OIDUSラボ:ハード開発課-PICNICの第三回
【公開日】:04/6/26 【著者】:OIDUS・DAVID・DEKURON 【記事コード】:0003 【Ver】:二版

【第三回】:メロディ発生器の作成

本体の写真

第二版。更新しました。更新部分は青色で表示します。

注意を呼びかける内容は赤色で表示します。

皆さん、こんにちは。「PICNIC」第三回です。今回は、前回の音発生器を改良し、音階を鳴らしてみようと思います。

さて、一口に音階と言っても、単に鳴らすだけなら周波数を意識して、方形波を出力するくらいですみますが、時間単位で音階や音量を変化させるのは、プログラムに一工夫も二工夫も必要になってきます。

今回は、メロディデータ(音階と音量設定情報のファイル)をパソコン側で作成しておき、PICのEEPROMに書き込んでおき、PICで読み込んで、それをPIC側で鳴らす方法を取りました。

音階データの入力で考えられる方法は二つ。

1つには、MIDIデータなどの手に入りやすい、また、作成しやすい音階ファイルを変換するかそのまま再生できるプログラムを組む方法。

2つ目は、メロディデータを作成するソフトを作ってしまい、PICのプログラムもそれにあわせる方法。

今回はMIDIの知識が薄いのと、その他のメロディ形式のほとんどが「単音でない」ことから、オリジナルのメロディデータを作る後者の方法をとりました。

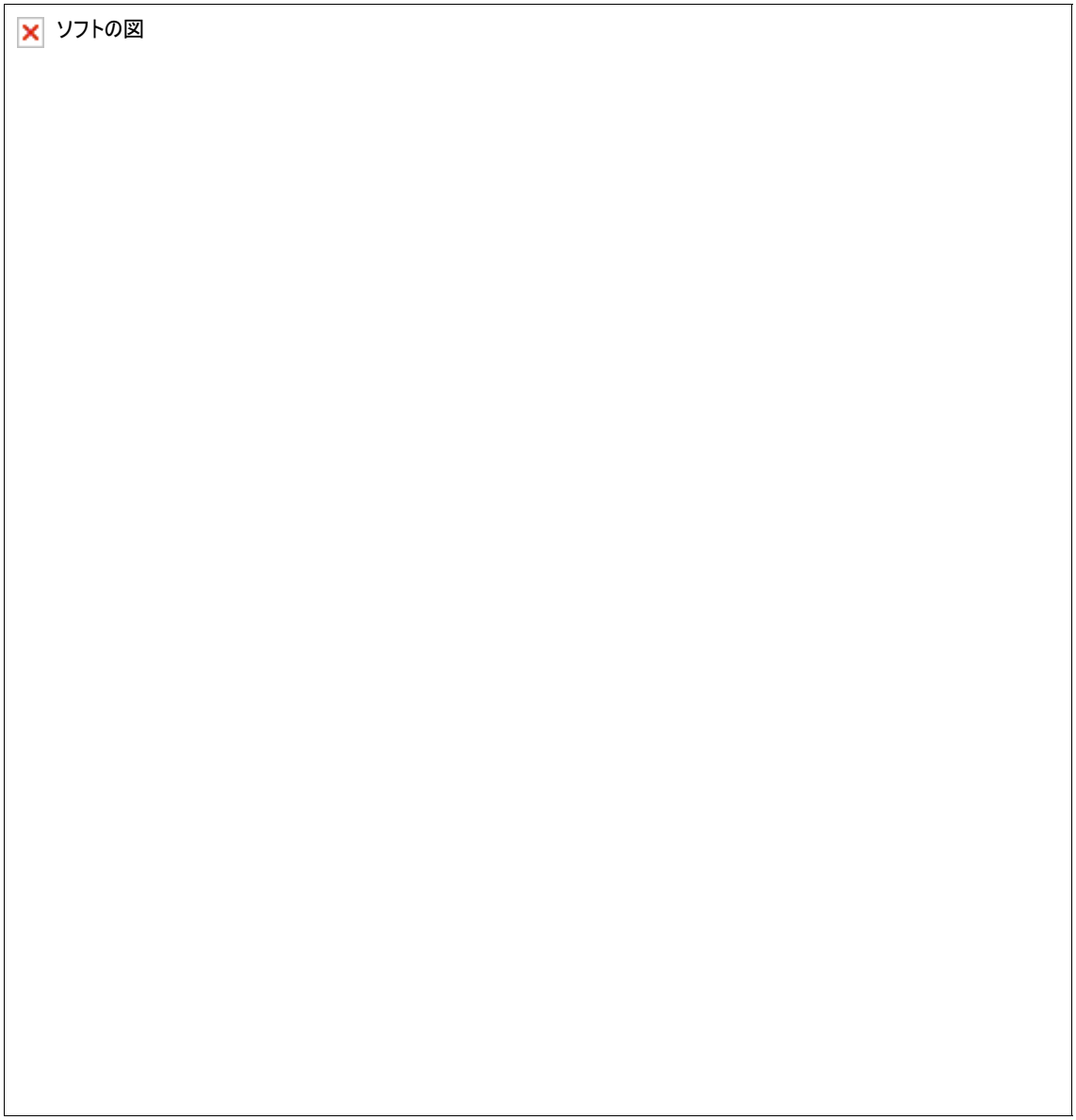
まず、この作成ソフトから作りましょう。

「Visual Basic 6.0」を使い、簡単に入力できるソフトを作りました。音階データと音量データは、まず音量が「4BIT D/A」を使ってる都合上、また、今後8BITにまで拡張できるように、1バイトで1つの音階の音量を表現することにします。一方、音階データは、一オクターブを0～144に区切って、それを1バイト分の幅を用意しました。

音階のほうは、1オクターブあたり、144の分解能がありますので、ピッチ調節なども容易です。ソフトのほうでは、音階に限って、キーボードの上下キーを一回押すごとに、半音階あがるような入力システムも加えました。

とりあえず、ソフトの様子を載せておきましょう。

ソフトの図



上の白いグラフが音量入力、下は音階入力です。保存ボタンを押すとデータを作成します。出力先のフォルダはソフトを起動したフォルダです。

データのビットフィールドを下に示します。

バイト｜1,2,3,4・・・
データ｜音階,音量,音階,音量・・・

上の図のような構造をしています。よって、PIC16F84Aでは64バイトのEEPROMを搭載していますので、半分の32音を記録できるわけです。Cで作るのなら（アセンブラ・他でも実現できますが）、「GETDATA」関数みたいなのを作ってしまえば、もっと記録できるでしょうね。具体的に、音階データ、音量データを、（アドレスを示す）引数の値の区別により、定数値として出力するような関数にします。たとえば、switch(address){}みたいにして、ソースに直接、値を記述してしまえば、プログラムメモリのサイズがあいてる分まで記録できるわけです。

・・・ちなみに、上の図のデータは「かえるの歌」です（笑）しかも、ぴったり収まりました。すごい！

このソフトはプロジェクトごと公開しますので、何かの役に立てて下さい。今回限りのソフトではかわいそうですからね・・・。

プロジェクト:「マウスで波形　EEPROM」　のダウンロード

さて、今度はPICのプログラミングです。

今回鳴らす音階は、いずれも「方形波」にします。方形波とは、SIN波形のSINカーブをパルスのON・OFFで置き換えたものです。かね？あまりいい説明が見つからなかったので、いろんなところでよりよい説明に出会ってください～（苦）

SIN波は、負になる部分があるので、回路上で一方向に出力するためには、SINの最低値を0にする、すなわち、Y方向に1だけ位相をずらす必要があります。その点、方形波は、ON・OFFの周期的な切り替えでこれをある程度近似できてしまうので、プログラムがだいぶ簡単になります。まあ、音は独特のコンピュータ的な音になりますけどね・・・。おまけに、合成波にするほど、SINの合成波とかけ離れてくるので、本当に単音での使用向きと言うわけですかね。

本体の写真2

上の写真は、完成したものです。まず、データをアナログにして、次に圧電スピーカを通過し、最後にGNDにたどり着きます。たったこれだけです。ちなみに、LEDが3本ほど固まって見えますが、あれは出力データの確認用につけたものですので、とってもかまわないですね。しかも、LEDに出力をとられて、音量が下がってしまいますから、音量メータなんかをつけるときは注意が必要です。

回路図は下にあります。

回路図

すこし見にくいでしょうか・・・。PICへの接続は、第一回で作ったボードの接続と同じです。

PORTBだけ違うので、違いのある、囲った部分（前と一緒）の外側だけ書きました。D/Aを通過した電流は、スピーカに伝わり、その後GNDに流れていきます。先ほどの説明と同じです。

肝になるPICのプログラムですが、今回もやはりCで作ってしまいました。慣れてしまうと、恐ろしいほどアセンブラを使わなくなりますね～。やっぱり面倒だし・・・。まあ、タイムクリティカルな物を作ることが少ないため、蔵で眠ってる状態ですけど。裏ではリンクとして働いているわけで、やっぱり縁の下の力持ちなのでしょうか。

さて、mainの関数で、やることはたった二つ、「putOTO」という音発生関数を呼ぶことと、EEPROMから逐一データを読み込むことです。

音量、音階のデータをchar型でそれぞれ、「onryou」「onkai」としました。これにEEPROMからデータを読み込み、代入したものを、putOTOの引数として渡します。プログラムは下です。

```
char eeread(char); //EEPROMからの読み込み関数。引数はアドレス。
void putoto(char,char);//音発生関数。引数は音階、音量の順
void setPORT(); //PORT（TRISの）設定関数。一部asm{}で記述します。

main()
{

    char add; //現在のアドレス
    char onryou; //音量
    char oto; //音階
    char over; //アドレス最大値。何故かCONST定義できない。

    setPORT(); //PORを全て出力に

    over=64; //最大値は64

    while(1)
    {

        add=add+2; //音を次にずらす

        if ( add >= over)
        {
            add=0;
        }

         oto=eeread(add);
         onryou=eeread(add+1);

         putoto(oto,onryou);

    }

}

char eeread(char address)
{

    eeadr=address; //アドレス代入。Cとアセンブラの連携もどき？
```

```
    asm
    {

        BSF         STATUS,RP0
        BSF         EECON1,RD
        BCF         STATUS,RP0
        MOVF     EEDATA,W
    }

    return eedata; //いや～、本当に便利ですね～

}

void setPORT() //そのままです。
{

    asm
    {

        BSF STATUS,RP0
        CLRF TRISA
        CLRF TRISB
        BCF STATUS,RP0
        MOVLW 0FFH

    }

}

void putoto(char sleep,char value)
{

     long i; //ループ制御
     int u;//ウェイトの回数

     long time; //方形波一周期をループして1秒になる回数＝1秒が音階の最小単位とします。

    i=10000; //1s=10000デシミリ秒とします。変な単位・・・。まあ、都合上。
    time=5*sleep;//まあ、これくらいかな？

    while (i>time)
    {

        output_port_b(value); //音階出力＝つまり、方形波の上がりの方

        u=sleep*3; //ループ回数
        while (u>0)
        {
            delay_us(144);//これくらいでOKだった・・・まあ、良しとしましょう・・・。
            u--;
        }

        output_port_b(0); //音階をゼロにする＝つまり方形波の下がりの方
```

```
        u=sleep*3;
        while (u>0)
        {
            delay_us(144);
            u--;
        }

        i=i-time;
    }

}
```

さて、大変なおおまつがいです！（この言い回し、やっぱり危ないと思われます・・・。でもいいや。）

初版で、一点ハのドを作る際、「12分の2かけて」と言いましたが、これが間違いでした。音階の求め方（標準的な）は、オクターブごとに周波数が2倍になり、その間を、12分割しているわけですが、その間と言うのは、2^（12分の～乗）の式で表されるようです。

つまり、12音でべき数が一増えるわけで、そうなると、2^1 倍増えます。これはつまり、2倍のこと。

今回の場合、一点ド ＝ 440Hz × {2^（－12分の9）}＝440 ×（1．6817分の一）≒ 261 Hz

となり、半周期にかかる時間 ＝ （1 ÷ 261 ）÷ 2 ≒ 1.9157ミリ秒になります。

八分音符なら、半分の回数だけループします。つまり、一点ドなら、261回の半分、つまり、約130回ON・OFFを繰り返すと、0.5秒間鳴るわけです。（テンポを1秒間の速さとしたとき、つまり、四分音符で一分間60回。）

第三版は、プログラムも直しますので、どうかお待ちくださいませ。

順調に行っていた様にも思えましたが、最後のputOTOが曲者でした。計算上、方形波の一周期を、標準の音である「一点イ」の440Hzに12分の2かけて一点ハにしたものにすれば、ウェイトの時間は半周期分（半周期なのは、ON・OFFする波形なので、半分ごとにウェイトを挟むのです。そのため、一回のウェイト自体は、半分の時間になります。）で1.91ミリ秒位になり、きちんと「ド」の音が鳴るはずだったのですが、それは鳴らず、一秒間鳴らすためのループも、変数のレンジ（long型は16BIT）を越えてしまい、それができなくなってしまいました。

ならば！と。まあ、平均律でしかあらわさないような、固定した音階でもないので、そこら辺は

「僕の耳が語った」

ということにしましょう。と言うことで、一所懸命、出力音とピアノの音を聞き比べです！（笑）

ピッチは440としましょう。そして、なぜかクロックに同期していないようにも思える途切れ途切れの音と格闘すること20分、おそらく最高に一点ハに最適化されたプログラムになりました。ウェイトの回数の誤差は、マイクロ秒レベルの影響しか与えないので、無視します！

やっと完成。とりあえず音階は鳴りました。いや～、良かったです。

おしまい。

いや、終わらない。

プロジェクト置いときます

プロジェクト:PICのプロジェクト(C言語) のダウンロード

リンク

OIDUSタウンートップ

PICNICートップ