RENESAS Tool News

RENESAS TOOL NEWS on June 1, 2011: 110601/tn7

Notes on Using the C/C++ Compiler Package for the M16C Series and the R8C Family (M3T-NC30WA) V.6.00 Release 00

When you use the C/C++ compiler package for the M16C series and the R8C family (M3T-NC30WA) V.6.00 Release 00, take note of the following problems:

- With passing a floating constant to a function as an argument
- With defining a function with the static specifier as an interrupt function

These problems are also described in Chapter 3, "Precautions", of the release note included with the compiler package.

---

1. Problem with Passing a Floating Constant to a Function as an Argument

1.1 Description

If an expression converting a negative floating constant to an unsigned integer is passed to a function as an argument, the following warning message appears, and the value after conversion goes to 0.

```
 sample.c(6) : C1841 (W) underflow in floating value
   converting to integer
     ===>    int j = func((unsigned int)-1.0);
```

1.2 Conditions

This problem arises if the following conditions are all satisfied:

(1) An argument to a function is a negative floating constant.
 Here, the constant can be a variable or expression that is replaced with a constant by optimization.

(2) The constant in (1) is cast to any of the following types:
 char
 unsigned char
 unsigned short
 unsigned int
 unsigned long

        unsigned long long

Example:

```
-------------------------------------------------------------------
#include <stdio.h>
int func(int x) { return x; }
void main(void)
 {
   int i = (unsigned int)-1.0;
   int j = func((unsigned int)-1.0);     /* Conditions (1) and (2) */
   if (i != j) {
      printf("NG (i, j) = (%d, %d)¥n", i, j);
              /* Constants i and j go to -1 and 0 respectively */
   } else {
      printf("OK¥n", i, j);
   }
}
-------------------------------------------------------------------
```

1.3 Workaround

Before calling the function involved in Condition (1), assign the constant that is cast in Condition (2) to a temporary variable, and then pass the temporary variable to the function as an argument.

Example:

```
-------------------------------------------------------------------
#include <stdio.h>
int func(int x) { return x; }
void main(void)
{
   int i = (unsigned int)-1.0;
   unsigned int tmp = (unsigned int)-1.0;    /* temp defined */
   int j = func(tmp);
   if (i != j) {
      printf("NG (i, j) =1(%d, %d)¥n", i, j);
   } else {
      printf("OK¥n", i, j);
   }
}
-------------------------------------------------------------------
```

1.4 Schedule of Fixing the Problem

We plan to fix this problem in the next version of the product.

2. Problem with Defining a Function with the Static Specifier as an Interrupt Function

2.1 Description

If an interrupt function is defined by using the #pragma interrupt preprocessing directive including a number of the interrupt vector, no code may be generated for the function, and the address of the function be not set in the variable interrupt vector table.
As a result, this function cannot be called if interrupts are generated.

2.2 Conditions

This problem arises if the following conditions are all satisfied:

(1) Compile option -OS_MAX (-OSM) is used.
Or, -Oforward_function_to_inline (-OFFTI) is used together with any of the following options:
-O, -O1 through -O5, -OR_MAX (-ORM), -OR, and -OS

(2) A function is defined by using the storage class specifier static.

(3) The function in (2) is defined by using #pragma interrupt including a number of the interrupt vector.

(4) No call is made to the function in (2) or no reference is made to its address.

Example:

```
----------------------------------------------------------------
#pragma interrupt func(vect=31)   /* Condition (3) */
static void func(void)            /* Condition (2) */
{
}
----------------------------------------------------------------
```

In this example, no code can be generated for the function, and the address of the function cannot be set in the variable interrupt vector table because the static function, which is not referenced, is removed by optimization.
So this function cannot be called if interrupts are generated.

2.3 Workaround

Do not use -OS_MAX (-OSM) or -Oforward_function_to_inline (-OFFTI).

2.4 Schedule of Fixing the Problem

We plan to fix this problem in the next version of the product.

[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.

© 2010-2016 Renesas Electronics Corporation. All rights reserved.