Tool News

RENESAS TOOL NEWS on August 5, 2010: 100805/tn1

Notes on Using the C/C++ Compiler Package V.1.00 Release 00 for the RX Family of MCUs

When you use the C/C++ compiler package V.1.00 Release 00 for the RX family of MCUs, take note of the six problems described below.

1. Problem with Initializing a Member/Element of a Structure-Type or Array-Type Variable Whose Storage Class Is 'auto' or 'static referenced within the function'

1.1 Description

If a member/element of a structure-type or array-type variable whose storage class is 'auto' or 'static referenced within the function' is initialized with the address of another variable, the value of the variable used for initialization may not correctly be read out.

1.2 Conditions

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

(1) Optimizing option -optimize=1, -optimize=2, or -optimize=max
is selected.
(2) A member/element of a structure-type or array-type variable whose
storage class is 'auto' or 'static referenced within the function'
is initialized with the address of another variable.
(3) The member initialized in (2) is the second or later member of
a structure or an array.
(4) A value is assigned to the variable used for the initialization
in (2) by using the member in (3) of the structure or the array.
(5) After the assignment in (4), the value of the variable used for
the initialization in (3) is referenced.

Example:

```
-----------------------------------------------
// ccrx -cpu=rx600 -optimize=1
#include <stdio.h>

int x;
typedef struct ST{
  int a;
  int* b;
}St;

void main(){
  St st = {     // Condition (2)
    0,
    &x          // Condition (3)
  };

  x   = 1;
  *st.b  = 2;   // Condition (4)

  if( x == 2 ){ // Condition (5)
    // Because all the conditions satisfied, value x is misread
    // in this evaluation.
  }
  else{
    // Because result of evaluation of if statement is false,
    // else block executed.
  }
}
-----------------------------------------------
```

1.3 Workarounds

To avoid this problem, use either of the following ways:

(1) Define and initialize the structure-type variable outside
the function.

Example modified:

```
--------------------------------------------------
int x;
typedef struct ST{
  int a;
  int* b;
}St;

St st = {  // Define and initialize variables as global; not local.
  0,
  &x
};
void main(){
  x = 1;
. . . . . . . . . . . . . . . . . . . . . . .
--------------------------------------------------
```

(2) Define the member involved of the structure or the array and
then assign the address of a variable to it.

Example modified:

```
--------------------------------------------------
void main(){
  St st;      // When structure-type variables defined,
             // do not initialize them.
  st.a = 0;
  st.b = &x;  // Address of a variable is assigned.
. . . . . . . . . . . . . . . . . . . . . . .
--------------------------------------------------
```

2. Problem with Declaring an Object Whose Type is Qualified by the __evenaccess or const Qualifier

2.1 Description

After an object whose type is qualified by the __evenaccess or const
qualifier is declared, the qualifier may be neglected.

2.2 Conditions

This problem arises if any of the following conditions is satisfied:

(1) An array whose type is qualified by __evenaccess is declared
in the C89 or C99 language.

(2) A type declarator is defined by typedef with the declarator's type
being qualified by const; then a variable is declared with the

type declarator.

(3) A type declarator is defined by typedef with the declarator's type being qualified by const; then a variable is declared with the type declarator qualified by const.

Example 1 (Condition (1) satisfied):

```
------------------------------------------------
__evenaccess char ary1[4];    // Condition (1)
void func1(void)
{
 // Qualifier __evenaccess is neglected in compilation. So assignments
 // to ary1[0] to ary1[3] below are brought into one assignment.
 ary1[0] = 0;
 ary1[1] = 1;
 ary1[2] = 2;
 ary1[3] = 3;
}
------------------------------------------------
```

Example 2 (Conditions (2) and (3) satisfied):

```
------------------------------------------------
typedef const long CARRAY[2];
CARRAY ary2 = { 0, 1 };          // Condition (2)
const CARRAY ary3 = { 2, 3 };    // Condition (3)
// Qualifier const is neglected in compilation. So ary2 and ary3 are
   stored in Section D.
------------------------------------------------
```

2.3 Workarounds

2.3.1 When Condition (1) Satisfied

To avoid the problem use either of the following ways:

(1) Compile treating source code as the C++ language.

(2) Access the elements of the array by using the pointer of the same type as them.

Example 1 modified:

```
------------------------------------------------
__evenaccess char ary1[4];
void func1(void)
{
  __evenaccess char *ptr1 = ary1;
  // Elements of array accessed by pointer ptr1.
  ptr1[0] = 0;
  ptr1[1] = 1;
  ptr1[2] = 2;
  ptr1[3] = 3;
}
------------------------------------------------
```

2.3.2 When Condition (2) or (3) Satisfied

To avoid the problem, do not use the const qualifier in the typedef statement, but use in the declaration of the variable whose type is to be qualified.

Example 2 modified:

```
------------------------------------------------
typedef long CARRAY[2];       // Qualifier const not used, but
const CARRAY ary2 = { 0, 1 }; // used in declaration of variables.
const CARRAY ary3 = { 2, 3 };
------------------------------------------------
```

3. Problem with Using a Loop-Controlling Variable of Type int That is 1 or 2 Bytes Long As an Argument to a Function

3.1 Description

If you use a loop-controlling variable of type int that is 1 or 2 bytes long as an argument passed to a function to be called, the argument cannot correctly be saved on the stack.

3.2 Conditions

This problem arises if the following conditions are all satisfied:

(1) Optimizing option -optimize=2 or -optimize=max is selected; or -optimize is not selected.

(2) In a loop exists a loop-controlling variable of type int that is 1 or 2 bytes long.

(3) The loop-controlling variable in (2) is not qualified to be volatile.

(4) A function to be called takes a list of two or more parameters. To one of them, the loop controlling variable in (2) is passed as an argument. Here, the argument satisfies all of the following four conditions:

(4-1) The type of the argument (that of the loop-controlling variable) is compatible with the prototype of the function. (see NOTE 1.)

(4-2) The argument is passed via the stack. (see NOTE 2.)

(4-3) The argument is not passed to the last of the parameters.

(4-4) The parameter that receives the loop-controlling variable and the one next to it are not variable parameters.

NOTES:

1. Both are of type int; and the size of type and the presence

   or absence of signs are the same.
  2. For the rules of allocating parameters, see Section 8.2.3, "Rules Concerning Setting and Referencing Parameters," in the User's Manual.

(5) Any one of the following three conditions is satisfied:
  (5-1) The offset value of the argument in (4) on the stack is not a multiple of 4. (See NOTE 2 above.)
  (5-2) The parameter next to the one receiving the argument in (4) is of type int and 1 byte or 2 bytes long.
  (5-3) Conditions (5-1) and (5-2) are not satisfied, and -endian=big is selected.

Example:

```
-----------------------------------------------
typedef unsigned char UC;
void func2(int, int, int, int, int, UC, UC, int);
void foo2(void)
{
  UC loop = 0;                // Condition (3)
  while (loop < 8) {          // Condition (2)
    func2(1,2,3,4,5,loop,6,7);   // Conditions (4) and (5-2)
    ++loop;
  }
}
-----------------------------------------------
```

Result of expanding Example above:

```
-----------------------------------------------
Block for calling function func2(1,2,3,4,5,loop,6,7);
  MOV.L  #00000007H,R5
  PUSH.L R5
  SUB    #04H,R0
  MOV.L  #00000005H,R15
  MOV.B  #06H,01H[R0]      ; 7th parameter
  MOV.L  R6,[R0]           ; Content of 7th parameter overwritten.
  PUSH.L R15
  MOV.L  #00000004H,R4
  MOV.L  #00000003H,R3
  MOV.L  #00000002H,R2
  MOV.L  #00000001H,R1
  BSR    _func2
-----------------------------------------------
```

3.3 Workaround

To avoid this problem, use any of the following ways:
(1) Select -optimize=0 or -optimize=1.
(2) Make the loop-controlling variable 4 bytes long.

Example:

```
----------------------------------------------------
void foo2(void)
{
  unsigned long loop = 0; // Made 4 bytes long.
  while (loop < 8) {
. . . . . . . . . . . . . . . . . . . . . . . .
----------------------------------------------------
```

(3) Qualify the loop-controlling variable to be volatile.

Example:

```
----------------------------------------------------
void foo2(void)
{
  volatile UC loop = 0; // volatile-qualified
  while (loop < 8) {
. . . . . . . . . . . . . . . . . . . . . . . .
----------------------------------------------------
```

(4) Assign the loop-controlling variable to another variable of a different type; then pass this variable as an argument to the function.
(5) Cast the loop-controlling variable to a different type; then pass it as an argument to the function.

4. Problem with Converting the Value of at Least One Operand in a Bitwise AND Operation between Two Integral-Type Variables

4.1 Description

In a bitwise AND operation (operator &) between two integral-type variables whose sizes are 1 byte and 2 bytes respectively, if the value of at least one operand--be it left or right to the & operator--has been converted to a different type with a different size, zero extension may not correctly be made.

4.2 Conditions

This problem may arise if the following conditions are all satisfied:
(1) Optimizing option -optimize=2 or -optimize=max is selected.
(2) Option -speed is not selected.
(3) A bitwise AND operation is performed between two operands that

are of type unsigned char and type unsigned short respectively.
(4) In this operation, the value of at least either one of two operands
is converted to a different type with a different size by using
a cast operator.

Example:

```
-----------------------------------------------
int a;
unsigned short b;
void func()
{
 a = (unsigned char)a & b; // Condition (3) and (4) satisfied because
                   // variable 'a' converted to type unsigned
                   // char and then ANDed with unsigned short b.
}
-----------------------------------------------
```

Result of expanding Example above:

```
-----------------------------------------------
_func:
 MOV.L  #_a,R2
 MOV.L  [R2],R5     ; Value of variable 'a' stored in R5.
 MOV.L  #_b,R3
 AND    [R3].UW,R5  ; An operand of bitwise AND operation not
                ; zero-extended, values of upper 3 bytes
                ; are incorrect.
 MOV.L  R5,[R2]
-----------------------------------------------
```

4.3 Workarounds

To avoid this problem, use either of the following ways:
(1) Select -optimize=0 or -optimize=1.
(2) Select -speed.

5. Problem with Using a Structure One of Whose Members Is a Bit Field As a Return Value of a Function

5.1 Description

Suppose that a bit field is a member of a structure, this structure
is a member of another structure, and the second structure is used
as the return value of a function. If the function has been called,
by using the return value of the function the bit field cannot
directly be referenced.

5.2 Conditions

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

(1) A structure has a bit field as one of its members.

(2) Another structure or any of the structures nested beyond 2 levels has the structure in (1) as one of its members.

(3) A call is made to a function whose return value is the structure in (2).

(4) The bit field in (1) included in the return value of the function to whom a call is made in (3) is directly referenced.

Example:

```
----------------------------------------------------
#include <stdio.h>
typedef struct{ struct{ int a : 8; }st; }ST;  // Conditions (1) and (2)
ST func(ST st){ return st; }                  // Condition (3)
void main(){
  int ret = 0;
  ST St;
  St.st.a = 1;
  ret = func( St ).st.a;     // Conditions (3) and (4)
                             // Correct value not returned to ret.
  if( ret == 1 ){
    printf( "OK¥n" );
  }
  else{
    printf( "NG(%d)¥n", ret );
  }
}
----------------------------------------------------
```

Result of expanding Example above:

```
----------------------------------------------------
_main:
  . . . . . . . . . . . . . . . . . . . . . . .
  BSR       _func
  MOV.L     R1,08H[R0]
  MOV.B     08H[R0],R4  ; Incorrect: must be MOVU.B  08H[R0],R5
  MOVU.B  [R4],R5
  CMP      #01H,R5
  BEQ      L12
  . . . . . . . . . . . . . . . . . . . . . . .
----------------------------------------------------
```

5.3 Workaround

Save the return value of the function on the temporary structure used for the return value; then reference the bit field.

In the above example, change

```
   ret = func( St ).st.a;
```

to

```
   ST temp;
   temp = func( St );
   ret = temp.st.a;
```

6. Problem with Assigning Constants to Members of a Structure by Using Indirect Reference via Pointers

6.1 Description

The order of execution of assigning constants to members of a structure by using indirect reference via pointers and making function calls may be interchanged.

6.2 Conditions

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

(1) Optimizing option -optimize=2 or -optimize=max is selected.

(2) A structure has two or more members, and the areas of two members are adjacent on memory.

(3) The two adjacent members in (2) are of type int with the same size, and the size is 1 or 2 bytes.

(4) By using pointers to the structure in (2), constants are assigned to the two adjacent members.

(5) Function calls are made.

(6) The two assignments of constants in (4) and the function calls in (5) are written in the same block.

Example:

```
------------------------------------------------
struct STR {              // Condition (2)
  unsigned short  member1;  // Condition (3)
  unsigned short  member2;  // Condition (3)
} s;

void main(struct STR * ps, int flg)
{
  if (flg)
  {
    ps->member1 = 7;       // Conditions (4) and (6)
    ps->member2 = 16;      // Conditions (4) and (6)

    func1(ps);             // Conditions (5) and (6)
  func1(ps);
 }
}
----------------------------------------
```

Result of expanding Example above:

```
----------------------------------------
_main:
 PUSH.L     R6
 MOV.L      R1,R6
 CMP        #00H,R2
 BEQ        L12
L11:
 MOV.L      R6,R1
 BSR        _func1
 MOV.L      R6,R1
 BSR        _func1
 MOV.L      #00100007H,[R6] ; Result of assignments to ps->member1
                            ; and ps->member2 is moved after call
                            ; to func1(), so this code is incorrect.
 RTSD       #04H,R6-R6
----------------------------------------
```

6.3 Workarounds

To avoid this problem, use any of the following ways:

(1) Select -optimize=0 or -optimize=1.

(2) Qualify the structure or its two members in Condition (2) to be volatile.

(3) Select -noschedule.

(4) Apply -optimize=0, -optimize=1, or -noschedule to the function involved by using #pragma option.

7. Schedule of Fixing the Problem

These six problems have already been resolved in V.1.00 Release 01.
For details of V.1.00 Release 01, see RENESAS TOOL NEWS Document No. 100805/tn2. This item of news is also accessible at:

https://www.renesas.com/search/keyword-search.html#genre=document&q=100805tn2
The Web page will be opened from August 17.