RENESAS ツールニュース

RENESAS TOOL NEWS 2014年02月01日：140201/tn1

SuperH RISC engineファミリ用 C/C++コンパイラパッケージ V.7～V.9 ご使用上のお願い

SuperH RISC engine ファミリ用C/C++コンパイラパッケージ V.7～V.9の使用上の注意事項 4 件を連絡します。

- 同一ループ内の異なる配列要素に、同一の添え字を使用した場合の注意事項 (SHC-0090)
- 命令並び替えの最適化を実施し、かつ組込関数を使用した場合の注意事項 (SHC-0091)
- 構造体または共用体型の関数内static変数を持つ関数がインライン展開された場合の注意事項 (SHC-0092)
- 標準ライブラリ関数 strcpy() を使用した場合の注意事項 (SHC-0093)

注: 各注意事項の後ろの番号は、注意事項の識別番号です。

---

1. 同一ループ内の異なる配列要素に、同一の添え字を使用した場合の注意事項 (SHC-0090)

1.1 該当バージョン

V.7.0B ～ V.9.04 Release 01

1.2 内容

同一ループ内の複数の配列要素に、同一の添え字を使用してアクセスした場合、誤った配列要素が参照されることがあります。

1.3 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=0 および -optimize=debug_only オプションを使用していない。
(2) ループ制御変数を持つループが存在する。
(3) (2)のループ内に配列要素 (注1) へのアクセスが存在する。
(4) (2)のループ内に(3)の配列とは異なる配列の配列要素 (注1) へのアクセスが存在する。
(5) (3)および(4)のそれぞれの配列要素へのアクセスは同じポインタ変数を介して行われる。

(6) (3)および(4)の配列要素へのアクセスの際の添え字式 (注2) は
以下(6-1)または(6-2)のいずれかを満たす。
(6-1) 同じ値の定数 (注3) である。
(6-2) 以下の全てを満たす。
- (2)のループ制御変数の1次式である。
- ループ制御変数と同じ型である。
- 更新時の増分値が同じである。

注1: 間接参照形式 (*(ary + index)) も含む。
注2: 間接参照形式*(ary + index) において、ary が配列変数名の場合
index の部分が添え字式である。
注3: 添え字式の値が結果として定数になる場合も含む。
例: int index = 0; ary[index];

1.4 発生例

```
----------------------------------------------------------------
long S=0;
void func(short *xxx, short *yyy) {
long i, t1, t2;
short *pt;
   for( i = 0 ; i < 2 ; ++i) {      // 発生条件(2)
      pt = xxx;                  // 発生条件(3)
      t1 = pt[i];                // 発生条件(5)および(6-2)
      pt = yyy;                  // 発生条件(4)
      t2 = pt[i];                // 発生条件(5)および(6-2)
      S += (t1 + t2);
   }
}
----------------------------------------------------------------
```

yyy[i]を誤ってxxx[i]とみなし、Sにxxx[i]+xxx[i]が加算される。

1.5 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 または -optimize=debug_only オプションを使用する。
(2) 以下のいずれかを volatile 修飾する。
- 発生条件(2)のループ制御変数
- 発生条件(3)の配列
- 発生条件(4)の配列
- 発生条件(5)のポインタ変数

(3) 発生条件(6-2)に該当する場合、添え字式の一方を異なる型を持つループ制御変数の1次式に変更する。
(4) 発生条件(4)の配列要素へ、発生条件(3)の配列要素へのアクセスに用いるポインタ変数とは別のポインタ変数を介してアクセスする。
(5) 発生条件(3)または発生条件(4)のいずれかの配列要素へ、ポインタ変数を介さずにアクセスする。

1.6 回避例

回避策(3)の場合の回避例:

```
-----------------------------------------------------------------
long i, t1, t2;
short *pt;
unsigned long k;                          // 回避策(3)
   for( i = 0, k = 0 ; i < 2 ; ++i, ++k) {   // 回避策(3)
      pt = xxx; t1 = pt[k];                 // 回避策(3)
      pt = yyy; t2 = pt[i];
      S += (t1 + t2);
   }
-----------------------------------------------------------------
```

回避策(4)の場合の回避例:

```
-----------------------------------------------------------------
short *xxx, *yyy, *pt, *pt2;   // 回避策(4)
pt = xxx;   t1=pt[i];
pt2 = yyy;  t2=pt2[i];        // 回避策(4)
-----------------------------------------------------------------
```

回避策(5)の場合の回避例:

```
-----------------------------------------------------------------
short *xxx,yyy[10], *pt;
pt = xxx;   t1=pt[i];
t2=yyy[i];                  // 回避策(5)
-----------------------------------------------------------------
```

2. 命令並び替えの最適化を実施し、かつ組込関数を使用した場合の注意事項 (SHC-0091)

2.1 該当バージョン

V.9.03 Release 00 ～ V.9.04 Release 01

2.2 内容

命令並び替えの最適化を実施し、かつ組込関数を使用している場合、
当該組み込み関数の後で、正しい命令が生成されないことがあります。

2.3 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) -cpu=sh2a または -cpu=sh2afpu オプションを使用している。
(2) -optimize=0 および -optimize=debug_only オプションを使用していない。
(3) -schedule=0 オプションを使用していない。
(4) 以下のいずれかの組み込み関数を使用している。
- bset()
- bclr()

- bcopy()
- bnotcopy()

2.4 発生例

-cpu=sh2aを使用した場合の発生例:

```
----------------------------------------------------------------
#include
void func()
{
   volatile char a[100];
   volatile char a100,a101,a102;
   a[55] = 0;
   a100 = 0;
   a101 = 0;
   bset((unsigned char *)(0xfffe3886),0); //発生条件(4)
   a102 = 0;
}
----------------------------------------------------------------
```

上記発生例のコンパイル結果:

```
----------------------------------------------------------------
MOV        #100,R2
MOV        #0,R1
ADD        R15,R2
MOV.B      R1,@(55:12,R15)
MOV.B      R1,@R2          ; a100に0を格納
MOV        R1,R0
MOV.B      R0,@(4,R2)      ; a101に0を格納
MOVI20     #-116602,R2     ; H'FFFE3886
BSET.B     #0,@(0,R2)
MOV.B      R0,@(8,R3)      ; 誤ってR3が参照されるため、
                           ; a102に0が格納されない
RTS
ADD        #112,R15
----------------------------------------------------------------
```

2.5 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 または -optimize=debug_only オプションを使用する。

(2) -schedule=0 オプションを使用する。

(3) 発生条件(4)の組み込み関数を使用しない記述に変更する。

2.6 回避例

回避策(3)の場合の回避例1:

bset()

【修正前のソースコード】

```
   bset(&a,0)
```

【修正後のソースコード】

```
   a|=0x01;
```

回避策(3)の場合の回避例2:
bclr()
【修正前のソースコード】

```
   bclr(&a,3)
```

【修正後のソースコード】

```
   a&=~(0x01 << 3);
```

回避策(3)の場合の回避例3:
bcopy()
【修正前のソースコード】

```
   bcopy(&a,1,&b,2);
```

【修正後のソースコード】

```
   if (a & (0x01 << 1)) {
      b |= (0x01 << 2);
   } else {
      b &= ~(0x01 << 2);
   }
```

回避策(3)の場合の回避例4:
bnotcopy()
【修正前のソースコード】

```
   bnotcopy(&a,1,&b,2);
```

【修正後のソースコード】

```
------------------------------------------------------------------
   if (!(a & (0x01 << 1))) {
      b |= (0x01 << 2);
   } else {
      b &= ~(0x01 << 2);
   }
------------------------------------------------------------------
```

3. 構造体または共用体型の関数内static変数を持つ関数がインライン展開された場合の注意事項(SHC-0092)

3.1 該当バージョン

V.7.0B ～ V.9.04 Release 01

3.2 内容

構造体または共用体型の関数内static変数を持つ関数が、インライン展開
された場合に、変数の正しい値を参照できない場合があります。

3.3 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=0 または -optimize=debug_only オプションを使用していない。
(2) 構造体または共用体型の関数内static変数が定義された関数が存在する。
(3) (2)の構造体または共用体はポインタ型のメンバを持つ。
(4) (3)のポインタ型メンバは初期値を持ち、かつその初期値は変数のアドレス
である。
(5) (2)の関数内で、(4)で初期値としてそのアドレスが参照される変数に対し、
以下の2種類のアクセス方法が混在する。
(5-1) 直接アクセス
(5-2) (3)のポインタ型メンバを経由した間接アクセス
(6) 以下のいずれかに該当し、かつ(2)の関数が呼出元の関数内にインライン
展開されている。 (注1)
(6-1) -inlineオプションを使用している。
(6-2) -speedオプションを使用し、かつ-noinlineオプションを使用して
いない。
(6-3) (2)の関数に#pragma inlineを指定している。
(7) (2)の関数の定義が削除されていない。 (注2)

注1: インライン展開が実施されたかどうかは、コンパイル結果のアセンブラ
ソースで確認できます。
注2: コンパイラの最適化の影響で、関数の定義が削除されることがあります。
関数の定義が削除されたかどうかは、コンパイル結果のアセンブラソース
で確認できます。

3.4 発生例

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

```
#pragma inline (func)          // 発生条件(6-3)
int xxx = 0;
struct ST {
   int * ppp;                  // 発生条件(3)
};

int func(void)
{
   static struct ST sss = {    // 発生条件(2)
      &xxx                     // 発生条件(4)
   };
      *(sss.ppp) = 1;          // 発生条件(5-2)
      return (xxx);            // 発生条件(5-1)
}

int main(void) {
   return (func());            // 発生条件(6)
}
```

上記発生例のコンパイル結果:

```
_main:
   MOV.L      L12,R5      ; sss のアドレスを取得
   MOV        #1,R2       ; H'00000001 を準備
   MOV.L      L12+4,R4    ; xxx のアドレスを取得
   MOV.L      @R5,R1      ; sss.ppp の値を取得
   MOV.L      @R4,R0      ; 戻り値用レジスタ(R0)にxxxの値を格納
   MOV.L      R2,@R1      ; *(sss.ppp)に 1 を格納
   RTS
```

---

*(sss.ppp)に1 を格納する前のxxxの値が、戻り値としてR0に格納される。
本来は、*(sss.ppp)に1を代入した後のxxxの値を、R0に格納。

3.5 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 または -optimize=debug_only オプションを使用する。
(2) 発生条件(3)のポインタ型メンバの値は、初期値ではなく、代入文で
   設定する。
(3) 発生条件(5)の 2 種類のアクセス方法を、いずれか一方に統一する。
(4) 発生条件(2)の関数内で宣言された構造体または共用体型のstatic変数を
   ファイル内static変数に変更する。
(5) 以下のいずれかを実施し、インライン展開を抑止する。
   (5-1) 発生条件(6-1)に該当する場合、-inlineを使用しない。
   (5-2) 発生条件(6-2)に該当する場合、-noinlineオプションを使用する。

(5-3) 発生条件(6-3)に該当する場合、#pragma inline指定を削除する。

3.6 回避例

回避策(2)の場合の回避例:

```
-----------------------------------------------------------------
static struct ST sss;
sss.ppp = &xxx;
-----------------------------------------------------------------
```

回避策(3)の場合の回避例1:
直接アクセスに統一する場合

```
-----------------------------------------------------------------
xxx = 1;
return (xxx);
-----------------------------------------------------------------
```

回避策(3)の場合の回避例2:
ポインタ型を経由した間接アクセスに統一する場合

```
-----------------------------------------------------------------
*(sss.ppp) = 1;
return (*(sss.ppp));
-----------------------------------------------------------------
```

4. 標準ライブラリ関数 strcpy() を使用した場合の注意事項(SHC-0093)

4.1 該当バージョン

V.7.0B ～ V.9.04 Release 01

4.2 内容

標準ライブラリ関数 strcpy() の第1引数に指定されたコピー先のメモリ
領域と、strcpy()の前後で使用している変数のメモリ領域が重なっている場合、
その変数の値が正しく更新されないことがあります。

4.3 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=debug_only オプションを使用していない。
(2) -blockcopy=inline オプションを使用している。 (注)
(3) 標準関数 strcpy() を呼び出している関数が存在する。
(4) (3)の関数内で、strcpy() の呼び出しより前に値の代入があり、
かつstrcpy()呼出しより後に参照される変数、構造体メンバ、
または共用体メンバのいずれかが存在する。
(5) (4)の変数、構造体メンバおよび共用体メンバの型は、
signed long long型およびunsigned long long型以外のスカラ型である。
(6) (3)の strcpy() 呼び出しの第1引数に指定されたコピー先のメモリ領域と、
(4)の変数、構造体メンバまたは共用体メンバが格納されたメモリ領域が
重なっている。

注: -nospeedまたはspeed オプションを使用している場合、
-blockcopy=inline オプションがデフォルトで使用されます。

4.4 発生例

```
---------------------------------------------------------------
#include
union
{
      char a[4];
      int  b;       // 発生条件(5)
} u;
   int c;
void func(void)       // 発生条件(3)
{
      u.b = 1;           // 発生条件(4)
      strcpy(u.a, "T");    // 発生条件(3)および(6)
      c = u.b;           // 発生条件(4)
}
---------------------------------------------------------------
```

本来、変数cには"T"が格納されるが、
誤って、strcpy()によって更新される前のu.bの値である1が格納される。

上記発生例のコンパイル結果:

```
---------------------------------------------------------------
_func:
   MOV        #1,R5
   MOV.L      L12+2,R1
   MOV.L      R5,@R1     ; u.bに1を代入
   MOV.L      L12+6,R6
   MOV.W       @R6,R7
   MOV.L      L12+2,R4   ; u
   MOV.L      L12+10,R2  ; c
   MOV.W       R7,@R4     ; strcpy(u.a, "T")によって u を更新
   RTS
   MOV.L      R5,@R2     ; 変数cに 1 (更新前の u.b の値)を代入
---------------------------------------------------------------
```

4.5 回避策

以下のいずれかの方法で回避してください。
(1) -optimize=debug_only オプションを使用する。
(2) -blockcopy=inline オプションを使用しない。 (注)
(3) 発生条件(4)の変数、構造体メンバまたは共用体メンバを以下の
いずれかの方法で修正する。
(3-1) volatile 修飾する。

(3-2) 長さ 1 の配列型に変更する。

注: -speedまたは-nospeedを使用している場合は、
デフォルトで-blockcopy=inlineが有効になりますので、
-blockcopy=runtimeオプションを使用してください。

4.6 回避例

回避策(3-2)の場合の回避例:

```
----------------------------------------------------------------
union {

   char a[4];
   int  b[1];
} u;
----------------------------------------------------------------
```

5. 恒久対策

4件の問題はすべてSuperH RISC engine ファミリ C/C++コンパイラパッケージ V.9.04 Release 02 で改修しました。
V.9.04 Release 02の詳細は、RENESAS TOOL NEWS 資料番号 140201/tn2 をご参照ください。以下のURLで参照できます。 (2月4日から公開予定)
https://www.renesas.com/search/keyword-search.html#genre=document&q=140201tn2