RENESAS Tool News

RENESAS TOOL NEWS on October 16, 2011: 111016/tn2

Notes on Using C/C++ Compiler Package for RX Family

When using the C/C++ compiler package for the RX family of MCUs, take note of the following problems:

- With performing bitwise OR (|) operation of -1 or bitwise AND (&) operation of 0 with a function call or a volatile-qualified variable (RXC#016)
- With executing a loop containing the conditional operator or an if statement with a loop counter (RXC#017)
- With updating the value of the variable referenced in a loop containing the conditional operator or an if statement by using any statement except an assignment one (RXC#018)
- With referencing an array that is a member of a structure by using a pointer to the structure (RXC#019)
- With referencing an array element of type char twice or more times in a function (RXC#020)

Here, RXC#XXX at the end of each item is a consecutive number for indexing the problems in the compiler concerned.

---

1. Problem with Performing Bitwise OR (|) Operation of -1 or Bitwise AND (&) Operation of 0 with a Function Call or a Volatile-Qualified Variable (RXC#016)

Versions Concerned:
V.1.00 Release 00 through V.1.01 Release 00

Description:
If bitwise OR (|) operation of -1 or bitwise AND (&) operation of 0 with a function call or a volatile-qualified variable is performed, the function may not be called or the volatile-qualified variable not be referenced.

Conditions:

This problem may arise if the following conditions are all satisfied:

(1) Bitwise OR (|) operation or bitwise AND (&) operation is performed whose operand is of any type of the following:

long long, signed long long, and unsigned long long

(2) The operation in (1) satisfies either of the following:

(2-1) An operand of bitwise OR operation is -1.

(2-2) An operand of bitwise AND operation is 0.

Here, the operand can be a constant substituted for a variable by optimization of constant propagation (including that of external constants qualified to be const).

(3) The other operand of the operation in (2) is any of the following:

(3-1) A variable qualified to be volatile

(3-2) An external variable with the -volatile option used

(3-3) A function call

(3-4) An expression containing any of the operands listed in (3-1), (3-2), and (3-3)

Example:

```
------------------------------------------------------------
long long a;
int sub();
main(){
  long long x;
  x = -1LL;                    // Condition (2-1)
  a = ((long long)(sub()+2)) | x;  // Conditions (1), (2-1), and (3-4)
}
------------------------------------------------------------
```

If the above example is compiled, the expression

((long long)(sub()+2)) | -1LL

is removed in error, and the function call sub() is not made.

Results of compilation:

```
------------------------------------------
_main:
  MOV.L      #0FFFFFFFFH,R4 ; -1LL
  MOV.L      #_a,R5
  MOV.L      R4,[R5]        ; part of variable a
  MOV.L      R4,04H[R5]     ; part of variable a
  MOV.L      #00000000H,R1
  RTS
------------------------------------------
```

Workaround:

To avoid this problem, assign constant -1 or 0 in Condition (2) to a volatile-qualified variable; then use it instead of the constant.

Example modified:

```
-----------------------------------------------------------
 long long a;
 int sub();
 main(){
  volatile long long x;      // Volatile-qualified variable used
                             instead of constant in Condition (2).
  x = -1LL;                  // Condition (2-1)
  a = ((long long)(sub()+2)) | x;  // Conditions (1) and (3-4)
 }
-----------------------------------------------------------
```

2. Problem with Executing a Loop Containing the Conditional Operator or an if Statement with a Loop Counter (RXC#017)

Versions Concerned:

V.1.00 Release 00 through V.1.01 Release 00

Description:

If a loop contains the conditional operator (?:) or an if statement with a loop counter, the conditional operator or the if statement may be incorrectly evaluated.

Here, a loop counter is a variable for controlling the iterations of a loop. It is incremented or decremented by a fixed integer value on each iteration and evaluated to decide when the iteration should be terminated.

Conditions:

This problem may arise if the following conditions are all satisfied:

(1) Neither option -optimize=0 nor -optimize=1 is selected.

(2) In the program exists a loop containing a loop counter.

(3) The initial and maximum values of the loop counter in (2) are constants.
Here, constants can be those substituted for variables by optimization of constant propagation (including that of external constants qualified to be const).

(4) In the loop in (2) exists the conditional operator (?:) or an if statement.

(5) The controlling expression of the conditional operator or the if

statement in (4) satisfies all the following:
(5-1) It is a comparison expression containing a relational operator <, >, <=, or >=.
(5-2) One operand is the loop counter in (2).
(5-3) The other operand is an integer constant equal to or smaller than the maximum value of the loop counter.
Here, an integer constant can be the one substituted for a variable by optimization of constant propagation (including that of external constants qualified to be const).

Example:

```
int main(void) {
  int  j;
  char a1[6],a2[6],a3[6];
  for ( j = 0; j < 6; j++ ){      // Conditions (2), (3), (4)
    if ((j >= 0 && j < 2) || (j >= 4 && j < 6))  // Condition (5)
      a1[j] = 1;
    else
      a1[j] = 0;

    if (j < 4)                          // Condition (5)
      a2[j] = 2;
    else
      a2[j] = 0;

    if (j >= 0 && j < 1)                    // Condition (5)
      a3[j] = 3;
    else
      a3[j] = 0;
  }
}
```

If the above example is compiled with the -cpu=rx600 -speed option used, the problem arises at the expression "j < 2," which exists in the first Condition (5).

Results of compilation:

In the following results of compilation, if j(R5) is 2, R1 becomes -1 at (B). So the program jumps to L11 at (C), and a1[3] is set to 1 at (A) in error.

If correctly executed, the program does not jump at (C), and a1[j] is set to 0 at (D).

```
    ADD       #0FFFFFFE8H,R0,R0
    MOV.L     #00000000H,R5
    MOV.L     #00000002H,R1
    MOV.L     R0,R2
    ADD       #08H,R0,R3
    ADD       #10H,R0,R4
  L11:
    MOV.B     #01H,[R3]    ; (A) a1[3] is set to 1 in error.
    MOV.B     #02H,[R2]
    CMP       #00H,R5
    BLT       L12
    CMP       #01H,R5
    BGE       L12
    MOV.B     #03H,[R4]
  L15:
    ADD       #01H,R5
    ADD       #01H,R4
    ADD       #01H,R3
    ADD       #01H,R2
    CMP       #06H,R5
    BGE       L16
    SUB       #01H,R1      ; (B) If j(R6) is 3, R1 becomes -1.
    BNE       L11          ; (C) Jumps to L11 in error.
    CMP       #04H,R5
    BGE       L20
    MOV.B     #00H,[R3]  ; (D) a1[3] should be set to 0.
    MOV.B     #02H,[R2]
    BRA       L12
  L20:
    CMP       #06H,R5
    BLT       L22
    MOV.B     #00H,[R3]
    BRA       L23
  L22:
    MOV.B     #01H,[R3]
  L23:
    MOV.B     #00H,[R2]
  L12:
    MOV.B     #00H,[R4]
    BRA       L15
  L16:
    MOV.L     #00000000H,R1
    RTSD      #18H
-----------------------------------------------------------------
```

Workaround:
To avoid this problem, use either of the following:
(1) Use option -optimize=0 or -optimize=1.
(2) Qualify the loop counter in Condition (2) to be volatile.

3. Problem with Updating the Value of the Variable Referenced in a Loop Containing the Conditional Operator or an if Statement by Using Any Statement except an Assignment One (RXC#018).

Versions Concerned:
V.1.00 Release 00 through V.1.01 Release 00

Description:
Suppose that the value of the external or static variable is referenced in a loop containing the conditional operator or an if statement. If the value of the above variable is updated by using any statement except the one assigned to the variable, after the loop is exited, the variable may resume the value before updated.

Conditions:
This problem may arise if the following conditions are all satisfied:
(1) Neither option -optimize=0 nor -optimize=1 is selected.
(2) Option -scope is valid.
Regardless of whether -optimize=2 is selected or -optimize is not selected, -scope is valid.
(3) In the program exist two or more functions whose optimizing ranges are divided.
NOTE1: When the -message option is selected, the following message is dispatched for the above functions:
C0101 (I) Optimizing range divided in function "function name"
NOTE2: The function specified in C0101 (I) is hereafter called the function in Condition (3)
(4) In the function in (3) exists a loop containing the conditional operator (?:) or an if statement.
(5) The loop in (4) does not have any loop inside of it and is not an infinite loop.
(6) Declared is an external or static variable that satisfies the following:
(6-1) In the loop in (4), it is neither defined nor referenced.
(6-2) In the function in (3), it is defined or referenced.

(7) The external or static variable in (6) is not qualified to be volatile, and the -volatile option is not used for the variable.
(8) The external or static variable in (6) is updated in the loop in (4) by using any statement except the one assigned to the variable. (An example of the above statement is a function call.)

Example:

```
-------------------------------------------------
int aaa,xxx=0,yyy=0,n;       // Condition (7)
void sub(void);
void func(void)              // Condition (3)
{
  int i;
. . . . . .
  aaa = 0;                   // Condition (6)
. . . . . .
  for (i = 0; i < n ;i++) {   // Conditions (4) and (5)
    if(xxx == yyy){
      sub();                 // Condition (8)
    }
  }
. . . . . .
}

void sub(void)
{
  aaa++;
}
-------------------------------------------------
```

Results of compilation:

In the following results of compilation, the value of aaa loaded into R7 from aaa at (A) is written from aaa to R7 at (C). So when the loop is exited, the value of aaa incremented at (B) resumes the value held at immediately before the loop begins.

```
-----------------------------------------------------------
   _func:
 . . . . . .
        MOV.L     #_aaa,R8
        MOV.L     [R8],R7       ; (A)
        MOV.L     #00000000H,R6
        BRA       L11
   L12:
        MOV.L     #_xxx,R2
        MOV.L     [R2],R4
```

```
        MOV.L     #_yyy,R3
        CMP       [R3],R4
        BNE       L14
    L13:
        BSR       _sub        ; aaa++; . . . (B)
    L14:
        ADD       #01H,R6
    L11:
        MOV.L     #_n,R1
        CMP       [R1],R6
        BLT       L12
    L15:
        MOV.L     R7,[R8]     ; (C)
  . . . . . .
    _sub:
        MOV.L     #_aaa,R4
        MOV.L     [R4],R5
        ADD       #01H,R5
        MOV.L     R5,[R4]
        RTS
```

Workaround:

To avoid this problem, use any of the following:

(1) Use option -optimize=0 or -optimize=1.

(2) Use option -noscope.

(3) Use option -volatile.

(4) Qualify the external or static variable in Condition (6) to be volatile.

(5) Reference the variable in Condition (6) within the loop in Condition (4).

(6) Divide the function in Condition (3) until the message C0101 (I) is not dispatched even if the -message option is selected.

4. Problem with Referencing an Array That Is a Member of a Structure by Using a Pointer to the Structure (RXC#019)

Versions Concerned:

V.1.00 Release 00 through V.1.01 Release 00

Description:

In a structure containing an array as one of its members, if an element of the array is referenced by a pointer to a structure through the array subscripted with a constant, C4098 Internal Error may arise, and the array not be referenced.

Conditions:

This problem may arise if the following conditions are all satisfied:

(1) Any of the options -optimize=1, -optimize=2, and -optimize=max is selected.

(2) A structure or union contains a 4-byte scalar-type array as one of its members.

(3) Another structure or union exists, and one of its members is:
   (a) of type pointer to the structure or union in (2)
   (b) of the 4-byte scalar type

(4) A function takes the structure or union in (3) as a parameter.

(5) The parameter in (4) is passed to the function via a register.
   Note that whether a register is used to pass the parameter to the function or not depends on the type of the parameter.
   For details, see Section 8.2.3, "Rules Concerning Setting and Referencing Parameters," in the User's Manual.

(6) Among the parameters in (5) exists an expression that references the member in (3) in either of the following ways:
   (a) The member in (3) is the pointer to the structure or union in (2) and is referenced with its type not converted.
   (b) The member in (3) is referenced after cast to a pointer to the structure or union in (2).

(7) When the expression in (6) is denoted by C, an expression that is equivalent to C->D[E] exists. Here D is the array as a member in (2) and E is any constant.

(8) The address of the parameter in (4) or the member of the structure or union in (3) is not referred in the function in (4).

Example 1:

Case where C4098 Internal Error arises

```
------------------------------------------------------------
typedef struct { unsigned long ul[2]; } S_A; // Condition (2)
typedef struct {
  S_A *p;
} S_B;              // Conditions (3)-(a) and (5)
int a;
void func(S_B pm)     // Conditions (4) and (5)
{
                  // Condition (8)
   a = pm.p->ul[1];  // Conditions (6)-(a) and (7)
}
```

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

Example 2:

```
  Case where array cannot be referenced
  ------------------------------------------------------------
  typedef struct { unsigned long ul[2]; } S_A; // Condition (2)
  typedef struct {
     S_A *p;
  } S_B;                    // Conditions (3)-(a) and (5)
  int a;
  void func(long m1, S_B pm2) // Conditions (4) and (5)
  {
     S_A *ptr;
     ptr = pm2.p;
                        // Condition (8)
     a = m1 + ptr->ul[0];    // Conditions (6)-(a) and (7)
   }
  ------------------------------------------------------------
  In this example, the code equivalent to (unsigned long)pm2.p is
  generated for ptr->ul[0] in error.
```

Workaround:

```
  To avoid this problem, use any of the following:
  (1) Refer the address of the parameter of the structure
     or union in Condition (4) so that Condition (8) cannot be met.
  (2) Reference the member in Condition (3) through the pointer to the
     structure or union in Condition (2) in the form of a subscripted
     array so that Condition (7) cannot be met.
  (3) Add a dummy member to the structure or union in Condition (3)
     to make its size greater than 17 bytes so that Condition (5)
     cannot be met.
  (4) Use option optimize=0.

  Modification 1 of Example 1:
  ------------------------------------------------------------------
  typedef struct { unsigned long ul[2]; } S_A;
  typedef struct {
    S_A *p;
  } S_B;
  int a;
  void func(S_B pm)
  {
    unsigned long *ptr = pm.p->ul;   // Pointer pointing to array in
                              structure or union in Condition (2)
    a = ptr[1];                    // Accessed by above pointer.
```

```
}
```

Modification 2 of Example 1:

```
 typedef struct { unsigned long ul[2]; } S_A;
    typedef struct {
      S_A *p;
   long long dummy[2];    // Member added to make size of structure
                          or union in Condition (3) greater than
                          17 bytes.
  } S_B;
  int a;
  void func(S_B pm)
  {
   a = pm.p->ul[1];
  }
```

Modification of Example 2 :

```
 typedef struct { unsigned long ul[2]; } S_A; // Condition (2)
 typedef struct {
  S_A *p;
 } S_B;
 int a;
 void func(long m1, S_B pm2)
 {
  S_B *dummy_ptr = &pm2;  // Address of parameter using structure or
                          union in Condition (3) is specified.
   S_A *ptr;
   ptr = pm2.p;
   a = m1 + ptr->ul[0];
 }
```

---

5. Problem with Referencing an Array Element of Type char Twice or More Times in a Function (RXC#019)

Versions Concerned:

V.1.00 Release 00 through V.1.01 Release 00

Description:

If an array element of type char, signed char, or unsigned char is referenced twice or more times in a function, the address of the array element may be wrong.

Conditions:

This problem may arise if the condition group A or B is satisfied:

Condition group A

The following conditions, (A1) to (A6), are all met:

(A1) Neither option -optimize=0 nor -optimize=1 is selected.

(A2) An array of type char, signed char, or unsigned char; or a pointer to char, signed char, or unsigned char is declared.

(A3) An element of the array in (A2) is referenced twice or more times in a function.
Here, the reference to an element can be an indirect reference expression equivalent to an array reference.
*(a+exp) is equivalent to a[exp]

(A4) The subscript expression of the array element in (A3) is an addition or subtraction expression, and one of its operands is a variable and the other is any constant except 0.
In the example of an indirect reference expression in (A3), the subscript expression is exp.
And a constant can be one substituted for a variable by optimization of constant propagation (including that of external constants qualified to be const).

(A5) The variable in (A4) is of type char, signed char, unsigned char, short, signed short, or unsigned short.

(A6) The value of the subscript expression in (A4) is greater than the maximum value expressible in the type of the variable in (A5).

Example condition group A met:

```
-----------------------------------------------------------------
  signed char S,*ary;                  // Condition (A2)
  void func(signed char par)             // Condition (A5)
  {
    if (ary[par+1] > 10) {        // Conditions (A3) and (A4)
      S = ary[par+1];             // Conditions (A3) and (A4)
    }
    return;
  }
-----------------------------------------------------------------
```

In the above example, Condition (A6) is met if par is 127.

Results of compilation:

```
------------------------------------------------------------
 _func:
      MOV.L      #_ary,R4
      ADD        #01H,R1
      MOV.L      [R4],R3      ; ary
      MOVU.B     R1,R5        ; par+1 in Example is sign-extended
                                 in error.
      MOVU.B     [R3,R5],R4   ; ary[]
      CMP        #0AH,R4      ; Compared with final value 10.
      BLE        L12
      MOV.L      #_S,R5
      MOV.B      R4,[R5]      ; S
 L12:
      RTS
------------------------------------------------------------
```

Condition group B

The following conditions, (B1) to (B6), are all met:

(B1) Neither option -optimize=0 nor -optimize=1 is selected.

(B2) A structure or union is declared a member of which is an array of type char, signed char, or unsigned char.

(B3) An element of the array in (B2) is referenced twice or more times in a function.

(B4) The subscript expression indicating the array element in (B3) is any of the following:

(B4-1) A variable

(B4-2) An addition expression; one of its operands is a variable and the other is a constant.

(B4-3) A subtract expression; one of its operands is a variable and the other is a constant.

The constant in either (B4-2) or (B4-3) can be one substituted for a variable by optimization of constant propagation (including that of external constants qualified to be const).

(B5) The variable in (B4) is of type char, signed char, unsigned char, short, signed short, or unsigned short.

(B6) The offset value of the array element in (B3) from the beginning of the structure or union in (B2) is greater than the maximum value expressible in the type of the variable in (B4).

Example condition group B met:

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

```
signed char S;

struct {
  signed char dummy[127];
  signed char mem[8];                // Condition (B6)
} *st;                       // Condition (B2)

void func(signed char par)             // Condition (B5)
{
  if (st->mem[par] > 10) {       // Conditions (B3) and (B4-1)
    S = st->mem[par];           // Conditions (B3) and (B4-1)
  }
  return;
}
```

In the above example, Condition (B6) is met if par is equal to or greater than 1.

Results of compilation:

```
_func:
    MOV.L       #_st,R4
    ADD         #7FH,R1,R5
    MOV.L       [R4],R3      ; st
    MOVU.B      R5,R5        ; par+127 in Example is sign-extended
                               in error.
    MOVU.B      [R3,R5],R4   ; st->mem[]
    CMP         #0AH,R4      ; Compared with final value 10.
    BLE         L15
    MOV.L       #_S,R5
    MOV.B       R4,[R5]      ; S
L15:
    RTS
```

Workaround:

To avoid this problem, use any of the following:

(1) Use option -optimize=0 or -optimize=1.

(2) Qualify any of the following to be volatile:

- The array in Condition (A2)
- The array as a member of the structure or union in Condition (B2)
- The structure or union in Condition (B2)
- The variable as an operand of the subscript expression in (A4) and (B4)

(3) Change the type of the variable in Condition (A4) or (B4) to one different from those specified in Condition (A5) or (B5).

Example modified in condition group B:

```
------------------------------------------------------------
    void func(signed int par)   // char changed to signed int.
------------------------------------------------------------
```

(4) If Condition (B4-1) is met, convert the type of the variable that is an operand of the subscript expression to one different from those specified in Condition (B5)

Example modified

```
------------------------------------------------------------
    st->mem[(signed int)par]   // par cast to signed int.
------------------------------------------------------------
```

6. Schedule of Fixing the Problems

All the above problems have already been fixed in the C/C++ compiler package for the RX family V.1.02 Release 00. For details of the latest version, see RENESAS TOOL NEWS Document No. 111016/tn3 on the Web page at:

http://tool-support.renesas.com/eng/toolnews/111016/tn3.htm

This page will be opened on November 7, 2011.