RENESAS Tool News

RENESAS TOOL NEWS [RSO-M3T-CC32R-030601D]

Notes on Using Cross-Tool Kit M3T-CC32R

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

- On referencing and updating variables in an infinite loop
- On converting a value by a printf-like standard library function

---

1. Problem on Referencing and Updating Variables in an Infinite Loop

1.1 Version Concerned
M3T-CC32R V.4.00 Release 1

1.2 Description
If a variable is referenced and then updated in an infinite loop, the previously referenced value of the variable may not be updated.

1.3 Conditions
This problem occurs if the following six conditions are satisfied:

(1) Any optimizing option covering the -O5 or -O6 option's function (-O5, -O6, -O7, -Otime only or -Ospace only) is selected at compilation.

Note: The -O7, -Otime only and -Ospace only options include the functions of the -O5 and -O6 options.

(2) Either of the following loop constructs exists in the program:

(a) a while, do-while, or for statement whose continuation condition is always TRUE

(b) a loop made by an unconditionally executed goto statement

(3) In the loop in (2) exists no break, goto or return statement used to exit from the loop.

(4) A variable is referenced in the loop in (2).

(5) The variable in (4) is updated after referenced in the loop in (2).

(6) This variable is not referenced after the update in (5) until the loop is ended.

1.4 Examples

Source file 1:

```
------------------------------------------------------------------
  void func1(int);

  void
  foo1(void)
  {
     int a = 0;
     while (1) {           /* Conditions(2)(a) and (3) */
        func1(a);          /* Condition (4) */
        ++a;               /* Condition (5) */
                        /* Condition (6) */
     }
  }
------------------------------------------------------------------
```

Source file 2:

```
------------------------------------------------------------------
  int func2(int);

  void
  foo2(void)
  {
     int a = 0;
     for (;;) {            /* Conditions (2)(a) and (3) */
        a = func2(a) + 1; /* Conditions (4) and (5) */
                        /* Condition (6) */
     }
  }
------------------------------------------------------------------
```

Source file 3:

```
------------------------------------------------------------------
  volatile int x;

  void
  foo3(void)
  {
     int a = 0;
  infinite_loop:           /* Conditions (2)(b) and (3) */
      x = a;               /* Condition (4) */
      ++a;                 /* Condition (5) */
      goto infinite_loop;   /* Conditions (2)(b) and (3) */
                          /* Condition (6) */
  }
-----------------------------------------------------------------
```

1.5 Solution

This problem has been fixed in the M3T-CC32R V.4.10 Release 1. So please download the fixed product from HERE.

1.6 Workaround in Using the M3T-CC32R V.4.00 Release 1

If you continue to use the M3T-CC32R V.4.00 Release 1, this problem can be circumvented in either of the following ways:

(1) Suppress the optimization equivalent to the -O5 or -O6 option.

(2) As the continuation condition, use an allocated address.
That is, after defining a variable, allocate a fixed address to it using the #pragma ADDRESS directive; then reference the address allocated to the variable (the allocated address) to use it as the continuation condition.

Modification of Source file 1:

```
-----------------------------------------------------------------
 void func1(int);

 int dummy1;
 #pragma ADDRESS dummy1 1    /* Allocate address 1 to dummy1 */

 void
 foo1(void)
 {
    int a = 0;
    while (&dummy1) { /*Use &dummy1 as continuation condition*/
       func1(a);
       ++a;
    }
 }
-----------------------------------------------------------------
```

Modification of Source file 2:

```
-----------------------------------------------------------------
 int func2(int);
 int dummy2;
 #pragma ADDRESS dummy2 1     /* Allocate address 1 to dummy2 */

 void
 foo2(void)
 {
    int a = 0;
    for (;&dummy2;) { /*Use &dummy2 as continuation condition*/
       a = func2(a) + 1;
    }
 }
-----------------------------------------------------------------
```

Modification of Source file 3:

```
-----------------------------------------------------------------
 volatile int x;

 int dummy3;
 #pragma ADDRESS dummy3 1    /* Allocate address 1 to dummy3 */

 void
 foo3(void)
 {
    int a = 0;
 infinite_loop:
    x = a;
    ++a;
    if (&dummy3)    /* Use &dummy3 as continuation condition */
       goto infinite_loop;
 }
-----------------------------------------------------------------
```

2. Problem on converting a value by a printf-like standard library function

2.1 Versions Concerned

M3T-CC32R V.1.00 Release 1 -- V.4.10 Release 1

2.2 Description

When a value less than the value which can be calculated from the specified precision is converted by the %f specification of a format output function such as printf, usually the number of zeros satisfying the precision given by the %f specification is outputted in decimal places. However, the number of zeros may be less by one than the satisfactory number of them depending on the value to be converted.

2.3 Conditions

This problem occurs if the following two conditions are satisfied:

(1) The %f specification of a printf-like function (printf, vprintf, fprintf, vfprintf, sprintf, or vsprintf) is used to convert a value.

(2) When the precision given by the %f specification in (1) is n, the value to be converted by a printf-like function is equal to or less than ten to the power of negative n.
Note that if no precision is explicitly specified in %f, six is used as the precision.

2.4 Examples

Source file:

```
-----------------------------------------------------------------
  #include <stdio.h>

  char bf1[32], bf2[32], bf3[32], bf4[32], bf5[32], bf6[32];

  int main()
  {
     double x;

     x = 0.01;                 /* Condition (2) */
     sprintf(bf1, "%5.1f¥n", x);   /* Conditions (1) and (2)
                               with precision 1 */
     /* A program containing bf1 */

     x = 0.005;                /* Condition (2) */
     sprintf(bf2, "%5.1f¥n", x);   /* Conditions (1) and (2)
                               with precision 1 */
     /* A program containing bf2 */

     x = 0.0064;               /* Condition (2) */
     sprintf(bf3, "%5.1f¥n", x);   /* Conditions (1) and (2)
                               with precision 1 */
     /* A program containing bf3 */

     x = 0.0007;               /* Condition (2) */
     sprintf(bf4, "%5.2f¥n", x);   /* Conditions (1) and (2)
```

```
                                 with precision 2 */
    /* A program containing bf4 */

    x = 8.1e-5;                 /* Condition (2) */
    sprintf(bf5, "%6.2f¥n", x);   /* Conditions (1) and (2)
                                 with precision 2 */
    /* A program containing bf5 */

    x = 7.3e-8;                 /* Condition (2) */
    sprintf(bf6, "%10f¥n", x);    /* Conditions (1) and (2)
                                 with precision 6 */
    /* A program containing bf6 */

    return 0;
  }
---------------------------------------------------------------
```

The output results of the above conversions (the contents of arrays bf1 through bf6)

```
      Output Result     Correct Output
bf1: "   0."          "  0.0"
bf2: "   0."          "  0.0"
bf3: "   0."          "  0.0"
bf4: "  0.0"           " 0.00"
bf5: "   0.0"          "  0.00"
bf6: "   0.00000"      "  0.000000"
```

2.5 Workaround

Discard the decimal places exceeding the specified precision using the floor standard library function in the math.h header file.

Example:

```
   Modify  printf("%5.1¥n, x)

       to  printf("%5.1¥n, floor(x * 1E1 + 0.5)/1E1)
```

Note:

The portion of 1E1 varies according to precisions.
Change %5.2f (precision 2) to floor(x * 1E2 + 0.5)/1E2
Change %8f (precision 6) to floor(x * 1E6 + 0.5)/1E6

Modification of Source file 1:

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

```
#include <stdio.h>
#include <math.h> /* Declaration for using the floor function */

char bf1[32], bf2[32], bf3[32], bf4[32], bf5[32], bf6[32];

int main()
{
   double x;

   x = 0.01;
     /* The decimal places exceeding the specified precision
          discarded in variable x (the same in the later) */
   sprintf(bf1, "%5.1f¥n", floor(x * 1E1 + 0.5)/1E1);
   /* A program containing bf1 */

   x = 0.005;
   sprintf(bf2, "%5.1f¥n", floor(x * 1E1 + 0.5)/1E1);
   /* A program containing bf2 */

   x = 0.0064;
   sprintf(bf3, "%5.1f¥n", floor(x * 1E1 + 0.5)/1E1);
   /* A program containing bf3 */

   x = 0.0007;
   sprintf(bf4, "%5.2f¥n", floor(x * 1E2 + 0.5)/1E2);
   /* A program containing bf4 */

   x = 8.1e-5;
   sprintf(bf5, "%6.2f¥n", floor(x * 1E2 + 0.5)/1E2);
   /* A program containing bf5 */

   x = 7.3e-8;
   sprintf(bf6, "%10f¥n", floor(x * 1E6 + 0.5)/1E6);
   /* A program containing bf6 */

   return 0;
}
```

The output results of the above conversions

```
      Output Result
 bf1: "  0.0"
 bf2: "  0.0"
```

```
bf3: "  0.0"
bf4: " 0.00"
bf5: "  0.00"
bf6: "  0.000000"
```

2.6 Schedule of Fixing the Problem

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