RL78/G14-FPB で遊ぶ(1)

RL78/G14-FPB のアプリケーション・ノートが発行されたので、そのプログラムを読んで少しいじってみようかと思います。

プログラムは 2 つアップされていますが、ここでは、「7 セグメント LED 点灯制御（Arduino API）」を対象にします。

このアプリケーション・ノートのプログラムは、3 桁の7SEG-LED と単体の LED8 個を接続して制御するプログラムのようです。対象のハードは、workshop Nak の製品で、製造はされているようですが、2016 年で通販はやめているようです。このボードの資料は以下にあります。

http://wsnak.com/kit/164/doc164_165.pdf

このボードはポートを使った制御では使いやすいのですが、入手に難があります。そこで、試してみたい人のために入手が簡単な「OSL40562-Lx」（4 桁の 7SEG-LED・カソードコモン）への置き換えを考えます。（手元に、OSL40562-LG（グリーン）があったので、これを使います）。これは、7SEG-LED4個が 1 パッケージに入り、セグメント信号は4桁分が接続されているので、12 本の信号だけで済みます。ただし、セグメントの電流制限抵抗は必要になります。

部品数を減らしたいので、ドライバ IC なしで直接ドライブしてみます。なぜ、カソードコモンかというと、RL78 のポートはハイレベル（10mA）よりもロウレベル（20mA）の方がドライブ能力は強いので、複数のセグメントの電流がまとまるコモンをドライブ能力が大きいロウレベルでドライブするためです（Arduino の解説本等でポートの電流が 40mA とかよく書かれているのがありますが、これは問題があります。40mA は絶対最大定格で、通常の動作条件では最大 20mA です。決して 40mA 流していい訳ではありません。このことを明記している本も一部にはありますが、殆どは 40mA が独り歩きしているようです）。

最近の LED は 2～3mA も流せば結構明るく光ります。そこで、3V 動作で、電流制限抵抗は 1kΩにしておきます。これだと、セグメントが全て点灯しても 20mA は超えないと考えられます。

なお、LED の特性として、順方向電圧（vf）が 3.3V（@20mA）とあり、この条件で判断すると 3V での駆動はできません。しかし、20mA も流さなければ、Vf はもっと低くてよくなります。発光する色によって Vf は異なってきますが、1.5～2V 程度と考えられます。

一方、RL78/G14 のポートですが、ハイレベル出力電圧は電気的特性では1. 5mA で 0.5V（MAX）降下すると規定されています。これは、どんなに製造のバラつきが最悪にぶれてもすべての使用条件範囲で保証されているものです。実際のデバイスの実力としては「設計支援情報」として公開されているポートの特性曲線を参照すると、ハイレベル出力特性は 25℃で 1.5mA 時の電圧降下は 0.1V 以下です。つまり、3V の電源電圧なら、2.9V 以上の出力電圧になります。また、ロウレベル出力は、20mA でロウレベル出力電圧は 1.3V と規定されていますが、特性曲線ではその半分程度です。つまり、セグメントとコモン間の電圧は 2.25V となります。ここから、LED の Vf と

して 1.5V を引くと、抵抗にかかる電圧は0. 75V となり、1kΩの抵抗だと、1mA も流れません。しかし、電流が減ると、すべてのセグメントを点灯させても合計は 10mA 以下なので、抵抗にかかる電圧は 1V 程度で釣り合ってきます。問題はこれで点灯するかです。さらに、セグメント1つと全セグメント点灯時の明るさの変化がきになるかどうかです。8.1 秒を表示したときの全セグメント点灯と点灯セグメントが少ないときの 1 表示が並んだときの写真を示します。明るさが変化していることは気になりません(それより、A セグメントと G セグメントに移りこんだ照明の方が気になりました)。

明るさも、まぶしくはなく自然に見えます。照明の映り込みがなければ、もっとはっきり見えます。

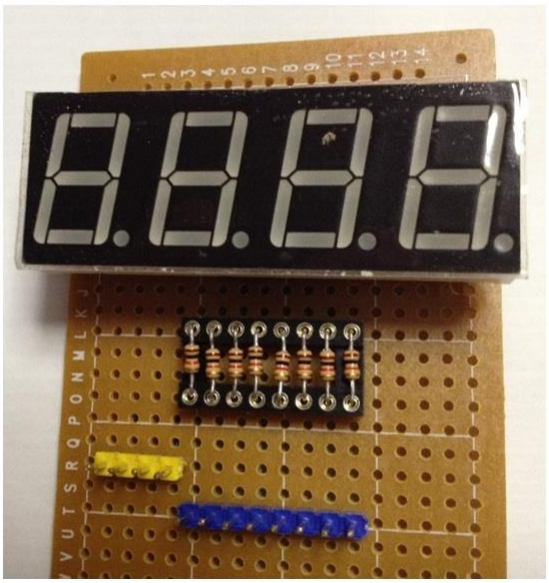
信号の接続を下の表に示します。OSL40562-Lx の桁は左から順に DIG1、DIG2、DIG3、DIG4 と並んでいるとデータシートには記載されていました。実際にプログラムを動かしたところ、DIG2 と DIG4(一番右端)の信号が入れ替わっているようです(表では赤字で修正しています)。

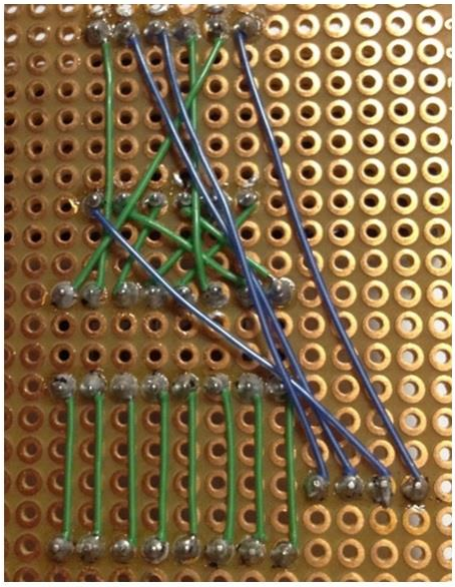
信号名	OSL40562-Lx のピン番号	G14-FPB の信号
セグメント A	11	D0
セグメント B	7	D1
セグメント C	4	D2
セグメント D	2	D3
セグメント E	1	D4
セグメント F	10	D5
セグメント G	5	D6
セグメント DP	3	D7
DIG1	12	D11
DIG2	6~~9~~	D10
DIG3	8	D9
DIG4	9~~6~~	D8

セグメント信号は 1kΩの抵抗を介して G14-FPB と接続し、桁(DIG)信号は直接 G14-FPB と接続します。

これらの配線(抵抗を含む)はブレッドボードを用いた方が簡単かもしれませんが、適当なブレッド

ボードの手持ちがなかったので、今回は手元にあったユニバーサルボード（50mm×72mm）で組み上げました。左が部品面、右が配線面の写真です（回路は修正済のものです）。電流制限抵抗は、16 ピンの IC ソケットを使って接続しています。ここに刺す抵抗の値を置き換えることで、簡単に明るさを変更できます。下側の青いヘッダピンがセグメント信号用で、右から順にセグメント A から並んでいます。黄色いヘッダピンが桁選択用の信号用です。信号線は、そのまま G14-FPB に平行ケーブルで接続できます。





以上がハードウェア関係の内容です。次にプログラムの見直し内容です。

プログラムで今回変更するのは、AR_SKETCH.c だけで済みます。最初に桁の選択信号をアクティブ・ハイからアクティブ・ロウに変更します（ついでに、元の状態に簡単に戻せるようにしています）。

AR_SKETCH.c の 41 行目辺りのグローバル変数を定義しているところの先頭に定義を追加します。最初に CATHOD_COM を定義し、続けて CATHOD_COM が定義されていたら、LED_OFF を 0x01 に LED_ON を 0x00 に定義します。CATHOD_COM が定義されていなかったら、LED_OFF を 0x00 に LED_ON を 0x01 に定義します。

画面イメージを下に示します。

```
/*****************************************************************************↓
Global variables and functions↓
*****************************************************************************/
#define CATHODE_COM↓
↓
#ifdef CATHODE_COM↓
#define LED_OFF  ( 0x01 )↓
#define LED_ON   ( 0x00 )↓
#else↓
#define LED_OFF  ( 0x00 )↓
#define LED_ON   ( 0x01 )↓
#endif↓
↓
```

こうすることで、CATHOD_COM の定義を残すかコメントアウトするかで簡単に切り替えることができるようにしています。

次に、loop の中の switch 文の3つの case 文と default の最初の digitalWrite の 2 番目の引数を OFF から LED_OFF に変更します。

```
        switch ( time_work )↓
        {↓
            case 0x00:                  // 10second digit timing↓
                digitalWrite(comPin3,LED_OFF);  // turn off minute digit↓
                seg_data = SEG_TABLE[(sec_data / 10)]; // get 10second digit data↓
                set_SEG( seg_data );    // set segment data↓
                com_sel = comPin0;      // set 10second digit↓
                break;↓
↓
            case 0x04:                  // second digit timing↓
                digitalWrite(comPin0,LED_OFF);  // turn off 10second digit↓
                seg_data = SEG_TABLE[(sec_data % 10)]; // get second digit data↓
                set_SEG( ( seg_data + 0x80 ) ); // set segment data↓
                com_sel = comPin1;      // set second digit↓
                break;↓
↓
            case 0x08:                  // second digit timing↓
                digitalWrite(comPin1,LED_OFF);  // turn off second digit↓
                seg_data = SEG_TABLE[precount2]; // get second digit data↓
                set_SEG( seg_data );    // set segment data↓
                com_sel = comPin2;      // set 100milli second digit↓
                break;↓
↓
            default:                    // minute digit timing↓
                digitalWrite(comPin2,LED_OFF);  // turn off second digit↓
                seg_data = mini_data;   // get minute display data↓
                set_SEG( seg_data );    // set segment data↓
                com_sel = comPin3;      // set minute digit↓
'
```

最後に switch 文が終わった次の digitalWrite の 2 番目の引数を ON から LED_ON に変更します。

```
        }↓
        digitalWrite(com_sel,LED_ON);   // enable display↓
```

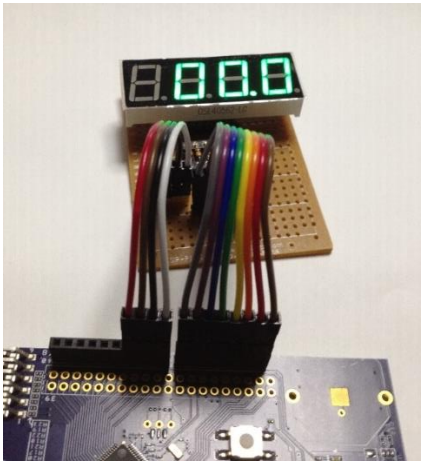
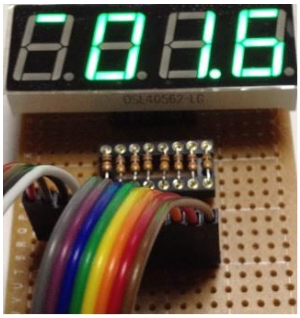
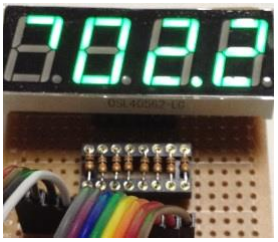
一応これだけの変更で、LED のカソードコモンへの置き換えに対応します。ここまでの変更でビルドして、エラーが発生しないことを確認したら、G14-FPB にダウンロードします。

G14-FPB には、12 本の(平行な)ケーブルで LED ボードを接続しておきます。単にダウンロードしただけでは、LED には何も表示されないので、プログラムを動作させます。

プログラムを動作させると、次ページの左側の写真に示すように、0 の表示が 3 つ並びます。ここでは、この表示になれば、一応は OK とします。

そこで、G14-FPB 上の SW_USR を押します。すると、カウントを開始します。すると、右端の桁と左から 2 番目の桁の動作が入れ替わっていることが分かりました。つまり、DIG2 と DIG4 の pin が逆になっていると考えられます。ここは、無条件に配線を変更します。

カウントして 1 分が経過すると左端のセグメント A が点灯し、さらに 1 分経過するとセグメント B も点灯します。これは、オリジナルの構成での個別の LED に対応した動作です。

分の表示(switch 文の default)での処理は分のデータ(mini_data)は、7 セグメント用データへ変換されていません。

```
default:                        // minute digit timing↓
    digitalWrite(comPin2,LED_OFF);  // turn off second digit↓
    seg_data = mini_data;       // get minute display data↓
    set_SEG( seg_data );        // set segment data↓
    com_sel = comPin3;          // set minute digit↓
```

そこで、この部分を他の桁と同じように変換します。

```
default:                        // minute digit timing↓
    digitalWrite(comPin2,LED_OFF);  // turn off second digit↓
    seg_data = SEG_TABLE[mini_data];    // get minute display data↓
    set_SEG( seg_data );        // set segment data↓
    com_sel = comPin3;          // set minute digit↓
```

これで表示は数字になるはずです。次は、分データのカウント処理です。その部分をいかに示します。この if 文で全ビットが 1 なら次のデータを 0 にし、そうでなければ、表示データを下位ビットからセットしていきます。

```
if ( 0xFF == mini_data )↓
{                               // over flow↓
    mini_data = 0;  // clear minute data↓
}↓
else↓
{                               // change minute data↓
    mini_data <<= 1; // shift 1bit left and↓
    mini_data++;    // set bit0↓
}↓
```

ここを単純に 9 なら次は 0 にし、そうでなければ＋1 するようにします。

```
if ( 9 == mini_data )↓
{                       // over flow↓
    mini_data = 0;      // clear minute data↓
}↓
else↓
{                       // change minute data↓
    ++mini_data; ›      // count minute data↓
}↓
```

ここまでで、再度ビルドして動作確認します。プログラムを起動すると、今度は 0 表示が 4 桁に変わりました。

SW_USR をおして、カウント開始すると、左端の桁は分のカウンタになっていて、9 分まで表示できるので、9 分 59 秒 9 までカウントできるようになります。

第 1 回目の内容しては、ここまでにしておきます。また、この段階のプロジェクトを添付しておきます。

次回は、プログラムの構造に手を加える予定です。

いろんなことをやろうとすると、ArduinoAPI でリアルタイム制御を行うのは、面倒な気がします。RL78G14 のハードウェア資源を活用すれば、よりシンプルにできそうなことが結構ありそうですが、ここではまだ ArduinoAPI ベースで考えていくことにします。

最初に手を付けるのは、スイッチ入力で、短押しと長押しを考えていきます。さらに、カウント値と表示用を分離することにします。これで、使い方が大きく広がるはずです。

以上