RENESAS Tool News

RENESAS TOOL NEWS on November 1, 2005: RSO-SHC-051101D

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

Please take note of the eleven problems described below in using the C/C++ compiler package V.9 for the SuperH RISC engine family of MCUs.

1. Versions Concerned

C/C++ compiler package V.9.00 Release 00 through V.9.00 Release 03 for the SuperH RISC engine family

2. Problems

2.1 On defining a structure- or union-type array using the typedef specifier (SHC-0039)

Consider the case where the typedef declaration is made of a structure or union using its tag name used in its tentative declaration, and then the structure or union is declared. If a structure- or union-type array is defined as of the type specified by the typedef declaration, the boundary alignment number may become incorrect.

Conditions

This problem occurs if the following conditions are all satisfied:

(1) A typedef declaration is made of a structure or union using its tag name used in its tentative declaration.

(2) After the typedef declaration in (1), a structure or union is declared.

(3) At least a member of the structure or union in (2) is not of type char, nor unsigned char.

(4) An array is defined as of the type specified by the typedef declaration in (1).

(5) The pack=1 option is not used. Or the array in (4) is not subject to the #pragma pack 1 preprocessor directive.

(6) The array in (4) is not qualified to be volatile, nor const.

(7) No member of the structure- or union-type array in (4) is referenced in a function defined within the program file.

Example C source file:

```
-----------------------------------------------
typedef struct ST ST_A;
struct ST {
   int x;
};
ST_A a[2];
-----------------------------------------------
```

Result of compilation:

```
-----------------------------------------------
_a:                     ; static: a
       .RES.B     8       ; To be .RES.L   2
-----------------------------------------------
```

Workaround

This problem can be circumvented any of the following ways:

(1) Make the typedef declaration after defining a structure or union.

(2) Reference a member of a structure- or union-type array in a function defined within the program file.

(3) Qualify a structure- or union-type array to be volatile.

(4) Define a structure- or union-type array as not of the type specified by the typedef declaration.

2.2 On making an assignment to a bit field of 1 bit wide that is a member of a structure- or union-type array (SHC-0040)

An incorrect value may be assigned to a bit field of 1 bit wide that is a member of a structure- or union-type array.

Conditions

This problem occurs if the following conditions are all satisfied:

(1) The cpu=sh2a or cpu=sh2afpu option is used.

(2) In the program exists a structure- or union-type array one of whose members is a bit field of 1 bit wide.

(3) No structure or union in the array in (2) is 1 or 4 bytes in width.

(4) An assignment is made to a bit field of 1 bit wide that is a member of the structure- or union-type array in (2).

(5) The assignment in (4) meets either of the following conditions:

a. Any of the following is used as s subscript of the array in (4); or the assignment is made using a pointer and any of the following is used in address calculation:
   - function call
   - member of a structure or union (including a bit field)
   - relational or equality operator
   - shift operator
   - multiplication, division, or residue operation
   - operation of type long long
   - type conversion from floating to int
   - include function such as addc() or shll() for replacing the value of the T bit in the SR register

b. After the SHLL instruction is executed, the assignment is made using the BST.B instruction.

Example C source file:

```
-----------------------------------------------
struct {
   unsigned short a:8;
   unsigned short b:1;
} ST1[100],ST2[100];

void func(int i,int j) {
   ST1[i].b=ST2[j].b;
}
-----------------------------------------------
```

Result of compilation:

```
-----------------------------------------------
_func:
        MOV.L     L11+2,R6   ; _ST2
        SHLL      R5
        ADD       R5,R6
       BLD.B     #7,@(1,R6) ; Loads value into bit T in register SR
        MOV.L     L11+6,R2   ; _ST1
        SHLL      R4         ; Replaces value of bit T in register SR
        ADD       R4,R2
        BST.B     #7,@(1,R2) ; Assigns replaced value
        RTS/N
-----------------------------------------------
```

Workaround

Assign the address of the structure- or union-type array concerned to a volatile-qualified variable of type pointer. Then using this pointer, make an assignment to

the bit field of 1 bit wide.

2.3 On the referencing order of volatile-qualified variables (SHC-0041)
When volatile-qualified variables are referenced, their referencing orders may be changed each other after compilation.

Conditions
This problem occurs if the following conditions are all satisfied:

(1) The optimize=1 option is used.

(2) In the program exists the definition of a funct ion that contains either of the following:
   a. function call
   b. selection statement, iteration statement, conditional operator, or logical operator, with any cpu option except cpu=sh1 being used

(3) Before or within the statement or operator in (2), a volatile-qualified variable is referenced.

(4) Before the statement or expression containing the reference to the variable in (3), another volatile-qualified variable is referenced.

Example C source file:

```
-----------------------------------------------
volatile int a,b;
int c,d;

void func() {
   int x,y;
   do {
     x=a;          // Condition (4)
     y=b;          // Condition (3)
  } while (x != a);    // Condition (2)-b
}
-----------------------------------------------
```

Result of compilation:

```
-----------------------------------------------
_func:
       MOV.L      L13,R4     ; _a
       MOV.L      L13+4,R1   ; _b
L11:
      MOV.L      @R4,R6     ; Reference in Condition (4)
                            ; Reference in Condition (3)
```

```
                    to be made here not referenced
      MOV.L     @R4,R2    ; Reference in Condition (2)-b
       CMP/EQ    R2,R6
       BF/S      L11
      MOV.L     @R1,R5    ; Here made reference in Condition (3)
       RTS/N
-----------------------------------------
```

Workaround
This problem can be circumvented either of the following ways:

(1) Use the optimize=0 option, not optimize=1.

(2) Immediately before the statement or expression in Condition (2)-a or -b, place an nop() include function or an assignment to another volatile-qualified variable than those in Conditions (3) and (4).

2.4 On an initial value of a union containing a bit field as one of its members (SHC-0042)

2.4.1 When a union containing a bit field as one of its members has an initial value, and the bits of a bit field are assigned to the data values to store from lower to upper, the initial value may become incorrect.

Conditions
This problem may occur if the following conditions are all satisfied:

(1) In the program exists a union whose first member is a bit field.

(2) The size of the bit field in (1) is not the one specific to the declared type of the bit field.

(3) The union in (1) has the initial value of a negative quantity.

(4) The bit_order=right option is used. Or the union in (1) is subject to the #pragma bit_order right directive.

Example C source file:

```
-----------------------------------------
#pragma bit_order right
union {
   int a:2;
   int b;
}u={-1};
-----------------------------------------
```

Result of compilation:

```
-----------------------------------------
_u:                     ; static: u
        .DATA.L     H'FFFFFFFF ; To be H'00000003
-----------------------------------------
```

Workaround

This problem can be circumvented either of the following ways:

(1) In front of the first member of the union in Condition (1), declare a structure that has only a bit field as its member whose size is the same as the bit field that is the first member of the union.

Example:

```
union {
    struct {
       int dummy:2;
    }s;
    int a:2;
    int b;
}u={-1};
```

(2) Mask the initial value with the bit size of the bit field; then use this as the new one.

Example:

```
union {
   int a:2;
   int b;
}u={-1&3};
```

2.4.2 When a union containing a bit field as one of its members has an initial value, and the initial value has redundant braces {}, the initial value may become incorrect.

Conditions

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

(1) In the program exists a union whose first member is a bit field.

(2) The size of the bit field in (1) is not the one specific to the declared type of the bit field.

(3) The union in (1) has an initial value.

(4) The initial value has redundant braces {}.

Example C source file:

```
-----------------------------------------
union {
   char a:3;
   short b;
}u={{0x8F}};
-----------------------------------------
```

Result of compilation:

```
-----------------------------------------
_u:                     ; static: u
        .DATA.B     H'8F      ; To be H'E0
        .DATAB.B    1,0
-----------------------------------------
```

Workaround

Remove the redundant braces {}.

2.5 On a function taking a variable number of arguments (SHC-0043)

When a recursive call is made of a function that takes a variable number of arguments, they may incorrectly be passed to the function.

Conditions

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

(1) The optimize=1 option is used.

(2) A recursive call is made of a function that takes a variable numberof arguments.

(3) The argument precedent by two to the variable part of arguments is passed to the function via a register.

(4) Before the argument in (3) exists another one passed to the function via the stack.

(5) The return value of the function in (2) is of type void, int, or floating.

(6) All the arguments in (2) are of type int or floating.

(7) A recursive call of a function is made:
   a. within a return statement, or
   b. at the end of the function or immediately before the return statement, with the return value of the function being of type void.

Example C source file:

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

```
long long S;
void f(long long a1, int a2, int a3, ...) {
   S += a1;
   if (a1!=0) {
      f(a1-1, 0, 0);    /*  Becomes f(0, 0, 0); */
   }
}
```

Workaround

This problem can be circumvented any of the following ways:

(1) Use the optimize=0 option, not optimize=1.

(2) Qualify at least one of the arguments to be volatile.

(3) Declare the function in Condition (2) using the #pragma regsave directive.

(4) Arrange arguments so that Condition (3) or (4) might not be met.

(5) If Condition (7)-b is met, place an nop() include function immediately after the function call.

2.6 On the controlling expression of a switch statement being of type long long or unsigned long long (SHC-0044)

When the controlling expression of a switch statement is of type long long or unsigned long long; and the result of its evaluation is a constant, control may jump to the statement of an incorrect case label or an address error may arise.

Conditions

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

(1) In the program exists a switch statement.

(2) The controlling expression of the switch statement in (1) is of type long long or unsigned long long.

(3) The controlling expression in (2) is a constant expression.

(4) The minimum value of case constants in the switch statement in (1) is 0.

(5) The switch statement in (1) is expanded as a table.

Example C source file:

```
int func() {
   long long d=1;
   switch(d) {      /* Evaluation of controlling expression
```

```
    case 0:                         always results in 1 */
    case 1:
    case 2:
    case 3:
    case 4:
    case 5:
    case 6:
    case 7:
    case 8:
      return 1;
      break;
    default:
      break;
    }
    return 0;
}
-------------------------------------------
_func:
        STS.L      PR,@-R15
        MOV        #1,R6     ; H'00000001
        MOV.L      R6,@-R15
        MOV        #0,R1     ; H'00000000
        MOV.L      R1,@-R15
        MOV        #8,R5     ; H'00000008
        MOV.L      R5,@-R15
        MOV.L      L25+2,R4   ; __cmpgt64u
        JSR        @R4
        MOV.L      R1,@-R15
        ADD        #16,R15
        CMP/EQ      #0,R0
        BF         L23
        MOV        #1,R6     ; H'00000001
        MOVA       L26,R0
        MOV.L      @(R0,R6),R1; Accesses an odd address,
                              resulting in an address error
        ADD        R1,R0
        JMP        @R0
        NOP

. . . . . .

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

Workaround

This problem can be circumvented any of the following ways:

(1) Assign the controlling expression of the switch statement to a volatile-qualified variable, and then substitute this variable for the controlling expression.

(2) Add a case label with a smaller value than 0.

(3) Use the case=ifthen option.

2.7 On Using Two or More Integer Constants (SHC-0045)

If two or more integer constants are used in a block, they may be replaced with incorrect ones after compilation.

Conditions

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

(1) The optimize=1 option is used.

(2) In a block is used an integer constant whose value is within a range from -2,147,483,648 to -129, or from 128 to 2,147,483,647.

(3) In the block that contains the integer constant in (2) exists another integer constant, and the difference between the former and the latter falls within a range from -128 to 127. *

(4) The types of the integer constants in (2) and (3) are different from each other.

* Note that there can exist more than one combination of integer constants in (2) and (3) in a block

Example C source file:

```
-----------------------------------------------
int func() {
   unsigned short a;
   unsigned short *p;
   a = 65535;
   p = &a;
   *p *= 32767;         // *p set to 32769
   if (*p == 32770) {    // 32769 compared with 32770
      return 1;
   }
   return 0;
}
```

```
-------------------------------------------
Result of compilation:
-------------------------------------------
_func:
        MOV.W       L11,R6     ; H'8001
        EXTU.W      R6,R2      ; R2 set to 32769
        ADD         #1,R6      ; R6 set to -32766
        CMP/EQ      R6,R2      ; 32769 compared with -32766
        RTS
        MOVT        R0
-------------------------------------------
```

Workaround

This problem can be circumvented either of the following ways:

(1) Assign either of the integer constants in Conditions (2) and (3) to a volatile-qualified variable, and then use it.

(2) Use the optimize=0 option, not optimize=1.

2.8 On an Assignment to an Element of a volatile-Qualified Array (SHC-0046)

If before an if statement and within its then statement exist two assignment expressions that assign variables to the same element of a volatile-qualified array, assignment may be made only once.

Conditions

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

(1) The optimize=1 option is used.

(2) In the program exists an iteration statement.

(3) In the iteration statement in (2) above exists an if statement not accompanied with an else statement.

(4) Before the if statement and within its then statement in (3) exist two assignment expressions that assign variables to the same element of an array.

(5) The array in (4) is qualified to be volatile. Or the array in (4) is an external variable, and the global_volatile=1 option is used.

Example C source file:

```
-------------------------------------------
int x, y, z;
volatile int V[10];
```

```
void func() {
   int i;
   for(i = 0; i < 10; i++) {
      V[i] = x + y;      // 1st assignment
      if(z) {
         V[i] = y + z;   // 2nd assignment
      }
   }

}
```

Result of compilation:

```
_func:
        MOV.L      R14,@-R15
        MOV.L      L15,R5     ; _x
        MOV.L      L15+4,R4   ; _y
        MOV.L      L15+8,R2   ; _z
        MOV.L      @R5,R14
        MOV.L      @R4,R1
        MOV.L      @R2,R4
        ADD        R1,R14
        ADD        R4,R1
        MOV        #0,R5      ; H'00000000
        MOV        #10,R6     ; H'0000000A
        MOV.L      L15+12,R7  ; _V
L11:
        TST        R4,R4


        MOV        R14,R2
        BT         L13
        MOV        R1,R2
L13:
        ADD        #-1,R6
        MOV        R5,R0
        TST        R6,R6
        MOV.L      R2,@(R0,R7); Assignment always made only once
        ADD        #4,R5
        BF         L11
        RTS
        MOV.L      @R15+,R14
```

Workaround

This problem can be circumvented any of the following ways:

(1) Place an nop() include function in the then statement of the if statement in Condition (3).

(2) Add an else statement to the if statement in Condition (3).

(3) Use the optimize=0 option, not optimize=1.

2.9 On an Iteration Statement Having a volatile-Qualified Controlled Variable (SHC-0047)

If an iteration statement has a volatile-qualified controlled variable, iteration may be made an incorrect number of times.

Conditions

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

(1) The optimize=1 option is used.

(2) In the program exists an iteration statement.

(3) The iteration statement in (2) has a controlled variable of type int; neither long long nor unsigned long long.

(4) The controlled variable in (3) is qualified to be volatile. Or the controlled variable in (3) is an external variable, and the global_volatile=1 option is used.

(5) The reset expression that resets the controlled variable in (3) is an additive expression (addition or subtraction).

(6) In the iteration statement in (2) is made a function call. Or in this iteration statement exists a pointer-type variable pointing to the controlled variable in (3), and this pointer-type variable is reset in the iteration statement.

Example C source file:

```
-------------------------------------------
 extern void sub();
 volatile int i;
 void func() {
    i = 1;
    while (i) {
       i--;

       sub();   // Though i may be reset at origin of function call,
              // number of times of iteration is always 1
```

```
    }
    return;
}
```

Workaround
This problem can be circumvented any of the following ways:

(1) Place an expression that refers to the controlled variable in Condition (3) after the reset expression in Condition (5).

(2) Declare the controlled variable in Condition (3) to be long long or unsigned long long.

(3) Use the optimize=0 option, not optimize=1.

2.10 On Using Comma Operators in the Controlling Expression of an Iteration Statement (SHC-0048)
If comma operators are used in the controlling expression of an iteration statement, the controlling expression may be evaluated from right to left.

Conditions
This problem may occur if the following conditions are all satisfied:

(1) The optimize=1 option is used.

(2) In the program exists an iteration statement.

(3) The iteration statement in (2) has a controlled variable of type int; neither long long nor unsigned long long.

(4) The reset expression that resets the controlled variable in (3) is an additive expression (addition or subtraction).

(5) The iteration statement in (2) is iterated only once.

(6) In the controlling expression of the iteration statement in (2) is used comma operators.

Example C source file:

```
int A, B;
void func() {
   int i;
   for (i=0; A++, B+=A, i<1; i++) {  // B+=A executed first; then A++
      B++;
   }
}
```

Workaround

This problem can be circumvented any of the following ways:

(1) Use no iteration statement.

(2) Declare the controlled variable in Condition (3) to be long long or unsigned long long.

(3) Use the optimize=0 option, not optimize=1.

2.11 On Using the Same Constant before and after a Function Call (SHC-0049)

If the same constant is used before and after a function call, the value of the constant after the function call may become incorrect.

Conditions

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

(1) The optimize=1 option is used.

(2) Neither the opt_range=noblock option nor the global_alloc=0 option is used.

(3) In the program exists an function call.

(4) The same constant is used before and after the function call in (3).

(5) The constant in (4) is used at least once in an iteration statement.

(6) A register whose value is indefinite before and after a function call is assigned to the constant in (4).

Example C source file:

```
-----------------------------------------------
int array[10];
void func(int j, int k) {
   int i;
   for (i = 0; i < 10; i++) {
      if (k < 20) {
         sub();

      }
      if (j < 20) {
         array[i] = j;
      }
. . . . . .
   }
```

```
}
-------------------------------------------
Result of compilation:
-------------------------------------------
_func:
. . . . . .
        MOV       R5,R11
        MOV       R4,R12
        MOV       #10,R14
        MOV.L     L17,R13    ; _array
L11:
        MOV       #20,R1     ; R1 set to 20
        CMP/GE    R1,R11
        BT        L13        ; (A)
        MOV.L     L17+4,R2   ; _sub
        JSR       @R2        ; (B)
        NOP
L13:
        CMP/GE    R1,R12     ; Even when R1 is reset at origin of
                             ; function call in (B), previous value
                             ; of R1 must be used if no jump is
                             ; made at (A); however, reset value of
                             ; R1 is used in error
        BT        L15

        MOV.L     R12,@R13

. . . . . .

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

Workaround
This problem can be circumvented any of the following ways:

(1) Use the opt_range=noblock option.

(2) Use the global_alloc=0 option.

(3) Use the optimize=0, not optimize=1.

3. **Schedule of Fixing the Problems**
   We plan to fix these problems at the next release of the products. (Scheduled at the end of March, 2006)

[Disclaimer]
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.