ＢＣＲＬ７８＿１０１　マイコン開発セット　マニュアル　抜粋

第１版２０１６．４．１３　　**第１版**

【　製品概要　】

本マニュアルはＢＣＲＬ７８＿１０１　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　ＣＡ，ＣＸ**における開発方法について多く記述してあります。

※本ＣＰＵボード開発にはルネサスエレクトロニクス社製Ｅ１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：ＢＣＲＬ７８＿１０１　ＣＰＵボードの特徴、ＢＣＲＬ７８＿１０４との違い

ｃ：Ｅ１エミュレータ（デバッカ）

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

ｆ：ＲＬ７８とＨ８／３００Ｈ、Ｒ８Ｃの速度比較

ｆ－１：ポートアクセス速度の比較

ｆ－２：乗除演算速度の比較

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：Ａ／Ｄ設計上の注意点

ｂ－２：自動生成されたプログラム

ｂ－３：Ｅ１から電源供給

ｂ－４：コード生成後の初期値の変更

ｂ－５：変数を見る

ｂ－６：変数変化を実行中に確認する

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）１文字送受信

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

１－１．開発環境

ａ：開発セット同梱物

ＢＣＲＬ７８＿１０１　ＣＰＵボード
ＣＤ（サンプルプログラム、ドキュメント）
マニュアル（本誌）
電源ケーブル、ＵＳＢ－ＳＩＯ絶縁変換器、ＵＳＢケーブル

※開発に必要なルネサスエレクトロニクス社製エミュレータＥ１は同封されておりません。別途必要です。

ｂ：ＢＣＲＬ７８＿１０１　ＣＰＵボードの特徴

●電源電圧： VDD = 1.6 - 5.5 V
最高動作周波数： 32 MHz（オンチップオシレータ　±1%精度）/41DMIPS　66μA/MHz
ROM 容量：96 KB
RAM 容量： 8 KB
データ・フラッシュ：非搭載
100 ピン・プラスチック LFQFP（14×14mm）

●クロック 32 MHz（TYP.）/24 MHz（TYP.）/16 MHz（TYP.）/12 MHz（TYP.）/8 MHz（TYP.）/4 MHz（TYP.）/1 MHz（TYP.）から選択可能
単電源のフラッシュ・メモリ内蔵（ブロック消去／書き込み禁止機能あり）
セルフ・プログラミング機能対応（ブート・スワップ／フラッシュ・シールド・ウインドウ機能あり）
オンチップ・デバッグ機能内蔵
パワーオン・リセット（POR）回路，電圧検出（LVD）回路内蔵
ウォッチドッグ・タイマ内蔵（低速内蔵発振クロックで動作可能）
乗除・積和演算命令に対応

●10 進補正（BCD）回路内蔵
I/O ポート：92 本
タイマ：16 ビット・タイマ　12 チャンネル
ウォッチドッグ・タイマ：1 チャネル
リアルタイム・クロック：1 チャネル
12 ビットインターバル・タイマ　：1 チャネル

タイマ出力　12 本（PWM 出力 10 本）

●シリアル・インタフェース：
CSI：2 チャンネル／簡易 I2C：2 チャンネル／UART：4ch／LIN　1ch

●A/D、D/A コンバータ
A/D コンバータ　20ch、8/10 ビット分解能、変換時間　2.1μsec
D/A コンバータ　なし

●スタンバイ機能：HALT, STOP, SNOOZE モード
動作周囲温度： TA=-40～85° C（A:民生用途）

●シリアルコネクタでＵＳＢ－ＳＩＯ絶縁変換器を接続し、ＵＳＢ使用可能。ＵＳＢ　Ｂコネクタ、ドライバＩＣ　ＦＴＤＩ社　ＵＳＢ－ＳＩＯ絶縁変換器はＦＴ２３２ＲＬ搭載。
●エミュレータＥ１によるデバック用コネクタ搭載。Ｃ言語による１行実行、ブレークポイント、変数参照等可能です。

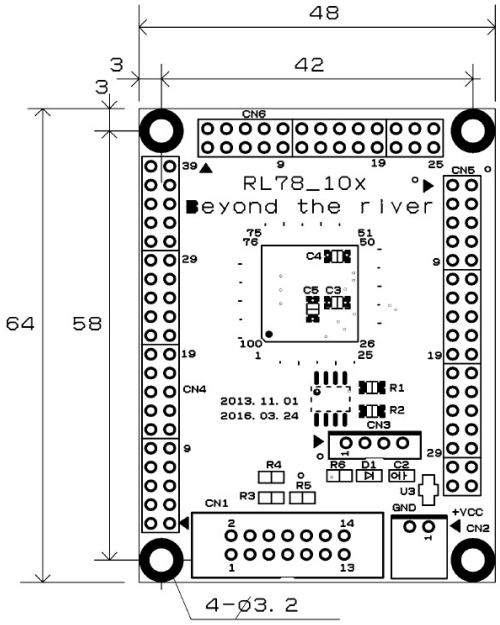
※１　速度比較は本マニュアル　１－１　ｆ：ＲＬ７８とＨ８／３００Ｈ、Ｒ８Ｃの速度比較をご参照下さい。

●弊社製品　ＲＬ７８＿１０１とＲＬ７８＿１０４の違い

ＲＬ７８＿１０１とＲＬ７８＿１０４に使用しているＣＰＵは同じ１００ピンで、形状、電源、ＧＮＤピン番号も同じでハードウエアの互換性がありますが、以下の部分がおおよそ違います。

機能　／　製品名	ＲＬ７８＿１０１	ＲＬ７８＿１０４
ＲＯＭ	９６ＫＢ	２５６ＫＢ
ＲＡＭ	８ＫＢ	２４ＫＢ
データ・フラッシュ	なし	８ＫＢ
Ｄ／Ａコンバータ	なし	８ｂｉｔ　２ｃｈ
コンパレータ	なし	２ｃｈ
３相相補ＰＷＭ　タイマーＲＤ	なし	あり
外部シリアルＥＥＰＲＯＭ	なし	２５ＬＣ２５６搭載済み
価格	２、８００円（税込み）	５、８００円（税込み）

基板大きさ（部品面）



ｃ：Ｅ１エミュレータ

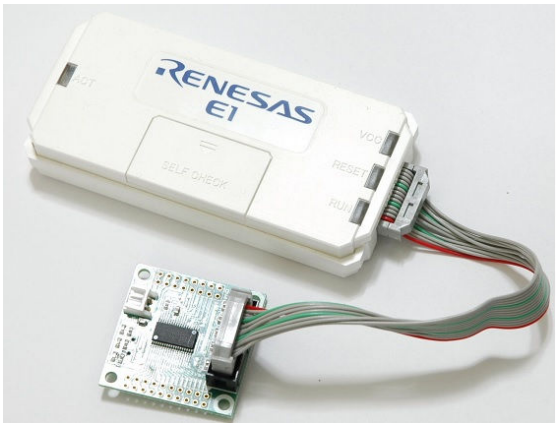
概要

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（インサーキットエミュレータ）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

- 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が添付されていて、ドライバーのインストールとセルフチェックを行った後に、ネットから開発環境ＣｕｂｅＳｕｉｔｅ＋とＣコンパイラのダウンロードを行います。

ｄ：無償版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+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＋　ＣＡ」の検索で表示されます。

統合開発環境とＣコンパイラが同時にダウンロードされます。

※ＣＳ＋には　ＣＳ＋　ｆｏｒ　ＣＣ（ＲＸ等）がありますので、混同にご注意下さい。

ｅ：開発セット添付ＣＤコピー、デバイスドライバ仮想ＣＯＭのインストゥール

以下省略

ｆ：ＲＬ７８とＨ８／３００Ｈ、Ｒ８Ｃの速度比較

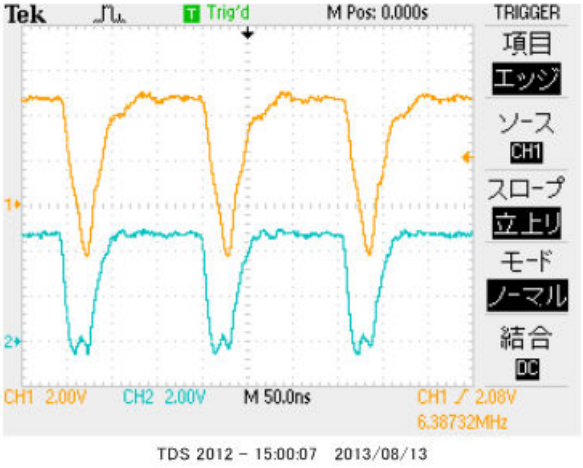
ＲＬ７８は、有名なＨ８／３０４８の代わりに検討される方も多いと思われますが、実行速度はどうなのでしょうか？　開発環境を含めて以前より進化していなければ使う意味がないとお考えの方も多いかと思われます。

ｆ－１　ポートアクセス速度比較

単純なポートアクセスプログラムで比較してみます。
ＲＬ７８のポートを１，０繰り返すプログラムです。

```
64                          void main(void)
65                          {
66      00187               R_MAIN_UserInit();
67                          /* Start user code. Do not edit comment generated here */
68                          while (1U)
69                          {
70      0018b                   P2 = 0x00;
71      0018d                   P2 = 0xff;
72
73
74      00190               }
75                          /* End user code. Do not edit comment generated here */
76      00192           }
```

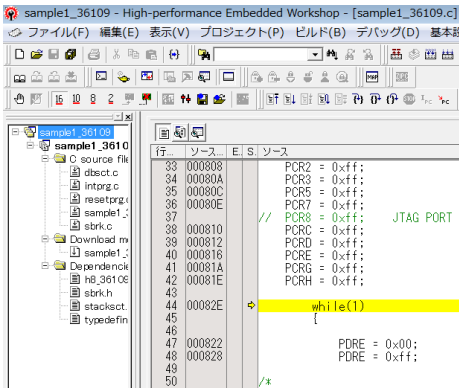
オシロスコープでＰ２０、Ｐ２１波形を観測すると６．３８７３２ＭＨｚという周波数でポートの１，０を繰り返すことが分かります。（クロック３２ＭＨｚ）

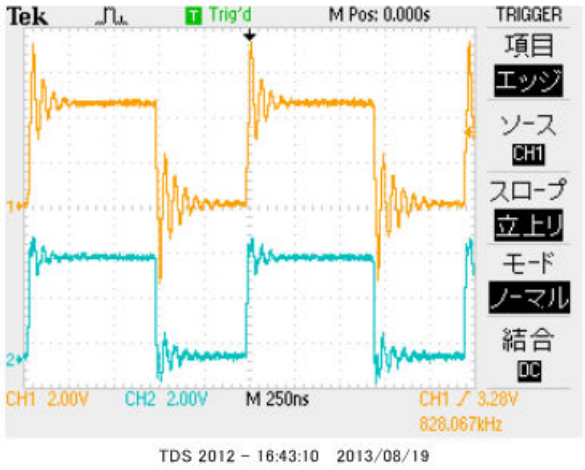


この命令の詳細は

ｗｈｉｌｅ（１Ｕ）
｛
Ｐ２　＝　０ｘ００；　　//ポートを０にする
Ｐ２　＝　０ｘｆｆ；　　//ポートを１にする
｝　　//上行にジャンプする
という３つの動作を行っています。波形が１から０に落ちて、上がる手前の時間が１命令の実行時間です。波形上約３０ｎｓｅｃ程度なので、カタログ値　３１．２５ｎｓｅｃと大きく相違は無いように思います。１クロックで１命令実行はＲＩＳＣ並みですね。１の時間が０に比べて長いのはポートを１にする、上行にジャンプするの２命令実行しているからです。

Ｈ８／３００Ｈコアを代表してＨ８／３６１０９を使用しました。基板名ＢＣＨ８３６１０９。ＨＥＷで同じ意味のコードを書き込みテストします。Ｈ８／３００ＨコアはＨ８／３０４８やＨ８／３０５２と同じです。

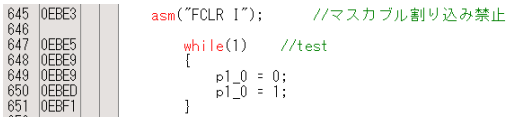


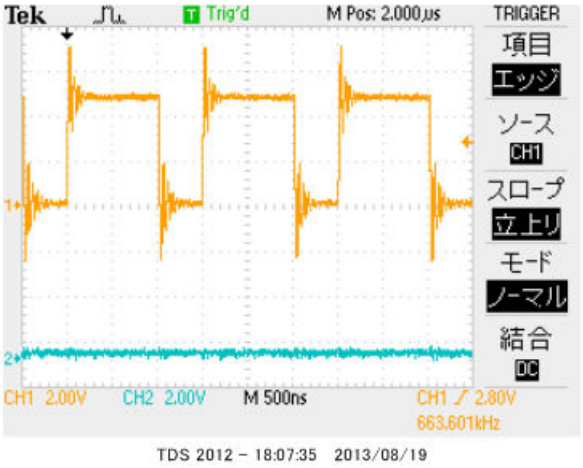


ポートＥを繰り返し、０、１しています。波形を観測すると８２８．０６７ＫＨｚとなりました。

６．３８７３２ＭＨｚ÷８２８．０６７ＫＨｚ≒７．７倍高速という驚きの結果になりました。（クロック２０ＭＨｚ）クロックを同じにしても、４．８倍違います。

次にＲ８Ｃを評価します。Ｒ８Ｃ/Ｍ１２Ａ（クロック２０ＭＨｚ）を使用して比較してみます。





６６３．６０１ＫＨｚとなりました。

ｆ－２　乗除演算速度の比較

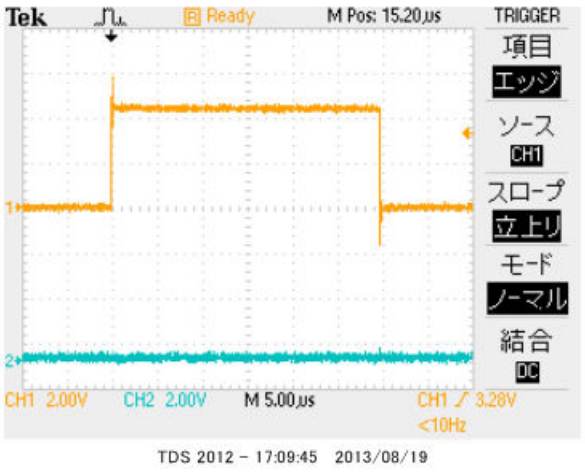
演算速度はどの程度違うでしょうか？　３２ｂｉｔの乗算、除算を行ってみました。
演算前にポートを立てて、演算後にポートを下ろすことにより、演算実行時間をオシロで観測しています。

Ｈ８－３６１０９　約３０μｓｅｃでした。

```
000874          while(1)
                {
00082E              ldata1 = 1234;
000838              ldata2 = 5678;

000840              PDRE = 0xff;
00084E              ldata3 = ldata1 * ldata2;
00085E              ldata4 = ldata2 / ldata1;
00086E              PDRE = 0;
                /*
                    PDR1 = 0;
                    PDR2 = 0;
                    PDR3 = 0;
                    PDR5 = 0;
                    PDR7 = 0;
                //  PDR8 = 0;
                    PDRC = 0;
                    PDRD = 0;
```

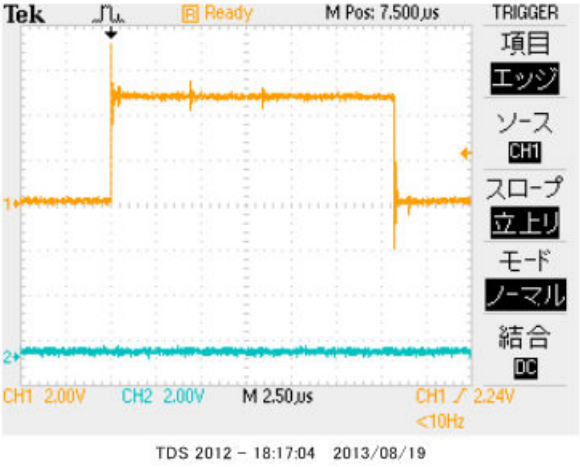
sample1...

Address	+0	+4	+8	+C
FFE40C	006AE9BC	00000004	00000000	00000000



Ｒ８Ｃ／Ｍ１２Ａの場合　約１５．５μｓｅｃでした。

```
0EBE3       asm("FCLR I");      //マスカブル割り込み禁止

0EBE5       while(1)    //test
0EBE9       {
0EBE9           ldata1 = 1234;
0EBF3           ldata2 = 5678;

0EBFD           p1_0 = 1;
0EC01           ldata3 = ldata1 * ldata2;
0EC1F           ldata4 = ldata2 / ldata1;
0EC3D           p1_0 = 0;
0EC41       }
```



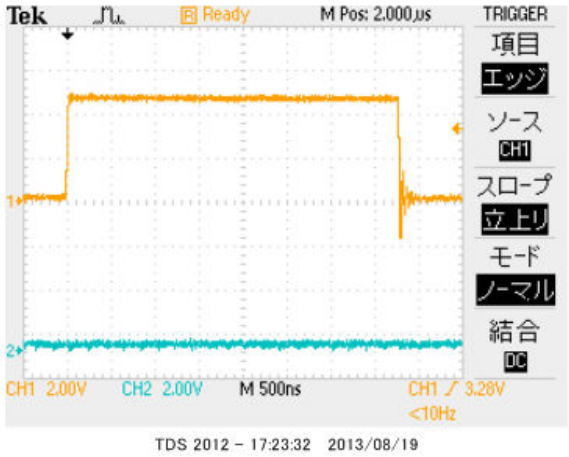
ＲＬ７８の場合　約３．８μｓｅｃでした。

ソースファイル

```
64
65                          unsigned long ldata1,ldata2,ldata3,ldata4;
66
67                          void main(void)
68                          {
69      001f1                   R_MAIN_UserInit();
70                              /* Start user code. Do not edit comment generated here */
71                              while (1U)
72                              {
73      001f5                       ldata1 = 1234;
74      001ff                       ldata2 = 5678;
75      00209                       P2 = 0xff;
76      0020c                       ldata3 = ldata1 * ldata2;
77      0022b                       ldata4 = ldata2 / ldata1;
78      0024a                       P2 = 0x0;
```

ソース＋逆アセンブラ

```
73:                 ldata1 = 1234;
   001f5    30d204      MOVW        AX,#4D2H
   001f8    bfbaef      MOVW        !_ldata1,AX
   001fb    f6          CLRW        AX
   001fc    bfbcef      MOVW        !0EFBCH,AX
74:                 ldata2 = 5678;
   001ff    302e16      MOVW        AX,#162EH
   00202    bfbeef      MOVW        !_ldata2,AX
   00205    f6          CLRW        AX
   00206    bfc0ef      MOVW        !0EFC0H,AX
75:                 P2 = 0xff;
   00209    cd02ff      MOV         P2,#0FFH
76:                 ldata3 = ldata1 * ldata2;
   0020c    afbaef      MOVW        AX,!_ldata1
   0020f    bdd8        MOVW        _@STBEG,AX
   00211    afbcef      MOVW        AX,!0EFBCH
   00214    bdda        MOVW        _@RTARG2,AX
   00216    afbeef      MOVW        AX,!_ldata2
   00219    bddc        MOVW        _@RTARG4,AX
   0021b    afc0ef      MOVW        AX,!0EFC0H
   0021e    fd8501      CALL        !@@lsmul
   00221    adda        MOVW        AX,_@RTARG2
   00223    bfc4ef      MOVW        !0EFC4H,AX
   00226    add8        MOVW        AX,_@STBEG
   00228    bfc2ef      MOVW        !_ldata3,AX
77:                 ldata4 = ldata2 / ldata1;
   0022b    afbeef      MOVW        AX,!_ldata2
   0022e    bdd8        MOVW        _@STBEG,AX
   00230    afc0ef      MOVW        AX,!0EFC0H
   00233    bdda        MOVW        _@RTARG2,AX
   00235    afbaef      MOVW        AX,!_ldata1
   00238    bddc        MOVW        _@RTARG4,AX
   0023a    afbcef      MOVW        AX,!0EFBCH
   0023d    fdb001      CALL        !@@ludiv
   00240    adda        MOVW        AX,_@RTARG2
   00242    bfc8ef      MOVW        !0EFC8H,AX
   00245    add8        MOVW        AX,_@STBEG
   00247    bfc6ef      MOVW        !_ldata4,AX
78:                 P2 = 0x0;
   0024a    f402        CLRB        P2
```



以上の結果をまとめると

ＣＰＵコア	クロック	ポートアクセス	乗除演算
ＲＬ７８	３２ＭＨｚ	６．３８ＭＨｚ	３．８μｓｅｃ
Ｈ８－３００Ｈ	２０ＭＨｚ	０．８２ＭＨｚ	３０μｓｅｃ
Ｒ８Ｃ	２０ＭＨｚ	０．６６ＭＨｚ	１５．５μｓｅｃ
結論		ＲＬ７８がＨ８－３００Ｈの７．７倍、Ｒ８Ｃの９．６倍高速。	ＲＬ７８がＨ８－３００Ｈの７．８倍、Ｒ８Ｃの４倍高速。

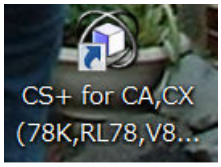
※測定結果はいずれも弊社製品比較です。

一般に設計が新しいCPUの方が、製造プロセスが微細化されている分、同じ機能であれば安価に製造できます。RL78は従来より優れたアーキテクチャのコアに、積和演算対応、10進補正回路等、高度な機能も内蔵し、かつ、今までより低消費電力、安価を目指して開発されたようです。

結論として、従来、Ｈ８／３０４８等をご使用の方々にも安心して使っていただける性能をもったＣＰＵだと思います。

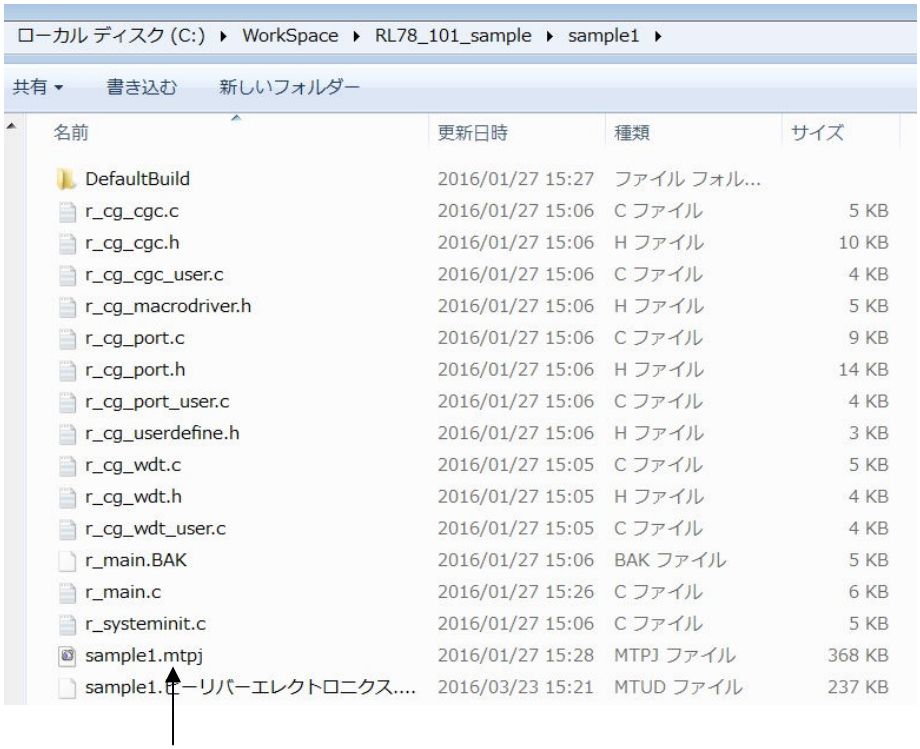
１－２　　動作、デバック

ａ：ＣＳ＋　ｆｏｒ　ＣＡ、ＣＸ起動、コンパイル、書き込み、動作



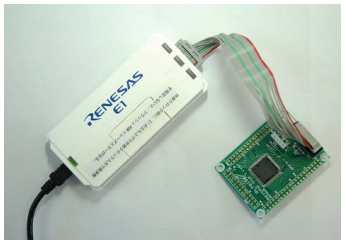
ＣＤに添付しているサンプルプログラムを使って、コンパイル、書き込み、動作の方法を示します。

ＣＳ＋　ｆｏｒ　ＣＡ，ＣＸを起動します。ここでは例としてＲＬ７８＿１０１￥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。
初めてのときは　ファイル　→　ファイルを開く　→　ｓａｍｐｌｅ１．ｍｔｐｊをダブルクリックします。

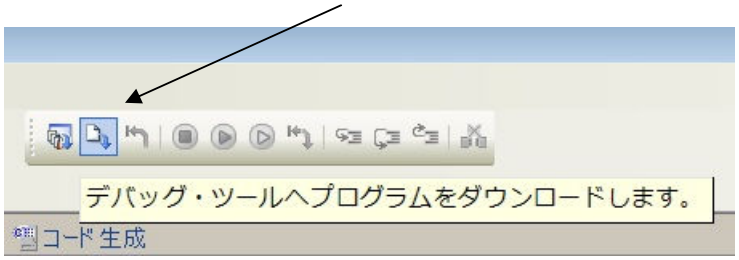


プロジェクトツリーとｒ＿ｍａｉｎ．ｃが表示されます。

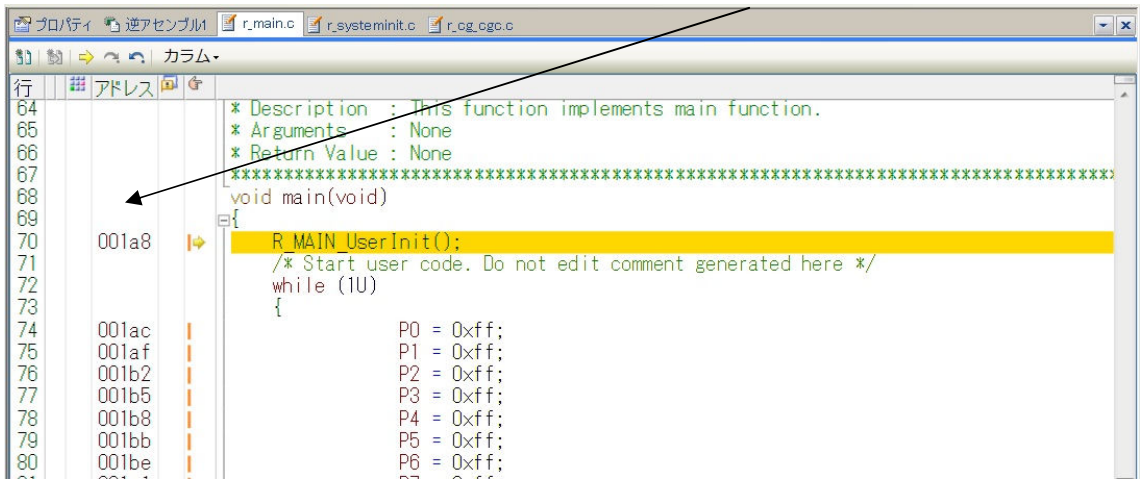
とりあえず、実行してみます。Ｅ１のケーブルを基板のＣＮ１に挿入します。電源はＥ１から供給しますので、不要です。（写真ご参考）



「デバック・ツールへプログラムを転送」をクリック。

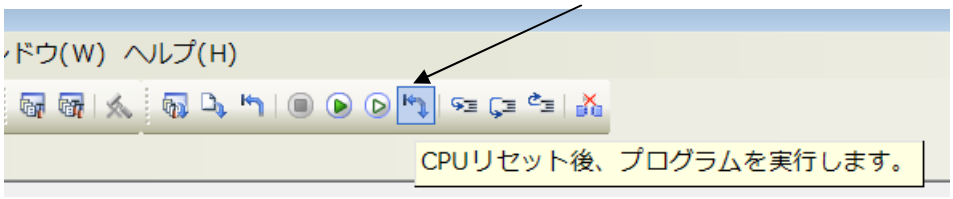


上手く転送できると、今まで表示されていなかったプログラムの絶対アドレスが表示されます。Ｅ１から電源がＣＰＵ基板に供給されます。Ｅ１のＶＣＣのＬＥＤの色が緑からオレンジに変わります。実行行を示す黄色いカーソルが表示されます。

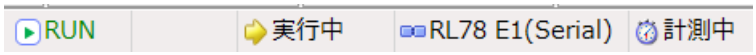


ここまでいかなかった場合、Ｅ１のインストゥールをご検証願います。

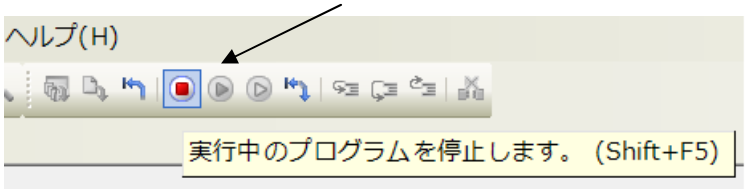
次に、プログラムを動作させます。「ＣＰＵリセット後、プログラムを実行」をクリック。



Ｅ１のＲＵＮ（赤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。



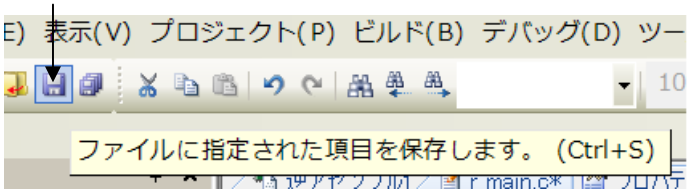
ここまで確認できましたら、一度止めます。



ｍａｉｎ関数のｌｗａｉｔの数値２箇所をキーボードを押して１桁０を増やしてみます。

```
    P0 = 0xff;
    P1 = 0xff;
    P2 = 0xff;
    P3 = 0xff;
    P4 = 0xff;
    P5 = 0xff;
    P6 = 0xff;
    P7 = 0xff;
    P8 = 0xff;
    P10 = 0xff;
    P11 = 0xff;
    P12.0 = 1;
    P13.0 = 1;
P14 = 0xff;
P15 = 0xff;
lwait(1000000);
    P0 = 0;
    P1 = 0;
    P2 = 0;
    P3 = 0;
    P4 = 0;
    P5 = 0;
    P6 = 0;
    P7 = 0;
    P8 = 0;
    P10 = 0;
    P11 = 0;
    P12.0 = 0;
    P13.0 = 0;
P14 = 0;
P15 = 0;

lwait(1000000);
```

セーブして



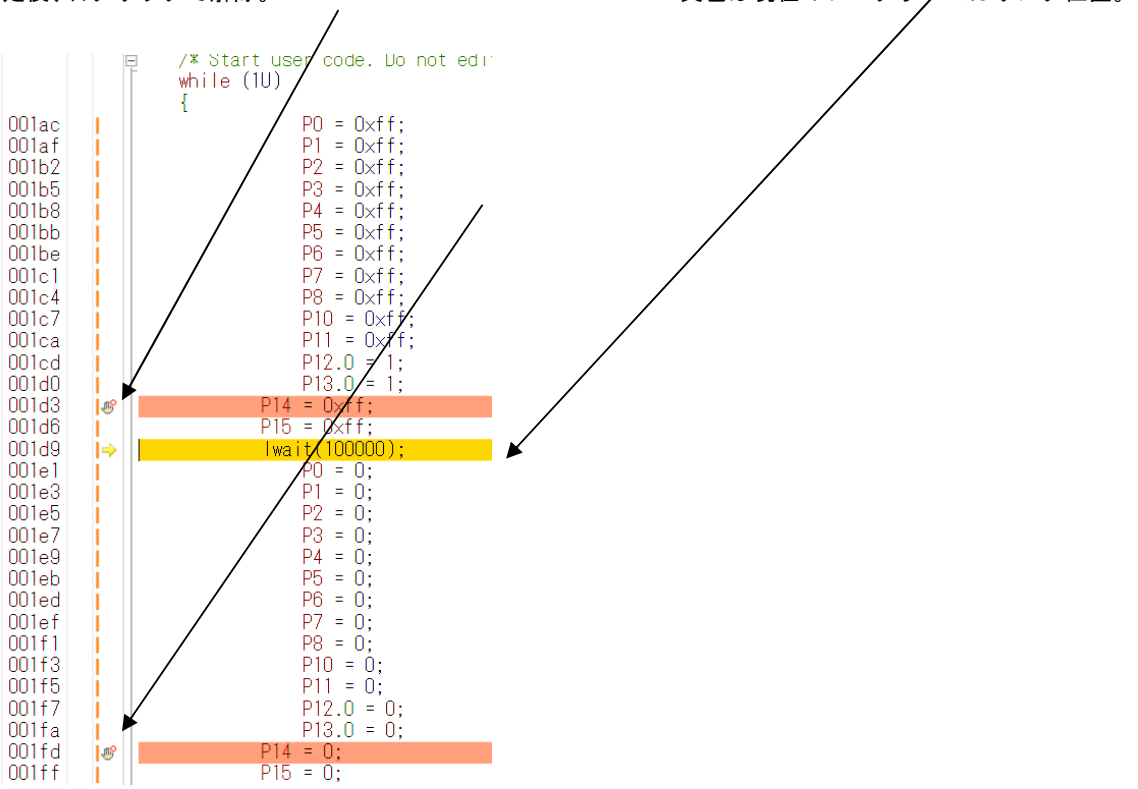
さきほどの、

「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。

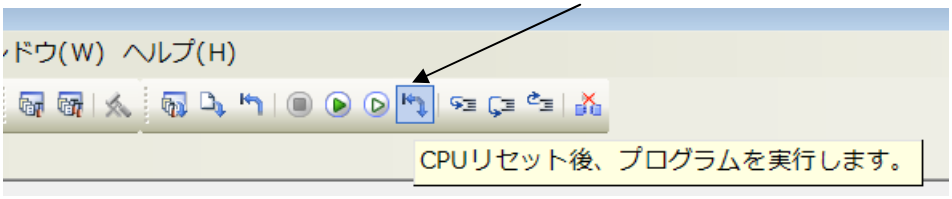
「ＣＰＵリセット後、プログラムを実行」をクリック。

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。設定はカーソルを行にもっていき、右クリック。設定後、右クリックで解除。　　黄色は現在のプログラムカウンタ位置。

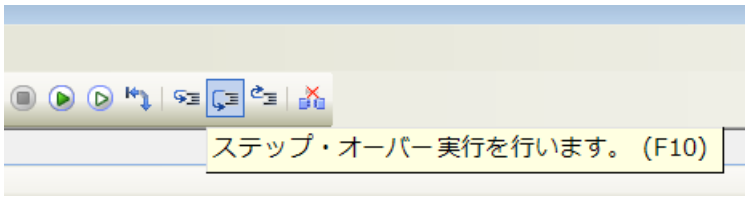


プログラムを動作させます。「ＣＰＵリセット後、プログラムを実行」をクリック。



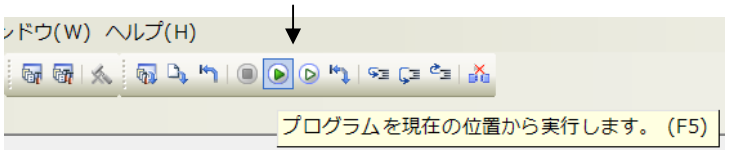

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    while (1U)
    {
                P0 = 0xff;
                P1 = 0xff;
                P2 = 0xff;
                P3 = 0xff;
                P4 = 0xff;
                P5 = 0xff;
                P6 = 0xff;
                P7 = 0xff;
                P8 = 0xff;
                P10 = 0xff;
                P11 = 0xff;
                P12.0 = 1;
                P13.0 = 1;
            P14 = 0xff;
            P15 = 0xff;
             lwait(100000);
```

プログラムの実行はブレークポイントで停止します。



ステップオーバー実行をクリックし、１行進めます。
ＬＥＤ　Ｄ１　は　Ｐ１４＝０ｘｆｆ；命令により点灯します。

「プログラムを現在の位置から実行」します。




```
                P13.0 = 1;
            P14 = 0xff;
            P15 = 0xff;
            lwait(100000);
                P0 = 0;
                P1 = 0;
                P2 = 0;
                P3 = 0;
                P4 = 0;
                P5 = 0;
                P6 = 0;
                P7 = 0;
                P8 = 0;
                P10 = 0;
                P11 = 0;
                P12.0 = 0;
                P13.0 = 0;
            P14 = 0;
            P15 = 0;

            lwait(100000);
        ;
    }
    /* End user code. Do not edit comment generated here */
```

次のブレークポイント設定行　Ｐ１４＝０でプログラムは停止します。この行はまだ実行されていません。
ステップオーバー実行をクリックし、１行進めます。
ＬＥＤの消灯が確認できると思います。

```
            P14 = 0xff;
            P15 = 0xff;
            lwait(100000);
                P0 = 0;
                P1 = 0;
                P2 = 0;
                P3 = 0;
                P4 = 0;
                P5 = 0;
                P6 = 0;
                P7 = 0;
                P8 = 0;
                P10 = 0;
                P11 = 0;
                P12.0 = 0;
                P13.0 = 0;
            P14 = 0;
            P15 = 0;

            lwait(100000);
        ;
    }
    /* End user code. Do not edit comment generated here */
```

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。

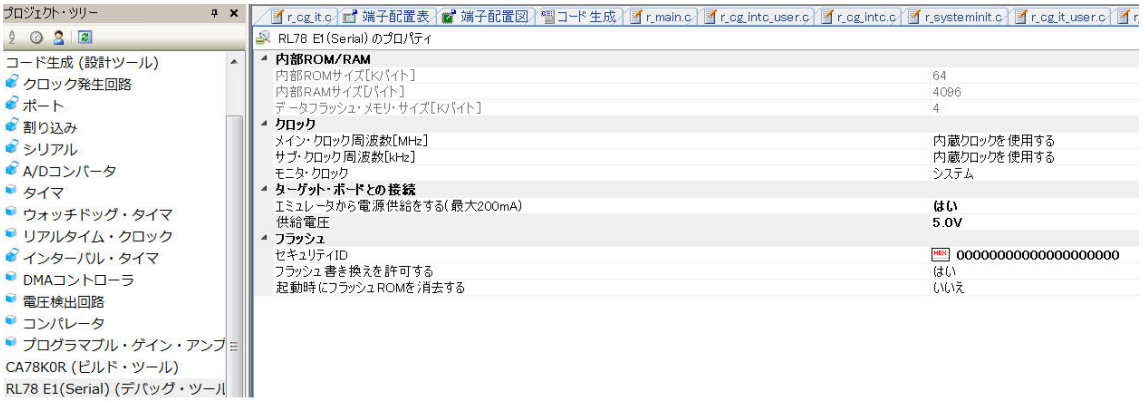
ｂ：新しいプログラムを作る

ＣＳ＋　ｆｏｒ　ＣＡ，ＣＸでのプログラム開発は、例えばＨＥＷと比べると大きく異なる部分があります。その一つはプログラムを書く前に、端子機能を入力すること（端子機能を入力しないと、プログラムが書けません）です。これはハードウエア的に端子の割り振りが終了していないといけないことになります。
二つ目は　割り振りにより決まる、端子を使用するための関数がコード生成機能で自動的に準備されることです。例えばＳＩＯを使用するように端子を割振り、コード生成でＳＩＯを使用すると設定し、「コード生成」ボタンをクリックすることにより、ＳＩＯを使用するためのイニシャル、送信、受信関数が作成されます。これによりプログラマはそれらを書く必要がありません。アプリケーションのみに集中できるように考えられています。
経験のある方ほど他と大きく異なる開発方法に戸惑いがあるかもしれません。すこし操作してみれば、ＣＳ＋の機能が、より簡単に、より短時間に、より正確に開発が行えるよう考慮されているのが理解できると思います。例えばハード的に入力しか使えない端子を出力で使おうとしても設定出来ませんので、ミスが発生しません。ソフトウエアの部分でも、提供される関数を使用することにより、品質が底上げされます。

開発の詳細は順を追って説明します。

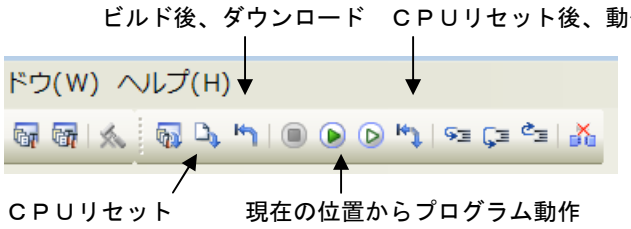
以下省略

ｂ－３：Ｅ１から電源供給
デバッカにはＥ１を使用します。Ｅ１から電源を供給する設定は



ＲＬ７８Ｅ１（Ｓｅｒｉａｌ）（デバックツール）を右クリック→プロパティで上記画面になりますので、エミュレータから電源を供給するを「はい」、電圧を「５Ｖ」にして下さい。３．３Ｖでも問題なく動作します。外部電源を使用する場合、ダウンロード前に電源をＯＮさせる必要があります。

ビルド後、ダウンロードを行いＥ１とうまく通信が出来るとデバックのためのボタンがアクティブになります。



ＣＰＵリセット後、動作をクリックするとプログラムが初めから動作します。

ｂ－４：コード生成後の初期値の変更

「コード生成」後、プログラムをある程度書いた後の仕様の変更に、再び「コード生成」を行うと、既にプログラムを書いた部分が初期化されて消えてしまう場合があります。

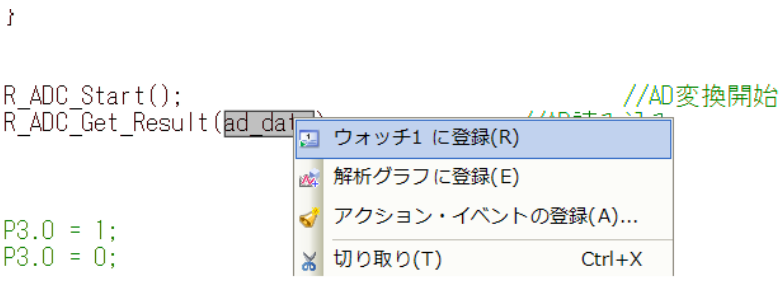
そうならないために Start　ｕｓｅｒ　ｃｏｄｅとＥｎｄ　ｕｓｅｒ　ｃｏｄｅの間にユーザープログラム、コメント、定数、変数宣言等を書いてください。

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

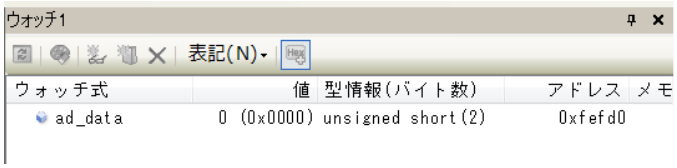
ここにユーザーコードを書けば、「コード生成」を繰り返しても、消されることはありません！！

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

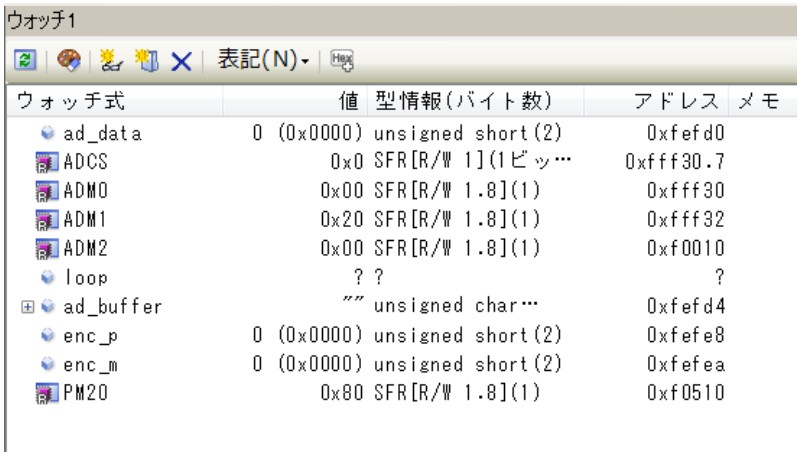
ｂ－５：変数を見る



見たい変数をコピーして右クリック→ウオッチ１に登録



ｂ－６：変数変化を実行中に確認する



ウオッチウインドウはデバックに非常に便利な窓ですが、そのままでは動作中は更新されません。そこで、

以下省略

２．サンプルプログラム

２－１　ｓａｍｐｌｅ１　出力ポートのＯＮ，ＯＦＦ

```
/***********************************************************************************************
* Function Name: main
* Description  : This function implements main function.
* Arguments    : None
* Return Value : None
***********************************************************************************************/

①void lwait(unsigned long time)
{
        while(time != 0)
        {
                time--;
        }

}



②void main(void)
{
   ③ R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
   ④ while (1U)
    {
                P0 = 0xff;
                P1 = 0xff;
                P2 = 0xff;
                P3 = 0xff;
                P4 = 0xff;
                P5 = 0xff;
                P6 = 0xff;
                P7 = 0xff;
                P8 = 0xff;
                P10 = 0xff;
                P11 = 0xff;
                P12.0 = 1;
                P13.0 = 1;
     ⑤          P14 = 0xff;
                P15 = 0xff;
     ⑥          lwait(100000);
                P0 = 0;
```

```
                P1 = 0;
                P2 = 0;
                P3 = 0;
                P4 = 0;
                P5 = 0;
                P6 = 0;
                P7 = 0;
                P8 = 0;
                P10 = 0;
                P11 = 0;
                P12.0 = 0;
                P13.0 = 0;
    ⑦           P14 = 0;
                P15 = 0;

    ⑧           lwait(100000);
        ;
    }
    /* End user code. Do not edit comment generated here */
}
```

【　解説　】

①void lwait(unsigned long time)

下のｍａｉｎ関数から呼ばれるウエイトルーチンです。

②void main(void)

{

メインルーチンです。

③ R_MAIN_UserInit();

コード生成によって自動的に作られた初期設定関数をコールしています。この初期設定はメインルーチンの下にあります。割込み許可ＥＩを実行しているだけです。

```
    /* Start user code. Do not edit comment generated here */
④   while (1U)
    {
```

以下を無限ループします。

⑤P14 = 0xff;

Ｐ１４に０ｘｆｆを設定しています。Ｐ０の出力設定は、コード生成により、ｒ＿ｓｙｓｔｅｍｉｎｉｔ．ｃの中のＲ￥ｓｙｓｔｅｍｉｎｉｔ（）関数の中にあり、リセット解除後、自動実行されます。

出力に設定されたＰ１４のポートは全て１になります。よってＰ１４５も１（電源電圧、５Ｖの場合、約５Ｖ、３．３Ｖの場合、約３．３Ｖ）が出力され、接続されているＬＥＤ　Ｄ１に電流が流れ点灯します。

⑥　　lwait(100000);

設定された数が０になるまでループするウエイト関数です。

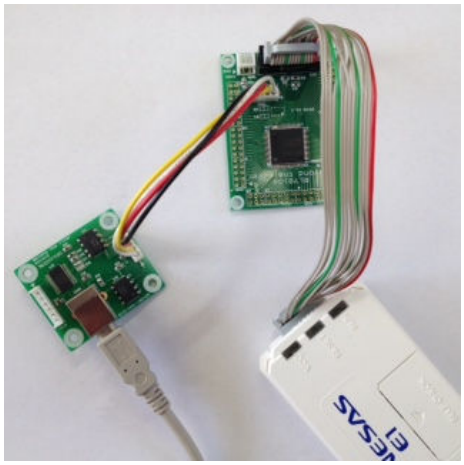
⑦　　P14 = 0;

Ｐ１４に０を設定しています。Ｐ１４５に接続されているＬＥＤ　Ｄ１は消灯します。

⑧　　lwait(100000);

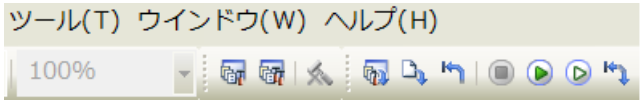
点灯も消灯と同じ時間、保持されます。

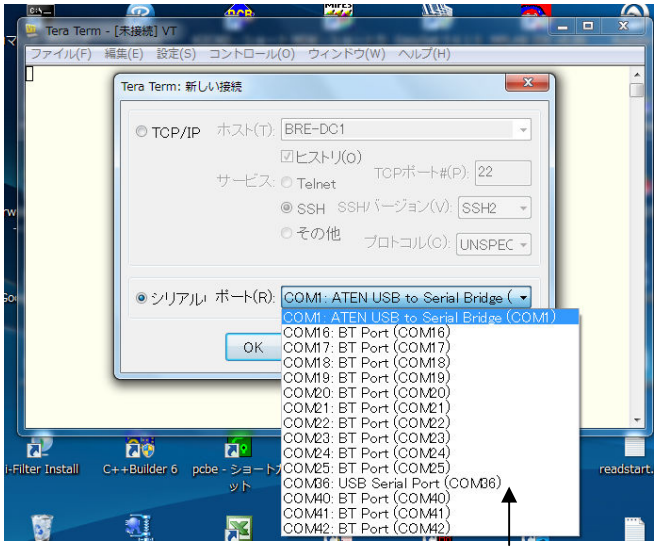
２－２　ｓａｍｐｌｅ２　ＳＩＯ（ＵＳＢ）１文字送受信



【　概要　】

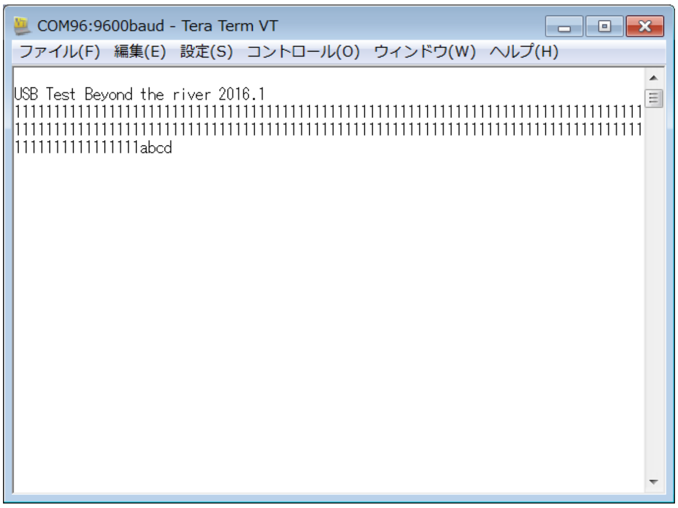
ＵＳＢ出力をパソコンと接続し、データのやり取りを行います。添付のＵＳＢ－ＳＩＯ絶縁変換器のＣＮ１とＣＰＵボードのＣＮ３を添付のケーブルで接続します。ＵＳＢミニケーブルをパソコンと接続します。お手数ですが、無料のターミナルプログラム、テラタームやハイパーターミナルなどのターミナルプログラムを使用しますので、無い方は、ネットで検索し、インストゥール願います。例ではテラタームで行います。９６００ｂｐｓに設定して下さい。ＵＳＢケーブルでパソコンとつなげて、認識するまで３０秒以上かかる場合があります。時間経過後テラタームを立ち上げて下さい。





テラタームの立ち上げでＵＳＢ　Ｓｅｒｉａｌ　Ｐｏｒｔと出てくればＦＴ２３２ＲＬが準備完了です。なお、マイコン基板の電源がＯＦＦになってもＵＳＢ－ＳＩＯ絶縁変換器の電源はＯＦＦになりません。ＵＳＢケーブルを抜いた時に再び、上記設定が必要になります。

ＣＰＵリセット後、プログラム実行で、テラターム画面に「ＵＳＢ　Ｔｅｓｔ　、、、、」と表示され、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、表示されるようでしたら正常です。



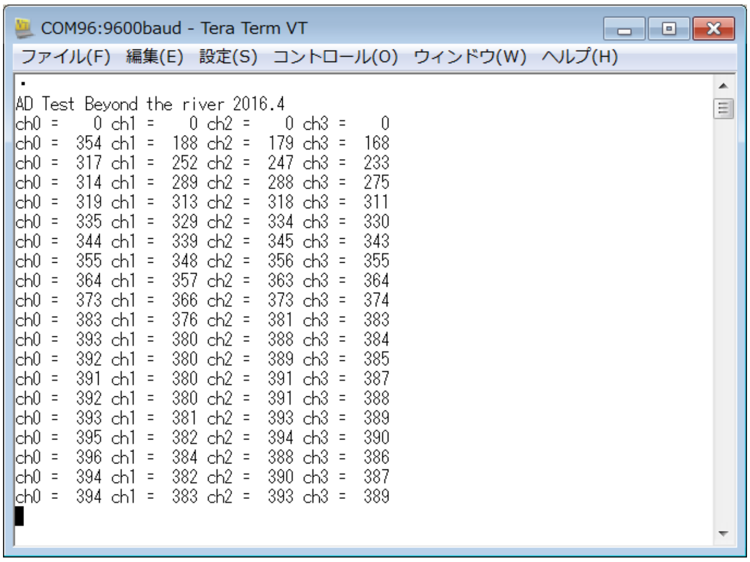
【　プログラム　】

以下省略

２－３　ｓａｍｐｌｅ３　Ａ／Ｄ変換をＵＳＢ出力

【　動作概要　】

ＡＮＩ０，１，２，３を入力とし、Ａ／Ｄ変換した値をＵＳＢからパソコンに送ります。



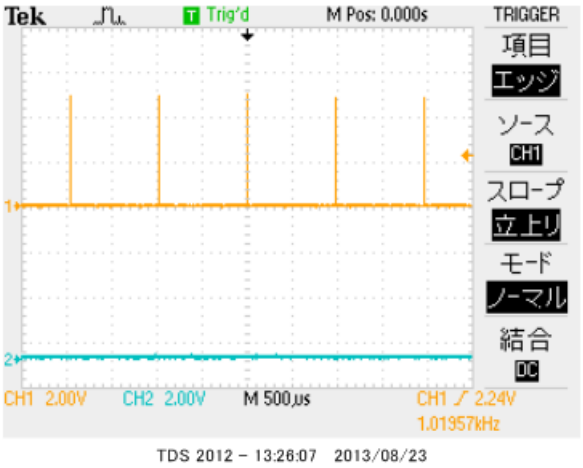
パソコン側のテラタームではＡＤの数値が繰り返し表示されます。初めの数回は０表示、ＡＮＩｘオープンで０以外、＋５Ｖ接続で１０２３、ＧＮＤ接続で０が表示されます。

以下省略

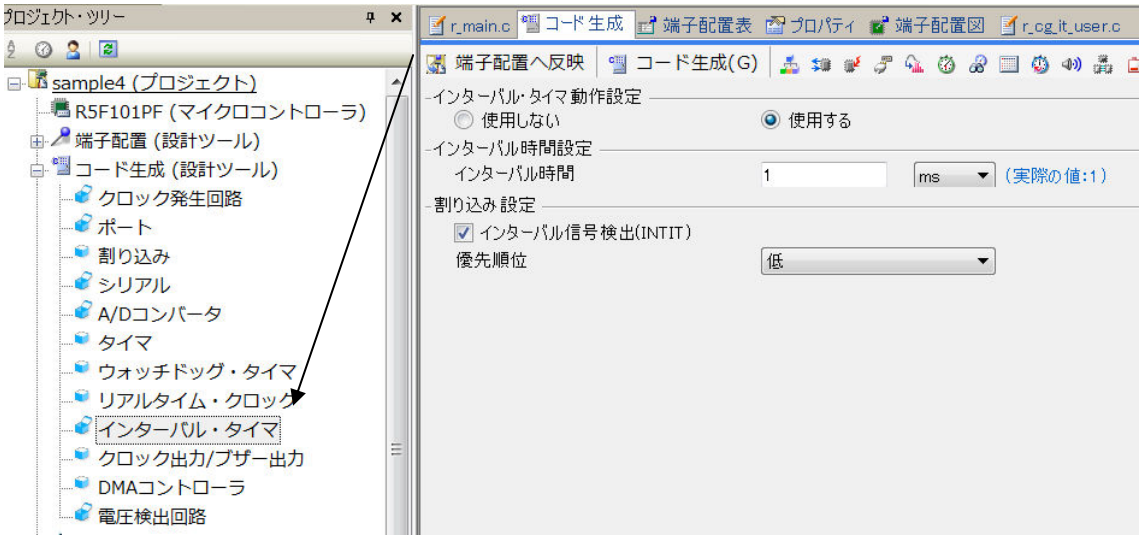
２－４　ｓａｍｐｌｅ４　割り込み

【　動作概要　】

ｓａｍｐｌｅ４を動作させます。オシロスコープがあればＣＮ４　１３番　Ｐ１４５を観測すると、以下のような波形が観測できます。



これはコード生成、インターバルタイマで定周期割り込みを設定したためです。

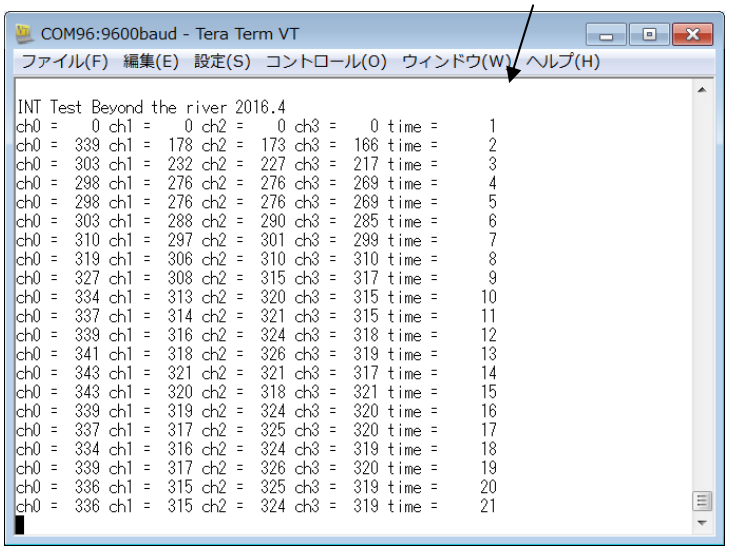


１ｍｓｅｃ毎に割り込みが入ります。ｒ＿ｃｇ＿ｉｔ＿ｕｓｅｒ．ｃの中に自動的に以下の関数のスケルトンが作成されますから、１ｍｓｅｃに１回実行したいことを書きます。下記例ではＰ０５のＯＮ，ＯＦＦ，タイマーをデクリメントしています。さきほどのオシロで観測された波形はここで作成されています。

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

    P14.5 = 1;                          //マーカー
```

```
    if(int_time != 0)
    {
        int_time--;
    }

    P14.5 = 0;                          //マーカー

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

ｍａｉｎではこのｉｎｔｔｉｍｅを使い、ｓａｍｐｌｅ３ではアバウトだった表示時間をちゃんと規定しています。　　１秒毎に表示



【　プログラム　】

以下省略

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

    R_UART3_Start();                    //UART3スタート
 ① R_IT_Start();                        //定周期割込みスタート

    R_UART3_Receive(rx_data,1);         //1文字受信
    rx_flg = 0;                         //フラグクリア
```

```
    tx_end_flg = 0;


//オープニングメッセージ出力

    R_UART3_Send(String_0,37);              //Opening message
        tx_end_wait();                      //送信終了まち

        int_time = 0;
        ltime = 0;

    while (1U)
    {
                if(int_time == 0)
                {
                ②       int_time = 1000;
                        ltime++;

                        ad_cnt = 0;
                        R_ADC_Start();                          //AD変換開始
                        while(ADCS)                             //変換待ち
                                ;

sprintf(ad_buffer, "ch0 = %4d ch1 = %4d ch2 = %4d ch3 = %4d time
= %6d¥r¥n",ad_data0,ad_data1,ad_data2,ad_data3,ltime);
                                                        //ad_buffに10進ASCII変換してセーブ
                        R_UART3_Send(ad_buffer,sizeof(ad_buffer));      //uart出力
                        tx_end_wait();                          //送信終了待ち
                }

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

【　解説　】

① R_IT_Start();//interval timer 定周期割り込みスタート　r_cg_it.c よりコピーして使用

インターバルタイマーを使用するときに、ｒ＿＿ｃｇ＿＿ｉｔ．ｃよりコピーするか、書いてください。

```
②inttime = 1000;                        /       /1msec×1000＝1秒
        while(inttime != 0)                     // 1 秒経過待ち
                ;
"¥n¥rINT Test Beyond the river¥n¥r"
```

をテラタームの画面上に１秒表示します。

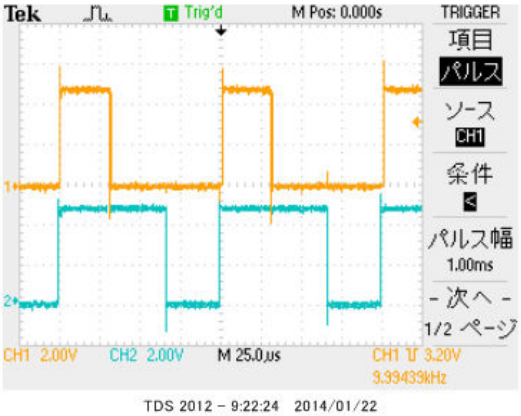
他の割り込みも同様に関数が作成されますので、そこに割り込み処理を書くことになります。ベクタ等、ハードウエア的な知識が不要で割り込みプログラムが作成でき、非常に便利です。

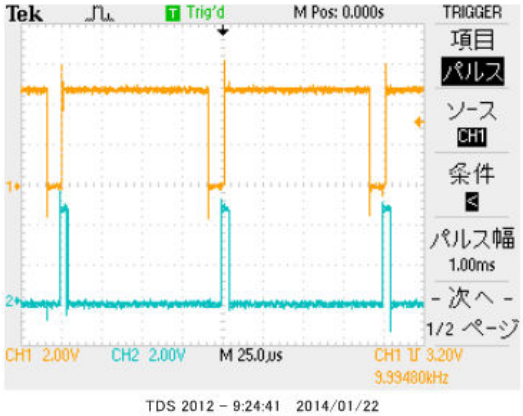
２－５　ｓａｍｐｌｅ５　PWM出力

【　動作　】

ＲＬ７８のタイマ・アレイ・ユニットを使用してPulse-Width Modulation（パルス幅変調）出力を製作します。波形はそれぞれＰ１６（ＴＯ０１　ＣＮ６　１２番）、Ｐ３１（ＴＯ０３　ＣＮ５　２０端子）から出力されます。

波形は下図のように、周期が変わらず、時間経過によって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ます。マイコンと親和性が良い、トランジスタをスイッチとして使用するのでエネルギー効率が良い、などの理由で現代では様々な用途に使われています。





プログラムはＰＷＭを出力するために以下の設定になっています。

【　ピン設定　】

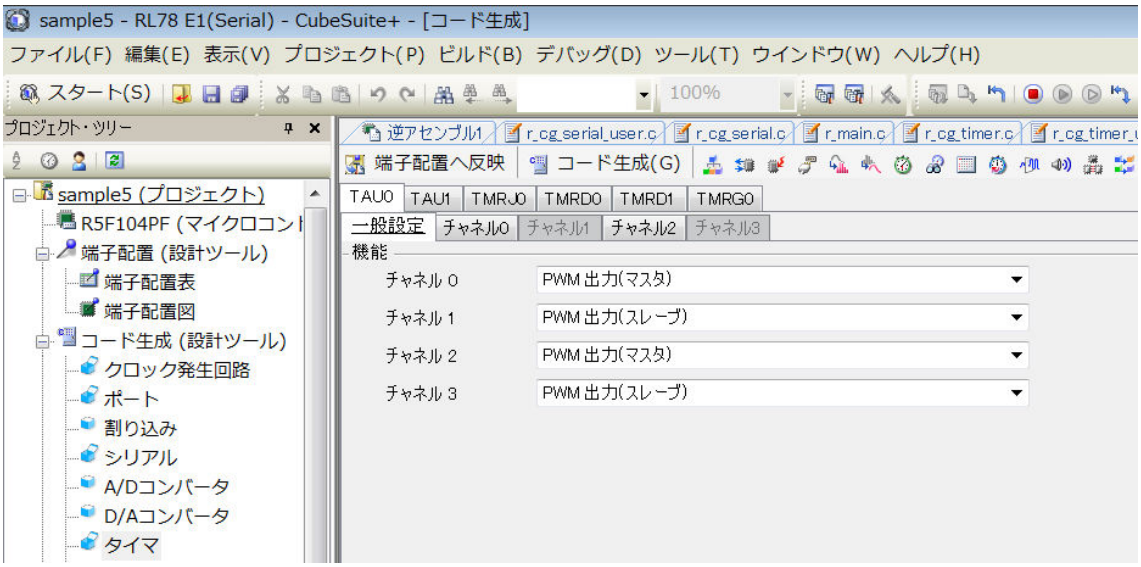
Ｐ１６をＴＯ０１　１６ビット・タイマ出力に設定。

62	P17/TI02/TO02/TRDIOA0/TRDCLK0/IVCMP0	Free	–	–		
63	P16/TI01/TO01/INTP5/TRDIOC0/IVREF0	TO01	O	–		16ビット・タイマ01 出力
64	P15/_SCK20/SCL20/TRDIOB0	Free	–	–		

Ｐ３１をＴＯ０３に設定。

27	P63/SDAA1	Free	–	–		
28	P31/TI03/TO03/INTP4	TO03	O	–		16ビット・タイマ03 出力
29	P64/TI10/TO10	Free	–	–		

【　コード生成設定　】



コード生成→タイマ→ＴＡＵ０を選択、チャンネル０と２をマスタ、チャンネル１と３をスレーブに設定。

以下省略

２－６　ｓａｍｐｌｅ６　三角、対数、平方根関数を使う

【　動作　】

浮動小数点３２ビットｄｏｕｂｌｅを使ってｌｏｇ、ｓｉｎ、√　の演算、キャストを行います。

【　プログラム　】

①#include <math.h>

②double d1,d2,d3;
③short s1,s2,s3;

④#define PI 3.14159265

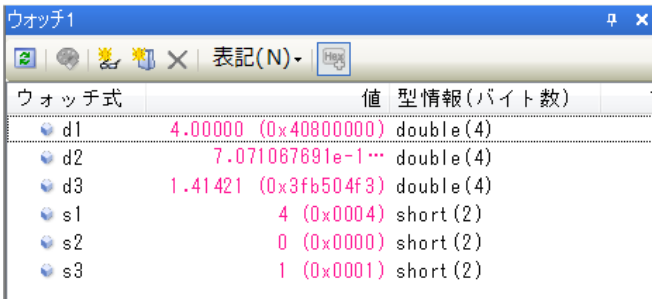
```
void main(void)
{
      R_MAIN_UserInit();
      R_UART3_Start();

       R_UART3_Receive(rx_data,1);
      rx_flg = 0;
      tx_end_flg = 0;

//オープニングメッセージ出力
```

```
        R_UART3_Send(String_0,37);          //Opening message
        tx_end_wait();                      //送信終了まち


⑤       P14.5 = 1;                          //時間測定マーカーＯＮ
⑥       d1 = log10(10000);
⑦       P14.5 = 0;                          //時間測定マーカーＯＦＦ

⑧       d2 = sin((PI/180)*45);
        P14.5 = 1;                          //時間測定マーカーＯＦＦ

⑨       d3 = sqrt(2);
        P14.5 = 0;                          //時間測定マーカーＯＦＦ

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

        while(1U)
        {

        }

}
```

【　解説　】

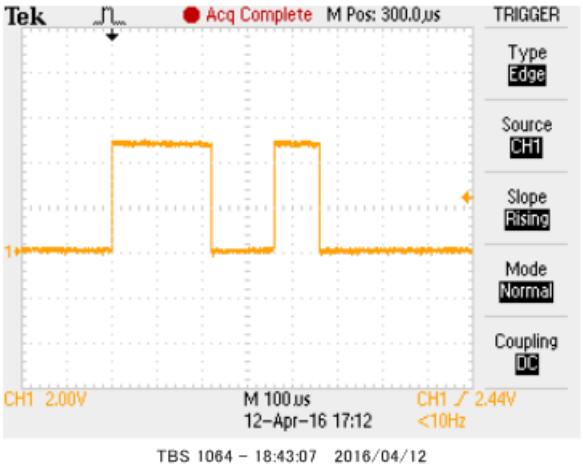
以下省略

演算結果ですが、事前予想通りとなりました。



演算速度ですが、
ｌｏｇ１０（１００００）が約２２０μｓｅｃ、ｓｉｎ（４５°）が１３０μｓｅｃ、√２が１００μｓｅｃ程度でした。

Ｐ１４５をオシロスコープで観測した波形。初めの山がｌｏｇの演算時間、谷がｓｉｎ、次の山が√の演算時間です。



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　