RENESAS ツールニュース

RENESAS TOOL NEWS 2005年03月16日：RSO-H8C-050316D

H8, H8S, および H8SXファミリ C/C++コンパイラパッケージ V.6.01 Release 00へのリビジョンアップのお知らせ

H8,H8S および H8SX ファミリ用C/C++コンパイラパッケージをV.6.00 Release 03からV.6.01 Release 00にリビジョンアップしました。

1. **該当製品**

   Renesas C/C++ Compiler Package for H8, H8S and H8SX family
   製品型名：

   Windows版 ： R0C40008XSW06R

   Solaris版 ： R0C40008XSS06R

   HP-UX版 ： R0C40008XSH06R

2. **リビジョンアップ内容**

   2.1 新機能および機能改善

   2.1.1 High-performance Embedded Workshop（Windows版のみ）をV.4.00.00へバージョンアップしました。

   詳細は、2005年1月26日発行RENESAS TOOL NEWS "High-performance Embedded WorkshopV.4.00.00 へのバージョンアップのお知らせ"（RSO-HEW-050126D）を参照ください。

   また、High-performance Embedded Workshopのオートアップデート機能をサポートしました。 オートアップデート機能の概要については、2005年3月1日発行の RENESAS TOOL NEWS "統合開発環境High-performance Embedded Workshop でのオートアップデート機能サポートのお知らせ"を参照してください。

   2.1.2 シミュレータ・デバッガ（Windows版のみ）

   (1) H8SXシミュレータ・デバッガでSYSCRレジスタのFETCHMD ビットをサポートしました。 実CPUと同様にFETCHMDビット

により、フェッチサイズを16ビットまたは32ビットから選択できます。

(2) H8Sシリーズシミュレータ・デバッガでCPUのタイマ機能をサポートしました。

(3) H8S/2000シリーズ用シミュレータ・デバッガで割り込みモードをサポートしました。

(4) プログラムをダウンロードする際にメモリリソースを自動で確保する機能を追加しました。

(5) メモリアクセスエラー発生時に、メモリアクセスエラーが発生したアドレスを表示する機能を追加しました。

2.1.3 コンパイラ

(1) AE-5シリーズCPUをサポートしました。

(2) CPU種別がH8Sの時に、H8SXと同様の最適化処理でコードを生成するようにしました。

(3) 以下の項目をANSIに準拠して解釈するようにしました。

1. 配列のインデックス

例：

```
------------------------------------------------------------
----
int iarray[10], i=3;
i[iarray] = 0;  /* iarray[i] = 0;と同じになります */
     /* 従来は、C2200 (E)およびC2233 (E)のエラーを出力
*/
------------------------------------------------------------
----
```

2. unionのビットフィールド指定

例：

```
------------------------------------------------------------
----
union u {
  int a:3;  /* 従来は、C2140 (E)のエラーを出力 */
};
------------------------------------------------------------
----
```

3. 定数演算

例：

```
------------------------------------------------------------
----
static int i=1||2/0;  /* 従来は、C2501 (E)のエラーを出力
*/
------------------------------------------------------------
----
```

4. ライブラリ関数およびマクロ追加
   ANSIに定義されているライブラリ関数strtoul、およびマクロFOPEN_MAXを追加しました。

(4) 以下のオプションを追加しました。

1. strict_ansiオプション：浮動小数点演算の結合則をANSIに準拠して解釈します。
   本オプション指定することにより、演算結果がVer.6.0までと異なる場合があります。
2. enable_registerオプション
   register記憶クラスを指定した変数を、優先的にレジスタに割り付けてコードを生成します。
3. legacy=v4オプション
   CPU種別がH8Sの時に、従来(Ver.6.00以前)と同じ最適化処理でコードを生成します。

(5) 変数を絶対アドレスに配置する拡張機能 #pragma addressを追加しました。

(6) 以下の制限値を緩和しました。(CPU種別がH8SXまたはH8Sで legacy=v4オプション指定無しの場合)

1. 繰り返し文(while文、do文、またはfor文)と選択文(if文またはswitch文)の組み合わせによるネストの深さを32レベルから4096レベルへ。
2. 1関数内で指定可能なgotoラベルの数を511個 から 2147483646個へ。
3. switch文のネストの深さを256レベル から 2048レベルへ。
4. 1つのswitch文内で指定可能なcaseラベルの数を511個 から 2147483646個へ。
5. 関数定義または関数呼び出しで指定可能な引数を63個 から 2147483646個へ。

2.1.4 アセンブラ

(1) AE-5シリーズCPUをサポートしました。

(2) .STACK制御命令をサポートしました。
アセンブラソース内に書かれた.STACKで定義したスタックサイズをスタック解析ツールが自動的に読み込みます。

(3) DEFINEオプションおよび.define制御命令の置換シンボルの文字数制限を32文字から無制限に変更しました。

(4) リスティングファイルに出力するソース行のすべてを改行して表示するようにしました。

2.1.5 最適化リンケージエディタ

次の機能を追加しました。

(1) binaryオプション入力セクションへの境界調整数指定機能
binaryオプションに指定するセクションに対して、境界調整数を指定することができます。

(2) クロスリファレンス情報出力機能
show=xreferenceオプション指定により、クロスリファレンス情報をリンケージリスト内に出力します。これにより、変数または関数がどこから参照されているのかを知ることができます。

(3) 参照されないシンボルの通知機能
msg_unusedオプション指定により、最適化を使用しない状況でも参照されないシンボルの存在を知ることができます。

2.2 改修内容

2.2.1 High-performance Embedded Workshop（Windows版のみ）

CPUシリーズをH8S/2600、H8S/2000、またはH8/300Hのいずれかとし、動作モードをNormalとしてプロジェクト生成した標準入出力用のファイルlowlvl.srcを使用すると、シミュレーション実行時にI/Oシミュレーションエラー(エラーメッセージ：System Call Error)の原因となる可能性のある箇所を改修しました。
コンパイラパッケージリビジョンアップ以前に同じ設定で生成したlowlvl.srcを含むプログラムをビルド後、シミュレーション実行時に"System Call Error"メッセージが出力された場合は、新規プロジェクトの作成により再度lowlvl.srcを生成してください。

2.2.2 コンパイラ

以降14点の問題を改修しました。

(1) int型のオーバフローを伴う演算(H8C-0004)
int型の変数においてオーバフローを伴う演算において、演算結果が異なる場合がある。
該当バージョン：V6.00 Release 00 ～ 03
発生条件：以下の条件をすべて満たした場合に、発生することがあります。

1. CPUオプションに、H8SXAまたはH8SXX(-cpu=h8sxaまたはh8sxx)を選択している。
2. 最適化オプション(-optimize=1)を選択している。
3. intまたはshort型の変数を加算し、その結果に対して

unsigned long型で乗算している式がある。

4. 3.の加算は、オーバフローを伴う演算である。

例:

```
------------------------------------------------------
-------
signed int sub(){
   return 32767;
}
unsigned long ul1,ul2;
main(){
   signed int i1,i2;
   i1 = sub();
   ul1 = 2;
   i2 = 1;
   ul2 = ul1 * (i1 + i2); // ul2=2*(unsigned
long)(32767+1)
}
------------------------------------------------------
-------
```

(2) 構造体配列サイズが32767バイトを超えたときのメンバアクセス(H8C-0005)

32767バイトを超える構造体型の配列をアクセスした時、不正なデータを
クセスする場合がある。
該当バージョン：V6.00 Release 00 ～ 03
発生条件：以下の条件をすべて満たした場合に、発生することがあります。

- CPUオプションに、H8SXN, H8SXM, H8SXAまたはH8SXX(-cpu=h8sxn, h8sxm, h8sxa, またはh8sxx)を選択している。
- 構造体のサイズが、2バイトまたは4バイトである。
- 条件2の構造体の配列のサイズが、32767バイトを超えている。

例:

```
------------------------------------------------------
-------
struct st2{
   char a;
   char b;
}st_2[32767];
void main(void){
```

```
    char i;
    for(i = 0 ; i < 10 ; i++){
        st_2[i].b = i;
            // 構造体内の指定領域とは異なる領域に値を設定
    }
}
------------------------------------------------------
-------
```

(3) ループ文内での配列アクセス(H8C-0006)

ループ文内で配列をアクセスすると、不正な要素をアクセスする場合がある。

該当バージョン: V4.0 ～ V5.0.06

発生条件：以下の1, 2, 4, および5 または 1, 3, 4, および6のすべての条件を満たした場合に、発生することがあります。

1. CPUオプションに、300HA, 2000A, 2600A, H8SXXまたはH8SXA(-cpu=300ha, 2000a, 2600a, h8sxx, またはh8sxa)を選択している。
2. 300HA, 2000A, または2600Aの場合：最適化(-optimize=1)オプション、スピードオプション(-speed, -speed=loop[1|2])を選択している。
3. H8SXXまたはH8SXAの場合：ポインタサイズ指定オプション(-ptr16)を選択している。
4. unsigned shortまたはunsigned int型の変数を宣言している。
5. 300HA, 2000A, または2600Aの場合：条件4の変数を以下のように使用している。
    - ループ変数の初期化時、定数で初期化している。
    - ループ内のループ変数として使用している。
    - 配列の添え字として（定数-変数）の形で使用している。
6. H8SXXまたはH8SXAの場合：条件4の変数を配列の添え字として(定数-変数)の形で使用している。または、(定数+<式>)の<式>が負数である。

例1:

```
------------------------------------------------------
-------
extern unsigned char a[20];
void sub(void){
    unsigned short j;
    for(j=0; j<10; j++){
        a[20-j] = 10; //配列領域外を参照
```

```
  }
}
------------------------------------------------------
-------
例2:
------------------------------------------------------
-------
unsigned short a[20];
unsigned short j;
void sub3(void){
  a[20-j] = 10;
}
------------------------------------------------------
-------
```

(4) switch文のテーブル展開コード(H8C-0007)
テーブル方式で展開されたswitch文の中と外で、同じ変数に同一値を設定すると、その変数の設定値が不定値になる場合がある。
該当バージョン：V6.00 Release 00 ～ 03
発生条件：以下の条件をすべて満たした場合に発生することがあります。

1. CPUオプションに、H8SXAまたはH8SXX(-cpu=h8sxa,h8sxx)を選択している。
2. switch文展開方式オプションで、自動(-case=auto)でテーブル方式に展開されているコードが存在する。またはテーブル展開方式(-case=table)を選択している。
3. 設定する変数がchar,unsigned char,short,unsigned short,intまたはunsigned intの型の配列である。
4. 同じ値の定数が複数箇所で使用されており、switch文の前とswitch文の中で使用されている。

(5) 可変個の引数を持つ関数の引数割り付け (H8C-0008)
構造体パラメタのレジスタ割り付けオプション(-structreg)を指定し、 4バイト以下の構造体を引数に指定すると、スタックで渡されるべき引数 がレジスタ渡しとなる場合がある。
該当バージョン：V6.00 Release 00 ～ 03
発生条件：以下の条件をすべて満たした場合に発生することがあります。

1. CPUオプションに、H8SXN, H8SXM, H8SXAまたはH8SXX(-cpu=h8sxn,h8sxm, h8sxa, またはh8sxx)を選択している。
2. 構造体パラメタのレジスタ割り付けオプション(-structreg)が選択されている。

3. 可変個の引数を持つ関数が存在し、4バイト以下の構造体変数が引数にある。
4. 条件3の構造体はスタックを介して関数呼び出しが実行される。
5. 条件3の構造体を格納できる引数割り付け用のレジスタに空きがある。

例:

```
-------------------------------------------------------
-------
struct A{
   int is_keyword ;
} flags;
void sub(const char *,...);
void main(void){
   sub("test", flags); // flagsの内容をレジスタにコピー
}
-------------------------------------------------------
-------
```

(6) switch文のtable展開時のセクション名(H8C-0009)

短絶対アドレスオプションが選択され、table方式に展開されるswitch文 が存在していると、セクション名が不正になる場合がある。

該当バージョン: V4.0 ～ V5.0.06

発生条件：以下の条件をすべて満たした場合に発生することがあります。

1. CPUオプションに、300HN, 2000Nまたは2600N(-cpu=300hn, 2000n, または2600n)を選択している。
2. 短絶対アドレスオプション(-abs16)を選択している。
3. switch文展開方式オプションで、自動(-case=auto)でテーブル方式に展開されているコードが存在する。またはテーブル展開方式(-case=table)を選択している。
4. オブジェクト形式オプションで、機械語出力(-code=machinecode)を選択している。

例:

```
-------------------------------------------------------
-------
char c;
void func(void){
   switch (c) {
      case 0:
         c-=2;
```

```
            break;
        case 1:
            c--;
            break;
        case 2:
            c++;
            break;
        case 3:
            c+=2;
            break;
        case 4:
            c+=3;
            break;
        case 5:
            c+=4;
            break;
    }
}
-------------------------------------------------------
-------
```

(7) speed優先時の乗算の演算(H8C-0010)

long型変数とint型変数との乗算を行うと、不正なオブジェクトが 生成される、または内部エラーが出力される場合がある。

該当バージョン: V4.0 ～ V5.0.06

発生条件：以下の条件をすべて満たした場合に発生することがあります。

1. CPUオプションに、300HN, 300HA, 2000N, 2000A, 2600N, または2600A (-cpu=300hn, 300ha, 2000n, 2000a, 2600n, または2600a)を選択している。
2. スピード優先最適化オプション(-speed)またはスピード優先最適化 の四則演算、比較、代入式の高速化オプション(-speed=expression)を選択している。
3. long型変数とshort型またはint型との乗算を行なっている。
4. 乗算の1項および2項が、いずれも関数呼び出しまたは演算式である。

例:

```
-------------------------------------------------------
-------
long  l;
```

```
short func_s(void);
long  func_l(void);
void func(void){
 l = func_s() * func_l();
    // short型とlong型の乗算
}
```

---

(8) __asm{}内でのDATA制御命令(H8C-0011)

__asm{}内で.DATA.B/.DATA.Wで変数を定義する際、記述した、DATA制御 命令のサイズよりも大きい値を記述すると、設定される値が不正となる場合がある。

該当バージョン：V6.00 Release 00 ～ 03

発生条件：以下の条件をすべて満たした場合に発生することがあります。

1. 埋め込みアセンブリ機能(__asm)内で、.DATA制御命令を記述している。
2. 1.の.DATA制御命令は、.DATA.B/.DATA.Wで宣言されている。
3. 1.の.DATA制御命令の整数データが、宣言した型のサイズを超えている。

例:

```
void func(void){
   __asm{
      L1: .data.w "0x12345678"
         //  実際には0xffffまでしか表現できません。
      L2: .data.b "0x1234"
         //  実際には0xffまでしか表現できません。
   }
}
```

---

(9) オーバフロー判定の組み込み関数使用時のアクセス(H8C-0012)

組み込み関数ovfaddまたはovfsubを使用すると、不当にオーバフローと出力される場合がある。

該当バージョン：V6.00 Release 00 ～ 03

発生条件：以下の条件をすべて満たした場合に発生すること

があります。

1. 組み込み関数ovfaddまたはovfsubを使用している。
2. 以下の演算を実施している。
   ovfadd：正の最大値 + 負の最大値
   ovfsub：正の最大値 - 負の最大値
   注: 正の最大値 = 2147483647(0x7FFFFFFF)
   負の最大値 = -2147483648(0x80000000)

例:

```
------------------------------------------------------
-------
#include<stdio.h>
#include<machine.h>
void main(void){
   if (!ovfaddl(0x7fffffff,0x80000000,0)){
      printf("OK¥n");
   } else {
      printf("NG¥n");
   }
}
------------------------------------------------------
-------
```

(10) 構造体型へのポインタ代入(H8C-0013)

メンバの属する構造体型へのポインタをメンバに代入すると、不正な データをアクセスする場合がある。
該当バージョン：V6.00 Release 00 ～ 03
発生条件：以下の条件をすべて満たした場合に、発生することがあります。

1. CPUオプションに、H8SXN,H8SXM,H8SXAまたはH8SXX(-cpu=h8sxn,h8sxm,h8sxa,h8sxx)を選択している。
2. 最適化オプション(-optimize=1)を選択している。
3. 構造体のメンバの型が、そのメンバの属している構造体型へのポインタである。
4. 条件3のメンバに、そのメンバが属している構造体のアドレスを代入している使用している。

例:

```
------------------------------------------------------
------
struct data {
  struct data *p ;
};
```

```
struct data *P1, *P2 ;
void fnc( void ){
  P1->p = P2->p = P2 ;
}
```

---

(11) cpuexpand指定時の乗算結果(H8C-0014)

cpuexpand指定時に、乗算結果が不正になる場合がある。
該当バージョン：V6.00 Release 00 ～ 03
発生条件：以下の条件をすべて満たした場合に、発生することがあります。

1. CPUオプションに、H8SXN,H8SXM,H8SXAまたはH8SXX(-cpu=h8sxn,h8sxm,h8sxa,h8sxx)を選択している。
2. 乗除算の拡張解釈オプション(-cpuexpand)を選択している。
3. 乗算式がある

例:

```
void  f (void){
  int a,b;
  long c,d;
  a=b=d=0;
  ++d;
  ++a;
  b+=2;
  c=(volatile long)(a*b);// a*aを計算
  ++d;
  if ((d==(volatile long)(a*b)) && (c==2)){
     printf("t026_04 OK¥n");
  } else {
     printf("t026_04 NG¥n");
  }
}
```

---

(12) abs16指定時のセクション名不一致(H8C-0015)

テーブル方式で展開されるswitch文が含まれるソースを、H8SXNまたは H8SXMのCPU用にコンパイルする

と、ABS16の接頭語の付いたセクションが 出力される場合がある。
該当バージョン：V6.00 Release 00 ～ 03
発生条件：以下の条件をすべて満たした場合、発生することがあります。

1. CPUオプションに、H8SXNまたはH8SXM(-cpu=h8sxn,h8sxm)を選択している。
2. 短絶対アドレスオプション(-abs16)を選択している。
3. switch文展開方式オプションで、自動(-case=auto)でテーブル方式に展開されているコードが存在する。またはテーブル展開方式(-case=table)を選択している。

例:

```
------------------------------------------------------
------
void main(void){
switch(l){
  case -3:
  case -1:
     l = 10;
     break;
  }
}
------------------------------------------------------
------
```

(13) 構造体代入(H8C-0016)
構造体メンバの代入において不正な値が代入される場合がある。
該当バージョン：V6.00 Release 00 ～ 03
発生条件：以下の条件をすべて満たした場合、発生することがあります。

1. CPUオプションに、H8SXX,H8SXA,H8SXNまたはH8SXM(-cpu=h8sxx,h8sxa,h8sxn,h8sxm)を選択している。
2. 最適化オプション(-optimize=1)を選択している。
3. 構造体サイズが4バイト以下(各メンバのサイズは4バイト未満)の変数がレジスタに割り付けられている。
4. 条件3の変数のメンバを使用している。
5. 変数を割り付けるために、レジスタの内容をスタック退避する。
6. 条件5で退避されるレジスタには、条件3の変数が割り付けられている。

7. 条件6で退避された変数にアクセスする式がある。

例:

```
-------------------------------------------------------
------
struct _ST{
  char c;
  int i;
}st1;
void f(){
  struct _ST lst1;
  . . . . . .
  lst1.i=1;
  . . . . . .
  switch(x){
     . . . . . .
     switch(y){
        case 3:
           . . . . . .
           lst1.i =2;
           . . . . . .
           break;
     }
     . . . . . .
  }
  . . . . . .
  st1=lst1;
  . . . . . .
}
-------------------------------------------------------
------
```

(14) その他

1. try-catchを使用したプログラムに対してコンパイルまたはリンク時にエラーを出力する場合があります。
2. コンパイル時に(C)4098を出力する場合があります。
3. デバッガのWatchまたはLocalsウィンドウに不正な値を表示する場合があります。
4. スタック解析ツールで"アドレス参照未解決関数"と表示する場合があります。

2.2.3 最適化リンケージエディタ

以降6点の問題を改修しました。

(1) 共通コード統合最適化指定時のデバッグ情報
共通コード統合最適化(optimize=same_code)が有効な場合、生成されたデバッグ情報をもとにデバッガでステップ実行を行うと意図しない関数へ飛ばされる場合がある。

(2) サブコマンドファイルのinputオプション記述に対する不正なウォーニング出力
サブコマンドファイル内で、例のような記述がされた場合、不正なウォーニングメッセージ(L1010)が出力される。

例：

```
-----------------
 -input=a.obj
 -input=b.obj
 -input=
-----------------
```

(3) 共通コード統合最適化による内部エラーの発生または不正なオブジェクト生成
共通コード統合最適化が有効な場合に不正なコードが生成される、 または、他のリンク時最適化の際に内部エラーとなる場合がある。
発生条件：以下の条件をすべて満たす場合、発生する可能性があります。
1. コンパイル時にgoptimizeオプションを指定している。
2. 共通コード統合最適化(optimize=same_code)が有効である。
3. optlnk V.8.00.03以降のバージョンを使用している。
4. 外部変数へのアクセスや関数呼び出しがないオブジェクトファイル(もしくはライブラリモジュール)がリンクされる。
5. 4.のコードが2.の最適化の対象となる。

(4) 短絶対アドレッシング活用の最適化によるエラー発生
短絶対アドレッシング活用の最適化が有効な場合、不正なエラー(L2330) が出力される場合がある。
発生条件：以下の条件をすべて満たす場合、発生する可能性があります。
1. コンパイル時にgoptimizeオプションを指定している。
2. 短絶対アドレッシング活用の最適化(optimize=variable_access)が有効である。
3. 下記のいずれかの条件に該当する。
   3a. sbrオプションの指定アドレスが16bit絶対アドレス範囲外である。
   3b. 16bit絶対アドレス範囲内に配列や構造体変数が割りついている。

(5) レジスタ退避および回復コードの最適化による不正なオブジェクト生成
コンパイル時に引数格納レジスタ指定を行った場合、レジスタ退避および回復 コードの最適化によって不正なコードが生成される場合がある。
発生条件：以下の条件をすべて満たす場合、発生する可能性があります。
1. コンパイル時にgoptimizeオプションを指定している。
2. コンパイル時にregparam=3オプションを指定している、または、__regparam3キーワードを指定している関数がある。
3. リンク時にレジスタ退避および回復コードの最適化(optimize=register)が有効である。

(6) 以下に該当する場合に内部エラーが発生する。
1. 未参照シンボル削除最適化(optimize=symbol_delete)が有効な場合にoutputオプションで分割出力指定する(内部エラー(L4000-7041)発生)。
2. 短絶対アドレッシング活用の最適化(optimize=variable_access)が有効な場合(内部エラー(L4000-8996)発生)。
3. レジスタ退避および回復最適化(optimize=register)が有効な場合(内部エラー(L4000-8416)発生)。

3. **リビジョンアップと購入方法**

3.1 リビジョンアップ（無償）

該当製品をお持ちの場合、無償でリビジョンアップできます。

(1) Windows版: R0C40008XSW06Rの場合
オンラインでリビジョンアップできます。
こちらからダウンロードしてください。

(2) R0C40008XSS06R(Solaris版) および R0C40008XSH06R(HP-UX版)の場合
以下の情報を最寄りのルネサス販売または特約店までご連絡ください。
最新版の製品パッケージを送付いたします。

製品型名：

Solaris版 ： R0C40008XSS06R

HP-UX版 ： R0C40008XSH06R

バージョン番号 ： V.6.01

リリース番号 ： Release 00

3.2 新規購入

ご注文の際には、以下の情報を最寄りのルネサス販売または特約店までご連絡ください。

製品型名：

Windows版 ： R0C40008XSW06R

Solaris版 ： R0C40008XSS06R

HP-UX版 R0C40008XSH06R

バージョン番号 ： V.6.01

リリース番号 ： Release 00

[免責事項]
過去のニュース内容は発行当時の情報をもとにしており、現時点では変更された情報や無効な情報が含まれている場合があります。
ニュース本文中のURLを予告なしに変更または中止することがありますので、あらかじめご承知ください。