RENESAS ツールニュース

RENESAS TOOL NEWS 2005年02月16日：RSO-SHC-050216D

SuperH RISC engine ファミリ C/C++コンパイラパッケージ V.8 ご使用上のお願い

SuperH RISC engine ファミリC/C++コンパイラパッケージV.8の使用上の注意事項を連絡します。

- コンパイラに関する注意事項

---

1. **該当製品**
   以下のバージョンのコンパイラが該当します。
   SuperH RISC engine C/C++コンパイラ V.8.00.00 ～ V.8.00.04

2. **内容**
   以下11点の問題があります。

   (1) 条件式を含むループにおいて、ループが不正に実行される場合がある(SHC-0008)。

   発生条件： 以下の条件をすべて満たした場合に発生することがあります。
   1. optimize=1オプションを使用している。
   2. ループが存在する。
   3. 2のループ制御変数の増分値が1または-1である。
   4. 2のループ内にif文が存在する。
   5. 以下5a、または5bのいずれかの条件を満たす。
      5a. 4のif文の条件式が2.のループ制御変数と、2.のループ内で値が不変の式(例1の変数c)の比較である。
      5b. 2のループ制御変数の初期値または上限値がループ内で値が不変の式(例2の変数c)である。
   6. 5のループ内不変式が整数型である。

例1

```
-----------------------------------------
int b[100];
unsigned int c=0;
void func1() {
  unsigned int i;
  for(i=0;i<=100;i++) {    // 無限ループになる
    if(i != c) {
      b[i]=0;
    }
  }
}
-----------------------------------------
```

例2

```
-----------------------------------------
int b[100];
unsigned int c=0;
void func2() {
  unsigned int i;
  for(i=0;i<c;i++) {       // c=0の時もループを1回以上まわる
    if(i != 5) {
      b[i]=0;
    }
  }
}
-----------------------------------------
```

回避策： 以下のいずれかの方法で回避してください。

1. optimize=0オプションを指定する。
2. sizeオプションを指定する。
3. 発生条件2.のループ制御変数をvolatile修飾する。
4. 発生条件5.のループ内不変式を、その値で置き換える。

---

(2) do-whileループで、正しくは1回で終了するものが2回以上ループする場合がある(SHC-0010)。

発生条件： 以下の条件をすべて満たした場合に発生することがあります。

1. optimize=1オプションを使用している。
2. do-whileループが存在する。
3. 2のループ制御変数の型がint, signed int, long, または signed longである。

4. 2のループ判定式が、定数値との大小比較(<,<=,>,>=)である。
5. 2のループの比較演算子、制御変数の初期値、更新値、および判定式の比較値が以下5a～5dのいずれかの条件を満たす。
    5a. 判定式が、ループ制御変数<定数 の場合に以下全てに該当する。
        ・更新値が正
        ・比較値<=初期値
        ・0x00000000<=(比較値-初期値-1)<=0x7FFFFFFF
    5b. 判定式が、ループ制御変数<=定数 の場合に以下全てに該当する。
        ・更新値が正
        ・比較値<初期値
        ・0x00000000<=(比較値-初期値-1)<=0x7FFFFFFF
    5c. 判定式が、ループ制御変数>定数 の場合に以下全てに該当する。
        ・更新値が負
        ・初期値<=比較値
        ・0x00000000<=(初期値-比較値-1)<=0x7FFFFFFF
    5d. 判定式が、ループ制御変数>=定数 の場合に以下全てに該当する。
        ・更新値が負
        ・初期値<比較値
        ・0x00000000<=(初期値-比較値-1)<=0x7FFFFFFF

例：

```
----------------------------------------
int func() {
int count=0;
int limit=0x60000000;
do {
  count++;
  limit += 0x10000000;
} while(limit < -0x60000000);  // 正しい動作では1回目のループ後の
                               // 判定で偽となり、ループを抜ける。
  return (count);              // count=1が正しいが、違う値に
```

なる。
}

回避策： 以下のいずれかの方法で回避してください。

1. optimize=0オプションを指定する。
2. 当該ループを前判定型に変更する。
3. 当該ループの制御変数をvolatile指定する。

---

(3) 1ビットの符号付きビットフィールドと1との比較を行ったとき、またはある比較結果に対して演算を行いその結果を1と比較したときに、比較結果が間違っている場合がある(SHC-0011)。

発生条件： 以下1、2のいずれかの条件を満たした場合に発生することがあります。

1. 以下のすべての条件を満たした場合。
   - 1a. optimize=1を指定している。
   - 1b. 1ビットの符号付きビットフィールドを使用している。
   - 1c. 1b.と1の比較(==,!=)を行っている。
2. 以下のすべての条件を満たした場合。
   - 2a. optimize=1を指定している。
   - 2b. 比較結果に対して以下のいずれかの演算を行っている。
     比較結果と1の減算、比較結果と1の排他的論理和、
     比較結果の符号反転、または比較結果のビット反転
   - 2c. 2b.の演算結果と1の比較(==,!=)を行っている。

例1：

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

```
----------------------------------------
例2：
----------------------------------------
int a;
void func2() {
  int t;
  t = ((a & 0x40) == 0);
  t = t - 1;
  t = -t;
  if(~t==1) {
     a = 1;
  } else {
     a = 2;
  }
}
----------------------------------------
```

回避策： 以下のいずれかの方法で回避してください。

1. optimize=0オプションを指定する。
2. 当該ビットフィールドまたは比較結果との演算の結果をvolatile指定した変数に代入してから比較する。

---

(4) 0との加算および減算、または1との乗算をした結果を他の演算で使用したときに、変数の値を誤って更新する場合がある(SHC-0012)。

発生条件： 以下の条件をすべて満たした場合に発生することがあります。

1. 変数と0の加算および減算、または変数と1の乗算をしている。
2. 1の結果を以下の演算で使用している。
   加算、減算、論理演算(&,|,^)、除算、剰余算、またはシフト

例：

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

回避策： 0および1との演算をしないで、演算結果を使用す

る。
　　例： a[3&b]=0;

---

(5)　pack=1を指定した構造体または共用体のdouble型メンバを参照したときに、 レジスタR2の値を誤って変更する場合がある(SHC-0013)。

発生条件： 以下の条件をすべて満たした場合に発生することがあります。

1. pack=1を指定した構造体または共用体がある。
2. 1.の構造体または共用体にdouble型のメンバが存在する。
3. cpu=sh4,sh4a,sh2afpuを指定している。
4. sizeまたはunaligned=runtimeオプションを使用している。
5. 実行時ルーチン_pack1_ld64の呼び出しがある。

例：

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

回避策： 以下のいずれかの方法で回避してください。

1. pack=1を指定しない。
2. speedオプションを指定する。
3. unaligned=inlineオプションを指定する。

---

(6)　定数乗算および定数除算を共に含む式において、演算結果が間違っている場合がある(SHC-0015)。

発生条件： 以下の条件をすべて満たした場合に発生することがあります。

1. unsigned型の式と定数の乗算式が存在する。
2. 1を含む式を1の定数の正の約数で除算している。

3. 1の乗算結果がその式の型の最大値を超える。

例：

```
----------------------------------------
unsigned int a=65536;
unsigned int b;
void func() {
  b=(a*65536)/8;  // b=0 ((65536*65536)/8→0/8=0)
が正しい
}               // 結果だが、b=65535<<13と置き換えられ
る。
----------------------------------------
```

回避策：発生条件1.の式をvolatile指定した変数に代入する。

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

---

(7) 被シフト数のビットサイズ未満の定数シフトを複数回行い、そのシフト数の合計が被シフト数のビットサイズ以上になった場合、 演算結果が間違っている場合がある(SHC-0016)。

発生条件： 以下の条件をすべて満たした場合に発生することがあります。

1. cpuオプションのパラメータにsh1, sh2, sh2e, またはsh2dsp以外を指定している。
2. 以下2a、2bのいずれかの条件を満たす。
   - 2a. 左シフトもしくは2のべき乗の乗算が2回以上連続し、かつ個々の シフト数、乗数は被シフト数のビットサイズ未満である。
   - 2b. 右シフトもしくは2のべき乗の除算が2回以上連続し、かつ個々の シフト数、乗数は被シフト数のビットサイズ未満である。
3. 2のシフト数と乗数の値の合計が被シフト数のビットサイズ以上となる。

例：

```
----------------------------------------
int x,y;
void func() {
  x=y<<31<<1;  // シフト数の合計は32で、int型のビット
```

サイズ
            // 以上になる
}

回避策：発生条件2.の式の演算結果をvolatile指定した変数に代入する。
      例： void func() {
            volatile int t=y<<31;
            x=t<<1;
          }

---

(8)　無限ループを含む関数において、変数の値をメモリ上にストアしていないにもかかわらず、 メモリから値をロードする場合がある(SHC-0020)。

発生条件： 以下の条件をすべて満たした場合に発生することがあります。
1. 関数内に無限ループするループがある。
2. 1.の関数内でvolatile指定のない変数に値を設定している(コンパイラ生成の一時変数を含む)。
   ただしauto変数またはコンパイラ生成一時変数の場合は、その変数にレジスタが割り付く場合は該当しない。

例：
```
---------------------------------------
int b;
void func() {
  int a = 0;
  if (b) {
     while(1) {
        a = sub() - a;    // aがスタックにストアされない
        sub();
        . . . . . .
        if (a > 1) {      // aがスタックにストアされているとして
                          // ロード
           sub();
        }
     }
  }
}
```

回避策： 関数内で無限ループと判断できないようにする。

例：

```
int loop_flag = 1;   // この変数の値は0にしない
int b;
void func() {
   int a = 0;
   if (b) {
      while(loop_flag) {
         a = sub() - a;
         sub();
. . . . . .
         if (a > 1) {
            sub();
         }
      }
   }
}
```

---

(9) ポインタを介してビットフィールドメンバに対して演算を行った場合、 その演算結果が間違っている場合がある(SHC-0023)。

発生条件： 以下の条件をすべて満たした場合に発生することがあります。

1. 構造体または共用体型を使用している。
2. 1.のメンバを直接参照(例ではun.b)している。
3. 2.と同じ領域のメンバ(例ではun.a)を指すポインタ変数(例ではp)が存在する。
4. 3.のポインタはローカル変数である。
5. 3.のポインタの間接参照(例では*p)が存在する。
6. 当該領域の値を更新する。
7. 構造体または共用体自身の参照を関数内で行っていない。

例：

```
typedef union {
  unsigned int a;
  unsigned int b:32;
} UN;
```

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

回避策： 以下のいずれかの方法で回避してください。

1. 構造体または共用体自身の参照を関数内で行う。

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

2. ポインタ変数をグローバルで定義する。
3. optimize=0オプションを指定する。

---

(10) 最後の処理が関数呼び出しである関数において、関数呼び出しを不当にJMP命令に変換する場合がある(SHC-0029)。

発生条件： 以下の条件をすべて満たした場合に発生することがあります。

1. 関数出口が複数ある。
2. 関数の最後以外で関数呼び出しで終わる出口がある。
3. 2の関数呼び出しの直前が関数呼び出しである。
4. 2の関数呼び出しと3の関数呼び出しはブロックが分かれている。
5. 当該関数でPR以外のレジスタ退避・回復がない。

例：
```
-----------------------------------------
// -speed指定
void func(int a, int s) {
   switch(a) {
   case 1:
      switch(s) {
      case 0:
         ng();    // 不正にJMPに変換される
         break;
      case 1:
```

```
            ok();
            break;
        }
        f1();
        break;
    case 2:
        f2();
        break;
    }
}
-----------------------------------------
```

回避策： 以下のいずれかの方法で回避してください。

1. 関数の最後に組み込み関数nop()を追加する。
2. 関数の最後を関数呼び出しにしない。

---

(11) 構造体または共用体コピーを行った場合、スタック領域が必要以上に確保される場合がある(SHC-0021)。

発生条件： 以下の条件をすべて満たした場合に発生することがあります。

1. 構造体または共用体型のメンバを持つ構造体または共用体配列が存在する。
2. 1の配列の要素数は2以上である。
3. 構造体または共用体代入文が存在する。
4. 3の左辺は1の構造体または共用体配列の構造体または共用体型メンバである。

例：

```
-----------------------------------------
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;  // スタックがA[1000]分余分に確保される
}
-----------------------------------------
```

回避策： 以下のいずれかの方法で回避してください。

1. 構造体または共用体代入文の左辺の構造体または共用体型メンバをポインタを用いて参照する。

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

2. 構造体(または共用体)代入文を各メンバの代入文に展開する。

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

3. **恒久対策**

   本内容は、SuperH RISC engine C/C++コンパイラ V.8.00.05で改修する予定です。