RENESAS Tool News

RENESAS TOOL NEWS on March 1, 2007: 070301/tn3

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

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

1. Product and Versions Concerned

The C/C++ compiler package for the SuperH RISC engine family
V.9.00 Release 00 through V.9.00 Release 04A

2. Descriptions

2.1 Problem with Incrementing Pointer Variables or Subscripts of Arrays (SHC-0069)

If the address of a pointer variable or an array is used in a controlling expression after incrementing the pointer variable or a subscript of the array, the address before the increment is made may be referenced.

Conditions:

This problem may occur if the following conditions are all satisfied:
(1) The optimize=1 option is selected.
(2) The cpu=sh2a or the cpu=sh2afpu option is also selected.
(3) In the program exists an assignment expression to an object pointed to by a pointer variable, and immediately after the assignment, the pointer variable is incremented.
Or, an assignment expression exists whose left term is an element of an array, and immediately after the assignment, the subscript of the array is incremented.
(4) The address of the pointer variable or the array in (3) is used in the controlling expression of an if statement or a loop.

Example:

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

```
int ST[100],ST2[100];
void func(void)
{
   int *p=ST, *q=ST2;
   do {
      *p++=*q;          // Condition (3)
      q++;
   } while (p < ST+100);  // Condition (4)
}
---------------------------------------------
L11:
     MOV.L      @R5+,R0
     CMP/HS      R4,R6      ; p referenced in a controlling
                           ; expression before incremented
     BF/S       L11
     MOV.L      R0,@R6+
     RTS/N
---------------------------------------------
```

Workarounds:
This problem can be avoided by any of the following ways:
(1) Place an nop() include function immediately after the increment in Condition (3).
(2) Place an expression assigning the pointer variable or the element of the array in Condition (3) to a volatile-qualified external variable immediately after the increment in Condition (3).
(3) Use the optimize=0 option.

2.2 Problem in Using Two or More Expressions Containing an Induction Variable in a Loop (SHC-0070)

When two or more operation expressions containing the induction variable of a controlling expression exist in the body of a loop, the results of operations of some operation expressions may be incorrect.

Conditions:
This problem may occur if the following conditions are all satisfied:
(1) The optimize=1 option is selected.
(2) In the program exists a loop.
(3) In the body of the loop in (2) exist the same two or more operation expressions (for example, i*4 and 4*i are considered as the same operation expressions) containing the induction variable of a controlling expression.
(4) The operators used in the expressions in (3) are only one kind of

addition, subtraction, multiplication, decrement, or left shift.
(5) At least one of the expressions in (3) is in the right term of an assignment expression, and the type of its left term is smaller in size than that of the operation expression.
(6) In the assignment expression in (5), the result of operation of the right term exceeds the maximum value expressible in the type of the left term.

Example:

```
---------------------------------------------
// a = 64*4 + (unsigned char)(64*4) = 256 + 0 = 256 is correct, but
// a = 64*4 + 64*4 = 256 + 256 = 512 obtained
int a;
f()
{
   int i;
   unsigned char temp;
   for (i=63; i<=64; i++) {  // Condition (2)
      temp = i*4;          // Conditions (3), (4), (5), and (6)
      a = i*4 + temp;       // Conditions (3) and (4)
   }
}
---------------------------------------------
```

Workarounds:
This problem can be avoided by either of the following ways:
(1) Use the optimize=0 option.
(2) Qualify the induction variable in Condition (3) to be volatile.

2.3 Problem in Using the file_inline Option (SHC-0071)

If a function is expanded inline between files using the file_inline option, incorrect integer constants may be used in another function that makes a call to the function to be expanded.

Conditions:
This problem may occur if the following conditions are all satisfied:
(1) The file_inline option is selected.
(2) The optimize=1 option is also selected.
(3) Two or more integer constants are used in the following two functions:
 - A function that exists in the file specified by file_inline and is to be expanded inline between files
 - Another function that exists in the file to be compiled and

makes a call to the function to be expanded inline

(4) At least one of the integer constants in (3) has a different value from the others.

Example of file to be compiled: a.c

```
---------------------------------------------
int c;
extern void sub();

void func()
{
   c = 1000;       // Conditions (3) and (4)
   sub();
}
---------------------------------------------
```

Example of file specified by file_inline: b.c

```
---------------------------------------------
int b;
void sub()
{
   b = 300;        // Conditions (3) and (4)
}
---------------------------------------------
```

Result of compilation:

```
---------------------------------------------
_func:
        MOV.L      L11+4,R5   ; _c
        MOV.L      L11+8,R6   ; _b
        MOV.W       L11,R2    ; H'03E8 (= 1000)
        MOV.L      R2,@R5     ; 1000 assigned to variable "c"
        RTS
        MOV.L      R2,@R6     ; sub() expanded inline here; then
                              ; incorrect value, 1000, assigned to
                              ; variable "b"
L11:
        .DATA.W     H'03E8
---------------------------------------------
```

Workarounds:

This problem can be avoided by either of the following ways:

(1) Do not use the file_inline option.

(2) Use the optimize=0 option.

3. Schedule of Fixing the Problems

We plan to fix these problems in the next release, V.9.01 Release 00, of the product.