RL78G14-FPB に shiftOut 機能を追加してみます。

以下の shiftOut を作成してみました。

```
012fa  void shiftOut (uint8_t dataPin, uint8_t clockPin, uint8_t bitOrder, uint8_t value)
       {

           uint8_t loopc;
           uint8_t work;
           uint8_t mask;
01307      sck_pin = clockPin;                          // シフト・クロックピンの設定
0130c      mosi_pin = dataPin;                          // データピンの設定
01311      work = value;                                // 転送データセット

01315      digitalWrite( sck_pin, LOW );                // SCKを立ち下げる

0131d      if ( MSBFIRST == bitOrder )
           {       // 上位ビットから転送モード
01326          for ( mask = 0x80 ; mask > 0 ; )
               {
0132e              digitalWrite( mosi_pin, ( work & mask ) ); // データを出力
01339              mask >>= 1;                          // マスクビットを更新
01340              digitalWrite( sck_pin, HIGH );       // SCKを立ち上げる
01348              digitalWrite( sck_pin, LOW );        // SCKを立ち下げる
               }

           }
           else
           {       // 下位ビットから転送モード
0135a          for ( loopc = 8 ; loopc > 0 ;  )
               {
0135f              digitalWrite( mosi_pin, ( work & 0x01 ) ); // データを出力
01370              loopc--;                             // カウンタを更新
01375              work >>= 1;                          // データをシフト
0137c              digitalWrite( sck_pin, HIGH );       // SCKを立ち上げる
01384              digitalWrite( sck_pin, LOW );        // SCKを立ち下げる
               }

           }

01394  }
```

これを呼び出しているところを以下に示します。202 行目～209 行目が shiftOut 関数を呼び出しているところです。その後にブレークポイントを設定して波形を観測します

```
                   case 0x08:                   // 100milli second digit timing
009bc                  dt_wk = precount2;               // get 100milli second digit data
009c1                  seg_data = LED_SEGMENT[dt_wk];   // set segment of 100milli second digit data
009c9                  com_sel = LED_DIGIT[3];          // set 100milli second digit
                       break;

                   default:                     // minute digit timing
009d0                  dt_wk = mini_data;               // get minute display data
009d5                  seg_data = LED_SEGMENT[dt_wk];   // get segment of minute display data
009dd                  com_sel = LED_DIGIT[0];          // set minute digit


               }

009e2          digitalWrite( SS_PIN, LOW );      // turn on SS signal(low level)

009e9          shiftOut (MOSI_PIN, SCK_PIN, LSBFIRST, seg_data); // segment data out

009f4          shiftOut (MOSI_PIN, SCK_PIN, LSBFIRST, com_sel);  // digit data out


009ff          digitalWrite( SS_PIN, HIGH ); // turn off SS signal(rise up)

           /*-------------------------------------
               every 4milli seconds timing
               if count enable, then count time up.
           -------------------------------------*/

00a06          if (( 25 <= ++precount1 ) && ( 1 == time_mode ))
```

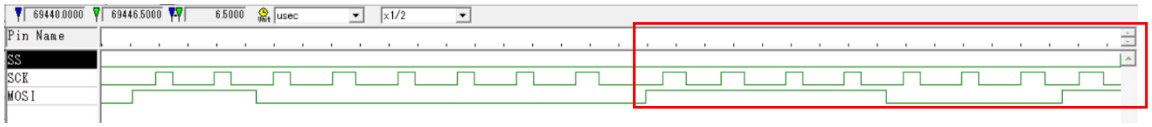
波形はシミュレータを用いて、タイミングチャートで確認します。

最初のブレークでは、time_work が 4 となり、下の波形のように桁3（秒の桁）の表示です。

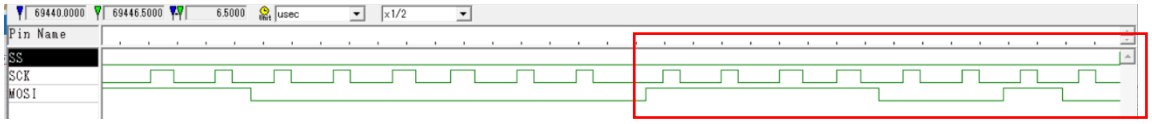
最初の 8 ビット分がセグメント（負論理）をセグメント DP からシフトアウトされています。2 つ目の 8 ビットが桁選択用で、6 ビット目が 1 であることから 3 桁目の秒の桁となります。この桁ではセグメント DP がローなので、小数点が点灯しています。

次のブレークでは、time_work が 8 となり、下の波形のように桁選択は 5 ビット目が 1 であることから、桁 4（0.1 秒の桁）の表示です。

次のブレークでは、time_work が 12 となり、分の桁（桁 1）の波形波形となります。



最後は、time_work が 0 で、十秒（桁 2）の波形となります。



このようにシミュレータで得られた波形は期待した波形と一致しています。

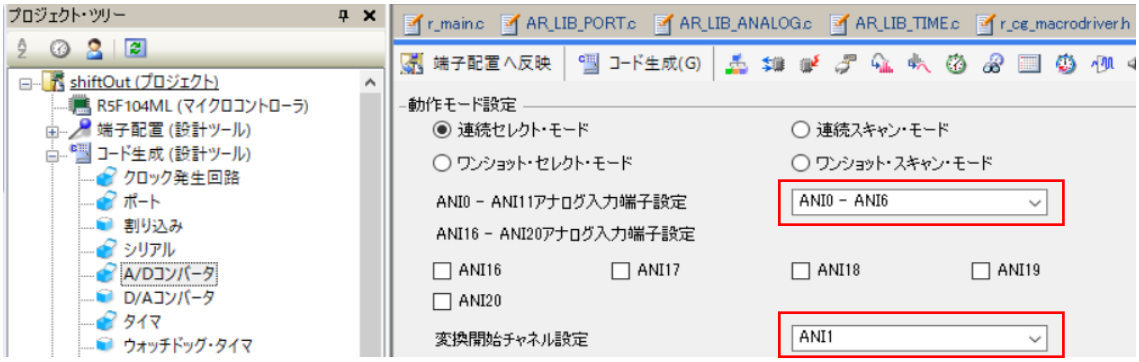
それでは、実際に RL78G14-FPB に MultiFunction シールドを接続して、タイマを動作させてみます。

下の写真が、実際に動作させたときのものです。

アナログ入力機能端子のデジタル入力機能には対応していないので、カウント開始／カウント停止は RL78G14-FPB 上の SW_USR を使う必要があります。

そこで、ArduinoAPI に勝手に手を加えることにします。元にしたのは、最新の AN6115（アナログ入出力(Arduino API)）です。

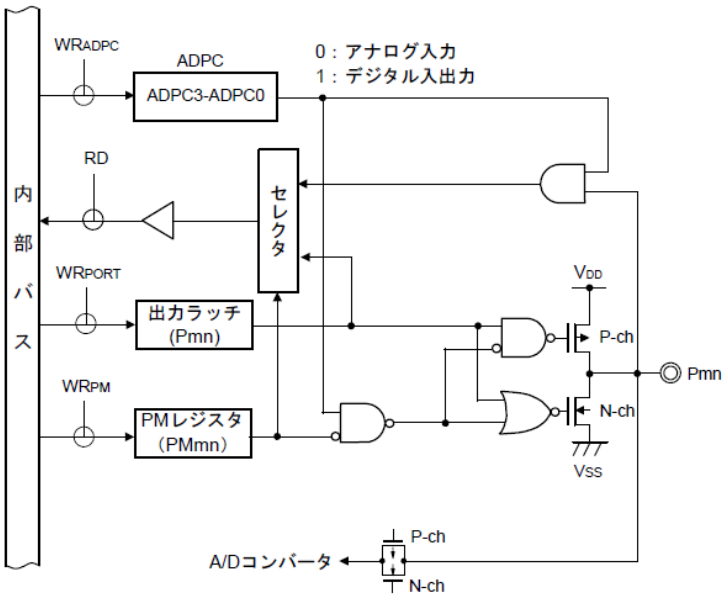
このプロジェクトを開くと、A/D コンバータは以下のようになっています。



ここで、MultiFunction シールドでスイッチに割り当てている A1（ANI5）～A3（ANI3）をデジタル入力に、ポテンショメーターに A0（ANI6）への対応を考えてみます。最低でも ANI23

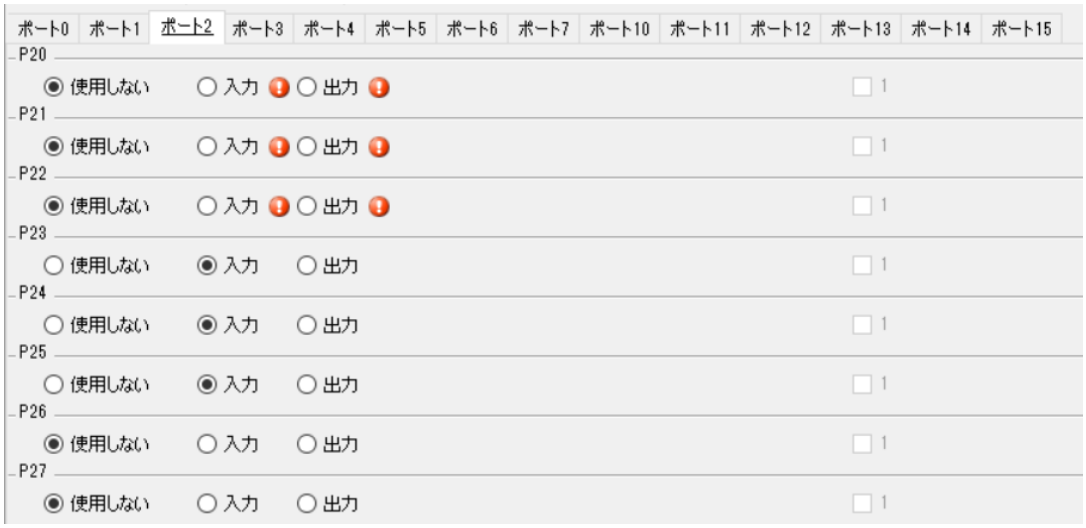
(P2.3)はデジタル入力にするために、ADPC を変更する必要があります。RL78/G1x は、困ったことに ANI0 の方からアナログ入力に割り当てるようになっています。つまり、ANI6 をアナログ入力に設定して、P23～P25 をデジタル入出力には設定できません。そこで、RL78/G13 の端子のブロック図を見ると、ADPC はデジタル機能を禁止しているだけのようです。

図2－5 端子タイプ 4-3-1 の端子ブロック図

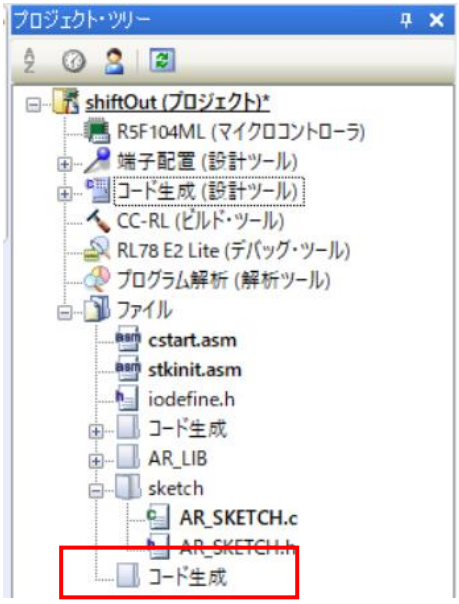


このことから、ANI22 までをアナログ入力に変更します。

これで、P26 を入力ポートに設定できます。(ここで、設定しても意味はないと思いますが。)

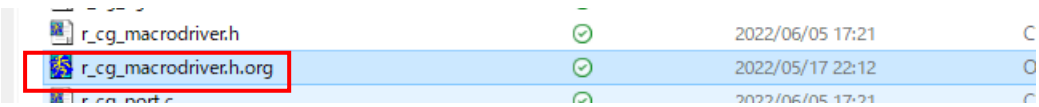


これで、コード生成すると、プロジェクト・ツリーの下の方に「コード生成」というのができますが、これは空っぽです。これが、KatoNagano さんが投稿されている、「「コード生成」カテゴリについて」だと思います。



目障りなだけで、他には影響はしないようです。コード生成する前にプロジェクトを保存しておいて、コード生成したら、プロジェクトを閉じるときに保存しないでください。

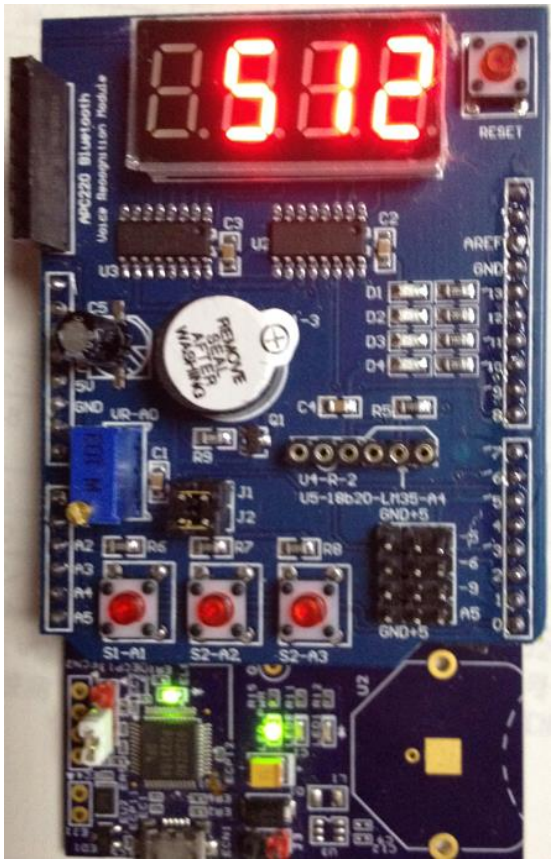
ArduinoAPI の場合には、この状態でビルドすると、大量にエラーが発生します。この原因は「r_cg_macrodriver.h」が書き換わってしまうからです。



この対策としては、前のページの図で赤い四角で囲んだ「r_cg_macrodriver.h.org」の内容を「r_cg_macrodriver.h」にコピペすればエラーはなくなります。

これは、同じソースを CS+だけでなく IAR 環境でも流用するためのものようです。

最後に、RL78G23-64PFPB で作成した半固定抵抗の A/D 変換です。この結果だけを示します。ゼロサプレス表示になっているので、時間表示とは違うことが分かります。また、ADPC で A0 端子はデジタル入出力に設定されているのにきちんと A/D 変換されているのが確認できました。



これで、RL78G23-64PFPB の主な使用例は RL78G14-FPB には持ってきたことになります。

以上