RENESAS ツールニュース

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

M3T-CC32R ご使用上のお願い

M32Rファミリ用クロスツールキット M3T-CC32R の使用上の注意事項を連絡します。

- 無限ループ中の変数の参照と更新に関する注意事項
- 標準ライブラリprintf系関数の%f変換で表示させる場合の注意事項

---

1. 無限ループ中の変数の参照と更新に関する注意事項

1.1 該当製品

M3T-CC32R V.4.00 Release 1

1.2 内容

無限ループの中で、変数の参照の後にその変数の更新を行う場合、変数の更新結果が参照に反映されない場合があります。

1.3 発生条件

以下6点の条件をすべて満たす場合に発生します。

(1) コンパイル時に -O5または-O6を含む最適化オプションを指定している (-O5, -O6, -O7, -Otimeのみ、または-Ospaceのみを指定している)。
※ -O5または-O6を含む最適化は、-O7, -Otimeのみ、または-Ospaceのみを指定しているときにも行われます。

(2) 以下の(a),(b)のいずれかに該当するループ構文がある。
(a) 継続条件が常に真である、while文、do～while文 あるいは for文
(b) 無条件に実行されるgoto文で作られたループ

(3) (2)のループ中に、ループ外に脱出するbreak文、goto文 および return文がいずれもない。

(4) (2)のループ中に変数の参照がある。

(5) (2)のループ中で、(4)の後に変数の更新を行っている。

(6) (5)の更新からループの末尾にかけてその変数の参照がない。

1.4 発生例

[ソースファイル例1]

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

  void
  foo1(void)
  {
     int a = 0;
     while (1) {          /* 発生条件(2)(a),(3) */
        func1(a);         /* 発生条件(4) */
        ++a;             /* 発生条件(5) */
                       /* 発生条件(6) */
     }
  }
---------------------------------------------------------------
```

[ソースファイル例2]

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

  void
  foo2(void)
  {
     int a = 0;
     for (;;) {           /* 発生条件(2)(a),(3) */
        a = func2(a) + 1; /* 発生条件(4),(5) */
                       /* 発生条件(6) */
     }
  }
---------------------------------------------------------------
```

[ソースファイル例3]

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

  void
  foo3(void)
  {
     int a = 0;
  infinite_loop:          /* 発生条件(2)(b),(3) */
     x = a;               /* 発生条件(4) */
```

```
    ++a;                /* 発生条件(5) */
    goto infinite_loop;  /* 発生条件(2)(b),(3) */
                        /* 発生条件(6) */
  }
```

1.5 恒久対策

本内容は、M3T-CC32R V.4.10 Release 1 で改修しました。
日本語版はこちらから ダウンロードしてください。
英語版はこちらから ダウンロードしてください。

1.6 M3T-CC32R V.4.00 Release 1での回避策

V.4.00 Release 1 を使用する必要がある場合、以下のいずれかの手段を適用ください。

(1) -O5または-O6を含む最適化を抑止する。

(2) 変数を配置されたアドレスを使って継続条件を設定する。
#pragma ADDRESS で固定アドレスに配置される変数を作成し、この配置アドレスを継続条件としてください。

[例1の変更例]

```
  void func1(int);

  int dummy1;
  #pragma ADDRESS dummy1 1  /* &dummy1 が 1 になるような変数を作成 */

  void
  foo1(void)
  {
    int a = 0;
    while (&dummy1) {     /* &dummy1 を継続条件にする */
      func1(a);
      ++a;
    }
  }
```

[例2の変更例]

```
  int func2(int);

  int dummy2;
  #pragma ADDRESS dummy2 1  /* &dummy2 が 1 になるような変数を作成 */
```

```
  void
  foo2(void)
  {
     int a = 0;
     for (;&dummy2;) {     /* &dummy2 を継続条件にする */
        a = func2(a) + 1;
     }
  }
```

[例3の変更例]

```
  volatile int x;

  int dummy3;
  #pragma ADDRESS dummy3 1  /* &dummy3 が 1 になるような変数を作成 */

  void
  foo3(void)
  {
     int a = 0;
  infinite_loop:
     x = a;
     ++a;
     if (&dummy3)          /* &dummy3 を継続条件にする */
        goto infinite_loop;
  }
```

ページの先頭へ

---

2. 標準ライブラリprintf系関数の%f変換で表示させる場合の注意事項

2.1 該当製品

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

2.2 内容

標準ライブラリprintf系関数の%f変換で、指定された精度に満たない数値を処理すると、通常、小数点以下には %f の精度の個数分の「0」が出力されますが、実際にはひとつ少なく出力される場合があります。

ただし、実際にこのような表示になるかどうかは、変換する数値に依存します。

2.3 発生条件

以下2点の条件をどちらも満たす場合に発生します。

(1) printf系関数(printf, vprintf, fprintf, vfprintf, sprintf, vsprintf) の%f変換を使用している。

(2) (1)の%f変換に指定された精度をnとする場合、printf系関数で表示させる数値が10のマイナスn乗以下の値である。
※ %f変換は、明示的に精度指定がない場合、精度として6を使用します。

2.4 発生例

[ソースファイル例]

```
#include <stdio.h>

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

int main()
{
   double x;

   x = 0.01;                  /* 条件(2) */
   sprintf(bf1, "%5.1f¥n", x);   /* 条件(1),(2) 精度1 */
   /* bf1を使った処理 */

   x = 0.005;                  /* 条件(2) */
   sprintf(bf2, "%5.1f¥n", x);   /* 条件(1),(2) 精度1 */
   /* bf2を使った処理 */

   x = 0.0064;                  /* 条件(2) */
   sprintf(bf3, "%5.1f¥n", x);   /* 条件(1),(2) 精度1 */
   /* bf3を使った処理 */

   x = 0.0007;                  /* 条件(2) */
   sprintf(bf4, "%5.2f¥n", x);   /* 条件(1),(2) 精度2 */
   /* bf4を使った処理 */

   x = 8.1e-5;                  /* 条件(2) */
   sprintf(bf5, "%6.2f¥n", x);   /* 条件(1),(2) 精度2 */
   /* bf5を使った処理 */

   x = 7.3e-8;                  /* 条件(2) */
```

```
    sprintf(bf6, "%10f¥n", x);     /* 条件(1),(2) 精度6 */
    /* bf6を使った処理 */

    return 0;
  }
```

[上記の出力結果 --- ソースファイル例 の bf1～bf6の内容]

	出力内容	正しい出力
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 回避策

精度外の桁を標準関数 math.h のfloorで切り捨ててください。

(例) printf("%5.1¥n, x)

↓

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

※ 1E1の部分は、精度に応じて変更してください。

(例) %5.2f(精度2) → floor(x * 1E2 + 0.5)/1E2

%8f (精度6) → floor(x * 1E6 + 0.5)/1E6

[ソースファイル例の回避例]

```
  #include <stdio.h>
  #include <math.h>       /* floor関数を用いるための指定 */

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

  int main()
  {
    double x;

    x = 0.01;
        /* x の精度外の部分を切り捨てる(以下同様) */
    sprintf(bf1, "%5.1f¥n", floor(x * 1E1 + 0.5)/1E1);
    /* bf1を使った処理 */
```

```
    x = 0.005;
    sprintf(bf2, "%5.1f¥n", floor(x * 1E1 + 0.5)/1E1);
    /* bf2を使った処理 */

    x = 0.0064;
    sprintf(bf3, "%5.1f¥n", floor(x * 1E1 + 0.5)/1E1);
    /* bf3を使った処理 */

    x = 0.0007;
    sprintf(bf4, "%5.2f¥n", floor(x * 1E2 + 0.5)/1E2);
    /* bf4を使った処理 */

    x = 8.1e-5;
    sprintf(bf5, "%6.2f¥n", floor(x * 1E2 + 0.5)/1E2);
    /* bf5を使った処理 */

    x = 7.3e-8;
    sprintf(bf6, "%10f¥n", floor(x * 1E6 + 0.5)/1E6);
    /* bf6を使った処理 */

    return 0;
  }
```

[上記の出力結果]

出力内容

```
bf1: "  0.0"
bf2: "  0.0"
bf3: "  0.0"
bf4: " 0.00"
bf5: "  0.00"
bf6: "  0.000000"
```

2.6 恒久対策

本内容は、次期バージョンで改修する予定です。