RENESAS ツールニュース

RENESAS TOOL NEWS 2005年11月01日：RSO-SHC_1-051101D

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 11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 typedefを使用して構造体配列または共用体配列を定義した場合の注意事項 (SHC-0039)
   構造体または共用体で仮宣言したタグ名を使用してtypedef宣言し、その後にその構造体または共用体を宣言した場合、このtypedef宣言した型を使用して 構造体配列または共用体配列を定義すると境界調整数が正しくない場合があります。

   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) 構造体または共用体で仮宣言したタグ名を使用してtypedef宣言を行っている。

   (2) (1)のtypedef宣言の後、構造体または共用体を宣言している。

   (3) (2)の構造体または共用体のメンバの少なくとも1つはchar型でもunsigned char型でもない。

   (4) (1)のtypedef宣言した型を使用して配列を定義している。

   (5) pack=1オプションを使用していない、かつ (4)の配

列は#pragma pack 1の対象ではない。

(6) (4)の配列はvolatile修飾もconst修飾もしていない。

(7) (4)の構造体配列のメンバまたは共用体配列のメンバの参照を行う関数を当該プログラムファイル内で定義していない。

例：

```
-----------------------------------------------
typedef struct ST ST_A;
struct ST {
   int x;
};
ST_A a[2];
-----------------------------------------------
```

コンパイル結果：

```
-----------------------------------------------
_a:                     ; static: a
        .RES.B     8        ; 正しくは.RES.L   2
-----------------------------------------------
```

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の定義の後にtypedef宣言を行う。

(2) 関数内で構造体配列または共用体配列のメンバ参照を行う。

(3) 構造体配列または共用体配列をvolatile修飾する。

(4) 構造体配列または共用体配列を、typedef宣言した型を使用しないで定義する。

2.2 1ビットのビットフィールドメンバへの代入を行った場合の注意事項(SHC-0040)

1ビットのビットフィールドメンバへの代入を行うと、正しくない値を代入する場合があります。

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) cpu=sh2aオプションまたはcpu=sh2afpuオプションを使用している。

(2) 1ビットのビットフィールドメンバを含む構造体配列

または共用体配列が存在する。

(3) (2)の配列の構造体または共用体のサイズは1バイトでも4バイトでもない。

(4) (2)の構造体配列または共用体配列の1ビットのビットフィールドメンバへの代入を行っている。

(5) (4)の代入が以下のaまたはbのいずれかの条件を満たしている。

a. 代入先の構造体配列または共用体配列の添え字に以下のいずれかを使用している。
もしくはポインタを介して代入を行い、かつアドレス計算に以下のいずれかを使用している。
- 関数呼び出し
- 構造体メンバまたは共用体メンバ(ビットフィールドを含む)
- 関係演算子または等値演算子
- シフト演算子
- 乗算、除算または剰余算
- long long型演算
- 浮動小数点型から整数型への型変換
- addc()、shll()などSRレジスタのTビットを更新する組み込み関数

b. 代入をBST.B命令で行い、かつBST.B命令の前にSHLL命令が存在する。

例：

```
-----------------------------------------------
struct {
   unsigned short a:8;
   unsigned short b:1;
} ST1[100],ST2[100];

void func(int i,int j) {
   ST1[i].b=ST2[j].b;
}
-----------------------------------------------
```

コンパイル結果：

```
-----------------------------------------------
_func:
        MOV.L      L11+2,R6   ; _ST2
        SHLL       R5
        ADD        R5,R6
        BLD.B      #7,@(1,R6) ; SRレジスタのTビットに値
```

を設定

```
        MOV.L      L11+6,R2   ; _ST1
        SHLL       R4         ; SRレジスタのTビットを更新
        ADD        R4,R2
        BST.B      #7,@(1,R2) ; 更新された値を代入
        RTS/N
```

回避策： 該当する構造体配列または共用体配列のアドレスをvolatile修飾したポインタ変数に代入して、そのポインタを使用して1ビットのビットフィールドメンバへの代入を行ってください。

2.3 volatile修飾した変数の参照順序に関する注意事項(SHC-0041)

volatile修飾した変数の参照が複数ある場合、コンパイルすると変数を参照する順序が変わる場合があります。

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) optimize=1オプションを使用している

(2) 以下のaまたはbのいずれかの条件を満たす関数の定義が存在する。
a. 関数呼び出しが存在する。
b. cpu=sh1以外のcpuオプションを使用し、かつ選択文、繰返し文、条件演算子および論理演算子のいずれかが存在する。

(3) (2)の文または演算子を含む式の中、もしくは前でvolatile修飾した変数の参照を行っている。

(4) (3)の変数の参照を含む式または文より前で、別のvolatile修飾した変数を参照している。

例：

```
volatile int a,b;
int c,d;

void func() {
   int x,y;
   do {
      x=a;            // 発生条件(4)
```

```
        y=b;              // 発生条件(3)
    } while (x != a);     // 発生条件(2)-b
}
```

コンパイル結果：

```
_func:
        MOV.L       L13,R4     ; _a
        MOV.L       L13+4,R1   ; _b
L11:
        MOV.L       @R4,R6     ; 発生条件(4)の参照。
                               ; ここで参照されるべき発生条件(3)の
                               ; 変数が参照されない。
        MOV.L       @R4,R2     ; 発生条件(2)-bの変数の参照。
        CMP/EQ      R2,R6
        BF/S        L11
        MOV.L       @R1,R5     ; 発生条件(3)の変数の参照。
        RTS/N
```

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)のaまたはbの文の直前に、nop()組み込み関数または 発生条件(3)および発生条件(4)とは別のvolatile修飾した変数への代入を追加する。

2.4 ビットフィールドメンバを含む共用体の初期値に関する注意事項(SHC-0042)

2.4.1 ビットフィールドメンバを含む共用体が初期値を持ち、かつビットフィールドメンバを下位ビット側から割り付けるように指示した場合、初期値が正しくない値になる場合があります。

発生条件： 以下の条件をすべて満たす場合に発生することがあります。

(1) 先頭のメンバがビットフィールドである共用体が存在する。

(2) (1)のビットフィールドメンバのビット長は、ビットフィールドを宣言した型のビット長ではない。

(3) (1)の共用体は負の数の初期値を持つ。

(4) bit_order=rightオプションを使用している。
もしくは(1)の共用体は#pragma bit_order rightの対象である。

例：

```
-----------------------------------------------
#pragma bit_order right
union {
   int a:2;
   int b;
}u={-1};
-----------------------------------------------
```

コンパイル結果：

```
-----------------------------------------------
_u:                          ; static: u
        .DATA.L     H'FFFFFFFF ; 正しくは
H'00000003
-----------------------------------------------
```

回避策： 以下のいずれかの方法で回避してください。

(1) 発生条件(1)のビットフィールドメンバと同じビット長のビットフィールド メンバのみを持つ構造体を共用体の先頭に宣言する。

```
例：    union {
                struct {
                   int dummy:2;
                }s;
                int a:2;
                int b;
             }u={-1};
```

(2) 初期値をビットフィールドメンバのビット長でマスクした値にする。

```
例：    union {
```

```
    int a:2;
    int b;
}u={-1&3};
```

2.4.2 ビットフィールドメンバを含む共用体が初期値を持ち、かつ初期値に冗長な {}がある場合に、初期値が正しくない値になる場合があります。

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) 先頭のメンバがビットフィールドである共用体が存在する。

(2) (1)のビットフィールドメンバのビット長はビットフィールドを宣言した型のビット長ではない。

(3) (1)の共用体は初期値を持つ。

(4) 初期値に冗長な{}がある。

例：

```
-----------------------------------------------
union {
   char a:3;
   short b;
}u={{0x8F}};
-----------------------------------------------
 コンパイル結果：
-----------------------------------------------
_u:                          ; static: u
        .DATA.B     H'8F       ; 正しくはH'E0
        .DATAB.B    1,0
-----------------------------------------------
```

回避策：冗長な{}を削除してください。

2.5 可変個数の引数を持つ関数に関する注意事項(SHC-0043)
可変個数の引数を持つ関数を再帰呼び出しした場合に、引数が正しく渡さ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す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) 可変個数の引数を持つ関数を再帰呼び出ししている。

(3) 引数の可変部分から2つ前の引数がレジスタ渡しになる。

(4) (3)の引数より前にスタック渡しになる引数がある。

(5) (2)の関数の戻り値の型はvoid型、整数型または浮動小数点型である。

(6) (2)のすべての引数の型は整数型または浮動小数点型である。

(7) 以下のaまたはbのいずれかの条件で再帰呼び出ししている。
   a. return文中で呼び出している。
   b. 関数の戻り値がvoid型でかつ関数の最後またはreturn文の直前で呼び出している。

例：

```
-----------------------------------------------
long long S;
void f(long long a1, int a2, int a3, ...) {
   S += a1;
   if (a1!=0) {
      f(a1-1, 0, 0);    /*  f(0, 0, 0);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オプションを使用する。

(2) 引数の少なくとも1つをvolatile修飾する。

(3) 発生条件(2)の関数を#pragma regsave宣言する。

(4) 引数の順番を発生条件(3)または発生条件(4)に該当しないようにする。

(5) 発生条件(7)のb項に該当する場合は、関数呼び出しの直後にnop()組み込み関数を追加する。

2.6 switch文の制御式がlong long型またはunsigned long long型の場合の注意事項(SHC-0044)

switch文の制御式がlong long型またはunsigned long long型で、かつ定数値になる場合に、誤ったcaseラベルの文へ制御が渡るか、またはアドレスエラーが発生する場合があります。

発生条件： 以下の条件をすべて満たす場合に発生することがあります。

(1) switch文が存在する。

(2) (1)のswitch文の制御式の型はlong long型またはunsigned long long型である。

(3) (2)の制御式は定数式である。

(4) (1)のswitch文のcase定数の最小値は0である。

(5) (1)のswitch文はテーブル方式で展開される。

例：

```
-----------------------------------------------
int func() {
   long long d=1;
   switch(d) {      /* 制御式の演算結果は常に1 */
   case 0:
   case 1:
   case 2:
   case 3:
   case 4:
   case 5:
   case 6:
   case 7:
   case 8:
      return 1;
      break;
   default:
      break;
   }
   return 0;
}
-----------------------------------------------
_func:
        STS.L      PR,@-R15
        MOV        #1,R6      ; H'00000001
```

```
        MOV.L      R6,@-R15
        MOV        #0,R1      ; H'00000000
        MOV.L      R1,@-R15
        MOV        #8,R5      ; H'00000008
        MOV.L      R5,@-R15
        MOV.L      L25+2,R4   ; __cmpgt64u
        JSR        @R4
        MOV.L      R1,@-R15
        ADD        #16,R15
        CMP/EQ     #0,R0
        BF         L23
        MOV        #1,R6      ; H'00000001
        MOVA       L26,R0
        MOV.L      @(R0,R6),R1; 奇数アドレスを参照してア
ドレスエラー
                              ; になる
        ADD        R1,R0
        JMP        @R0
        NOP
. . . . . .
-----------------------------------------
```

回避策： 以下のいずれかの方法で回避してください。

(1) switch文の制御式をvolatile修飾した変数に代入し、制御式をその変数に置き換える。

(2) 0より小さいcase定数を追加する。

(3) case=ifthenオプションを使用する。

2.7 整数定数を複数使用した場合の注意事項 (SHC-0045)

1つのブロック内で整数定数を複数使用した場合に、正しくない定数値を使用する場合があります。

発生条件： 以下の条件をすべて満たす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) -2, 147, 483, 648から-129の範囲または128から2, 147, 483, 647の範囲の整数定数を使用している。

(3) (2)の整数定数が含まれるブロックに、(2)の整数定数との差が-128から127の範囲である別の整数定数が存

在する。

(4) (2)の整数定数と(3)の整数定数は異なる型である。

例：

```
-----------------------------------------
int func() {
   unsigned short a;
   unsigned short *p;
   a = 65535;
   p = &a;
   *p *= 32767;
   if (*p == 32770) {    // 32769と32770の比較
      return 1;
   }
   return 0;
}
-----------------------------------------
```

コンパイル結果：

```
-----------------------------------------
_func:
        MOV.W       L11,R6     ; H'8001
        EXTU.W      R6,R2      ; R2に32769を設定
        ADD         #1,R6      ; R6に-32766を設定
        CMP/EQ      R6,R2      ; 32769と-32766の比較
        RTS
        MOVT        R0
-----------------------------------------
```

回避策： 以下のいずれかの方法で回避してください。

(1) 発生条件(2)の整数定数または発生条件(3)の整数定数をvolatile修飾した変数に代入して使用する。

(2) optimize=0オプションを使用する。

2.8 volatile修飾した配列への代入に関する注意事項 (SHC-0046)

if文の前とthen節に、volatile修飾した配列の同じ要素への代入式が存在する場合に、要素の参照回数が変わる場合があります。

発生条件： 以下の条件をすべて満たす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) 繰り返し文が存在する。

(3) (2)の繰り返し文の中にelse節を持たないif文が存在する。

(4) (3)のif文の前とthen節に配列の同じ要素への代入式が存在する。

(5) (4)の配列をvolatile修飾している。
または(4)の配列は外部変数で、かつ
global_volatile=1オプションを使用している。

例：

```
-----------------------------------------------
int x, y, z;
volatile int V[10];
void func() {
   int i;
   for(i = 0; i < 10; i++) {
      V[i] = x + y;      // 1回目の代入
      if(z) {
         V[i] = y + z;   // 2回目の代入
      }
   }
}
-----------------------------------------------
```

コンパイル結果：

```
-----------------------------------------------
_func:
        MOV.L      R14,@-R15
        MOV.L      L15,R5     ; _x
        MOV.L      L15+4,R4   ; _y
        MOV.L      L15+8,R2   ; _z
        MOV.L      @R5,R14
        MOV.L      @R4,R1
        MOV.L      @R2,R4
        ADD        R1,R14
        ADD        R4,R1
        MOV        #0,R5      ; H'00000000
        MOV        #10,R6     ; H'0000000A
        MOV.L      L15+12,R7  ; _V
L11:
        TST        R4,R4
```

```
        MOV       R14,R2
        BT        L13
        MOV       R1,R2
L13:
        ADD       #-1,R6
        MOV       R5,R0
        TST       R6,R6
        MOV.L     R2,@(R0,R7); 代入は常に1回しか行われ
ない
        ADD       #4,R5
        BF        L11
        RTS
        MOV.L     @R15+,R14
```

回避策： 以下のいずれかの方法で回避してください。

(1) 発生条件(3)のif文のthen節にnop()組み込み関数を追加する。

(2) 発生条件(3)のif文にelse節を追加する。

(3) optimize=0オプションを使用する。

2.9 volatile修飾したループ制御変数を持つ繰り返し文に関する注意事項 (SHC-0047)

volatile修飾したループ制御変数を持つ繰り返し文の繰り返し回数が正しくない場合があります。

発生条件： 以下の条件をすべて満たす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) 繰り返し文が存在する。

(3) (2)の繰り返し文はlong long型でもunsigned long long型でもない整数型のループ制御変数を持つ。

(4) (3)のループ制御変数をvolatile修飾している。
または(3)のループ制御変数は外部変数でかつglobal_volatile=1オプションを使用している。

(5) (3)のループ制御変数の更新式は加算式もしくは減算式である。

(6) (2)の繰り返し文の中に関数呼び出しがある。
もしくは(3)のループ制御変数をさすポインタ型変数が存在し、かつ(2)の繰り返し文の中でこのポインタ変数を更新している。

例：

```
-----------------------------------------------
 extern void sub();
 volatile int i;
 void func() {
    i = 1;
    while (i) {
       i--;
       sub();    // 関数呼び出し先でiが更新される可能性があるが、
                 // 繰り返し回数は常に1回になる
    }
    return;
 }
-----------------------------------------------
```

回避策： 以下のいずれかの方法で回避してください。

(1) 発生条件(5)の更新式の後に発生条件(3)のループ制御変数を参照する式を追加する。

(2) 発生条件(3)のループ制御変数をlong long型またはunsigned long long型にする。

(3) optimize=0オプションを使用する。

2.10 ループ制御式にカンマ演算子を使用した場合の注意事項 (SHC-0048)

繰り返し文のループ制御式にカンマ演算子を使用した場合に、式の評価を右側から行う場合があります。

発生条件： 以下の条件をすべて満たす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) 繰り返し文が存在する。

(3) (2)の繰り返し文はlong long型でもunsigned long long型でもない整数型のループ制御変数を持つ。

(4) (3)のループ制御変数の更新式は加算式もしくは減算式である。

(5) (2)の繰り返し文の繰り返し回数は1回である。

(6) (2)の繰り返し文のループ制御式でカンマ演算子を使用している。

例：

```
-----------------------------------------------
int A, B;
void func() {
   int i;
   for (i=0; A++, B+=A, i<1; i++) {    // B+=A, A++の順に実行される
      B++;
   }
}
-----------------------------------------------
```

回避策： 以下のいずれかの方法で回避してください。

(1) 繰り返し文を使用しない。

(2) 発生条件(3)のループ制御変数をlong long型またはunsigned long long型にする。

(3) optimize=0オプションを使用する。

2.11 関数呼び出しの前後で同じ定数を使用した場合の注意事項 (SHC-0049)

関数呼び出しの前後で同じ値の定数を使用した場合に、関数呼び出し後の定数値が正しくない場合があります。

発生条件： 以下の条件をすべて満たす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) opt_range=noblockオプションもglobal_alloc=0オプションも使用していない。

(3) 関数呼び出しが存在する。

(4) (3)の関数呼び出しの前後で同じ値の定数を使用している。

(5)　(4)の定数の少なくとも1つは繰り返し文で使用している。

(6)　(4)の定数に関数呼び出し前後で保証しないレジスタが割り付いている。

例：

```
-------------------------------------------
int array[10];
void func(int j, int k) {
   int i;
   for (i = 0; i < 10; i++) {
      if (k < 20) {
         sub();
      }
      if (j < 20) {
         array[i] = j;
      }
. . . . . .
   }
}
-------------------------------------------
```

コンパイル結果：

```
-------------------------------------------
_func:
. . . . . .
        MOV        R5,R11
        MOV        R4,R12
        MOV        #10,R14
        MOV.L      L17,R13    ; _array
L11:
        MOV        #20,R1     ; R1に20を設定
        CMP/GE     R1,R11
        BT         L13        ; (A)
        MOV.L      L17+4,R2   ; _sub
        JSR        @R2        ; (B)
        NOP
L13:
        CMP/GE     R1,R12     ; (A)で分岐しなかった場合、
(B)の関数
                              ; 呼び出し先でR1が更新される可能性
が
                              ; あるにもかかわらず使用
        BT         L15
```

```
        MOV.L      R12,@R13
. . . . . . . . . . . . . . . . . . . . . . . .

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

回避策： 以下のいずれかの方法で回避してください。

(1)　opt_range=noblockオプションを使用する。

(2)　global_alloc=0オプションを使用する。

(3)　optimize=0オプションを使用する。

3. **恒久対策**

次期リビジョンアップで改修する予定です。(2006年1Q予定)
