# RENESAS Tool News

**RENESAS TOOL NEWS on October 20, 2011: 111027/tn1**

## A Note on Using the C/C++ Compiler Package for the SuperH RISC engine MCU Family V.9

When using the C/C++ compiler package V.9 for the SuperH RISC engine family of MCUs, take note of the following problems:

- With using the -global_volatile=1 option (SHC-0082)
- With referencing the address of the function whose name is the same as that of a local static variable (SHC-0083)
- With performing bitwise OR (|) operation of -1 or bitwise AND (&) operation of 0 with a function call or a volatile-qualified variable (SHC-0084)
- With executing a loop containing the conditional operator or an if statement with a loop counter (SHC-0085)
- With updating the value of the variable referenced in a loop containing the conditional operator or an if statement by using any statement except an assignment one (SHC-0086)
- With using the function that accesses both a parameter to be saved on the stack and the stack area reserved in the function within the same control block (SHC-0087)
- With referencing an array element of type char twice or more times in a function (SHC-0088)

Here, SHC-XXXX at the end of each item is a consecutive number for indexing the problem in the compiler concerned.

---

### 1. Problem with Using the -global_volatile=1 Option (SHC-0082)

Versions Concerned:
V.9.00 Release 00 through V.9.04 Release 00

Description:
If the -global_volatile=1 option is used, incorrect values may be stored on registers R0-R7 or FR0-FR11, and the program not be operated properly.

Conditions:

This problem may arise if the following conditions are all satisfied:

(1) Neither option -optimize=0 nor -optimize=debug_only is selected.

(2) Option -global_volatile=1 is selected.

(3) Option -noscope is not selected.

(4) In the program exist a number of functions whose optimizing ranges are divided.
Here, the following message is dispatched when the -message option is selected:
C0101 (I) Optimizing range divided in function "function name"

Example:

```
-------------------------------
. . . . .
 member[9].func(0x1000, -1, 0);
. . . . .
-------------------------------
```

Correct results of compilation:

```
-----------------------------------------------------------------------
 MOV       #72,R1
 SHLL2     R1
 MOV       #16,R5
 ADD       R1,R12
 MOV       #0,R7     ; On 4th argument of func(), 0 stored.
 AND       R11,R2
 MOV       #-1,R6    ; On 3rd argument of func(), -1 stored.
 SHLL8     R5        ; On 2nd argument of func(), 0x1000 stored.
 LDS       R2,FPSCR
 MOV.L     L28+10,R1  ; _func__5ClassFUiiT1
 JSR       @R1
 MOV       R12,R4    ; On 1st argument of func() (this pointer),
                     &member[9] stored.
-----------------------------------------------------------------------
```

If the problem arises, not &member[9] but an indefinite value 0x120 is stored on R4. As a result, "this pointer" points to an incorrect address within the func().

Results of compilation where problem arises:

```
-----------------------------------------------------------------------
 MOV       #72,R4
 MOV       #16,R5
 SHLL2     R4        ; On 1st argument (this pointer) of func(),
                     0x120 stored in error.
 ADD       R12,R1
 MOV       #0,R7     ; On 4th argument of func(), 0 stored.
```

```
AND        R11,R2
MOV        #-1,R6     ; On 3rd argument of func(), -1 stored.
SHLL8      R5         ; On 2nd argument of func(), 0x1000 stored.
MOV.L      L28+8,R1   ; _func__5ClassFUiiT1
JSR        @R1
LDS        R2,FPSCR
```

---

Workaround

To avoid this problem, use any of the following:

(1) Use option -optimize=0 or -optimize=debug_only.

(2) Use option -global_volatile=0.

(3) Use option -noscope.

(4) Divide the function in Condition (4) until the message C0101 (I) is not dispatched even if the -message option is selected.

## 2. Problem with Referencing the Address of the Function Whose Name is the Same as That of a Local Static Variable (SHC-0083)

Versions Concerned:

V.9.00 Release 00 through V.9.04 Release 00

Description:

If a static variable declared in a function is referenced, and then the address of another function with the same name as the variable is referenced in the same file, not the address of the function but that of the variable may be referenced.

Conditions:

This problem may arise if the following conditions are all satisfied:

(1) Function A references a static variable declared.

(2) In the file where function A is defined exists function B, and in function B, the reference to the address of function C with the same name as the static variable in (1) or a function call is made.

(3) The reference to the address of function C or the function call in (2) is made after the reference to the static variable in (1). This condition includes the case where functions A and C are the same.

Example:

---

```
extern void clock(void);
void foo(void)
 {
   static int clock = 1; // Condition (1)
   clock++;
 }
void foo2(void)
 {
   clock();      // Conditions (2) and (3)
 }
 ------------------------------------------------
```

Results of compilation:

```
 ------------------------------------------------------------------
  _foo2:
     MOV.L      L12,R2   ; __$clock$2   ; References static variable
                                        in error.
     JMP        @R2
     NOP
 ------------------------------------------------------------------
```

Workaround:

To avoid this problem, use any of the following:

(1) Give a different name to function C in Condition (2) from the static variable in Condition (1).

(2) Assign the address of function C to a volatile variable before the reference to the static variable in Condition (1).

(3) Change the order of placing functions so that Condition (3) cannot be met. For instance, interchange foo() with foo2() in Example above.

Example modified where Workaround (2) used:

```
 ------------------------------------------
 extern void clock(void);

//// Add the following block ////
 void dummy(){
    volatile FUN_t f = &clock;  // Address of function (C) in
                                Condition (3) assigned to
                                volatile variable
    f();
 }
//// Block added ends ////

 void foo(void)
```

```
{
   static int clock = 1; // Condition (1)
   clock++;
}

void foo2(void)
{
   clock();      // Conditions (2) and (3)
}
```
---

### 3. Problem with Performing Bitwise OR (|) Operation of -1 or Bitwise AND (&) Operation of 0 with a Function Call or a Volatile-Qualified Variable (SHC-0084)

Versions Concerned:
V.9.02 Release 00 through V.9.04 Release 00

Description:
If bitwise OR (|) operation of -1 or bitwise AND (&) operation of 0 with a function call or a volatile-qualified variable is performed, the function may not be called or the volatile-qualified variable not be referenced.

Conditions
This problem may arise if the following conditions are all satisfied:
(1) Bitwise OR (|) operation or bitwise AND (&) operation is performed whose operand is of any type of the following:
long long, signed long long, and unsigned long long
(2) The operation in (1) satisfies either of the following:
(2-1) An operand of bitwise OR operation is -1.
(2-2) An operand of bitwise AND operation is 0.
Here, the operand can be a constant substituted for a variable by optimization of constant propagation (including that of external constants qualified to be const).
(3) The other operand of the operation in (2) is any of the following:
(3-1) A variable qualified to be volatile
(3-2) An external variable with the -volatile=1 option used
(3-3) A function call
(3-4) An expression containing any of the operands listed in (3-1), (3-2), and (3-3)

Example:

```
------------------------------------------------------------
long long a;
int sub();
main(){
  long long x;
  x = -1LL;                    // Condition (2-1)
  a = ((long long)(sub()+2)) | x;  // Conditions (1), (2-1), (3-4)
}
------------------------------------------------------------
```

If the above example is compiled, the expression

```
          ((long long)(sub()+2)) | -1LL
```

is deleted in error, and the function call sub() is not made.

Results of compilation:

```
----------------------------------------
 _main:
     MOV.L      L11+2,R6   ; _a
     MOV        #-1,R2     ; H'FFFFFFFF
     MOV.L      R2,@R6     ; part of variable a
     RTS
     MOV.L      R2,@(4,R6) ; part of variable a
----------------------------------------
```

Workaround:

To avoid this problem, assign constant -1 or 0 in Condition (2) to a volatile-qualified variable; then use it instead of the constant.

Example modified:

```
------------------------------------------------------------
  long long a;
  int sub();
  main(){
    volatile long long x;  // Volatile-qualified variable used
                           instead of constant in Condition (2).
    x = -1LL;                    // Condition (2-1)
    a = ((long long)(sub()+2)) | x;  // Conditions (1) and (3-4)
  }
------------------------------------------------------------
```

**4. Problem with Executing a Loop Containing the Conditional Operator or an if Statement with a Loop Counter (SHC-0085)**

Versions Concerned:
V.9.00 Release 00 through V.9.04 Release 00

Description:
If a loop contains the conditional operator (?:) or an if expression including a loop counter, the conditional operator or the if statement may be incorrectly evaluated.

Here, a loop counter is a variable for controlling the iterations of a loop. It is incremented or decremented by a fixed integer value on each iteration and evaluated to decide when the iteration should be terminated.

Conditions:
This problem may arise if the following conditions are all satisfied:
(1) neither option -optimize=0 nor -optimize=debug_only is selected.
(2) In the program exists a loop containing a loop counter.
(3) The initial and maximum values of the loop counter in (2) are constants.
Here, constants can be those substituted for variables by optimization of constant propagation (including that of external constants qualified to be const).
(4) In the loop in (2) exists the conditional operator (?:) or an if statement.
(5) The controlling expression of the conditional operator or the if statement in (4) satisfies all the following:
(5-1) It is a comparison expression containing a relational operator <, >, <=, or >=.
(5-2) One operand is the loop counter in (2).
(5-3) The other operand is an integer constant equal to or smaller than the maximum value of the loop counter.
Here, an integer constant can be the one substituted for a variable by optimization of constant propagation (including that of external constants qualified to be const).

Example:
```
------------------------------------------------------------
 int main(void) {
   int  j;
   char a1[6],a2[6],a3[6];
   for ( j = 0; j < 6; j++ ){  // Conditions (2), (3), (4)
     if ((j >= 0 && j < 2) || (j >= 4 && j < 6))  // Condition (5)
       a1[j] = 1;
```

```
    else
      a1[j] = 0;
    if (j < 4)    // Condition (5)
      a2[j] = 2;
    else
      a2[j] = 0;
    if (j >= 0 && j < 1)    // Condition (5)
       a3[j] = 3;
    else
      a3[j] = 0;
  }
 }
```

---

If the above example is compiled with the -cpu=sh4a -speed option used, the problem arises at the expression "j < 2," which exists in the first Condition (5).

Results of compilation:

```
    MOV       #0,R6     ; H'00000000
    MOV       #2,R1     ; H'00000002
    MOV       R15,R4
    MOV       R15,R5
    ADD       #8,R4
    MOV       R1,R13
    ADD       #16,R5
    MOV       R6,R7
    MOV       R15,R2
    MOV       #1,R12    ; H'00000001
    MOV       #3,R9     ; H'00000003
    MOV       #6,R14    ; H'00000006
    MOV       #4,R10    ; H'00000004
L11:
    CMP/PZ    R6
    MOV.B     R12,@R4   ; a1[] . . . (A) a1[3] is set to 1 in error.
    BF/S      L12
    MOV.B     R13,@R2   ; a2[]
    CMP/GE    R12,R6
    BT        L12
    MOV.B     R9,@R5    ; a3[]
L15:
    ADD       #1,R6
    CMP/GE    R14,R6
    ADD       #1,R5
    ADD       #1,R4
```

```
    BT/S       L17
    ADD        #1,R2
    DT         R1        ; (B) If j(R6) is 3, R1 becomes -1.
    BF         L11       ; (C) Jumps to L11 in error.
    CMP/GE     R10,R6
    BT         L20
    MOV.B      R7,@R4    ; a1[] . . . (D) a1[3] should be set to 0.
    BRA        L12
    MOV.B      R13,@R2   ; a2[]
L20:
    CMP/GE     R14,R6
    BF         L22
    BRA        L23
    MOV.B      R7,@R4    ; a1[]
L22:
    MOV.B      R12,@R4   ; a1[]
L23:
    MOV.B      R7,@R2    ; a2[]
L12:
    BRA        L15
    MOV.B      R7,@R5    ; a3[]
L17:
. . . . . .
```

---

In the above results of compilation, if j(R6) is 3, R1 becomes -1 at (B). So the program jumps to L11 at (C), and a1[3] is set to 1 at (A) in error.

If correctly executed, the program does not jump at (C), and a1[3] is set to 0 at (D).

Workaround:

To avoid this problem, use either of the following:

(1) Use option -optimize=0 or -optimize=debug_only.

(2) Qualify the loop counter in Condition (2) to be volatile.

## 5. Problem with Updating the Value of the Variable Referenced in a Loop Containing the Conditional Operator or an if Statement by Using Any Statement except an Assignment One (SHC-0086)

Version Concerned:

V.9.04 Release 00

Description:

Suppose that the value of the external or static variable is referenced in a loop containing a conditional operator or an if statement. If the value of the above variable is updated by using any statement except the one assigned to the variable, after the loop is exited, the variable may resume the value before updated.

Conditions:

This problem may arise if the following conditions are all satisfied:

(1) Neither option -optimize=0 nor -optimize=debug_only is selected.

(2) Option -noscope is not selected.

(3) In the program exist a number of functions whose optimizing ranges are divided.
Here, the following message is dispatched when the -message option is selected:
C0101 (I) Optimizing range divided in function "function name"
NOTE: The function specified in C0101 (I) is hereafter called the function in Condition (3)

(4) In the function in (3) exists a loop containing the conditional operator (?:) or an if statement.

(5) The loop in (4) does not have any loop inside of it and is not an infinite loop.

(6) Declared is an external or static variable that satisfies the following:
(6-1) In the loop in (4), it is neither defined nor referenced.
(6-2) In the function in (3), it is defined or referenced.

(7) The variable in (6) is not qualified to be volatile, and the -global_volatile=1 option is not used.

(8) The external or static variable in (6) is updated in the loop in (4) by using any statement except the one assigned to the variable. (An example of the above statement is a function call.)

Example:

```
-------------------------------------------------
int aaa,xxx=0,yyy=0,n;       // Condition (7)
void sub(void);
void func(void)              // Condition (3)
{
 . . . . . .
  aaa = 0;                   // Condition (6)
 . . . . . .
  for (i = 0; i < n ;i++) {   // Conditions (4) and (5)
    if(xxx == yyy){
```

```
     sub();              // Condition (8)
    }
   }
  . . . . . . . . . . . . . . . . . . .
 }
 void sub(void)
 {
  aaa++;
 }
```

---

Results of compilation:

---

```
 _func:
 . . . . . . . . . . . .
     MOV.L      @R9,R13    ; aaa . . . (A)
     MOV.L      L17+8,R12  ; _n
     MOV.L      L17+12,R10 ; _yyy
     MOV.L      L17+16,R11 ; _xxx
     BRA        L11
     MOV        #0,R14     ; H'00000000
 L12:
     MOV.L      @R10,R6    ; yyy
     MOV.L      @R11,R2    ; xxx
     CMP/EQ     R6,R2
     BF         L14
     BSR        _sub       ; aaa++; . . . (B)
     NOP
 L14:
     ADD        #1,R14
 L11:
     MOV.L      @R12,R2    ; n
     CMP/GE     R2,R14
     BF         L12        ; if (i < n)
     MOV.L      R13,@R9    ; aaa . . . (C)
  . . . . . . . . . .
 _sub:
     MOV.L      L17,R6     ; _aaa
     MOV.L      @R6,R2     ; aaa
     ADD        #1,R2
     RTS
     MOV.L      R2,@R6     ; aaa
```

---

In the above results of compilation, the value of aaa loaded into R13 from aaa at (aA) is written from aaa to R13 at (C). So when the

loop is exited, the value of aaa incremented at (B) resumes the value held at immediately before the loop begins.

Workaround:

To avoid this problem, use any of the following:

(1) Use option -optimize=0 or -optimize=debug_only.
(2) Use option -noscope.
(3) Use option -global_volatile=1.
(4) Qualify the variable in Condition (6) to be volatile.
(5) Reference the variable in Condition (6) within the loop in Condition (4).
(6) Divide the function in Condition (3) until the message C0101 (I) is not dispatched even if the -message option is selected.

## 6. Problem with Using the Function That Accesses Both a Parameter to Be Saved on the Stack and the Stack Area Reserved in the Function within the Same Control Block (SHC-0087)

Versions Concerned:

V.9.00 Release 00 through V.9.04 Release 00

Description:

If a function accesses both a parameter to be saved on the stack and the stack area reserved in the function within the same control block, incorrect access may be made.

Conditions:

This problem may arise if the following conditions are all satisfied:

(1) Neither option -optimize=0 nor -optimize=debug_only is selected.
(2) Option -cpu=sh2afpu, -cpu=sh4, or -cpu=sh4a is selected.
(3) Option -fpu=single is not selected.
(4) A function takes a parameter to be saved on the stack.
(5) The type of the parameter in (4) is any of the following and is not qualified to be volatile:
  (5-1) double
  (5-2) long double
  (5-3) structure or union one of whose members is of type double or long double
(6) A stack area is reserved in the function in (4) for other purposes than for saving and restoring registers.
  The following case meets this condition:

In the compile list, the value of the .STACK instruction of the function in (4) is greater than the size of the stack area used to save registers at the front of the function.

(7) The variable of type float, double, or long double that has been assigned to the stack area in (6) is accessed.

(8) The parameter in (5) is referenced.

Note that when Condition (5-3) is met, Condition (8) is met only if a member of type double or long double is accessed; neither the structure nor union.

(9) Between the access to the stack area in (7) and the reference in (9) exists none of the following:

if statement, switch statement, for statement, while statement, do-while statement, goto statement, and conditional operator

Example when option -cpu=sh2afpu used:

```
------------------------------------------------------------
struct tbl {
     double x[2];
    };
    void sub(float a, double b);
    void func(struct tbl s){     // Conditions (4) and (5-3)
     float y[4];              // Condition (6)
     sub(y[3],s.x[1]);    // Conditions (7), (8), and (9)
    }
------------------------------------------------------------
```

Results of compilation:

```
------------------------------------------------------------
_func:
    .STACK      _func=16              ; Condition (6)
    ADD         #-16,R15
    FMOV.S      @(24,R15),FR6
    MOV         #28,R0     ; H'0000001C
    FMOV.S      @(R0,R15),FR7; s.x[]
    FMOV.S      FR7,FR4   ;  (A) Lower 4 bytes of s.x[1] transferred
                          in error.
    MOV.L       L11+2,R2   ; _sub
    JMP         @R2
    ADD         #16,R15
------------------------------------------------------------
```

In the above results of compilation, y[3] is not transferred to FR4 at (A), but the lower 4 bytes of s.x[1] are done.

Workaround:

To avoid this problem, use any of the following:

(1) Use option -optimize=0 or -optimize=debug_only.
(2) Use option -fpu=single.
(3) Qualify the parameter in Condition (4) to be volatile.
Example modified

```
-----------------------------------------------
  void func(volatile struct tbl s){
-----------------------------------------------
```

## 7. Problem with Referencing an Array Element of Type char Twice or More Times in a Function (SHC-0088)

Versions Concerned:
V.9.04 Release 00

Description:
If an array element of type char, signed char, or unsigned char is referenced twice or more times in a function, the address of the array element may be wrong.

Conditions:
This problem may arise if the condition group A or B is satisfied:

Condition group A
The following conditions, (A1) to (A6), are all met:
(A1) Neither option -optimize=0 nor -optimize=debug_only is selected.
(A2) An array of type char, signed char, or unsigned char; or a pointer to char, signed char, or unsigned char is declared.
(A3) An element of the array in (A2) is referenced twice or more times in a function.
Here, the reference to an element can be an indirect reference expression equivalent to an array reference.
Example of an indirect reference expression:
*(a+exp) is equivalent to a[exp]
(A4) The subscript expression indicating the array element in (A3) is an addition or subtraction expression, and one of its operands is a variable and the other is any constant except 0.
In the example of an indirect reference expression in (A3), the subscript expression is exp.
And a constant can be one substituted for a variable by optimization of constant propagation (including that of external constants qualified to be const).
(A5) The variable in (A4) is of type char, signed char, unsigned

char, short, signed short, or unsigned short.
(A6) The value of the subscript expression in (A4) is greater than the maximum value expressible in the type of the variable in (A5).

Example condition group A met:

```
---------------------------------------------------------------
char S,*ary;              // Condition (A2)
void func(char par)       // Condition (A5)
    {
     if (ary[par+1] > 10) {    // Conditions (A3) and (A4)
       S = ary[par+1];         // Conditions (A3) and (A4)
     }
     return;
    }
---------------------------------------------------------------
```

In the above example, Condition (A6) is met if par is 127.

Results of compilation:

```
---------------------------------------------------------------
_func:
   MOV.L      L13,R7     ; _ary
   ADD        #1,R4
   MOV.L      @R7,R0     ; ary
   EXTS.B     R4,R1      ; par+1 in Example is sign-extended
                            in error.
   MOV.B      @(R0,R1),R6; ary[]
   MOV        #10,R5     ; H'0000000A
   CMP/GT     R5,R6
   BF         L12
   MOV.L      L13+4,R2   ; _S
   MOV.B      R6,@R2     ; S
---------------------------------------------------------------
```

Condition group B

The following conditions, (B1) to (B6), are all met:
(B1) Neither option -optimize=0 nor -optimize=debug_only is selected.
(B2) A structure or union is declared a member of which is an array of type char, signed char, or unsigned char.
(B3) An element of the array in (B2) is referenced twice or more times in a function.
(B4) The subscript expression indicating the array element in (B3) is any of the following:
(B4-1) A variable
(B4-2) An addition expression; one of its operands is a variable

and the other is a constant.

(B4-3) A subtract expression; one of its operands is a variable and the other is a constant.

The constant in either (B4-2) or (B4-3) can be one substituted for a variable by optimization of constant propagation (including that of external constants qualified to be const).

(B5) The variable in (B4) is of type char, signed char, unsigned char, short, signed short, or unsigned short.

(B6) The offset value of the array element in (B3) from the beginning of the structure or union in (B2) is greater than the maximum value expressible in the type of the variable in (B4).

Example condition group B met:

```
---------------------------------------------------
char S;
struct {
  char dummy[127];
  char mem[8];              // Condition (B6)
} *st;                    // Condition (B2)
void func(char par)         // Condition (B5)
{
  if (st->mem[par] > 10) {    // Conditions (B3) and (B4-1)
    S = st->mem[par];         // Conditions (B3) and (B4-1)
  }
  return;
}
---------------------------------------------------
```

In the above example, Condition (B6) is met if par is equal to or greater than 1.

Results of compilation:

```
---------------------------------------------------
_func:
      MOV.L      L13,R7     ; _st
      ADD        #127,R4
      MOV.L      @R7,R0     ; st
      EXTS.B     R4,R1      ; par+127 in Example is sign-extended
                            in error.
      MOV.B      @(R0,R1),R6; st->mem[]
      MOV        #10,R5     ; H'0000000A
      CMP/GT     R5,R6
      BF         L12
```

```
        MOV.L      L13+4,R2   ; _S
        MOV.B      R6,@R2     ; S
    ------------------------------------------------------
```

Workaround:
To avoid this problem, use any of the following:
(1) Use option -optimize=0 or -optimize=debug_only.
(2) Qualify any of the following to be volatile:
- The array in Condition (A2)
- The array as a member of the structure or union in Condition (B2)
- The structure or union in Condition (B2)
- The variable as an operand of the subscript expression in (A4) and (B4)

(3) Change the type of the variable in Condition (A4) or (B4) to one different from those specified in Condition (A5) or (B5).

Example modified in condition group B:

```
    ------------------------------------------------------------
        void func(signed int par)   // char changed to signed int.
    ------------------------------------------------------------
```

(4) If Condition (B4-1) is met, convert the type of the variable that is an operand of the subscript expression to one different from those specified in Condition (B5)

Example modified:

```
    ------------------------------------------------------------
        st->mem[(signed int)par]   // par cast to signed int.
    ------------------------------------------------------------
```

## 8. Schedule of Fixing the Problems

All the above problems have already been fixed in the C/C++ compiler package V.9.04 Release 01 for the SuperH RISC engine family.
For details of the latest version, see RENESAS TOOL NEWS Document No. 111027/tn2 on the Web page at:
http://tool-support.renesas.com/eng/toolnews/111027/tn2.htm
This page will be opened on October 27, 2011.

The past news contents have been based on information at the time of publication. Now changed or invalid information may be included. The URLs in the Tool News also may be subject to change or become invalid without prior notice.