RENESAS Tool News

RENESAS TOOL NEWS on March 16, 2012: 120316/tn2

Notes on Using C/C++ Compiler Package for RX Family (for High-performance Embedded Workshop)

When using C/C++ Compiler Package for RX Family, take note of the following problems:

- With casting the address for indirectly referencing an element of an array to a pointer whose type is different from that of the array (RXC#021)
- With making function calls immediately before all the exits from a function (RXC#022)

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

1. Problem with Casting the Address for Indirectly Referencing an Element of an Array to a Pointer Whose Type is Different from That of the Array (RXC#021)

Versions Concerned:
V.1.00 Release 00 through V.1.02 Release 00

Description:
If the address of an element of an array is cast to a pointer whose type is different from that of the array, and a constant is added to the cast result, an incorrect code is read out from the address.

Conditions:
This problem may arise if the following conditions are all satisfied:
(1) A pointer variable or an array whose dimension is greater than one is defined.
(2) The address of any element of the following array that is referenced with the subscript is given with the & operator:
(a) An array with the same type as that of any object pointed to

        by the pointer variable in (1), or
    (b) The array in (1), whose dimension is greater than one
(3) The address in (2) is cast to a pointer whose type is different
    from that of the pointer variable or the array in (1).
(4) Either of the following is added to the cast result in (3):
    (a) Any constant except 0
    (b) The result of any operational expression except
        a constant of 0
(5) The address obtained in (4) is used to read the memory by indirect
    referencing.
(6) Items (2) through (5) are represented by a single expression.
(7) The result in (4)-(b) is not 0 if Condition (4)-(b) is satisfied.

Example 1:

```
-----------------------------------------------------------------
long data01[3] = { 0x11223344, 0x55667788, 0x99aabbcc };
char a, b, c;
void func01(void)
{
    long *p = data01;     /* Pointer variable in Condition (1) */
    a = *((char *)&p[2] + 0); /* Condition (4)-(a) unsatisfied
                               because constant 0 is added */
    b = *((char *)&p[2] + 1); /* Conditions (2), (3), (4)-(a),
                               (5), and (6) */
    c = *((char *)&p[2] + 2); /* Conditions (2), (3), (4)-(a),
                               (5), and (6) */
}
-----------------------------------------------------------------
```

Though the correct results are a = 0xcc, b = 0xbb, and c = 0xaa,
both b and c take the same value that a does.
(The above values are those in the little endian system.)

Example 2:

```
-----------------------------------------------------------------
short data02[2][3] =       /* 2-dimensional array in Condition (1) */
{{0x1122,0x3344,0x5566},
{0x7788,0x99aa,0xbbcc}};
char a, b, c;
void func02(int x)
{
   a = *((char *)&data02[1] + 0); /* Conditions (2), (3), (4)-(a),
                                   (5), and (6) */
   b = *((char *)&data02[1] + 2); /* Conditions (2), (3), (4)-(a),
                                   (5), and (6) */
   c = *((char *)&data02[1] + x); /* Conditions (2), (3), (4)-(b),
                                   (5), (6), and (7) */
}                              /* See NOTE */
-------------------------------------------------------------
```
NOTE:
  Condition (7) is satisfied only when variable x is not 0. For example, when variable x is 1, b takes the same value that a does though the correct results are a = 0x88, b = 0xaa, and c = 0x77.
  And when x takes any value except 0, c takes the same value that a does though the values of c and a must be different from each other.
  (The above values are those in the little endian system.)

Workarounds:
  To avoid this problem, do either of the following:
  (1) Assign the cast result in Condition (3) to a temporary variable, and then use it when an offset is added.
    Workaround for Example 1:
```
-------------------------------------------------------------
char *temp = (char *)&p[2];
*(temp + 1);
-------------------------------------------------------------
```

  (2) Insert "+ 0" immediately before the + operator of the addition expression in Condition (4). However, do not enclose the inserted 0 and the addition expression in parentheses.
    Workaround for Example 1:
```
-------------------------------------------------------------
*((char *)&p[2] + 0 + 1 )
-------------------------------------------------------------
```

    Example where problem is not avoided:
```
-------------------------------------------------------------
*((char *)&p[2] + (0 + 1) )
-------------------------------------------------------------
```

2. Problem with Making Function Calls Immediately before All the Exits from a Function (RXC#022)

Versions Concerned:
  V.1.00 Release 00 through V.1.02 Release 00

Description:
  If two functions A and B are defined contiguously, and calls are made to function B or other functions immediately before the exits from

function A, calls to function B may be deleted.

Conditions:
This problem may arise if the following conditions are all satisfied:
(1) The optimizing option optimzie=2 or optimize=max is used.
(2) Two functions A and B are defined contiguously.
(3) Function A comes with two or more exits: the end of the function and its return statements
(4) Immediately before all the exits from function A, function calls are made.
(5) One or more of the function calls in (4) are made to function B.

Example:
```
----------------------------------------------------------------
void FuncB();
void FuncC(),FuncD();
long long funcLL();
void FuncA()                    /* Function A */
{
  long long v1 = funcLL();
  if (v1) {
     FuncB();                  /* Conditions (4) and (5)
                            /* This call to FuncB() deleted */
     return;                  /* Condition (3) */
  } else if (v1==1){
     FuncC();                  /* Condition (4) */
     return;                  /* Condition (3) */
  } else {
     FuncD();                  /* Condition (4) */
     return;                  /* Condition (3) */
  }
}
void FuncB(){ }                  /* Condition (2) */
----------------------------------------------------------------
```
Note that when two or more calls have been made to function B, only the ones called immediately before return statements are deleted.

Result of compilation
```
----------------------------------------------------------------
_FuncA:
        CMP        #00H,R1
        BEQ        L12
L11:
        ADD        #08H,R0
        ; <==  BRA _FuncB deleted.
```

```
L12:
. . . . . . . . . . . . . . . . . . . . .
        BEQ        L15
        ADD        #08H,R0
        BRA        _FuncC
L15:
        ADD        #08H,R0
        BRA        _FuncD
_FuncB:
. . . . . . . . . . . . . . . . . . . . .
----------------------------------------------------------------
```

Workarounds:

To avoid this problem, do any of the following:

(1) Use optimzie=0 or optimize=1.

(2) Immediately after Function A, define a dummy function.
For example, define a dummy function Dummy immediately before FuncB.

Example of Workaround (2):

```
----------------------------------------------------------------
void FuncB();
void FuncC(),FuncD();
long long funcLL();
void FuncA()
{
 long long v1 = funcLL();
 if (v1) {
   FuncB();
   return;
 } else if (v1==1){
   FuncC();
   return;
 } else {
   FuncD();
   return;
 }
}
void DummyFunc(){ } // Dummy function defined.
void FuncB(){ }
----------------------------------------------------------------
```

(3) Immediately before an exit from function A, place a dummy instruction. For example, place the built-in function nop().

Example of Workaround (3):

```
------------------------------------------------------------------
#include      // for nop();
void FuncB();
void FuncC(),FuncD();
long long funcLL();
void FuncA()
{
 long long v1 = funcLL();
 if (v1) {
    FuncB();
    return;
 } else if (v1==1){
    FuncC();
    return;
 } else {
    FuncD();
    nop();   // Built-in function nop() placed as dummy.
    return;
 }
}
void FuncB(){ }
------------------------------------------------------------------
```

3. Schedule of Fixing Problems

The above problems have already been fixed in V.1.02 Release 01.
For details of V.1.02 Release 01, see RENESAS TOOL NEWS Document No. 120316/tn3. You can also see this news on the Web page at:
https://www.renesas.com/search/keyword-search.html#genre=document&q=120316tn3

This page will be published on March 21, 2012.