# 目次

# 第1章　　概説

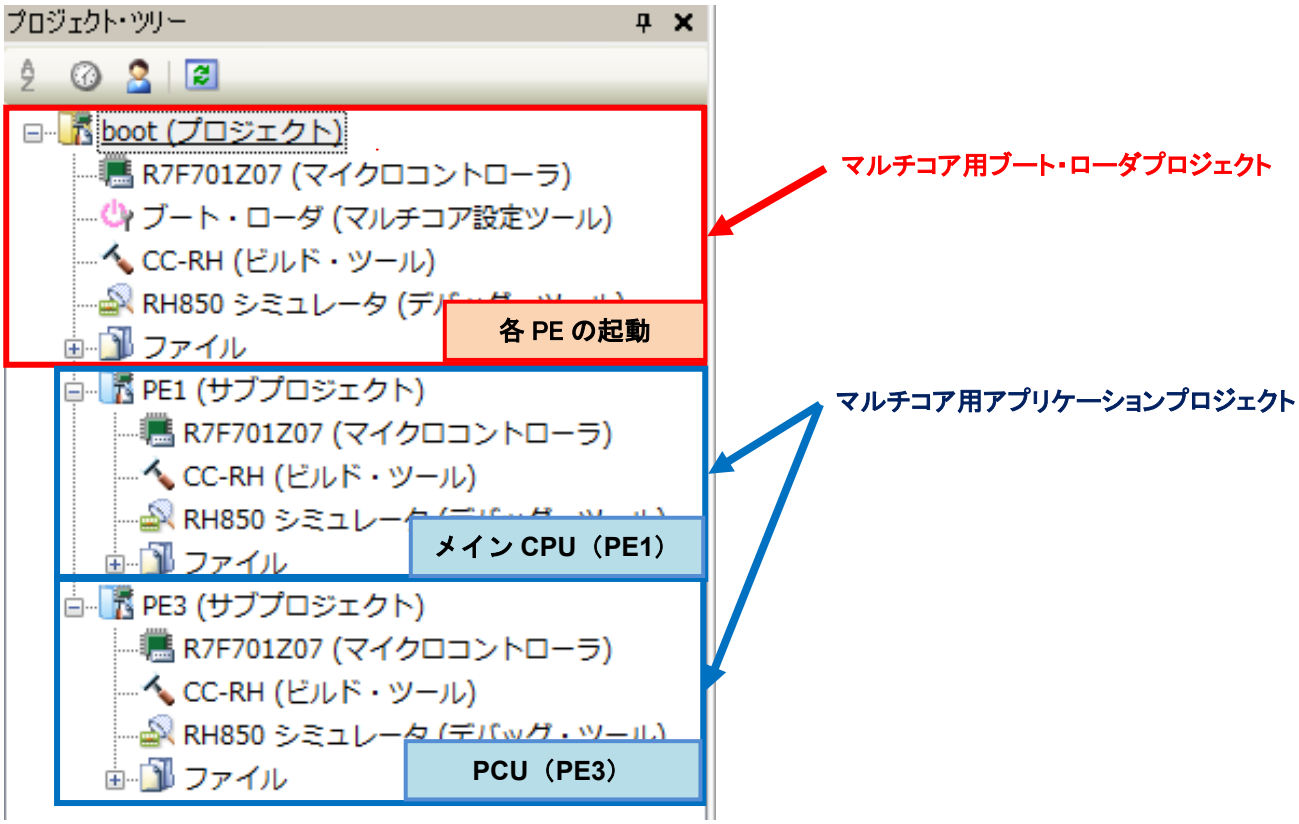
本チュートリアルでは、RH850/E1x-FCC1（R7F701Z07）をターゲットとしたマルチコアプロジェクトをCubeSuite+で新規作成してビルドするまでの手順を説明します。

本章では、マルチコアプロジェクトの概要を説明します。マルチコアプロジェクトは、1 つのマルチコア用ブート・ローダプロジェクトと、マイコンに搭載しているCPUコアの個数分のマルチコア用アプリケーションプロジェクトから構成されます。

## 1.1　マルチコアプロプロジェクトの構成

マルチコアプロジェクトを作成する場合、マルチコア用ブート・ローダプロジェクト（以降、ブート・ローダプロジェクト）とマルチコア用アプリケーションプロジェクト（以降、アプリケーションプロジェクト）を作成してください。ブート・ローダプロジェクトではリセットから各アプリケーションプロジェクトに分岐するまでの処理を実行し、アプリケーションプロジェクトではPE（プロセッサ・エレメント）ごとの処理を実行します。

CubeSuite+のプロジェクト・ツリーでは以下のような構成となります。メインプロジェクトとしてブート・ローダプロジェクト、サブプロジェクトとしてPE数分のアプリケーションプロジェクトから構成します。このようなプロジェクト構成とすることにより、一方のPEのみのデバッグや、両PEの同期デバッグが可能となります。



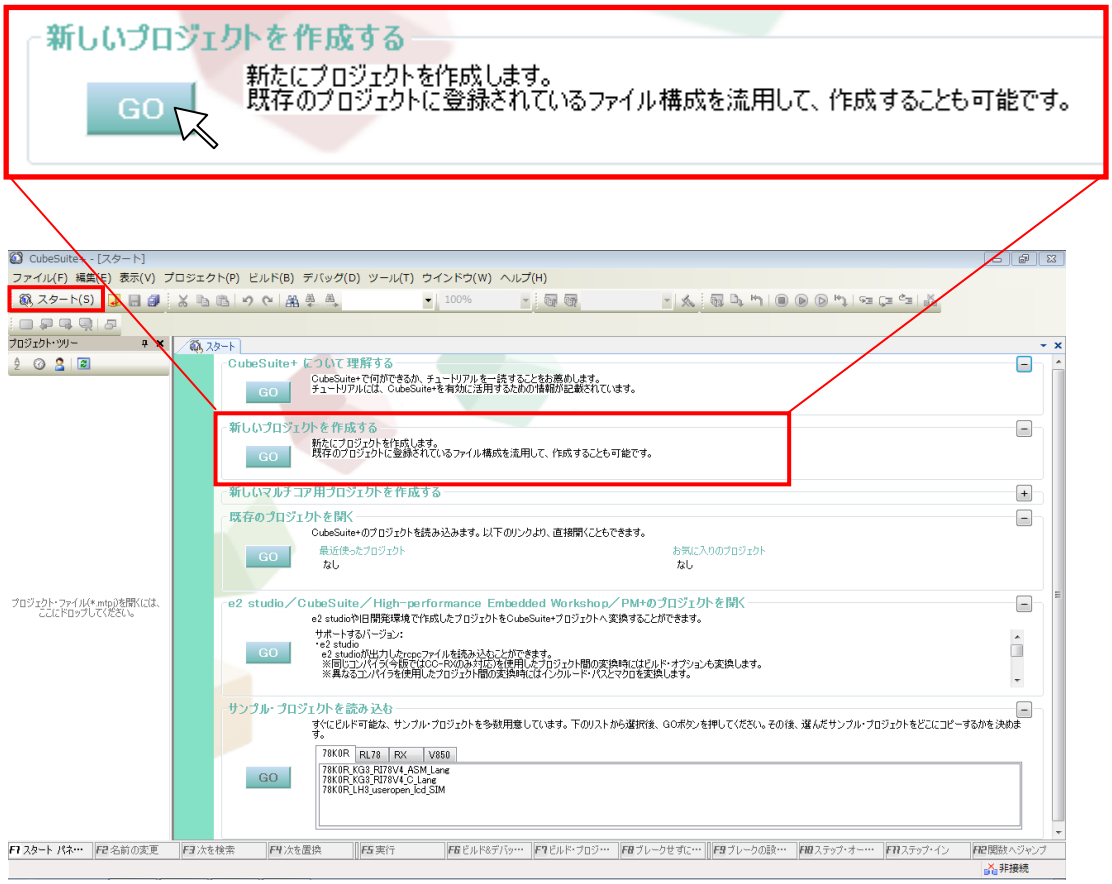
# 第2章　　ブート・ローダプロジェクト

本章では、ブート・ローダプロジェクトの作成方法を説明します。ブート・ローダプロジェクトではリセットから各アプリケーションプロジェクトに分岐するまでの処理を実行します。

## 2.1　新規ブート・ローダプロジェクト作成

以下の手順でブート・ローダプロジェクトを作成します。

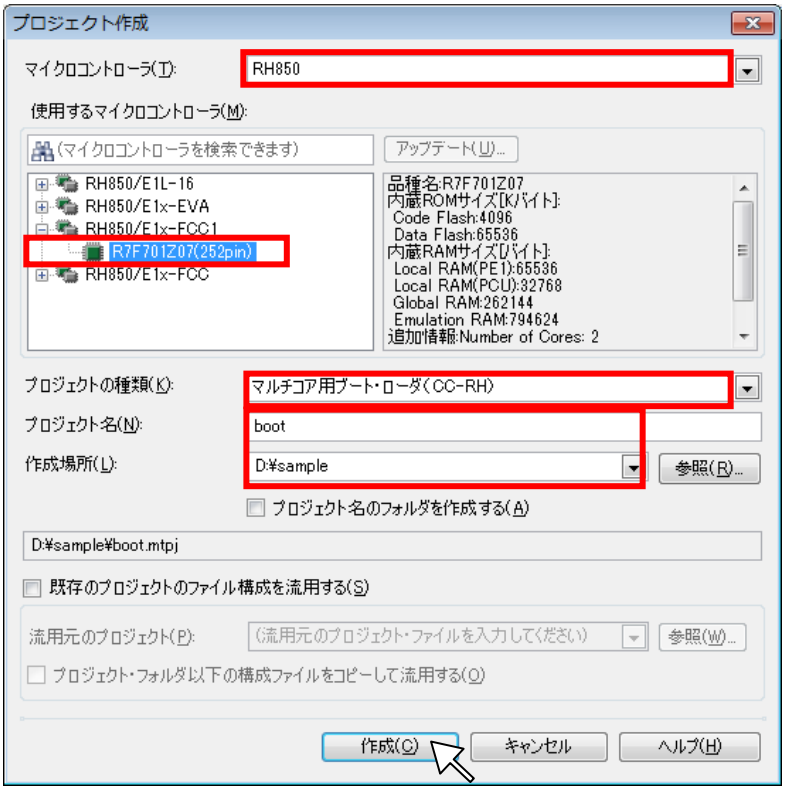
① 新規プロジェクトの作成
CubeSuite+を起動して、[スタート]ボタンをクリックして、スタートパネル上の[新しいプロジェクトを作成する]の[GO]ボタンをクリックしてください。




**備考**　スタートパネルの[新しいマルチコア用プロジェクトを作成する]からプロジェクトを作成することも可能です。この場合、プロジェクト作成ダイアログで[アプリケーション・プロジェクトも同時に作成する]を選択すると、ブート・ローダプロジェクトと、1 つのアプリケーションプロジェクトが作成されます。アプリケーションプロジェクト名は「*ブート・ローダプロジェクト名*_App1」となります。プロジェクト名は変更可能です。

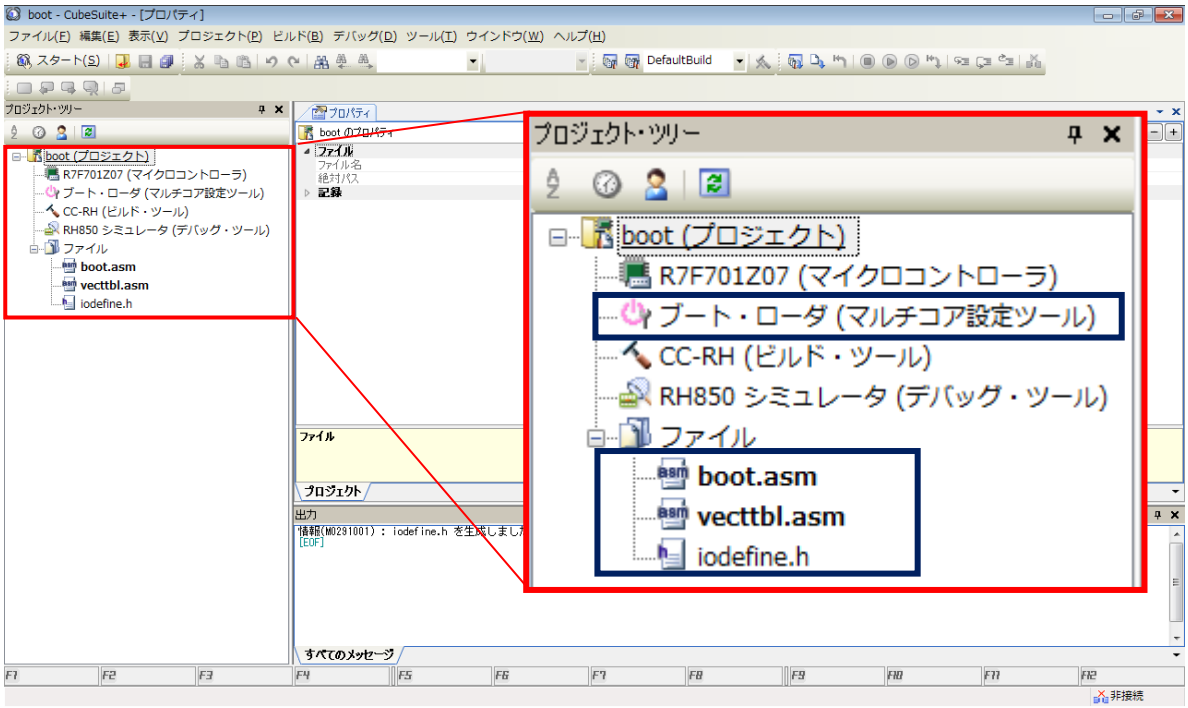
② プロジェクトの設定

プロジェクト作成ダイアログが起ち上がります。本ダイアログで、[マイクロコントローラ]を「RH850」に指定してください。また、[使用するマイクロコントローラ]にて対象マイコンを選択してください。続いて、[プロジェクトの種類]を「マルチコア用ブート・ローダ(CC-RH)」に指定してください。最後にプロジェクト名・作成場所を指定の上、作成ボタンをクリックしてください。




③ ブート・ローダプロジェクトの起動

ブート・ローダプロジェクトが起動します。プロジェクト・ツリーにはブート・ローダ・ノードが表示されます。ファイル・ノードにはboot.asm、vecttbl.asm、iodefine.hが自動で登録されます。



## 2.2 ソース登録

CubeSuite+で新規にマルチコア用ブート・ローダプロジェクト作成時に、プロジェクト・ツリーのファイル・ノードには以下の 3 つのファイルが自動で登録されます。

・ブート・ローダ用スタートアップルーチン（boot.asm）
・例外/割り込みベクタテーブル（vecttbl.asm）
・I/O ヘッダ・ファイル（iodefine.h）

マルチコア用ブート・ローダプロジェクトのソースファイルはboot.asm、cstartm.asm、iodefine.h のみで構成されます。boot.asm、cstartm.asm、iodefine.hについて以降の項で説明します。

### 2.2.1 ブート・ローダ用スタートアップルーチン

ブート・ローダ用スタートアップルーチン（boot.asm）では、リセットから各アプリケーションプロジェクトに分岐するまでの以下の処理を行います。必要に応じてカスタマイズしてください。

#### (1) PE共通のエントリルーチン

複数PEのうち、どのPEで実行されているのか識別するため、PEID（プロセッサ・エレメント番号）を取得します。取得したPEIDに応じて、各PEのエントリルーチンに分岐します。PEIDが 1 の場合、PE1 のエントリルーチン（__start_PE1）に分岐し、PEIDが2の場合、PE2のエントリルーチン（__start_PE2）に分岐し、PEIDが3の場合、PE3 のエントリルーチン（__start_PE3）に分岐します。

```
        ; jump to entry point of each PE
        stsr    0, r10, 2           ; get HTCFG0
        shr     16, r10             ; get PEID

        cmp     1, r10
        bz      __start_PE1
        cmp     2, r10
        bz      __start_PE2
        cmp     3, r10
        bz      __start_PE3
```

#### (2) PE1 のエントリルーチン（__start_PE1）

RAMをクリアするルーチン（_hdwinit_PE1）、EIレベル割り込みをテーブル参照方式に変更するルーチン（_init_eiint）に分岐し、その後、PE1 用アプリケーションプロジェクトへ分岐します。

```
__start_PE1:
        jarl    _hdwinit_PE1, lp            ; initialize hardware
$ifdef USE_TABLE_REFERENCE_METHOD
        jarl    _init_eiint, lp             ; initialize exception
$endif

        mov     #_pm1_setting_table, r13
        ld.w    .OFFSET_ENTRY[r13], r10 ; r10 <- #__start
        jmp     [r10]                       ; jump to #__start
```

_hdwinit_PE1 の呼び出し

ECC機能向けにグローバルRAMとローカルRAM(PE1)をゼロ初期化します。初期化に使用しているシンボル（GLOBAL_RAM_ADDR，GLOBAL_RAM_END，LOCAL_RAM_PE1_ADDR，LOCAL_RAM_PE1_END）はファイル冒頭で定義しています。この値は必要に応じて対象マイコン用のアドレスに変更してください。

```
        ; The following is the addresses in R7F701Z07.
        ; Specify values suitable to your system if needed.
        GLOBAL_RAM_ADDR     .set    0xfeee0000
        GLOBAL_RAM_END      .set    0xfef1ffff

        LOCAL_RAM_PE1_ADDR  .set    0xfedf0000
        LOCAL_RAM_PE1_END   .set    0xfedfffff
```

```
;---------------------------------------------------------------------------
;       hdwinit_PE1
;---------------------------------------------------------------------------
        .section    ".text", text
        .align      2
_hdwinit_PE1:
        mov     lp, r14             ; save return address

        ; clear Global RAM
        mov     GLOBAL_RAM_ADDR, r6
        mov     GLOBAL_RAM_END, r7
        jarl    _zeroclr4, lp

        ; clear Local RAM PE1
        mov     LOCAL_RAM_PE1_ADDR, r6
        mov     LOCAL_RAM_PE1_END, r7
        jarl    _zeroclr4, lp

        mov     r14, lp
        jmp     [lp]
```

```
;---------------------------------------------------------------------------
;       zeroclr4
;---------------------------------------------------------------------------
        .align  2
_zeroclr4:
        br      .L.zeroclr4.2
.L.zeroclr4.1:
        st.w    r0, [r6]
        add     4, r6
.L.zeroclr4.2:
        cmp     r6, r7
        bh      .L.zeroclr4.1
        jmp     [lp]
```

_init_eiintの呼び出し

割り込み優先度 0～2 のEIレベル割り込みを、直接分岐方式からテーブル参照方式に変更します。テーブル参照方式への変更のため、EIレベル割り込み制御レジスタEIC0・EIC1・EIC2 の割り込みベクタ方式選択ビットをセットします。EIC0 のアドレスを値として持つシンボルICBASEを定義し、このICBASEからのオフセットを使用してセットしていますので、必要に応じて対象マイコン用のアドレスに変更してください。なお、別の優先度のEIレベル割り込みをテーブル参照方式に変更する場合、各EIレベル割り込み制御レジスタの割り込みベクタ方式選択ビットをセットしてください。
また、INTBPレジスタにEIINTTBLセクションの先頭アドレスを設定します。EIINTTBLセクションはvecttbl.asmにて定義されています。なお、デフォルトではコメントアウトしていますので、本処理を有効にするにはファイル冒頭で定義しているマクロ"USE_TABLE_REFERENCE_METHOD"を有効にする必要があります。

【無効時】

```
        ; example of using eiint as table reference method
        ;USE_TABLE_REFERENCE_METHOD .set 1
```

【有効時】「;」を削除

```
        ; example of using eiint as table reference method
        USE_TABLE_REFERENCE_METHOD .set 1
```

```
$ifdef USE_TABLE_REFERENCE_METHOD
;-------------------------------------------------------------------------
;       init_eiint
;-------------------------------------------------------------------------
        ; interrupt control register address
        ICBASE  .set    0xfffeea00

        .align  2
_init_eiint:
        mov     #__sEIINTTBL, r10
        ldsr    r10, 4, 1           ; set INTBP

        ; Some interrupt channels use the table reference method.
        mov     ICBASE, r10         ; get interrupt control register address
        set1    6, 0[r10]           ; set INT0 as table reference
        set1    6, 2[r10]           ; set INT1 as table reference
        set1    6, 4[r10]           ; set INT2 as table reference

        jmp     [lp]
$endif
```

PE1 用アプリケーションプロジェクトのエントリルーチンへの分岐

PE1 用アプリケーションプロジェクトで作成するアプリケーション情報テーブル（cstartm.asm の _pm1_setting_table）から、PE1 用アプリケーションプロジェクトのエントリルーチン（__start_pm）のアドレスを読み出して、このエントリルーチンに分岐します。

## (3) PE2 のエントリルーチン（__start_PE2）

RH850/E1x-FCC1（R7F701Z07）はPE2 が実装されていないマイコンのため、__exitルーチンへの分岐で処理を終了します。__exitルーチンは、使用しないPEを待機させておくために自身への分岐を繰り返すルーチンです。

```
__start_PE2:
        br      __exit              ; PE2 does not exist in R7F701Z07
```

```
__exit:
        br      __exit
```

## (4) PE3 のエントリルーチン（__start_PE3）

RAMをクリアするルーチン（_hdwinit_PE3）、EIレベル割り込みをテーブル参照方式に変更するルーチン（_init_eiint）に分岐し、その後、PE3 用アプリケーションプロジェクトへ分岐します。
なお、これらの処理はコメントアウトしており、デフォルトではPE3 を使用しないものとして__exitルーチンへの分岐で処理を終了しています。PE3 を使用する場合はこれらのコメントアウトを解除し、「br　__exit」をコメントアウトしてください。

```
__start_PE3:
;       jarl    _hdwinit_PE3, lp    ; initialize hardware
;$ifdef USE_TABLE_REFERENCE_METHOD
;       jarl    _init_eiint, lp             ; initialize exception
;$endif
;       mov     #_pm3_setting_table, r13
;       ld.w    .OFFSET_ENTRY[r13], r10 ; r10 <- #__start
;       jmp     [r10]                       ; jump to #__start

        br      __exit
```

### _hdwinit_PE3 の呼び出し

ECC 機能向けにローカル RAM(PE3)をゼロ初期化します。初期化に使用しているシンボル（LOCAL_RAM_PE3_ADDR, LOCAL_RAM_PE3_END）はファイル冒頭で定義しています。この値は必要に応じて対象マイコン用のアドレスに変更してください。

```
        LOCAL_RAM_PE3_ADDR  .set    0xfedf8000
        LOCAL_RAM_PE3_END   .set    0xfedfffff
```

### _init_eiintの呼び出し

PE1 と同じ_init_eiintを呼び出します。本処理を有効にするにはファイル冒頭で定義しているマクロ「USE_TABLE_REFERENCE_METHOD」を有効にする必要があります。

### PE3 用アプリケーションプロジェクトのエントリルーチンへの分岐

PE1 と同じくPE3 用アプリケーションプロジェクトを作成し、アプリケーション情報テーブル（cstartm.asmの_pm3_setting_table）から、PE3 用アプリケーションプロジェクトのエントリルーチン（__start_pm）のアドレスを読み出して、このエントリルーチンに分岐します。
なお、cstartm.asmはPE1 用のため、PE3 用にアプリケーション情報テーブル名を_pm1_setting_tableから_pm3_setting_tableに変更する必要があります。

### 2.2.2 例外/割り込みベクタテーブル

リセット処理や例外/割り込み処理は、ベクタ方式で実行します。ハンドラ・アドレスは、直接ベクタ方式、及びテーブル参照方式の2種類の方式で指定することができます。使用割り込みチャネルごとの例外ハンドラ・アドレス選択方式の設定については、各製品搭載の割り込みコントローラ仕様を参照してください。

CubeSuite+で新規プロジェクト作成時に、プロジェクト・ツリーのファイル・ノードに自動で登録される例外/割り込みベクタテーブル（vecttbl.asm）について以下で説明します。必要に応じてカスタマイズしてください。

#### (1) RESET

RESETのアドレスは、RBASEレジスタで示されるベース・アドレスに、例外要因のオフセット（RESETの例外要因オフセットは0）を加算した値を使用します。RH850/E1x-FCC1（R7F701Z07）のRESETハンドラ・アドレスは、ユーザ領域から起動する場合は0x00000000番地、ユーザブート領域から起動する場合は、0x01000000番地です。

```
.section "RESET", text
.align    512
jr32      __start ; RESET
```

上記の定義により、RESETセクションの先頭に「jr32 __start」が埋め込まれます。
CubeSuite+で新規プロジェクトを作成した場合、リンカオプション”-start”によりRESETセクションは 0x01000000 番地に配置指定されています。

#### (2) 直接ベクタ方式の例外/割り込み

直接ベクタ方式の場合は、割り込み優先度に従って固定のハンドラ・アドレスへ分岐します。ハンドラ・アドレスの基準位置は、RBASEレジスタ、またはEBASEレジスタで示されるベース・アドレスに例外要因のオフセットを加算した値を使用します。いずれをベース・アドレスとして利用するかは、PSW.EBVビットによって選択します。
CubeSuite+で新規プロジェクトを作成した場合、RBASEをベース・アドレスであるものとして、RESETセクションの直後から割り込み/例外ハンドラを配置しています。

```
.section "RESET", text
.align    512
jr32      __start ; RESET

.align    16                    <-- RESET の直後から配置
jr32      _Dummy ; SYSERR

.align    16
jr32      _Dummy ; HVTRAP
...
```

デフォルトでは、SYSERR/HVTRAP/FETRAP等の対応するオフセット位置に、ダミー関数_Dummyに分岐させる命令を配置しています。_Dummyは自分自身への分岐を繰り返すルーチンであり、vecttbl.asm内で定義しています。必要に応じてカスタマイズしてください。

カスタマイズする例外/割り込みに対応するオフセット位置の「__Dummy」を「_割り込み関数名」に変更してください。また、割り込み関数を定義してください。Cソースファイル上で割り込み関数を定義する場合は、#pragma interrupt 指令にて指定してください。詳細な記述方法につきましては、コーディング編マニュアルをご参照ください。

【記述例】例外”SYSERR”発生時に割り込み関数”func1”を実行する場合

```
        .section "RESET", text
        .align    512
        jr32      __start ; RESET
                                  「__Dummy」を「_割り込み関数名」に変更
        .align    16
        jr32      _func1 ; SYSERR

        .align    16
        jr32      _Dummy ; HVTRAP

        .align    16
        jr32      _Dummy ; FETRAP

        …
```

```
#pragma interrupt func1(priority=SYSERR, callt=true, fpu=true)
void func1(unsigned long feic)
{
        …
}
```

## (3) テーブル参照方式の例外/割り込み

RH850の場合、割り込みの拡張仕様として、テーブル参照方式による割り込みを指定できます。直接ベクタ方式では、EIレベル割り込みのハンドラ・アドレスはそれぞれの割り込み優先度ごとに1つであり、複数の同一優先度を示す割り込みチャネルは同じ割り込みハンドラ・アドレスへ分岐します。しかし、アプリケーション上、割り込みハンドラごとに異なるコード領域を利用したい場合などがあります。RH850では、このような使用方法を想定した割り込みに関するテーブル参照方式を定義しています。

CubeSuite+で新規プロジェクトを作成した場合、例外/割り込みテーブルをEIINTTBLセクションとして、EIINTTBLセクションの先頭から4の倍数の領域にダミー関数_Dummy_EIの配置アドレスが埋め込まれています。これにより、割り込み優先度n（nは 0 から 512）のテーブル参照方式の例外/割り込みが発生した場合、_Dummy_EIに分岐します。_Dummy_EIは自分自身への分岐を繰り返すルーチンであり、vecttbl.asm内で定義しています。必要に応じてカスタマイズしてください。

```
        .section "EIINTTBL", const
        .align    512
        .dw       #_Dummy_EI ; INT0
        .dw       #_Dummy_EI ; INT1
        .dw       #_Dummy_EI ; INT2
        .rept     512 - 3
        .dw       #_Dummy_EI ; INTn
        .endm
```

CubeSuite+で新規プロジェクトを作成した場合、リンカオプション”-start”によりEIINTTBLセクションは0x00番地に配置指定されていますので、必要に応じて配置アドレスを指定してください。

カスタマイズする例外/割り込みに対応するオフセット位置の「#_Dummy_EI」を「#_割り込み関数名」に変更してください。また、割り込み関数を定義してください。Cソースファイル上で割り込み関数を定義する場合は、#pragma interrupt指令にて指定してください。詳細な記述方法につきましては、コーディング編マニュアルをご参照ください。

【記述例】EIINT 割り込みチャネル 9”EIINT9”発生時に割り込み関数”func2”を実行する場合

```
        .section "EIINTTBL", const
        .align    512
        .dw       #_Dummy_EI ; INT0
        .dw       #_Dummy_EI ; INT1
        .dw       #_Dummy_EI ; INT2
        .dw       #_Dummy_EI ; INT3
        .dw       #_Dummy_EI ; INT4
        .dw       #_Dummy_EI ; INT5
        .dw       #_Dummy_EI ; INT6
        .dw       #_Dummy_EI ; INT7
        .dw       #_Dummy_EI ; INT8
        .dw       #_func2          ; INT9
        .rept     512 - 9
        .dw       #_Dummy_EI ; INTn
```

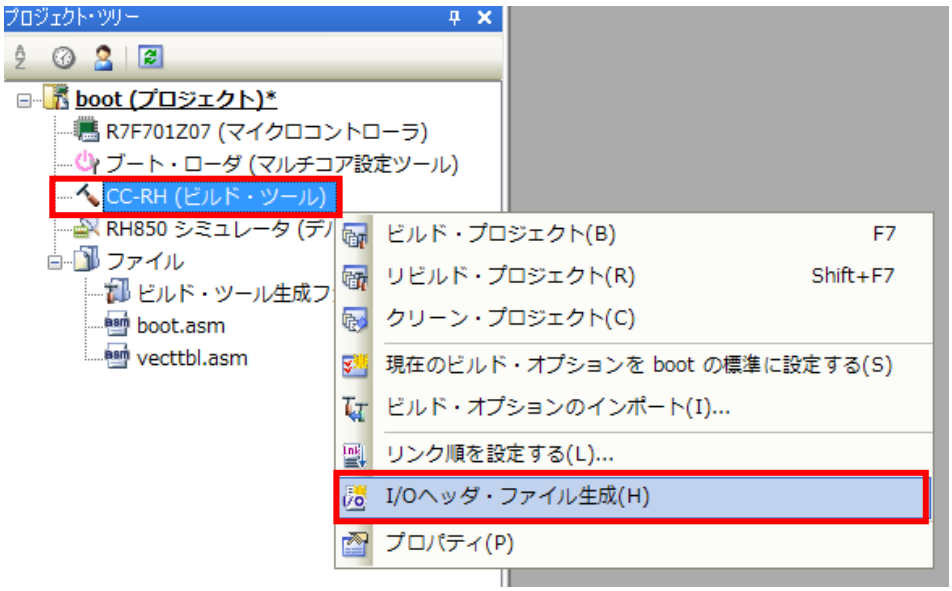
「#_Dummy_EI」を
「#_割り込み関数名」に変更

```
#pragma interrupt func2(channel=9, enable=true, callt=true, fpu=true)

void func2(unsigned long eiic)
{
        ...
}
```

### 2.2.3 I/Oヘッダ・ファイル

新規ブート・ローダプロジェクト作成時に、プロジェクトで指定している該当マイコン用のI/Oヘッダ・ファイル(iodefine.h)を生成し、プロジェクトに自動で登録します。I/Oヘッダ・ファイルでは、マイコンが持つレジスタ名と、そのアドレスが定義されています。ブート・ローダプロジェクトでこのファイルを使用しないのであればプロジェクトから外してください。

CubeSuite+プロジェクト・ツリーの[CC-RH(ビルド・ツール)]ノードを右クリック=> [I/Oヘッダ・ファイル生成]を押下して生成させることも可能です。

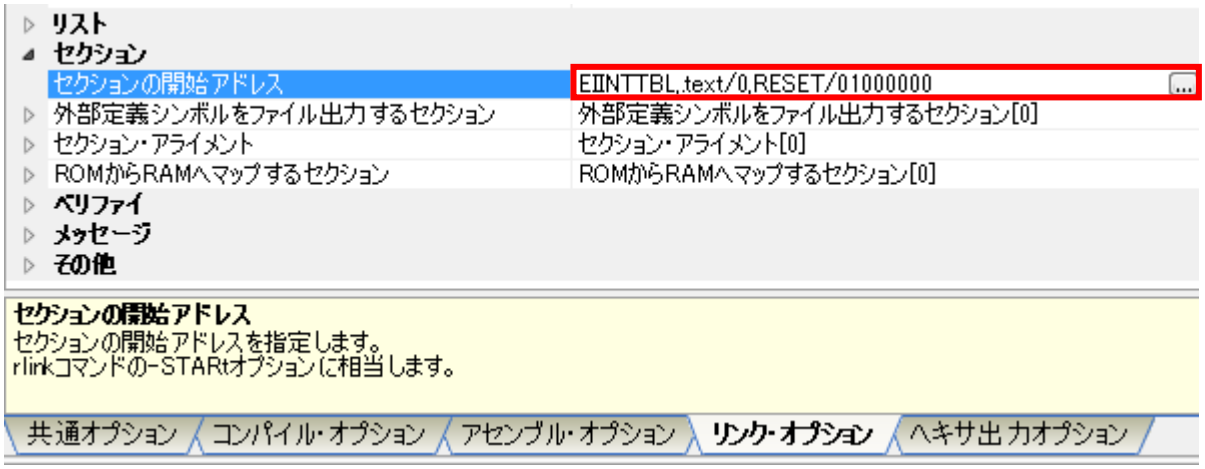


## 2.3 オプション設定

ブート・ローダプロジェクトを作成する上で、特に使用するオプションについて説明します。

### 2.3.1 リンクオプション

セクションの開始アドレスを[リンク・オプション]タブ=> [セクション]カテゴリ=> [セクションの開始アドレス]プロパティで指定してください。デフォルトでは以下のように指定されています。この文字列はリンクオプション"-start"の引数としてリンカに渡されます。




[セクションの開始アドレス]プロパティの右端の[...]ボタンを押下すると、以下のようなセクション設定ダイアログが起ち上がります。このダイアログから指定することも可能です。


このセクション設定によって、0x00000000 番地から上位方向にEIINTTBL=>.textセクションを配置し、0x01000000番地からRESETセクションを配置します。本ダイアログ上で、所望のアドレス配置となるようにカスタマイズしてください。

# 第3章　　アプリケーションプロジェクト

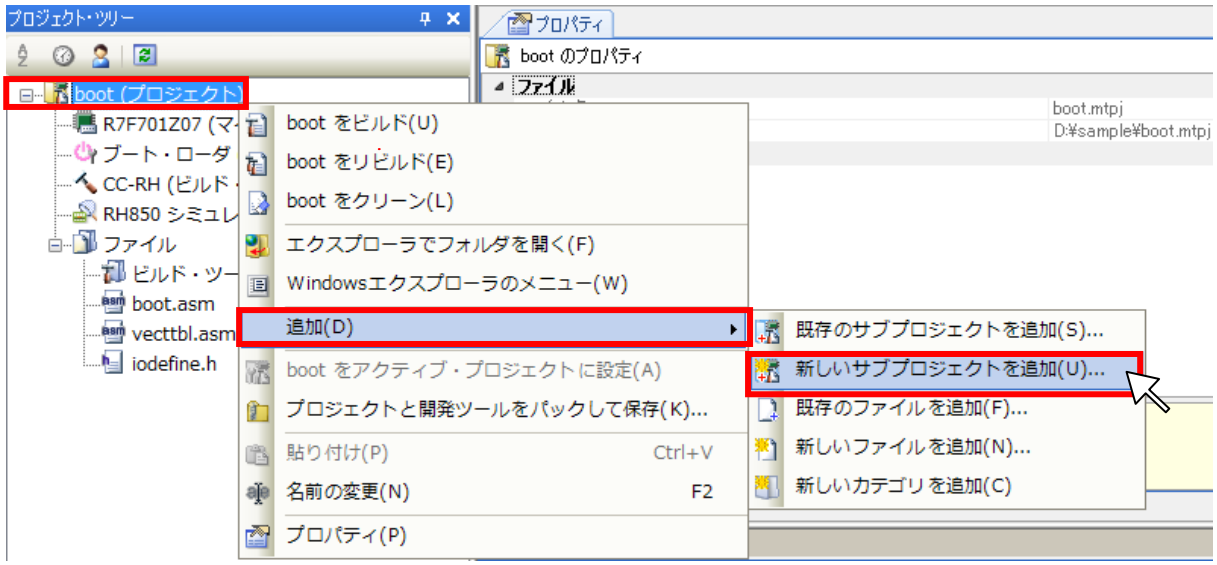
本章では、マルチコア用アプリケーションプロジェクトの作成方法を説明します。アプリケーションプロジェクトでは、PEごとの処理を実行します。

メインCPU用プロジェクト名をPE1、PCU用プロジェクト名をPE3 として 2 つのアプリケーションプロジェクトを作成します。新規プロジェクトの作成から、オプション設定、ビルド方法に至るまで一通りの手順を説明します。

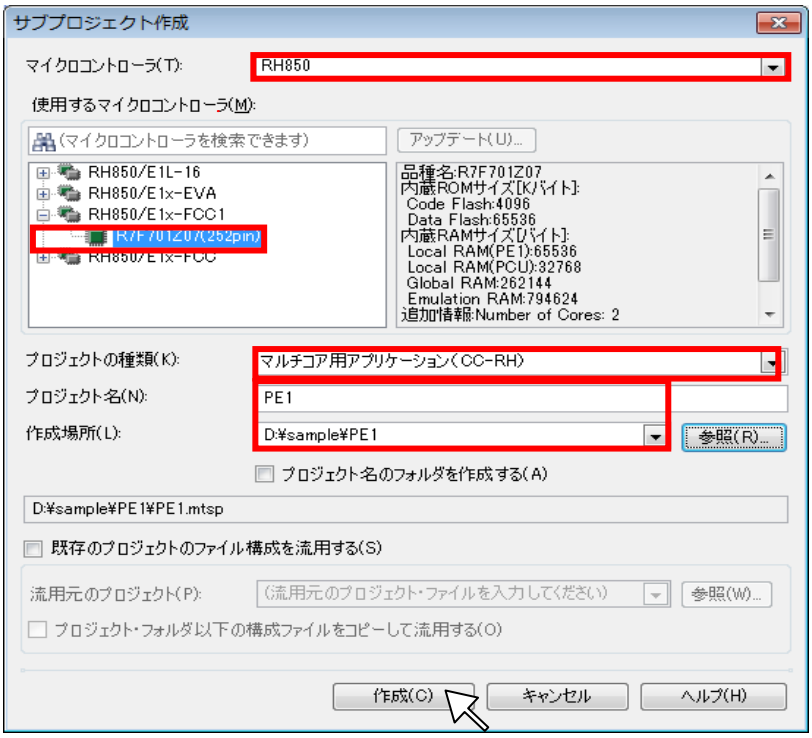
## 3.1　新規アプリケーションプロジェクト作成

ブート・ローダプロジェクトが作成されている場合に、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を作成する手順を説明します。RH850/E1x-FCC1 はデュアルコアですので、2 つのアプリケーションプロジェクトを作成します。

① サブプロジェクトの追加
メインプロジェクトであるブート・ローダプロジェクトにサブプロジェクトを追加します。プロジェクト・ツリーの[プロジェクト]ノードを右クリック=> [追加]=> [新しいサブプロジェクトを追加]から追加してください。既に作成済みのサブプロジェクトを追加する場合は[既存のサブプロジェクトを追加]から追加してください。



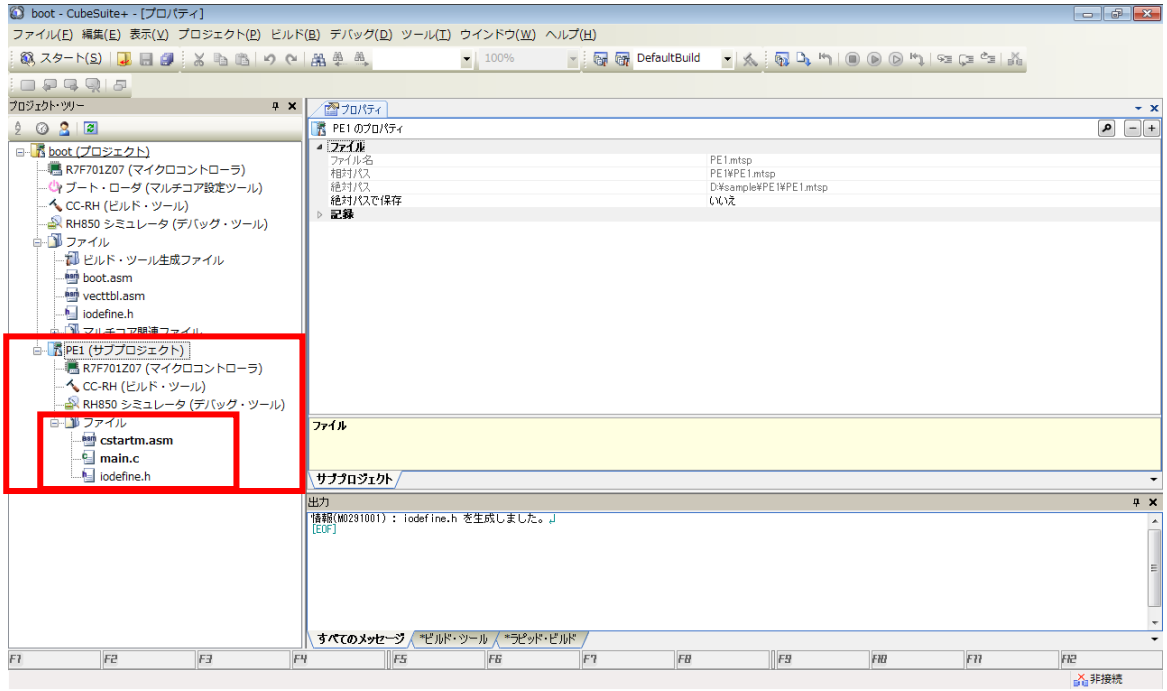

② サブプロジェクトの設定
サブプロジェクト作成ダイアログが起ち上がります。
[使用するマイクロコントローラ]では、ブート・ローダプロジェクトと同じマイクロコントローラを指定してください。続いて、プロジェクトの種類を「マルチコア用アプリケーション(CC-RH)」に指定してください。最後にプロジェクト名・作成場所を指定の上、作成ボタンをクリックしてください。
ここでは、メインCPU用のプロジェクトであるものとして、プロジェクト名をPE1とします。




※　スタートパネルの「新しいマルチコア用プロジェクトを作成する」からプロジェクトを作成した場合、ブート・ローダプロジェクトと、1 つのマルチコア用アプリケーションプロジェクトがサブプロジェクトとして作成されます。

③　アプリケーションプロジェクトの登録

ブート・ローダプロジェクトの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が登録されます。サブプロジェクトのプロジェクト・ツリーのファイル・ノードにはcstartm.asm、main.c、iodefine.hが自動で登録されています。




同様の手順で、PCU用のプロジェクトとしてサブプロジェクトをもう1つ追加してください。ここではプロジェクト名をPE3とします。

## 3.2 ソース登録

CubeSuite+で新規にマルチコア用アプリケーションプロジェクト作成時に、プロジェクト・ツリーのファイル・ノードには以下の 3 つのファイルが自動で登録されます。

・アプリケーション用スタートアップルーチン（cstartm.asm）
・空の main 関数（main.c）
・I/O ヘッダ・ファイル（iodefine.h）

プロジェクト･ツリーのファイル･ノードに、必要なソースファイルを登録してください。ドラッグ &ドロップ、あるいはファイル･ノードを右クリック=>[追加]から登録できます。アプリケーション用スタートアップルーチン（cstartm.asm）とI/Oヘッダ・ファイルについて以降の項で説明します。

### 3.2.1 アプリケーション用スタートアップルーチン

アプリケーション用スタートアップルーチン（cstartm.asm）では、PE ごとのスタートアップ処理を行います。必要に応じてカスタマイズしてください。

#### (1) アプリケーション情報テーブルの定義

PE1 用のアプリケーション情報テーブル（_pm1_setting_table）を定義しています。このアプリケーション情報テーブルによって、ブート･ローダ用スタートアップルーチン（boot.asm）の__start_PE1 内から分岐するPE1 用アプリケーションプロジェクトのエントリルーチン（__start_pm）を設定します。

```
;-----------------------------------------------------------------------
;           processing module setting table
;-----------------------------------------------------------------------
            .section   ".const.cmn", const
            .public    _pm1_setting_table
            .align     4
_pm1_setting_table:
            .dw        #__start_pm        ; ENTRY ADDRESS
```

【参考】 boot.asm の__start_PE1

```
__start_PE1:
            …
            mov        #_pm1_setting_table, r13
            ld.w       .OFFSET_ENTRY[r13], r10  ; r10 <- #__start_pm
            jmp        [r10]                    ; jump to #__start_pm
```

アプリケーション情報テーブルは.const.cmnセクションに配置されていますので、-fsymbolオプションの引数に.const.cmnを指定することにより、_pm1_setting_tableのアドレスをブート･ローダプロジェクトに渡すことができます。ブート･ローダプロジェクトでは、_pm1_setting_tableに配置されたPE1 用アプリケーションプロジェクトのエントリルーチンのアドレス（#__start_pm）をロードし、このアドレスに分岐します。

PE3 用アプリケーションプロジェクトの場合は、アプリケーション情報テーブル名を_pm1_setting_table から _pm3_setting_tableに変更してください。

```
            .section   ".const.cmn", const
            .public    _pm3_setting_table
            .align     4
_pm3_setting_table:
            .dw        #__start_pm        ; ENTRY ADDRESS
```

## (2) スタック領域の確保

各PE毎に、コンパイラ生成コードが利用するスタック領域を 0x200 バイト分確保します。スタック領域は.stack.bssセクションに配置されています。

```
;------------------------------------------------------------------------
;               system stack
;------------------------------------------------------------------------
STACKSIZE       .set        0x200
                .section    ".stack.bss", bss
                .align      4
                .ds         (STACKSIZE)
                .align      4
_stacktop:
```

## (3) RAMセクション初期化テーブルの定義

RAMセクションの初期値コピーとゼロクリアを行う関数”_INITSCT_RH”の引数に指定するテーブルを定義しています。テーブルにはセクションの先頭ラベルのアドレス値(#__sセクション名)と終端ラベルのアドレス値(#__eセクション名)を利用します。デフォルトでは、初期値付き変数のセクションが.dataセクション、初期値なし変数のセクションが.bssセクションに配置されており、-rom=.data=.data.Rが指定されている場合のテーブルです。

```
;------------------------------------------------------------------------
;               section initialize table
;------------------------------------------------------------------------
                .section    ".INIT_DSEC.const", const
                .align      4
                .dw         #__s.data,          #__e.data,          #__s.data.R

                .section    ".INIT_BSEC.const", const
                .align      4
                .dw         #__s.bss,           #__e.bss
```

RAMセクションを新たに追加した場合は、追加したセクションをテーブルに定義してください。テーブルに追加したセクションも”_INITSCT_RH”関数によるコピーとゼロクリアの対象となります。

【記述例】.sdata セクションと.sbss セクションを追加した場合（-rom=.sdata=.sdata.R 指定時）

```
;------------------------------------------------------------------------
;               section initialize table
;------------------------------------------------------------------------
                .section    ".INIT_DSEC.const", const
                .align      4
                .dw         #__s.data,          #__e.data,          #__s.data.R
                .dw         #__s.sdata,         #__e.sdata,         #__s.sdata.R


                .section    ".INIT_BSEC.const", const
                .align      4
                .dw         #__s.bss,           #__e.bss
                .dw         #__s.sbss,          #__e.sbss
```

## (4) アプリケーションプロジェクトのエントリルーチン（__start_pm）

ブート・ローダ用スタートアップルーチン（boot.asm）の__start_PE1 内から分岐してくるアプリケーションプロジェクト用のエントリルーチンです。main関数に分岐するまでの以下の処理を行います。必要に応じてカスタマイズしてください。

### レジスタの設定

SP/GP/EPレジスタに値を設定します。

```
;-----------------------------------------------------------------------
;       startup
;-----------------------------------------------------------------------
        .section  ".text", text
        .public   __start_pm
        .align    2
__start_pm:
        mov       #_stacktop, sp            ;   set sp register
        mov       #__gp_data, gp            ;   set gp register
        mov       #__ep_data, ep            ;   set ep register
```

### _hdwinitの呼び出し

必要に応じてhdwinit関数を定義して周辺装置の初期化処理を行ってください。この定義が無い場合は、標準ライブラリ（libc.lib）中の空のhdwinit関数をリンクして呼び出します。

### __INITSCT_RHの呼び出し

RAMセクション初期化テーブルで指定されたセクションのコピーとゼロクリアを行います。

```
        mov       #__s.INIT_DSEC.const, r6
        mov       #__e.INIT_DSEC.const, r7
        mov       #__s.INIT_BSEC.const, r8
        mov       #__e.INIT_BSEC.const, r9
        jarl32    __INITSCT_RH, lp          ;   initialize RAM area
```

### FPUの設定

PSW.CU0 をセットしてFPUの使用を許可します。また、FPU 機能レジスタ(FPSR・FPEPC)を初期化します。コプロセッサとしてFPUを実装していないCPUのスタートアップルーチンとして使用する場合は、この処理削除してください。

```
        ; set various flags to PSW via FEPSW

        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
```

main関数への遷移

以下の2つの処理はコメントアウトしています。いずれもFEPSWレジスタ値を設定する処理で、feret命令の実行によってPSWに反映される値となります。必要に応じてコメントを削除してこの処理を有効にしてください。

- PSW.IDをクリアして割り込みを有効 ※リセット後のPSW.IDは1のため
- PSW.UMをセットしてSV(スーパーバイザモード)からUM(ユーザモード)へ遷移

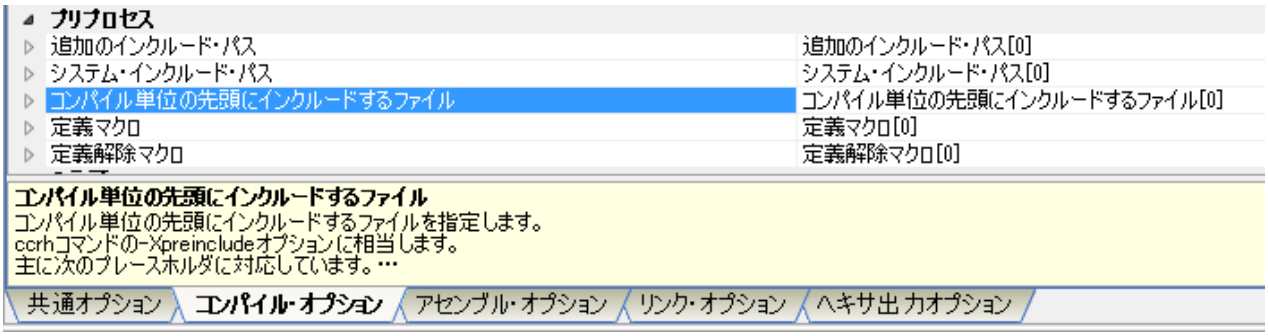
また、自身への分岐を繰り返すルーチン_exitのアドレス(#_exit)をlpに、PE1用main関数の先頭アドレス(#_main)をFEPCレジスタに設定します。その後、feret命令の実行によって、FEPSWレジスタ値をPSWに、FEPCレジスタ値をPCに反映し、main関数へ遷移します。

```
        ;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
        ldsr     r10, 2, 0          ; FEPC <- #_main

        ; apply PSW and PC to start user mode
        ferret

_exit:
        br       _exit                      ;   end of program
```

### 3.2.2 I/Oヘッダ・ファイル

新規アプリケーションプロジェクト作成時に、プロジェクトで指定している該当マイコン用のI/Oヘッダ・ファイル(iodefine.h)を生成し、プロジェクトに自動で登録します。I/Oヘッダ・ファイルでは、マイコンが持つレジスタ名と、そのアドレスが定義されています。

プログラム中でI/Oをアクセスする場合、I/Oヘッダ・ファイルをインクルードしてください。なお、-Xpreincludeオプションの引数に本ファイルを指定することにより、ソースファイル中に#include指定する必要がなくなります。-Xpreincludeオプションは、[コンパイル・オプション]タブ=> [プリプロセス]カテゴリ=> [コンパイル単位の先頭にインクルードするファイル]にて指定可能です。本プロパティで該当マイコン用のI/Oヘッダ・ファイルを指定してください。



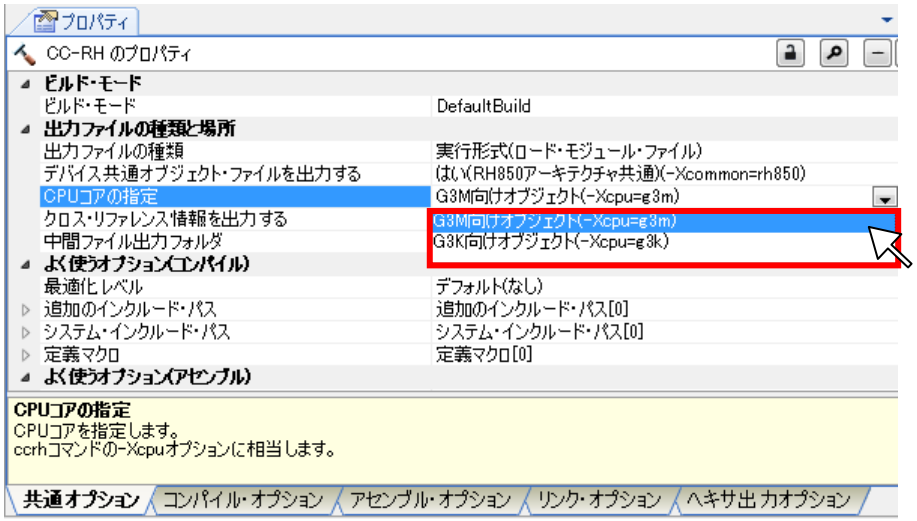
## 3.3 オプション設定

アプリケーションプロジェクトを作成する上で、特に使用するオプションについて説明します。

### 3.3.1 コンパイラオプション

**-Xcpuオプション**

CPUコアを指定するオプションです。指定したコア向けのオブジェクトを生成します。[共通オプション]タブ=> [出力ファイルの種類と場所]カテゴリ=> [CPUコアの指定]にて[G3M向けオブジェクト]／[G3K向けオブジェクト]のいずれかを指定してください。デフォルトでは[G3K向けオブジェクト]が指定されています。

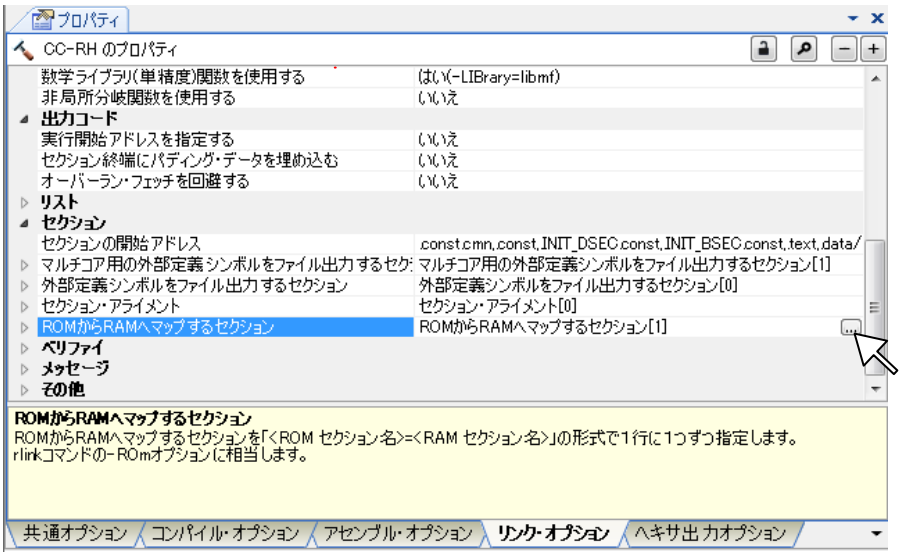



メ インCPUプロジェクトの場合は[G3M向けオブジェクト]を、PCU用プロジェクトの場合は[G3K向けオブジェクト]を選択してください。メインCPUとPCUの両方から実行される関数を含むファイルをコンパイルする場合は[G3K向けオブジェクト]を選択してください。

### 3.3.2　リンクオプション

**-romオプション**

初期値付き変数が配置されるセクションは、リセット時にはROMに配置しておき、実行時にはRAMにコピーされている必要があります。この処理をROM化といいます。-romオプションは、ROM化によってROMからRAMへマップするセクションを指定するオプションです。[リンク・オプション]タブ=> [セクション]カテゴリ=> [ROMからRAMへマップするセクション]にて右端の[...]ボタンを押下し、ROMからRAMへマップするセクションを「*<ROMセクション名>=<RAMセクション名>*」の形式で、1行に1つずつ指定してください。



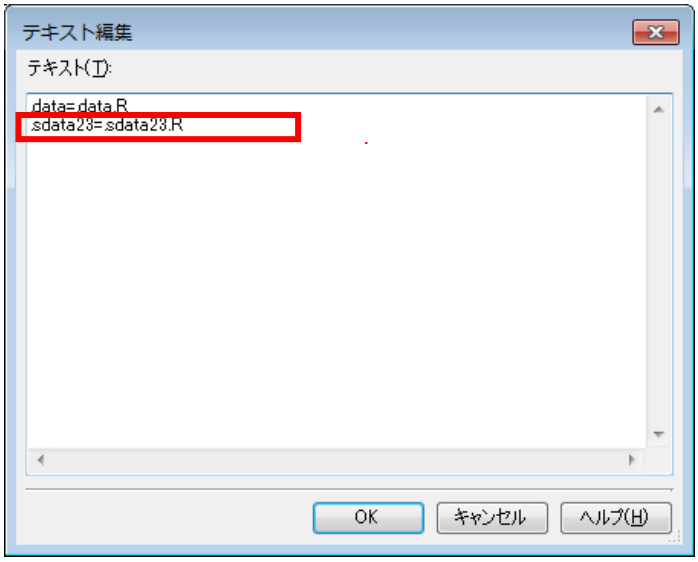

*<ROMセクション名>*がROM化対象のセクションとなります。-startオプションによって、*<ROMセクション名>*で指定したセクションはROMに配置指定し、*<RAMセクション名>*で指定したセクションはRAMに配置指定してください。

デフォルトでは以下が指定されています。

```
.data=.data.R
```

.dataセクション以外にROM化が必要なセクションを追加した場合は、本オプションを追加指定してください。

【例】.sdata23 を追加して ROM 化対象とする場合（-rom=.sdata23 =.sdata23.R 指定）



また、スタートアップルーチンcstartm.asm のセクション初期化テーブル(.INIT_DSEC.const)にも、追加したセクションの初期化テーブルを追記してください。セクション名の頭に"__s"を付けることで、そのセクションの先頭アドレスを値として持つ予約シンボルとなります。同様にセクション名の頭に"__e"を付けることで、そのセクションの終端アドレスを値として持つ予約シンボルとなります。初期化テーブルへの追加はこの予約シンボルを使用頂くことを推奨します。

【例】.sdata23 を追加した場合 (-rom=.sdata23=.sdata23.R 指定時)

```
;------------------------------------------------------------------------
;               section initialize table
;------------------------------------------------------------------------
                .section   ".INIT_DSEC.const", const
                .align     4
                .dw        #__s.data,          #__e.data,          #__s.data.R
                .dw        #__s.sdata23,       #__e.sdata23,       #__s.sdata23.R
```

ROM セクションの先頭アドレス　ROM セクションの終端アドレス　RAM セクションの先頭アドレス

また同様に、初期値なし変数が配置されるセクションにつきましても、セクションを追加した場合にはセクション初期化テーブル(.INIT_BSEC.const)に追加してください。

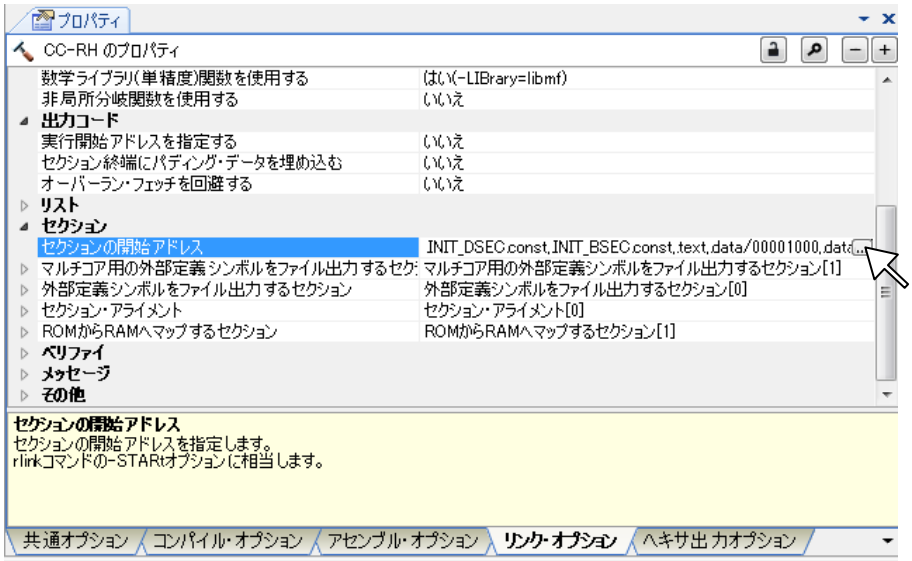
【例】.sbss23 を新たに追加した場合

```
                .section   ".INIT_BSEC.const", const
                .align     4
                .dw        #__s.bss,           #__e.bss
                .dw        #__s.sbss23,        #__e.sbss23
```

RAM セクションの先頭アドレス　RAM セクションの終端アドレス

## -startオプション

セクションの開始アドレスを指定するオプションです。[リンク・オプション]タブ=> [セクション]カテゴリ=> [セクションの開始アドレス] にて指定してください。

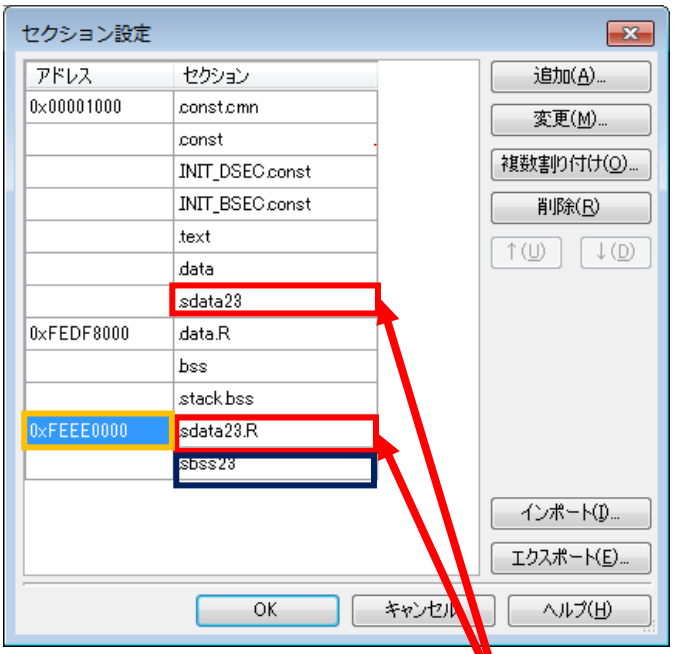


右端の[...]ボタンを押下すると、セクション設定ダイアログが起ち上がります。デフォルトでは以下のように指定されています。


この指定は 0x1000 番地から上位方向に.const.cmn -> .const -> INIT_DSEC.const -> INIT_BSEC.const -> .text -> .data セクションを配置し、0xFEDF8000 番地から上位方向に.data.R -> .bss -> .stack.bssセクションを配置させる指定です。0xFEDF8000 番地はPCUのローカルRAMセルフ領域の先頭アドレスを想定しております。本ダイアログ上で、所望のアドレス配置となるようにカスタマイズしてください。例えば、メインCPU用のプロジェクトであれば、ローカルRAMセルフ領域は0xFEDF0000番地からですので、0xFEDF0000番地に変更しますと、RAM領域を効率的に使用出来ます。

デフォルトで指定されているセクション以外にセクションを追加した場合は、本オプションを追加指定してください。

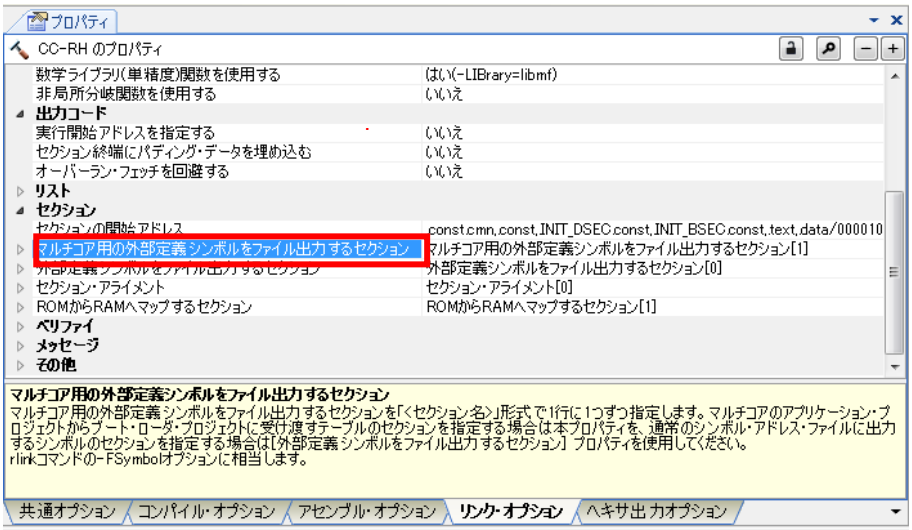
【例】.sdata23／.sbss23 を新たに追加し（-rom=.sdata23=.sdata23.R 指定）、これらの変数をグローバル RAM 領域である 0xFEEE0000 番地に配置指定する場合




-rom オプションの引数<*ROM セクション名*>で指定した.sdata23 は ROM 領域に、<*RAM セクション名*>で指定した.sdata23.R は RAM 領域に配置指定

## -fsymbolオプション

外部定義シンボルをシンボル・アドレス・ファイル(*.fsy)に出力させるオプションです。本オプションの引数として指定したセクション内の外部定義シンボルをアセンブラ制御命令形式で*.fsyファイルに出力します。[リンク・オプション]タブ=> [セクション]カテゴリ=> [マルチコア用の外部定義シンボルをファイル出力するセクション]にて指定してください。




[マルチコア用の外部定義シンボルをファイル出力するセクション]では、アプリケーションプロジェクト間、あるいはブート・ローダプロジェクトとアプリケーションプロジェクト間で共有する外部シンボルが含まれるセクション名を指定してください。

デフォルトでは.const.cmnセクションが指定されています。つまり、外部定義シンボル_pm1_setting_table のアドレスが.fsyファイルに出力されます。.fsyファイルとは、外部定義シンボルをアセンブラ制御命令で記述したアセンブリ・ソース・ファイルです。この外部定義シンボルを共有したいプロジェクトに.fsyファイルを入力して一緒にビルドすることにより、外部定義シンボルをプロジェクト間で共有することが可能となります。

【デフォルトの cstartm.asm】

```
;------------------------------------------------------------------------
;               processing module setting table
;------------------------------------------------------------------------
                .section   ".const.cmn", const
                .public    _pm1_setting_table
                .align     4
_pm1_setting_table:
                .dw        #__start  ; ENTRY ADDRESS
```

【*.fsy ファイルの出力例】

```
;SECTION NAME = .const.cmn
 .public _pm1_setting_table
_pm1_setting_table .equ 0x1000
```

外部定義シンボル名　　配置アドレス

この*.fsyファイルをブート・ローダプロジェクトに入力することにより、_pm1_setting_tableをブート・ローダプロジェクトからも参照することが可能となります。ブート・ローダプロジェクトのboot.asm の以下の記述により、0x1000番地の値(__start のアドレス)に分岐します。

```
        .OFFSET_ENTRY .set      0
        …
        mov     #_pm1_setting_table, r13
        ld.w    .OFFSET_ENTRY[r13], r10 ; r10 <- #__start
        jmp     [r10]                   ; jump to #__start
```

.const.cmnセクションの外部シンボル以外にもプロジェクト間で共有する外部シンボルがある場合は、その外部シンボルが含まれるセクション名を追加で指定してください。

## 3.4 変数の共有

シンボル・アドレス・ファイル(*.fsy)を使用することで、変数をプロジェクト間で共有させることが可能です。つまり、コア間で変数を共有させることが可能です。

本節では、初期値付き変数val1と初期値なし変数val2をプロジェクトPE1で定義し、プロジェクトPE3から参照する方法を説明します。

① セクション名変更
デフォルトでは初期値付き変数は.dataセクションに、初期値なし変数は.bssセクションに配置されます。プロジェクトPE1にソース登録されているCソースファイルにて、プロジェクトPE3と共有する変数のセクション名を変更してください。

【C ソースファイルの記述例】

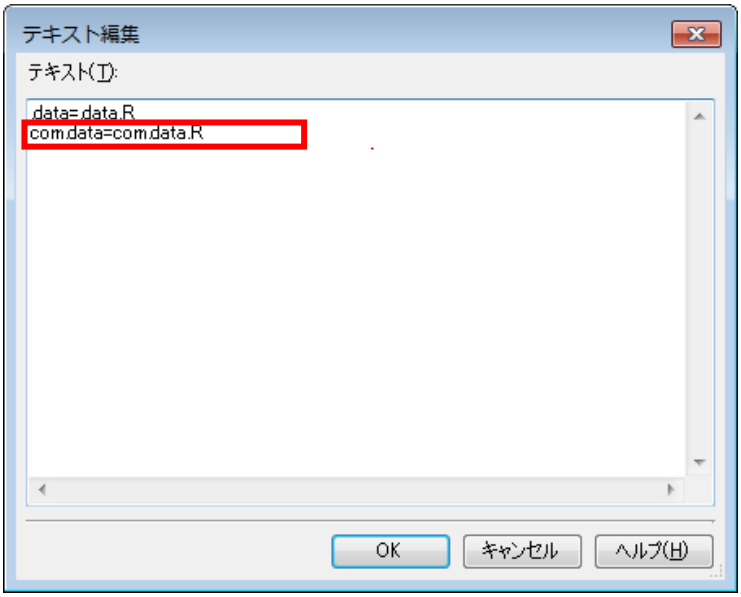
```
#pragma section r0_disp32 "com"

int val1 = 1;          <- com.data  セクション
int val2;              <- com.bss   セクション

#pragma section default
```

-Xmulti_level=0（デフォルト）オプションを指定している場合、val1はcom.dataセクションに、val2はcom.bssセクションに配置されます。

② ROM化
-romオプションによって、初期値付き変数が配置されるセクションをROM化対象としてください。
CubeSuite+では、[リンク・オプション]タブ=> [セクション]カテゴリ=> [ROMからRAMへマップするセクション]にて右端の[...]ボタンを押下し、[テキスト編集]ダイアログ上で指定してください。

【com.data の RAM セクション名を com.data.R に指定する場合】

```
com.data=com.data.R
```

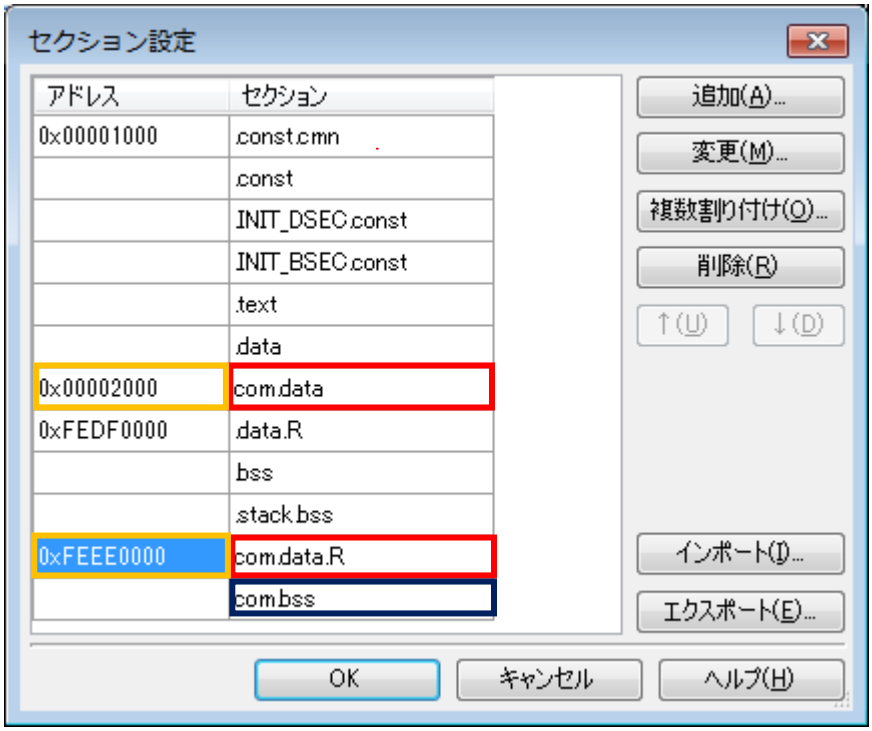


③　セクション配置

-startオプションによって、com.dataセクションをROM領域に、com.data.Rセクションを RAM領域に配置指定してください。同様に、com.bssセクションをRAM領域に配置指定してください。

CubeSuite+では、[リンク・オプション]タブ=> [セクション]カテゴリ=> [セクションの開始アドレス] にて右端の[...]ボタンを押下し、[セクション設定]ダイアログ上で指定してください。

【例】com.data セクションを 0x2000 番地に、com.data.R と com.bss セクションをグローバル RAM 領域である 0xFEEE0000 番地に配置指定する場合

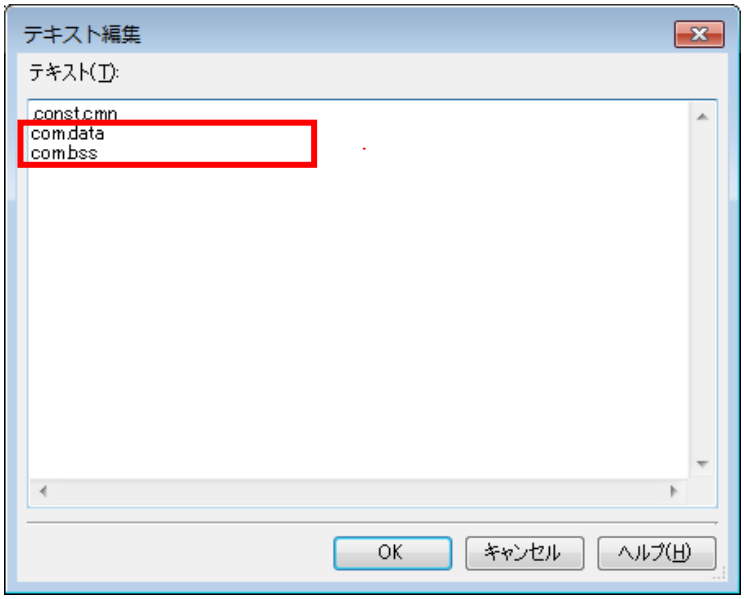



④　セクション初期化テーブルへの追記

cstartm.asm 内で定義しているセクション初期化テーブルに、追加したセクションの先頭ラベルのアドレス値(#__sセクション名)と終端ラベルのアドレス値(#__eセクション名)を追記してください。

```
        .section  ".INIT_DSEC.const", const
        .align    4
        .dw       #__s.data,        #__e.data,        #__s.data.R
        .dw       #__scom.data,     #__ecom.data,     #__scom.data.R

        .section  ".INIT_BSEC.const", const
        .align    4
        .dw       #__s.bss,         #__e.bss
        .dw       #__scom.bss,      #__ecom.bss
```

⑤　*.fsyファイルの出力
-fsymbol オプションによって、*.fsyファイルに変数名とその配置アドレスを出力させてください。CubeSuite+では、[リンク・オプション]タブ=> [セクション]カテゴリ=> [マルチコア用の外部定義シンボルをファイル出力するセクション]にて右端の[...]ボタンを押下し、[テキスト編集]ダイアログ上でcom.dataセクションとcom.bssセクションを指定してください。




リビルドすると、以下のような*.fsyファイルが出力します。ファイル名は”*プロジェクト名.fsy*”です。外部定義シンボルとして”_val1”が0xFEEE0000番地に、”_val2”が0xFEEE0004番地に配置されていることを示しています。
この*.fsyファイルを他プロジェクトにソース登録することによって、外部定義シンボル、つまり変数”_val1”,”_val2”を他プロジェクトからも参照可能となります。

```
;SECTION NAME = .const.cmn
  .public _pm1_setting_table
_pm1_setting_table .equ 0x1000
;SECTION NAME = com.data
  .public _val1
_val1 .equ 0xfeee0000
;SECTION NAME = com.bss
  .public _val2
_val2 .equ 0xfeee0004
```

⑥　*.fsyファイルのソース登録
外部定義シンボル”_val1”,”_val2”を参照するプロジェクトPE3のプロジェクト・ツリーのファイル・ノードに、⑤で出力した*.fsyファイルを登録してください。ファイル・ノードへのドラッグ &ドロップ、あるいはファイル・ノードを右クリック=> [追加]から登録できます。

## 3.5 関数の共有

シンボル・アドレス・ファイル(*.fsy)を使用することで、関数をプロジェクト間で共有させることが可能です。つまり、コア間で関数を共有させることが可能です。

本節では、関数funcをプロジェクトPE1で定義し、プロジェクトPE3から参照する方法を説明します。

① セクション名変更
デフォルトでは関数は.textセクションに配置されます。プロジェクトPE1にソース登録されているCソースファイルにて、プロジェクトPE3と共有する関数のセクション名を変更してください。

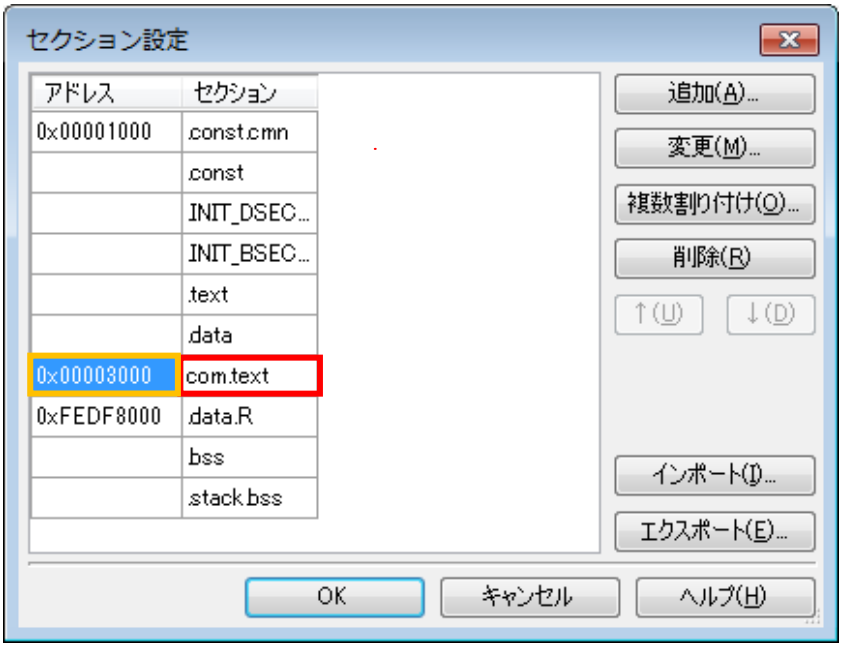
【C ソースファイルの記述例】

```
#pragma section text "com"

void func (void) {
  ・・・
}

#pragma section default
```

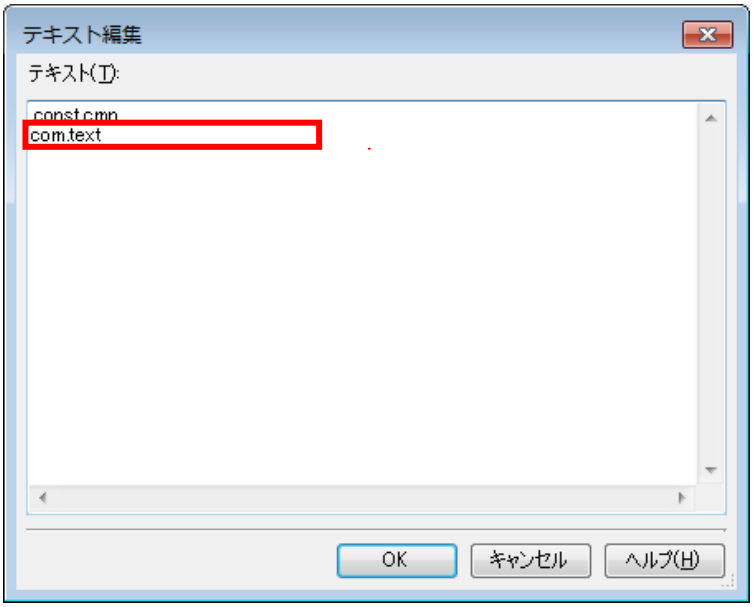
funcはcom.textセクションに配置されます。

② セクション配置
-startオプションによって、com.textセクションを配置指定してください。
CubeSuite+では、[リンク・オプション]タブ=> [セクション]カテゴリ=> [セクションの開始アドレス]にて右端の[...]ボタンを押下し、[セクション設定]ダイアログ上で指定してください。

【com.text セクションを 0x3000 番地に配置指定する場合】



③　*.fsyファイルの出力
-fsymbol オプションによって、*.fsyファイルに関数名とその配置アドレスを出力させてください。
CubeSuite+では、[リンク・オプション]タブ=> [セクション]カテゴリ=> [マルチコア用の外部定義シンボルをファイル出力するセクション]にて右端の[...]ボタンを押下し、[テキスト編集]ダイアログ上でcom.textセクションを指定してください。




リビルドすると、以下のような*.fsyファイルが出力します。ファイル名は"*プロジェクト名.fsy*"です。外部定義シンボルとして"_func"が0x3000番地に配置されていることを示しています。この*.fsyファイルを他プロジェクトにソース登録することによって、外部定義シンボル"_func"を他プロジェクトからも参照可能となります。

```
;SECTION NAME = .const.cmn
  .public _pm1_setting_table
_pm1_setting_table .equ 0x1000
;SECTION NAME = com.text
  .public _func
_func .equ 0x3000
```

④　*.fsyファイルの入力
外部定義シンボル"_func"を参照するプロジェクトPE3のプロジェクト・ツリーのファイル・ノードに、③で出力した*.fsyファイルを登録してください。ファイル・ノードへのドラッグ &ドロップ、あるいはファイル・ノードを右クリック=>[追加]から登録できます。

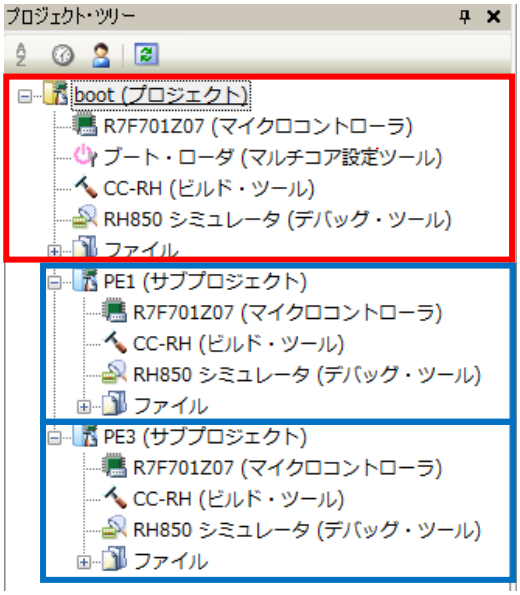
⑤　-Xcpu=g3kオプションの指定
共有関数を含むソースファイルをコンパイルする場合は、-Xcpu=g3kオプションを個別指定してください。
CubeSuite+では、プロジェクト・ツリーの該当ソースファイルを右クリック=> [プロパティ]=> [ビルド設定]タブ=> [個別コンパイル・オプションを設定する]を「はい」に選択し、[個別コンパイル・オプション]タブ=> [その他]カテゴリ=> [その他の追加オプション] にて-Xcpu=g3kオプションを直接指定してください。

# 第4章　　リビルド

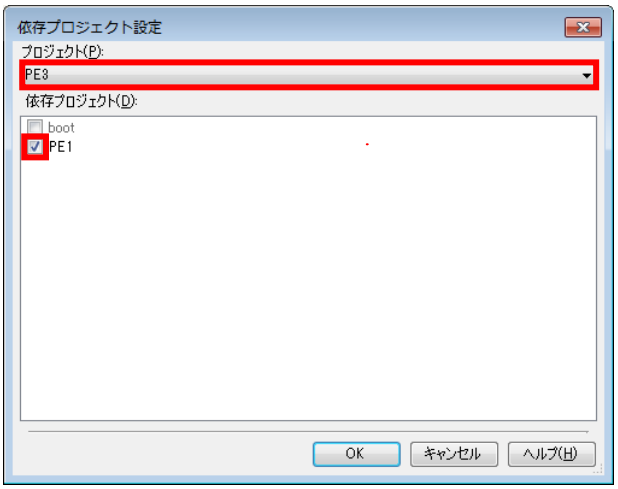
本章では、ブート・ローダプロジェクトとアプリケーションプロジェクトをリビルドするに方法ついて説明します。

## 4.1　複数プロジェクトのリビルド

ブート・ローダプロジェクトと2つのアプリケーションプロジェクトをリビルドしてください。なお、最初にリビルドされるサブプロジェクトで共有変数・共有関数を定義することを推奨します。
デフォルトではPE1(サブプロジェクト)=>PE3(サブプロジェクト)=>boot(メインプロジェクト)の順番でリビルドします。このとき、最初にリビルドされるPE1で共有変数・共有関数を定義してシンボル・アドレス・ファイル(*.fsy)を生成し、PE3に登録すると、リビルド時にはPE3には常に更新された*.fsyファイルが入力されることになり、1度のリビルドで済むためです。

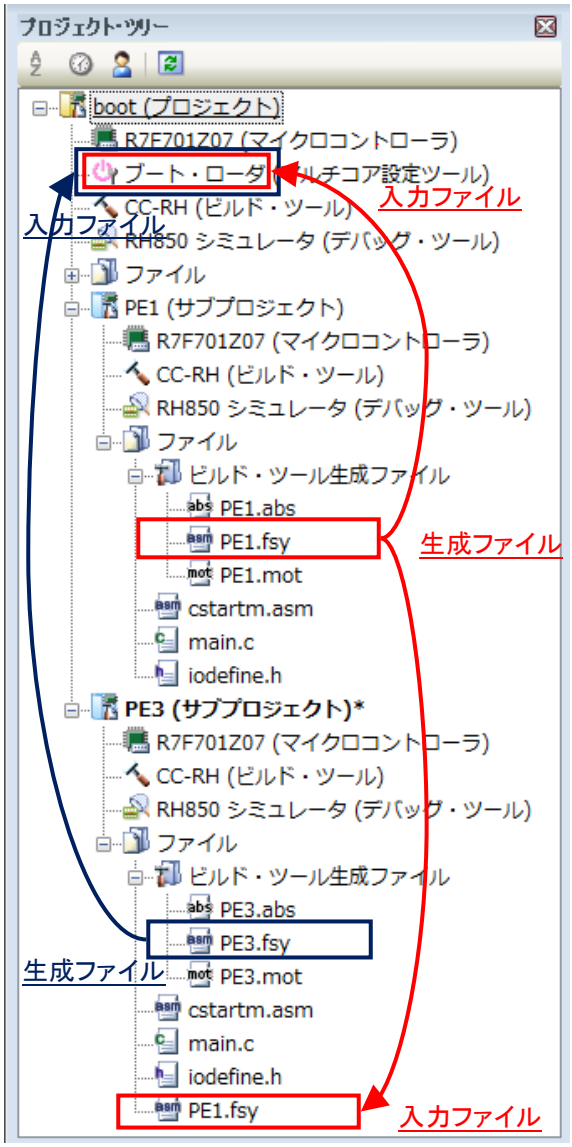



なお、CubeSuite+ではプロジェクトのビルド順を制御することが可能です。[プロジェクト]メニュー=> [依存プロジェクト設定]を押下し、[依存プロジェクト設定]ダイアログ上に設定します。




このダイアログ設定は、PE3はPE1に依存しているプロジェクトであることを指定しています。つまり
PE1=>PE3=>boot　の順番でリビルドします。

プロジェクト構成例を以下に示します。”boot”がブート・ローダプロジェクト、”PE1”がメインCPU用アプリケーションプロジェクト、”PE3”がPCU用アプリケーションプロジェクトとして構成しています。

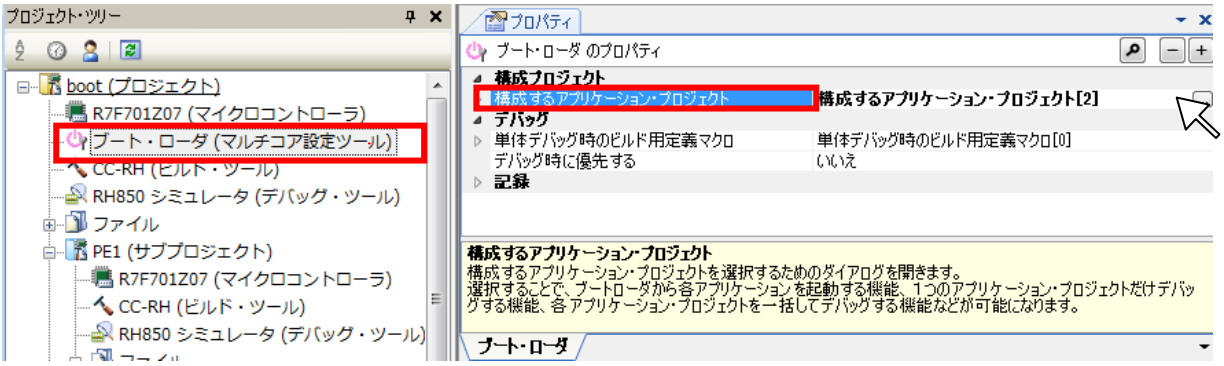



① プロジェクト全体のリビルドにより、まずプロジェクト”PE1”がリビルドされます。これによって、”PE1”で定義しているアプリケーション情報テーブル(_pm1_setting_table)と共有変数・共有関数の配置アドレスが指定されたシンボル・アドレス・ファイル(PE1.fsy) が生成されます。

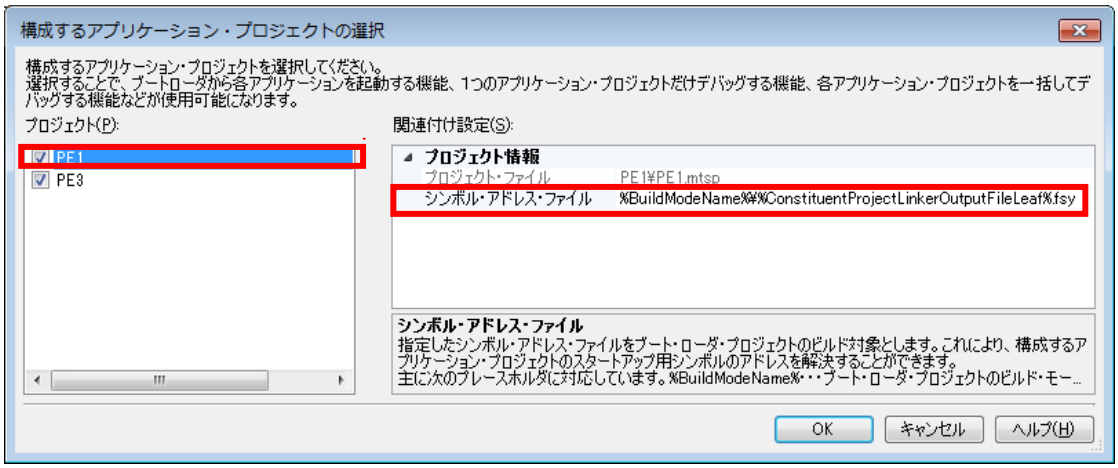
② 続いてプロジェクト”PE3”がリビルドされます。PE1.fsyを”PE3”へ入力することによって、”PE3”にて”PE1”で定義した共有変数と共有関数にアクセスすることが可能となります。また、”PE3”で定義しているアプリケーション情報テーブル(例えば”_pm3_setting_table”等)の配置アドレスが指定されたシンボル・アドレス・ファイル(PE3.fsy) が生成されます。

③ 最後にプロジェクト”boot”がリビルドされます。PE1.fsy とPE3.fsy を”boot”に入力することによって、”PE1”と”PE3”で定義しているアプリケーション情報テーブルの配置アドレスを解決します。
なお、”boot”の構成プロジェクトとして”PE1”と”PE3”が登録されているのであれば、これらの*.fsy は自動で登録されています。

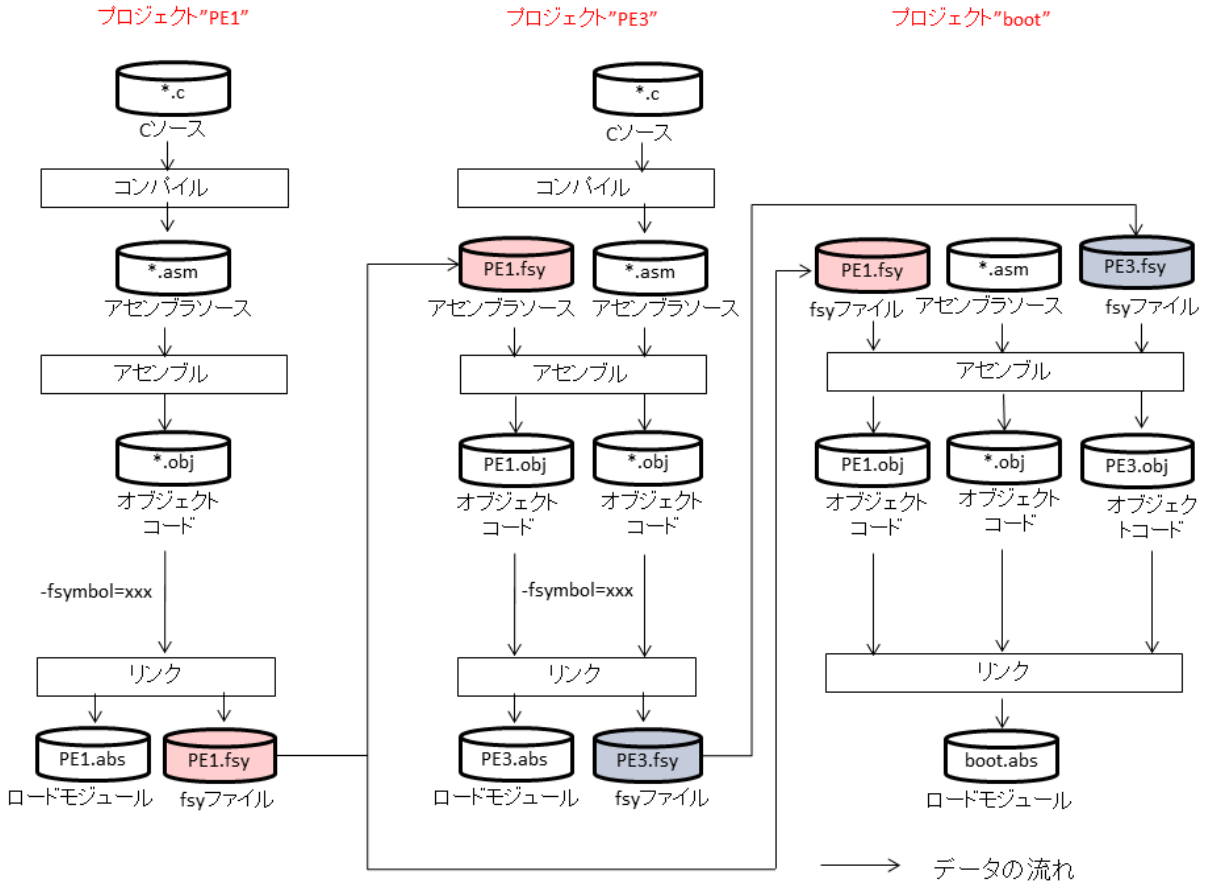
ブート・ローダプロジェクトに*.fsyが登録されていることを確認するには、[ブート・ローダ(マルチコア設定ツール)]を右クリック=>[プロパティ]を選択し、プロパティパネル上で[構成プロジェクト]カテゴリ=>[構成するアプリケーション・プロジェクト] の右端の[...]ボタンを押下してください。



以下のような[構成するアプリケーション・プロジェクトの選択]ダイアログが起ち上がります。ブート・ローダプロジェクトに追加したアプリケーションプロジェクトにチェックが入っており、そのアプリケーションプロジェクトで生成した*.fsyファイルが関連付けられていることを確認できます。




なお、プロジェクト"PE1"、"PE3"、"boot"に対するリビルドの流れは以下の通りです。

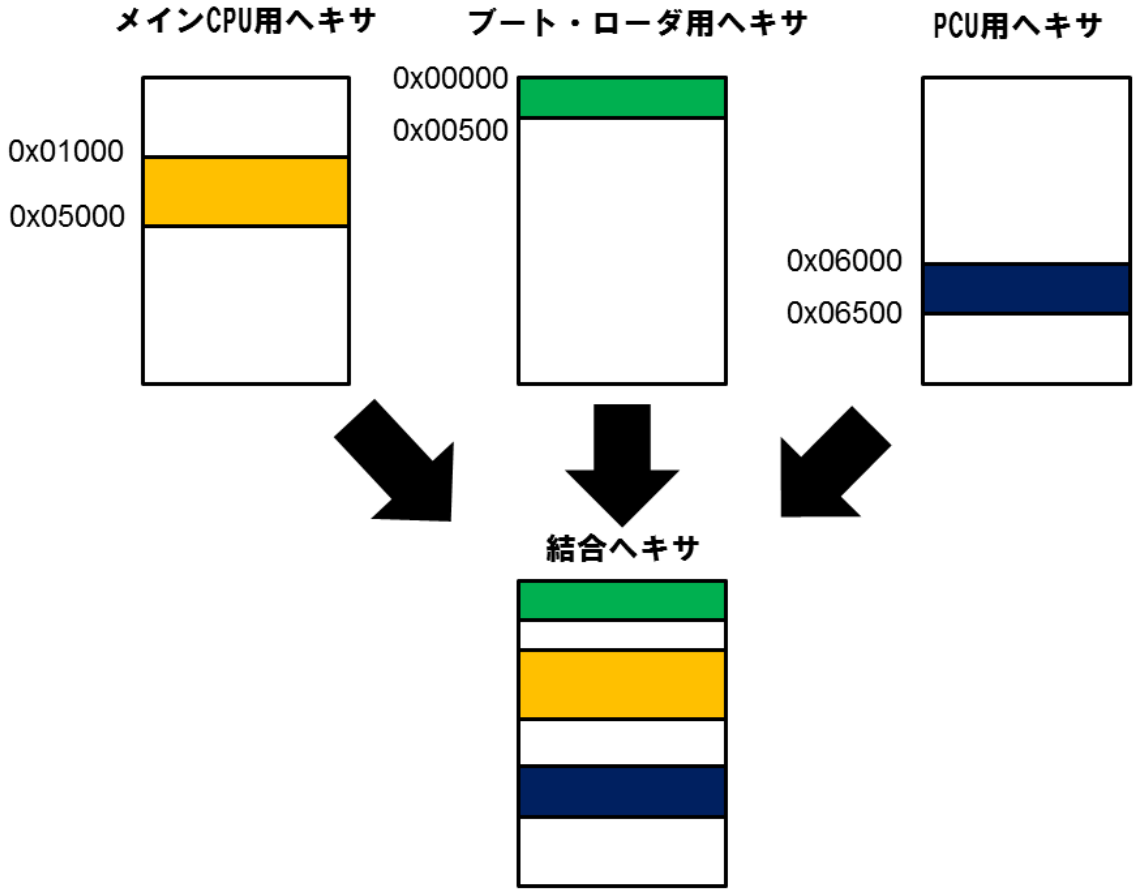


# 第5章　　オブジェクト結合

本章では、構成アプリケーションの選択と複数のオブジェクトを結合する機能について説明します。

## 5.1　オブジェクト結合機能とは

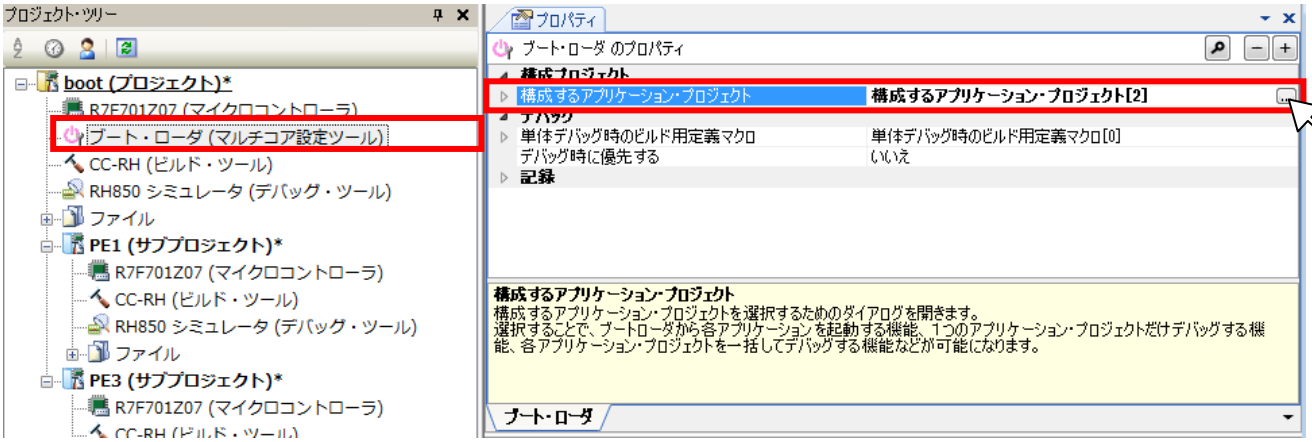
ブート・ローダプロジェクトと2つのアプリケーションプロジェクトをリビルドすると、3つのロードモジュールと3つのヘキサ・ファイル(インテル拡張ヘキサ・ファイル、またはモトローラ・S タイプ・ファイル)が生成されます。生成された複数のヘキサ・ファイルをそれぞれ結合し、全体として1つのヘキサ・ファイルを生成することが可能です。この機能をオブジェクト結合機能といいます。オブジェクト結合機能を使用することでヘキサ・ファイルを1ファイルで管理することが可能です。ただし、インテル拡張ヘキサ・ファイルとモトローラ・Sタイプ・ファイルを混在させることはできませんので、ブート・ローダプロジェクトとアプリケーションプロジェクトのヘキサ・フォーマットは合わせてください。
なお、複数のヘキサを結合する際、アドレスが重複している場合はオーバラップエラーが発生します。ただし、データが存在しないRAM領域のチェック出来ませんので、お客様自身でマップファイルを参照する等してオーバーラップしていないかチェックしてください。セクションの割り付けアドレスをチェックする”-cpu”オプション等でチェック可能です。



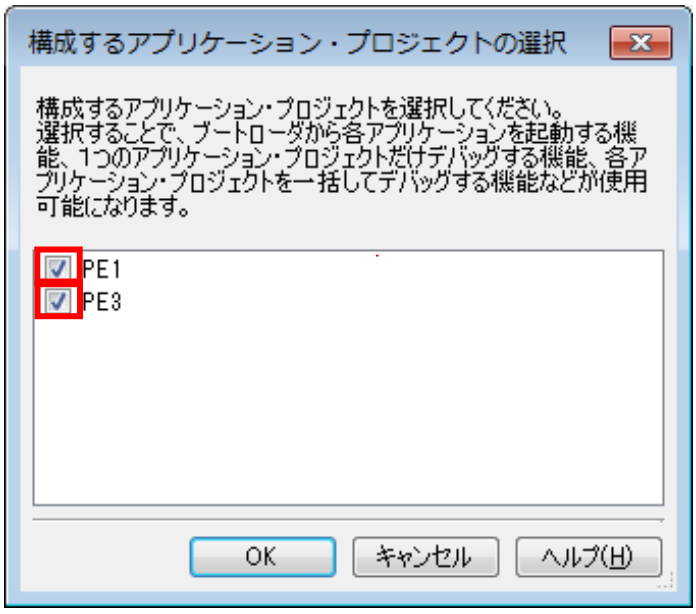
## 5.2 構成アプリケーション・プロジェクトの選択

作成したアプリケーションプロジェクトが、ブート・ローダプロジェクトの「構成アプリケーション・プロジェクト」として登録されている必要があります。なお、ブート・ローダプロジェクトに対して追加したアプリケーションプロジェクトは、デフォルトで構成アプリケーション・プロジェクトとして登録されています。

構成アプリケーションプロジェクトは、プロジェクト・ツリーの[ブート・ローダ(マルチコア設定ツール)]を右クリック=>[プロパティ]を選択し、プロパティパネル上で[構成プロジェクト]カテゴリ=>[構成するアプリケーション・プロジェクト]で変更することが可能です。



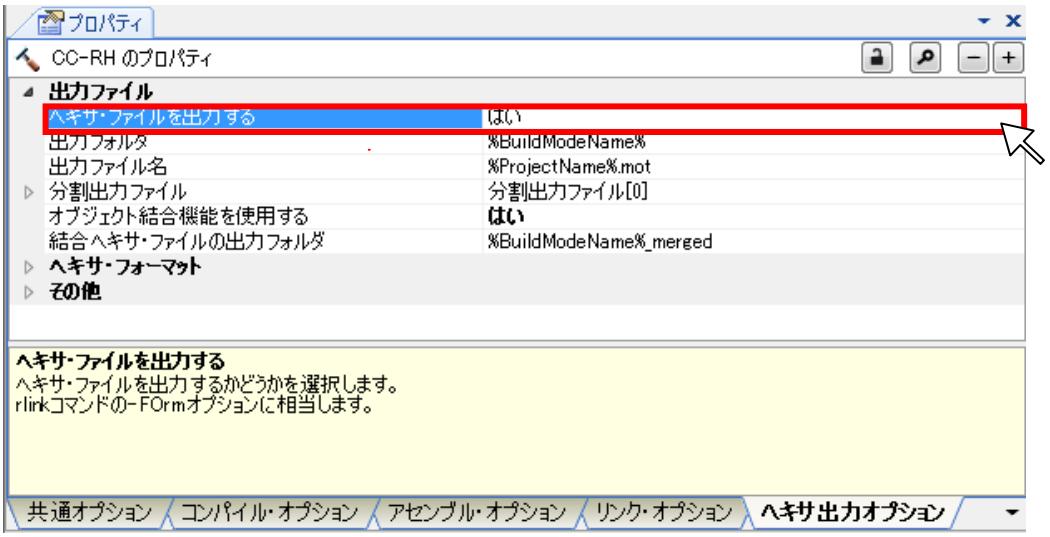

右端の[...]ボタンをクリックしてください。以下のような[構成するアプリケーション・プロジェクトの選択]ダイアログが起ち上がります。ブート・ローダプロジェクト”boot”に追加したアプリケーションプロジェクト”PE1”と”PE3”にチェックが入っており、”boot”の構成アプリケーションとして登録されています。



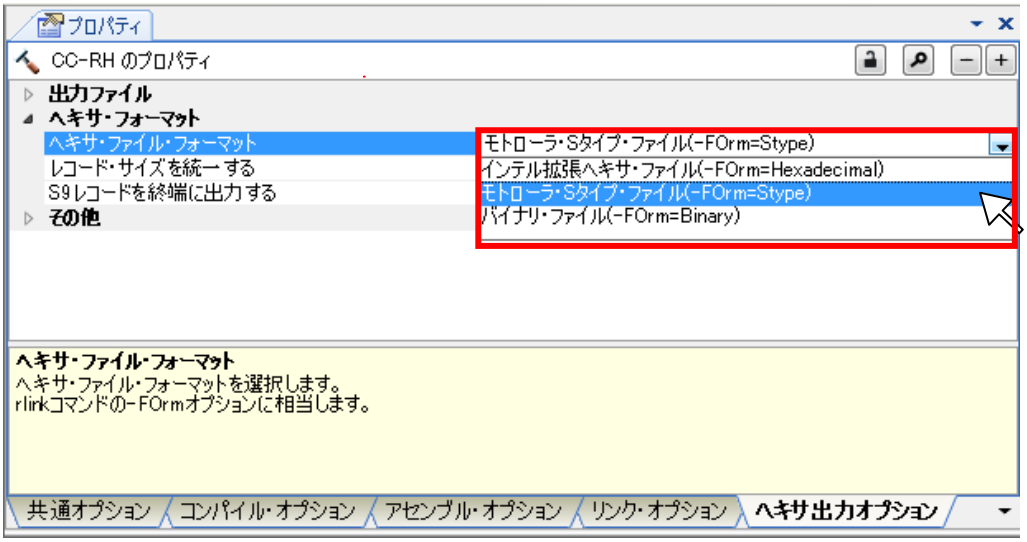
## 5.3 オブジェクトの結合

以下の手順で、生成したヘキサ・ファイル(インテル拡張ヘキサ・ファイル、またはモトローラ・S タイプ・ファイル)を結合できます。これをオブジェクト結合機能といいます。オブジェクト結合機能により、ブート・ローダプロジェクトで生成したヘキサ・ファイルとアプリケーションプロジェクトで生成したメインCPUとPCU用のヘキサ・ファイルを結合し、1つのヘキサ・ファイルを生成することが可能です。

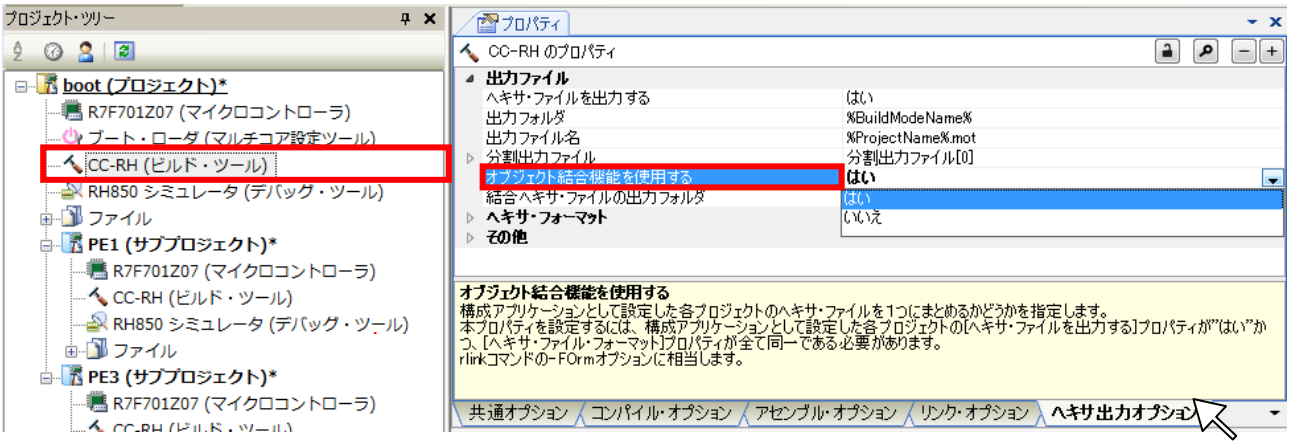
ブート・ローダプロジェクト、および各アプリケーションプロジェクトにおいて、ヘキサ・ファイルの出力およびヘキサ・フォーマットの設定を行ってください。[ヘキサ出力オプション]タブ=> [出力ファイル]カテゴリ=> [ヘキサ・ファイルを出力する]で「はい」を選択してください。デフォルトでは「はい」が選択されています。




続いて[ヘキサ出力オプション]タブ=> [ヘキサ・フォーマット]カテゴリ=> [ヘキサ・ファイル・フォーマット]でインテル拡張ヘキサ・ファイル、またはモトローラ・S タイプ・ファイルのいずれかを選択してください。なお、インテル拡張ヘキサ・ファイルとモトローラ・Sタイプ・ファイルを結合することはできません。ブート・ローダプロジェクト、および各アプリケーションプロジェクトのヘキサ・フォーマットは統一してください。




最後に、ブート・ローダプロジェクトにて1つのヘキサ・ファイルに結合する指定を行います。ブート・ローダプロジェクトの[出力ファイル]カテゴリ=>[オブジェクト結合機能を使用する]で「はい」を選択してください。




ブート・ローダプロジェクトをリビルドすると、ブート・ローダプロジェクトと、構成アプリケーションプロジェクトとして指定されたプロジェクトのヘキサ・ファイルが結合されます。
結合されたヘキサ・ファイルは[出力ファイル]カテゴリ->[結合ヘキサ・ファイルの出力フォルダ]に生成されます。デフォルトでは"DefaultBuild_merged"フォルダに生成されます。

以上

# 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、お客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
2. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
3. 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に関し、当社は、何らの責任を負うものではありません。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. 当社製品を改造、改変、複製等しないでください。かかる改造、改変、複製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）、交通用信号機器、
   防災・防犯装置、各種安全装置等
   当社製品は、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（原子力制御システム、軍事機器等）に使用されることを意図しておらず、使用することはできません。たとえ、意図しない用途に当社製品を使用したことによりお客様または第三者に損害が生じても、当社は一切その責任を負いません。なお、ご不明点がある場合は、当社営業にお問い合わせください。
6. 当社製品をご使用の際は、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他の保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障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、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、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。お客様が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. 本資料に記載されている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。また、当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途に使用しないでください。当社製品または技術を輸出する場合は、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。
10. お客様の転売等により、本ご注意書き記載の諸条件に抵触して当社製品が使用され、その使用から損害が生じた場合、当社は何らの責任も負わず、お客様にてご負担して頂きますのでご了承ください。
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。

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その総株主の議決権の過半数を直接または間接に保有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。


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　〒100-0004　千代田区大手町2-6-2（日本ビル）

■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：http://japan.renesas.com/contact/




Colophon 3.0