RA0 シリーズ

RA0 シリーズ チュートリアル

要旨

ルネサス製 RA0 シリーズ MCU を対象に、統合開発環境 e^2 studio と Fast Prototyping Board を用いてプロジェクト新規作成からプログラム作成、およびデバッグまでの操作手順を説明します。

本書では、RA0E1 と FPB-RA0E1 を具体例として取り上げますが、他 RA0 シリーズの MCU も操作方法は同様ですので参考としてください。

対象デバイス

RA0 シリーズ

参考ドキュメント

[1] Renesas RA ファミリ RA0E1 グループ ユーザーズマニュアル ハードウェア編(R01UH1040)

[2] Renesas RA ファミリ FPB-RA0E1 v1 ユーザーズマニュアル(R20UT5378)

[3] 統合開発環境 e^2 studio 2022-07 以上 ユーザーズマニュアル クイックスタートガイド ルネサスマイクロコントローラ RA ファミリ(R20UT5210)

目次

1. 開発環境

本アプリケーションノートでは、下記の開発環境を用いて説明します。

1.1 ハードウェア環境

以下のハードウェアを使用します。

- ボード：FPB-RA0E1(製品型名：RTK7FPA0E1S00001BJ)
  ボードと PC を USB ケーブルで接続して下さい。

1.2 ソフトウェア環境

本書では以下のソフトウェアを使用します。あらかじめソフトウェアをインストールしてください。

- 統合開発環境
  — e^2 studio : 2024-01.1 以降
- コンパイラ
  — GNU ARM Embedded : 13.2.1.arm-13.7 以降
- Flexible Software Package
  — Version : 5.2.0 以降

2. ソフトウェア概要

本アプリケーションノートで作成するプログラムの仕様を説明します。

2.1 作成するプログラム

RA0E1 グループのタイマ機能を用いて 500ms 周期を作り、500ms ごとに発生する割り込みの中で FPB-RA0E1 ボード上の 2 つの LED を交互に点灯するプログラムを作成します。

使用するボードと LED の位置を示します。

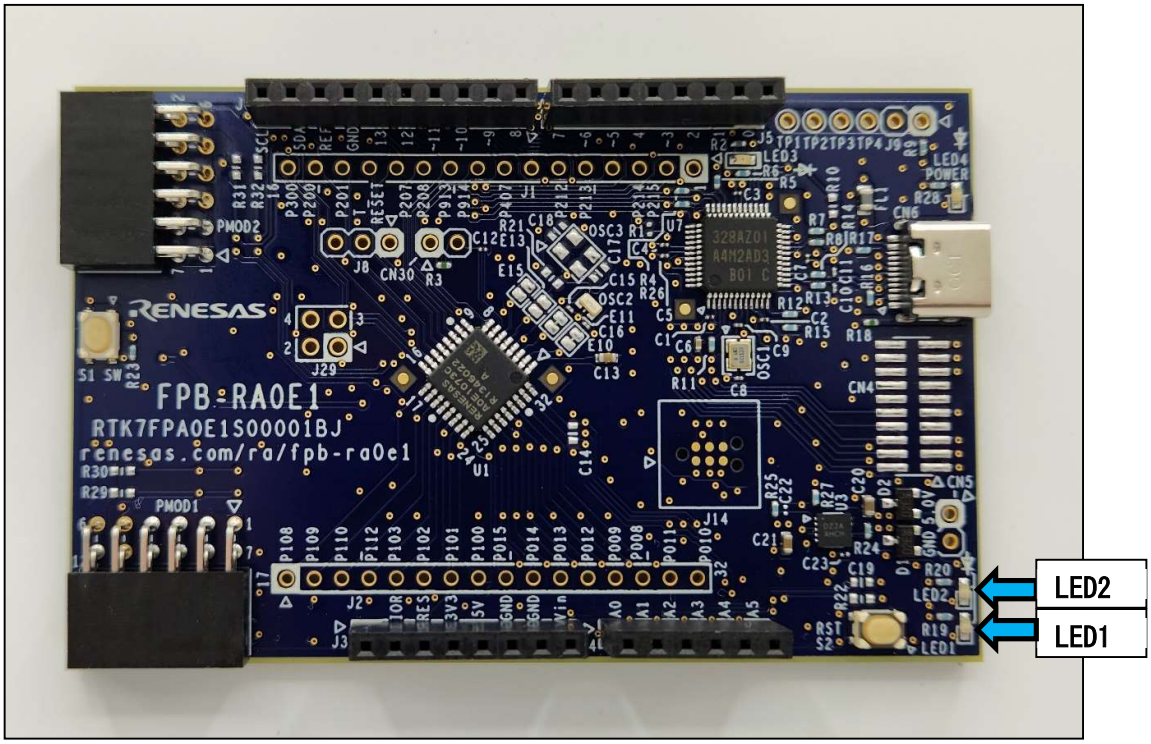


図 2-1 ボード上の LED 位置

2.2 使用するリソース

プログラム作成にあたり、RA0E1 グループの以下のリソースを使用します。

2.2.1 クロック

HOCO クロック周波数: 32MHz

HOCO クロック分周: 1 分周

ICLK 選択ソース：HOCO

ICLK 周波数：32MHz

TAU CK00 供給クロック ：62.5kHz

2.2.2 タイマ

使用するタイマ機能：	TAU チャンネル 0
チャンネル 0 に供給する TAU クロック：	CK00
チャンネル 0 の動作モード：	インターバルタイマモード
インターバルタイマ周期：	500ms
使用する FSP のソフトウェアスタック：	r_tau

2.2.3 汎用ポート

使用するポート ：	P008(LED1) / P009(LED2)
使用する FSP のソフトウェアスタック ：	r_ioport

3. プログラム作成方法

この章ではプロジェクトの作成から FSP を用いた周辺機能の設定方法、コーディング、ビルドの方法を説明します。

3.1 プロジェクト新規作成

3.1.1 プロジェクト起動

1. e² studio のメニューバーから
「新規」→「Renesas C/C++ Project」→「Renesas RA」を選択します。

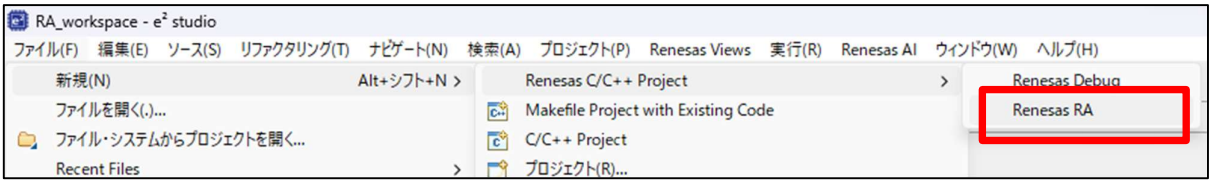


図 3-1 プロジェクト起動画面

2. 「Renesas RA C/C++ Project」を選択し、「次へ」をクリックします。

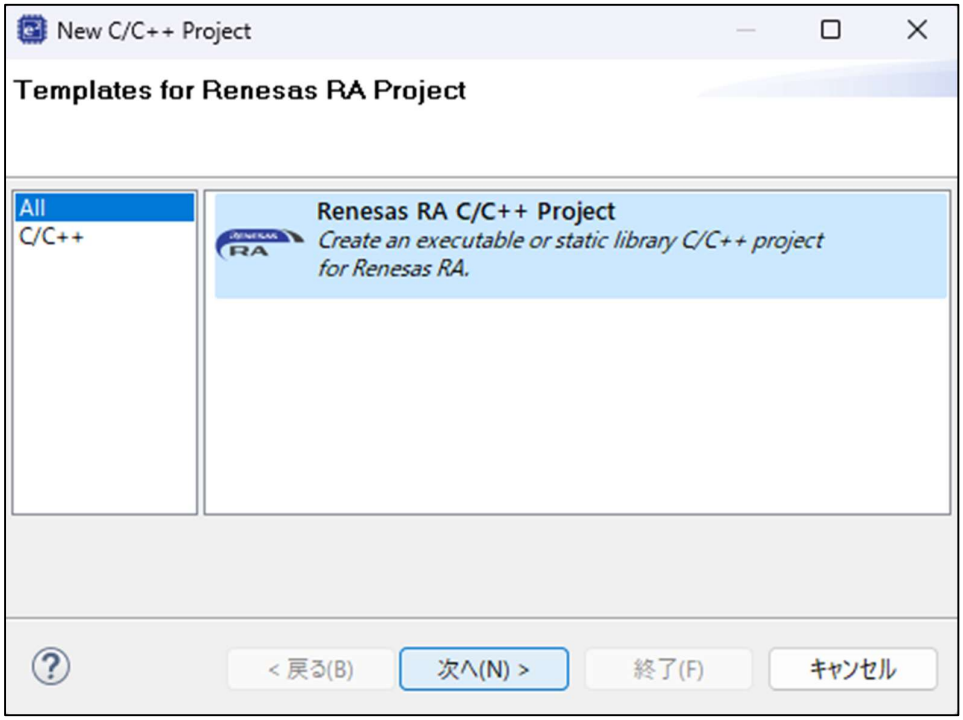


図 3-2 プロジェクト起動画面

3. Project name に 任意の名前を記述し、「次へ」をクリックします。
”デフォルト・ロケーションの使用”にチェックを入れた場合、プロジェクトの作成場所は下に表示されているパスに作成されます。任意の場所に作成したい場合は、チェックを外してパスを設定してください。

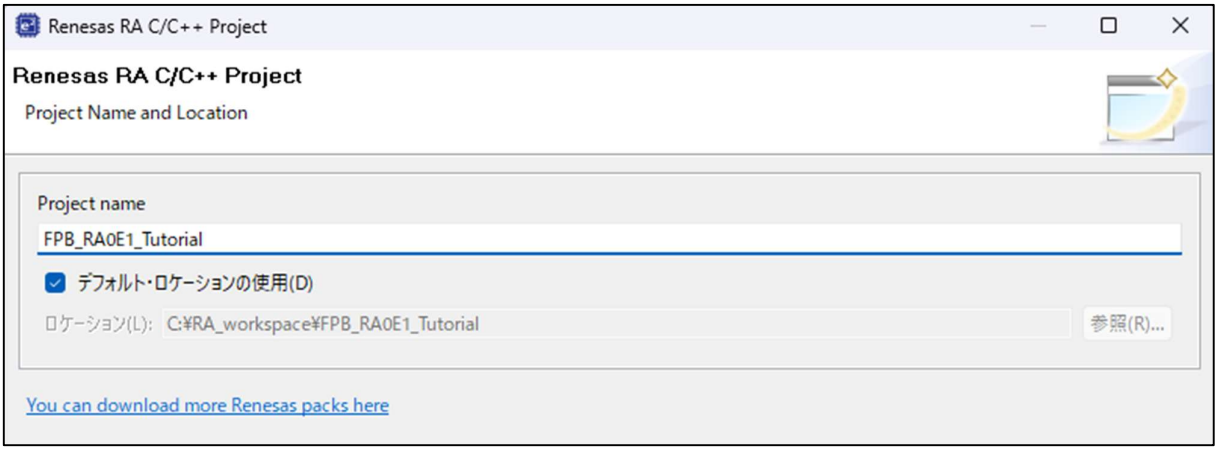


図 3-3 プロジェクト起動画面

3.1.2　デバイス・ツール設定画面

Device and Tools Selection　画面では以下のように設定します。

1. FSP Version：最新版を選択します。
   Language　：　C を選択します。

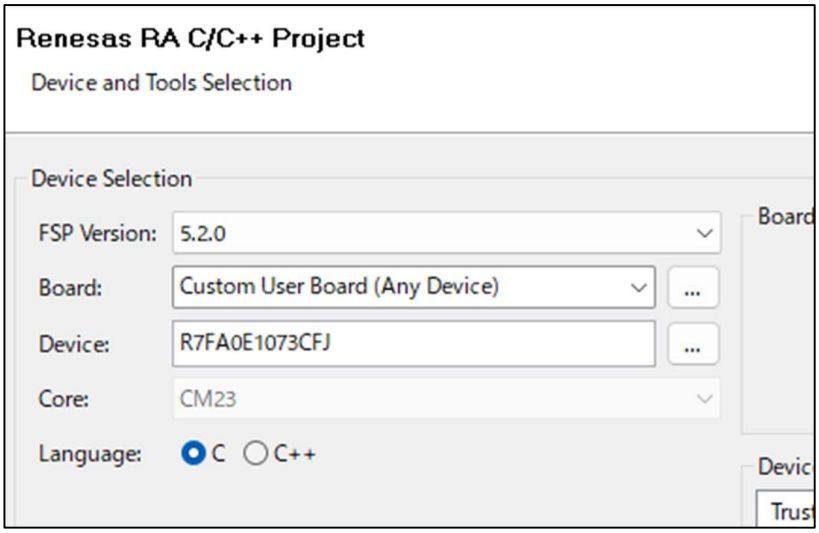


図 3-4　デバイス・ツール設定画面

2. Board：一覧から **FPB-RA0E1** を選択します。
   (Core は自動的に設定されます。)

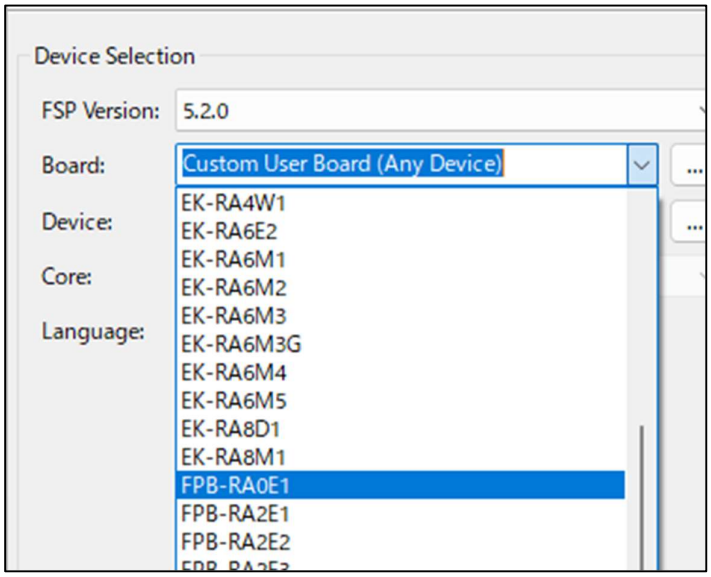


図 3-5　　デバイス・ツール設定画面

3. Toolchains の設定：“GNU ARM Embedded”を選択します。バージョンの一覧より、最新版を選択します。

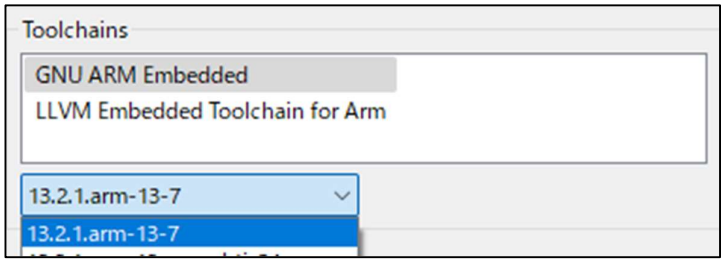


図 3-6 デバイス・ツール設定画面

4. Debugger の設定：J-Link ARM を選択します。

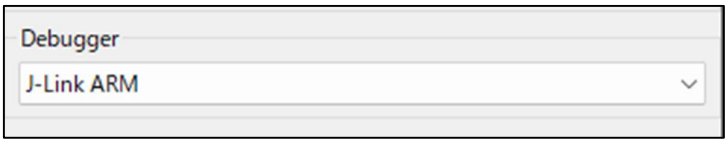


図 3-7 デバイス・ツール設定画面

設定は完了です。

5. 赤枠部分の設定を確認して「次へ」をクリックします。

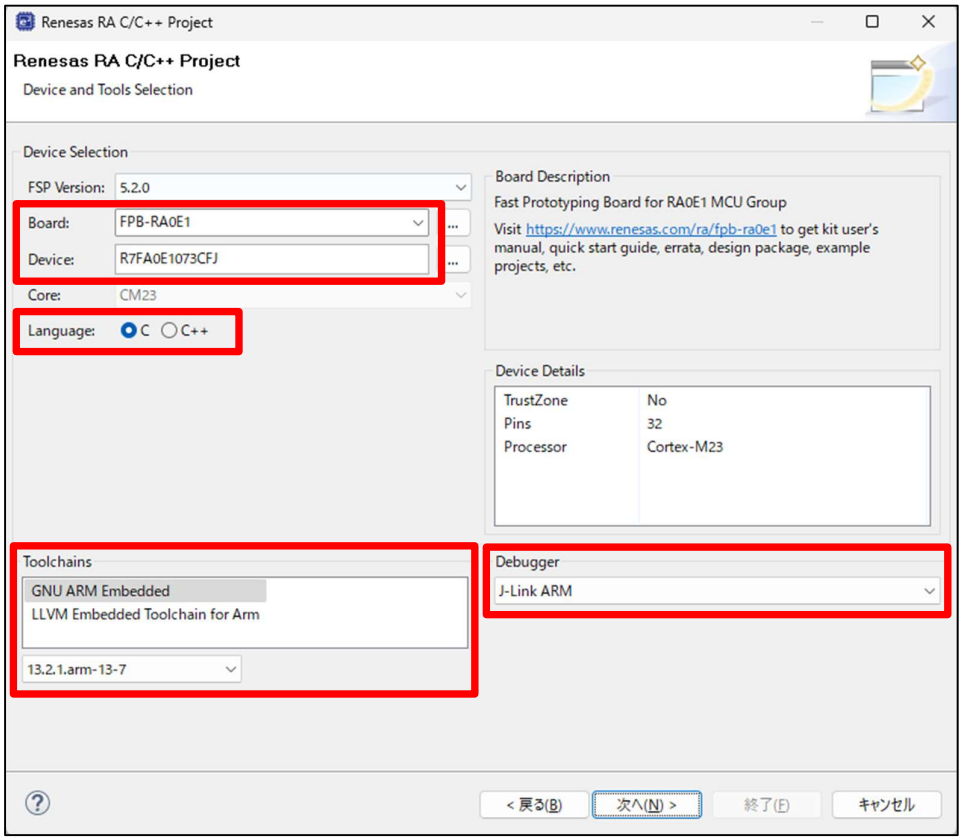


図 3-8 デバイス・ツール設定画面

3.1.3　ビルドアーティファクト設定

本アプリケーションノートではプログラムから実行ファイルを生成しますので Executable を選択します。

RTOS は使いませんので No RTOS を選択します。

「次へ」をクリックします。

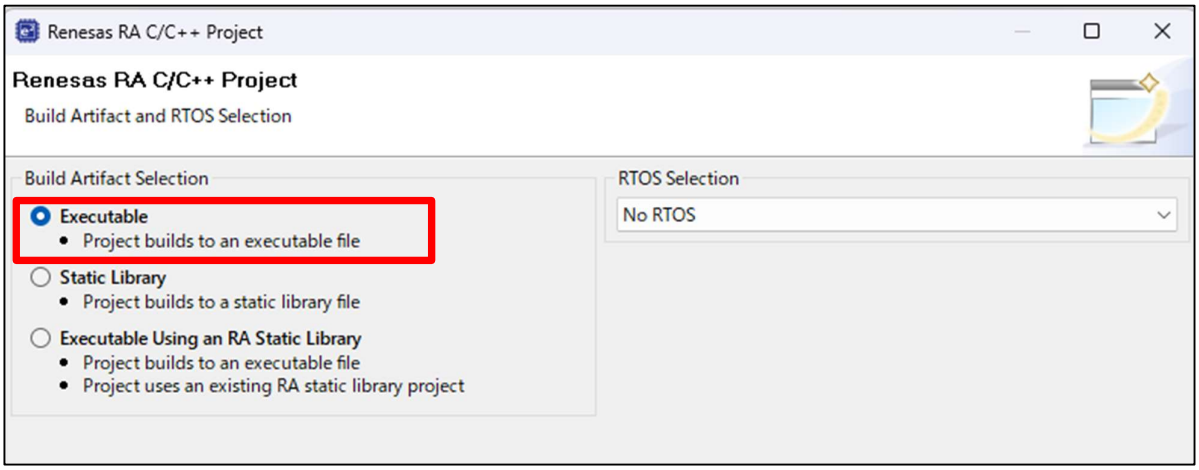


図 3-9　ビルドアーティファクト設定画面

3.1.4 テンプレートタイプの設定

1. 本アプリケーションノートでは FSP の使い方を交えて説明しますので今回は Bare Metal - Minimal を選択します。その後、「終了」をクリックしてください。

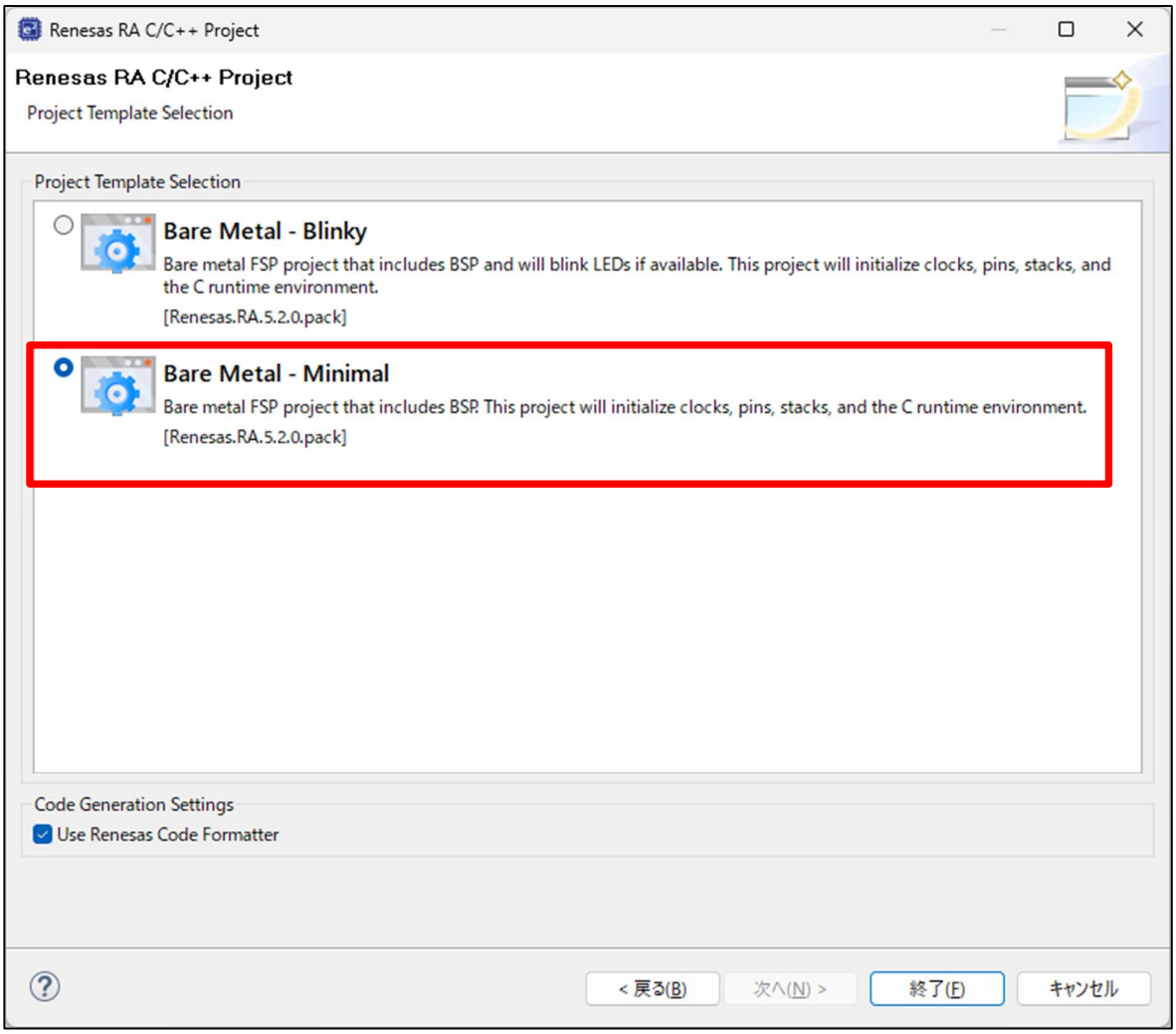


図 3-10　テンプレートタイプの設定画面

2. パースペクティブ画面表示のポップアップが開きますので、「パースペクティブを開く」をクリックします。
   ※「いいえ」をクリックしても問題ありません。

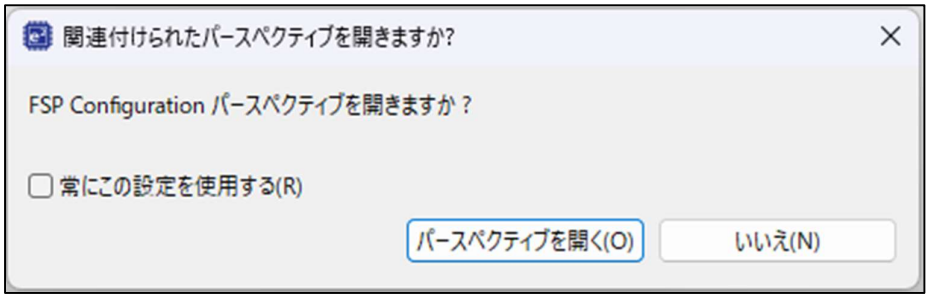


図 3-11　テンプレートタイプの設定画面

3. プロジェクトの作成が完了しました。

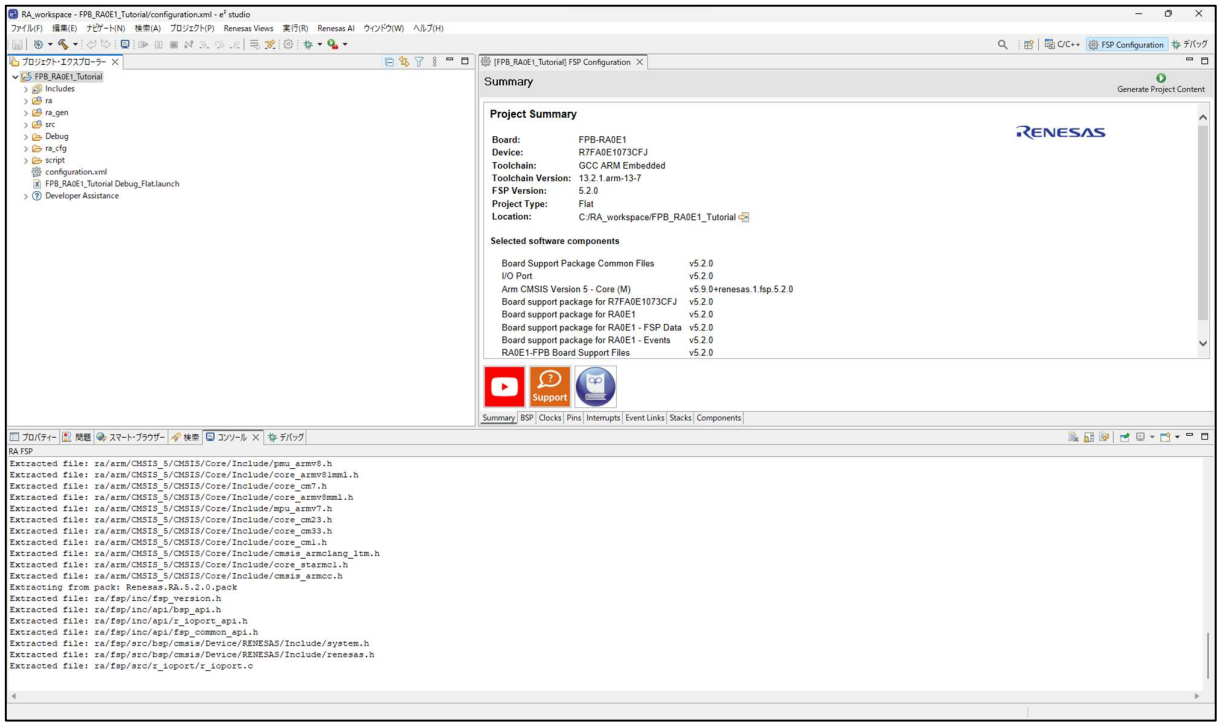


図 3-12 テンプレートタイプの設定画面

3.2 FSP Configurator 設定

FSP Configurator を用いて、システムのクロック設定や、周辺機能の初期設定を行います。

3.2.1 FSP Configurator 画面の呼び出し方法

プロジェクト・エクスプローラーの configuration.xml ファイルをダブルクリックして、設定画面を表示します。

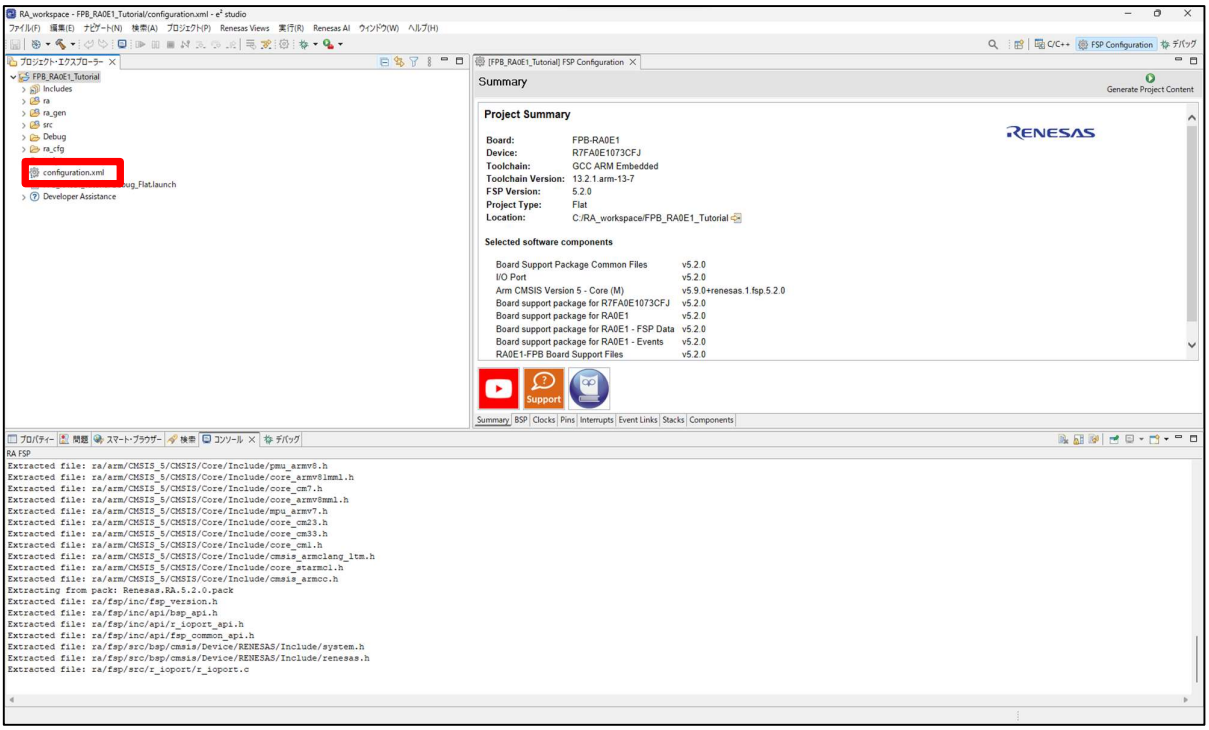


図 3-13 FSP Configurator 画面の呼び出し

3.2.2　クロック設定画面

1. “Clocks”タブを選択してクロック設定画面を開きます。デフォルト画面は以下のようになっていることを確認します。
   各クロックの経路は太い矢印で示されています。

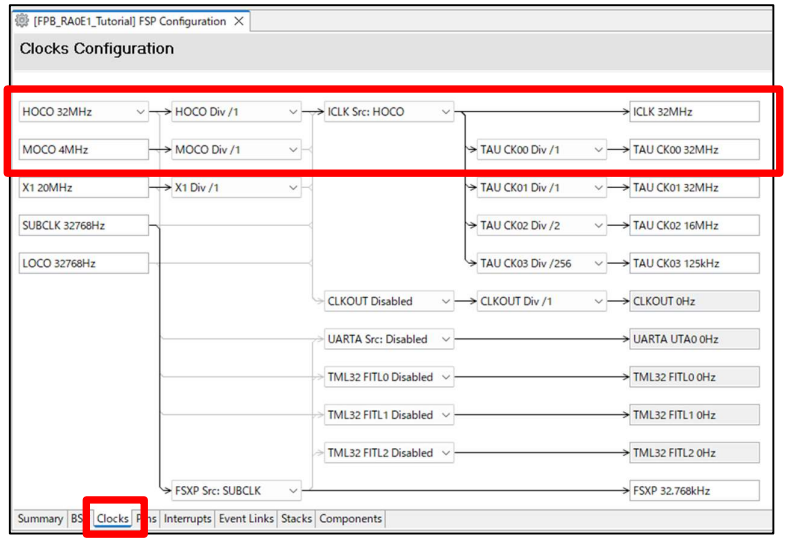


図 3-14　クロック設定画面

2. 今回作成するプロジェクトでは 500ms 周期を作成します。供給クロック CK00 が 32MHz のままでは作成できませんので、62.5kHz までダウンクロックする必要があります。
   TAU CK00 Div /1 をクリックし、リストから **TAU CK00 Div /512** を選択します。

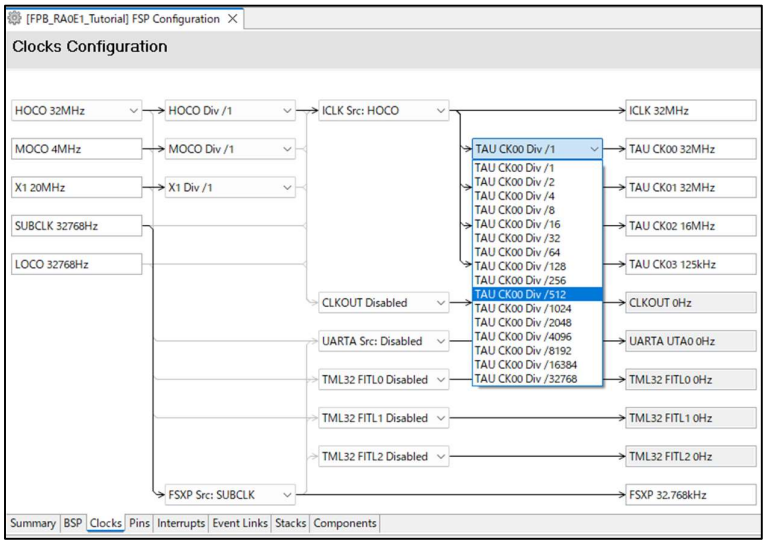


図 3-15　クロック設定画面

3. TAU CK00 の供給クロックが 62.500kHz になっていることを確認します。

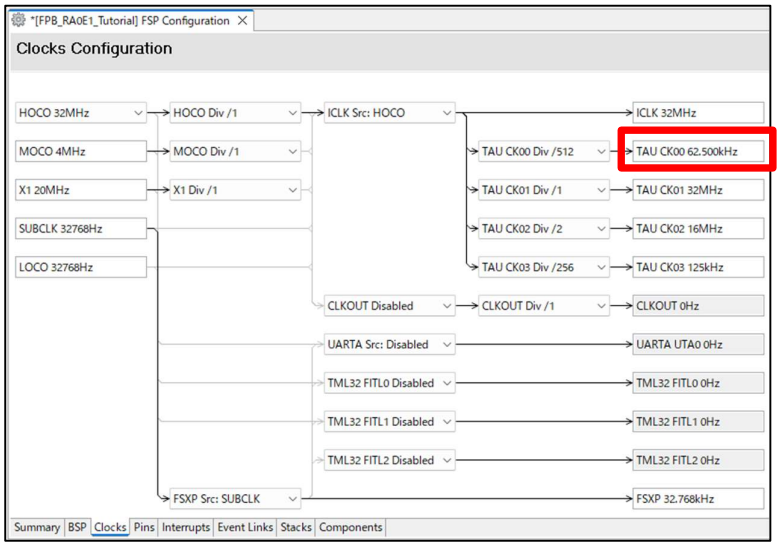


図 3-16　クロック設定画面

3.2.3　ピンの設定

1. 「Pins」タブ、「端子機能」タブと選択し、Pin Selection の一覧から LED1 に割り当たっている P008 を選択します。
「3.1.2 デバイス・ツール設定画面」でボードを選択しているため、一部の端子はシンボル名や入出力設定が施されています。
P008 は “LED1”というシンボルが設定されており、プログラム実装時にこの名前を使用することができます。

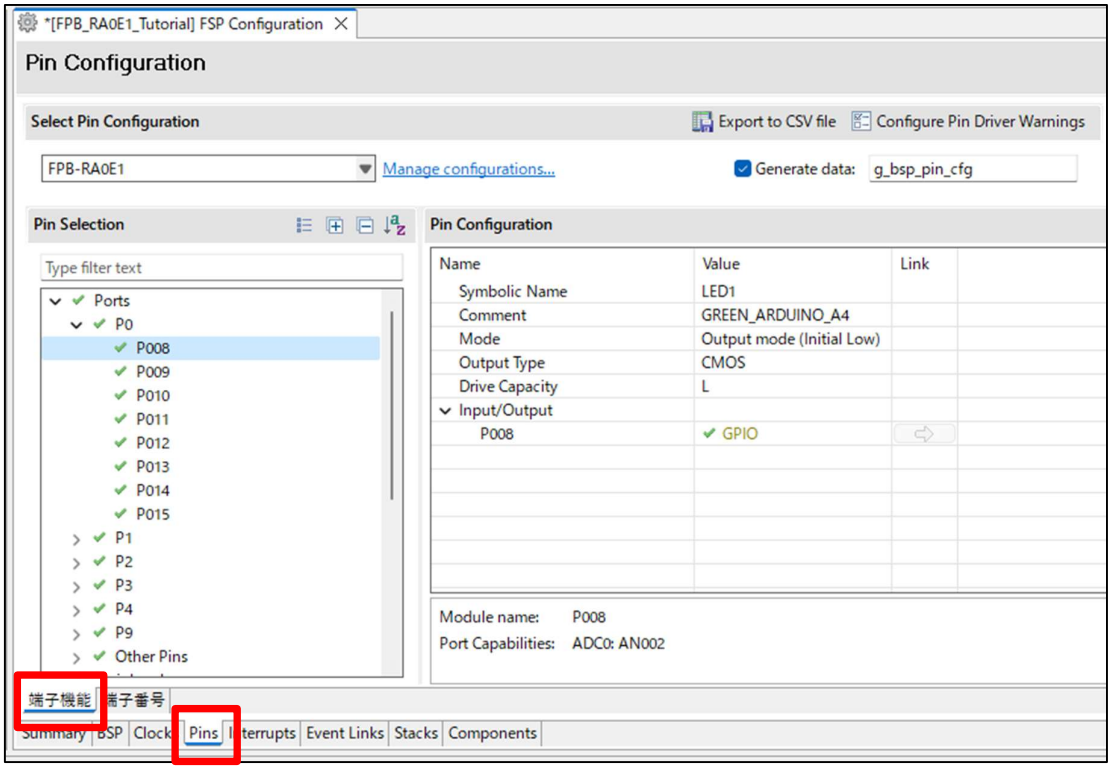


図 3-17　ピンの設定画面

2. P008 は初期状態を点灯(High 出力)とするため、Mode の値をリスト一覧から 「**Output mode (Initial High)**」にします。

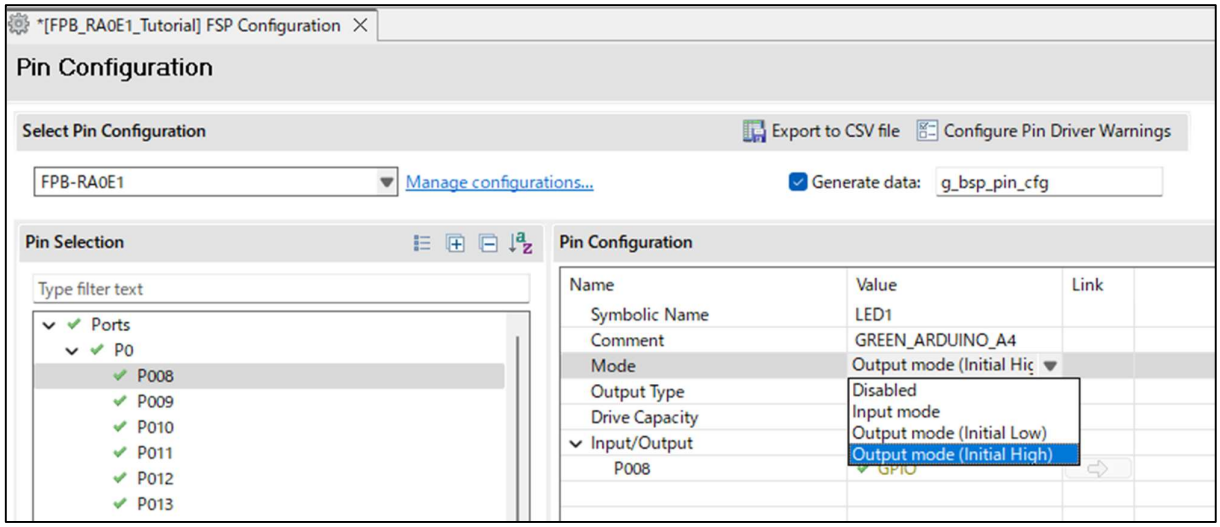


図 3-18 ピンの設定画面

3. P009(LED2)は初期状態を消灯(Low 出力)とするため、以下の設定とします。

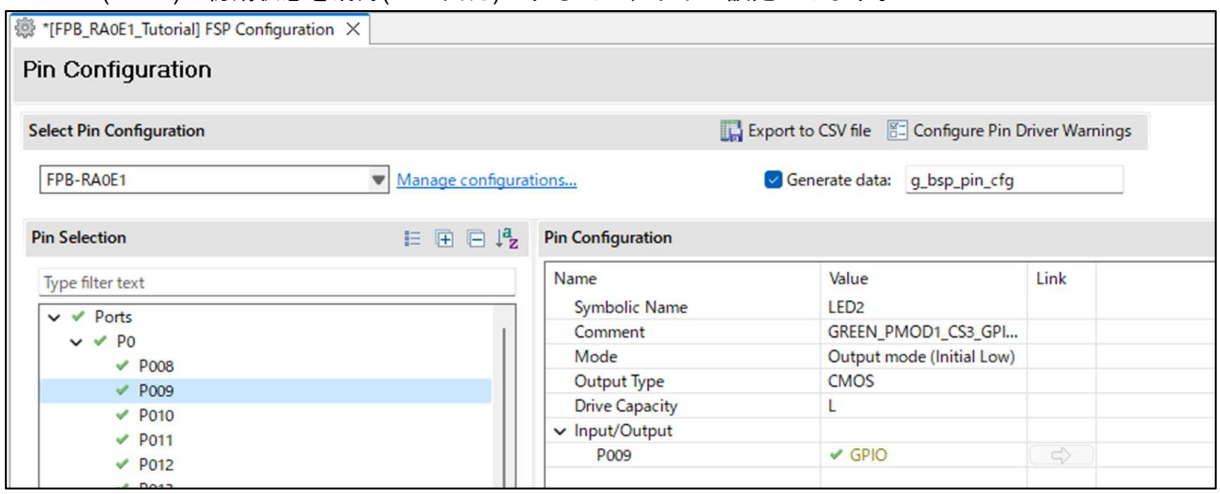


図 3-19 ピンの設定画面

3.2.4　タイマ機能の設定

1. “Stacks”タブを選択します。この画面でタイマの機能を追加していきます。

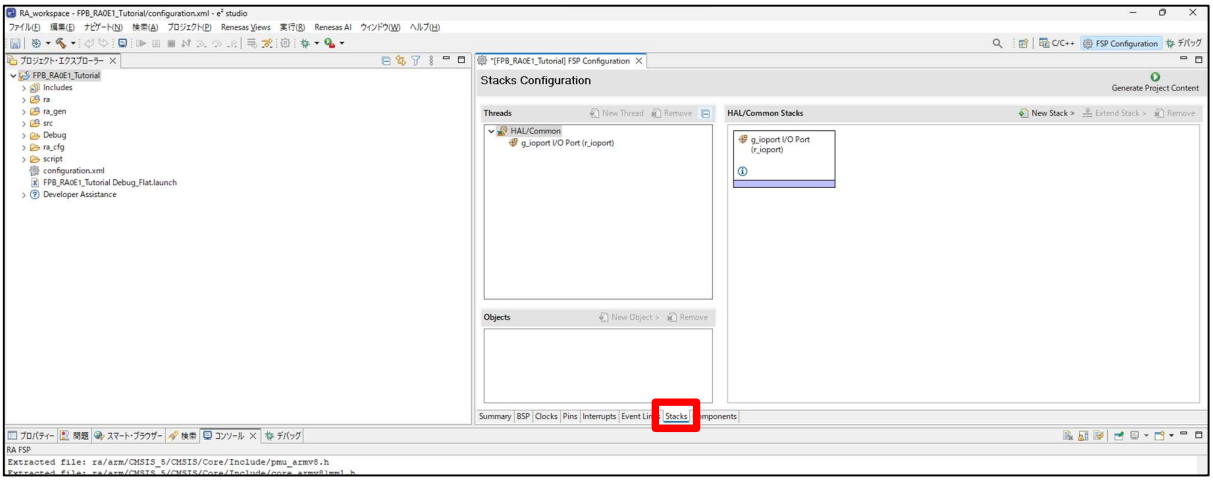


図 3-20　タイマ機能の設定画面

2. 「New Stack」を押すと、機能一覧が表示されますので、「Timers」 >「Timer, Independent Channel, 16-bit and 8-bit Timer Operation (r_tau)」の順に選択します。

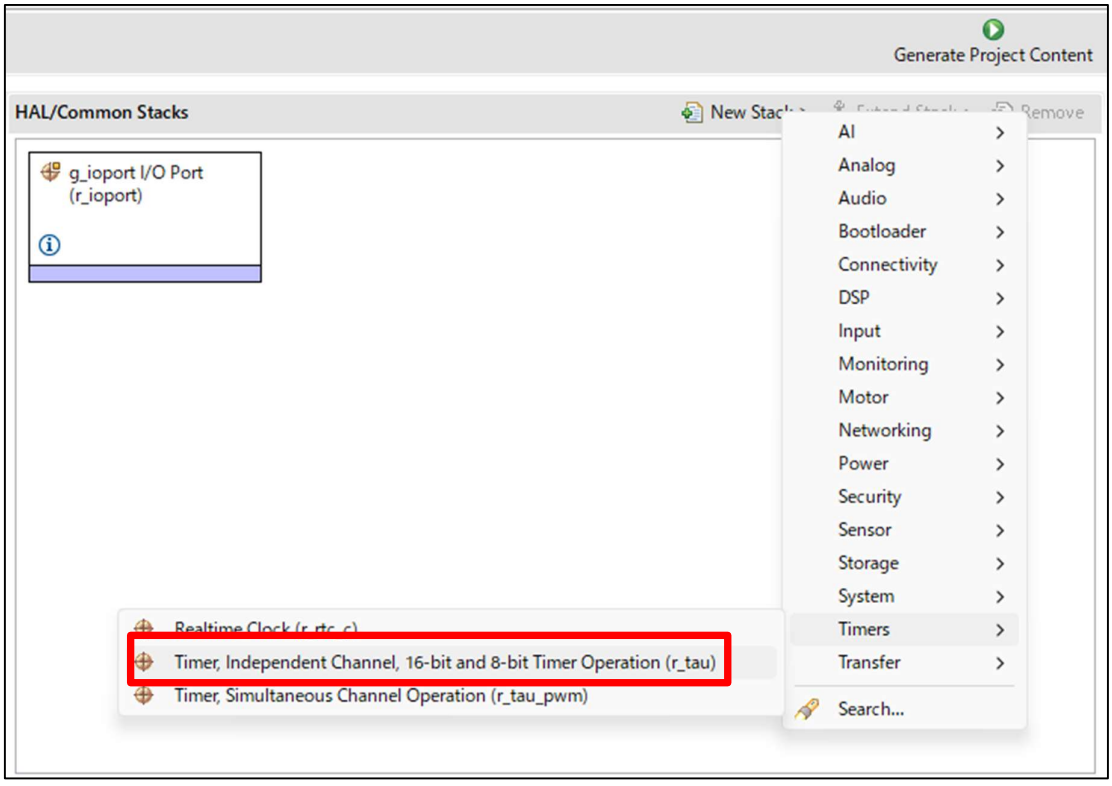


図 3-21　タイマ機能の設定画面

3. g_timer0 という名前のスタックが追加されていることを確認します。
「プロパティー」ウインドウを表示した状態で g_timer0 のブロックをクリックすると、タイマの詳細な設定項目を表示します。

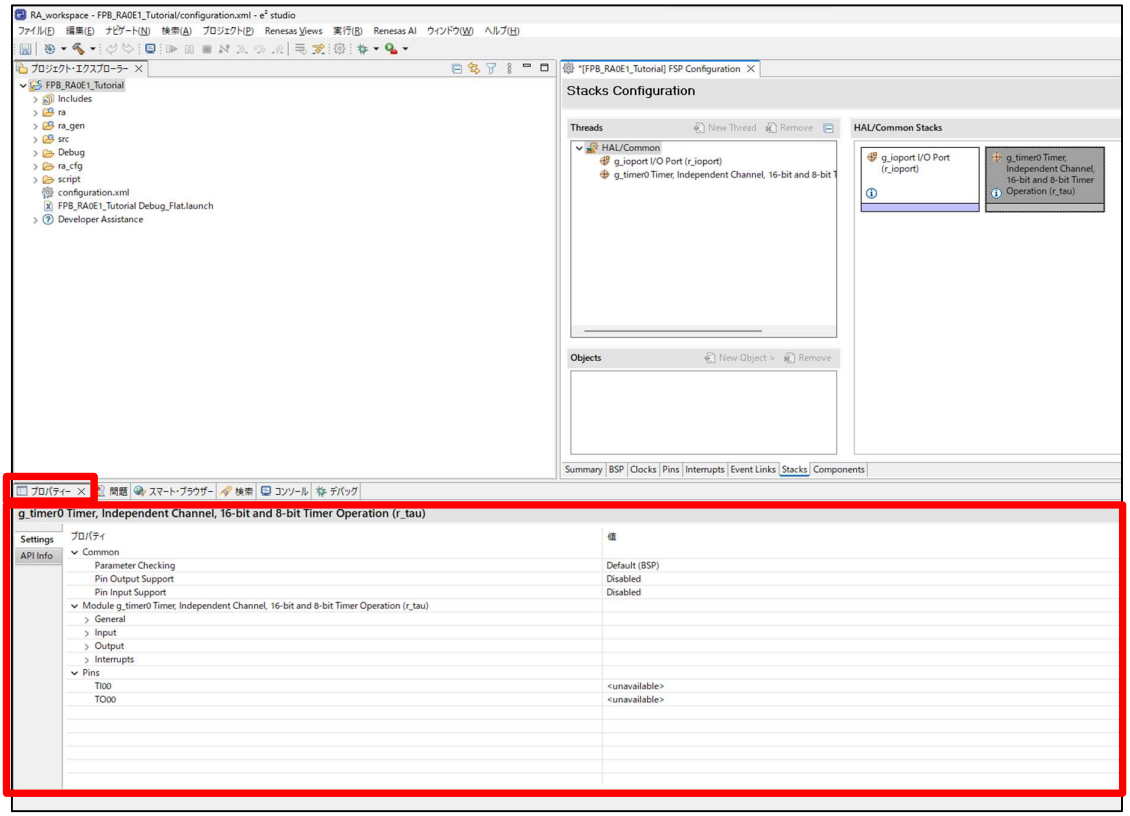


図 3-22　タイマ機能の設定画面

「プロパティー」ウインドウが見つからない場合は、e² studio のメニューバーから「ウィンドウ」→「ビューの表示」→「プロパティー」を選択すると表示します。

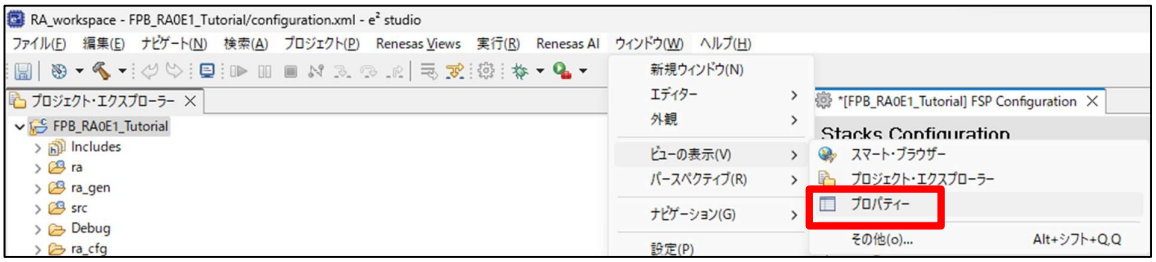


図 3-23　タイマ機能の設定画面

4. General のリストを表示します。この画面で基本的なタイマの設定を行います。

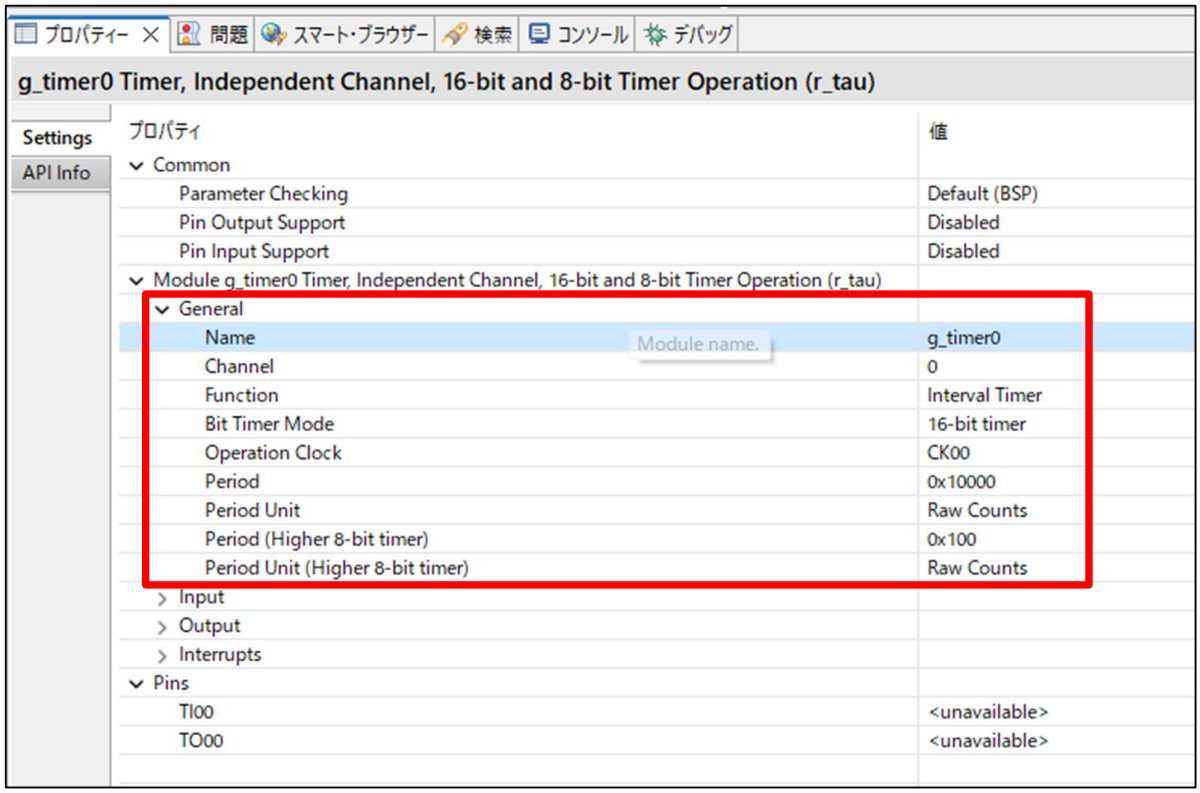


図 3-24　タイマ機能の設定画面

5. はじめに、タイマモジュールの名前を設定します。
   Name：タイマモジュールの名前を設定します。今回は 「MyTimer」という名前で作成します。

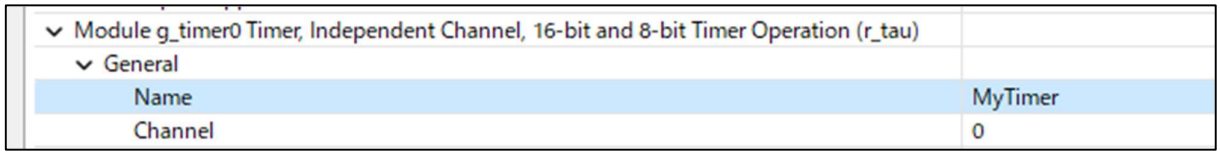


図 3-25　タイマ機能の設定画面

6. 続けて下記の項目を設定します。
   Channel : 使用する TAU のチャンネル番号
   今回はチャンネル 0 を使用します。「**0**」を設定します。
   Function : 使用する TAU チャンネルの機能
   今回はインターバルタイマを使用します。「**Interval Timer**」を選択します。
   Bit Timer: 使用する TAU チャンネルのカウンタビット幅
   今回は 16 ビット幅を使用します。「**16-bit timer**」を選択します。
   Operation Clock: 使用する TAU チャンネルに供給するクロックを設定
   今回は CK00 を使用します。「**CK00**」を選択します。
   Period / Period Unit : 16bit タイマを使用する際の周期を設定します。
   今回は 500ms 周期を作るため
   Period = **500**
   Period Unit = **Milliseconds**
   を設定します。

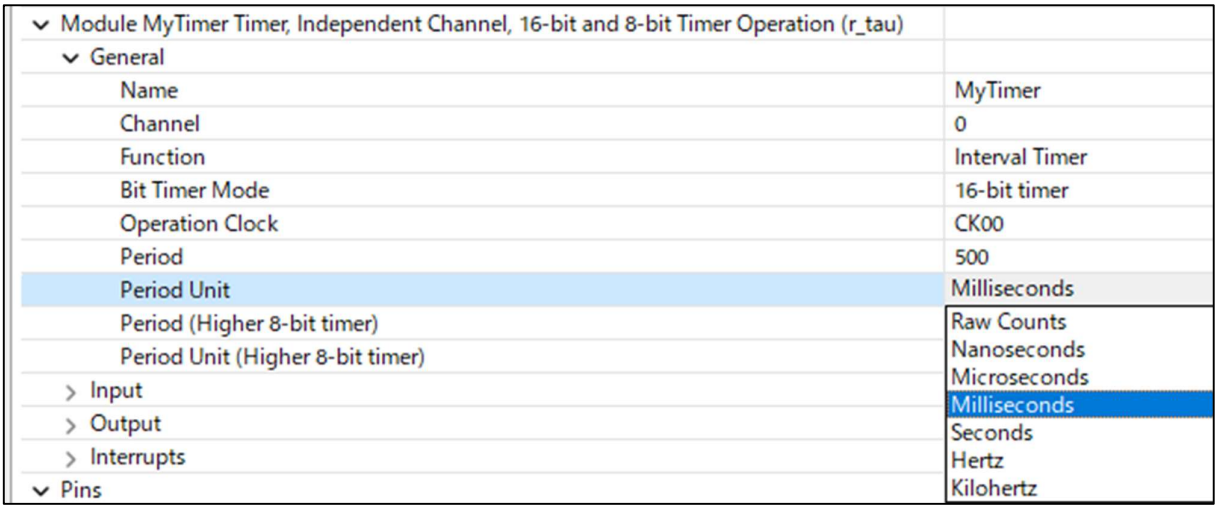


図 3-26　タイマ機能の設定画面

7. 次に割り込みの設定を行います。
   Interrupts のリストを表示します。

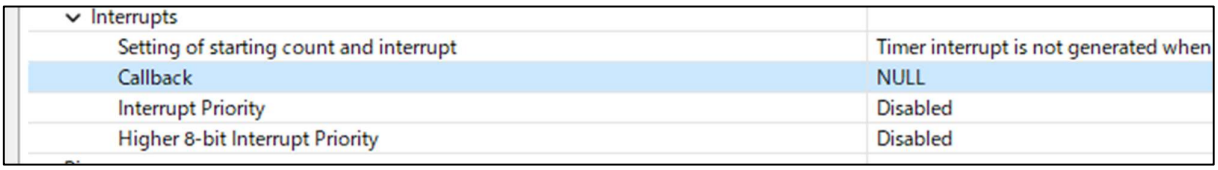


図 3-27　タイマ機能の設定画面

8. Callback にユーザが実装可能な割り込み処理関数を設定します。
   デフォルト設定の「NULL」は実装する関数がないことを意味します。
   今回は「**MyISR**」 という名前のコールバック関数を作成します。

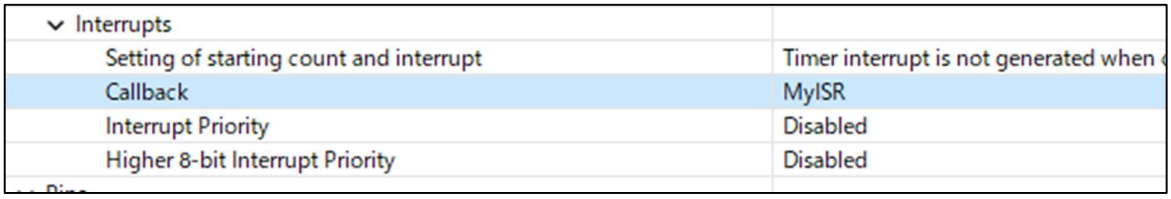


図 3-28　タイマ機能の設定画面

9. Interrupt Priority に割り込み優先度を設定します。
   デフォルトの「Disabled」は割り込み禁止を意味します。
   今回は割り込みを使用しますので、**Priority 0 ~ 3 のいずれか**に設定します。

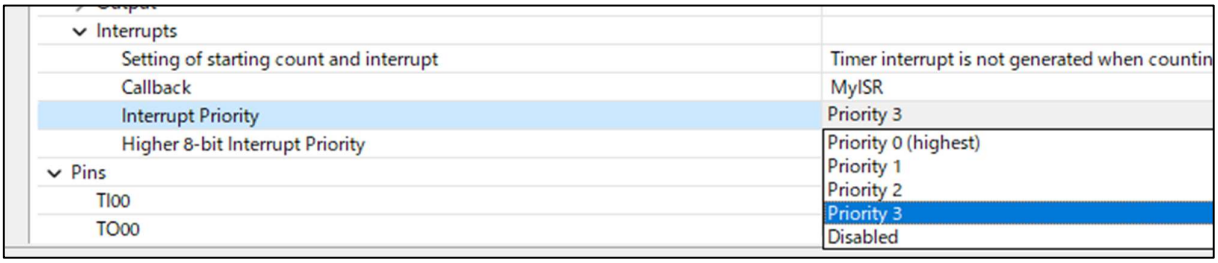


図 3-29　タイマ機能の設定画面

Timer の設定は以上です。

10. 今回のプログラム作成で必要な設定は完了しましたので、「Generate Project Content」ボタンを押してコード生成します。

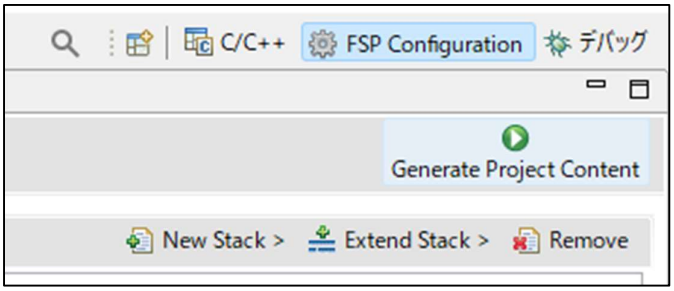


図 3-30　タイマ機能の設定画面

11. Configuration.xml ファイルに保存するかどうかを質問されますので「続行」ボタンを押して保存します。

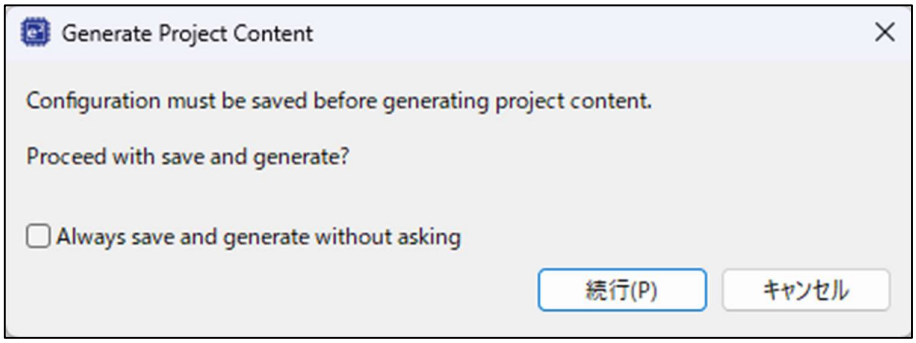


図 3-31 タイマ機能の設定画面

3.3 コーディング

main プログラム、割り込み関数にコードを実装します。

実装する内容は以下の通りです。

・main プログラム

　タイマの起動と開始

・割り込みプログラム

　LED 端子の出力反転処理

3.3.1 main プログラムの実装

1. FSP を使用する場合、ユーザが記述するファイルは プロジェクトフォルダ¥src¥hal_entry.c です。
   hal_entry.c をダブルクリックして、ファイルを開きます。

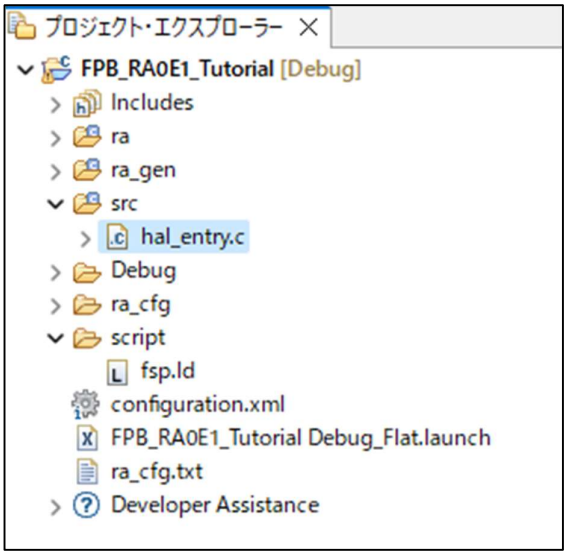


図 3-32　main プログラムの実装画面

2. hal_entry.c の void hal_entry(void)の中にソースコードを記述していきます。
   IO ポートの初期設定は hal_entry 関数が呼び出される前に実行されますので、記述不要です。
   ※画面の灰色で示された部分は、ソースコードが無効であることを示しており、無視してかまいません。

```
void hal_entry(void)
{
    /* TODO: add your own code here */

#if BSP_TZ_SECURE_BUILD
    /* Enter non-secure code */
    R_BSP_NonSecureEnter();
#endif
}
```

図 3-33　main プログラムの実装画面

3. まず、FSP が提供する関数が定義している戻り値の変数を定義します。
```
fsp_err_t err;
```
と記述します。

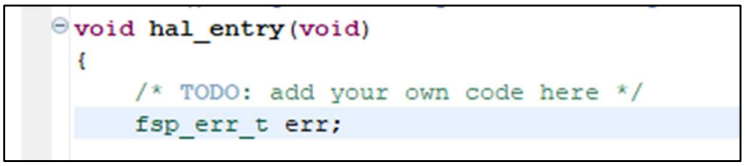


図 3-34　main プログラムの実装画面

4. 次にタイマのオープン関数を実装します。
「プロジェクト・エクスプローラー」ウインドウにある「Developer Assistance」のリストを開くと、「3.2.4 タイマ機能の設定」で設定したタイマモジュール MyTimer が表示されます。
さらにリストを開くと、関数一覧が表示されます。
タイマをオープンする関数は R_TAU_Open()です。マウスでこの関数をソースファイルにドラッグアンドドロップします。

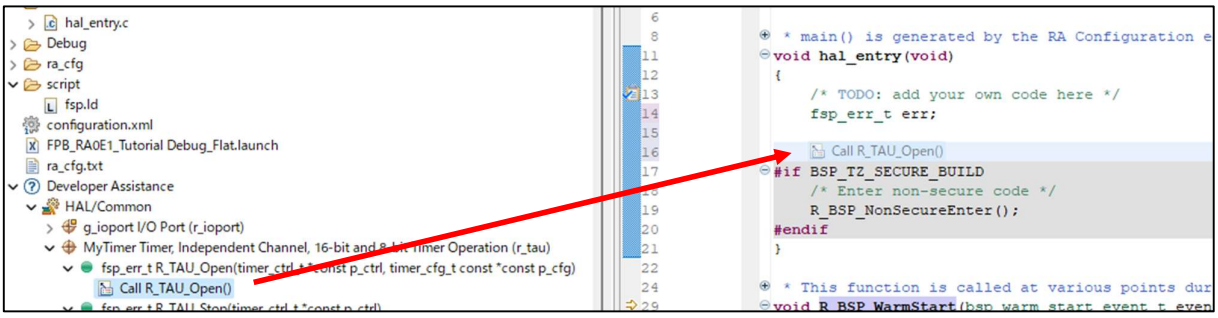


図 3-35　main プログラムの実装画面

5. ソースコードにオープン関数が実装されます。
事前に戻り値変数を宣言しておくことで、代入文を含んだコードを生成します。

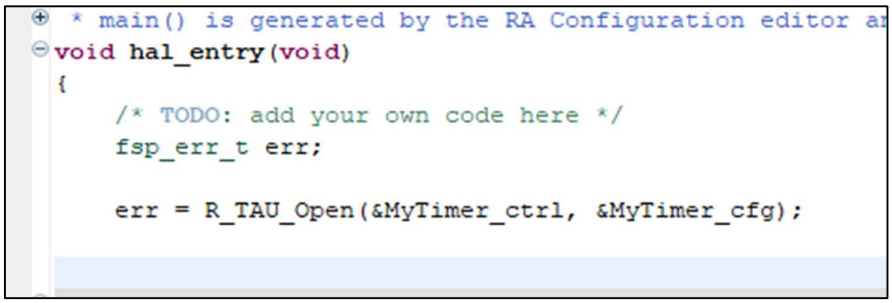


図 3-36　main プログラムの実装画面

6. 続けて、タイマのスタート関数を実装します。
   スタート関数は R_TAU_Start()です。
   同じように、マウスでソースファイル内の R_TAU_Open()関数呼び出しの下にドラッグアンドドロップします。

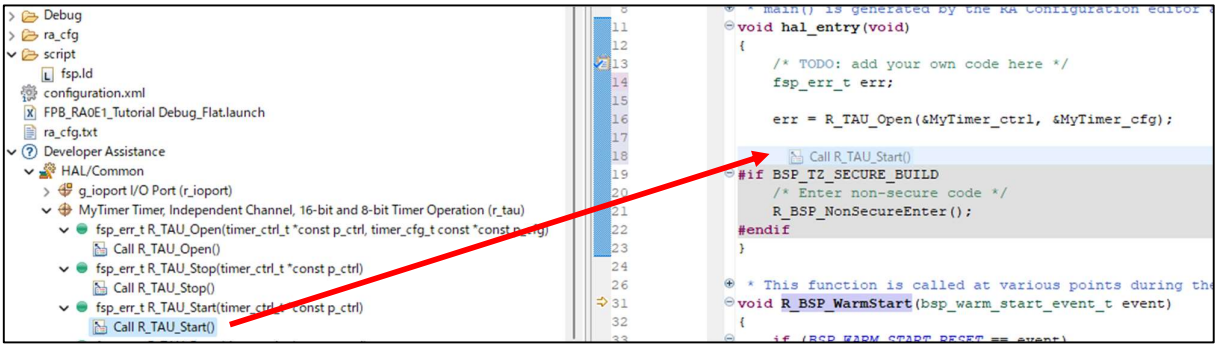


図 3-37　main プログラムの実装画面

7. ソースコードにスタート関数が実装されます。

```
void hal_entry(void)
{
    /* TODO: add your own code here */
    fsp_err_t err;

    err = R_TAU_Open(&MyTimer_ctrl, &MyTimer_cfg);

    err = R_TAU_Start(&MyTimer_ctrl);
#if BSP_TZ_SECURE_BUILD
    /* Enter non-secure code */
    R_BSP_NonSecureEnter();
#endif
}
```

図 3-38　main プログラムの実装画面

8. オープン関数、スタート関数が異常終了を返した場合を検出するために、関数呼び出し後に

```
while(err);
```

を実装しておきます。異常終了した場合はこの文で無限ループします。
また、正常終了の確認と main ループを実装するため

```
while (1)
{
  __NOP();
}
```

を実装します。

hal_entry()関数のソースコードは以下のようになります。

```
 * main() is generated by the RA Configuration editor a
void hal_entry(void)
{
    /* TODO: add your own code here */
    fsp_err_t err;

    err = R_TAU_Open(&MyTimer_ctrl, &MyTimer_cfg);
    while(err);

    err = R_TAU_Start(&MyTimer_ctrl);
    while(err);

    while(1)
    {
        __NOP();
    }
#if BSP_TZ_SECURE_BUILD
    /* Enter non-secure code */
    R_BSP_NonSecureEnter();
#endif
}
```

図 3-39　main プログラムの実装画面

9. 8 までの内容をテキストで掲載します。(灰色部分の無効コードは除いています)

```
void hal_entry(void)
{
    /* TODO: add your own code here */
    fsp_err_t err;

    err = R_TAU_Open(&MyTimer_ctrl, &MyTimer_cfg);
    while(err);

    err = R_TAU_Start(&MyTimer_ctrl);
    while(err);

    while(1)
    {
        __NOP();
    }
}
```

3.3.2　割り込みプログラムの実装

1. 割り込みプログラムを記述します。記述するファイルは引き続き hal_entry.c とします。
   Developer Assistance のリストからタイマモジュールの「Callback function definition」をソースファイルの最終行にドラッグアンドドロップします。

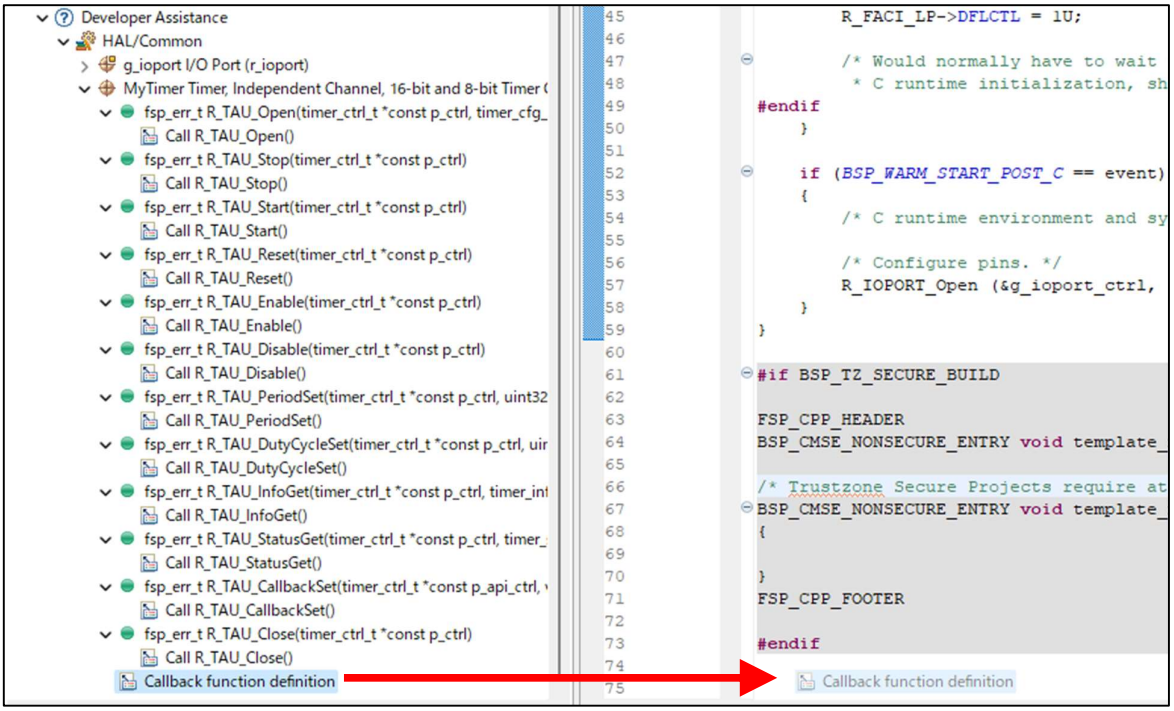


図 3-40　割り込みプログラムの実装画面

2. FSP で設定したコールバック関数名が現れます。
   この関数は「3.2.4 タイマ機能の設定」で設定した通り、500ms ごとに呼び出されます。
   この関数の中に LED 反転出力処理を追加していきます。

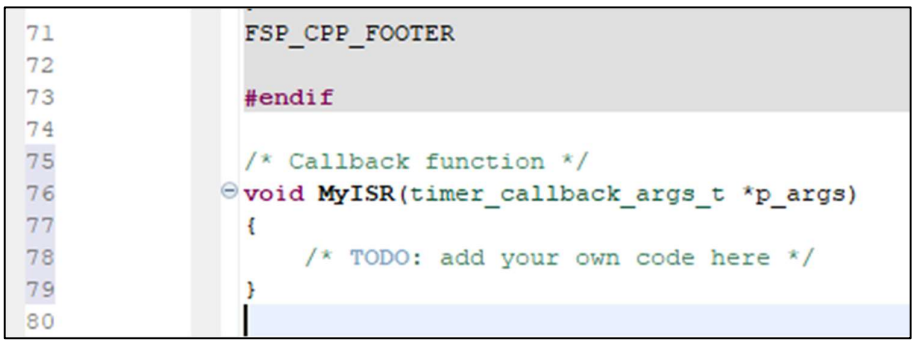


図 3-41　割り込みプログラムの実装画面

3. FSP の戻り値型の変数を記述します。
   `fsp_err_t status;`

```
/* Callback function */
void MyISR(timer_callback_args_t *p_args)
{
    /* TODO: add your own code here */
    fsp_err_t status;

}
```

図 3-42 割り込みプログラムの実装画面

4. LED のピンの出力状態を読み出します。
   ピン読み出しの関数は R_IOPORT_PinRead()です。

   Developer Assistance のリストから IO ポートモジュールの R_IOPORT_PinRead()を割り込み関数内にドラッグアンドドロップします。

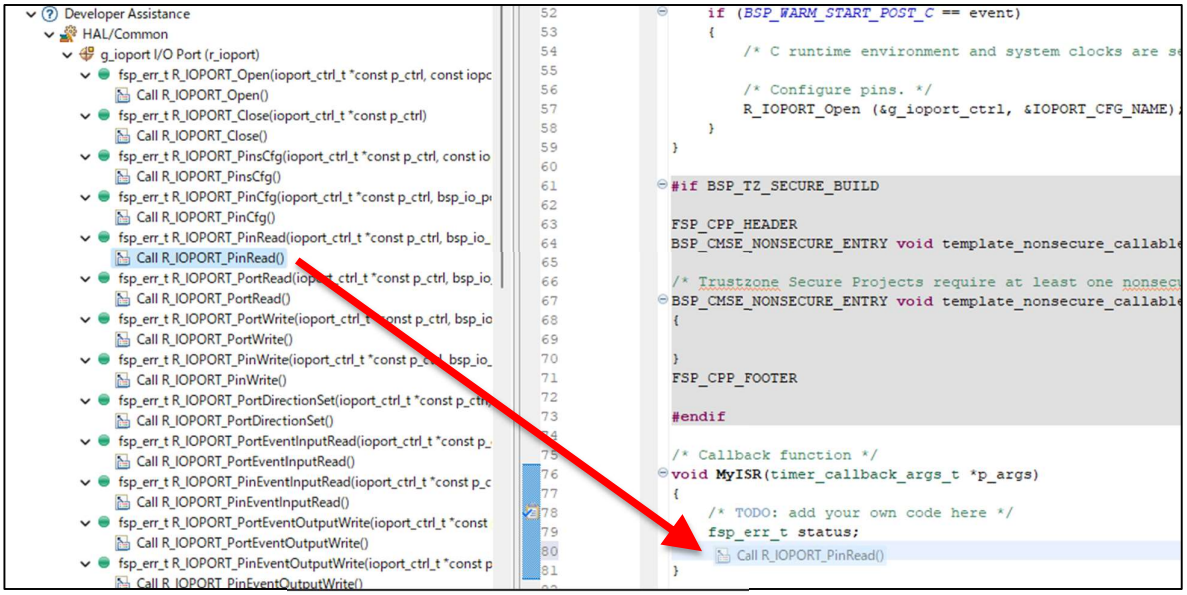


図 3-43 割り込みプログラムの実装画面

5. ピン読み出しの関数が実装されます。

```
/* Callback function */
void MyISR(timer_callback_args_t *p_args)
{
    /* TODO: add your own code here */
    fsp_err_t status;
    status = R_IOPORT_PinRead(&g_ioport_ctrl, pin, p_pin_value);

}
```

図 3-44 割り込みプログラムの実装画面

6. R_IOPORT_PinRead()の行の上に
```
bsp_io_level_t value;
```
を記述します。

7. R_IOPORT_PinRead()の第 2 引数には読み出すピン番号 = “LED1”、
第 3 引数には読み出した結果を格納する変数(bsp_io_level_t value)のアドレスを設定します。
```
status = R_IOPORT_PinRead(&g_ioport_ctrl, LED1, &value);
```
と記述します。

```
63  }
64  FSP_CPP_FOOTER
65
66  #endif
67
68  /* Callback function */
69  void MyISR(timer_callback_args_t *p_args)
70  {
71      /* TODO: add your own code here */
72      fsp_err_t status;
73
74      bsp_io_level_t value;
75      status = R_IOPORT_PinRead(&g_ioport_ctrl, LED1, &value);
76
```

図 3-45　割り込みプログラムの実装画面

8. “LED1”というシンボルは、プロジェクトツリーの ra_cfg¥fsp_cfg¥bsp¥bsp_pin_cfg.h に宣言されており、hal_entry.c から参照可能です。

```
9   #define LED1 (BSP_IO_PORT_00_PIN_08) /* GREEN_ARDUINO_A4 */
10  #define LED2 (BSP_IO_PORT_00_PIN_09) /* GREEN_PMOD1_CS3_GPIO10_ARDUINO_A3 */
```

図 3-46　割り込みプログラムの実装画面

9. 読み出した値を反転して出力する値を変数 next_led1 に格納します。

```
bsp_io_level_t next_led1;
```

を宣言し、読み出した値を反転して出力する処理を記述します。

```
while(status);
if(BSP_IO_LEVEL_HIGH == value)
{
   next_led1 = BSP_IO_LEVEL_LOW;
}
else
{
   next_led1 = BSP_IO_LEVEL_HIGH;
}
```

```
void MyISR(timer_callback_args_t *p_args)
{
    /* TODO: add your own code here */
    fsp_err_t status;

    bsp_io_level_t value;
    bsp_io_level_t next_led1;
    status = R_IOPORT_PinRead(&g_ioport_ctrl, LED1, &value);
    while(status);
    if(BSP_IO_LEVEL_HIGH == value)
    {
        next_led1 = BSP_IO_LEVEL_LOW;
    }
    else
    {
        next_led1 = BSP_IO_LEVEL_HIGH;
    }

}
```

図 3-47　割り込みプログラムの実装画面

10. LED1 にピン出力する処理を追加します。
ピン書き込みの関数は R_IOPORT_PinWrite()です。

IO ポートモジュールの R_IOPORT_PinWrite()を割り込み関数内にドラッグアンドドロップします。

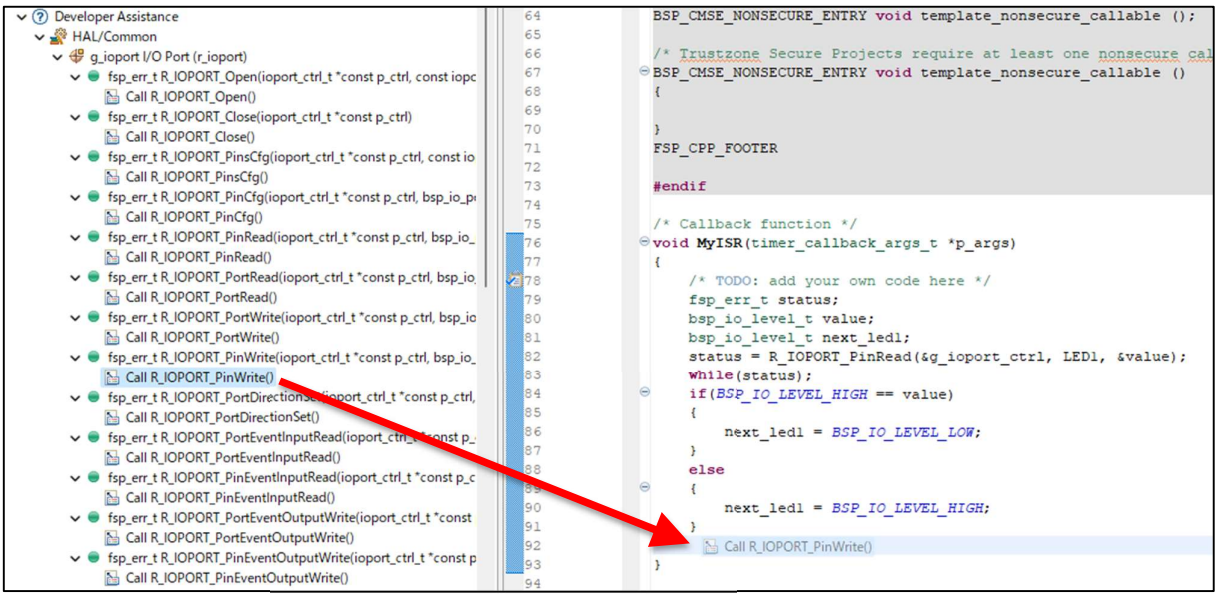


図 3-48　割り込みプログラムの実装画面

11. ピン書き込みの関数が実装されます。

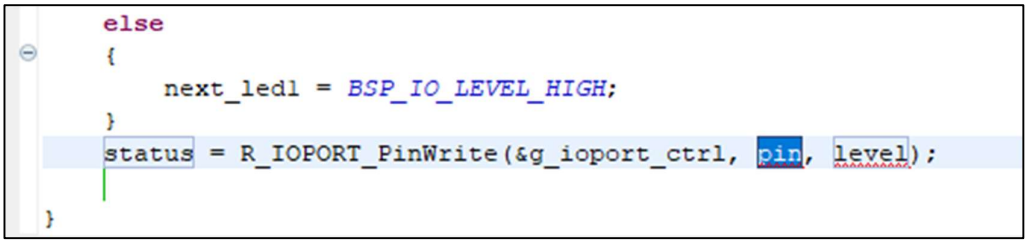


図 3-49　割り込みプログラムの実装画面

12. R_IOPORT_PinWrite()の第 2 引数に”LED1”、第 3 引数に出力値(=next_led1)を設定します。

```
status = R_IOPORT_PinWrite(&g_ioport_ctrl, LED1, next_led1);
while(status);
```

と実装します

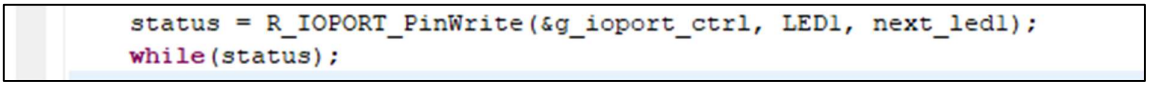
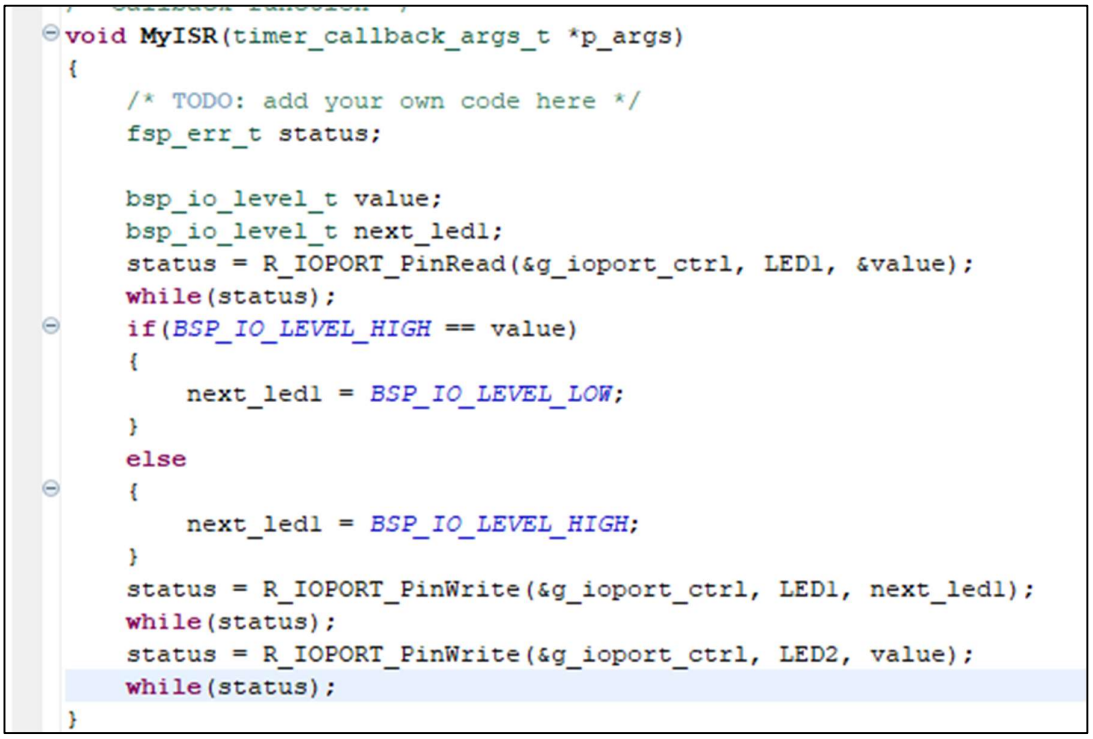


図 3-50　割り込みプログラムの実装画面

13. 同様に LED2 に対して前回の LED1 の値を出力する処理を以下のように記述します。

```
status = R_IOPORT_PinWrite(&g_ioport_ctrl, LED2, value);
```

14. 割り込み関数の実装は以下のようになります。



```
void MyISR(timer_callback_args_t *p_args)
{
    /* TODO: add your own code here */
    fsp_err_t status;

    bsp_io_level_t value;
    bsp_io_level_t next_led1;
    status = R_IOPORT_PinRead(&g_ioport_ctrl, LED1, &value);
    while(status);
    if(BSP_IO_LEVEL_HIGH == value)
    {
        next_led1 = BSP_IO_LEVEL_LOW;
    }
    else
    {
        next_led1 = BSP_IO_LEVEL_HIGH;
    }
    status = R_IOPORT_PinWrite(&g_ioport_ctrl, LED1, next_led1);
    while(status);
    status = R_IOPORT_PinWrite(&g_ioport_ctrl, LED2, value);
    while(status);
}
```

図 3-51　割り込みプログラムの実装画面

15. 14 までの内容をテキストで掲載します。

```
/* Callback function */
void MyISR(timer_callback_args_t *p_args)
{
    /* TODO: add your own code here */
    fsp_err_t status;
    bsp_io_level_t value;
    bsp_io_level_t next_led1;
    status = R_IOPORT_PinRead(&g_ioport_ctrl, LED1, &value);
    while(status);
    if(BSP_IO_LEVEL_HIGH == value)
    {
        next_led1 = BSP_IO_LEVEL_LOW;
    }
    else
    {
        next_led1 = BSP_IO_LEVEL_HIGH;
    }
    status = R_IOPORT_PinWrite(&g_ioport_ctrl, LED1, next_led1);
    while(status);
    status = R_IOPORT_PinWrite(&g_ioport_ctrl, LED2, value);
    while(status);
}
```

3.4 オプション設定メモリ

オプション設定メモリは、低電圧検出回路の設定など、MCU のリセット後の状態を設定するメモリです。e2studio では、ツール上から設定が可能です。本書のプログラムではデフォルト設定のままで設定変更の必要はないのですが、設定箇所を説明します。なお、オプション設定メモリの設定値は参考資料[1]の 6 章オプション設定メモリをご参照ください。

1. 「プロパティー」ウインドウを選択した状態で FSP Configurator 画面にある BSP タブを選択すると、プロパティーに詳細な設定項目が表示されます。

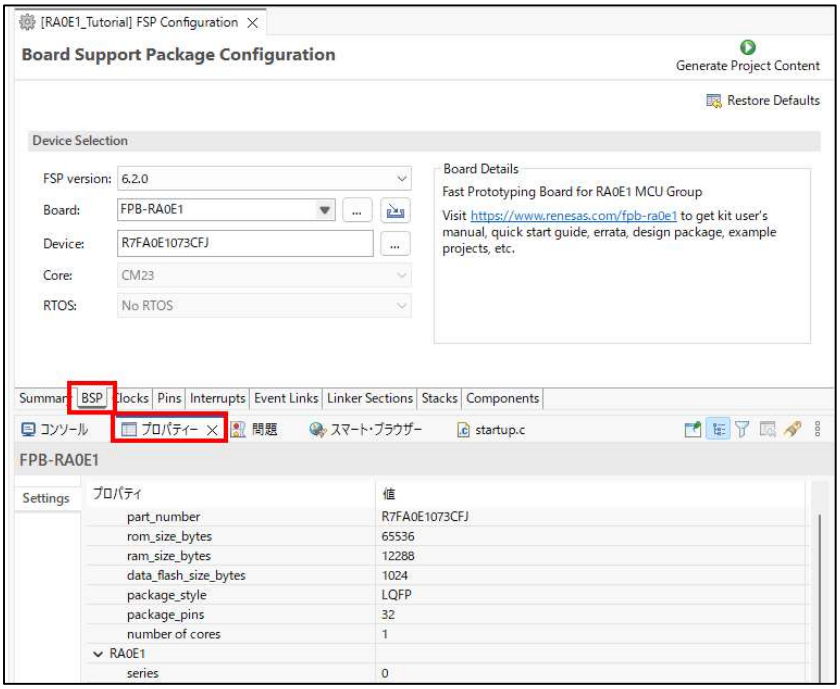


図 3-52　オプション設定メモリ

2. プロパティー内にある RA0E1 Device Options からオプション設定メモリを設定できます。

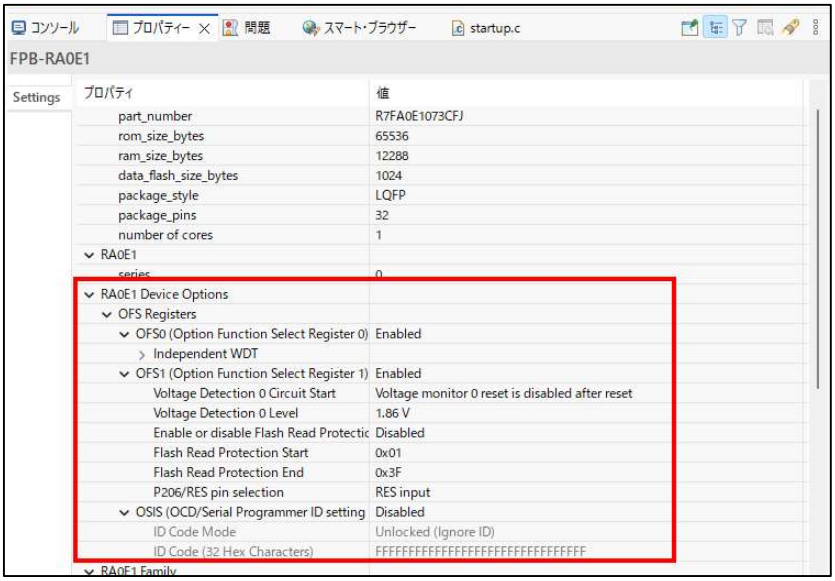


図 3-53　オプション設定メモリ

3.5 ビルド

コーディング完了後に、プロジェクトをビルドします。

1. プロジェクトツリーのプロジェクト名を右クリックした後、「プロジェクトのビルド」を選択してください。

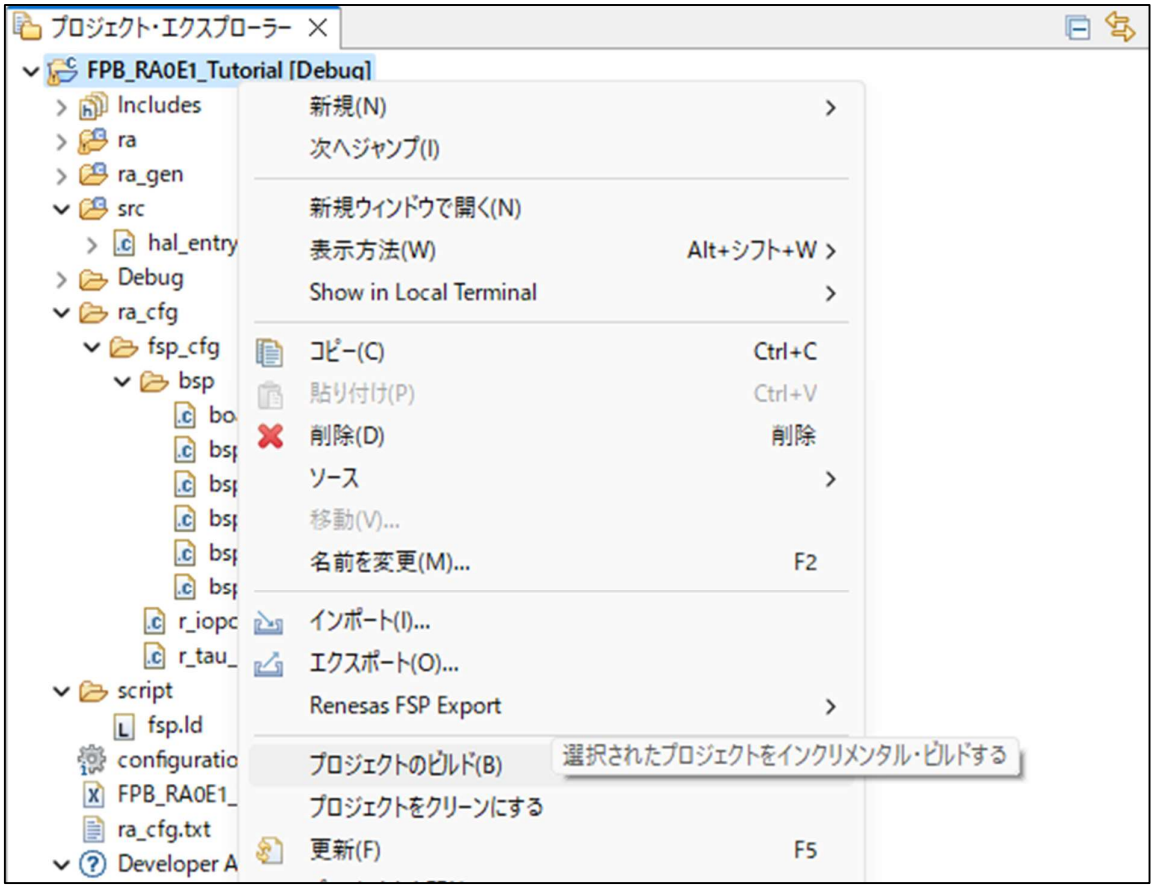


図 3-54 ビルド画面

または、以下のアイコンをクリックすることでもビルド可能です。

図 3-55 ビルド画面

2. 「コンソール」ウインドウにビルドログが出力されます。
最後に “Build Finished. 0 errors, “と表示されていればビルドが正常終了したことを意味します。
黄帯はコンパイラがワーニングを検出して表示しています。ワーニングの内容を確認して必要に応じて修正してください。

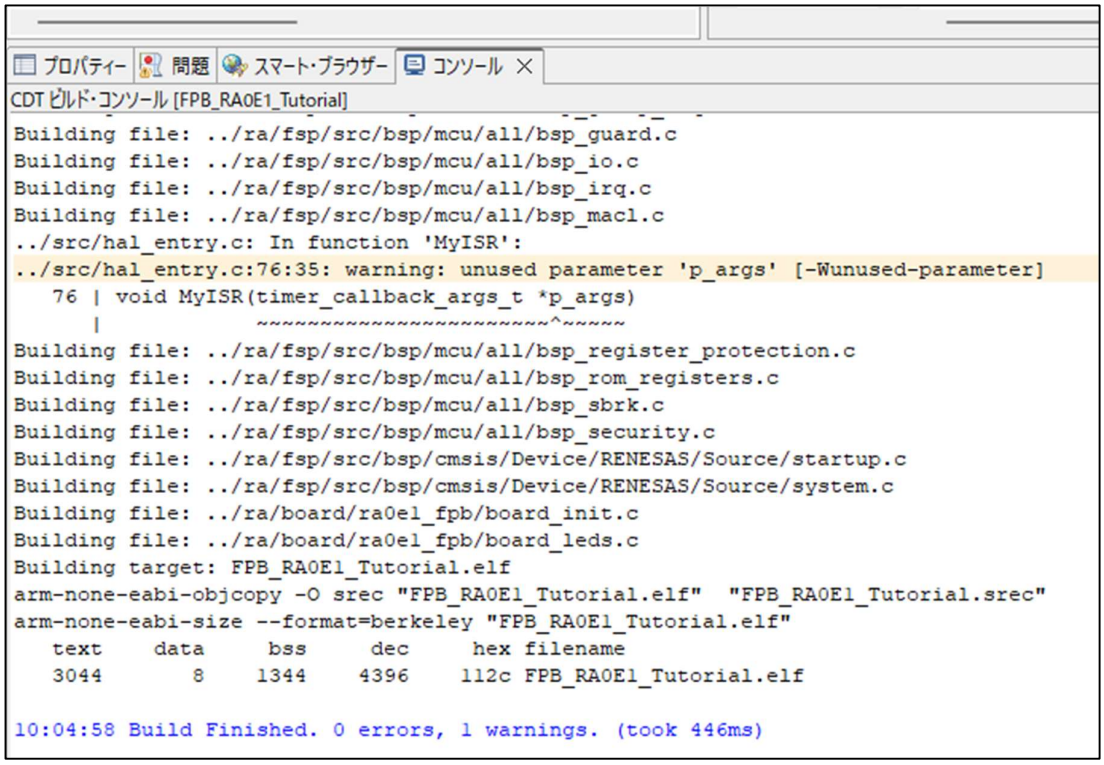


図 3-56 ビルド画面

何らかのコーディングミスがあった場合、赤帯が表示されます。該当行を確認してソースコードを見直してください。

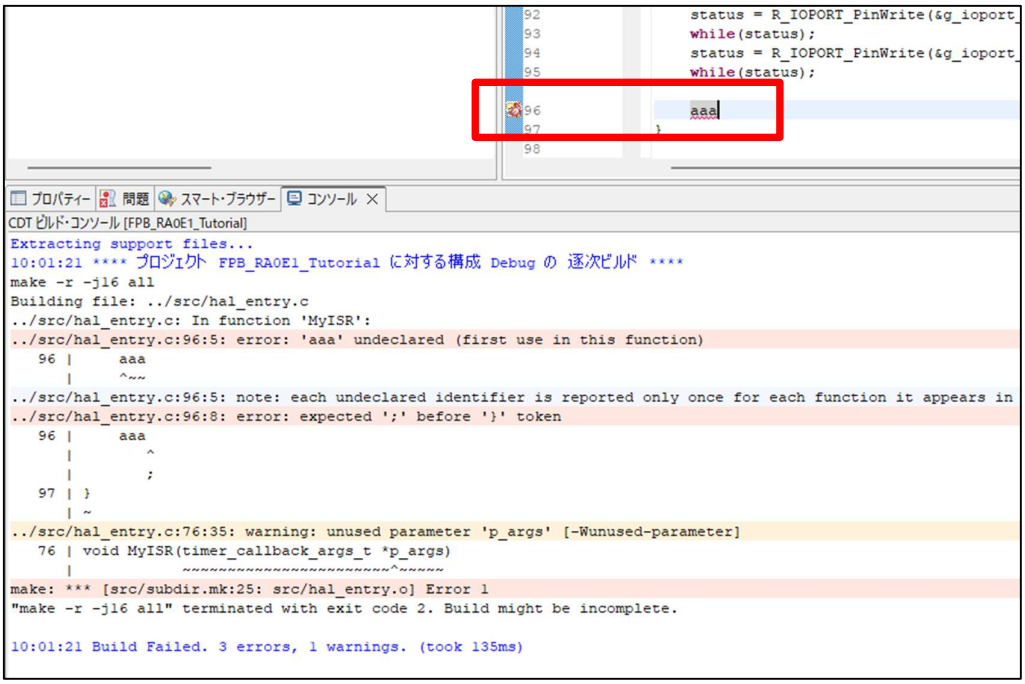


図 3-57 ビルド画面

4. デバッグ方法

この章ではプログラム書き込みに必要な設定、プログラムの実行方法を説明します。

4.1 デバッグ設定と起動

ビルドが完了した後にプログラムをボード上の MCU に書き込みます。

初回のみ、書き込みのための設定を確認します。

PC と FPB ボードを USB ケーブルで接続してください。

1. プロジェクトのプロパティから、「デバッグ」→「デバッグの構成」を選択します。

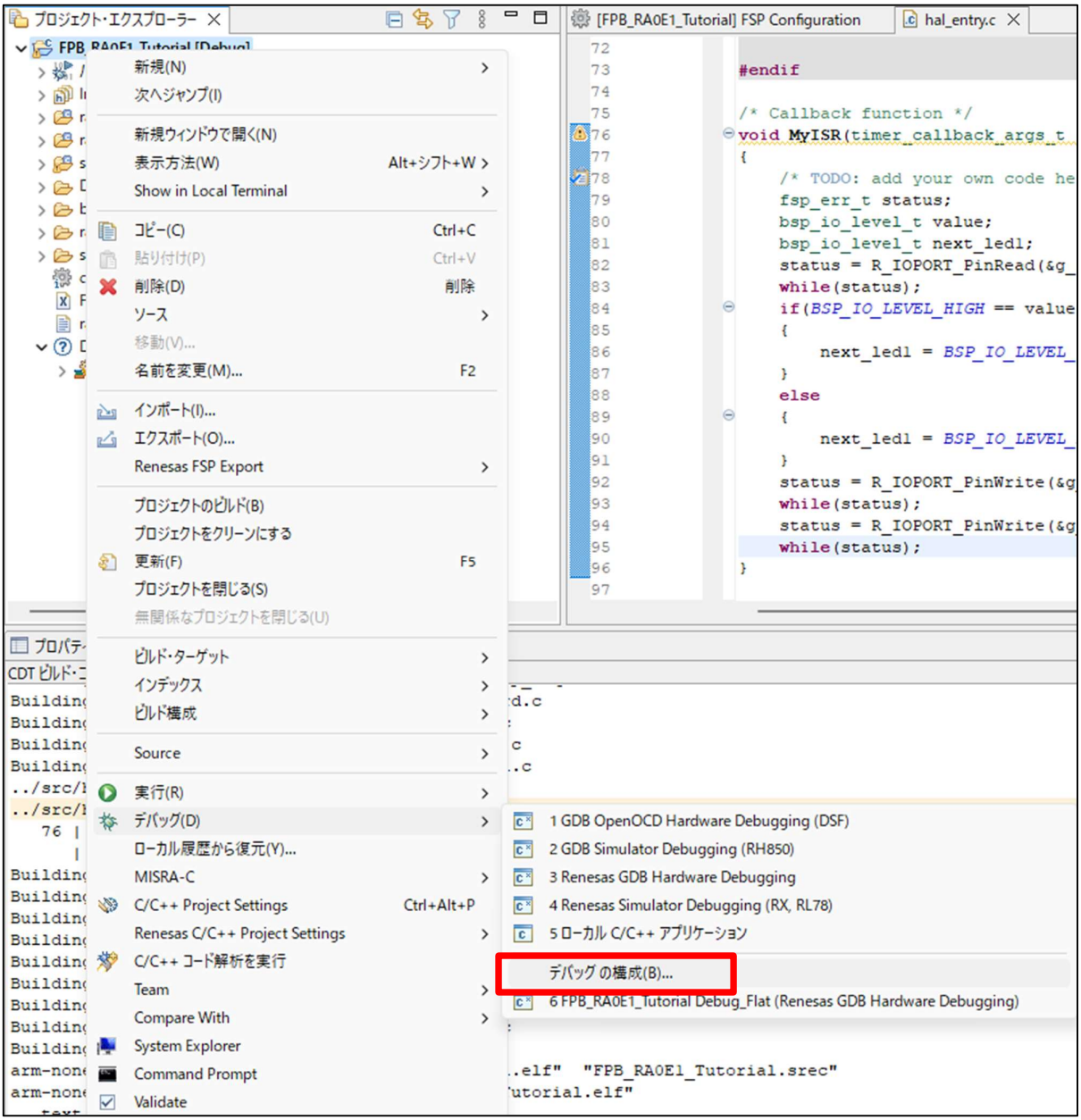


図 4-1　デバッグ設定画面

2. 構成画面が表示されますので、左側のツリーの中から Renesas GDB Hardware Debugging の下にある、プロジェクト名を選択します。

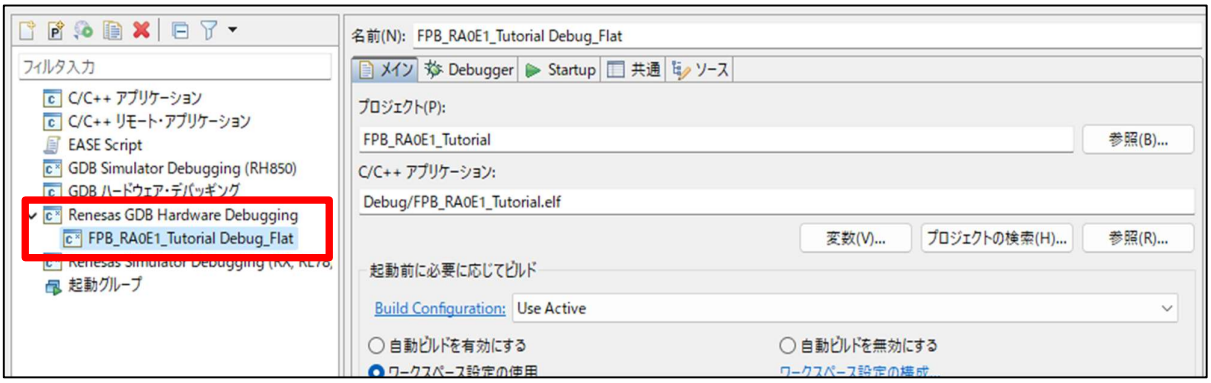


図 4-2　デバッグ設定画面

3. 次に、右側の画面で Debugger タブを選択し、
Debug hardware = “J-link ARM”
Taget Device = R7FA0E107
であることを確認してください。
「デバッグ」ボタンを押します。マイコンにプログラムの書き込みを開始します。

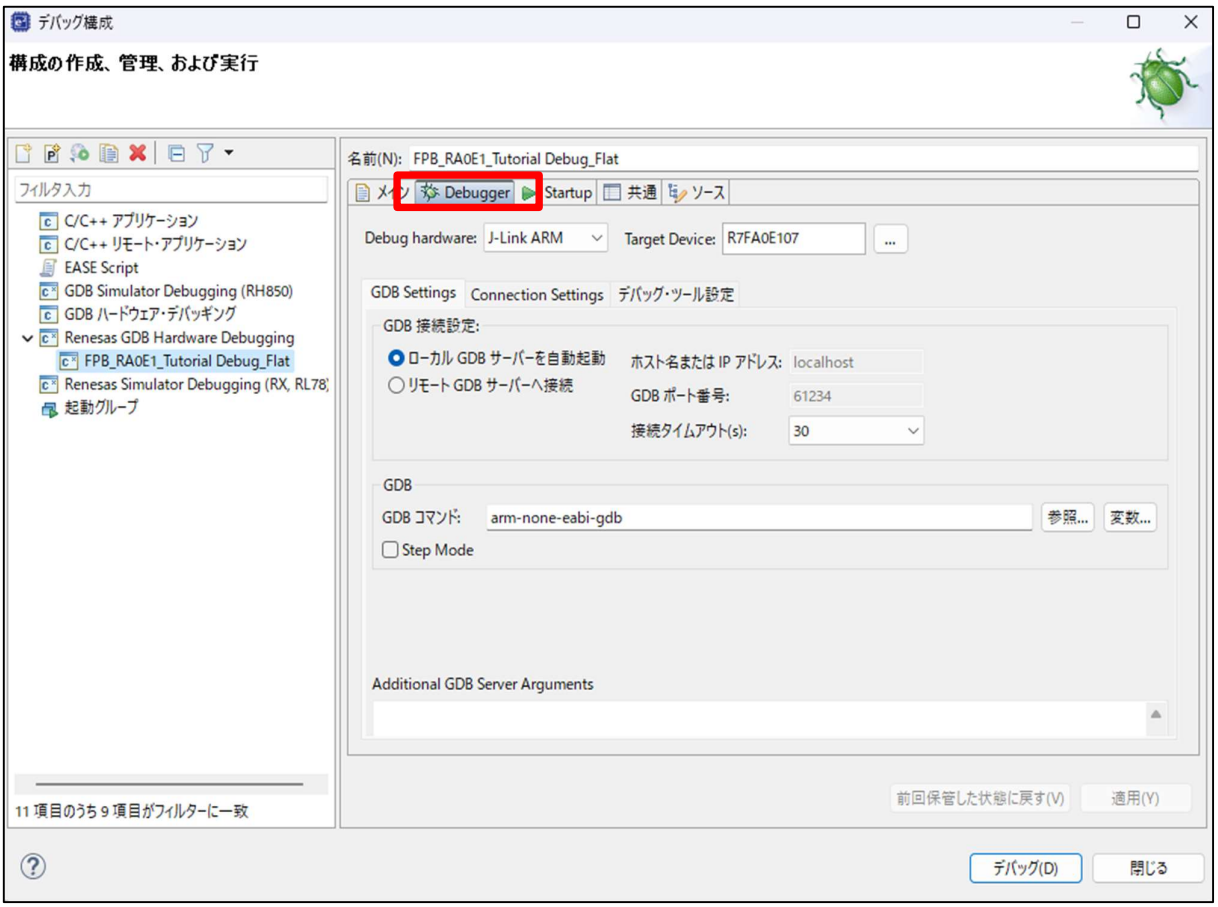


図 4-3　デバッグ設定画面

4. 書き込みが終わった後、画面切り替えのポップアップが表示されるので、「切り替え」ボタンを押します。e² studio のレイアウトがデバッグに適した画面配置になります。

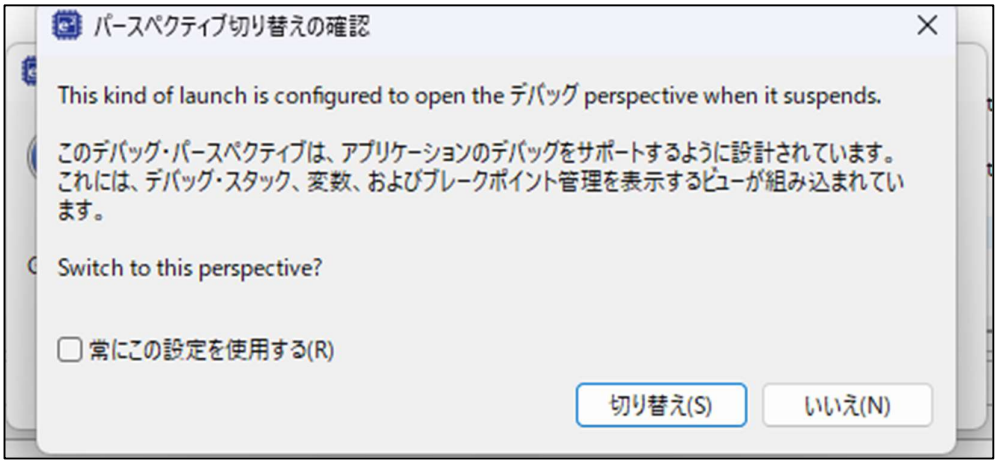


図 4-4　デバッグ設定画面

※「いいえ」を押した場合でも 後から e² studio の右上にある「デバッグ」ボタンを押すことで、画面切り替えができます。

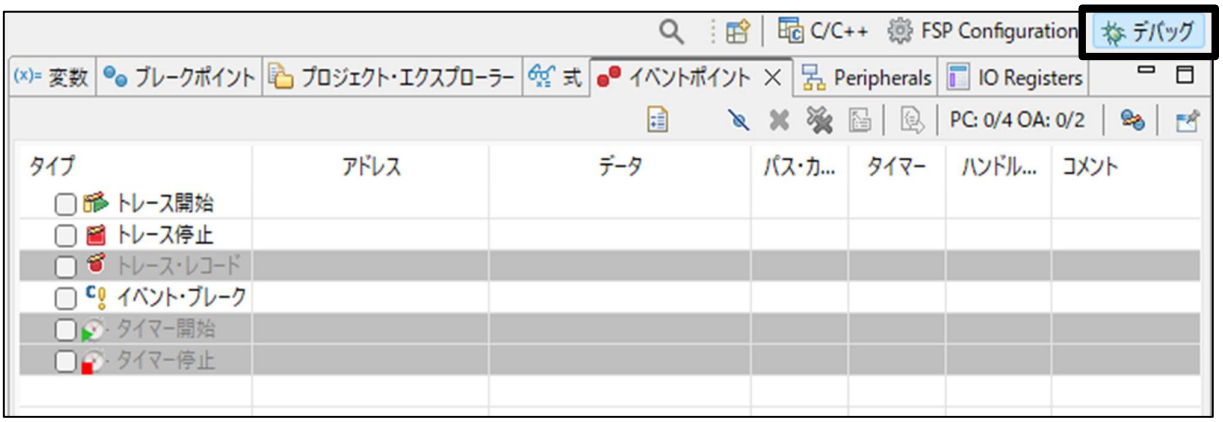


図 4-5　デバッグ設定画面

5. プログラムの書き込みが完了すると、startup.c の SystemInit()関数で一時停止します。

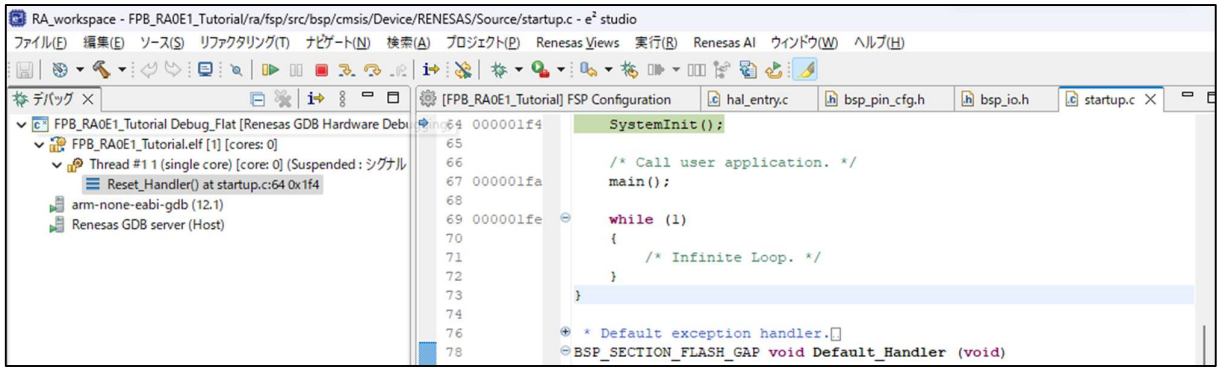


図 4-6　デバッグ設定画面

4.2 実行

この状態は、MCU がリセットした状態で、プログラムはまだ実行していません。

ボード上の LED1、LED2 はどちらも点灯していないことを確認できます。

1. 赤枠の再開ボタン、またはキーボードの F8 キーを押すとプログラムを実行します。

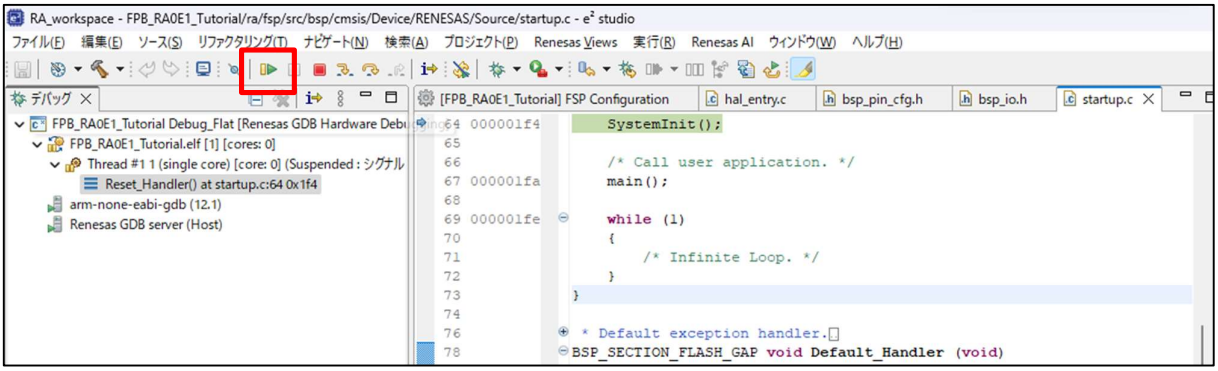


図 4-7　実行画面

プログラムが main 関数の先頭で一時停止します。(e² studio プロジェクトのデフォルトが main 関数で一時停止するよう設定されているためです。)

この状態は、SystemInit()が完了して、初期設定が完了した状態です。
ボードを確認すると、LED1 が点灯していることが確認できます。

2. 再度、再開ボタンを押して、プログラムを続行してください。

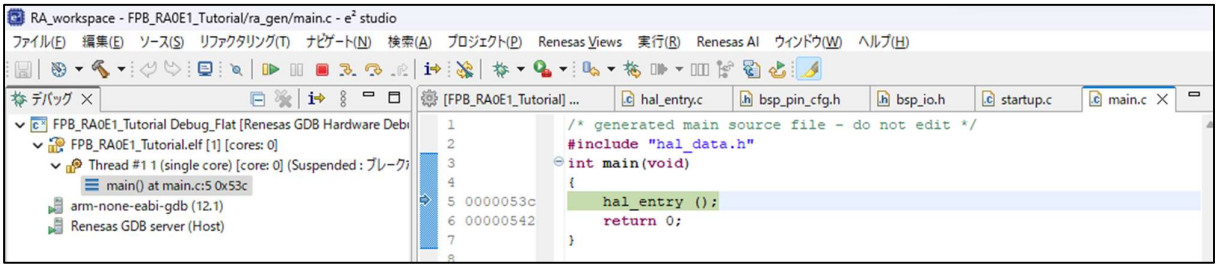


図 4-8　実行画面

e² studio の左下に実行ステータスが表示されます。”実行中”と表示しているときは、プログラムが動作している状態になります。

3. ボードを確認すると、LED1 と LED2 が 500ms ごとに交互に点灯していることを確認できます。

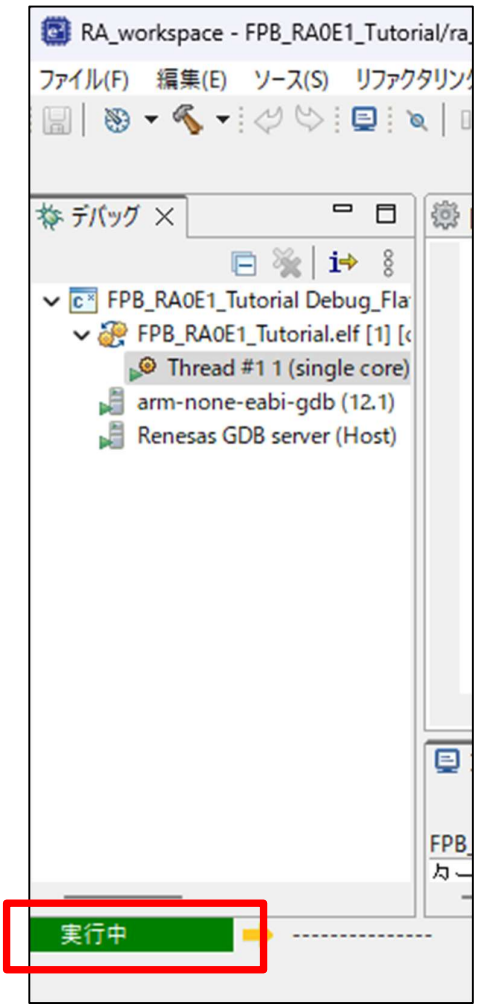


図 4-9 実行画面

4.3　デバッグの終了と再起動

1. デバッグを終了したい場合は、「終了」ボタンを押します。

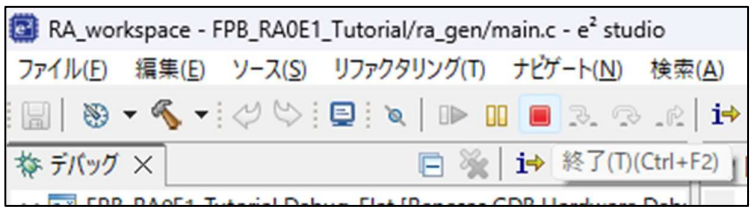


図 4-10　デバッグの終了画面

2. デバッグ終了時、パースペクティブが”デバッグ”の状態になっているため、「プロジェクト・エクスプローラー」ウインドウが所定の位置に表示されていません。
   所定の位置に戻したい場合は e² studio の右上にある パースペクティブ設定を “C/C++”に設定してください。

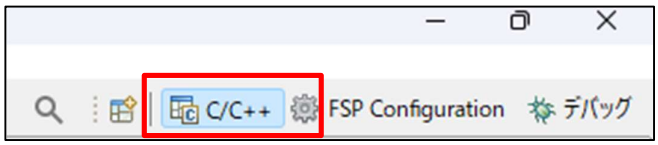


図 4-11　デバッグの終了画面

3. 再度デバッグする場合、デバッグ設定は記憶されています。
   プロジェクトのプロパティから「デバッグ」→ 「6 “プロジェクト名”」を選択して、デバッグ開始することが可能です。

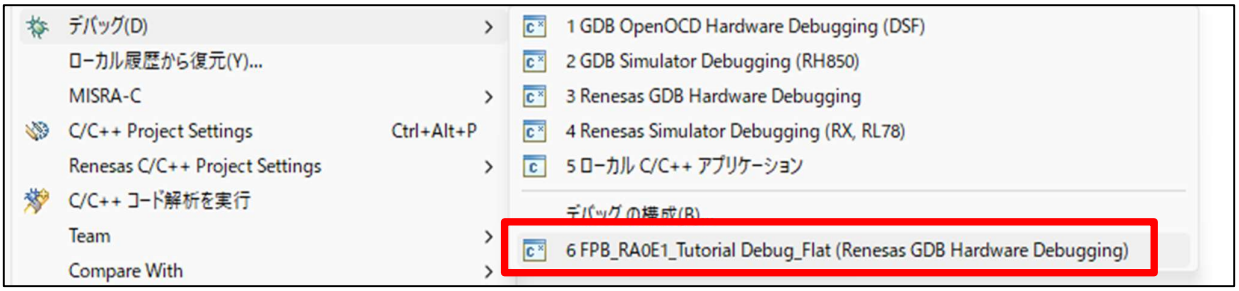


図 4-12　デバッグの再起動画面

または、以下のアイコンをクリックすることでもデバッグ開始可能です。

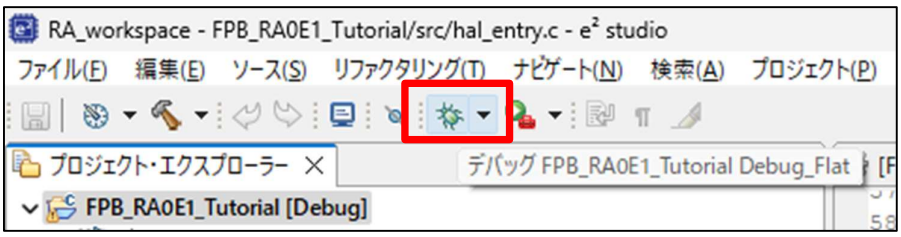


図 4-13　デバッグの再起動画面

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2026.2.17	-	初版発行

製品ご使用上の注意事項

ここでは、マイコン製品全体に適用する「使用上の注意事項」について説明します。個別の使用上の注意事項については、本ドキュメントおよびテクニカルアップデートを参照してください。

1. 静電気対策
   CMOS 製品の取り扱いの際は静電気防止を心がけてください。CMOS 製品は強い静電気によってゲート絶縁破壊を生じることがあります。運搬や保存の際には、当社が出荷梱包に使用している導電性のトレーやマガジンケース、導電性の緩衝材、金属ケースなどを利用し、組み立て工程にはアースを施してください。プラスチック板上に放置したり、端子を触ったりしないでください。また、CMOS 製品を実装したボードについても同様の扱いをしてください。
2. 電源投入時の処置
   電源投入時は、製品の状態は不定です。電源投入時には、LSI の内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。
3. 電源オフ時における入力信号
   当該製品の電源がオフ状態のときに、入力信号や入出力プルアップ電源を入れないでください。入力信号や入出力プルアップ電源からの電流注入により、誤動作を引き起こしたり、異常電流が流れ内部素子を劣化させたりする場合があります。資料中に「電源オフ時における入力信号」についての記載のある製品は、その内容を守ってください。
4. 未使用端子の処理
   未使用端子は、「未使用端子の処理」に従って処理してください。CMOS 製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI 周辺のノイズが印加され、LSI 内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。
5. クロックについて
   リセット時は、クロックが安定した後、リセットを解除してください。プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。
6. 入力端子の印加波形
   入力ノイズや反射波による波形歪みは誤動作の原因になりますので注意してください。CMOS 製品の入力がノイズなどに起因して、V_{IL}（Max.）から V_{IH}（Min.）までの領域にとどまるような場合は、誤動作を引き起こす恐れがあります。入力レベルが固定の場合はもちろん、V_{IL}（Max.）から V_{IH}（Min.）までの領域を通過する遷移期間中にチャタリングノイズなどが入らないように使用してください。
7. リザーブアドレス（予約領域）のアクセス禁止
   リザーブアドレス（予約領域）のアクセスを禁止します。アドレス領域には、将来の拡張機能用に割り付けられている リザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。
8. 製品間の相違について
   型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。同じグループのマイコンでも型名が違うと、フラッシュメモリ、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ幅射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

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。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、お客様の責任において、お客様の機器・システムを設計ください。これらの使用に起因して生じた損害（お客様または第三者いずれに生じた損害も含みます。以下同じです。）に関し、当社は、一切その責任を負いません。
2. 当社製品または本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害またはこれらに関する紛争について、当社は、何らの保証を行うものではなく、また責任を負うものではありません。
3. 当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を組み込んだ製品の輸出入、製造、販売、利用、配布その他の行為を行うにあたり、第三者保有の技術の利用に関するライセンスが必要となる場合、当該ライセンス取得の判断および取得はお客様の責任において行ってください。
5. 当社製品を、全部または一部を問わず、改造、改変、複製、リバースエンジニアリング、その他、不適切に使用しないでください。かかる改造、改変、複製、リバースエンジニアリング等により生じた損害に関し、当社は、一切その責任を負いません。
6. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、各品質水準は、以下に示す用途に製品が使用されることを意図しております。

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

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

   当社製品は、データシート等により高信頼性、Harsh environment 向け製品と定義しているものを除き、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（宇宙機器と、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に使用することは想定していません。たとえ、当社が想定していない用途に当社製品を使用したことにより損害が生じても、当社は一切その責任を負いません。
7. あらゆる半導体製品は、外部攻撃からの安全性を 100%保証されているわけではありません。当社ハードウェア／ソフトウェア製品にはセキュリティ対策が組み込まれているものもありますが、これによって、当社は、セキュリティ脆弱性または侵害（当社製品または当社製品が使用されているシステムに対する不正アクセス・不正使用を含みますが、これに限りません。）から生じる責任を負うものではありません。当社は、当社製品または当社製品が使用されたあらゆるシステムが、不正な改変、攻撃、ウイルス、干渉、ハッキング、データの破壊または窃盗その他の不正な侵入行為（「脆弱性問題」といいます。）によって影響を受けないことを保証しません。当社は、脆弱性問題に起因しまたはこれに関連して生じた損害について、一切責任を負いません。また、法令において認められる限りにおいて、本資料および当社ハードウェア／ソフトウェア製品について、商品性および特定目的との合致に関する保証ならびに第三者の権利を侵害しないことの保証を含め、明示または黙示のいかなる保証も行いません。
8. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は、データシート等において高信頼性、Harsh environment 向け製品と定義しているものを除き、耐放射線設計を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
10. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。かかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
12. お客様が当社製品を第三者に転売等される場合には、事前に当該第三者に対して、本ご注意書き記載の諸条件を通知する責任を負うものといたします。
13. 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。
14. 本資料に記載されている内容または当社製品についてご不明な点がございましたら、当社の営業担当者までお問合せください。

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

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

(Rev.5.0-1　2020.10)

本社所在地

〒135-0061 東京都江東区豊洲 3-2-24（豊洲フォレシア）

www.renesas.com

お問合せ窓口

弊社の製品や技術、ドキュメントの最新情報、最寄の営業お問合せ窓口に関する情報などは、弊社ウェブサイトをご覧ください。

www.renesas.com/contact/

商標について

ルネサスおよびルネサスロゴはルネサス エレクトロニクス株式会社の商標です。すべての商標および登録商標は、それぞれの所有者に帰属します。