ＲＬ７８　マイコン学習セット　マニュアル　入門編

第４版　２０２１．１０．１１　詳細下記　　第１版 2015.4.16

【　製品概要　】

本マニュアルはＲＬ７８／Ｉ１Ａ　Ｒ５Ｆ１０７ＤＥ（３８ピン）マイコンを使ったマイコン学習セットの開発環境構築、ソフトウエアインストゥール手順、添付ＣＤのサンプルプログラムの動作について解説されています。

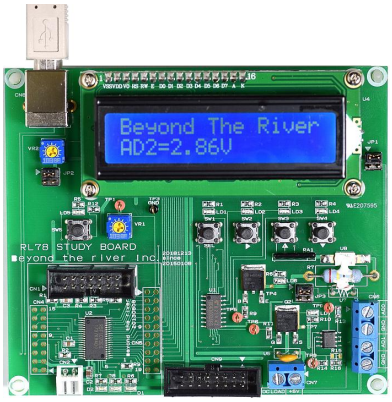
入門編ではマイコンの基本的なハードウエアのアクセス方法、プログラムの書き方をサンプルプログラムを参考に学び、習熟度をチェックするために、演習プログラムの課題を自分で考えます。

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

※本学習セット開発にはルネサスエレクトロニクス社製Ｅ２　Ｌｉｔｅが必要です。

【　第４版での変更点　】

１．デバックツールＥ１の製造中止に伴いましてＥ２　Ｌｉｔｅ（以降　Ｅ２Ｌ）でのデバックに変更しました。



１．学習環境、事前準備

１－１．　学習環境

ａ：学習セット　同梱物

ｂ：ＢＣＲＬ７８１０７　ＣＰＵ部の特徴

ｃ：Ｅ２　Ｌｉｔｅ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　ＣＳ+　操作

ｂ－１：Ａ／Ｄ、リセット、ウオッチドッグ設計上の注意点

ｂ－２：自動生成されたプログラム

ｂ－３：Ｅ２Ｌ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．キー入力　ｓａｍｐｌｅ１

プログラム　　：点滅する LED をキー入力で消灯

演習プログラム　：押されたキーの LED を点滅

２－２．ＵＳＢ通信 ｓａｍｐｌｅ２

プログラム　　：ＡＢＣＤをパソコン側に送信

演習プログラム　：キー入力でＡＢＣＤと送信

２－３．Ａ／Ｄ変換 ｓａｍｐｌｅ３

プログラム　　：Ａ／Ｄ変換データをパソコン側に送信

演習プログラム　：Ａ／Ｄ値を０－５Ｖに換算しパソコン側に送信。

２－４．ＰＷＭ　ｓａｍｐｌｅ４

プログラム　　：ＬＥＤ輝度連続可変

演習プログラム　：ＬＥＤ輝度階段状可変

２－５．割り込み　ｓａｍｐｌｅ５

プログラム　　：割り込みでＬＤ１点灯

演習プログラム　：メインでＬＤ１点滅、割り込みでＬＤ２点灯、消灯

１−１．学習環境

ａ：学習セット同梱物

ＲＬ７８学習ボード	１
ＣＤ（サンプルプログラム、デバイスドライバ、ドキュメント）	１
マニュアル（本誌） 入門、実用	各１
電源ケーブル、ＵＳＢケーブル	各１
モーター	１
サーミスタ	１

※開発に必要なルネサスエレクトロニクス社製デバッカＥ２Ｌは同封されておりません。別途必要です。但し、プログラムの検討、コンパイルは、無料のＣＳ＋（後述）で行うことが出来ます。

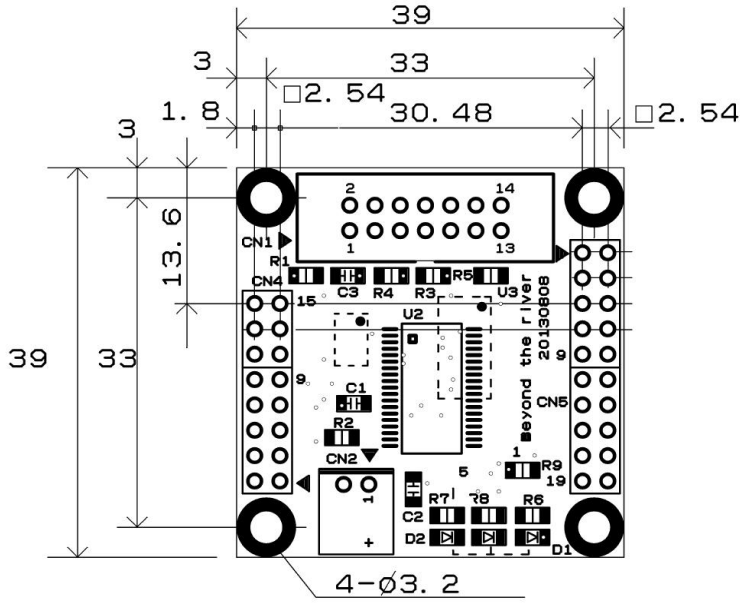
複数の人間の学習において
Ａ．Ｅ２Ｌ＋本ボード＋ＣＳ＋インストゥール済みパソコンを用意
Ｂ．プログラムの検討、コンパイルは他のパソコンで行い、実行だけＡのパソコンに席を移る
といった使い方で、人数分用意しなくても効率よく学習することは可能だと思います。もちろん、各人に各台数あるのが、時間的な効率は一番良いです。

ｂ：ＢＣＲＬ７８１０７　ＣＰＵボード部の特徴
学習ボードのマイコン部分は弊社ＢＣＲＬ７８１０７ＣＰＵボードと同じです。
●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
●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　６

ｃｈ、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　ドライバＩＣ　ＦＴＤＩ社　ＦＴ２３２ＲＬ搭載。※２
●デバッカＥ２Ｌ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。

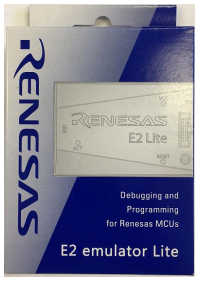
※１　速度比較は本マニュアル　１－１　ｆ：ＲＬ７８とＨ８／３００Ｈ、Ｒ８Ｃの速度比較をご参照下さい。
※２　学習ボードはＣＰＵ＋デバック用コネクタ、ＵＳＢインターフェイス＋ＥＥＰＲＯＭ搭載のＢＣＲＬ７８１０７Ｍが使用されています。

ＣＰＵ部大きさ（部品面）



ＵＳＢミニＢコネクタ、ＦＴ２３２ＲＬ、２５ＬＣ２５６は裏面搭載。

ｃ：Ｅ２　Ｌｉｔｅ　エミュレータ



概要

Ｅ２Ｌ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。変数をウオッチ窓に登録し、実行中を含めて数値を見ながらデバック出来ます。

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。

対応MPU　RL78,RX,RA,RE

Ｅ２ＬをＰＣのＵＳＢに接続するとｗｉｎｄｏｗｓが自動的にデバイスドライバをインスツールします。続いて、ネットから開発環境ＣｕｂｅＳｕｉｔｅ＋とＣコンパイラの最新版ダウンロードを行います。

ｄ：無償のＣＳ+、ＲＬ７８用Ｃコンパイラのダウンロード

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

　ネット検索で→「ＣＳ＋　無償ダウンロード」の検索で表示されます。

統合開発環境 CS+ (旧CubeSuite+)

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

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

ダウンロード出来ましたら、指示に従い展開して下さい。

ｅ：開発セット添付ＣＤコピー、デバイスドライバＤ２ＸＸのインストゥール

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

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

正常にインストゥールされると、以下のように２つのデバイスが確認出来ます。

ｆ：ＲＬ７８とＨ８／３００Ｈ、Ｒ８Ｃの速度比較

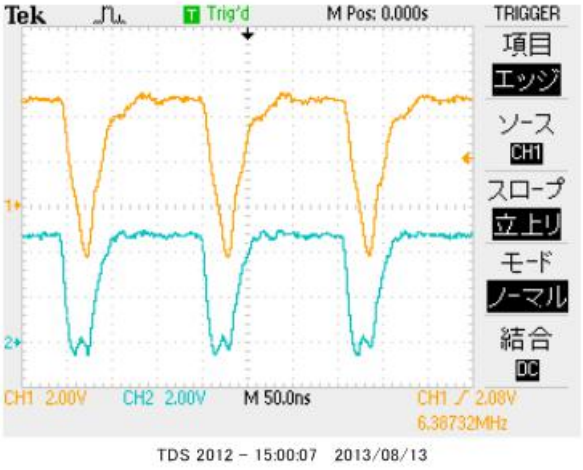
ＲＬ７８は、製造中止がアナウンスされているＨ８／３０４８の替わりに検討される方も多いと思われますが、実行速度はどうなのでしょうか？　開発環境を含めて以前より進化していなければ使う意味がないとお考えの方も多いかと思われます。

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）

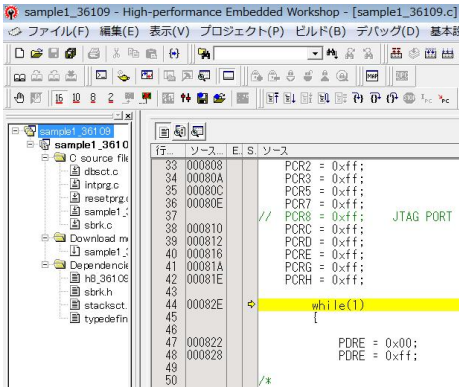


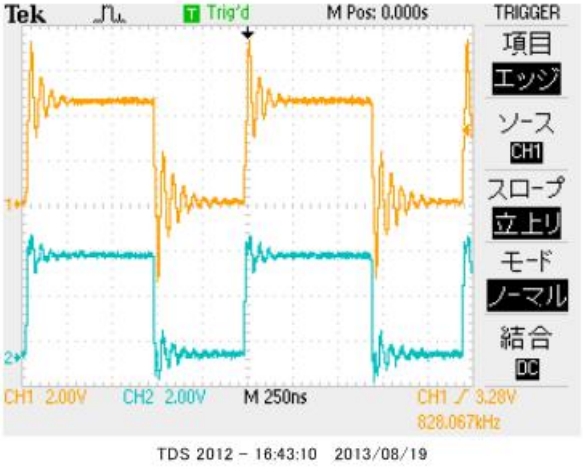
この命令の詳細は

```
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。

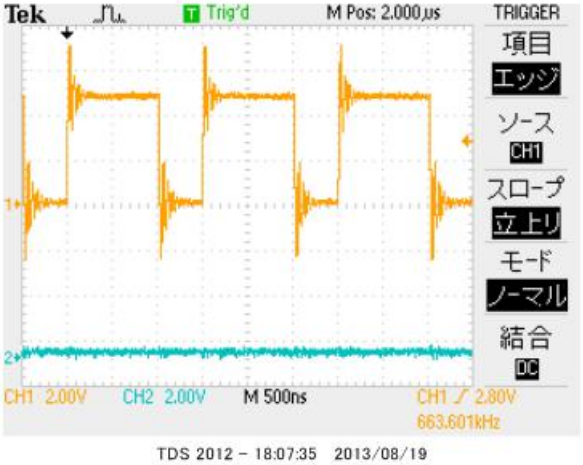
６．３８７３２ＭＨｚ÷８２８．０６７ＫＨｚ≒７．７倍高速という驚きの結果になりました。（クロック２０ＭＨｚ）クロックを同じにしても、４．８倍違います。

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。

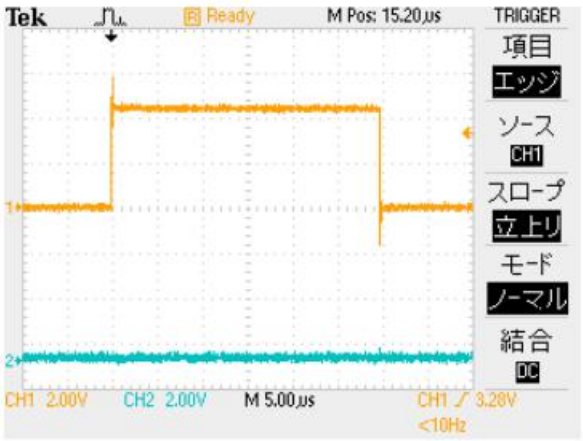
Ｈ８－３６１０９　約３０μｓｅｃでした。

```
000874          while(1)
                {
00082E              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

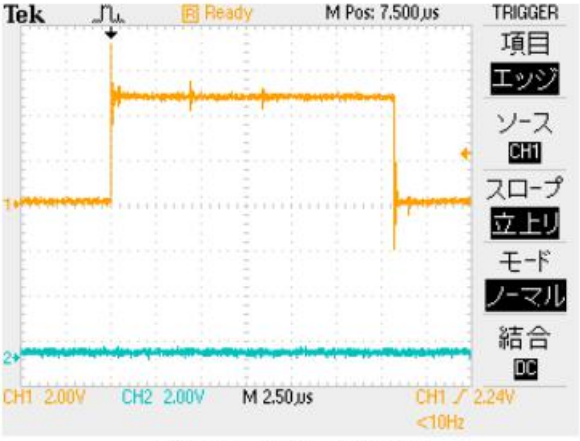


Ｒ８Ｃ／Ｍ１２Ａの場合　約１５．５μｓｅｃでした。

```
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       }
```



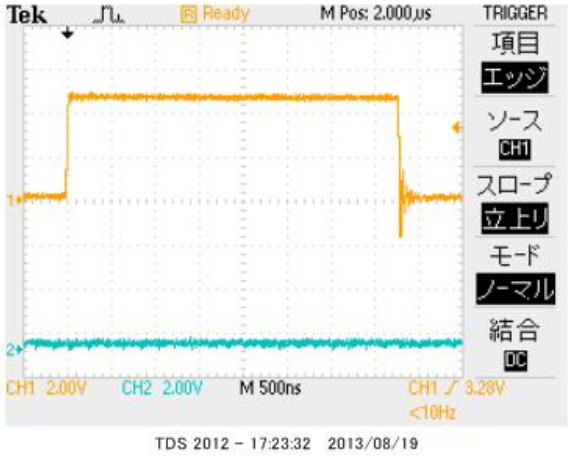
ＲＬ７８の場合　約３．８μｓｅｃでした。

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、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。

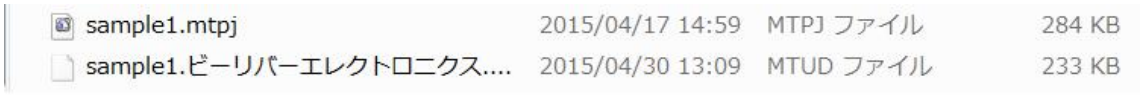
結論として、従来、Ｈ８／３０４８等をご使用の方々にも安心して使っていただける性能をもったＣＰＵだと思います。

１－２　　動作、デバック

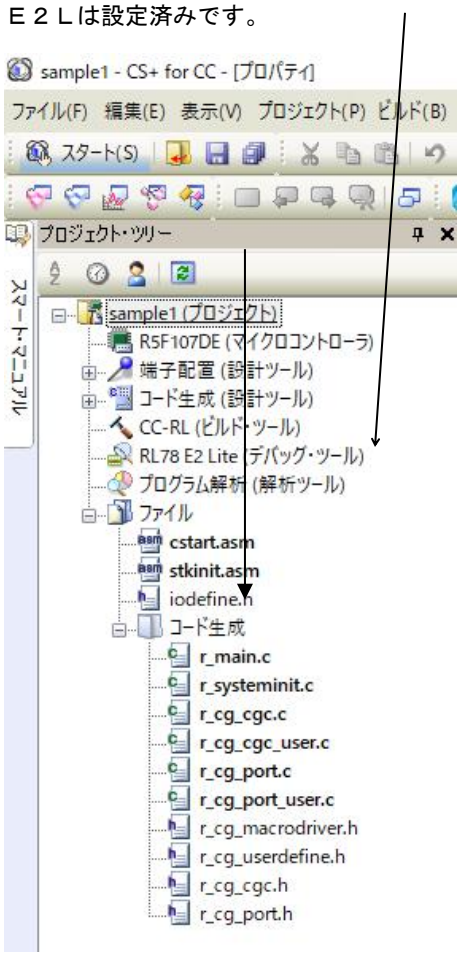
ａ：ＣＳ＋起動、コンパイル、書き込み、動作

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＋を起動します。ここでは例としてＲＬ７８ＳＴＵＤＹ￥ｓａｍｐｌｅ１を動作させます。キーを押すと上のＬＥＤ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。
Ｅ２Ｌ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。Ｅ２Ｌのケーブルを基板のＣＮ

１に挿入します。電源はＥ２Ｌから供給しますので、ＵＳＢ接続は不要です。（写真ご参考）

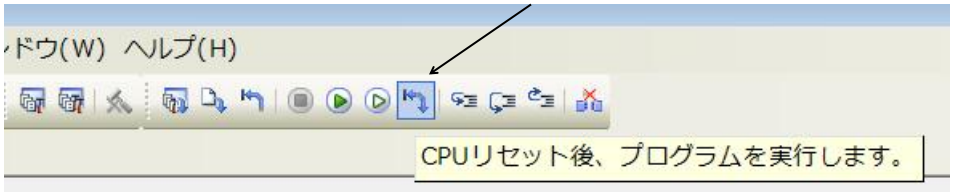
「デバック・ツールへプログラムを転送」をクリック。

　正常に転送できると、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。Ｅ２Ｌから電源３．３ＶがＣＰＵ基板に供給されます。

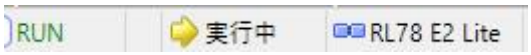
```
78                  /*****************************************************************************
79                   * Function Name: main
80                   * Description  : This function implements main function.
81                   * Arguments    : None
82                   * Return Value : None
83                   *****************************************************************************/
84                  void main(void)
85                  {
86                      R_MAIN_UserInit();
87                      /* Start user code. Do not edit comment generated here */
88                      while (1U)
89                      {
90      0018c                   P2 = 0x00;                      //LD1,2,3,4 OFF
91      0018e                   wait(1000000);                  //時間待ち
92      00197                   P2 = 0xf0;                      //LD1,2,3,4 ON
93      0019a                   wait(1000000);                  //時間待ち
94
95      001a3                   while(P12.1 == 0)               //SW1が押されているとループ
96                              {
97      001a8                           P2 = 0x00;              //LD1,2,3,4 OFF
98                              }
99                      }
100
101                     /* End user code. Do not edit comment generated here */
102                 }
```

正常にダウンロード出来ると命令のアドレス等が表示されます。ここまでいかなかった場合、Ｅ２Ｌ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。

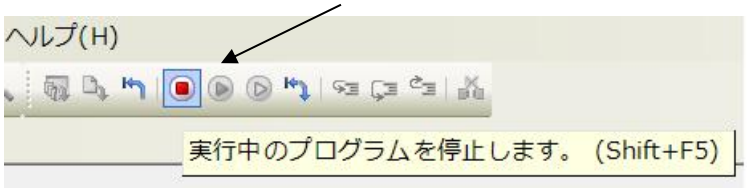


４つのＬＥＤ　Ｄ１，２，３，４が点滅したら正常に動作しています。ＣＳ＋の右下部「ＲＵＮ　実行中」が表示されます。



ここで　キー　ＳＷ１が押されている間、ＰＯＲＴ２には０が出力され、ＬＥＤは消灯するはずです。そうなりますでしょうか？

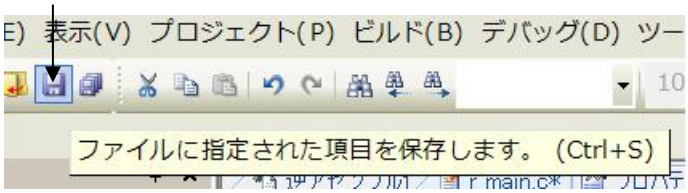
ここまで確認できましたら、一度止めます。



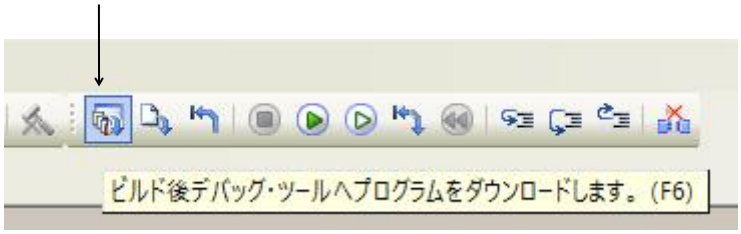
ｍａｉｎ関数のｗａｉｔの数値に２か所、０を減らして

```
84                  void main(void)
85                  {
86                      R_MAIN_UserInit();
87                      /* Start user code. Do not edit comment generated here */
88                      while (1U)
89                      {
90      0018c                   P2 = 0x00;                  //LD1,2,3,4 OFF
91      0018e                   wait(100000);               //時間待ち
92      00195                   P2 = 0xf0;                  //LD1,2,3,4 ON
93      00198                   wait(100000);               //時間待ち
94
95      0019f                   while(P12.1 == 0)           //SW1が押されているとループ
96                              {
97      001a4                           P2 = 0x00;          //LD1,2,3,4 OFF
98                              }
99                      }
100
101                     /* End user code. Do not edit comment generated here */
102             }
```

セーブして



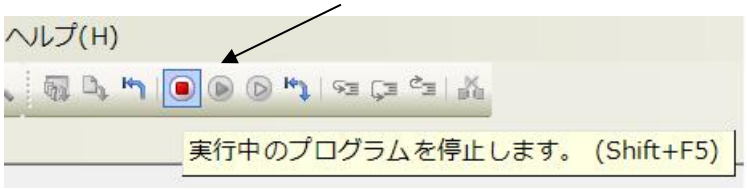
プログラムを変更しましたから「ビルド後、デバック・ツールへプログラムを転送」をクリック。



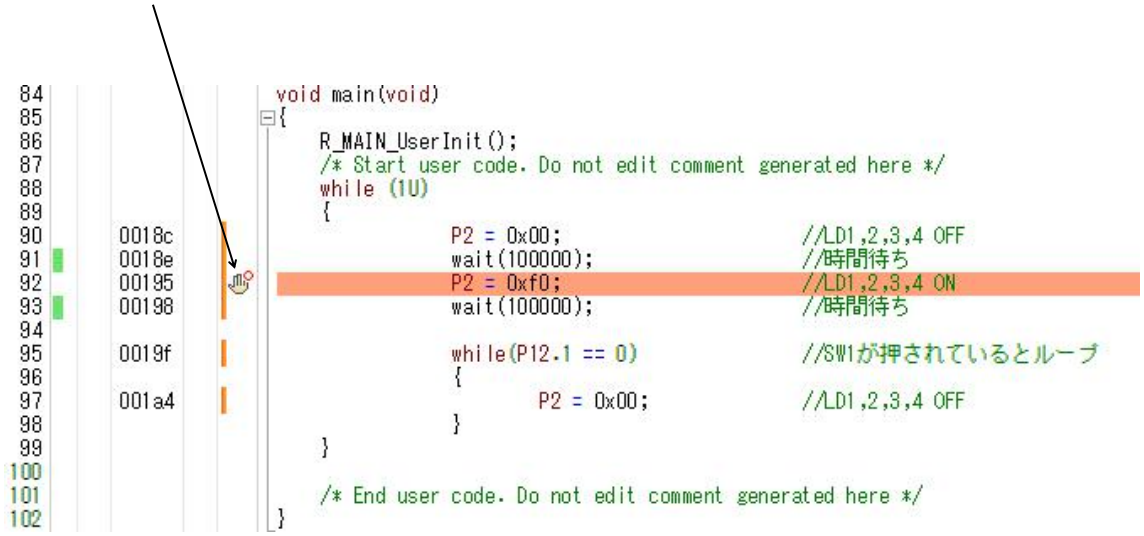
「ＣＰＵリセット後、プログラムを実行」をクリック。
ＬＥＤの点灯が先ほどより、早く点滅するのが目視できましたでしょうか？

Ｐ１２のデータを読み込んだり、その値を３ビット左にシフトしたり、反転させたりしているので、値が変動します。

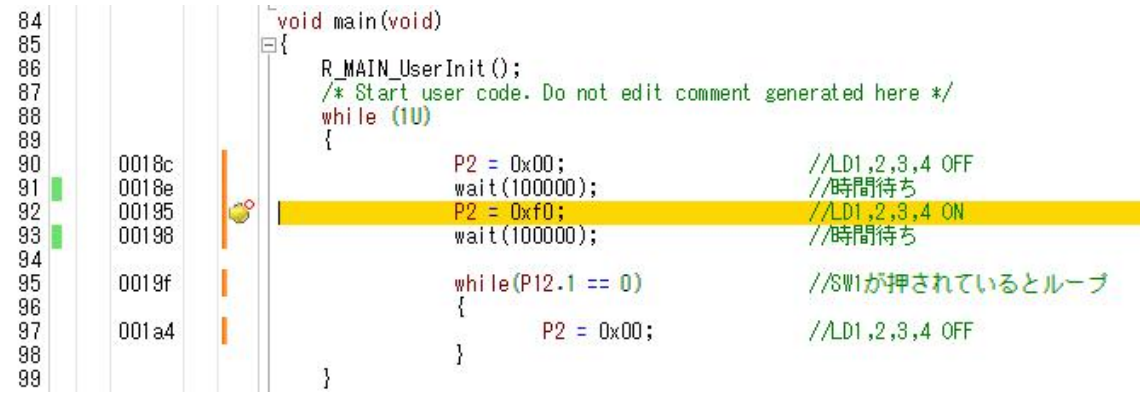
次に、ブレークポイントの設定を行ってみます。一度、プログラムを停止させます。



マウスを 00195 番地のここにもって行き、左クリックで設定です。手の形が出ます。解除は設定後、再クリック。



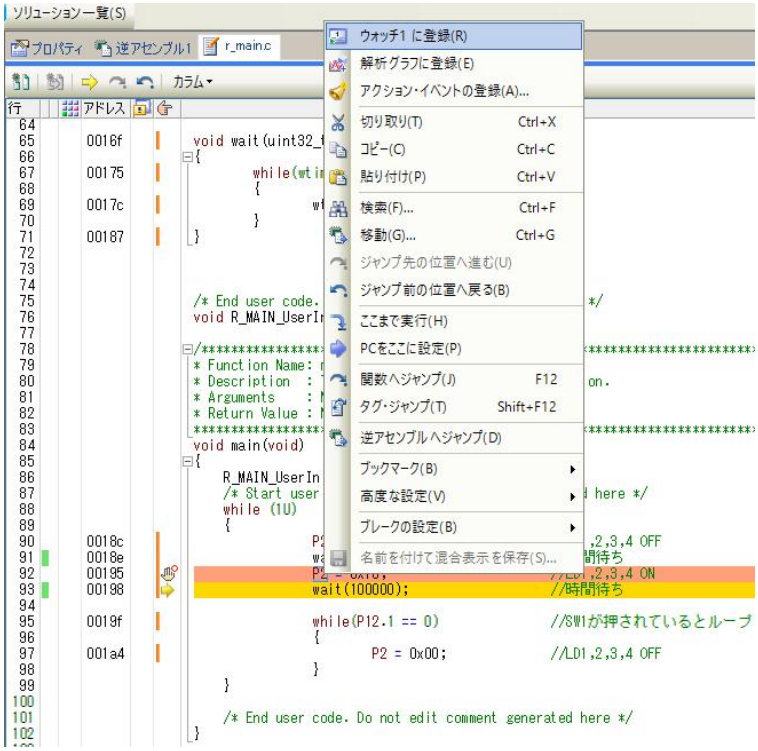
「ＣＰＵリセット後、プログラムを実行」をクリック。先ほど設定した行でプログラムカウンタが停止します。黄色は現在のプログラムカウンタの位置です。まだコードは実行されていません。



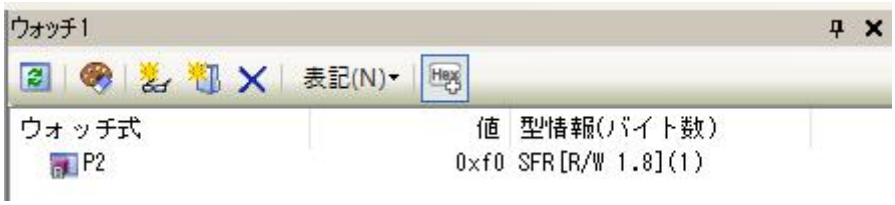
ステップオーバーで１行実行。

```
84                  void main(void)
85                  {
86                      R_MAIN_UserInit();
87                      /* Start user code. Do not edit comment generated here */
88                      while (1U)
89                      {
90      0018c                   P2 = 0x00;                  //LD1,2,3,4 OFF
91      0018e                   wait(100000);               //時間待ち
92      00195                   P2 = 0xf0;                  //LD1,2,3,4 ON
93      00198                   wait(100000);               //時間待ち
94
95      0019f                   while(P12.1 == 0)           //SW1が押されているとループ
96                              {
97      001a4                       P2 = 0x00;              //LD1,2,3,4 OFF
98                              }
99                      }
100
101                     /* End user code. Do not edit comment generated here */
102             }
```

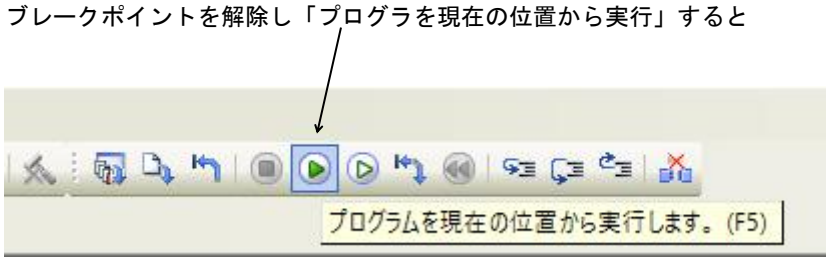
プログラムカウンターは 00198 で、１行前のＰ１２　＝　０ｘｆ０　が実行されました。ＬＤ１，２，３，４の点灯が確認できると思います。



Ｐ１２　＝　０ｘｆ０；
のＰ１２の部分をマウスの左ボタンを押したままでドラッグ、右クリックで「ウオッチ１に登録」で現在のＰ１２の内容が表示されます。



ブレークポイントを解除し「プログラを現在の位置から実行」すると



　ウオッチ１窓にあるＰ２の値がプログラムの進行に伴い、０ｘ００、０ｘｆ０と繰り返すのが確認できると思います。

以上、大急ぎでしたが、プログラムの書き換え、コンパイル、Ｅ２Ｌへのダウンロード、実行、ブレークポイント設定、ウオッチ窓設定、動作の概要です。

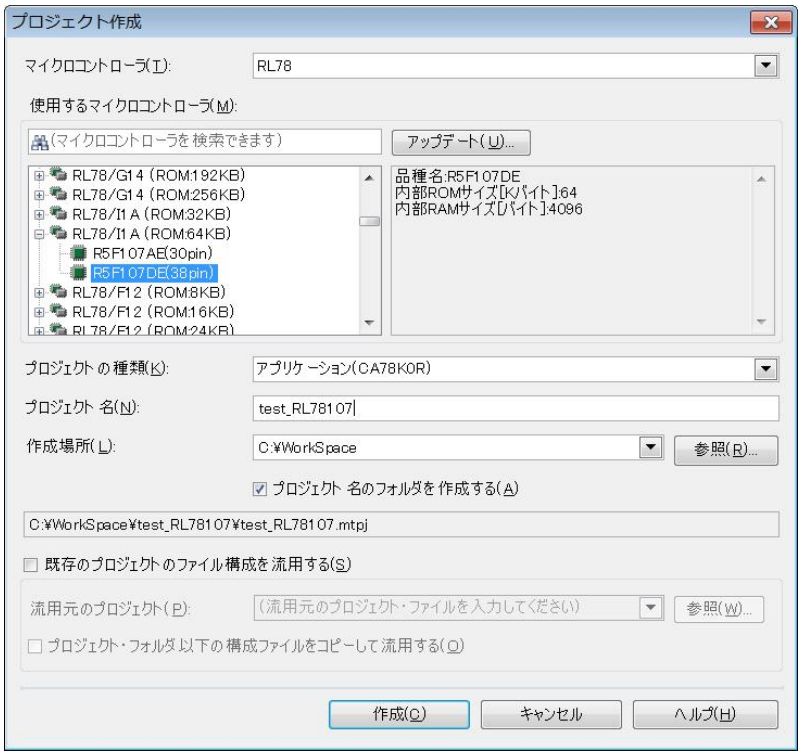
ｂ：新しいプログラムを作る

　ＣＳ＋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生成されることです。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、バグの発生を抑え、品質が底上げされます。

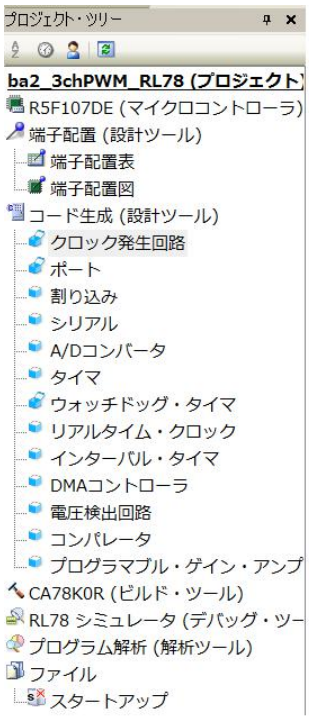
開発の詳細は順を追って説明します。

ＣＳ＋を開くと、以下のような画面が表示されます。新しいプロジェクト作成はファイル→新規作成→新しいプロジェクトを作成　を実行します。

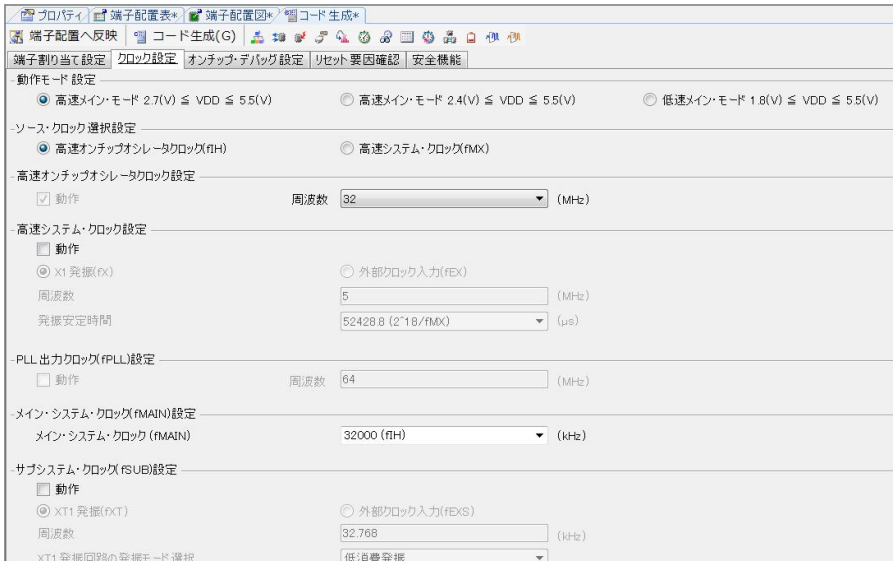
使用するマイクロコントローラはＲ５Ｆ１０７ＤＥ（３８ｐｉｎ）を設定します。プロジェクト名を設定して、「作成」をクリックします。



「ツール」→「プラグインの管理」→「追加機能」で「コード生成プラグイン」「端子配置プラグイン」にチェックが入っていることを確認、無い場合入れます。
次にコード生成（設計ツール）→クロック発生回路をクリックします。
クロック発生回路より順次設定していきます。



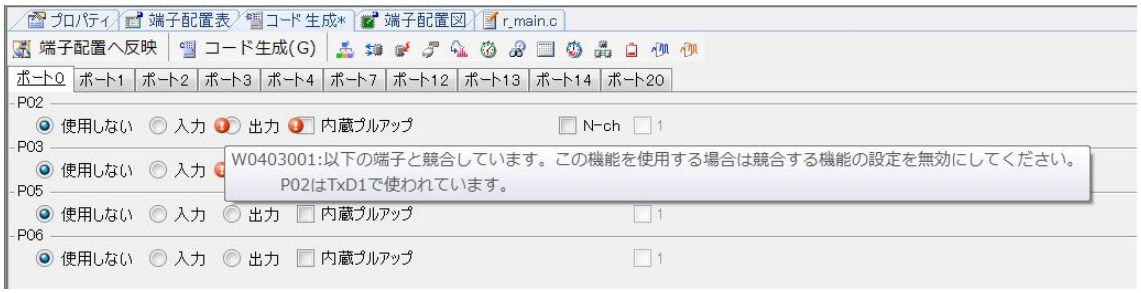
ここでは高速メインモード２．７Ｖ～５．５Ｖを選択し、内蔵３２ＭＨｚクロックを選択します。



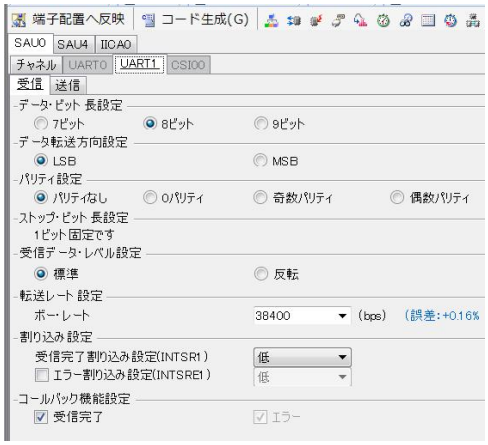
オンチップ・デバック設定は使用するを選択

他はＲＬ７８学習ボードでは特に変更する必要はありません。

次にポートの設定を行います。端子を入力で使うのか、出力で使うのか等を設定します。ポート以外で使用する場合、使用しないを選択します。例Ｐ０３→ＲＸＤ１で使用するので、ポートとしては使用しない。後に説明するシリアルでアクティブにしているので、警告が表示されます。

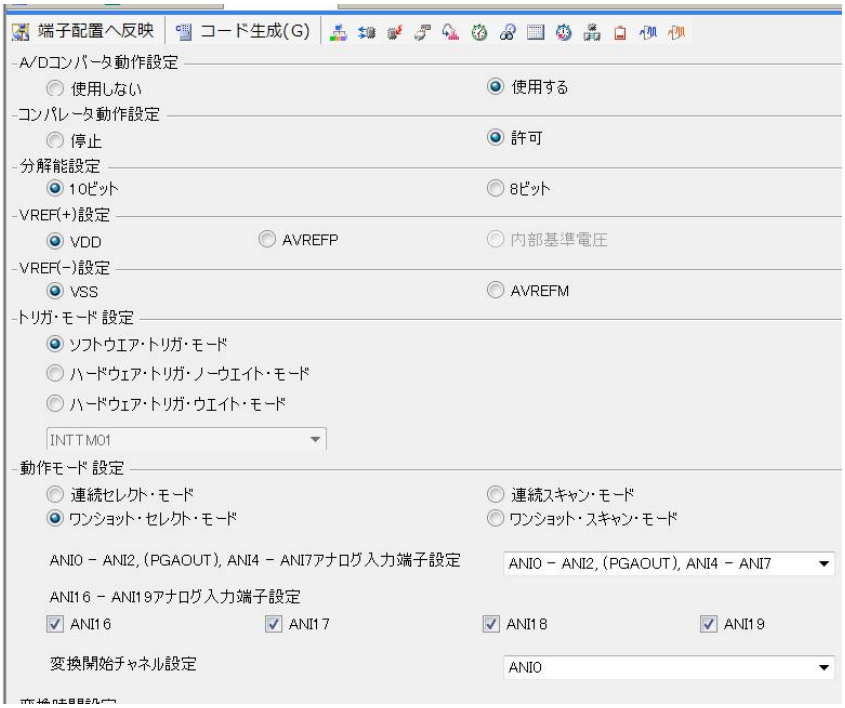


次にシリアルを設定します。Ｐ０３，Ｐ０２のＲＸＤ１、ＴＸＤ１を使用しますから、ＵＡＲＴ１、送信／受信機能を選択します。データビット長、ボーレート等それぞれ送受信で設定がありますので、注意願います。



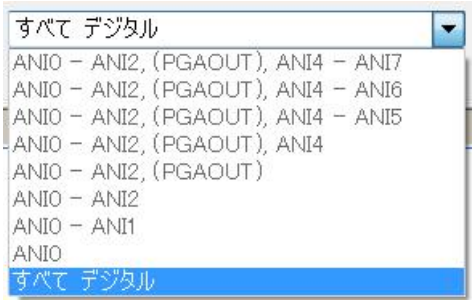
以下は上記設定し「コード生成」後、Ｐ０２、Ｐ０３を入力とか出力に設定しようとした場合に出される警告。設定できません。

次にＡ/Ｄコンバータを設定します。今回は基準電圧（+）＝電源電圧、基準電圧（－）＝ＧＮＤ電位、ソフトウエアトリガモードです。コンパレーター動作「許可する」。



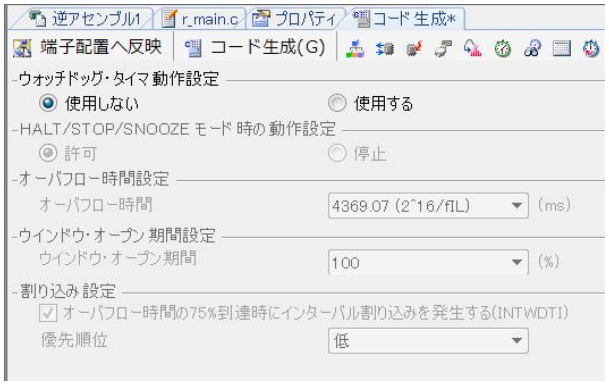
ｂ－１：Ａ／Ｄ設計上の注意点

１点注意しなくてはならないのが、例えばＡＮＩ５を使いたい場合、ＡＮＩ０～ＡＮＩ２、ＡＮＩ４もＡ／Ｄコンバータ入力になってしまうので、ハード設計時に注意が必要です。
（選択できるＡ／Ｄ入力）



特に割り込みは使用しません。

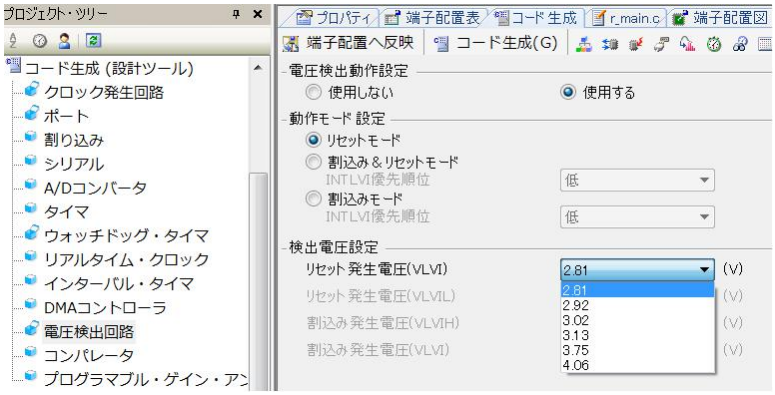
タイマ、ウオッチドグタイマ、リアルタイムクロックは特に使用しません。ウオッチドグタイマはデホルトで「使用する」がチェックされていますので、「使用しない」をチェックします。**これを忘れると定期的にリセットが発生しますので、注意して下さい。**



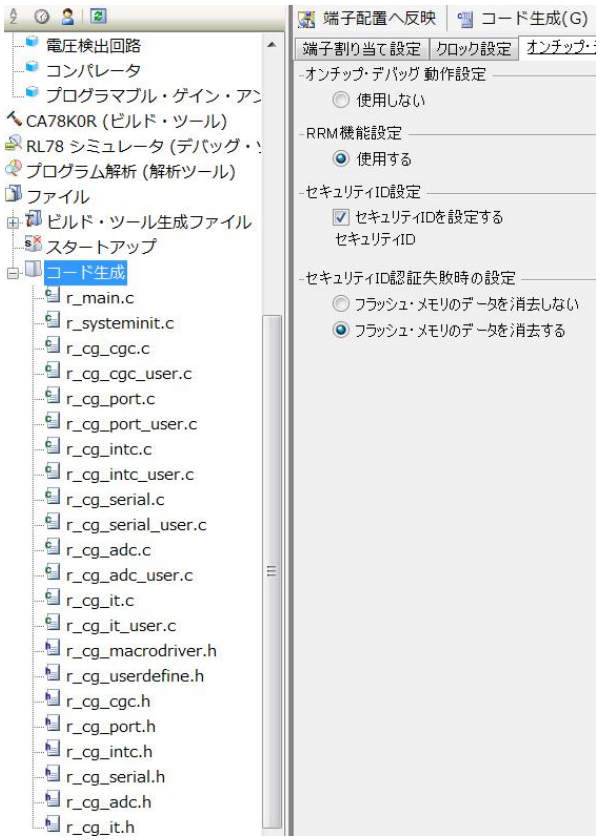
インターバルタイマを使用する場合、以下のように設定します。
例として１ｍｓｅｃに１回、割り込みが入るインターバルタイマを設定します。

電圧検出回路の電圧を設定します。本基版は電源ＯＮ時のリセットを内蔵のパワーオンリセット回路（ＰＯＲ）と電圧検出回路（ＬＶＤ）の併用で行っています。ＰＯＲだけですと、１．５１Ｖでリセット解除されますので、ＬＶＤで電圧を設定します。一般に、電池動作（３．０Ｖあたり）で動作させる場合はＰ

ＯＲのみ、３．３Ｖや５Ｖで動作させる場合はＬＶＤを設定してリセット電圧を２．８１Ｖ～で動作させます。デフォルトでは「使用しない」になっていて、**「使用する」に変更しないと、デバック中は問題なくても、単独で動作させたときに、電源投入時、不安定な動作になります。**



上記設定を終えたら「コード生成」をクリックします。



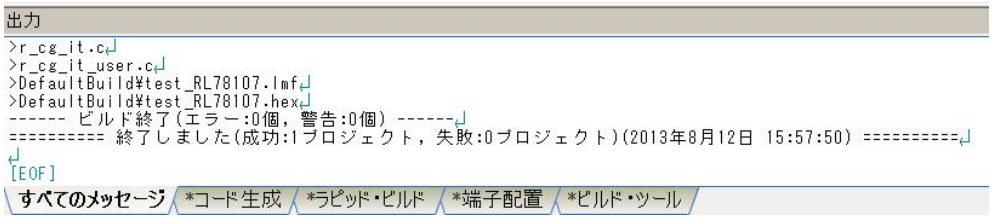
ｒ__ｍａｉｎ．ｃを初め、設定内容を反映したソースファイルが生成されます。

プロジェクトをリビルドしてみます。

ビルド　リビルド　デバッカにダウンロード

ビルド後、デバッカにダウンロード

ビルド終了（エラー：０個、警告：０個）と出れば正常です。



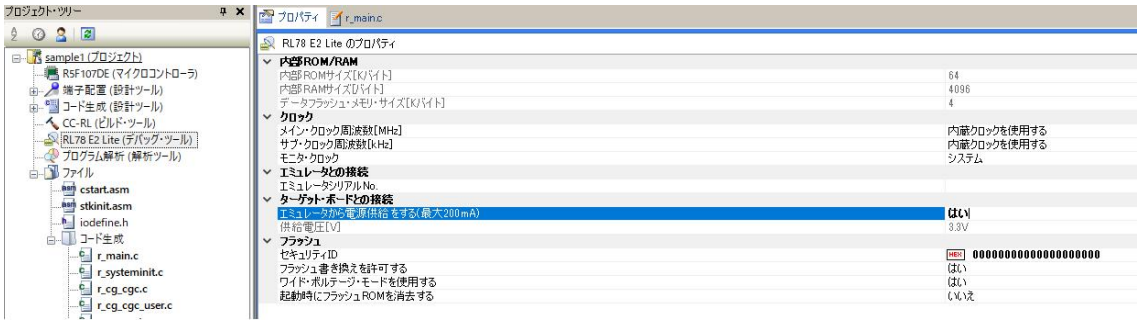
ｂ－２：自動生成されたプログラム

以下省略

Ｓｙｓｔｅｍｉｎｉｔ（ｖｏｉｄ）関数の中身

Ｒ＿ＰＯＲＴ＿Ｃｒｅａｔｅ（）；　　//ポート初期化
Ｒ＿ＣＧＣ＿Ｃｒｅａｔｅ（）；　　//クロック初期化
；
Ｒ＿ＡＤＣ＿Ｃｒｅａｔｅ（）；　　//Ａ／Ｄコンバータ初期化
；
などが自動生成され、電源ＯＮ時に自動的に実行されます。

ｂ－３：Ｅ２Ｌから電源供給
デバッカにはＥ２Ｌを使用します。Ｅ２Ｌから電源を供給する設定は



ＲＬ７８Ｅ２Ｌｉｔｅ（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」、電圧は３．３Ｖ固定です（Ｅ１は３．３Ｖと５Ｖが選べました）。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Ｅ２Ｌ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います！

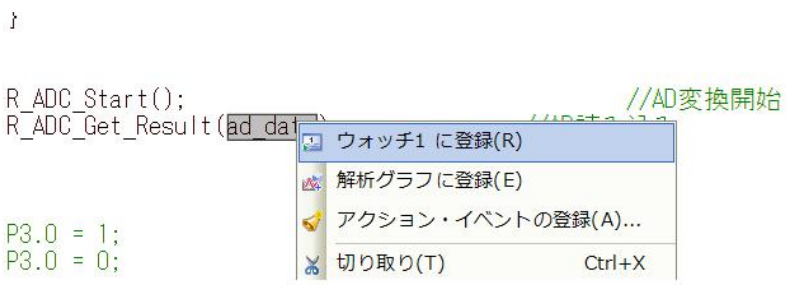
そうならないためにＳｔａｒｔ　ｕｓｅｒ　ｃｏｄｅ、、、とＥｎｄ　ｕｓｅｒ　ｃｏｄｅ、、、の間にユーザープログラム、コメント、定数、変数宣言等を書いてください。

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

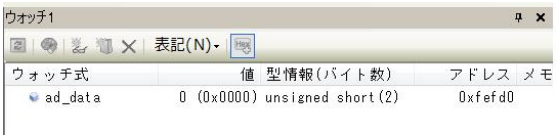
ここにユーザーコードを書けば、「コード生成」を繰り返しても、消されることはありません！！

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

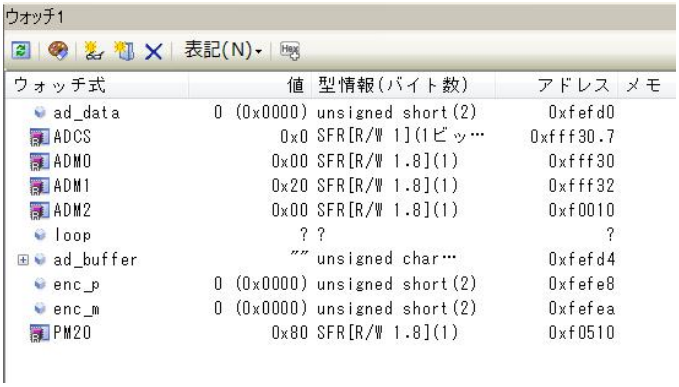
ｂ－５：変数を見る



見たい変数をコピーして右クリック→ウオッチ１に登録



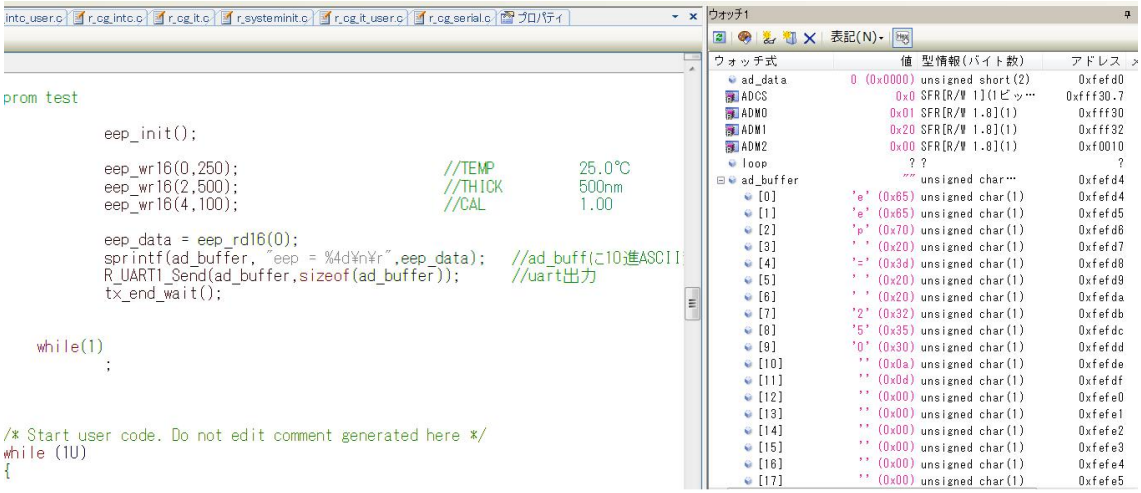
ｂ－６：変数変化を実行中に確認する



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、初期設定では動作中は更新されません。そこで、

ＲＬ７８　Ｅ２Ｌ（Ｓｅｒｉａｌ）（デバック・ツール）→プロパティ→デバックツール設定で実行中のメモリアクセス、表示更新を「はい」にすると、実行中でも変数の変化が確認できます。

下記例はｓｐｒｉｎｔｆでａｄ＿ｂｕｆｆにｅｅｐ＿ｄａｔａの値が格納されるのをリアルタイムで表示しています。



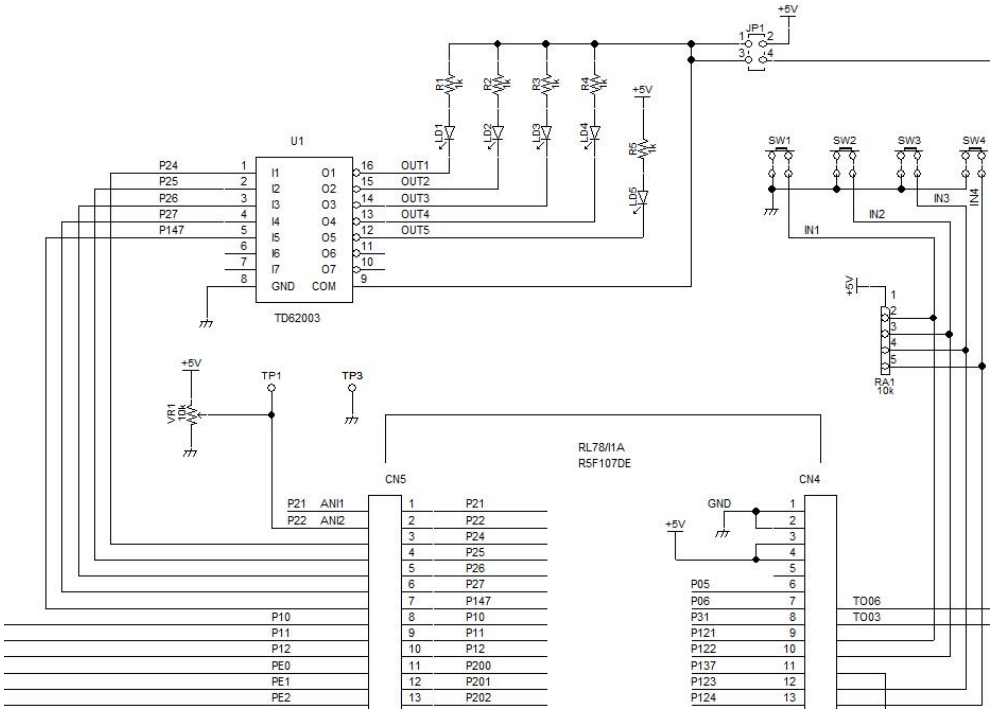
なお、使用端子や動作プログラムが同じような構成のものの場合、ホルダをコピーし、ホルダ名、ｍｔｐｊファイルの名前を変更すれば、それまでの設定はそのまま使えます。変更もその上から行うことが出来ます。

２．サンプルプログラム

２－１　ｓａｍｐｌｅ１　キー入力でＬＥＤを点灯

【　概要　】

ＬＥＤ１，２，３，４が点滅し、ＳＷ１キーを押すと消灯します。マイコンの基礎であるポートの入出力を学習します。電源はＥ２Ｌから供給しますので、ＵＳＢケーブルは接続しないでください。以下の写真ご参照。



【　ハードウエア　】

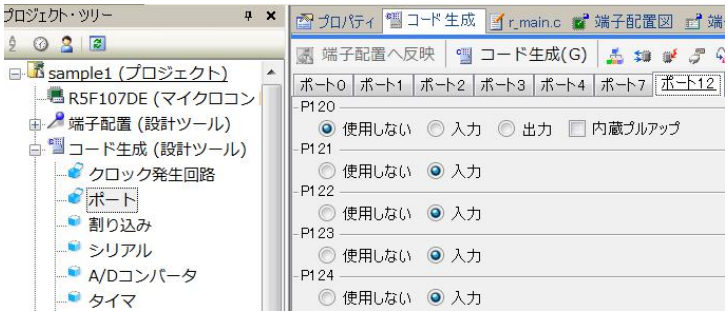
キーはＲＡ１でプルアップされており、入力ポートＰ１２１，１２２，１２３，１２４は押されないとき

１（＋３．３Ｖ）、押されたとき０（０Ｖ　ＧＮＤレベル）になります。出力ポートＰ２３，２５，２６，２７はトランジスタバッファＴＤ６２００３の入力に接続されていて、入力Ｉｘが１（＋３．３Ｖ）で出力Ｏｘがｏｎ（出力が０Ｖ近傍まで落ち、ＬＥＤが点灯します）、０（０Ｖ　ＧＮＤレベル）でＯＦＦ（出力が＋３．３Ｖ近傍まで上がり、ＬＥＤが消灯します）
ｘ＝１～７

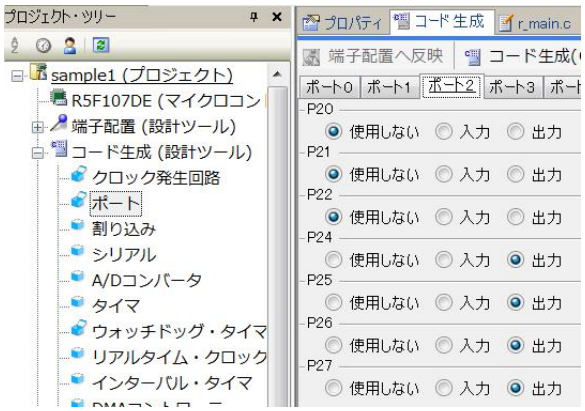
回路図に５Ｖとあるのは、ＵＳＢケーブルをさして使用するｓａｍｐｌｅ２以降はＵＳＢからの電圧を電源にするので５Ｖになります。このｓａｍｐｌｅ１だけはＥ２Ｌから電源を供給するので３．３Ｖになります。

【　コード生成　ポートの設定　】

あらかじめ、ポートＰ１２１，１２２，１２３，１２４は入力に設定されています。



ポート２４，２５，２６，２７は出力に設定されています。



【　プログラム　】

```
①void main(void)
{
②      R_MAIN_UserInit();
      /* Start user code. Do not edit comment generated here */
③      while (1U)
      {
④                        P2 = 0x00;                    //LD1,2,3,4 OFF
⑤                        wait(1000000);                //時間待ち
⑥                        P2 = 0xf0;                    //LD1,2,3,4 ON
                         wait(1000000);                //時間待ち
```

```
⑦                       while(P12.1 == 0)               //SW1 が押されているとループ
                        {
                                P2 = 0x00;              //LD1,2,3,4 OFF
                        }
    }

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

【　解説　】

①void main(void)

メイン関数です。電源ＯＮで自動的に実行されます。

②　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（同じファイル）のところにあります。

③　while (1U)

以下の{　}の中を無限ループします。１Ｕは数字の１で、ｕｎｓｉｇｎｅｄ型であると明示しています。

④　P2 = 0x00;　//LD1,2,3,4 OFF

ポート２の出力データを０にしています。Ｐ２．４、Ｐ２．５、Ｐ２．６、Ｐ２．７が０になるので、ＬＥＤが４つ共消灯します。

バイト	Ｐ２							
ビット	Ｐ２．７	Ｐ２．６	Ｐ２．５	Ｐ２．４	Ｐ２．３	Ｐ２．２	Ｐ２．１	Ｐ２．０
２進数表記	０	０	０	０	０	０	０	０
１６進数表記	０				０			

```
⑤                       wait(1000000);                  //時間待ち
```

人間の目でみて点滅が分かるように時間待ちをしています。１０００００００という数を－１して、０になったら抜けます。

```
void wait(uint32_t wtime)
{
        while(wtime != 0)
        {
                wtime--;
        }
}

⑥                       P2 = 0xf0;                      //LD1,2,3,4 ON
```

ポート２の上位４ビット出力データを１にしています。Ｐ２．４、Ｐ２．５、Ｐ２．６、Ｐ２．７が１になるので、ＬＥＤが４つ共点灯します。

バイト	Ｐ２							
ビット	Ｐ２．７	Ｐ２．６	Ｐ２．５	Ｐ２．４	Ｐ２．３	Ｐ２．２	Ｐ２．１	Ｐ２．０

重み	8	4	2	1	8	4	2	1
２進数表記	1	1	1	1	0	0	0	0
１６進数表記	f				0			

２進数１１１１が１６進数ではｆになる理由ですが、４ビット毎にデータが１の重みを加算します。
Ｐ２．７は重み８、Ｐ２．６は４、Ｐ２．５は２、Ｐ２．４は１　ですので、８＋４＋２＋１＝１５となります。Ｐ２．３～Ｐ２．０は０です。

１０進数	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
１６進数	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f

１０進数表記で１５は１６進数表記ではｆ（Ｆ）になります。なので、０ｘｆ０と書きます。頭に付ける０ｘは以降の数が１６進表記であることを表しています。

⑦
```
while(P12.1 == 0)                //SW1 が押されているとループ
{
        P2 = 0x00;               //LD1,2,3,4 OFF
}
```

ＳＷ１は押さないときは何もしません。Ｐ１２．１の配線がＲＡ１の１０ＫΩでプルアップ（電源に吊られること）されていますので、電圧は＋３．３Ｖとなり、入力したデータは１となります。

ＳＷ１は押されるとＧＮＤと接続されますので、０Ｖ、入力データは０となります。
プログラムでは押されたとき、データ０のときにｗｈｉｌｅで無限ループとなり、Ｐ２に０を出していますので、ＬＥＤが全て消灯します。離すと点滅が再開します。

【　演習　】
ｓａｍｐｌｅ１で動作を理解できた方は、演習問題に進んで下さい。課題は
キー入力で押したところのＬＥＤを点滅させて下さい。

回答例　ｓａｍｐｌｅ１＿ａ

回答例は例であり、課題に合致していれば、書き方がこの通りである必要はありません。

以下省略

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