ＲＬ７８_Ｇ２４　マイコン学習セット　マニュアル　入門、応用編

初版　２０２５．１１．１０

【　製品概要　】

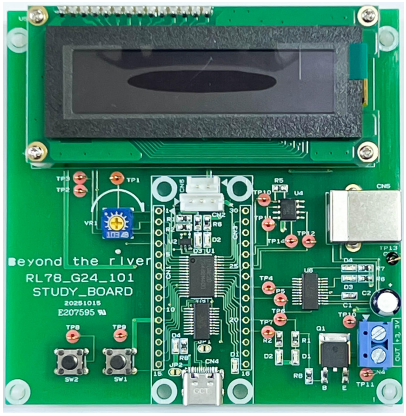
本マニュアルはＲＬ７８／Ｇ２４＿１０１ＧＡ（３０ピン）マイコンを使ったマイコン学習セットの開発環境構築、ソフトウエアインストゥール手順、添付ＣＤのサンプルプログラムの動作について解説しています。

入門、応用編はマイコンの基本的なハードウエアのアクセス方法、プログラムの書き方をサンプルプログラムを参考に学びます。習熟度をチェックするために、演習問題をプログラムで書くことにより、大きく理解が進み、様々な応用に対応できるようになります。

ルネサスエレクトロニクス社の統合開発環境ＣＳ+　ｆｏｒ　ＣＣ　における開発方法について記述してあります。

機能的な最大の特徴は、統合開発環境ＣＳ＋に新しく追加されたＣＯＭポートデバックです。従来はプログラムのダウンロード、デバックにＥ２Ｌｉｔｅ等、エミュレータが必要でした。ＣＯＭポートデバック機能は、パソコンとＣＰＵボードを直接ＵＳＢケーブルでつなぎ、ダウンロード、動作、ブレークポイントの設定、動作中の変数を見る機能などをＥ２Ｌｉｔｅなしで実現したものです（詳細後述）

弊社の学習ボードの最大の特徴は「質問が出来る」ことです。雑誌やネットの多くのマイコン学習教材は一方通行で質問出来ない、出来ても満足な回答が得られない製品が多く、知識のある人が周囲にいない場合、折角の興味がそこで断念せざるを得ない場合があります。この学習ボードで学習している最中、疑問に思ったこと、分からないかとがあれば遠慮なく弊社に何回でもお問い合わせください。担当が元気にしている限り必ず回答いたします。



１．学習環境、事前準備

１－１．　学習環境

ａ：学習セット　同梱物

ｂ：習得知識

ｃ：ＲＬ７８＿Ｇ２４＿１０１ＧＡ　ＣＰＵ部の特徴

ｄ：無償のＣＳ＋、ＲＬ７８用Ｃコンパイラのダウンロード

ｅ：ＣＤコピー、デバイスドライバのインストゥール

２．マイコン内蔵主要ペリフェラル（Ｉ/Ｏ、ＵＳＢ（ＵＡＲＴ）、A/D、PWM、割り込み、ＦＲＡＭ）の設定と動作

１．Ｉ／Ｏポート制御
（１）ＣＳ＋　コード生成（設計ツール）の使い方
（２）入出力ポートの初期設定
（３）出力ポートでＬＥＤ点灯、入力ポートでスイッチの読み込み【サンプルプログラム】
【演習】課題

２．ＵＳＢ通信
（１）ＳＩＯ（シリアルアイオー）ペリフェラルの初期設定
（２）ターミナルソフトを使用したＰＣ－マイコン間のデータ通信【サンプルプログラム】
（３）ＣＳ＋　ウオッチ窓を使用した送受信データの確認
【演習】課題

３．Ａ／Ｄ変換
アナログ値をデジタル値に変換します。
（１）Ａ／Ｄ（エーデーコンバータ）ペリフェラルの設定
（２）Ａ／Ｄ値の読み込み、ＣＳ＋のウオッチ窓を使用したリアルタイムでのデータの確認　【サンプルプログラム】
（３）Ａ／Ｄ値を０－３．３Ｖに変換しＵＳＢからパソコン側に送信する
【演習】課題

４．PWM制御
（１）PWM（パルス幅変調）が現代に多用される理由
（２）インプットキャプチャペリフェラル　PWMペリフェラルの設定　分解能の設定
（３）デューティ比を変えてＬＥＤの輝度変化を見る　【サンプルプログラム】
【演習】課題

５．ＦＲＡＭの読み書き
（１）ＳＰＩインターフェイスを使って、次世代のメモリであるＦＲＡＭ（強誘電体メモリ）の読み書きを行います。ＦＲＡＭは書き換え速度が高速、電源を切っても内容が保持される特徴を持ちます。　【サンプルプログラム】
【演習】課題

６．割り込み動作
（１）インターバルタイマペリフェラル　定周期割り込みの設定
（２）組込みマイコンに必須のマルチタスクの基礎、考え方
（３）割り込み、多重割り込みによる動作実習　【サンプルプログラム】
【演習】課題

応用

１．有機ＥＬ表示器を使った表示　　【サンプルプログラム】
（１）ソフトウエアで作るＩ２Ｃインターフェイス

【演習】課題

２．ＰＷＭを使用したモーター速度制御　　【サンプルプログラム】
（１）外部にトランジスタを追加する理由
【演習】課題

質問、お問い合わせ

本製品に対する質問やお問い合わせは以下にお願いします。どんな簡単なことでも大丈夫です。専門家がお答えします。

〒3501213
埼玉県日高市高萩 1141-1
有限会社ビーリバーエレクトロニクス
TEL：042（985）6982　EMAIL:info@beriver.co.jp

１−１．学習環境

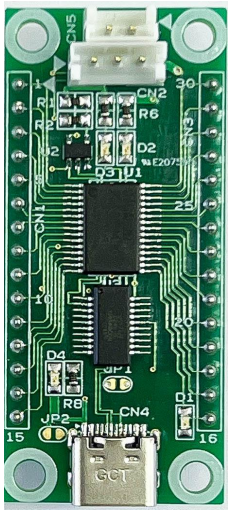
ａ：学習セット同梱物

ＲＬ７８＿Ｇ２４＿１０１学習ボード	１
ＣＤ（サンプルプログラム、デバイスドライバ、ドキュメント）	１
マニュアル（本誌） 入門、応用	１
電源ケーブル、ＵＳＢ　Ａ－Ｂケーブル、ＵＳＢ　Ｃケーブル	各１
モーター	１

ｂ：習得知識

習得知識	ルネサスエレクトロニクス社のＲＬ７８／Ｇ２４マイコンを使用し、マイコン、開発環境選定、使い方、Ｃ言語プログラムの作り方の基本を理解できます。 ＣＳ＋　コード生成（設計ツール）を使った、現在の主流であるプログラムを書かない内蔵ペリフェラルの初期設定の方法、基本を理解できます。 作成したプログラムを実機にダウンロード、実行し、ハードウエアの動作を確認。ソフトウエアとハードウエアの基本を理解できます。

ｃ．ＲＬ７８＿Ｇ２４＿１０１　ＣＰＵボード部の特徴



１．３０ピンとコンパクトですが、コードフラッシュメモリ　６４ＫＢ、ＲＡＭ　１２ＫＢ　と大容量です。
２．ＰＣとＵＳＢ　Ｃケーブルで接続するだけで、今までＥ２ｌｉｔｅが無いと出来なかったプログラムダウンロード、ブレークポイント設定、変数をウオッチ窓で見たり、ＣＳ＋の新しい機能、ＣＯＭデバックに対応します（詳細後述）。Ｅ２ｌｉｔｅが不要です。オプションボードでＥ２ｌｉｔｅでの従来型デバックも可能です。
４．開発環境をＡｒｄｕｉｎｏ®ＩＤＥ　※２、ルネサスエレクトロニクス社ＣＳ＋、ｅ２ｓｔｕｄｉｏなどで開発出来ます。
５．ルネサスエレクトロニクス社製のＲＦＷソフトで書き込み出来ます。量産時などに便利です。

※２　Ａｒｄｕｉｎｏ®ＩＤＥからプログラムのダウンロード、実行、シリアルモニタ機能が使用できます。各種ライブラリの動きを保証するものではありません。

電源：1．６Ｖ～５．５Ｖ　ＣＯＭデバック時はＵＳＢポートからの電源５Ｖを基板上の３端子電源で３．３Ｖに下げて使っています。単一　５．５ｍＡ（３．３Ｖ／４８ＭＨｚ時　ＴＹＰＥ　ＨＳモード）。

デバックコネクタ：ＵＳＢ－Ｃ　Ｔｙｐｅ　ＣＯＭデバックコネクタ実装済み。オプション　Ｅ２ｌｉｔｅ用コネクタ
ルネサスフラッシュライトプログラマ対応ポート実装済み。
基板サイズ：　　２３．６×５３×１５（Ｈ）ｍｍ
ＲｏＨＳ指令：　　基板、部品、半田付け全ての工程でＲｏＨＳ指令準拠仕様。

【　ＣＯＭポートデバック詳細　】

以前はＣＳ＋とマイコンボードを接続するのに、エミュレータが必要でしたが、最近（２０２５年頃）、統合開発環境ＣＳ＋にＣＯＭポートデバック機能が追加され、パソコンから直接、ＵＳＢで対応したハードを搭載したマイコン基板に接続してプログラムダウンロード、実行、ブレークポイント設定、変数の確認等が行えるようになりました。今回はそのハードを搭載したＲＬ７８＿Ｇ２４＿１０１ＧＡ　ＣＰＵマイコンをＣＯＭ接続で使います。※１

表 1-1　デバッグ機能一覧

項目		対応		内容
		COM Port	E2 Lite	
プログラム実行中のメモリ参照/変更				
疑似リアルタイム RAM モニタ(RRM)		○*1	○	参照時に CPU 占有
Dynamic Memory Modification(DMM)		○*1	○	変更時に CPU 占有
イベント機能		最大 2 点	最大 2 点	ハードウェアブレーク (もしくはトレース機能*2)に使用可能
ブレーク機能	ソフトウェアブレーク	○	○	最大 2000 点
	ハードウェアブレーク	○	○	実行アドレスもしくはデータアクセス
	強制ブレーク	○	○	-
トレース機能	取得情報	○	○	分岐元 PC 情報
	開始イベント	○	○	ユーザプログラム実行開始、イベント開始
	終了イベント	○	○	ユーザプログラム停止、イベント終了、トレース Full
実行時間計測機能		×	○	-
ホットプラグイン		×	○	-
カバレッジ機能		×	×	-

○:サポート、×：未サポート

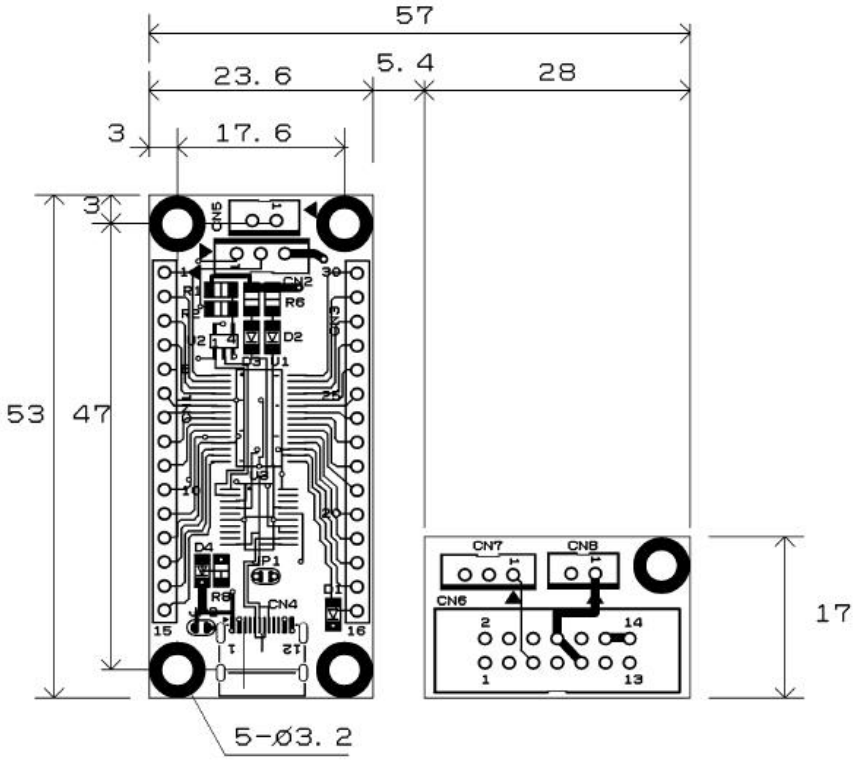
*1　RRM/DMM は変数単位では可能ですが、メモリ・パネル表示の表示量は 4,5 行程度、更新間隔は 5000msec 以上で使用されることを推奨します。表示量が多いとデバッガが無反応になる場合があります。

5秒から0.1秒にVer8.14.00から改善されました。

*2　トレース機能搭載デバイスのみ

ＣＰＵ部大きさ（部品面）

■ＲＬ７８＿Ｇ２４＿１０１ＧＡ　、Ｅ２ｌｉｔｅアダプタ　外形寸法図



ｄ：無償のＣＳ+、ＲＬ７８用Ｃコンパイラのダウンロード

プログラムの開発はルネサスエレクトロニクス社の統合開発環境ＣＳ+　ｆｏｒ　ＣＣ　でＣ言語を用い動作させることができます。ＣＤ添付のサンプルプログラムはこの環境下で作成されています。無償版をダウンロードして使用します。

ネット検索で→「ＣＳ＋　無償ダウンロード」の検索で表示されます。

統合開発環境 CS+ (旧CubeSuite+)

製品名	仕様・性能	試用期限
統合開発環境 CS+ for CC (RL78, RX, RH850用) 製品ページ 評価版ダウンロード	• 試用期限内は製品版(professional版)と同じ。試用期限を過ぎると各MCUにより以下の制限があります。 [RH850ファミリ] リンクサイズを256Kバイト以内に制限しています。professional版の機能は使用できません。 [RXファミリ] リンクサイズを128Kバイト以内に制限しています。professional版の機能は使用できません。 [RL78ファミリ] リンクサイズを64Kバイト以内に制限しています。professional版の機能は使用できません。 • コンパイラ※、デバッガを同梱。 • ※ CC-RL, CC-RX, CC-RH	60日 初めて評価版ソフトウェアツールをインストールした後、最初にビルドを行った日から60日間の試用期間があります。 試用期間内は、機能に制限はありません。 61日目以降は、リンクサイズ、professional版の機能が制限されます。

Ｃコンパイラ等も同梱されていています。ルネサスエレクトロニクス株式会社に登録が必要ですが、質問のときにも必要なのでしておいて、損はないと思います。

ダウンロード出来ましたら、指示に従い展開して下さい。

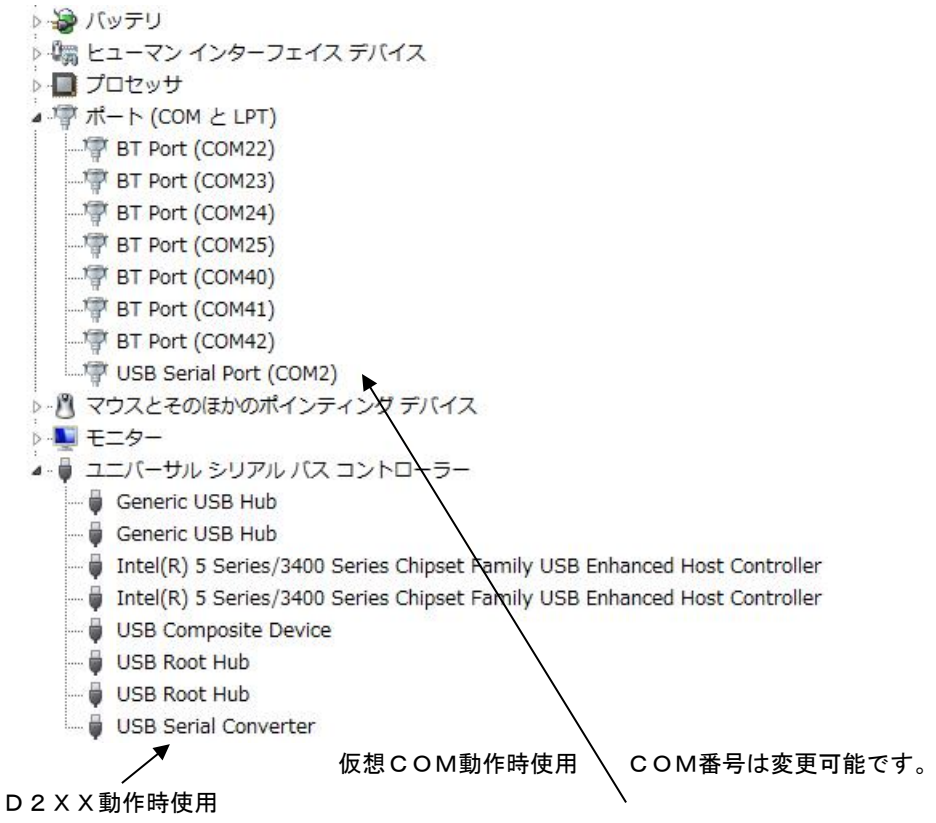
更に、ＲＬ７８用スマートコンフィグレータダウンロードが必要です。

ｅ：開発セット添付ＣＤコピー、デバイスドライバのインストゥール

事前にＣＤの中のホルダを例えばC:¥WrokSpace¥にコピーしてください。WorkSpaceはCS+をインストゥールすると自動形成されます。

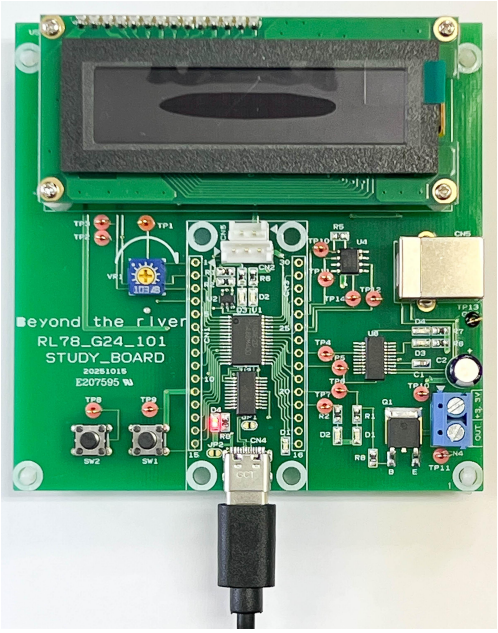
初めて、ＲＬ７８学習ボードをパソコンにＵＳＢケーブルで接続するとＯＳがＦＴ２３２ＲＬのデバイスドライバを要求してきますが、Ｗｉｎｄｏｗｓ　Ｕｐｄａｔｅに登録されているため、ユーザーは何もしなくても最新のデバイスドライバが自動的にインスツールされます。

正常にインスツールされると、以下のように２つのデバイスが確認出来ます。



社内ＬＡＮ等の制限で自動的にデバイスドライバがインスツールされない場合、添付するＦＴＤＩデバイスドライバを使用するか、ＯＳに合ったＶＣＰ　デバイスドライバをＦＴＤＩのサイトからダウンロード、インスツールしてください。https://ftdichip.com/drivers/
ＵＳＢケーブルでＰＣと基板を接続すると上記のようにＣＯＭ番号が確認できれば成功です。

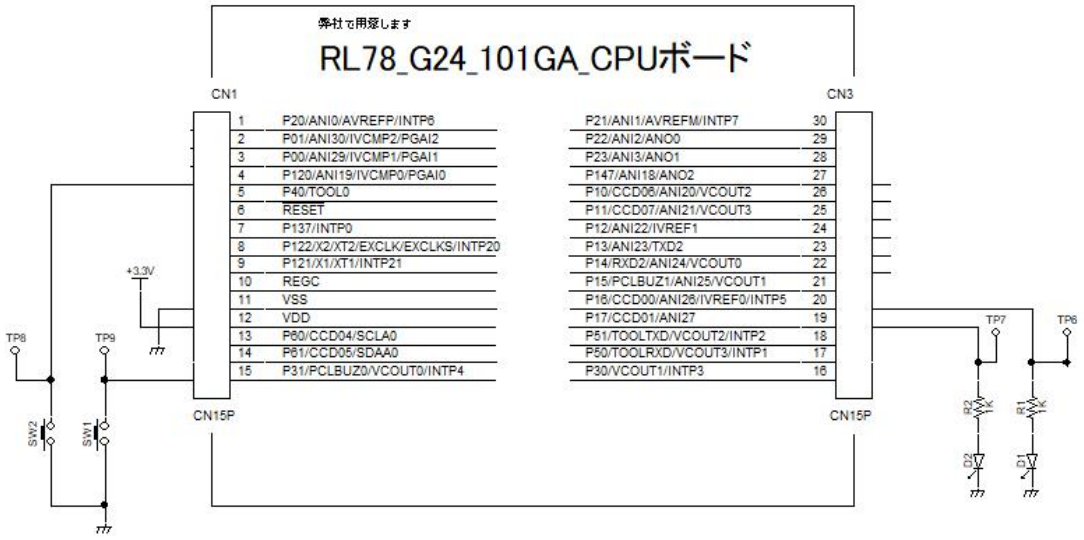
２．マイコン内蔵主要ペリフェラル（Ｉ/Ｏ、ＵＳＢ（ＵＡＲＴ）、Ａ/Ｄ、ＰＷＭ、割り込み、ＦＲＡＭ）の設定と動作



1．I／Oポート制御

【　動作概要　】

■サンプルプログラム名　RL78__G24__seminar1__IO

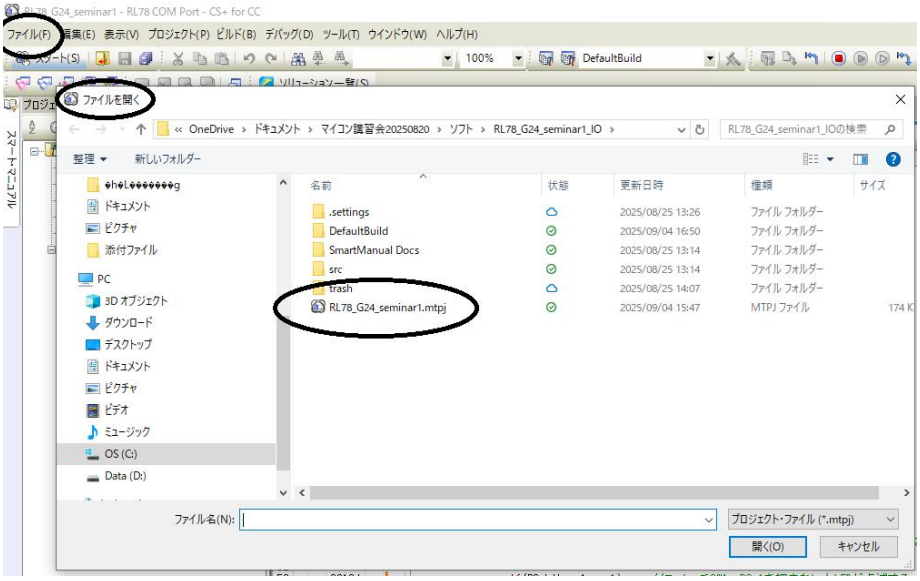


ＬＥＤ２つとスイッチを２つを使います。スイッチを押さないときは約５００ｍｓｅｃ周期で点滅しますが、ＳＷ１を押すと消灯、ＳＷ２を押すと点灯するプログラムです。

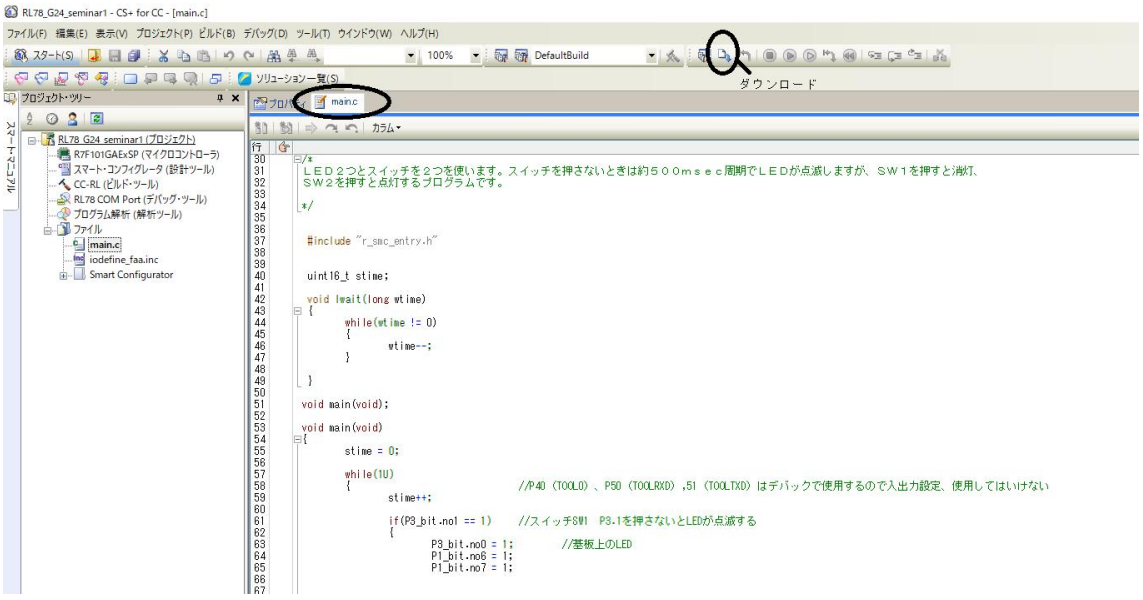
初めにブレッドボード上で上記を参考に配線を行って下さい。ＴＰ７等の端子は実在するわけではなく、テスターやオシロで測定の時に使う名称です。配線が終了したら次に移ります。

（１）ＣＳ＋　コード生成（設計ツール）の使い方

始めにサンプルプログラムを開きます。ファイル→ファイルを開く→RL78_G24_seminar1.mtpj をダブルクリック。

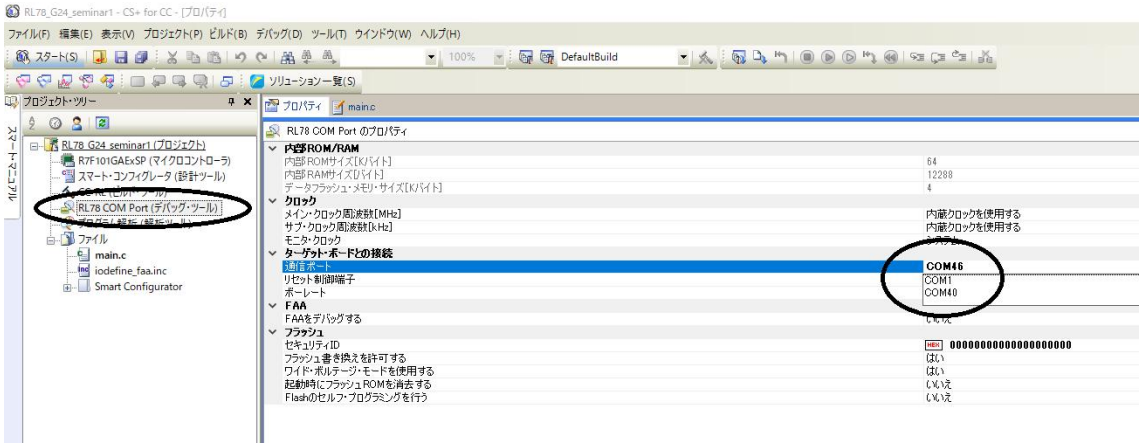


下記のような画面が開いたら、サンプルプログラムをダウンロードします。

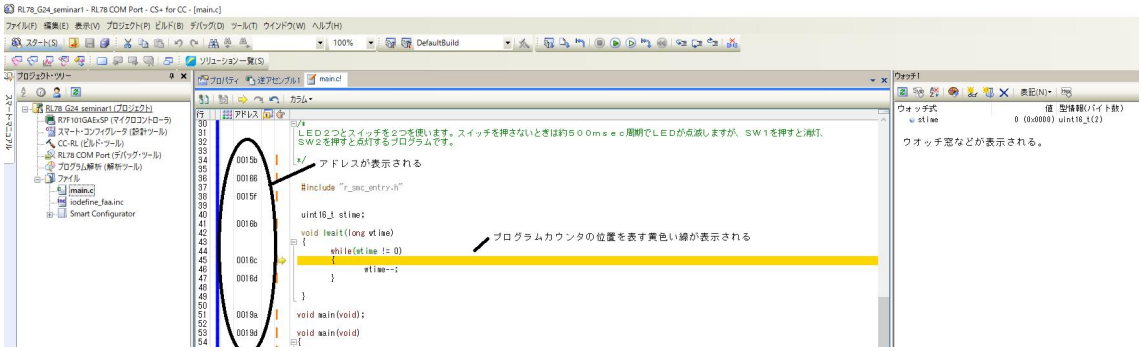


通常、初めは以下のようなメッセージが出ます。理由はプログラムが作成された COM 番号と実行者の COM 番号が一致しないからです。そこで

RL78COMPort（デバックツール）→ターゲットボード接続→現在 COM46 になっていますが、これを PC が現在認識している COM1、または COM40 にします。（個々のパソコンで番号は違います、個々の表示に合わせて下さい）



再び、ダウンロード。成功すると
あ）プログラムの左にアドレスが表示されます。
い）プログラムカウンタの位置を示す黄色のカーソルが表示されます。
う）右にウオッチ窓が表示されます。



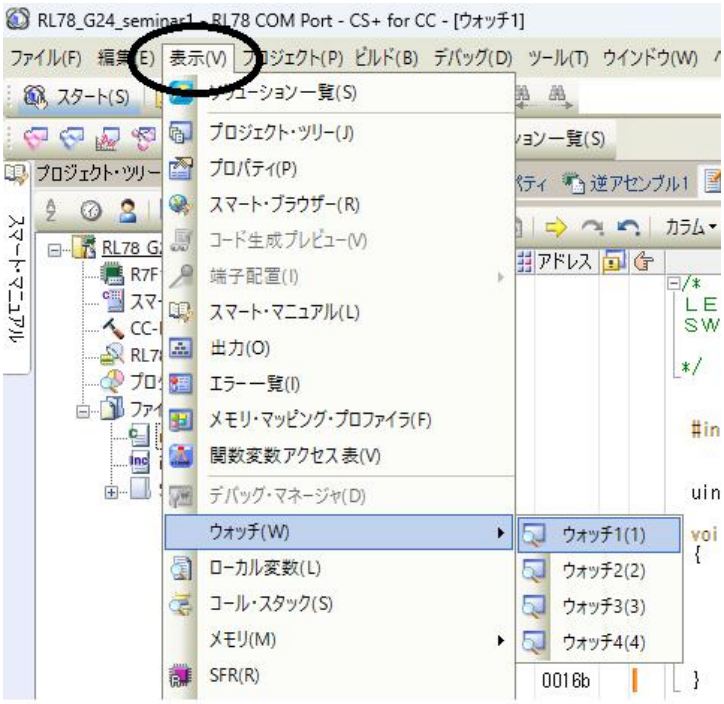
CPUリセット後、プログラムを実行します。

基板上のＤ１が０．５秒周期で動作すれば、プログラムは正常に動いています。ブレットボードに追加したＬＥＤが点滅しない、スイッチを押してもＬＥＤの点滅が止まらない場合、配線の間違いです。

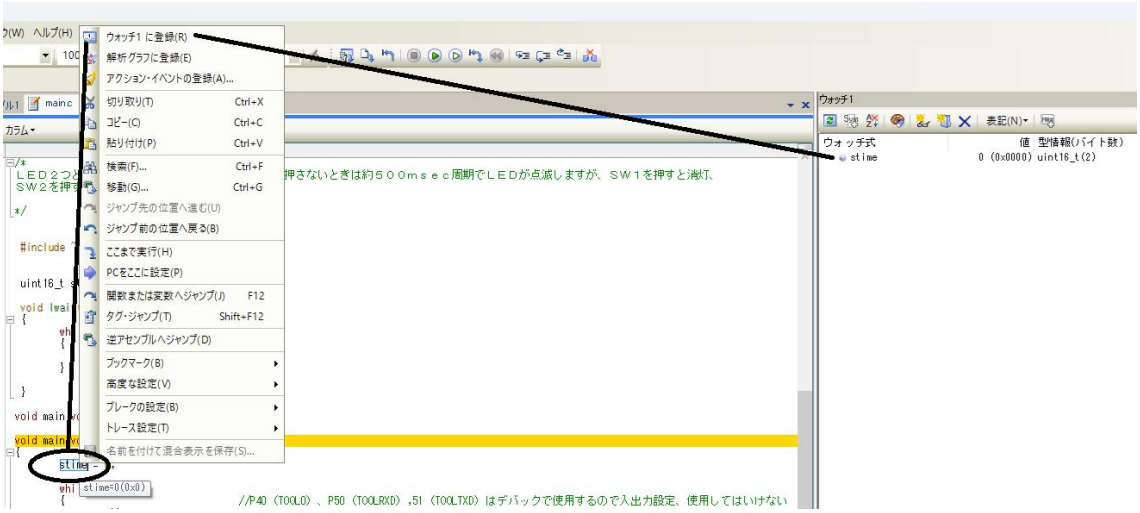
●ウオッチ窓の使い方

プログラムの初めに　　stime++;　というコードがありますが、これはウオッチ窓の使い方の例として使用する変数です。

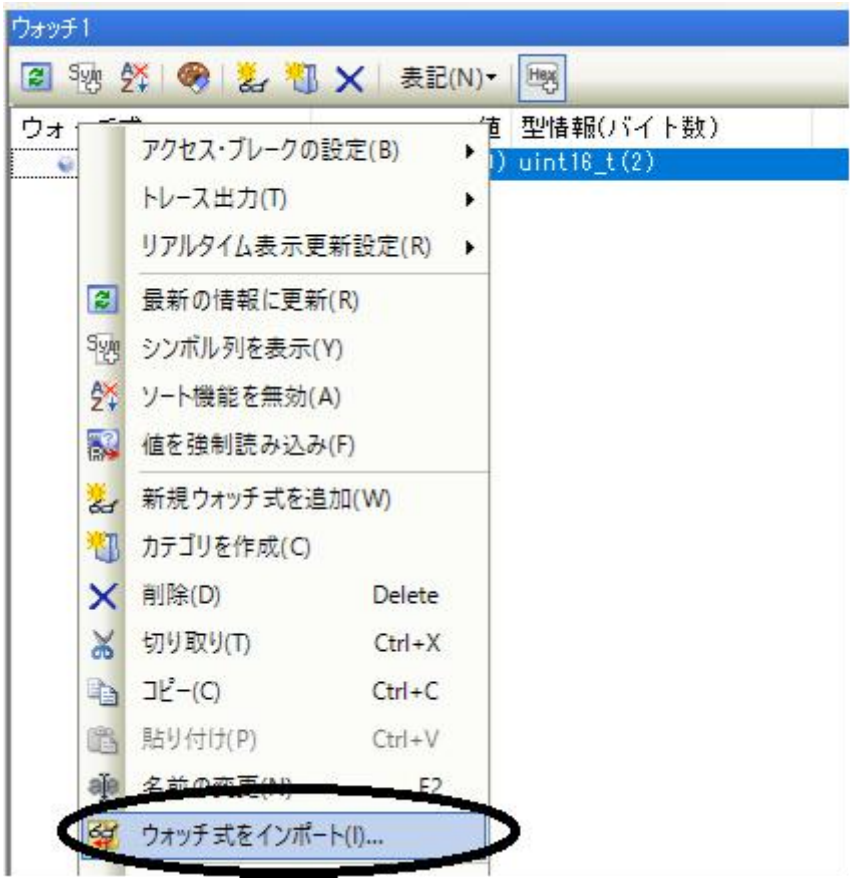
もし、プログラムがダウンロードされた後でも、ウオッチ窓が表示されていない場合、以下の方法で表示できます。



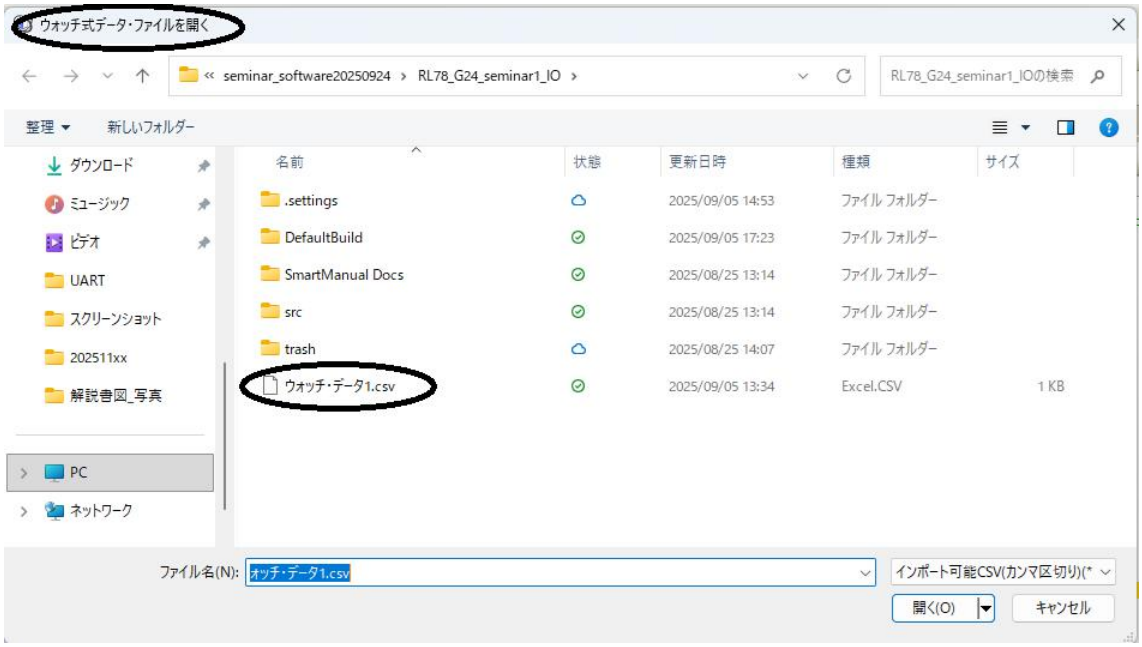
ウオッチ窓の使い方には２種類あります。１つは以下のように変数を左ドラッグし、右クリックでウオッチ窓に登録する方法。



もう1つはマウスをウオッチ窓上に置いて、右クリック。ウオッチ式をインポート

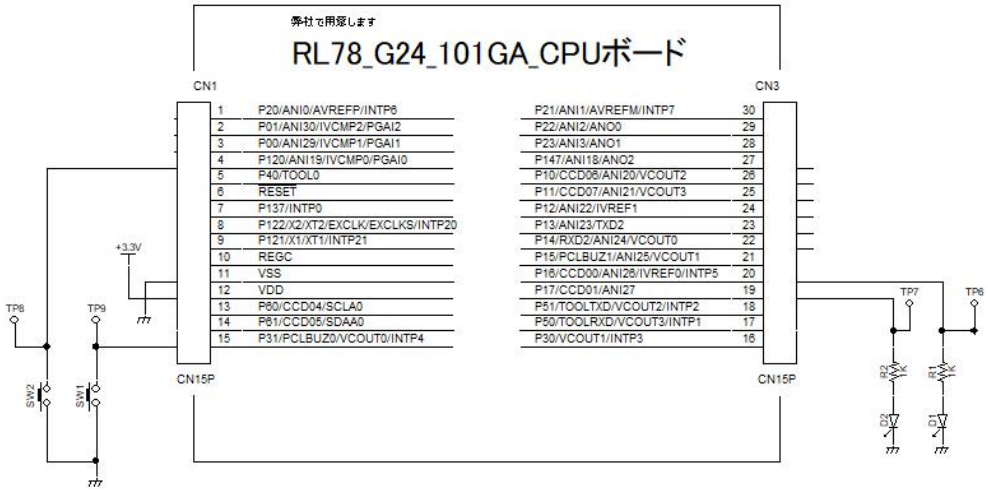


ウオッチ式がある場合（プログラムの作者が製作していた場合）、それを読み込むとウオッチ窓に変数等が表示されます。



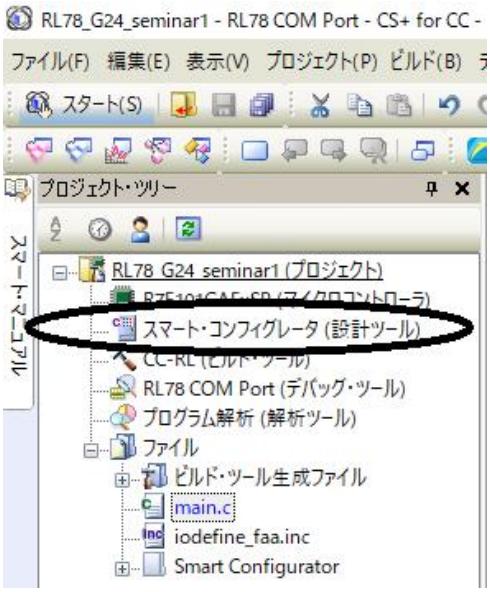
（２）入出力ポートの初期設定

このプログラムを製作するにあたり、行った手順を示します。まず、ポートの入出力を設定します。

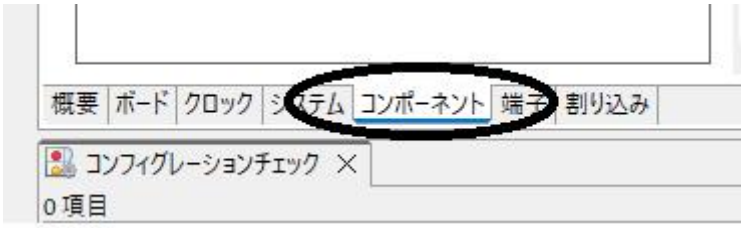


回路図からＰ１２０、Ｐ３１を入力、Ｐ１６，Ｐ１７を出力に設定します。

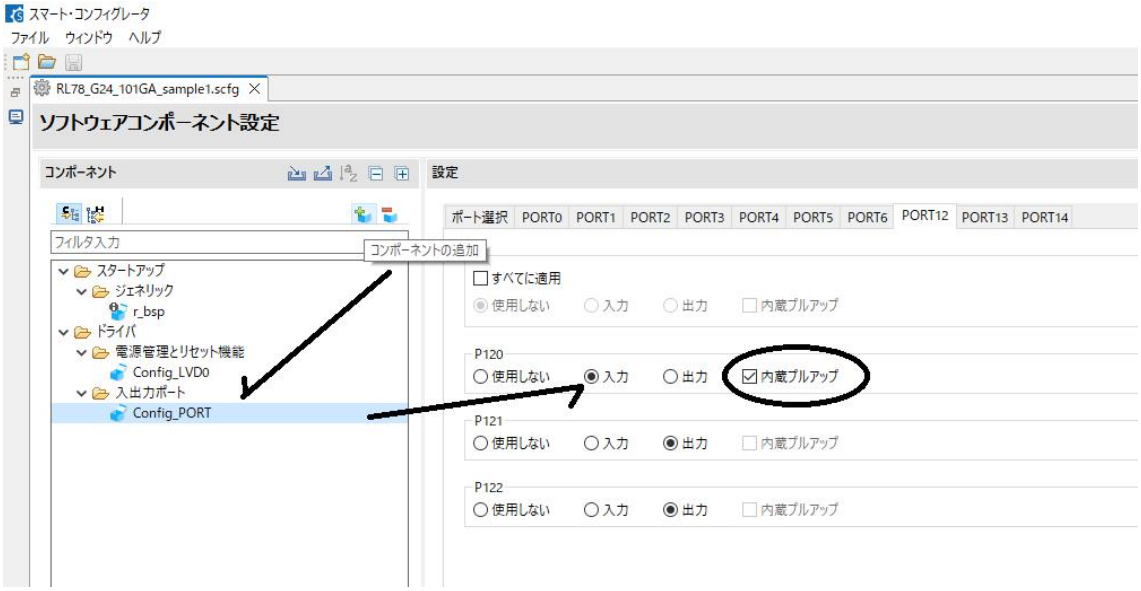
スマートコンフイグレータを起動します。



開いたら、コンポーネントを選択

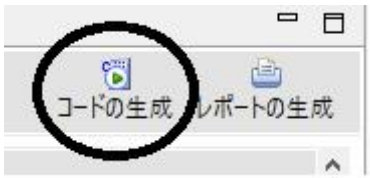


初めは何もないので、「Config__PORT」の追加、P120 ポートを入入力、内蔵プルアップ ON



同様の方法で、Ｐ３１を入力、Ｐ１６，Ｐ１７を出力に設定します。

一通り、設定したら「コードの生成」をクリックします。



これでポートの初期設定は終わりです。

（３）出力ポートでＬＥＤ点灯、入力ポートでスイッチの読み込み【サンプルプログラム】

プログラムです。

```
/*
ＬＥＤ２つとスイッチを２つを使います。スイッチを押さないときは約５００ｍｓｅｃ周期でＬＥＤが点滅しますが、ＳＷ１を押すと消灯、
ＳＷ２を押すと点灯するプログラムです。
*/
 #include "r_smc_entry.h"


 uint16_t stime;

 void lwait(long wtime)
 {
        while(wtime != 0)
        {
            wtime--;
        }

 }

void main(void);
```

```
void main(void)
{
        stime = 0;

        while(1U)
        {       //P40（TOOL0）、P50（TOOLRXD）,51（TOOLTXD）はデバックで使用するので入出力設定、使用してはいけない
                stime++;

                if(P3_bit.no1 == 1)  //スイッチ SW1   P3.1 を押すと LED が消える
                {
                        P3_bit.no0 = 1;       //基板上の LED
                        P1_bit.no6 = 1;                 //
                        P1_bit.no7 = 1;


                        lwait(800000);

                        if(P12_bit.no0 == 1)//スイッチ SW2   P12.0 を押すと LED が点灯する
                        {
                                P3_bit.no0 = 0;       //基板上の LED
                                P1_bit.no6 = 0;
                                P1_bit.no7 = 0;
                        }

                        lwait(800000);
                }
        }
}
```

【 解説 】

```
if(P3_bit.no1 == 1)  //スイッチ SW1   P3.1 を押すと LED が消える
{
    P3_bit.no0 = 1;       //基板上の LED
    P1_bit.no6 = 1;       //
    P1_bit.no7 = 1;
```

SW1 を押さないと P3.1 は 1（3.3V）なので、P3.0,P1.6,P1.7 ポートが 1（3.3V）になり、LED が点灯します。押すと以降のプログラムはスルーされます。前回、消灯で終わっているので、消灯が継続します。

```
lwait(800000);
```

ウエイト時間、点灯が継続し、SW2 を押さないと P12.0 が1になり、3つの LED が消灯します。

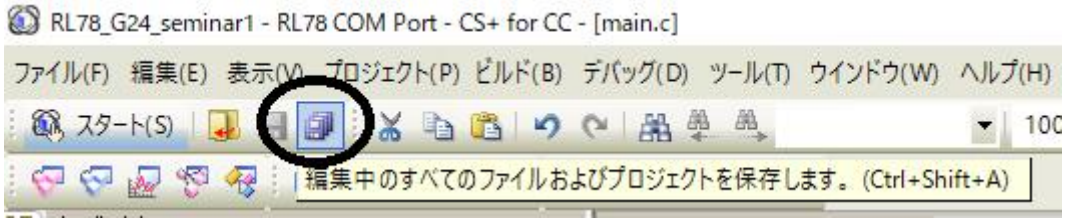
```
if(P12_bit.no0 == 1)    //スイッチSW2  P12.0を押すとLEDが点灯する
{
        P3_bit.no0 = 0; //基板上のLED
        P1_bit.no6 = 0; //0でポートから0Vが出力されます。
        P1_bit.no7 = 0;
}
```

消灯を継続します。時間経過後、while(1u)頭に戻り、繰り返します。押さないと消灯プログラムをスルーするので、点灯が継続します。

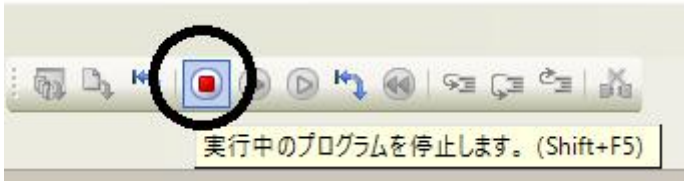
【 演習 】

1. 今、上記プログラムでは D1,D2 は同じときに光り、同じときに消えますが、D1 が光るときには D2 は消灯、D1 が消える時には D2 が点灯に変えてみて下さい。

プログラムを修正したらセーブします。



実行中止。



ビルド後、デバックツールへプログラムをダウンロード。修正部にエラーが無ければ、ダウンロードされます。エラーがあればエラーの行番号と内容が示されるので、チェックしてください。

「実行」で演習課題の動きになれば OK です。

参考サンプルプログラムは演習ホルダにあります。

ところで、ポートから出力できる電流はハードウエアマニュアルに書かれていて、直流で10mA、交流で19mA(デューティ ≦70%)です。今回はLEDの FV(順方向降下電圧)が 2V とすると、(3.3V-2V)/1K=1.3mA で十分低い値で問題ありません。

43.3　DC特性

43.3.1　端子特性

（TA = –40～+105°C, 1.6 V≦EVDD0≦VDD≦5.5 V, VSS = EVSS0 = 0 V）　(1/7)

項目	略号	条件		Min.	Typ.	Max.	単位
ハイ・レベル許容出力電流注1	IOH1	P00-P06, P10-P17, P30, P31, P40-P43, P50-P55, P62, P63, P70-P77, P120, P130, P140, P141, P146, P147 1端子	1.6 V≦EVDD0≦5.5 V			–10.0注2	mA
		P00-P04, P40-P43, P120, P130, P140, P141　合計 （デューティ≦70%時注3）	4.0 V≦EVDD0≦5.5 V			–55.0注4	mA
			2.7 V≦EVDD0＜4.0 V			–10.0	mA
			1.8 V≦EVDD0＜2.7 V			–5.0	mA
			1.6 V≦EVDD0＜1.8 V			–2.5	mA
		P05, P06, P10-P17, P30, P31, P50-P55, P62, P63, P70-P77, P146, P147　合計 （デューティ≦70%時注3）	4.0 V≦EVDD0≦5.5 V			–80.0注5	mA
			2.7 V≦EVDD0＜4.0 V			–19.0注7	mA
			1.8 V≦EVDD0＜2.7 V			–10.0	mA
			1.6 V≦EVDD0＜1.8 V			–5.0	mA
		全端子合計 （デューティ≦70%時注3）	1.6 V≦EVDD0≦5.5 V			–135.0 注6	mA

2. USB通信

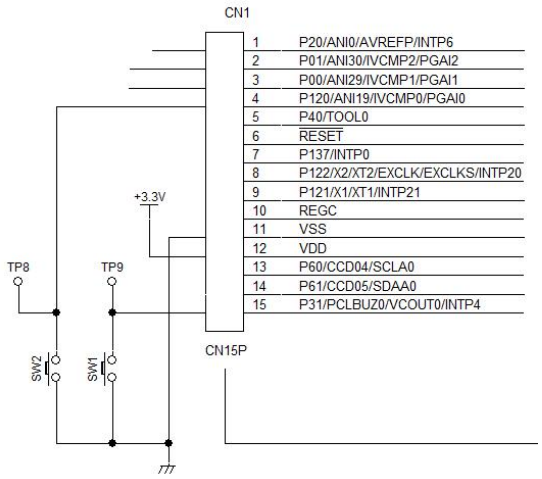
【 動作概要 】

■サンプルプログラム名　RL78__G24__seminar2_UART

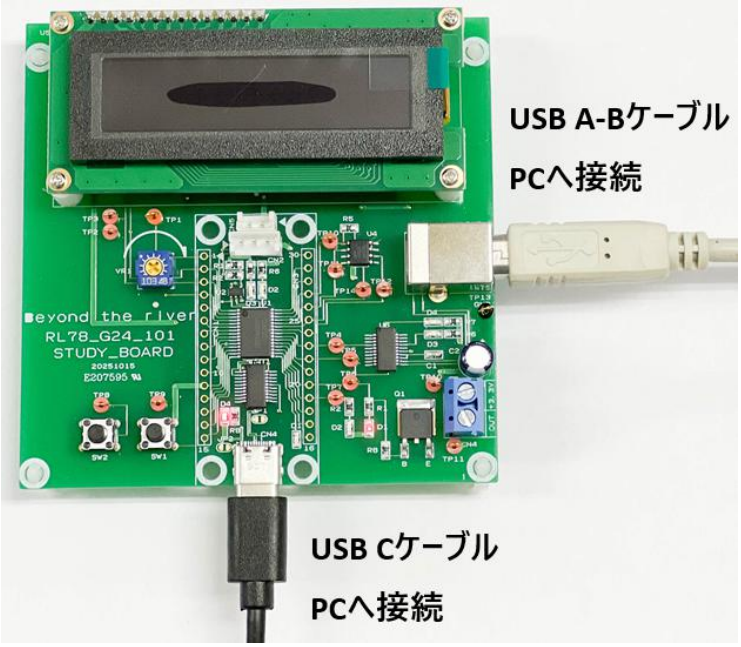
SW1（P31）を押すと RL78→UART→USB→PC 側に ASCII 文字が出力されます。
SW2（P120）を押すと RL78→UART→USB→PC 側にシフト JIS 漢字が表示されます。
PC 側から RL78 に送信するとエコーバックされます。

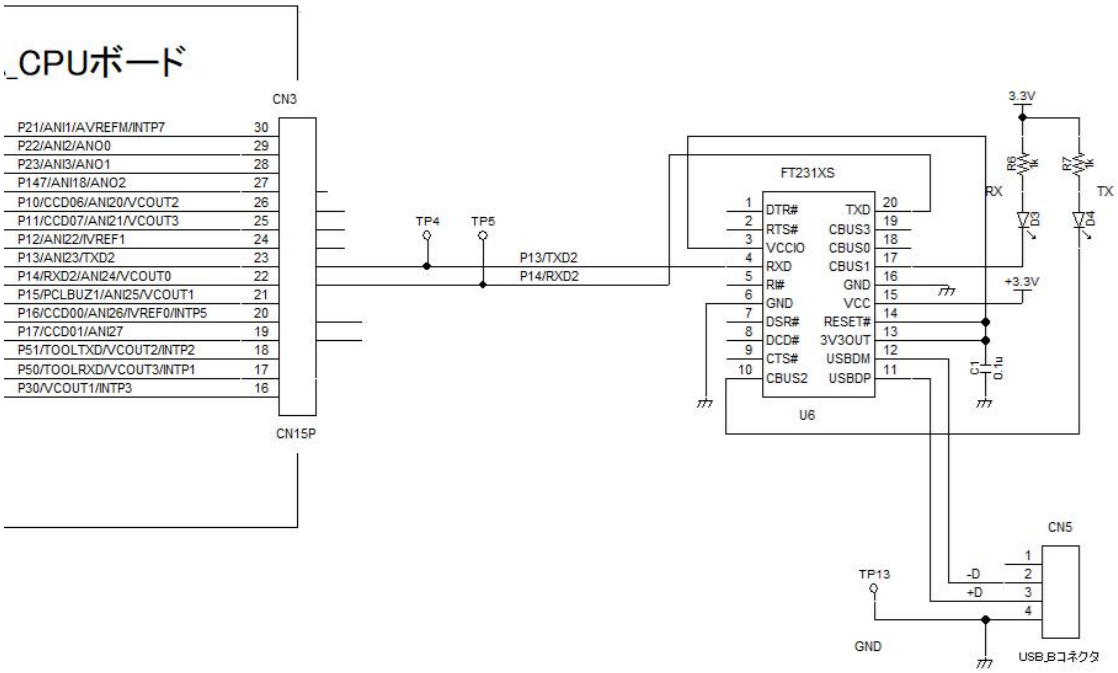
【 配線 】

ＳＷ１とＳＷ２はＩＯポート制御のをそのまま使います。



USB－SIO2基板への配線はP13、P14、3．3V、GNDの線が基板上で接続されています。添付のUSB　A－Bケーブルで学習ボードと PC を接続します。合計 2 本のケーブルが PC の USB ポートにつながります。

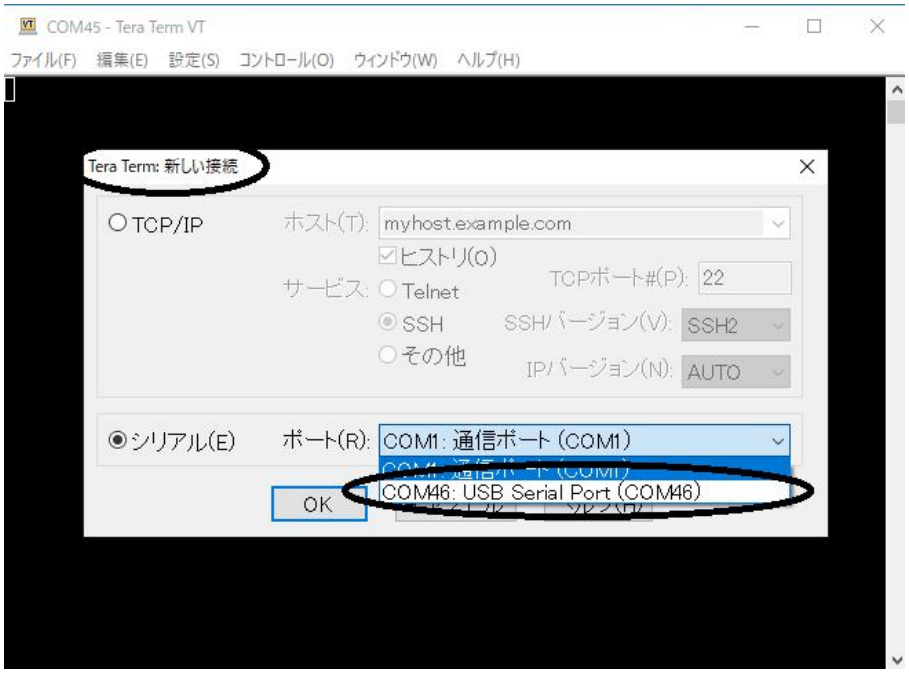



_CPUボード
CN3
P21/ANI1/AVREFM/INTP7 30
P22/ANI2/ANO0 29
P23/ANI3/ANO1 28
P147/ANI18/ANO2 27
P10/CCD06/ANI20/VCOUT2 26
P11/CCD07/ANI21/VCOUT3 25
P12/ANI22/IVREF1 24
P13/ANI23/TXD2 23
P14/RXD2/ANI24/VCOUT0 22
P15/PCLBUZ1/ANI25/VCOUT1 21
P16/CCD00/ANI26/IVREF0/INTP5 20
P17/CCD01/ANI27 19
P51/TOOLTXD/VCOUT2/INTP2 18
P50/TOOLRXD/VCOUT3/INTP1 17
P30/VCOUT1/INTP3 16
CN15P
TP4
TP5
P13/TXD2
P14/RXD2
FT231XS
DTR# TXD
RTS# CBUS3
VCCIO CBUS0
RXD CBUS1
R# GND
GND VCC
DSR# RESET#
DCD# 3V3OUT
CTS# USBDM
CBUS2 USBDP
U6
3.3V
R6
R7
RX
TX
D3
D4
+3.3V
C1
0.1u
CN5
TP13
-D
+D
GND
USB_Bコネクタ

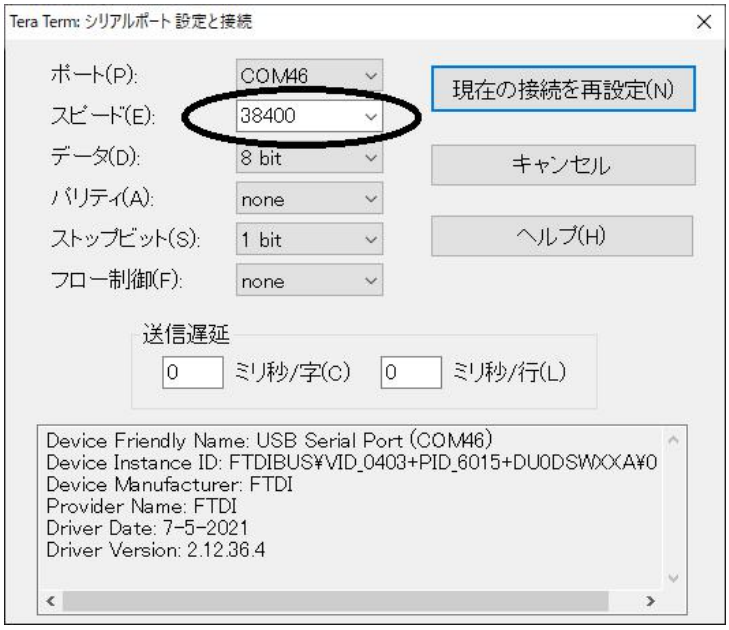
【ターミナルソフトの立ち上げ】

ターミナルソフトは何でも構いませんが、例ではＴｅｒａＴｅｒｍを使いました。

ファイル→TeraTarm 新しい接続→シリアル　お使いのPCの COM 番号に合わせて下さい。例では COM46。

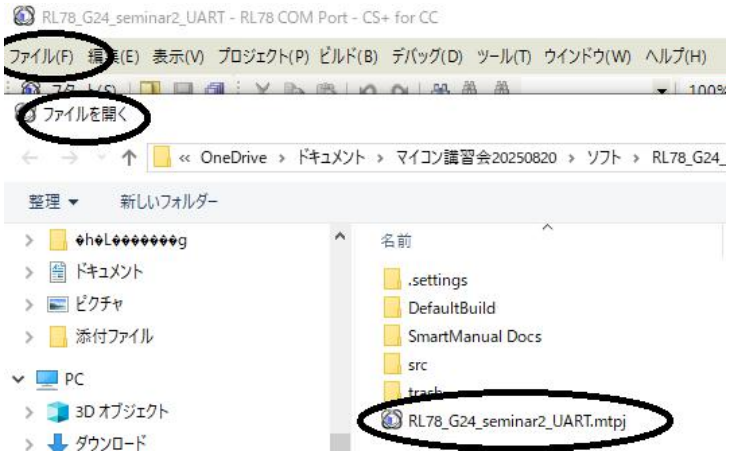


ボーレートは38400bpsにします。

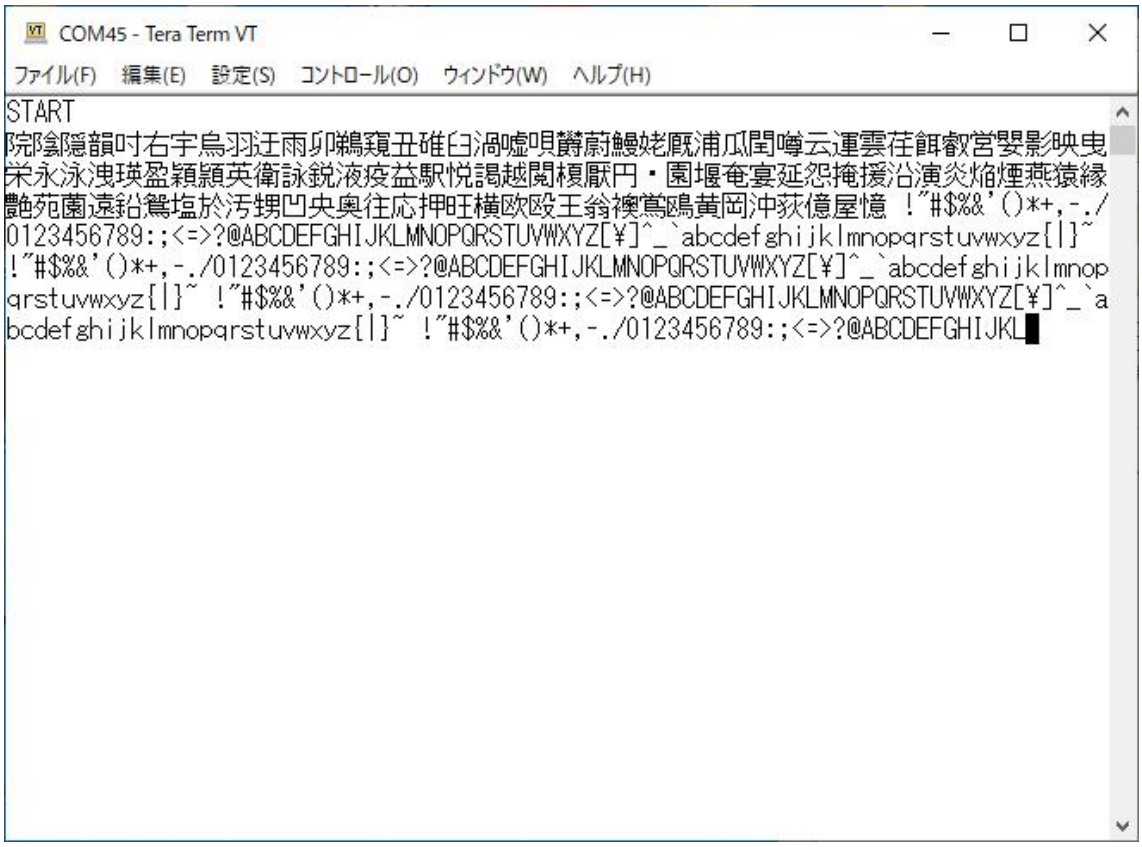


更に、設定→端末の設定→漢字受信、送信とも Shift-JIS にします。

ここまで出来たらサンプルプログラムをダウンロードし、実行してみます。
ファイル→ファイルを開く→RL78_G24_seminar_UART.mtpj　をダブルクリック。
うまくダウンロード出来たら「実行」します。

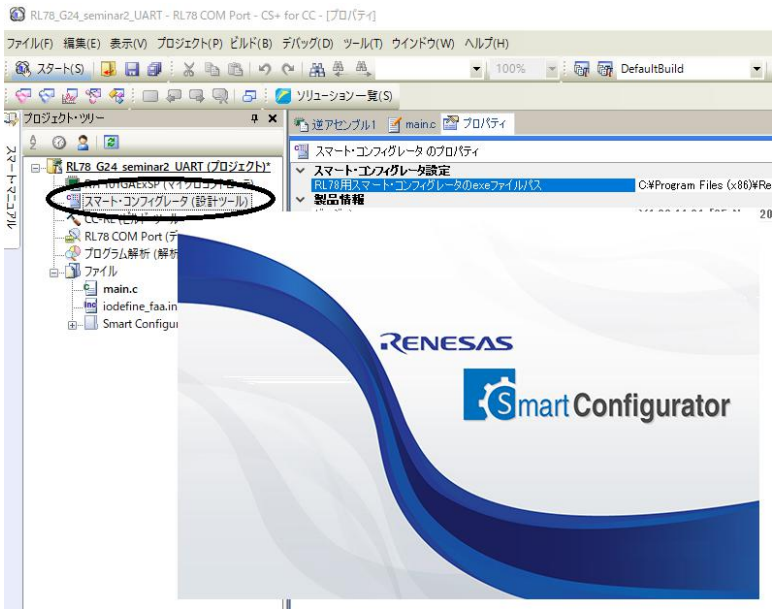


実行し TeraTarm の画面に「START」（オープニングメッセージ）が表示されると、配線もプログラムも正常に動作しています。SW1 や SW2 を押すと、ASCII 文字や漢字が表示されると思います。

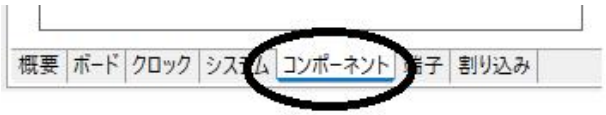


（４）ＳＩＯ（シリアルアイオー）ペリフェラルの初期設定

ＲＬ７８側の初期設定は「スマートコンフイグレータ」で行います。

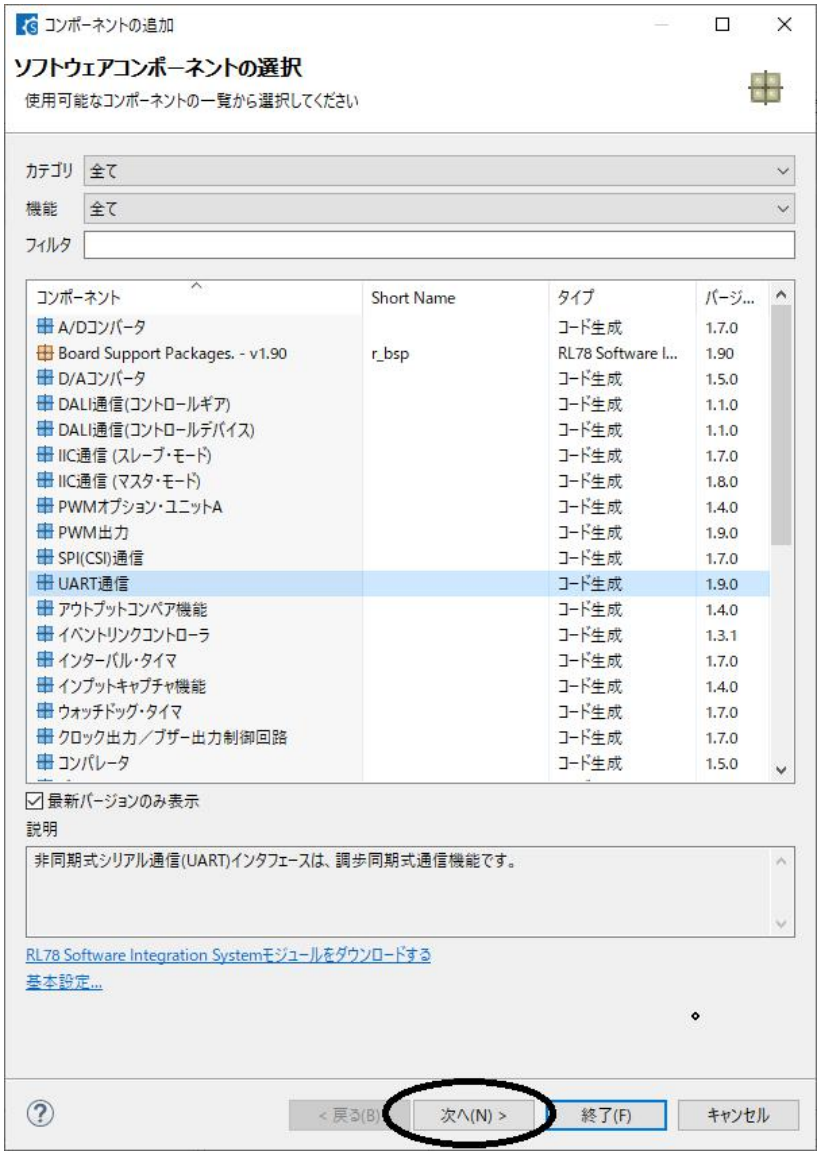


画面が開いたら、コンポーネンツを選択。



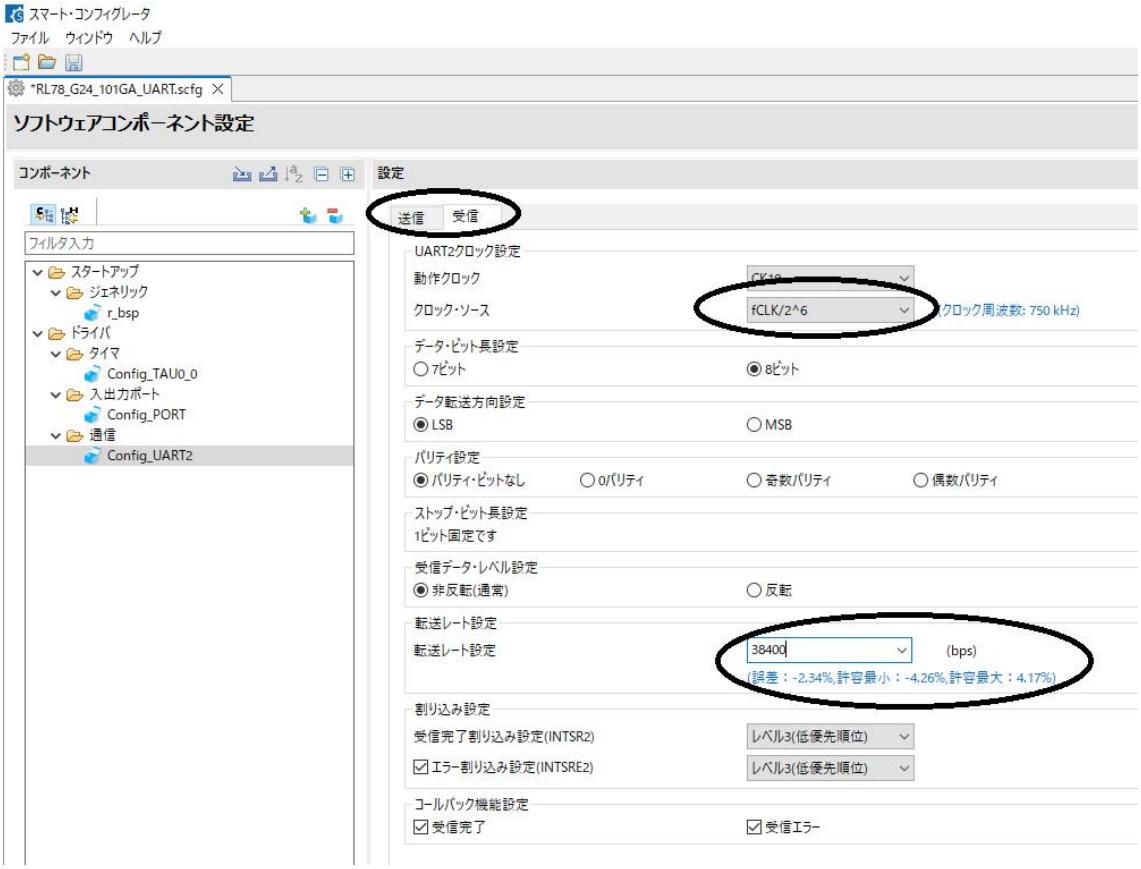
コンポーネンツの追加

UART 通信を選択、次へ

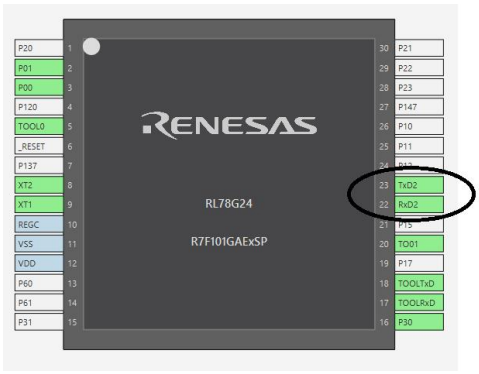


今回は使えるピンの関係でUART2にしました。動作は送受信です。

転送レートは３８４００ｂｐｓにしました。転送レートが赤に表示（規格外）されないようにクロックソース分割を設定します。送受信とも同じに設定します。

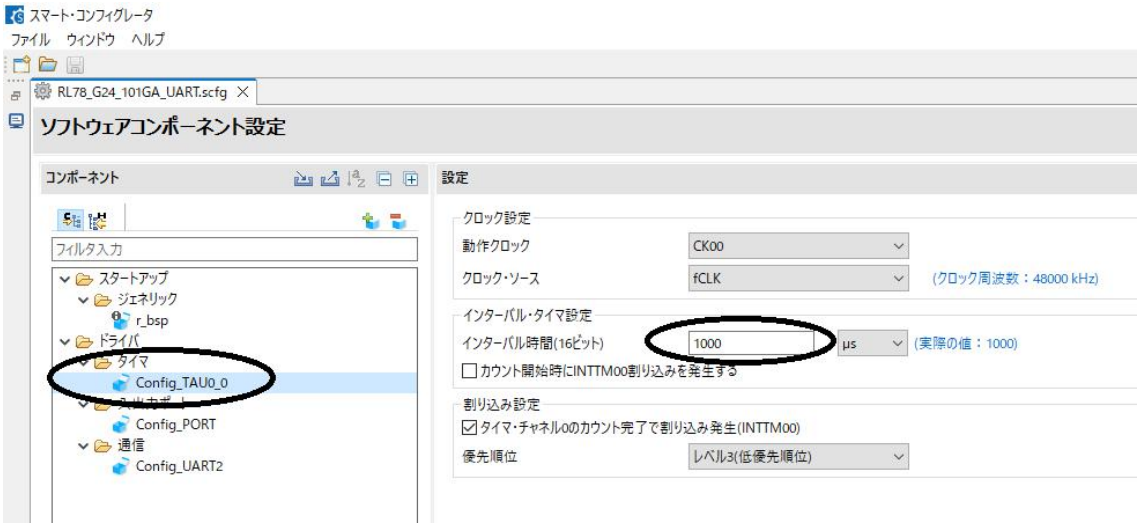


IC の図に23ピンが TXD2、22ピンが RXD2 に指定されたことが分かります。ピンが赤で表示されると、他の機能で既に用途が設定されている可能性がありますので、注意してください。



本プログラムでは上記に加え、正確な時間を作るために定周期割り込みタイマを動作させ

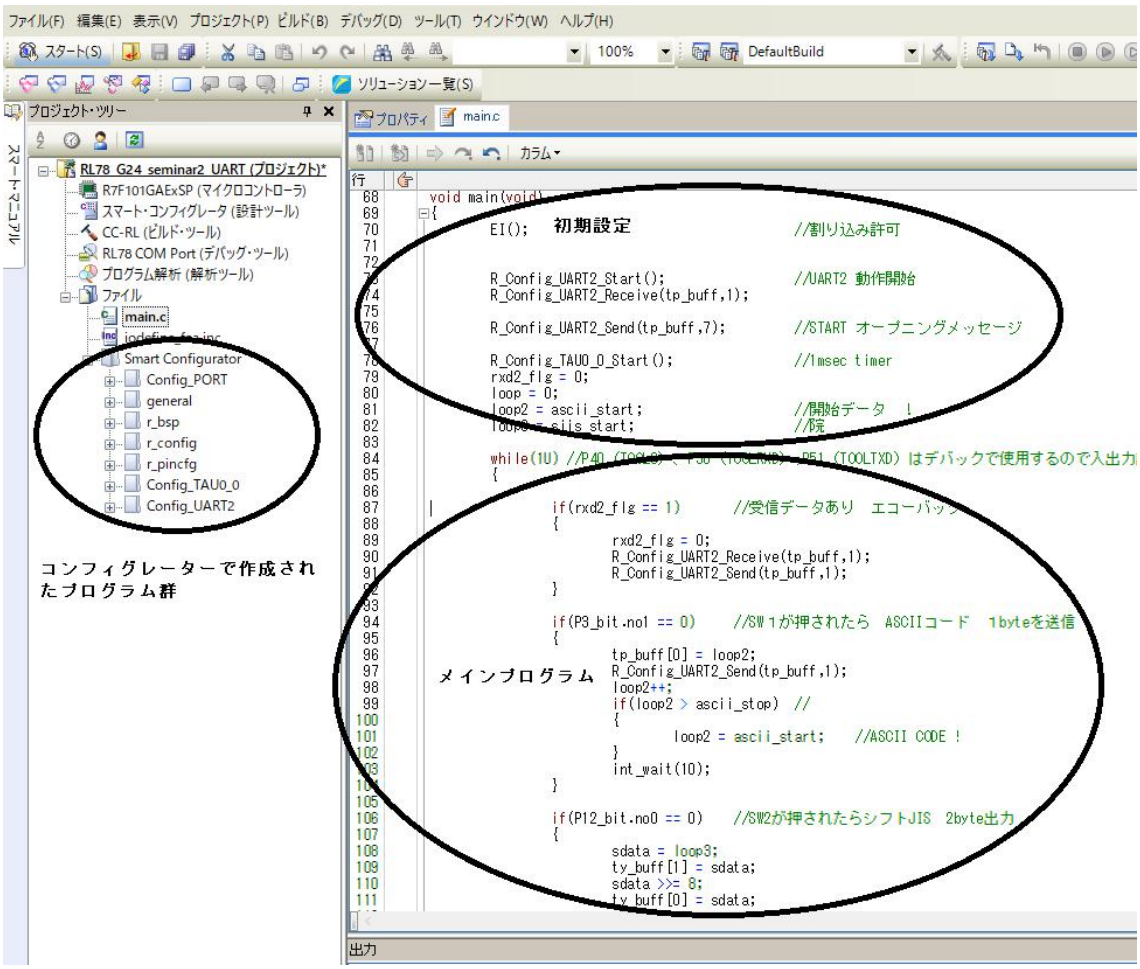
ています。



全部、設定しましたら、最後に「コードの生成」をクリック、エラー無く終了したことを確認します。コードの生成はソースファイルの変更になるので、下記の場合、エラーが出ます。

1. プログラム実行中は変更できません。
2. ファイルを外部エディタで開いている場合は書き換えできません。
3. コードの生成後、必ずコンパイルしないと変更が更新されません。

【 プログラム 】



初期設定の部分から説明します。

```
EI();        ①                          //割り込み許可


R_Config_UART2_Start();      ②          //UART2 動作開始
R_Config_UART2_Receive(tp_buff,1);  ③

R_Config_UART2_Send(tp_buff,7);      ④      //START オープニング メッセージ

R_Config_TAU0_0_Start();                ⑤      //1msec timer
rxd2_flg = 0;                                        ⑥
loop = 0;
loop2 = ascii_start;                          ⑦      //開始データ　！
loop3 = sjis_start;                           ⑧      //院
```

【 解説 】

EI(); ① //割り込み許可

様々な割り込みを使用するので割り込みを許可します。デホルトは DI()割り込み不許可です。

R_Config_UART2_Start(); ② //UART2 動作開始

UART2を使用開始するときに書きます。

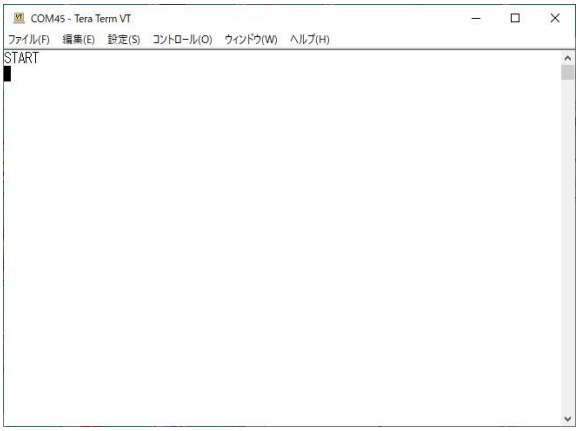
R_Config_UART2_Receive(tp_buff,1); ③

1 文字受信します。必要です。

R_Config_UART2_Send(tp_buff,7); ④ //START オープニング メッセージ

オープニングメッセージを PC 側に出力します。TeraTarm 画面に START の文字が表示されれば、 RL78 プログラム、配線、TeraTarm 設定が正常に動作しています。

```
volatile char tp_buff[20] = "START¥n¥r";
```

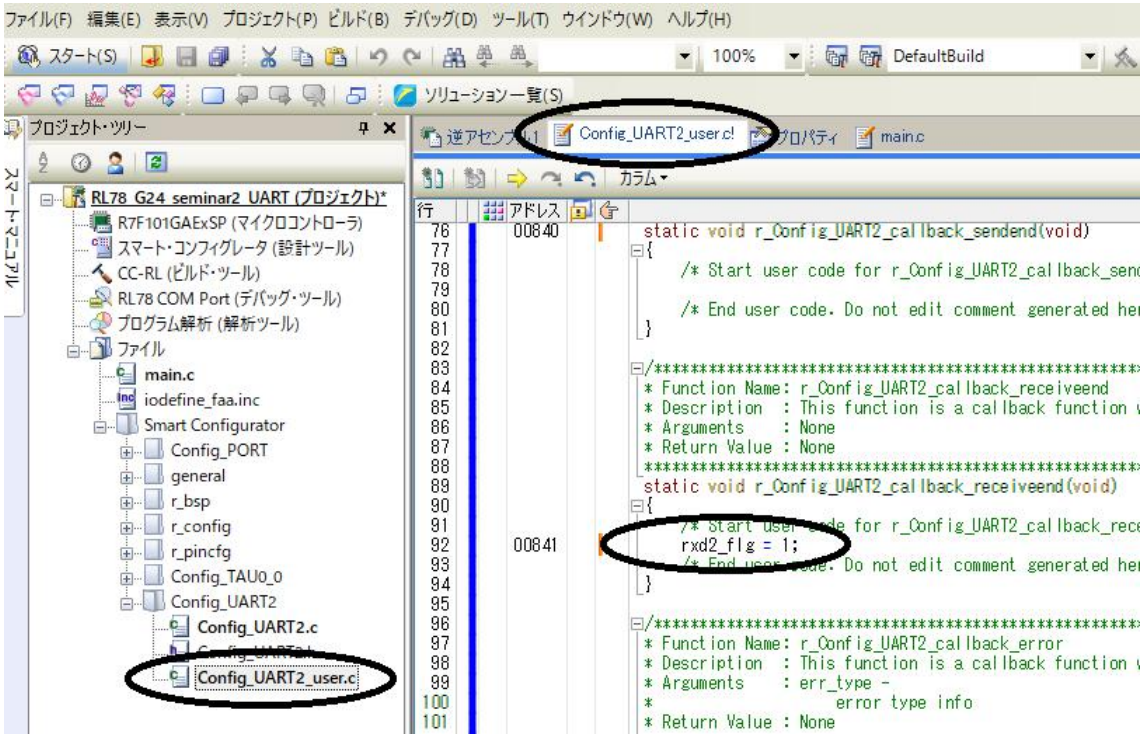


R_Config_TAU0_0_Start(); ⑤ //1msec timer

1msec 定周期割り込みをスタートさせます。

rxd2_flg = 0;　　⑥

UART2が受信データありの場合、割り込みで1にセットされます。　　データを受信すると、Config_UART2_user.c の中にある r_Config_UART2_callback_receiveend()関数が呼ばれますので、そこで1を立てて、メインプログラムで処理しています。



loop2 = ascii_start;　　⑦　　//開始データ　！

SW1 を押すと RL78 からパソコンに ASCII データ、1byte を送信しますが、その初期値 0x21 を設定しています。

loop3 = sjis_start;　　⑧　　//院

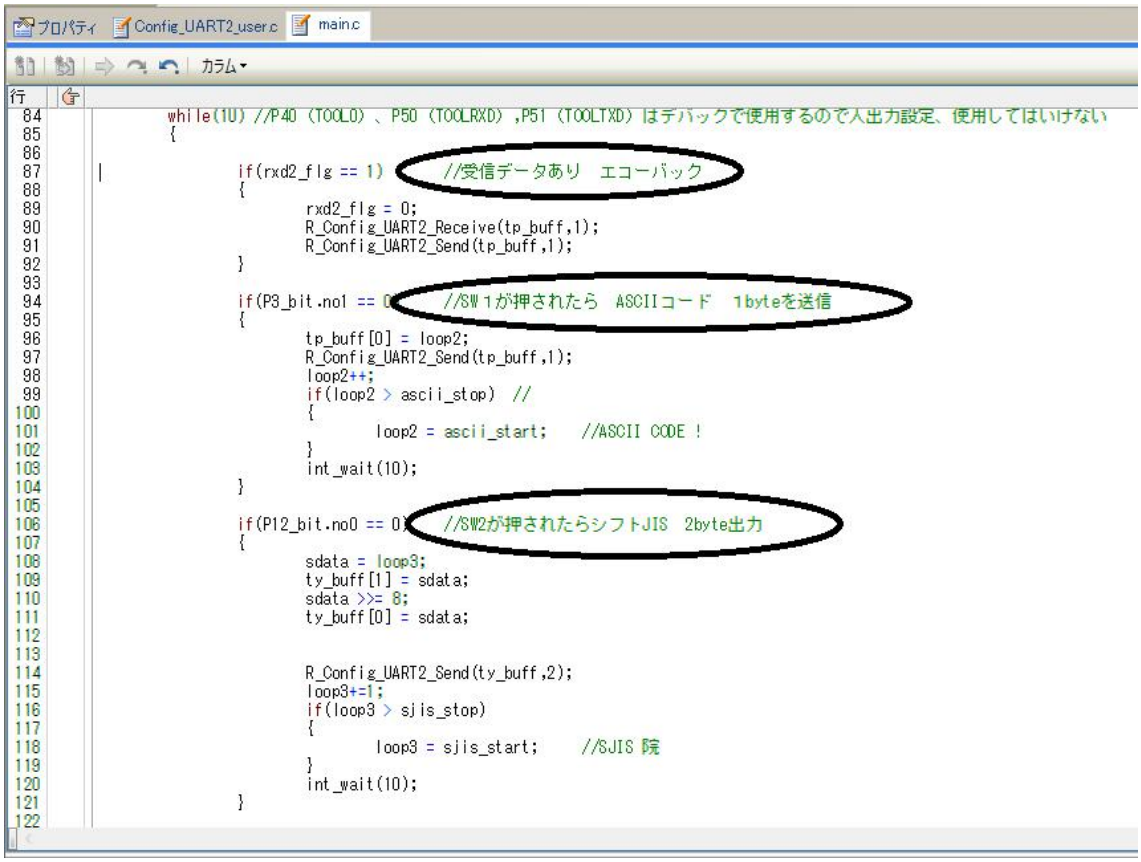
SW2 を押すと漢字が表示されます。これはシフト JIS コード(2byte)を RL78 から TeraTarm に送信すると表示されます。

#define sjis_start　　0x8940

0x8940 は漢字「院」のシフト JIS コードです。

【メインプログラム 】

メインプログラムは PC から RL78 にデータが送信されると、受信し、PC 側に返送するルーチン、SW1 を押すと ASCII 文字を PC 側に送信するルーチン、SW2 を押すとシフト JIS 文字を PC 側に送信するルーチンの3つあります。



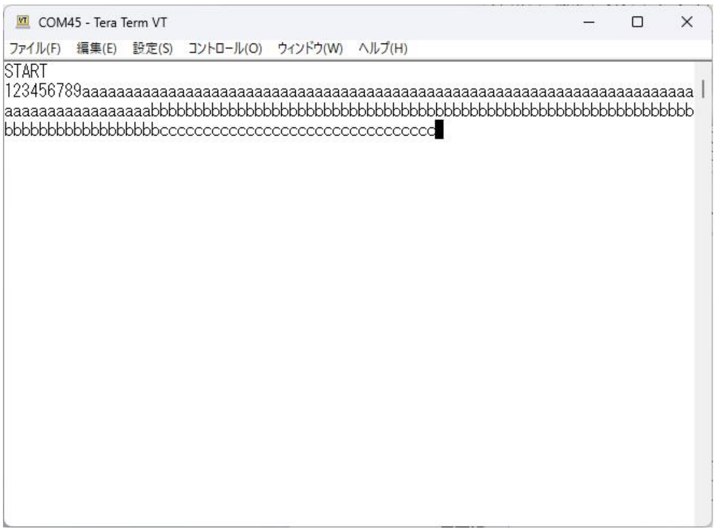
```
プロパティ  Config_UART2_user.c  main.c
カラム
行
 84          while(1U) //P40 (TOOL0) 、P50 (TOOLRXD) ,P51 (TOOLTXD) はデバックで使用するので入出力設定、使用してはいけない
 85          {
 86
 87  |               if(rxd2_flg == 1)     //受信データあり  エコーバック
 88                  {
 89                          rxd2_flg = 0;
 90                          R_Config_UART2_Receive(tp_buff,1);
 91                          R_Config_UART2_Send(tp_buff,1);
 92                  }
 93
 94                  if(P3_bit.no1 == 0      //SW1が押されたら  ASCIIコード  1byteを送信
 95                  {
 96                          tp_buff[0] = loop2;
 97                          R_Config_UART2_Send(tp_buff,1);
 98                          loop2++;
 99                          if(loop2 > ascii_stop)  //
100                          {
101                                  loop2 = ascii_start;    //ASCII CODE !
102                          }
103                          int_wait(10);
104                  }
105
106                 if(P12_bit.no0 == 0)    //SW2が押されたらシフトJIS  2byte出力
107                 {
108                         sdata = loop3;
109                         ty_buff[1] = sdata;
110                         sdata >>= 8;
111                         ty_buff[0] = sdata;
112
113
114                         R_Config_UART2_Send(ty_buff,2);
115                         loop3+=1;
116                         if(loop3 > sjis_stop)
117                         {
118                                 loop3 = sjis_start;     //SJIS 院
119                         }
120                         int_wait(10);
121                 }
122
```

【 解説 】

1．受信したら返信 エコーバック

TeraTarm 画面にカーソルがあるときに、PC のキーボードを押すと押した文字が RL78 側に送信されます。それを RL78 は受信して、返信しますので、画面に押した文字が表示されます。

```
if(rxd2_flg == 1)        ①        //受信データあり エコーバック
```
受信データありのフラグがあれば
```
        {
                rxd2_flg = 0;
                R_Config_UART2_Receive(tp_buff,1);②
```
フラグをクリアして 1 文字受信、
```
                R_Config_UART2_Send(tp_buff,1);③
```
その文字を送信しています。
```
        }
```



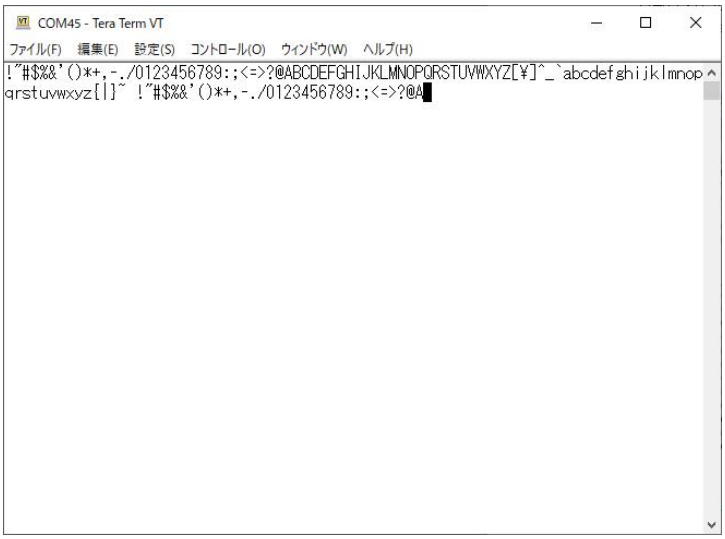
2．SW1 が押されたら ASCII 文字を送信する

```
if(P3_bit.no1 == 0)④ //SW 1 が押されたら  ASCII コード  1 byte を送信
        {
```
SW1のスイッチが押されたら
```
                tp_buff[0] = loop2;⑤
```
loop2の値を1文字
```
                R_Config_UART2_Send(tp_buff,1);⑥
```
送信しています。
```
                loop2++;
                if(loop2 > ascii_stop)  //⑦
                {
                        loop2 = ascii_start;      //ASCII CODE !
```

```
                    }
```

loop2はascii__startからascii__stopまで変わります。

画面初めに 0x21 に対応する文字「！」が表示されています。順次+1 されて対応する ASCII コードが表示されます。



3. SW2 が押されたらシフト JIS　2byte　出力

SW2 が押されたら TeraTarm 画面に漢字が表示されます。これはシフト JIS コードを RL78 から TeraTerm 側に2byte送信しているためです。

```
if(P12_bit.no0 == 0)    ⑧//SW2 が押されたらシフト JIS　2byte 出力
        {
```

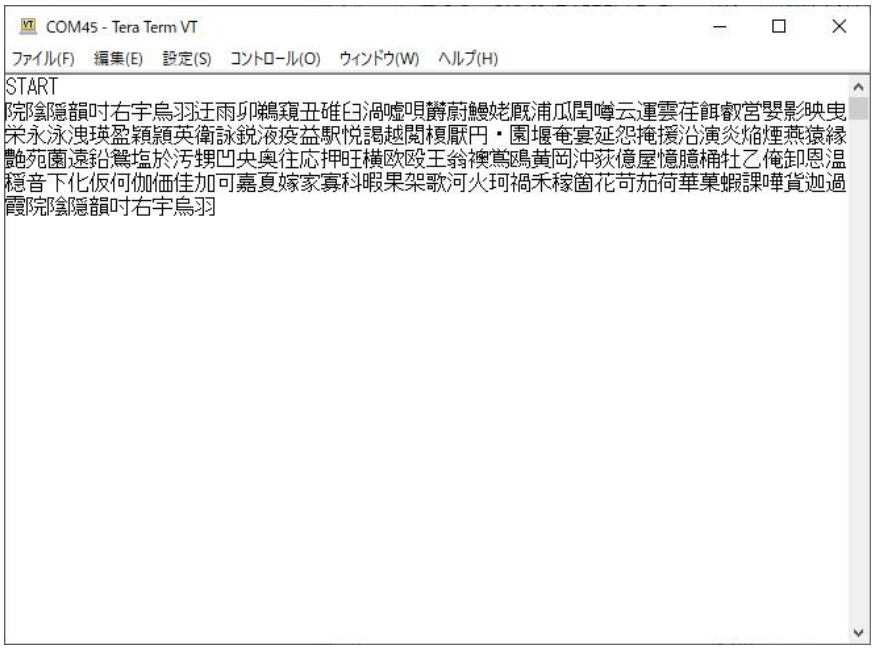
SW2 が押されたらloop3の値、16ビットを

```
    sdata = loop3;
                ty_buff[1] = sdata;
                sdata >>= 8;
                ty_buff[0] = sdata;

                R_Config_UART2_Send(ty_buff,2);    ⑨
```

2 文字送信しています。
sjis__startからsjis__stopまで変わります。

```
loop3+=1;
                if(loop3 > sjis_stop)
                {
                        loop3 = sjis_start;        //SJIS 院
                }
```

```
                int_wait(10);
            }
        }
```



【演習】

1.

SW1 を押すと　1 ガオサレマシタ　と表示する
SW2 を押すと　一網打尽　と表示するソフト　を製作してみて下さい。

2.

1. ではキーを押している間、連続して文字が出力されますが、1 回押したら1単位出力されるように変更してください。

回答例は演習ホルダの中にあります。

3. A／D変換

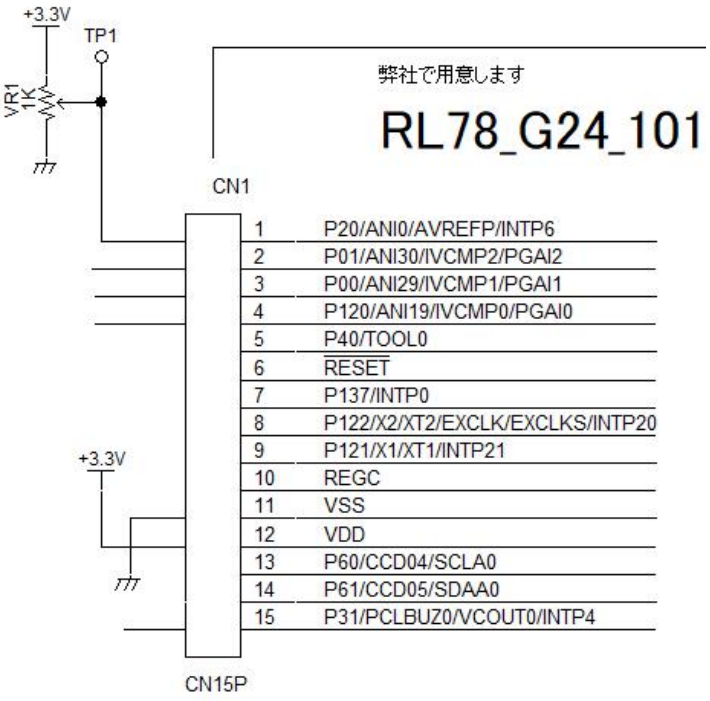
【 動作概要 】

VR1の0－3．3Vのアナログ値をデジタル値に変換し、USBに出力します。

■サンプルプログラム名　RL78__G24__seminar3__AD

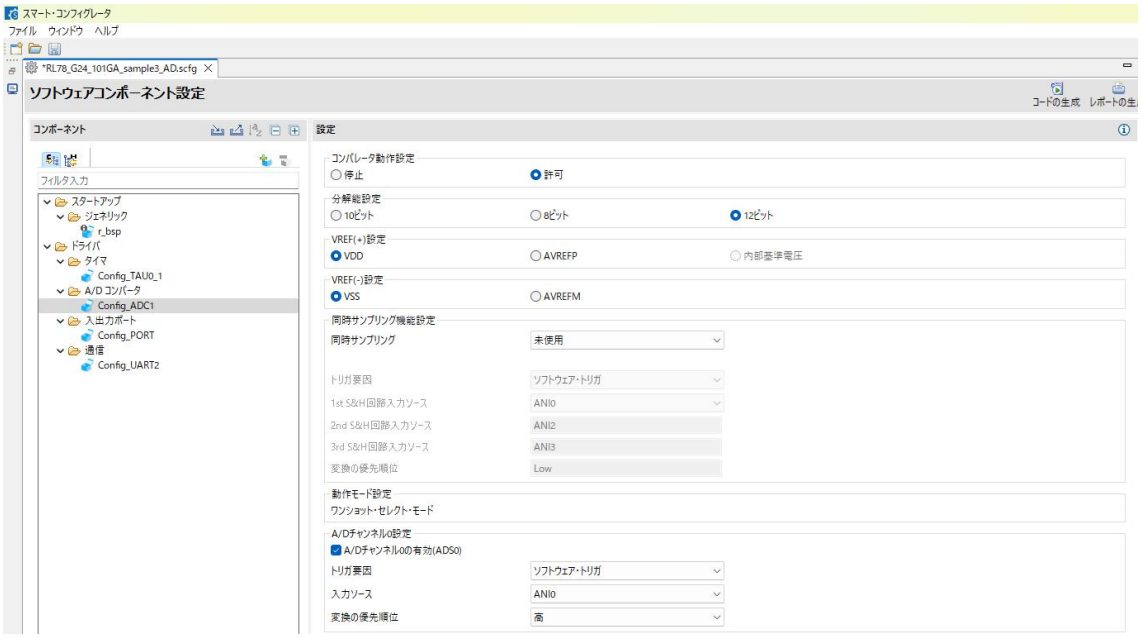
【 ボリウム 】

プログラム動作確認のために、回路図のようにボリウムが配線されています



【　A／D（エーデーコンバータ）ペリフェラルの設定　】

コンフイグレーターで今回は動作クロックを48MHzにして、AD変換はアドバンストモードにしました。



【 プログラム 】

```
void main(void)
{
        rxd2_flg = 0;

        EI();                                           //割り込み許可

        R_Config_ADC1_Set_OperationOn();①       //AD 動作開始
        R_Config_ADC1_Start();                          //AD スタート

        R_Config_UART2_Start();                         //UART2  動作開始
        R_Config_UART2_Receive(tp_buff,1);

        R_Config_UART2_Send(tp_buff,10);        //START オープニングメッセージ
        while(rxd2_flg == 0)                    //文字列送信終了まち
        ;

        R_Config_TAU0_1_Start();                //1msec timer
        rxd2_flg = 0;

        while(1U)  //P40（TOOL0）、P50（TOOLRXD）,51（TOOLTXD）はデバックで使用するので入出力設定、使用してはいけない
        {

                ADTRSWT = 1;②                          //AD 変換開始
                while(ADINTST0S==0)                     //AD 変換成功ステータス 1
                ADINTST0S = 0;
                R_Config_ADC1_ADS0_Get_Result_12bit(ad0);

                sdata = ad0[0];
                fdata1 = sdata/(4095/3.3);③     //4095→3.30 に変換
                sprintf(cvdat,"AD0 = %.2fV¥n¥r",fdata1);④  //ASCII 変換
                R_Config_UART2_Send(cvdat,sizeof(cvdat));       //データを USB 送信

                P3_bit.no0 = 1;
                int_wait(500);
                P3_bit.no0 = 0;
                int_wait(500);

        }

}
```

【 プログラムの注意点 】

```
R_Config_ADC1_Set_OperationOn();①      //AD 動作開始
R_Config_ADC1_Start();                                    //AD スタート
R_Config_ADC1_Set_OperationOn();       //Set Operation On
```

①アドバンスト変換モードの場合、AD 開始に 2 行必要になるようです。

```
ADTRSWT = 1;②                         //AD 変換開始
        while(ADINTST0S==0)                    //AD 変換成功ステータス 1
        ADINTST0S = 0;
        R_Config_ADC1_ADS0_Get_Result_12bit(ad0);
```

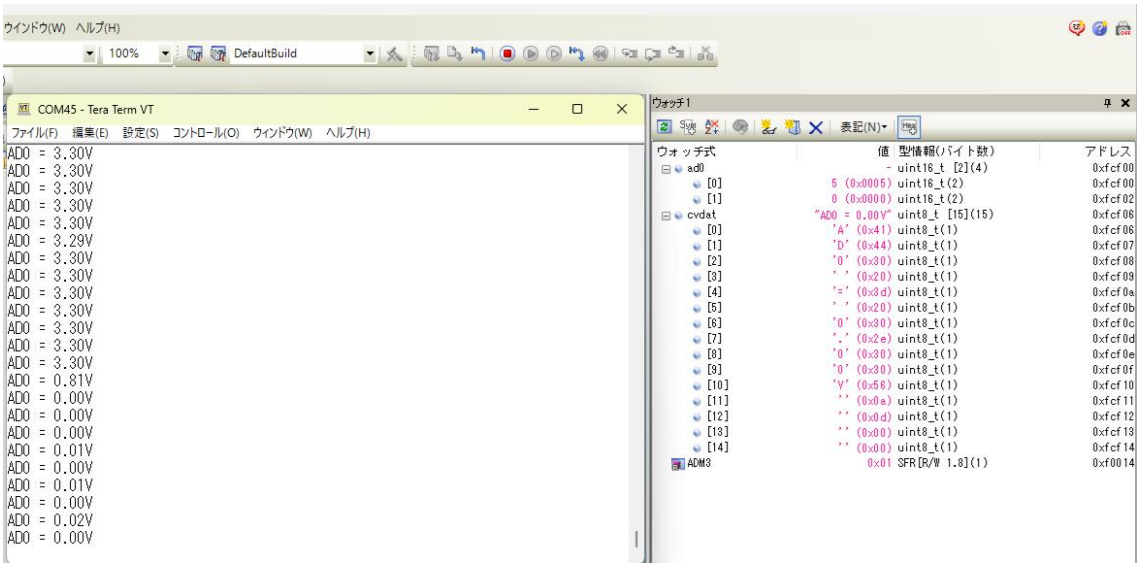
②AD 変換の終了フラグもノーマルモードとは異なります。

```
sdata = ad0[0];
        fdata1 = sdata/(4095/3.3);③    //4095→3.30 に変換
```

③読み込んだデータ（0－4095）を0－3．3Vに変換します。

```
        sprintf(cvdat,"AD0 = %.2fV¥n¥r",fdata1);④      //ASCII 変換
        R_Config_UART2_Send(cvdat,sizeof(cvdat));      //データを USB 送信
```

下はデータを受信しているTeraTerm画面（左）とウオッチ窓（左）のADデータ　ad0[0]　とASCII変換後　cvdat[]　データ。



【　演習　】

1．AD入力が2V以上でD1をONさせて下さい。2V未満はOFFにしてください。

2．AD 入力が2V以上でD1　ON，1．5V以下でOFF　0．5V幅のヒステリシスを持たせます。実際の負荷ではよく使用される方法です。ヒステリシスが無いと設置値の際（きわ）でON／OFFを繰り返し、品質の悪い制御になる場合があります。

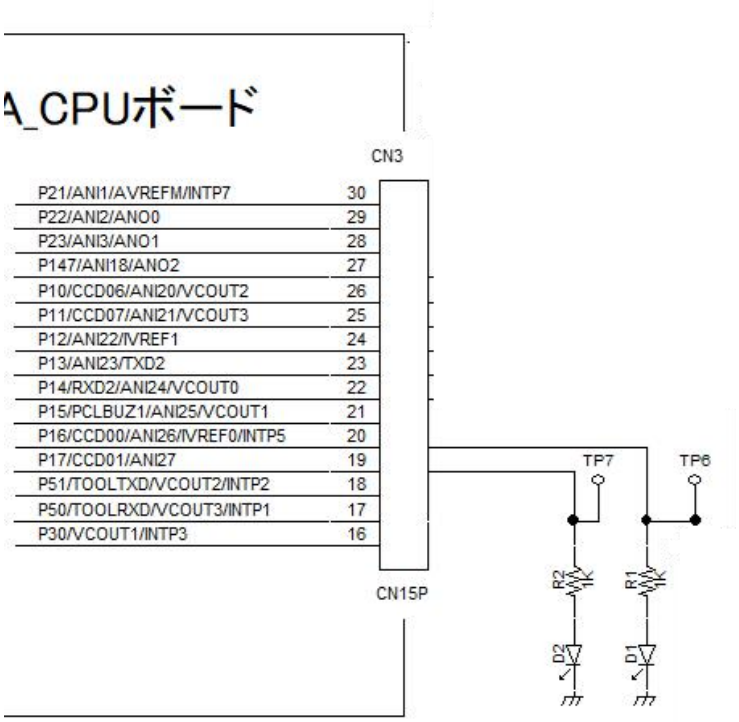
回答例は演習ホルダにあります。

4. PWM制御

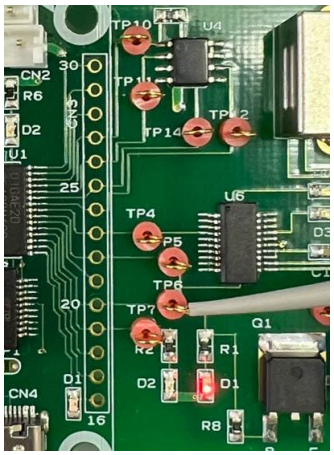
【 動作概要 】

■サンプルプログラム名　RL78__G24__seminar4__PWM

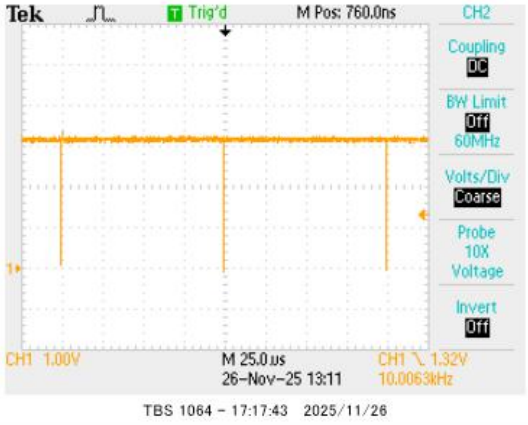
P16の出力を PWM 出力に設定します。PWM周期は 100μsec、分解能は 1/4800 です。変数を 10msec 毎に+1 して 4800 以上で0にし、繰り返します。負荷の LED の輝度が PWM のデューティにより変化する様子が目視出来ます。オシロスコープがあればTP6信号を観測することで周期の変化が目視出来ます。



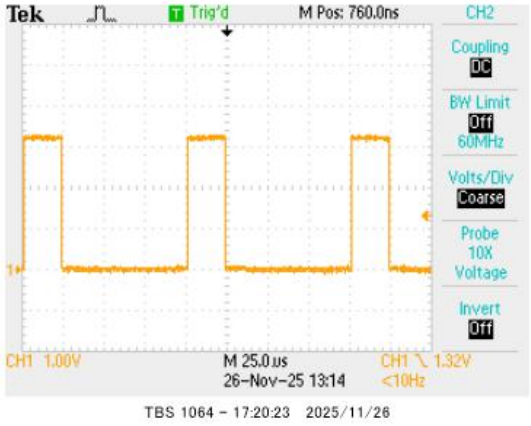
TP6 の波形を観測すると PWM 波形と LD1 の明るさが同期していることが分かると思います。



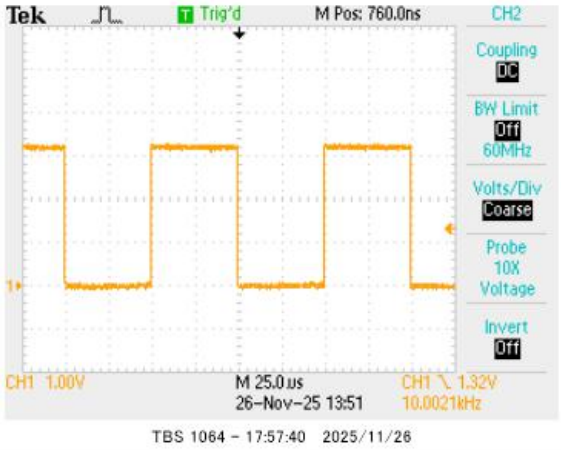
H 幅がほぼ100%、3.3V の直流と等価で一番 LD1 が明るくなります。



H 幅が 25%で LED は暗い

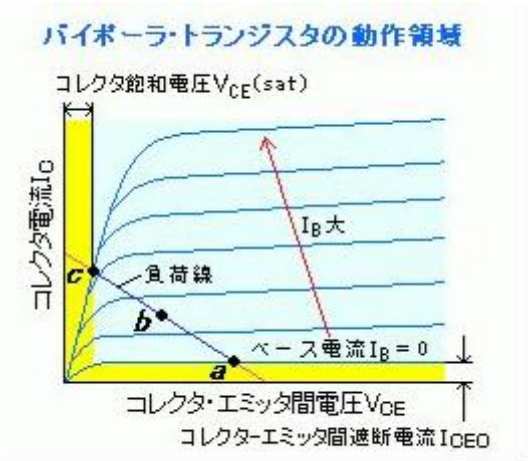


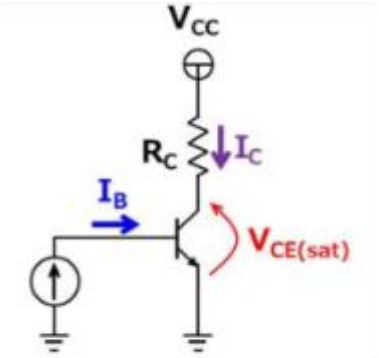
H幅50%、L 幅50%で LD1 は半分消えて、半分点灯しています。



【 PWM が現代に多用される理由 】

PWM が現代に多用される理由 を AI で検索すると以下の内容が出てきました。
●高い電力効率



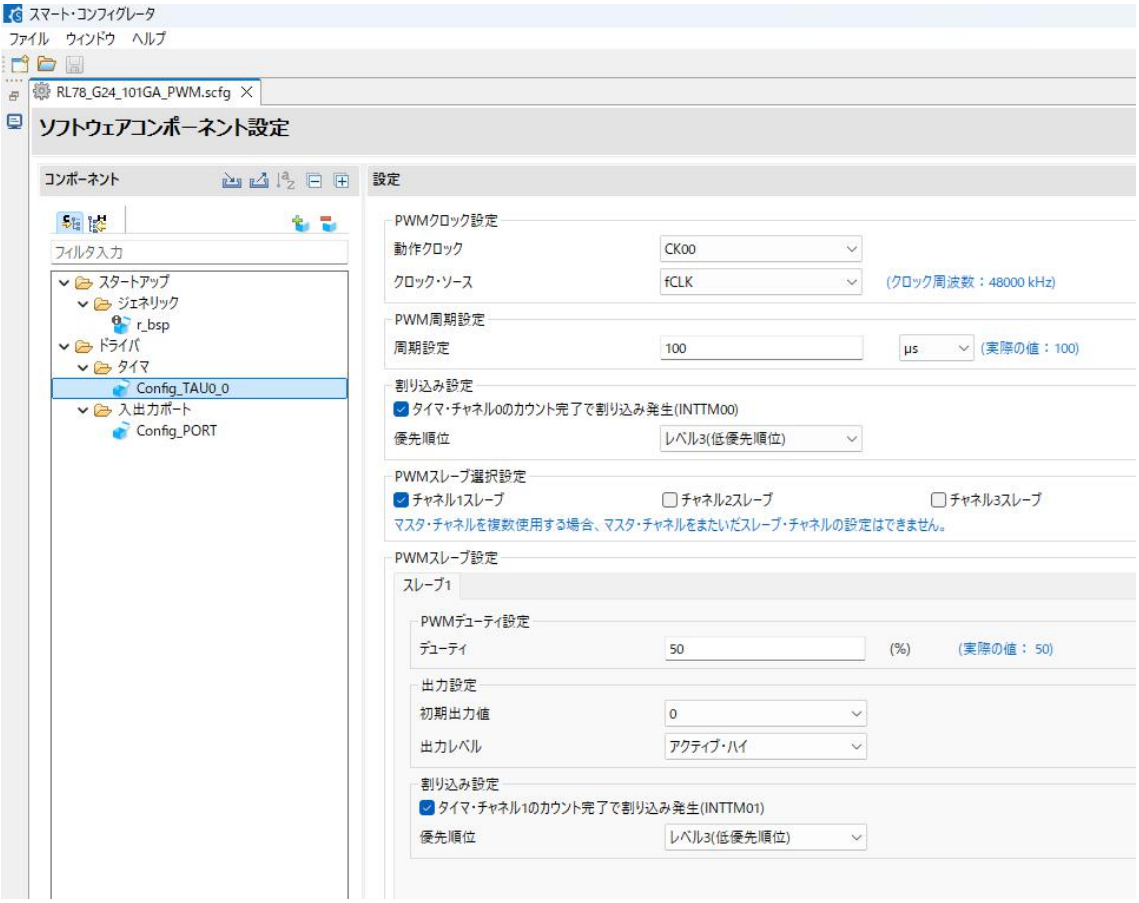


飽和領域、活性領域、遮断領域 PWM はトランジスタの飽和領域（トランジスタ ON）、と遮断領域（トランジスタ OFF）しか使わないので、活性領域で信号を伝達する機構より効率が良い。

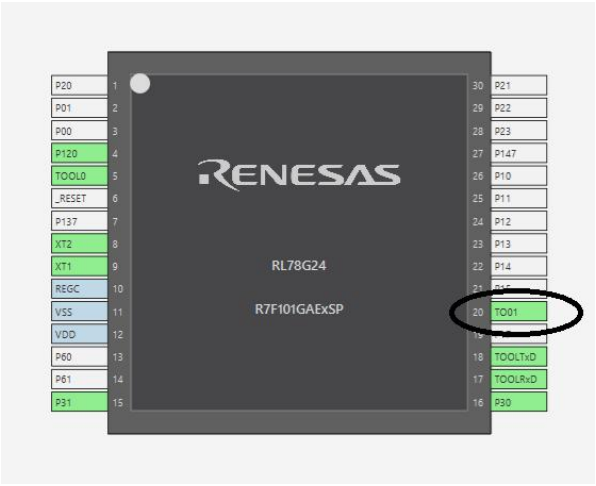
●発熱の抑制
上記理由。
●デジタル制御との親和性
マイコンで扱いやすい。
●精密な制御
分解能を増やすのが容易。
●低コスト
マイコンなので。
●広範な応用分野

【 スマートコンフィグレータの設定 】

PWM の設定はクロック48MHz、周期 100μsec にしたので、分解能は 100μsec／（1／48MHz）＝4800になります。デューティ50%の値で2400。



この周期１００μsec は、時間計測用タイマーとしても動作させています。



【 プログラム 】

```
void main(void)
{
        R_Config_TAU0_0_Start();        //PWM スタート
        pdata = 2400;                   //2400 で 50%、50%Duty
```

```
        EI();                                   //割り込み許可

//      while(SW1 == 1)
//              ;


                while(1U)               //P40（TOOL0）、P50（TOOLRXD）,51（TOOLTXD）はデバッ
クで使用するので入出力設定、使用してはいけない
                {

                        TDR01 = pdata;
                        int_wait(10);     //1msec 毎に
                        pdata++;    //pdata を+1 して
                        if(pdata > 4800)//4800 以上で
                        {
                                pdata = 0;  //0 にクリア
                                TDR01 = pdata;
                                int_wait(20000);//2 秒待つ
                        }
                }

}
```

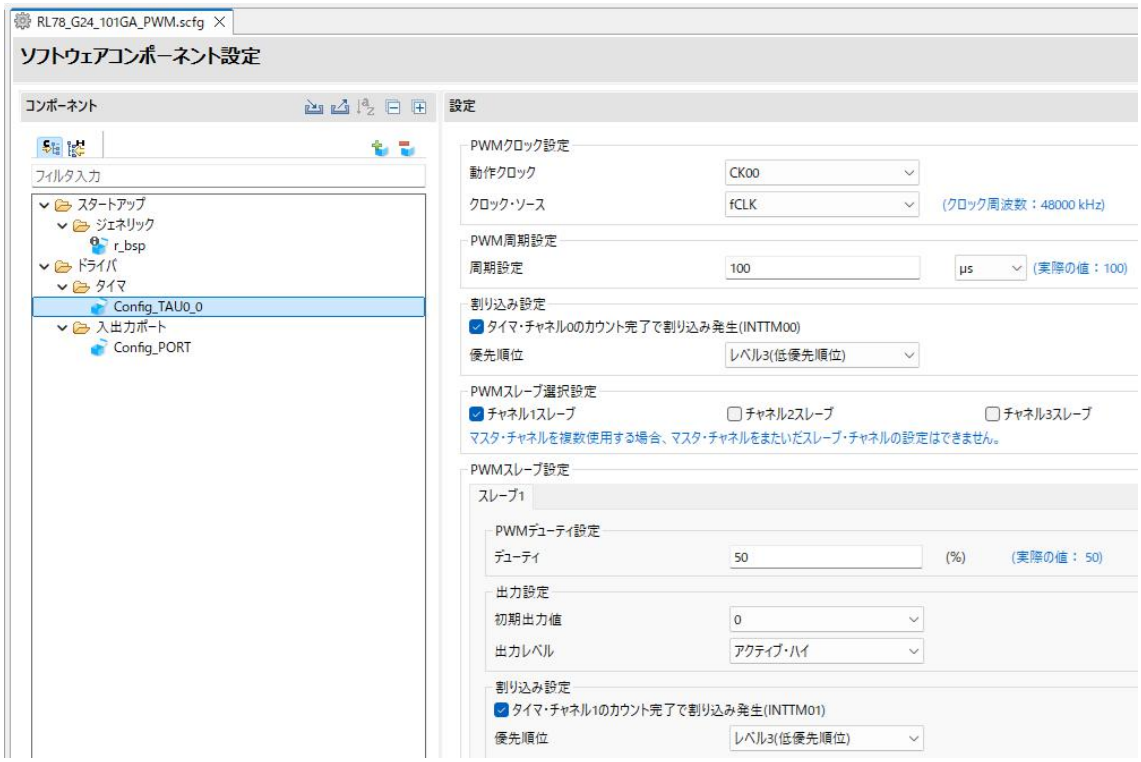
コメントに書いてあることで、内容的には理解いただけるかと思います。

【　ｉｎｔ＿＿ｗａｉｔの動作　】

int_wait(20000);//2 秒待つ　について少し解説します。

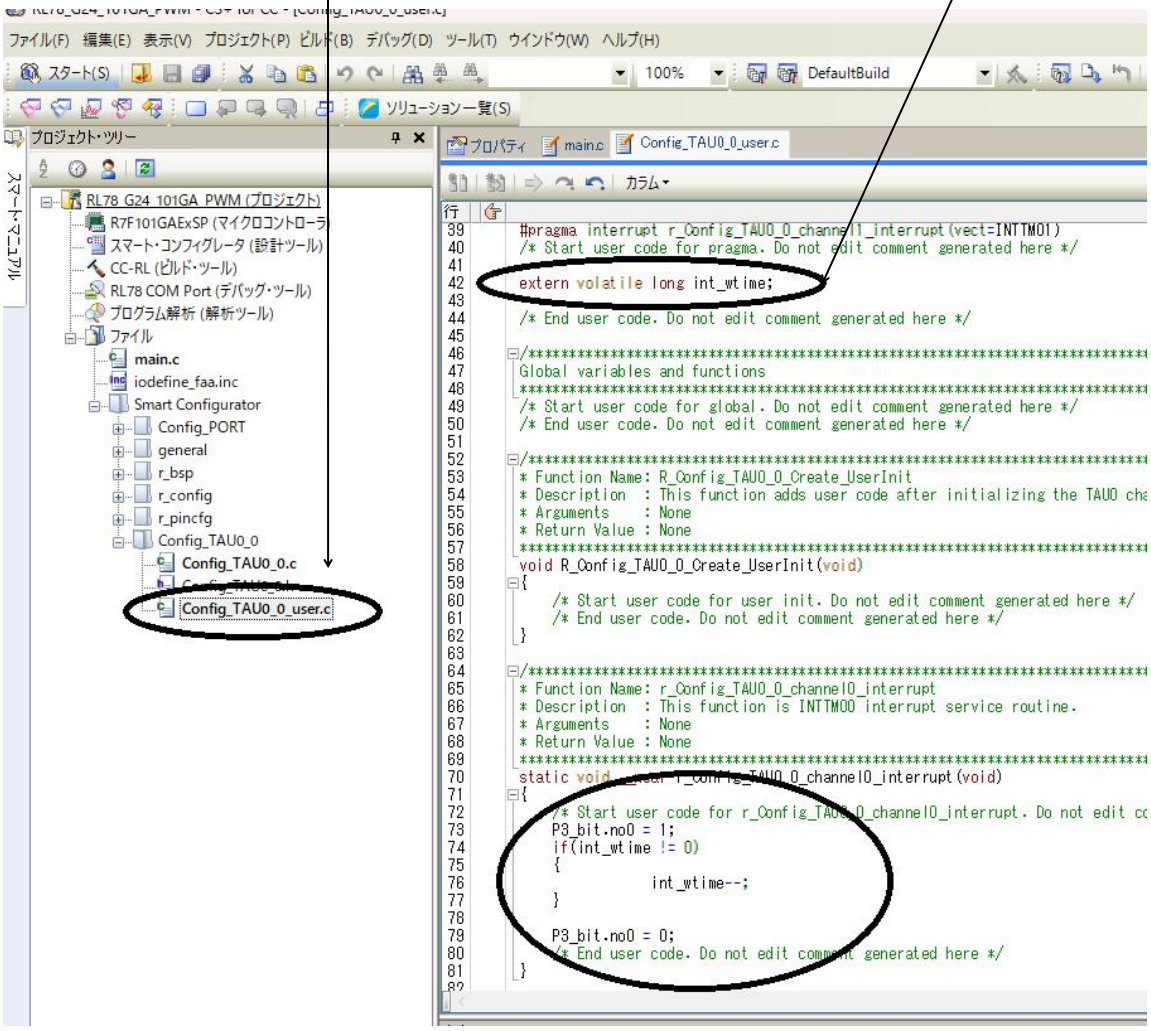
```
volatile long int_wtime;                //int__wtime  はメインのここで定義されている。
volatile uint16_t pdata;

void int_wait(long wtime)
 {
          int_wtime = wtime;

        while(int_wtime != 0)           //PWM 周期割り込みでデクリメントせさている
        ;
 }
```

プログラムは引数ｌｏｎｇ　ｗｔｉｍｅの設定を int_wtime に入れて、それが０になるまでループしている、と言う形ですが、int_wtime はどこで操作されているのでしょうか？
スマートコンフイグレータ→コンポーネント設定　で　Config__TAU0__０でＰＷＭ周期　１００μsec に設定していますが、割り込み設定→タイマ・チャネル０のカウント完了で割り込み発生の設定により、この周期毎に割り込みが入ります。


RL78_G24_101GA_PWM.scfg
ソフトウェアコンポーネント設定
コンポーネント
設定
フィルタ入力
スタートアップ
ジェネリック
r_bsp
ドライバ
タイマ
Config_TAU0_0
入出力ポート
Config_PORT
PWMクロック設定
動作クロック
CK00
クロック・ソース
fCLK
(クロック周波数：48000 kHz)
PWM周期設定
周期設定
100
µs
(実際の値：100)
割り込み設定
タイマ・チャネル0のカウント完了で割り込み発生(INTTM00)
優先順位
レベル3(低優先順位)
PWMスレーブ選択設定
チャネル1スレーブ
チャネル2スレーブ
チャネル3スレーブ
マスタ・チャネルを複数使用する場合、マスタ・チャネルをまたいだスレーブ・チャネルの設定はできません。
PWMスレーブ設定
スレーブ1
PWMデューティ設定
デューティ
50
(%)
(実際の値： 50)
出力設定
初期出力値
0
出力レベル
アクティブ・ハイ
割り込み設定
タイマ・チャネル1のカウント完了で割り込み発生(INTTM01)
優先順位
レベル3(低優先順位)

int_wtime の減算は Config__TAU0__user. cの中に書いてあります。

long　int_wtime　の宣言はmain. cで行われているので、外部で宣言されているとexternを付けます。これにより、mainで使う int_wtime と、ここで使う int_wtime が共有化された変数となります。



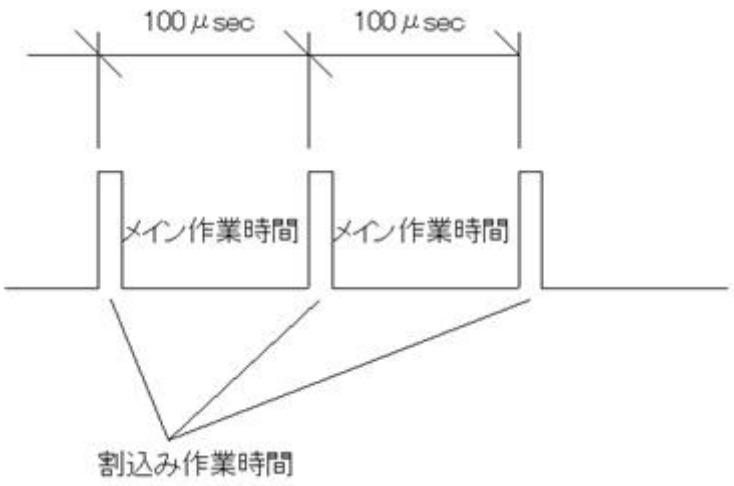
100μsec の割り込みのたびにこの関数が呼ばれます。int_time が０でなければー１し、０になるまで繰り返します。割り込みの精度はこの場合、高速オンチップオシレータの精度で、±1%なので、かなり正確な時間が作れることが分かります。

43.2.2　オンチップ・オシレータ特性

（TA = –40～+105°C, 1.6 V≦VDD≦5.5 V, VSS = 0 V）

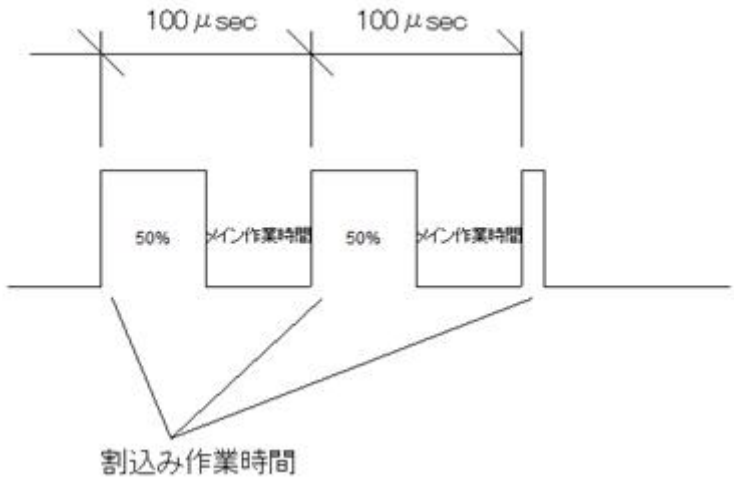
項目	略号	条件			Min.	Typ.	Max.	単位
高速オンチップ・オシレータ・クロック周波数	fIH				1		48	MHz
高速オンチップ・オシレータ・クロック周波数精度注1		HIPREC = 1	+85～+105°C	1.8 V≦VDD≦5.5 V	–1.5		+1.5	%
				1.6 V≦VDD≦5.5 V	–6.0		+6.0	%
			–20～+85°C	1.8 V≦VDD≦5.5 V	–1.0		+1.0	%
				1.6 V≦VDD≦5.5 V	–5.0		+5.0	%
			–40～–20°C	1.8 V≦VDD≦5.5 V	–1.5		+1.5	%
				1.6 V≦VDD≦5.5 V	–5.5		+5.5	%
		HIPREC = 0注4			–15		0	%

また、時間軸を見ると、プログラムは以下のように動作しているイメージです。これは長い時間軸で見れば２つのプログラムが同時に動いているように見えます。「マルチタスク」です。

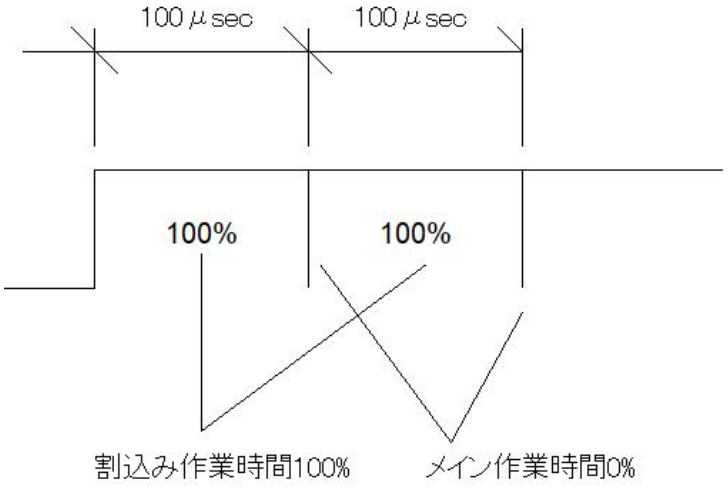


例えば、割込みでの作業時間とメイン関数での作業時間を 50%ずつにしたとすると、等価的に動作速度は1／2になりますが、メイン関数と割込みで異なるプログラムを2本、動作させることが出来ます。また、マイコンの動作速度を倍に出来れば、メイン関数と割込みで前と同じ速度で動作することになります。このようなマルチタスクの方式をプリエンプティブ方式といいます。

特にOS（オペレーティングシステム）を使わなくても、このような方法で複数のプログラムを見掛け上、複数動かせることは可能です。



時々、コンサートチケットの予約などでシステム障害が発生して、コンピュータが正常に動かない話がありますが、これは以下のように割込み処理が１００%を超えるとメイン関数が動作出来なくなるために発生する場合があります。予想される割込みの数、処理時間を上回った場合、このような障害が発生します。

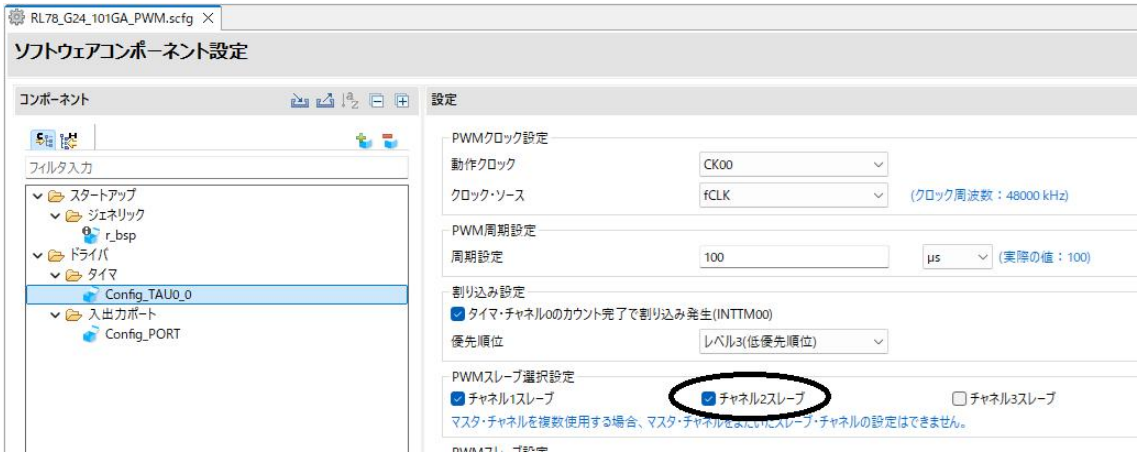


【 演習 】

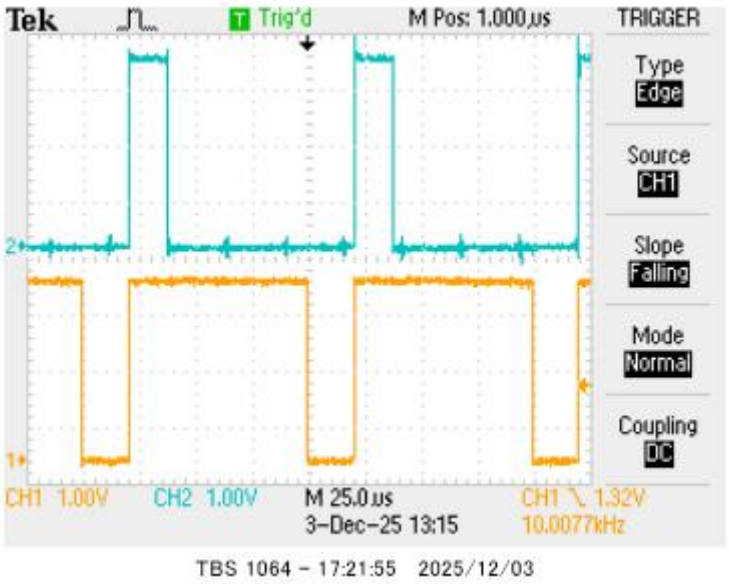
CH2 スレーブも有効にして P17 にもPWM波形を出力し、CH1 とは逆にデータをマイナスしていって0になったら 4800 を設定して繰り返す動作を作ってみて下さい。

【 ヒント 】

スマートコンフイグレータの Config_TAU0_0 のチャンネル 2 スレーブを ON し、「コード生成」します。

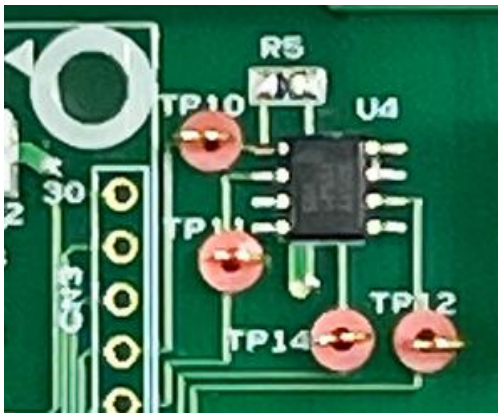


オシロスコープで見た場合、TP7（上　ブルー）が時間と共に L 幅が増え、TP6（下　オレンジ）が時間と共に H 幅が増えれば正解です。



回答例は演習ホルダにあります。

5. FRAMの読み書き



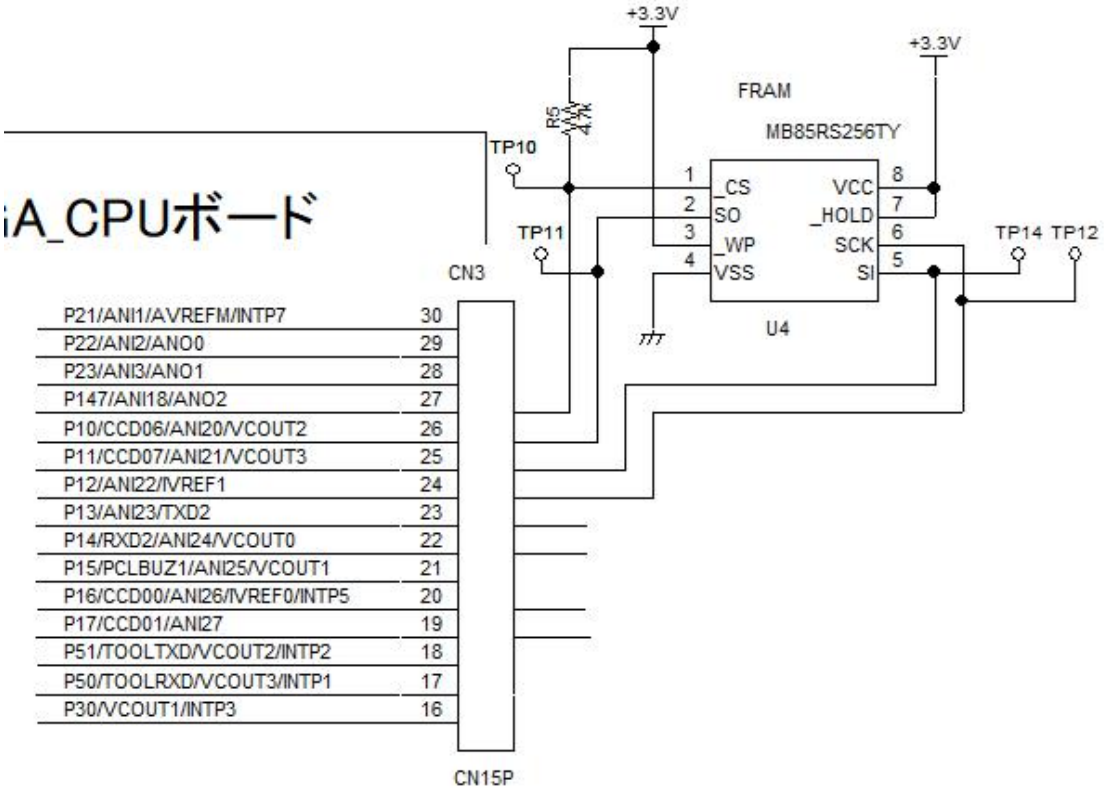
【 動作概要 】

■サンプルプログラム名　RL78__G24__seminar4__FRAM

SPIインターフェイス（Serial Peripheral Interface）を使って、次世代のメモリであるFRAM（強誘電体メモリ）の読み書きを行います。FRAMは書き換え速度が高速、電源を切っても内容が保持される特徴を持ちます。

【 配線 】

RL78から 4 本の信号線と電源、GND線を配線してあります。



【サンプルプログラム】

```
#include "eeprom25256_rl78.c"
```

サンプルプログラムは上記関数を使います。元々、eeprom 用の関数がそのまま使えます。

＞EEPROM（Electrically Erasable Programmable Read-Only Memory）は、電気的に消去・書き換えが可能な不揮発性メモリ

FRAM は EEPROM や FLASH に比べて書換サイクルが早く、データの消去作業が要らないというメリットがありますが、コストの面で採用が進まない状況でした。プロセスの微細化に伴い、書き込みに高電圧を必要としない FRAM に注目が集まり、今後、組込みマイコンのフラッシュ ROM も FRAM に置き換えられていくようです。

FRAMと他メモリの特性比較

		FRAM	EEPROM	Flash Memory
データ保持期間		不揮発	不揮発	不揮発
データ保持		10年	10年	10年
セル構成		2T／2C 1T／1C	2T	1T
読み出し時間		180ns	200ns	<120ns
書き込み電圧		2V～5V	14V	9V
データの書き換え	書き換え方法	重ね書き	消去あるいは書き込み	書き込みと消去の組み合わせ
	書き換えサイクル	180ns	10ms (byte 単位)	1s (sector 単位) ※2
データの消去動作		不要	必要	必要
書き換え回数		$>10^{10}$ ※1	10^5	10^5
データ保持電流		不要	不要	不要
待機時電流		10μA	20μA	5μA
読出動作時電流（最大）		10mA	5mA	12mA
書込動作時電流（最大）		10mA	8mA	35mA

上記は典型値であり、実際は製品により異なります。
※1. FRAM 書き換え回数：読み出しの場合は、破壊読み出しになるため、読み出しと再書き込みの合計回数。
※2. Flash Memory 書き換えサイクル：チップ内部でのプリプログラミング時間を除く。

ライブラリは 4 本の線の定義をマイコンに合わせて変えるだけで、他のＣＰＵでも使用出来ます。専用のハードを必要とせず、ＳＰＩインターフェイスをソフトウエアで実現しています。

```
/*
        EEPROM 25LC256 用ヘッダ

        RX71m

        2016.8.30
```

```
            RL78/G24  2025.09.01
*/

//RL78/G24
#define     sk_l        P1_bit.no2 = 0
#define     sk_h        P1_bit.no2 = 1
#define      si_l       P1_bit.no1 = 0
#define     si_h        P1_bit.no1 = 1
#define     cs_l        P14_bit.no7 = 0
#define     cs_h        P14_bit.no7 = 1

#define IN_PORT P1_bit.no0

/*
//RX230 用
#define     sk_l        PORT1.PODR.BIT.B6 = 0
#define     sk_h        PORT1.PODR.BIT.B6 = 1
#define si_l            PORT1.PODR.BIT.B7 = 0
#define si_h            PORT1.PODR.BIT.B7 = 1
#define cs_l            PORT1.PODR.BIT.B5 = 0
#define cs_h            PORT1.PODR.BIT.B5 = 1

#define IN_PORT PORT1.PIDR.BIT.B4
*/
```

【メインプログラム】

```
void main(void)
{
                    eep_init();                     //FRAM 初期化


                    sdata1 = eep_rd16(0);①

                    if(sdata1 == 0)                 ②          //始めだけ初期化　フラッシュ E2PROM と
逆！
                    {
                            eep_wr16(0,1234);       //0 番地に 2byte データを書き込み
                            eep_wr16(2,5678);       //2 番地に 2byte データを書き込み
                            eep_wr16(4,0x9abc);     //4 番地に 2byte データを書き込み
                            eep_wr16(6,0xdef0);     //6 番地に 2byte データを書き込み
                            eep_wr32(8,12345678);           //8 番地に 4byte データを書き込み
                    }
```

```
sdata1 = eep_rd16(0);          ③//0 番地データを 2byte 読み込み
sdata2 = eep_rd16(2);          //2 番地データを 2byte 読み込み
sdata3 = eep_rd16(4);          //4 番地データを 2byte 読み込み
sdata4 = eep_rd16(6);          //6 番地データを 2byte 読み込み

ldata1 = eep_rd32(8);          //8 番地データを 4byte 読み込み


while(1)
{

}


}
```

【　解説　】

sdata1 = eep_rd16(0);①

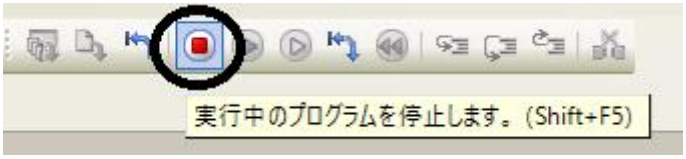
0 番地のデータを読み、0 だったら書き込みされてないと判断し、データを書き込み

```
if(sdata1 == 0)                ②        //始めだけ初期化　フラッシュ E2PROM と
逆！
{
        eep_wr16(0,1234);      //0 番地に 2byte データを書き込み
        eep_wr16(2,5678);      //2 番地に 2byte データを書き込み
        eep_wr16(4,0x9abc);    //4 番地に 2byte データを書き込み
        eep_wr16(6,0xdef0);    //6 番地に 2byte データを書き込み
        eep_wr32(8,12345678);          //8 番地に 4byte データを書き込み
}
```

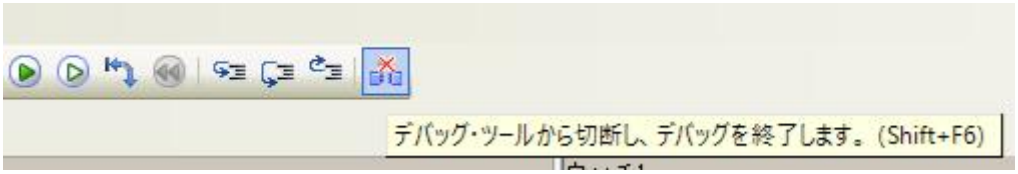
0 番地からデータを読み込み、ウオッチ窓に表示される値が書き込んだ値と同じならOK

```
sdata1 = eep_rd16(0);          ③//0 番地データを 2byte 読み込み
sdata2 = eep_rd16(2);          //2 番地データを 2byte 読み込み
sdata3 = eep_rd16(4);          //4 番地データを 2byte 読み込み
sdata4 = eep_rd16(6);          //6 番地データを 2byte 読み込み

ldata1 = eep_rd32(8);          //8 番地データを 4byte 読み込み
```

電源 OFF にしてもデータが記憶されているかですが、まず、動作を止めて



デバックツールから切断。



USB ケーブルを抜きます。これで基板の電源は供給されていません。USB ケーブルを再び差します。プログラムの書き込みをコメントにし、コンパイル。

```
void main(void)
{
        eep_init();                     //FLASH領域初期化
    /*
        sdata1 = eep_rd16(0);

        if(sdata1 == 0)                 //始めだけ初期化　フラッシュE2PROMと逆！
        {
                eep_wr16(0,1234);       //0番地に2byteデータを書き込み
                eep_wr16(2,5678);       //2番地に2byteデータを書き込み
                eep_wr16(4,0x9abc);     //4番地に2byteデータを書き込み
                eep_wr16(6,0xdef0);     //6番地に2byteデータを書き込み
                eep_wr32(8,12345678);   //8番地に4byteデータを書き込み
        }

    */
        sdata1 = eep_rd16(0);           //0番地データを2byte読み込み
        sdata2 = eep_rd16(2);           //2番地データを2byte読み込み
        sdata3 = eep_rd16(4);           //4番地データを2byte読み込み
        sdata4 = eep_rd16(6);           //6番地データを2byte読み込み

        ldata1 = eep_rd32(8);           //8番地データを4byte読み込み


        while(1)
        {

        }

}
```

コメント化

実行して先ほどと同じように、ウオッチ窓にデータが表示されたら、電源を切ってもデータが保持されていたことになります。

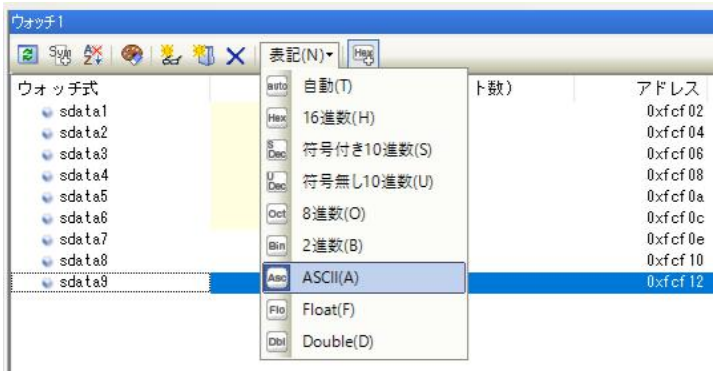
【 演習 】

FRAM に アイウエオ 諸行無常 を書いて、読みだしてください。

回答例は演習ホルダにあります。

【 ヒント 】

ウオッチ窓の表記は ASCII にしてください。

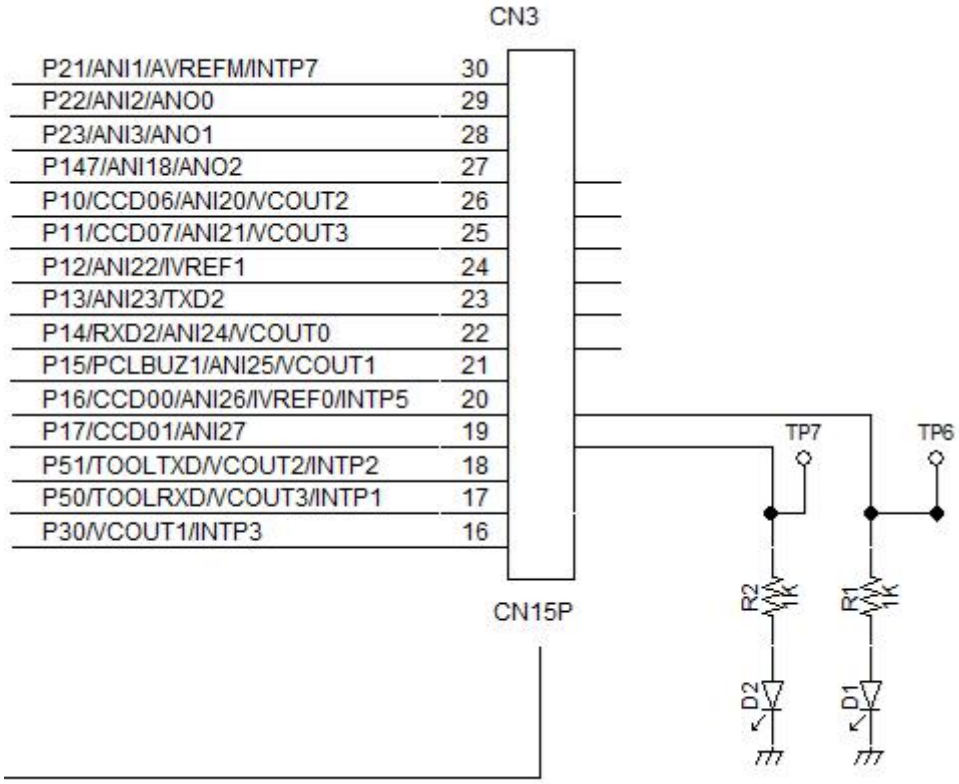


6. 割り込み動作

【 動作概要 】

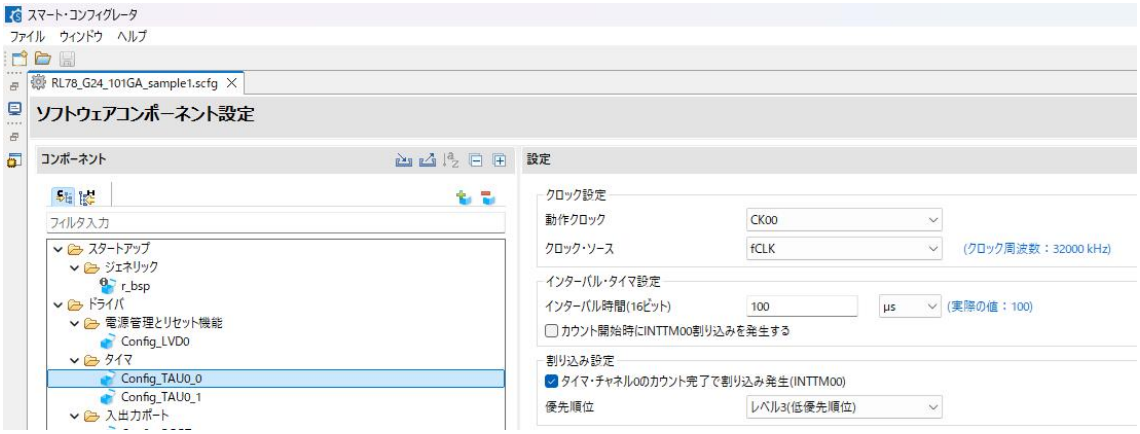
■サンプルプログラム名 RL78__G24__seminar5__INT

今までのサンプルプログラムでも、定周期割り込みを使用してタイマとして使ってきましたが、現実のプログラムでは複数の割り込みが同時にかかる多重割り込みが普通です。その考え方について学びます。既に配線済みの P16,P17 を割り込みの出力として使用します。

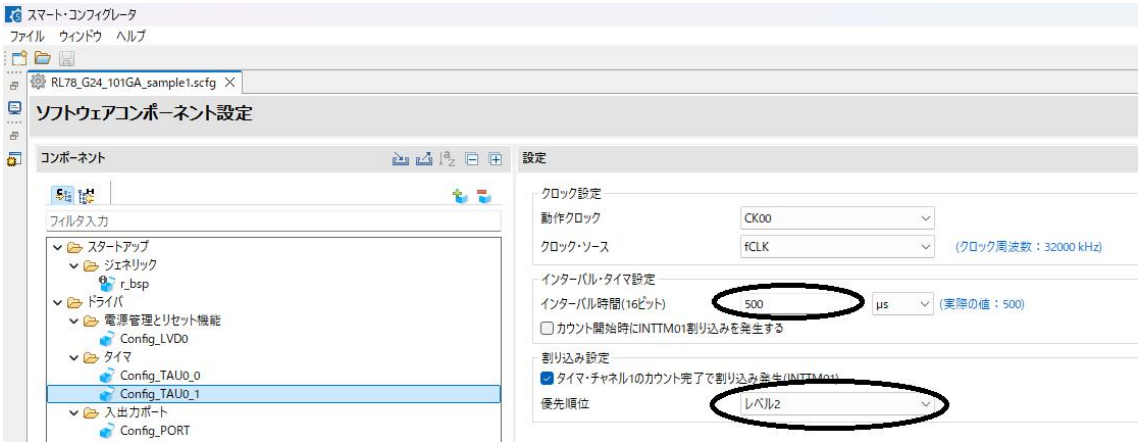


【 スマートコンフィグレータ設定 】

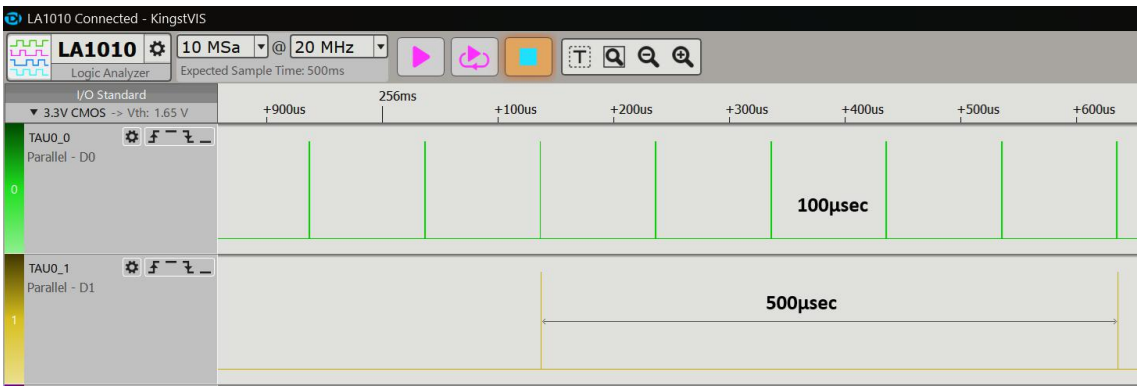
タイマ Config__TAU0__0 を１００μsec 毎に割り込みを入れます。



タイマ Config__TAU0__1 は500μsec 毎に割り込みを入れます。



割り込みの優先順位をＣｏｎｆｉｇ＿ＴＡＵ０＿０より上げます。出力波形はＴＡＵ０＿０波形が１００μsec 毎、ＴＡＵ０＿１波形が５００μsec 毎に出力され、５回に１回重なりあう（割り込みが多重に入る）部分があります。



【　メインプログラム　】

メインプログラムではタイマーをスタートさせているだけで、何もしていません。

```
void main(void)
{
        EI();

        R_Config_TAU0_0_Start();        //100usec INT
        R_Config_TAU0_1_Start();        //500usec INT

        while(1)
        {

        }
}
```

【　割り込みプログラム　】

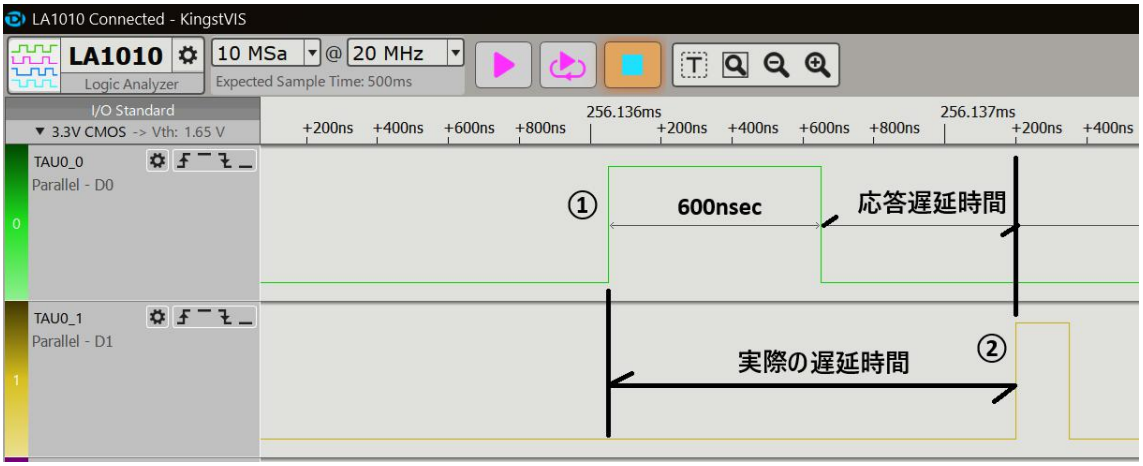
100μsec 毎の割り込みは P1.6 を上げて下げています。同じ命令が連続しているのはH幅を確保するだめです。

```
static void __near r_Config_TAU0_0_interrupt(void)
{
    /* Start user code for r_Config_TAU0_0_interrupt. Do not edit comment generated here */

        P1_bit.no6 = 1;         //100usec
//      EI();
        P1_bit.no6 = 1;
        P1_bit.no6 = 1;
        P1_bit.no6 = 1;
        P1_bit.no6 = 1;
        P1_bit.no6 = 1;
        P1_bit.no6 = 1;
        P1_bit.no6 = 1;
        P1_bit.no6 = 1;
        P1_bit.no6 = 1;



        P1_bit.no6 = 0;

        /* End user code. Do not edit comment generated here */
}
```

500μsec 毎の割り込みはP1．7を上げて、下げています。

```
static void __near r_Config_TAU0_1_interrupt(void)
{
    /* Start user code for r_Config_TAU0_1_interrupt. Do not e

    P1_bit.no7 = 1;             //500usec
    P1_bit.no7 = 1;
    P1_bit.no7 = 0;

    /* End user code. Do not edit comment generated here */
}
```

5回に1回、割り込みが重なった部分の時間軸を拡大すると①TAU0_0の割り込みが入り、P1．6がH、Lになり、応答遅延時間経過後、②TAU0_1の割り込みが入っているのが分かります。
本来、TAU0_1もTAU0_0と同じ時間に割り込みが入るはずですが、シングルコアCPUは1度に2つの命令を実行できませんから、①が終わったら、応答遅延時間経過後、②が開始されるイメージです。



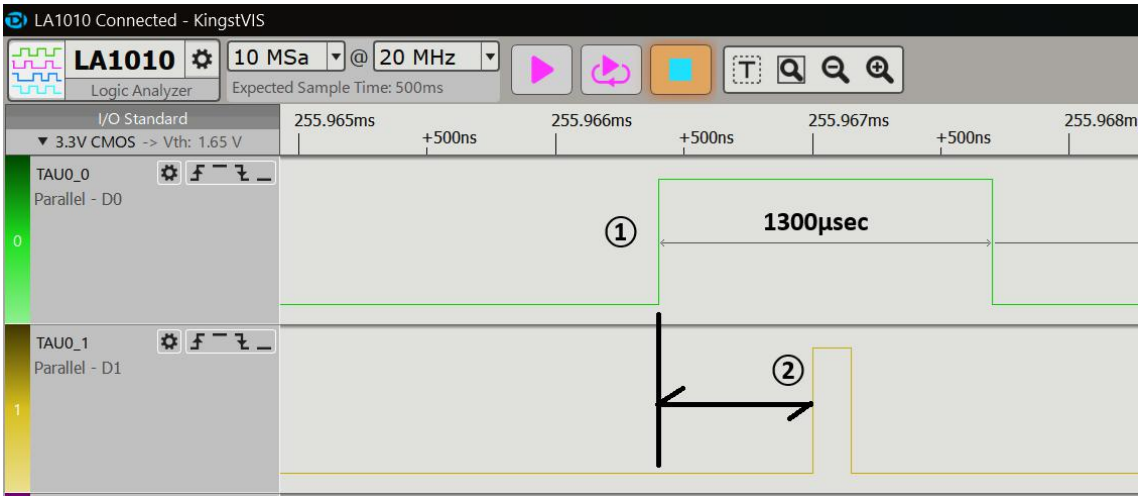
ここで、問題になるのは①にかかる時間が長い場合、②の割り込みが情報などを取りこぼさないかどうかです。
仮に、②の応答をなるべく早くしたい場合、次のようにします。
（1）②の割り込み優先順位を①より上げる
（2）①の割り込みが入った後に、割り込みを許可する（割り込みが入ると、コンパイラは自動的に

割り込みを不許可にします）

```
66      0015b   static void __near r_Config_TAU0_0_interrupt(void)
67              {
68                  /* Start user code for r_Config_TAU0_0_interrupt
69
70      0015f           P1_bit.no6 = 1;         //100usec
71      00161           EI();
72      00164           P1_bit.no6 = 1;
73      00166           P1_bit.no6 = 1;
74      00168           P1_bit.no6 = 1;
75      0016a           P1_bit.no6 = 1;
76      0016c           P1_bit.no6 = 1;
77      0016e           P1_bit.no6 = 1;
78      00170           P1_bit.no6 = 1;
79      00172           P1_bit.no6 = 1;
80      00174           P1_bit.no6 = 1;
81
82
83
84      00176           P1_bit.no6 = 0;
```

EI命令を実行する

EI 命令の追加により、TAU0__0割り込み実行中①にTAU0__1の割り込みが入る様子②が確認できました。多重割り込みで、②の応答時間が問題になる場合、上記のように割り込みの中で割り込み許可を実行すると早い応答が可能です。



このように組込みマイコンでは、起こる複数の事象を想定し、時間的な検討を行うのが普通です。クロックが高いマイコンは対応が楽になるので設計者には好まれますが、消費電力、コストが上がりますので、経営側に嫌われる場合があり、トレードオフの関係にあります。コストを重視過ぎると、ソフトウエア作業者の負担が増えますので適切なマネジメントが必要です。

【 演習 】

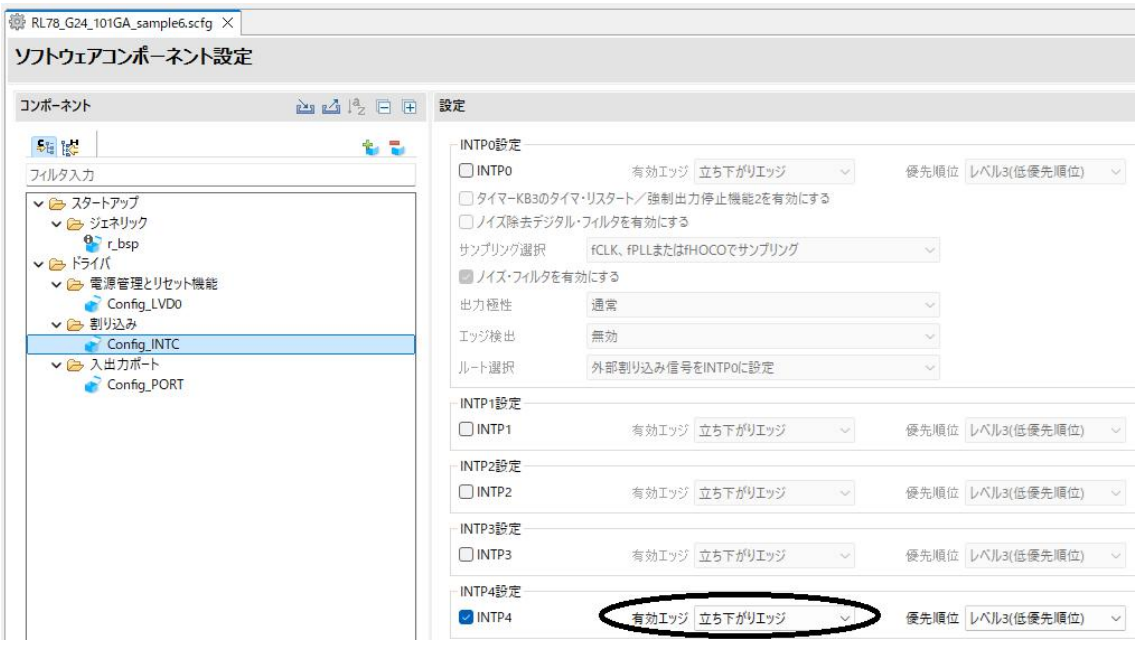
SW1 が繋がっている P31 は INTP4 端子として使うことも出来ます。

SW1 を押して割り込みで LED　D1（P1.6）を点灯させてください。

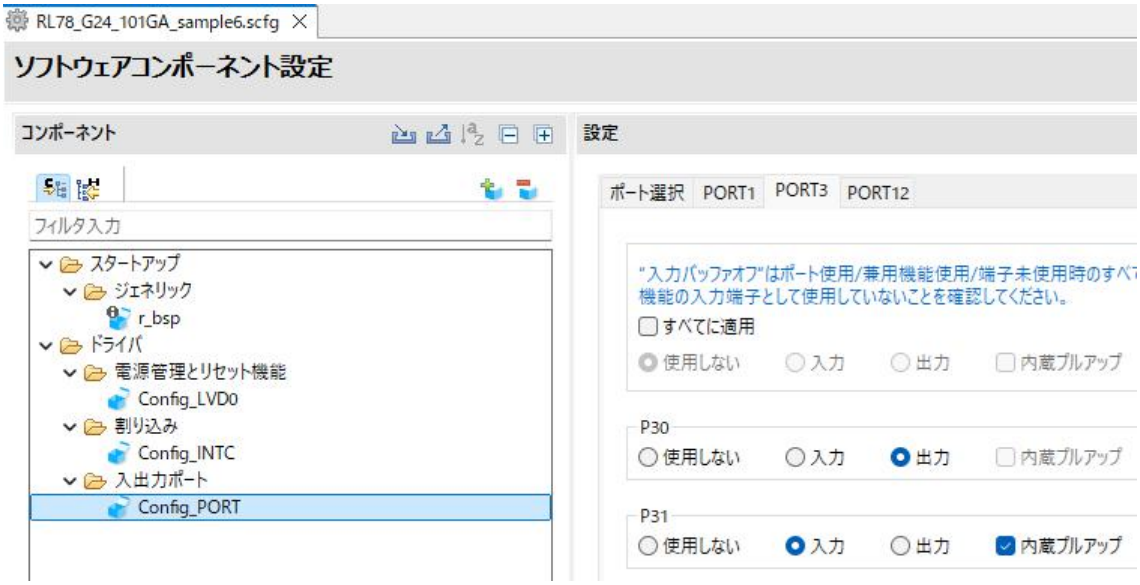
メインプログラムで SW2（P120）を押して D1 を消灯させて下さい。

【 ヒント 】

初めにスマートコンフイグレータで割込みの設定を行います。Config__INTC を+追加し、INTP4 を立下りエッジで使います。



SW1 を受ける P31,P120 は入力、プルアップしてください。



コードの生成を行います。

メインプログラムは SW2 が押されたら D1 を消灯します。

```
46                  void main(void)
47                  {
48      00160               EI();
49      00163               R_Config_INTC_INTP4_Start();
50
51
52                          while(1)
53                          {
54                                  if(P12_bit.no0 == 0)    //SW2が押されたら
55                                  {
56      00169                               P1_bit.no6 = 0; //D1をクリア
57                                  }
58
59
60                          }
61
62                  }
```

割込みプログラムは Config_INTC_user.c の中の
static void __near r_Config_INTC_intp4_interrupt(void)
に記入します。

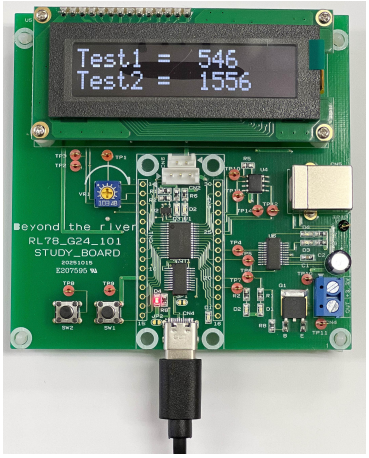

```
41
42                  /***********************************************************
43                  Global variables and functions
44                  ***********************************************************
45                  /* Start user code for global. Do not edit comment generated
46                  /* End user code. Do not edit comment generated here */
47
48                  /***********************************************************
49                  * Function Name: R_Config_INTC_Create_UserInit
50                  * Description  : This function adds user code after initiali
51                  * Arguments    : None
52                  * Return Value : None
53                  ***********************************************************
54      00401       void R_Config_INTC_Create_UserInit(void)
55                  {
56                      /* Start user code for user init. Do not edit comment ge
57                      /* End user code. Do not edit comment generated here */
58                  }
59
60                  /***********************************************************
61                  * Function Name: r_Config_INTC_intp4_interrupt
62                  * Description  : This function is INTP4 interrupt service ro
63                  * Arguments    : None
64                  * Return Value : None
65                  ***********************************************************
66                  static void __near r_Config_INTC_intp4_interrupt(void)
67                  {
68                      /* Start user code for r_Config_INTC_intp4_interrupt. Do
69      0015b           P1_bit.no6 = 1;     //D1を点灯します
70
71                      /* End user code. Do not edit comment generated here */
72
```

【 応用 】
1. 有機 EL 表示器を使った表示

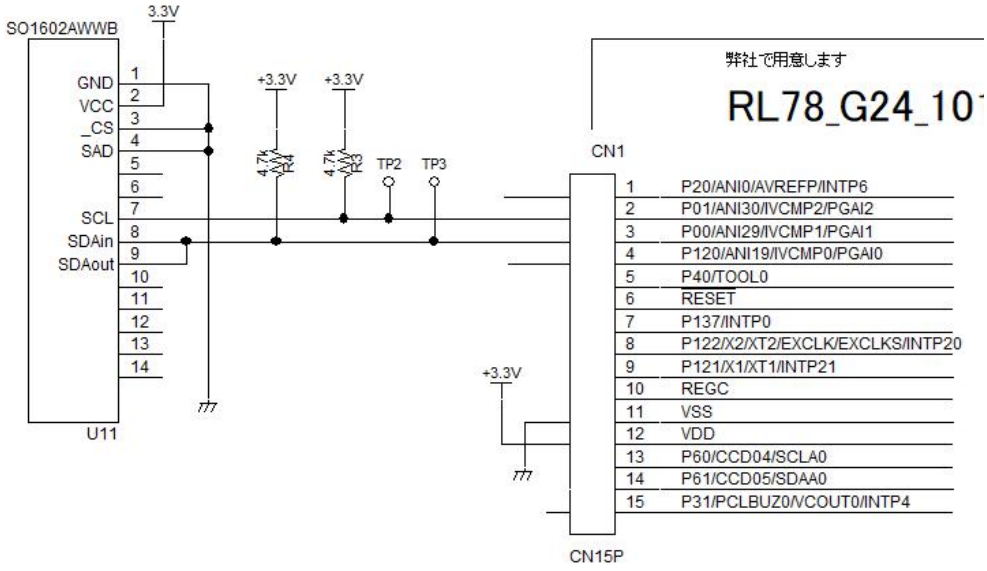
【 動作概要 】
■サンプルプログラム名　RL78__G24__seminar__ouyou__OLED

有機 EL の英語表記は、現象そのものには organic electro-luminescence（OEL）、素子やディスプレイ製品には一般的に OLED（Organic Light Emitting Diode）が使われます。「organic」が有機物、「luminescence」が発光現象、「diode」がダイオードを意味し、いずれも電気によって有機物が光る仕組みを利用していることを指します。

サンプルプログラムは変数１，２を変化させ表示させます。



【 配線 】
RL78 から OLED 基板への配線は2本ですが、GND、３．３V、4.7KΩのプルアップ配線も忘れないで接続して下さい。



【 ソフトウエア 】

変数data1、data2を表示しています。

```
52      005bf   void main(void)
53              {
54              //      EI();                   //割り込み許可
55
56      005c1           init_SO1602();          //有機EL　イニシャル
57
58      005c7           data1 = 23;             //特に意味のある数字ではありません
59                      data2 = 78;             //特に意味のある数字ではありません
60
61                      while(1U)               //P40 (TOOL0) 、P50 (TOOLRXD) ,51 (
62                      {
63
64      005e1                           sprintf(tp_buff,"Test1 = %4d",data1);
65      00604                           lcd_puts(0,tp_buff);
66      0060b                           sprintf(tp_buff,"Test2 = %5d",data2);
67      0062e                           lcd_puts(1,tp_buff);
68      00636                           data1++;
69      00639                           if(data1 > 1000){data1 = 23;}
70      005d3                           data2++;
71      005d6                           if(data2 > 10000){data2 = 78;}
72                      }
73
74              }
```

#include "lcd_SO1602_RL78.c"　のライブラリの中に有機 EL 駆動ライブラリがあります。

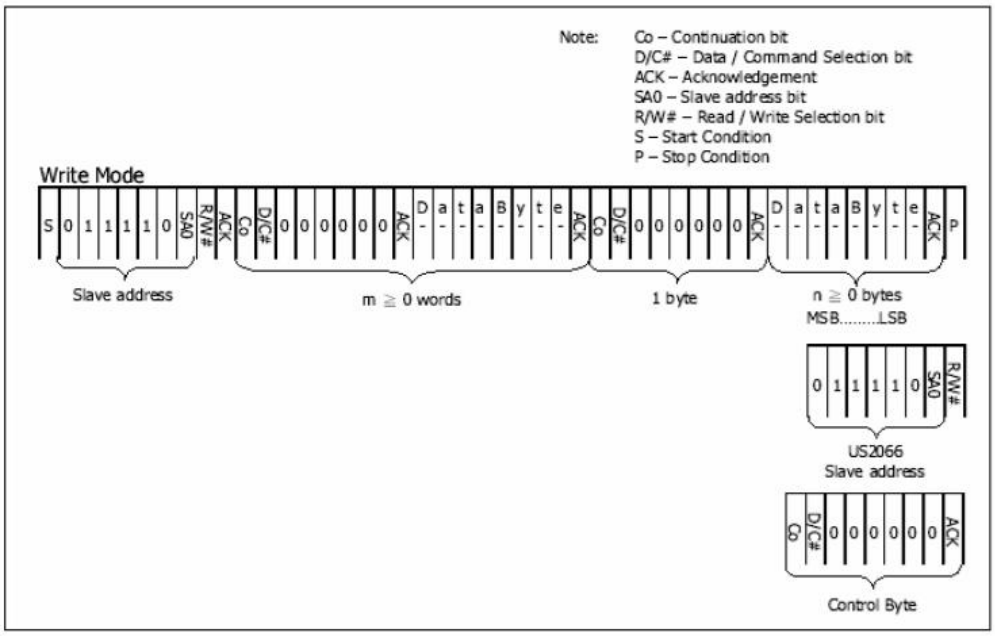
動作させると有機 EL の表示がどんどん変わるのが確認できると思います。

【 I2C の詳細 】

有機 EL のインターフェイス(I2C)フォーマットです。

SUNLIKE DISPLAY Mode No: SO1602A

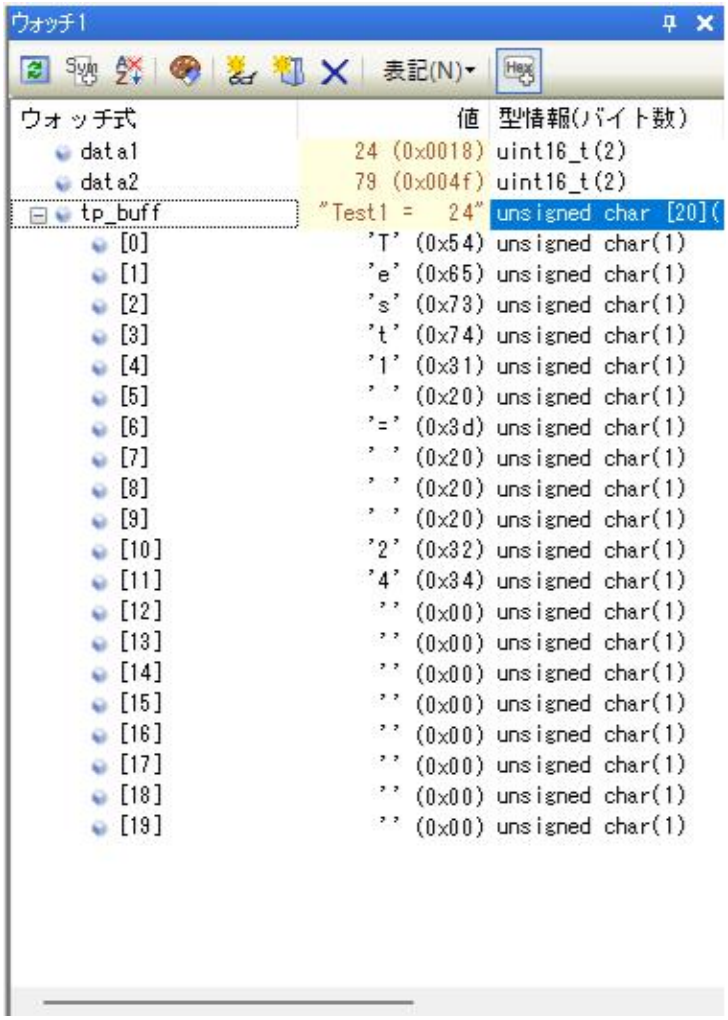
8. Read/Write Timing Chart



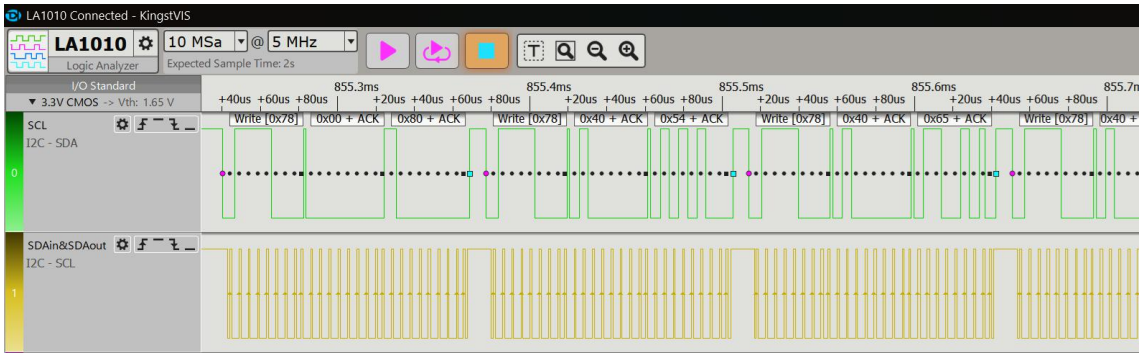
Lcd__puts()関数の具体的な動きを見てみましょう。下記はブレークポイントを設定し、関数の実行前までプログラムカウンタを進めた例。

```
61                      while(1U)               //P40 (TOOL0) 、P50 (TOOLRXD) ,51
62                      {
63
64      005e1                           sprintf(tp_buff,"Test1 = %4d",data1);
65      00604                           lcd_puts(0,tp_buff);
66      0060b                           sprintf(tp_buff,"Test2 = %5d",data2);
67      0062e                           lcd_puts(1,tp_buff);
68      00636                           data1++;
69      00639                           if(data1 > 1000){data1 = 23;}
70      005d3                           data2++;
71      005d6                           if(data2 > 10000){data2 = 78;}
72                      }
73
74  }
```

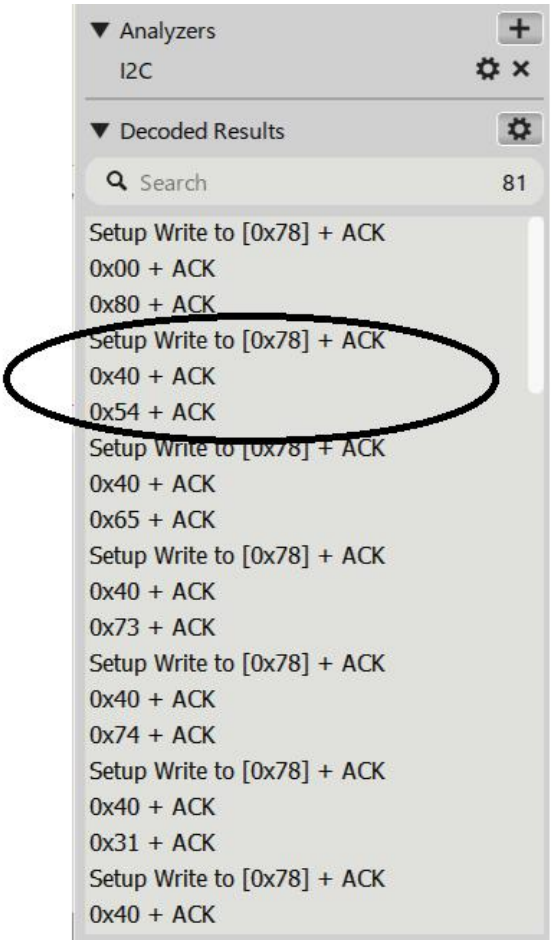
書き込まれるデータはTest1　＝　　24 です。

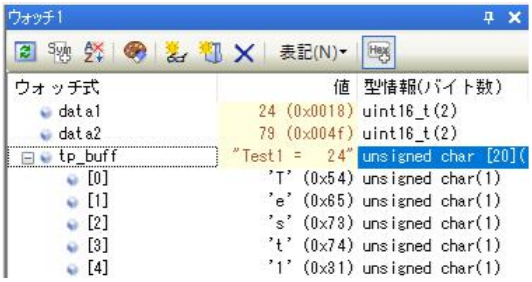


シリアルアナライザでSCL、SDAの 2 本のラインをサンプリングした I2C 波形です。



I2Cラインを見るシリアルアナライザで受けたデータは Test1 ＝ 、、、 の ASCII 文字 0x54, 0x65、0x73、0x74、0x31、、が受信できています。ウオッチ窓にあるRL78が出力したデータと同じなら問題ないわけです。





Setup Write to「0x78」というのは Slaveaddress（有機 EL のアドレス ＋ SAD＝0, __Write）0x78 を示しています。次の 0x40+ACK は D/_C （Data/Command Selection bit）です。1を立てて、以降がデータであるこを示しています。大文字 T の ASCII コードは0x54で、それが書かれることにより有機ELに「 T 」が表示されます。順次、Test1 ＝ 24 と書き込まれます。

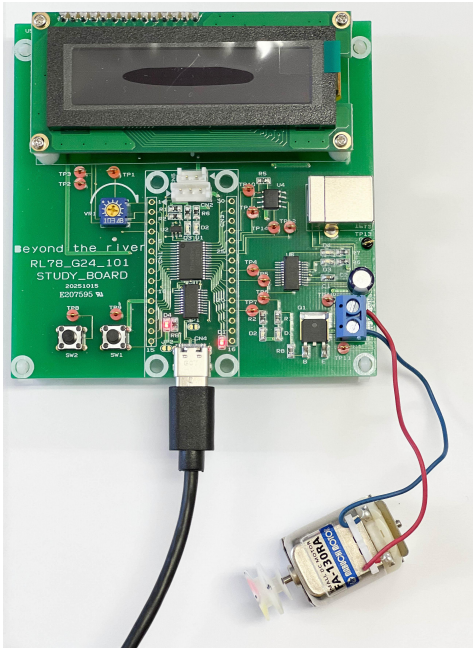
【 演習 】

SW1 を押すと 1 ガオサレマシタ SW2 を押すと 2 ガオサレマシタ SW1 と SW2 を同時に押すと消えるプログラム

プログラム例は演習ホルダーにあります。

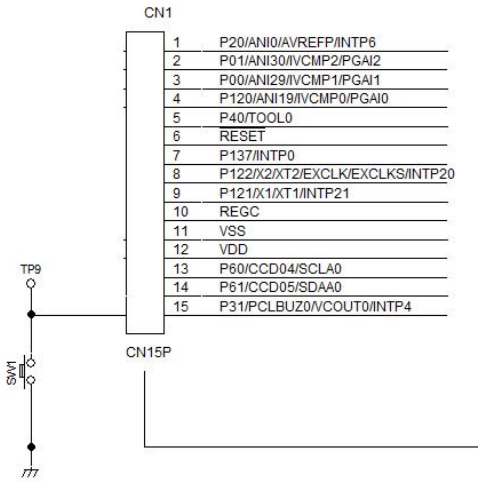
【 応用 】

2. PWM を使用したモーター速度制御

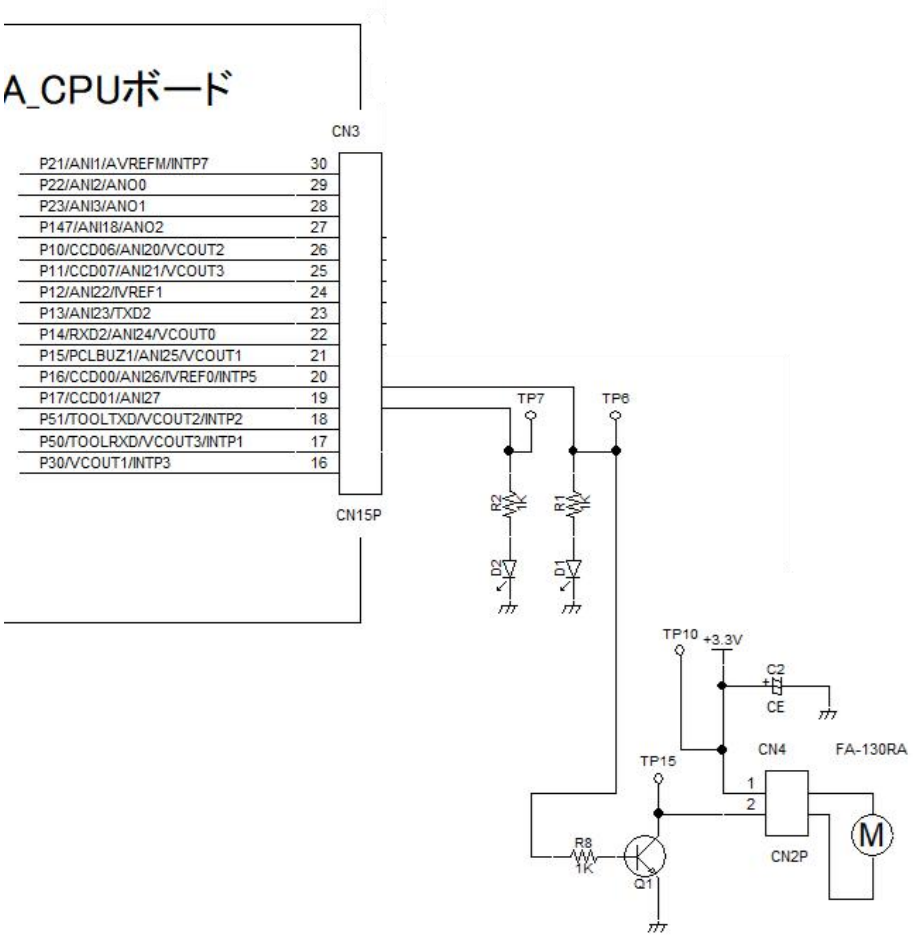


【 配線 】

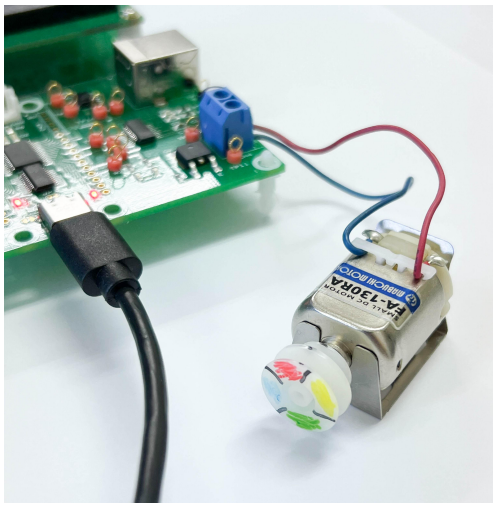
モーター動作開始に SW1 を使います。



モーターの配線はCN4に取り付けます。電気的な極性は特にありませんが、CN4の1にモーターの赤、2に青を接続するとSW1を押したときにCW（クロックワイズ　時計方向）に回転します。



なお、モーターに添付するプーリーに色を塗ったり、線を引いたりすると回転しているか、否かが分かりやすくなります。

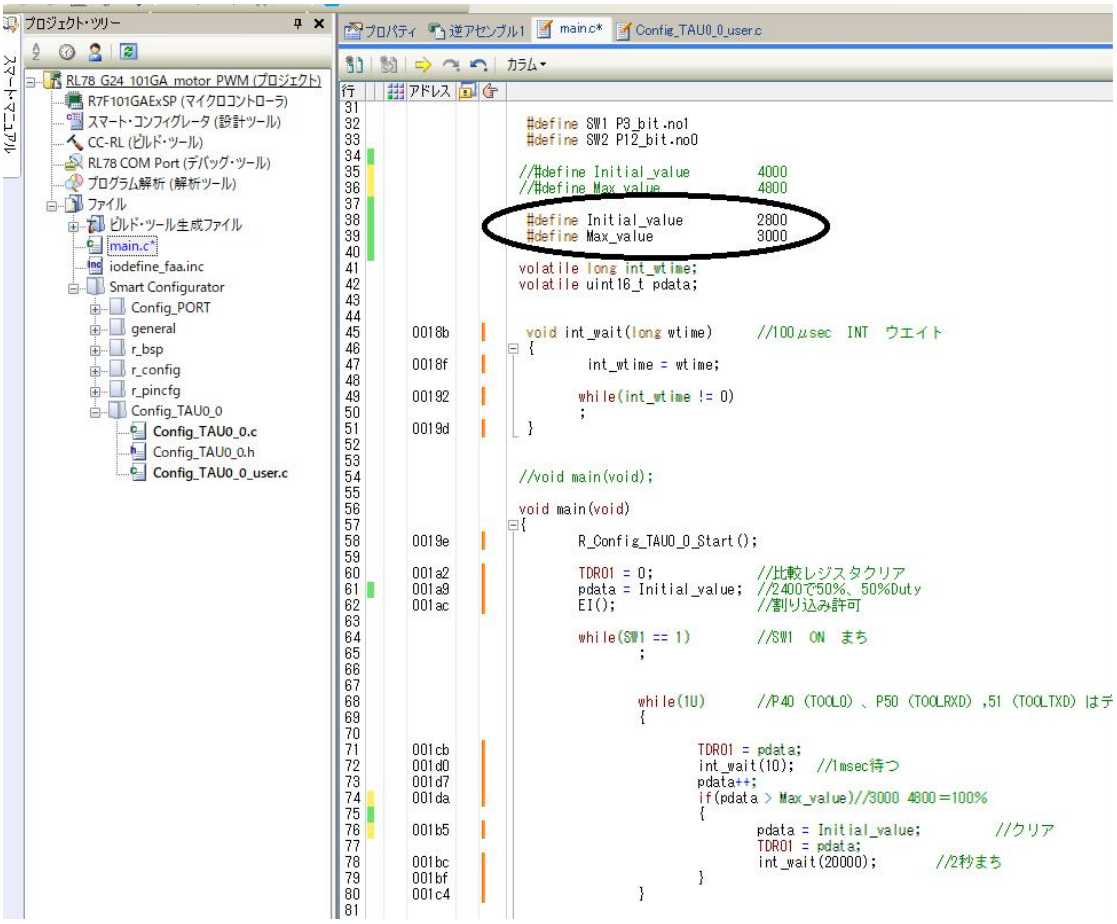


【 動作概要 】

■サンプルプログラム名　RL78__G24__seminar__ouyou__Motor

SW1 を押すと Initial_value から動作が始まります。速度を上げて、出力 Max_value で停止 2 秒、を繰り返します。

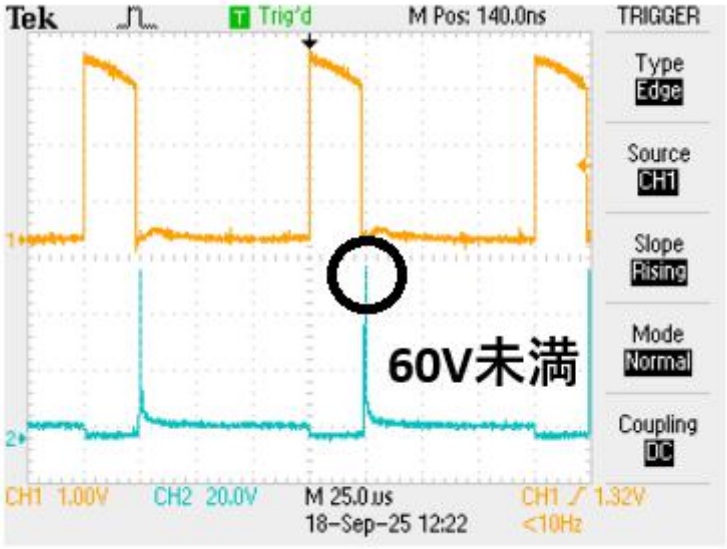
【 プログラム 】

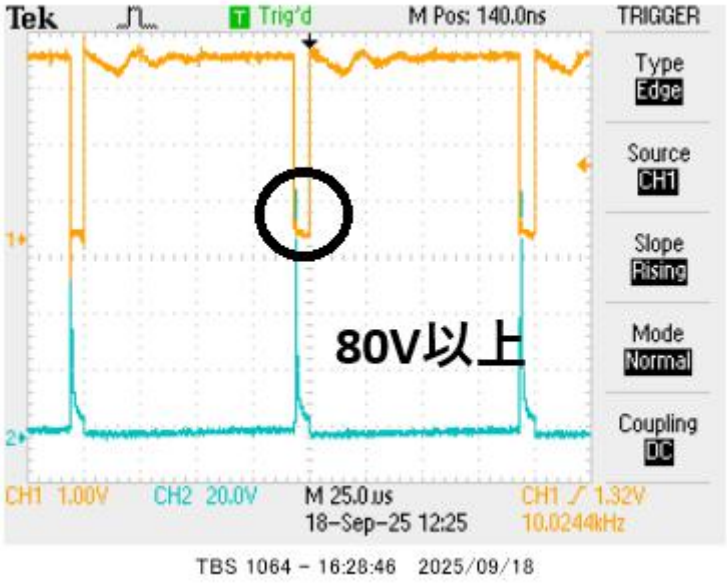


【 波形 質問 】

図は2CHの オシロスコープ で TP6 トランジスタにベース与えるPWM波形（オレンジ色 CH1）と TP11 トランジスタのコレクタを観測した波形（青 CH2）です。

何か分かりますか？ また、PWM の H 幅が長いほど電圧が上がっている理由が分かりますか？





【 答え 】

1. CH2 で観測できる波形は逆起電力といいます。モーターにエネルギーを与え（CH1 オレンジ色波形Hのとき）、切れた時点（CH1 オレンジ色波形Lのとき）で、コイルに蓄えられたエネルギーで切れる前と逆方向に電圧が発生します。

2. H 幅が長いほどモーターの回転は速くなっています。逆起電力は

コイルの巻数を N としたとき、1 箇所あたりの逆起電力は下記の式で表されます。(フレミング右手の法則)

e＝NBLrω
e:逆起電力(発生電圧)[V]
N:コイルの巻数
B:磁束密度[T]
L:磁界中の電線長[m](磁束に対して交差する方向のコイルの長さ)
r:回転半径[m]
ω:回転速度[rad/s]

これにより回転速度が速いほど逆起電力eは大きくなることが分かります。

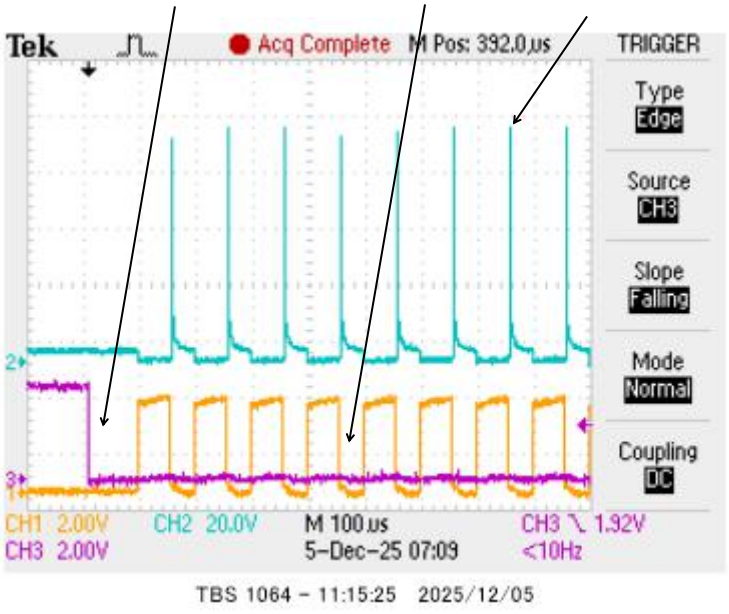
コイルを使った制御はこの逆起電力の収束方法が問題になる場合があり、注意が必要です。一方、この特性を生かした機器が入力電圧より昇圧する DCDC コンバータなど多数あります。

【 演習 】

応用のプログラムよりも最高速度が高くなるプログラムを作成してみて下さい。例は演習ホルダの中にあります。

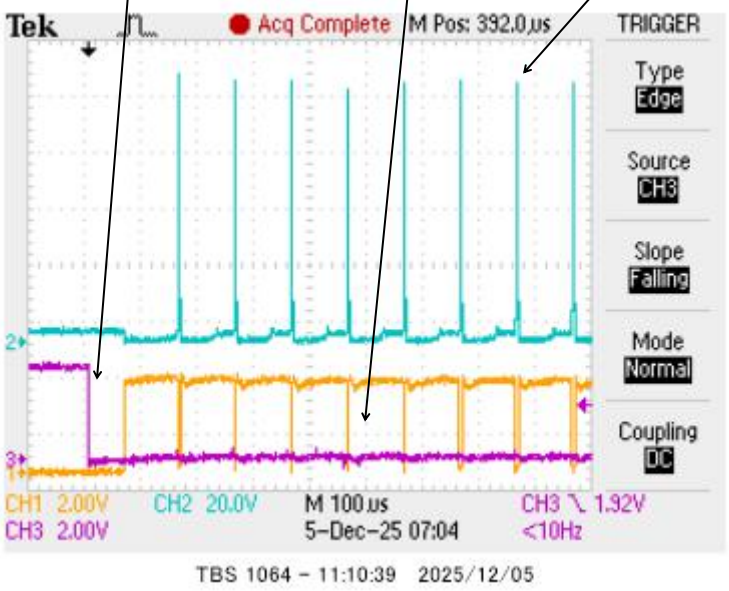
【 検証 】

下図が応用プログラムを動作させたときの各部の波形です。
スイッチ SW1 を押されたとき　　TP6:PWM 出力　TP11:逆起電力



下図が演習プログラムを動作させたときの各部の波形です。

スイッチ SW1 を押されたとき　　TP6：PWM 出力　TP11:逆起電力



応用プログラムは初動の数値が

#define Initial_value	2800
#define Max_value	3000

演習プログラム例の数値は

#define Initial_value	4600
#define Max_value	4800

です。

応用の TP11：逆起電力の波形の高さが、明らかに演習の方が高く(80V 以上)、よって **e＝NBLrω**　式により、回転速度が応用プログラムより、演習プログラムの方が速いことになります。

質問、お問い合わせ

本製品に対する質問やお問い合わせは以下にお願いします。どんな簡単なことでもＯＫです。専門家がお答えします。

〒3501213
埼玉県日高市高萩 1141-1
有限会社ビーリバーエレクトロニクス
TEL：042（985）6982　EMAIL：info@beriver.co.jp