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

発行日：2004 年 06 月 07 日

RENESAS TECHNICAL UPDATE

〒100-0004　東京都千代田区大手町 2-6-2 日本ビル
株式会社 ルネサス テクノロジ
問合せ窓口　E-mail: csc@renesas.com

製品分類	開発環境	発行番号	TN-CSX-075A/JA	Rev.	第 1 版
題名	H8S, H8/300 シリーズ C/C++コンパイラ Ver.4.0 不具合のご連絡		情報分類	使用上の注意事項	
適用製品	PS008CAS4-MWR PS008CAS4-SLR PS008CAS4-H7R PS008CAS5-MWR	対象ロット等 Ver.4.0 以降	関連資料	H8S, H8/300 シリーズ C/C++コンパイラ、アセンブラ、最適化リンケージエディタ ユーザーズマニュアル RJJ10B0049-0100H Rev.1.00	

H8S,H8/300 Series C/C++コンパイラ Ver.4.0 以降には別紙に示す不具合があります。

詳しい修正内容については、添付資料の PS008CAS4-040601J をご参照ください。

添付：PS008CAS4-040601J

H8S,H8/300 シリーズ C/C++コンパイラ Ver.4.0 不具合内容

H8S,H8/300 シリーズ C/C++コンパイラ Ver.4.0
不具合内容

H8S,H8/300 シリーズ C/C++コンパイラ Ver.4.0 における不具合の内容を以下に示します。

1)ビットフィールドの設定・参照不正

ビットフィールドへの設定および参照を行った場合、値の設定や参照が正しく行えない場合があります。

[例]

```
typedef struct {
    char c:8;
    char c2:8;
    int i;
}ST;

main()
{
    ST a;
    a.c=10;
    if (a.c==10) {                    /* a.c を不正参照し、比較が FALSE になる                    */
        func1();
    } else {
        func2();
    }
}
```

[発生条件]

以下の a),b)いずれかの条件を全て満たす場合、発生することがあります。

a)

i. CPU 種別として 300HN, 300HA のいずれかを指定している。
ii. 最適化あり(optimize=1:デフォルト)でコンパイルしている。
iii. 引数または局所変数の構造体を宣言している。
iv. iii.の構造体が[unsigned] char 型で、サイズが 8bit のビットフィールドメンバを持つ。
v. iii.の構造体はレジスタ上に割りつき、iv.のビットフィールドメンバは En 上に割りついている。

b)

i. CPU 種別として 300HN, 300HA, 2000N, 2000A, 2600N, 2600A のいずれかを指定している。
ii. 最適化あり(optimize=1:デフォルト)でコンパイルしている。
iii. 引数または局所変数の構造体を宣言している。
iv. iii.の構造体が[unsigned] short または[unsigned] int で、サイズが 16bit のビットフィールドメンバを持つ。
v. iii.の構造体は、境界調整数が 1 である(pack=1 オプション、#pragma pack 1 が指定されている)。
vi. iii.の構造体はレジスタ上に割りつき、iv.のビットフィールドメンバは En の下位 8bit と RnH に割りついている。

[回避方法]

下記のいずれかの方法で回避することができます。

a) ビットフィールドの指定をやめ、スカラ型として宣言する。

```
struct ST {                          struct ST {
    char c:8;                            char c;
    char c2:8              →             char c2;
    int i;                               int i;
};                                   };
```

b) 最適化なしでコンパイルする。

[対象バージョン]

・Ver.4.0 以降

2) &構造体.配列[0] 等構造体メンバのアドレス参照時エラー

&構造体.配列[0](&構造体->配列[0])形式で先頭アドレスを参照した場合、正しいアドレスが求められない場合があります。
または内部エラーが出力される場合があります。

[例]

```
typedef struct ST {
      char array[12];
}ST;

ST st;
int b,c;
char *p;

void sub()
{
    p= &st.array[0] + b + c;                    /* アドレスではなく、st.array[0]の値を加算するコードを出力    */
}
```

[発生条件]

以下の a),b)いずれかの条件を全て満たす場合、発生することがあります。

a)

i. CPU 種別として 300HN, 300HA, 2000N, 2000A, 2600N, 2600A のいずれかを指定している。
ii. 構造体メンバが配列で定義されている。
iii. ii.の先頭のアドレス値を用いて 2 回以上の加減算を実施する。
iv. 先頭アドレスは&構造体.配列[0]、または&構造体->配列[0]形式で求めている。

b)

i. CPU 種別として 300HN, 300HA, 2000N, 2000A, 2600N, 2600A のいずれかを指定している。
ii. 最適化あり(optimize=1:デフォルト)でコンパイルする。
iii. 変数の配列を定義している。
iv. iii.の先頭のアドレス値を用いて、2 回以上の加減算を実施する。
v. 先頭アドレスを&配列[0]形式で求めている。

[回避方法]

下記の方法で回避することができます。

・配列のアドレスを、構造体.配列(または構造体->配列)形式または、配列で求める。

[対象バージョン]

・Ver.4.0 以降

3) &=0、|=0xFFFF の不正アドレスアクセス

[unsigned]short/int型の変数に対して複合論理演算を実施した場合、不正なアドレス(本来のアドレス+2)に値を設定するコードが生成される場合があります。

[例]

```
typedef struct {
    short       w1;
}*PST;

volatile PST pst = (PST)0xC40000;
short * volatile p;

void sub(void)
{
    pst->w1 &= 0;
   *p &= 0;
    pst->w1 |= 0xffff;
   *p |= 0xffff;
}
```

［不正コード］

```
_sub:
        MOV.L       @_pst:32,ER0
        SUB.W       R1,R1
        MOV.W       R1,@(2:16,ER0)
        MOV.L       @_p:32,ER0
        MOV.W       R1,@(2:16,ER0)
        MOV.L       @_pst:32,ER0
        MOV.W       #-1,R1
        MOV.W       R1,@(2:16,ER0)
        MOV.L       @_p:32,ER0
        MOV.W       R1,@(2:16,ER0)
        RTS
```

[正常コード]

```
→   MOV.W   R1,@ER0
→   MOV.W   R1,@ER0
→   MOV.W   R1,@ER0
→   MOV.W   R1,@ER0
```

[発生条件]

下記条件を全て満たす場合、発生することがあります。

a) CPU種別として 300, 300HN, 300HA, 2000N, 2000A, 2600N, 2600Aのいずれかを指定している。
b) 変数を[unsigned] short/int型で宣言している。
c) 変数は volatile 宣言されたポインタである。
d) 下記の複合代入演算を記述している。
- 変数 &=0;
- 変数 |= 0xffff;

[回避方法]

下記のいずれかの方法で回避することができます。

a) ポインタの指す先の領域に volatile を付加する。

```
typedef volatile struct {                   /* volatile を付加する。                    */
    short   w1;
}*PST;

volatile PST pst = (PST)0xC40000;
volatile short * volatile p;                /* volatile を付加する。                    */
```

b) &=,|=演算を単純代入(=)に変更する。

```
pst->w1 = 0;

*p = 0;
pst->w1 = 0xffff;
*p = 0xffff;
```

[対象バージョン]

・Ver.3.0 以降

4) 0 クリアコード不正削除

分岐ごとに 0 を設定するコード(SUB 命令)を記述した場合、0 を設定するコードを不正に削除してオブジェクト不正となることがあります。

[例]

```
      sub.b     R0H, R0H              ; R0H に 0 を設定
           :
           :
    L55:
           :
      bls       L36
      sub.b     R0H, R0H              ; R0H が 0 のままなので削除可能な命令となる。
      add.w     R0, R0                ; ここで R0H の値が変わる可能性がある。
           :
    L48:
      sub.b     R0H, R0H              ; 不正に削除される
      add.w     R0, R0                ; R0H の値が 0 でない値の場合、オブジェクト不正となる。
```

[発生条件]

下記条件を全て満たす場合、発生することがあります。

a) CPU 種別として H8/300, H8/300L のいずれかを指定している。
b) 最適化あり(optimize=1:デフォルト)を指定している。
c) 分岐毎に 0 クリアを行うコードを記述している。

[回避方法]

下記の方法で回避することができます。

a) 最適化なし(optimize=0)でコンパイルする。

[対象バージョン]

・Ver.4.0.04 以降