RENESAS ツールニュース

RENESAS TOOL NEWS 2006年04月01日：RSO-SHC_2-060401D

SuperH RISC engineファミリ C/C++コンパイラパッケージ V.9 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.9の使用上の注意事項 4件を連絡します。

1. **該当製品**
   SuperH RISC engine ファミリ C/C++コンパイラパッケージ
   V.9.00 Release 00 ～ V.9.00 Release 03

2. **内容**

   2.1 **構造体型または共用体型の配列を宣言してからその構造体または共用体を定義した場合の注意事項 (SHC-0064)**

   extern宣言された構造体型または共用体型の配列の宣言より後にその構造体 または共用体の定義がある場合に、 その配列の先頭以外の要素をアクセスすると、配列の先頭要素をアクセスするコードを生成する場合があります。

   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) extern宣言された構造体型もしくは共用体型の配列がある。

   (2) (1)の配列の宣言より後に、(1)の構造体もしくは共用体が定義されている。

   (3) (1)の配列の先頭以外の要素へアクセスしている。

   例：

```
-----------------------------------------------
extern struct TBL g[3];  // 発生条件(1)
struct TBL {             // 発生条件(2)
```

```
    int m;
};
struct TBL tbl;

void func() {
    tbl.m = g[1].m;      // 発生条件(3)
}
-------------------------------------------
コンパイル結果
-------------------------------------------
_func:
        MOV.L       L11+2,R1    ; _g
        MOV.L       L11+6,R4    ; _tbl
        MOV.L       @R1,R6      ; 本来ならg[1].mへのアクセス
が正しいが
                                ; 間違ってg[0].mをアクセスする。
        RTS
        MOV.L       R6,@R4
-------------------------------------------
```

回避策：
発生条件に該当する構造体もしくは共用体を、発生条件(1)の配列宣言より前に定義してください。

2.2 仮引数への代入を行なった場合の注意事項 (SHC-0065)

仮引数への代入を行なった場合に、その仮引数への代入が正しく行なわれない場合があります。

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) opitmize=1オプションを使用している。

(2) schedule=0オプションを使用していない。

(3) noscopeオプションを使用していない。

(4) 仮引数の型がsigned char型、unsigned char型、signed short型またはunsigned short型である。

(5) (4)の仮引数のアドレスを参照していない。

(6) (4)の仮引数の値が呼び元関数からレジスタ経由で渡される。

(7) (4)の仮引数への代入を行っている。

例：

```
-------------------------------------------
void func(unsigned short ss) {  // 発生条件(4) および (6)
. . . . . . . . . . . . . .
   ss = 0xffff - ss;          // 発生条件(7)
. . . . . . . . . . . . . .
}
-------------------------------------------
```

コンパイル結果

```
-------------------------------------------
_func1:
. . . . . . . . . . . . . . . . . .
        MOV        #-1,R1       ; H'FFFFFFFF
        EXTU.W      R1,R1
        SUB        R4,R1
        MOV        #30,R0       ; H'0000001E
        MOV.W       R1,@(R0,R15)  ; ss = 0xffff - ss (*)
        MOV.L       R4,@(28,R15)  ; *の左辺を上書き
. . . . . . . . . . . . . . . . . .
-------------------------------------------
```

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オプションを使用する。

(2) schedule=0オプションを使用する。

(3) noscopeオプションを使用する。

(4) 該当する仮引数をvolatile修飾する。

(5) 該当する仮引数を関数の先頭でローカル変数に代入して使用する。

2.3 スタック渡しの仮引数を持つ関数を使用した場合の注意事項 (SHC-0066)

スタック渡しの仮引数を持つ関数内で比較 (==または!=)またはビットフィールドを使用した場合に比較またはビットフィールドアクセスが正しく行なわれない場合があります。

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) -128以上127以下の定数値との等価比較 (==また

は!=) を行っている、またはビットフィールドを使用している。

(3) 当該関数にスタック渡しの仮引数が存在する。

例：

```
-----------------------------------------------
struct {
   unsigned int a:31;
   unsigned int b:1;
} *p;

void func(unsigned int a, unsigned int b, unsigned int c,
        unsigned int d, unsigned int g) {
   volatile int q[16];
   q[15]=0;
   p->b = g;     // 発生条件(2) および (3)
}
-----------------------------------------------
 コンパイル結果：
-----------------------------------------------
        ADD         #-64,R15
        MOV.L       L13,R4      ; _p
        MOV         #1,R7       ; H'00000001
        MOV.L       @R4,R6
        MOV         #0,R1       ; H'00000000
        MOV.B       @(3,R6),R0
        MOV         #64,R0      ; R0を破壊
        MOV.L       @(R0,R15),R5
        MOV.L       R1,@(60,R15)
        TST         R7,R5
        OR          #1,R0
        BF          L12
-----------------------------------------------
```

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オプションを使用する。

(2) 当該関数先頭において、すべてのスタック渡しの仮引数をvolatile変数に代入する。

2.4 条件文のthen節およびelse節でvolatile変数に0または1を代入した場合の注意事項(SHC-0067)

volatile 修飾した変数に対して条件分岐により0または1を設定するときに 不要なメモリアクセスを生成する場合があります。

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) if文もしくは条件演算子(?:)を使用している。

(3) (2)のif文または条件演算子の、then節およびelse節に、同じvolatile変数への代入文が1文ずつ存在する。(global_volatileオプションを指定している場合も含む)

(4) (3)の代入文では、一方は0を、もう一方は1を代入している。

(5) (2)のif文または条件演算子の、then節およびelse節の内容で異なるのは(4)の文だけである。

例：

```
-----------------------------------------------
   volatile int a;              // 発生条件(3)
   int b;
   void func(){
      if (b == 0)               // 発生条件(2)
      {
         a = 0;                 // 発生条件(3)、(4)および(5)
      } else {
         a = 1;                 // 発生条件(3)、(4) および(5)
      }
   }
-----------------------------------------------
```

コンパイル結果：

```
-----------------------------------------------
    MOV.L      L11,R5     ; _b
    MOV.L      @R5,R1
    TST        R1,R1
    MOVT       R4
    MOV.L      L11+4,R7   ; _a
    MOV.L      R4,@R7     ; ここで割り込みが発生した場合
    MOV.L      @R7,R0     ; a の内容が正しくない状態
    XOR        #1,R0
    RTS
    MOV.L      R0,@R7
```

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オプションを使用する。

(2) 発生条件(2)がif文である場合、then節またはelse節のどちらか一方に、nop()組み込み関数、 または発生条件(3)とは別のvolatile変数への代入文を追加する。

(3) 発生条件(2)が条件演算子である場合、if文に変更してから回避策(2)を適用する。

3. 恒久対策

本内容は、4月1日にリビジョンアップしたV.9.00 Release 04で 改修しました。

V.9.00 Release 04の詳細は、2006年4月1日発行のRENESAS TOOL NEWS 資料番号：RSO-SHC_1-060401D 「SuperH RISC engine ファミリC/C++コンパイラパッケージV.9.00 Release 04 へのリビジョンアップのお知らせ」を参照ください。