CS+ Ver.3.02.00 RH850 マルチコア環境用チュートリアル（ビルド編 2）

目次

第1章　　概説

本書は、CS+のマルチコアプロジェクト構成で RH850 マルチコアマイコンのプロジェクトを開発されるユーザ向けチュートリアルです。また、CPU コア間で共通の例外/割り込みベクタを持つ RH850 マイコンが対象です。マルチコアプロジェクトを新規作成してビルドするまでの手順を説明します。

対象マイコン ： RH850/C1H（R7F701270）

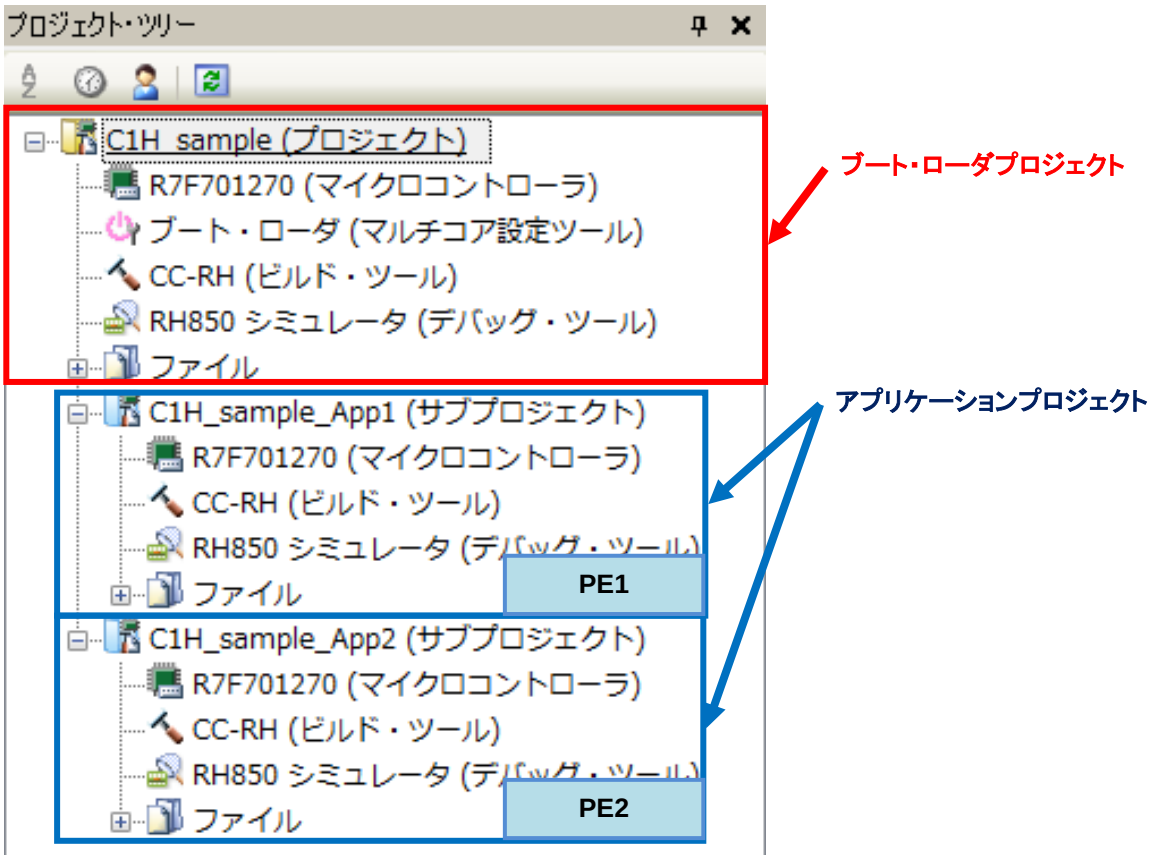
本章ではマルチコアプロジェクトの概要を説明します。マルチコアプロジェクトは、1 つのマルチコア用ブート・ローダプロジェクトと、マイコンに搭載している PE（プロセッサ・エレメント）数分のマルチコア用アプリケーションプロジェクトから構成されます。

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 ごとの処理を実行します。

CS+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。ブート・ローダプロジェクトと、その下層の PE 数分のアプリケーションプロジェクトから構成されます。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。

なお、RH850/C1H は PE1 と PE2 から構成されますが、例えば RH850/E1L は PE1 と PE3 から構成されます。そのため、対象デバイスが RH850/E1L の場合は PE2 を PE3 に読み替えてカスタマイズしてください。



第2章　　マルチコア用プロジェクトの作成

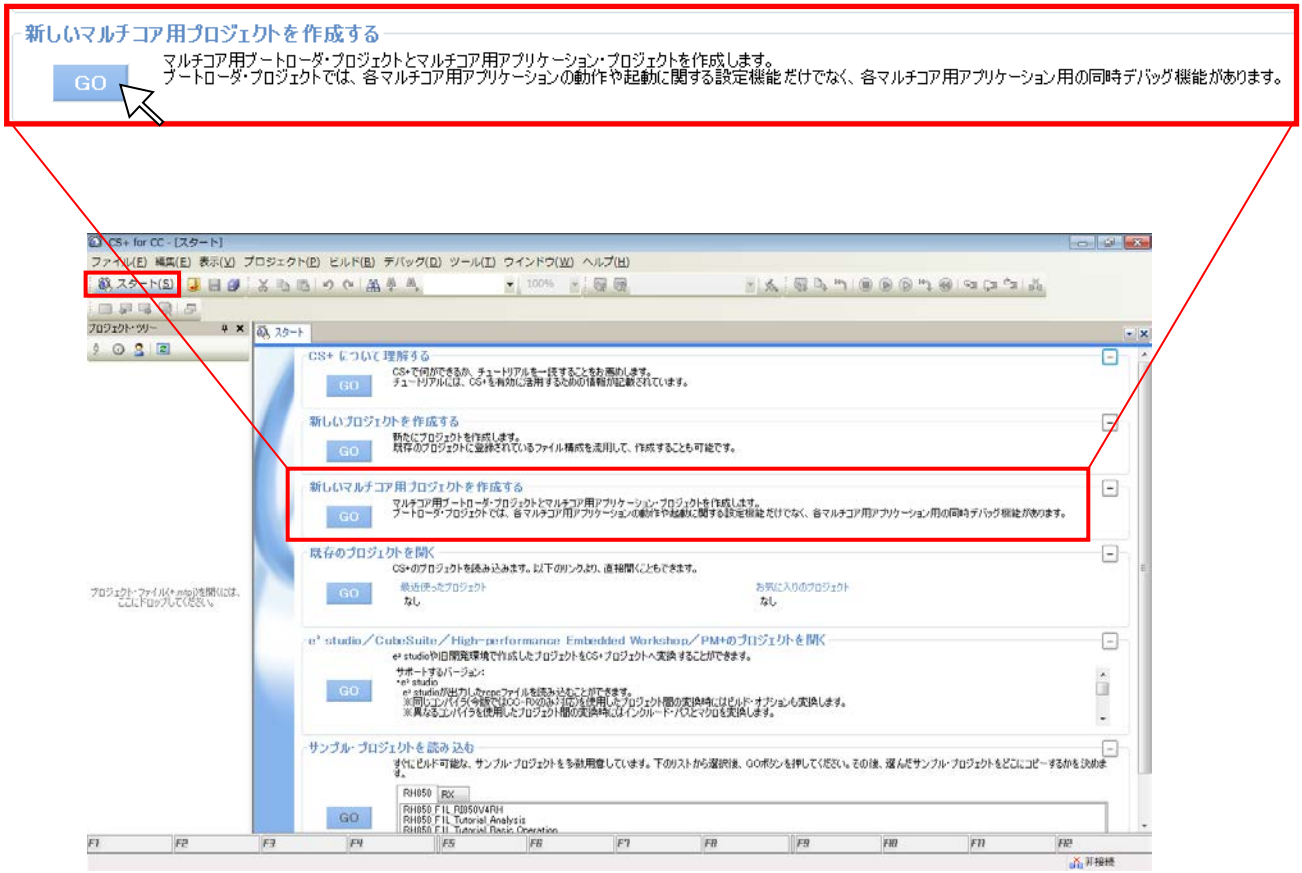
本章では、マルチコア用プロジェクトの作成方法を説明します。

2.1　新規マルチコア用プロジェクトの作成

以下の手順で新しくマルチコア用プロジェクトを作成します。

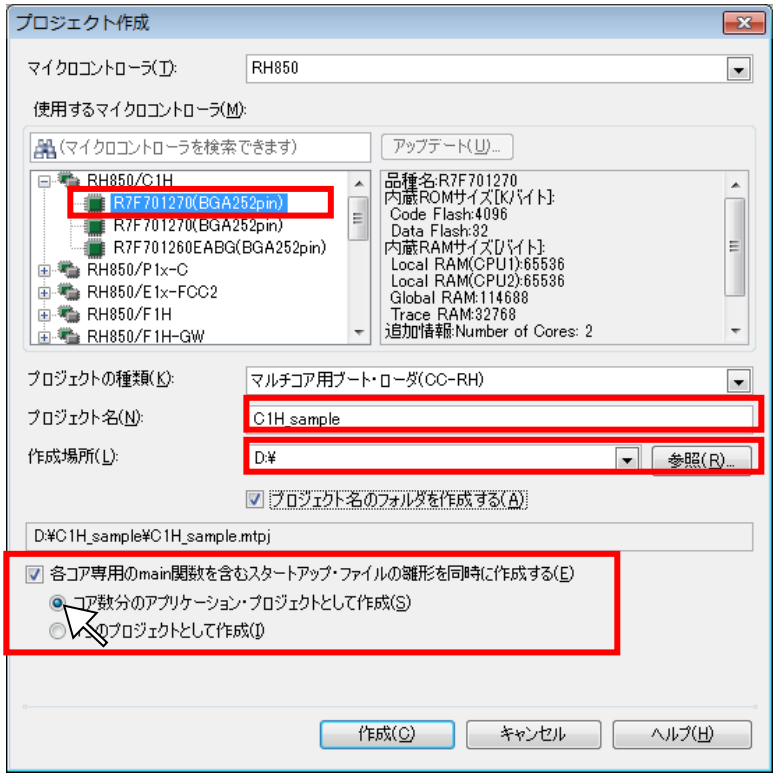
(1) 新規マルチコア用プロジェクトの作成

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[新しいマルチコア用プロジェクトを作成する]の[GO]ボタンを押下してください。



(2) プロジェクトの設定

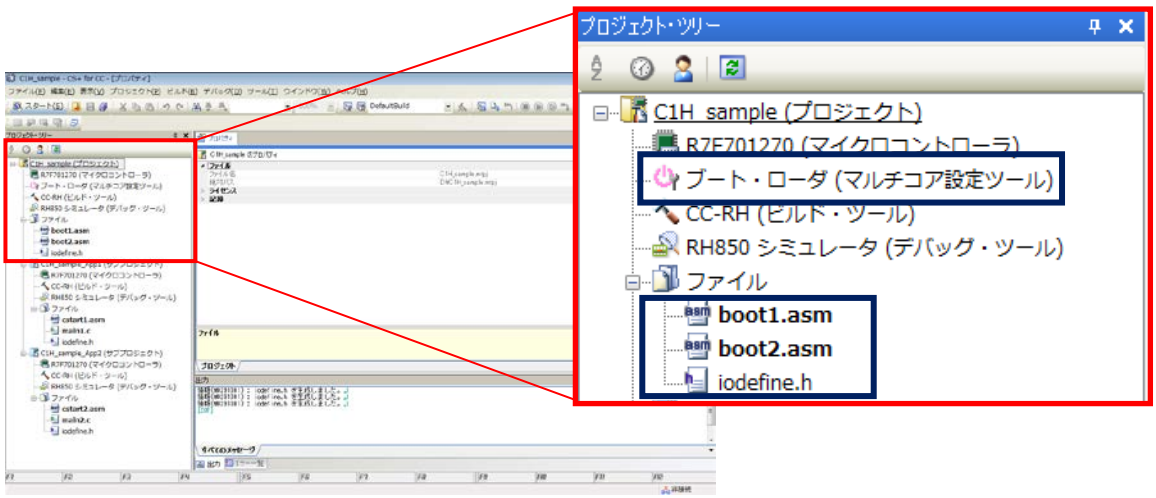
プロジェクト作成ダイアログが起ち上がります。本ダイアログ上で、[使用するマイクロコントローラ]にて対象マイコンを選択し、プロジェクト名・作成場所を指定してください。最後に[各コア専用の main 関数を含むスタートアップ・ファイルの雛形を同時に作成する]にチェックを入れて、[コア数分のアプリケーション・プロジェクトとして作成]のラジオボタンを選択してください。最後に[作成]ボタンを押下してください。



(3) マルチコア用プロジェクトの起動

マルチコア用プロジェクトとして、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が起動します。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_App*n*（*n* は PE 番号）」となります。プロジェクト名は変更可能です。

ブート・ローダプロジェクトのプロジェクト・ツリーには[ブート・ローダ]ノードが表示されています。ファイル・ノードには boot1.asm、boot2.asm、iodefine.h が自動で登録されます。



第3章　　ブート・ローダプロジェクト

本章では、マルチコア用プロジェクトの内、ブート・ローダプロジェクトのカスタマイズ方法を説明します。

3.1.1　ソース登録

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

・PE1 のブート・ローダ用スタートアップルーチン（boot1.asm）
・PE2 のブート・ローダ用スタートアップルーチン（boot2.asm）
・I/O ヘッダ・ファイル（iodefine.h）

boot1.asm、boot2.asm では各コア用の例外/割り込みベクタテーブルが定義されています。RH850/C1H は PE1・PE2 とも例外/割り込みベクタが共通のため boot2.asm は不要です。そのため、プロジェクト・ツリー上で boot2.asm を選択して右クリック=>[プロジェクトを外す]により、本ファイルをプロジェクトから外してください。

ブート・ローダプロジェクトのソースファイルは boot.1asm、iodefine.h のみから構成されます。boot1.asm、iodefine.h について以降の項で説明します。

3.1.2　ブート・ローダ用スタートアップルーチン(boot1.asm)

ブート・ローダ用スタートアップルーチン（boot1.asm）では、各コアのリセット発生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と、例外/割り込みベクタテーブルの定義を行っています。

(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（.L.entry_PE1）に分岐します。同様に PEID が n の場合、PEn のエントリルーチン（.L.entry_PEn）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      .L.entry_PE1
        cmp     2, r10
        bz      .L.entry_PE2
        cmp     3, r10
        bz      .L.entry_PE3
        cmp     4, r10
        bz      .L.entry_PE4
        cmp     5, r10
        bz      .L.entry_PE5
        cmp     6, r10
        bz      .L.entry_PE6
        cmp     7, r10
        bz      .L.entry_PE7
```

(2) PE1 のエントリルーチン（.L.entry_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（_set_table_reference_method）に分岐します。その後、PE1 用アプリケーションプロジェクトのスタートアップルーチン(cstart1.asm)で定義しているエントリルーチン（__cstart_pm1）に分岐します。

```
.L.entry_PE1:
        jarl    _hdwinit_PE1, lp    ; initialize hardware
$ifdef USE_TABLE_REFERENCE_METHOD
        mov     #__sEIINTTBL_PE1, r6
        jarl    _set_table_reference_method, lp ; set table reference method
$endif

        jr32    __cstart_pm1
```

_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 等）は、.set 疑似命令を使用してユーザ自身で対象マイコン用のアドレスを指定してください。

```
;-------------------------------------------------------------------------
;       hdwinit_PE1
; Specify RAM addresses suitable to your system if needed.
;-------------------------------------------------------------------------
        GLOBAL_RAM_ADDR     .set    0
        GLOBAL_RAM_END      .set    0
        LOCAL_RAM_PE1_ADDR  .set    0
        LOCAL_RAM_PE1_END   .set    0

        .align  2
_hdwinit_PE1:
        mov     lp, r29                 ; 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     r29, lp
        jmp     [lp]
```

【記述例】グローバル RAM のアドレスが 0xFEEF8000～0xFEF13FFF、ローカル RAM(PE1)のアドレスが 0xFEBF0000～0xFEBFFFFF の場合（RH850/C1H）

```
GLOBAL_RAM_ADDR      .set      0xFEEF8000
GLOBAL_RAM_END       .set      0xFEF13FFF

LOCAL_RAM_PE1_ADDR   .set      0xFEBF0000
LOCAL_RAM_PE1_END    .set      0xFEBFFFFF
```

_set_table_reference_method の呼び出し

EIINT 割り込みチャネル 0～2 の割り込みを、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 レジスタにテーブル参照方式のベクタの先頭アドレスを設定します。r6 レジスタには PE1 用テーブル参照方式のベクタの先頭アドレスが(#__sEIINTTBL_PE1)が設定されています。EIINTTBL_PE1 セクションは boot1.asm 内の割り込みベクタテーブルで定義しています。

なお、デフォルトではこれらの処理は無効です。これらの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。本マクロを有効にするには、コメントを削除してください。

```
        ; if using eiint as table reference method,
        ; enable next line's macro.

        USE_TABLE_REFERENCE_METHOD .set 1
```

```
$ifdef USE_TABLE_REFERENCE_METHOD
;---------------------------------------------------------------------------
;       set table reference method
;---------------------------------------------------------------------------
        ; interrupt control register address
        ICBASE  .set    0xfffeea00

        .align  2
_set_table_reference_method:
        ldsr    r6, 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のスタートアップルーチン(cstart1.asm)で定義しているエントリルーチン（__cstart_pm1）に分岐します。

(3) PE2 のエントリルーチン（.L.entry_PE2）

RAM をクリアするルーチン（_hdwinit_PE2）、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（_set_table_reference_method）に分岐します。その後、PE2 用アプリケーションプロジェクトのスタートアップルーチン(cstart2.asm)で定義しているエントリルーチン（__cstart_pm2）に分岐します。

なお、デフォルトでは_set_table_reference_method への分岐処理と PE2 用アプリケーションプロジェクトへ分岐処理はコメントアウトしており、__exit ルーチンへの分岐で処理を終了しています。PE2 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。

```
.L.entry_PE2:
        jarl    _hdwinit_PE2, lp    ; initialize hardware
$ifdef USE_TABLE_REFERENCE_METHOD
        mov     #__sEIINTTBL_PE2, r6
        jarl    _set_table_reference_method, lp ; set table reference method
$endif
        jr32    __cstart_pm2
;       br      __exit
```

_hdwinit_PE2 の呼び出し

ECC 機能向けにローカル RAM(PE2)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_PE2_ADDR, LOCAL_RAM_PE2_END ）は、.set 疑似命令を使用してユーザ自身で対象マイコン用のアドレスを指定してください。

```
;-----------------------------------------------------------------------
;       hdwinit_PE2
; Specify RAM addresses suitable to your system if needed.
;-----------------------------------------------------------------------
        LOCAL_RAM_PE2_ADDR  .set    0
        LOCAL_RAM_PE2_END   .set    0

        .align  2
_hdwinit_PE2:
        mov     lp, r14                     ; save return address

        ; clear Local RAM PE2
        mov     LOCAL_RAM_PE2_ADDR, r6
        mov     LOCAL_RAM_PE2_END, r7
        jarl    _zeroclr4, lp

        mov     r14, lp
        jmp     [lp]
```

【記述例】ローカル RAM(PE2)のアドレスが 0xFE9F0000～0xFE9FFFFF の場合（RH850/C1H）

```
LOCAL_RAM_PE2_ADDR  .set    0xFE9F0000
LOCAL_RAM_PE2_END   .set    0xFE9FFFFF
```

_set_table_reference_method の呼び出し

PE1 と同じ_set_table_reference_method 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。

また、INTBP レジスタにテーブル参照方式のベクタの先頭アドレスを設定します。r6 レジスタには PE2 用テーブル参照方式のベクタの先頭アドレスが(#__sEIINTTBL_PE2)がデフォルトで設定されています。EIINTTBL_PE2 セクションはプロジェクトから外した boot2.asm 内の割り込みベクタテーブルで定義しています。そのため、デフォルトでは Undefined external symbol エラーとなります。PE1 と PE2 で同じベクタを使用する場合は、#__sEIINTTBL_PE2 を#__sEIINTTBL_PE1 に変更してください。PE1 と PE2 で異なるベクタを用意する場合は、boot2.asm 内の割り込みベクタテーブルをコピーする等、ユーザ自身でテーブルを EIINTTBL_PE2 セクションとして定義してください。

```
$ifdef USE_TABLE_REFERENCE_METHOD
        mov     #__sEIINTTBL_PE2, r6
        jarl    _set_table_reference_method, lp ; set table reference method
$endif
```

【記述例】PE2 用テーブル参照方式のベクタを定義する場合

```
        .section "EIINTTBL_PE2", const
        .dw     #_int0 ; EIINT0
        .dw     #_int1 ; EIINT1
        …
```

PE2 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

PE2 用アプリケーションプロジェクトのスタートアップルーチン(cstart2.asm)で定義しているエントリルーチン（__cstart_pm2）に分岐します。

(4) PE*n* のエントリルーチン（.L.entry_PE*n*）

RH850/C1H は PE*n*(*n*=3～7)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*n* 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。

```
.L.entry_PEn:
        br      __exit
```

```
__exit:
        br      __exit
```

(5) 例外/割り込みベクタテーブル

RESET

RBASEレジスタ値をRESETのアドレスとします。

```
        .section "RESET_PE1", text
        .align  512
        jr32    __start ; RESET
```

上記の定義により、RESET_PE1 セクションの先頭に「jr32 __start」が埋め込まれます。

CS+で新規マルチコア用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_PE1 セクションは%ResetVectorPE1%番地に配置指定されています。%ResetVectorPE1%は、ブート・ローダプロジェクトの[マイクロコントローラ]ノード=>[マイクロコントローラ情報タブ] =>[マイクロコントローラ情報]カテゴリ=>[リセット・ベクタ・アドレス]で指定した値です。

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。

CS+で新規マルチコア用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_PE1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。

```
;-------------------------------------------------------------------------------
;           exception vector table
;-------------------------------------------------------------------------------
            .section "RESET_PE1", text
            .align      512
            jr32        __start ; RESET

            .align      16
            jr32        _Dummy ; SYSERR

            .align      16
            jr32        _Dummy ; HVTRAP

            ...
```

RESET の直後から配置

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 あるいは _Dummy_EI に分岐させる命令を配置しています。ダミー関数は自分自身への分岐を繰り返すルーチンです。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ダミー関数を「_割り込み関数名」に変更し、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 指令により指定してください。

【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_PE1", text
            .align      512
            jr32        __start ; RESET

            .align      16
            jr32        _func1 ; SYSERR

            .align      16
            jr32        _Dummy ; HVTRAP

            ...
```

「__Dummy」を「**_割り込み関数名**」に変更

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

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、割り込みチャネル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。

boot1.asm では、テーブル参照方式の例外/割り込みテーブルを EIINTTBL_PE1 セクションとして、EIINTTBL_PE1 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、EIINT 割り込みチャネル 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す。必要に応じてカスタマイズしてください。

```
        .section "EIINTTBL_PE1", const
        .align      512
        .dw         #_Dummy_EI ; INT0
        .dw         #_Dummy_EI ; INT1
        .dw         #_Dummy_EI ; INT2
        .rept       512 - 3
        .dw         #_Dummy_EI ; INTn
        .endm
```

CS+で新規マルチコア用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_PE1 セクションは RESET_PE1 セクションの直後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変更してください。

カスタマイズする EI レベル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、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 指令により指定してください。

【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_PE1", 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 - 10
        .dw         #_Dummy_EI ; INTn
```

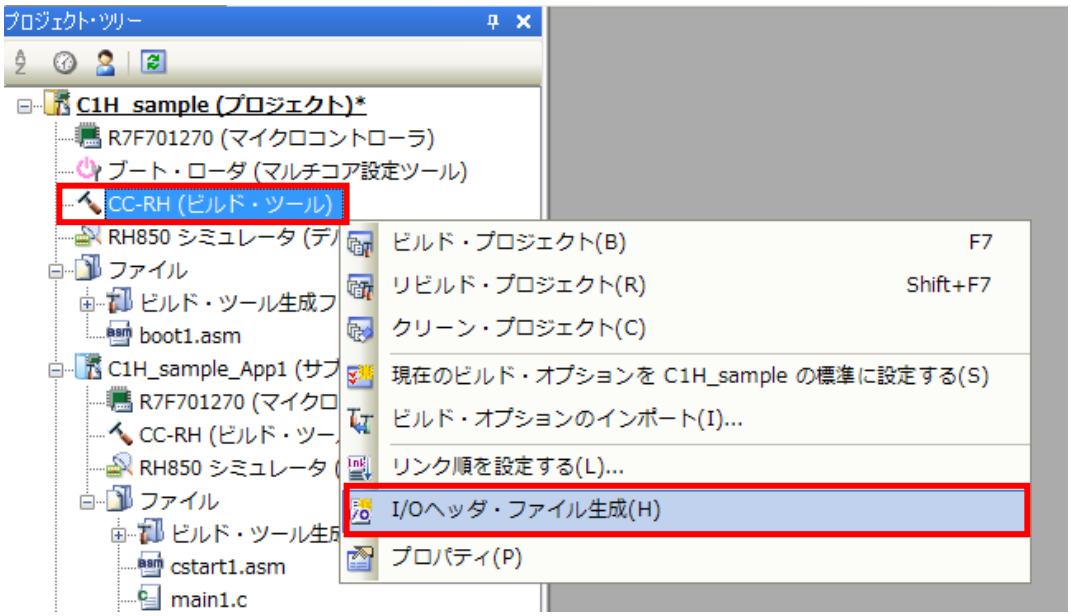
「#_Dummy_EI」を
「#_割り込み関数名」に変更

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

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

3.1.3 I/Oヘッダ・ファイル

新規マルチコア用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、ブート・ローダ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。

CS+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。

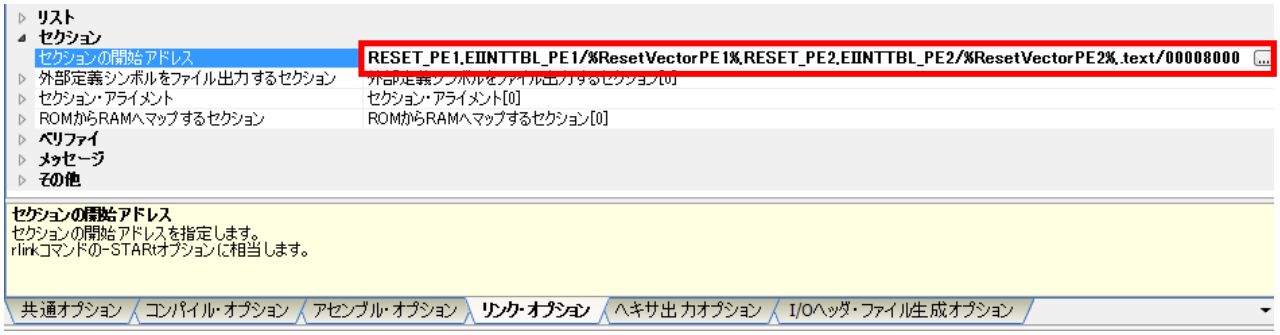


3.2 オプション設定

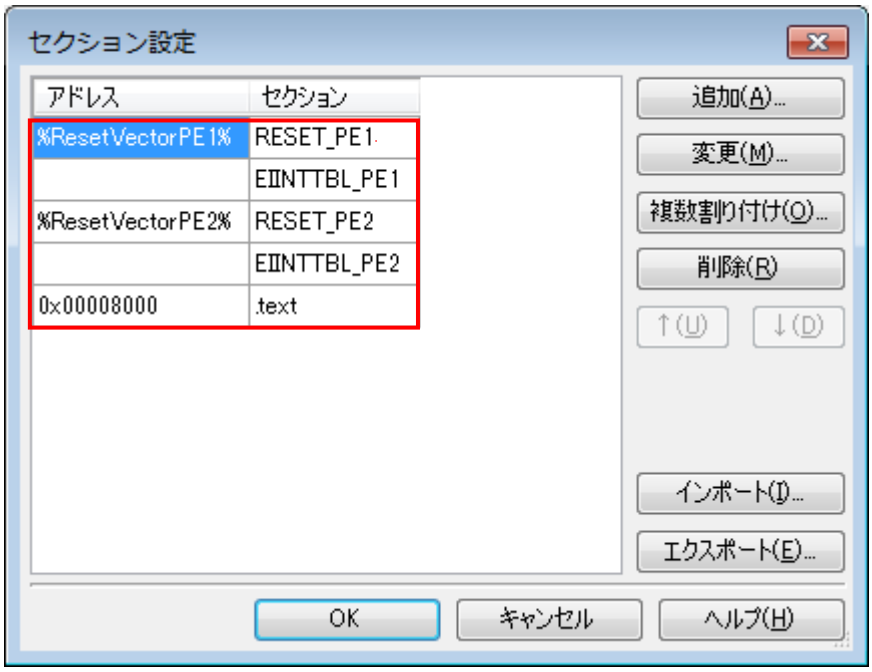
ブート・ローダプロジェクトにおいて特に使用するオプションについて説明します。

3.2.1 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。

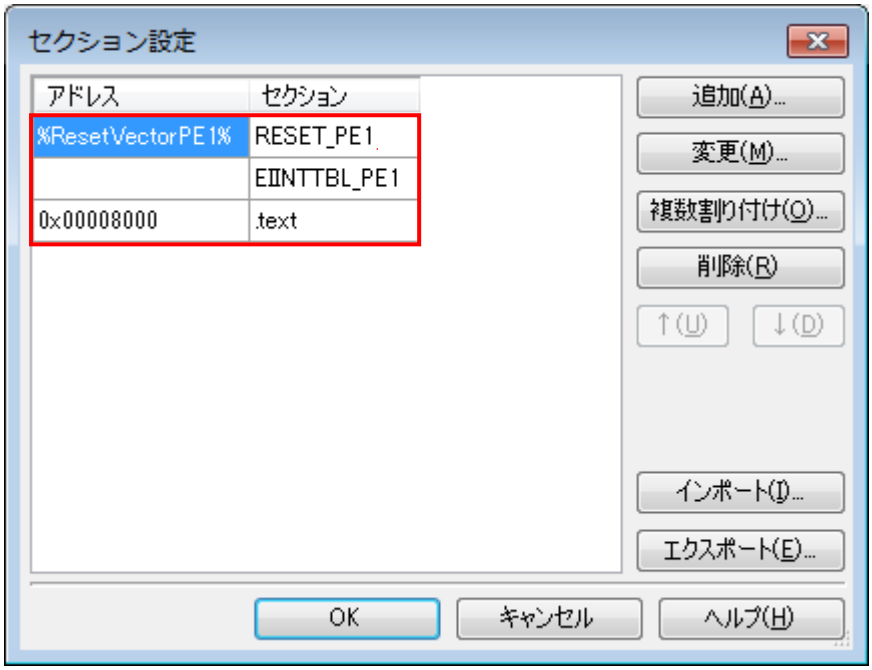


[セクションの開始アドレス]プロパティの右端の[...]ボタンを押下すると、以下のようなセクション設定ダイアログが起ち上がります。このダイアログから指定することも可能です。

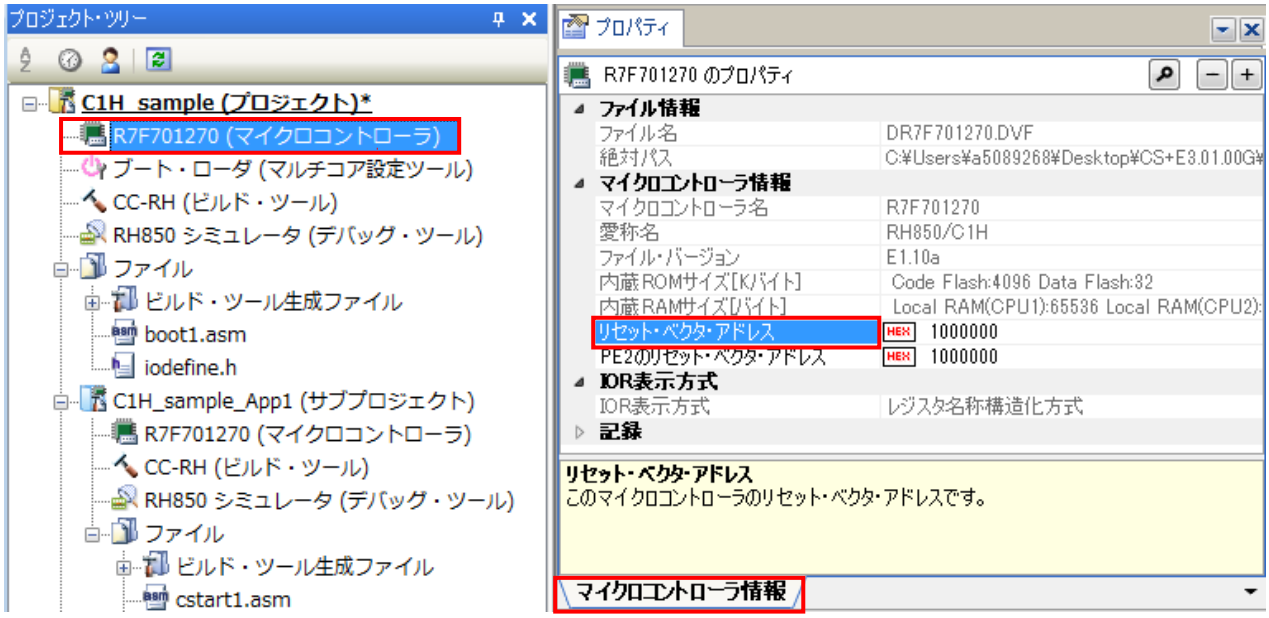


このセクション設定によって、%ResetVectorPE1%番地から上位方向にRESET_PE1=>EIINTTBL_PE1セクションを配置し、%ResetVectorPE2%番地から上位方向にRESET_PE2=>EIINTTBL_PE2セクションを配置し、0x08000番地から.textセクションを配置します。本ダイアログ上で、所望のアドレス配置となるようにカスタマイズしてください。

ブート・ローダプロジェクトから外したboot2.asm内で定義されているRESET_PE2、EIINTTBL_PE2セクションが配置指定されていますので、本セクションは削除してください。



%ResetVectorPE1%は、ブート・ローダプロジェクトの[マイクロコントローラ]ノード=>[マイクロコントローラ情報タブ]=>[マイクロコントローラ情報]カテゴリ=>[リセット・ベクタ・アドレス]で指定した値が渡されます。RESET_PE1 セクションのアドレスを変更する場合は本プロパティ値を変更してください。



第4章　　アプリケーションプロジェクト

本章では、マルチコア用プロジェクトの内、アプリケーションプロジェクトのカスタマイズ方法を説明します。

4.1　ソース登録

CS+で新規マルチコア用プロジェクト作成時、アプリケーション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（cstart*n*.asm）注
・空の main 関数（main*n*.c）注
・I/O ヘッダ・ファイル（iodefine.h）

注:ファイル名の末尾 *n* はPE番号を意味します。

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（cstart*n*.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。

4.1.1　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（cstart*n*.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。以降では、PE1 のアプリケーション用スタートアップルーチン（cstart1.asm）について説明しますが、他 PE のスタートアップルーチンも同様に読み替えてください。

(1) スタック領域の確保

各 PE 毎に、スタック領域を 0x200 バイト分確保しています。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:
```

(2) 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
```

(3) アプリケーションプロジェクトのエントリルーチン

ブート・ローダ用スタートアップルーチン（boot1.asm）の.L.entry_PE1 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。

セクション配置

エントリルーチンを.text.cmn セクションに配置します。

```
;------------------------------------------------------------------------
;               startup
;------------------------------------------------------------------------
                .section  ".text.cmn", text
                .public   __cstart_pm1
                .align    2
__cstart_pm1:
```

-fsymbol オプションの引数に.text.cmn を指定することにより、外部定義シンボル__cstart_pm1 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*.fsy をブート・ローダプロジェクトに入力して一緒にビルドすることにより、ブート・ローダプロジェクトで__cstart_pm1 のアドレスを共有することが可能となります。つまり、boot1.asm 内の__cstart_pm1 のアドレス解決に使用します。

レジスタの設定

SP/GP/EP レジスタに値を設定します。

```
        mov     #_stacktop, sp          ;   set sp register
        mov     #__gp_data, gp          ;   set gp register
        mov     #__ep_data, ep          ;   set ep register
```

__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。FPU を実装していない PE 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

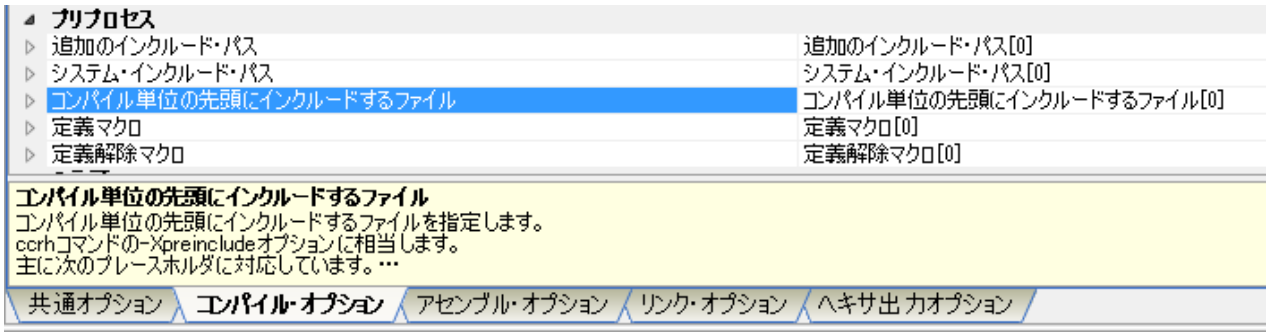
また、自身への分岐を繰り返すルーチン_exitのアドレス(#_exit)をlpに、PE1用main関数の先頭アドレス(#_main)をFEPCレジスタに設定します。その後、feret命令の実行によってFEPSWレジスタ値をPSWに、FEPCレジスタ値をPCに反映しmain関数へ遷移します。

```
        ;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
        feret
```

4.1.2 I/Oヘッダ・ファイル

新規マルチコア用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、アプリケーション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 指定する必要がありません。-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]で指定可能です。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。



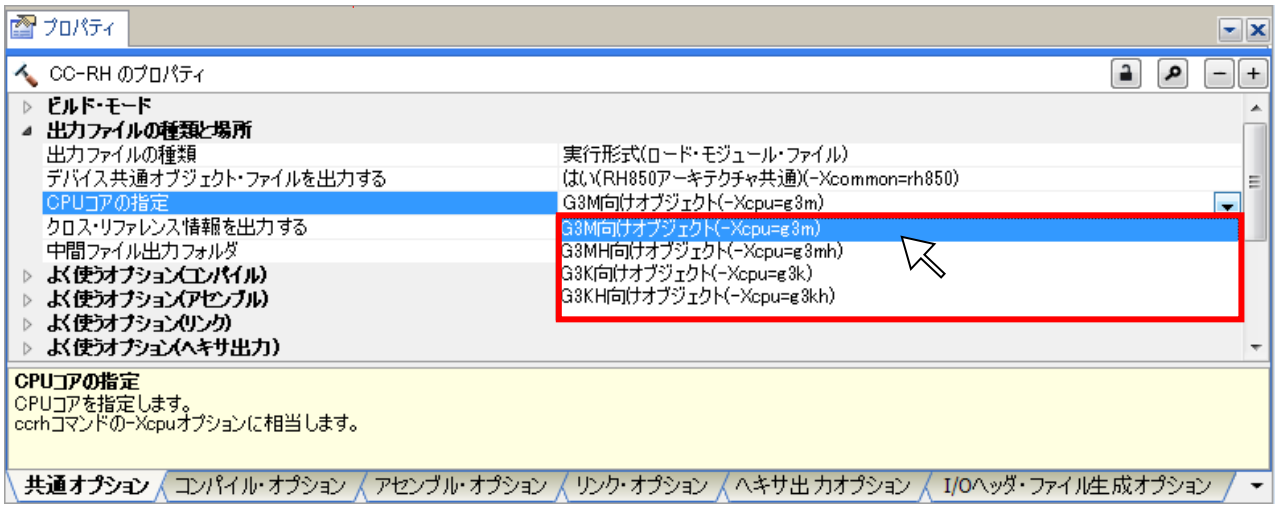
4.2 オプション設定

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。

4.2.1 コンパイラオプション

-Xcpu オプション

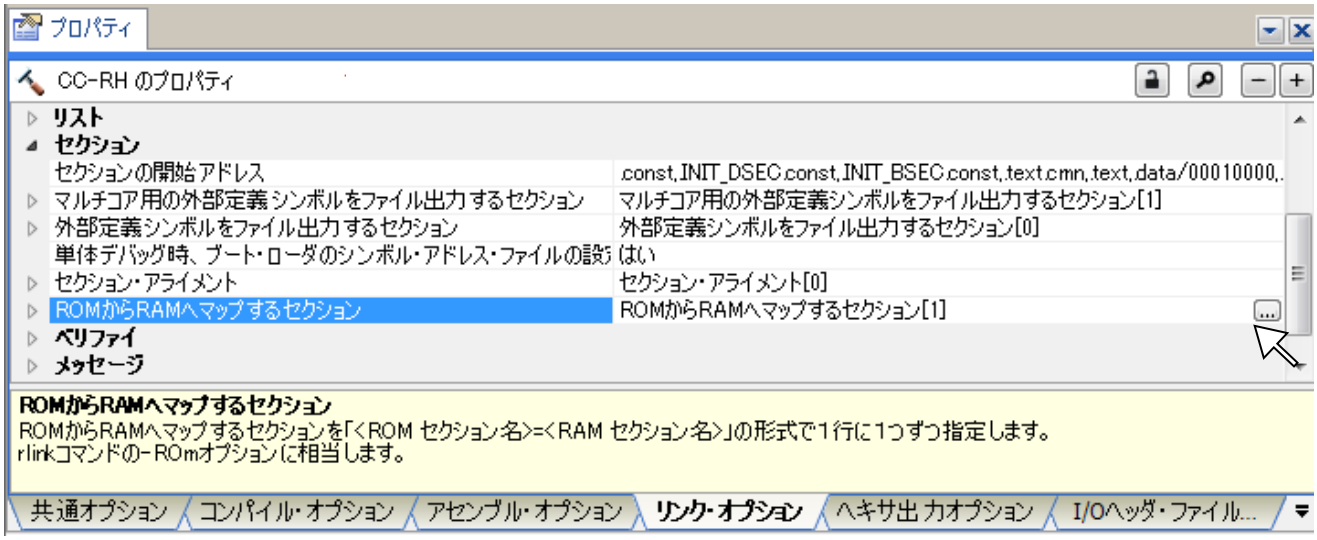
CPU コアを指定するオプションです。指定したコア向けのオブジェクトを生成します。[共通オプション]タブ=>[出力ファイルの種類と場所]カテゴリ=>[CPU コアの指定]で[G3M 向けオブジェクト]／[G3MH 向けオブジェクト]／[G3K 向けオブジェクト]／[G3KH 向けオブジェクト]のいずれかを指定してください。デフォルトでは[G3M 向けオブジェクト]が指定されています。



4.2.2 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。



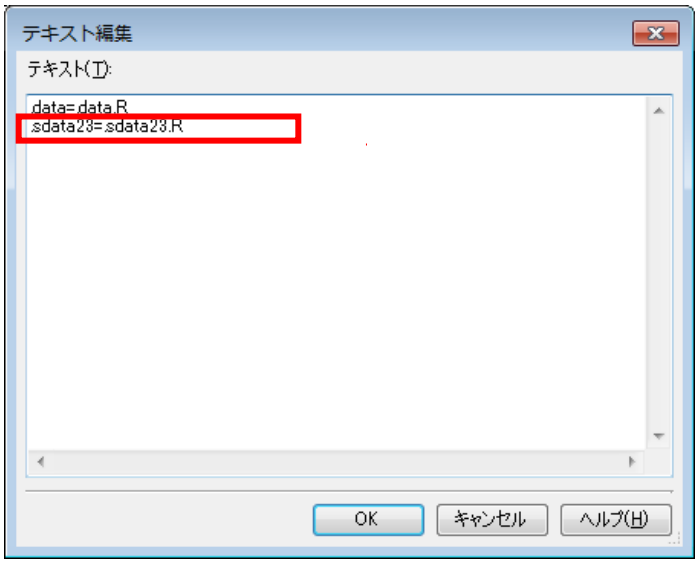
*<ROMセクション名>*がROM化対象のセクションとなります。-startオプションによって、*<ROMセクション名>*で指定したセクションはROMに配置指定し、*<RAMセクション名>*で指定したセクションはRAMに配置指定してください。

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、アプリケーション用スタートアップルーチンcstart*n*.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。

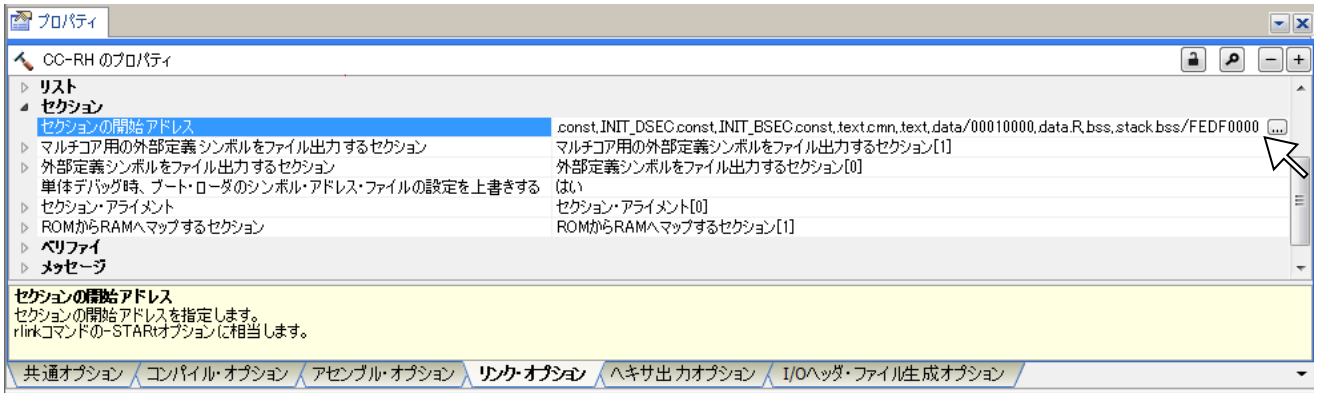
【例】.sbss23 を新たに追加した場合

```
                .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]で指定してください。

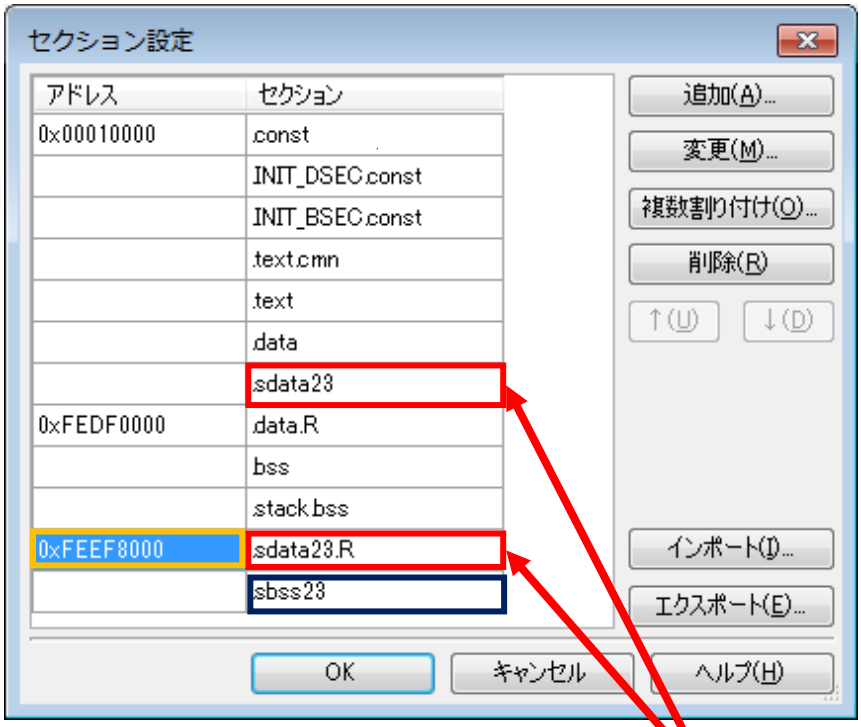


右端の[...]ボタンを押下すると、セクション設定ダイアログが起ち上がります。PE1用アプリケーションプロジェクトの場合はデフォルトでは以下のように指定されています。

この指定は 0x10000 番地から上位方向に.const => INIT_DSEC.const => INIT_BSEC.const => .text.cmn => .text => .data セクションを配置し、0xFEDF0000 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。0xFEDF0000 番地はローカル RAM セルフ領域の先頭アドレスを想定しています。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。

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。

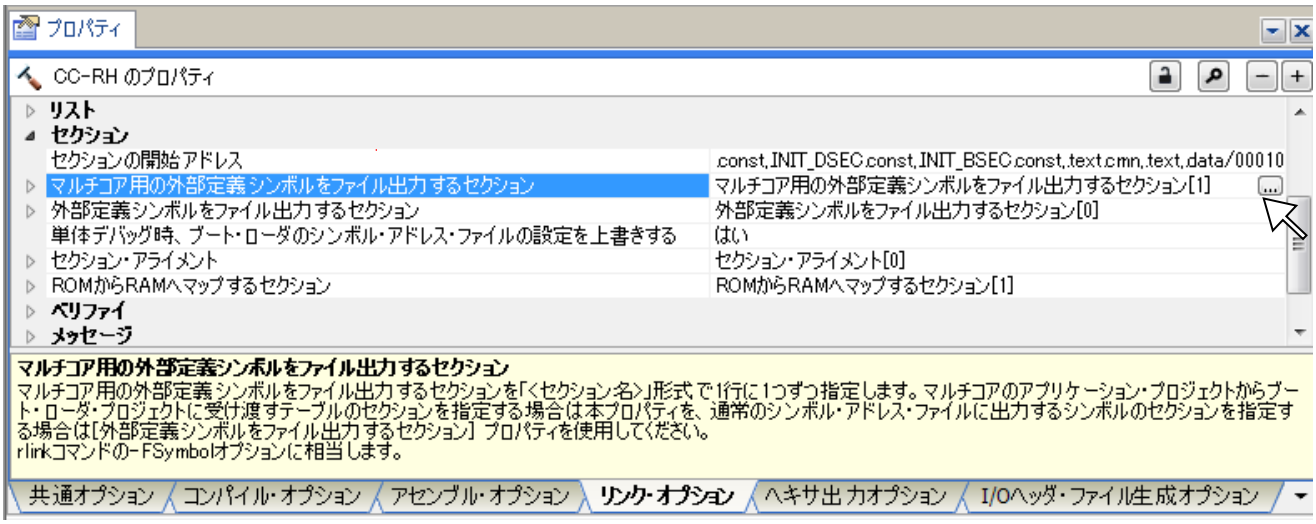
【例】.sdata23／.sbss23 を新たに追加し（-rom=.sdata23=.sdata23.R 指定）、これらの変数をグローバル RAM 領域である 0xFEEF8000 番地に配置指定する場合



-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]で指定してください。



[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、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.text.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__cstart_pm1 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 cstart1.asm】

```
;------------------------------------------------------------------------------
;               startup
;------------------------------------------------------------------------------
                .section   ".text.cmn", text
                .public    __cstart_pm1
                .align     2
__cstart_pm1:
```

【*.fsy ファイルの出力例】

```
;SECTION NAME = .text.cmn
  .public __cstart_pm1
__cstart_pm1 .equ 0x10500
```

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、__cstart_pm1 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。

.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。

4.3 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PE2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ソースファイルでプロジェクトPE2と共有する変数のセクション名は変更してください。

【例】val1 を com.data セクションに、val2 を com.bss セクションに配置させる場合

```
#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
```

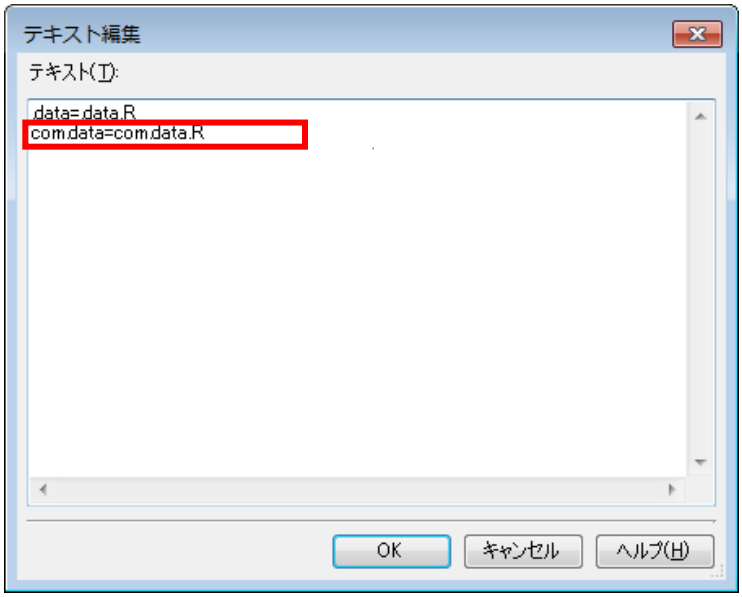
(2) ROM 化

-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。

CS+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。

【例】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
```

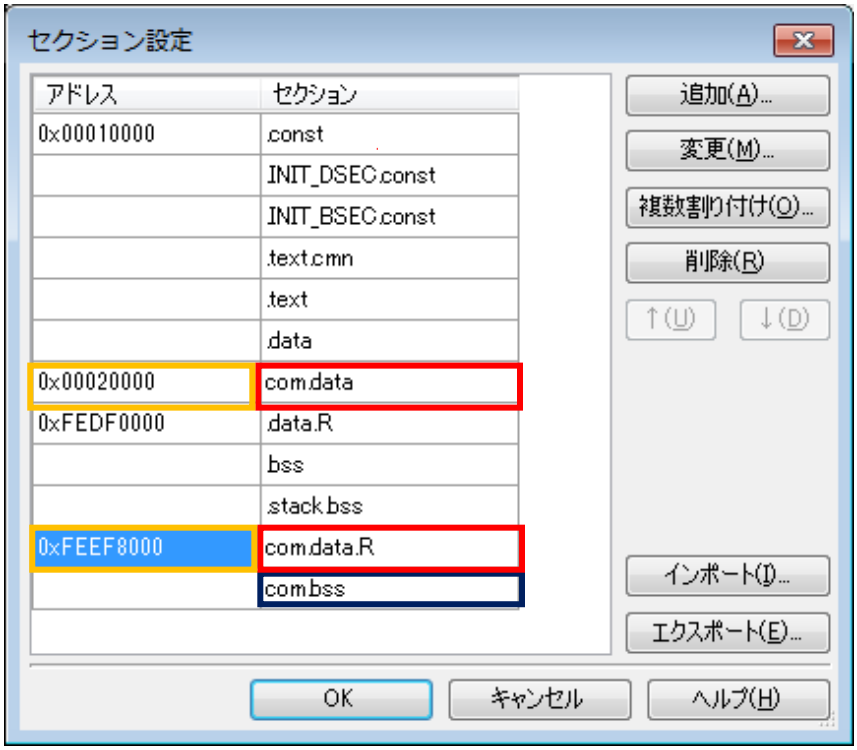


(3) セクション配置

-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。

CS+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 0x20000 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 0xFEEF8000 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加

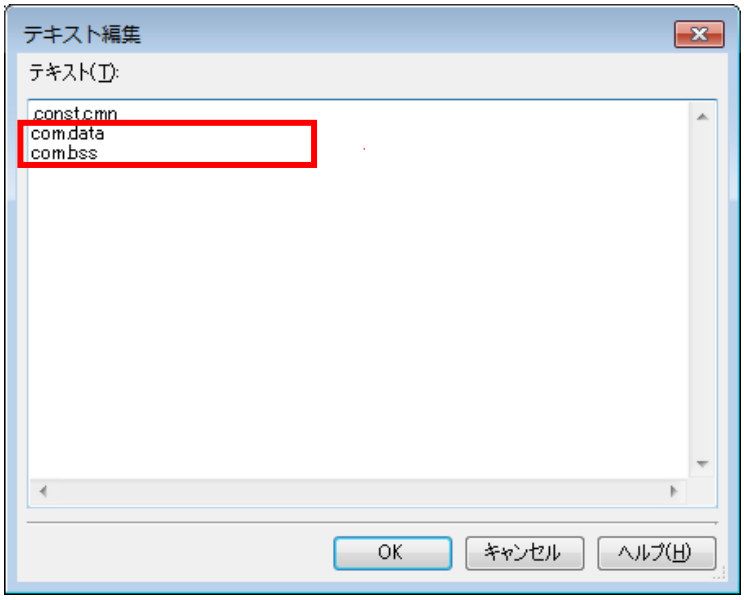
cstartm.asm 内で定義しているセクション初期化テーブルに、追加したセクションの先頭ラベルのアドレス値(#__sセクション名)と終端ラベルのアドレス値(#__eセクション名)を追記してください。

```
        .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。

CS+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0xFEEF8000番地に、"_val2"が0xFEEF8004番地に配置されていることを示しています。

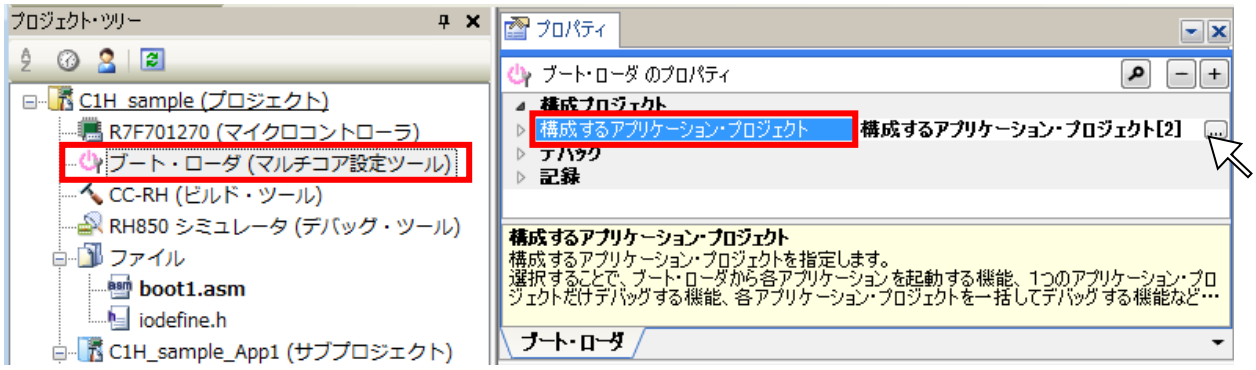
```
;SECTION NAME = .text.cmn
  .public __cstart_pm1
__cstart_pm1 .equ 0x10500
;SECTION NAME = com.data
  .public _val1
_val1 .equ 0xfeef8000
;SECTION NAME = com.bss
  .public _val2
_val2 .equ 0xfeef8004
```

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。

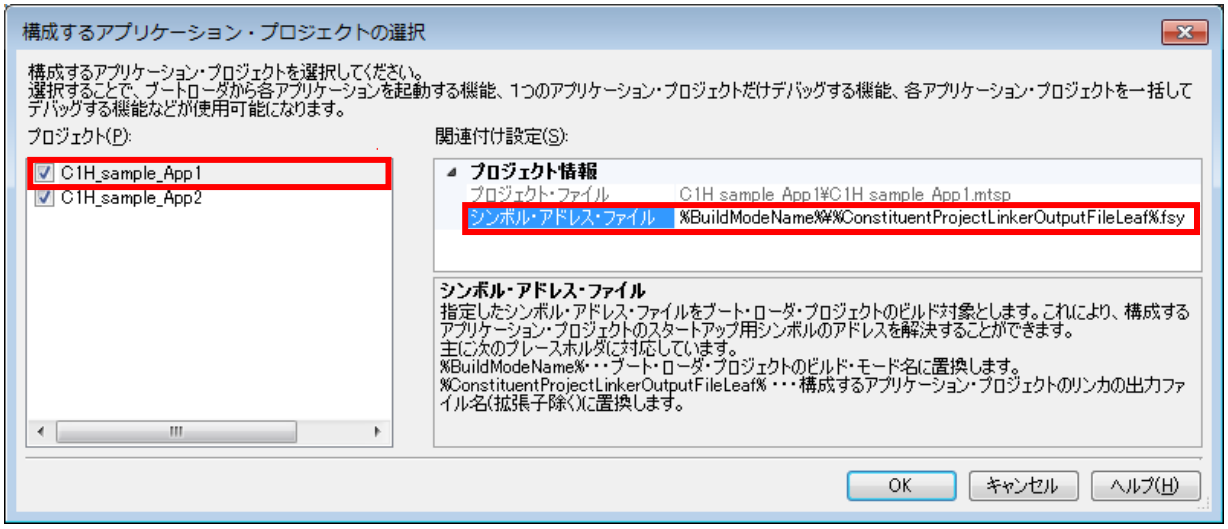
(6) *.fsy ファイルの入力

外部定義シンボル"_val1","_val2"を参照するプロジェクトPE2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。

ブート・ローダプロジェクトの場合、アプリケーションプロジェクトで生成した*.fsyファイルがデフォルト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下にある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。



4.4 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PE2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ソースファイルで、プロジェクトPE2と共有する関数のセクション名を変更してください。

【例】関数 func を com.text セクションに配置させる場合

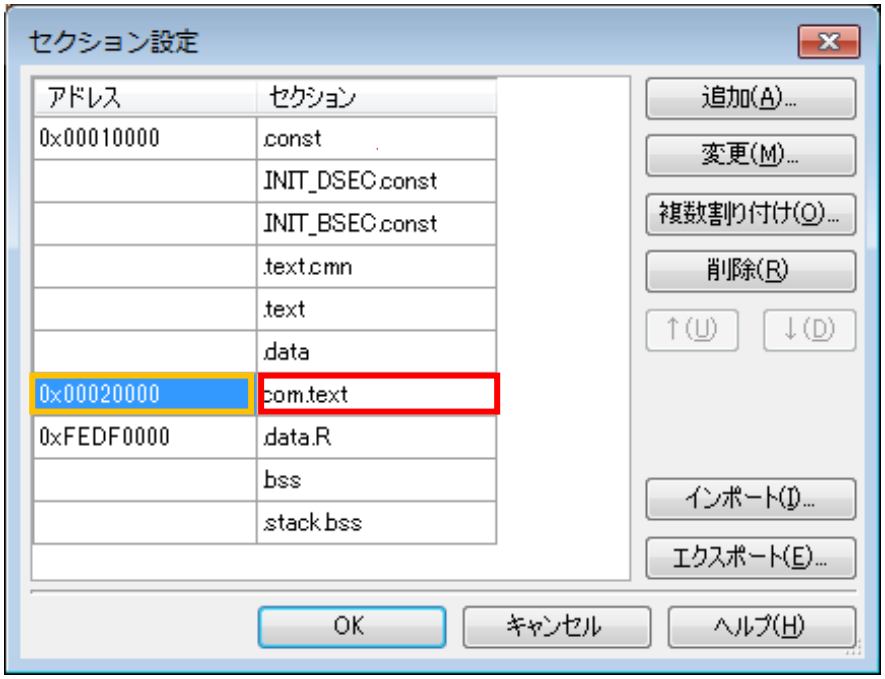
```
#pragma section text "com"

void func (void) {
  ・・・
}

#pragma section default
```

(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。

CS+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。

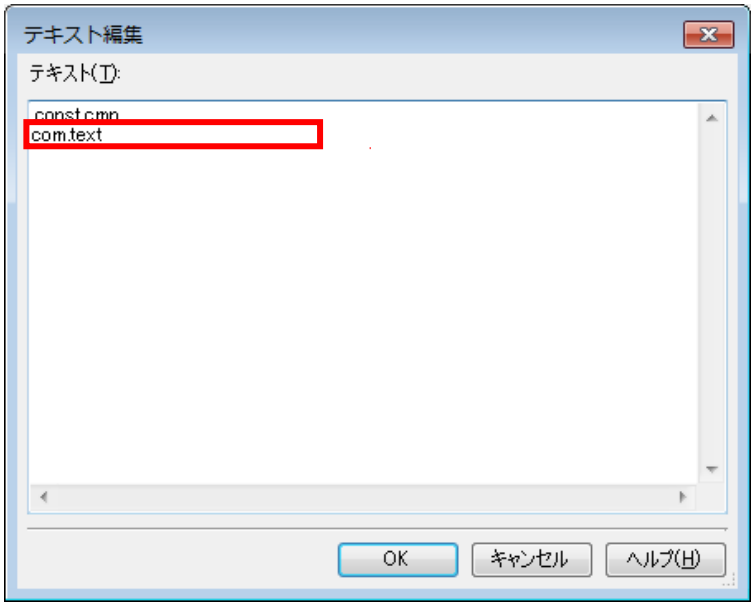
【例】com.text セクションを 0x20000 番地に配置指定する場合



(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。

CS+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0x20000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 = .text.cmn
  .public __cstart_pm1
__cstart_pm1 .equ 0x10500
;SECTION NAME = com.text
  .public _func
_func .equ 0x20000
```

(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PE2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。ブート・ローダプロジェクトの場合、アプリケーションプロジェクトで生成した*.fsyファイルがデフォルトで登録されています。

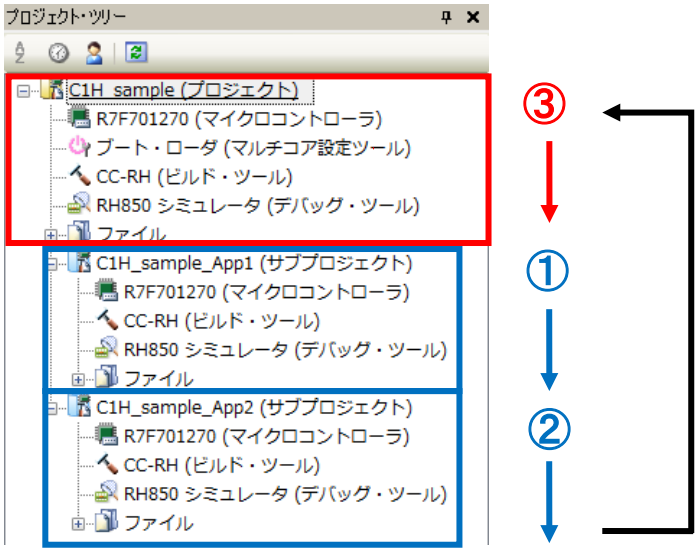
第5章　　リビルド

本章では、マルチコア用プロジェクト(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。

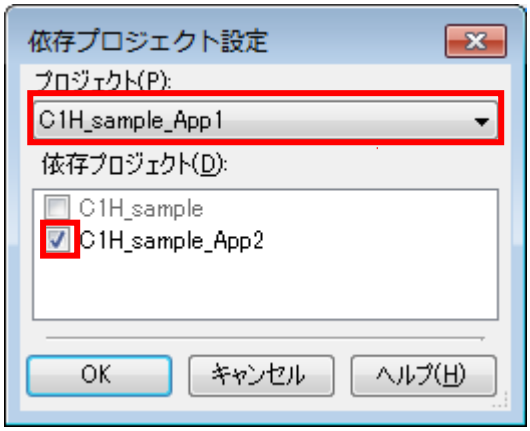
5.1　複数プロジェクトのリビルド

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アプリケーション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。

CS+でリビルドすると、デフォルトではPE1(C1H_sample_App1)=>PE2(C1H_sample_App2)=>ブート(C1H_sample)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、PE2に登録すると、リビルド時にはPE2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度のリビルドで済みます。

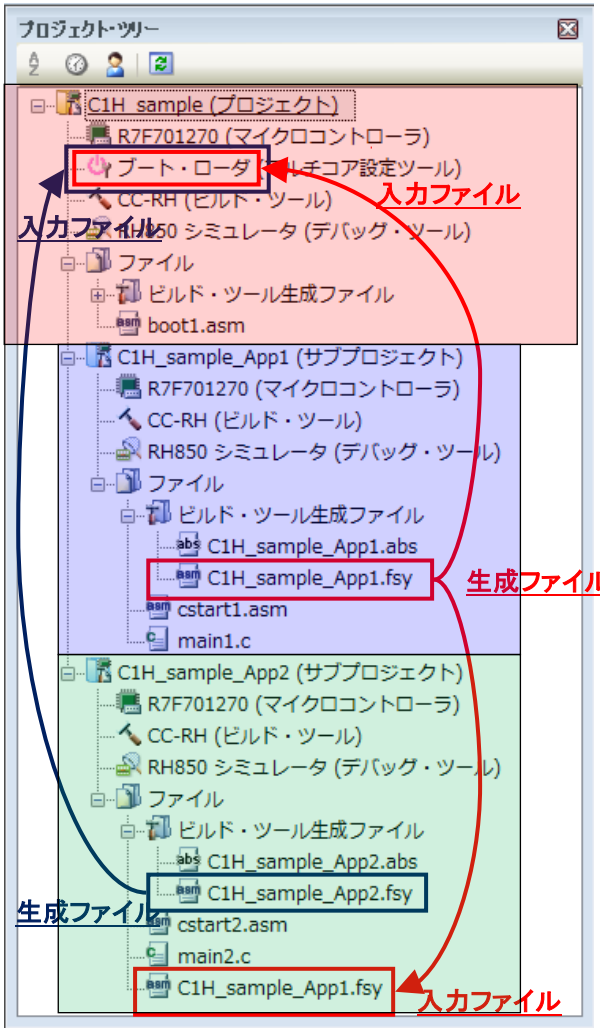


なお、CS+ではプロジェクトのビルド順を制御することが可能です。[プロジェクト]メニュー=>[依存プロジェクト設定]を押下し、[依存プロジェクト設定]ダイアログ上に設定します。



このダイアログ設定は、C1H_sample_App1(PE1)はC1H_sample_App2(PE2)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PE2=>PE1=>ブート・ローダ(C1H_sample)の順番でリビルドします。

アプリケーションプロジェクトのリビルドの流れを以下に示します。



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 C1H_sample_App1(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プロジェクト用エントリルーチン(__cstart_pm1)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(C1H_sample_App1.fsy) が生成されます。

② 続いてプロジェクト C1H_sample_App2(PE2)がリビルドされます。C1H_sample_App1.fsy を PE2 へ入力することによって、PE2 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、PE2 で定義しているアプリケーションプロジェクト用エントリルーチン(__cstart_pm2)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(C1H_sample_App2.fsy) が生成されます。

③ 最後にブート・ローダ(C1H_sample)がリビルドされます。C1H_sample_App1.fsy と C1H_sample_App2.fsy をブート・ローダに入力することによって、PE1 と PE2 で定義しているアプリケーションプロジェクト用エントリルーチンの配置アドレスを解決します。

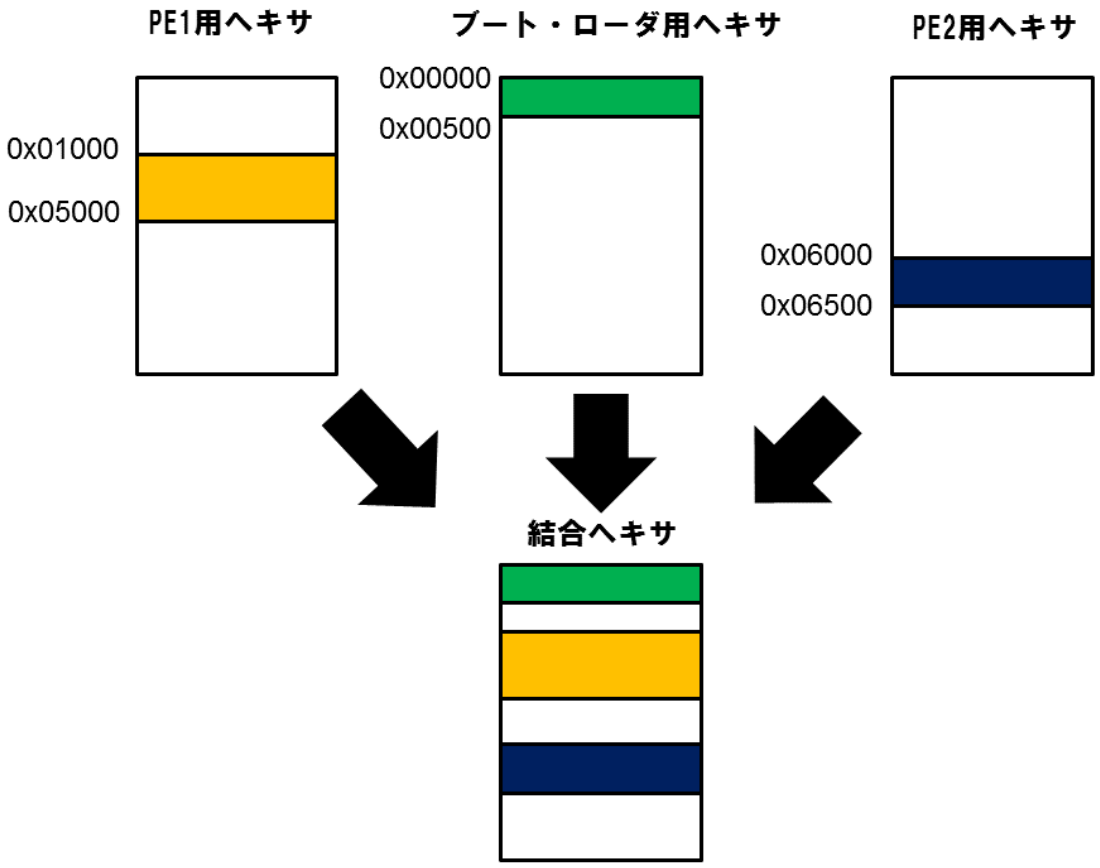
第6章　　オブジェクト結合

本章では、マルチコア用プロジェクトで作成した3つのヘキサ・ファイルを結合して1つのヘキサ・ファイルを作成する手順について説明します。

6.1　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。

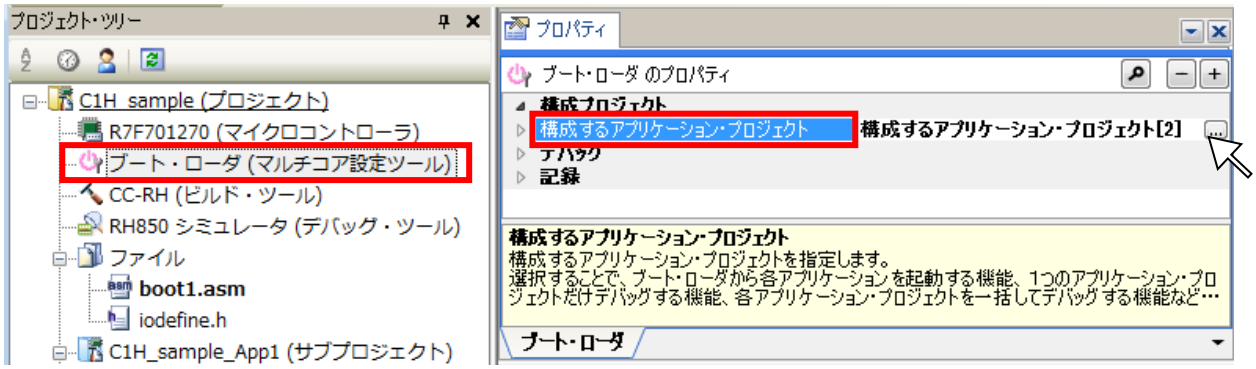
なお、複数のヘキサを結合する際、アドレスが重複している場合はオーバラップエラーが発生します。ただし、データが存在しないRAM領域のチェック出来ませんので、お客様自身でマップファイルを参照する等してオーバーラップしていないかチェックしてください。セクションの割り付けアドレスをチェックする”-cpu”オプション等でチェック可能です。



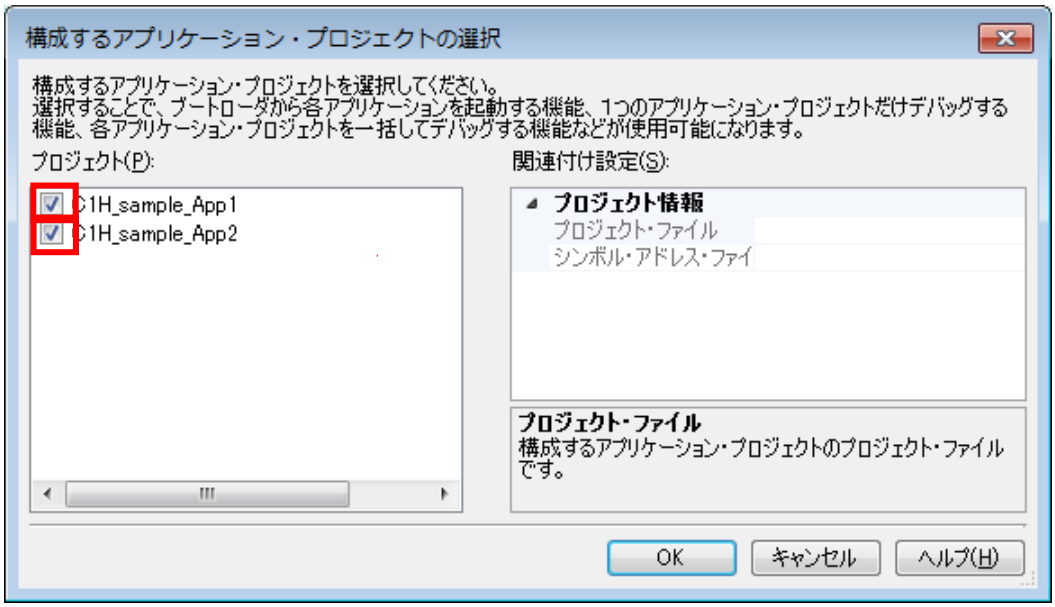
6.2 構成アプリケーションプロジェクトの選択

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。登録することにより、ブート・ローダから各アプリケーションを起動する機能、1つのアプリケーションプロジェクトだけをデバッグする機能、各アプリケーションプロジェクトを一括してデバッグする機能等が使用可能になります。

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。



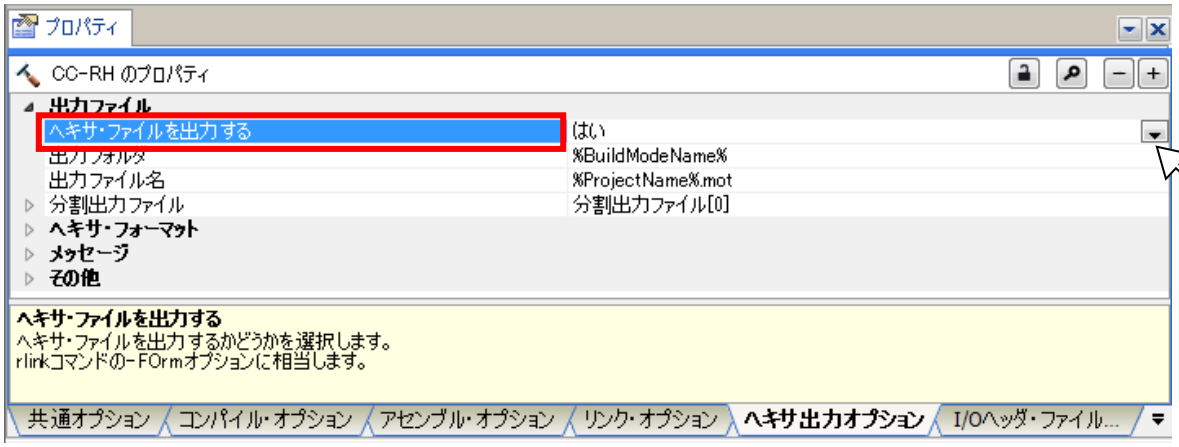
右端の[…]ボタンをクリックしてください。以下のような[構成するアプリケーション・プロジェクトの選択]ダイアログが起ち上がります。アプリケーションプロジェクト"C1H_sample_App1"と" C1H_sample_App2"にチェックが入っており、ブート・ローダプロジェクト(C1H_sample)の構成アプリケーションとして登録されています。



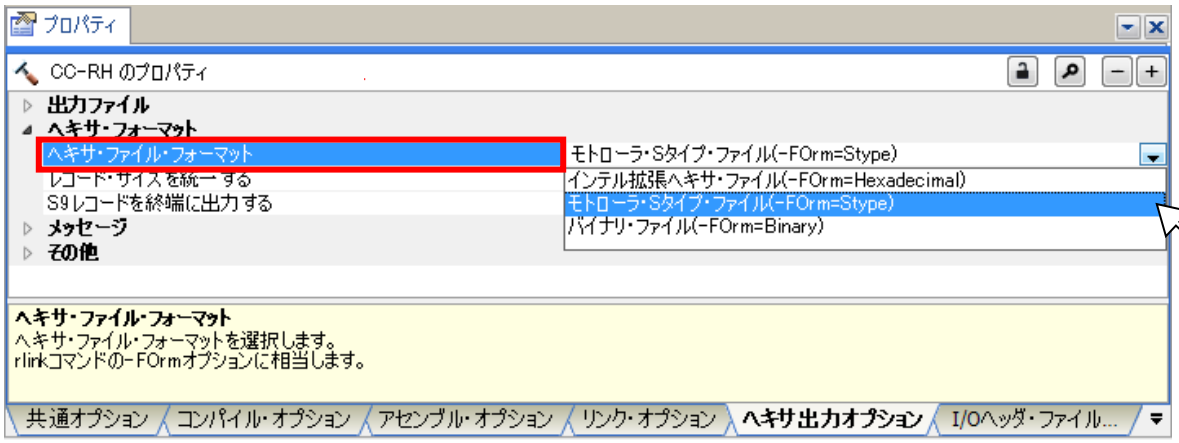
6.3 オブジェクトの結合

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PE1とPE2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。

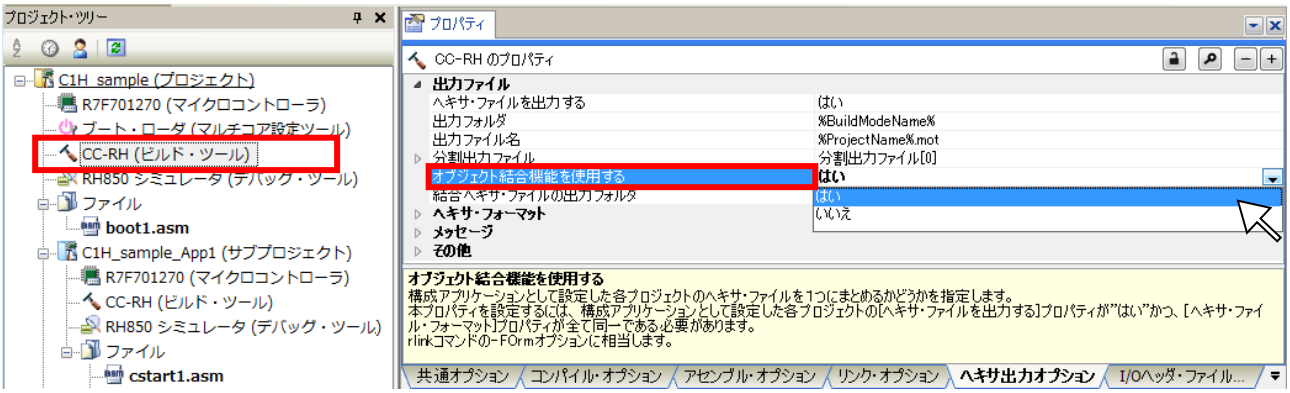
ブート・ローダプロジェクト、および各アプリケーションプロジェクトにおいて、ヘキサ・ファイルの出力およびヘキサ・フォーマットの設定を行ってください。各プロジェクトのビルド・ツールプロパティの[ヘキサ出力オプション]タブ=>[出力ファイル]カテゴリ=>[ヘキサ・ファイルを出力する]で「はい」を選択してください。デフォルトでは「はい」が選択されています。



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[ヘキサ出力オプション]タブ=>[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、各PE用のアプリケーションプロジェクトで生成したヘキサ・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。

改定記録

Rev.	発行日	改定内容	
		ページ	ポイント
1.00	2015.3.31	-	初版発行
1.01	2015.10.20	6, 7, 8, 9, 10	ブート・ローダ用スタートアップルーチン(boot1.asm)の処理を一部変更しました。
		20	-Xcpu=g3kh オプションを追加しました。

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　〒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：http://japan.renesas.com/contact/