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

2002 年　3 月　27 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-035A		
題名	SuperH RISC engine C/C++コンパイラパッケージ Ver.7.0.02 (PC 版)/Ver.7.0.03 (UNIX 版) リビジョンアップのお知らせ		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++コンパイラパッケージ Ver.7.0.02(PC 版)/Ver.7.0.03(UNIX 版)にリビジョンアップしました。
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。

Windows 版 SuperH RISC engine C/C++コンパイラパッケージ　(型名：P0700CAS7-MWR)
SPARC 版 SuperH RISC engine C/C++コンパイラパッケージ　(型名：P0700CAS7-SLR)
HP9000 版 SuperH RISC engine C/C++コンパイラパッケージ　(型名：P0700CAS7-H7R)

の Ver.7.0B、Ver.7.0.01(PC 版)、Ver.7.0.02(UNIX 版)

添付 ： P0700CAS7-030312J
SuperH RISC engine C/C++ compiler package
Ver.7.0.02(PC 版)/Ver.7.0.03 (UNIX 版)
アップデート内容

SuperH RISC engine C/C++ compiler package
Ver.7.0.02 (PC 版)/Ver.7.0.03 (UNIX 版)
アップデート内容

本パッケージのアップデート内容を以下に示します。
ただし、項番 1 は PC 版のみです。

1. Hitachi Embedded Workshop

1.1 HIM プロジェクトの変換不正

C コンパイラフェーズを含まない HIM プロジェクトを HEW2.0 用に変換した時、C コンパイラの CPU タブ上のデフォルトオプションを正しく設定できない問題を対策しました。

1.2 Denormalize アセンブラオプションのビルド時設定不正

ワークスペースが、アセンブラオプション Denormalize"=ON"と"=OFF"の両プロジェクトを持つ場合、Denormalize"=ON"のプロジェクトをビルド(アセンブル)時、Denormalize"=ON"を設定しない問題を対策しました。

1.3 プロジェクトジェネレータ生成データの修正

以下の CPU の I/O 定義ファイル(iodefine.h)を修正しました。

SH7708 : st_sci 定義
SH7709S : st_scif, st_irda 定義他

1.4 Runtime Library のスタックサイズ表示不正

ユーティリティツールの一つ、スタック解析ツールで Runtime Library のスタックサイズを正しく表示できない問題を対策しました。

2. コンパイラ (Ver.7.0.03 -> Ver.7.0.04)

2.1 副作用のある式を含む if 文の不正削除

if 文の削除最適化で、副作用のある条件式が削除される場合がある不具合を解決しました。

＜例＞

```
extern int sub();
main() {
        int i=0;
        if(sub()) {         /* if 文内の関数 sub()の呼び出しが不正に削除されます        */
              i=10;         /*  ローカル変数 i はこの後どこでも使用されないので        */
        }                   /*  最適化により i=10 が削除されます                       */
}
```

【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) if 文の then 節、else 節とも空文である(最適化により空文になる場合を含む)。

2.2 スタックアクセス不正

スタックに対するロード・ストアが不正になる場合がある不具合を解決しました。

＜例＞

```
          :
MOV         R15,R2
ADD         #72,R2
MOV         #104,R0       ; H'00000068
MOV.L       R2,@(R0,R15)                <- (SP+72)にデータを保存
MOV         R15,R3
ADD         #92,R3                      <- 正しくは#72
MOV.L       @R3,R6                       <- (SP+92)からデータ取り出し NG
MOV.L       L37+60,R4    ; L52
BSR         _func
MOV         #52,R5        ; H'00000034
          :
```

【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) スタックへのアクセスがある。

(2) 1 関数の最適化対象範囲が複数に分割されている
(分割される条件は、サイズ、変数、関数呼び出しの数等に起因します)。

2.3 #pragma interrupt **関数のレジスタ退避・回復不正**

#pragma interrupt 関数から#pragma inline_asm 指定関数を呼び出し時、#pragma inline_asm 関数内で使用したレジスタが破壊されることがある不具合を解決しました。

【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) #pragma interrupt 関数から#pragma inline_asm 指定関数の呼び出しがある

(2) 当該#pragma interrupt 関数内に他の関数呼び出しがない

2.4 MAC **レジスタ退避・回復不正**

macsave=1 オプション指定時、#pragma regsave 指定関数で MAC レジスタが退避・回復されない不具合を解決しました。

【発生条件】

以下の条件をすべて満たす場合、発生します。

(1) macsave=1 を指定する。

(2) #pragma regsave を指定する。

2.5 2 **重ループ内式のオブジェクト不正**

2 重ループのある式で speed オプションまたは loop オプションを指定すると、コード不正になる場合がある不具合を解決しました。

＜例＞

```
        :
for(...)
    for(...)
        x = x + i;      /* x の値が不正になることがある */
        :
```

【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) speed オプションまたは loop オプションを指定する。

(3) 2 重ループがあり、ループの内側に x=x+a;のように左辺と同じ変数を右辺で使用している。

2.6 拡張命令の削除不正

不要なロード・ストア命令削除または複数ロード命令削除で不正に EXTS 命令を削除してしまう場合がある不具合を解決しました。

【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) 代入先(または複数ロード命令の２回目以降)サイズ < 参照元サイズ <= 4byte である。

2.7 switch 文展開不正

switch 文がテーブル方式展開される場合、遅延スロットにケーステーブルのリテラルデータが入り、コード不正になる場合がある不具合を解決しました。

＜例＞

```
            SHLL        R6
            MOVA         L204,R0
            MOV.W        @(R0,R6),R0
            BRAF         R0
            NOP
    L203:
            .RES.W      1
            .DATA.L     _fl
            .DATA.L     _f2
            .DATA.L     _f3
            .DATA.L     _f4
            .DATA.L     _f5
            .DATA.L     _f6
    L204:
            .DATA.W     L143-L203
            .DATA.W     L143-L203
            BRA         L180
            .DATA.W     L143-L203       ←遅延スロットに入る
```

【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) switch 文があり、分岐方式がテーブル展開である。

2.8 定数演算不正

定数演算でコード不正になる場合がある不具合を確認しました。

＜例＞

```
void f() {
        sl = 0x80000001l;
                :
        if (*scp16==(char)(0x80000001L))   /* 0x80000001 を 1byte に型変換した結果(1)と比較 */
                :                          /*  すべきところ、0x80000001 と比較              */
}
```

【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) 同一関数内に、サイズの異なる部分的に等しい定数式がある。

2.9 ポインタ指示先への設定不正

ポインタの指示先への値の設定が正しく行われない場合がある不具合を解決しました。

＜例＞

```
int *gp;
void g(int *p) {
        gp = p;
}
int f(int *q) {
        static int i, *p = &i;
        g(p);              /* gp が i を指すように設定される */
              :
        i = 1;
        *gp = 2;         /* *gp と i が同一である可能性があることが認識できていない */
              :
        return i;       /* i = 1 が定数伝播され、2 ではなく 1 が返される */
}
```

【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) 静的変数及び外部変数の初期値にアドレス参照式を指定、または&演算子を用いない配列のアドレス参照がある

2.10 #pragma interrupt 関数の FPSCR 退避・回復不正

#pragma interrupt 指定関数で FPSCR が退避・回復されない不具合を解決しました。

【発生条件】

以下の条件をすべて満たす場合、発生します。

(1) cpu=sh2e/sh4 を指定する。

(2) #pragma interrupt 関数内に関数呼び出しがない。

(3) #pragma interrupt 関数内にオペランドに FPSCR のあるコードが無い。

(4) #pragma interrupt 関数内に FPU 演算(FABS,FSQRT,FNEG,FCNVDS,FCNVSD,FLOAT, FTRC,FADD,FSUB,FMUL,FDIV,FCMP/EQ,FCMP/GT,FMAC)がある。

2.11 組込み関数の移動不正

以下の組み込み関数を跨いで命令が移動することがある不具合を解決しました。

set_cr, get_cr, set_imask, get_imask, set_vbr, get_vbr, trapa, trapa_svc, prefetch

【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) 以下の組み込み関数を使用。

set_cr, get_cr, set_imask, get_imask, set_vbr, get_vbr, trapa, trapa_svc, prefetch

2.12 乗算結果の拡張命令削除不正

乗算結果を 1/2byte に代入時、不正に拡張命令が削除される場合がある不具合を解決しました。

＜例＞

```
unsigned short eus=32768,eus2=32769;
void func () {
      unsigned short aus;
      aus= (eus--) * (eus2--);          /*  乗算結果のゼロ拡張が不正に削除          */
      if (aus!=(unsigned short)((1+eus)*(1+eus2))){
            :                           /* (1+eus)*(1+eus2)の結果はゼロ拡張される  */
}
```

【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) 乗算結果を 1byte または 2byte に代入する。

2.13 volatile へのキャスト不正

volatile へのポインタに型変換を指定時、ループ内の式の volatile 指定が無効になる場合がある不具合を解決しました。

＜例＞

```
#define ADDRESS1 0xf0000000
#define ADDRESS2 0xf0000004
#define BIT0        0x0001
void shc_test(void) {
    unsigned short dataW,read_val;
    read_val = *((unsigned short*)ADDRESS1);
    while(1) {
        /* volatile が無効になりループ外に移動 */
        dataW = *((volatile unsigned short*)ADDRESS2);

        if(dataW & BIT0) {
            /* volatile が無効になりループ外に移動 */
            *((volatile unsigned short*)ADDRESS2) &= ~BIT0;
            break;
        }
    }
}
```

【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) 指示先が volatile であるポインタキャストが存在する。

(3) そのポインタの間接参照式がループ内で使用されている。

2.14 GBR **相対論理演算化不正**

外部変数の複合論理演算指定時に、コード不正になる場合がある不具合を解決しました。

＜例＞

```
unsigned char guc1=255;
void func001() {
    if (guc1 & 254) {        /* guc1 をロード(A)                                    */
        guc1 &= 253;         /* AND.B #253,@(R0,GBR)  メモリ上の guc1 を書き換え    */
    }
    if (guc1==253) {         /* guc1 を再ロードせず、(A)をそのまま使用              */
        ok(1);
    } else {
        ng(1);
    }
}
```

【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) gbr=auto を指定している、
または gbr=user 指定時#pragma global_base/#pragma global_base1 を使用している。

(3) 外部変数を複合論理演算式(&=, |=, ^=)で使用している。

(4) 複合論理演算式を「op.B #xx,@(R0,GBR)」(op:AND/OR/XOR)でコード生成している。

2.15 #pragma global_register **指定時の拡張命令削除不正**

1/2byte 変数を#pragma global_register 指定すると、不正に EXTS/EXTU 命令が削除される場合がある不具合を解決しました。

【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) (2) (unsigned) char/short 型変数を#pragma global_register に指定している。

2.16 構造体・共用体配列メンバを含む構造体・共用体使用時のオブジェクト不正

構造体・共用体に構造体・共用体配列メンバがある場合にオブジェクト不正になる場合がある不具合を解決しました。

＜例＞

```
void func() {
    struct tag {
        union {
            short    u11;
            char     u12[2];
        } u1[2];
    } st= {{{0x1234},{0x5678}}};
    char ans1;
    ans1=st.u1[1].u12[1];        /* st.u1[1].u12[1]設定前に ans1 をコピー(A)    */
    printf("%x¥n",ans1);        /* (A)の ans1 をそのまま使用                   */
    if(ans1 == 0x78)             /* (A)の ans1 をそのまま使用                   */
            :
}
```

【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) 構造体・共用体に構造体・共用体配列メンバが存在する。

(3) 上記の領域を配列メンバとそれ以外の直接参照式(構造体全体を表す式や共用体の別メンバ)の両方で参照している。

2.17 goptimize 指定時の optlnk 最適化不正

goptimize 指定時に、レジスタ退避・回復コードが遅延スロットに移動した場合、optlnk で不正に退避・回復コードを削除する場合がある不具合を解決しました。

【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) goptimize を指定する。

(3) optlnk で optimize=register を指定する。

(4) 遅延スロットにレジスタ退避・回復コードがある
(MOV.L @R15+,Rn/MOV.L Rn,@-R15 など)。

2.18 組込み関数 nop の遅延スロット移動抑止

組込み関数 nop を引用した場合、不当に遅延スロットに NOP 命令を移動し指定位置に設定できない不具合を解決しました。

【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) 組込み関数 nop を使用する。

2.19 符号付き整数型定数値のキャスト不正

定数値を char または short 型に明示的型変換指定をした場合、コード不正になる場合がある不具合を解決しました。

【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) 定数値を符号付きの char, short 型に型変換している。

(3) その定数値は型変換後、負の値となる。

2.20 副作用のある実引数の演算不正

関数パラメタに volatile 変数に対する後置++/--式を記述した時、呼び出し関数内で当該変数が後置++/--前の値になる場合がある不具合を解決しました。

【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) 関数のパラメタが volatile で変数である。

(3) 当該パラメタが副作用(後置++/--)式である。

2.21 rtnext 指定時の返却値の符号/ゼロ拡張不正

rtnext オプション指定時、1/2byte のリターン値が不正になる場合がある不具合を解決しました。

＜例＞

```
int a;
unsigned char func(){
    return a;          /* a の上位 3byte は不定値が返る */
}
```

【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) rtnext を指定している。
(2) 関数のリターン値が 1byte または 2byte である。
(3) 関数の原型宣言を行っていない。

2.22 無名ビットフィールドデータ配置不正

endian=little 指定時、構造体メンバに無名ビットフィールドがあるとデータ配置が不正となる場合がある不具合を解決しました。

【発生条件】

以下の(1)-(4)もしくは(5)-(8)の条件をすべて満たす場合、発生することがあります。

(1) endian=little を指定する。
(2) 構造体メンバにビット幅が 8 以上の無名ビットフィールドがある。
(3) その無名ビットフィールドは構造体先頭ではない。
(4) 構造体変数の定義で初期値が設定されている。

(5) endian=little を指定する。
(6) 構造体に配列メンバがあり、その次がビットフィールドメンバである。
(7) 配列メンバとビットフィールドメンバの間に境界調整のギャップが入る。
(8) 構造体変数の定義で初期値が設定されている。

2.23 volatile 指定変数参照削除の抑止

volatile 宣言された変数について以下のような式を記述した場合、変数の参照を保証するように変更しました。

```
a &= 0;
```

2.24 デバッグ情報不正

ローカル配列、構造体・共用体(スタックに領域を確保した時)のデバッグ情報が出力されなかった不具合を解決しました。

2.25 インターナルエラー

以下の場合、インターナルエラーが発生することがある不具合を解決しました。

(1) code=machinecode 指定時に、コンパイル実行するドライブのルートディレクトリにファイルを生成する権限がない場合(PC 版のみ)

(2) 無限ループがあり関数出口に到達する制御フローのないプログラムをコンパイルした場合。

3. 最適化リンケージエディタ (Ver.7.1.02 -> Ver7.1.04)

3.1 compress オプション指定時のデバッグ情報不正

compress オプション指定によりデバッグ情報を圧縮した際、型情報の参照先が不正となる不具合を解決しました。

3.2 外部変数アクセス最適化に関するアドレスチェック

コンパイラの map オプション指定(外部変数アクセス最適化)時のシンボルアドレスについて、リンク時に map オプションを指定したときのみチェックしていたのを、常にアドレスチェックを行うように修正しました。

3.3 binary ファイル入力指定時のセクション属性不正

下記条件全てを満たした際に、セクションの属性が不正となる不具合を解決しました。

(1) object ファイルと binary ファイルを入力

(2) object ファイル内でサイズ 0 のセクションを定義

(3) binary オプションでサイズ 0 のセクションを指定

(4) ファイルの入力指定順：(2)の object ファイル -> (3)の binary ファイル

3.4 relocate **ファイル生成時の異常終了**

下記条件全てを満たした際に異常終了する不具合を解決しました。

(1) 入力 object ファイルの先頭：goptimize オプション指定ありのファイル
入力 object ファイルの 2 番目以降の入力：goptimize オプション指定なしのファイル
もしくはアセンブラ出力ファイル

(2) 出力形式に relocate ファイルを指定

(3) profile オプション指定あり

(4) 最適化指定あり

3.5 **未参照シンボル削除とレジスタ最適化実施時の内部エラー**(1703)

下記条件全てを満たした際に内部エラーが生じる不具合を解決しました。

(1) goptimize オプション指定ありの object ファイルを入力

(2) (1)の object ファイル内に分岐幅が境界(-4096 または 4094)の BSR、または BRA 命令が存在

(3) 未参照シンボル削除最適化、およびレジスタ最適化を指定

3.6 **複数関数が同一リテラルを指す場合のリテラル参照不正**

下記条件全てを満たした際に参照するリテラルの値が不正となる不具合を解決しました。

(1) goptimize オプション指定ありの object ファイルを入力

(2) (1)の object ファイル内で、複数関数が同一リテラルを参照

(3) レジスタ退避/回復コード最適化を指定

以上