ブート-フラッシュ領域の分割方法

RL78、78K0R用 Cコンパイラ CA78K0R 再リンク機能

株式会社ルネサス ソリューションズ
ツールビジネス本部 ツール技術部

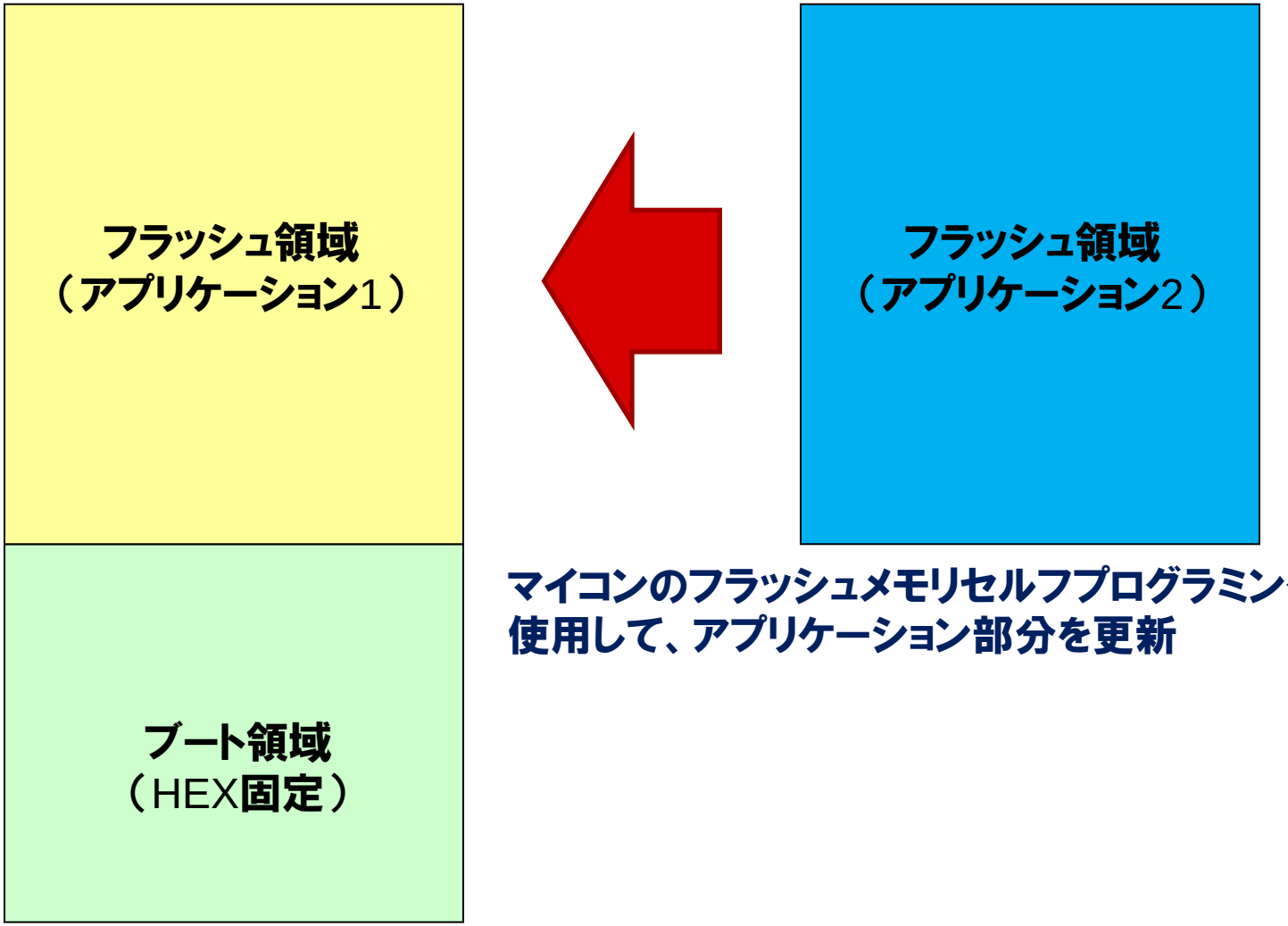
2014/6/20　　Rev. 1.00

R20UT3040JJ0100

- ブートーフラッシュの再リンク機能について
- ブートーフラッシュ領域の配置
  - プロジェクトの作成
  - ブート領域用プログラムの設定
  - フラッシュ領域用プログラムの設定
  - ソースファイルへの記述
  - E1エミュレータのモニタ領域の確保
- ブート領域用とフラッシュ領域用のヘキサ・ファイル
- 初期化フロー
- 分岐テーブルを2000H以外にする場合
- 変数／関数情報ファイル生成ツールを使用する場合
- ブートーフラッシュ領域とミラー元領域によるnear/farの扱い
- 再リンク機能とROM化機能を併用について

ブートーフラッシュの再リンク機能について(1/3)

- システム上でのイメージ



マイコンのフラッシュメモリセルフプログラミング機能を使用して、アプリケーション部分を更新

ブートーフラッシュの再リンク機能について(2/3)

- ブート領域とフラッシュ領域
  - ブート領域：システム上、書き換えが不可能な領域
  - フラッシュ領域：システム上、書き換え／取り換えが可能な領域

- 再リンク機能
  - ブート領域上のプログラムの再構築を行わず、フラッシュ領域上のプログラムのみを変更し、ブート領域とフラッシュ領域間の関数呼び出しを行う機能

- 再リンク機能におけるブート領域とフラッシュ領域間の参照可否

関数

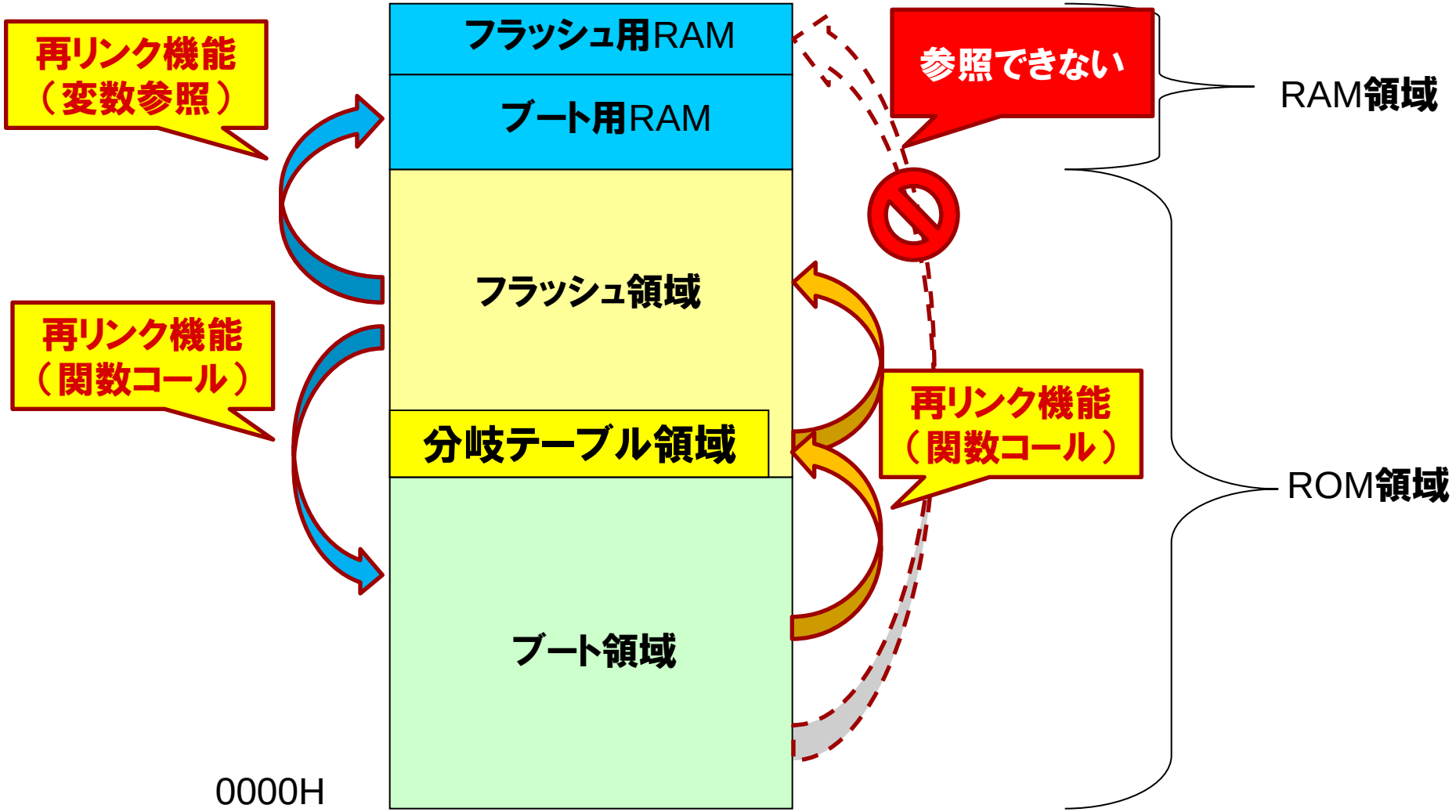
参照元 ＼ 参照先	ブート領域	フラッシュ領域
ブート領域	○	○（再リンク機能）
フラッシュ領域	○（再リンク機能）	○

外部変数

参照元 ＼ 参照先	ブート領域	フラッシュ領域
ブート領域	○	×（アクセスできません）
フラッシュ領域	○（再リンク機能）	○

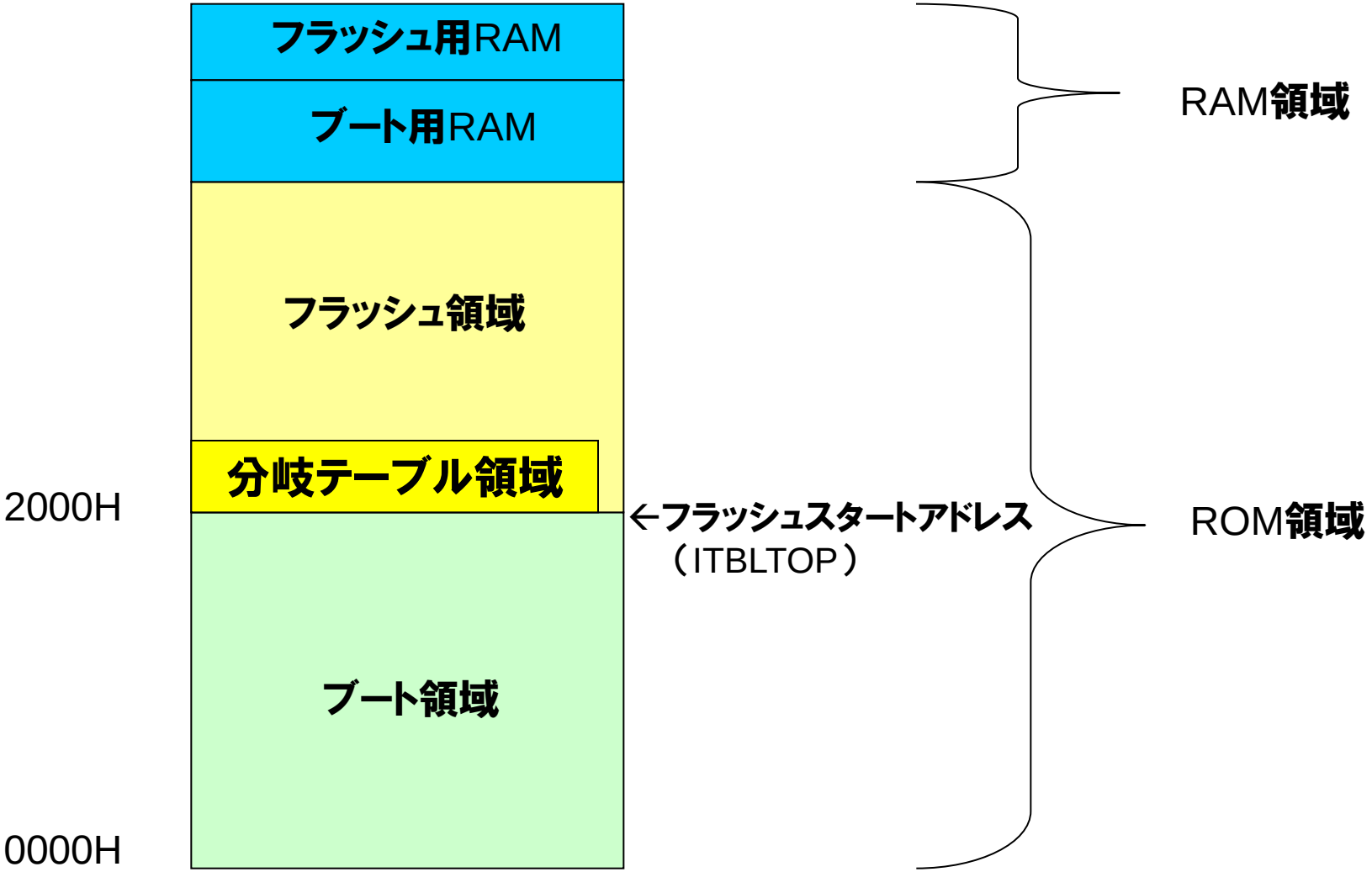
ブートーフラッシュの再リンク機能について(3/3)

(例)ブートーフラッシュ領域間の変数、関数の参照



ブートーフラッシュ領域の配置

（例）次のようにブート領域とフラッシュ領域に配置



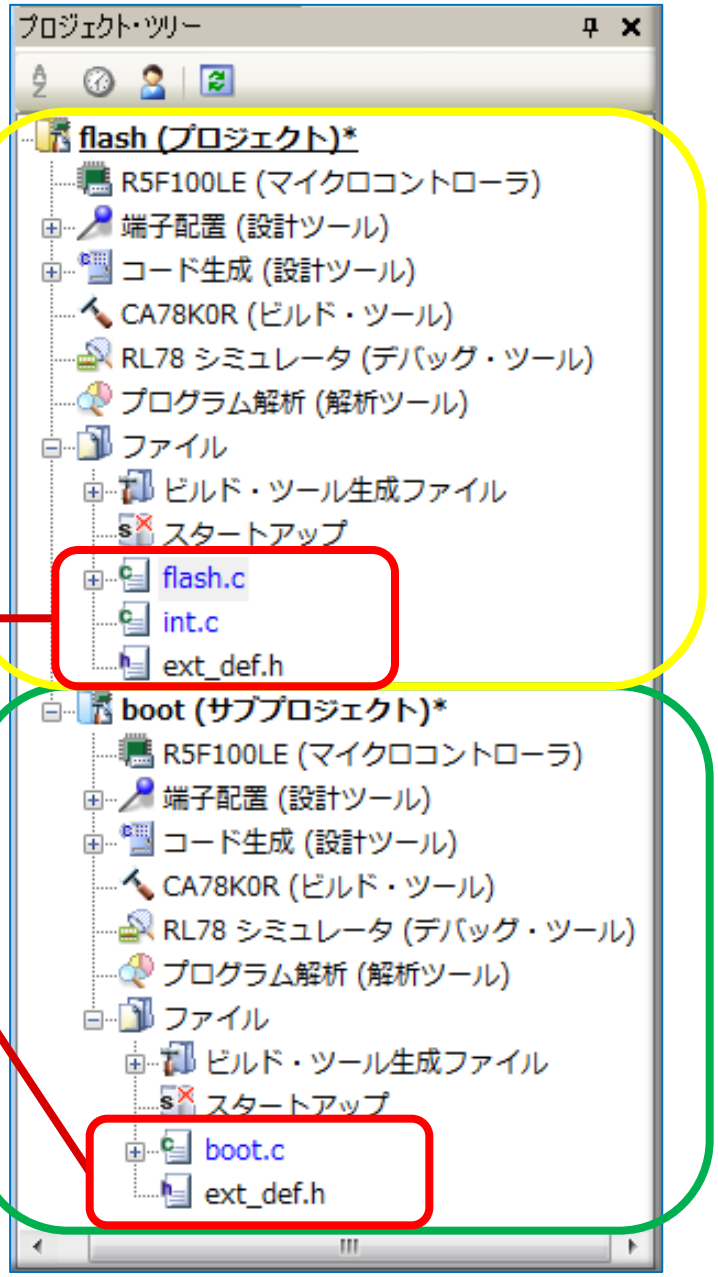
再リンク機能(ブート-フラッシュ機能)を使用するために

- ブート領域用のプロジェクトの作成
  - ブート領域とするためのオプション設定
    - コンパイラ、リンカ、オブジェクト・コンバータのオプション設定
  - フラッシュ領域のプロジェクトをビルドする際に必要なため、フラッシュ領域用プロジェクトよりも前にビルドが必要
- フラッシュ領域用のプロジェクトの作成
  - フラッシュ領域とするためのオプション設定
    - コンパイラ、リンカ、オブジェクト・コンバータのオプション設定
- ソースファイルに、ブート領域から呼び出すフラッシュ領域関数の指定
  - #pragma ext_func指令の記述
- E1エミュレータを使用する場合
  - デバッグ用モニタ領域の確保
- 分岐テーブルwO2000H番地以外にする場合
  - スタートアップルーチンの再構築

プロジェクトの作成

- **プロジェクトの作成※**
  - **メインプロジェクト**
    - **フラッシュ領域のプロジェクト**
  - **サブプロジェクト**
    - **ブート領域のプロジェクト**

- **ビルド対象ファイルの追加**



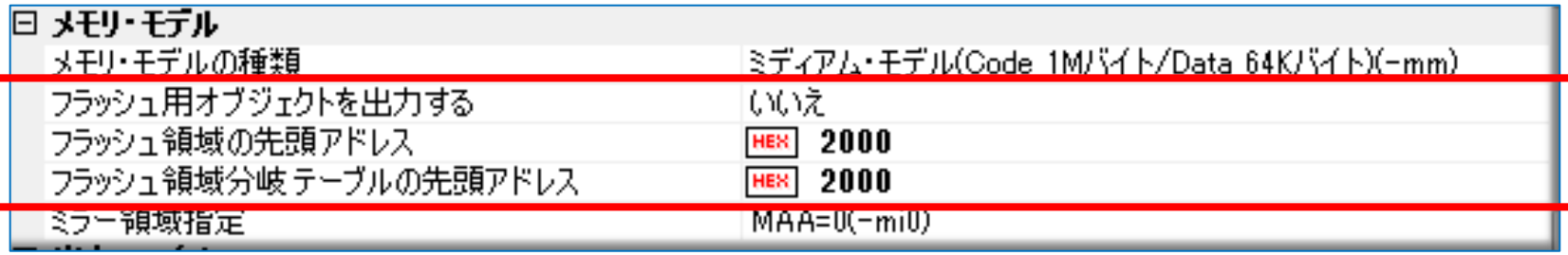
※補足

(1) CubeSuite+**のビルド順は、**
「サブプロジェクト」→「メインプロジェクト」

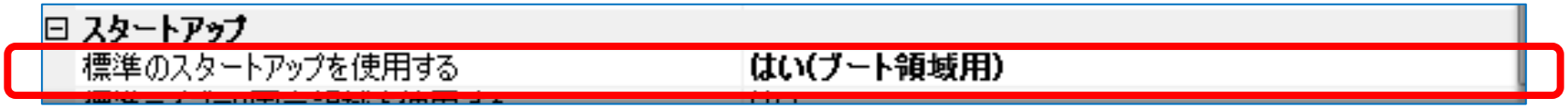
(2) **ブート領域のプログラムは**1**度作成したら変更しないため、フラッシュ領域の**2**世代目以降の作成時には、サブプロジェクトを削除することが可能**

ブート領域用プログラムの設定(1/4)

- [コンパイルオプション]タブの「メモリモデル」
  - 「フラッシュ用オブジェクトを出力する」で「いいえ」を選択(デフォルト)
  - 「フラッシュ領域の先頭アドレス」の入力
  - 「フラッシュ領域分岐テーブルの先頭アドレス」の入力



- [コンパイルオプション]タブの「スタートアップ」
  - 「標準のスタートアップを使用する」で「はい(ブート領域)」を選択

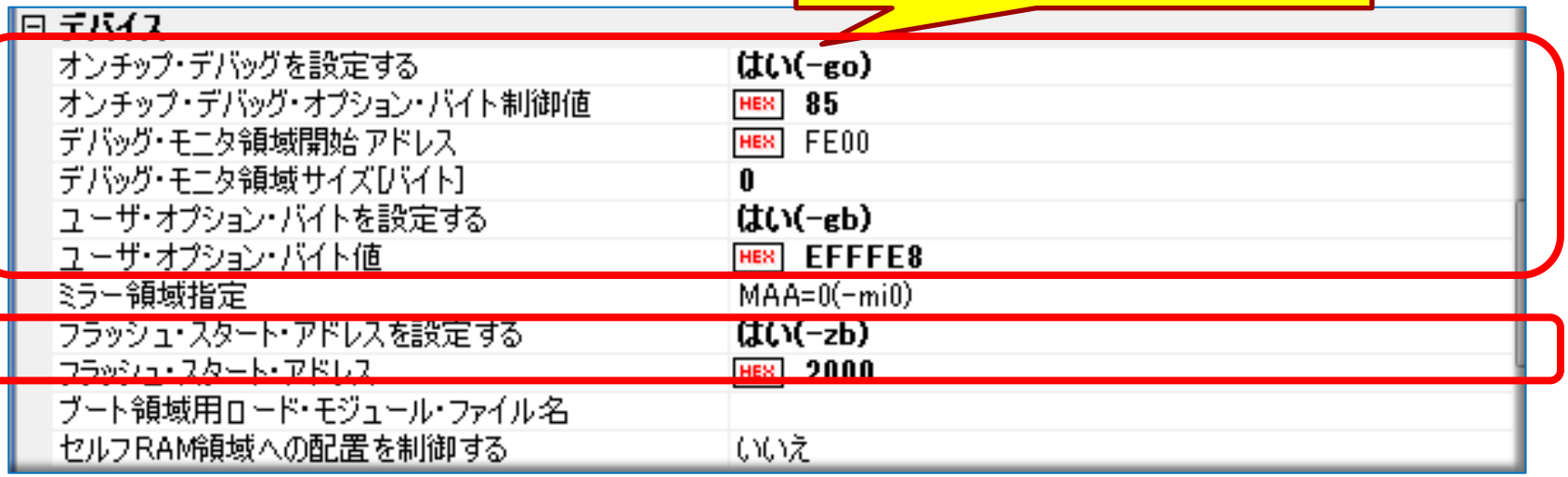


ブート領域用プログラムの設定(2/4)

- [リンクオプション]タブの「デバイス」
  - 「フラッシュスタートアドレスを設定する」で「はい」を選択
  - 「フラッシュスタートアドレス」を入力(コンパイラオプションと同じにする)
- [リンクオプション]タブの「デバイス」(E1エミュレータを使用する場合)
  - 「オンチップデバッグを設定する」で「はい」を選択
  - 「オンチップデバッグオプションバイト制御値」を設定
  - 「デバッグモニタ領域開始アドレス」を入力(デフォルト値使用)
  - 「デバッグモニタサイズ」に「0」を入力
- [リンクオプション]タブの「デバイス」
  - 「ユーザオプションバイトを設定する」で「はい」を選択
  - 「ユーザオプションバイト値」を設定

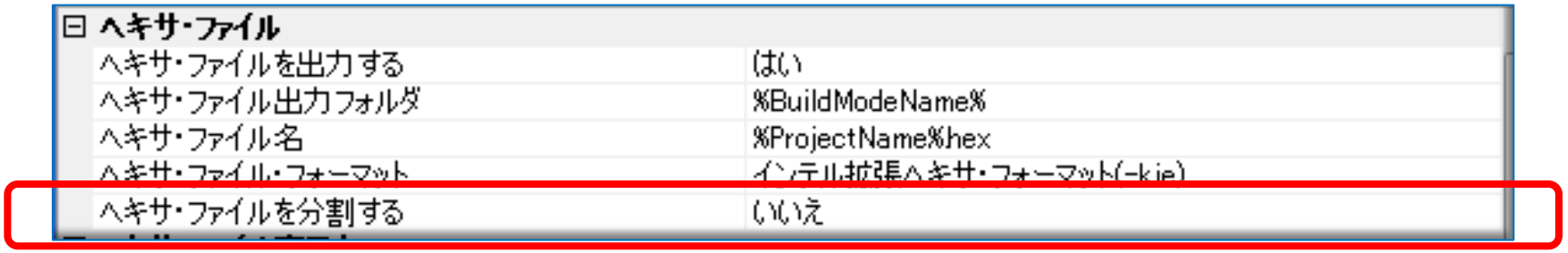
ブート領域用プログラムの設定(3/4)

E1エミュレータを使用する際に設定する



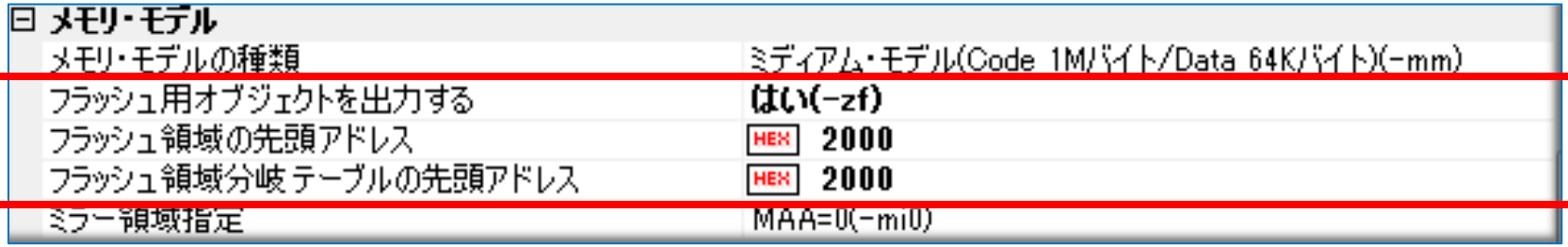
ブート領域用プログラムの設定(4/4)

- [オブジェクトコンバートオプション]タブの「ヘキサファイル」
  - 「ヘキサファイルを分割する」で「いいえ」を選択

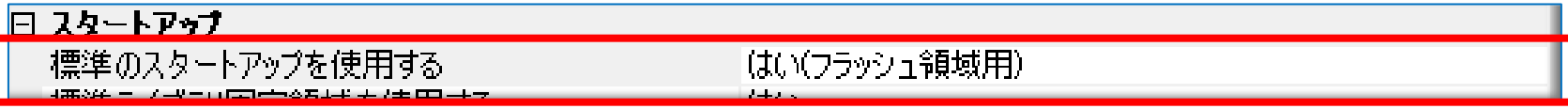


フラッシュ領域用プログラムの設定(1/3)

- [コンパイルオプション]タブの「メモリモデル」
  - 「フラッシュ用オブジェクトを出力する」で「はい」を選択
  - 「フラッシュ領域の先頭アドレス」の入力(ブート領域用プロジェクトと同じにする)
  - 「フラッシュ領域分岐テーブルの先頭アドレス」の入力(ブート領域用プロジェクトと同じにする)

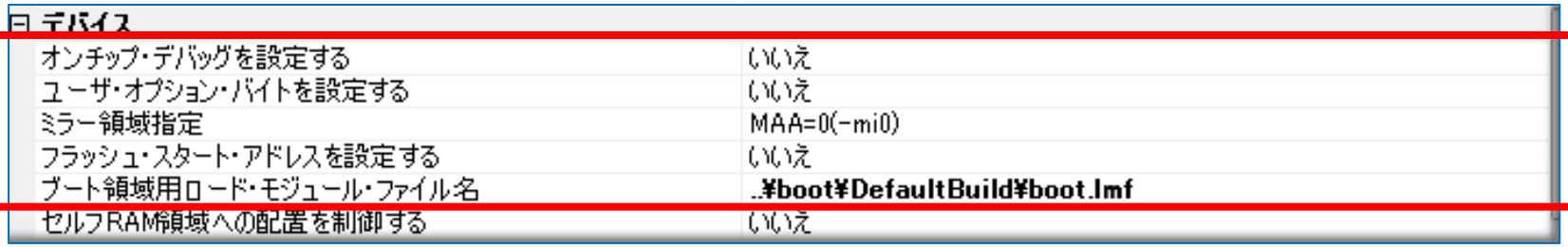


- [コンパイルオプション]タブの「スタートアップ」
  - 「標準のスタートアップを使用する」で「はい(フラッシュ領域用)」を選択



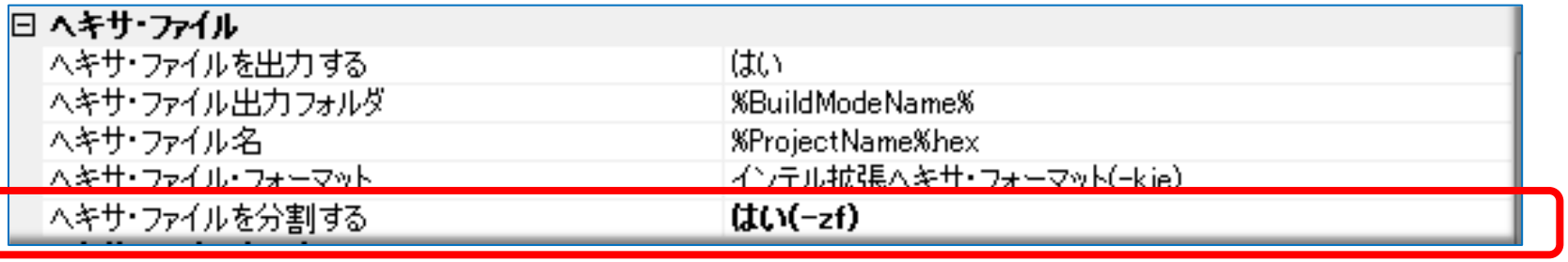
フラッシュ領域用プログラムの設定(2/3)

- [リンクオプション]タブの「デバイス」(E1エミュレータを使用する場合も)
  - 「オンチップデバッグを設定する」で「いいえ」を選択
- [リンクオプション]タブの「デバイス」
  - 「ユーザオプションバイトを設定する」で「いいえ」を選択
- [リンクオプション]タブの「デバイス」
  - 「ブート領域用ロードモジュールファイル名」でブート側のリンカ出力の.lmfを指定



フラッシュ領域用プログラムの設定(3/3)

- [オブジェクトコンバートオプション]タブの「ヘキサファイル」
  - 「ヘキサファイルを分割する」で「はい」を設定



ソースファイルへの記述(1/2)

- **#pragma ext_func指令の記述**
  - **実体がフラッシュ領域に存在し、ブート領域から呼び出される関数に対して、ID値を指定**
  - #pragma ext_func**指令は、ブート領域の参照、および、フラッシュ領域の定義の両領域のプログラムへの記述が必要**
  - **推奨**
    - **記述漏れやソースファイル間の矛盾が生じることを防ぐため、**#pragma ext_func**指令の記述は1つのファイルにまとめ、ブート領域、フラッシュ領域を問わず、全Cソース・ファイルにインクルードすることを推奨します。**

ソースファイルへの記述(2/2)

（例）次のようなソースをブート領域とフラッシュ領域に配置

ブート領域

```
------ boot.c-----
#include "ext_def.h"

int boot_a = 0x12;
int boot_b = 0x34;

extern int f1 ( int ) ;
extern int f2 ( int ) ;


void boot_main (void )
{

}

void func(void)
{
  int k;
  boot_a = f1 ( boot_a ) ;
  boot_b = f2 ( boot_b ) ;
}
```

```
------ ext_def.h -----
#pragma ext_func f1 1
#pragma ext_func f2 2
```

ブート領域から呼び出す
フラッシュ領域の関数を定義

```
------ int.c -----
#include "ext_def.h"


#pragma interrupt INTP0 int_INTP0

 char        f;

void int_INTP0 ( )

{

/* 割り込み処理*/
                f = 1;
}
…
```

フラッシュ領域

```
------ flash.c -----
#include "ext_def.h"


int flash_a = 0x56;
int flash_b = 0x78;

extern void func(void);

void main(void)
{
                func();
}

int f1(int a)
{
        ….
}
int f2(int b)
{
        ….
}
….
```

E1エミュレータのモニタ領域の確保

- **アセンブラ・ソース・ファイルでのモニタ領域の確保**
  - E1エミュレータを使用する場合には、デバッグ用のモニタ領域を確保する必要があります。モニタ領域が512バイトの場合で説明します。

- **モニタ領域（ROMの最後512バイト）に、“0FFH”を埋め込む**
  - **次のようなファイルをCubeSuite+にソース登録する**

（例）マイコンのROMのサイズが64Kバイトの場合

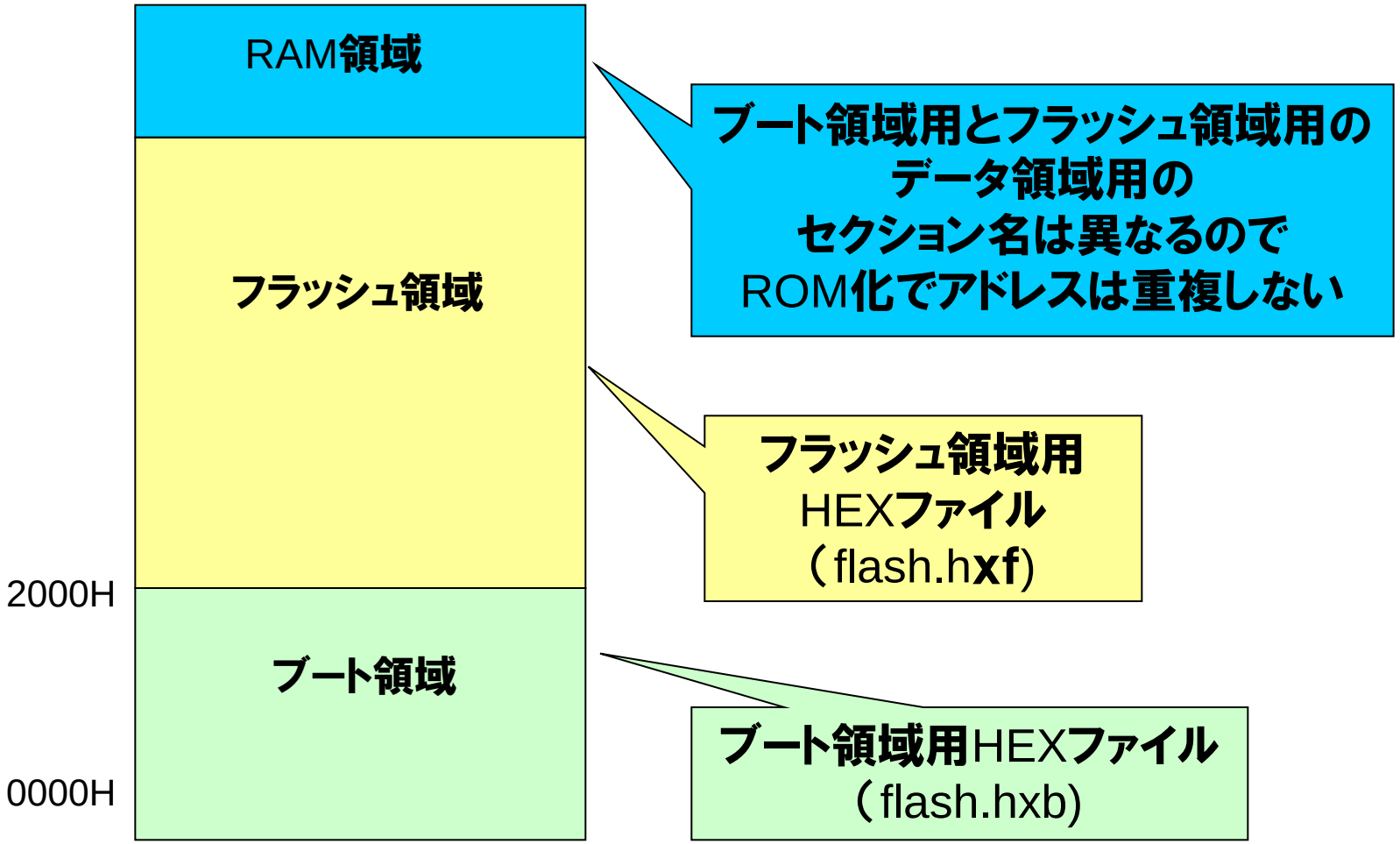
```
E1M   CSEG   AT 0FE00H
      REPT   20H
      DB     0FFH,0FFH,0FFH,0FFH,0FFH,0FFH,0FFH,0FFH,0FFH,0FFH,0FFH,0FFH,0FFH,0FFH,0FFH,0FFH
      ENDM
      END
```

32回繰り返す

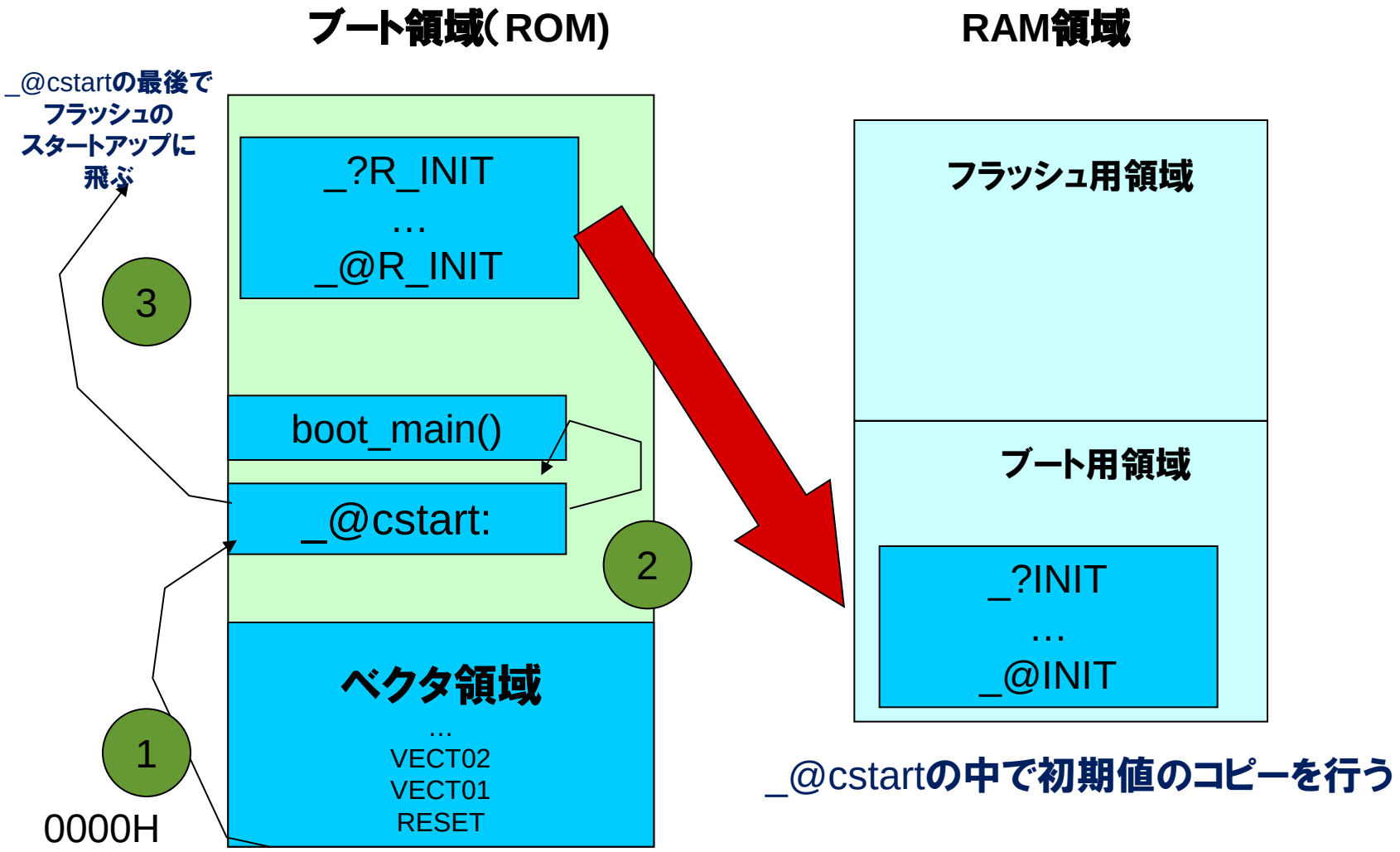
元のROMのサイズから、512バイトを減算した値を開始アドレスとする

16バイト分、0FFHで初期化

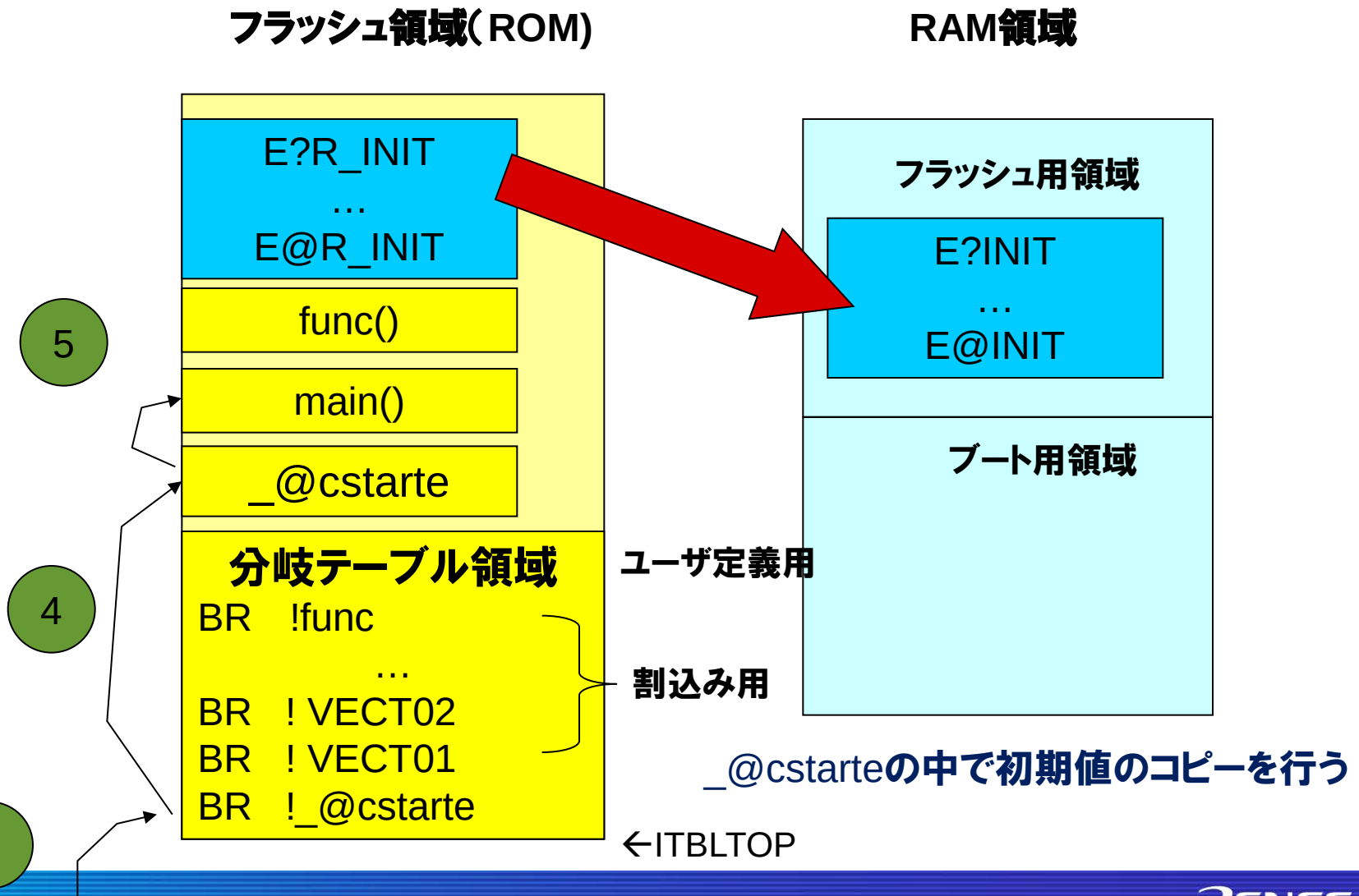
ブート領域用とフラッシュ領域用のヘキサ・ファイル



初期化フロー(1/2)



初期化フロー(2/2)



分岐テーブルを2000H以外にする場合

- 分岐テーブルの先頭アドレスに2000H以外を指定する場合
  - 割込みベクタの処理も変更が必要
    - vect.inc のアドレス値をエディタで変更
    - コマンドプロンプト上で、batファイル「repvect.bat」を実行して、スタートアップ、ライブラリを更新
    - ブート領域、フラッシュ領域のプロジェクトに登録

分岐テーブルを2000H以外にする場合～vect.incの編集

- **システムフォルダからのbatファイルに関するファイルのコピー**
  - CubeSuite+**のインストールフォルダの**
    「¥CubeSuite+¥CA78K0R¥V1.50¥Src¥cc78k0r」**を**
    任意のフォルダにコピーする

- vect.inc**の編集**
  - **コピー先の**「cc78k0r¥src¥vect.inc」**をエディタで編集**

```
ITBLTOP    EQU   2000H
```

分岐テーブルのアドレス※を記述

注意※
16**進数を記述するには、数値の最後に**「H」**を追加する**
（例）2000H
16**進数で最上位桁がアルファベットの場合、**0**を追加する**
（例）0F000H

分岐テーブルを2000H以外にする場合～batファイルの実行(1/2)

- Windowsのコマンドプロンプトを起動する
- コマンドプロンプト上で次の操作をする
  - コピー先のフォルダに移動する

（例） cd *C:¥sample*¥cc78k0r¥bat

コピー先のフォルダ

（例）
  - PATHを追加する（batファイル中のra78k0r.exeを実行するため）

set path=%PATH%;*C:¥Program Files¥Renesas Electronics*¥CubeSuite+¥CA78K0R¥*V1.50*¥bin

CubeSuite+の
インストールフォルダ

CA78K0Rのバージョン

分岐テーブルを2000H以外にする場合～batファイルの実行(2/2)

- batファイルの「repvect」を実行する
  - 次の形式で実行してください
  - repvect　品種指定名　DIFファイルパス指定オプション

（例）

```
repvect f100le -y"C:¥Program Files¥Renesas Electronics¥CubeSuite+¥Device¥RL78¥Devicefile"
```

品種指定名

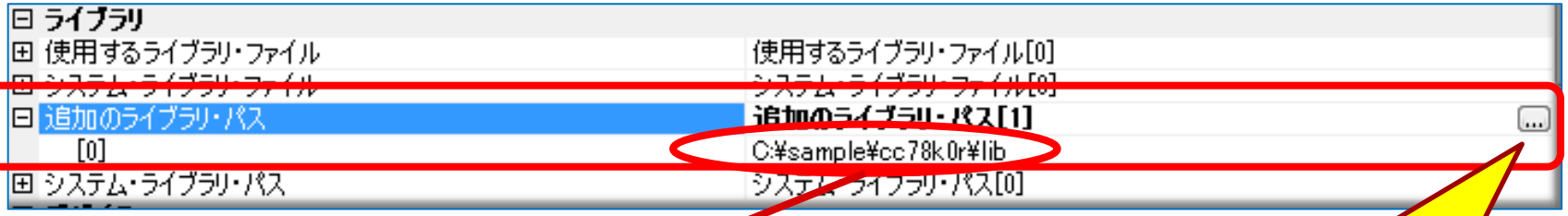
CubeSuite+の
インストールフォルダ

RL78
(78K0Rの場合は78K0R)

  - 「¥cc78k0r¥lib」にスタートアップ、ライブラリのファイルが生成される

分岐テーブルを2000H以外にする場合～プロジェクトへの登録(1/3)

- ライブラリパスの指定
  - ブート領域、フラッシュ領域のプロジェクトに指定
  - [リンクオプション]タブの「ライブラリ」
    - 「追加のライブラリ・パス」に新たに作成したライブラリのパスを指定



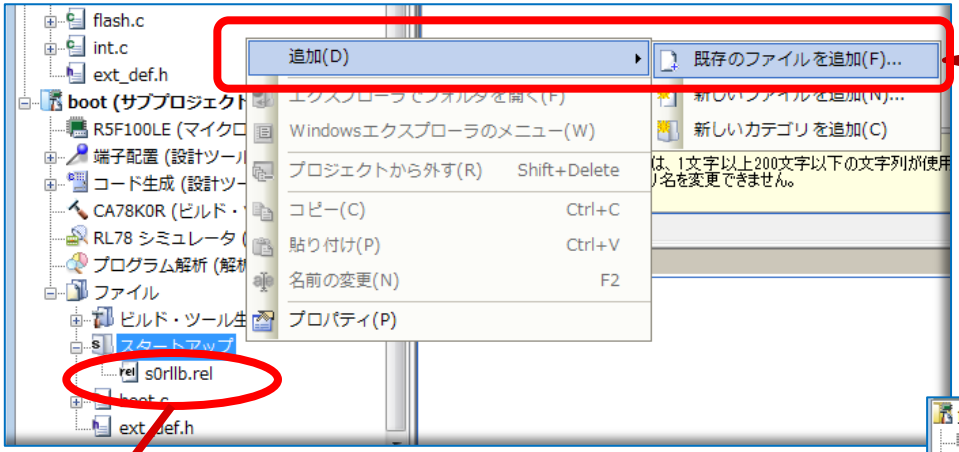
登録完了するとファイルが表示される

ボタンを押すことにより
パスを入力するダイアログが開く

分岐テーブルを2000H以外にする場合～プロジェクトへの登録(2/3)

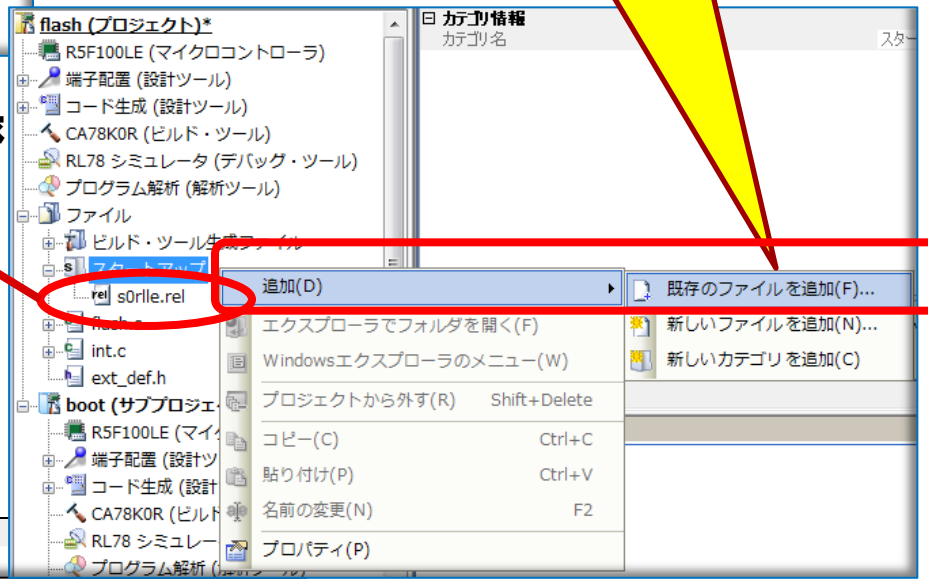
■ スタートアップの登録

● ブート領域プロジェクトへの登録



プロジェクトツリーパネルの「スタートアップ」を選択し右クリックして「既存のファイルを追加」を選択しファイルを指定する

● フラッシュ領域プロジェクトへの登録



登録完了するとファイルが表示される

登録完了すると、[コンパイルオプション]タブの「スタートアップ」の「標準のスタートアップを使用する」は、「いいえ」になる

分岐テーブルを2000H以外にする場合～プロジェクトへの登録(3/3)

- **スタートアップ・ファイルの命名規則**
  - lib78k0r¥s0r<*model*><*lib*><*flash*>.rel
    - <*model*>
      - m　：**ミディアム・モデル（スモール・モデル兼用）**
      - l　：**ラージ・モデル**
    - <*lib*>
      - **なし：標準ライブラリ固定領域を使用しない場合**
      - l　：**標準ライブラリ固定領域を使用する場合**
    - <*flash*>
      - **なし：通常用**
      - b　：**ブート領域用**
      - e　：**フラッシュ領域用**

変数／関数情報ファイル生成ツールを使用する場合

- [変数／関数配置オプション]タブの「出力ファイル」
  - 「変数／関数情報ファイルを出力する」で「はい」を設定

- 空の変数／関数情報ファイルが生成され、プロジェクトに追加されます

- 注意事項
  - 「VF78K0R error」が出力された場合は、一旦、[変数／関数配置オプション]タブの「変数／関数情報ファイルを出力する」で[いいえ]を選択してください。
  - また、プロジェクト・ツリーに登録されている変数／関数情報ファイルをプロジェクト・ツリーから外してください。

ブートーフラッシュ領域とミラー元領域によるnear/farの扱い

ブート領域にミラー元領域がない場合のROMデータの扱い

領域	定義	extern宣言	ポインタの指す先
ブート	far固定	far固定	far固定
フラッシュ	near/far指定可	far固定	far固定

フラッシュ領域にミラー元領域がない場合のROMデータの扱い

領域	定義	extern宣言	ポインタの指す先
ブート	near/far指定可	near/far指定可	far固定
フラッシュ	far固定	far固定	far固定

フラッシュ領域の先頭が64Kバイト以内にない場合の関数の扱い

領域	定義	extern宣言	ポインタの指す先
ブート	near/far指定可	near/far指定可※	far固定
フラッシュ	far固定	far固定	far固定

※ #pragma ext_funcで指定した関数は，関数本体がフラッシュ領域にあるのでfar固定となります。

再リンク機能とROM化機能を併用について（1/3）

- 再リンク機能とROM化機能を併用する場合、ブート領域用ロードモジュールファイルとフラッシュ領域用ロードモジュールファイルを結合することはできません。次の方法でご使用ください。
  - フラッシュ領域のプロジェクトでのブート領域用ロードモジュールファイルは、リンカ出力（ROM化前）のlmfファイルを指定する。
  - HEXファイルは、次のものを使用する。
    - ブート領域
      - ブート用域プロジェクトで出力したHEXファイル
    - フラッシュ領域
      - フラッシュ領域プロジェクトで出力したフラッシュ部分のHEXファイル（hxf）
  - フラッシュ領域プロジェクトでのオブジェクトコンバータのCRC機能は、フラッシュ領域プロジェクトのアドレス範囲で行う

再リンク機能とROM化機能を併用について(2/3)

- **ビルドツールでの設定方法**
  1. **フラッシュ領域用のプロジェクトにて、CubeSuite+のプロジェクトツリーパネルで、ビルドツールを右クリックして、プロパティを選択する。**
  2. **プロパティパネルで、リンクオプションタブを選択する。**
  3. **デバイスのブート領域用ロードモジュールファイル名には、ROM化前のブート領域用のlmfを指定する。**
     ※ROM化プロセッサ出力のlmfを指定しないでください。

再リンク機能とROM化機能を併用について(3/3)

- ブート領域のプロジェクトのhexファイルとフラッシュ領域のプロジェクトのフラッシュ領域部分のhexファイルをダウンロードする方法
  1. フラッシュ領域用のプロジェクトにて、CubeSuite+のプロジェクトツリーパネルで、デバッグツールを右クリックして、プロパティを選択する。
  2. プロパティパネルで、ダウンロードファイル設定タブを選択する。
  3. ダウンロードするファイルを選択し、右欄の[...]ボタンを選択する。
  4. ダウンロード一覧にフラッシュ領域プロジェクトのlmfファイルが表示されているので選択する
  5. 右側のダウンロードファイル情報の「オブジェクトをダウンロードする」を「いいえ」にする
  6. 左下の「追加」ボタンを選択する。
  7. ダウンロード一覧の「-」を選択し、右側の「ファイルの種類」で「ヘキサファイル」を選択する
  8. ファイルを選択し、右欄の[...]ボタンを選択し、ブート領域プロジェクトのhexファイルを指定する。
  9. 上記の(6)(7)と同じ手順で、フラッシュ領域プロジェクトのフラッシュ領域部分のhexファイル(hxf)を選択する。
  10. OKボタンを選択する。

ルネサス ソリューションズ株式会社