ＧＫＲＸ７＿１Ｍ　マイコン学習セット　入門編

初版　２０１８．１．２６

【　製品概要　】

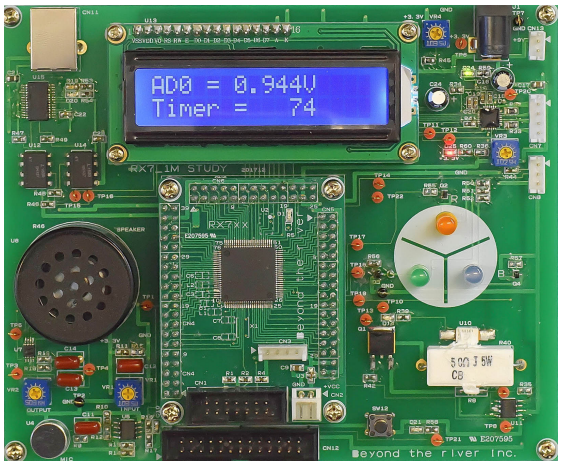
本マニュアルはマイコンの学習を行うためのハードウエア、ソフトウエアの入門編です。

入門編では開発環境の構築からマイコンプログラムの作成、実行までを詳細に解説。また、基本的な周辺機能である　Ｉ／Ｏポート、ＵＳＢ、Ａ／Ｄコンバータ、Ｄ／Ａコンバータの使い方を平易なサンプルプログラムをもとに説明します。

ハードウエアはマイコン基板　ＢＣＲＸ７＿１Ｍ　ＣＰＵボード＋専用ハードウエアで構成される学習ボードです。

ソフトウエアはルネサス社の無償統合開発環境ＣＳ＋　ｆｏｒ　ＣＣ上で動くサンプルプログラムです。Ｃ言語で書かれています。めんどくさい初期設定はＣＳ＋の「コード生成」機能で簡単に行え、２４０ＭＨｚクロック、ＦＰＵ内蔵で驚異の演算速度を体験出来ます。

※本学習ボード開発にはルネサスエレクトロニクス社製Ｅ１が別途必要です。



１．開発環境、事前準備

１－１．　開発環境

ａ：開発セット　同梱物

ｂ：ＢＣＲＸ７＿１Ｍ　ＣＰＵボードの特徴

ｃ：Ｅ１エミュレータ（デバッカ）

ｄ：無償のＣＳ＋、ＲＸ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

１－２　　動作、デバック

ａ：ＣＳ＋起動、コンパイル、書き込み、動作

ｂ：新しいプログラムを作る　ＣＳ＋　操作

ｂ－１：初めにＩＯポートの設定

ｂ－２：プログラムの書き込み、書く場所の注意
ｂ－３：パワーオンリセットを設定する

ｂ－４：デバッカの設定がデフォルトはエミュレータなので注意
ｂ－５：Ｅ１から電源供給
ｂ－６：クロック発生回路を設定する必要があります

ｃ：その他
ｃ－１：動作中に変数の変化を見るには？
ｃ－２：サンプルを走らせるときにｖｅｃｔ．ｈの重複するアドレスを削除
ｃ－３：三角関数ｍａｔｈ．ｈはインクルードもＣＳ＋の設定も必要
ｃ－４：割り込みが入っているか？周期は？簡単なチェック方法
ｃ－５：既存のプログラムを雛形として新しいプログラムを作る
ｃ－６：コード生成と見落としがちな注意点

２．サンプルプログラム

２－１．	ｓａｍｐｌｅ１　出力ポートのＯＮ，ＯＦＦ	ＬＥＤをチカチカさせます。
２－２．	ｓａｍｐｌｅ２　ＳＩＯ（ＵＳＢ）	ＵＳＢ経由でパソコンとやりとり
２－３．	ｓａｍｐｌｅ３　Ａ／Ｄ変換をＵＳＢ出力	アナログ量をデジタルに変換し、パソコンに出力します。
２－４．	ｓａｍｐｌｅ４　割り込み	１ｍｓｅｃ間隔定周期タイマ割込み
２－５．	ｓａｍｐｌｅ５　ＰＷＭ出力	ＰＷＭでＬＥＤの明るさを変えます
２－６．	ｓａｍｐｌｅ６　三角、対数、平方根関数	関数演算速度を検証
２－７．	ｓａｍｐｌｅ７　Ｄ／Ａにｓｉｎ演算した正弦波を出力する	音の高低をスピーカーで聞く
２－８．	ｓａｍｐｌｅ８　液晶表示	ＡＤの値を液晶表示します。
２－９．	ｓａｍｐｌｅ９　ステージを動かす	パルスモータ制御の基本を学習

１−１．開発環境

ａ：学習セット同梱物

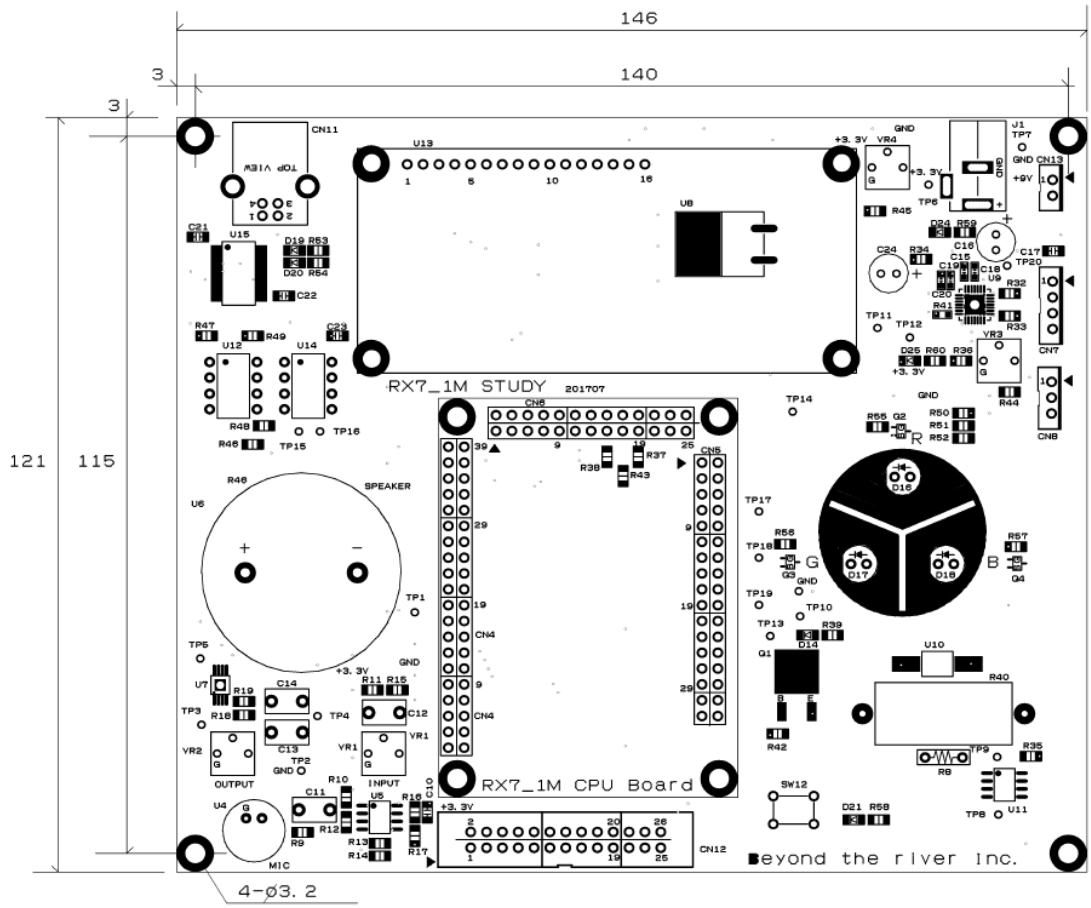
ＢＣＲＸ７＿１Ｍ　学習ボード
ＣＤ（サンプルプログラム、デバイスドライバ、ドキュメント）
マニュアル（入門編　本誌、応用編）
ＡＣアダプタ、ＵＳＢ（フルサイズ）ケーブル、パルスモータステージ
※開発に必要なルネサスエレクトロニクス社製デバッカＥ１は同封されておりません。別途必要です。

ｂ：ＢＣＲＸ７＿１Ｍ　ＣＰＵボードの特徴（Ｒ５Ｆ５７１ＭＬＣＤＦＰ　１００ピン搭載）

●ルネサス独自のＲＸ　ＣＰＵコア、内部３２ビットデータバス幅マイクロコンピュータ。３．３Ｖ　２４０ＭＨｚ動作可能。従来のＲＸ製品に搭載されたコアとの互換性を踏襲しながらも更に強力に進化したＲＸｖ２コアを採用し、フラッシュ内蔵マイコンとして最高クラスとなる１０４４ｃｏｒｅｍａｒｋを実現。フラッシュメモリ向けに最適化したキャッシュ（ＡＦＵ）により２４０ＭＨｚノーウェイト相当のフラッシュメモリアクセスが可能。
●ＦＰＵ　単精度浮動小数点数（３２ビット）ＩＥＥＥ７５４に準拠したデータタイプ、および例外
●メモリ容量　内蔵フラッシュＲＯＭ　４Ｍバイト、内蔵ＲＡＭ５１２Ｋバイト　内蔵データフラッシュ６４Ｋバイト
●Ａ／Ｄコンバータ　：１２ビット分解能×２２　　変換速度０．４８μｓｅｃ／１ｃｈ（ＡＤＣＬＫ＝６０ＭＨｚ）
●Ｄ／Ａコンバータ：１２ビット分解能×１
●外部バス拡張機能：あり（外部にデータバス、アドレスバス等出力できます）
●Ｉ／Ｏポート：入出力：７８、入力：１、プルアップ抵抗：７８　オープンドレイン出力：７８　５Ｖトレラント：１７
●タイマ：１６ビットタイマパルスユニット（ＴＰＵａ）（１６ビット×６チャンネル）、ポートアウトプットイネーブル３（ＰＯＥ３ａ）、汎用ＰＷＭタイマ（ＧＰＴａ）（１６ビット×４チャンネル）、プログラマブルパルスジェネレータ（ＰＰＧ）、８ビットタイマ（ＴＭＲｂ）（８ビット×２チャンネル）×２ユニット、コンペアマッチタイマ（ＣＭＴ）（１６ビット×２）×２ユニット、コンペアマッチタイマＷ（ＣＭＴＷ）（３２ビット×１チャンネル）×２ユニット、リアルタイムクロック（ＲＴＣｄ）、ウオッチドッグタイマ（ＷＤＴＡ）、独立ウオッチドグタイマ（ＩＷＤＴＡａ）
●イーサネットコントローラ（ＥＴＨＥＲＣ）２チャンネル、ＵＳＢ２．０ＦＳホスト／ファンクションモジュール（ＵＳＢｂ）１ポート、シリアルコミュニュケーションインターフェイス（ＳＣＩｇ、ＳＣＩｈ）９チャンネル、ＦＩＦＯ内蔵シリアルコミュニュケーションインターフェイス（ＳＣＩＦＡ）４チャンネル、ＩＩＣバスインターフェイス（ＲＩＩＣａ）２チャンネル、ＣＡＮモジュール（ＣＡＮ）３チャ

ンネル、シリアルペリフェラルインターフェイス（ＲＳＰＩａ）２チャンネル、クワッドシリアルペリフェラルインターフェイス（ＱＳＰＩ）１チャンネル
●シリアルサウンドインターフェイス（ＳＳＩ）２チャンネル、サンプリングレートコンバータ（ＳＲＣ）、ＭＭＣホストインターフェイス（ＭＭＣＩＦ）、パラレルデータキャプチャユニット（ＰＤＣ）、温度センサ等内蔵。
●オンチップデバッキングシステム：（ＦＩＮＥインターフェイス）
●動作周囲温度：－４０～＋８５℃
●ＥＥＰＲＯＭ：　２５ＬＣ２５６（３２Ｋバイト）　電源ＯＦＦでもデータ保持。　※オプション（実装品はご相談下さい）
●電源　　２．７Ｖ～３．６Ｖ　単一　４０ｍＡ（２４０ＭＨｚ動作　ＴＹＰＥ）
　　Ｅ１デバッカを使用して動作させる時Ｅ１から３．３Ｖの電源を供給できます。
　　デバック時など２００ｍＡ以内の使用であれば他に用意する必要はありません。
●クリスタル：メイン　１２ＭＨｚ（×４逓倍で５０ＭＨｚ作成）実装済み。
●デバックコネクタ：Ｅ１用（ＦＩＮＥインターフェイス）デバックコネクタ実装済み。
●基板サイズ　６４×４８×１３（Ｈ）ｍｍ
●基板仕上げ　金メッキ　ＲｏＨＳ指令準拠　基板、部品、半田付け全てのエ程でＲｏＨＳ指令準拠仕様。

基板大きさ（部品面）



ｃ：Ｅ１エミュレータ

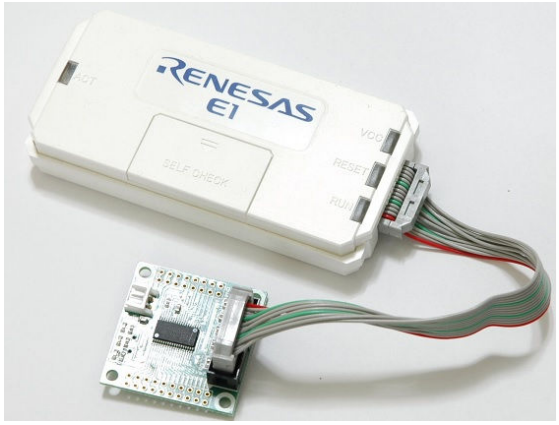
概要

Ｅ１エミュレータは、ルネサス主要マイコンに対応したオンチップデバッギングエミュレータです。基本的なデバッグ機能を有した低価格の購入しやすい開発ツールで、フラッシュプログラマとしても使用可能です。
Ｃ言語ソースデバックが可能で、１行実行、ブレークポイント設定、変数、レジスタ、メモリ参照等々、従来であれば高価なＩＣＥしか出来なかった機能が、安価に実現されています。また、使い方もＨＥＷ（統合開発環境）のＥ８ａと同じで、経験があれば半日で、無くても１日で必要な操作を会得することが出来ると思います。

マイコンとの通信として、シリアル接続方式とＪＴＡＧ接続方式の２種類に対応しています。使用可能なデバッグインタフェースは、ご使用になるマイコンにより異なります。
また、基本デバッグ機能に加え、ホットプラグイン機能（動作中のユーザシステムに後からＥ１エミュレータを接続して、プログラムの動作確認を行うことが可能）を搭載しているため、プログラムのデバッグ・性能評価に大きく貢献できます。

対応MPU

- RH850,V850 ファミリ
- RX ファミリ
- RL78 ファミリ
- R8C ファミリ
- 78K ファミリ



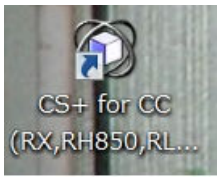
Ｅ１を購入するとＣＤが添付されています。インターネットが動作する環境で、パソコンにＣＤをセットし、ＲＥＡＤＭＥ＿ｊ．ＨＴＭを実行、ネットから最新Ｅ１ドライバーのインストールを行ってください。
続いて、ネットから開発環境ＣＳ＋とＣコンパイラの最新版ダウンロードを行います。

ｄ：無償版ＲＸ用Ｃコンパイラのダウンロード

省略

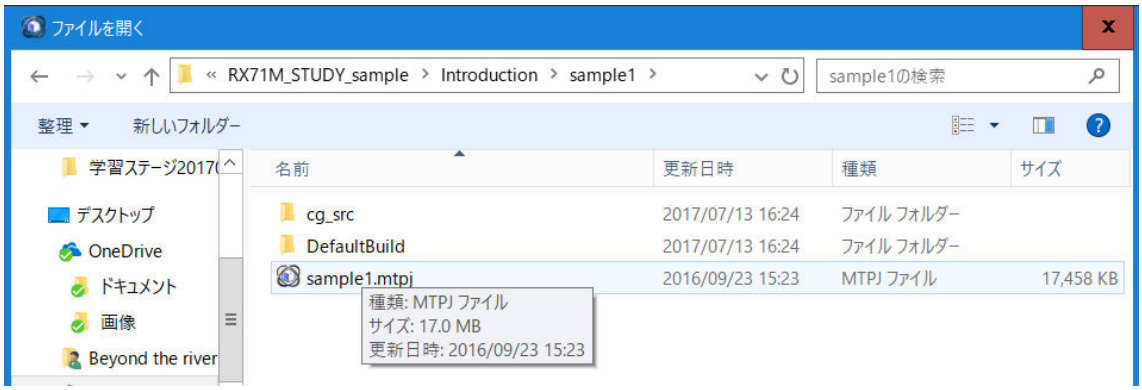
１－２　　動作、デバック

ａ：ＣＳ＋　ｆｏｒ　ＣＣ　起動、コンパイル、書き込み、動作

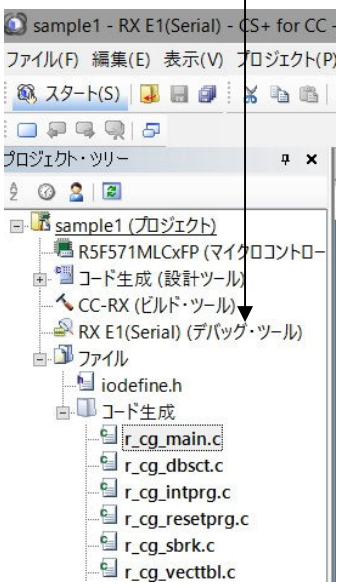


ＣＤに添付しているサンプルプログラムを使って、コンパイル、書き込み、動作の方法を示します。

ＣＳ＋　ｆｏｒ　ＣＣを起動します。ここでは例としてＲＸ７＿１Ａ＿ｓｕｔｄｙ＿ｓａｍｐｌｅ￥入門￥ｓａｍｐｌｅ１を動作させます。基板上のＬＥＤ　Ｄ１が点滅するプログラムです。
ファイル　→　ファイルを開く　→　ｓａｍｐｌｅ１．ｍｔｐｊをダブルクリックします。



プロジェクトツリーが表示されます。

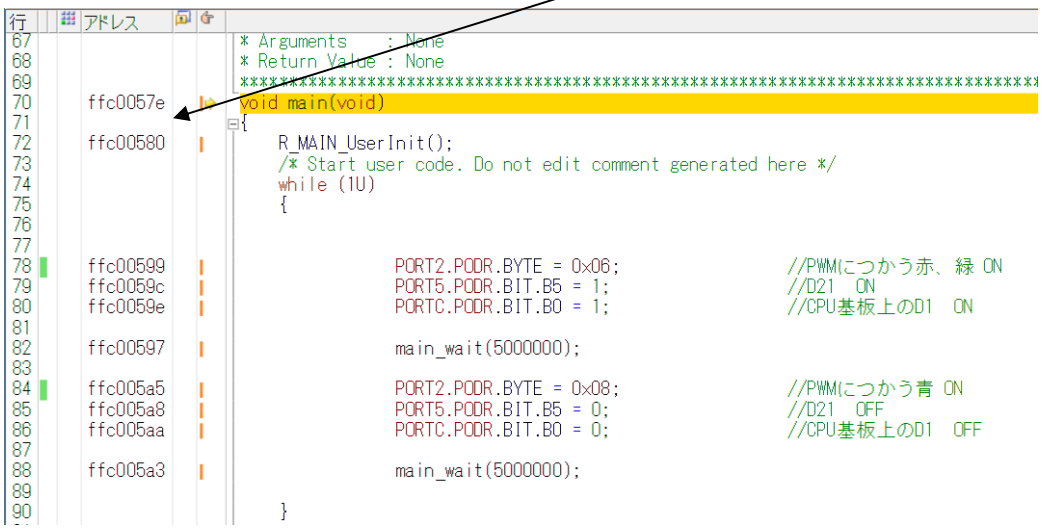


ｓａｍｐｌｅ１．ｃが中央に表示されます。とりあえず、実行してみます。Ｅ１のケーブルを基板のＣＮ１に挿入します。電源は添付のＡＣアダプタより供給します。（写真ご参考）
電源アダプタをコンセントに入れて電源を供給すると９Ｖ側ＬＥＤ　Ｄ２４が緑色に点灯、３．３Ｖ側ＬＥＤ　Ｄ２５が赤色に点灯します。Ｅ１のＶＣＣが緑色に点灯します。→　以上３点のＬＥＤが点灯しな

い場合、異常です。何か理由があります。コンセントからＡＣアダプタを抜き、原因不明の場合、弊社にご連絡ください。

「デバック・ツールへプログラムを転送」をクリック。

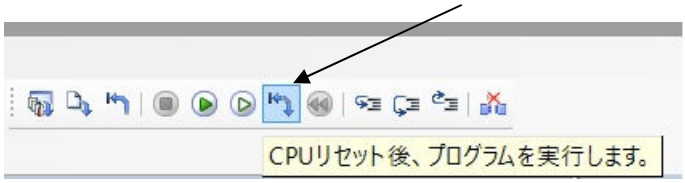
上手く転送できると、今まで表示されていなかったプログラムの絶対番地が表示されます。カーソルの黄色い帯が表示されます。



```
行  アドレス
67                  * Arguments    : None
68                  * Return Value : None
69                  ***********************************************************************************
70      ffc0057e    void main(void)
71                  {
72      ffc00580        R_MAIN_UserInit();
73                      /* Start user code. Do not edit comment generated here */
74                      while (1U)
75                      {
76
77
78      ffc00599                PORT2.PODR.BYTE = 0x06;             //PWMにつかう赤、緑 ON
79      ffc0059c                PORT5.PODR.BIT.B5 = 1;              //D21  ON
80      ffc0059e                PORTC.PODR.BIT.B0 = 1;              //CPU基板上のD1  ON
81
82      ffc00597                main_wait(5000000);
83
84      ffc005a5                PORT2.PODR.BYTE = 0x08;             //PWMにつかう青 ON
85      ffc005a8                PORT5.PODR.BIT.B5 = 0;              //D21  OFF
86      ffc005aa                PORTC.PODR.BIT.B0 = 0;              //CPU基板上のD1  OFF
87
88      ffc005a3                main_wait(5000000);
89
90                      }
```

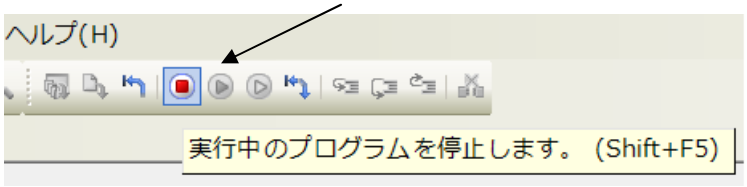
ここまでいかなかった場合、Ｅ１のデバイスドライバインストゥールをご検証願います。

次に、プログラムを動作させます。「ＣＰＵリセット後、プログラムを実行」をクリック。



Ｅ１のＲＵＮ（緑ＬＥＤ）が点灯し、ＣＰＵ基板のＤ１、学習ボードのＤ１６，１７，１８、Ｄ２１が点滅したら動作しています。ＣＳ＋の右下部にも表示されます。

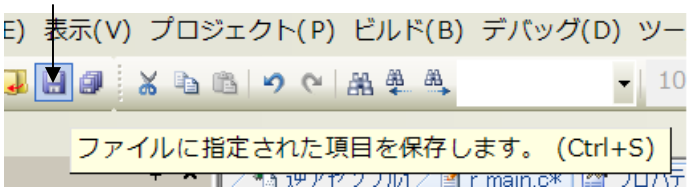
ここまで確認できましたら、一度止めます。



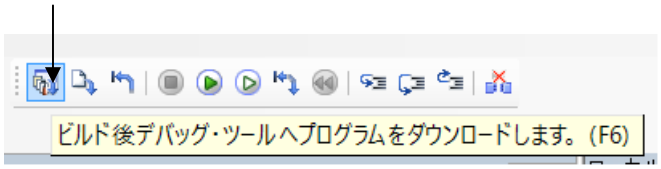
ｍａｉｎ関数のｌｗａｉｔの数値２箇所を１桁０を増やしてみます。

```
R_MAIN_UserInit();
/* Start user code. Do not edit comment generated here */
while (1U)
{

        PORT2.PODR.BYTE = 0x06;          //PWMにつかう赤、緑 ON
        PORT5.PODR.BIT.B5 = 1;           //D21  ON
        PORTC.PODR.BIT.B0 = 1;           //CPU基板上のD1  ON

        main_wait(50000000);

        PORT2.PODR.BYTE = 0x08;          //PWMにつかう青 ON
        PORT5.PODR.BIT.B5 = 0;           //D21  OFF
        PORTC.PODR.BIT.B0 = 0;           //CPU基板上のD1  OFF

        main_wait(50000000);

}
```

セーブして

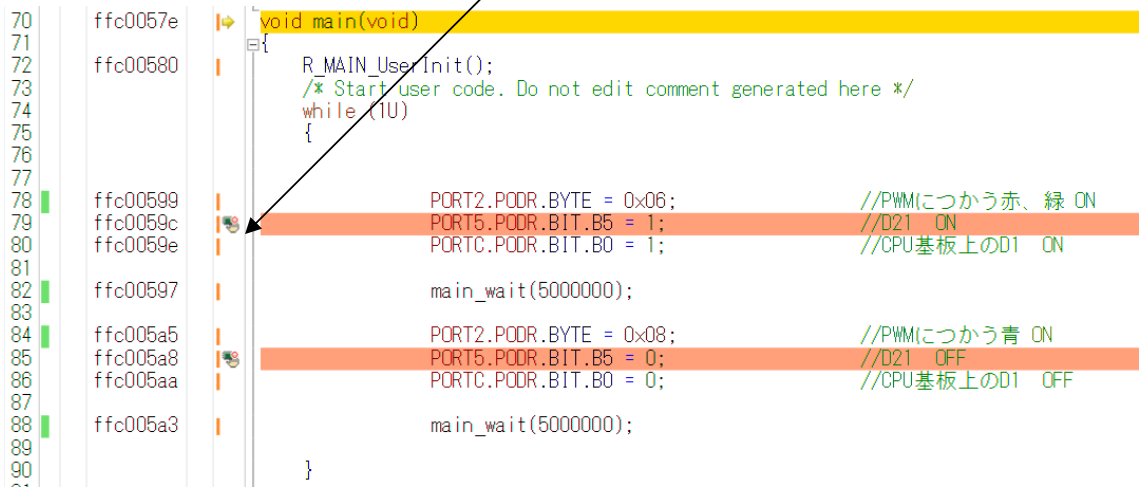


「ビルド後、デバック・ツールへプログラムを転送」をクリック。



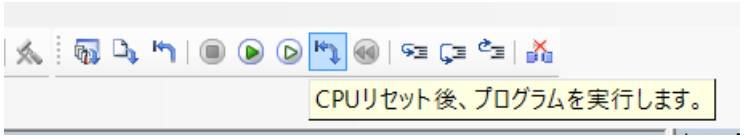
「ＣＰＵリセット後、プログラムを実行」をクリック。
ＬＥＤの点滅が先ほどより、遅くなったのが目視できましたでしょうか？

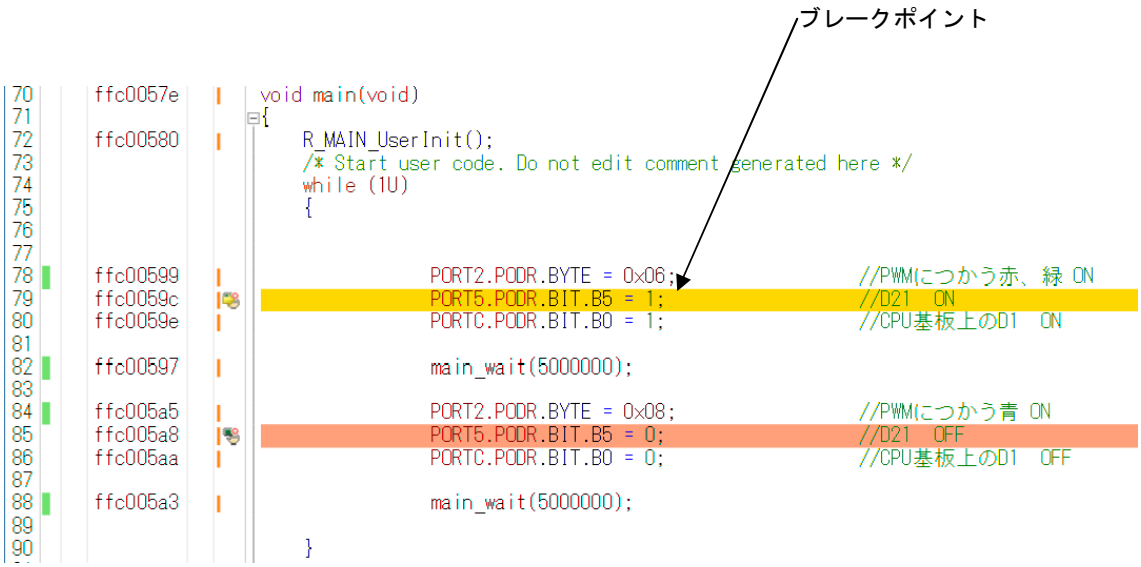
次に、ブレークポイントの設定を行ってみます。一度、プログラムを停止させます。
ブレークポイントを２か所　設定しました。ここをマウスでダブルクリック。ブレークポイント設定マークとオレンジのカーソルが表示されます。



```
70      ffc0057e    void main(void)
71                  {
72      ffc00580        R_MAIN_UserInit();
73                      /* Start user code. Do not edit comment generated here */
74                      while (1U)
75                      {
76
77
78      ffc00599                    PORT2.PODR.BYTE = 0x06;             //PWMにつかう赤、緑 ON
79      ffc0059c                    PORT5.PODR.BIT.B5 = 1;              //D21  ON
80      ffc0059e                    PORTC.PODR.BIT.B0 = 1;              //CPU基板上のD1  ON
81
82      ffc00597                    main_wait(5000000);
83
84      ffc005a5                    PORT2.PODR.BYTE = 0x08;             //PWMにつかう青 ON
85      ffc005a8                    PORT5.PODR.BIT.B5 = 0;              //D21  OFF
86      ffc005aa                    PORTC.PODR.BIT.B0 = 0;              //CPU基板上のD1  OFF
87
88      ffc005a3                    main_wait(5000000);
89
90                  }
```

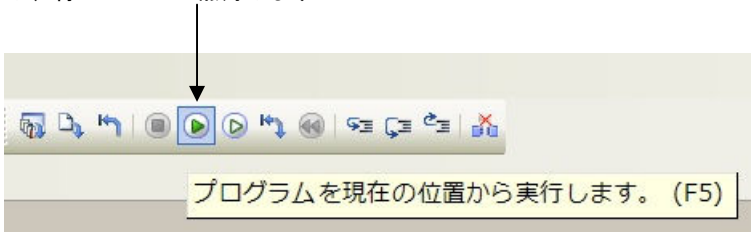
「ＣＰＵリセット後、プログラムを実行」します。

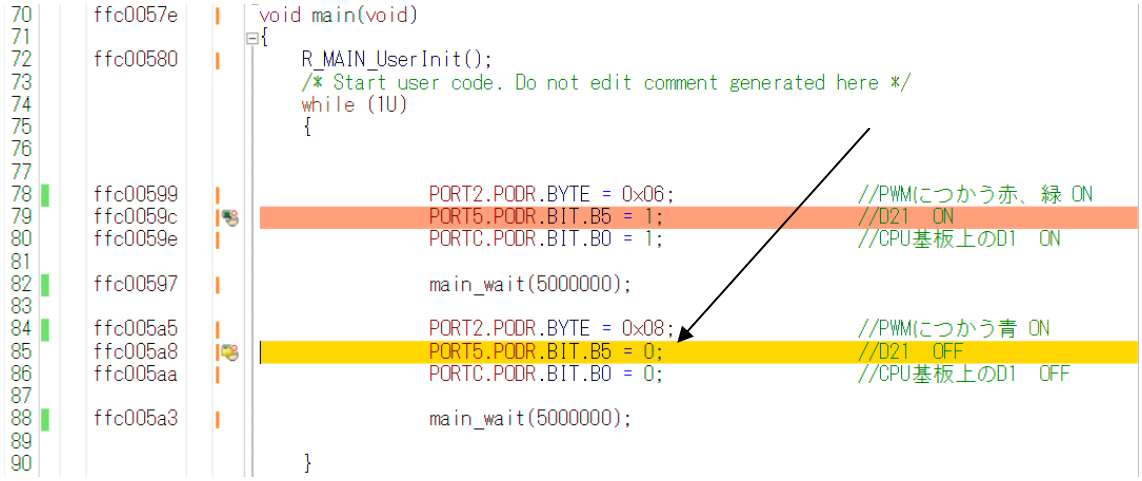




プログラムはブレークポイントで停止し、黄色いカーソル（現在のプログラムカウンタの位置）で停止します。カーソルのある行はまだ実行されていません。その１行前のＰＯＲＴ２．ＰＯＤＲ．ＢＹＴＥ＝０ｘ０６；命令によりＰＯＲＴ２　＝　０ｘ０６となるので、赤と緑のＬＥＤが点灯します。理由、詳細はｓａｍｐｌｅ１の解説で行います。

更に「プログラムを現在の位置から実行」をクリックすると、もう一つのブレークポイントで停止し、ＰＯＲＴ２．ＰＯＤＲ．ＢＹＴＥ＝０ｘ０８；命令実行によりＰＯＲＴ２は先ほどの赤と緑のＬＥＤは消灯し、青のＬＥＤが点灯します





以上が、プログラムのコンパイル、Ｅ１へのダウンロード、実行、修正、ブレークポイント設定、動作の概要です。

ｂ：新しいプログラムを作る

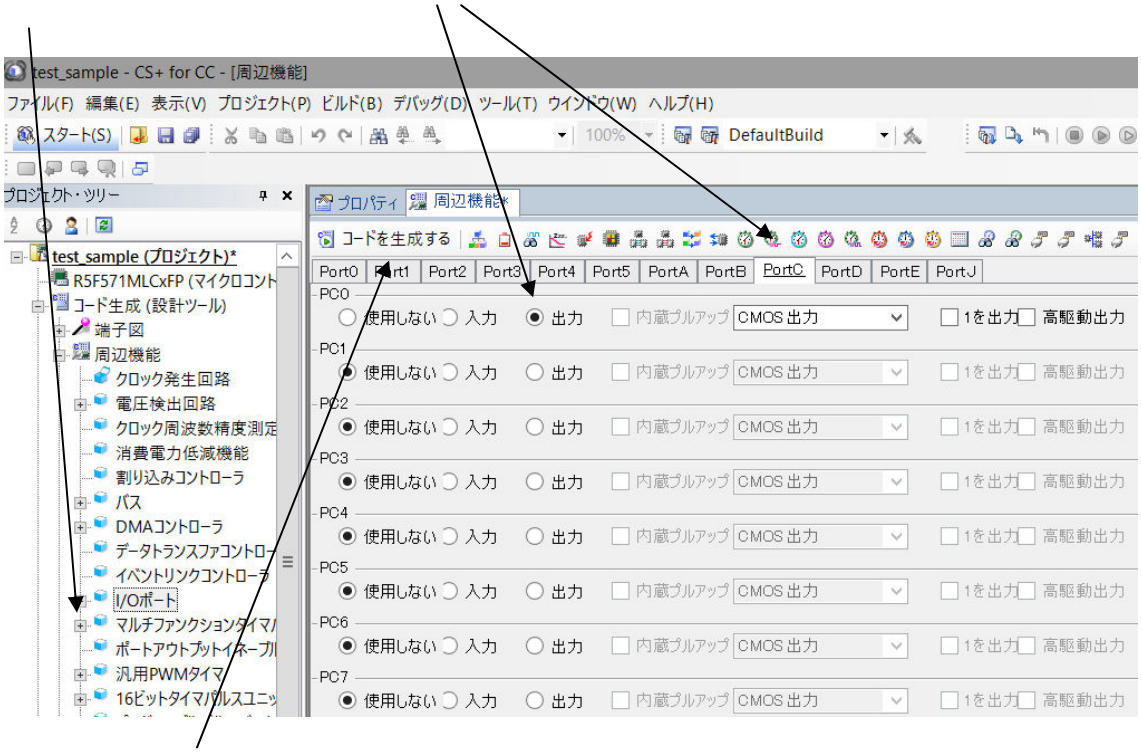
省略

ｓａｍｐｌｅ１．ｃを参考にもっとも簡単なプログラムを記入してみます。基板上のＬＥＤを点滅させるプログラムです。基板上のＬＥＤ　Ｄ１はＰＯＲＴＣのＰＣ０に接続されています。

ｂ－１：初めにＩＯポートの設定

ここが従来のＲＸマイコン開発と変わった部分になります。コード生成ツールが完備してＩＯ設定に限らず、あらゆるペリフェラルの初期設定をプログラムレスで開発できるようになりました（２０１６．９）。ＲＬ７８のＣＳ＋開発と同じように行えます。

ここではＰＯＲＴＣのＰＣ０を出力設定します。コード生成（設計ルーツ）→　周辺機能　→Ｉ／Ｏポートを選択。ＰＯＲＴＣのＰＣ０を出力にチェック。



これで「コードを生成する」をクリックするとＰＯＲＴＣのＰＣ０を出力に設定する初期化プログラムが自動生成されます。

ｂ－２：プログラムの書き込み、書く場所の注意

ＬＥＤ　Ｄ１が接続されているＰＯＲＴＣのＰＣ０をＯＮ，ＯＦＦさせるプログラムを書き込みます。

```
/***********************************************************************
Global variables and functions
***********************************************************************/
/* Start user code for global. Do not edit comment generated here */
void lwait(long wdata)
{
        while(wdata != 0)
        {
                wdata--;
        }
}
/* End user code. Do not edit comment generated here */

void R_MAIN_UserInit(void);
/***********************************************************************
* Function Name: main
* Description  : This function implements main function.
* Arguments    : None
* Return Value : None
***********************************************************************/
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */

    while (1U)
    {
        PORTC.PODR.BIT.B0 = 1;
        lwait(500000);
        PORTC.PODR.BIT.B0 = 0;
        lwait(500000);
    }
    /* End user code. Do not edit comment generated here */
}
```

このときに、必ず

/* Start user code for global. Do not edit comment generated here */

プログラム、ｄｅｆｉｎｅ等定義、ｉｎｔ，ｓｈｏｒｔ等変数定義等　ユーザーが書くもの全て

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

の間にプログラムや定義文を書くようにしてください。これは新しいペリフェラルを追加したり、変更したりした場合、新たに「コード生成」させる必要がありますが、このコメントサンドイッチ中のコードは「コード生成」しても消えません。それ以外は消えますので、注意が必要です。

ビルド後、「デバックツールへダウンロード」　をクリックするとノーエラーでコンパイルされ、ダウンロード、実行されますが、ＬＥＤは光りません。

ｂ－３：パワーオンリセットを設定する

省略

ｂ－４：デバッカの設定がデフォルトはエミュレータなので注意

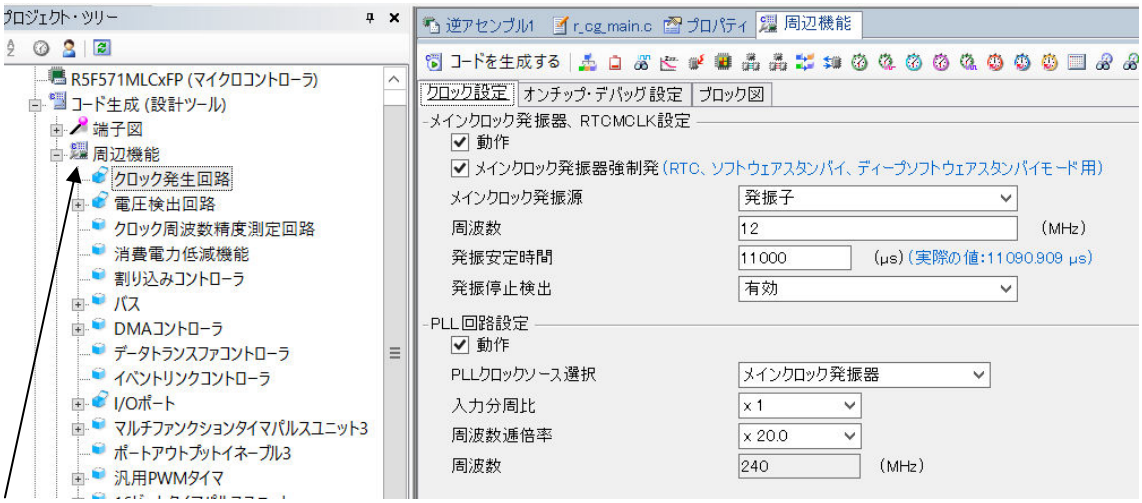
省略

ｂ－　５：Ｅ１から電源供給

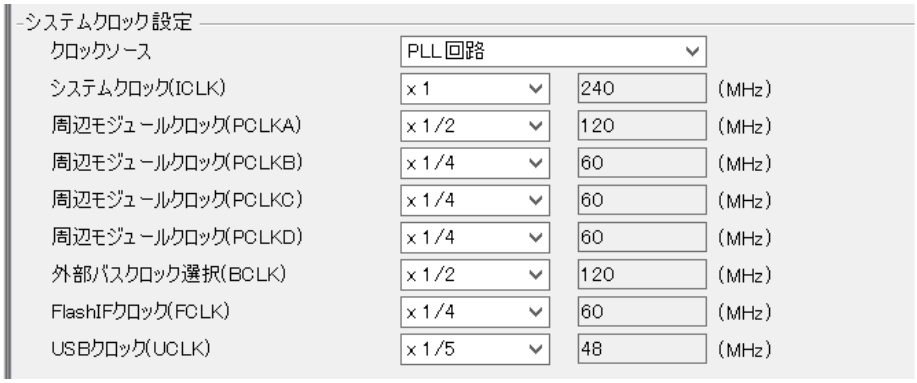
省略

ｂ－６：クロック発生回路を設定する必要があります

ここから



クロック発生をクリック。クロック設定で上記のように設定します。

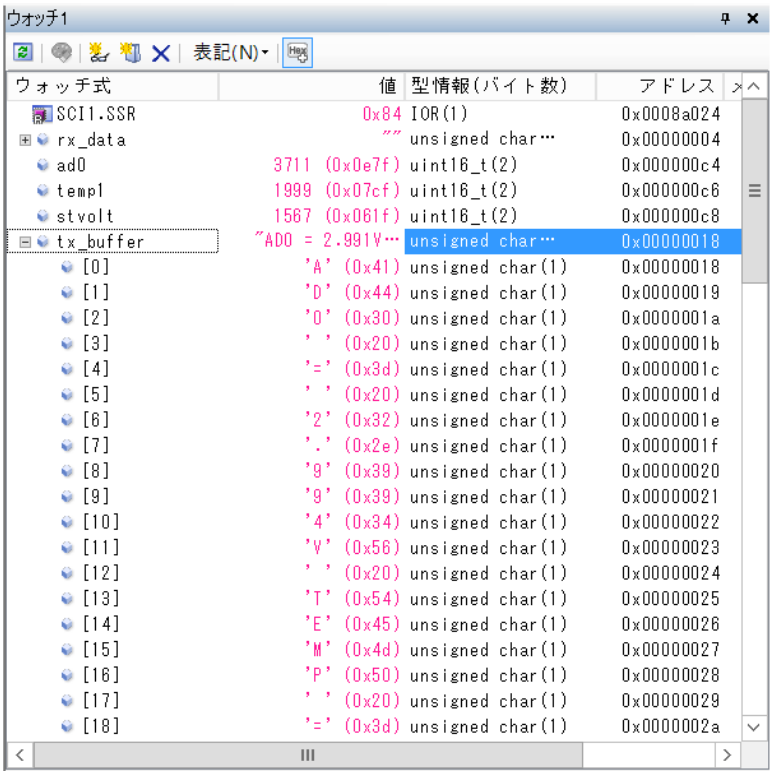


更に、各々の部分のクロックを上記のように設定します。「コード生成」を行い、全てのファイルを変更、コンパイル、ダウンロード、実行　でＣＰＵはやっと２４０ＭＨｚで動作し、ｓａｍｐｌｅ１とほぼ同じ間隔でＬＥＤが点滅するのが確認できます。

ｃ：その他

ｃ－１：動作中に変数の変化を見るには？

省略



ｃ－２：サンプルを走らせるときにｖｅｃｔ．ｈの重複するアドレスを削除

省略

ｃ－３：三角関数ｍａｔｈ．ｈはインクルードもＣＳ＋の設定も必要

省略

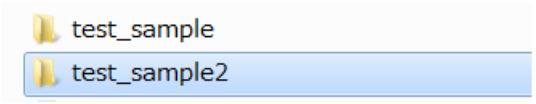
ｃ－４：割り込みが入っているか？周期は？簡単なチェック方法

省略

ｃ－５：既存のプログラムを雛形として新しいプログラムを作る

省略

今まで見てきたように新規でプロジェクトを設定すると、新たに設定しなくてはならない項目が多く、従来のプロジェクトをもとに新しいプロジェクトを作成できると便利です。ｔｅｓｔ＿ｓａｍｐｌｅを元に製作してみます。



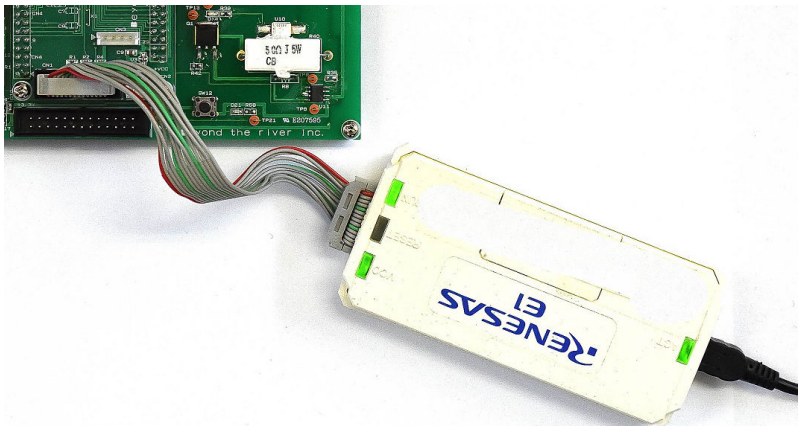
ｃ－６：コード生成と見落しがちな注意点

省略

コード生成で変更したつもりになっていても、これを忘れると変わっていないので、ご注意下さい。

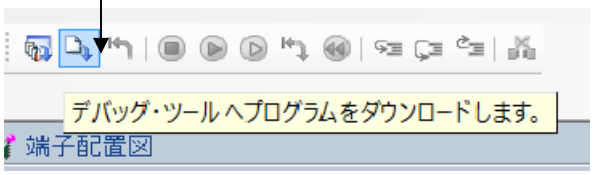
２．サンプルプログラム

初めにＣＮ１にＥ１のケーブルを差します。Ｅ１はＣＳ+　ｆｏｒ　ＣＣが動作しているパソコンのＵＳＢに接続して下さい。

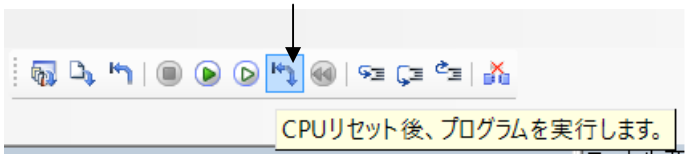


次に添付のＡＣアダプターをＪ１に差して下さい。ＡＣアダプタをＡＣ１００Ｖ　コンセントに挿入し、ＬＥＤ　Ｄ２４　が緑（９Ｖ）、ＬＥＤ　Ｄ２５が赤（３．３Ｖ）に点灯すれば正常です。

サンプルプログラムは全てコンパイル、動作確認済みです。「デバックツールへプログラムをダウンロード」後



ＣＰＵリセット後、実行で動作します。



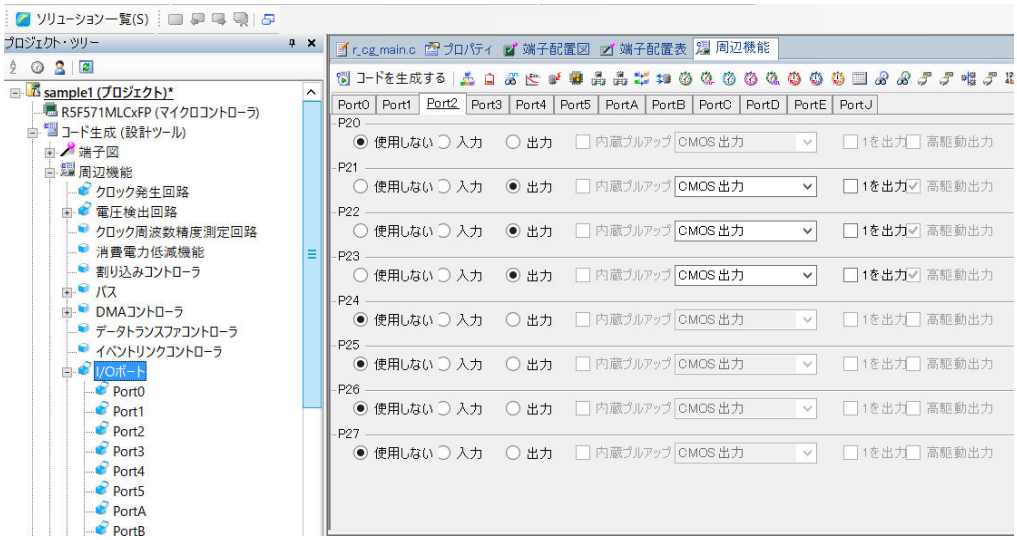
ｓａｍｐｌｅ１　ポートのＯＮ/ＯＦＦ

【　概要　】

ＣＰＵ基板上のＬＥＤ　Ｄ１（ポートＰＣ０）、Ｄ２１（ポートＰ５５）をビット命令で、３色のＬＥＤ　Ｄ１６、Ｄ１７、Ｄ１８（ポートＰ２１，２２，２３）をバイト命令で、ポートのＯＮ/ＯＦＦにより点滅を繰り返します。

【周辺機能の説明】

サンプルプログラムは周辺機能→Ｉ／Ｏポート、ポートＰＣ０、Ｐ５５、Ｐ２１，２２，２３を出力に設定し、「コード生成」してあります。この機能により、ユーザーはポートの初期設定を文章で記入する必要が無く、「コード生成」で自動的に作成され、電源投入時、自動的に実行されます。



【　プログラム　】

```
/***********************************************************************************************
Global variables and functions
***********************************************************************************************/
①/* Start user code for global. Do not edit comment generated here */
②void main_wait(long ltime)
{
        while(ltime != 0)
        {
                ltime--;
        }
}
/* End user code. Do not edit comment generated here */

void R_MAIN_UserInit(void);
/***********************************************************************************************
* Function Name: main
* Description   : This function implements main function.
* Arguments     : None
```

```
* Return Value : None
***********************************************************************************************/
void main(void)
{
    R_MAIN_UserInit();
 ③   /* Start user code. Do not edit comment generated here */
     while (1U)
    {
④                   PORT2.PODR.BYTE = 0x06;                  //PWMにつかう赤、緑　ON
                    PORT5.PODR.BIT.B5 = 1;                   //D21　ON
                    PORTC.PODR.BIT.B0 = 1;                   //CPU基板上のD1　ON

⑤                   main_wait(5000000);


⑥                   PORT2.PODR.BYTE = 0x08;                  //PWMにつかう青　ON
                    PORT5.PODR.BIT.B5 = 0;                   //D21　OFF
                    PORTC.PODR.BIT.B0 = 0;                   //CPU基板上のD1　OFF

⑦                   main_wait(5000000);

    }

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

【　解説　】

①/* Start user code for global. Do not edit comment generated here */

ユーザープログラム

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

先も書きましたが、ユーザープログラムはこのコメントの内側に作成してください。でないと、「コード生成」を行うと消えてしまいます。

```
②void main_wait(long ltime)
{
        while(ltime != 0)
        {
                ltime--;
        }
}
```

ＬＥＤのＯＮ，ＯＦＦを人間の目で確認するためには時間の早すぎるＯＮ，ＯＦＦではだめで、数１００ｍｓｅｃの時間（ウエイト）を作るためのプログラムです。

```
③   /* Start user code. Do not edit comment generated here */
     while (1U)
```

{

プログラムは/* Start user code.　以下に記入してください。

④

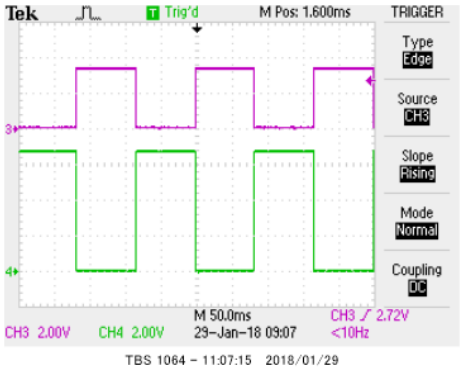
```
PORT2.PODR.BYTE = 0x06;                    //PWMにつかう赤、緑　ON
PORT5.PODR.BIT.B5 = 1;                     //D21　ON
PORTC.PODR.BIT.B0 = 1;                     //CPU基板上のD1　ON
```

バイト命令で０ｘ０６＝００００１１０Ｂです。１が立つビットはＰ２１とＰ２２で、それぞれの

Ｐ２７	Ｐ２６	Ｐ２５	Ｐ２４	Ｐ２３	Ｐ２２	Ｐ２１	Ｐ２０
０	０	０	０	０	１	１	０

ポートに１　＝　３．３Ｖが出力され、トランジスタＱ２、Ｑ３がＯＮし、赤、緑のＬＥＤが点灯します。

ＰＯＲＴ５のＢ５、ＰＯＲＴＣのＰＣ０に繋がれているＬＥＤ　もビット命令’１’で電流が流れて点灯します。ＣＨ３はＴＰ１９　Ｑ４トランジスタをドライブしている信号、ＣＨ４はＱ４コレクタ側の信号です。



⑤

```
main_wait(5000000);
```

５０００００００という数を減算して０になるまでの間、ここで時間が消費されます。

⑥

```
PORT2.PODR.BYTE = 0x08;                    //PWMにつかう青 ON
PORT5.PODR.BIT.B5 = 0;                     //D21　OFF
PORTC.PODR.BIT.B0 = 0;                     //CPU基板上のD1　OFF
```

０ｘ０８を出力しています。
０ｘ０８＝００００１０００Ｂです。ＰＣ３のみ１でＰ２３に接続された青のＬＥＤ　Ｄ１８がＯＮします。

ＰＣ７	ＰＣ６	ＰＣ５	ＰＣ４	ＰＣ３	ＰＣ２	ＰＣ１	ＰＣ０
０	０	０	０	１	０	０	０

ＰＯＲＴ５のＰ５５に繋がれているＬＥＤ　Ｄ２１もデータ’０’で電流が止まり、消灯します。
ＰＯＲＴＣのＰＣ０に繋がれているＬＥＤ　Ｄ１もデータ’０’で電流が止まり、消灯します。

⑦

```
main_wait(5000000);
```

消灯している間も点灯同様に時間を保持しています。

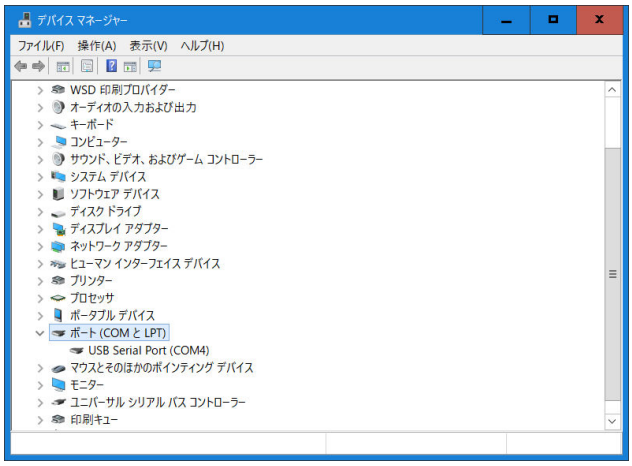
ｗａｉｔ（　）に入れる数を変えたり、一方だけ変えて、セーブ、コンパイル、ＣＰＵリセット後、実行でＬＥＤの点灯時間が変わるのを確認できます。

２－２　ｓａｍｐｌｅ２　ＳＩＯ（ＵＳＢ）　ＵＳＢ経由でパソコンとやりとり

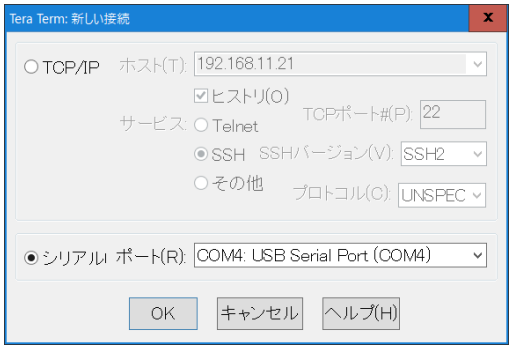
【　概要　】

ＵＳＢ出力をパソコンと接続し、データのやり取りを行います。お手数ですが、テラタームやハイパーターミナルなどのターミナルプログラムを使用しますので、無い方は、ネットで検索し、インストゥール願います。例ではテラタームで行います。ボーレートは３８４００ｂｐｓに設定して下さい。

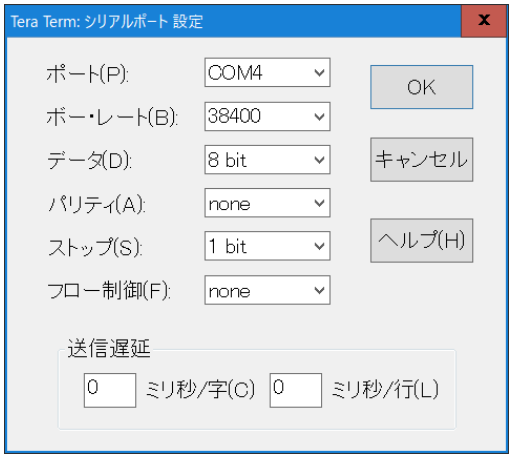
スタート→右クリック→デバイスマネージャー　→　ポート（ＣＯＭとＬＰＴ）でＵＳＢ　Ｓｅｒｉａｌ　Ｐｏｒｔ（ＣＯＭｘｘ）があることを確認して下さい（Ｗｉｎｄｏｗｓ１０の場合）。例ではＣＯＭ４となっています。



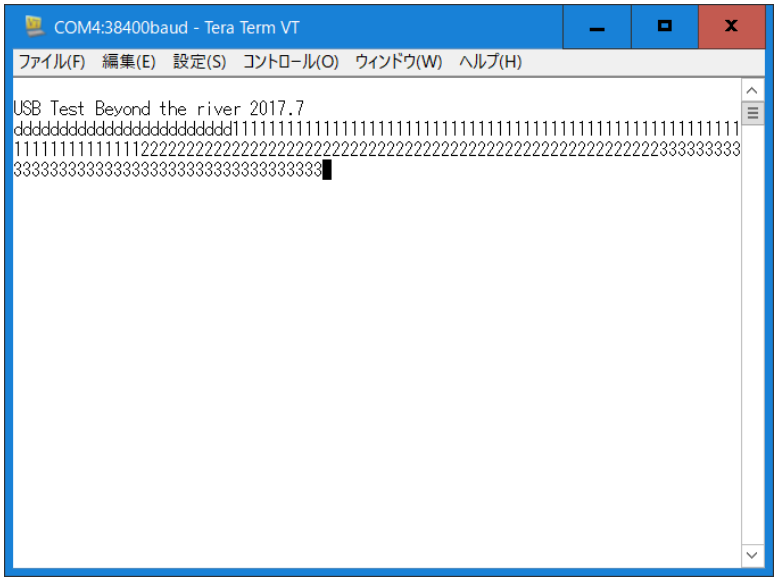
Ｔｅｒａ　Ｔｒｅｍをシリアルポート　ＣＯＭ４　→ＯＫとします。



設定→シリアルポート→ボーレート３８４００として下さい。



ＣＳ＋でｓａｍｐｌｅ２を開き、デバック・ツールへプログラムダウンロード→ＣＰＵリセット後、プログラム実行。



ＵＳＢ　Ｔｅｓｔ、、、　と表示され、ＰＣのキーボードを何か押すたびに、押した文字が表示されると動作としてはＯＫです。

プログラムはパソコンのキーボードを押した文字がＣＰＵ基板に送信され、それを返信（エコーバック）し、表示されるようになっています。

【周辺機能の説明】

省略

【　プログラム　】

省略

【　解説　】

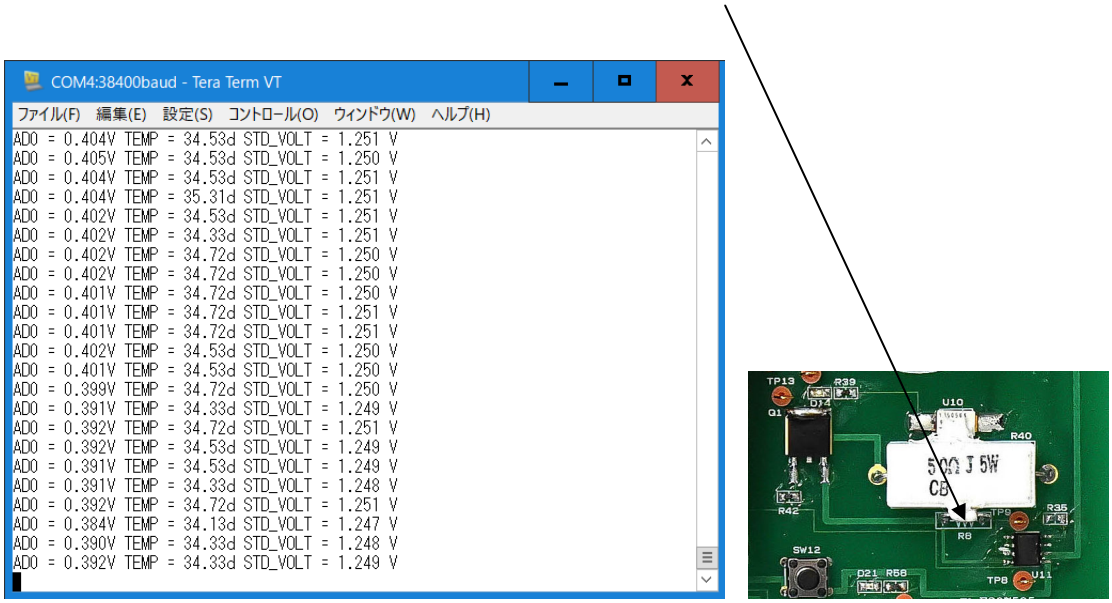
省略

２－３　ｓａｍｐｌｅ３　Ａ／Ｄ変換をＵＳＢ出力

【　動作概要　】

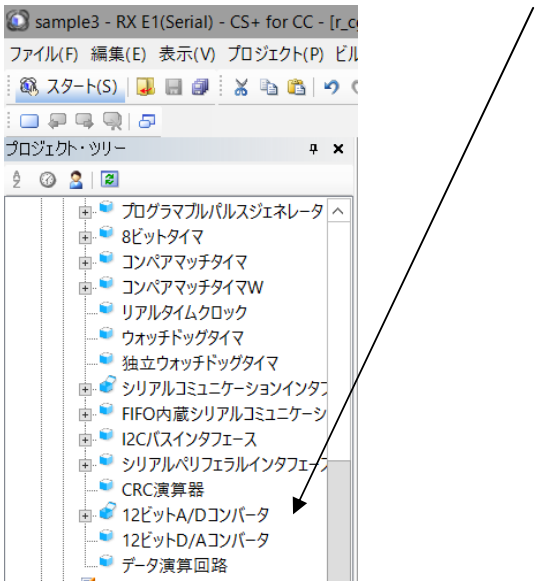
ＲＸ７１Ｍ　ＣＰＵは２ユニットの１２ｂｉｔ　ＡＤコンバータを持っています。ユニット０は最大８ｃｈのアナログ入力を選択できます。ユニット１は最大２１チャンネルのアナログ入力、内部温度、内部基準電圧を選択できます。

ハードウエアはサーミスタが繋がれているＡＮ０００（Ｐ４０　ＣＮ４　１６番)、ＣＰＵ内部温度、基準電圧を入力とし、Ａ／Ｄ変換した値をＵＳＢからパソコンに送り、表示しています。
また、温度の変化はＲ８サーミスタを指で触ると加熱され、抵抗値が下がり、分圧されている電圧値が下がるのが目視出来ます。

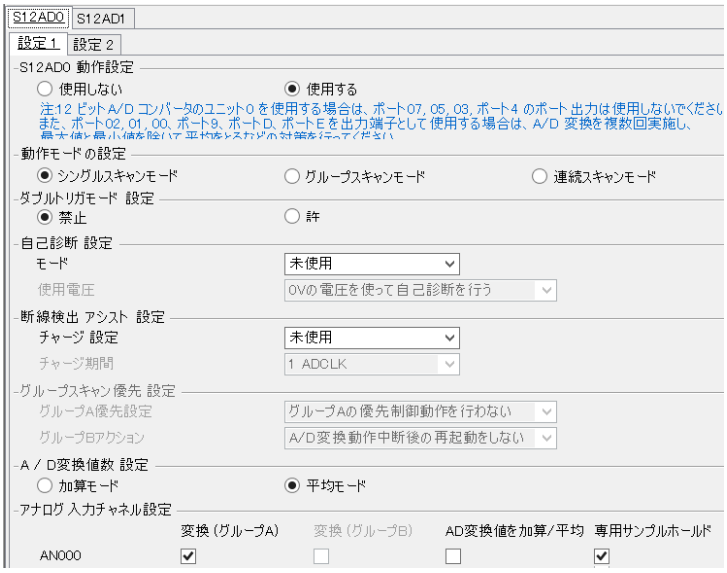


【周辺機能の説明】

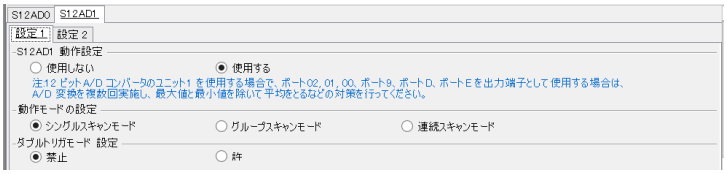
ＳＣＩ２に加えて、１２ビットＡ／Ｄコンバータを使用しています。



ＡＮ０００入力のための設定です。Ｓ１２ＡＤ０を選択しています。



温度センサ、内部基準電圧を読み込むためのＳ１２ＡＤ１の設定です。



レジスタ等の詳細はハードウエアマニュアル　ＲＥＶ１．００　２６７２頁等参照。

【　プログラム　】

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */

    R_SCI2_Start();                 //SCI2動作開始
```

```
①      R_S12AD0_Start();          //AD0動作開始
②      R_S12AD1_Start();          //AD1動作開始

       R_SCI2_Serial_Receive(rx_data,1);
       rx_flg2 = 0;
       tx_end_flg2 = 0;


       PORTC.PODR.BIT.B0 = 0;              //LED1 OFF

       R_SCI2_Serial_Send(String_0,37);    //Opening message
       tx_end_wait2();                     //送信終了まち


       while (1U)
       {

③              S12AD.ADCSR.BIT.ADST = 1;                  //ADスタート
               S12AD1.ADCSR.BIT.ADST = 1;                 //AD1スタート

④              while(S12AD.ADCSR.BIT.ADST)
                              ;

                              ad0 = S12AD.ADDR0;          //AD値


⑤              while(S12AD1.ADCSR.BIT.ADST)
                              ;

                      temp1 = S12AD1.ADTSDR;              //温度
                      stvolt = S12AD1.ADOCDR;             //基準電圧

⑥                     fdata1 = ad0/(4095/3.3);            //データ→電圧Ｖ換算

⑦                     fdata2 = temp1/(4.095/3.3);         //データ電圧ｍＶ換算
                      fdata2 /= 4.1;                      //温度＝（データ／４．１ｍＶ）－277.4
                      fdata2 -= 277.4;                    //0℃電圧１１３７．５ｍＶ／４．１ｍＶ

⑨                     fdata3 = stvolt/(4095/3.3);         //データ→電圧換算  type1.25V 1.20～1.30V

⑩          sprintf(tx_buffer,"AD0 = %.3fV TEMP = %.2fd STD_VOLT = %.3f V¥n¥r",fdata1,fdata2,fdata3);
⑪                    R_SCI2_Serial_Send(tx_buffer,sizeof(tx_buffer));      //データをＵＳＢ出力
                      tx_end_wait2();                     //送信終了まち
```

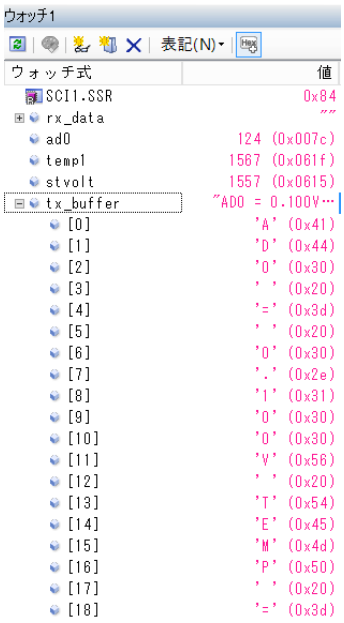
⑫

```
                    PORTC.PODR.BIT.B0 = 1;      //LED1 ON
                    main_wait(10000000);
                    PORTC.PODR.BIT.B0 = 0;      //LED1 OFF
                    main_wait(10000000);
```

}

【　解説　】

省略

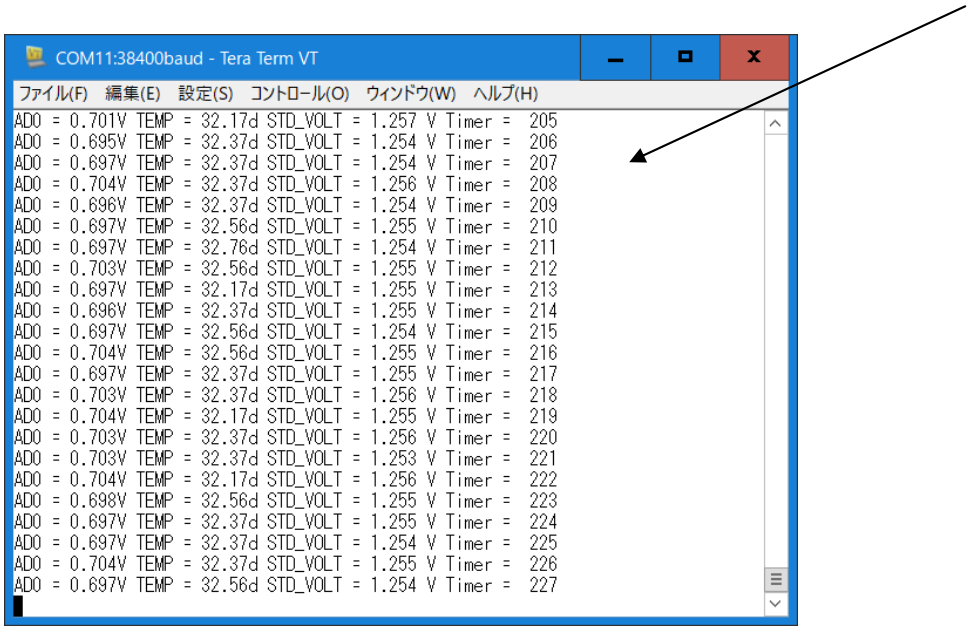


以上が、ＡＤデータの取り込み、ＵＳＢ出力、制御の簡単なプログラムになります。

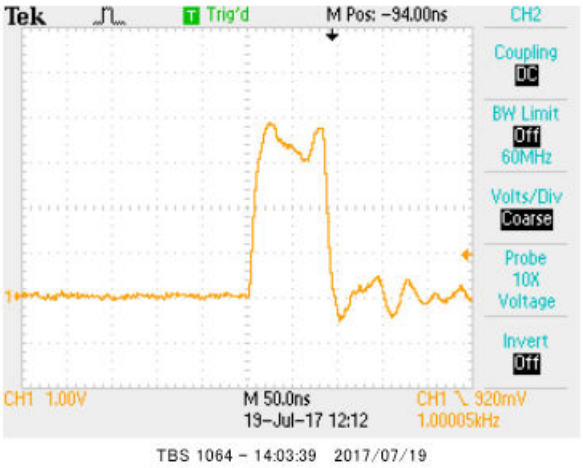
２－４　ｓａｍｐｌｅ４　割り込み

【　動作概要　】

ｓａｍｐｌｅ４を動作させます。ここでは定周期割り込みについてサンプルを示します。ｓａｍｐｌｅ３のループ時間を割込みを使って正確に１秒とし、データを出力しています。新たに経過秒と出力を示しています。

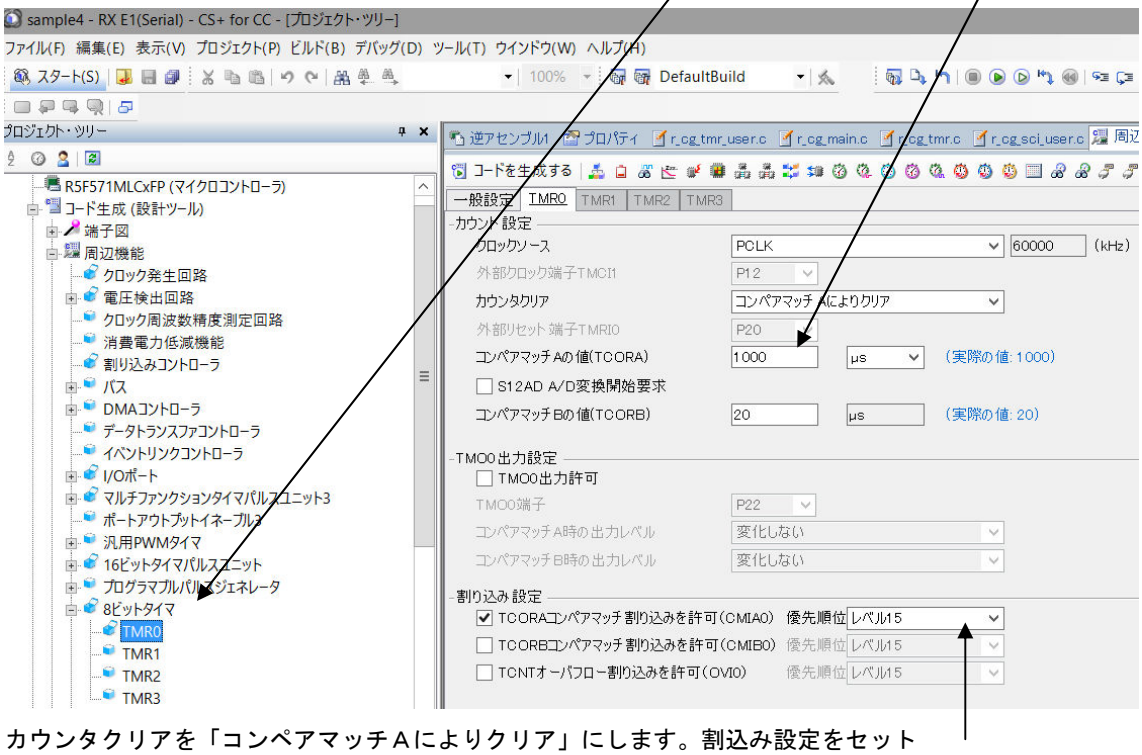


オシロスコープがあればＰＣ０　ＣＮ６　２０番を観測すると、以下のような幅１００ｎｓｅｃ、１ｍｓｅｃ周期（１．００００５ＫＨｚ）の波形が観測できます。



【周辺機能の説明】

ｓａｍｐｌｅ３で使用している周辺機能に加えて、８ビットタイマ　ＴＭＲ０を使って、１ｍｓｅｃ　定周期割り込みを実現させています。



カウンタクリアを「コンペアマッチＡによりクリア」にします。割込み設定をセット

「コード生成」で
ｒ＿ｃｇ＿ｔｍｅｒ．ｃ
ｒ＿ｃｇ＿ｔｍｅｒ＿ｕｓｅｒ．ｃ
２つのプログラムが自動作成されます。

【　プログラム　】

省略

【　解説　】

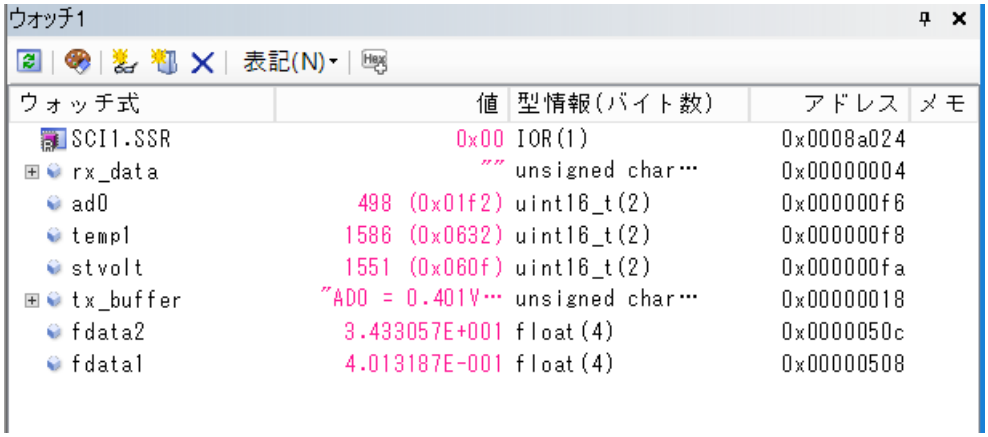
省略

このように、割り込みで時間を作り、メインで使用すれば、極めて正確なタイマーが多数作成可能です。定周期割り込みはその中に様々なプログラムを作成することも可能で、機能的なプログラム作成に不可欠な知識、要素です。

⑦ sprintf(tx_buffer,"AD0 = %.3fV TEMP = %.2fd STD_VOLT = %.3f V Timer = %4d ¥n¥r",fdata1,fdata2,fdata3,timer);

サーミスタの電圧換算ＡＤ値、ＣＰＵ内部温度、基準電圧に加えて、経過時間(秒）の４つの情報をＵＳＢに出力しています。

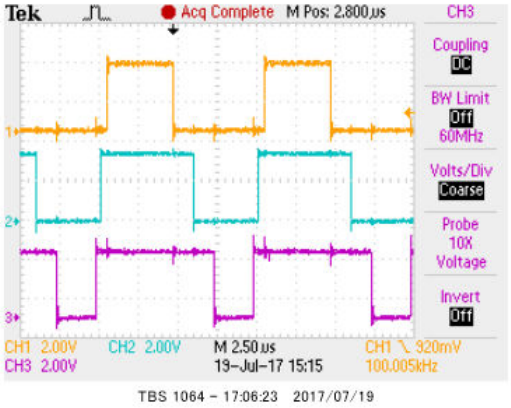
なお、各種変数、バッファの内容はプログラム実行中にウオッチ窓でリアルタイムに見ることが出来ます。

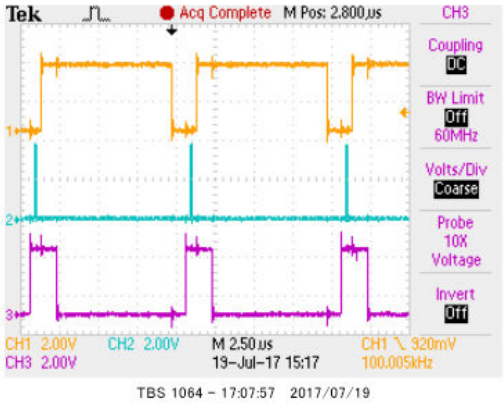


２－５　ｓａｍｐｌｅ５　PWM出力

【　動作　】

汎用PWMタイマを使って、PWM波形を作ります。GTIOC0A端子　P23　CN5　20番、GTIOC1A端子　P22　CN5　19番、GTIOC2A端子　P21　CN5　18番に出力されます。それぞれ、青、緑、赤のLEDがドライブされます。波形はチェックピンTP17，18，19を観測したものです（詳細は回路図ご参照ください）





ＰＷＭ駆動により、Ｒｅｄ，Ｇｒｅｅｎ，Ｂｌｕｅ＝ＲＧＢの色の混合比が変わりますので、丸い白色のドームを被せて見たりすると色が混色し、単色が変化するように見えるかもしれません。ＴＶやスマホのフルカラー表示は、光の３原色　ＲＧＢの強度の違いで様々な色を見せています。

ＰＷＭ波形は上図のように、周期が変わらず、設定値によってＨ、Ｌの幅の比率が変化します。この出力でＬＥＤやモーターをドライブすると明るさや速度を効率よく変えることが出来るので、現代では様々な用途に使われています。

【周辺機能の説明】

省略

【　プログラム　】

省略

【　解説　】

省略

２－６　ｓａｍｐｌｅ６　三角、対数、平方根関数　関数演算速度を検証

【　概要　】

ｌｏｇ、ｓｉｎ、√　演算を行い、演算結果の確認とその速度を測定します。

【周辺機能の説明】

ＣＳ＋側の設定はｃ－３↓をご参照ください。
ｃ－３：三角関数ｍａｔｈ．ｈはインクルードもＣＳ＋の設定も必要

【　プログラム　】

```
;
①#include <math.h>

②double d1,d2,d3;
③short s1,s2,s3;

④#define PI 3.14159265

;

void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */

//重複省略

⑤                  PORTC.PODR.BIT.B0 = 1;          //時間マーカーON
⑥                  d1 = log10(10000);
⑦                  PORTC.PODR.BIT.B0 = 0;          //時間マーカーOFF

⑧                  d2 = sin((PI/180)*45);
                   PORTC.PODR.BIT.B0 = 1;          //時間マーカーON

⑨                  d3 = sqrt(2);
                   PORTC.PODR.BIT.B0 = 0;          //時間マーカーOFF

⑩                  s1 = d1;
                   s2 = d2;
                   s3 = d3;

        while (1U)
    {
```

```
    }
}
```

【　解説　】

①#include <math.h>　　//sqr 等演算を行うのに必要　math.h有効と共にこの表記も必要。

三角関数や、対数、平方根を使うためにはｍａｔｈ．ｈをインクルードする必要があります。

②double d1,d2,d3;

演算結果をセーブするダブル（浮動小数点３２ビット）データです。

③short s1,s2,s3;

演算結果をキャストしてセーブするショート（１６ビット）データです。

④#define PI 3.14159265

三角関数計算で角度を入力して数値を出すために使います。

```
⑤                PORTC.PODR.BIT.B0 = 1;          //時間マーカーON
⑥                d1 = log10(10000);
⑦                PORTC.PODR.BIT.B0 = 0;          //時間マーカーOFF
```

ｄ１＝ｌｏｇ１０（１００００）を行うのですが、演算速度を測定するためにポートを使用しています。ｄ１は４になるはずです。

```
⑧                d2 = sin((PI/180)*45);
```

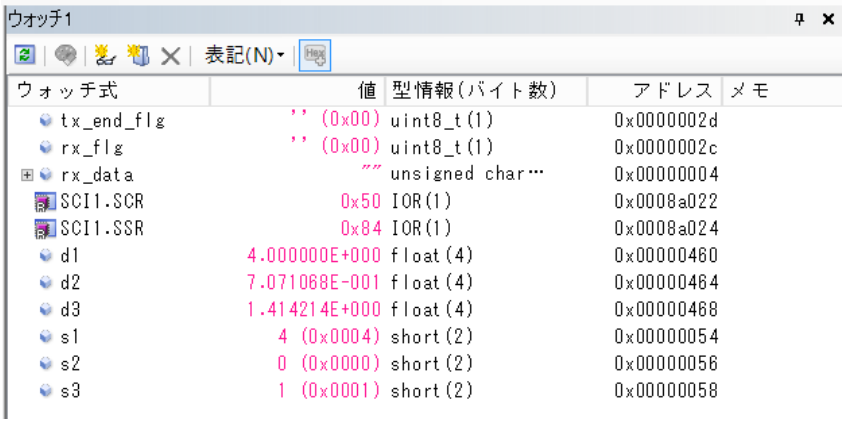
ｓｉｎ（４５°）という意味です。ｄ２＝０．７０７１０６７、、となるはずです。

```
⑨                d3 = sqrt(2);
```

√２という意味です。ｄ３＝１．４１４２１、、となるはずです。

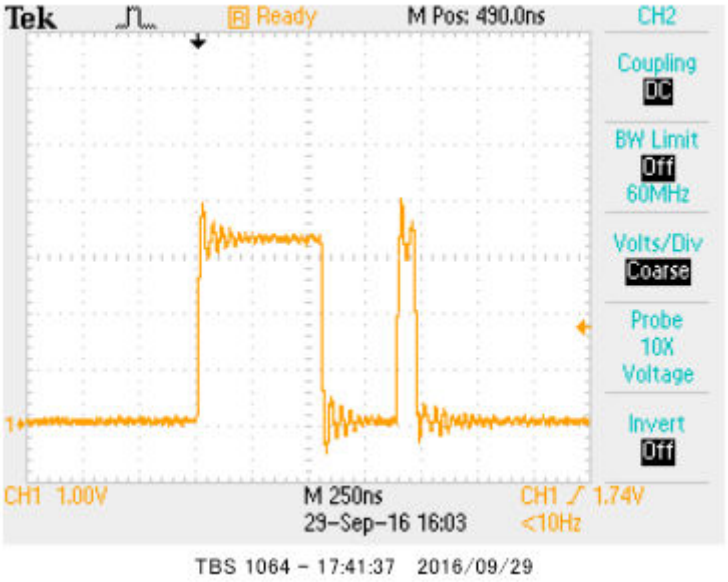
```
⑩                s1 = d1;
                 s2 = d2;
                 s3 = d3;
```

３２ビット浮動小数点データを１６ビット整数にキャストしています。それぞれ、４，０，１となるはずです。例えば演算結果をＤＡコンバータに出力する場合、浮動小数点のままでは設定できません。小数点以下何桁まで使用したいかに応じて、ｄｏｕｂｌｅデータを加工してからｓｈｏｒｔに移せば最大の精度、有効数値を得ることが出来ます。



演算結果はそれぞれｄ１、ｄ２、ｄ３に入ります。正しいですね。

演算速度ですが、ｌｏｇ１０（１００００）が約５５０ｎｓｅｃ、ｓｉｎ（４５°）が３５０ｎｓｅｃ、√２が１００ｎｓｅｃ程度かかるようでした。



例えばＲＬ７８（３２ＭＨｚ）では約２２０μｓｅｃ、ｓｉｎ（４５°）が１３０μｓｅｃ、√２が１００μｓｅｃ程度ですので、それぞれ４００倍、３７１倍、１０００倍も速いことになります。マイコンに必要な能力が演算処理速度の場合、ＲＸを使用するのが圧倒的に有利であることが分かります。ＲＸ２１ＡがＲＸ７１Ｍ、ＲＸ６３０に比べて遅いのはＦＰＵ内蔵、非内蔵の差と思われます。

ＣＰＵ別演算速度例

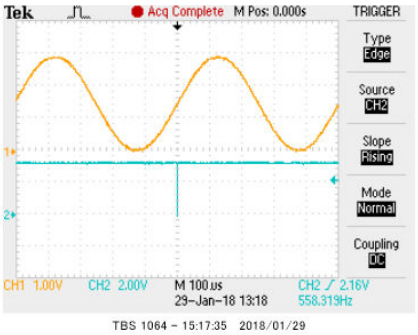
ＣＰＵ　クロック	ｌｏｇ	ｓｉｎ	√
ＲＸ７１Ｍ　２４０ＭＨｚ	５５０ｎｓｅｃ	３５０ｎｓｅｃ	１００ｎｓｅｃ
ＲＸ６３０　１００ＭＨｚ	１．８μｓｅｃ	８００ｎｓｅｃ	１．２μｓｅｃ
ＲＸ２１Ａ　５０ＭＨｚ	３３μｓｅｃ	２．５μｓｅｃ	３μｓｅｃ
ＲＬ７８　３２ＭＨｚ	２２０μｓｅｃ	１３０μｓｅｃ	１０００μｓｅｃ

２－７　ｓａｍｐｌｅ７　Ｄ／Ａにｓｉｎ演算した正弦波を出力する

【　概要　】

ＲＸ７１Ｍがもつ、１ｃｈ　１２ビットＤ／Ａにｓｉｎ演算結果（最大±１）を０－３．３Ｖに変換し出力します。正弦波オシレーターになります。サンプルでは２０７Ｈｚ～１０ＫＨｚまで周波数を変化させて、デジタルアンプを経てスピーカーＵ６に出力します。周波数変化を耳で音を聞くことが出来ます。音量はＶＲ２で調整出来ます。

波形は実行時の　上　ＴＰ４　下　ＴＰ２２を観測。



【周辺機能の説明】

省略

【　プログラム　】

省略

【　解説　】

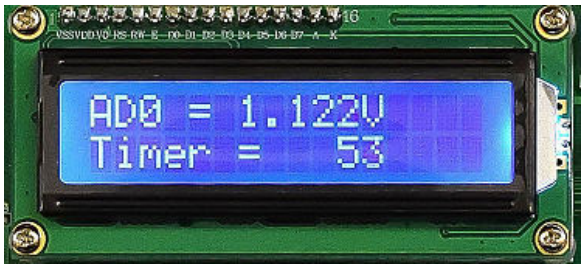
省略

格納された配列のデータを１個ずつ読み込んでＤ／Ａに出力しています。この方式で最高８２ＫＨｚ程度の正弦波が作成できています。人間の可聴帯域程度であれば十分使用可能です。

２－８　ｓａｍｐｌｅ８　液晶表示

【　概要　】

ｓａｍｐｌｅ３、４　プログラムをベースに１６文字×２行の液晶に表示させるプログラムを追加しました。簡単な液晶表示の例を示します。



【　プログラム　】

①　　lcd_disp_0();

【　解説　】

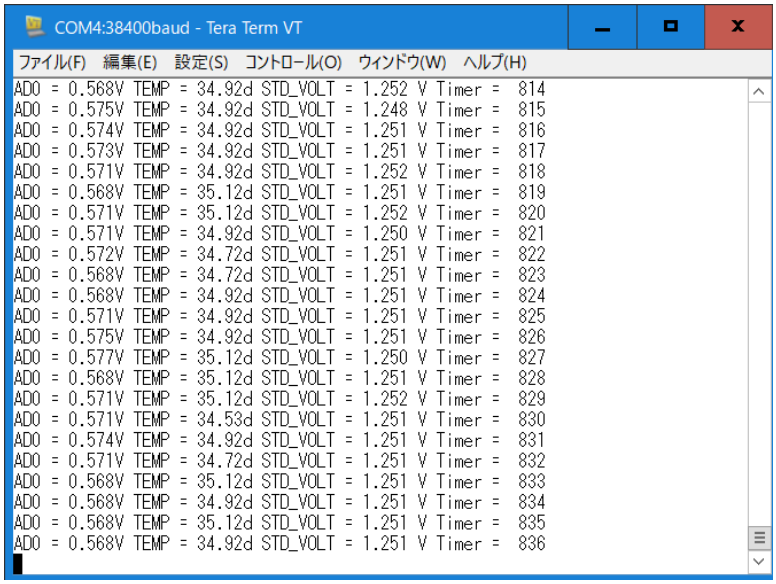
①　　lcd_disp_0();

液晶に表示させている関数ですが、内容は以下の通りです。

```
void lcd_disp_0(void)
{
        lcd_cursor_1();
        lcd_data_write(tx_buffer[0]);
        lcd_data_write(tx_buffer[1]);
        lcd_data_write(tx_buffer[2]);
        lcd_data_write(tx_buffer[3]);
        lcd_data_write(tx_buffer[4]);
        lcd_data_write(tx_buffer[5]);
        lcd_data_write(tx_buffer[6]);
        lcd_data_write(tx_buffer[7]);
        lcd_data_write(tx_buffer[8]);
        lcd_data_write(tx_buffer[9]);
        lcd_data_write(tx_buffer[10]);
        lcd_data_write(tx_buffer[11]);

        lcd_cursor_2();

        lcd_data_write(tx_buffer[46]);
        lcd_data_write(tx_buffer[47]);
        lcd_data_write(tx_buffer[48]);
        lcd_data_write(tx_buffer[49]);
        lcd_data_write(tx_buffer[50]);
        lcd_data_write(tx_buffer[51]);
        lcd_data_write(tx_buffer[52]);
```

```
        lcd_data_write(tx_buffer[53]);
        lcd_data_write(tx_buffer[54]);
        lcd_data_write(tx_buffer[55]);
        lcd_data_write(tx_buffer[56]);
        lcd_data_write(tx_buffer[57]);
}
```

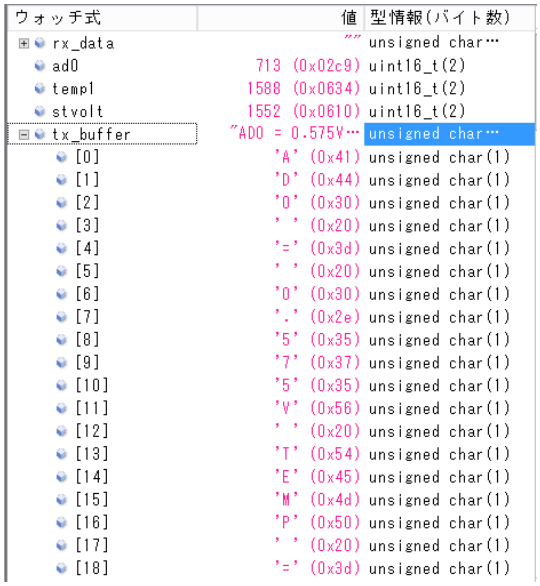
単純に tx_buffer［ｎ］にある文字を液晶に書き込んでいるだけです。



ＵＳＢには以下の命令で

sprintf(tx_buffer,"AD0 = %.3fV TEMP = %.2fd STD_VOLT = %.3f V Timer = %4d¥n¥r",fdata1,fdata2,fdata3,timer);

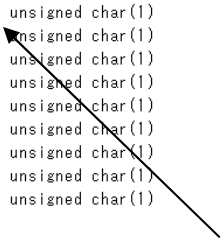
テラターム画面のように出力されているのですが、液晶は１行１６文字文字しかないので



上の行はｔｘ＿ｂｕｆｆｅｒ［０］から［１１］までを表示しています。

下の行はｔｘ＿＿ｂｕｆｆｅｒ［４６］から［５７］までを表示しています。

ウォッチ式	値	型情報(バイト数)
[37]	' ' (0x20)	unsigned char(1)
[38]	'1' (0x31)	unsigned char(1)
[39]	'.' (0x2e)	unsigned char(1)
[40]	'2' (0x32)	unsigned char(1)
[41]	'5' (0x35)	unsigned char(1)
[42]	'1' (0x31)	unsigned char(1)
[43]	' ' (0x20)	unsigned char(1)
[44]	'V' (0x56)	unsigned char(1)
[45]	' ' (0x20)	unsigned char(1)
[46]	'T' (0x54)	unsigned char(1)
[47]	'i' (0x69)	unsigned char(1)
[48]	'm' (0x6d)	unsigned char(1)
[49]	'e' (0x65)	unsigned char(1)
[50]	'r' (0x72)	unsigned char(1)
[51]	' ' (0x20)	unsigned char(1)
[52]	'=' (0x3d)	unsigned char(1)
[53]	' ' (0x20)	unsigned char(1)
[54]	'1' (0x31)	unsigned char(1)
[55]	'4' (0x34)	unsigned char(1)
[56]	'2' (0x32)	unsigned char(1)
[57]	'2' (0x32)	unsigned char(1)
[58]	'' (0x0a)	unsigned char(1)
[59]	'' (0x0d)	unsigned char(1)
[60]	'' (0x00)	unsigned char(1)
[61]	'' (0x00)	unsigned char(1)

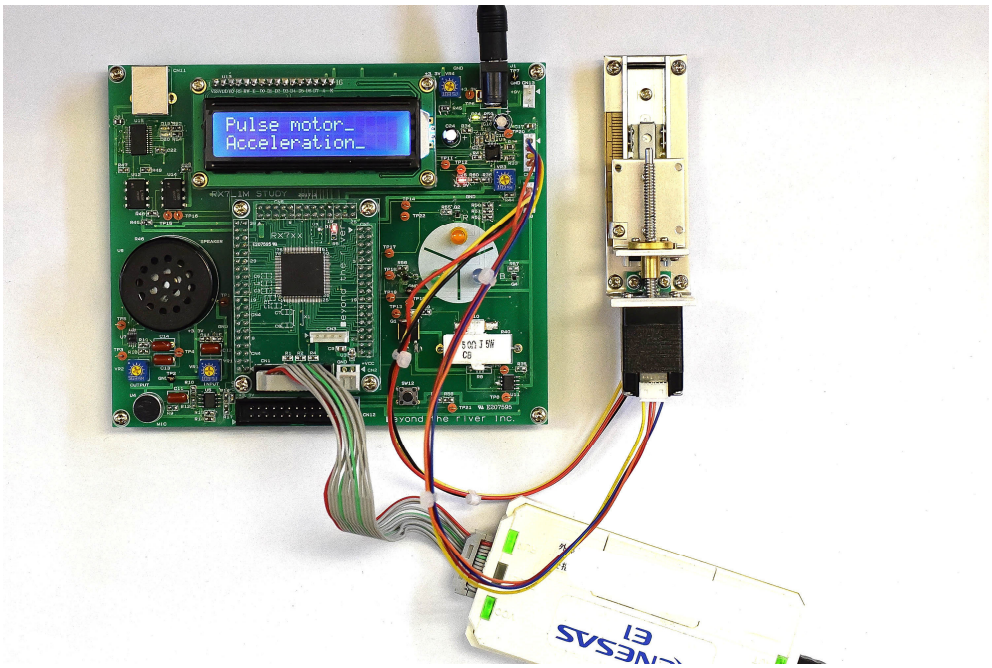


なお、通信でも液晶表示でも扱う文字はＡＳＣＩＩコードで、１を表示したいときには０ｘ３１を設定します。１を設定しても１は表示されないことに注意してください。

２－９　ｓａｍｐｌｅ９　ステージを動かす

【　概要　】

パルスモータ制御の基本を学びます。

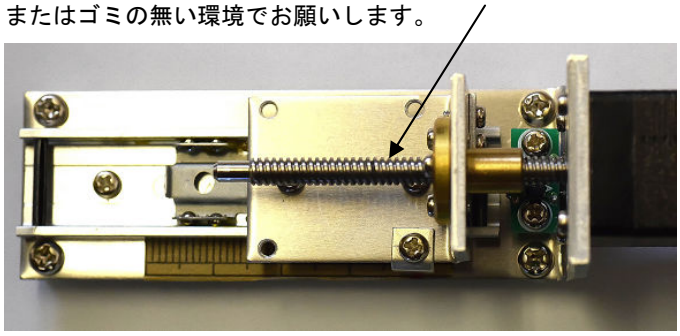


使用しているモーターは２相、ステップ角　３．６°、リードピッチ１mmです。３６０／３．６＝１００パルスで１mm移動する計算になります。

ドライバーＩＣは最大１／１６ステップまで選択できるマイクロステップドライバで、１／１６を選択した場合、１パルスで１．８／１６＝０．１１２５°、３６０／０．１１２５＝１６００パルス１mm移動することになります。

※液晶の表示は応用　ｓａｍｐｌｅ３５を動作させた時のものです。

※ステージの回転ねじの部分にゴミが入ると回転がスムースに行かなくなりますの保存は必ず付属の箱、またはゴミの無い環境でお願いします。



【　プログラム　】

省略

【　解説　】

省略

【　演習　】

省略

お問い合わせ先：
〒３５０－１２１３　埼玉県日高市高萩１１４１－１
TEL　０４２（９８５）６９８２
FAX　０４２（９８５）６７２０
Homepage：http//beriver.co.jp
e-mail：info@beriver.co.jp
有限会社ビーリバーエレクトロニクス　