06
23
2019

RXマイコンで、Unityによる単体テスト環境を作ってみた（前編）

RXマイコン テスト

Qiitaの記事「TDDによるマイコンのLチカ開発」を、ルネサス製のRXマイコンとIDE(e2studio)の組み合わせで、真似してみました。

1. RXマイコンで、Unityによる単体テスト環境をセットアップする。
2. シミュレータ環境、ターゲットボード環境の2通りを用意する。
3. 元の記事にあった、ホスト環境の構築、CMockの導入は省略。

- 前編では、RXマイコンのプロジェクト生成とprintf()での文字出力を行います。
- 後編はこちら。テストフレームワーク `Unity` を導入し、テストを行います。

環境

- Board: Target Board for RX231
  - マルツで3000円くらいで購入できる
  - エミュレータ機能内蔵(E2Liteとして認識)。USB接続だけで電源供給・デバック可能
- Device: RX231(R5F52318ADFP)
- IDE: e2Stdio V7.4.0
- Compiler: CC-RX V3.01.00
- Unit Test Framework: Unity (https://github.com/ThrowTheSwitch/Unity)

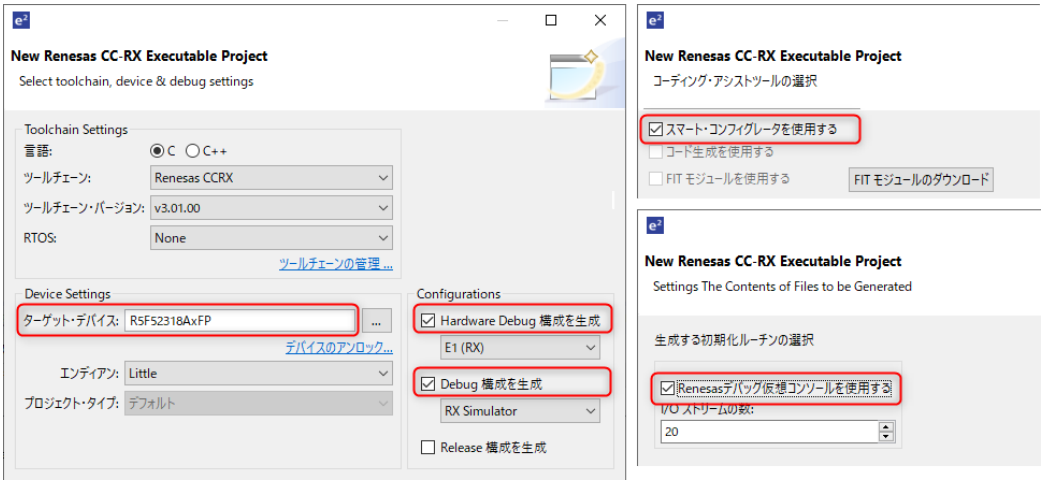
プロジェクト作成とprintf()での文字出力

e2studio(V7.3.0以降)では、プロジェクトを生成時に「Renesasデバック仮想コンソールを使用する」をチェックすると、IDE内の専用コンソールを標準出力として使えます。※但し、かなりの出力遅延あり。

1. プロジェクト作成

`メニュー` > `ファイル` > `新規` > `C/C++プロジェクト` から作成し、下記を設定。

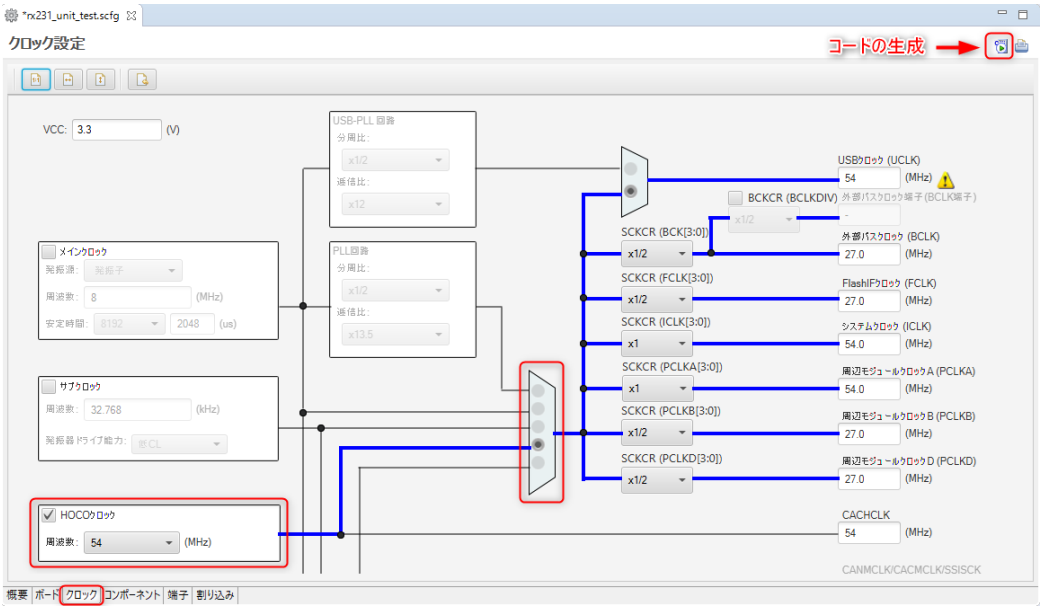
- Templete for New C/C++ Project
  - `Renesas RX` > `Renesas CC-RX C/C++ Executable Project` を選択
- Select toolchain, device&debug settings
  - Device Settingsのターゲット・デバイスは、 `RX231-100pin` > `R5F52318AxFP` を選択
  - Configrationsの `Hardware Debug構成を生成` と `Debug構成を生成` の２つにチェック
    ※それぞれ、ターゲットボード環境、シミュレータ環境に相当
- コーディングアシストツールの選択
  - `スマートコンフィグレータを使用する` にチェック
- 生成する初期化ルーチンの選択
  - `Renesasデバック仮想コンソールを使用する` をチェック



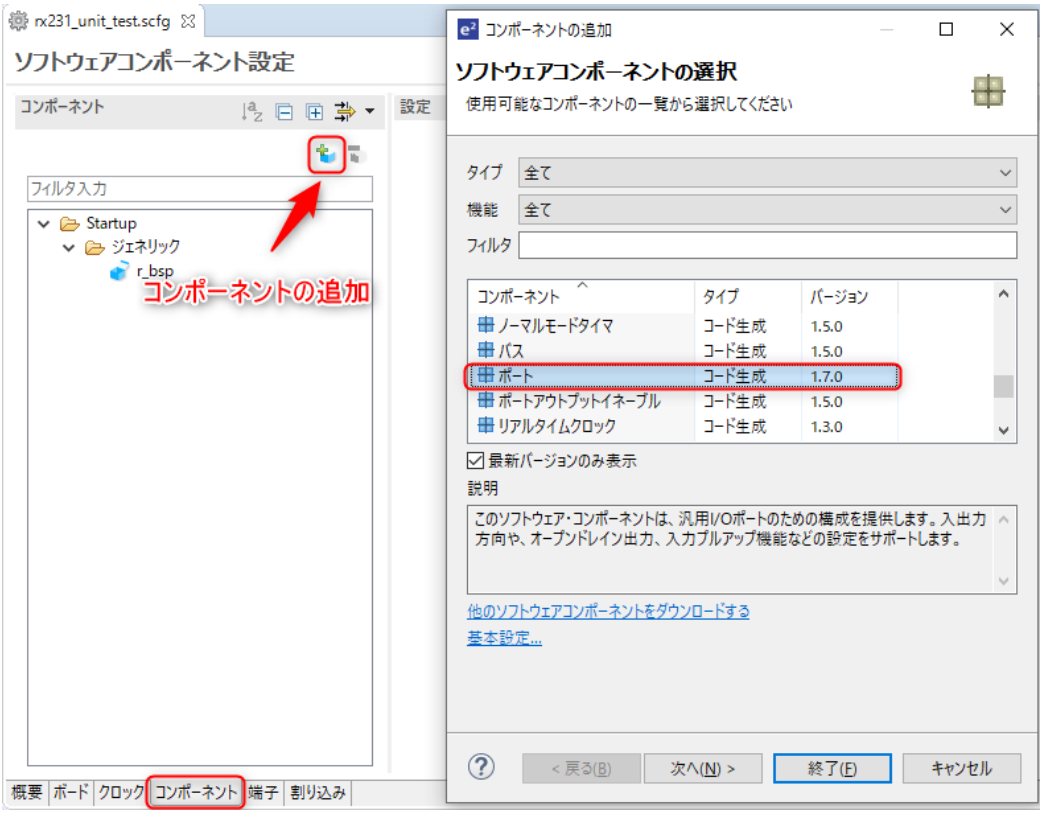
2. ハードウェア初期化コードの生成

スマートコンフィグレータによるコード生成を利用し、クロックとGPIOの初期化コードを生成します。

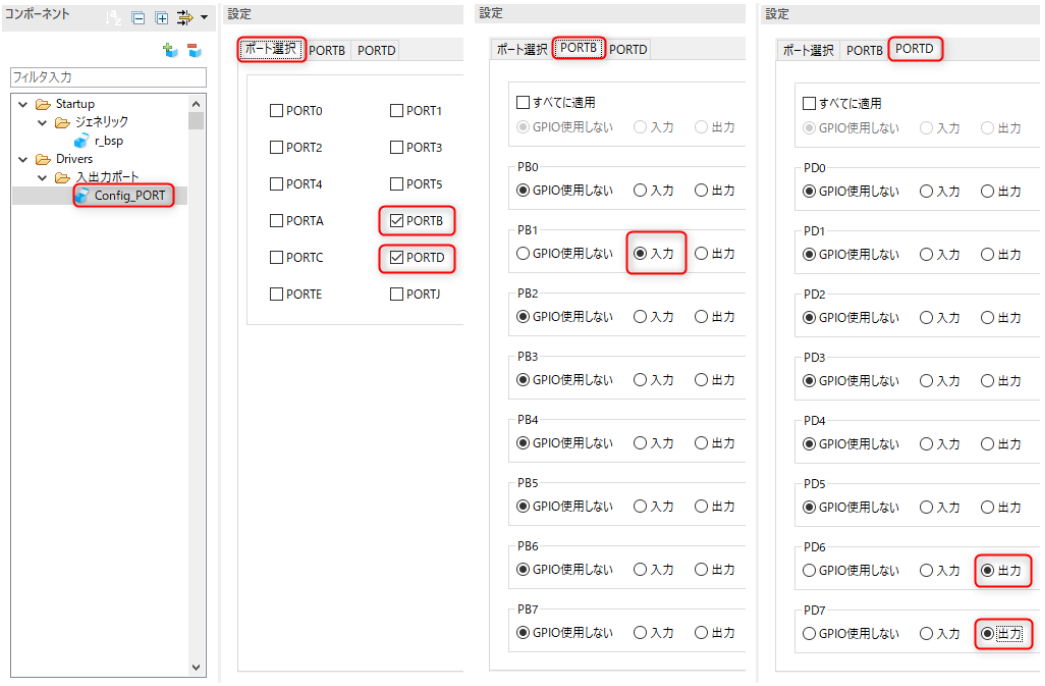
- `[クロック]` タブ
  - メインクロックのチェックを外す(実装されていないので)
  - 高速オンチップオシレータ `HOCOクロック(54MHz)` を選択



- ［コンポーネント］タブ
  - コンポーネントの追加 ボタンで、 ソフトウェアコンポーネントの選択 ダイアログを表示。
  - コンポーネント ［ポート］ を追加



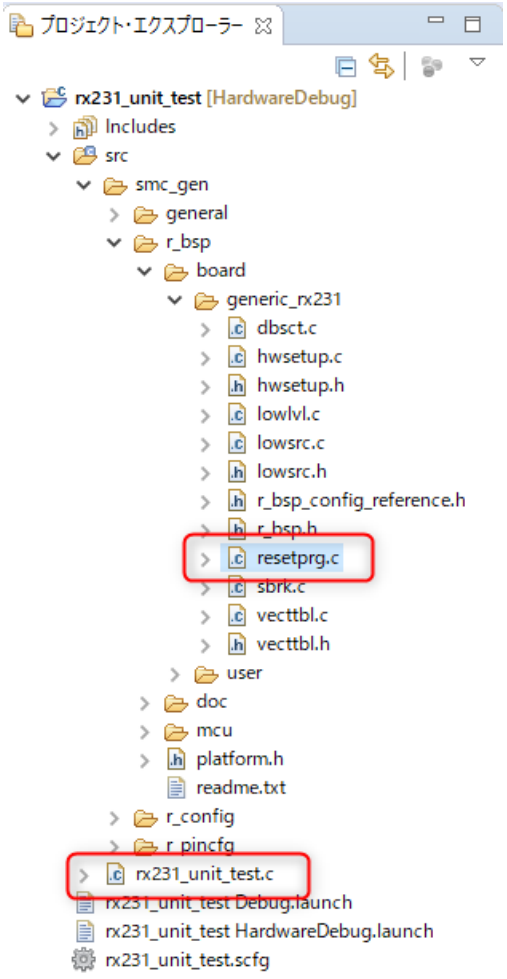
- コンポーネント `Config_PORT` を選択。LED出力2点(PD6,7)、SW入力1点(PB1)のGPIOを設定



- 右上にある コードの生成 ボタンを押すと、初期化コードが生成される

3. コード修正

プロジェクトの構成は下記のようになっています。ここでは２か所のコードを修正します。



- main関数に、printf() と Unityで使う putchar() の動作確認コードを追加。

/src/rx231_unit_test.c

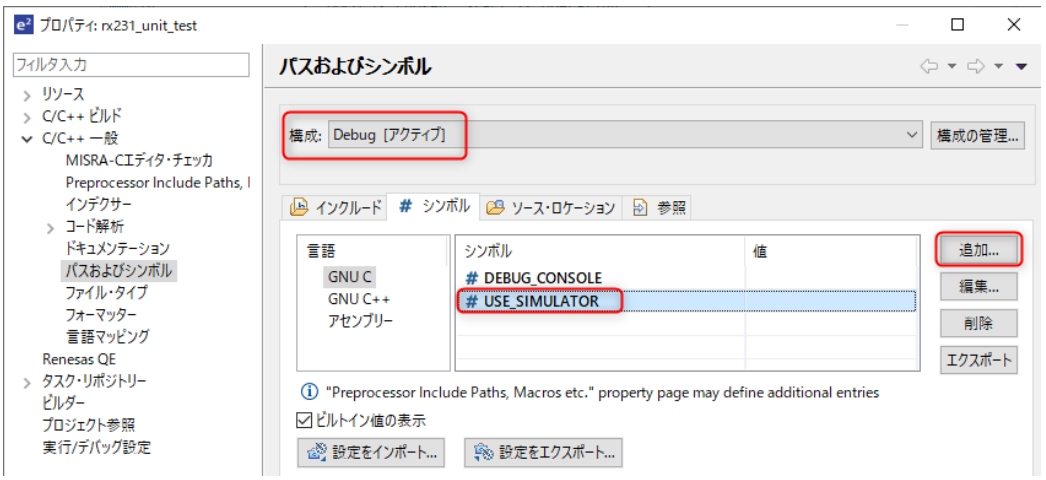
```
#include "r_smc_entry.h"
#include <stdio.h>

void main(void)
{
    printf("Hello World!\n");
    putchar('p');
    while(1);
}
```

- Debug構成(シミュレータ環境)に、コンパイルスイッチ `USE_SIMULATOR` を追加し、クロック初期化コードを一部無効にします[1]。

/src/smc_gen/r_bsp/board/generic_rx231/resetprg.c

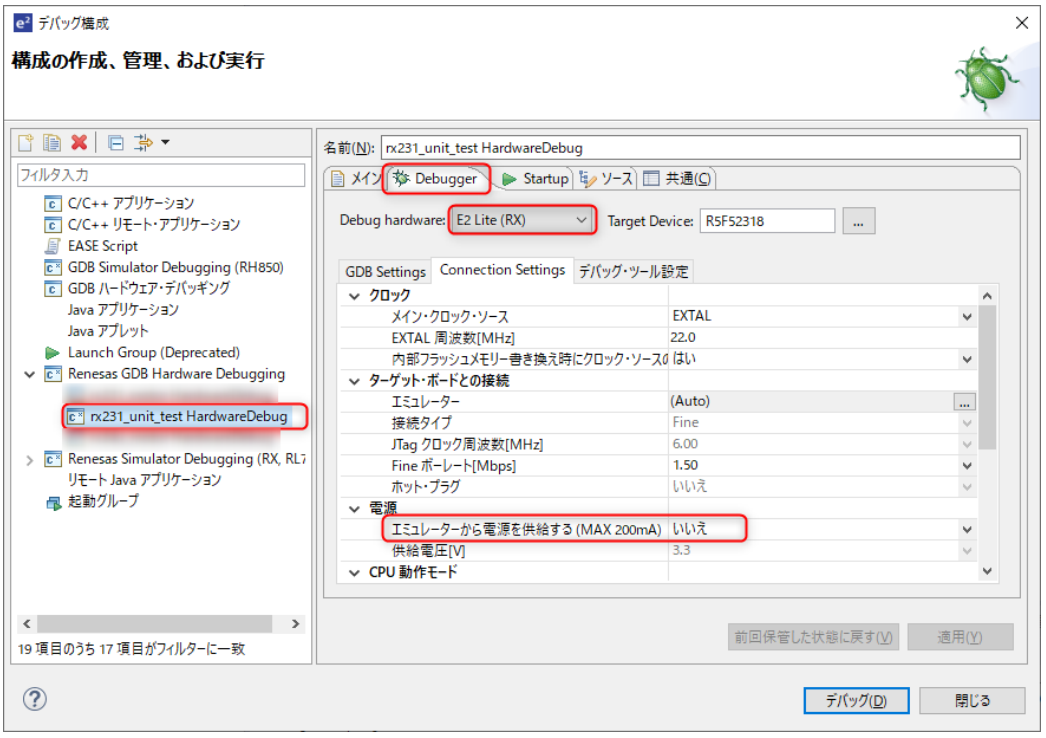
```
static void clock_source_select (void)
{
    ・・・・
    /* Make sure HOCO is stopped before changing frequency. */
    SYSTEM.HOCOCR.BYTE = 0x01;

    /* Set frequency for the HOCO. */
    SYSTEM.HOCOCR2.BIT.HCFRQ = BSP_CFG_HOCO_FREQUENCY;

    /* HOCO is chosen. Start it operating. */
    SYSTEM.HOCOCR.BYTE = 0x00;
    /* WAIT_LOOP */
#ifndef USE_SIMULATOR
    // シミュレータでは、ここでハードウェアの変化を無限に待ってしまう
    while (SYSTEM.OSCOVFSR.BIT.HCOVF != 1)
    {
        ;   // wait for stabilization
    }
#endif
    ・・・・
}
```

- プロジェクトエクスプローラから、 `プロパティ` > `C/C++一般` > `パスおよびシンボル` を選択。
  - `構成:` を、 `Debug` に変更
  - `＃シンボル` タブより、コンパイルスイッチ `USE_SIMULATOR` を追加

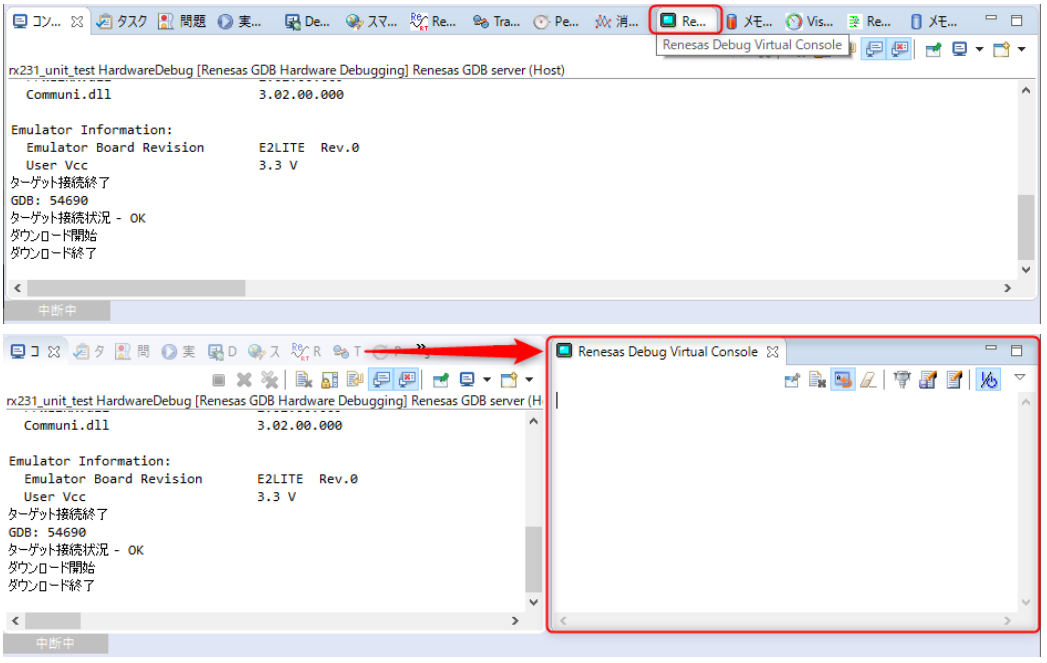


4. ターゲットボード環境でのビルドと動作確認

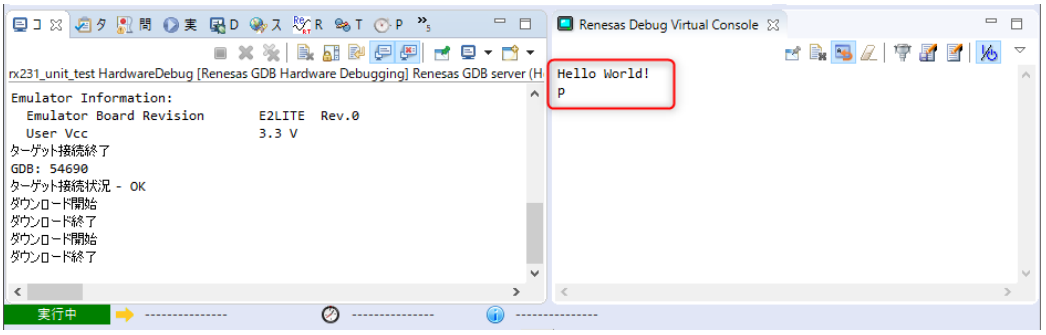
- ターゲットボードをPCに接続しておく
- ビルドする
  - プロジェクトエクスプローラから、 `ビルド構成` > `アクティブにする` > `HardwareDebug(Debug on hardware)` を選択。その後、 `プロジェクトのビルド` を選択。
- デバック設定とデバックの開始
  - `メニュー` > `実行` > `デバックの構成` より、 `～ HardwareDebug` の構成を選択し、 `Debugger` タブを開く
  - Debugger hardware を `E2 Lite(RX)` に設定
  - 電源 > エミュレータから電源を供給する を `いいえ` に設定
  - `デバック` のボタンを押下して、デバックを開始する。(再度デバックするときは `F11` で可)



- ターゲットへの接続・ダウンロードが完了したら、 `Renesas Debug Virtual Console` のウィンドウを開く。見当たらない時は `メニュー` > `Renesas Views` > `デバック` > `Renesas Debug Virtual Console` で表示。
  - ウィンドウをドラックして、コンソールと同時に表示したほうが使いやすい感じです。

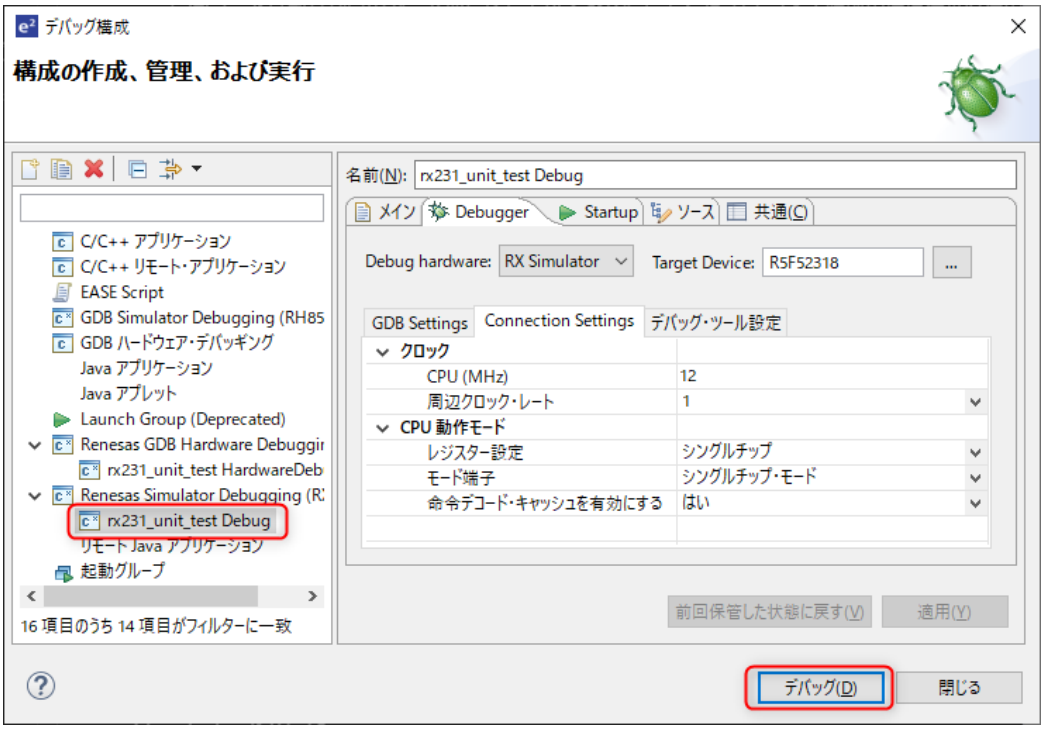


- プログラム実行(F8 を押下)。コンソールに"Hello World!"と"p"が表示されればＯＫ

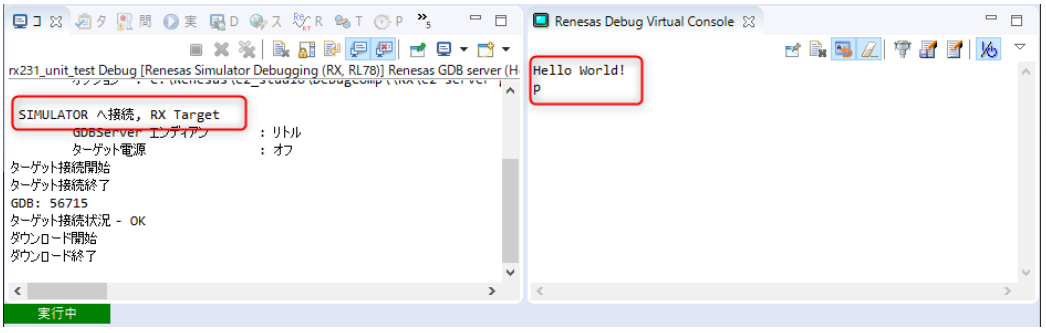


5.シミュレータ環境の設定と動作確認

- ターゲットボードをPCから外しておく
- ビルドする
  - プロジェクトエクスプローラから、 ビルド構成 > アクティブにする > Debug(Debug) を選択。その後、 プロジェクトのビルド を選択。
- デバッグ設定とデバッグの開始
  - メニュー > 実行 > デバッグの構成 より、 ～ Debug の構成を選択。(特に設定項目なし)
  - デバッグ のボタンを押下して、デバッグを開始する。(再度デバッグするときは F11 で可)



- プログラム実行(F8 を押下)。 Renesas Debug Virtual Console に"Hello World!"と"p"が表示されればＯＫ



最後に

e2studioのバージョンアップで `Renesas Debug Virtual Console` が簡単に使えるようになりました。このため、IDE上でのprintf()出力まではスムーズです。次は、Unityを導入して、単体テスト環境を構築します。

参考にした情報

1. ルネサスのドキュメント
   - e2 studio V7.3.0 リリースノート(R20UT4471EE0100)
   - Target Board for RX231 ユーザーズマニュアル(R20UT4168JJ0101)
2. ルネサスのFAQ
   - FAQ 3000062 : (e² studio)コンソール表示でprintfデバッグする方法

---

1. シミュレータでは、CPUの周辺ハードウェア動作はサポートされないため、ハードウェアの変化を待つループがあると、無限待ちになります。最初にシミュレータを利用した時、main()関数が呼ばれなくて困りました・・・↩

« RXマイコンで、UNITYによる単体テスト環境...
ANDROIDで、VUFORIAの POSITIONAL DEVICE ... »

プロフィール

sonoka_gi

電子工作やプログラミングなど、やってみたことのメモ

読者になる 1

検索

記事を検索

リンク

はてなブログ
ブログをはじめる
週刊はてなブログ
はてなブログPro

最新記事

Logicool製Webカメラ C922n のカメラ設定
C言語で、符号やサイズが異なる場合のキャスト動作を確認してみた
Processingで、学戦都市アスタリスクのクレジット表示を真似してみた
RXマイコンで、Unityを使ってハードウェアの動作確認をしてみた
RXマイコンで、Unityによる単体テストをやってみた

月別アーカイブ

▼ 2020 (3)
2020 / 12 (1)
2020 / 10 (1)
2020 / 7 (1)
▶ 2019 (4)
▶ 2018 (2)