RENESAS ツールニュース

RENESAS TOOL NEWS [2003年11月16日：RSO-M3T-CC32R_2-031116D]

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。

- 並列の関係にある文または式でfloat型の演算のみが異なる場合に関する注意事項

---

1. **該当製品**

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

2. **内容**

   並列の関係(どれか一つだけが実行されるような関係)にある文または式で、float型またはdouble型(*)の四則演算のみが異なる場合、レベル2の最適化が有効な状態でコンパイルすると、いずれかの文または式を誤って削除してしまいます。

   *：-float_onlyオプションが有効な場合のみ

   2.1 発生条件

   以下の条件をすべて満たす場合に発生します。

   (1) -O2と同じ機能を含む最適化オプションを指定している。
   (-O2, -O3, -O6, -O7のいずれかを指定している。もしくは-Otimeのみまたは-Ospaceのみを指定している。)

   (2) 以下のいずれか、または組み合わせからなる並列の関係にある文または式がある。

   (a) 同じif文に属する2つの副文

   (b) switch文の副文の中の、caseラベルと次のbreakに挟まれている文、またはcaseラベルと最後の実行文の末尾に挟まれている文

   (c) 条件演算子(?:)の第2パラメータと第3パラメータ

   (3) (2)の並列の関係にある文または式には、入出力がいずれもfloat型(*1)またはdouble型(*2)の演算があり、その演算の種類 (+ - * /)がそれぞれ異なる以外は内容が同じであ

る。

*1： -float_onlyオプションが無効で、以下のいずれかの場合はdouble型の演算になるため発生条件に該当しません。

(a) 演算のパラメータに、「f」指定のない浮動小数点定数を使用している。

(b) 演算結果(演算の出力)をdouble型として扱っている。

*2： -float_onlyオプションが有効な場合のみ。

2.2 発生例

[ソースファイル例1]
[sample1.c]

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

[ソースファイル例2]
[sample2.c]

```
------------------------------------------------------------------
  float f2;
  void func2(int flag)
  {
     switch(flag) {
        case 0:
           f2 += 2.0f;       /* 条件(2b),(3) */
           break;
        case 1:
           f2 = f2 - 2.0f;    /* 条件(2b),(3) */
           break;
        case 2:
           f2 = f2 * 2.0f;    /* 条件(2b),(3) */
           break;
```

```
        case 3:
          f2 += -2.0f;      /* この式は該当しない */
          break;
    }
  }
```

[ソースファイル例3]
[sample3.c]

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

[ソースファイル例4]
[sample4.c]

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

```
          }
        }
      }
      return result;
  }
```

[オプション指定例]

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

(% は プロンプトを表します)

3. **回避策**

以下のいずれかの方法で回避してください。

(1) float型または-float_onlyオプションが有効な場合のdouble型の演算式を演算結果が等価になる別の式に変更する。

[ソースファイル例1の変更例]

[sample1.c]

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

---

(2) 並列の関係にある文または式の構成を変更する。
例えば、asm関数などを利用して、並列の関係にある文または式に意味のない処理を追加してください。

※ ただし、V.3.00 Release 1 ～ V.3.20 Release 1でasm関数を使用すると、-O4レベルの最適化を有効にできませんのでご注意ください。

[ソースファイル例2の変更例]

[sample2.c]

```
#pragma keyword asm on            /* asm関数を有効にするため追加 */
#define DUMMY asm(" ",__LINE__)    /* 中身のないasm関数を定義 */

float f2;
void func2(int flag)
{
   int dummy;
   switch(flag) {
      case 0:
         f2 += 2.0f;
         DUMMY;          /* DUMMY; の内容はそれぞれ異なる */
         break;
      case 1:
         f2 = f2 - 2.0f;
         DUMMY;          /* DUMMY; の内容はそれぞれ異なる */
         break;
      case 2:
         f2 = f2 * 2.0f;
         DUMMY;          /* DUMMY; の内容はそれぞれ異なる */
         break;
      case 3:
         f2 += -2.0f;
         DUMMY;          /* DUMMY; の内容はそれぞれ異なる */
         break;
   }
}
```

[ソースファイル例3の変更例]

[sample3.c]

```
#pragma keyword asm on            /* asm関数を有効にするため追加 */
#define DUMMY asm(" ",__LINE__)    /* 中身のないasm関数を定義 */

float f31, f32, f33;
void func3(void)
{
   f31 = (f32 == 0.0f ?
        (DUMMY, f32 - f33) :     /* DUMMYを追加 */
        (DUMMY, f32 + f33));     /* 優先順序があるので括弧でくくることを
                                    忘れないよう注意 */
```

```
  }
```
---

(3) volatileな領域へのポインタの間接演算代入を用いる。

[ソースファイル例4の変更例]

```
  float f4;
  int func1(int flag)
  {
     int result = 0;
     volatile float *ptr;       /* volatile領域へのポインタを作成 */
     ptr = &f4;                /* *ptr が f4 に等価になるよう設定 */
     switch (flag) {
      case 1:
       result = 1;
       *ptr += 4.0f;          /* f4 ==> *ptr に変更 */
       break;
      case 2:
       result = 1;
       *ptr -= 4.0f;          /* f4 ==> *ptr に変更 */
       break;
      default:
       if (flag >= 3) {
          if (flag == 3) {
             result = 1;
             *ptr *= 4.0f;   /* f4 ==> *ptr に変更 */
          } else {
             result = 1;
             *ptr /= 4.0f;   /* f4 ==> *ptr に変更 */
          }
       }
     }
     return result;
  }
```
---

(4) -O2を含む最適化を抑止する。
-O2を含む最適化は、-O3,-O6,-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。-Otimeまたは-Ospaceを使用する場合は、同時に-O0,-O1,-O4,-O5のいずれかを指定してください。

4. **恒久対策**

本問題は、次期バージョンで改修する予定です。
