# ＢＣＲＸ７＿１Ｍ　マイコン開発セット　マニュアル

第２版　２０２０．１２．１７／２０１６．８．３０　Ｅ２ lite の使用を追加

## 【　製品概要　】

本マニュアルはＢＣＲＸ７＿１Ｍ　ＣＰＵボードのソフトウエア開発を行うために必要な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されています。特に新しい統合開発環境ＣＳ＋　ｆｏｒ　ＣＣにおける開発方法について多く記述してあります。「コード生成」機能でＲＬ７８のように簡単に、ＦＰＵ内蔵で驚異の演算速度をご体験下さい。

※本ＣＰＵボード開発にはルネサスエレクトロニクス社製Ｅ１またはＥ２　ｌｉｔｅが必要です。

## 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：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－４：Ｅ１から電源供給

ｂ－５：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

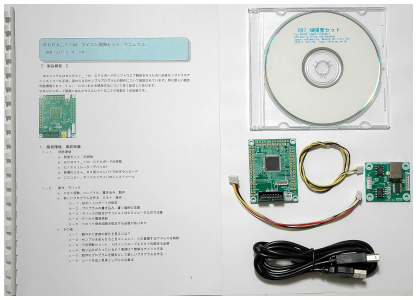
## ２．サンプルプログラム

２－１．　ｓａｍｐｌｅ１　出力ポートのＯＮ，ＯＦＦ
２－２．　ｓａｍｐｌｅ２　ＳＩＯ（ＵＳＢ）でパソコンとのやりとり
２－３．　ｓａｍｐｌｅ３　Ａ／Ｄ変換をＵＳＢ出力
２－４．　ｓａｍｐｌｅ４　割り込み
２－５．　ｓａｍｐｌｅ５　ＰＷＭ出力
２－６．　ｓａｍｐｌｅ６　三角、対数、平方根関数を使う
２－７．　ｓａｍｐｌｅ７　Ｄ／Ａにｓｉｎ演算した正弦波を出力する

## １−１．開発環境

### ａ：開発セット同梱物

ＢＣＲＸ７＿１Ｍ　ＣＰＵボード
ＳＩＯ−ＵＳＢ絶縁変換器
ＤＶＤ（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（本誌）
電源ケーブル、ＵＳＢ（フルサイズ）ケーブル
**※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またはＥ２　ｌｉｔｅは同封されておりません。別途必要です。Ｅ１は２０１９年末に製造中止となりました。**

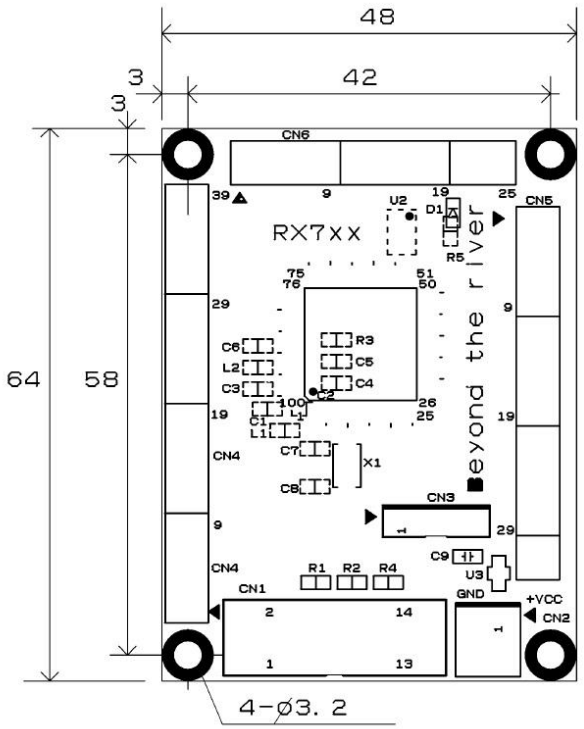


### ｂ：ＢＣＲＸ７＿１Ｍ　ＣＰＵボードの特徴（Ｒ５Ｆ５７１ＭＬＣＤＦＰ　１００ピン搭載）

●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。

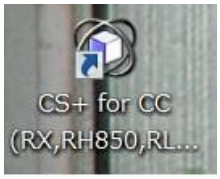
## 基板大きさ（部品面）




# 省略

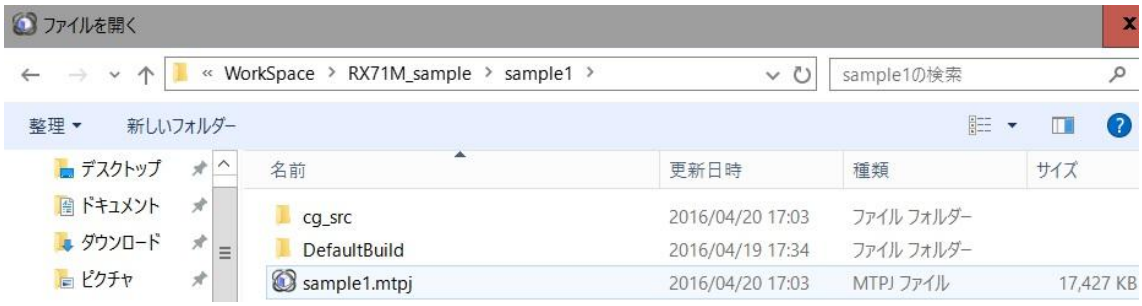
## １－２　　動作、デバック

### ａ：ＣＳ＋　ｆｏｒ　ＣＣ　起動、コンパイル、書き込み、動作

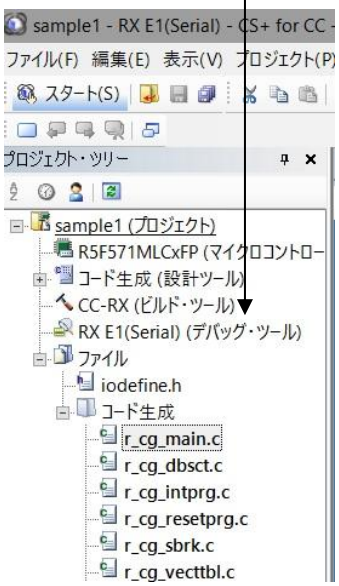



ＣＤに添付しているサンプルプログラムを使って、コンパイル、書き込み、動作の方法を示します。

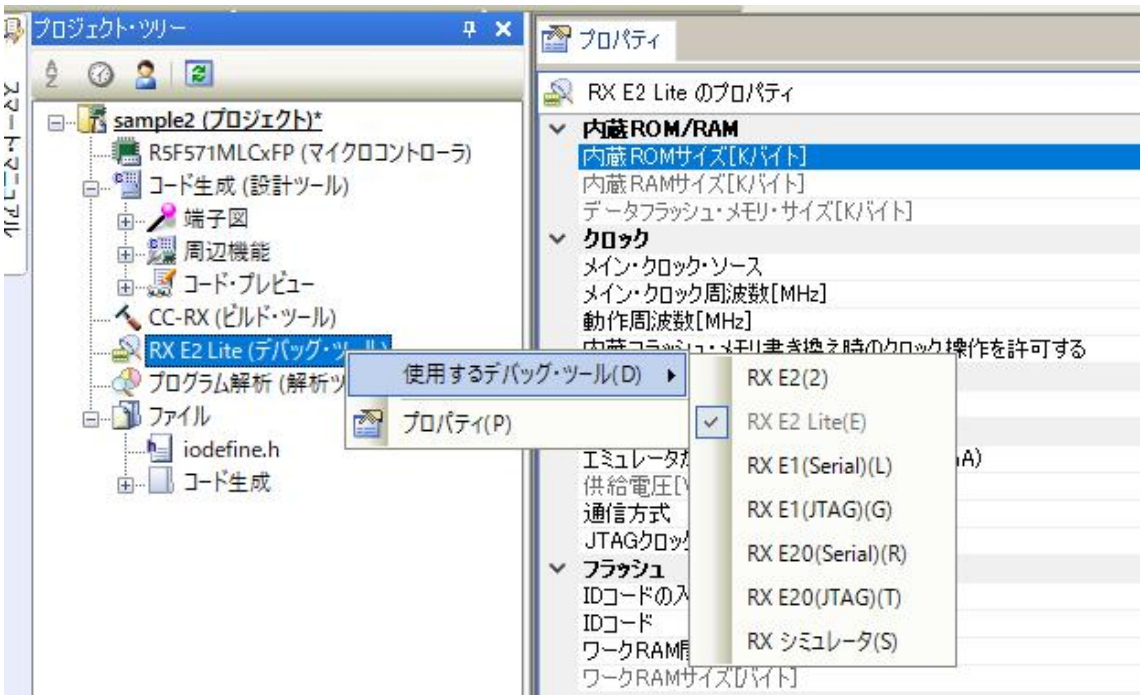
ＣＳ＋　ｆｏｒ　ＣＣを起動します。ここでは例としてＲＸ７＿１Ａｓａｍｐｌｅ￥ｓａｍｐｌｅ１を動作させます。基板上のＬＥＤ　Ｄ１が点滅するプログラムです。
ファイル　→　ファイルを開く　→　ｓａｍｐｌｅ１．ｍｔｐｊをダブルクリックします。



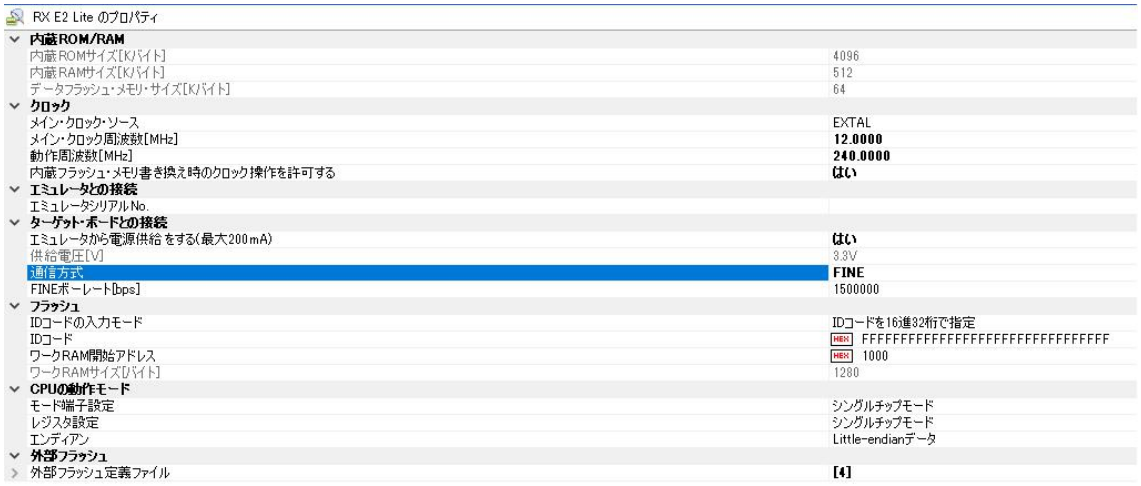

プロジェクトツリーが表示されます。
Ｅ１は設定済みです。



もし、Ｅ２ｌｉｔｅを使う場合は、右クリックで→使用するデバックツール→Ｅ２　ｌｉｔｅを選択します。

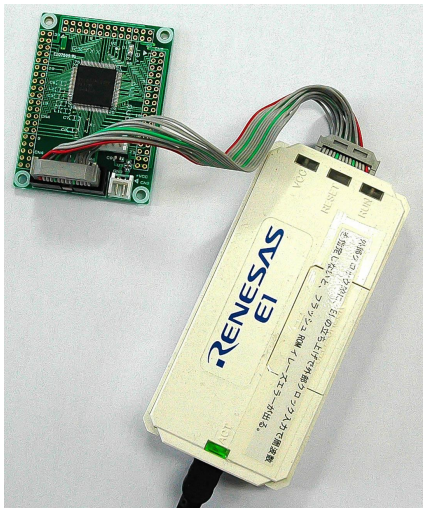



多くの設定はＥ１のものが継承されますが、




通信方式はＪＴＡＧになってしまうので、ＦＩＮＥに切り替えて下さい。

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Ｅ１から供給しますので、不要です。（写真ご参考）



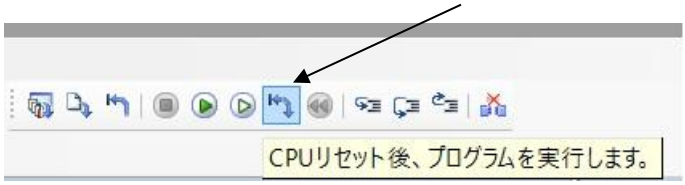

「デバック・ツールへプログラムを転送」をクリック。


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。Ｅ１から電源がＣＰＵ基板に供給されます。　　（Ｅ１　ＶＣＣ　オレンジＬＥＤ点灯）

```
64                      void R_MAIN_UserInit(void);
65                    /***********************************************
66                     * Function Name: main
67                     * Description  : This function implements m
68                     * Arguments    : None
69                     * Return Value : None
70                     ***********************************************
71      ffc0057e       void main(void)
72                    {
73      ffc00580           R_MAIN_UserInit();
74                         /* Start user code. Do not edit comment
75                         while (1U)
76                         {
77
78
79      ffc005ac                   PORT0.PODR.BYTE = 0x55;
80      ffc005ae                   PORT1.PODR.BYTE = 0x55;
81      ffc005b0                   PORT2.PODR.BYTE = 0x55;
82      ffc005b2                   PORT3.PODR.BYTE = 0x55;
83      ffc005b4                   PORT4.PODR.BYTE = 0x55;
84      ffc005b6                   PORT5.PODR.BYTE = 0x55;
```

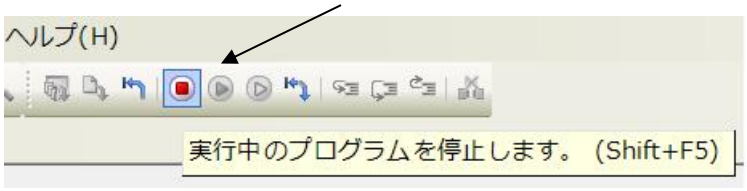
ここまでいかなかった場合、Ｅ１のデバイスドライバインストゥールをご検証願います。

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、基板のＤ１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。

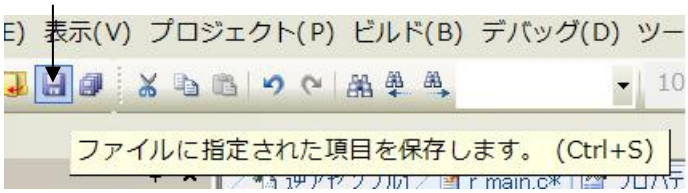

ここまで確認できましたら、一度止めます。



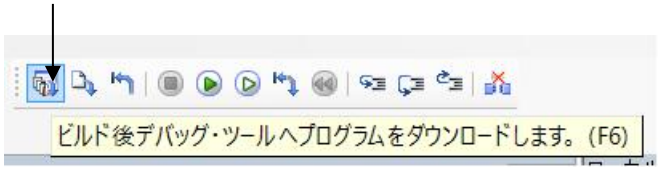

ｍａｉｎ関数のｌｗａｉｔの数値２箇所を１桁０を増やしてみます。

```
ffc005f7 |  |              PORTF.PODR.BYTE = 0x55;
ffc005fe |  |              PORTJ.PODR.BYTE = 0x55;


ffc00605 |  |              main_wait(5000000);


ffc0060d |  |              PORT0.PODR.BYTE = 0xaa;
ffc0060f |  |              PORT1.PODR.BYTE = 0xaa;
ffc00611 |  |              PORT2.PODR.BYTE = 0xaa;
ffc00613 |  |              PORT3.PODR.BYTE = 0xaa;
ffc00615 |  |              PORT4.PODR.BYTE = 0xaa;
ffc00617 |  |              PORT5.PODR.BYTE = 0xaa;

ffc00619 |  |              PORT6.PODR.BYTE = 0xaa;
ffc00620 |  |              PORT7.PODR.BYTE = 0xaa;
ffc00627 |  |              PORT8.PODR.BYTE = 0xaa;
ffc0062e |  |              PORT9.PODR.BYTE = 0xaa;
ffc00635 |  |              PORTA.PODR.BYTE = 0xaa;
ffc0063c |  |              PORTB.PODR.BYTE = 0xaa;
ffc00643 |  |              PORTC.PODR.BYTE = 0xaa;
ffc0064a |  |              PORTD.PODR.BYTE = 0xaa;
ffc00651 |  |              PORTE.PODR.BYTE = 0xaa;
ffc00658 |  |              PORTF.PODR.BYTE = 0xaa;
ffc0065f |  |              PORTJ.PODR.BYTE = 0xaa;

ffc00666 |  |              main_wait(5000000);
```

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。

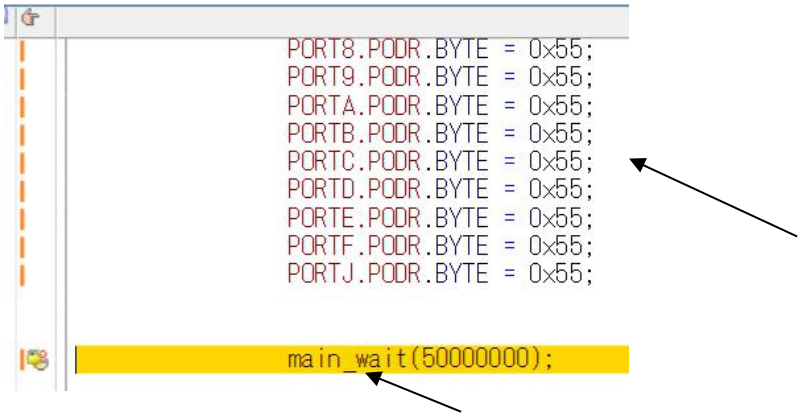



「ＣＰＵリセット後、プログラムを実行」をクリック。
ＬＥＤの点滅が先ほどより、遅くなったのが目視できましたでしょうか？

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点設定しました。手のマークの部分をマウスでダブルクリック。

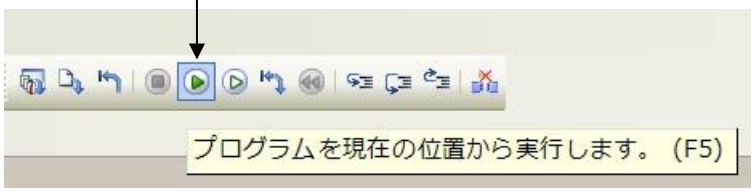
```
 99    ffc00606                main_wait(50000000);
100
101
102    ffc0060e                PORT0.PODR.BYTE = 0xaa;
103    ffc00610                PORT1.PODR.BYTE = 0xaa;
104    ffc00612                PORT2.PODR.BYTE = 0xaa;
105    ffc00614                PORT3.PODR.BYTE = 0xaa;
106    ffc00616                PORT4.PODR.BYTE = 0xaa;
107    ffc00618                PORT5.PODR.BYTE = 0xaa;
108
109    ffc0061a                PORT6.PODR.BYTE = 0xaa;
110    ffc00621                PORT7.PODR.BYTE = 0xaa;
111    ffc00628                PORT8.PODR.BYTE = 0xaa;
112    ffc0062f                PORT9.PODR.BYTE = 0xaa;
113    ffc00636                PORTA.PODR.BYTE = 0xaa;
114    ffc0063d                PORTB.PODR.BYTE = 0xaa;
115    ffc00644                PORTC.PODR.BYTE = 0xaa;
116    ffc0064b                PORTD.PODR.BYTE = 0xaa;
117    ffc00652                PORTE.PODR.BYTE = 0xaa;
118    ffc00659                PORTF.PODR.BYTE = 0xaa;
119    ffc00660                PORTJ.PODR.BYTE = 0xaa;
120
121    ffc00668                main_wait(50000000);
```

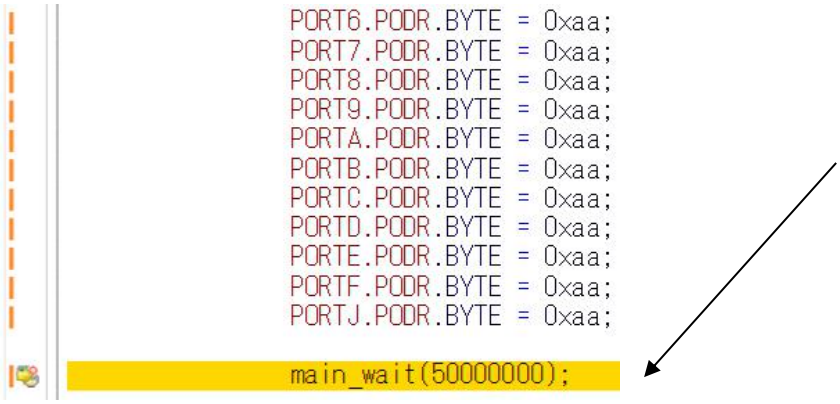
「ＣＰＵリセット後、プログラムを実行」します。




プログラムの実行はブレークポイントで停止し、ＬＥＤ　Ｄ１　は　ＰＯＲＴＤ．ＰＯＤＲ．ＢＹＴＥ＝０ｘ５５；命令によりＰＤ０＝１となるので、点灯します。

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、ＰＯＲＴＤ．ＰＯＤＲ．ＢＹＴＥ＝０ｘａａ；命令実行によりＰＤ０＝０になり、ＬＥＤは消灯します。




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。

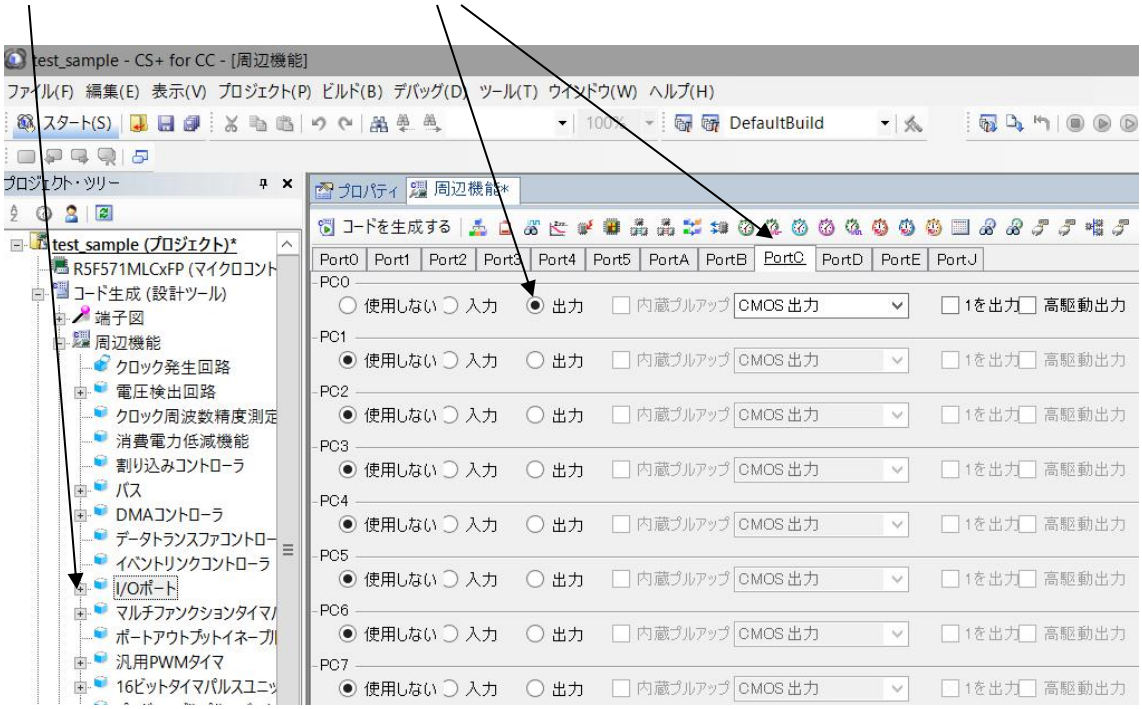
## ｂ：新しいプログラムを作る

# 省略

## 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。



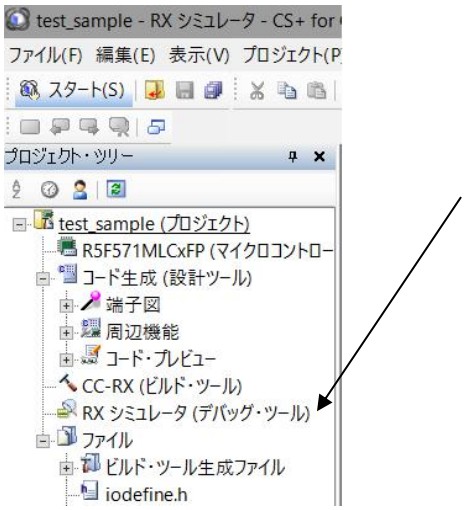

これでコードを生成するをクリックするとＰＯＲＴＣのＰＣ０を出力に設定する初期化プログラムが自動生成されます。

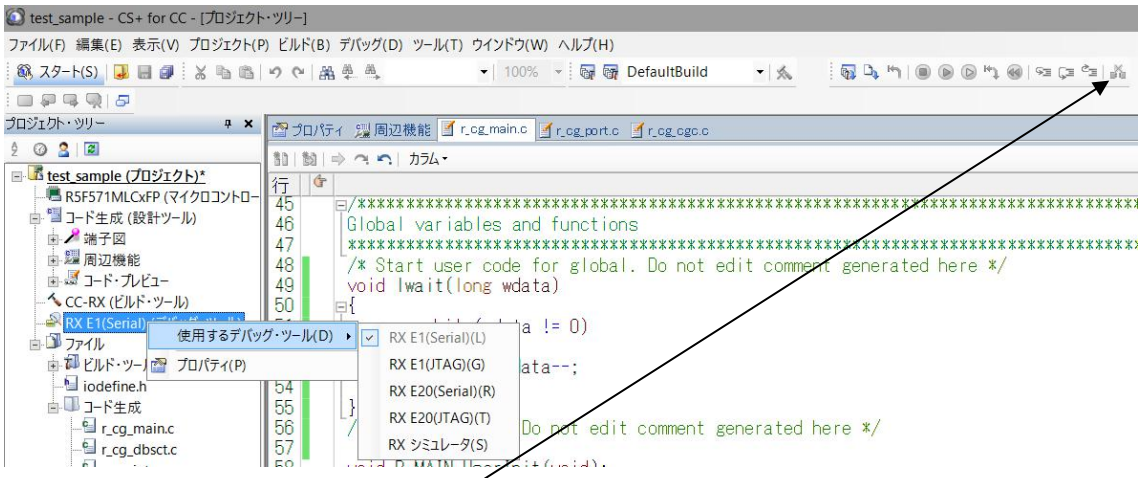
## ｂ－２：プログラムの書き込み、書く場所の注意

# 省略

## ｂ－３：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

デフォルトのデバック・ツールがシュミレータとなっていて、Ｅ１を使う設定になっていません。




Ｅ１からの電源供給が遮断されている状態で「使用するデバック・ツール」→　ＲＸ　Ｅ１を選択して下さい。

### ｂ－４：Ｅ１から電源供給

# 省略

綺麗にＬＥＤが点滅できたと思います。しかし、ｓａｍｐｌｅ１と比べると遅い。実はまだ設定が十分ではありません。

### ｂ－５：クロック発生回路を設定する必要があります

# 省略

これで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

ルネサス等から提供されているライブラリを自分で新規に製作したプロジェクトで走らそうとするときに、ｖｅｃｔ．ｈの内容が重複定義となり、エラーが出る場合があります。ｖｅｃｔ．ｈの中の定義を削除して下さい。
以下、ＲＸ２＿１Ａのｓａｍｐｌｅ３．ｃの中で定義されている割込み。

```
/***********************************************************************
* Function Name: dsadi0_isr
* Description   : DSAD channel0 interrupt rutine
* Arguments      : none
* Return Value : none
***********************************************************************/
#pragma interrupt dsadi0_isr(vect = VECT(DSAD, DSADI0))
static void dsadi0_isr(void)
{
    /* Read conversion data of DSAD channel0
    DSADDR0        Delta-Sigma Data Register 0
    b31-b0         Holding A/D results. Read only register. */
    g_dsad_data[0] = (int32_t)DSAD.DSADDR0;


    /* Clear Interrupt Request Register assigned DSADI0. */
    IR(DSAD, DSADI0) = 0;
}
```

ｖｅｃｔ．ｈの中をコメントにしないと２重定義でエラーがでます。

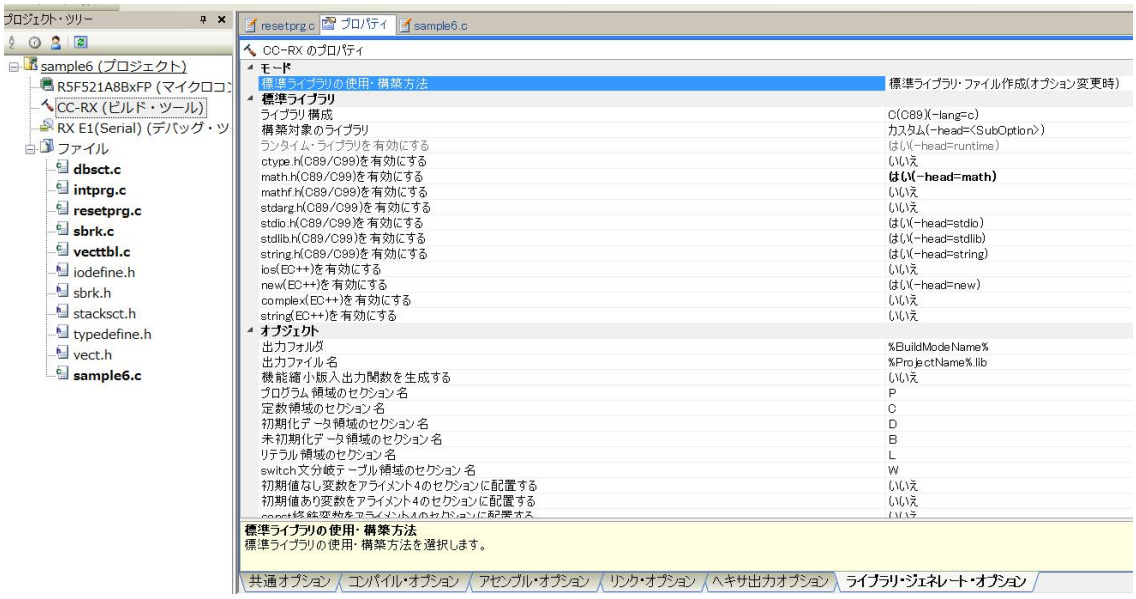
```
/*
// DSAD DSADI0
#pragma interrupt (Excep_DSAD_DSADI0(vect=207))
void Excep_DSAD_DSADI0(void);

// DSAD DSADI1
#pragma interrupt (Excep_DSAD_DSADI1(vect=208))
void Excep_DSAD_DSADI1(void);

// DSAD DSADI2
#pragma interrupt (Excep_DSAD_DSADI2(vect=209))
```

```
void Excep_DSAD_DSADI2(void);

// DSAD DSADI3
#pragma interrupt (Excep_DSAD_DSADI3(vect=210))
void Excep_DSAD_DSADI3(void);

// DSAD DSADI4
#pragma interrupt (Excep_DSAD_DSADI4(vect=211))
void Excep_DSAD_DSADI4(void);

// DSAD DSADI5
#pragma interrupt (Excep_DSAD_DSADI5(vect=212))
void Excep_DSAD_DSADI5(void);

// DSAD DSADI6
#pragma interrupt (Excep_DSAD_DSADI6(vect=213))
void Excep_DSAD_DSADI6(void);
*/
```

## ｃ－３：三角関数ｍａｔｈ．ｈはインクルードもＣＳ＋の設定も必要

ｓａｍｐｌｅ６は三角、対数、平方根関数を使用しますが、ソースファイルにインクルードを記入するだけでなく、

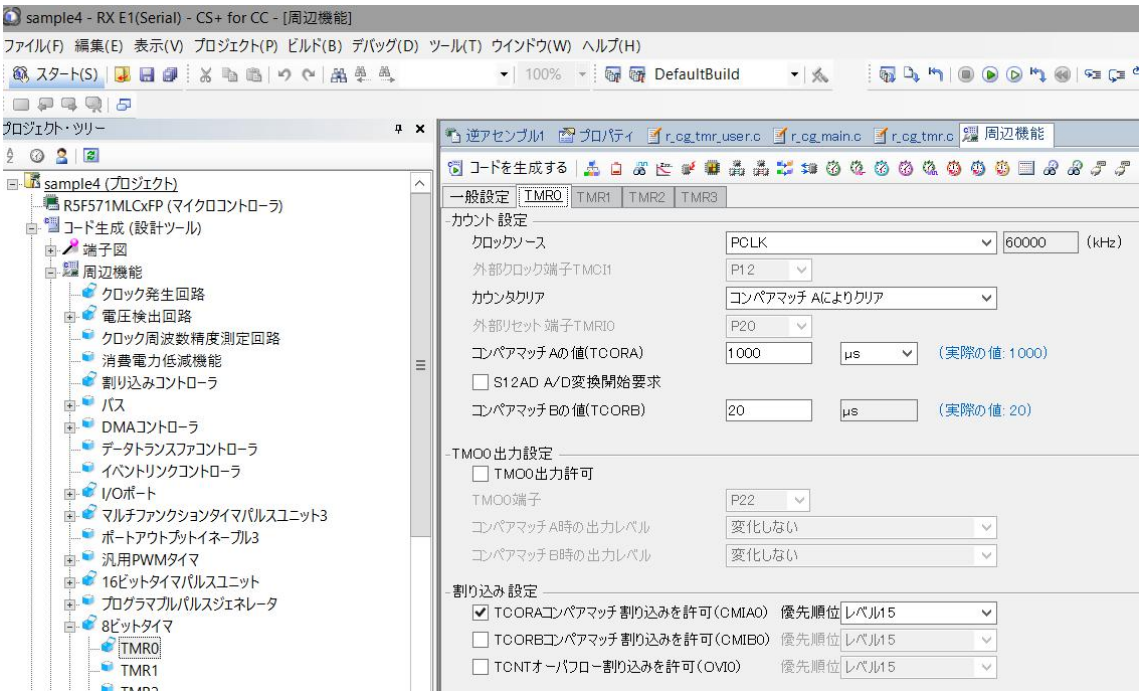
#include <math.h>　　//sqr 等演算を行うのに必要　math.h 有効と共にこの表記も必要。



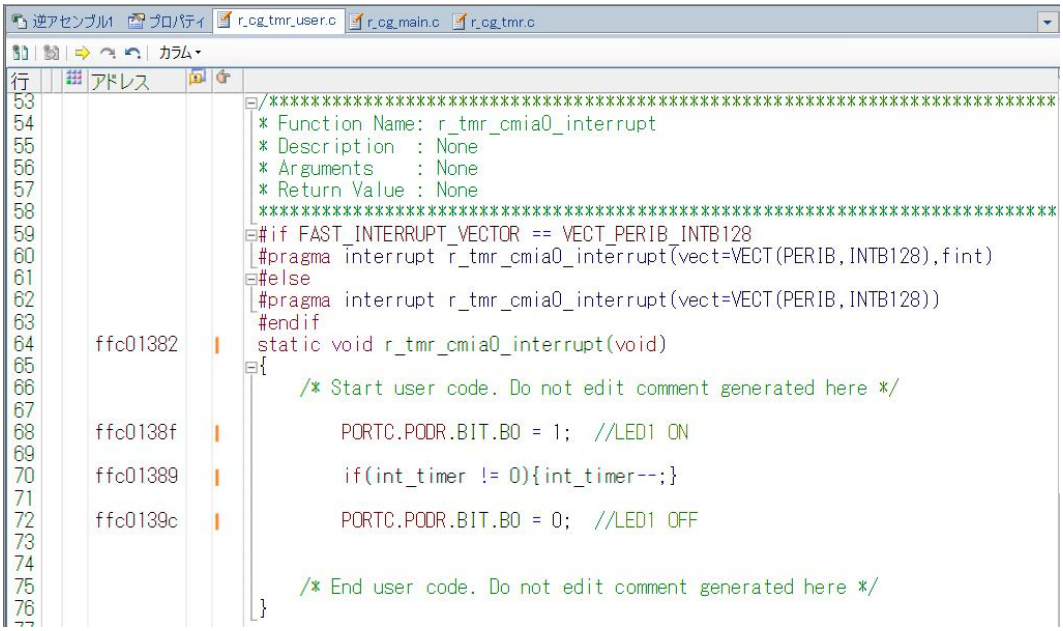

ＣＣ-ＲＸ（ビルド・ツール）→　ライブラリ・ジェネレート・オプション　→　ｍａｔｈ．ｈを有効にする→はい　としてください。

## 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

ｓａｍｐｌｅ４で１ｍｓｅｃ毎の定周期タイマー割り込みを使用しています。これがちゃんと割り込まれているか、周期、割り込み中の消費時間が分かる方法があります。８ビットタイマ→ＴＭＲ０→１００００μsec＝１ｍｓｅｃ毎にＣＭＩＡ０割り込みが入るように設定されています。




「コード生成」でｒ＿ｃｇ＿ｔｍｒ．ｃとｒ＿ｃｇ＿ｔｍｒ＿ｕｓｅｒ．ｃファイルが新たに生成されますが、ｒ＿ｃｇ＿ｔｍｅｒ．ｃの方は初期化設定＝メインの初めに１回だけコールします、ｒ＿ｃｇ＿ｔｍｒ＿ｕｓｅｒ．ｃのほうにユーザーがプログラムを書き込むべきＣＭＩＡ０割り込み関数が生成されます。

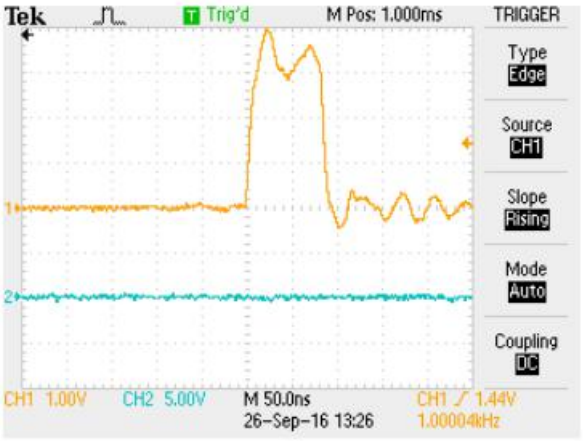



ｒ＿ｔｍｒ＿ｃｍｉａ０＿ｉｎｔｅｒｒｕｐｔ割り込みの中でポートを立てて、抜け出す前に下ろしてい

ます。

この波形をオシロスコープで観測します。


細い筋が割込み周期です。ほぼ１ｍｓｅｃ毎になっています。また、１つの波形を時間軸を拡大すると

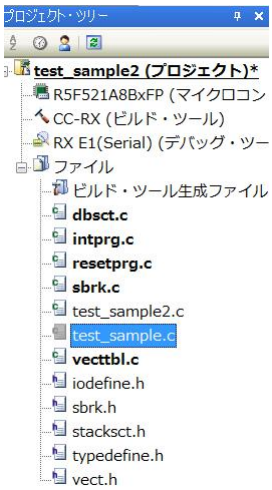



割り込みプログラムで５０ｎｓｅｃ×１．８　≒　９０ｎｓｅｃ程度の時間を消費していることが分かります。このポートがＨでない部分がメインルーチンのプログラムが走っている時間です。

## 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

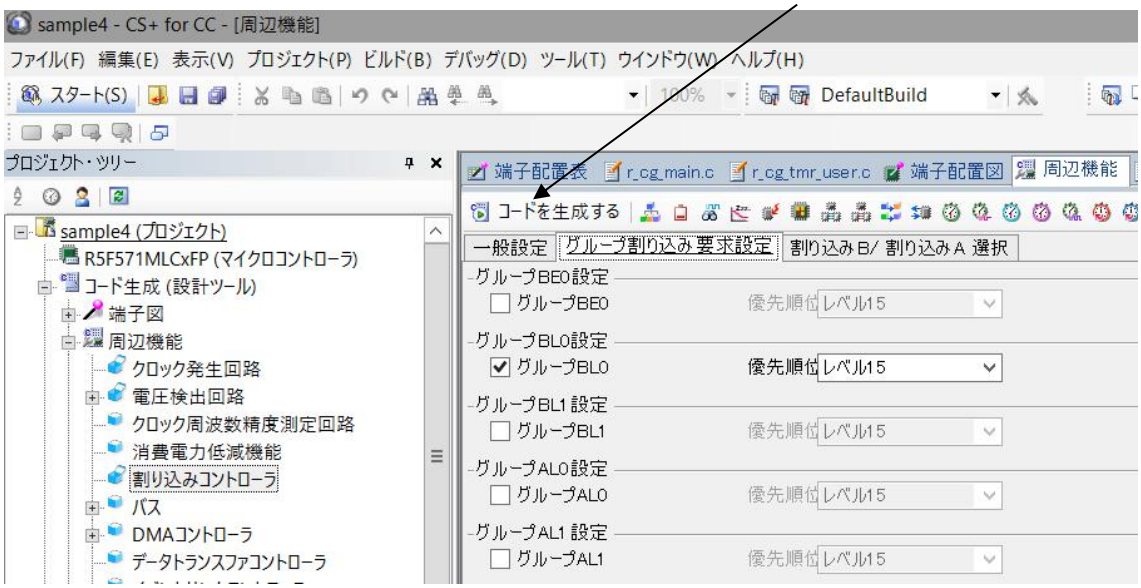
# 省略

⑤ｔｅｓｔ＿ｓａｍｐｌｅ．ｃを削除。これで完成です。気になる方は、ＤｅｆａｕｌｔＢｕｉｌｄホルダの中のｔｅｓｔ＿ｓａｍｐｌｅファイル群も削除して下さい。

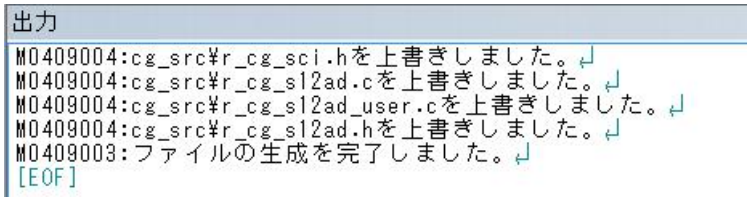


## ｃ－６：コード生成と見落しがちな注意点

例えば、下記のように割り込み要求を変更し→「コードを生成する」で




全てのプログラムソースが上書きされるのですが、




# 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

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。ＣＮ１にＥ１のケーブルを差し、「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。


電源はＥ１から供給しますので、新たに準備する必要はありません。

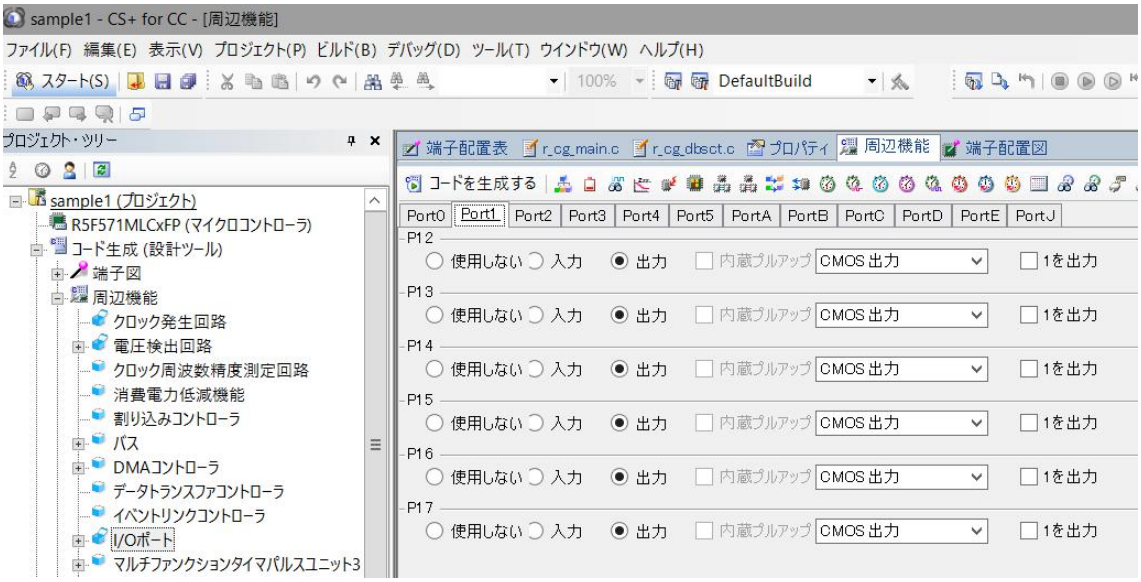
### ｓａｍｐｌｅ１　ポートのＯＮ/ＯＦＦ

**【　概要　】**

出力可能な全ポートのＯＮ/ＯＦＦを繰り返します。ＰＯＲＴＣのＰＣ０に接続された基板上のＬＥＤ　Ｄ１が点滅します。

**【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０からポートＪまで出力可能なポートは全て出力に設定し、「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          PORT0.PODR.BYTE = 0x55;
                   PORT1.PODR.BYTE = 0x55;
                   PORT2.PODR.BYTE = 0x55;
                   PORT3.PODR.BYTE = 0x55;
                   PORT4.PODR.BYTE = 0x55;
                   PORT5.PODR.BYTE = 0x55;

                   PORT6.PODR.BYTE = 0x55;
                   PORT7.PODR.BYTE = 0x55;
                   PORT8.PODR.BYTE = 0x55;
                   PORT9.PODR.BYTE = 0x55;
                   PORTA.PODR.BYTE = 0x55;
                   PORTB.PODR.BYTE = 0x55;
                   PORTC.PODR.BYTE = 0x55;
```

```
                PORTD.PODR.BYTE = 0x55;
                PORTE.PODR.BYTE = 0x55;
                PORTF.PODR.BYTE = 0x55;
                PORTJ.PODR.BYTE = 0x55;

⑤               main_wait(5000000);

⑥               PORT0.PODR.BYTE = 0xaa;
                PORT1.PODR.BYTE = 0xaa;
                PORT2.PODR.BYTE = 0xaa;
                PORT3.PODR.BYTE = 0xaa;
                PORT4.PODR.BYTE = 0xaa;
                PORT5.PODR.BYTE = 0xaa;

                PORT6.PODR.BYTE = 0xaa;
                PORT7.PODR.BYTE = 0xaa;
                PORT8.PODR.BYTE = 0xaa;
                PORT9.PODR.BYTE = 0xaa;
                PORTA.PODR.BYTE = 0xaa;
                PORTB.PODR.BYTE = 0xaa;
                PORTC.PODR.BYTE = 0xaa;
                PORTD.PODR.BYTE = 0xaa;
                PORTE.PODR.BYTE = 0xaa;
                PORTF.PODR.BYTE = 0xaa;
                PORTJ.PODR.BYTE = 0xaa;

⑦               main_wait(5000000);

    }

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

**【　解説　】**

# 省略

```
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
```
PORT0.PODR.BYTE = 0x55;
PORT1.PODR.BYTE = 0x55;
PORT2.PODR.BYTE = 0x55;
PORT3.PODR.BYTE = 0x55;
PORT4.PODR.BYTE = 0x55;
 ;
PORTC.PODR.BYTE = 0x55;
```

０ｘ５５＝０１０１０１０１Ｂです。１ビット毎に１を立てています。ＰＯＲＴＣのＰＣ０に繋がれているＬＥＤ　Ｄ１もデータ’１’で電流が流れて点灯します。

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

⑤
```
main_wait(5000000);
```

５０００００００という数を減算して０になるまでの間、ポートが０ｘ５５に保たれます。ＬＥＤ　Ｄ１も点灯が保たれます。

⑥
```
PORT0.PODR.BYTE = 0xaa;
PORT1.PODR.BYTE = 0xaa;
PORT2.PODR.BYTE = 0xaa;
PORT3.PODR.BYTE = 0xaa;
PORT4.PODR.BYTE = 0xaa;
PORT5.PODR.BYTE = 0xaa;
```

０ｘ５５のビット反転数０ｘａａを出力しています。
０ｘａａ＝１０１０１０１０Ｂです。１ビット毎に１を立てています。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。

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

０ｘ５５と０ｘａａを交互に出す理由は、仮に隣のポートと接触しているとレベルの変化がありませんので、例えばＬＥＤが点滅しません。それによりハードウエアの異常が検出できます。（隣は必ず異なる論理

なので０，１でも１，０でも接触していると０になります。弊社出荷検査にて使用しています）

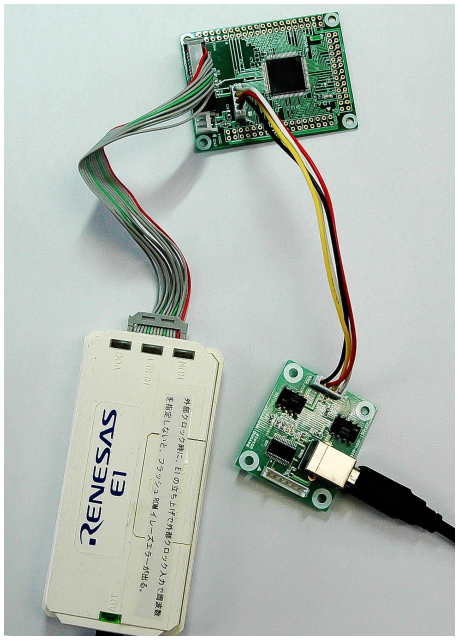
⑦　　main_wait(5000000);
消灯している間も点灯同様に時間を保持しています。

## ２－２　ｓａｍｐｌｅ２　ＳＩＯ（ＵＳＢ）でパソコンとのやりとり

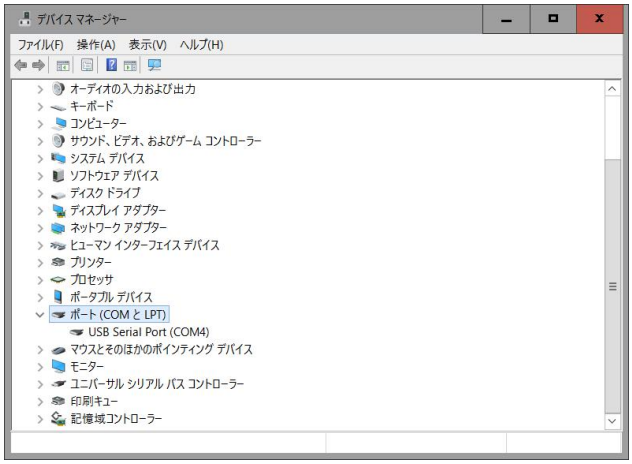
**【　概要　】**

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９６００ｂｐｓに設定して下さい。

ＵＳＢ－ＳＩＯ絶縁変換器基板とＣＰＵボードを接続します。ＵＳＢケーブルでパソコンとつなげると、ＵＳＢ基板側に電源が入ります。



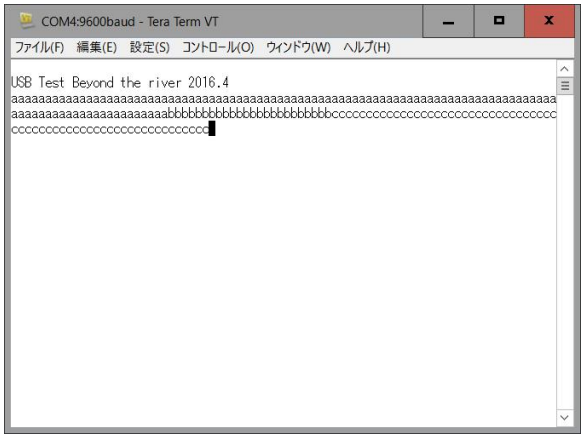

スタート→右クリック→デバイスマネージャー　→　ポート（ＣＯＭとＬＰＴ）でＵＳＢ　Ｓｅｒｉａｌ　Ｐｏｒｔ（ＣＯＭｘｘ）があることを確認して下さい（Ｗｉｎｄｏｗｓ１０の場合）。例ではＣＯＭ４となっています。




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９６００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。

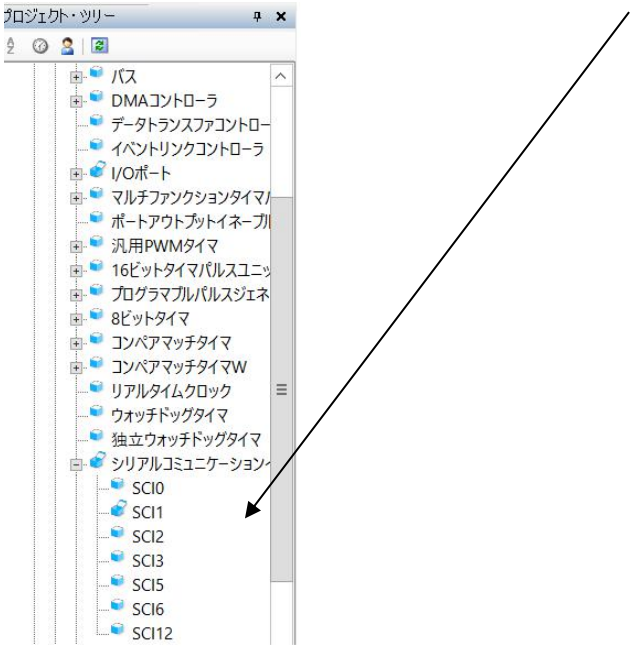



ＵＳＢ　Ｔｅｓｔ、、、　と表示され、ＰＣのキーボードを何か押すたびに、押した文字が表示されると動作としてはＯＫです。

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。

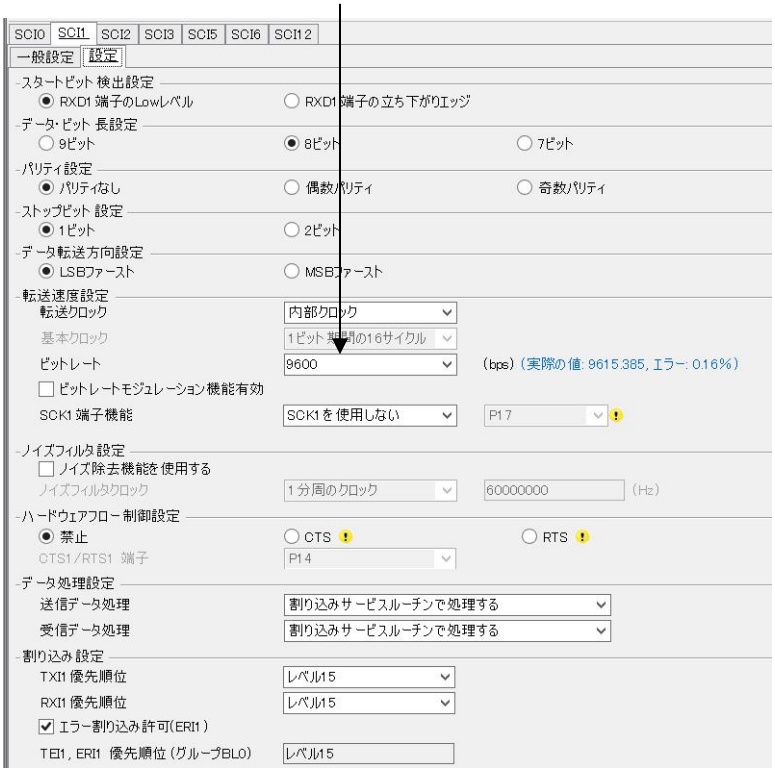
**【周辺機能の説明】**

ここではＳＣＩ１を使用しています。青箱のふたが空いたように見える機能が使用されている機能です。

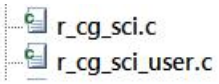



ＳＣＩ１をクリック　→　一般設定　調歩同期式、ＴＸＤ１端子としてＰ１６，ＲＸＤ１端子としてＰ１５が選択されています。

ボーレートは９６００ｂｐｓです。８ビットデータ、パリティなし、１ビットストップビット、ハードウエアフロー制御なし。




この条件で「コード生成」が行われ、以下の２つの関数が生成されています。




ｒ＿ｃｇ＿ｓｃｉ．ｃは電源投入時に自動的に実行される void R_SCI1_Create(void)関数と、ユーザーが使い初めに１回だけコールする void R_SCI1_Start(void)関数が自動生成されています。

# 省略

割込みコントローラも設定する必要があります。

# 省略

**【　プログラム　】**

```
void main(void)
{
    R_MAIN_UserInit();
```

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


①R_SCI1_Start();

②R_SCI1_Serial_Receive(rx_data,1);
 rx_flg = 0;
 tx_end_flg = 0;


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

 ③R_SCI1_Serial_Send(String_0,37);               //Opening message
 ④tx_end_wait();                                 //送信終了待ち


⑤  while (1U)
 {
⑥       if(rx_flg == 1)
        {
                PORTC.PODR.BIT.B0 = 1;           //LED1 ON

                rx_flg = 0;                      //受信フラグクリア

⑦               R_SCI1_Serial_Receive(rx_data,1);  //1文字受信
                R_SCI1_Serial_Send(rx_data,1);     //1文字送信
        }
 }
```

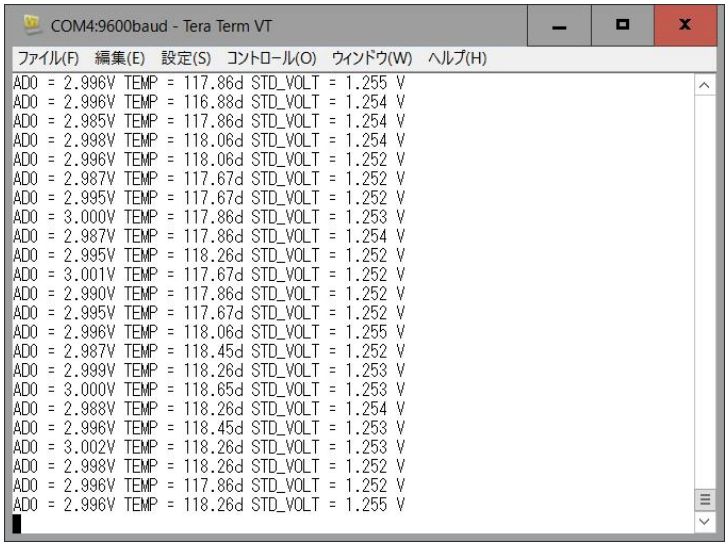
**【　解説　】**

# 省略

## ２－３　ｓａｍｐｌｅ３　Ａ／Ｄ変換をＵＳＢ出力

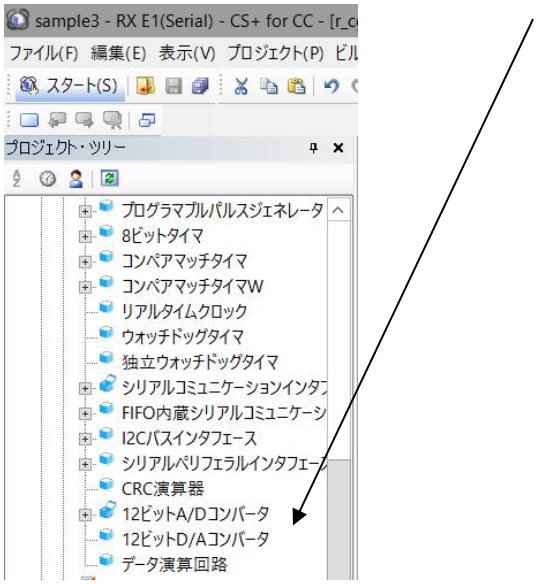
**【　動作概要　】**

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。




**【周辺機能の説明】**

ＳＣＩ１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。

# 省略

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。

# 省略

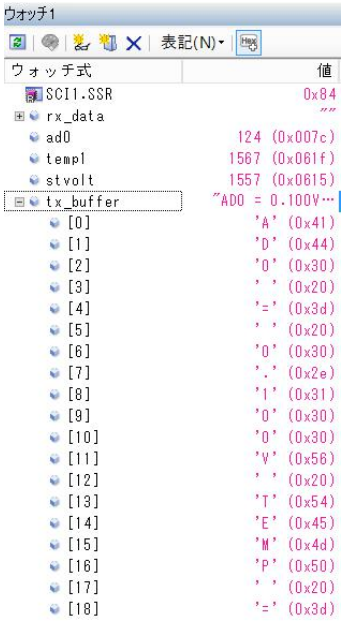
**【　プログラム　】**

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

    R_SCI1_Start();                 //SCI1動作開始

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

    R_SCI1_Serial_Receive(rx_data,1);
    rx_flg = 0;
    tx_end_flg = 0;


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

    R_SCI1_Serial_Send(String_0,37);    //Opening message
    tx_end_wait();                      //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


⑧               fdata3 = stvolt/(4095/3.3);     //データ→電圧換算  type1.25V 1.20～1.30V

⑨       sprintf(tx_buffer,"AD0 = %.3fV TEMP = %.2fd STD_VOLT = %.3f V¥n¥r",fdata1,fdata2,fdata3);
⑩               R_SCI1_Serial_Send(tx_buffer,sizeof(tx_buffer));    //データをＵＳＢ出力
                tx_end_wait();                  //送信終了まち


⑪                  PORTC.PODR.BIT.B0 = 1;      //LED1 ON
                   main_wait(10000000);
                   PORTC.PODR.BIT.B0 = 0;      //LED1 OFF
                   main_wait(10000000);

        }
```

**【　解説　】**

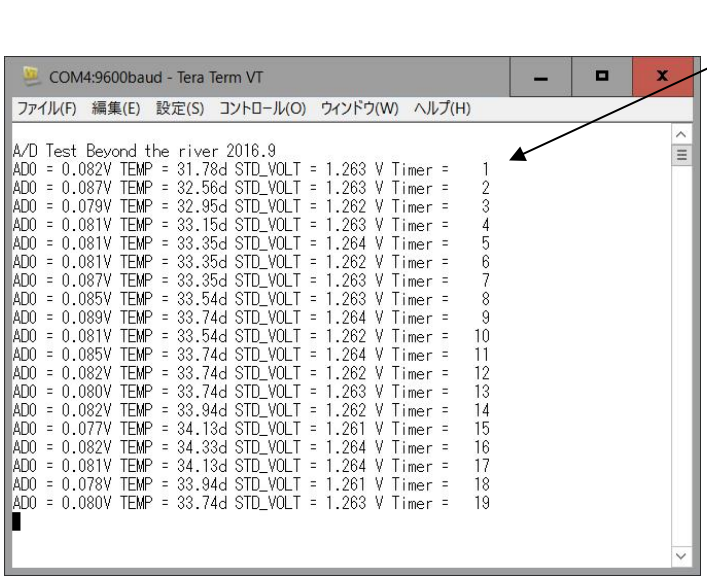
# 省略


ウォッチ1
表記(N)
ウォッチ式
値
SCI1.SSR
0x84
rx_data
""
ad0
124 (0x007c)
temp1
1567 (0x061f)
stvolt
1557 (0x0615)
tx_buffer
"AD0 = 0.100V…
[0]
'A' (0x41)
[1]
'D' (0x44)
[2]
'0' (0x30)
[3]
' ' (0x20)
[4]
'=' (0x3d)
[5]
' ' (0x20)
[6]
'0' (0x30)
[7]
'.' (0x2e)
[8]
'1' (0x31)
[9]
'0' (0x30)
[10]
'0' (0x30)
[11]
'V' (0x56)
[12]
' ' (0x20)
[13]
'T' (0x54)
[14]
'E' (0x45)
[15]
'M' (0x4d)
[16]
'P' (0x50)
[17]
' ' (0x20)
[18]
'=' (0x3d)

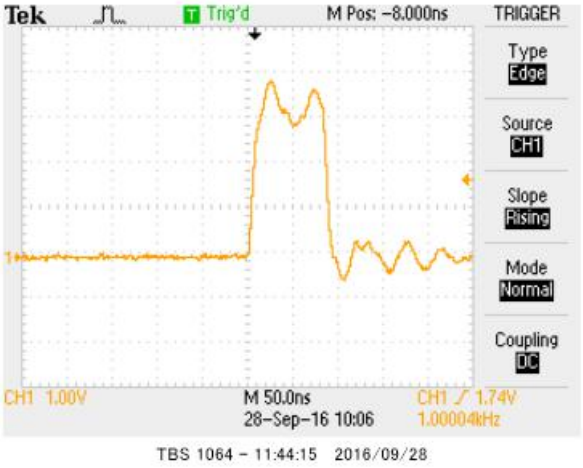
## ２－４　ｓａｍｐｌｅ４　割り込み

**【　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。

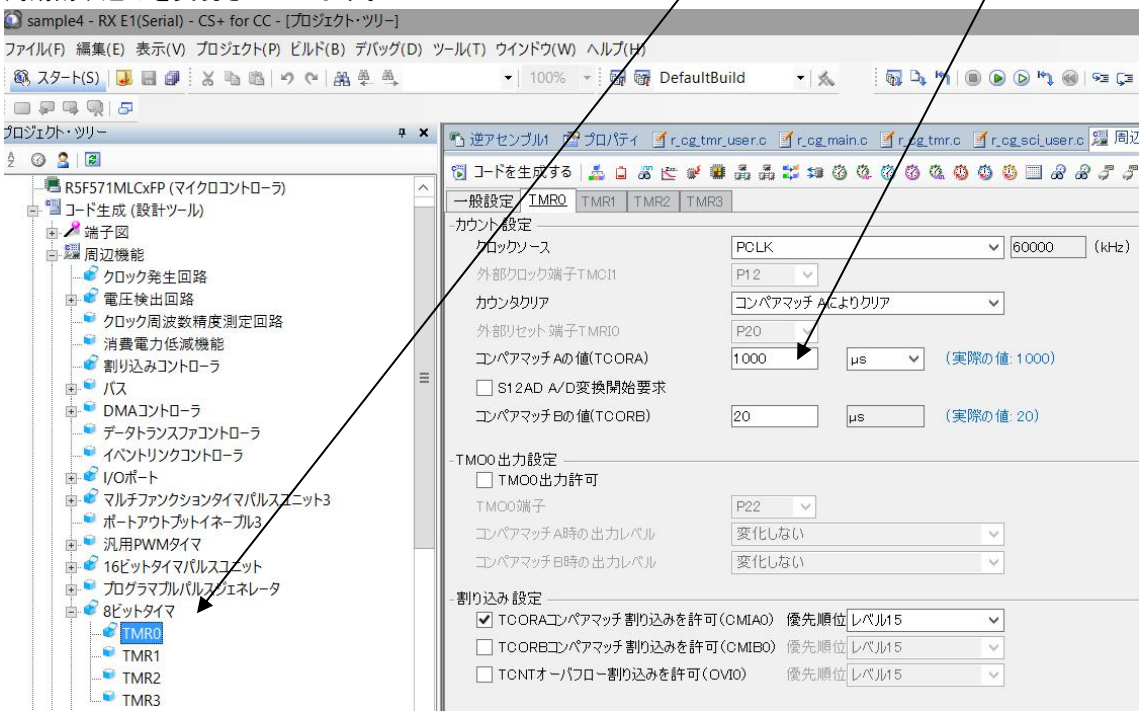



オシロスコープがあればＰＣ０　ＣＮ６　２０番を観測すると、以下のような１ｍｓｅｃ毎の波形が観測できます。



**【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。




「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　】**

ｓａｍｐｌｅ３と比べて変わった所だけ書きます。

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

    R_SCI1_Start();  //SIO初期化

    R_S12AD0_Start();          //AD初期化
    R_S12AD1_Start();          //AD初期化

①     R_TMR0_Start();          //8 ビットタイマー初期化

;
```

```
    while (1U)
    {
②           if(int_timer == 0)
            {
③               timer++;
④               int_timer = 1000;                   //1msec*1000=1 秒
```

ｒ＿＿ｃｇ＿＿ｔｍｅｒ＿＿ｕｓｅｒ．ｃ　の割り込みの部分

```
static void r_tmr_cmia0_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */

⑤           PORTC.PODR.BIT.B0 = 1;          //LED1 ON

⑥           if(int_timer != 0){int_timer--;}

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


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

**【　解説　】**

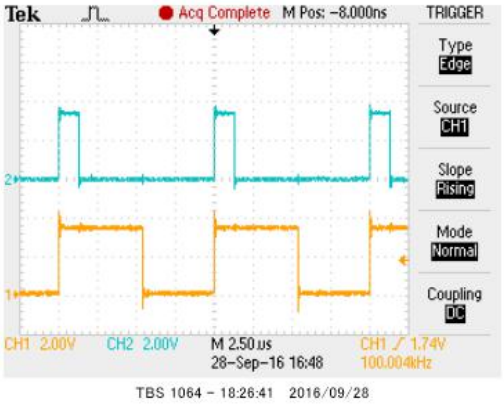
# 省略

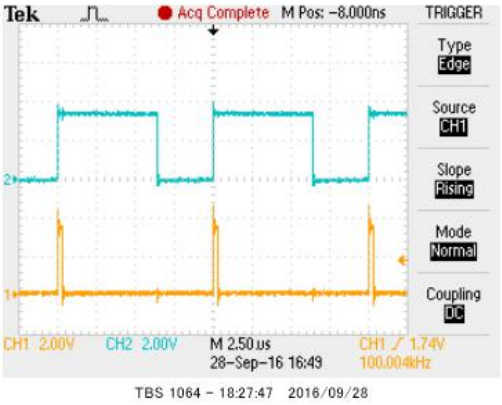
このように、割り込みで時間を作り、メインで使用すれば、極めて正確なタイマーが多数作成可能です。定周期割り込みはその中に様々なプログラムを作成することも可能で、機能的なプログラム作成に不可欠な知識、要素です。

## 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、ＧＴＩＯＣ０Ｂ端子　Ｐ１７　ＣＮ５　１６番に出力されます。



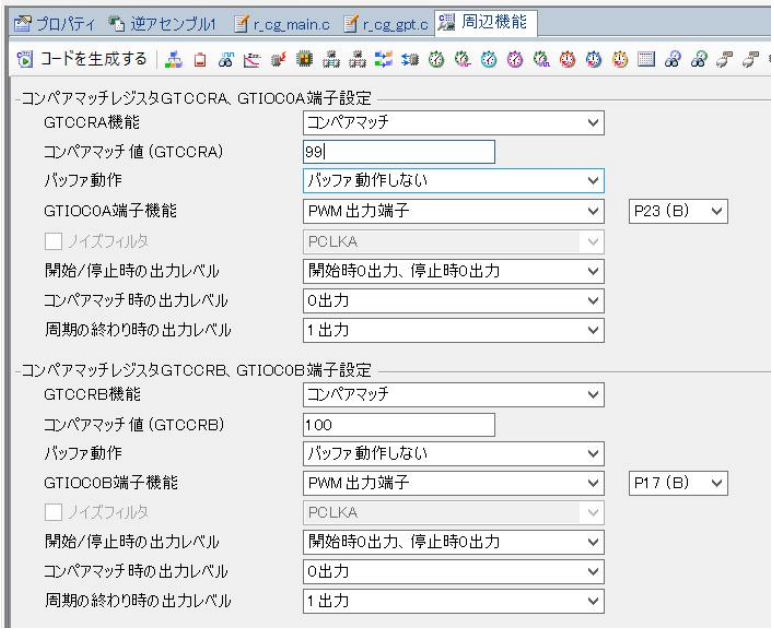

ＰＷＭ波形は上図のように、周期が変わらず、設定値によってＨ、Ｌの幅の比率が変化します。この出力でＬＥＤやモーターをドライブすると明るさや速度を変えることが出来るので、現代では様々な用途に使われています。

**【周辺機能の説明】**

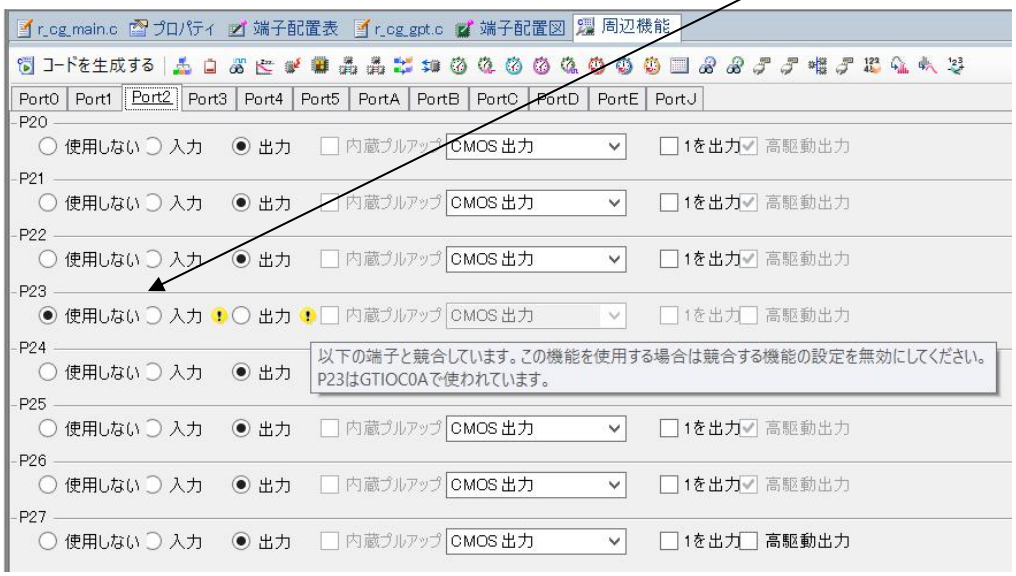
汎用ＰＷＭタイマ→ＧＰＴ０でタイマ周期レジスタを１０μｓｅｃに設定。ｆ（周波数）＝１／周期　で周波数は１００ＫＨｚとなります。周期レジスタ（ＧＴＰＲ０）の値が自動的に５９９になります。一般的に変化の応答を早くしたいときは周期を短くし、分解能を重視したいときは周期を長くします。

# 省略

コンペアマッチレジスタＧＴＣＣＲＡに書く値を０～５９９まで変えることによりＰＷＭの幅が変ります。初期値として９９をセットしています。ＰＷＭ出力端子をＰ２３，Ｐ１７に設定しています。




なおＰ１７，Ｐ２３は周辺機器→Ｉ／Ｏポート→（Ｉ／Ｏとしては）使用しない　を選択する必要があります。



【　プログラム　】

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


    R_SCI1_Start();              //SCI1スタート
①   R_GPT0_Start();              //汎用PWMタイマースタート

    R_SCI1_Serial_Receive(rx_data,1);
    rx_flg = 0;
    tx_end_flg = 0;


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

    R_SCI1_Serial_Send(String_0,37);                    //Opening message
    tx_end_wait();                                      //送信終了待ち


②   while (1U)
    {
        if(GPT0.GTCCRA != 599)
        {
③                   GPT0.GTCCRA++;
        }
        else
        {
④                   GPT0.GTCCRA = 0;
        }

        if(GPT0.GTCCRB != 0)
        {
⑤                   GPT0.GTCCRB--;
        }
        else
        {
⑥                   GPT0.GTCCRB = 599;
        }

⑦       main_wait(1000000);

    }
```

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

**【　解説　】**

# 省略

従来のＲＸのイニシャルプログラム作成時はレジスタライトプロテクション有効、解除の頻繁な書き込みが必要でしたし、ＰＷＭを使うためのレジスタの詳細を掴んでからでないとプログラムが作成できませんでしたが、このようにＲＬ７８と全く同じようにＲＸマイコンの開発が行える環境がそろいました。

## ２－６　三角、対数、平方根関数を使う

**【　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　】**

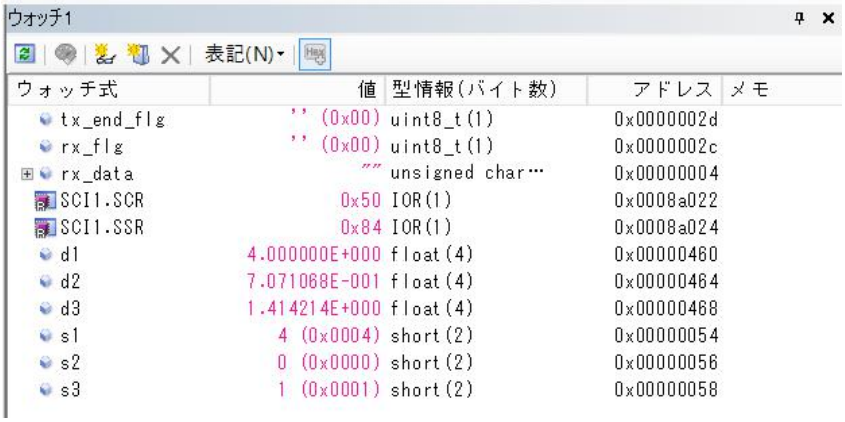
```
;
①#include <math.h>

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　】**

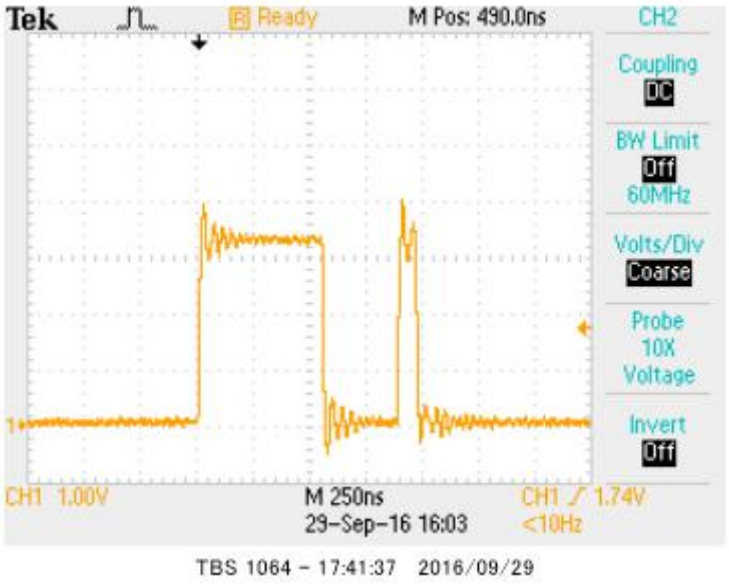
# 省略

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。

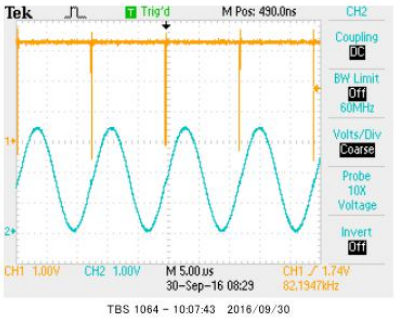
ＣＰＵ別演算速度例

| 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　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。




**【周辺機能の説明】**

Ｄ／Ａコンバータの設定は以下の通りです。

# 省略

**【　プログラム　】**

```
①unsigned short sin_data[370],kakudo;
 ;

void main(void)
{
②//演算して結果をメモリにセーブ

                for(kakudo = 0;kakudo < 360 ;kakudo++)
                {
                        d2 = sin((PI/180)*kakudo);      //1から－1まで変動
                        d2 +=1;                         //オフセット＋１→　0～2の変化になる
                        d2 *= 2047.5;                   //2を最大電圧3.3Vにする。
                        sin_data[kakudo] = d2;          //sin信号をDAOUT P05

                }

③//演算結果をD/Aに出力

    while (1U)
    {
```

```
        PORTC.PODR.BIT.B0 = 1;                                  //LED1 ON

                for(kakudo = 0;kakudo < 360 ;kakudo++)
                {
                        DA.DADR1 = sin_data[kakudo];   //sin信号をDAOUT P03

                }

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

**【　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。

---



**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　