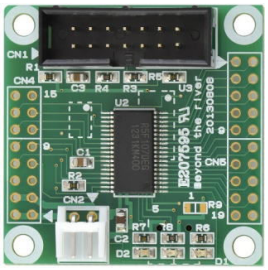
ＢＣＲＬ７８１０７　マイコン開発セット　マニュアル

第２版　２０１５．５．２１　CubeSuite+　→　CS+変更　　第１版２０１３．８．１３

【　製品概要　】

本マニュアルはＢＣＲＬ７８１０７　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：ＢＣＲＬ７８１０７　ＣＰＵボードの特徴
ｃ：Ｅ１エミュレータ（デバッカ）
ｄ：無償のＣＳ＋　ｆｏｒ　ＣＡ、ＣＸ、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Ｄ２ＸＸのインストゥール
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　ＣｕｂｅＳｕｉｔｅ+　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）、ＥＥＰＲＯＭ読み書き
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

ＢＣＲＬ７８１０７　ＣＰＵ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（本誌）
電源ケーブル、ＵＳＢ（ミニ）ケーブル

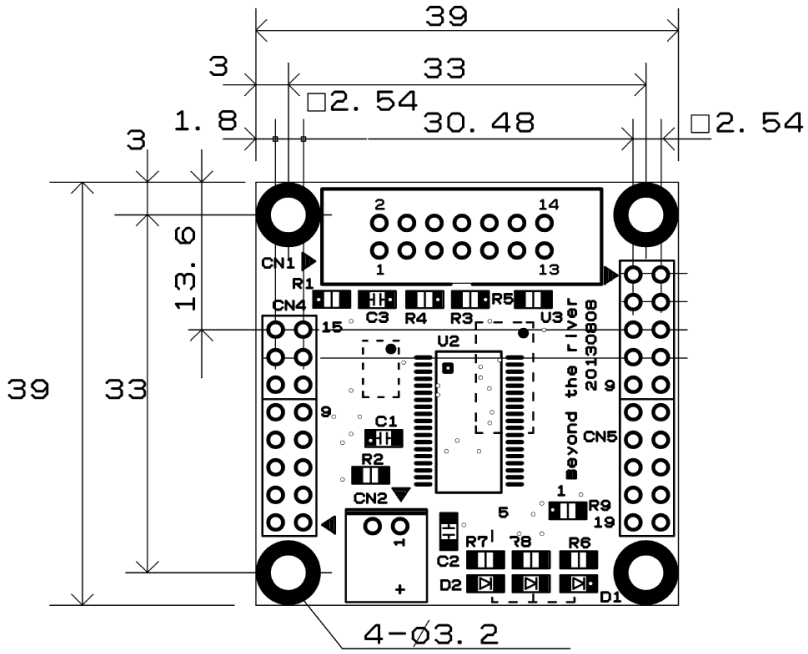
※開発に必要なルネサスエレクトロニクス社製デバッカＥ１は同封されておりません。別途必要です。

ｂ：ＢＣＲＬ７８１０７　ＣＰＵボードの特徴

●高性能、低消費電力、低コストな新設計ＲＬ７８コアを使用。１．３９ＤＭＩＰＳ/ＭＨｚ、４６μＡ／ＭＨｚ。３２ＭＨｚ±１%の高精度内蔵オシレータ　※１
●ＲＬ７８/Ｉ１Ａ（Ｒ５Ｆ１０７ＤＥ）は産業、インフラ、情報アプリケーションに特化した強力な周辺機能（高性能ＰＷＭタイマ、ＬＩＮ－ｂｕｓ、ＤＡＬＩ通信機能）を搭載。３８ピン。
●内蔵高速オシレータ　　３２ＭＨｚ（２．７～５．５Ｖ）。最小命令実行時間３１．２５ｎｓｅｃ。
●内蔵低速オシレーター　１５ＫＨｚ（ＴＹＰ）　ＣＰＵクロックとしては使用不可。
●メモリ容量　フラッシュＲＯＭ６４Ｋバイト、ＲＡＭ４Ｋバイト、データフラッシュ４Ｋバイト。
電源を切ってもデータが保持されるＥＥＰＲＯＭ　２５ＬＣ２５６（容量３２、７６８ＢＹＴＥ）搭載　ライブラリ添付※２
●基板大きさ、超小型３９×３９×１５ｍｍ
●動作電圧電流　３．３Ｖ～５．５Ｖ、１６ｍＡ　ＴＹＰＥ（５Ｖ、ＵＳＢ使用、３２ＭＨｚ動作時）
　　最低２．７Ｖから動作可能（ＢＣＲＬ７８１０７Ｓタイプ　※２）
●豊富な周辺機能
Ｉ／Ｏポート　合計３４、Ａ／Ｄ変換器：１０ビット分解能　１１ｃｈ、プログラマブルゲインアンプ　６ｃｈ、ＵＡＲＴ　３ｃｈ（１ｃｈはＬＩＮ－ｂｕｓ、ＤＭＸ５１２、ＤＡＬＩ通信対応）
タイマ８ｃｈ（ＰＷＭ出力３ｃｈ、１ｎｓｅｃ分解能可能、６４ＭＨｚＰＬＬ＋ディザリング）、乗除算・積和演算器内蔵、オンチップデバック機能内蔵
●ＵＳＢ搭載　ミニＢコネクタ、ドライバＩＣ　ＦＴＤＩ社　ＦＴ２３２ＲＬ搭載。※２
●デバッカ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。

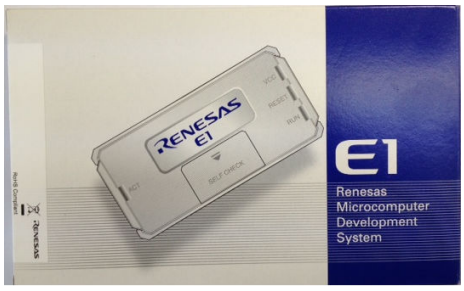
※１　速度比較は本マニュアル　１－１　ｆ：ＲＬ７８とＨ８／３００Ｈ、Ｒ８Ｃの速度比較をご参照下さい。
※２　製品はＣＰＵ＋デバック用コネクタ実装済みのＢＣＲＬ７８１０７Ｓ、ＳにＵＳＢインターフェイス＋ＥＥＰＲＯＭを追加したＢＣＲＬ７８１０７Ｍがあります。開発セットはＭが同梱されています。

基板大きさ（部品面）



ＵＳＢミニＢコネクタ、ＦＴ２３２ＲＬ、２５ＬＣ２５６は裏面搭載。

ｃ：Ｅ１エミュレータ



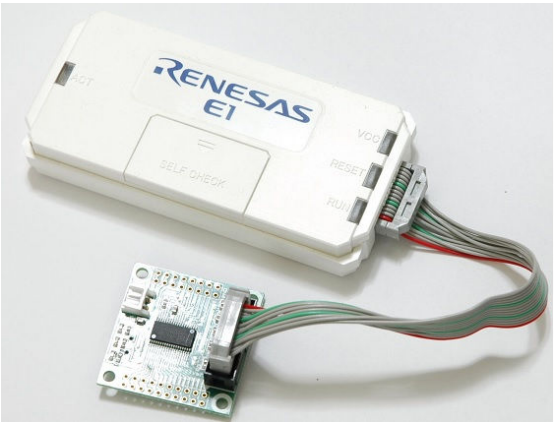
概要

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

- 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。

いずれかのＣＳ＋　ｆｏｒ　ＣＡ，ＣＸをダウンロードし、指示に従い展開して下さい。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。なお、ＣＳ＋は以前、ＣｕｂｅＳｕｉｔｅ＋という名称でしたが、２０１４年にＣＳ+となりました。大きな変更点は
１．ＲＬ７８用はＣＳ＋　ｆｏｒ　ＣＡ，ＣＸ、ＲＸ用はＣＳ＋　ｆｏｒ　ＣＣ　と別環境に分割されました。

２．設定等も変更されています。但し、上位互換性はあり、ＣｕｂｅＳｕｉｔｅ＋で作成されたソフトはＣＳ＋　ｆｏｒ　ＣＡ，ＣＸでコンパイル、実行可能です。

３．２０１５　３／２０に出たＣＳ＋　ｆｏｒ　ＣＣ　はＲＬ７８の開発が行えますが、１，２の上位互換性が無いので本セットには使用できません。ご注意願います。（２０１５．５．２１）

以下省略

ｆ：ＲＬ７８とＨ８／３００Ｈ、Ｒ８Ｃの速度比較

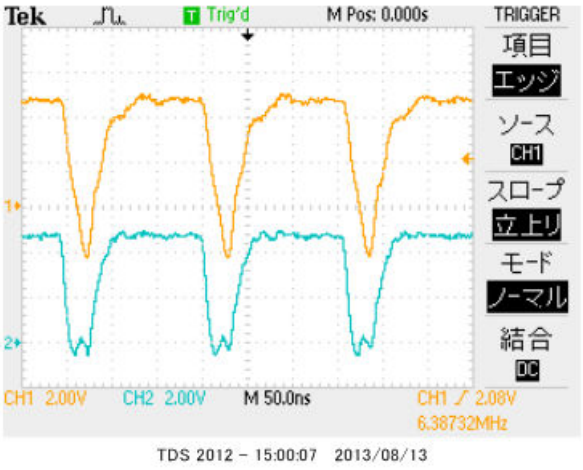
ＲＬ７８は、製造中止がアナウンスされているＨ８／３０４８の替わりに検討される方も多いと思われますが、実行速度はどうなのでしょうか？　開発環境を含めて以前より進化していなければ使う意味がないとお考えの方も多いかと思われます。

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。

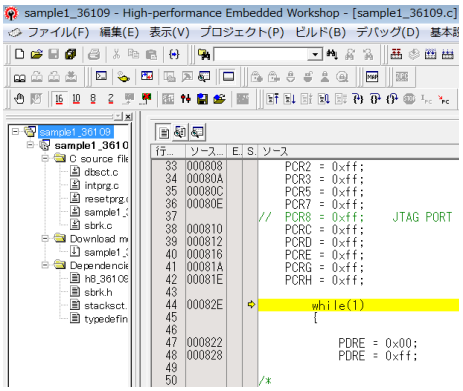
```
64                          void main(void)
65                          {
66      00187               R_MAIN_UserInit();
67                          /* Start user code. Do not edit comment generated here */
68                          while (1U)
69                          {
70      0018b       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

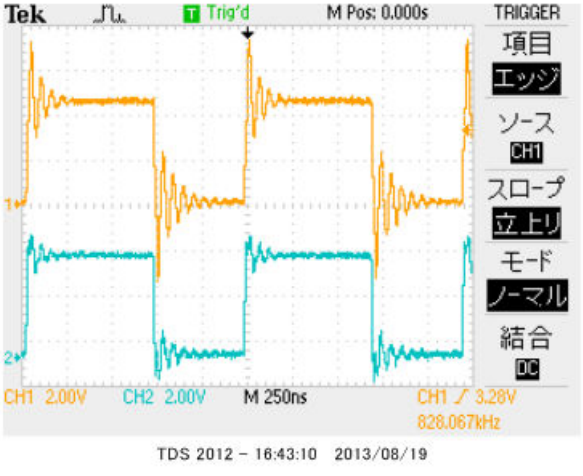
オシロスコープでＰ２０、Ｐ２１波形を観測すると６．３８７３２ＭＨｚという周波数でポートの１，０を繰り返すことが分かります。（クロック３２ＭＨｚ）



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ＢＣＨ８３６１４０９。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。





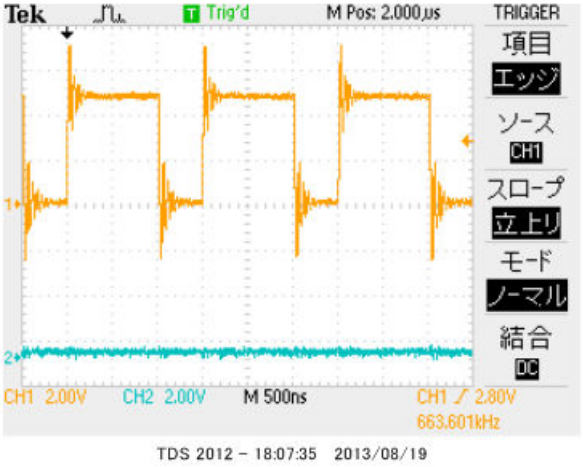
ポートＥを繰り返し、０、１しています。波形を観測すると８２８．０６７ＫＨｚとなりました。

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。

```
645 0EBE3
646
647 0EBE5
648 0EBE9
649 0EBE9
650 0EBED
651 0EBF1
```

```
asm("FCLR I");      //マスカブル割り込み禁止

    while(1)    //test
    {
        p1_0 = 0;
        p1_0 = 1;
    }
```



６６３．６０１ＫＨｚとなりました。

ｆ－２　乗除演算速度の比較

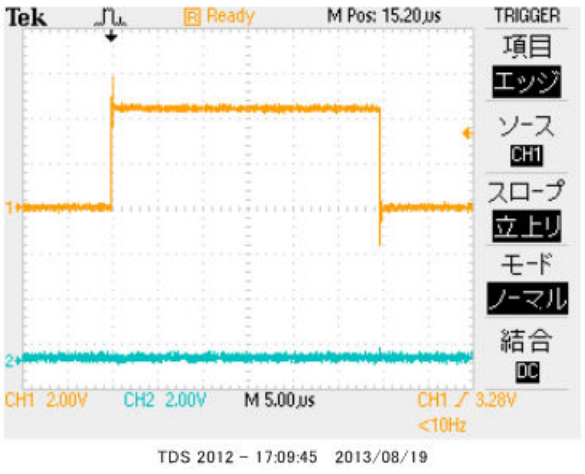
演算速度はどの程度違うでしょうか？　３２ｂｉｔの乗算、除算を行ってみました。
演算前にポートを立てて、演算後にポートを下ろすことにより、演算実行時間をオシロで観測しています。

Ｈ８－３６１０９　約３０μｓｅｃでした。

```
000874          while(1)
                {
00082E              Idata1 = 1234;
000838              Idata2 = 5678;

000840              PDRE = 0xff;
00084E              Idata3 = Idata1 * Idata2;
00085E              Idata4 = Idata2 / Idata1;
00086E              PDRE = 0;
                /*
                    PDR1 = 0;
                    PDR2 = 0;
                    PDR3 = 0;
                    PDR5 = 0;
                    PDR7 = 0;
                //  PDR8 = 0;
                    PDRC = 0;
                    PDRD = 0;
```

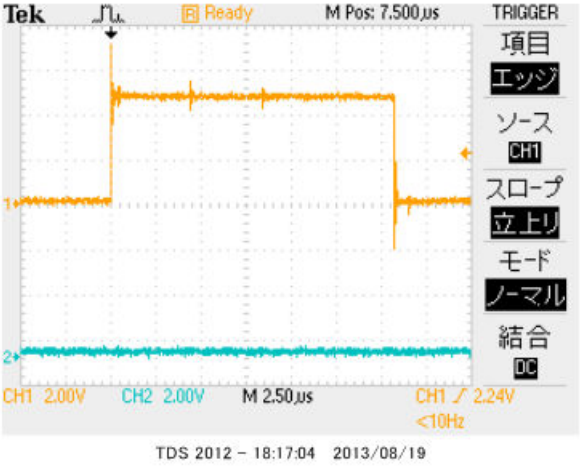
sample1

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           Idata1 = 1234;
0EBF3           Idata2 = 5678;

0EBFD           p1_0 = 1;
0EC01           Idata3 = Idata1 * Idata2;
0EC1F           Idata4 = Idata2 / Idata1;
0EC3D           p1_0 = 0;
0EC41       }
```



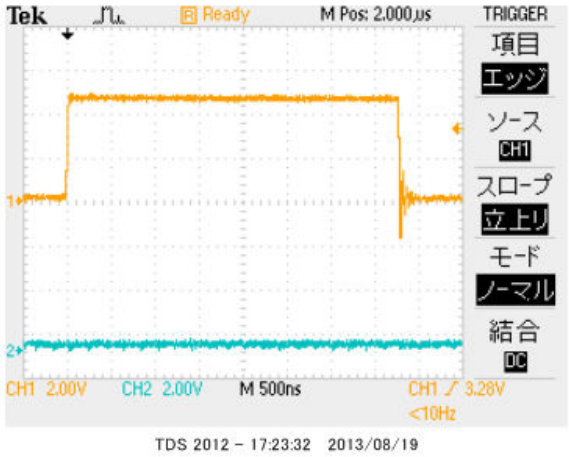
ＲＬ７８の場合　約３．８μｓｅｃでした。

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。

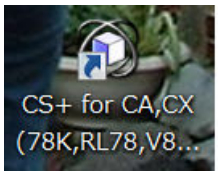
※測定結果はいずれも弊社製品比較です。

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、乗除・積和演算器、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。

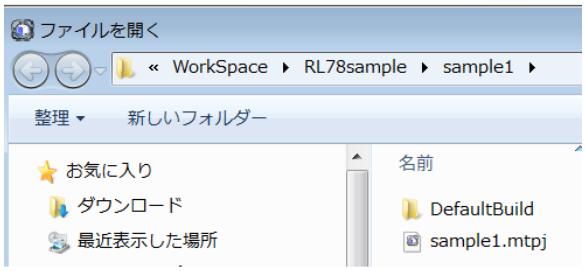
１－２　　動作、デバック

ａ：ＣＳ＋　ｆｏｒ　ＣＡ、ＣＸ起動、コンパイル、書き込み、動作

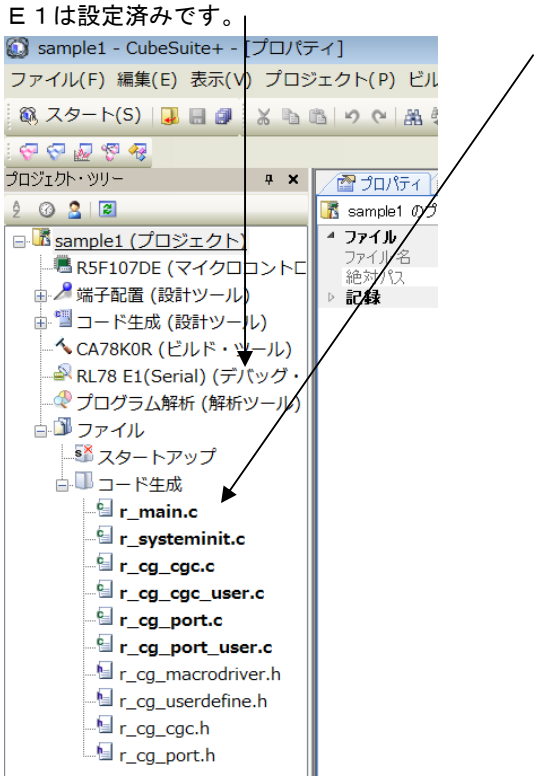


ＣＤに添付しているサンプルプログラムを使って、コンパイル、書き込み、動作の方法を示します。

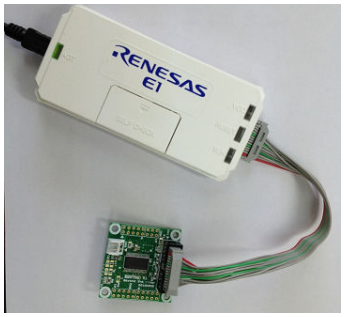
ＣｕｂｅＳｕｉｔｅ＋（以降ＣＳ+）を起動します。ここでは例としてＲＬ７８￥ｓａｍｐｌｅ１を動作させます。基板上のＬＥＤ　Ｄ１が点滅するプログラムです。
初めてのときは　ファイル　→　ファイルを開く　→　ｓａｍｐｌｅ１．ｍｔｐｊをダブルクリックします。



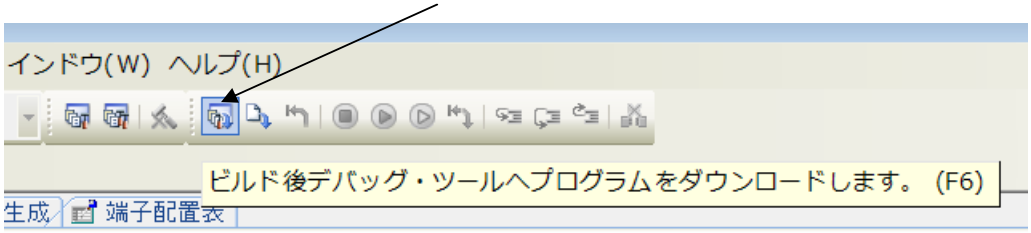
プロジェクトツリーが表示されます。ｒ__ｍａｉｎ．ｃをダブルクリック。
Ｅ１は設定済みです。



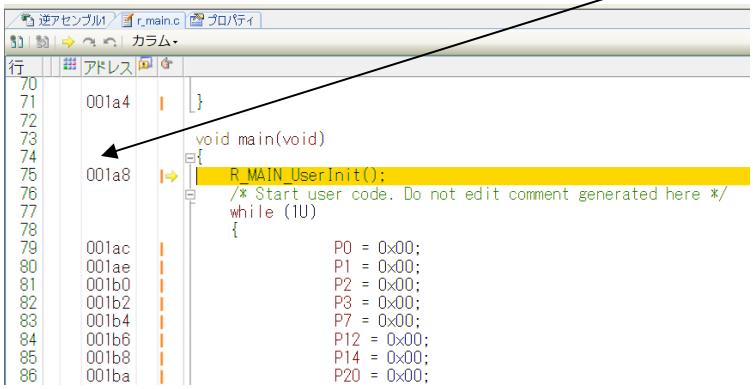
ｒ__ｍａｉｎ．ｃが中央に表示されます。とりあえず、実行してみます。Ｅ１のケーブルを基板のＣＮ１に挿入します。電源はＥ１から供給しますので、不要です。（写真ご参考）



「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。

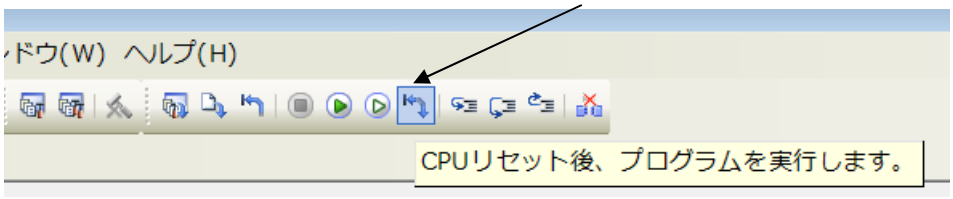


上手く転送できると、今まで表示されていなかったプログラムの絶対アドレス等が表示されます。Ｅ１から電源がＣＰＵ基板に供給されます。

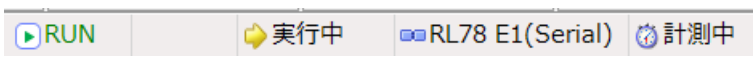


ここまでいかなかった場合、Ｅ１のインストゥールをご検証願います。

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。

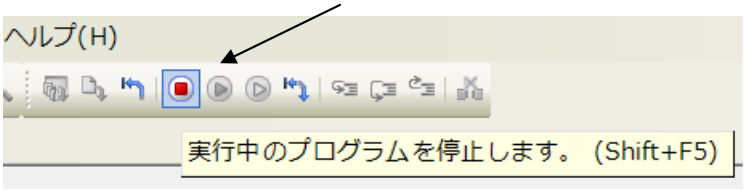


Ｅ１のＲＵＮ（緑ＬＥＤ）が点灯し、基板のＤ１が点滅したら動作しています。ＣＳ＋の右下部にも表示されます。



「ＣＰＵリセット後、プログラムを実行」をクリック。

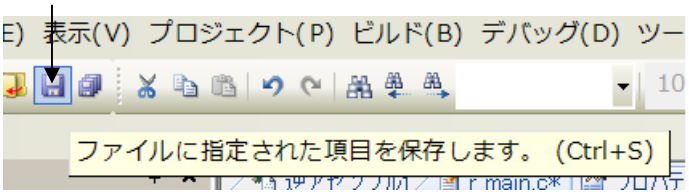
ここまで確認できましたら、一度止めます。



ｍａｉｎ関数のＩｗａｉｔ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。

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not e
    while (1U)
    {
                P0 = 0x00;
                P1 = 0x00;
                P2 = 0x00;
                P3 = 0x00;
                P7 = 0x00;
                P12 = 0x00;
                P14 = 0x00;
                P20 = 0x00;

                Iwait(1000000);

                P0 = 0xff;
                P1 = 0xff;
                P2 = 0xff;
                P3 = 0xff;
                P7 = 0xff;
                P12 = 0xff;
                P14 = 0xff;
                P20 = 0xff;

                Iwait(1000000);
    }
```

セーブして



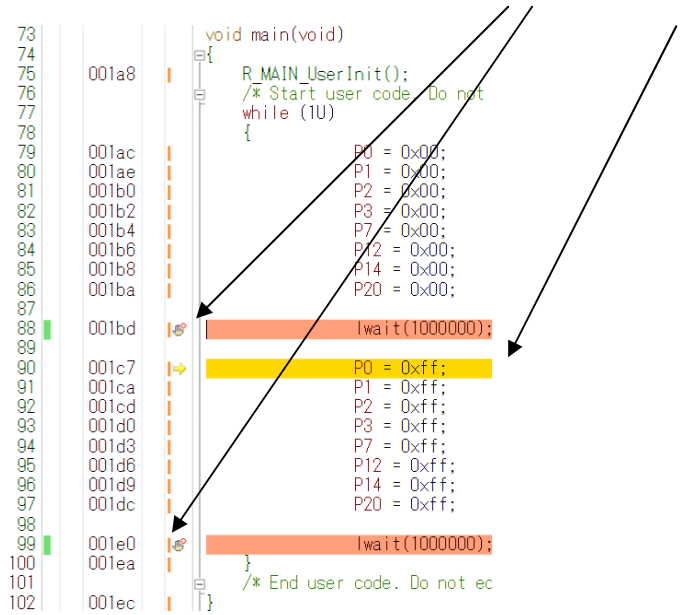
さきほどの、

「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。

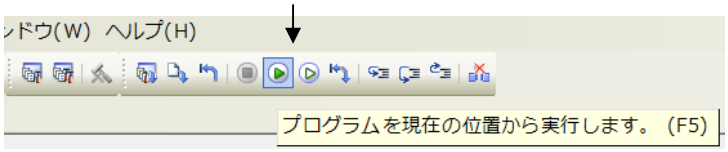
「ＣＰＵリセット後、プログラムを実行」をクリック。

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？

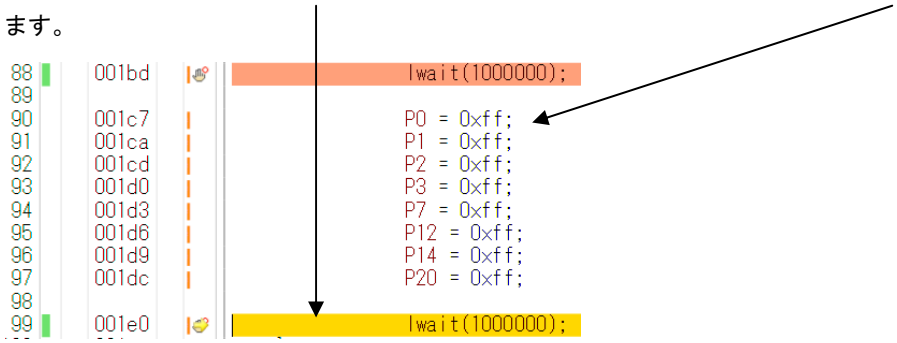
次に、ブレークポイントの設定を行ってみます。一度、プログラムを停止させます。
ブレークポイントを２点設定しました。手のマーク、黄色が現在のプログラムカウンタ位置。



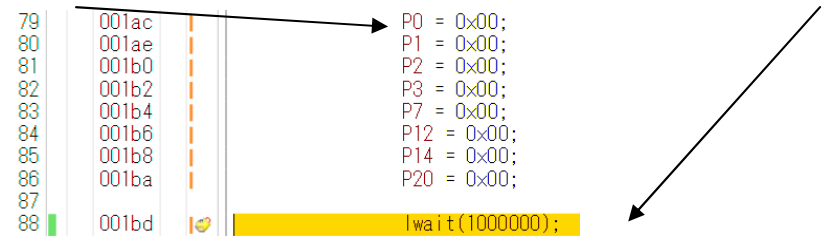
「プログラムを現在の位置から実行」します。



プログラムの実行はブレークポイントで停止し、ＬＥＤ　Ｄ１　は　Ｐ０＝０ｘｆｆ；命令により点灯します。



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、Ｐ０＝０ｘ００；命令実行により、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

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 ltime)
{
        while(ltime != 0)
        {
                ltime--;
        }
}

②void main(void)
{
  ③  R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    ④while (1U)
    {
        ⑤       P0 = 0x00;
                P1 = 0x00;
                P2 = 0x00;
                P3 = 0x00;
                P7 = 0x00;
                P12 = 0x00;
                P14 = 0x00;
                P20 = 0x00;

        ⑥       lwait(100000);

        ⑦       P0 = 0xff;
                P1 = 0xff;
                P2 = 0xff;
                P3 = 0xff;
                P7 = 0xff;
                P12 = 0xff;
                P14 = 0xff;
                P20 = 0xff;
```

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

【　解説　】

①void lwait(unsigned long ltime)

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。

④while (1U)

{

以下を無限ループします。

⑤　　P0 = 0x00;

Ｐ０に０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。

Ｐ０．５に接続されているＬＥＤ１は消灯します。

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　　P0 = 0xff;

Ｐ０に０ｘｆｆを設定しています。Ｐ０．５に接続されているＬＥＤ１は点灯します。

⑧　　lwait(100000);

点灯も消灯と同じ時間、保持されます。

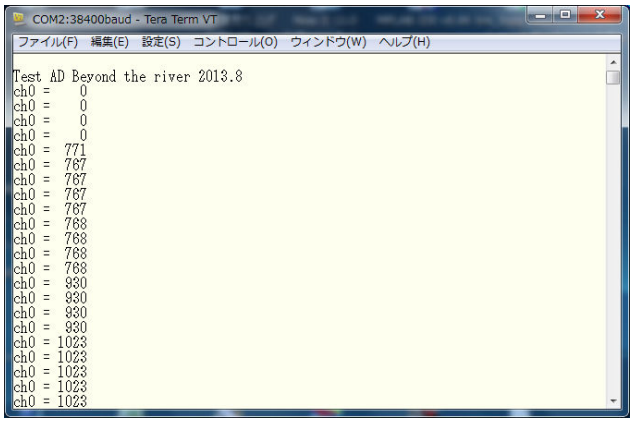
２－２　ｓａｍｐｌｅ２　ＳＩＯ（ＵＳＢ)、ＥＥＰＲＯＭ読み書き

以下省略

２－３　ｓａｍｐｌｅ３　Ａ／Ｄ変換をＵＳＢ出力

【　動作概要　】

ＡＮＩ０（Ｐ２０）　ＣＮ４　１４番　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、ＡＮＩ０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。

【　プログラム　】

```
volatile uint16_t ad_data,eep_data;
unsigned char ad_buffer[20];
volatile uint16_t enc_p,enc_m;

void main(void)
{

    R_MAIN_UserInit();

    R_UART1_Start();                                    //UART動作開始

//UART(USB)初期化
        R_UART1_Receive(rx_data, 1);                    //1文字受信→絶対必要！
        rx_flg = 0;                                     //受信フラグクリア
        tx_end_flg = 0;                                 //送信終了フラグクリア

        R_UART1_Send(String_0,35);                      //Opening message
        tx_end_wait();                                  //送信終了まち

        wait(1000000);                                  //オープニングメッセージ表示時間

//      ADPC = 0x07;                                    //AD0-5までＡＤ端子　不用なようです
①      ADCE = 1;                                       //A/D電圧コンパレータ動作許可
```

```
    /* Start user code. Do not edit comment generated here */
    while (1U)
    {
②                   R_ADC_Start();                  //AD変換開始
                    while(ADCS)                     //変換待ち
                    R_ADC_Get_Result(&ad_data);     //AD読み込み  ad_dataに&（アンバサンド）を付ける

③                   sprintf(ad_buffer, "ch0 = %4d¥n¥r",ad_data); //ad_buffに10進ASCII変換してセーブ

④                   R_UART1_Send(ad_buffer,sizeof(ad_buffer));          //uart出力
                    tx_end_wait();                                      //送信終了待ち

⑤                   wait(1000000);

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

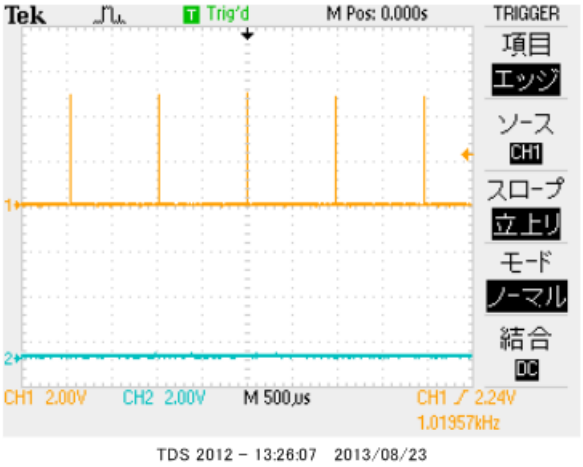
【　解説　】

以下省略

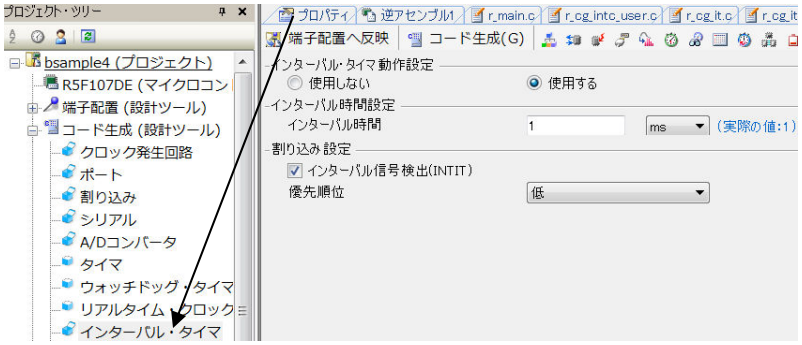
２－４　ｓａｍｐｌｅ４　割り込み

【　動作概要　】

ｓａｍｐｌｅ４を動作させます。オシロスコープがあればＰ０５　ＣＮ４　６番を観測すると、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。

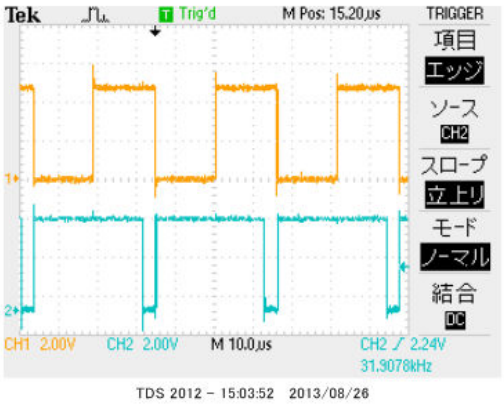
以下省略

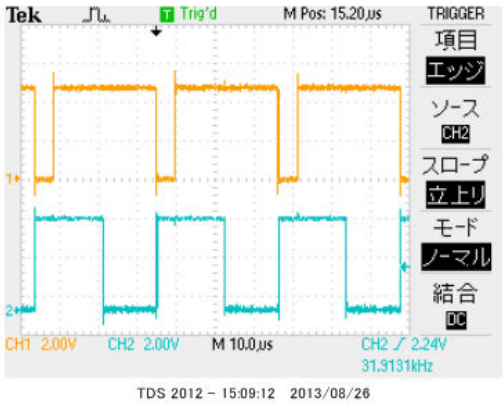
２－５　ＰＷＭ出力

【　動作　】

ＲＬ７８の１６ビットタイマＫＢ０，ＫＢ１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Ｐ２００（ＴＫＢＯ００）、Ｐ２０１（ＴＫＢＯ０１）、Ｐ２０２（ＴＫＯＢＯ１０）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、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。





【　プログラム　】

以下省略

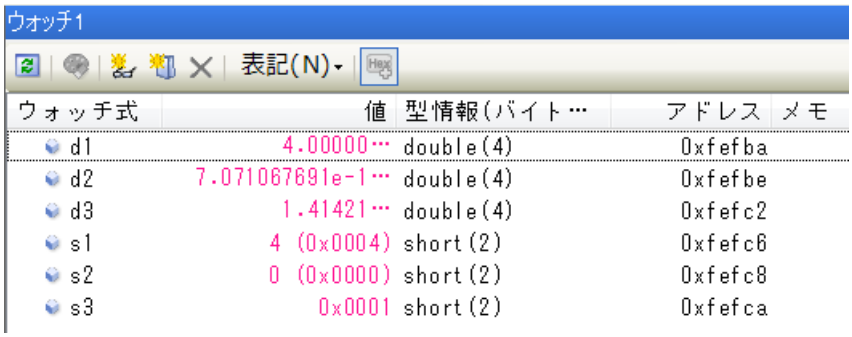
２－６　三角、対数、平方根関数を使う

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

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 */

⑤       P0 = 0x20;                                      //時間測定マーカーＯＮ
⑥       d1 = log10(10000);
⑦       P0 = 0x00;                                      //時間測定マーカーＯＦＦ

        P0 = 0x20;                                      //時間測定マーカーＯＮ
⑧       d2 = sin((PI/180)*45);
        P0 = 0x00;                                      //時間測定マーカーＯＦＦ

        P0 = 0x20;                                      //時間測定マーカーＯＮ
⑨       d3 = sqrt(2);
        P0 = 0x00;                                      //時間測定マーカーＯＦＦ

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


    while (1U)
                ;

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

【　解説　】

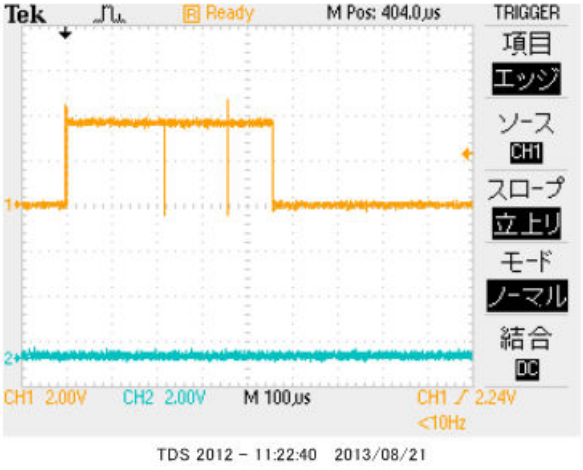
省略

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程度かかるようでした。



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　