お客様各位

カタログ等資料中の旧社名の扱いについて

2010年4月1日を以ってNECエレクトロニクス株式会社及び株式会社ルネサステクノロジが合併し、両社の全ての事業が当社に承継されております。従いまして、本資料中には旧社名での表記が残っておりますが、当社の資料として有効ですので、ご理解の程宜しくお願い申し上げます。

ルネサスエレクトロニクス ホームページ（http://www.renesas.com）

2010年4月1日
ルネサスエレクトロニクス株式会社

【発行】ルネサスエレクトロニクス株式会社（http://www.renesas.com）

【問い合わせ先】http://japan.renesas.com/inquiry

（注）本文中のURLは、以下の様に変更になりました。
http://tool-support.renesas.com/jpn/toolnews/shc/shcv7/dr_shcv7_1.html

2002 年 5 月 31 日

― 日立半導体技術情報 ―

〒１００－０００４
東京都千代田区大手町 2 丁目 6 番 2 号
(日本ビル)
TEL (03)5201-5022 (ダイヤルイン)
株式会社 日立製作所 半導体グループ

製品分類	開発環境		発行番号	TN-CSX-038A		
題名	SuperH RISC engine C/C++コンパイラ Ver.7.0 不具合のご連絡(2)		情報分類	1. 仕様変更 2. ドキュメント訂正追加等 ③. 使用上の注意事項 4. マスク変更 5. ライン変更		
適用製品	SH-1,SH-2,SH-2E,SH2-DSP, SH-3,SH3-DSP,SH-4	対象ロット等 全ロット	関連資料	―	Rev. 第１版	有効期限 永年

SuperH RISC engine C/C++コンパイラ Ｖｅｒ.7.0 に別紙に示す不具合があります。
次に示す製品を御使用のお客様につきましては周知願います。

型名	パッケージバージョン	コンパイラバージョン
P0700CAS7-MWR	7.0B	7.0B
	7.0.01	7.0.03
	7.0.02	7.0.04
P0700CAS7-SLR	7.0B	7.0B
	7.0.02	7.0.03
	7.0.03	7.0.04
P0700CAS7-H7R	7.0B	7.0B
	7.0.02	7.0.03
	7.0.03	7.0.04

なお、プログラムが本不具合に該当しているかを検出するチェックツールを以下よりダウンロードできます。
http://www.hitachisemiconductor.com/sic/jsp/japan/jpn/PRODUCTS/MPUMCU/TOOL/download/crosstool/release/rest_shcv7002.html

添付 ： P0700CAS7-020514J
SuperH RISC engine C/C++コンパイラ Ver.7 不具合内容(2)

SuperH RISC engine C/C++ コンパイラ Ver.7
不具合内容 (2)

SuperH RISC engine C/C++コンパイラ Ver.7.0B.002、Ver.7.0.03 および Ver.7.0.04 における不具合内容を以下に示します。以下の不具合は 4、7、9、12 を除いてチェックツールを使用することにより、プログラムに当該ケースが存在するか確認することができます。チェックツールは以下 URL より入手できます。

http://www.hitachisemiconductor.com/sic/jsp/japan/jpn/PRODUCTS/MPUMCU/TOOL/download/crosstool/release/rest_shcv7002.html

1. PR レジスタ退避・回復命令の不当削除

【現象】

以下 C ソースを speed オプションでコンパイル時、不当に PR レジスタの退避・回復命令を削除し、実行時無限ループする場合がある。

＜例＞

```
int x;
extern void f1();
extern void f2();
void f() {
   if (x == 2){
      f1();          // then 節が関数呼び出しで終わっている
   }
   f2();             // 関数の最後の処理が関数呼び出し
   return;
}
```

```
_f:
        MOV.L      L14,R6          ; _x
        MOV.L      @R6,R0
        CMP/EQ     #2,R0
        BT         L11
L12:
        MOV.L      L14+4,R2        ; _f2
        JMP        @R2             ; 関数の最後が関数呼び出しのため、JMP 命令に
        NOP                        ; 変換し、RTS を削除
L11:
        MOV.L      L14+8,R2        ; _f1
        JSR        @R2             ; ここに関数呼び出し命令があるにも関わらず
        NOP                        ; PR レジスタの退避回復が行われていない
        BRA        L12
        NOP
```

【発生条件】

以下の条件をすべて満たす場合、発生することがあります。

(1) speed オプションを指定している。

(2) 関数の最後の処理が関数呼び出しである。

(3) (2)の関数呼び出しの直前に if-then 節があり、then 節の最後が関数呼び出しである。

(4) (2)の関数呼び出しがインライン展開されない。

【回避方法】

該当個所が存在するかを確認するチェックツールを配布いたします。

該当個所が存在した場合､以下のいずれかの方法で回避していただきますようお願いします。

(1) 該当ファイルを nospeed オプションまたは size オプションを指定してコンパイルする。

(2) nop 組み込み関数を関数末尾に追加する。

＜例＞

```
#include <machine.h>        // 組み込み関数を使用するため追記
void f() {
    if (x == 2) {
        f1();
    }
    f2();
    nop();                  // nop 組み込み関数を追加
    return;
}
```

2. T ビットの不当参照

【現象】

以下のケースにおいて、不当に条件分岐を行う場合がある。

(1) 以下 C プログラムをコンパイルした場合

＜例＞

```
#include <machine.h>
extern void f();
int a;

void func() {
    int b;
    b = (a == 0);      // 比較結果を変数に格納(A)
    f();               // 関数呼び出しまたは組み込み関数 set_cr()
    if (b) {           // (A)の変数を 0/1 比較
        a=1;
    }
}

_func:
        STS.L      PR,@-R15
        MOV.L      L13,R6          ; _a
        MOV.L      @R6,R2
        TST        R2,R2           ; 比較結果を T ビットに格納
        MOV.L      L13+4,R2        ; _f
        JSR        @R2             ; 関数呼び出し先で T ビットが
        NOP                        ; 破壊される可能性あり
        BF         L12             ; T ビットを参照し分岐
        MOV.L      L13,R6          ; _a
        MOV        #1,R2
        MOV.L      R2,@R6
L12:
        LDS.L      @R15+,PR
        RTS
        NOP
```

(2) C++プログラムで、関数内のローカルブロック内で宣言されたデストラクタ呼び出しのあるクラスを記述した場合

【発生条件】

以下の(1)～(3)または(4)～(5)の条件をすべて満たす場合、発生することがあります。

(1) optimize=1 を指定する。

(2) C プログラムで比較結果を変数に格納している。

(3) (2)の変数を関数呼び出しまたは組み込み関数 set_cr()使用後に 0/1 比較している。

または

(4) optimize=1 を指定する。

(5) C++プログラムで関数内のローカルブロック内で宣言されたデストラクタ呼び出しのあるクラスを記述している。

【回避方法】

該当個所が存在するかを確認するチェックツールを配布いたします。

該当個所が存在した場合､以下のいずれかの方法で回避していただきますようお願いします。

＜C プログラムの場合＞

(1) 該当ファイルを optimize=0 を指定してコンパイルする。

(2) 比較結果を格納する変数を volatile 宣言する。

(3) 比較結果を格納する式(上記例(A))を以下(a)または(b)の通り記述する。

(a)
```
b = (a == 0) ? 1 : 0;
```

(b)
```
if (a == 0) {
      b = 1;
  } else {
      b = 0;
  }
```

＜C++プログラムの場合＞

該当ファイルを optimize=0 を指定してコンパイルする。

3. 定数値の不当共通化

【現象】

以下 C ソースを optimize=1 でコンパイル時、定数値を不当に共通化し、参照時の値が実際と異なる場合がある。

＜例＞

```
#define a (*(volatile unsigned short *)0x400)
#define b (*(volatile unsigned short *)0x4000)
#define c (*(volatile unsigned short *)0x402)
int d;

void func() {
   a = 0x8000;        /* (A)   */
   b = 0x8000;        /* (A')  */
   d = c + 0x8000;    /* (B)   */
}
```

```
_func:
        MOV.W     L15,R6          ; H'8000  R6 に 0xFFFF8000 を設定
        MOV       #4,R5
        MOV       #64,R2
        SHLL8     R5
        SHLL8     R2
        MOV.W     R6,@R5          ; (A)  a に 0x8000 を設定
        MOV.W     R6,@R2          ; (A') b に 0x8000 を設定
        MOV.W     @(2,R5),R0
        EXTU.W    R0,R2
        ADD       R6,R2           ; R2 に c+0xFFFF8000 の結果を設定
        MOV       R2,R6
        MOV.L     L15+4,R2        ; _d
        RTS
        MOV.L     R6,@R2          ; (B) d に c+0xFFFF8000 の結果を格納し NG
```

【発生条件】

以下の条件をすべて満たす場合、発生することがあります。

(1) optimize=1 を指定する。

(2) 関数内で 128～255 もしくは 32768～65535 の範囲の同じ定数値を複数回使用している。

(3) (2)の定数値が無符号で、異なるサイズで使用される。

上記例では(A)(A')では 2 バイト、(B)では 4 バイト

【回避方法】

該当個所が存在するかを確認するチェックツールを配布いたします。

該当個所が存在した場合､以下のいずれかの方法で回避していただきますようお願いします。

(1) 該当ファイルを optimize=0 を指定してコンパイルする。

(2) 定数値を初期値ありの 4 バイト外部変数にする。

＜例＞

```
int value = 0x8000;
void func() {
   a = value;      /* (A)   */
   b = value;      /* (A')  */
   d = c + value;  /* (B)   */
}
```

4. リテラルプール出力位置不正

【現象】

align16 オプションを指定してコンパイルした場合、リテラルプールの参照が届かなくなる場合がある。

(1) code=machinecode を指定した場合

goptimize オプションを指定時はリンク時内部エラー、指定しない場合はオブジェクト不正になる場合がある。

(2) code=asmcode を指定した場合

アセンブル時エラーになる場合がある。

＜例＞

```
                :
        MOV.L       L154+2,R2     ; L158    リテラル(A)を参照(1)
        MOV.L       R2,@R15
        MOV.L       L154+6,R2     ; _printf リテラル(B)を参照(2)
        JSR         @R2
        NOP
                :
        .ALIGN      16
L86:
        ADD         #1,R2
        BRA         L153          ; 無条件分岐を生成しリテラルを出力
        MOV.L       R2,@R4
L154:
        .RES.W      1
        .DATA.L     L158          ; (A) (1)から disp が届かない位置にリテラル出力
        .DATA.L     _printf       ; (B) (2)から disp が届かない位置にリテラル出力
        .ALIGN      16
L153:
                :
```

【発生条件】

以下の条件をすべて満たす場合、発生することがあります。

(1) align16 オプションを指定する。

(2) 無条件分岐命令を生成してリテラルプールを出力する。

【回避方法】

以下の方法で回避していただきますようお願いします。

(1) align16 オプションを指定しないでコンパイルする。

5. 遅延スロットへの不当命令移動

【現象】

optimize=1 を指定してコンパイルした場合、不当に遅延スロットに命令を移動し、レジスタを破壊する場合がある。

＜移動前＞

```
            :
    SHLL        R2
    MOV         R2,R0
    MOVA        L88,R0
    BRA         L144
    NOP
            :
```

＜移動後＞

```
            :
    SHLL        R2
                            ; 遅延スロットに移動
    MOVA        L88,R0      ; R0 に値を設定
    BRA         L144
    MOV         R2,R0       ; MOVA で設定した R0 を破壊
            :
```

【発生条件】

以下の条件を満たす場合、発生することがあります。

(1) 同一レジスタに値を設定するコードが連続する。

【回避方法】

該当個所が存在するかを確認するチェックツールを配布いたします。

該当個所が存在した場合、以下の方法で回避していただきますようお願いします。

(1) 該当ファイルを optimize=0 を指定してコンパイルする。

6. GBR 相対論理演算生成時の offset 不正

【現象】

optimize=1 を指定してコンパイルし、かつ 1 バイト構造体メンバの論理演算を GBR 相対で行うコードを生成した場合、その offset 値が不正になる場合がある。

＜例＞

```
struct {
   int a;
   unsigned char b;
} ST;
char c;

void f() {
   ST.b |= 1;
   c &= 1;
}
```

```
_f:
       STC        GBR,@-R15
       MOV        #0,R0          ; H'00000000
       LDC        R0,GBR
       MOV.L      L11+2,R0       ; H'00000008+_ST  <- 実際は(ST+4)
       OR.B       #1,@(R0,GBR)
       MOV.L      L11+6,R0       ; _c
       AND.B      #1,@(R0,GBR)
       RTS
       LDC        @R15+,GBR
```

【発生条件】

以下の条件をすべて満たす場合、発生することがあります。

(1) optimize=1 を指定する。

(2) gbr=user を指定しかつ#pragma gbr_base/gbr_base1 を使用している、または gbr=auto を指定しかつ map オプションを指定していない。

(3) サイズが 1 のメンバを含むグローバル構造体が存在する。

(4) サイズ 1 の構造体メンバは構造体先頭でない。

(5) (3)のメンバが関数内で論理演算で使用されている。

(6) (3)のメンバは論理演算以外では使用されていない。

(7) 関数内に論理演算のみで使用される外部変数が(3)以外に存在する。

【回避方法】

該当個所が存在するかを確認するチェックツールを配布いたします。

該当個所が存在した場合、以下の方法で回避していただきますようお願いします。

(1) 該当ファイルを optimize=0 を指定してコンパイルする。

(2) 該当メンバの参照を論理演算以降に追加する。

＜例＞

```
struct {
   int a;
   unsigned char b;
} ST;
char c;
unsigned char temp;  // 参照用

void f() {
  ST.b |= 1;
  c &= 1;
  temp = ST.b;        // ST.b の参照を追加
}
```

7. SWAP 命令直後の不当ゼロ拡張

【現象】

optimize=1 を指定してコンパイルした場合、組み込み関数 swapb、swapw、end_cnvl の結果代入先にポインタを指定すると、不当にゼロ拡張を行う場合がある。

＜例＞

```
#include <machine.h>
unsigned short *a,*b;

void func() {
   *b=swapb(*a);
}

_func:
       MOV.L      L13+2,R2   ; _a
       MOV.L      L13+6,R5   ; _b
       MOV.L      @R2,R6
       MOV.W      @R6,R2
       SWAP.B     R2,R6
       MOV.L      @R5,R2
       EXTU.B     R6,R6     ; SWAP 結果を不正にゼロ拡張
       RTS
       MOV.W      R6,@R2
```

【発生条件】

以下の条件をすべて満たす場合、発生することがあります。

(1) optimize=1 を指定する。

(2) 組み込み関数 swapb、swapw、end_cnvl を使用している。

(3) (2)の組み込み関数の結果代入先がポインタである。

【回避方法】

以下の方法で回避していただきますようお願いします。

(1) 該当ファイルを optimize=0 を指定してコンパイルする。

(2) 組み込み関数の結果をローカル変数にコピーしてからポインタに代入する。

＜例＞

```
void func() {
   unsigned short temp;
   temp=swapb(*a);
   *b=temp;
}
```

8. ビットフィールドデータ出力不正

【現象】

無名ビットフィールドを含む構造体に初期値を設定した場合、その初期値が不正になる場合がある。

＜例＞

```
struct st {
   short a:4;
   short b;
   short :12;   // 無名ビットフィールド
   short c:4;
} ST={1,1,3};

_ST:
        .DATA.W    H'1000
        .DATA.W    H'0001
        .DATAB.B   1,0          ; 正しくは
        .DATA.W    H'0300       ; .DATA.W H’0003
```

【発生条件】

以下の条件をすべて満たす場合、発生することがあります。

(1) 構造体の中にビットフィールド、無名ビットフィールド、ビットフィールドでないメンバが混在し、以下の順番で定義されている。

```
struct A{
          :
   ビットフィールド
          :
   ビットフィールドでないメンバ
   無名ビットフィールド        // (A)
   ビットフィールド            // (B)
          :
};
```

(2) (A)(B)のメンバの型サイズが 2 バイト以上である。

(3) (A)の無名ビットフィールドのサイズが 8 ビット以上である。

(4) (A)(B)のビットフィールドのサイズの合計が以下の通りである。

(a) (A)(B)の型サイズがともに 2 バイトの場合　　：16 ビット以下

(b) (A)(B)の型サイズがともに 4 バイトの場合　　：32 ビット以下

(5) 構造体の変数が定義時に初期値設定されている。

【回避方法】

該当個所が存在するかを確認するチェックツールを配布いたします。

該当個所が存在した場合、以下の方法で回避していただきますようお願いします。

(1) 無名ビットフィールド全体をダミーとする。

＜例＞

```
struct st {
    short a:4;
    short b;
    short dummy:12;  // 無名ビットフィールド全体をダミーとする。
    short c:4;
} ST = {1,1,0,3};
```

9. ループ展開不正

【現象】

speed オプションまたは loop オプション指定時、ループ展開最適化でループ判定式を不当に置きかえる場合がある。

＜例＞

```
int a[100];
void main(int n) {
   int i;

   for (i=0; i<n; i++) {
      a[i] = 0;
   }
}
```

```
_main:
        MOV        R4,R7
        ADD        #-1,R4     ; R4 が 0x80000000 の場合、アンダフローとなり
                              ; R4 は 0x7FFFFFFF となる
        MOV        R4,R6
        CMP/PL      R4        ; 比較結果が実際の結果と逆になる
        MOV        #0,R4
        BF         L12
        ADD        #-1,R6
               :
```

【発生条件】

以下の条件をすべて満たす場合、発生することがあります。

(1) speed オプションまたは loop オプションを指定している。

(2) 関数内でループ文を使用している。

(3) ループ上限値が以下の通りである。

(a) 制御変数がインクリメントされていくループの場合(i += step)

ループ上限値が 0x80000000～0x80000000+step-1 の範囲である。

(b) 制御変数がデクリメントされていくループの場合(i -= step)

ループ上限値が 0x7FFFFFFF～0x7FFFFFFF-step+1 の範囲である。

上限値が変数の場合、変数に(a)または(b)を満たす値が設定されている場合に動作不正になる。

【回避方法】

以下の方法で回避していただきますようお願いします。

(1) speed オプション、loop オプションを指定しない。

(2) noloop オプションを指定する。

10. 構造体、配列引数の不当参照

【現象】

関数の引数に構造体/共用体または配列を使用した場合、そのメンバや配列要素を参照する時に、不正なアドレスを参照する場合がある。

＜例＞

```
typedef  struct{
   int A[10];
   double B;
   char F[20];
} ST;
extern ST f(ST a,ST b);
ST S;
extern int X;
void func(ST a,ST b) {
   ST t;
   if (a.B!=f(S,t).B){
      X++;
   }
   if (a.B!=b.B){
      X++;
   }
}
```

```
               :
L12:
        MOV        R15,R2
        MOV.W      L15+2,R0   ; H'014C
        ADD        R0,R2
        MOV        R15,R0
        MOV.W      L15+4,R0   ; H'0190  R0 を不正に破壊
        ADD        R0,R0
        MOV.L      @R2,R4
        MOV.L      @(4,R2),R7
        MOV        R0,R2
        MOV.L      @R2,R6     ; b.B でない場所をアクセス
               :
```

【発生条件】

以下の条件をすべて満たす場合、発生することがあります。

(1) 引数が構造体/共用体または配列である関数が存在する。

(2) 該当関数内で引数の構造体/共用体メンバまたは配列要素を参照している。

【回避方法】

該当個所が存在するかを確認するチェックツールを配布いたします。

該当個所が存在した場合、以下の方法で回避していただきますようお願いします。

(1) 構造体/共用体または配列の引数をポインタにする。

＜例＞

```
void func(ST *a,ST *b) {
    ST t;
    if (a->B!=f(S,t).B){
        X++;
    }
    if (a->B!=b->B){
        X++;
    }
}
```

11. JMP 命令の不当削除

【現象】

speed オプションを指定してコンパイルした場合、JMP 命令を不当に削除する場合がある。

＜例＞

```
void f(){
   int i,j=0;
   for (i=0; i<10; i++)
       if (j%2) j++;
   sub();
}

void sub() {
   :
}
```

```
_f:
               :
L20:
        ADD        #-1,R5
        TST        R5,R5
        BF         L11
        MOV.L      L23,R2       ; _sub  以下 3 命令が削除される
        JMP        @R2          ;               |
        NOP                     ;               V
L18:
        MOV        R6,R0
        AND        #1,R0
        BRA        L16
        MOV        R0,R2
L13:
        MOV        R6,R0
        AND        #1,R0
        BRA        L14
        MOV        R0,R2
_sub:
               :
```

【発生条件】

以下の条件をすべて満たす場合、発生することがあります。

(1) speed オプションを指定する。

(2) 該当関数の最後の処理が関数呼び出しである。

(3) (2)の関数呼び出しがインライン展開されない。

(4) (2)の関数呼び出し先が該当関数の直下にある。

(5) 該当関数内でループ文や条件文を使用している。

【回避方法】

該当個所が存在するかを確認するチェックツールを配布いたします。

該当個所が存在した場合、以下の方法で回避していただきますようお願いします。

(1) 該当ファイルを speed オプションを指定しないでコンパイルする。

(2) 関数の配置を変更し、関数呼び出し先が直下にこないようにする。

(3) nop 組み込み関数を関数末尾に追加する。

＜例＞

```
#include <machine.h>    // for nop
void f(){
    int i,j=0;
    for (i=0; i<10; i++)
        if (j%2) j++;
    sub();
    nop();              // 追加
}
```

12. ループ判定式不正

【現象】

以下 C ソースを optimize=1 でコンパイル時、ループ判定式を不当に置きかえる場合がある。

＜例＞

```
void f1()
{
   int  i;

   for (i=-1; i<INT_MAX; i++) {
        a[i] = 0;
   }
}

_f1:
        MOV.L      L13,R2    ; _a
        MOV        #-4,R6    ; H'FFFFFFFC
        MOV        R6,R5
        MOV        #0,R4     ; H'00000000
        ADD        #-4,R2
L11:
        ADD        #4,R6
        MOV.L      R4,@R2
        CMP/GE     R5,R6     ; H’FFFFFFFC と比較
        ADD        #4,R2
        BF         L11
        RTS
        NOP
```

【発生条件】

以下の条件をすべて満たす場合、発生することがあります。

(1) optimize=1 を指定している。

(2) 関数内でループ文を使用している。

(3) ループ上限値 – ループ下限値がオーバフローする。

例えば、for (i=-1; i < 0x7FFFFFFF; i++)

【回避方法】

以下の方法で回避していただきますようお願いします。

(1) optimize=0 を指定する。

(2) ループを分割する。

＜例＞

```
for (i =-1;i<0x7FFFFFFF;i++) {
    func(i);
)
        ↓
func(-1);
for(i=0;i<0x7FFFFFFF;i++) {
    func(i);
}
```