuperH RISC engine family of MCUs:

- With requesting an interrupt in a function that calls another function (SHC-0078)
- With assigning the result of two or more tilde (~) operations (bitwise 1's complement operations) to a variable of specific types (SHC-0079)

---

1. Problem with Requesting an Interrupt in a Function That Calls Another Function (SHC-0078)

Versions Concerned:
V.9.00 Release 00 through V.9.02 Release 00

Symptom:
If an interrupt is requested in a function that calls another during the period from when the stack area used by the function is released to when the function returns to the calling function, values saved on another stack are rewritten, and these incorrect values may be referenced.

Conditions:
This problem may arise if the following conditions are all satisfied:
(1) The optimize=1 option is selected.

(2) A function calls another.

(3) In the function in (2), a stack area is used and released.

(4) In the function in (2), none of the registers from R8 through R14 is referenced.

(5) In the function in (2) exists at least any one of the following instructions, where n represents any number from 1 to 7:
- MOV R15,Rn
- MOV.L R15,@-Rn
- MOV.L R15,@Rn
- MOV.L R15,@(R0,Rn)
- MOV.L R15,@(disp,Rn)
- MOV.L R15,@(disp12,Rn)
- ADD R15,Rn

(6) In the function in (2), a formal parameter or automatic variable that is saved on the reserved stack area is referenced.

(7) The function in (2) returns to the calling function by the RTS instruction.

(8) An interrupt is requested during the period from when the stack area is released in the function in (2) to when the RTS instruction in (7) is executed.

(9) In the delay slot of the RTS instruction in (7) exists the LOAD instruction from the stack area released in (3) to R0 or FR0.
Note that this condition is satisfied only when the operand of the return statement within the function in (2) takes a parameter or local variable.
If your compiler is any one of those from V.9.00 Release 00 through Release 04A, and the operand of the return statement is a scalar-type variable, this condition is never satisfied.

Example:

```
-----------------------------------------
int *p;
int func()
{
   int array[2]={0};

   sub();          // Function called by function in Condition (2).
   p = &(array[1]);   // Condition (6)
   return (*p);
}
-----------------------------------------
```

Results of compilation:

```
_func:
        STS.L   PR,@-R15
        ADD     #-8,R15         ; Condition (3):
                                ; Stack area reserved.
        MOV.L   L11+2,R5        ; L12
        MOV.L   L11+6,R4        ; _sub
        MOV.L   @R5,R1
        MOV.L   @(4,R5),R2
        MOV.L   R1,@R15         ; array[]
        JSR     @R4
        MOV.L   R2,@(4,R15)     ; array[]
        MOV     R15,R7          ; Condition (5)
                                ; R7 = Value pointed
                                ; to by stack pointer
                                ; before stack area
                                ; released.
        MOV.L   L11+10,R6       ; _p
        ADD     #4,R7
        MOV.L   R7,@R6          ; p
        ADD     #8,R15          ; Condition (3) Stack
                                ; area released -> A
        LDS.L   @R15+,PR
        RTS                     ; Condition (7) -> B
        MOV.L   @R7,R0          ; *(p) Condition (9)
```

NOTE: If an interrupt is requested between A and B above, Condition (8) is satisfied.

Workarounds

Avoid this problem in either of the following ways:

(1) Before and after the function in Condition (2) is called,

disable interrupts.
(2) Use optimize=0 or optimize=debug_only instead of optimize=1.

2. Problem with Assigning the Result of Two or More tilde (~) Operations (Bitwise 1's Complement Operations) to a Variable of Specific Types (SHC-0079)

Versions Concerned:
V.9.00 Release 00 through V.9.02 Release 00

Symptom:
If the right term of an assignment statement is the result of two or more tilde (~) operations and the left term is a variable of type signed char, unsigned char, signed short, or unsigned short, and the variable of the left term is used after the assignment statement, the variable may not be converted into the data type of the assignment statement and the value of the right term may be used instead.

Conditions:
This problem may arise if the following conditions are all satisfied:
(1) A internal or external variable of type signed char, unsigned char, signed short, or unsigned short is in the left term of an assignment statement.

(2) A tilde operation in a fanction is performed on another variable or statement.

(3) On the result of tilde operation in (2) another tilde operation is performed to assign the result to the variable in (1).

(4) Between two tilde operations in (2) and (3) the result of tilde operation in (2) does not make any changes.

Example1:

```
------------------------------------------------------------------
unsigned char t = ~~(X+a);      // Condition (1), (2), (3), and (4)
------------------------------------------------------------------
```

Example2:

```
------------------------------------------------------------------
unsigned char t = ~(~X);      // Condition (1), (2), (3), and (4)
------------------------------------------------------------------
```

Example3:

```
----------------------------------------------------------------
unsigned char t;
unsigned char temp;
temp = ~X;              // Condition (2)
     :                  // Condition (4)
                          No change made to temp.
t = ~temp;              // Condition (1) and (3)
----------------------------------------------------------------
```

(5) Either of the following is satisfied:

a. The value of the expression performed before the tilde operation in (3) are performed is a value that cannot be presented by the type of the variable in (1), to which an assignment is made.

b. The result of the tilde operation in (3) is a value that cannot be presented by the type of the variable in (1), to which an assignment is made.

Here, "a value that cannot be presented by the type of the variable" is any of the following:

Type signed char: Less than -128 or greater than 127
Type unsigned char: A negative value or greater than 255
Type signed short: Less than -32,768 or greater than 32,767
Type unsigned short: A negative value or greater than 65,535

(6) After the assignment statement in (1), the variable to which this assignment has been made is referenced. Note, however, that the case where the above variable is assigned to a variable of the same type or a variable with a smaller data size is excluded from this condition.

Example:

```
-----------------------------------------------
int X = -1;             // Condition (3)
int Y;

func()
{
   unsigned char t = ~~X; // Conditions (1) and (2)
   Y = t;
}
-----------------------------------------------
```

Results of compilation:

```
-----------------------------------------
    MOV.L     L11+2,R1   ; _X
    MOV.L     L11+6,R4   ; _Y
    MOV.L     @R1,R2     ; X
    RTS
    MOV.L     R2,@R4     ; Y  Though to be 255, Y is set to -1.
-----------------------------------------
```

Workaround:

Avoid this problem in either of the following ways:

(1) Assign the result of tilde operation in Condition (2) to a volatile-qualified variable. Then use this volatile-qualified variable.

Example:

```
-----------------------------------------
volatile unsigned char temp = ~X;
unsigned char t = ~temp;
Y = t;
-----------------------------------------
```

(2) Delete the unnecessary tilde operations out of those in Condition (2) and (3).

Example:

```
-----------------------------------------------------------------
unsigned char t = X;   // Two tildes (~~) deleted
Y = t;
-----------------------------------------------------------------
```

3. Schedule of Fixing the Problem

We plan to fix this problem in the C/C++ compiler package V.9.03 Release 00 for the SuperH RISC engine family.