RENESAS Tool News

RENESAS TOOL NEWS on February 16, 2005: RSO-SHC-050216D

A Note on Using the C/C++ Compiler Package V.8 for the SuperH RISC Engine Family

Please take note of the following problems in using the C/C++ compiler package V.8 for the SuperH RISC engine family:

- Eleven problems experienced in the compiler

---

1. **Product and Versions Concerned**
   The C/C++ compiler Packages V.8.00.00 through V.8.00.04 for the SuperH RISC engine family are concerned.

2. **Description**
   The following 11 problems arise:

   (1) Loops containing a controlling expression may be executed an incorrect number of times (SHC-0008).

   - Conditions

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

   1. The optimize=1 option is selected.
   2. A program loop exists.
   3. The increment or decrement of the controlled variable in the loop in Condition 2 is 1.
   4. The above loop contains an if statement.
   5. Either condition 5a or 5b below is met.

      5a. The controlling expression in the if statement in Condition 4 compares the controlled variable in the loop in Condition 2 with an invariant in the loop (variable c in Example 1

below).

5b. The initial value or the upper limit of the controlled variable in the loop in Condition 2 is an invariant in the loop (variable c in Example 2 below).

6. The invariant in Condition 5a or 5b is of type int.

Example 1:

```
-----------------------------------------------
int b[100];
unsigned int c=0;
void func1() {
  unsigned int i;
  for(i=0;i<=100;i++) {     // Executed infinite times.
     if(i != c) {
        b[i]=0;
     }
  }
}
-----------------------------------------------
```

Example 2:

```
-----------------------------------------------
int b[100];
unsigned int c=0;
void func2() {
  unsigned int i;
  for(i=0;i<c;i++) {       // Executed once or more
                           // even if c = 0.
     if(i != 5) {
        b[i]=0;
     }
  }
}
-----------------------------------------------
```

- Workaround

This problem can be circumvented in any of the following ways:

1. Use the optimize=0 option.
2. Use the size option.
3. Qualify the controlled variable in the loop in Condition 2 as volatile.
4. Replace the invariant in Condition 5 with its value.

(2) Do-while loops to be executed only once may be repeated twice or more (SHC-0010).

- Conditions

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

1. The optimize=1 option is selected.
2. A do-while loop exists.
3. The controlled variable in the loop in Condition 2 is of type int, signed int, long, or signed long.
4. The controlling expression in the loop in Condition 2 checks whether its controlled variable is less than, less than or equal to, greater than, or greater than or equal to a constant.
5. The comparison operator, the initial value of the controlled variable, its incremented or decremented value in the loop, and the constant to be compared in the controlling expression satisfy any of the following conditions, 5a through 5d:

   5a. If the controlled variable of the loop is less than the constant to be compared, the following relations are all held:
   - The incremented or decremented value is positive.
   - The constant to be compared is less than or equal to the initial value of the controlled variable.
   - 0x00000000 <= (constant - initial value - 1) <= 0x7FFFFFFF.

   5b. If the controlled variable of the loop is less than or equal to the constant to be compared, the following relations are all held:
   - The incremented or decremented value is positive.
   - The constant to be compared is less than the initial value of the controlled variable.
   - 0x00000000 <= (constant - initial value - 1) <= 0x7FFFFFF F.

   5c. If the controlled variable of the loop is greater than the constant to be compared,

the following relations are all held:
- The incremented or decremented value is negative.
- The constant to be compared is greater than or equal to the initial value of the controlled variable.
- 0x00000000 <= (initial value - constant - 1) <= 0x7FFFFFFF.

5d. If the controlled variable of the loop is greater than or equal to the constant to be compared the following relations are all held:
- The incremented or decremented value is negative.
- The constant to be compared is greater than the initial value of the controlled variable.
- 0x00000000 <= (initial value - constant - 1) <= 0x7FFFFFFF.

Example:

```
----------------------------------------
int func() {
int count=0;
int limit=0x60000000;
do {
  count++;
  limit += 0x10000000;
} while(limit < -0x60000000);
       // If executed correctly, the expression is FALSE
       // after the first looping, and the loop is exited.
  return (count);            // "count" takes another value
                          // than the correct "1".
}
----------------------------------------
```

- Workaround

This problem can be circumvented in any of the following ways:

1. Use the optimize=0 option.
2. Change the type of the loop in Condition 2 to a pre-tested one.
3. Qualify the controlled variable in the loop as volatile.

(3) When 1 is compared with a signed bit field of 1 bit wide or with the result of an operation performed on that of any comparison, the incorrect result may be obtained (SHC-0011).

- Conditions

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

1. The conditions stated below are all met.
   1a. The optimize=1 option is selected.
   1b. A signed bit field of 1 bit wide is used.
   1c. Equality or inequality (== or !=) between 1 and the signed bit field in Condition 1b is tested.
2. The conditions stated below are all met.
   2a. The optimize=1 option is selected.
   2b. Any of the following operations is performed on the result of any comparison: (1) subtraction between it and 1, (2) XOR operation of it and 1, (3) its sign inversion, and (4) its bitwise inversion.
   2c. Equality or inequality (== or !=) between 1 and the result of the operation in Condition 2b is tested.

Example1:

```
----------------------------------------
struct {
  char b0:1;
} ST;
void func() {
  if (ST.b0 != 1) {
  . . . . . .
  }
}
----------------------------------------
```

Example 2:

```
----------------------------------------
int a;
void func2() {
```

```
  int t;
  t = ((a & 0x40) == 0);
  t = t - 1;
  t = -t;
  if(~t==1) {
     a = 1;
  } else {
     a = 2;
  }
}
----------------------------------------
```

- Workaround

This problem can be circumvented in either of the following ways:

1. Use the optimize=0 option.
2. Assign the bit field in Condition 1b or the result of the operation performed in Condition 2b to a variable qualified as volatile, and then make the comparison in Condition 2c.

---

(4) When the result of an add or subtract operation between a variable and 0 or a multiply operation of a variable by 1 is used in another operation, a change may incorrectly be made to the value of the variable (SHC-0012).

- Conditions

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

1. An addition/subtraction of 0 to/from a variable or a multiplication of a variable by 1 is performed.
2. The result of the operation in Condition 1 is used in another operation such as addition, subtraction, bitwise AND, bitwise OR, bitwise XOR, division, remainder or shift.

Example:

```
----------------------------------------
int a[4], b;
void func() {
  a[3&(b-0)]=0;
}
```

- Workaround

Don't describe any operation concerned to the operation in Condition 1.

Example: a[3&b]=0;

---

(5) When a double-type member of a structure or union for which "pack" is defined as 1 is referenced, a change may incorrectly be made to the value of register R2 (SHC-0013).

- Conditions

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

1. A structure or union exists for which pack=1 is used.
2. The structure or union in Condition 1 contains a member of type double.
3. CPU options cpu=sh4, sh4a, and sh2afpu are used.
4. Option "size" or "unaligned=runtime" is selected.
5. The runtime-routine "_pack1_ld64" is called at referencing.

Example:

```
---------------------------------------
#pragma pack 1
struct {
  double d;
} ST;
int t;
double d[2];
void func() {
  d[t]=ST.d;
}
---------------------------------------
```

- Workaround

This problem can be circumvented in any of the following ways:

1. Don't define pack as 1.
2. Use the speed option.

3. Use the unaligned=inline option.

---

(6) In an expression containing both a multiplication and division by constants, an incorrect result may be obtained (SHC-0015).

- Conditions

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

1. A multiplication of an unsigned-type expression by a constant exists.
2. The expression in Condition 1 is divided by a positive factor of the constant in Condition 1.
3. The result of the multiplication in Condition 1 exceeds the maximum value allowed to the type of the expression.

Example:

```
----------------------------------------
unsigned int a=65536;
unsigned int b;
void func() {
  b=(a*65536)/8;   // The correct result b=0
((65536*65536)/8 -> 0/8=0)
}               // replaced by b=65535<<13.
----------------------------------------
```

- Workaround

Assign the multiplicative expression in Condition 1 to a variable qualified as volatile.

Example:

```
  void func() {
    volatile unsigned int t=a*65536;
    b=t/8;
  }
```

---

(7) In shift operations, consider each shift count is less than the bit size of the value to be shifted. When such shifts are performed more than once, and the total number of shifts exceeds the bit size of the value to be shifted, an incorrect result of operation may be obtained (SHC-

0016).

- Conditions

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

1. Any CPU option parameters except cpu=[sh1|sh2|sh2e|sh2dsp] are used.
2. Either condition 2a or 2b below is met.
   2a. Left shifts and multiplications by powers of 2 are performed twice or more, where every shift count and exponent in powers of 2 is less than the bit size of the value to be shifted.
   2b. Right shifts and divisions by powers of 2 are performed twice or more, where every shift count and exponent in powers of 2 is less than the bit size of the value to be shifted.
3. The total sum of the shift counts and the exponents in powers of 2 in 2a or 2b exceeds the bit size of the value to be shifted.

Example:

```
----------------------------------------
int x,y;
void func() {
  x=y<<31<<1;  // he total of shift counts, 32, exceeds
               // the bit size of type int.
}
----------------------------------------
```

- Workaround

Assign the result of shifts and/or multiplications in Condition 2 to a variable qualified as volatile.

Example:

```
  void func() {
    volatile int t=y<<31;
    x=t<<1;
  }
```

---

(8) In a function containing an infinite loop, the value of a variable may be loaded from memory even though the

value does not reside in memory (SHC-0020).

- Conditions

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

1. An infinite loop exists in a function.
2. In the function in Condition 1, a value is assigned to a variable not qualified as volatile1.
   The above variable can be a temporary variable generated by the compiler. However, if a register is assigned to such a temporary variable or an auto variable, this condition is not met.

Example:

```
----------------------------------------
int b;
void func() {
  int a = 0;
  if (b) {
     while(1) {
        a = sub() - a;    // "a" is not saved on stack.
        sub();
        . . . . . .
        if (a > 1) {      // "a" is restored as if it were
                          // saved on stack.
           sub();
        }
     }
  }
}
----------------------------------------
```

- Workaround

Don't make the infinite loop be interpreted as so in the function.

Example:

```
        int loop_flag = 1;   // Value of this variable must
not be 1.
        int b;
        void func() {
           int a = 0;
           if (b) {
              while(loop_flag) {
```

```
            a = sub() - a;
            sub();
        . . . . . .
            if (a > 1) {
               sub();
            }
          }
        }
      }
```

---

(9) When an operation is performed on a member of a bit field through a pointer, an incorrect result may be obtained (SHC-0023).

- Conditions

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

1. A structure or union is used.
2. A member of the structure or union in Condition 1 is referenced directly (un.b in Example below).
3. A pointer variable (p in Example) exists which points to a member of the structure or union (un.a in Example) in the same area as the member referenced in Condition 2.
4. The pointer in Condition 3 is defined as a local variable.
5. An indirect reference using the pointer in Condition 3 (*p in Example) exists.
6. Updated is the value of the area where the member referenced in Conditions 2 and 3 exist.
7. The structure or union itself is not referenced in the function where references in Conditions 2 and 3 are made.

Example:

```
----------------------------------------
typedef union {
  unsigned int a;
  unsigned int b:32;
} UN;
UN un;
void func() {
```

```
  int *p=(int *)&un.a;
  un.b=1;
  *p+=1;
}
----------------------------------------
```

- Workaround

This problem can be circumvented in any of the following ways:

1. Reference the structure or union itself inside the function.

   Example:

```
void func() {
   int *p=(int *)&un.a;
   un;  // Inserted
   un.b=1;
   *p+=1;
}
```

2. Define the pointer as a global variable.
3. Use the optimize=0 option.

---

(10) In a function whose last processing is to call another function, the function call may incorrectly be replaced by a JMP instruction (SHC-0029).

- Conditions

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

1. A function has two or more exits.
2. This function has an exit placed after a function call, not at the last of the description of the function.
3. Immediately before the function call in Condition 2 exists another function call.
4. The function calls in Condition 2 and 3 are placed in different blocks from each other.
5. In the function in Condition 1, the contents of no registers except the procedure register are not saved on or restored from the stack.

Example:

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

```
// Option -speed selected.

void func(int a, int s) {
   switch(a) {
   case 1:
      switch(s) {
      case 0:
         ng();     // Incorrectly replaced with JMP.
         break;
      case 1:
         ok();
         break;
      }
      f1();
      break;
   case 2:
      f2();
      break;
   }
}
----------------------------------------
```

- Workaround

This problem can be circumvented in either of the following ways:

1. Add an include function nop() at the last of the description of the function concerned.
2. Place no function call at the last of the description of the function concerned.

---

(11) Copying a structure or union may reserve the stack area more than necessary (SHC-0021).

- Conditions

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

1. An array of structures or unions exists which include members of type structure or union.
2. The array in Condition 1 has two or more elements.
3. An assignment expression to a structure or union exists.
4. The left term of the assignment expression in Condition 3 is a member of a structure or union in

the array of structures or unions.

Example:

```
----------------------------------------
typedef struct {
   unsigned char c;
} ST0;
typedef struct {
  ST0 s;
} ST;
extern ST A[1000];
extern unsigned short i;
void func(ST *d) {
   A[i-1].s=d->s;  // Stack reserved redundantly for
A[1000].
}
----------------------------------------
```

- Workaround

This problem can be circumvented in either of the following ways:

1. Reference the member of a structure or union stated in Condition 4 using a pointer.

   Example:

   ```
   void func(ST *d) {
      ST *pa=&A[i-1];
      pa->s=d->s;
   }
   ```

2. Expand the assignment expression to a structure or union in Condition 3 to the one to each member of the structure or union.

   Example:

   ```
   void func(ST *d) {
      A[i-1].s.c=d->s.c;
   }
   ```

3. **Schedule of Fixing the Problems**

   We plan to fix the above problems in the release of the C/C++ compiler package V.8.00.05 for the SuperH RISC engine family.

[Disclaimer]
The past news contents have been based on information at the time of publication. Now changed or invalid information may be included. The URLs in the Tool News also may be subject to change or become invalid without prior notice.