RENESAS Tool News

RENESAS TOOL NEWS on February 1, 2014: 140201/tn1

Notes on Using the C/C++ Compiler Package for the SuperH RISC engine MCU Family V.7 through V.9

When using the C/C++ compiler package V.7 through V.9 for the SuperH RISC engine family of MCUs, take note of the following problems:

- With accessing the multiple array elements using the same subscript (SHC-0090)
- With using instruction scheduling and intrinsic functions (SHC-0091)
- With inlining a function which has a static variable of the structure type or the union type within a function (SHC-0092)
- With using the standard library function strcpy() (SHC-0093)

Here, SHC-XXXX at the end of each item is a consecutive number for indexing the problem in the compiler concerned.

---

1. Problem with Accessing the Multiple Array Elements Using the Same Subscript (SHC-0090)

1.1 Product and Versions Concerned

V.7.0B through V.9.04 Release 01

1.2 Description

When the multiple array elements in the same loop are accessed by using the same subscript, the address of the array element may be wrong.

1.3 Conditions

This problem may arise if the following conditions are all met:
(1) Neither option -optimize=0 nor -optimize=debug_only is used.
(2) In the program exists a loop containing a loop counter.
(3) An array element in the loop in (2) is accessed. (NOTE 1.)
(4) An array element of array that is different from (3) is accessed in the loop (2). (NOTE 1.)
(5) Each array element of (3) and (4) is accessed using the same pointer.
(6) The subscript expressions for accessing the array elements in (3)

and in (4), satisfy either (6-1) or (6-2). (NOTE 2.)
(6-1) A same constant value. (NOTE 3.)
(6-2) The following conditions are all met:
- A linear expressions of a loop counter in (2)
- Types is the same as the loop counter
- Increments is the same as the loop counter

NOTES:
1. Including that an array element accessed using an indirect reference expression (*(ary + index)).
2. In an indirect reference expression *(ary + index), the subscript expression is "index" when "ary" is array variable.
3. This condition is also met when the subscript expression becomes a substituted constant.
Example: int index = 0; ary[index];

1.4 Example

```
----------------------------------------------------------------
long S=0;
void func(short*xxx, short *yyy) {
long i, t1, t2;
short *pt;
   for( i = 0 ; i < 2 ; ++i) {      // Condition (2)
      pt = xxx;                  // Condition (3)
      t1 = pt[i];                // Conditions (5) and (6-2)
      pt = yyy;                  // Condition (4)
      t2 = pt[i];                // Conditions (5) and (6-2)

      S += (t1 + t2);
   }
}
----------------------------------------------------------------
```

yyy[i] is processed as xxx[i] in error. xxx[i] + xxx[i] is added to S.

1.5 Workarounds

To avoid this problem, do any of the following:
(1) Use option -optimize=0 or -optimize=debug_only.
(2) Qualify either of the following to be volatile.
- The loop counter in Condition (2)
- The array in Condition (3)
- The array in Condition (4)
- The pointer variable in Condition (5)

(3) When the Condition (6-2) is met, change either of the subscript expressions for accessing the array elements to a linear expression of a loop counter with a different type.

(4) Access the array element in Condition (4) using a pointer variable different from a pointer variable for accessing the array elements in Condition (3).
(5) Access the array element either in Condition (3) or in Condition (4) not using a pointer variable.

1.6 Example

Example of modified workaround (3):

```
-----------------------------------------------------------
long i, t1, t2;
short *pt;
unsigned long k;                        // Workaround (3)
   for( i = 0, k = 0 ; i < 2 ; ++i, ++k) {   // Workaround (3)
      pt = xxx; t1 = pt[k];              // Workaround (3)
      pt = yyy; t2 = pt[i];
      S += (t1 + t2);
   }
-----------------------------------------------------------
```

Example of modified workaround (4):

```
-----------------------------------------------------------
short *xxx, *yyy, *pt, *pt2;   // Workaround (4)
pt = xxx;   t1=pt[i];
pt2 = yyy;  t2=pt2[i];        // Workaround (4)
-----------------------------------------------------------
```

Example of modified workaround (5):

```
-----------------------------------------------------------
short *xxx,yyy[10], *pt;
pt = xxx;   t1=pt[i];
t2=yyy[i];                  // Workaround (5)
-----------------------------------------------------------
```

2. Problem with Using Instruction Scheduling and Intrinsic Functions (SHC-0091)

2.1 Product and Versions Concerned

V.9.03 Release 00 through V.9.04 Release 01

2.2 Description

When the instruction scheduling and intrinsic functions are used, instructions after the intrinsic function may not be generated correctly.

2.3 Conditions

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

(1) Either option -cpu=sh2a or -cpu=sh2afpu is used.
(2) Neither option -optimize=0 nor -optimize=debug_only is used.
(3) Option -schedule=0 is not used.
(4) Any of the following intrinsic functions are used.
- bset()
- bclr()
- bcopy()
- bnotcopy()

2.4 Example

When option -cpu=sh2a is used:

```
------------------------------------------------------
#include
void func()
{
   volatile char a[100];
   volatile char a100,a101,a102;
   a[55] = 0;
   a100 = 0;
   a101 = 0;
   bset((unsigned char *)(0xfffe3886),0); // Condition(4)
   a102 = 0;
}
------------------------------------------------------
```

Results of compilation:

```
------------------------------------------------------
MOV        #100,R2
MOV        #0,R1
ADD        R15,R2
MOV.B      R1,@(55:12,R15)
MOV.B      R1,@R2          ; On a100, 0 stored.
MOV        R1,R0
MOV.B      R0,@(4,R2)      ; On a101, 0 stored.
MOVI20     #-116602,R2     ; H'FFFE3886
BSET.B     #0,@(0,R2)
MOV.B      R0,@(8,R3)      ; R3 is referenced in error.
                           ; And 0 is not stored on a102.
RTS
ADD        #112,R15
------------------------------------------------------
```

2.5 Workaround

To avoid this problem, do any of the following:
(1) Use option -optimize=0 or -optimize=debug_only.

(2) Use option -schedule=0.
(3) Modify the program not using intrinsic function in Condition(4).

2.6 Example

Example 1 of modified workaround (3):
bset()
[Original]

```
------------------------------------------------------
   bset(&a,0)
------------------------------------------------------
```

[Modified]

```
------------------------------------------------------
   a|=0x01;
------------------------------------------------------
```

Example 2 of modified workaround (3):
bclr
[Original]

```
------------------------------------------------------
   bclr(&a,3)
------------------------------------------------------
```

[Modified]

```
------------------------------------------------------
   a&=~(0x01 << 3);
------------------------------------------------------
```

Example 3 of modified workaround (3):
bcopy()
[Original]

```
------------------------------------------------------
   bcopy(&a,1,&b,2);
------------------------------------------------------
```

[Modified]

```
------------------------------------------------------
   if (a & (0x01 << 1)) {
      b |= (0x01 << 2);
   } else {
      b &= ~(0x01 << 2);
   }
------------------------------------------------------
```

Example 4 of modified workaround (3):
bnotcopy()

[Original]

```
bnotcopy(&a,1,&b,2);
```

[Modified]

```
if (!(a & (0x01 << 1))) {
   b |= (0x01 << 2);
} else {
   b &= ~(0x01 << 2);
}
```

3. Problem with Inlining a Function Which Has a Static Variable of the Structure Type or the Union Type Within a Function (SHC-0092)

3.1 Product and Versions Concerned

V.7.0B through V.9.04 Release 01

3.2 Description

When a function which has a static variable of structure or union type within the function is inlined, the correct value of a variable may not be referenced.

3.3 Conditions

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

(1) Neither option -optimize=0 nor -optimize=debug_only is used.
(2) In the program exists a function which has a static variable of structure or union type within the function.
(3) The structure or union in (2) has a member of pointer type.
(4) A member of type pointer in (3) has an initial value, and the initial value is an address of a variable.
(5) In the function in (2), the variable whose address is referenced as an initial value of a member of type pointer in (4) is accessed by both of the following:
   (5-1) Direct access
   (5-2) Indirect access through the member of type pointer in (3)
(6) Any of the following are met, and the function in (2) is inlined in a calling function. (NOTE 1.)
   (6-1) Option -inline is used.
   (6-2) Option -speed is used and option -noinline is not used.
   (6-3) #pragma inline is used in the function in (2).
(7) The function definition in (2) is not eliminated. (NOTE 2.)

NOTES:

1. It can be confirmed from an assembly source program output by the compiler whether an inlining is performed.
2. It can be confirmed from an assembly source program output by the compiler whether the function definition is eliminated.

3.4 Example

```
---------------------------------------------------------------
#pragma inline (func)          // Condition (6-3)
int xxx = 0;
struct ST {
   nt * ppp;                 // Condition (3)
};

int func(void)
{
   static struct ST sss = {    // Condition (2)
      &xxx                  // Condition (4)
   };
      *(sss.ppp) = 1;        // Condition (5-2)
      return (xxx);          // Condition (5-1)
}

int main(void) {
   return (func());          // Condition (6)
}
---------------------------------------------------------------

Results of compilation:
---------------------------------------------------------------
_main:
   MOV.L      L12,R5      ; Acquisition of sss address
   MOV        #1,R2       ; H'00000001
   MOV.L      L12+4,R4    ; Acquisition of xxx address
   MOV.L      @R5,R1      ; Acquisition of value of sss.ppp
   MOV.L      @R4,R0      ; Value of xxx is assigned a return
                          ; value(R0).
   MOV.L      R2,@R1      ; On *(sss.ppp), 1 stored.
   RTS
---------------------------------------------------------------
xxx is referenced as a return value before setting assignment value
of *(sss.ppp).
If correctly executed, the value of xxx after assigning 1 to *(sss.ppp)
is assigned to R0.
```

3.5 Workaround

To avoid this problem, do any of the following:
(1) Use option -optimize=0 or -optimize=debug_only.
(2) Set a value to the member of type pointer in Condition (3) by assignment statement.
(3) Unify two kinds of access methods in Condition (5) into either one.
(4) Change the static variable in Condition (2) to a static variable in file-scope.
(5) To disable inlining, do any of the following:
(5-1) When Condition (6-1) is met, do not use option -inline.
(5-2) When Condition (6-2) is met, use option -noinline.
(5-3) When Condition (6-3) is met, eliminate #pragma inline.

3.6 Example

Example of modified workaround (2):

```
-------------------------------------------------------
static struct ST sss;
sss.ppp = &xxx;
-------------------------------------------------------
```

Example 1 of modified workaround (3):
When direct access is used:

```
-------------------------------------------------------
xxx = 1;
return (xxx);
-------------------------------------------------------
```

Example 2 of modified workaround (3):
When indirect access through the member of type pointer is used:

```
-------------------------------------------------------
*(sss.ppp) = 1;
return (*(sss.ppp));
-------------------------------------------------------
```

4. Problem with Using the Standard Library Function strcpy() (SHC-0093)

4.1 Product and Versions Concerned

V.7.0B through V.9.04 Release 01

4.2 Description

When the storage area for copying, specified on the 1st argument of the standard library function strcpy(), and the storage area of a variable used before and after strcpy() are overlapped, the value of the variable may not be updated correctly.

4.3 Conditions

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

(1) Option -optimize=debug_only is not used.
(2) Option -blockcopy=inline is used.
Here, when option -nospeed or option -speed is used,
the default for this option is -blockcopy=inline.
(3) In the program exists a function where the standard library
function strcpy() is called.
(4) In the function in Condition (3) exists a variable, a structure
member or a union member, which are assigned a value before
strcpy() and referenced after strcpy();
(5) The variable, the structure member or the union member in (4) are
types of scalar except signed long long and unsigned long long.
(6) The storage area for copying, specified on the 1st argument of
strcpy() in (3), and the storage area of the variable used before
and after strcpy() in (4) are overlapped.

4.4 Example

```
------------------------------------------------------------------
#include
union
{
      char a[4];
      int  b;       // Condition (5)
} u;
   int c;
void func(void)       // Condition (3)
{
      u.b = 1;           // Condition (4)
      strcpy(u.a, "T");    // Conditions (3) and (6)
      c = u.b;           // Condition (4)
}
------------------------------------------------------------------
```

If correctly executed, u.b is assigned the value of "T".
On c, 1 that is the value of u.b before updating is stored in error.

Results of compilation:

```
------------------------------------------------------------------
_func:
   MOV        #1,R5
   MOV.L      L12+2,R1
   MOV.L      R5,@R1     ; on u.b, 1 stored.
   MOV.L      L12+6,R6
   MOV.W       @R6,R7
   MOV.L      L12+2,R4   ; u
   MOV.L      L12+10,R2  ; c
   MOV.W       R7,@R4     ; u is updated by calling strcpy(u.a, "T")
```

```
    RTS
    MOV.L       R5,@R2     ; on c, 1 which is a value of u.b before
                           ; being updated is stored.
-----------------------------------------------------------------
```

4.5 Workaround

To avoid this problem, do any of the following:

(1) Use option -optimize=debug_only.

(2) Do not use option -blockcopy=inline.
When option -speed or option -nospeed is used, use option -blockcopy=runtime, since the default for this option is -blockcopy=inline.

(3) Modify the variable, the structure member or the union member in Condition (4) with either of the following:
(3-1) Qualify to be volatile.
(3-2) Change to an array of length 1.

Example:

Example of modified workaround (3-2):

```
-----------------------------------------
union {
    char a[4];
    int  b[1];
} u;
-----------------------------------------
```

5. Schedule for Fixing the Problems

All the above problems have already been fixed in the C/C++ compiler package V.9.04 Release 02 for the SuperH RISC engine family.
For details of the latest version, see RENESAS TOOL NEWS Document No. 140201/tn2 on the Web page at:

https://www.renesas.com/search/keyword-search.html#genre=document&q=140201tn2

This page will be opened on February 5, 2014.