RENESAS Tool News

RENESAS TOOL NEWS on January 16, 2009: 090116/tn1

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

Please take note of the following nine problems in using the C/C++ compiler package V.9 for the SuperH RISC engine MCU family:

1. With referencing addresses in an iteration statement (SHC-0073)
2. With using two or more bit field members of 1 bit wide that are referenced and then whose values are set (SHC-0074)
3. With setting a variable to 0, 1, or -1 in an if statement (SHC-0075)
4. With referencing or setting the value of a variable to which #pragma global_register has been issued (SHC-0076)
5. With referencing bit fields of type signed long long or unsigned long long (SHC-0077)
6. With using the map option in linking (LNK-0002)
7. With performing CRC calculation (LNK-0003)
8. With overlaying sections with different alignment numbers each other (LNK-0004)
9. With using the data_stuff and nooptimize options in linking (LNK-0005)

---

1. Products and Versions Concerned

The C/C++ compiler packages for the SuperH RISC engine family
V.9.00 Release 00 through V.9.02 Release 00

2. Five Problems in C/C++ Compiler

2.1 With Referencing Addresses in an Iteration Statement (SHC-0073)

Versions concerned:

V.9.00 Release 00 through V.9.02 Release 00

Symptom:

The address of an element of an array or of a member of a structure or union is referenced by using a pointer variable within the loop of an iteration statement, an incorrect value may be used for the pointer variable.

Conditions:

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

(1) The optimize=1 option is used.

(2) A formal parameter, an automatic variable, or a static variable within a function is used as a pointer variable.

(3) The address of the pointer variable in (2) is not referenced.

(4) The address of an element of an array, or that of a member of a structure, or union is referenced by using the pointer variable in (2).

(5) The offset value from the beginning of the array, structure, or union one of whose elements or members has been referenced in (4) falls within a range from 1 to 127 bytes.

(6) In the program exists an iteration statement.

(7) Within the loop of the iteration statement in (6) exist two or more address referencing expressions that satisfy the conditions (2), (3), (4), and (5) above.

(8) Within the loop in (7) exists another expression containing the same pointer variable that is used for the address references in (7).

(9) Outside the loop in (7) are referenced no addresses that have been referenced in (7).

(10) Any of the registers R8, R9, R10, R11, R12, R13, and R14 is assigned to the pointer variable used in (8)

Example:

```
-----------------------------------------------
struct ST {
   int mem1;
   int mem2;
};
int a,b,*pa,*pb,*pc;
void func(struct ST *pst)       // Condition (2)
```

```
{
   while (a > 0) {
      sub();
      a = pst->mem1;         // Condition (8)
. . . . . . . . . . . . . . . . . . . . . . . .
      pa = &(pst->mem2);     // Conditions (4), (5), and (7)
      pb = &(pst->mem2);     // Conditions (4), (5), and (7)
. . . . . . . . . . . . . . . . . . . . . . . . . . .
      if (b > 0) {
         pc = &(pst->mem2);  // Conditions (4), (5), and (7)
      }
   }
}
-------------------------------------------
```

Results of compilation:

```
-------------------------------------------
. . . . . . . . . . . . . . . . . . . . . . . .
       MOV       R4,R13          ; Condition (10): R13 is assigned
                                   ; to variable pst.
. . . . . . . . . . . . . . . . . . . . . . . . . .
       MOV.L     @(4,R15),R14 ; Address value of pst->mem2 is
                                   ; referenced from stack on which
                                   ; no values are saved.

       MOV.L     R14,@R9       ; Assigned to pa
       MOV.L     R14,@R10      ; Assigned to pb
. . . . . . . . . . . . . . . . . . . .
       MOV       R13,R14
       ADD       #4,R14
       MOV.L     R14,@R8       ; Assigned to pc
. . . . . . . . . . . . . . . . . . . .
-------------------------------------------
```

Workarounds:

Avoid the problem in either of the following ways:

(1) Use the optimize=0 or optimize=debug_only option instead of optimize=1.

(2) Reference the address of the pointer variable described in the above conditions.

Example:

```
-------------------------------------------
. . . . . .
void func(struct ST *pst)
{
   &pst;              // Address of pointer variable reference.
   while (a > 0) {
. . . . . .
-------------------------------------------
```

2.2 With Using Two or More Bit Field Members of 1 Bit Wide That are Referenced and Then Whose Values are Set (SHC-0074)

Versions concerned:

V.9.01 Release 00 through V.9.02 Release 00

Symptom:

If two or more bit field members of 1 bit wide are referenced and then their values are set, these values may be incorrect.

Conditions:

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

(1) The optimize=1 option is used.

(2) In addition, the cpu=sh2a or cpu=sh2afpu option is used.

(3) In the program exist two or more bit field members of 1 bit wide.

(4) Among the bit field members in (3), some members are referenced and then their values are set.

(5) Among several of the members in (4), referencing members and setting their values are not performed adjacently.

Example:

```
-------------------------------------------
struct aa {
   unsigned char bf1:1;  // Condition (3)
   unsigned char bf2:1;  // Condition (3)
} aaf;

void main()
```

```
{
   unsigned char getfl1;
   unsigned char getfl2;

   // Conditions (4) and (5): bf1 is referenced.
   getfl1 = (*(volatile struct aa *)0x80FFFD00).bf1;

   // Conditions (4) and (5): bf2 is referenced.
   getfl2 = (*(volatile struct aa *)0x80FFFD00).bf2;

   // Conditions (4) and (5): bf1 is set.
   (*(volatile struct aa *)0x80FFFD00).bf1 = getfl1;

   // Conditions (4) and (5): bf2 is set.
   (*(volatile struct aa *)0x80FFFD00).bf2 = getfl2;
}
```

Results of compilation:

```
    MOV.L   L11,R5          ; H'80FFFD00
    BLD.B   #7,@(0,R5)      ; Status register is
                            set to aaf.bf1.
    MOV.B   @R5,R0          ; aaf
    TST     #64,R0          ; Value in status
                            register is
                            corrupted.
    MOVRT   R6
    BST.B   #7,@(0,R5)      ; Corrupted value in
                            status register is
                            ; restored to
                            aaf.bf1.

    BLD     #0,R6
    BST.B   #6,@(0,R5)
    RTS/N
```

Workarounds:
Avoid the problem in any of the following ways:

(1) Use the optimize=0 or optimize=debug_only option instead of optimize=1.

(2) Perform either of the following operations for all the references made in Condition (4):
(2-1) If the value of a referenced member is assigned to a variable, qualify the variable to be volatile in advance.
(2-2) Otherwise, assign the value of a referenced member to a volatile-qualified automatic variable, and then replace the reference to the member variable with that to the automatic variable.

Examples:
- Original

```
-----------------------------------------------
st.b1 &= st.b2;        // b1 is referenced and set.
-----------------------------------------------
```

- Modified

```
-----------------------------------------------
// Assign member variable to volatile-qualified variable
   vtemp; then replace reference to member variable with
   that to vtemp.
 volatile unsigned char vtemp = st.b1;
 st.b1 = vtemp & st.b2;
-----------------------------------------------
```

(3) Change the program so that referencing a member and setting its value can be adjacent for all the members in Condition (4).

Example:

```
-----------------------------------------------
  // Set bf1 immediately after referencing it.
  getfl1 = (*(volatile struct aa *)0x80FFFD00).bf1;
  (*(volatile struct aa *)0x80FFFD00).bf1 = getfl1;
  // Set bf2 immediately after referencing it.
  getfl2 = (*(volatile struct aa *)0x80FFFD00).bf2;
```

```
  (*(volatile struct aa *)0x80FFFD00).bf2 = getfl2;
-----------------------------------------
```

2.3 With Setting a Variable to 0, 1, or -1 in an if Statement (SHC-0075)

Versions concerned:

V.9.01 Release 00 through V.9.02 Release 00

Symptom:

If a variable is set to 0, 1, or -1 in the block of an if statement, a memory area where correct values have not been stored may be referenced.

Conditions:

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

(1) The optimize=1 option is used.

(2) In the program exists an if statement that satisfies any of the following:
  (2-1) In the then statement exist no expression statements.
  (2-2) In the else statement exist no expression statements.
  (2-3) The if statement have no else statement.

(3) In the if statement in (2), the part in which exists a statement contains an assignment expression only.

(4) In the assignment statement in (3), a constant of 0, 1, or -1 is assigned to a variable.

(5) Before the if statement in (2) exists an assignment expression where 0, 1, or -1 is assigned to the same variable as in (4).

(6) The combination of the constant assigned to the variable in (4) and the one assigned to the variable in (5) are as follows:

Constant in (4)	Constant in (5)
0	1
0	-1
1	0

-1 0

(7) The version number of the compiler is any of those concerned except V.9.01 Release 00 and V.9.01 Release 01, and either of the following conditions are satisfied:
(7-1) The variables to which a constant is assigned in (4) and (5) are not external ones.
(7-2) The opt_range=all or opt_range=noloop option is used.

(8) Between the assignment expression in (5) and the if statement in (2), the variables to which a constant is assigned in (4) and (5) or the memory area where the variables have been stored are referenced. At this time, correct values may have not been stored in this memory area.

Example:

```
-----------------------------------------------
int x;
void sub(int a)
{
   union U { int a; short b; } u;  // Condition (7-1)
   u.a = -1;                  // Conditions (5) and (6)
   x = u.b;                   // Condition (8)
   if (a == 0) {               // Condition (2-3)
      u.a = 0;                // Conditions (3), (4), and (6)
   }
   x += u.a;
}
-----------------------------------------------
```

Results of compilation:

```
-----------------------------------------------

   ADD     #-4,R15
   TST     R4,R4
   MOV.W   @R15,R5   ; Assigned to u.b while
                     @R15 not set yet.
   SUBC    R2,R2
   MOV     #-1,R6    ; H'FFFFFFFF
```

```
    XOR     R6,R2
    MOV.L   L16+2,R1  ; _x
    ADD     R2,R5
    MOV.L   R5,@R1    ; x
    RTS
    ADD     #4,R15
```

Workarounds:
Avoid the problem in any of the following ways:

(1) Use the optimize=0 or optimize=debug_only option instead of optimize=1.

(2) Qualify the variable in Condition (4) to be volatile.

(3) Before or after the assignment expression in Condition (3), place an include function nop().

(4) If Condition (2-1) is satisfied, place an include function nop() in the then statement.

(5) If Condition (2-2) is satisfied, place an include function nop() in the else statement.

(6) If Condition (2-3) is satisfied, add an else statement with an include function nop().

(7) If Condition (7-2) only is satisfied, use the opt_range=noblock option instead of opt_range=all and opt_range=noloop.

2.4 With Referencing or Setting the Value of a Variable to Which #pragma global_register Has Been Issued (SHC-0076)

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

Symptom:
When #pragma global_register has been issued to a variable,

the value of the variable may be referenced or set incorrectly.

Conditions:

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

(1) The optimize=1 option is used.

(2) The value of a variable to which #pragma global_register has been issued is referenced or set.

Example:

```
-----------------------------------------------
#pragma global_register val_R8=R8  // Condition (2)

int val_R8;
int x;

void sub(int xxx, int yyy)
{
   unsigned short zzz = xxx;
   if (yyy) {
      x = zzz;
   } else {
      val_R8 = zzz;            // Condition (2)
   }
}
-----------------------------------------------
```

Results of compilation:

```
-----------------------------------------------


          TST       R5,R5
          EXTU.W    R4,R8       ; val_R8(R8) is
                                  set regardless of
                                ; value of yyy.
          BT        L12
          MOV.L     L20+2,R6    ; _x
          MOV.L     R8,@R6      ; x
    L12:
```

```
        RTS
        NOP
```

Workarounds:
Avoid the problem in either of the following ways:

(1) Use the optimize=0 or optimize=debug_only option instead of optimize=1.

(2) Do not issue #pragma global_register.

2.5 With Referencing Bit Fields of Type signed long long or unsigned long long (SHC-0077)

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

Symptom:
The value of a bit field of type signed long long or unsigned long long may be referenced incorrectly. (NOTE)
NOTE: Even if "signed" has been omitted in signed long long, it is interpreted as signed long long.

Conditions:
This symptom may arise if the following conditions are all satisfied:

(1) A bit field member of type signed long long or unsigned long long is referenced.

(2) In the structure variable declared to be of type signed long long or unsigned long long in (1), the upper 32 bits are not 0s, and the lower 32 bits are 0s.

(3) The bit field member in (1) consists of any of the lower 32 bits in (2) except the lowermost bit.

Example:

```
struct tbl {
   unsigned long long bit1:32;
```

```
    unsigned long long bit2:1;
    unsigned long long bit3:29;
    unsigned long long bit4:1;
    unsigned long long bit5:1;
} s = { 1, 0, 0, 0, 0};         // Conditions (1), (2), and (3)

#include<stdio.h>
void main(void){
    if (s.bit2 != 0) {      // Condition (1)
        printf("NG¥n");
    }
}
```

Workarounds:

Avoid the problem in either of the following ways:

(1) Set any of the lower 32 bits of the bit field involved in the symptom to a nonzero value. In the above example, s = { 1, 0, 0, 0, 1}, s = { 1, 0, 1, 0, 0}, and so on.

(2) Convert the type of the bit field involved to any except signed long long and unsigned long long. In the above example, convert the types of all the members from unsigned long long to unsigned long.

3. Four Problems in Optimizing Linkage Editor

3.1 With Using the Map Option in Linking (LNK-0002)

Versions concerned:

V.9.00 Release 04A through V.9.02 Release 00

Symptom:

If the map option is used in combination with the overlay function, variables at incorrect addresses may be referenced.

Conditions:

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

(1) The map option is used in first linking and in second compiling.

(2) The start option in linking specifies the following items:

(2-1) Overlaying sections

Example: -start=(B:P),D1,D2/100
Here, B and P are section names, and (B:P) represents an overlay.

(2-2) Placing two or more sections adjacently after the overlay in (2-1)
Example (2-1) shows the data sections D1 and D2 are placed adjacently.

(2-3) Making the alignment number of a section greater than that of its preceding one among the adjacently placed sections in (2-2)
In Example (2-1), the alignment number of D1 is 2 bytes and that of D2 is 4 bytes.

Workaround:

If overlaying sections, do not use the map option.

3.2 With Performing CRC Calculation (LNK-0003)

Versions concerned:

V.9.02 Release 00

Symptom:

If an available area exists between modules in a section, or the size of the last module in a section after linking is zero, incorrect results of CRC calculation are obtained.

Conditions:

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

(1) The crc option is selected in linking.

(2) The files input by the linker are relocatable.

(3) Condition (3-1) or (3-2) below is satisfied.

(3-1) The following conditions are all satisfied:

(3-1-1) An available area exists between modules in a section.

(3-1-2) The beginning address of CRC calculation is within the available area in (3-1-1).

(3-1-3) The available area in (3-1-1) is for a data section with initial values or a program section.

(3-2) The following conditions are all satisfied:

(3-2-1) The size of the last module in a section after linking is zero.

(3-2-2) CRC calculation covers the whole section in (3-2-1), including the module whose size is zero.
(3-2-3) The section in (3-2-1) is a data section with initial values or a program section.

Workaround:
Select the -crc option for absolute load modules and rebuild them.

3.3 With Overlaying Sections with Different Alignment Numbers (LNK-0004)

Version concerned:
V.9.00 Release 04A through V.9.02 Release 00

Symptom:
If sections with alignment numbers different from each other are overlaid, aligning the section first to be overlaid will place other sections at incorrect addresses.

Conditions:
This symptom may arise if the following condition (1) or (2) is satisfied:

(1) The following conditions are all satisfied:
(1-1) Overlays are specified by using parentheses '()', and there is a section before or behind parentheses '().'
(1-2) One or more overlays are specified for groups of sections (that is, one overlay for one group), and, the alignment numbers of the sections in a group are different from each other.
(1-3) The start address specified by the start option is not a common multiple of the alignment number of sections.
Example: -start=(P1:P2),P3,(P4:P5),P6/100
Here, one or more of the alignment numbers of sections P1, P3, P4, or P6 is different from others.
The start address of assignment is 100, and it is not a common multiple of numbers of the alignment number of sections P1, P3, P4, and P6.

(2) The following conditions are all satisfied.
(2-1) Overlays are not specified by using parentheses '()'.
The alignment number of the front section of the group first to be overlaid is greater than that of any front section of the other groups (overlaid later).

(2-2) The address specified by the start option is not a multiple of the alignment number of the front section of the group first to be overlaid.

Example: -start=A1,A2:B1,B2:C1,C2/152

Here, the alignment number of A1 is greater than those of B1 and C1.

The start address of assignment is 152, and it is not a multiple of the alignment number of section A1.

Workarounds:

Avoid the problem in any of the following ways:

(1) If the condition (1) is satisfied: Specify same value for all alignment number of the section.

(2) If the condition (2) is satisfied: Overlay the section group whose front section has the least alignment number first.

(3) Specify the common multiple of the alignment number of sections to the start address by start option.

3.4 With Using the data_stuff and nooptimize Options in Linking (LNK-0005)

Versions concerned:

V.9.00 Release 00 through V.9.02 Release 00

Symptom:

If options data_stuff and nooptimize are selected to stuff data, odd addresses may be assigned to variables of even bytes in length.

Conditions:

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

(1) The data_stuff option is selected in linking.

(2) The nooptimize option is selected at the same time.

(3) Two or more object files are linked.

(4) At least two object files contain a data section with the same name.

(5) An object file specified second or later by the input file

contains a data section with the same name in (4), and the last data item of the data section is 1 byte long.

(6) In the data section in (5) does not exist the symbol with same alignment number as the alignment number of the section.

Example:

```
-------------------------------------------
//a.c    // Condition (4)
char a;
//b.c     // Condition (4)
short b; // Condition (6)
char c;  // Conditions (5) and (6)
-------------------------------------------
```

Linker command

```
-------------------------------------------
optlnk a.obj b.obj ?data_stuff -nooptimize
-------------------------------------------
```

Workarounds:

Avoid the problem in any of the following ways:

(1) Do not select the data_stuff option in linking.

(2) Select the source file where this symptom arises; then, in the section in which exist symbols that odd addresses are assigned to, define dummy variables whose length are the same as the alignment number of the section.

Example:

```
-------------------------------------------
//b.c
long dummy;
short b;
char c;
-------------------------------------------
```

(3) Select the source file where this symptom arises; then, define dummy variables of 2 bytes long such a way that those are placed at the end of the section in which exist symbols that odd addresses are assigned to.

Example:

```
-------------------------------------------
```

```
//b.c
short b;
char c;
short dummy;
-----------------------------------------
```

4. Schedule of Fixing the Problems

We plan to fix these problems in the V.9.03 Release 00 compiler package.
