RENESAS Tool News

RENESAS TOOL NEWS on November 16, 2003: RSO-M3T-CC32R_2-031116D

A Note on Using Cross-Tool Kit M3T-CC32R

Please take note of the following problem in using the M3T-CC32R cross-tool kit for the M32R family MCUs:

- On statements or expressions in parallelism where only the types of operators for the operations of type float are different from each other

---

1. **Versions Concerned**

   M3T-CC32R V.3.00 Release 1--V.4.20 Release 1

2. **Description**
   In statements or expressions in parallelism (only one of them is executed depending on the evaluation of the condition), if only the types of operators for the arithmetic operations of type float or double (*) are different from each other, any of the statements or expressions is deleted in error when the program is compiled with a level-2 optimizing option selected.

   NOTE: * Only when the -float_only option is selected.

   2.1 Conditions
   This problem occurs if the following three conditions are satisfied:

   (1) An optimizing option including the function equivalent to the -O2 option is selected; that is, any of the options -O2, -O3, -O6, and -O7 or -Otime only or -Ospace only is used.

   (2) There exists any type of the following statements and/or expressions in parallelism or a combination of them:

   (a) The two substatements (statements enclosed with curly braces) in an if statement

   (b) The statement(s) between a case label and its break or between a case label and the end of its last program statement in the substatement of a switch statement (Here, case labels include the default label.)

(c) The second and third operands of a conditional operator (? :)

(3) In the statements or expressions in parallelism in (2) above exist the operations where the input and output operands are all of type float (*1) or double (*2), and only the type of each operator is different (+, -, *, or /).

*1. This condition is not imposed if the -float_only option is not selected and either of the following is the case because the double-type operation is performed:

(a) A floating-point constant not followed by a floating suffix f is used for an operand of any operator.

(b) The result of an operation is handled as of type double.

*2. This condition is imposed only when the -float_only option is selected.

2.2 Examples

1. Source file sample1.c

```
----------------------------------------------------------------
float f11, f12;
void func1(int flag)
{
   if (flag) {
      f11 = f12 + 1.0f;    /* Conditions (2a) and (3) */
   } else {
      f11 = f12 - 1.0f;    /* Conditions (2a) and (3) */
   }
}
----------------------------------------------------------------
```

2. Source file sample2.c

```
----------------------------------------------------------------
float f2;
void func2(int flag)
{
   switch(flag) {
      case 0:
         f2 += 2.0f;       /* Conditions (2b) and (3) */
         break;
      case 1:
            f2 = f2 - 2.0f;    /* Conditions (2b) and (3) */
            break;
        case 2:
            f2 = f2 * 2.0f;    /* Conditions (2b) and (3) */
            break;
        case 3:
            f2 += -2.0f;       /* Not met the condition */
            break;
    }
}
-----------------------------------------------------------------
```

3. Source file sample3.c

```
-----------------------------------------------------------------
float f31, f32, f33;
void func3(void)
{
    f31 = (f32 == 0.0f ?
           f32 - f33 :      /* Conditions (2c) and (3) */
           f32 + f33);      /* Conditions (2c) and (3) */
}
-----------------------------------------------------------------
```

4. Source file sample4.c

```
-----------------------------------------------------------------
float f4;
int func1(int flag)
{
    int result = 0;
    switch (flag) {
      case 1:
        result = 1;          /* Conditions(2a),(2b),and (3) */
        f4 += 4.0f;          /* Conditions(2a),(2b),and (3) */
        break;
      case 2:
        result = 1;          /* Conditions(2a),(2b),and (3) */
        f4 -= 4.0f;          /* Conditions(2a),(2b),and (3) */
        break;
      default:
        if (flag >= 3) {
            if (flag == 3) {
                result = 1;  /* Conditions(2a),(2b),and (3) */
                f4 *= 4.0f;  /* Conditions(2a),(2b),and (3) */
            } else {
                result = 1;  /* Conditions(2a),(2b),and (3) */
                f4 /= 4.0f;  /* Conditions(2a),(2b),and (3) */
            }
        }
    }
    return result;
}
------------------------------------------------------------------
```

Examples of specifying options:

```
------------------------------------------------------------------
 % cc32R -c -O2 sample1.c         /* Condition (1) */
 % cc32R -c -O2 sample2.c         /* Condition (1) */
 % cc32R -c -O2 sample3.c         /* Condition (1) */
 % cc32R -c -O2 sample4.c         /* Condition (1) */
------------------------------------------------------------------
```

(Here a % denotes a prompt.)

3. **Workaround**

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

(1) Change the operational expression of type float or of type double with the -float_only option selected to another expression that gives the same result.

Modified source file sample1.c

```
------------------------------------------------------------------
float f11, f12;
void func1(int flag)
{
    if (flag) {
        f11 = f12 + 1.0f;
    } else {
        f11 = f12 + (-1.0f);  /* The same result as f12f - 1.0f */
    }
}
------------------------------------------------------------------
```

(2) Change the construction of the statements or expressions in parallelism. For example, add meaningless lines to these statements or expression using asm functions.

NOTICE: Using an asm function in The M3T-CC32R V.3.00 Release 1 through V.3.20 Release 1 invalidates the optimization by the -O4 level option.

Modified source file sample2.c

```
----------------------------------------------------------------
#pragma keyword asm on          /* Added to validate asm function */
#define DUMMY asm(" ",__LINE__)  /* Dummy asm function defined */

float f2;

void func2(int flag)
{
   int dummy;
   switch(flag) {
      case 0:
         f2 += 2.0f;
         DUMMY;        /* DUMMY; The descriptions of dummies
         break;                  are different from each other */
      case 1:
         f2 = f2 - 2.0f;
         DUMMY;          /* DUMMY */
         break;
      case 2:
         f2 = f2 * 2.0f;
         DUMMY;          /* DUMMY */
         break;
      case 3:
         f2 += -2.0f;
         DUMMY;          /* DUMMY */
         break;
   }
}
----------------------------------------------------------------
```

Modified source file sample3.c

```
----------------------------------------------------------------
#pragma keyword asm on          /* Added to validate asm function */
#define DUMMY asm(" ",__LINE__)  /* Dummy asm function defined */

float f31, f32, f33;
void func3(void)

{
   f31 = (f32 == 0.0f ?
        (DUMMY, f32 - f33) :     /* Add DUMMY */
```

```
          (DUMMY, f32 + f33));      /* Enclose with parentheses
                                       to give ":" precedence
                                       over "," */
}
------------------------------------------------------------------
```

(3) Use the indirection operator pointing to a volatile area.

Modified source file sample4.c

```
------------------------------------------------------------------
float f4;
int func1(int flag)
{
   int result = 0;
   volatile float *ptr;    /* Define a pointer to a volatile area */
   ptr = &f4;              /* Make *ptr equivalent to f4 */
   switch (flag) {
     case 1:
       result = 1;
       *ptr += 4.0f;          /* Change f4 to *ptr */
       break;
     case 2:
       result = 1;
       *ptr -= 4.0f;          /* Change f4 to *ptr */
       break;
     default:
       if (flag >= 3) {
           if (flag == 3) {
               result = 1;
               *ptr *= 4.0f;   /* Change f4 to *ptr */
           } else {
               result = 1;
               *ptr /= 4.0f;   /* Change f4 to *ptr */
           }
       }
   }
   return result;
}
------------------------------------------------------------------
```

(4) Suppress the optimization covering -O2.
The optimization covering -O2 is performed when -O3, -O6, -O7, -Otime only, or -Ospace only is selected.

If you want to use -Otime or -Ospace, use any of the options -O0, -O1, -O4, and -O5 at the same time.

4. Schedule of Fixing the Problem

We plan to fix this problem in our next release of the product.
