RENESAS Tool News

RENESAS TOOL NEWS on June 1, 2011: 110601/tn6

Notes on Using the C Compiler Packages for the R8C and M16C Families

When you use the C compiler packages for the R8C and M16C families, 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
- With declaring a function or variable to be extern within another function
- With assigning constants successively to members of a union
- With using an array name as the operand of the sizeof operator
- With comparing a floating constant of 0.0 with NaN (Not a Number)
- With passing a parameter of type float to a K&R-style (the older style) function
- With passing a parameter of type _Bool to a K&R-style (the older style) function
- With using signed char as the type of the condition expression in a switch statement

---

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

1.1 Products and Versions Concerned

- C compiler package for the R32C series
  V.1.01 Release 00 through V.1.02 Release 01A
- C compiler package for the M32C series (M3T-NC308WA)
  V.1.00 Release 1 through V.5.42 Release 00A
- C compiler package for the M16C series and the R8C family (M3T-NC30WA)
  V.2.00 Release 1 through V.5.45 Release 01

1.2 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.

```
[Warning(ccom):sample.c,line 6] underflow in floating value
  converting to integer
    ===>    int j = func((unsigned int)-1.0);
```

1.3 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:
(a) char
Except when compile option -fsigned_char (-fSC) is used in the C compiler for the R32C series.
(b) unsigned char
(c) unsigned short
(d) unsigned int
(e) unsigned long
(f) 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);
   }
}
------------------------------------------------------------------------
```

1.4 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);
    }
}
```

1.5 Schedule of Fixing the Problem

In the C compiler package for the R32C series and the one for the M32C series (M3T-NC308WA), we have no plan to fix this problem. In the C compiler package for the M16C series and the R8C family (M3T-NC30WA), we are fixing this problem in the next release of the V.5X product.

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

2.1 Products and Versions Concerned

- C compiler package for the R32C series
  V.1.01 Release 00 through V.1.02 Release 01A
- C compiler package for the M32C series (M3T-NC308WA)
  V.5.20 Release 1 through V.5.42 Release 00A
- C compiler package for the M16C series and the R8C family (M3T-NC30WA)
  V.5.30 Release 1 through V.5.45 Release 01

2.2 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.

2.3 Conditions

This problem arises if the following conditions are all satisfied:
(1) The following compile options are used:
In the C compiler package for the R32C series:
-Ofile_inline[=<filename>[,...]] (-OFI[=<filename>[,...]])
In the C compiler package M3T-NC308WA or M3T-NC30WA:
-Oforward_function_to_inline (-OFFTI) together with

any of the following:
-O, -O1 through -O5, -OR_MAX (-ORM), -OR, -OS_MAX (-OSM), 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.

2.4 Workaround

Do not use -Ofile_inline[=<filename>[,...]] and -Oforward_function_to_inline (-OFFTI).

2.5 Schedule of Fixing the Problem

In the C compiler package for the R32C series and the one for the M32C series (M3T-NC308WA), we have no plan to fix this problem. In the C compiler package for the M16C series and the R8C family (M3T-NC30WA), we are fixing this problem in the next release of the V.5X product.

3. Problem with Declaring a Function or Variable to Be extern within Another Function

3.1 Products and Versions Concerned

- C compiler package for the R32C series
  V.1.01 Release 00 through V.1.02 Release 01A
- C compiler package for the M32C series (M3T-NC308WA)
  V.5.00 Release 1 through V.5.42 Release 00A
- C compiler package for the M16C series and the R8C family (M3T-NC30WA)
  V.1.00 Release 1 through V.5.45 Release 01

3.2 Description

When a function or variable (B) that has the same name as the function or variable (A) declared at the file scope is declared to be extern within another function (C), no error arises and incorrect code is generated even if the types of both functions or variables (A and B) are different from each other. As a result, the following symptoms arise:

- For the function:
  If a function (B) that has been declared within another function (C) is called, an incorrect argument may be passed to it (B). Or, the return value of this function (B) may be received incorrectly.
- For the variable:
  The variable (B) declared within a function (C) or its adjacent variable may take an incorrect value.

3.3 Conditions

This problem arises if the following conditions are all satisfied:

(1) A variable or function (A) is declared at the file scope.

(2) A variable or function (B) that has the same name as and the different type from the variable or function (A) in (1) is declared to be extern in another function (C).

(3) Within "another function" (C) and at a position where the extern declaration is valid in (2), the variable (B) declared to be extern in (2) is accessed, or the function (B) declared to be extern is called.

Example in function:

```
------------------------------------------------------------------
#include <stdio.h>
static long double func(long double, ...);  /* Condition (1) */
void main(void)
{
   extern long double func(float, ...);    /* Condition (2) */
   long double rtn = func(1.0f);          /* Condition (3) */
   if (rtn != 1.0f) {
      printf("NG¥n");
   } else {
      printf("OK¥n");
   }
}
static long double func(long double x, ...) /* Condition (1) */
{
   return x;
}
```

```
-------------------------------------------------------------
```
In this example, no error arises, and argument x of type float is passed instead of the correct argument of type long double.

Example in variable:

```
-----------------------------------------------------
#include <stdio.h>
static short x = 0;        /* Condition (1) */
static short y = 0;
void main(void)
{
   extern long x;         /* Condition (2) */
   x = 0xffff0000UL;       /* Condition (3) */
   if (y != 0) {
      printf("NG¥n");
   } else {
      printf("OK¥n");
   }
}
-----------------------------------------------------
```
In this example, no error arises, and a change is made to the value of variable y instead of the correct value of variable x.

3.4 Workaround

Match up the type of the function or variable (A) declared at the file scope with that of the function or variable (B) that has the same name as the function or variable (A) and has been declared to be extern within another function (C).

Example in function:

```
-------------------------------------------------------------------
#include <stdio.h>
static long double func(long double, ...);
void main(void)
{
   extern long double func(long double, ...);
                  /* Argument type matched up with long double */
 . . . . . . . . . . . . . . . . . . . . . . . . . .
}
 . . . . . . . . . . . . . . . . . . . . . . . . . .
-------------------------------------------------------------------
```

Example in variable:

```
-------------------------------------------------------------------
#include <stdio.h>
```

```
static short x = 0;
static short y = 0;
void main(void)
{
   extern short x;    /* Variable type matched up with short */
 . . . . . .
}
 . . . . . .
```

3.5 Schedule of Fixing the Problem

In the C compiler package for the R32C series and the one for the M32C series (M3T-NC308WA), we have no plan to fix this problem. In the C compiler package for the M16C series and the R8C family (M3T-NC30WA), we are fixing this problem in the next release of the V.5X product. (In V.6, the problem has already been fixed.)

4. Problem with Assigning Constants Successively to Members of a Union

4.1 Products and Versions Concerned

- C compiler package for the R32C series
  V.1.01 Release 00 through V.1.02 Release 01A
- C compiler package for the M32C series (M3T-NC308WA)
  V.3.10 Release 1 through V.5.42 Release 00A
- C compiler package for the M16C series and the R8C family (M3T-NC30WA)
  V.5.00 Release 1 through V.5.45 Release 01

4.2 Description

If constants are assigned successively to members of a union, assignment may be made in incorrect order.

4.3 Conditions

This problem arises if the following conditions are all satisfied:

(1) Any of the following optimizing options is used:
  -O, -O1 through -O5, -OS, -OR, -OS_MAX (-OSM), and -OR_MAX (-ORM)

(2) In the program exist three or more successive assignment expressions in which constants are assigned.
  Here, constants can be variables or expressions that are replaced with constants by optimization.

(3) In the successive assignment expressions in (2) exist two assignment expressions in which constants are assigned to members of the same union, and these two assignment destinations are adjacent to each other.
  In the two assignment expressions, the first is hereafter called A,

and the second is B.
(4) Between A and B exists assignment expression C, in which a constant is assigned to a member of the same union.
(5) The assignment destinations of A and C are not adjacent.
(6) The assignment destinations of B and C are overlapped.
(7) The assignment destinations of A, B, and C are not qualified to be volatile.

Example:

```
------------------------------------------------------------------
 union{
    unsigned short unim01;
    unsigned short unim02;
    unsigned long  unim05;  /* In this example, uni[0].unim05 and
                               uni[1].unim01 are adjacent */
 } uni[2];               /* Condition (7) */
 int x;
 void func()
 {
    uni[1].unim01 = 0x1111;     /* Condition (2), (3), A, and (5) */
    uni[0].unim02 = 0x2222;     /* Condition (2), (4), and C */
    x          = 0x1234;     /* Condition (2) */
    uni[0].unim05 = 0x55555555;  /* Conditions (2), (3), B, and (6) */
 }
------------------------------------------------------------------
```

In this example, assignment expression B is performed ahead of C.
So the result of assignment in B is modified by C.

4.4 Workaround

Place a dummy asm function immediately before assignment expression B.
Example:

```
-----------------------------------------------------
void func()
{
   uni[1].unim01 = 0x1111;
   uni[0].unim02 = 0x2222;
   x          = 0x1234;
   asm();                  /* Dummy asm() placed */
   uni[0].unim05 = 0x55555555;
}
-----------------------------------------------------
```

4.5 Schedule of Fixing the Problem

In the C compiler package for the R32C series and the one for the M32C series (M3T-NC308WA), we have no plan to fix this problem.

In the C compiler package for the M16C series and the R8C family (M3T-NC30WA), we are fixing this problem in the next release of the V.5X product. (In V.6, the problem has already been fixed.)

5. Problem with Using an Array Name as the Operand of the sizeof Operator

5.1 Products and Versions Concerned

- C compiler package for the R32C series V.1.01 Release 00
- C compiler package for the M32C series (M3T-NC308WA)
  V.1.00 Release 1 through V.5.41 Release 01A
- C compiler package for the M16C series and the R8C family (M3T-NC30WA)
  V.1.00 Release 1 through V.5.45 Release 01

5.2 Description

If the addition expression of an integer and an array name is used as the operand to the sizeof operator, not pointer size but array size is obtained as the result.

5.3 Conditions

This problem arises if the following conditions are all satisfied:

(1) The size of type of the addition expression of an integer constant and an array name is obtained by using the sizeof operator.

(2) In the addition expression in (1), the left operand of the addition operator is an integer constant, and the right one is an array name. Here, the left operand can be an constant expression that is changed to an integer constant by folding constants.

Example:

```
short far arr[30], i;
void func(void)
{
   i = sizeof(0+arr);    /* Conditions (1) and (2) */
}
```

In this example, the sizeof operator does not give a correct value of 4, the size of expression &arr[0], but an incorrect value of 60, the size of the array.

5.4 Workaround

Interchange the left and right operands of the addition operator; that is, put the array name at the left and the integer constant at the right.

Example:

```
------------------------------------------------------------
short arr[30],i;
void test(void)
{
   i = sizeof(arr+0); /* Left and right operands interchanged */
}
------------------------------------------------------------
```

5.5 Schedule of Fixing the Problem

In the C compiler package for the R32C series V.1.02 Release 00 and the one for the M32C series (M3T-NC308WA) V.5.42 Release 00, we have already fixed this problem.
In the C compiler package for the M16C series and the R8C family (M3T-NC30WA), we are fixing this problem in the next release of the V.5X product. (In V.6, the problem has already been fixed.)

6. Problem with Comparing a Floating Constant of 0.0 with NaN (Not a Number)

6.1 Products and Versions Concerned

- C compiler package for the R32C series V.1.01 Release 00
- C compiler package for the M32C series (M3T-NC308WA)
  V.1.00 Release 1 through V.5.41 Release 01A
- C compiler package for the M16C series and the R8C family (M3T-NC30WA)
  V.2.00 Release 1 through V.5.45 Release 01

6.2 Description

If an equality or inequality operation is performed between 0.0 and NaN, the result may be interpreted to be equal.
Here, NaN (Not a Number) is a special value represented by a type of floating-point number and is the value returned after such invalid operations as division of zero by zero, multiplication of zero by an infinity, and addition of positive and negative infinities.

6.3 Conditions

This problem arises if the following conditions are all satisfied:
(1) Any of the following compile options is used:
  -O, -O1 through -O5, -OR, -OS, -OR_MAX (-ORM) and -OS_MAX (-OSM)
(2) An equality or inequality operation is performed
(3) The one operand of the operation in (2) is the expression of a floating constant whose value is NaN.
  Here, the expression of a floating constant can be a variable or

expression that is replaced with a constant by optimization.

(4) The other operand of the operation in (2) is a constant of 0.0.
Here, the constant can be a variable or expression that is replaced with a constant by optimization.

Example:

```
-----------------------------------------------------------------------
int test(void)
{
   float f  = 1.797e+308L + 0.001e+308L;
   float f1 = 1.797e+308L + 0.001e+308L;
                /* Overflow makes variables f and f1 infinities */
   f /= f1;         /* Infinity divided by infinity results in NaN */
   if (f == 0.0f) {  /* Conditions (2), (3) and (4) */
      return 0;
   }
   return 1;
}
-----------------------------------------------------------------------
```

In this example, the result of the equality operation in the if statement is always interpreted to be true in error by optimization.

6.4 Workaround

Change the constant in Condition (4) to a volatile variable with a value of 0.

Example:

```
-----------------------------------------------------------------
volatile float zero = 0.0f;
int test(void)
{
   float f  = 1.797e+308L + 0.001e+308L;
   float f1 = 1.797e+308L + 0.001e+308L;
   f /= f1;
   if (f == zero) {
            /* A volatile variable with a value of 0 used */
      return 0;
   }
   return 1;
}
-----------------------------------------------------------------
```

6.5 Schedule of Fixing the Problem

In the C compiler package for the R32C series V.1.02 Release 00 and the one for the M32C series (M3T-NC308WA) V.5.42 Release 00,

we have already fixed this problem.
In the C compiler package for the M16C series and the R8C family (M3T-NC30WA), we are fixing this problem in the next release of the V.5X product. (In V.6, the problem has already been fixed.)

7. Problem with Passing a Parameter of Type float to a K&R-Style (the older style) Function

7.1 Products and Versions Concerned

- C compiler package for the R32C series
  V.1.01 Release 00 through V.1.02 Release 01A
- C compiler package for the M32C series (M3T-NC308WA)
  V.1.00 Release 1 through V.5.42 Release 00A
- C compiler package for the M16C series and the R8C family (M3T-NC30WA)
  V.1.00 Release 1 through V.5.45 Release 01

7.2 Description

If the prototype declaration for a function taking a parameter of type double is made, the function is defined in the K&R style (the older style), and the parameter is defined as of type float, the value of the parameter cannot be referenced correctly within the function.

7.3 Conditions

This problem arises if the following conditions are all satisfied:
(1) None of the following compile options is used:
  -fdouble_32 (-fD32)
  -OR_MAX (-ORM)
  -OS_MAX (-OSM)
(2) A prototype declaration is made for a function taking a parameter of type double.
(3) The function in (2) is defined in the K&R style (the older style).
(4) In the definition of the function in (3), the parameter in (2) is defined as of type float.
(5) The first access to the parameter in (4) is a read.
(6) In the C compiler package for the R32C series, the parameter in (4) is the third or later parameter.

Example:

```
#include <stdio.h>
float func(int, int, double); /* Condition (2) */
float func(x, y, f)           /* Conditions (3), (4), and (6) */
int x;
int y;
```

```
 float f;                  /* Condition (4) */
 {
    long z = 0;
    if (x) {
       z = 1;
    }
    return f;              /* Condition (5) */
 }
 void main(void)
 {
    float r = func(0, 0, 1.0);
    if (r == 1.0) {
       printf("OK¥n");
    } else {
       printf("NG¥n");
    }
 }
```

In this example, the parameter declared to be of type double in the prototype declaration is converted at the entrance of the function in Condition (3) to a parameter of type float declared in the K&R-style definition. However, the parameter after conversion cannot be read.

7.4 Workaround

Do not define the function in Condition (2) in the K&R style (the older style) but define it including a prototype.

Example:

```
 float func(double f)  /* Defined including a prototype */
 {
. . . . . .
    return f;
 }
```

7.5 Schedule of Fixing the Problem

In the C compiler package for the R32C series and the one for the M32C series (M3T-NC308WA), we have no plan to fix this problem
In the C compiler package for the M16C series and the R8C family (M3T-NC30WA), we are fixing this problem in the next release of the V.5X product. (In V.6, the problem has already been fixed.)

8. Problem with Passing a Parameter of Type _Bool to a K&R-Style

(the older style) Function

8.1 Products and Versions Concerned

- C compiler package for the R32C series
  V.1.01 Release 00 through V.1.02 Release 01A
- C compiler package for the M32C series (M3T-NC308WA)
  V.5.00 Release 1 through V.5.42 Release 00A
- C compiler package for the M16C series and the R8C family (M3T-NC30WA)
  V.5.00 Release 1 through V.5.45 Release 01

8.2 Description

If an even number of 2 or greater is passed as a parameter of type _Bool to a function that has been defined in the K&R style (the older style), and a call is made to this function, the parameter cannot be converted to 1.

8.3 Conditions

This problem arises if the following conditions are all satisfied:
(1) A function is defined in the K&R style (the older style).
(2) The parameter to the function in (1) is of type _Bool.
(3) The first access to the parameter in (2) is a read.
(4) The argument corresponding to the parameter in (2) is an even number of 2 or greater.

Example:

```
#include <stdio.h>
_Bool func(x)        /* Condition (1) */
_Bool x;             /* Condition (2) */
{
   return x;         /* Condition (3) */
}
void main(void)
{
   _Bool x = func(2); /* Condition (4) */
   if (x == 1) {
      printf("OK¥n");
   } else {
      printf("NG¥n");
   }
}
```

In this example, the type of the parameter cannot be converted to _Bool

at the entrance of the function in Condition (1). As a result, a value of 2 or greater is still referenced.

8.4 Workaround

Do not define the function in Condition (2) in the K&R style (the older style) but define it including a prototype.

Example:

```
-----------------------------------------------------
  _Bool func(_Bool x) /* Defined including a prototype */
  {
    return x;
  }
-----------------------------------------------------
```

8.5 Schedule of Fixing the Problem

In the C compiler package for the R32C series and the one for the M32C series (M3T-NC308WA), we have no plan to fix this problem
In the C compiler package for the M16C series and the R8C family (M3T-NC30WA), we are fixing this problem in the next release of the V.5X product. (In V.6, the problem has already been fixed.)

9. Problem with Using Signed Char as the Type of the Condition Expression in a Switch Statement

9.1 Products and Versions Concerned

- C compiler package for the M32C series (M3T-NC308WA)
  V.1.00 Release 1 through V.3.10 Release 3
- C compiler package for the M16C series and the R8C family (M3T-NC30WA)
  V.2.00 Release 1 through V.4.00 Release 2 and
  V.5.10 Release 1 through V.5.45 Release 01

9.2 Description

If the condition expression of a switch statement is of type signed char, the program may not branch to the correct case label.

9.3 Conditions

This problem arises if the following conditions are all satisfied:
(1) The condition expression of a switch statement is of type signed char.
(2) The switch statement in (1) satisfies any of the following:
  (a) The minimum of the case values is -127, the maximum is 127, and between them, the case values continue in step of 1.

(b) The minimum of the case values is -128, the maximum is 126, and between them, the case values continue in step of 1.

(c) The minimum of the case values is -128, the maximum is 127, and the total number of cases is 135 or greater. Here, default is not included in cases.

(3) The value of the condition expression (1) is less than zero except when Condition (2)-(a) is satisfied, and the condition expression takes a value of -128.

Example:

```
-----------------------------------------------------------------
#include <stdio.h>
signed char d = -1;                     /* Condition (3) */
void main( void )
{
   switch( d ) {                       /* Condition (1) */
   case -127 : printf( "NG[-127]¥n"); break; /* Condition (2)-(a) */
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
   case -1 : printf( "OK¥n"); break;
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
   case 127 : printf( "NG[127]¥n"); break;   /* Condition (2)-(a) */
   }
}
-----------------------------------------------------------------
```

In this example, the program branches to default because the value of the case label to be branched is zero-extended when it is converted to the number of the corresponding element in the branch table.

9.4 Workaround

Cast the condition expression in Condition (1) to int.

Example:

```
-----------------------------------------------------------
#include <stdio.h>
signed char d = -1;
void main( void )
{
   switch( (int)d ) {       /* Cast to int */
   case -127 : printf( "NG[-127]¥n"); break;
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
   case -1 : printf( "OK¥n", d); break;
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
   case 127 : printf( "NG[127]¥n"); break;
   }
}
```

9.5 Schedule of Fixing the Problem

In the C compiler package for M32C series (M3T-NC308WA) V.5.00 Release 01, we have already fixed this problem.

In the C compiler package for the M16C series and the R8C family (M3T-NC30WA), we are fixing this problem in the next release of the V.5X product. (In V.6, the problem has already been fixed.)
