RH850 用 C コンパイラパッケージ(CC-RH)
PIC/PID 機能編

R20AN0505JJ0100
Rev.1.0
2018.07.20

要旨

PIC/PID 機能の概要と使用方法、応用例について説明します。

対象リビジョン

CC-RH V1.07.00 以降

目次

1. PIC/PID 機能とは

PIC/PID 機能とは、一度リンク処理が完了して配置アドレスが確定したコードやデータを、リンクし直すことなく任意のアドレスに配置して実行できるようにする機能です。

本アプリケーションノートでは、PIC/PID 機能を利用するために、「アプリケーション・プログラム」と「マスタ・プログラム」と呼ばれる 2 つのプログラムを用意する例を説明します。アプリケーション・プログラムは、PIC/PID 機能を利用することでメモリ上の任意のアドレスに配置して実行できるプログラムであり、マスタ・プログラムはアプリケーション・プログラムを実行するためのプログラムです。

1.1 機能の概要

CC-RH では、以下 3 つの機能をサポートしています。

PIC(Position Independent Code)機能

メモリ上の任意のアドレスにコード(関数)を配置して、実行できる機能.です。

-pic オプションを指定することにより、関数用のセクションを位置独立にします。

PIROD(Position Independent Read Only Data)機能

メモリ上の任意のアドレスに定数データ(const 変数)を配置して、参照できる機能です。

-pirod オプションを指定することにより、定数データ用のセクションを位置独立にします。

PID(Position Independent Data)機能

メモリ上の任意のアドレスにデータ(変数)を配置して、参照できる機能です。

-pid オプションを指定することにより、データ用のセクションを位置独立にします。

1.2 活用例

PIC 機能を使用すると、実行中のアプリケーションに影響を与えずに別の場所に更新アプリケーションを配置し、そのまま更新アプリケーションを動作させることができます。

なお、アプリケーション・プログラムから標準ライブラリ等の PIC でない共通ルーチンを呼び出すことも可能ですが、その場合は共通ルーチンのプロジェクトをアプリケーション・プログラムより先にビルドし、アプリケーション・プログラムから共通ルーチン側の絶対アドレスを参照する必要があります。

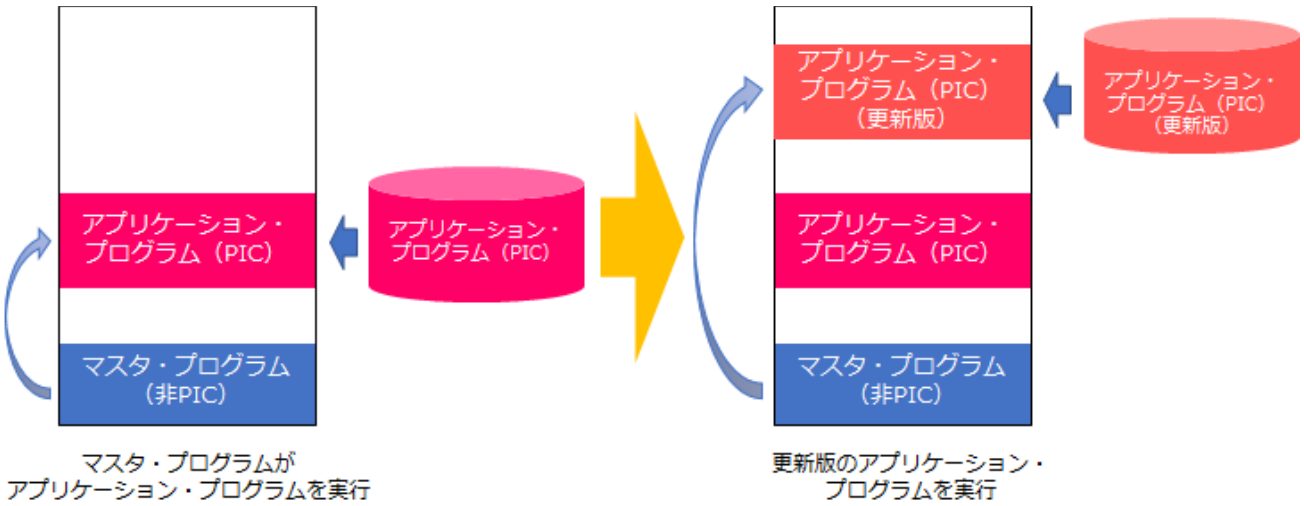

図 1-1 PIC/PID の活用例

2. PIC 機能

2.1 コンパイラ・オプション

コンパイラ・オプション”-pic”を指定することで、PIC 機能を使用できます。

なお、本オプションは”-pirod”オプションと同時に指定してください。

CS+を使用している場合は [共通オプション]タブ → [PIC/PID]カテゴリ → [はい(-pic -pirod)] を選択することで、PIC 機能が有効になります。

CS+上で PIC 機能を有効にした場合、”-pic”と”-pirod”オプションが常に同時に指定されます。

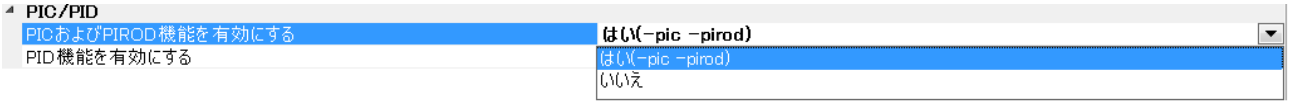

図 2-1 -pic、-pirod オプションの設定

2.2 PIC 機能で使用するセクション

”-pic”オプションを指定した場合、コードの配置先のデフォルト・セクション名が.text から.pctext になります。

.text に配置した関数の呼び出しやアドレス参照に対しては PC 相対あるいは r0(0 番地)からの 32 ビット長相対でアクセスします。r0 からの 32 ビット長相対でアクセスする場合は位置独立ではありません。.pctext に配置した場合は必ず PC 相対でアクセスすることで位置独立にします。

”-start”オプションによるセクションのアドレス指定も、.text から.pctext へ変更してください。このときのアドレスは、PC 相対セクション同士の位置関係を決定するために指定します。実行時のアドレスである必要はありません。

表 2-1 PIC 機能で使用するセクション

セクション再配置属性	デフォルト・セクション名	アクセス方法	整列条件
pctext	.pctext	PC からの 32 ビット長相対	2

セクション名は #pragma section 指令によって変更可能です。

下記の例ではセクション名を”test.pctext”に変更しています。

```
#pragma section pctext "test"
void func(void) {    //  test.pctext
…
}
```

2.3 PIC 関数の使用例

2.3.1 非 PIC 関数から PIC 関数の呼び出し

非 PIC 関数から PIC 関数を呼び出す例については 6.1.3 で説明します。

2.3.2 PIC 関数から PIC 関数の呼び出し

C ソースからは通常関数と同様に関数名で呼び出すことができます。

C ソース例

```
void func1(void) {
func2();
}
```

関数を PC 相対で呼び出します。

コンパイル結果

```
_func1:
        .stack   _func1 = 4
        prepare 0x00000001, 0x00000000
        jarl    _func2, r31
        dispose 0x00000000, 0x00000001, [r31]
```

2.3.3 PIC 関数から非 PIC 関数の呼び出し

C ソースからは通常関数と同様に関数名で呼び出すことが可能です。

C ソース例

```
#pragma section text  // マスタ・プログラム(非 PIC)で定義したセクション
void nopic_func();
#pragma section default

void func1(){         // func1 を.pctext セクションに配置
  nopic_func();       // 非 PIC 関数の呼び出し
}
```

非 PIC 関数の絶対アドレスを r0 相対で呼び出すコードを生成します。アプリケーション・プログラムからマスタ・プログラムの関数を呼び出す場合は、マスタ・プログラム側で生成したシンボル・アドレス・ファイル(*.fsy)をアプリケーション・プログラム側で参照することによりアドレス解決します。非 PIC 関数を実行後は、r31 レジスタ(lp)を使用して PIC 関数に復帰します。

コンパイル結果

```
_func1:
    .stack   _func1 = 4
    prepare 0x00000001, 0x00000000
    mov     #_nopic_func, r2  // 非 PIC 関数の絶対アドレスを取得
    jarl    [r2], r31
    dispose  0x00000000, 0x00000001, [r31]
```

なお、-pic オプション指定時、関数の定義は pctext 属性セクション内でのみ可能です。

3. PIROD 機能

3.1 コンパイラ・オプション

コンパイラ・オプション”-pirod”を指定することで、PIROD 機能を使用できます。

なお、本オプションは”-pic”オプションと同時に指定してください。また、”-Omap”、”-Osmap”オプションと同時に指定することはできません。

CS+上での設定方法は 2.1 をご参照ください。

3.2 PIROD 機能で使用するセクション

”-pirod”オプションを指定した場合、定数データの配置先のデフォルト・セクション名が.const から.pcconst32 になります。

.const に配置した定数データの呼び出しやアドレス参照に対しては r0(0 番地)からの 32 ビット長相対でアクセスしますので位置独立ではありません。.pcconst32 には PC 相対でアクセスしますので位置独立になります。PIC から PIROD へはリンク時に決定した相対アドレスを用いてアクセスするため、位置関係を変更することはできません。

”-start”オプションによるセクションのアドレス指定も、.const から.pcconst32 へ変更してください。このときのアドレスは、PC 相対セクション同士の位置関係を決定するために指定します。実行時のアドレスである必要はありません。

表 3-1 PIROD 機能で使用するセクション

セクション再配置属性	デフォルト・セクション名	アクセス方法	整列条件
pcconst32	.pcconst32	__pc_data シンボルからの 32 ビット長相対	4

セクション名は #pragma section 指令によって変更可能です。

下記の例ではセクション名を”test.pcconst32”に変更しています。

```
#pragma section pcconst32 "test"
const int a = 1;   //  test.pcconst32
```

また、配置先セクションを.pcconst16 セクションや.pcconst23 セクションに変更することで、より短い命令長で定数データを参照できます。

```
#pragma section pcconst16
const int a = 1;  //  .pcconst16
```

3.3 PIROD 機能の使用例

C ソースからは通常の定数と同様に定数名で呼び出すことが可能です。

C ソース例

```
const int a = 3;
int func() {
return a;
}
```

PIC 関数から PIROD 対象の変数を参照する場合は、PC 相対で参照します。

コンパイル結果

```
.public   _a, 4
.public   _func
.section  ".pctext", pctext
func:
     .stack   func = 0
     jarl     .BB.LABEL.1_1, r2      ; .LABEL.1_1 の実行時アドレスを r2 に設定

.BB.LABEL.1_1:
     mov      #.BB.LABEL.1_1-#__pc_data, r5   ; リンク時に決定した相対アドレス
    sub      r5, r2
     movhi    HIGHW1(#_a-#__pc_data), r2, r2  ; _a を PC 相対で参照
     ld.w     LOWW(#_a-#__pc_data)[r2], r10
     jmp      [r31]
     .section  ".pcconst32", pcconst32
     .align   4
a:
     .dw      0x00000003
```

__pc_data : リンカが自動的に生成する、PC 相対アクセス用のベースシンボルです。

4. PID 機能

4.1 コンパイラ・オプション

コンパイラ・オプション”-pid”を指定することで、PID 機能を使用できます。

なお、”-r4=none”および”-Omap”、”-Osmap”オプションと同時に指定することはできません。

CS+を使用している場合は [共通オプション]タブ → [PIC/PID]カテゴリ→ [はい(-pid)] を選択することで、PID 機能が有効になります。

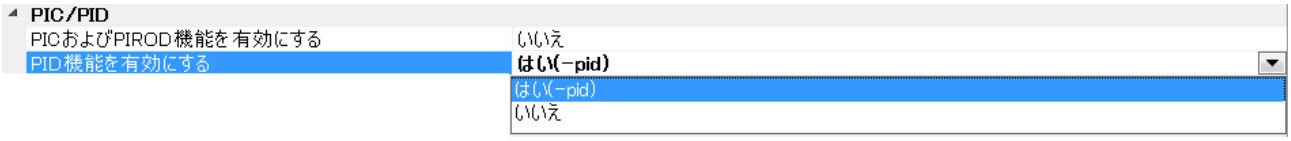

図 4-1 –pid オプションの設定

4.2 PID 機能で使用するセクション

”-pid”オプションを指定した場合、変数データの配置先のデフォルト・セクション名が.data、.bss から.sdata32、.sbss32 になります。

.data、.bss に配置した変数データの呼び出しやアドレス参照に対しては r0(0 番地)からの 32 ビット長相対でアクセスしますので位置独立ではありません。、.sdata32、.sbss32 へは GP 相対でアクセスしますので位置独立になります。

なお、.sdata32、.sbss32、.edata32、.ebss32 は PID 機能専用のセクションです。デフォルトでは、PID 用のセクションに配置した変数データに対する参照は、すべて GP 相対で行いますが、＃pragma section により EP 相対指定した場合は、.edata32、ebss32 に配置して EP 相対で行います。

上記以外の GP 相対セクション、EP 相対セクションを PID 機能で使用することも可能です。使用可能なセクションについてはユーザーズ・マニュアルをご参照ください。

”-start”オプションによるセクションのアドレス指定も、.data, .bss から.sdata32, .sbss32 へ変更してください。このときのアドレスは、GP 相対または EP 相対参照のベースシンボルとの位置関係を決定するために指定します。実行時のアドレスである必要はありません。

表 4-1 PID 機能で使用するセクション

セクション再配置属性	デフォルト・セクション名	割り当て対象	アクセス方法	整列条件
sdata32	.sdata32	初期値あり変数	r4（GP）からの 32 ビット長相対	4
sbss32	.sbss32	初期値なし変数		

#pragma section 指令によって、セクション名を変更可能です。

下記の例ではセクション名を”test.sdata32”、 ”test.sbss32”に変更しています。

```
#pragma section sdata32 "test"
int a = 1;    //  test.sdata32
int b ;       //   test.sbss32
```

4.3 PID 機能の使用例

PIC 関数から PID 対象の変数を参照する場合、C ソースからは通常の変数と同様に変数名で呼び出すことが可能です。

C ソース例

```
int a = 1;
int func() {
    return a;
}
```

GP 相対または EP 相対で参照します。

コンパイル結果

```
.public   _a, 4
.public   _func1
_func1:
      .stack    _func1 = 0
      movhi     HIGHW1($_a), r4, r2  // GP 相対で参照
      ld.w      LOWW($_a)[r2], r10
      jmp       [r31]
.section .sdata32, sdata32
.align    4
_a:
      .dw 0x00000001
```

5. スタートアップ

PIC/ PID 機能を使用する場合、標準のスタートアップ・ルーチンは使用できません。スタートアップ・ルーチン内の次の処理を変更する必要があります。

- ベース・レジスタ初期化
- RAM セクションの初期化
- main 関数への分岐

スタートアップのコーディング例は、付録を参照ください。以降、付録のコーディング例について詳細を説明します。

5.1 ベース・レジスタ初期化

PID 機能を使用する場合は、まず事前に、リンク時に指定した RAM セクションの開始アドレスからのオフセット情報（以降 **RAM オフセット値**と呼びます）を、受け渡す手段を決めておきます。例えば、特定の RAM 領域や、データ・フラッシュ領域に RAM オフセット値を書き込んでおきます。[注]

【注】 この場合、その特定の領域は絶対アドレスで参照する必要があるため、PID、PIROD 機能の対象外になります。

または、マイコンの電源を遮断しないでプログラムを再起動する場合は、特定のレジスタに RAM オフセット値を格納しておきます。受け取った RAM オフセット値を、ベース・レジスタに加算して、実行時のベース・アドレスとして使用します。

```
$ifdef __PID
      mov    #_PID_offset, r28        ; 受け渡し用のメモリのアドレス
                                      ; このアドレスに RAM オフセット値を格納
      ld.w   0[r28], r28              ; リンク時のデータ配置と実行時のデータ配置の間の
                                      ; オフセット（RAM オフセット値）を r28 レジスタに
格納
$endif
           ; PIC 側で gp/ep を両方使用する場合
             mov    #_stacktop, sp             ; set sp register
             mov    #__gp_data, gp             ; set gp register
             mov    #__ep_data, ep             ; set ep register
$ifdef __PID
             add    r28, sp     ; gp/ep をずらした際のスタック領域へのオーバーラップを
回避
                                ; オーバーラップしないのであれば不要
             add    r28, gp
             add    r28, ep
$endif
```

5.2 RAM セクションの初期化

_INITSCT_RH()関数はセクション情報のテーブルが入力であるため、PID 機能使用時のセクション初期化には使用できません。そこで、スタートアップ・ルーチン内で直接初期値をコピーします。事前準備として、コード領域、定数データ領域の、リンク時と実行時の配置オフセットを求めます。以降 **ROM オフセット値**と呼びます。

```
      jarl   .pic_base, r29      ; r29 に実行時の.pic_base ラベルのアドレスを格納する
.pic_base:
      mov    #.pic_base, r10     ; r10 にリンク時の.pic_base ラベルのアドレスを格納す
る
      sub    r10, r29            ; r29 - r10 の値が ROM オフセット値になる
```

次に、初期値ありセクションの初期化を行います。初期化の対象となるセクションの、初期値のコピー元の先頭アドレス、終端アドレス、コピー先アドレスを r6, r7, r8 レジスタに格納します。(*1)

PIROD 機能を使用している場合、初期値のコピー元の先頭アドレス、終端アドレス（r6, r7 レジスタ）に ROM オフセット値を加算します。(*2)

PID 機能を使用している場合、データのコピー先アドレス（r8 レジスタ）に RAM オフセット値を加算します。(*3)

```
      mov    #__s.sdata32, r6    ; (*1) コピー元セクションアドレス先頭の格納
      add    r29, r6             ; (*2) ROM オフセット値の加算

      mov    #__e.sdata32, r7    ; (*1) コピー元セクションアドレス終端の格納
      add    r29, r7             ; (*2) ROM オフセット値の加算

      mov    #__s.sdata32.R, r8  ; (*1) コピー先セクションアドレスの格納
      add    r28, r8             ; (*3) RAM オフセット値の加算
```

ここまででコピーの準備ができたので、コピールーチンを呼び出します。

```
      jarl         _copy4, lp
....
      ; r6: source begin (4-byte aligned)
      ; r7: source end (r6 <= r7)
      ; r8: destination begin (4-byte aligned)
            .align       2
copy4:
      sub    r6, r7
.copy4.1:
      cmp    4, r7
      bl     .copy4.2
      ld.w   0[r6], r10
      st.w   r10, 0[r8]
      add    4, r6
      add    4, r8
      add    -4, r7
      br     .copy4.1
.copy4.2:
      cmp    2, r7
      bl     .copy4.3
      ld.h   0[r6], r10
      st.h   r10, 0[r8]
      add    2, r6
      add    2, r8
      add    -2, r7
.copy4.3:
      cmp    0, r7
      bz     .copy4.4
      ld.b   0[r6], r10
      st.b   r10, 0[r8]
.copy4.4:
      jmp     [lp]
```

ここまでの処理を、初期値が必要なセクションの数だけ繰り返します。

次に初期値無しセクションを 0 で初期化します。対象セクションの先頭アドレス、終端アドレスを、r6, r7 レジスタに格納します。PID 機能を使用している場合、先頭アドレス、終端アドレス（r6, r7）に RAM オフセット値を加算します。

```
$ifdef __PID
      mov    #__s.sbss32, r6
      mov    #__e.sbss32, r7
      add    r28, r6           ; RAM オフセット値の加算
      add    r28, r7           ; RAM オフセット値の加算
$else
      mov    #__s.sbss, r6
      mov    #__e.sbss, r7
$endif
```

初期化ルーチンを呼び出して、対象セクションを 0 で初期化します。

```
      jarl         _clear4, lp
....
      ; r6: destination begin (4-byte aligned)
      ; r7: destination end (r6 <= r7)
.align      2
_clear4:
      sub    r6, r7
.clear4.1:
      cmp    4, r7
      bl     .clear4.2
      st.w   r0, 0[r6]
      add    4, r6
      add    -4, r7
      br     .clear4.1
.clear4.2:
      cmp    2, r7
      bl     .clear4.3
      st.h   r0, 0[r6]
      add    2, r6
      add    -2, r7
.clear4.3:
      cmp    0, r7
      bz     .clear4.4
      st.b   r0, 0[r6]
.clear4.4:
      jmp     [lp]
```

ここまでの処理を、初期化が必要なセクションの数だけ繰り返します。

5.3 main 関数への分岐

PIC 機能を使用していて、FERET 命令で main 関数へ分岐する場合、FEPC に格納する値に ROM オフセット値を加算します。

```
      mov    #_exit, lp                ; lp <- #_exit
      mov    #_main, r10
$ifdef __PIC
      add    r29, lp                   ; ROM オフセット値の加算
      add    r29, r10                  ; ROM オフセット値の加算
$endif
      ldsr   r10, 2, 0                 ; FEPC <- #_main
                                       ; apply PSW and PC to start user mode
      feret
```

6. PIC/PID 機能の応用例

この章では、PIC/PID 機能を使用した CS+プロジェクトの作成方法について説明します。

6.1 PIC/PID 機能を使用したプロジェクトの構築

マスタ・プログラム(非 PIC)からアプリケーション・プログラム(PIC)を呼び出す場合の CS+のプロジェクトの作成方法について説明します。

6.1.1 CS+プロジェクトの構成

マスタ・プロジェクト(非 PIC)を通常の CS+プロジェクトの作成手順で作成した後、サブプロジェクトとしてアプリケーション・プロジェクト(PIC)を追加します。本アプリケーションノートでは、アプリケーション・プログラムを 1 プロジェクトで構成するプログラム例を説明します。つまり、アプリケーション・プログラム内での相対アドレスの関係は常に一定であるものとします。

CS+のプロジェクト・ツリーでは以下の構成になります。

図 6-1 プロジェクト構成

6.1.2 マスタ・プロジェクトの作成

CS+を起動し、[スタート]ボタンを押下してスタートパネル上の[新しいプロジェクトを作成する]の[GO]ボタンを押下し、プロジェクトを作成してください。

6.1.3 マスタ・プログラムからアプリケーション・プログラムの起動

実行時に配置した PIC のエントリ・ポイントを指定して、マスタ・プログラムからアプリケーション・プログラムを起動してください。

PID 機能を使用する場合、ベース・レジスタ(GP/EP)を初期化するために RAM オフセット値をアプリケーション・プログラムに受け渡す必要があります(5.1 を参照)。以下は付録のコーディング例を使用する場合の例です。

特定のアドレスに RAM オフセット値を格納することで受け渡しを行います。リンカオプションに –start=PID_OFFSET.bss/<受け渡し用のアドレス> を追加して、PID_offset を受け渡し用のアドレスに配置してください。

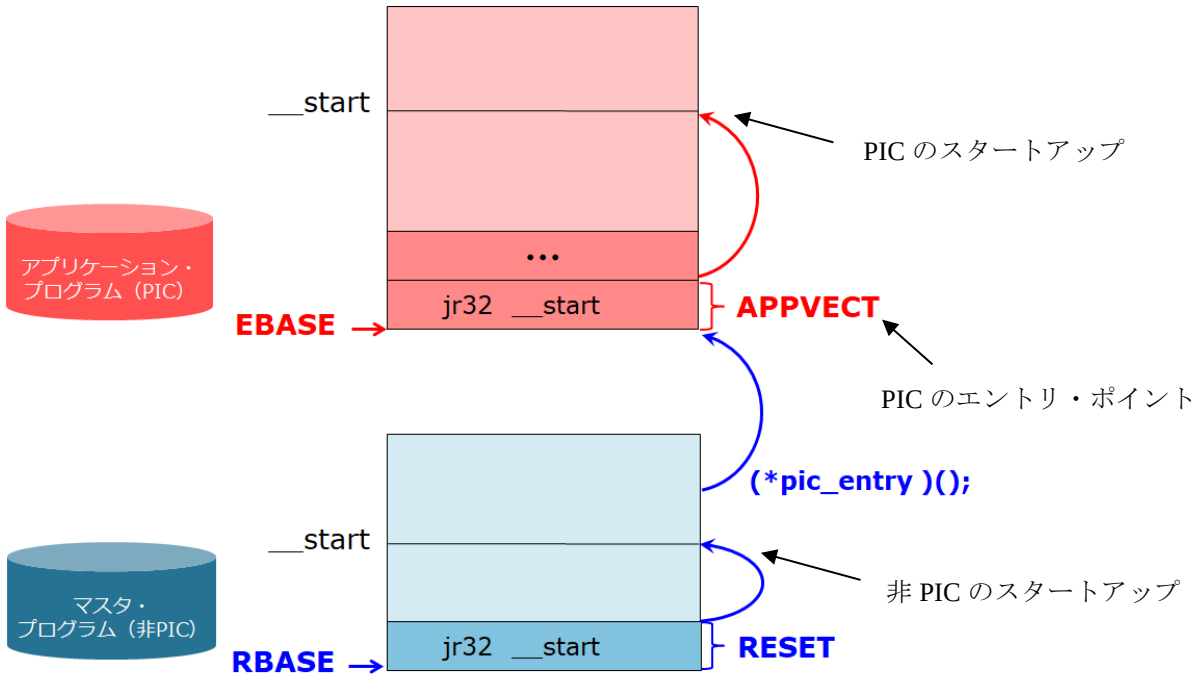

図 6-2 付録のコーディング例によるマスタ・プログラムからアプリケーション・プログラムへの分岐例

C ソース例

```
#pragma section PID_OFFSET
unsigned long PID_offset;                    // RAM オフセット値の格納場所
#pragma section default

void main() {
  void (*pic_entry)(void) = (void*)<実行時のエントリ・ポイント>;// APPVECT のアドレス
  PID_offset = <実行時の RAM オフセット値>;
  (*pic_entry)();
}
```

<実行時のエントリ・ポイント>は、アプリケーション・プログラム(PIC)を書き込んだ先頭アドレスを動的に取得できるようなプログラムにしてください。

6.1.4 アプリケーション・プロジェクトの追加

プロジェクト・ツリーでプロジェクト・ノードを選択後、コンテキスト・メニューの［追加］→［新しいサブプロジェクトを追加...］または［既存のサブプロジェクトを追加...］を選択して、アプリケーション・プロジェクトを追加してください。

アプリケーション・プロジェクトを新規に作成する場合は、2~5 章を参照し、オプションやセクションの設定、およびスタートアップ・ルーチンを編集してください。

6.1.5 アプリケーション・プログラムからマスタ・プログラムの参照

(1) マスタ・プログラムの外部定義シンボルの参照

アプリケーション・プログラムからマスタ・プログラムの関数や変数を参照する場合、参照したい関数、変数の宣言と参照する処理を記述します。この宣言が所属するセクションについては、マスタ・プログラムの関数、変数のセクションと合わせておく必要があります。参照元であるアプリケーション・プログラムの関数は、PIC 用のセクションに定義してください。

アプリケーション・プログラムの C ソース例

```
#pragma section text "comm"
extern void *nopic_func(void);  // 非 PIC 関数

#pragma section pctext          // PIC 関数を定義するときは、
                                // セクション再配置属性を PIC 用に戻す
void pic_func(void) {
  nopic_func();
}
```

アプリケーション・プログラムのビルド時には、マスタ・プログラムの外部定義シンボルの情報が必要になります。マスタ・プログラムをビルドする時に、アプリケーション・プログラムから参照したい関数や変数のアドレスをシンボル・アドレス・ファイル(*.fsy)に出力してください。

マスタ・プログラムの[リンク・オプション]タブ →[セクション]カテゴリ →[外部定義シンボルをファイル出力するセクション)]の右端の[...]ボタンを押下し、外部定義シンボルのセクション名を指定してください。

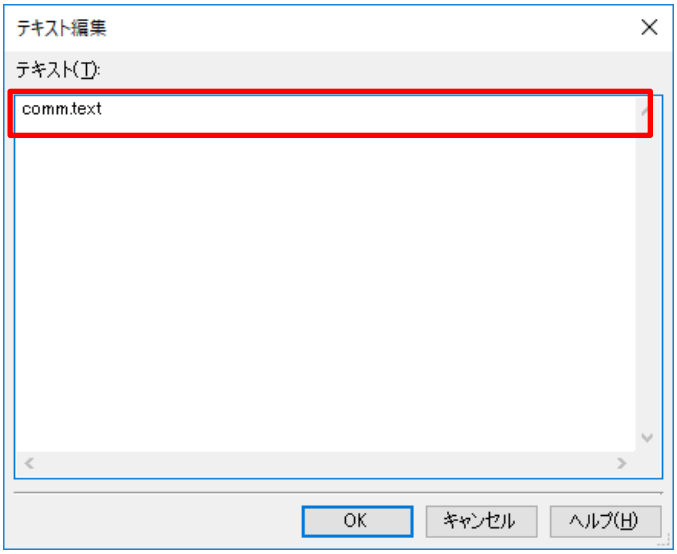

図 6-3 外部定義シンボルのセクションの設定画面

*.fsy の出力例

```
;SECTION NAME = comm.text
.public _nopic_func
_nopic_func .equ 0xXXXXXXXX
```

外部定義シンボル名　　配置アドレス

次に、*.fsy ファイルをアプリケーション・プロジェクトに登録します。プロジェクト・ツリーのファイル・ノードを右クリック → [追加]から登録できます。

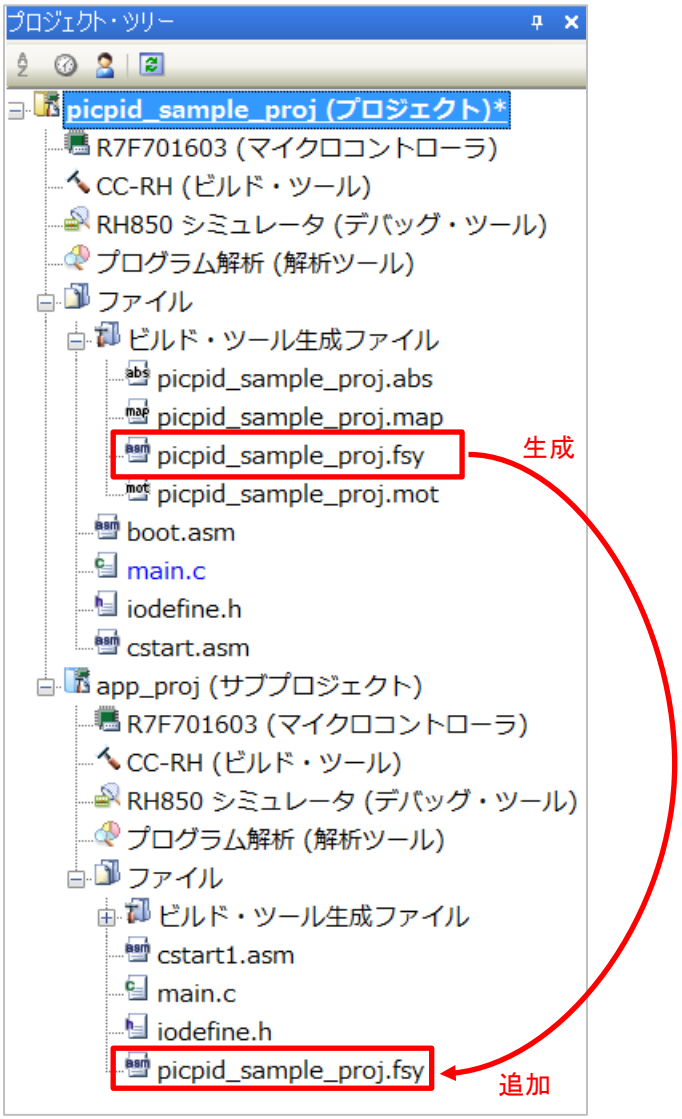

図 6-4　アプリケーション・プログラムに*.fsy ファイルを追加した後のプロジェクト・ツリー

【補足】マスタ・プログラムの標準ライブラリを参照する場合

アプリケーション・プログラムで標準ライブラリを参照する場合は、マスタ・プログラムのリンク・マップ・ファイルを参照して、関数および変数のシンボル名とアドレスの情報を、*.fsy ファイルに直接記述してください。

C ソース例

アプリケーション・プログラムからマスタ・プログラムのライブラリ関数を参照する場合、以下のような疑似的な参照コードを記述し、マスタ・プログラムにライブラリ関数をリンクさせます。

```
#include <string.h>
void* const dummy_libcall[] = {&memcpy, &memcmp, &strcpy};
```

リンク・マップ・ファイルの出力例

```
  SYMBOL                          ADDR          SIZE      INFO    COUNTS  OPT
FILE = memcpy
                                  00002024  0000203b        18
  _memcpy
                                  00002024         0   none ,g ※
```

*.fsy の記述例

```
;SECTION NAME = text
 .public _memcpy
_memcpy .equ 0x00002024
```

(2) 依存プロジェクトの設定

アプリケーション・プログラムからマスタ・プログラムを参照する場合、マスタ・プロジェクト → アプリケーション・プロジェクトの順でビルドする必要があります。

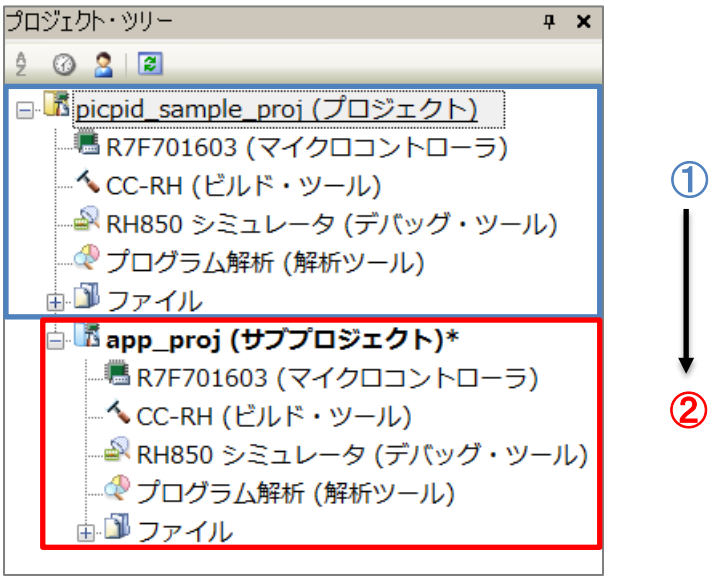

図 6-5 プロジェクトのビルド順のイメージ

なお、CS+ではプロジェクトのビルド順を制御することが可能です。[プロジェクト]メニュー → [依存プロジェクト設定] を押下し、[依存プロジェクト設定] ダイアログ上に設定します。

以下の設定により、app_proj は picpid_sample_proj に依存するため、picpid_sample_proj → app_proj の順にビルドします。

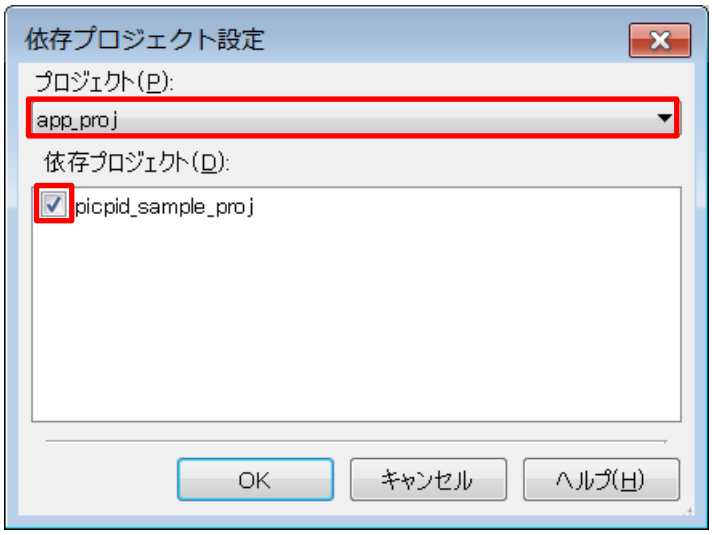

図 6-6 依存プロジェクトの設定

6.2 割り込み/例外を PIC にする

アプリケーション・プログラムで割り込み/例外を PIC にする場合(※)、ハンドラ・アドレスの基準位置は、EBASE レジスタで示されるアドレス(ROM の配置先のアドレスを設定)に要因のオフセットを加算した値を使用します。EBASE にベクタテーブルのアドレスを設定し、PSW.EBV に 1 を設定してください。また、セクションの境界は 512 バイトに調整してください。

(※) RESET ベクタは PIC にできないため、対象外です。RESET のかわりに、エントリ・ポイントを設定してください。

スタートアップ・ルーチンの記述例

```
$ifdef __PIC
; EBASE にベクタテーブルのアドレスを設定
  mov      #__sAPPVECT, r10      ; PIC のエントリ・ポイント・アドレスを r10 に格納
  add      r29, r10              ; ROM オフセット値の加算
  ldsr     r10, 3, 1             ; EBASE <- r10
  stsr     5, r10, 0             ; r10 <- PSW
  mov      0x00008000, r11
  or       r11, r10
  ldsr     r10, 5, 0             ; PSW.EBV に 1 を設定
```

ベクタテーブルの記述例

```
;------------------------------------------------------
;     exception vector table
;------------------------------------------------------
      .section "APPVECT", pctext   ; PIC のエントリ・ポイント
      .align    512
      jr32       __start            ; PIC のスタートアップに分岐

      .align    16
      jr32      _Dummy1               ; 割り込み/例外処理 1

      .align    16
      jr32       _Dummy2              ; 割り込み/例外処理 2
…
```

7. 注意事項

この章では、PIC/PID 機能を使用する際に注意すべき点について説明します。

7.1 変数および関数の参照

アプリケーション・プログラム(PIC)から参照できるマスタ・プログラム(非 PIC)の変数や関数には、参照方法に制約があります。

アプリケーション・プログラム内の関数、変数同士、またアプリケーション・プログラムとマスタ・プログラムとの間の、参照可能な関係と参照方法を次の表に示します。

表 7-1　変数および関数の参照可否と参照方法

		参照先					
		PIC 関数	非 PIC 関数	PIROD 変数	非 PIROD 変数	PID 変数[注 2]	非 PID 変数
参照元	PIC 関数	PC 相対	R0 相対	PC 相対	R0 相対	GP,EP 相対	GP,EP 相対 R0 相対
	非 PIC 関数	不可[注 1]	PC 相対 R0 相対	不可[注 1]	R0 相対	GP,EP 相対	GP,EP 相対 R0 相対

【注】 1. 非 PIC 関数のリンク時には、リンカが PIC 関数、PIROD 変数の実行時のアドレスを特定できないため、あらゆる直接参照ができません。ただし、実行時にポインタを受け取って、ポインタ経由で参照することは可能です。

2. PID 変数とは、GP 相対、EP 相対セクションに配置している変数全般ではなく、-pid オプションを指定してコンパイルした変数を指します。

7.2 静的なアドレスの取得

PIC/PID であるコードやデータは、リンク時とは異なるアドレスで動作します。そのため、これらのコードやデータのアドレスを、静的変数の初期化子に指定することはできません。

以下のように記述した場合、エラーが出力されます。

```
const int c;
int d = 0;

// PIROD 変数のアドレス指定はエラー
void* vp1 = &c;
// PIROD 変数のリテラルはエラー
const char* cp const = "string";
// PID 変数のアドレス指定はエラー
void* vp2 = &d;
```

7.3 GP 相対、EP 相対セクションの使用

GP 相対、EP 相対セクションは、PID のデータと、非 PID のデータの両方で使用可能です。ただし、GP、EP レジスタを共有しているため、PID 機能を使用するために GP、EP レジスタの値を変更すると、非 PID のデータの参照アドレスも変更されます。GP、EP レジスタはそれぞれについて、PID 用に使うか、非 PID 用に使うかを、プログラム全体で統一することを推奨します。

7.4 標準ライブラリの使用

標準ライブラリは、PIC/PID 機能に対応していません。マスタ・プログラムに配置して使用してください。

7.5 コンパイル・オプション

アプリケーション・プログラムとマスタ・プログラムを連携させる場合、次のコンパイル・オプションを合わせる必要があります。

表 7-2 アプリケーション・プログラムとマスタ・プログラムで合わせるオプション

オプション	説明
-Xenum_type	列挙型に対して、どの整数型として扱うかを指定します。
-Xdbl_size	double 型および long double 型の精度を指定します。
-Xpack	構造体パッキングを行います。
-Xbit_order	ビット・フィールドのメンバの並び順を指定します。
-Xreg_mode	レジスタ・モードを指定します。
-Xreserve_r2	r2 レジスタを予約します。
-Xep	ep レジスタの扱い方を指定します。
-Xfloat	浮動小数点演算命令の生成を制御します。
-Xround	浮動小数点定数の丸めモードを指定します。

付録

PIC/PID 機能を使用する場合のアプリケーション・プログラムのベクタテーブルとスタートアップ・ルーチンとのコーディング例

```
;-------------------------------------------------------
;    exception vector table
;-------------------------------------------------------
     .section "APPVECT", pctext   ; PIC のエントリ・ポイント
     .align   512
     jr32      __start

     .align   16
     jr32     _Dummy1               ; 割り込み/例外処理 1

     .align   16
     jr32     _Dummy2              ; 割り込み/例外処理 2
…
;-------------------------------------------------------
;    startup
;-------------------------------------------------------
     .section     ".pctext", pctext
     .align 2
__start:                    ; PIC のスタートアップ
     jr32   __cstart
```

```
$ifdef __PIC
  .TEXT .macro
    .section .pctext, pctext
  .endm
$else
  .TEXT .macro
    .section .text, text
  .endm
$endif


$ifdef __PID
  .STACK_BSS .macro
    .section .stack.bss, sbss32
  .endm
$else
  .STACK_BSS .macro
    .section .stack.bss, bss
  .endm
$endif

;------------------------------------------------------------------------------
; system stack
;------------------------------------------------------------------------------
STACKSIZE  .set   0x200
```

```
    .STACK_BSS
    .align    4
    .ds        (STACKSIZE)
    .align    4
_stacktop:

;-------------------------------------------------------------------------------
;  startup
;-------------------------------------------------------------------------------
    .TEXT
    .public __cstart
    .align    2
__cstart:

$ifdef __PIC
    jarl      .pic_base, r29
.pic_base:
    mov       #.pic_base, r10
    sub       r10, r29
$endif

$ifdef __PID
    mov       0xfedf0000, r28         ; 受け渡し用のメモリのアドレス
    ld.w      0[r28], r28             ; リンク時のデータ配置と、実行時のデータ配置の間の
                                      ; オフセット（RAM オフセット）
$endif

    mov       #_stacktop, sp          ; set sp register
    mov       #__gp_data, gp          ; set gp register
    mov       #__ep_data, ep          ; set ep register
$ifdef __PID
    add       r28, sp
    add       r28, gp
    add       r28, ep
$endif

    ; initialize .sdata32 section
$ifdef __PID
  $ifdef __PIROD
    mov       #__s.sdata32, r6
    add       r29, r6
    mov       #__e.sdata32, r7
    add       r29, r7
    mov       #__s.sdata32.R, r8
    add       r28, r8

  $else
    mov       #__s.sdata32, r6
    mov       #__e.sdata32, r7
    mov       #__s.sdata32.R, r8
    add       r28, r8
  $endif
$else
  $ifdef __PIROD
    mov       #__s.data, r6
    add       r29, r6
    mov       #__e.data, r7
    add       r29, r7
    mov       #__s.data.R, r8
```

```
  $else
    mov       #__s.data, r6
    mov       #__e.data, r7
    mov       #__s.data.R, r8
  $endif
$endif
    jarl      _copy4, lp

    ; initialize .sbss32 section
$ifdef __PID
    mov       #__s.sbss32, r6
    mov       #__e.sbss32, r7
    add       r28, r6
    add       r28, r7
$else
    mov       #__s.bss, r6
    mov       #__e.bss, r7
$endif
    jarl      _clear4, lp

    ; enable FPU
$if 1 ; disable this block when not using FPU
    stsr      6, r10, 1               ; r10 <- PID
    shl       21, r10
    shr       30, r10
    bz        .L1                     ; detecting FPU
    stsr      5, r10, 0               ; r10 <- PSW
    movhi     0x0001, r0, r11
    or        r11, r10
    ldsr      r10, 5, 0               ; enable FPU

    movhi     0x0002, r0, r11
    ldsr      r11, 6, 0               ; initialize FPSR
    ldsr      r0, 7, 0                ; initialize FPEPC
.L1:
$endif

    ; set various flags to PSW via FEPSW

    stsr      5, r10, 0               ; r10 <- PSW
    ;xori     0x0020, r10, r10        ; enable interrupt
    ;movhi    0x4000, r0, r11
    ;or       r11, r10                ; supervisor mode -> user mode
    ldsr      r10, 3, 0               ; FEPSW <- r10
    mov       #_exit, lp              ; lp <- #_exit
    mov       #_main, r10

$ifdef __PIC
    add       r29, lp
    add       r29, r10
$endif
    ldsr      r10, 2, 0               ; FEPC <- #_main

    ; apply PSW and PC to start user mode
    feret

_exit:
    br        _exit                   ;  end of program
```

```
;-------------------------------------------------------------------------------
; copy routine
;-------------------------------------------------------------------------------
    ; r6: source begin  (4-byte aligned)
    ; r7: source end  (r6 <= r7)
    ; r8: destination begin (4-byte aligned)
    .align    2
_copy4:
    sub       r6, r7
.copy4.1:
    cmp       4, r7
    bl        .copy4.2
    ld.w      0[r6], r10
    st.w      r10, 0[r8]
    add       4, r6
    add       4, r8
    add       -4, r7
    br        .copy4.1
.copy4.2:
    cmp       2, r7
    bl        .copy4.3
    ld.h      0[r6], r10
    st.h      r10, 0[r8]
    add       2, r6
    add       2, r8
    add       -2, r7
.copy4.3:
  cmp         0, r7
  bz          .copy4.4
  ld.b        0[r6], r10
  st.b        r10, 0[r8]
.copy4.4:
  jmp         [lp]

;-------------------------------------------------------------------------------
; clear routine
;-------------------------------------------------------------------------------
    ; r6: destination begin (4-byte aligned)
    ; r7: destination end  (r6 <= r7)
    .align    2
_clear4:
    sub       r6, r7
.clear4.1:
    cmp       4, r7
    bl        .clear4.2
    st.w      r0, 0[r6]
    add       4, r6
    add       -4, r7
    br        .clear4.1

.clear4.2:
    cmp       2, r7
    bl        .clear4.3
    st.h      r0, 0[r6]
    add       2, r6
    add       -2, r7
.clear4.3:
    cmp       0, r7
    bz        .clear4.4
```

```
    st.b      r0, 0[r6]
.clear4.4:
    jmp       [lp]

;-------------------------------------------------------------------------------
; dummy section
;-------------------------------------------------------------------------------
$ifdef __PID
    .section .sdata32, sdata32
.L.dummy.sdata32:
    .section .sbss32, sbss32
.L.dummy.sbss32:
$else
    .section .data, data
.L.dummy.data:
    .section .bss, bss
.L.dummy.bss:
$endif

$ifdef __PIROD
    .section .pcconst32, pcconst32
.L.dummy.pcconst32:
$else
    .section .const, const
.L.dummy.const:
$endif
;-------------------- end of start up module -------------------;
```

ホームページとサポート窓口
ルネサス エレクトロニクスホームページ
https://www.renesas.com/

お問合せ先
https://www.renesas.com/contact/

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2018.07.20	-	初版発行

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器・システムの設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因して生じた損害（お客様または第三者いずれに生じた損害も含みます。以下同じです。）に関し、当社は、一切その責任を負いません。
2. 当社製品、本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害またはこれらに関する紛争について、当社は、何らの保証を行うものではなく、また責任を負うものではありません。
3. 当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を、全部または一部を問わず、改造、改変、複製、リバースエンジニアリング、その他、不適切に使用しないでください。かかる改造、改変、複製、リバースエンジニアリング等により生じた損害に関し、当社は、一切その責任を負いません。
5. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、各品質水準は、以下に示す用途に製品が使用されることを意図しております。

   標準水準： コンピュータ、OA機器、通信機器、計測機器、AV機器、
   家電、工作機械、パーソナル機器、産業用ロボット等

   高品質水準： 輸送機器（自動車、電車、船舶等）、交通制御（信号）、大規模通信機器、
   金融端末基幹システム、各種安全制御装置等

   当社製品は、データシート等により高信頼性、Harsh environment向け製品と定義しているものを除き、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（宇宙機器と、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に使用することは想定していません。たとえ、当社が想定していない用途に当社製品を使用したことにより損害が生じても、当社は一切その責任を負いません。
6. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合および事故につきましては、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は、データシート等において高信頼性、Harsh environment向け製品と定義しているものを除き、耐放射線設計を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
8. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制するRoHS指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。かかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
9. 当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
10. お客様が当社製品を第三者に転売等される場合には、事前に当該第三者に対して、本ご注意書き記載の諸条件を通知する責任を負うものといたします。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。
12. 本資料に記載されている内容または当社製品についてご不明な点がございましたら、当社の営業担当者までお問合せください。

注1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社が直接的、間接的に支配する会社をいいます。

注2. 本資料において使用されている「当社製品」とは、注1において定義された当社の開発、製造製品をいいます。

(Rev.4.0-1 2017.11)

ルネサス エレクトロニクス株式会社

■営業お問合せ窓口

http://www.renesas.com

※営業お問合せ窓口の住所は変更になることがあります。最新情報につきましては、弊社ホームページをご覧ください。

ルネサス エレクトロニクス株式会社　〒135-0061　東京都江東区豊洲3-2-24（豊洲フォレシア）

■技術的なお問合せおよび資料のご請求は下記へどうぞ。
総合お問合せ窓口：https://www.renesas.com/contact/

Colophon 6.0