RENESAS Tool News

RENESAS TOOL NEWS on April 1, 2006: RSO-SHC_2-060401D

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

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

1. Versions Concerned

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

2. Descriptions

2.1 Problem on Defining a Structure or Union after Declaring an Array of the Type of the Said Structure or Union (SHC-0064)

When a structure or union is defined after an array is declared which is of the type of the said structure or union and whose storage class is extern, code for accessing the first element of the array may be generated if any element other than the first is accessed.

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

(1) In the program exists an array that is of the type of a structure or union and whose storage class is extern.

(2) After the declaration of the array in (1), the structure or union in (1) is defined.

(3) Any element except the first one of the array in (1) is accessed.

Example:

```
-----------------------------------------------
extern struct TBL g[3];  // Condition (1)
struct TBL {           // Condition (2)
   int m;
};
struct TBL tbl;

void func() {
   tbl.m = g[1].m;     // Condition (3)
}
-----------------------------------------------
```

Result of compilation:

```
-----------------------------------------------
_func:
        MOV.L      L11+2,R1   ; _g
        MOV.L      L11+6,R4   ; _tbl
        MOV.L      @R1,R6     ; In spite of g[1].m, g[0].m
                              ; accessed in error
        RTS
        MOV.L      R6,@R4
-----------------------------------------------
```

Workaround:
Define the structure or union involved before declaring the array in Condition (1).

2.2 On Assigning a Variable to a Parameter of a Function (SHC-0065)

When a variable is assigned to a parameter of a function, the assignment may be performed incorrectly.

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

(1) The opitmize=1 option is selected.

(2) The schedule=0 option is not selected.

(3) The noscope option is not selected.

(4) A function takes a parameter of type signed char, unsigned char, signed short, or unsigned short.

(5) The address of the parameter in (4) is not accessed.

(6) A value is passed to the parameter in (4) from the calling function via a register.

(7) An assignment is made to the parameter in (4).

Example

```
-----------------------------------------------
void func(unsigned short ss) {  // Conditions (4) and (6)
 . . . . . .
   ss = 0xffff - ss;          // Condition (7)
 . . . . . .
}
-----------------------------------------------
```

Result of compilation:

```
-----------------------------------------------
_func1:
 . . . . . .
        MOV        #-1,R1       ; H'FFFFFFFF
        EXTU.W     R1,R1
        SUB        R4,R1
        MOV        #30,R0       ; H'0000001E
        MOV.W      R1,@(R0,R15)  ; ss = 0xffff - ss
        MOV.L      R4,@(28,R15)  ; Left term of above
overwritten
 . . . . . .
-----------------------------------------------
```

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

(1) Use the optimize=0 option.

(2) Use the schedule=0 option.

(3) Use the noscope option.

(4) Qualify the parameter involved to be volatile.

(5) Use the parameter involved after assigning it to a local variable at the beginning of the function involved.

2.3 On Using a Function That Takes a Parameter Passed via the Stack (SHC-0066)

When a comparison (== or !=) is made or a bit field is used within a function that takes a parameter passed via the stack, the comparison or the access to the bit field may incorrectly be performed.

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

(1) The optimize=1 option is selected.

(2) Equality or inequality (== or !=) is evaluated between a given value and a constant within a range of -128 to 127 inclusive, or a bit field is used in a function.

(3) The function in (2) takes a parameter passed via the stack.

Example:

```
-----------------------------------------------
struct {
   unsigned int a:31;
   unsigned int b:1;
} *p;

void func(unsigned int a, unsigned int b, unsigned int c,
        unsigned int d, unsigned int g) {
   volatile int q[16];
   q[15]=0;
   p->b = g;     // Conditions (2) and (3)
}
-----------------------------------------------
```

Result of compilation:

```
-----------------------------------------------
        ADD         #-64,R15
        MOV.L       L13,R4      ; _p
        MOV         #1,R7       ; H'00000001
        MOV.L       @R4,R6
        MOV         #0,R1       ; H'00000000
        MOV.B       @(3,R6),R0
```

```
        MOV       #64,R0      ; R0 corrupted
        MOV.L     @(R0,R15),R5
        MOV.L     R1,@(60,R15)
        TST       R7,R5
        OR        #1,R0
        BF        L12
```

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

(1) Use the optimize=0 option.

(2) Assign all the parameters passed via the stack to volatile variables at the beginning of the function involved.

2.4 On Assigning a 0 or 1 to a Volatile Variable in the True and False Statements of a Control Statement (SHC-0067)

When a 0 or 1 is assigned to a volatile-qualified variable according to the evaluation of the controlling expression of a control statement, unnecessary accesses to memory may be performed.

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

(1) The optimize=1 option is selected.

(2) An if statement or a conditional operator (?:) is used.

(3) In each of the true and else statements of the if statement or the conditional operator exists an assignment expression to the same volatile variable (included the case where the global_volatile option is selected).

(4) A 0 is assigned to the variable in one assignment expression in (3) and a 1 to the variable in the other.

(5) In the true and false statements of the if statement or the conditional operator in (2), only

the assignment expressions are different each other.

Example

```
-----------------------------------------------
   volatile int a;          // Condition (3)
   int b;
   void func(){
      if (b == 0)           // Condition (2)
      {
         a = 0;            // Conditions (3), (4), and (5)
      } else {
         a = 1;            // Conditions (3), (4), and (5)
      }
   }
-----------------------------------------------
```

Result of compilation:

```
-----------------------------------------------
    MOV.L     L11,R5    ; _b
    MOV.L     @R5,R1
    TST       R1,R1
    MOVT      R4
    MOV.L     L11+4,R7  ; _a
    MOV.L     R4,@R7    ; If interrupt generated here,
                        ; value of a is incorrect.
    MOV.L     @R7,R0
    XOR       #1,R0
    RTS
    MOV.L     R0,@R7
-----------------------------------------------
```

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

(1) Use the optimize=0 option.

(2) When an if statement is used in Condition (2), place an nop() include function or an assignment expression to a different volatile variable from the one in Condition (3) in either the true or the false statement.

(3) When a conditional operator is used in Condition (2), change it to an if statement and then follow

the way in (2) above.

3. Schedule of Fixing the Problem

The above problems have already been resolved in the product of V.9.00 Release 04 (revised on this April 1).

For details of the revision, see RENESAS TOOL NEWS No. RSO-SHC_1-060401D, published on April 1, 2006.
