R20UT3384JJ0100
Rev.1.00
2015.03.13

CS+ Ver.3.01.00 RH850 マルチコア環境用チュートリアル（デバッグ編）

（株）ルネサス システムデザイン
ツールビジネス事業部
ツール技術部

はじめに

CS+をご使用いただき、誠にありがとうございます。
本チュートリアルでは、統合開発環境 CS+のご紹介と、使い方を E1(オンチップデバッギングエミュレータ)と RH850/C1X評価基板（株式会社サニー技研社製）を用いた例で説明します。プログラムの作成からデバッグまでを本チュートリアルの手順通りに操作していただくことにより、誰でも気軽に CS+を体験していただくことが可能です。
実際に CS+を用いたマイコンシステム開発を体験してみましょう。

CS+の特徴

CS+とは、コーディング、ビルド、デバッグまでのマイコン開発環境を一つのツールで実現した新統合開発環境です。

GUI のカスタマイズが簡単

CS+の各パネルを自由自在に操る「ドッキング」「フローティング」「自動で隠す」などの機能で、画面をお好きなようにカスタマイズすることが可能です。また、従来のプロジェクト環境を保存する機能に加え、開発環境を含めた保存も可能になりました。マイコンシステム開発をよりスムーズに行っていただけます。

開発環境の準備が簡単

システム開発を行うための開発環境が統合されており、必要なツールのインストールが簡単にできます。また、オートアップデート機能がついていますので、ワンクリックで最新の情報（ドキュメントを含む）に更新することも簡単にできます。

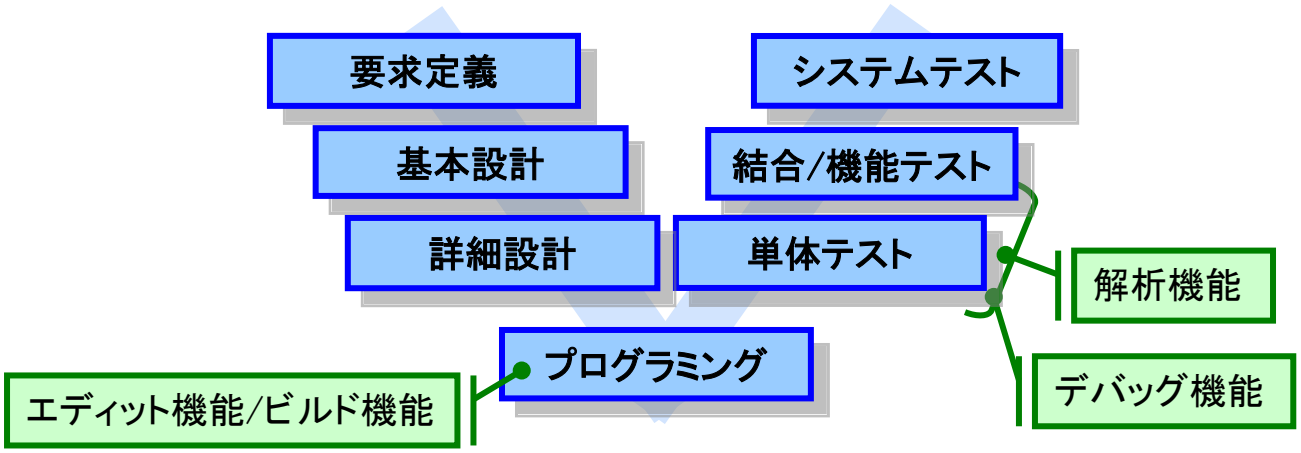
本チュートリアルと下記を合わせてお読みいただくことで、RH850 マルチコアのプログラミングを習得できます。

RH850 マルチコア向けプログラミング概要編

マイコンシステム開発の流れ

CS+を使ったシステム開発の流れを説明します。

一般的なシステム開発フロー（V 字モデル）



各システム開発フローに応じた、CS+の機能を説明します。

＜CS+機能＞	＜内容＞
エディット機能/ビルド機能	プログラムの編集を行う機能です。プログラムの作成が終了したらビルドをします。
デバッグ機能	ビルドしたプログラムをダウンロードして実行し、期待動作になっているかデバッグを行う機能です。
解析機能	解析結果を視覚的に確認し、プログラムの性能や品質の向上を支援するための機能です。

サンプルプログラムの概要

サンプルプログラムと、ターゲットボード(RH850/C1X評価基板)の概要を説明します。

1. サンプルプログラムの概要

今回使用するプログラムは、RH850/C1H の各々のコア（CPU1 と CPU2）で異なる LED を制御（点灯/消灯）します。

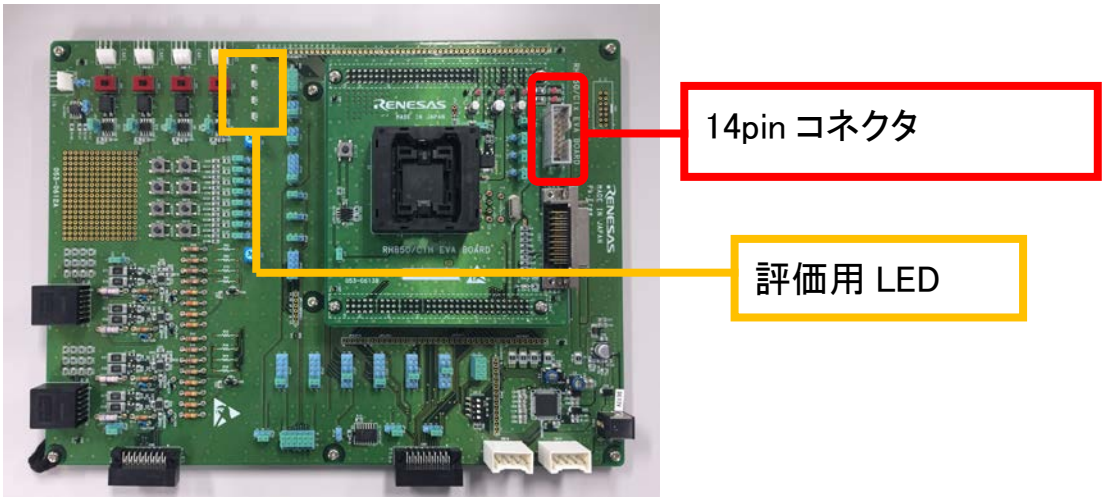
プログラムの詳しい説明は付録の「サンプルプログラムの説明」を参照してください。

CPU1コア：LED1を制御し、LED1を点滅させます
CPU2コア：LED2 を制御し、LED2 を点滅させます

2. ターゲットボード(RH850/C1X評価基板)の概要

ターゲットボードとして用いる RH850/C1X評価基板の概要は以下のとおりです。

RH850/C1X評価基板



評価用 LED1～4　：ポートグループ 4 の P4_n(n=8-11)が High で点灯します。
14pinコネクタ　：オンチップデバッグや書き込み時に使用します。

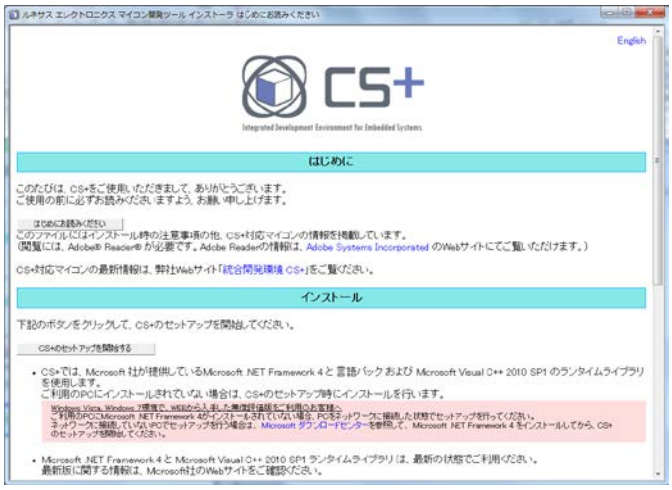
インストール

CS+をインストールする手順を説明します。

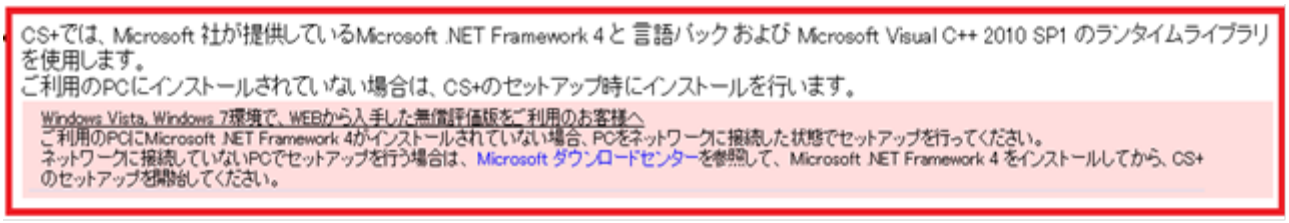
1. Microsoft 社製ソフトウエアの事前インストール

CS+をインストールするには、「.NET Framework」と「Visual C++ のランタイムライブラリ」の事前インストールが必要です。ご使用の PC にインストールされていない場合には、CS+のセットアップ時にインストールを行ないます。

CS+製品の DVD を PC のドライブに挿入してください。
以下のような画面が自動的に立ち上がります。



必要なソフトウエアをインストールしてください。



インストール

2. 統合インストーラの実行

統合インストーラを実行することにより，CS+製品をインストールします。

[CS+のセットアップを開始する]をクリックして、CS+のセットアップを開始してください。

インストールウィザードに従って、設定を行ってください。最後に[完了]ボタンをクリックしてインストールを終了します。
※インストール後に PC を再起動してください。

CS+の起動

CS+ の起動からプロジェクトの作成までを行います。

1. CS+の起動

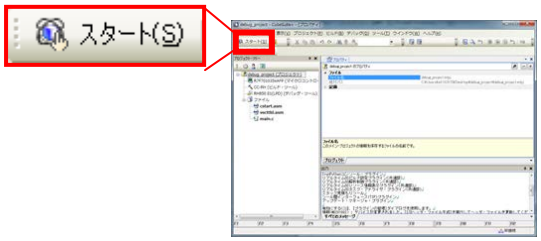
[スタート] → [すべてのプログラム] → [Renesas Electronics CS+] → [CS+] を選択して CS+を起動します。

起動時に「ワンポイントアドバイス」ダイアログが起ち上がります。参照したい方は[次へ]ボタンをクリックして参照してください。[OK]ボタンをクリックすると CS+の起動画面が表示されます。

ワンポイントアドバイス

スタートパネルについて

新たな開発でCS+を使い始めるときは、”スタートパネル”ボタン（下図）をクリックしてください。スタートパネルが表示され、新しいプロジェクトを作成したり、最近使ったプロジェクトや、お気に入りのプロジェクトを開いたりなど、簡単にプロジェクトを作成/開くことが可能です。（はじめてCS+をインストールして、起動した場合には、スタートパネルが表示されますが、一度プロジェクトを作成した後は、起動後に最新のプロジェクトが開きます。）

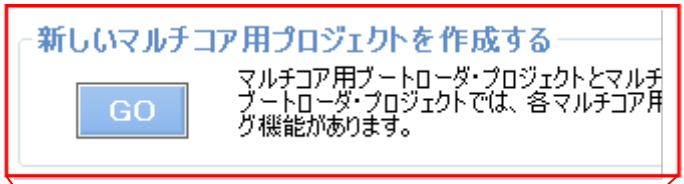


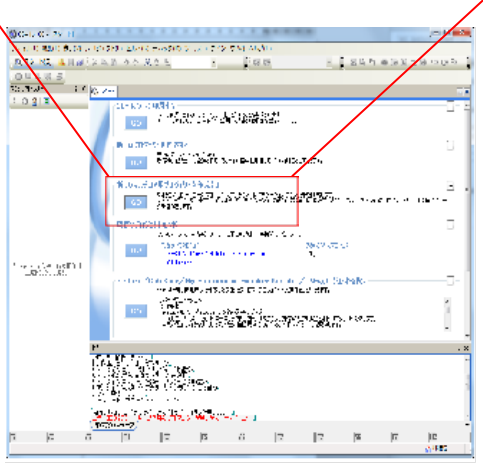
CS+の起動

2. プロジェクトの読み込み

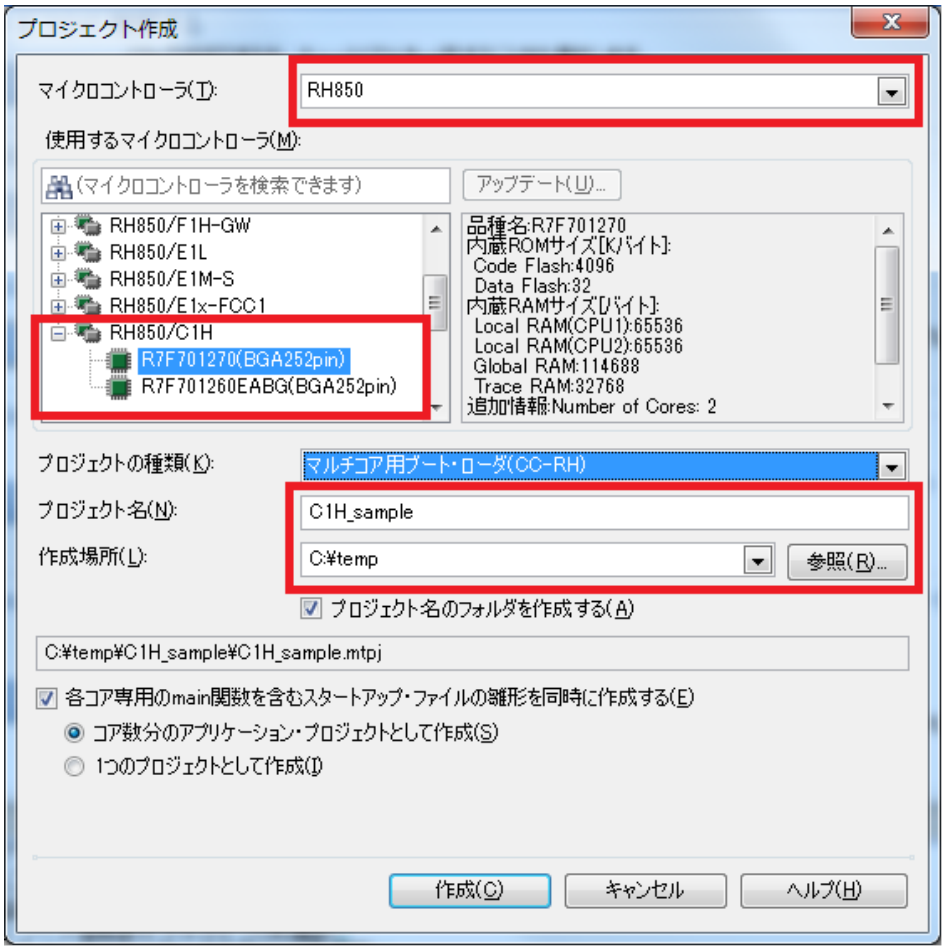
プロジェクトの読み込みを行ないます。
本資料は、CS+で新規に作成したプロジェクトを使用して説明します。
詳細については、CS+を用いたプロジェクトの構築方法を参照ください。
CS+ Ver.3.01.00 RH850 マルチコア環境用チュートリアル（ビルド編 2）

「新しいマルチコア用プロジェクトを作成する」欄の[GO]ボタンをクリックして、プロジェクトを作成してください。





「マイクロコントローラに」に[RH850]、「使用するマイクロコントローラ」に[RH850/C1H]、[R7F701270]を選択し、「プロジェクト名」と「作成場所」を設定してください。本書では、プロジェクト名を「C1H_sample」に設定します。



ワンポイントアドバイス

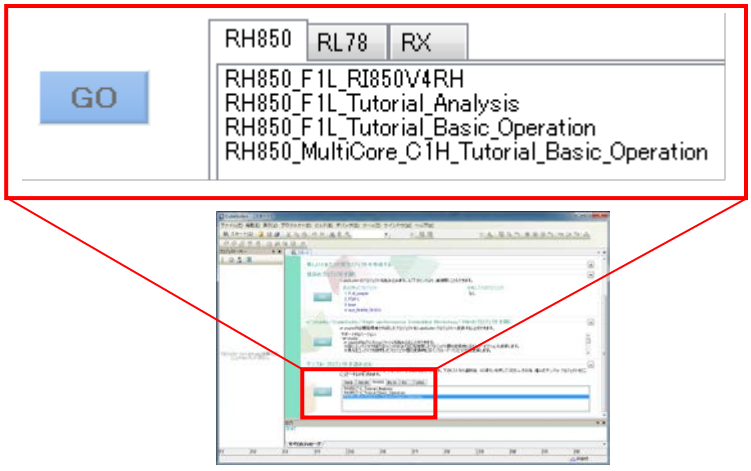
サンプル・プロジェクトについて

CS+は、サンプル・プロジェクトを提供しています。

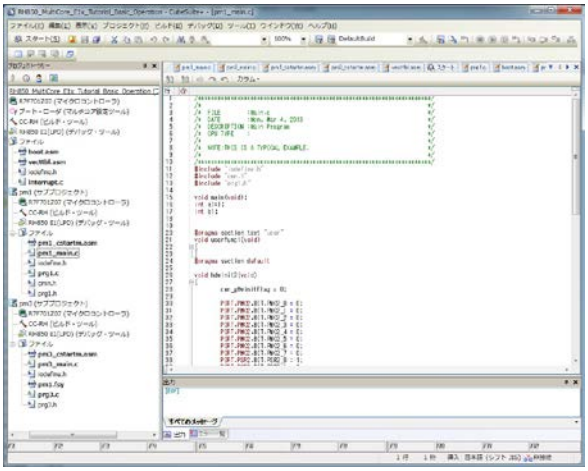
サンプル・プロジェクトは、本資料の“プログラムの編集”操作後の状態になっています。

CS+が提供するサンプル・プロジェクトを使用する場合は、下記のようにサンプル・プロジェクトの読み込みを行なってください。

「サンプル・プロジェクトを読み込む」欄の[RH850]タブから「RH850_Multicore_C1H_Tutorial_Basic_Operation」を選択し、[GO]ボタンをクリックしてください。



指示に従っていくと，下図のようにサンプル・プロジェクトが開きます。

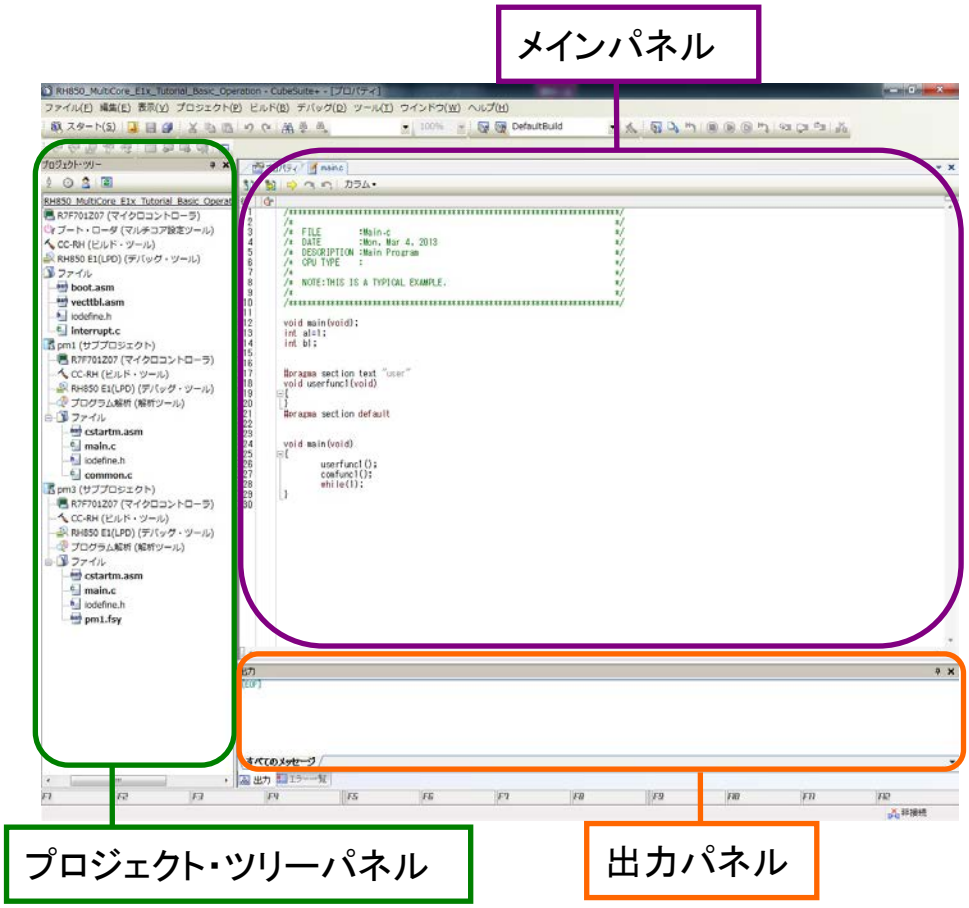


ウインドウ操作

CS+では、ウインドウを自由自在にカスタマイズすることが出来ます。ここではウインドウ構成と「自動で隠す」、「フローティング」、「ドッキング」などのウインドウカスタマイズに関して説明します。

1. ウインドウ構成

CS+のウインドウ構成を説明します。



プロジェクト・ツリーパネル
　　：システム開発のフロー順に、CS+の機能を表示します。
メインパネル
　　：プロジェクト・ツリーから選択した機能に対応するパネルを表示します。（エディタパネル　など）
出力パネル
　　：出力結果を表示します。

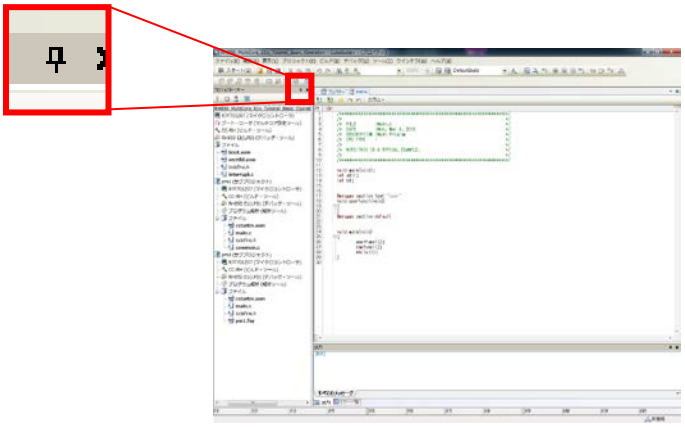
ウインドウ操作

2. 自動で隠す

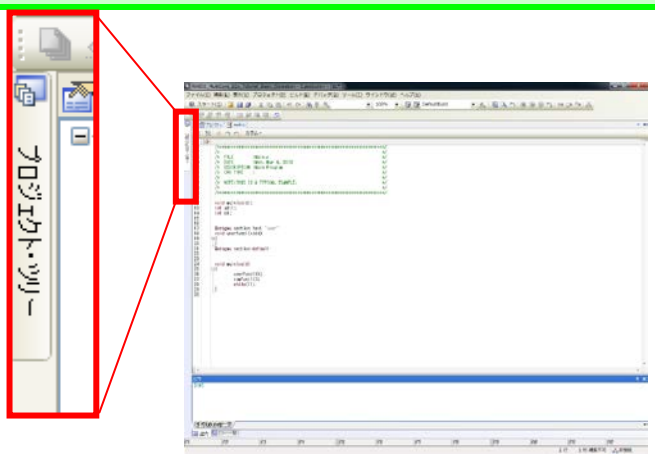
各パネルのタイトルバーのピンアイコンをクリックすることにより、パネルを自動的に隠す、隠さないといった設定を簡単に切り替えることが可能です。操作上、不要なパネルを自動的に隠すことにより、画面を有効に使うことが出来ます。

(a)プロジェクト・ツリーパネルを自動的に隠す場合

プロジェクト・ツリーのピンアイコンをクリックしてください。

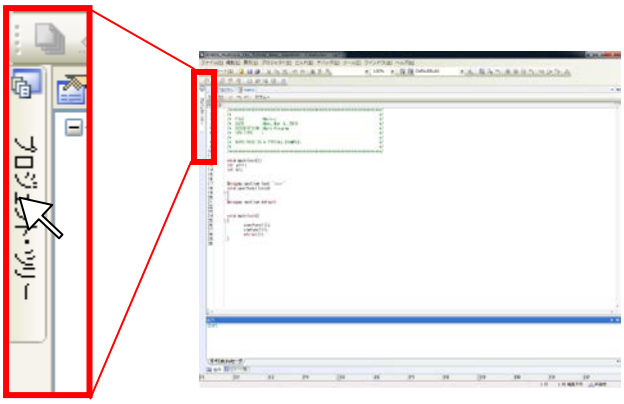


プロジェクト・ツリーが自動的に隠れ、タブができます。

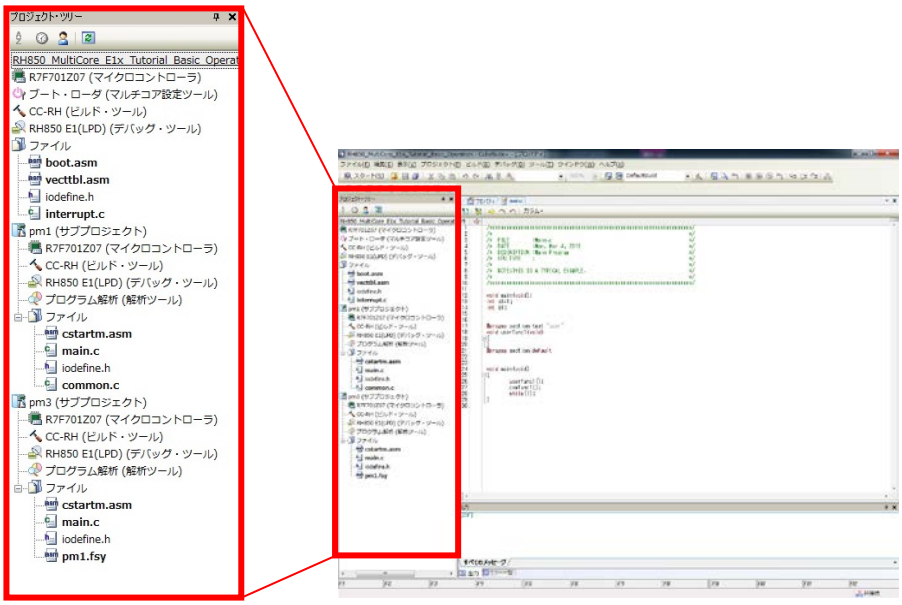


(b)隠したプロジェクト・ツリーパネルを表示する場合

[プロジェクト・ツリー]タブにポインタを合わせてください。



プロジェクト・ツリーがスライド表示されます。



ワンポイントアドバイス

パネルの隠し場所について

　パネルはウインドウの左側、右側、下側の三箇所に隠すことが可能です。また、同じ箇所に複数のパネルを隠すこともできます。

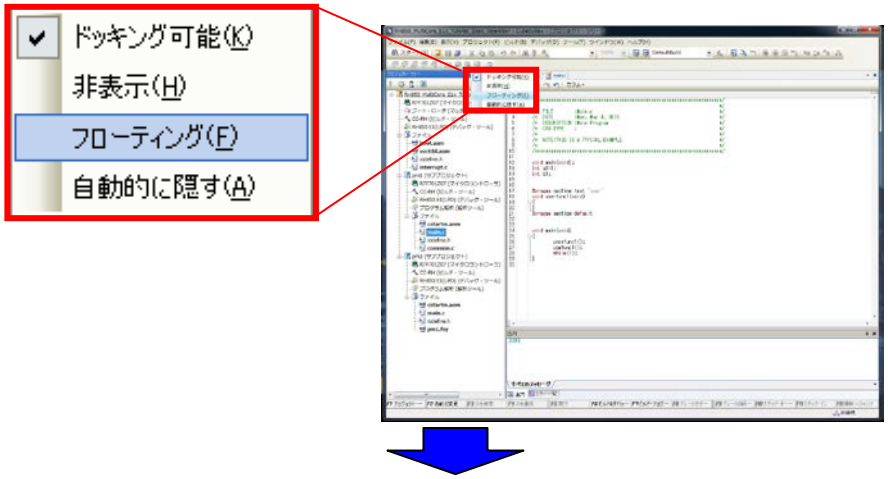
ウインドウ操作

3. フローティング

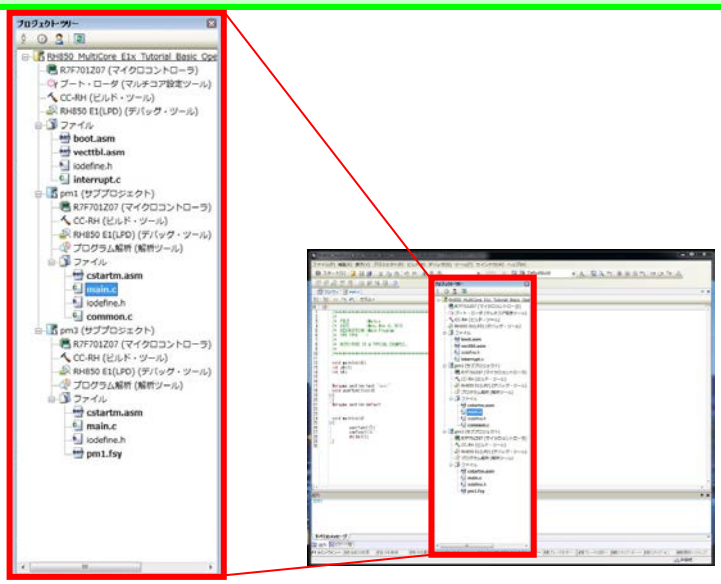
タイトルバーで右クリックして[フローティング]を選択すると、パネルを自由に移動させることができます。

(a)プロジェクト・ツリーパネルをフローティング状態にする場合

タイトルバー上で右クリックして[フローティング]を選択します。



パネルがフローティング状態になります。



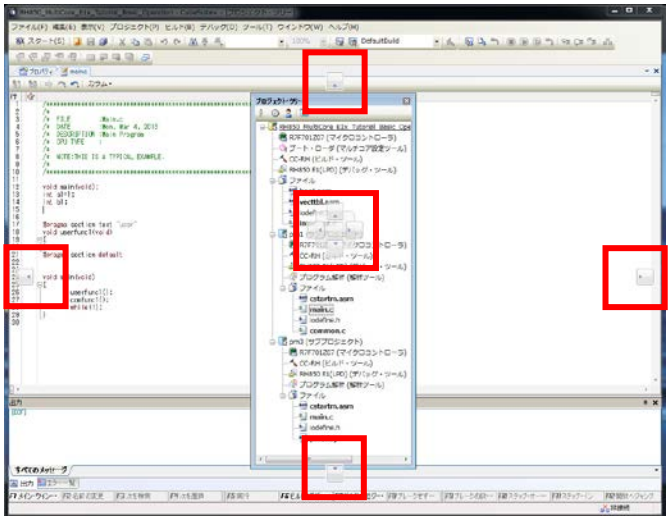
ウインドウ操作

4. ドッキング

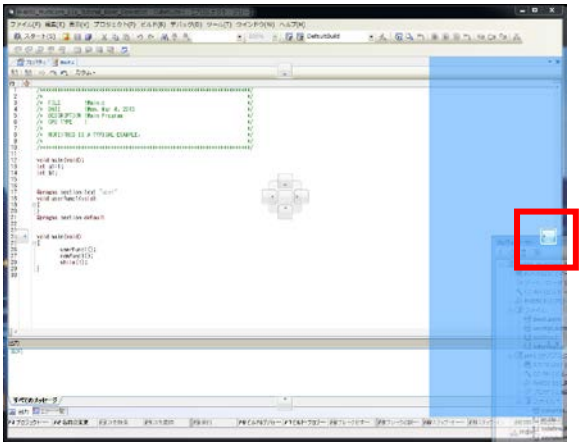
フローティング状態から、パネルをメインパネルや各パネルの上下左右に配置することが出来ます。パネルをナビゲーター表示に従い、好きな場所にドラック&ドロップすることで簡単にパネルの配置を変更することが可能です。

(a)プロジェクト・ツリーパネルの配置を変更する場合

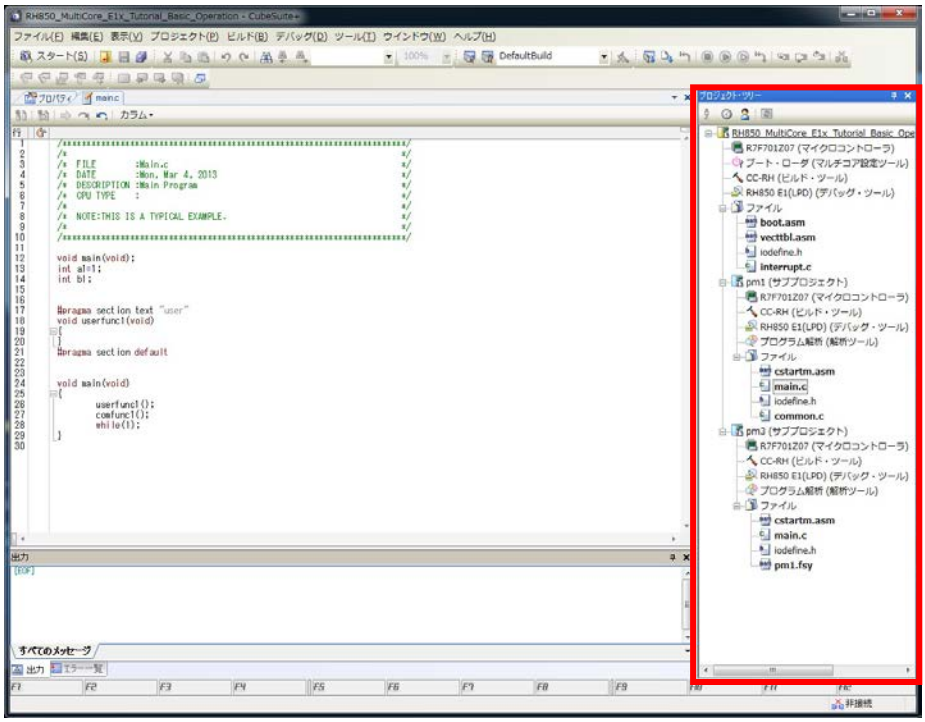
フローティング状態のパネルをドラックするとナビゲーターが表示されます。



移動したい位置のナビゲーターにポインタを合わせると配置される領域が青く表示されます。



そのままドロップすることで、プロジェクト・ツリーの配置が変更されます。（下図はメインパネルの右に配置した場合）

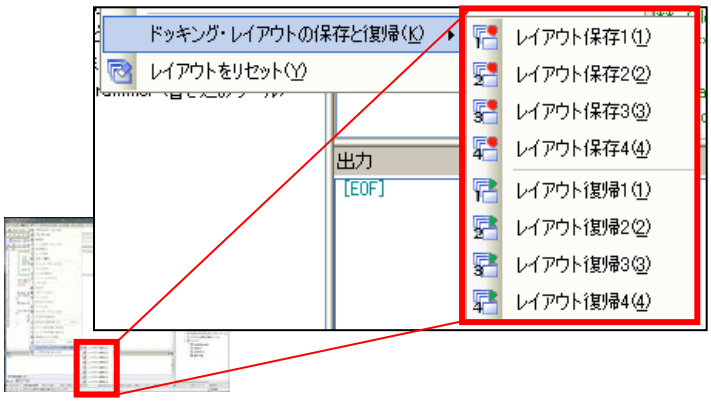


ワンポイントアドバイス

レイアウトの保存、復帰について

レイアウト（パネルの配置情報）は、メニューバーの「表示」→「ドッキングレイアウトの保存と復帰」で、デバッグ・ツール接続前/接続後のそれぞれに対して、4 つの状態を保存することができます。

※デバッグ・ツール接続時のみ、デバッグ専用のレイアウトになります。



プログラムの編集

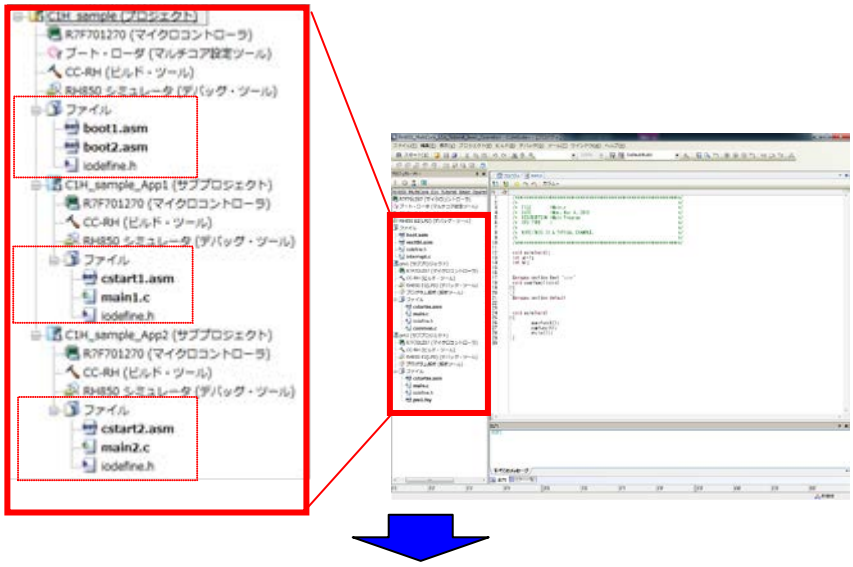
ユーザのプログラム編集を行います。

最初に基本的な編集方法を説明しますが，今回はコピー＆ペーストで簡単に行っていただきます。手順に従い，プログラムのエディットを行ってください。

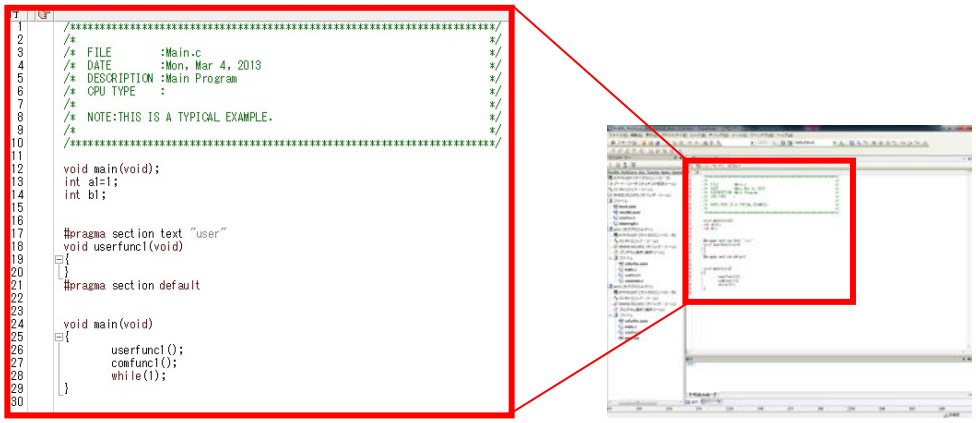
1. ソースの開き方

ソースの開き方を説明します。

プロジェクト・ツリーの「ファイル」中にソースがあるので，エディットしたいソースをダブルクリックしてください。



ソースがメインパネルに表示されます。



プログラムの編集

2. エディット

手順に従い，プログラムをエディットしてください。

以下の記述をコピーして main1.c の main()関数内にペーストしてください。

```
void main(void)
{
                    hwinit();
                    g_flag = 1;
                    set();
                    g_counter = 0;

                    while(1){
                                        g_counter++;
                                        if(g_counter == 0x1FFFFF){
                                                            g_counter = 0;
                                                            outputLED1();
                                        }
                    }
}
```

main1.c にインクルード文を追加します。以下の記述をコピーしてファイルの先頭にペーストしてください。

```
#include "iodefine.h"
#include "common.h"
```

以下の記述をコピーして main2.c の main()関数内にペーストしてください。

```
void main(void)
{
                hwinit();
                wait();
                g_counter_pe2 = 0;

                while(1){
                                g_counter_pe2++;
                                if(g_counter_pe2 == 0x1FFFFF){
                                                g_counter_pe2 = 0;
                                                outputLED2();
                                }
                }
}
```

main2.c にインクルード文を追加します。以下の記述をコピーしてファイルの先頭にペーストしてください。

```
#include "iodefine.h"
#include "common.h"
```

boot1.asm の下記のコメント行の先頭のセミコロンを削除して有効にしてください。

```
.L.entry_PE1:
                        jarl                _hdwinit_PE1, lp      ; initialize hardware
```

```
.L.entry_PE2:
                        jarl                _hdwinit_PE2, lp      ; initialize hardware
$ifdef USE_TABLE_REFERENCE_METHOD
                        jarl                _set_table_reference_method, lp ; set table reference method
$endif
                        jr32                __cstart_pm2
```

```
;-----------------------------------------------------------------------------
;                       hdwinit_PE1
; Specify RAM addresses suitable to your system if needed.
;-----------------------------------------------------------------------------
                        .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]
```

以下の記述をコピーして boot1.asm の hdwinit_PE1 の直前にペーストしてください。

```
;-----------------------------------------------------------------------------
;                       RAM address
;-----------------------------------------------------------------------------
                        GLOBAL_RAM_ADDR         .set                    0xfeef0000
                        GLOBAL_RAM_END          .set                    0xfef0bfff

                        LOCAL_RAM_PE1_ADDR      .set                    0xfebf0000
                        LOCAL_RAM_PE1_END       .set                    0xfebfffff

                        LOCAL_RAM_PE2_ADDR      .set                    0xfe9f0000
                        LOCAL_RAM_PE2_END       .set                    0xfe9fffff
```

以下の記述をコピーして boot1.asm の zeroclr4 の直前にペーストしてください。

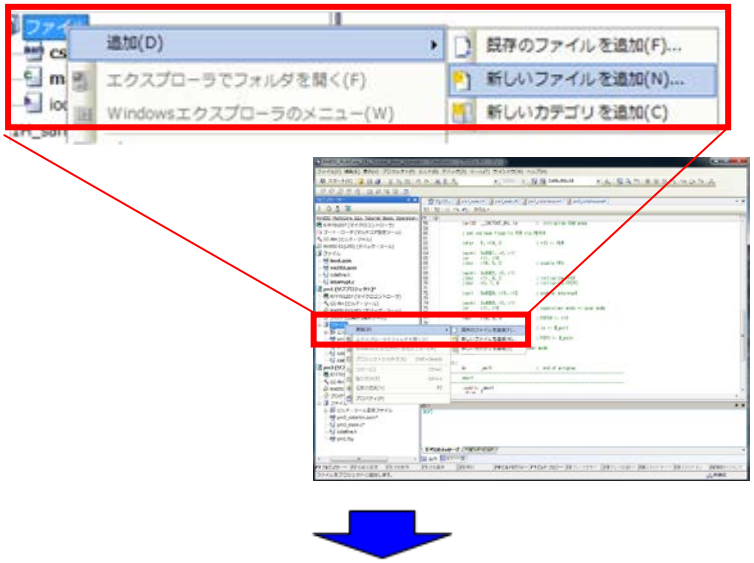
```
;-----------------------------------------------------------------------------
;                       hdwinit_PE2
; Specify RAM addresses suitable to your system if needed.
;-----------------------------------------------------------------------------
                        .section                ".text", text
                        .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]
```

プログラムの編集

3. ファイルの追加

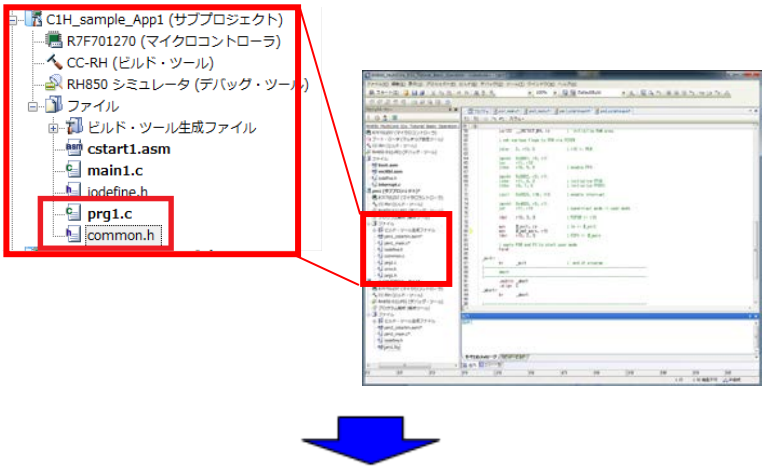
手順に従い，プログラムを追加してください。

プロジェクト・ツリーの C1H_sample_App1（サブプロジェクト）のファイルを選択し、右クリックで開くポップアップメニューから“新しいファイルを追加”を選択してください。



下記の 2 つのファイルを追加してください。なお、ファイルの作成場所は、C1H_sample¥C1H_sample_App1 フォルダに設定してください。

C1H_sample_App1
- prg1.c
- common.h



以下の記述をコピーして prg1.c にペーストしてください。

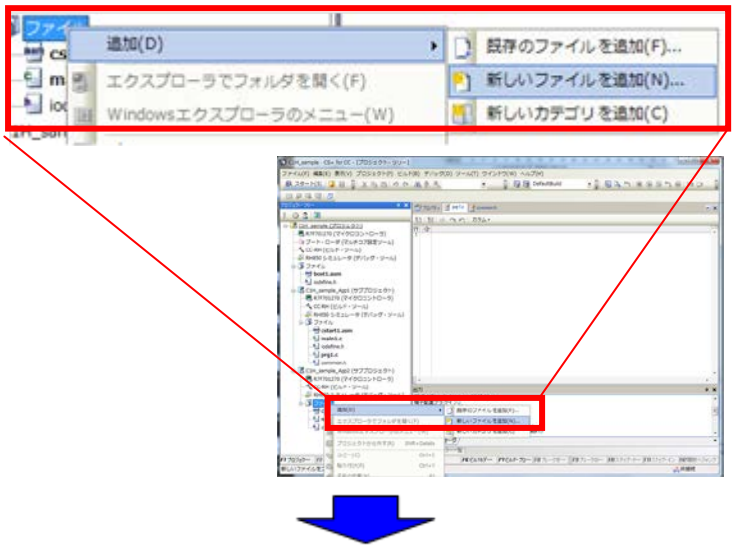
```
#include "iodefine.h"
#include "common.h"

#define LED1 PORT.P4.BIT.P4_11
#define LED2 PORT.P4.BIT.P4_10
#define LED3 PORT.P4.BIT.P4_9
#define LED4 PORT.P4.BIT.P4_8

/******************************************************************************/
/* Definition of unshared variable for PE1                                    */
/******************************************************************************/
int led1 = 0;
int led2 = 0;
int led3;
int led4;

/******************************************************************************/
/* Definition of shared variable                                              */
/******************************************************************************/
#pragma section r0_disp32 "com"
int g_flag = 0;
int g_counter;
int g_counter_pe2;
#pragma section default

/******************************************************************************/
/* Definition of unshared function for PE1                                    */
/******************************************************************************/
void hwinit()
{
                        PBG.FSGD3DPROT5.UINT32 = 0x000e02d5;
                        PORT.PM4.UINT16 = 0;

                        led3 = 0;
                        led4 = 0;

                        LED1 = led1;
                        LED2 = led2;
```

```
                        LED3 = led3;
                        LED4 = led4;
}

void set()
{
                        MEV.GOMEV0 = 0;
}

void outputLED1()
{
                        led1 = g_eor1(led1);
                        LED1 = led1;
}

void outputLED2()
{
                        led2 = g_eor1(led2);
                        LED2 = led2;
}

void outputLED3()
{
                        led3 = g_eor1(led3);
                        LED3 = led3;
}

void outputLED4()
{
                        led4 = g_eor1(led4);
                        LED4 = led4;
}

/***********************************************************************/
/* Definition of shared function                                       */
/***********************************************************************/
#pragma section text "com"
int g_eor1(int num)
{
                        return num^1;
}
#pragma section default
```

以下の記述をコピーして common.h にペーストしてください。

```
/***********************************************************************/
/* Declaration of shared variable                                      */
/***********************************************************************/
#pragma section r0_disp32 "com"
extern int g_flag;
extern int g_counter;
extern int g_counter_pe2;
#pragma section default

/***********************************************************************/
/* Declaration of shared function                                      */
/***********************************************************************/
#pragma section text "com"
int g_eor(int num);
#pragma section default
```

プロジェクト・ツリーの C1H_sample_App2（サブプロジェクト）のファイルを選択し、右クリックで開くポップアップメニューから“新しいファイルを追加”を選択してください。



下記のファイルを追加してください。なお、ファイルの作成場所は、C1H_sample¥C1H_sample_App2 フォルダに設定してください。

C1H_sample_App2
　－prg2.c

以下の記述をコピーして prg2.c にペーストしてください。

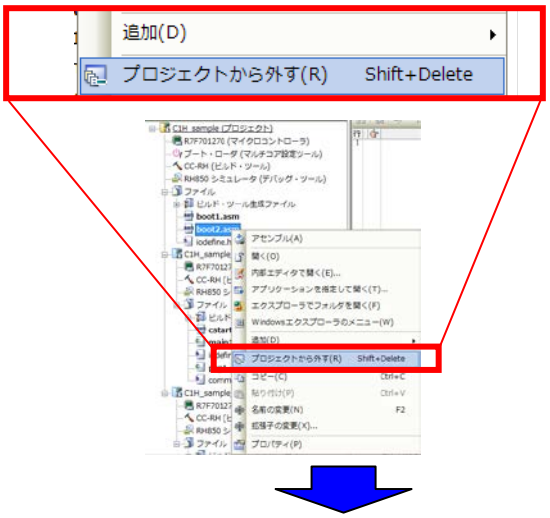
```
#include "iodefine.h"
#include "common.h"

#define LED1 PORT.P4.BIT.P4_11
#define LED2 PORT.P4.BIT.P4_10
#define LED3 PORT.P4.BIT.P4_9
#define LED4 PORT.P4.BIT.P4_8

/*****************************************************************************/
/* Definition of unshared variable for PE2                                   */
/*****************************************************************************/
int led1;
int led2;
int led3 = 0;
int led4 = 0;

/*****************************************************************************/
/* Definition of unshared function for PE2                                   */
/*****************************************************************************/
void hwinit()
{
                        led1 = 0;
                        led2 = 0;
                        MEV.GOMEV0 = 1;
}

void wait()
{
                        while(MEV.GOMEV0);
}

void outputLED1()
{
                        led1 = g_eor1(led1);
                        LED1 = led1;
}

void outputLED2()
{
                        led2 = g_eor1(led2);
                        LED2 = led2;
}

void outputLED3()
{
                        led3 = g_eor1(led3);
                        LED3 = led3;
}

void outputLED4()
{
                        led4 = g_eor1(led4);
                        LED4 = led4;
}
```

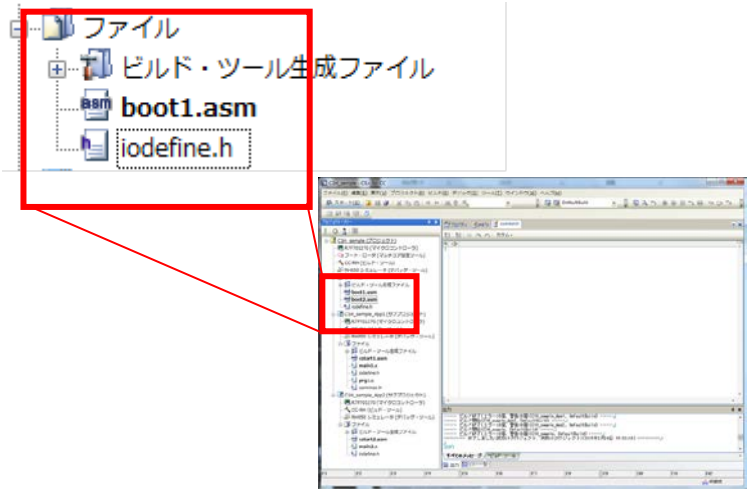
プログラムの編集

4. ファイルの削除

手順に従い，プログラムを削除してください。

プロジェクト・ツリーの C1H_sample（プロジェクト）の boot2.asm を選択し、右クリックで開くポップアップメニューから“プロジェクトから外す”を選択してください。



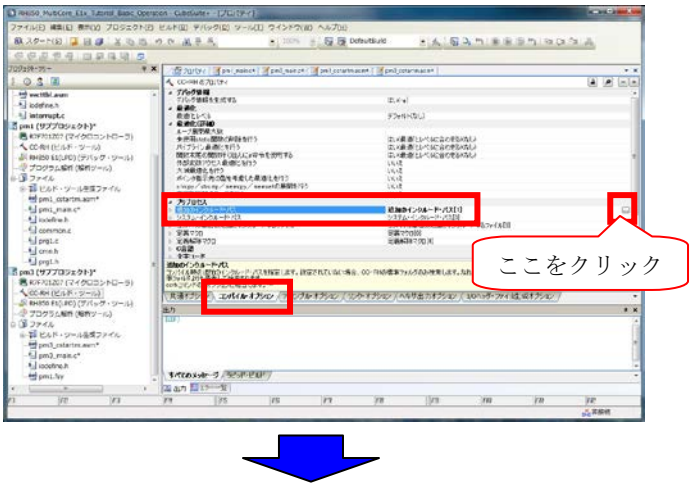


プログラムの編集

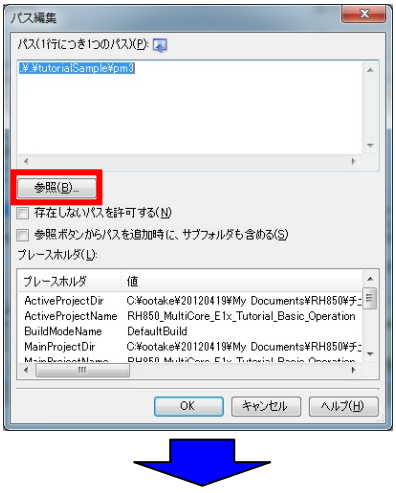
5. コンパイラ（CC-RH）のプロパティの変更

手順に従い，CC-RH のプロパティを変更してください。

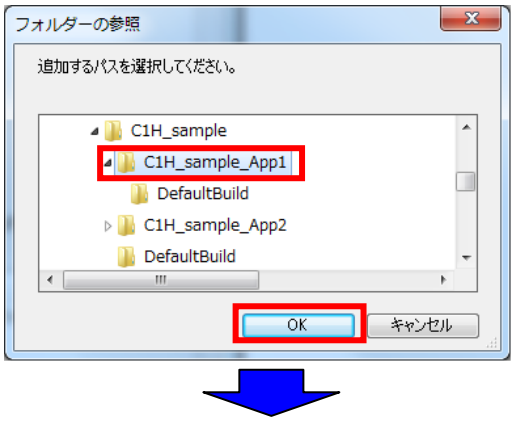
プロジェクト・ツリーの C1H_sample_App2（サブプロジェクト）の CC-RH のプロパティを表示し、コンパイル・オプションの追加インクルード・パスの追加で追加ボタンを押します。



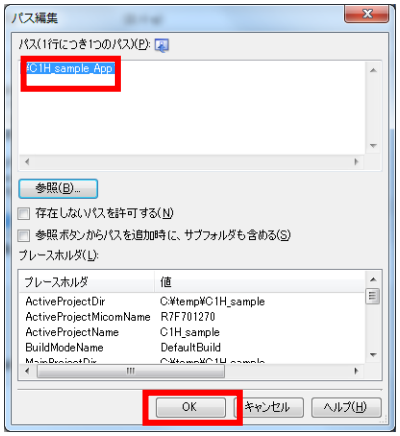
パス編集ダイアログが表示されますので、参照ボタンを押します。



フォルダの参照ダイアログが表示されますので、下記フォルダを選択します。
C1H_sample¥ C1H_sample _App1



フォルダが追加されたことを確認して OK ボタンを押してください。

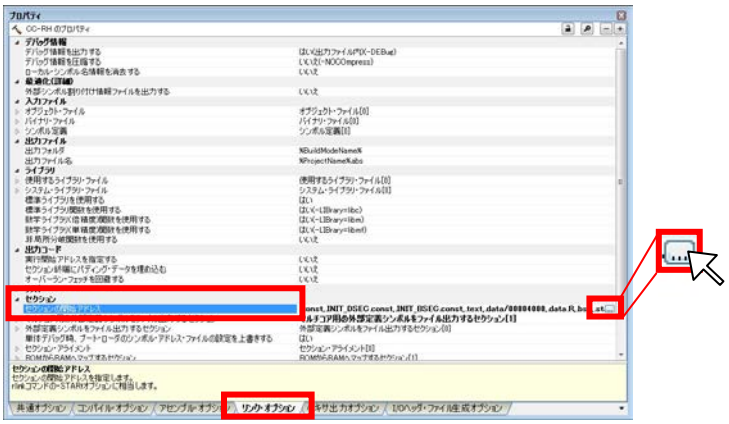


プログラムの編集

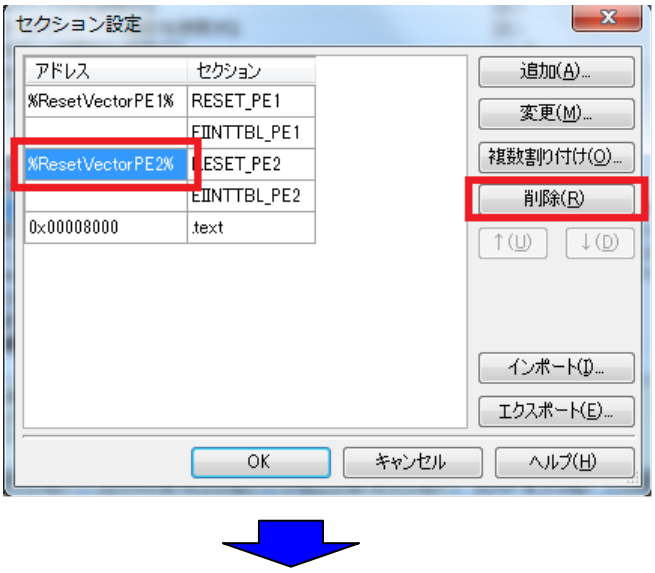
6. セクションの削除

手順に従い，セクションを削除してください。

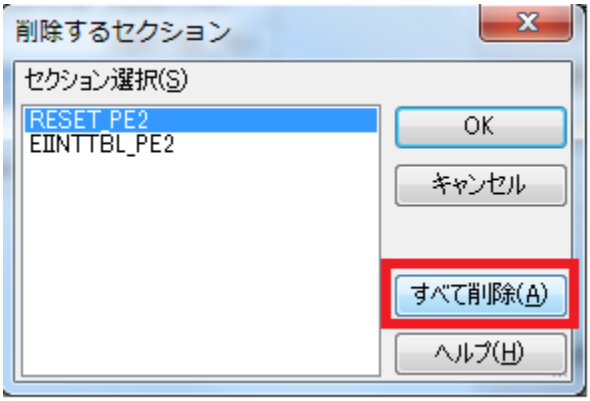
プロジェクト・ツリーの C1H_sample（プロジェクト）の CC-RH のプロパティを表示し、リンク・オプションのセクショングループのセクションの開始アドレスで編集ボタンを押します。



セクション設定ダイアログが表示されますので、%ResetVectorPE2%を選択し、削除ボタンを押してください。



削除するセクションダイアログが表示されますので、すべて削除ボタンを押してください。

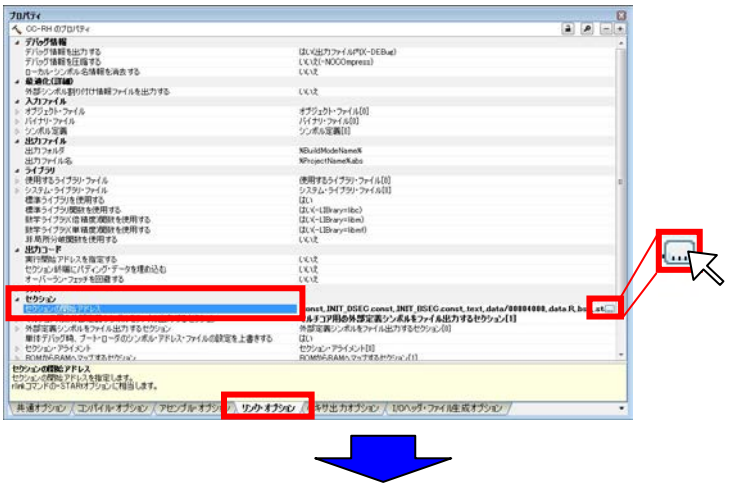


プログラムの編集

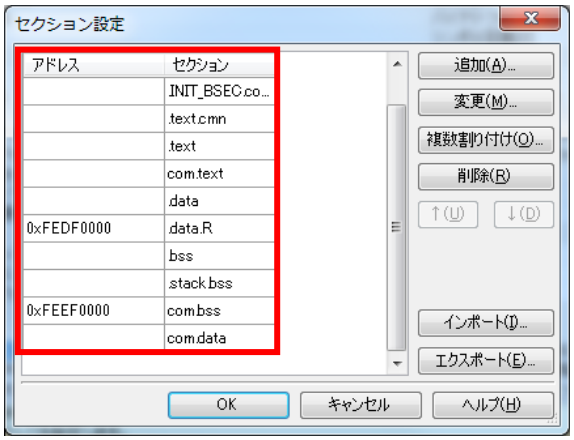
7. セクション開始アドレスの編集

手順に従い，セクション開始アドレスを変更してください。

プロジェクト・ツリーの C1H_sample_App1（サブプロジェクト）の CC-RH のプロパティを表示し、リンク・オプションのセクショングループのセクションの開始アドレスで編集ボタンを押します。



セクション設定ダイアログが表示されますので、追加ボタンを押して、“com.text”，“com.bss”，“com.data”の 3 つのセクションを追加し、下記のように設定・編集をしてください。

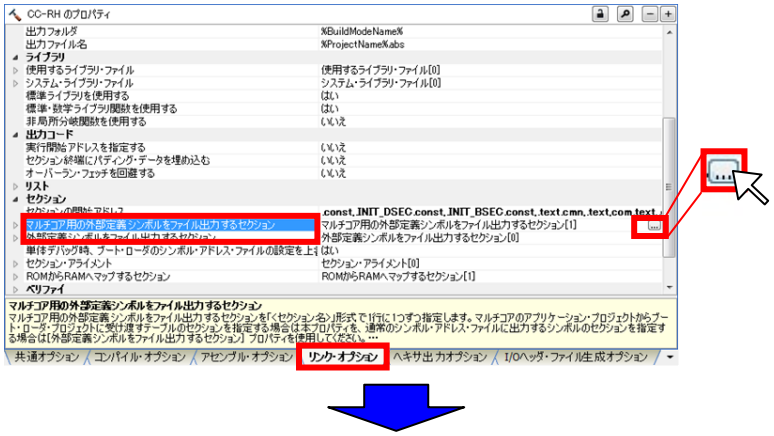


プログラムの編集

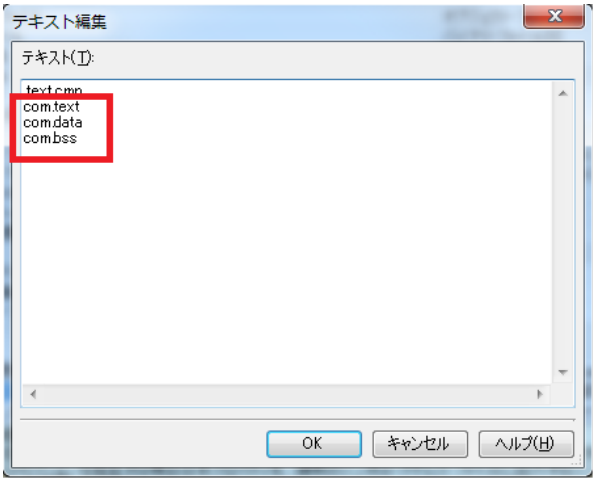
8. マルチコア用の外部定義シンボルをファイル出力するオプションの編集

手順に従い，マルチコア用の外部定義シンボルを出力するオプションを変更してください。

プロジェクト・ツリーの C1H_sample_App1（サブプロジェクト）の CC-RH のプロパティを表示し、“リンク・オプション”のセクショングループの“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”で編集ボタンを押します。



セクション設定ダイアログが表示されますので、追加ボタンを押して、“com.text”，”com.bss”，”com.data”の 3 つのセクションを追加し、下記のように設定・編集をしてください。

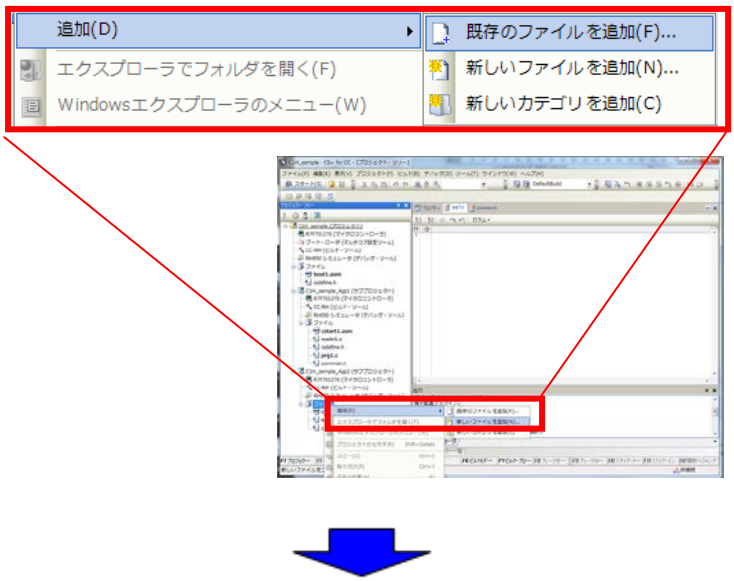


プログラムの編集

9. ファイルの追加

手順に従い，C1H_sample_app2 に C1H_sample_app 1.fsy ファイルを追加してください。

プロジェクト・ツリーの C1H_sample_App2（サブプロジェクト）のファイルを選択し、右クリックで開くポップアップメニューから“新しいファイルを追加”を選択してください。



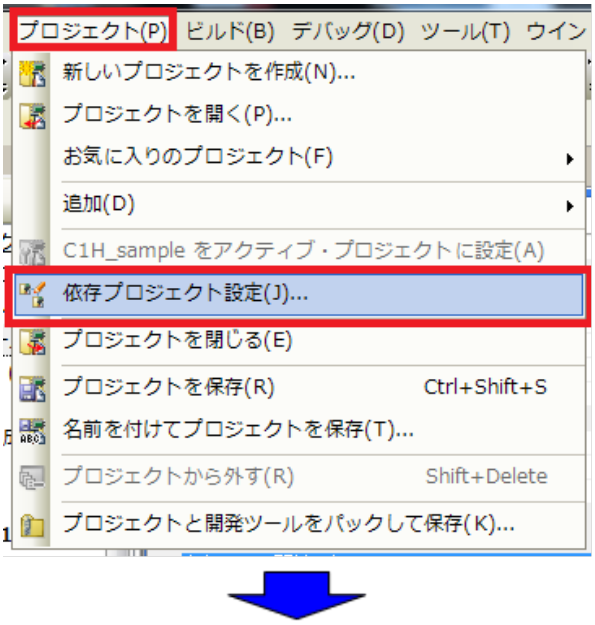
アセンブリ・ソース・ファイルを選択し、C1H_sample_App1 、DefaultBuild と開き、“C1H_sample_App1.fsy”を選択してファイルを追加してください。

プログラムの編集

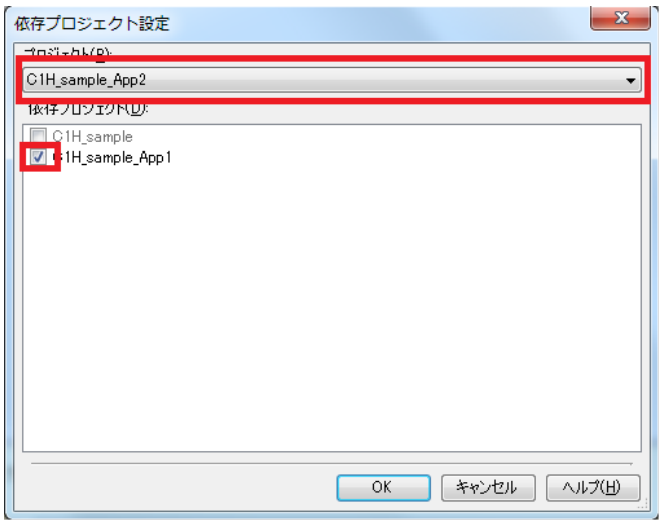
10. 依存プロジェクトの設定

手順に従い，依存プロジェクトを設定してください。

メニューのプロジェクトから“依存プロジェクト設定”を選択してください。



プロジェクトに“C1H_sample_App2”を選択し、”C1H_sample_App1”を選択してください。



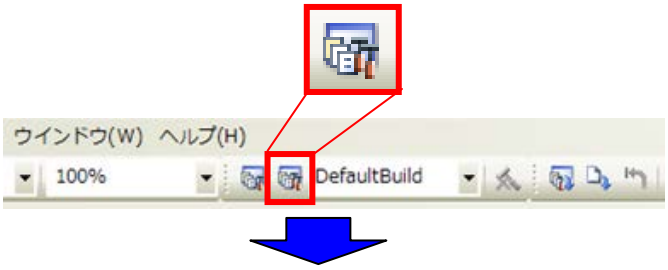
プログラムのリビルド

読み込んだサンプル・プロジェクトのプログラムをリビルドします。

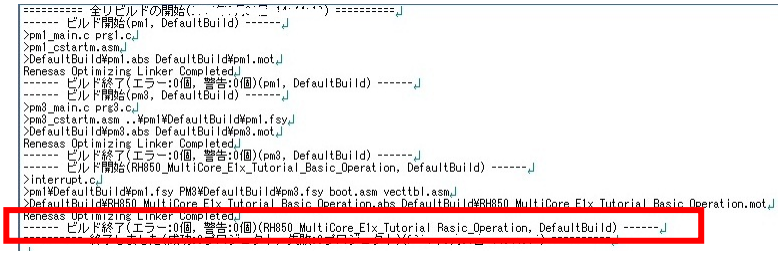
1. ビルドプロジェクト

読み込んだサンプル・プロジェクトのプログラムをリビルドします。

[リビルドプロジェクト]ボタンをクリックしてください。



リビルドが正常に完了したかを確認してください。正常にリビルドが完了した場合、ロードモジュールファイルが作成されます。



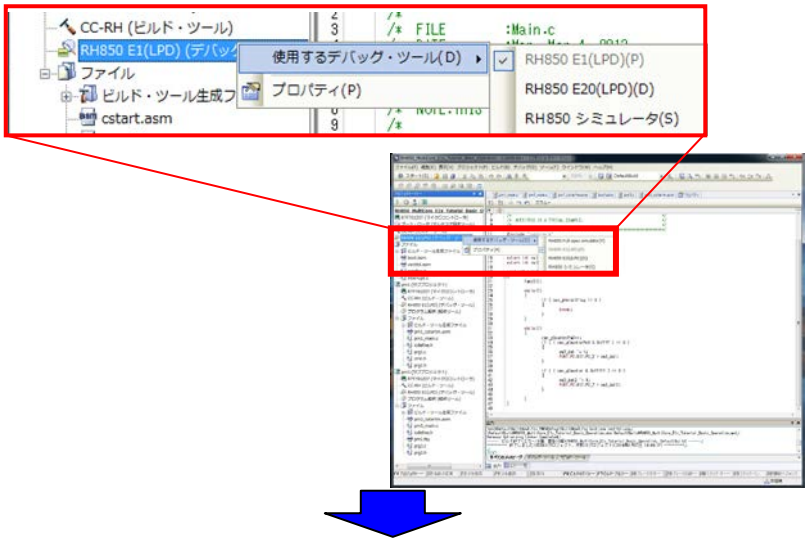
デバッガの接続とダウンロード

ここでは E1 を用いて、プログラムをデバッグします。まずはデバッグを開始するにあたって準備を行います。

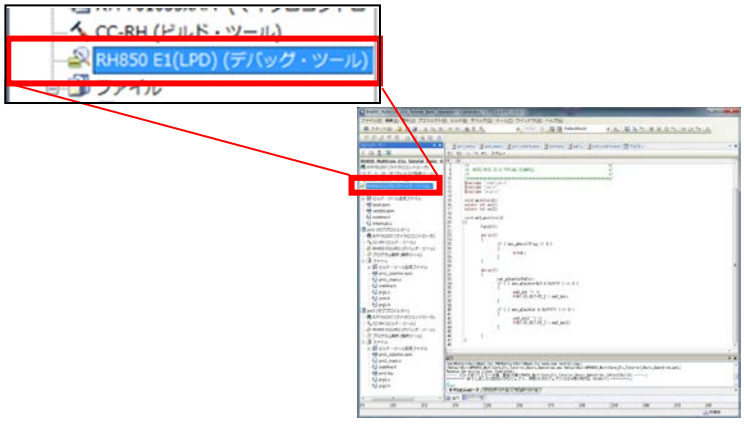
1. デバッグ・ツールの選択

使用するデバッグ・ツールとして[RH850 E1(LPD)（デバッグ・ツール）]を選択します。

プロジェクト・ツリーのデバッグ・ツール上で右クリックして、[使用するデバッグ・ツール]→[RH850 E1(LPD)]を選択してください。



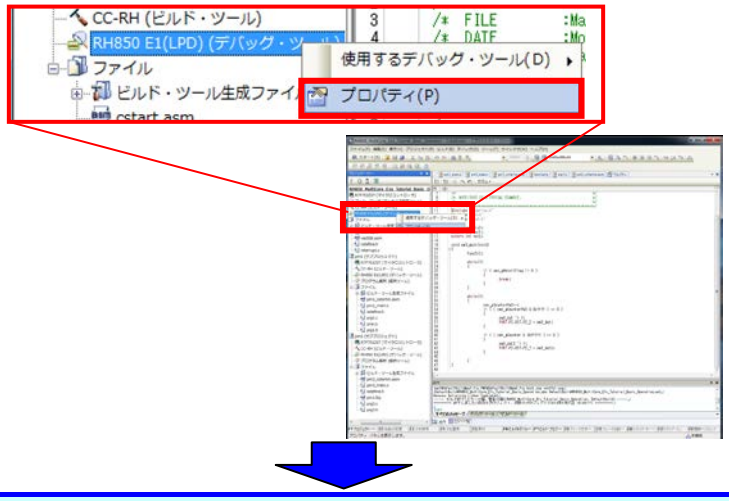
デバッグ・ツールとして[RH850 E1(LPD)（デバッグ・ツール）]が選択されます。



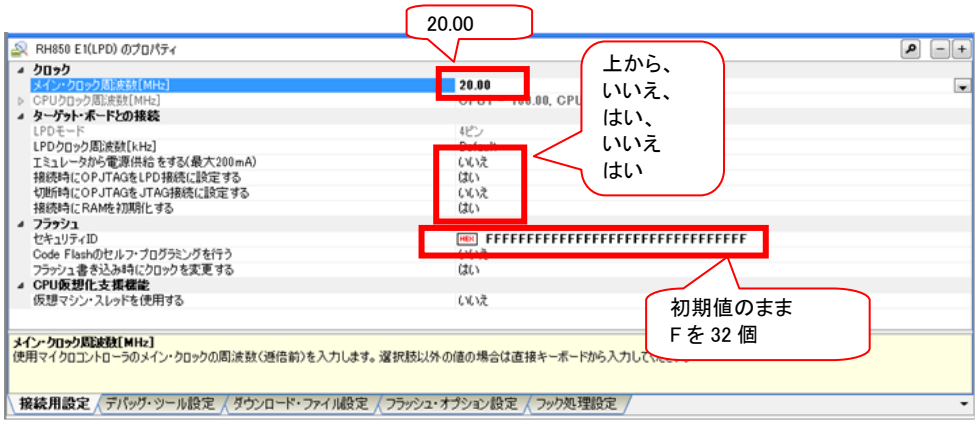
デバッガの接続とダウンロード

2. E1 とターゲットボードとの接続設定

プロジェクト・ツリーのデバッグ・ツール上で右クリックして、[プロパティ]を選択してください。



[接続用設定]タブを選択し、以下のように設定してください。

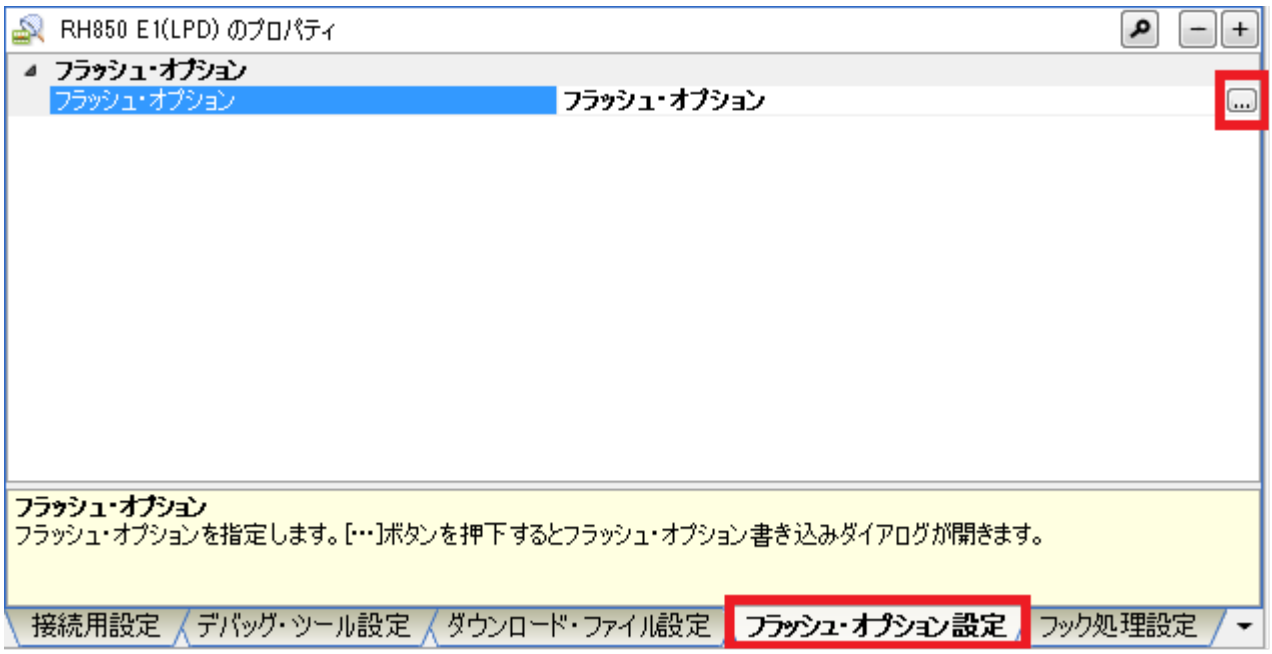


ワンポイントアドバイス

セキュリティ ID について

フラッシュメモリの内容が権限のないユーザにリードされないように、128 ビットのID コードをマイクロコントローラにライトすることができます。デバッガ起動時にユーザが入力するコードがマイクロコントローラにライトされたID コードに一致しない場合は、フラッシュメモリにアクセスできません。設定は、フラッシュプログラマで行ないます。ブランク（全面消去）品をご使用の場合は、セキュリティIDは、ALL Fを入力ください。

[フラッシュ・オプション]タブを選択し、編集ボタンを押します。



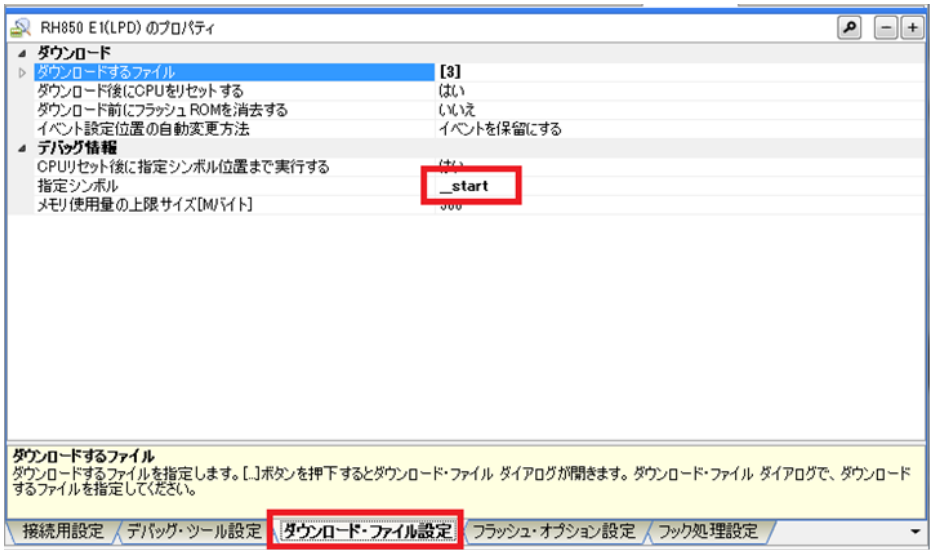
オプション・バイトを以下のように設定し、書き込みボタンを押してください。

ワンポイントアドバイス

オプション・バイトについて

フラッシュメモリにはさまざまな目的でユーザが指定したデータを保持する拡張領域（オプション・バイト）があります。RH850/C1Hでは、デバッグI/Fの設定だけでなく、WDT関連の機能設定やマイコンの動作モード・起動領域の設定等も行ないます。
本チュートリアルプログラム使用時は、OPBT0レジスタはH’7FFFFFFDに、OPBT2レジスタはH’BFFFFFFFに設定してください。

[ダウンロード・ファイル]タブを選択し、デバッグ情報グループのシテにシンボルに”_start”を設定してください。

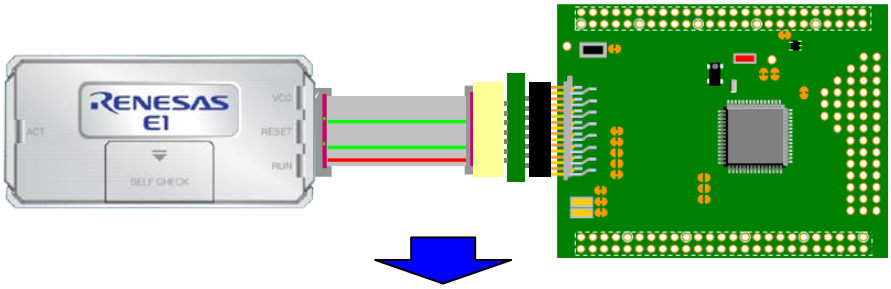


デバッガの接続とダウンロード

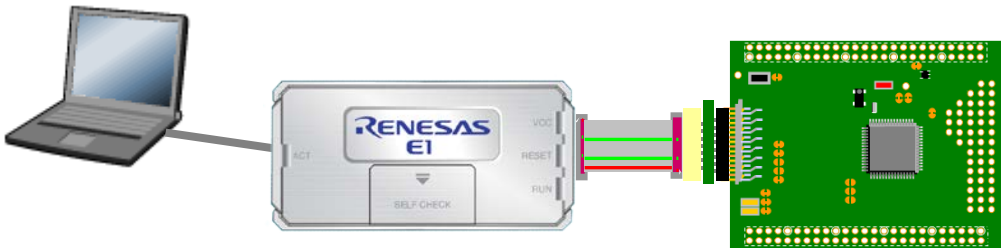
3. E1 の接続

E1 を用いてオンチップデバッグを行います。

RH850/C1X評価基板と E1 を接続してください。コネクタの 1pin と合わせて接続します。



E1 と PC を接続してください。（初めて接続する場合、では、「新しいハードウエアの検出」ウィザードが開きますので、“ソフトウエアを自動でインストールする”を選択し、指示に従ってUSBドライバのインストールを行ってください。）



RH850/C1X評価基板の電源を入れてください。

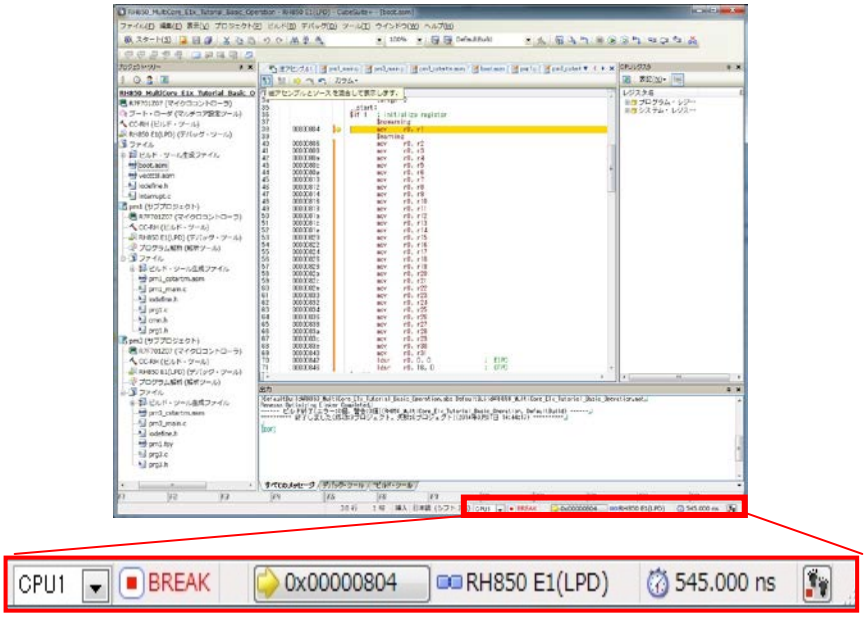
デバッガの接続とダウンロード

4. E1 へのロードモジュールファイルのダウンロード

ビルドにより生成したロードモジュールファイルをターゲットにダウンロードします。
ダウンロードが完了すると、プログラムを実行させることが可能になります。

メニューのダウンロードボタンをクリックしてください。

メイン・ウインドウのステータスバーが、下図のように変化します。



コアを切り替える

ターゲットへロードモジュールファイルのダウンロードが完了しましたので、デバッグ対象コアを切り替えてみましょう。

1. コアの切り替え

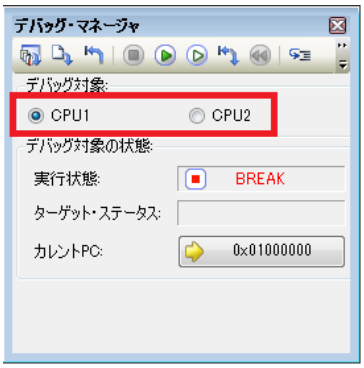
デバッグ対象コアを切り替えるには2つの方法があります。

a. ステータスバーで切り替える

メイン・ウインドウのステータスバー上のドロップダウンリストにより、切り替えることができます。

b. デバッグ・マネージャパネルで切り替える

[表示]メニュー→[デバッグ・マネージャ]を選択すると、デバッグ・マネージャパネルがオープンします。デバッグ・マネージャパネルで切り替えることが出来ます。



ここでは、CPU1 を選択した状態にしておいてください。

ワンポイントアドバイス

デバッグ対象コアについて

プログラムの実行や停止は、一方のコアのみ実行または停止することは出来ません。
プログラムの実行や停止は、必ず、両コアが協調して動作します。
デバッグ対象コアを CPU1 に設定している状態での CS+上のメモリの参照や変更は、CPU1 に対してのみ有効です。他方のコアに対し、操作を行なう場合は、デバッグ対象コアを切り替えて行なってください。

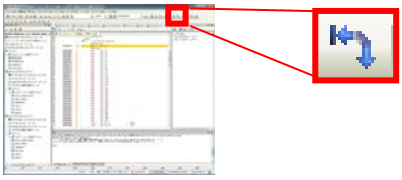
プログラムの実行と停止

ターゲットへロードモジュールファイルのダウンロードが完了しましたので、プログラムを実行することが可能です。まずは、プログラムの実行と停止を行います。

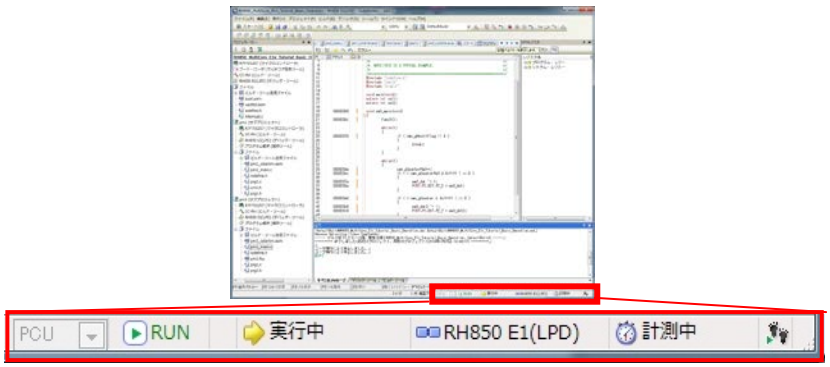
1. プログラムの実行

CPU をリセット後、プログラムを実行させます。

メニューの「リセット＆実行ボタン」をクリックしてください。



プログラムが実行され、ステータスバーに[RUN]と表示されます。



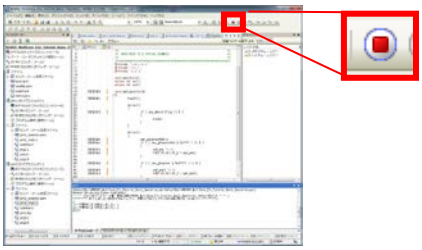
ボード上の LED1 と LED2 が点滅します。

プログラムの実行と停止

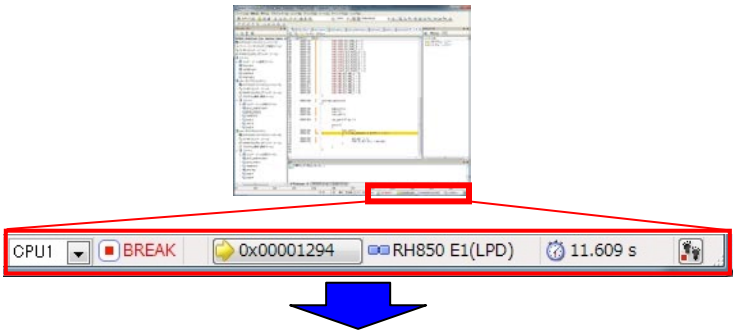
2. プログラムの停止

プログラムを停止させます。

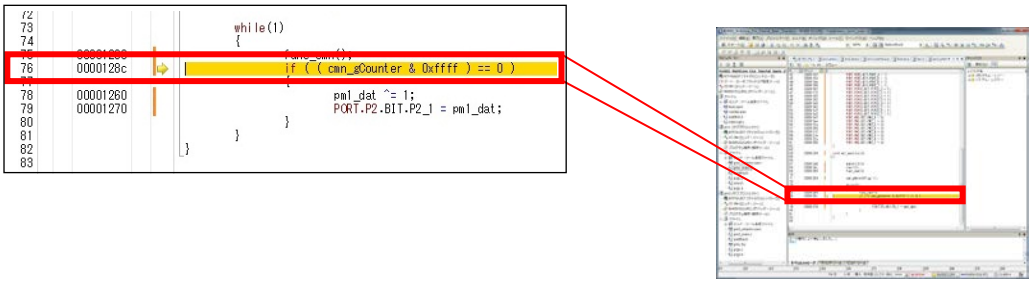
「停止ボタン」をクリックしてください。



プログラムの実行が停止され、ステータスバーに[BREAK]と表示されます。



プログラムの停止したソース行(プログラムカウンタ(PC)の現在位置)が黄色で表示されます。

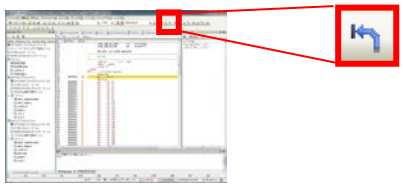


プログラムの実行と停止

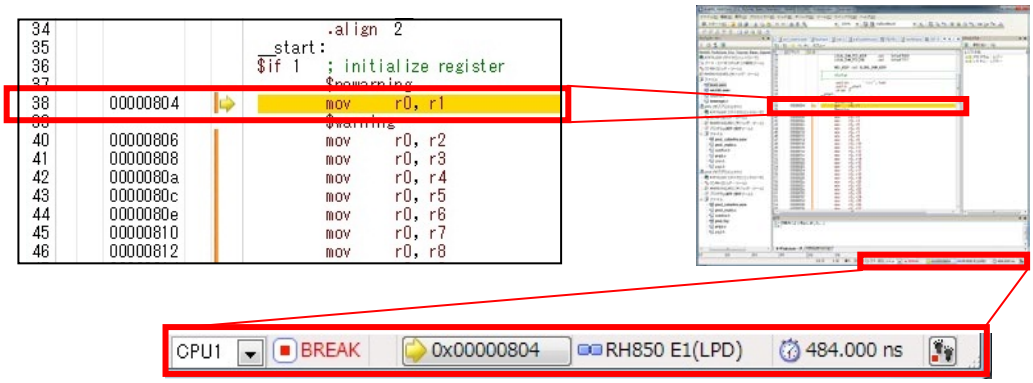
3. プログラムのリセット

1.ではプログラムのリセットと実行を 1 つのボタンで同時に行いましたが、プログラムのリセットのみを行うことも可能です。

「リセットボタン」をクリックしてください。



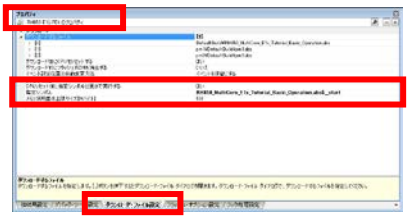
プログラムのリセットが行われ、プログラムカウンタ（PC）がプログラムの先頭に戻ります。



ワンポイントアドバイス

プログラムカウンタについて

プログラムカウンタ(PC)とは、次に実行するプログラムのアドレス情報を保持する制御レジスタです。RH850/C1H では、リセット信号の発生により、ユーザモードでは 00000000H が、ユーザブートモードでは 01000000H が PC にセットされます。本チュートリアルプログラムでは、リセット後に__start 関数まで実行する設定を行なっているため、__start 関数まで実行した後、ブレークした状態となります。このような動作を変更する場合は RH850 E1（LPD）のプロパティの[ダウンロード・ファイル設定]で行うことができます。

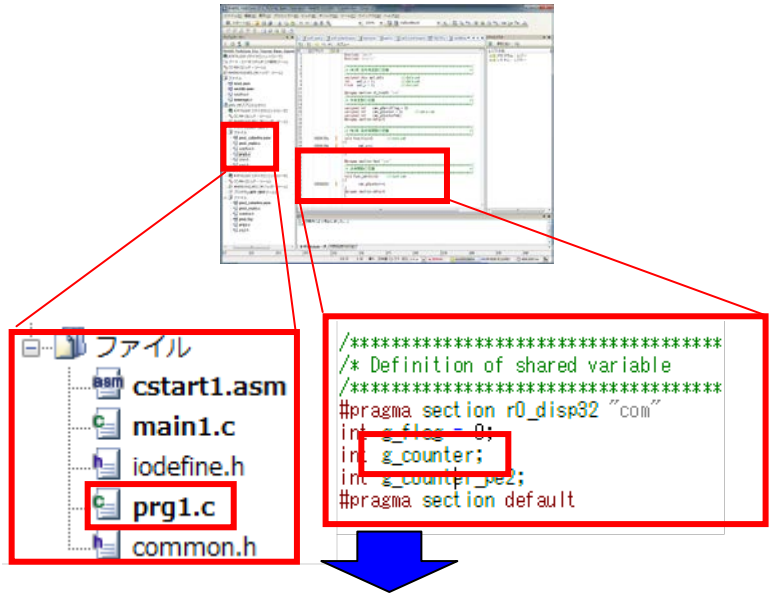


変数値の参照

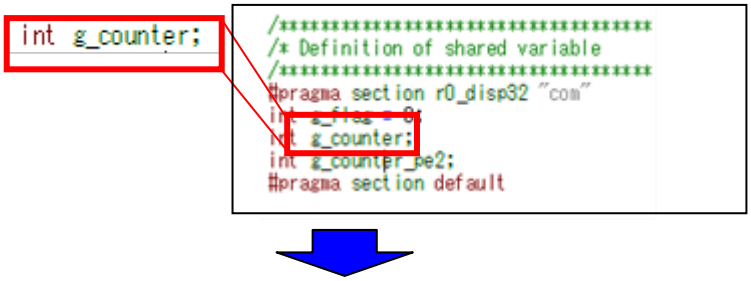
ウォッチ機能

変数を「ウォッチ登録」することにより、変数値を表示させることが可能です。ここでは「g_counter」と「g_counter_pe2」の2つの共有変数（CPU1 コアからも CPU2 コアからも参照できる共有領域に配置されている変数）をウォッチ登録し、変数値がインクリメントされていることを確認します。「g_counter」は CPU1 コアが、「g_counter_pe2」は CPU2 コアがそれぞれプログラム中でカウントアップする変数です。

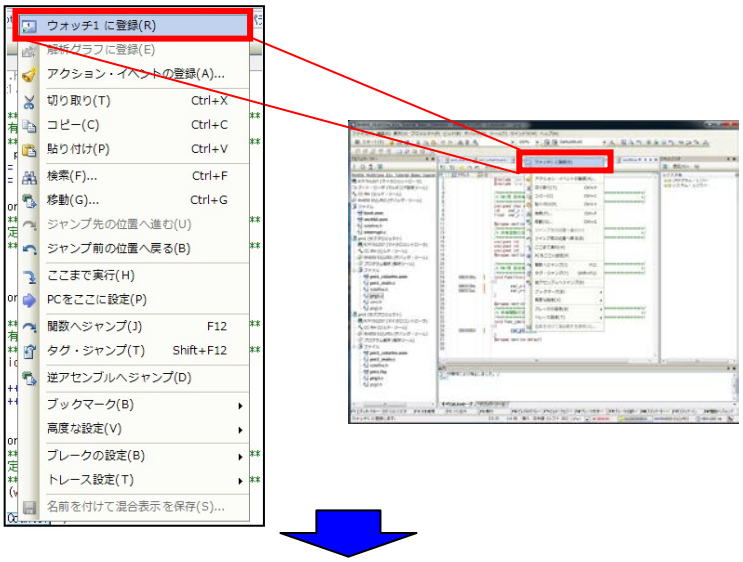
プロジェクト・ツリー上の prg1.c をダブルクリックして、ソースを表示してください。



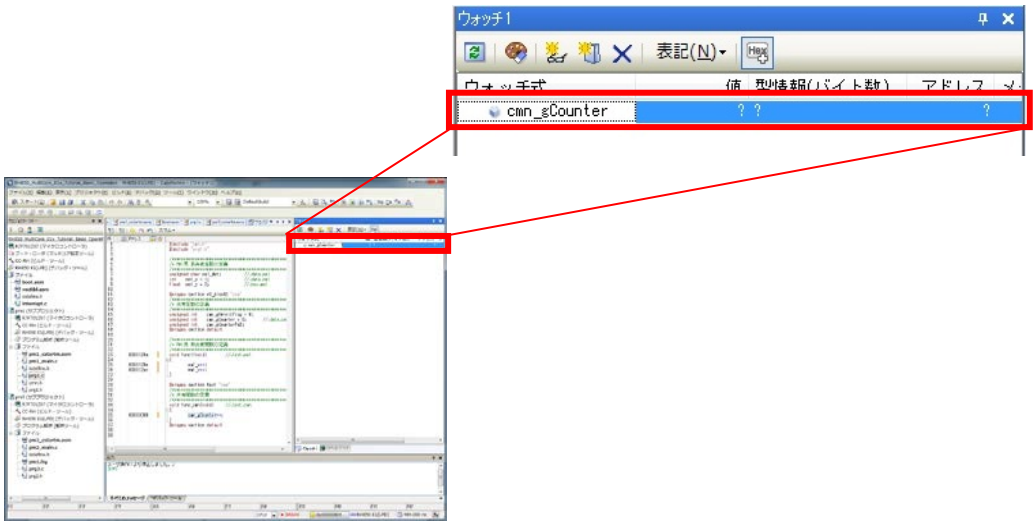
ソース内の変数 g_counter を選択してください。



g_counter を選択した状態で、右クリックして、[ウォッチ 1 に登録]を選択してください。

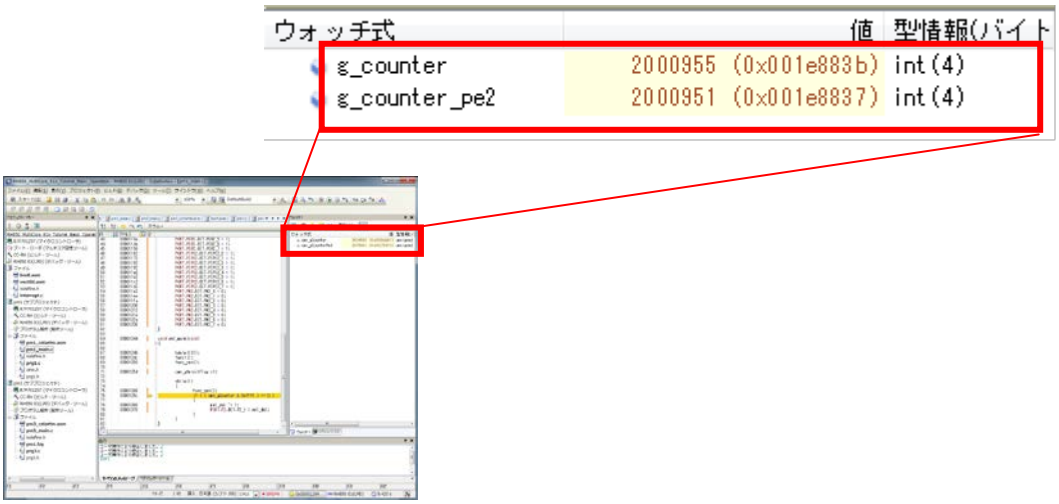


ウォッチパネルが表示されるので、登録されたことを確認してください。現在の g_counter の値は ？ です。



同様にmain2.cからg_counter_pe2をウォッチパネルに登録してください。

メニューの「リセット＆実行ボタン」をクリックしてください。数秒経過後、「停止ボタン」をクリックしてください。


ウォッチ式
値
型情報(バイト
g_counter
2000955 (0x001e883b)
int(4)
g_counter_pe2
2000951 (0x001e8837)
int(4)

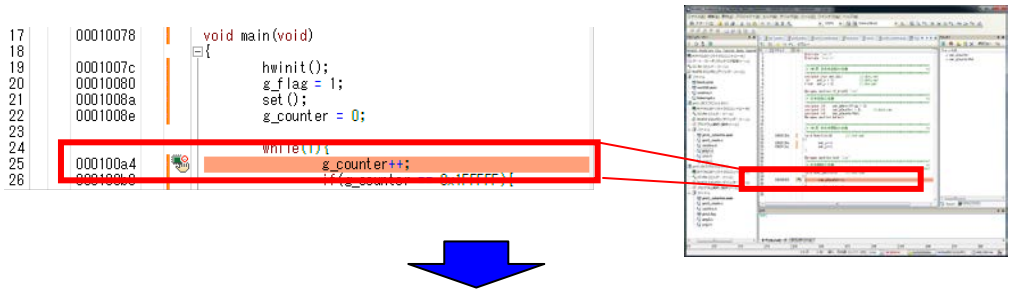
ブレークポイントの設定

ブレークポイントの設定

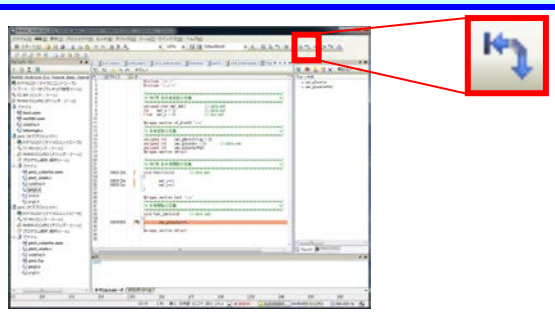
ソース中の意図的な場所でプログラムを停止させたい場合は、ブレークを設定することで、実行前ブレークすることができます。

先ほどウォッチ登録した変数（g_counter）が、どのような値に変化するのかをプログラムを実行→ブレークすることで確認しましょう。

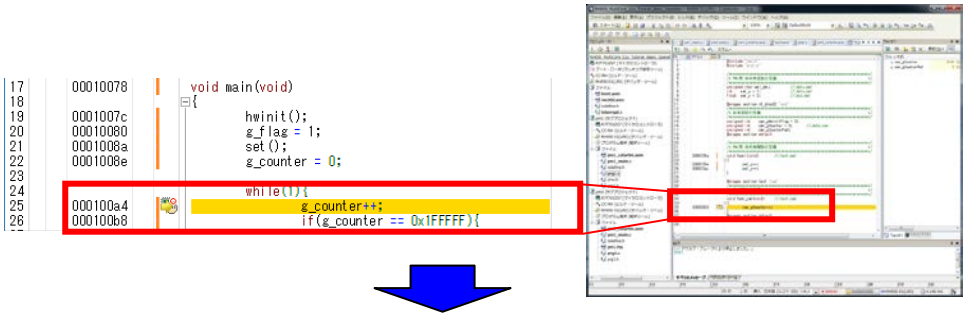
下図のようにソース行の左の空欄をクリックしてください。ハードウエア・ブレークが設定され、行が赤色で表示されます。



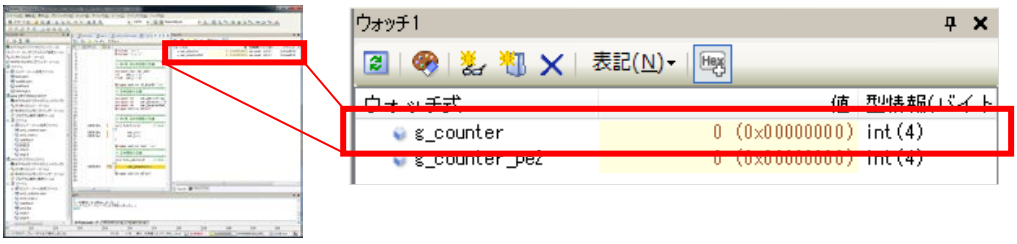
メニューの「リセット＆実行ボタン」をクリックしてください。



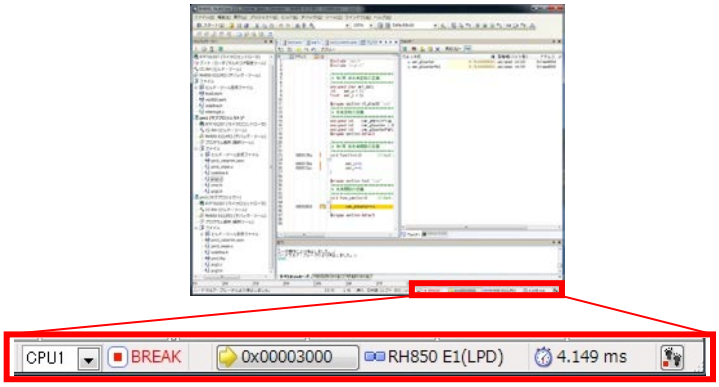
ブレーク設定行でプログラムがブレークし、ブレーク行が黄色で表示されます。



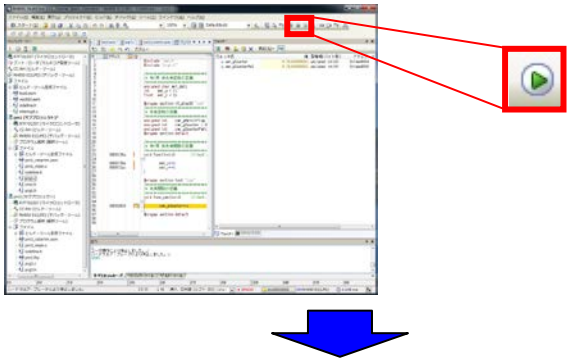
ウォッチパネルを確認すると、[g_counter]の値が 0x0 に設定されています。



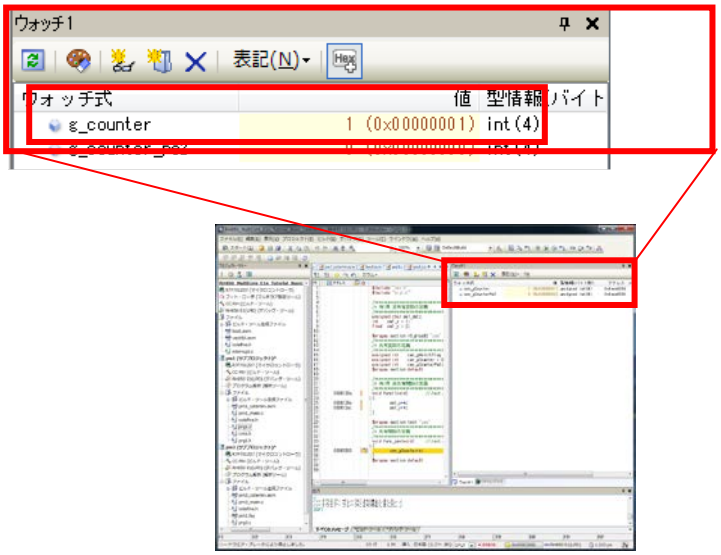
ステータスバーに[BREAK]と表示され、ブレーク時の PC 値が表示されています。



メニューの「実行ボタン」をクリックしてください。



再度、ブレーク設定行でプログラムがブレークし、ウォッチパネルを確認すると、[g_counter]の値が 0x1 にカウントアップされています。

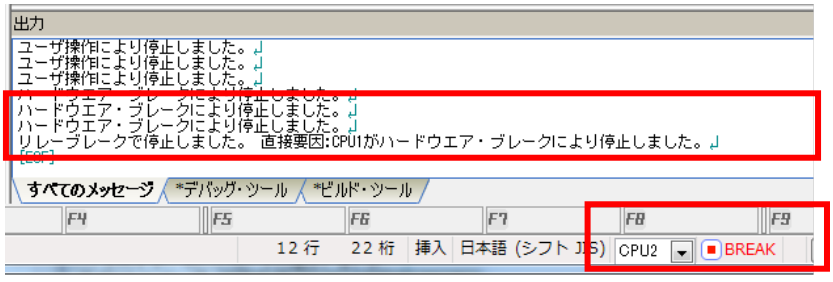


ワンポイントアドバイス

マルチコアのブレークについて

通常、ブレーク時はブレークした PC 位置のプログラムを表示します。マルチコアにおいては、他コア（デバッグ対象で無い方）のブレーク要因でブレークした場合は、自コア（デバッグ対象コア）はブレーク条件の設定されていないアドレスでブレークします。出力パネルでブレーク要因を確認することができます。

下記例では、他コアによるブレーク（リレーブレーク）がブレーク要因であることを示します。



実行履歴の収集

実行履歴の収集

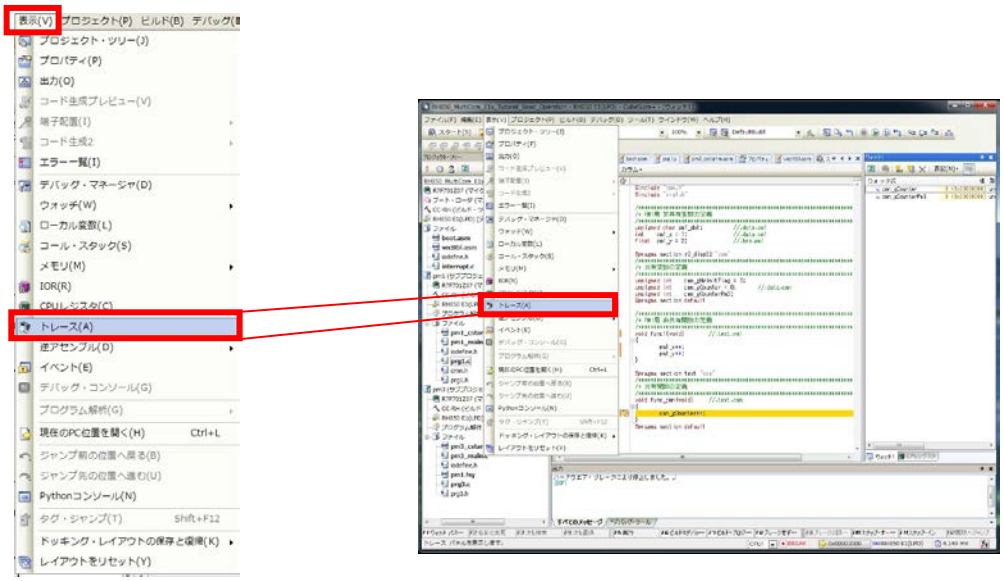
一般的に、プログラムの実行履歴をトレースと呼びます。プログラムが暴走した場合、暴走後のメモリ内容やスタック情報などの情報のみで原因を探ることは非常に困難ですが、トレースを使用し、収集したトレースの内容を解析することにより、暴走するまでの過程を直接探ることができます。

実行履歴の収集

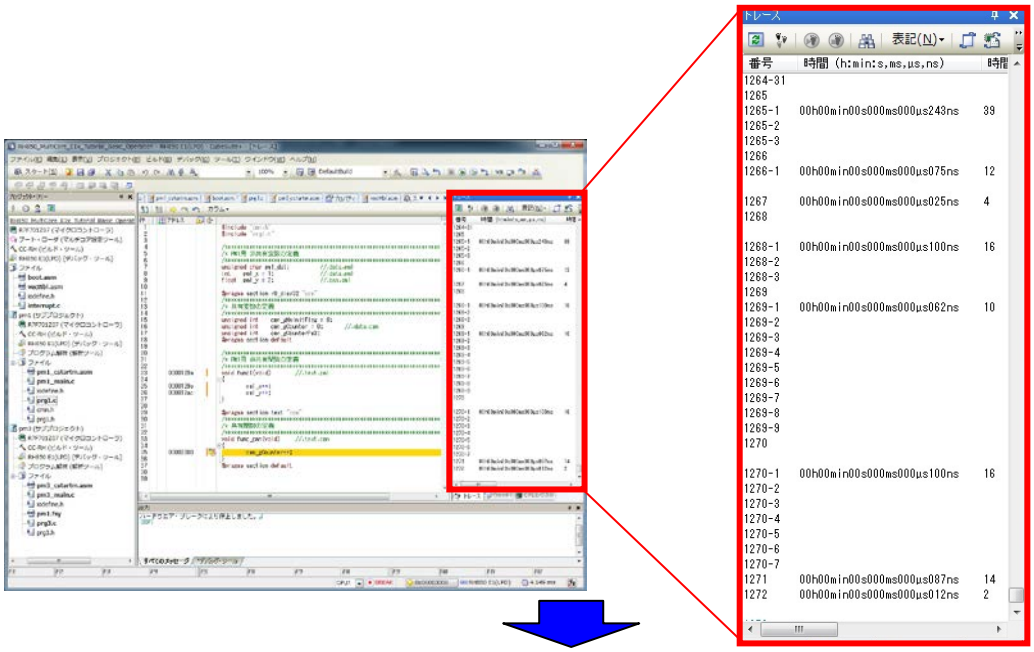
トレース動作の設定

トレース機能が記録を開始すると、現在実行中のプログラムの実行過程をトレースメモリに記録します（プログラムの実行が停止すると、自動的にトレース機能も停止します）。
トレース機能を使用するためには、あらかじめトレースの動作に関する設定を行う必要があります。

メニューの[表示]から[トレース]を選択してください。



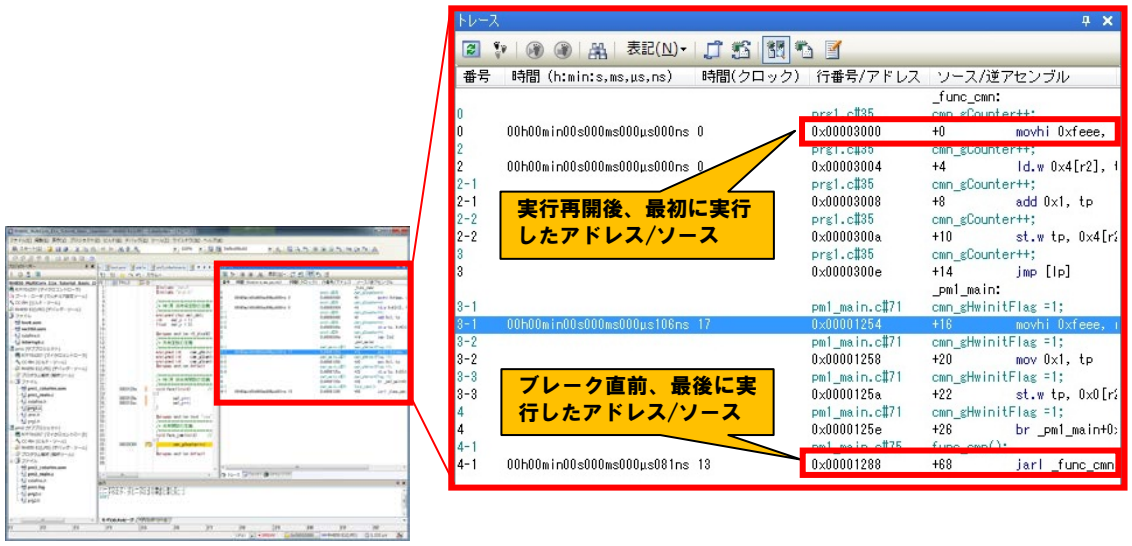
トレースパネルが表示されます。



トレースの設定は、プロパティパネルの[デバッグ・ツールの設定]タブ上の[トレース]カテゴリ内で行います。
[デバッグ・ツールの設定]タブを選択し、以下のように設定してください。

メニューの「実行ボタン」をクリックしてください。

ブレークが発生し、トレースウィンドウに実行履歴が表示されます。

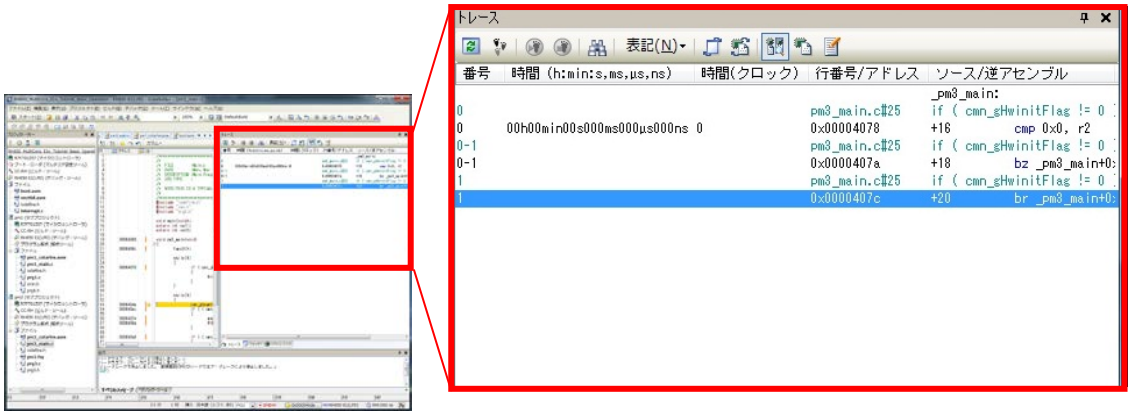


ワンポイントアドバイス

マルチコアのトレースについて

デバッグ対象コアを CPU1 にしてトレースを取得した場合、CPU1 側のトレース情報しか見えません。CPU2 側のトレースを取得したい場合は、デバッグ対象コアを CPU2 に切り替えてプログラムを実行する必要があります。

また、デバッグ対象コアを CPU1 にしてトレースを取得した後、デバッグ対象コアを CPU2 に切り替えても、CPU2 側のトレースは見えません。



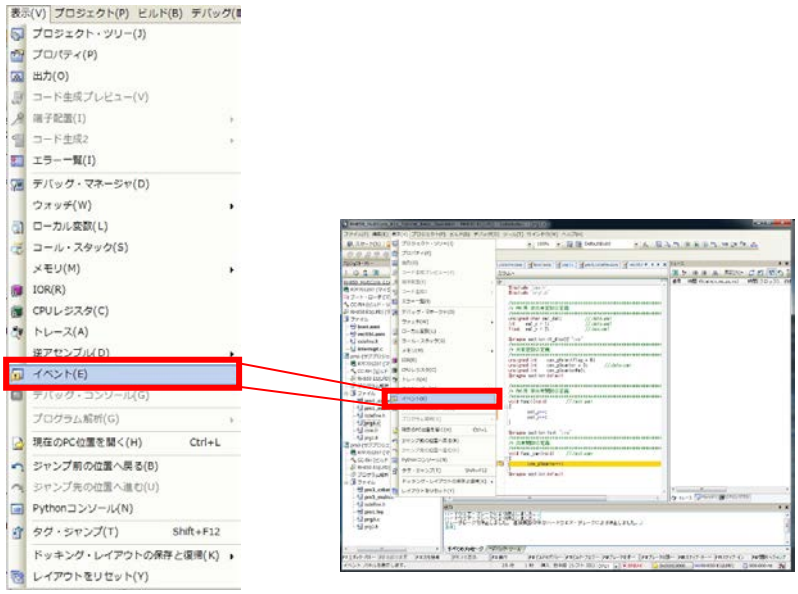
ブレークの解除

ブレークの解除

先ほど設定したブレークの解除を行います。先ほど設定したブレークは、ハードウェアブレークとして設定されています。ハードウェアブレークは、イベントとして、登録されています。イベントを削除することで、ハードウェアブレークを解除します。

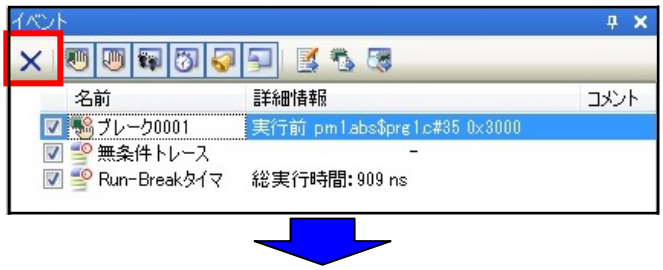
イベントとは、フェッチ、リード、ライトなどマイコンの動作を指しています。そして、イベントはブレーク、トレース等の各デバッグ機能のアクショントリガとして利用できます。先ほどのハードウェアブレークは、（ある特定の番地）をフェッチしたら（実行前）ブレークするというイベントでした。

メニューの[表示]から[イベント]を選択してください。

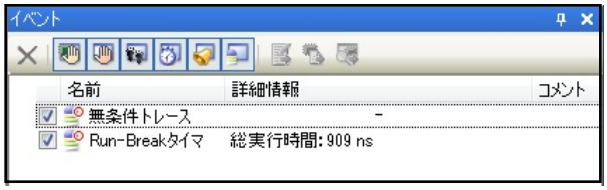


イベントパネルが表示されるので[ブレーク0001]を選択してください。

削除ボタンを左クリックしてください。アクセスブレークの解除ができます。



イベントパネルから[ブレーク 0001]が削除されたことを確認してください。

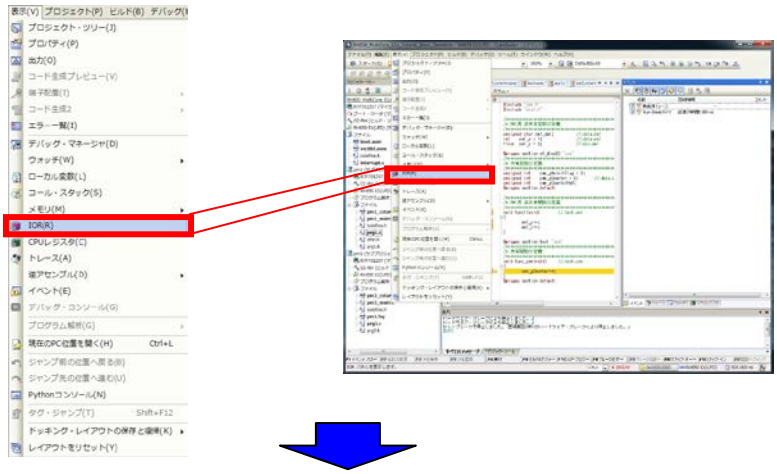


特殊機能レジスタ（IOR）の表示

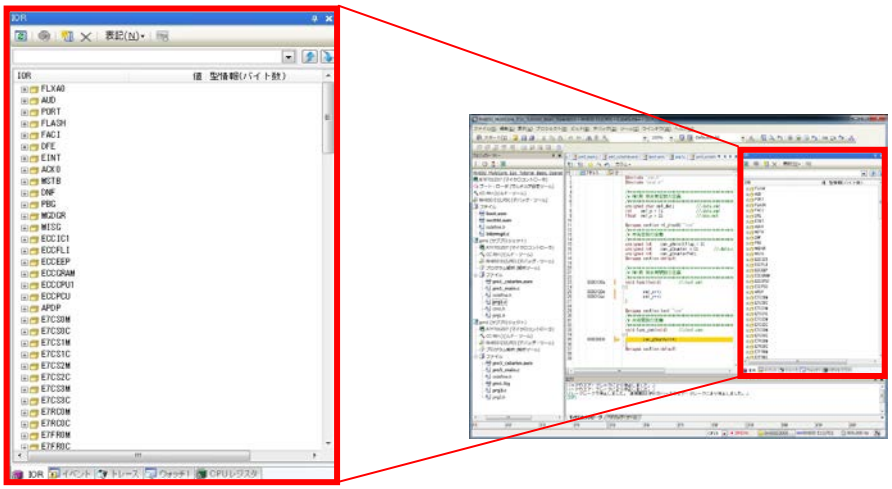
IOR の表示

マイコンの内蔵周辺機能を動作させるレジスタの値が表示されます。見やすいようにフローティングさせてみましょう。

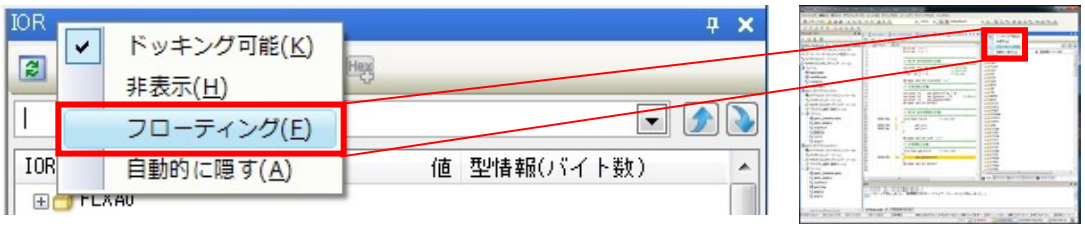
メニューの[表示]から[IOR]を選択してください。



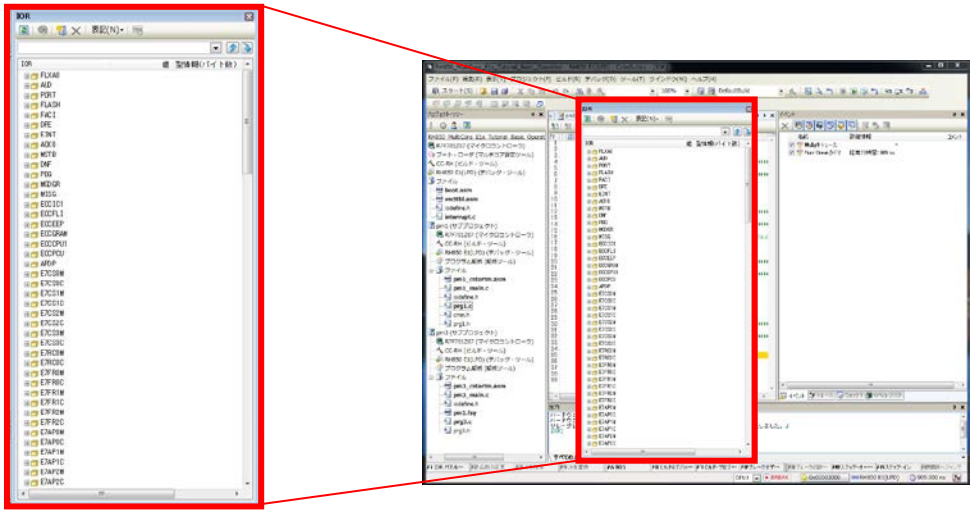
IOR パネルが表示されます。



IOR パネルのタイトルバー上で右クリックし、[フローティング]にチェックしてください。



IOR パネルがフローティングされ、見やすくなりました。



ワンポイントアドバイス

IOR のビット表示について

IOR パネルは、IOR のビット表示に対応していません。このため、IOR をビット表示で確認したい場合は、ウォッチパネルに登録して参照する必要があります。
ウォッチパネルのコンテキストメニューから”新規ウォッチ式を追加”を選択して、ウォッチ式を入力します。ビットレジスタを指定する場合は、下記のように入力します。

AAA0.BBB.CCC
＜モジュール名＞．＜レジスタ名＞．＜ビット名＞

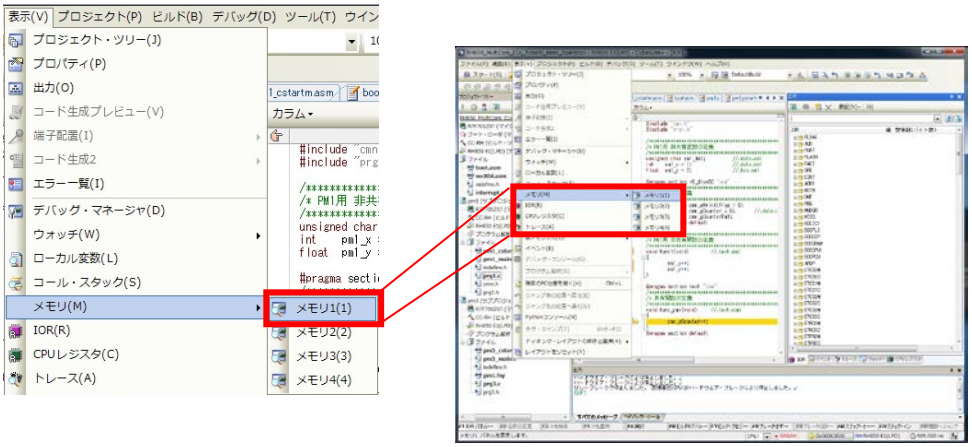
【例】
（汎用 I/O）ポートの P2 レジスタの P2_1 ビットをウォッチパネルに登録するウォッチ式
PORT.P2.P2_1

メモリの表示

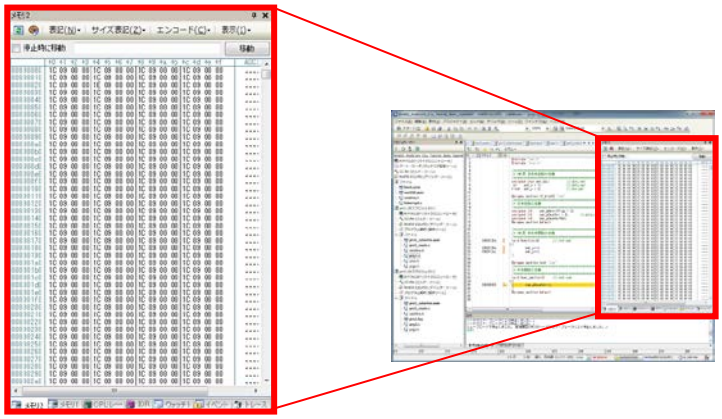
メモリパネルの表示

メモリ状態が表示されます。メモリパネルは4つありますが、今回は 2つを表示させます。[メモリ1]、[メモリ2]を同時に表示させると、デフォルトではタブ表示になり、どちらか一方しか見ることができません。これを、並べて見えるようにドッキングさせてみましょう。

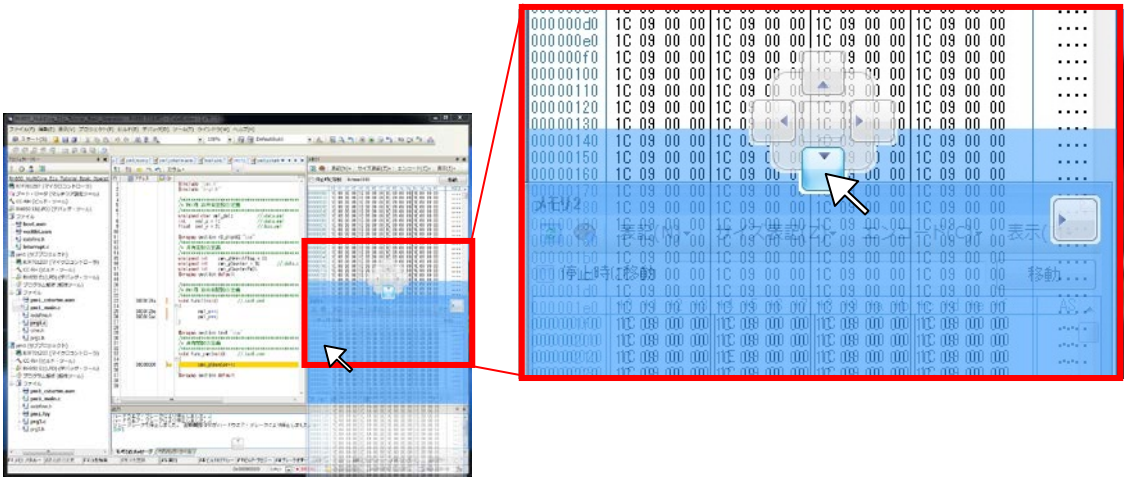
[表示]メニュー → [メモリ 1]を選択してください。



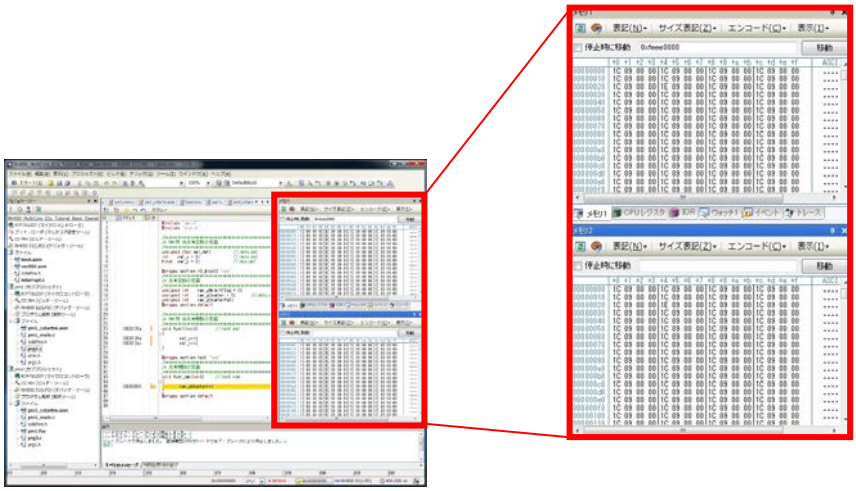
メモリ 1 パネルが表示されます。（メモリ 2 パネルも同様に表示させてください。）



メモリ 2 パネルをフローティング状態にして、出力パネルにドッキングしてください。



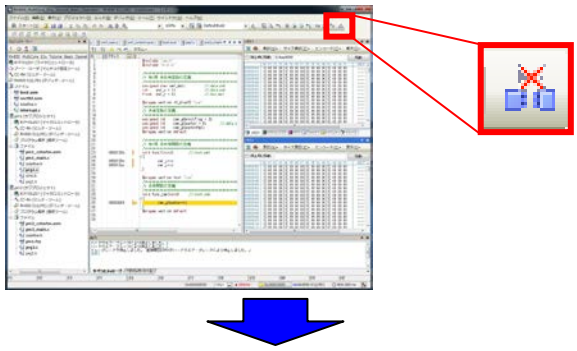
メモリ 1 とメモリ 2 を同時に参照することが可能になりました。



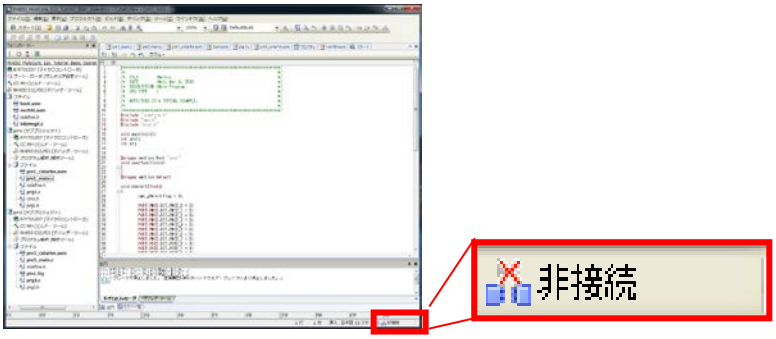
デバッグ・ツールからの切断

デバッグを終了する場合は、デバッグ・ツールの切断を行います。

「デバッグ・ツールから切断ボタン」を選択してください。



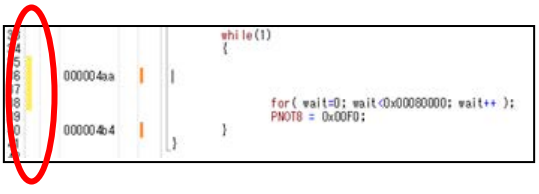
メイン・ウインドウのステータスバーが下図のように[非接続]と表示され、デバッグが終了します。



ワンポイントアドバイス

プログラムのダウンロードについて

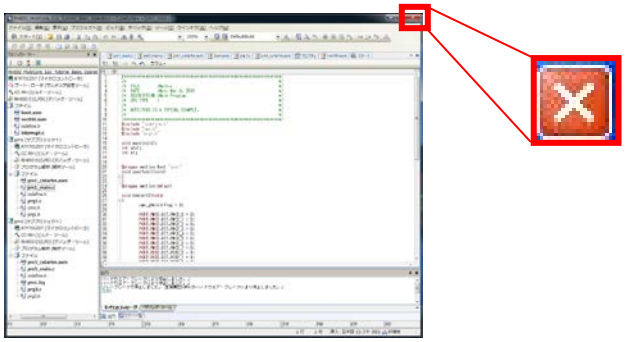
プログラムがダウンロードされた状態で、プログラムの変更をした場合は、再度ビルドしてダウンロードを行う必要があります。ダウンロード後にプログラムを変更した場合、図のように黄色く（もしくは緑）なり、ブレーク設定ができなくなります。



終了の方法

終了の方法を説明します。

メイン・ウインドウの「閉じるボタン」をクリックしてください。

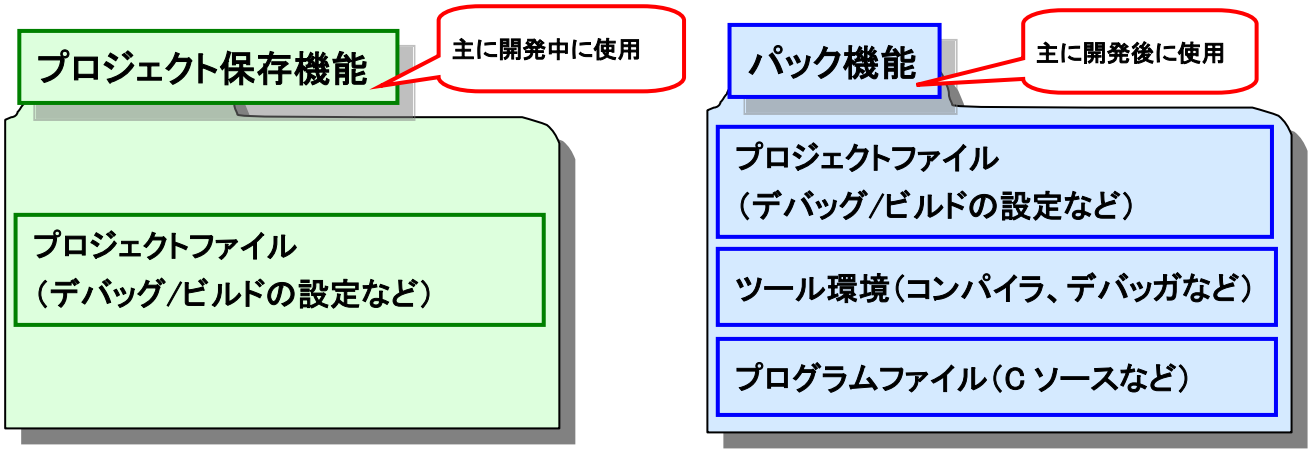


メイン・ウインドウが閉じ、終了します。環境が保存されていない場合は、保存を行う/行わないの選択ウインドウが表示されますので、指示に従い進めてください。

ワンポイントアドバイス

開発環境の保存について(プロジェクト保存機能とパック機能)

開発環境の保存として、CS+では2つの機能(プロジェクト保存機能とパック機能)をサポートしています。それぞれ、下図に示す内容が保存されます。お客さまの開発フェーズに合わせて保存機能の使い分けをすると便利です。



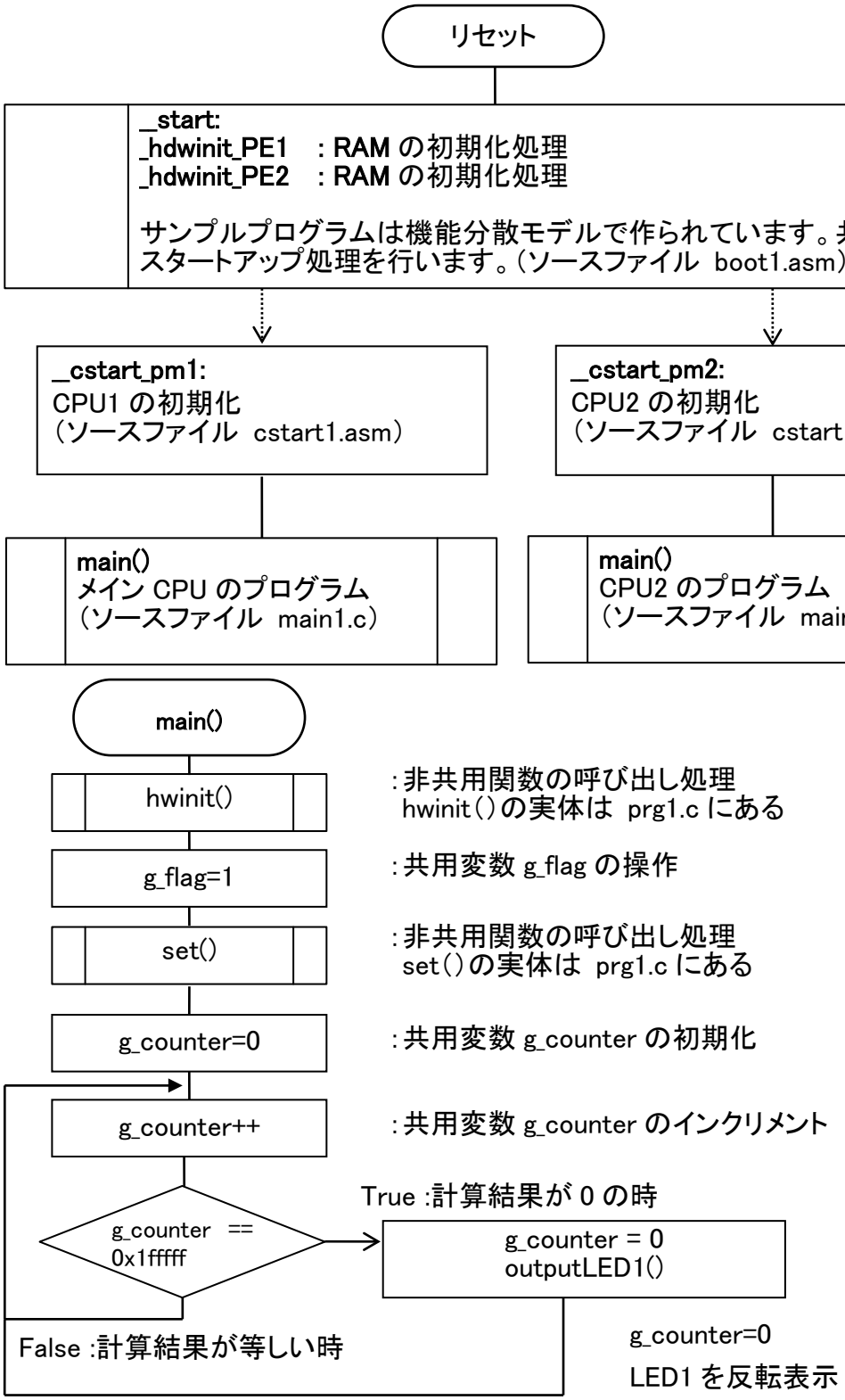
書き込みについて

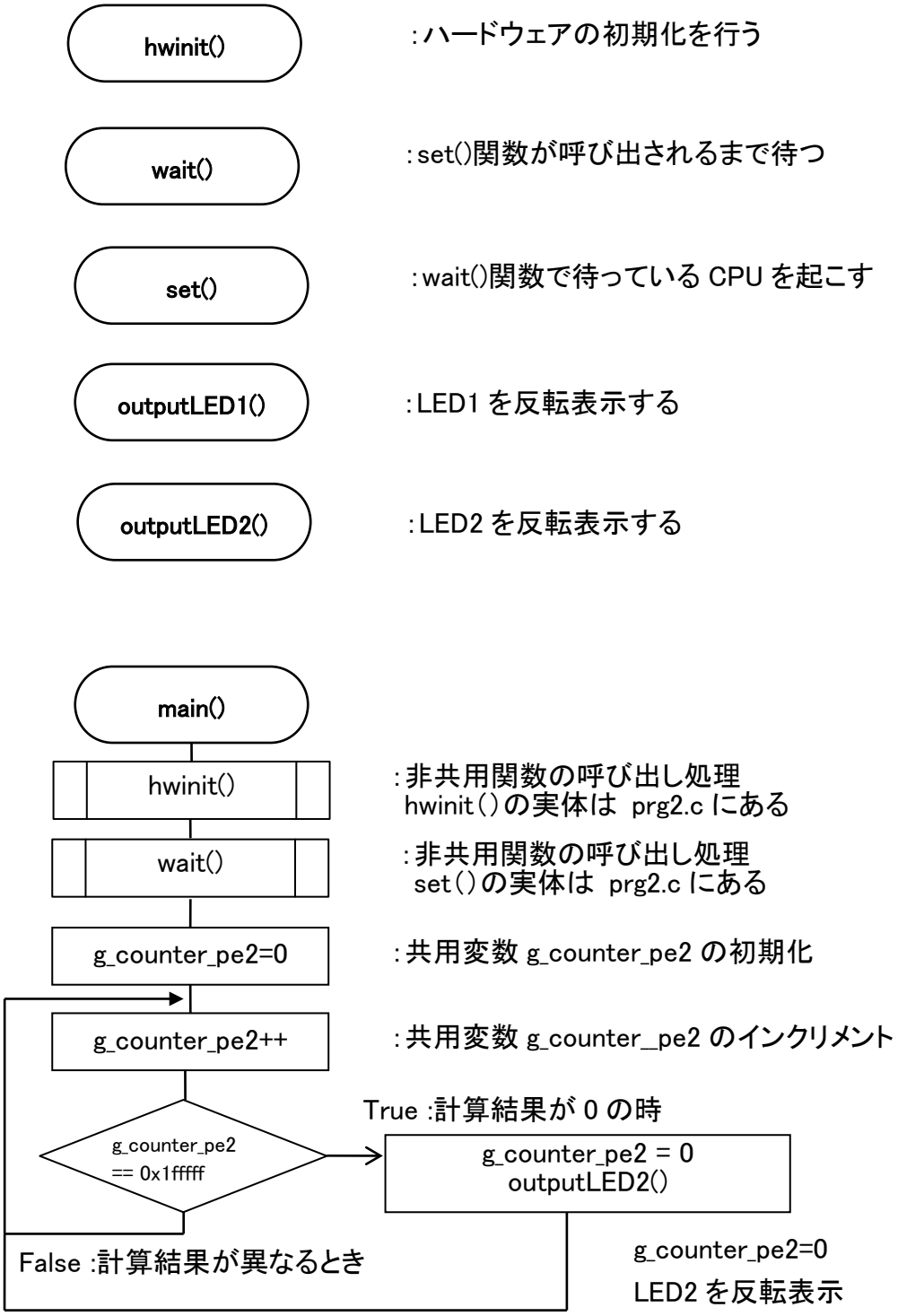
E1 エミュレータでマイコンに.hex ファイルを書き込む場合は、Renesas Flash Programmer(RFP)をご使用ください。

・Renesas Flash Programmer(RFP)は、[スタート] → [すべてのプログラム] → [Renesas Electronics Utilities] → [書き込みツール]から起動することができます。
・使用方法は、ユーザーズマニュアルをご参照ください。

サンプルプログラムの説明

サンプルプログラムの流れ図を以下に示します。




hwinit()
:ハードウェアの初期化を行う
wait()
:set()関数が呼び出されるまで待つ
set()
:wait()関数で待っている CPU を起こす
outputLED1()
:LED1 を反転表示する
outputLED2()
:LED2 を反転表示する
main()
hwinit()
:非共用関数の呼び出し処理
hwinit()の実体は prg2.c にある
wait()
:非共用関数の呼び出し処理
set()の実体は prg2.c にある
g_counter_pe2=0
:共用変数 g_counter_pe2 の初期化
g_counter_pe2++
:共用変数 g_counter_pe2 のインクリメント
True :計算結果が 0 の時
g_counter_pe2
== 0x1fffff
g_counter_pe2 = 0
outputLED2()
False :計算結果が異なるとき
g_counter_pe2=0
LED2 を反転表示

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/