RL78/G10 キャンペーンのプレゼントの使い方3

1　10pin の RL78/G10 での圧電スピーカー応用

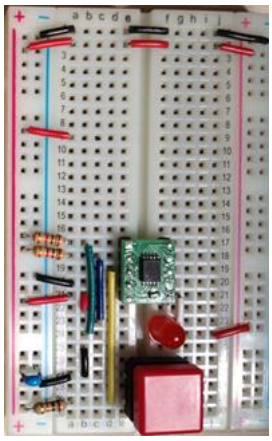

Fig1

Fig1 に「RL78/G10 キャンペーンのプレゼントの使い方」で作成した 10pin の RL78/G10 を使ったボードを示します。

今回はこれに回路を追加して、スイッチを押下している時間だけ圧電スピーカーから音を出すようにします。出す音としては、まずはドレミのラの音（440Hz）にします。

作成した回路を Fig2 に示します。これは、「RL78/G10 キャンペーンのプレゼントの使い方2」の Fig17 の回路を 10pin の RL78/G10 に変更したものです。

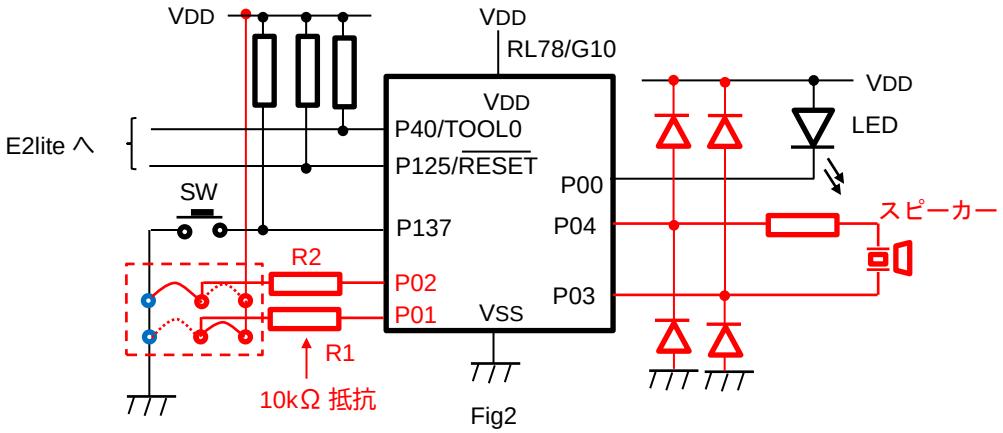

Fig2

2　10pin の RL78/G10 での変更点

2.1　pin 数の不足

10pin の RL78/G10 では、pin数が少ないので、音程の指定に 3 本のポートを割り当てることはできません。そこで、SW（スイッチ）を流用することにします。

SW の設定	R2 の接続		R1 の接続		設定値	音程
オン	j18	「−」	j19	「−」	000	ド(C4)
オン	j18	「−」	i19	「+」	001	レ(D4)
オン	i18	「+」	j19	「−」	010	ミ(E4)
オン	i18	「+」	i19	「+」	011	ファ(F4)
オフ	j18	「−」	j19	「−」	100	ソ(G4)
オフ	j18	「−」	i19	「+」	101	ラ(A4)
オフ	i18	「+」	j19	「−」	110	シ(B4)
オフ	i18	「+」	i19	「+」	111	ド(C5)

残念ながら、音程の補正用の可変抵抗器は接続できません。

2.2　10pin の応用回路の配線

Fig3 に配線にイメージを示します。今回は右側の配線です。

① g8 と g17 の間を 1kΩの抵抗で接続します。
② h12 と h18 を接続します(白色)。
③ j10 と右の－を接続します(黒色)。
④ f10 から f12 をダイオードで接続します。
⑤ h10から h8 をダイオードで接続します。
⑥ j8 から右の＋をダイオードで接続します。
⑦ j12 から右の＋をダイオードで接続します。
⑧ i8とi12に圧電スピーカーを接続します(配線が隠れてしまうので、圧電スピーカーを接続する場所だけ○で示しています)。
注:ダイオードは電流が流れる方向(カソード)を矢印で示しています。
⑨ 音程選択部の抵抗の配置は前ページの表を参照してください。

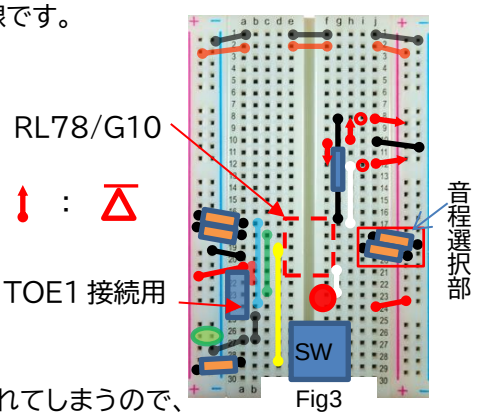

Fig3

2.3　PWM 出力チャネル数の不足

16pin の RL78/G10 にはタイマが 4 チャネル内蔵されていたので、2 本の PWM 信号を使うことができました。しかし、10pin の RL78/G10 にはタイマが 2 チャネルしか内蔵されていないので PWM 信号は 1 本しか使えません。このため、AppliletEZ PL for RL78 では対応できません。

ただし、音を出すためには、PWM は必須の機能ではありません。同じような波形の信号を出力すればいいだけです。このためには、「方形波出力機能」が使用可能です（本来はこちらが、一般的な方法です）。また、ソフトウェア（プログラム）を使って、必要なタイミング（間隔）でポートを操作して（一定間隔で 0 と 1 の出力を繰り返すことで）も、同じような波形を出力することができます。

2.4　出力する信号の発生方法

「ラの音」に対応する 440Hz についても Fig4 の（a）、（b）,（c）、（e）に示すものがあります。（a）の正弦波は 440Hz の成分だけしか含まない音です。（c）の方形波信号がデジタルでよく使用する 440Hz の信号です。この中には 440Hz の正弦波（基本波）が含まれています。それ以外に 3 倍の周波数（1320Hz）の正弦波成分、5 倍の周波数（2200Hz）の正弦波成分などの奇数倍の高調波成分が含まれています。この場合にも、音程は、同じく「ラの音」になります。高調波成分が音色になります。

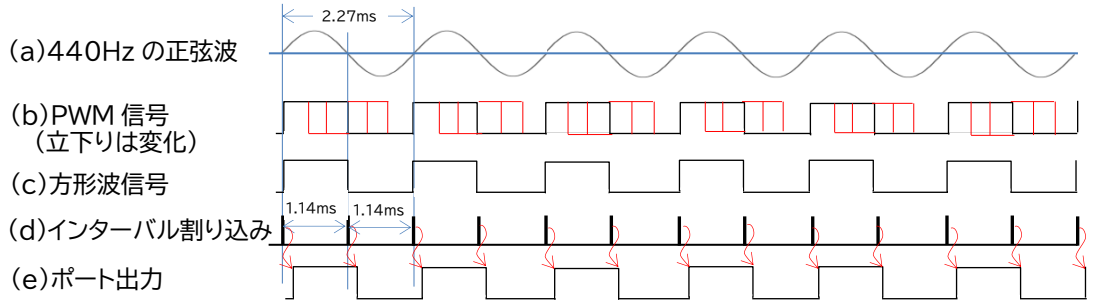

Fig4

Fig4 の（b）PWM 信号で赤く書かれた部分がPWM信号のデューティを変化させた場合の例です。デューティが 50%の場合は方形波信号と同じになります。

（e）ポート出力は、（d）のインターバル割り込みでポートをプログラム（ソフトウェア）で反転させることで、方形波信号と同じものを出力させようとするものです。割り込みからポート反転までのプログラムの実行時間が同じなら、方形波信号と同じものが得られます。

今回は、動作が分かりやすいポートをプログラムで操作する方法を使うことにします。なお、この場合でも、タイマは使う必要があります。この場合に使用するタイマの機能は、「インターバルタイマ」と呼ばれる機能です。インターバルタイマは、一定の時間間隔（インターバル）で割り込みを発生させる機能です。そこから、ポートが実際に変化するまでは遅くなりますが、変化する間隔が同じなら問題ありません。

3　10pin の RL78/G10 のプログラム

3.1　RL78/G10 での処理内容

RL78/G10 はインターバルタイマの割り込みが発生する度にポートの出力を反転します。そうすると、出力される信号の周期は、インターバルタイマの割り込みの発生する間隔の 2 倍になります（つまり、インターバルタイマに設定する時間間隔は発生したい信号の周期の半分にする必要があります。これは、方形波出力でも同じです）。

また、RL78/G10 は、割り込み要求（インターバルタイマが要求）から実際のプログラムの実行開始までの時間が 11～18 クロックと割り込み要求が発生していた命令によって 7 クロック分バラつきます。これは、インターバル時間に対して 1/1000 程度なので無視してもいいのですが、HALT（停止状態）で割り込みを待てば、1 クロックのバラつきに抑えられます。

これよりも影響が大きいのが、ほかの割り込みの実行による保留状態（ほかの割り込み処理が完了するまで実行が待たされた状態）です。これを避けるには、ほかの割り込みは使わないようにする必要があります。

そうすると、問題になるのが SW（スイッチ）が押されたことで起動される音出しの開始処理の割り込みとチャタリング防止のタイマです。このうち、音出し開始の割り込みはインターバルタイマと同時には使わないので無視して構いません。チャタリング防止は音の出力と同時に処理する必要があります。そもそも、チャタリング防止のタイマは、それほど厳密な時間は必要ありません。そこで、音を出すためのインターバルタイマの割り込みのポート反転処理の後で処理することにします。

チャタリング防止のための時間間隔はポート反転用の割り込み間隔よりもはるかに長いので、ポート反転処理の回数をカウントして、20 回に 1 回だけ SW（スイッチ）の状態を確認するようにします。

3.2　プログラムでの大まかな処理

RL78/G10 がスタートすると、内蔵されている機能（回路）の初期設定が行われ、その後に作成したプログラム（main と言う名前の関数（プログラム）が実行されます。main 関数は以下のように始まっています。

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    {
        uint16_t work;

        Tone_sel();                     /* 音程選択処理                 */

        while ( P13_bit.no7 == 0 )      /* スイッチが離されるのを待つ   */
        {
            NOP();
        }

        for ( work = 10000 ; work > 0 ; work-- )
        {
            NOP();
        }
```

最初の赤い四角で囲んだ部分が出す音の音程を指定している部分です。次の四角は SW(スイッチ)が押されていた場合に離されるのを待つ処理です。3 つ目は、SW(スイッチ)のチャタリング対策部分です。ここでは、単純にソフトウェアで 10,000 回ループしているだけです。

これらの処理が完了すると、下に示す無限ループがスタートします。

```
        while (1U)
        {

            R_INTC0_Start();            /* スイッチ割り込み許可          */
            STOP();                     /* 省電力モードでスイッチ待ち    */
        /* STOPモード    */

/***************************************************************************/
/*                                                                         */
/* スイッチが押された場合の処理                                            */
/*                                                                         */
/***************************************************************************/

            while ( g_SW_input != 0xFF )
            {
                HALT();                 /* HALTモードで離されるのを待つ */

            }

        }
```

ループの最初で、SW(スイッチ)を押したことを検出する割り込みを許可します。その後、RL78／G10は STOP モード(省エネルギー状態)で SW(スイッチ)が押されるまで停止します。

SW(スイッチ)が押されたら、次は SW(スイッチ)が離されるのをHALTモード(少し省エネ)で待ち、離されたら無限ループの先頭に戻ります。

このように、main 関数での処理は非常に単純です。実際のいろんな処理は、割り込み処理の中で実行されています。

3.3　使用している関数の概要

殆どの処理は2つの割り込みで処理しています(ただし、両方の割り込みが同時に掛かることはなく、どちら片方しか動作しません)。

r_intc0_interrupt 関数は SW(スイッチ)が押されたことを検出した割り込みを処理する関数です。タイマを起動して、タイマ割り込み(INTTM01)を有効にします。最後に、自分自身の割り込みを禁止します。

```
static void __near r_intc0_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */

    if ( P13_bit.no7 == 0 )             /* スイッチはまだ押されているか */
    {
        g_SW_input = 0xFE;              /* スイッチ入力初期値をセット    */
        g_P_image = 0b00010000;         /* ポート出力イメージ初期値設定 */
        g_SW_timing = SW_TIMING;        /* SWチェックタイミング初期設定 */
        R_TAU0_Channel1_Start();        /* 動作タイミングタイマ起動      */
        P0_bit.no0 = 0;                 /* LEDを点灯する                 */
        R_INTC0_Stop();                 /* スイッチでの割り込み禁止      */
    }
```

r_tau0_channel1_interrupt 関数は、実際に音を出す処理を行っている割り込み(INTTM01)処理関数で、システムの中心と言えるプログラムです。

割り込み発生から、ポートへの出力時間をできるだけ短くするために、最初にポートへ出力するようになっています。そのため、出力データは前もって変数 g_P_image に格納しておきます。

変数g_P_imageに設定する値は、初期値として 0b00010000 が設定されています。音を出す場合には 0b00011000 との排他的論理和をとることで、0b00010000 と 0b00001000 が交互に設定されます。音を止める時には 0b00000001 が設定されます。

下に示すプログラムでは、赤い四角で囲んだ部分では変数 g_P_image が0x01(＝0b00000001)なら、タイマを停止して変数 g_SW_input に 0xFF を設定することで、SW(スイッチ)が離されたことをmain関数に通知しています。

その下の青い四角で囲んだ部分が、SW(スイッチ)が離されたこと(P13.7 が 3 回連続して 1)を検出したときの処理です。ここで設定した 0x01 を次の INTTM01 のタイミングで、上の赤い四角で囲んだ部分でチェックしています。

```
static void __near r_tau0_channel1_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */

    P0 = g_P_image;                     /* ポート出力設定                */
    if ( g_P_image == 0x01 )            /* 発音停止か？                  */
    {                                   /* 発音停止の場合                */
        R_TAU0_Channel1_Stop();         /* タイマ動作停止                */
        g_SW_input = 0xFF;              /* スイッチを停止状態に設定      */
    }
    else
    {                                   /* 発音継続の場合                */
        if ( ( g_SW_input & 0x07 ) == 0x07 )    /* スイッチの状態確認    */
        {                               /* スイッチが離された            */
            g_P_image = 0x01;           /* 次のタイミングで発音停止      */
        }
        else
        {                               /* スイッチは押されている        */
```

引き続き SW(スイッチ)が押されていた場合の処理が以下の部分です。

```
            g_P_image = 0x01;           /* 次のタイミングで発音停止      */
        }
        else
        {                               /* スイッチは押されている        */
            g_P_image ^= 0b00011000;    /* 次の出力を反転させる          */

            g_SW_timing--;              /* タイミングをカウント          */
            if ( g_SW_timing == 0 )
            {                           /* スイッチ確認タイミング        */
                g_SW_timing = SW_TIMING;/* SWチェックタイミング初期設定  */

                g_SW_input <<= 1;       /* スイッチ入力の準備            */
                if ( P13_bit.no7 == 1 )
                {
                    g_SW_input++;       /* スイッチオフ状態を設定        */
                }
            }
        }
    }
```

ここで赤い四角で囲んだ部分が次のタイミングでのポートの出力を設定している部分です。変数 g_P_image に対して、0b00011000 との排他的論理和をとることで P04、P03 の状態を反転するようにしています。

その後の青い四角で囲んだ部分が SW(スイッチ)の状態を確認するタイミングを作っている部分です。INTTM01 が発生するごとに変数 g_SW_timing を−1 して、0 になったら、実際の SW(スイッチ)の状態をチェックしています。結果は変数 g_SW_input の LSB(最下位)から取り込まれていきます。

変数 g_SW_input の下位のビットが最新の結果で、上位に移るほど古い結果になります。SW(スイッチ)が離されたときに下位から 3bit が、すべて 1 になったときに、離されたと判断しています。

これ以外に、ポートの設定で音程指定する Tone_sel 関数やそこで使っている Set_TDR_data 関数がありますが、それらについては「RL78 セミナ 2」を参照してください。

3.4　使用している変数の意味

ここで、使用している変数について説明します。デバッグ(プログラムの動作確認)時に、いつでも状態が確認できるように、ほとんどの変数をグローバル変数(大域変数)として定義しています。メモリの使い方としては無駄が多くなりますが、デバッグ効率を優先するためにこのような使い方をしています。

・g_SW_input:SW(スイッチ)の状態の変化を記憶している変数です。初期値は SW(スイッチ)が押されていないことを意味する 0xFF(すべてのビットが1)です。SW(スイッチ)が押されると下位のビットから順に 0 に変化していきます。逆に SW(スイッチ)が離されたときには、0x01→0x03→0x07 と変化し、0x07 になったら、SW(スイッチ)が離されたと判断するようにします。

・g_TM_count:現状では、定義しただけで未使用(将来の拡張用)

・g_SW_timing:タイマ割り込み(INTTM01)の回数をカウントして、SW(スイッチ)の状態をチェックするタイミングを作っています。初期値は SW_TIMING(20)で、割り込みごとに−1し、0 になったら SW(スイッチ)の状態を変数 g_SW_input に取り込んでいきます。

・g_P_image:スピーカーを駆動している P04 と P03 および LED を駆動している P00 に出力するデータを保持しています。初期状態では、0x01 になっています。

3.5　使用している定数の意味

このプログラムでは、2つの定数を定義しています。1 つは、r_cg_userdefine.h で定義している SW_TIMING(20)です。2 つ目が main.c で定義している音程データ(タイマに設定するデータ)の配列(TDR00_DATA[8])です。16pin の RL78/G10 の PWM 信号の周期を指定する定数名をそのまま使ったのでこの名前になっていますが、このデータは TDR01 に設定してインターバルタイマの周期設定に使用しています。