RENESAS Tool News

RENESAS TOOL NEWS on February 01, 2015: 150201/tn5

Note on Using C/C++ Compiler Package for RX Family (IDE: CubeSuite+) and C/C++ Compiler Package for RX Family (without IDE)

When using the C/C++ Compiler Package for RX Family (for the CubeSuite+ IDE) and of the C/C++ Compiler Package for RX Family (without an IDE), take note of the problem regarding the following points.

- Point to note regarding static aggregates and unions within a function having initial values of address constant expression (RXC#035)
- Point to note regarding the use of both judgment of a match and greater or less than for variables (RXC#036)

Note: The number at the end of the above item is from a consecutive index of problems in the compiler package for the RX family of MCUs.

---

1. Static Aggregates and Unions within a Function Having Address Constant Expressions as Initial Values (RXC#035)

1.1 Products and Versions Concerned

- C/C++ Compiler Package for RX Family (IDE: CubeSuite+) V2
  Order type name: RTCRX0000CL02WDR and RTCRX0000CL02WNR
  CC-RX compiler V2.02.00

- C/C++ Compiler Package for RX Family (without IDE) V2
  Order type name: RTCRX0000CC02WRR and RTCRX0000CC02WNR
  CC-RX compiler V2.02.00

1.2 Description

The initialization of the variables or arrays which were declared as

static or as const without extern might not operate correctly.

1.3 Conditions

This problem arises if the following conditions are all met.

(1) The program is compiled as a C++ program.

(2) A function was declared as static and holds automatic variables.

(3) Variables or arrays were declared as static, or as const without extern. In addition, the expressions for initializing these variables or arrays include addresses of a function (or functions) satisfying condition (2).

(4) A function was declared as static and does not hold automatic variables, and the code within the function refers to a variable or array satisfying condition (3).

(5) A function satisfying condition (2) contains processing which uses an automatic variable* mentioned in (2) at one or more places.

*: The expression "uses an automatic variable" covers the cases of initializing the given automatic variable at the same time as it is declared and of using the given variable in an expression for initializing another variable, as well as reading from or writing to the automatic variable, or obtaining its address.

(6) The functions satisfying conditions (2) and (4) are not referred by any functions or variables declared with extern.

Example of the condition:

In the case of compiling the following as C++ code: Condition (1)

```
-----------------------------------------------------------------
static void OtherFunc(void);
static int LocalFunc(void);
int (*const sf_Table[] )(void) = {   // Condition (3)
     LocalFunc, 0,                // Condition (3)
};
extern int (*const *pFunc)(void);
static void OtherFunc(void)
{
    pFunc = sf_Table;             // Condition (4)
}
static int LocalFunc(void)          // Condition (2)
{
    int ret;                    // Condition (2)
    return ret;                   // Condition (5)
}
-----------------------------------------------------------------
```

1.4 Workaround

To avoid this problem, do any of the following.

(1) If you are not using functions peculiar to the C++ programming language, compile the program as a C program.
(2) When calling any of the functions that satisfy condition (2) or (4) for the problem to arise, add functions which are not static.
(3) Delete any of the functions that satisfy condition (2) or (4). Note, when deleting a function that satisfies condition (2), the initializing expression for a variable or array that satisfies condition (3) should not include the address of the function that satisfied condition (2).
(4) Change the qualifier in the declaration of any function that satisfies (2) or (4) to extern from static.
(5) Avoid using automatic variables covered by condition (2).

2. Point to Note Regarding the Use of Both Judgment of a Match and Greater or Less Than for Variables (RXC#036)

2.1 Product and Versions Concerned

- C/C++ Compiler Package for RX Family (IDE: CubeSuite+) V2
  Order type name: RTCRX0000CL02WDR and RTCRX0000CL02WNR
  CC-RX compiler V2.00.00 to 2.02.00

- C/C++ Compiler Package for RX Family (without IDE)
  Order type name: RTCRX0000CC02WRR and RTCRX0000CC02WNR V2
  CC-RX compiler V2.01.00 to 2.02.00

2.2 Description

Within a given function, a given if statement or loop might be resolved wrongly when the if statement or other loop-control expression includes a combination of expressions to compare for a match ("!=" or "==") between, and for other types of comparison of a constant and variable.

2.3 Conditions

This problem arises if the following conditions are all met:
(1) Any of -optimize=1, 2 or max is designated.
(2) A comparison expression "!=" or "=="comparing a constant (Note 1) and a variable (Note 2) is present.
Note 1: Includes expressions in which the constant is statically known to be a constant.
Note 2: Includes array variables, structure members, and union members.
(3) An expression that applies "<", ">", "<=", or ">=" to compare the constant and variable covered by (2), within a function which contains a comparison expression of the type described in (2).
(4) The variable in (2) is not modified by volatile.
(5) The comparison expressions covered by (2) and (3) meet any of the

following conditions.
(5-1) The comparison expressions covered by (2) and (3) are connected by "||" or "&&".
(5-2) Loops or "if" statements which include the comparison expressions covered by (2) and (3) are executed in succession.

(6) There is no another expression between the comparison expressions covered by (2) and (3).

Example of condition:

```
-----------------------------------------------------------------------
int ZZZ[3] = {0x7FFF,0x7FFF,0x7FFF};          // Condition (4)
void test(void){
    if(( ( ZZZ[0] <= 180  ) &&              // Condition (3),
                                  // Condition (5-1)
        ( (unsigned int)ZZZ[0] != (unsigned int)0x8000U ) ) ){
                                  // Condition (2) (6)
    }
    .........
}
-----------------------------------------------------------------------
```

2.4 Workaround

To avoid this problem, take any of the following steps.

(1) Designate the -optimize=0 option

(2) Modify the variable in Condition (2) by declaring it as volatile.

(3) In the comparisons covered by (2) and (3), refer to the dummy volatile variable just before the next expression to be executed.

Application example of the workaround: case of the workaround (3)

```
-----------------------------------------------------------------------
volatile int dummy;                    // Declaration of dummy as a
                                  // volatile variable
int ZZZ[3] = {0x7FFF,0x7FFF,0x7FFF};
void test(void){
    if(( ( ZZZ[0] <= 180  ) &&
        (dummy,((unsigned int)ZZZ[0] != (unsigned int)0x8000U ) )) )
    {                             // Workaround (3)
    }
    .........
}
-----------------------------------------------------------------------
```

3. Schedule for Fixing the Problem

- C/C++ Compiler Package for RX Family (IDE: CubeSuite+) and

C/C++ Compiler Package for RX Family (without IDE)

This problem will be fixed in V2.03.00 of CC-RX (we intend to release it on or soon after February 5).
