RENESAS Tool News

RENESAS TOOL NEWS on November 16, 2013: 131116/tn1

Note on Using CubeSuite+ CC-RX Compiler

When using CubeSuite+ CC-RX compiler, take note of the following problem:

- When the storage area of a structure variable is not allocated correctly (RXC#025)

NOTE:
The number at the end of the above item is for indexing the problem in this compiler.

---

1. Product and Version Concerned

CubeSuite+ CC-RX Compiler V2.00.00 and V2.00.01

2. Description

Due to optimization by the compiler, the area which is allocated for storing a structure variable to which the initial value is set may become smaller than the required size and a code which does not allocate the area correctly may be generated.
In this case, the program which manipulates that structure cannot be executed correctly.

3. Conditions

This problem may arise if the following conditions are all met:
(1) There is a structure that has a union member.
(2) Padding is performed between the union which is a member of the structure in (1) and the member located immediately before it. (NOTE 1.)
(3) The alignment number of the union which is a member of the structure in (1) does not match. (NOTE 2.)
(4) The initial value is set for the union which is a member of the structure in (1).
(5) The alignment number of the member of the structure for which the initial value is set is smaller than the alignment number of the union.

NOTES:
1. When the ending location of the area for the prior member does

not match the alignment number (boundary adjustment number) of the union, padding is performed to achieve alignment.

2. For a structure that does not include even one member whose basic size is one or two bytes, all of the alignment numbers are 4. Therefore, condition (3) is not met, and this problem is irrelevant.

Example:

```
---------------------------------------------------------------------
typedef struct {
   char x;   // Member immediately before the union
             // (alignment number is 1)
             // The ending location of the area for member x which is
             // immediately U1 is before union at the location of
             // (beginning of structure + 1 byte).
             // Accordingly, padding for three bytes is inserted
             // between x and U1: Condition (2)
   union {       // Alignment number of the union is 4: Condition (1)
      char m;    // Alignment number of member m is 1: Condition (3)
      long n;    // Alignment number of member n is 4: Condition (3)
   } U1;
} Str;

Str S1 = {
   0x77,
   {
      0x88  // The initial value of the union is set to member m
   }        // whose alignment number is smaller than the union's
};          // alignment number of 4: Conditions (4) and (5)
---------------------------------------------------------------------
```

In the above description, padding is not output after member x in structure S1. As a result, the latter four bytes corresponding to member U1 will not be placed in the correct address.

4. Workarounds

To avoid this problem, use either of the following methods described below:

(1) Add a dummy variable instead of padding immediately before the relevant union.

Example:

```
   ---------------------------------------------------------------
   typedef struct {
      char  x;
      char  dummy1; // Add a dummy variable that will substitute for
      short dummy2; // padding of three bytes.
```

```
    union {
      char m;
      long n;
    } U1;
  } Str;
```

---

(2) Set the alignment number of the member in the relevant structure to 1 using the pack option or the #pragma pack (see NOTE) facility.

NOTE:
#pragma pack cannot be applied when the structure is an auto variable.

(3) Shift the order of the members in the structure so that the padding immediately before the relevant union becomes unnecessary.
If the parent structure has only a single union that meets condition (2), move that union to the beginning of the structure.
Example:

```
typedef struct {
  union {      // Move the declaration of the union to a location
    char m;   // where no padding is inserted immediately
    long n;   // before it. (Alignment is unnecessary if placed
  } U1;       // at the beginning of the structure)
  char x;
} Str;
```

---

(4) Change the setting so that the initial value of condition (4) is set for the member with the largest alignment number in the union.
(a) For C89 and C++ (when compiling without lang=c99 selected)
Example:

```
typedef struct {
  char  x;
  union {
    long n;  // Declare the member with the largest
    char m;  // alignment number in the union at the
  } U1;      // beginning so that the initial value is set
} Str;       // for that member.

Str S1 = {
  0x77,
  {
```

```
        0x88    // NOTE: For big endian, the initial value
    }           //      needs to be 0x88000000.
};
-----------------------------------------------------------------
```

(b) For C99 (when compiling with lang=c99 selected)

Example:

```
-----------------------------------------------------------------
typedef struct {
    char  x;
    union {
        char m;
        long n;
    } U1;
} Str;

Str S1 = {
    0x77,
    {
        .n = 0x88  // To make an expression to initialize
                   // the member with the largest alignment
                   // number in the union, specify
                   // initialization with the member name.
    }           // NOTE: For big endian, the initial value
};              //      needs to be 0x88000000.
-----------------------------------------------------------------
```

5. Schedule for Fixing the Problem

This problem has been fixed in CubeSuite+ CC-RX compiler V2.01.00.
For details on V2.01.00, refer to RENESAS TOOL NEWS Document No. 131116/tn2.
The details can also be referenced from the following URL.

https://www.renesas.com/search/keyword-search.html#genre=document&q=131116tn2
This page will be published on November 20.
