RENESAS ツールニュース

RENESAS TOOL NEWS 2009年01月16日：090116/tn1

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

1. 繰り返し文内でアドレスを参照した場合の注意事項(SHC-0073)
2. 参照と設定が行われている1ビットのビットフィールドメンバが複数ある場合の 注意事項(SHC-0074)
3. if文で、変数に0、1、または-1を設定した場合の注意事項(SHC-0075)
4. #pragma global_register を宣言した変数の値を参照または設定した場合の 注意事項(SHC-0076)
5. signed long long型またはunsigned long long型のビットフィールドを 参照した場合の注意事項(SHC-0077)
6. リンク時のmapオプション使用時の注意事項(LNK-0002)
7. CRC演算を行う場合の注意事項(LNK-0003)
8. 異なる境界調整数のセクションのオーバーレイに関する注意事項(LNK-0004)
9. リンク時のdata_stuffオプションおよびnooptimizeオプション選択に関する 注意事項(LNK-0005)

---

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.9.02 Release 00

2. C/C++コンパイラ

2.1 繰り返し文内でアドレスを参照した場合の注意事項(SHC-0073)

該当バージョン:
V.9.00 Release 00 ～ V.9.02 Release 00

現象:
繰り返し文のループ本体で配列要素のアドレス、もしくは構造体または

共用体のメンバのアドレスを、ポインタ変数を使って参照する場合、ポインタ変数の値が間違っている場合があります。

発生条件:

以下の条件をすべて満たした時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) 仮引数、自動変数、もしくは関数内static変数のいずれかをポインタ変数として使用している。

(3) (2)のポインタ変数のアドレスを参照していない。

(4) (2)のポインタ変数を用いて配列、構造体、または共用体メンバのアドレスを参照している。

(5) (4)でアドレスを参照したメンバの配列、構造体、または共用体の先頭からのオフセット値が1バイト以上127バイト以下になっている。

(6) 繰り返し文が存在する。

(7) (6)の繰り返し文のループ本体に、(2)、(3)、(4)、および(5)全てに該当するアドレス参照が複数個ある。

(8) (7)と同じループ本体で、(7)のアドレス参照の他にも(7)と同じポインタ変数を使用している式がある。

(9) (7)と同じループ本体以外では(7)のアドレス参照と同一のアドレスを参照していない。

(10) (8)で使用しているポインタ変数がR8, R9, R10, R11, R12, R13, またはR14レジスタのいずれかに割りついている。

例:

```
-----------------------------------------------
struct ST {
   int mem1;
   int mem2;
};
int a,b,*pa,*pb,*pc;
void func(struct ST *pst)       // 発生条件 (2)
{
   while (a > 0) {
```

```
      sub();
      a = pst->mem1;         // 発生条件 (8)
. . . . . . . . . . . . . . . . . . . . . . . .
      pa = &(pst->mem2);     // 発生条件 (4),(5),(7)
      pb = &(pst->mem2);     // 発生条件 (4),(5),(7)
. . . . . . . . . . . . . . . . . . . . . . . . . . .
      if (b > 0) {
         pc = &(pst->mem2);  // 発生条件 (4),(5),(7)
      }
   }
}
```

コンパイル結果

```
. . . . . . . . . . . . . . . . . . . . . . . .
       MOV        R4,R13      ; 発生条件(10):変数pstがR13に
                              ; 割り付いている。
. . . . . . . . . . . . . . . . . . . . . . . . . .
       MOV.L      @(4,R15),R14 ; 値が設定されていないスタック
                              ; 領域からpst->mem2のアドレス値を
                              ; 参照している。
       MOV.L      R14,@R9     ; paに代入
       MOV.L      R14,@R10    ; pbに代入
. . . . . . . . . . . . . . . . . . . . .
       MOV        R13,R14
       ADD        #4,R14
       MOV.L      R14,@R8     ; pcに代入
. . . . . . . . . . . . . . . . . . . . .
```

回避策：

以下のいずれかの方法で回避してください。

(1) optimize=1の代わりに、optimize=0またはoptimize=debug_only
オプションを使用する。

(2) 発生条件に該当するポインタ変数のアドレス参照をおこなう。

例：

```
. . . . . . . . . . . . . . . . . .
void func(struct ST *pst)
{
```

```
    &pst;                   // 追加
    while (a > 0) {
. . . . . .
```

2.2 参照と設定が行われている1ビットのビットフィールドメンバが複数ある場合の注意事項(SHC-0074)

該当バージョン:

V.9.01 Release 00 ～ V.9.02 Release 00

現象:

参照と設定が行われている1ビットのビットフィールドメンバが複数ある場合
そのビットフィールドメンバに間違った値が設定される場合があります。

発生条件:

以下の条件をすべて満たした時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) cpu=sh2a、またはcpu=sh2afpuオプションを使用している。

(3) 1ビットのビットフィールドメンバが複数存在する。

(4) (3)の複数のメンバのうち、値の参照と設定をしているメンバ
が複数ある。

(5) (4)のメンバのうちで、参照と設定が連続していないメンバが存在する。

例:

```
struct aa {
    unsigned char bf1:1;  // 発生条件(3)
    unsigned char bf2:1;  // 発生条件(3)
} aaf;

void main()
{
    unsigned char getfl1;
    unsigned char getfl2;

    // 発生条件(4)(5): bf1参照
    getfl1 = (*(volatile struct aa *)0x80FFFD00).bf1;
```

```
    // 発生条件(4)(5): bf2参照
    getfl2 = (*(volatile struct aa *)0x80FFFD00).bf2;

    // 発生条件(4)(5): bf1設定
    (*(volatile struct aa *)0x80FFFD00).bf1 = getfl1;

    // 発生条件(4)(5): bf2設定
    (*(volatile struct aa *)0x80FFFD00).bf2 = getfl2;
}
```

コンパイル結果

```
    MOV.L   L11,R5          ; H'80FFFD00
    BLD.B   #7,@(0,R5)      ; aaf.bf1をステータスレジスタに設定
    MOV.B   @R5,R0          ; aaf
    TST     #64,R0          ; ステータスレジスタの値を破壊
    MOVRT   R6
    BST.B   #7,@(0,R5)      ; aaf.bf1に、破壊されたステータスレジスタの
                            ; 値をストア
    BLD     #0,R6
    BST.B   #6,@(0,R5)
    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=1の代わりに、optimize=0、またはoptimize=debug_only オプションを使用する。

(2) 発生条件(4)の全ての参照について、以下のいずれかを実施する。

(2-1) 参照したメンバの値を変数に代入する場合は、代入先の変数をvolatile修飾する。

(2-2) (2-1)以外の場合は、そのメンバの値を、volatile修飾した

自動変数に代入し、メンバの参照をその自動変数の参照で置き換える。

例:
<変更前>
```
-----------------------------------------------
st.b1 &= st.b2;         // b1の参照、設定
-----------------------------------------------
```
<変更後>
```
-----------------------------------------------
// volatile変数vtempに代入して、元の参照をvtempで置換
 volatile unsigned char vtemp = st.b1;
 st.b1 = vtemp & st.b2;
-----------------------------------------------
```

(3) 発生条件(4)の全てのメンバについて、参照と設定が連続するようにプログラムを変更する。
例: 以下のように変更する。
```
-----------------------------------------------
  // 参照の直後に設定: bf1
  getfl1 = (*(volatile struct aa *)0x80FFFD00).bf1;
  (*(volatile struct aa *)0x80FFFD00).bf1 = getfl1;
  // 参照の直後に設定: bf2
  getfl2 = (*(volatile struct aa *)0x80FFFD00).bf2;
  (*(volatile struct aa *)0x80FFFD00).bf2 = getfl2;
-----------------------------------------------
```

2.3 if文で、変数に0、1、または-1を設定した場合の注意事項(SHC-0075)

該当バージョン:
V.9.01 Release 00 ～ V.9.02 Release 00

現象:
if文のブロック内で変数に0、1、または-1を設定する場合に、
正しい値が設定されていないメモリ領域を参照することがあります。

発生条件:
以下の条件をすべて満たした時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) 以下のいずれかを満たすif文が存在する。
(2-1) then節に式文が無い。

(2-2) else節に式文が無い。
(2-3) else節が無い。

(3) (2)のif文において、文が存在する方の節には、1つの代入文だけが存在する。

(4) (3)の代入文で変数に0, 1, または-1のいずれかを代入している。

(5) (2)のif文より前に、(4)と同じ変数に0, 1, または-1のいずれかを代入する文がある。

(6) (4)および(5)において、代入する値の組み合わせは、以下のいずれかである。

(4)の値	(5)の値
0	1
0	-1
1	0
-1	0

(7) 使用しているコンパイラがV.9.01 Release 00 および V.9.01 Release 01 以外の該当バージョンの場合、以下のいずれかを満たす。
(7-1) (4)および(5)の代入先変数は、外部変数ではない。
(7-2) opt_range=allまたはopt_range=noloopオプションを使用している。

(8) (5)の代入文から(2)のif文までの間で、(4)および(5)の代入先変数、またはその変数がストアされている同じメモリ領域を参照している。

例:

```
-----------------------------------------------
int x;
void sub(int a)
{
   union U { int a; short b; } u;  // 発生条件(7-1)
   u.a = -1;                       // 発生条件(5)(6)
   x = u.b;                        // 発生条件(8)
   if (a == 0) {                     // 発生条件(2-3)
      u.a = 0;                     // 発生条件(3)(4)(6)
   }
   x += u.a;
```

```
}
-----------------------------------------------
```

コンパイル結果

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

    ADD       #-4,R15
    TST       R4,R4
    MOV.W     @R15,R5     ; u.b ※@R15の値は未
                          設定
    SUBC      R2,R2
    MOV       #-1,R6      ; H'FFFFFFFF
    XOR       R6,R2
    MOV.L     L16+2,R1    ; _x
    ADD       R2,R5
    MOV.L     R5,@R1      ; x
    RTS
    ADD       #4,R15

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

回避策:

以下のいずれかの方法で回避してください。

(1) optimize=1の代わりに、optimize=0またはoptimize=debug_only
オプションを使用する。

(2) 発生条件(4)の変数をvolatile修飾する。

(3) 発生条件(3)の代入文の前または後に、組み込み関数nop()を挿入する。

(4) 発生条件(2-1)を満たす場合、then節に組み込み関数nop()を挿入する。

(5) 発生条件(2-2)を満たす場合、else節に組み込み関数nop()を挿入する。

(6) 発生条件(2-3)を満たす場合、組み込み関数nop()を含むelse節を
記述する。

(7) 発生条件(7-2)のみに該当する場合、opt_range=allおよび
opt_range=noloop の代わりに、opt_range=noblockオプションを

使用する。

2.4 #pragma global_register を宣言した変数の値を参照または設定した場合の注意事項(SHC-0076)

該当バージョン:

V.9.00 Release 00 ～ V.9.02 Release 00

現象:

#pragma global_register を宣言した変数の値を参照または設定した場合、
その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した時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) #pragma global_register を宣言した変数の値を参照または
設定している。

例:

```
-----------------------------------------------
#pragma global_register val_R8=R8  // 発生条件(2)

int val_R8;
int x;

void sub(int xxx, int yyy)
{
   unsigned short zzz = xxx;
   if (yyy) {
      x = zzz;
   } else {
      val_R8 = zzz;            // 発生条件(2)
   }
}
-----------------------------------------------
```

コンパイル結果

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

            TST       R5,R5
            EXTU.W    R4,R8       ; yyyの値に関わらず
                                  val_R8(R8)を設定
```

```
        BT      L12
        MOV.L   L20+2,R6  ; _x
        MOV.L   R8,@R6    ; x
L12:
        RTS
        NOP
```

回避策:

以下のいずれかの方法で回避してください。

(1) optimize=1の代わりに、optimize=0、またはoptimize=debug_only オプションを使用する。

(2) #pragma global_register を使用しない。

2.5 signed long long型またはunsigned long long型のビットフィールドを参照した場合の注意事項(SHC-0077)

該当バージョン:

V.9.00 Release 00 ～ V.9.02 Release 00

現象:

signed long long型(注)またはunsigned long long型のビットフィールドを参照した場合、その値が正しくない場合があります。

注: signedを省略した場合、signed long long型として扱います。

発生条件:

以下を全て満たす場合に発生することがあります。

(1) signed long long型、またはunsigned long long型のビットフィールドメンバを参照している。

(2) (1)のsigned long long型、またはunsigned long long型の変数は、上位32ビットが0でなく、かつ下位32ビットが0である。

(3) (1)のビットフィールドメンバは、(2)の最下位ビットを除く下位32ビットのいずれかである。

例:

```
-------------------------------------------
struct tbl {
    unsigned long long bit1:32;
    unsigned long long bit2:1;
    unsigned long long bit3:29;
    unsigned long long bit4:1;
    unsigned long long bit5:1;
} s = { 1, 0, 0, 0, 0};          // 発生条件(1)(2)(3)

#include<stdio.h>
void main(void){
    if (s.bit2 != 0) {       // 発生条件(1)
        printf("NG¥n");
    }
}
-------------------------------------------
```

回避策:

以下のいずれかの方法で回避してください。

(1) 発生条件に該当するビットフィールドの下位32ビットに0以外の値を
設定する。
上記例の場合、 s = { 1, 0, 0, 0, 1}, s = { 1, 0, 1, 0, 0}
など。

(2) 発生条件に該当するビットフィールドメンバの型を、signed long long
およびunsigned long long型以外の型に変更する。
上記例の場合、unsigned long long を全てunsigned long に変更する。

3. 最適化リンケージエディタ

3.1 リンク時のmapオプション使用時の注意事項(LNK-0002)

該当バージョン:

V.9.00 Release 04A ～ V.9.02 Release 00

現象:

オーバーレイ機能とmapオプションを組み合わせて使用したときに、
誤ったアドレスで変数を参照することがあります。

発生条件:

以下を全て満たす場合に発生することがあります。

(1) 初回リンク時および2回目コンパイル時にmapオプションを
使用している。

(2) リンク時のstartオプションの指定内容が以下の条件をすべて
満たしている。
(2-1) startオプションでオーバーレイ宣言をしている。
例: -start=(B:P),D1,D2/100
BおよびPがセクション名で、(B:P)でオーバーレイを宣言している。
(2-2) (2-1)のオーバーレイ宣言の後ろで、複数のデータセクションを
連続して配置する宣言をしている。
(2-1)の例ではD1とD2のデータセクションを連続して配置している。
(2-3) (2-2)の連続しているセクションのうちで直前のセクションよりも
境界調整数が大きくなるよう設定したセクションがある。
(2-1)の例では、D1の境界調整数は2バイト、D2の境界調整数は
4バイトである。

回避策:
オーバーレイ宣言をする場合は、mapオプションを使用しないでください。

3.2 CRC演算を行う場合の注意事項(LNK-0003)

該当バージョン:
V.9.02 Release 00

現象:
一つのセクション内のモジュール間に空き領域がある場合や、リンク後の
セクションの末端のモジュールのサイズが0の場合に、誤ったCRC演算結果
になります。

発生条件:
以下を全て満たす場合に発生する場合があります。

(1) リンク時にcrcオプションを選択している。

(2) リンカに入力するファイルがリロケータブル形式である。

(3) 以下の(3-1)または、(3-2)を満たしている。
(3-1) 以下の(3-1-1)から(3-1-3)を全て満たしている。
(3-1-1) セクション内のモジュール間に空き領域がある。
(3-1-2) CRC演算対象の開始アドレスが、(3-1-1)の空き領域内である。
(3-1-3) (3-1-1)の空き領域は、初期値ありのデータセクションまたは、
プログラムセクションである。

(3-2) 以下の(3-2-1)から(3-2-3)を全て満たしている。
(3-2-1) リンク後のセクション内の末端のモジュールのサイズが0で
ある。
(3-2-2) CRC演算対象の範囲が(3-2-1)のセクションを対象としており、
演算対象の範囲が、(3-2-1)のサイズ0のモジュールを含む。
(3-2-3) (3-2-1)のセクションは、初期値ありのデータセクション
またはプログラムセクションである。

回避策:
アブソリュートロードモジュールに対して、-crcオプションを選択し
再ビルドする。

3.3 異なる境界調整数のセクションのオーバーレイに関する注意事項(LNK-0004)

該当バージョン:
V.9.00 Release 04A ～ V.9.02 Release 00

現象:
異なる境界調整数のセクションをオーバーレイ宣言した場合に、最初に宣言
したオーバーレイセクションの境界調整が、誤って他のオーバーレイ
セクションを割り付けます。

発生条件:
以下(1)または(2)いずれかの条件を満たす場合に発生する場合があります。

(1) 以下の全ての条件を満たす。
(1-1) オーバーレイ指定で、()を使用していて、()の前後のいずれかに
セクションがある。
(1-2) セクションごとの境界調整数が少なくともひとつは異なる。
(1-3) startオプションで指定した開始アドレスが、各セクションの
境界調整数の公倍数ではない。
例) -start=(P1:P2),P3,(P4:P5),P6/100
- P1,P3,P4,P6の境界調整数が少なくともひとつは異なる。
- 開始アドレス100がP1,P3,P4,P6の境界調整数の公倍数ではない。

(2) 以下の全ての条件を満たす。
(2-1) オーバーレイ指定した先頭の境界調整数がオーバーレイ関係にある
ほかのいずれかの先頭のセクションの境界調整数より大きい。
(2-2) startオプションで指定したアドレスが先頭に指定したセクションの
境界調整数の倍数ではない。
例) -start=A1,A2:B1,B2:C1,C2/152
- A1がB1,C1の境界調整数より大きい。
- 開始アドレス152がセクションA1の境界調整数の倍数ではない。

回避策:

以下のいずれかを実施して回避してください。

(1) 発生条件(1)に該当する場合、セクションの境界調整数を全て同じにする。

(2) 発生条件(2)に該当する場合、境界調整数の一番小さい先頭セクションを最初にオーバーレイ宣言する。

(3) startオプションで各セクションの境界調整数の公倍数を開始アドレスに指定する。

3.4 リンク時のdata_stuffオプションおよびnooptimizeオプション選択に関する注意事項(LNK-0005)

該当バージョン:

V.9.00 Release 00 ～ V.9.02 Release 00

現象:

data_stuffオプションおよびnooptimizeオプションを選択し、データの詰め込みを行ったときに偶数サイズの変数を、誤って奇数番地に割り付けることがあります。

発生条件:

以下を全て満たす場合に発生します。

(1) リンク時にdata_stuffオプションを選択している。

(2) リンク時にnooptimizeオプションを選択している。

(3) 複数のオブジェクトファイルをリンクしている。

(4) (3)のオブジェクトファイルのうち、少なくとも2つに同一名称のデータセクションがある。

(5) (4)の同一名称のデータセクションのうち、入力ファイルで指定された2つ目以降のオブジェクトにある当該セクションの最後尾が1バイトのデータである。

(6) (5)のセクション内に、セクションの境界調整数と同じシンボルが存在しない。

発生例：

```
-----------------------------------------------
//a.c    //発生条件(4)
char a;
//b.c     //発生条件(4)
short b; //発生条件(6)
char c;  //発生条件(5),(6)
-----------------------------------------------
```

リンカコマンド

```
-----------------------------------------------
optlnk a.obj b.obj ?data_stuff -nooptimize
-----------------------------------------------
```

回避策:

以下のいずれかの方法で回避してください。

(1) リンク時にdata_stuffオプションを選択しない。

(2) 本現象の発生したソースファイル内で、奇数番地が割り付けられたシンボルがあるセクションに、セクションの境界調整数と同じサイズのダミーの変数を定義する。

例：

```
-----------------------------------------------
//b.c
long dummy;
short b;
char c;
-----------------------------------------------
```

(3) 本現象の発生したソースファイル内で、奇数番地が割り付けられたシンボルがあるセクションの末端に割り付くように2バイトのダミー変数を定義する。

例：

```
-----------------------------------------------
//b.c
short b;
char c;
short dummy;
-----------------------------------------------
```

4. 恒久対策

本内容は、SuperH RISC engine ファミリ C/C++コンパイラパッケージ

V.9.03 Release 00で改修する予定です。
