音楽放送とノイズ

一、　音とは何だろう?

多くの章に筆を割いてきたが、本章では異なる主題について語ろうと思う。音とは、多くの場合、その妙なる調べによって人の心を揺り動かす。そしてまた EduCake もまた、その例に漏れない

まず、初めにご理解していただきたいのは、音とは透明性（空気、水等）を持ちながら伝達される一種の波動であり、異なる媒体における速度により、耳に届き、鼓膜を震わし、そうすることで、私たちは音を感知し、その内容を知りうることが可能となるのである。

一般的に、人類の耳が聞き取ることのできる音の周波数は 20Hz～20000Hz（ヘルツ）の間であると言われており、この範囲内を超えたものを超音波と言い、反対にこれを下回るものを超低波音と言う。犬は 40～50000Hz の音を聞き取ることが可能であるため、人には聞き取れない音を彼らが察知し、大声で吠え出すという事が時として起こる。彼らが人の聞き取れない音を感じとことが可能なことは、超音波モジュールを用いた、簡単なテストで知ることが出来る。その中の Hz は一種の音の周波数の単位であり、例えば、下の図 1 のようなものであると言える。

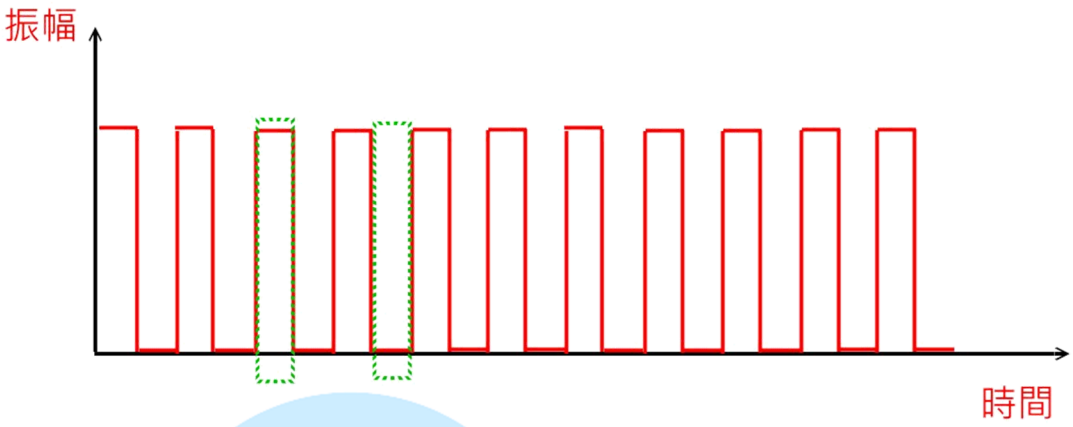

図 1. 簡単な音の波動図

図中の音は、簡単な高低を表した図表であり、やや、先程述べた PWM に似ている。個の図の波動は 2 種類だけであり、高いものと低いもののみを表している。これに限らず、全て第一次震動は、震動の振れ幅の高低を表すのみであり、つまり、周波数の定義とは次のようになる：一秒間に発生する幾度かのこの様な波動は、EX: 300Hz の時、一秒間に 300 回震動したことを示している。波動がより高く、震動の回数が多いほど、私達の耳に届く音はより鋭いものとなり、反対に、波動が低くなるほど、その音は重く低いものとなってゆく。これはあたかも、私達が水面を叩く様に似ている。それが軽ければ、水は優しく揺れ動き、強ければ、水の動きもまた激しいものとなる。

次に述べるのは、音色（音の特性）についてである。例えば、図 2 は、同じ音の高さと強度下において、音が伝わっていく様を表しているが、異なる楽器や異なる人間によって、その周波数が同一のものであっても、それが私たちの耳に届くころには、そのさまざまな素材の特性によって、異なるものとして伝わってゆくのである。これはあたかも歌の様で、Do、Re、Me、Fa、So…と歌った時、男女が混声合唱をした場合、その中には男性特有の声の低さと、また、女性の特有の声の高さが垣間見られる。この違いははっきりと明らかであり、各種のチェロ、フルート、ギター、ピアノ等の楽器が、コンチェルトの時、それぞれが異なるメロディーとして認識できるのも、同じ原理である。それぞれがことなる特性を持ち、それを活かしながら、妙なる調べとなり、私たちの耳に届くのだ。

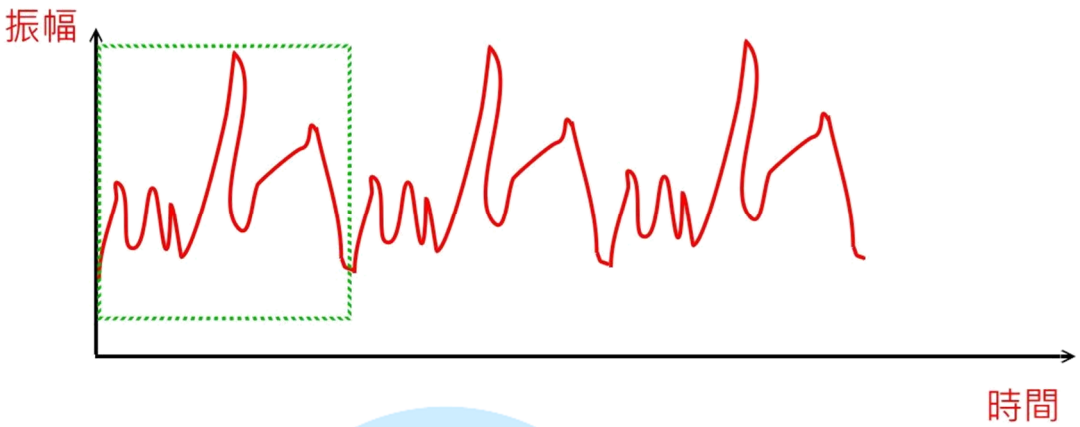

図 2 音の特色

周波数、震動、音色を持つことにより、やっと私たちはその美しい調べを楽しむことが出来るのだ。その振幅をコントロールすることは、また、その音量の大小(これをすなわち、PWM と言う)のことであり、音の発生の変化の度合いを模している。そして変化の速度は、人の耳もそれを察知することが可能である。それはあたかもパトカーや救急車が、近づいたり接近したときに耳に入る音の感覚と一緒である。その他のものについては、ここでは、音量の変化に伴う周波数の変化について紙面を割きたいので、ここではやめておきたい。

ここでは、これで一旦筆を置き、後の章で、また、改めて詳しく読者のかたがたへご説明したいと思う。

二、　音の発生

音の特性を理解したら、EduCake の内部機器を使って、実際に音を発生させてみよう。まず、tone( )、notone( )の二つの指令を出すこと。

tone( )は、digital pin 上に、周波数の方形の波（作業周期が 50%の PWM シグナル）が、異なる周波数の音を擬することにより、音の時間あるいは noTone()と呼ばれる状態を持続することを指定することを可能にする。このピンには通常のブザー等に接続されており、音を発生させることが可能となっており、音のクオリティは大変高い。また、同一の時間で発生する音についてであるが、もし、ある一つの pin の上で音を流した場合、異なる pin で tone() を出すことは無効である為、もし多数の pin で異なる音を発生させたい場合は、先にまず noTone()を使用して pin をオフ状態にし、再度 tone( )を用いて別の pin で音を流す必要がある。この動作により、音は左右のスピーカーから流れだすようになるため、大変重要である。tone( )の使用方法は、主に下記に挙げる二つである。

まず回路を図 3 の様に接続すること:

tone(pin, frequency)

tone(pin, frequency, duration)

pin: 音を発する pin の指定

frequency: 音の周波数(Hz), unsigned int, 人の聞き取れる 20~20000 の範囲内であることに注意すること

duration: リズムの持続時間, 単位は秒, unsigned long 型別（この関数は特に必要ない）

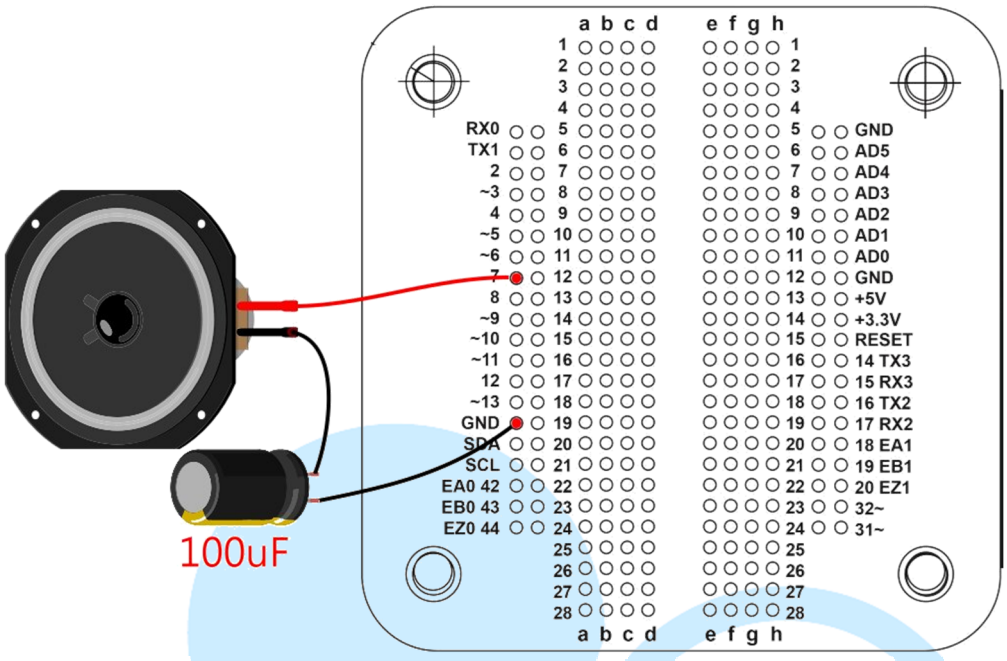

図 3. 単独スピーカーの接続図

回路中の 100uF 的電界電融率は、つまりはフィルター効果によるものであり，人によっては直接電気抵抗を利用するだけでもよく，主にスピーカーの電気抵抗が低すぎれば、制御盤の電流は大きくなりすぎてしまうが，しかしながらその音量を僅かながら下げれば，組み合わせた電融の直列配属や，または使用するスピーカーの大小，制御方法等々により，大きく変わってくるため、ここは読者自身にぜひ、色々試していただきたい。きっと様々な効果がえられるはずだ。もしスピーカーに異なる効果を与えれば、音の大小にも影響してくるだろうし、最新型のスピーカーを使用したにしても、その効果が変わってくるのは当然である。また、スピーカーには pin 7 を接続し、tone()からの指令が pin 7 に達すると、音が発生し、またその制御をすることが可能となる。この様な種々様々なことが、この回路図で可能となるのだ。簡単なテストプログラミングコードを、下記に記しておく。

```
    void setup()
    {   }
    void loop()
    {
      tone(7, 500); // pin7 に 500Hz の音を発生させる
      delay(500); // この delay 等は 5 秒の音を発生させる指令を出す
      noTone(7); // pin 7 の音のスイッチ
      delay(500);
}
```

もし、二つのスピーカーのエミュレーションを左右して音の音量に変更を加える場合は、回路を変更する必要が出てくる。電圧を変更して音量を調節し、回路を図 4 の様に変更すること。

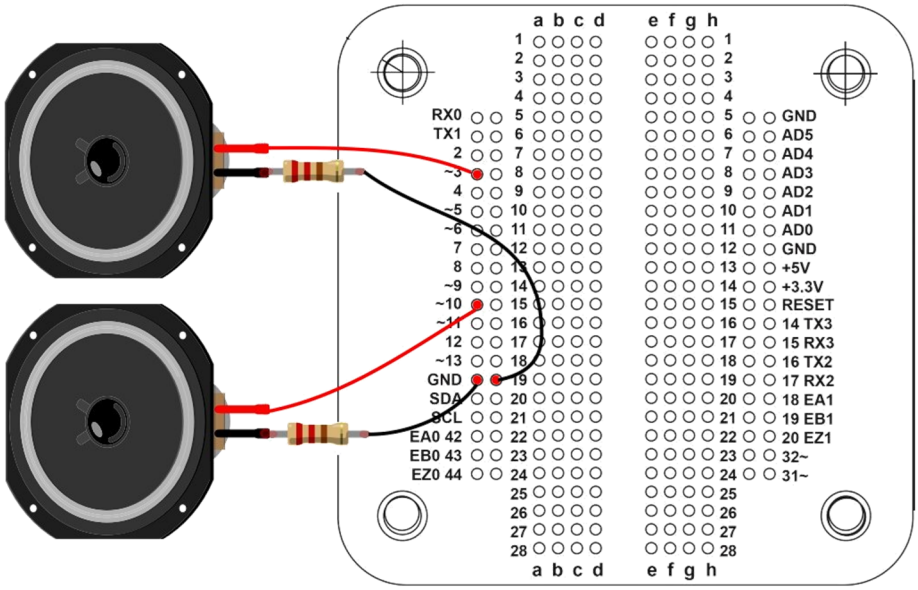

図 4 二つのスピーカーの接続図

プログラミングは下図の通りにすること。

```
    int left_sound=10;
    int right_sound=3;
    void setup()
    {   }
    void loop()
```

```
    {
      tone(left_sound, 500); // 500Hz の音を発生する
      delay(500); // この delay 等により 0.5 秒の音を発生させる
      noTone(left_sound); // 左側のスピーカのスイッチを切断
      tone(right_sound, 500); // 右側のスピーカーを開き 500Hz の音を
発生させる
      delay(500); // この delay 等により 0.5 秒の音を発生させる
      noTone(right_sound); //右側のスピーカのスイッチを切断
    }
```

電圧と変電圧についてであるが、もし前章で述べた serial を使う場合は、音量の調整が可能である。使用するパソコンの UI +マウスをへんこうするだけで、音量の調整が可能となるのだ。応用方法は非常にたくさんある。

読者自身にもぜひ、自分自身でぜひ色々と試して頂きたい。

三、　音符とリズム

続いて音楽放送についてお話ししようと思う。まず先に音符とリズムについて理解して頂きたい。リズムとは簡単にお話すれば、音の長短速度であり、通常一分間に 180 回である。という事はつまり一分間に 180 個の音符を流すことが可能であるという事である。その為、速度を速めたり弱めたりすることは、先に述べた tone と notone と delay()を使用することにより自由にコントロールすることが可能である。

音符はリズムに加えやや煩雑で、一般にピアノの鍵盤の白い部分（黒い部分はとりあえず置いておくとする）は、簡単に分けると低音（Do、Re、Me、Fa、So、La、Se)、中音、高音，各七個計 21 音に分けることが出来る。すなわち:

低音部分：262、294、330、349、392、440、494(単位 Hz)

中音部分：523、587、659、698、784、880、988

高音部分：1046、1175、1318、1397、1568、1760、1976

これら音符の周波数はウィキペディアで調べることが可能である。

(http://zh.wikipedia.org/wiki/音符)

音符は大変複雑であるが、ここでは簡潔に述べられており、非常に興味深い。興味を持った読者は、これを機に、ぜひ、色々と調べてみることをお勧めしたい。

音符の周波数表が分かったら、プログラミングについて再びお話ししよう。まず、先に低音の Do～Se をご覧いただきたい:

```
int frequency[]={
  262,294,330,349,392,440,494}; // 低音部の 7 つの音を整列させる

void setup()
{
}
void loop()
{
  int a;
  for (a=0;a<7;a++)
  {
```

```
        tone(7, frequency[a]); // pin7 より音を発生させる
        delay(500); // 0.5 秒ごとに音を発生させる
        noTone(7);
      }
    }
```

この様に低音の Do~Se を放送することが出来るようになったら，大小のスピーカーを用いて，音を聴き比べてみよう。

もし、高音から低音までの一連の変化を聞き比べてみたいという読者がいたら、frequency を下記の様に変更すること。

その後 for ループを下図の様に変更する。

```
int frequency[]={
  262, 294, 330, 349, 392, 440, 494,
  523, 587, 659, 698, 784, 880, 988,
  1046, 1175, 1318, 1397, 1568, 1760, 1976};
for (a=0;a<7;a++)  を  for (a=0;a<21;a++)に変更
```

この様にして、私たちは高音から低音までの標準的な音符の周波数を知ることができることになる。これにより、EduCake に歌を歌わせることが、ようやく可能となったのである。

四、　ループの高低及びドップラー効果、弦波音

符の放送の仕方が分かったら、その効果について調べてみよう。

連続、断続、高低などによる様々な効果により、それらは数学的な曲線を描く。

中々興味深い分野であるので、興味のある読者は、ぜひ、調べていただきたい。

先に述べた 20～20000Hz の間で、私たちはその音色を聞き取ることが出きる。

```
void setup()
{
}
void loop()
{
  int a;
  for(a=20;a<20000;a+=10) // 周波数 20~20000, 各 10Hz
  {
    tone(7,a);
    delay(5);
  }
  noTone(7);
}
```

筆者が試したところ、通常のパソコン用の 6cm の小型スピーカーで、

大体 12000Hz 以上の音を発すると、聞き取りづらくなる。おそらくこれは、

音が鋭くなりすぎているのが原因であると思われる。この様なスピーカー

と音の相関関係も、興味のある方は、試してみると、非常に面白いと思う。

```
void setup()
{
}
void loop()
{
    tone(7 ,900); // 900 に調整しノイズを無くす
    delay(200); // ノイズ音制御
    noTone(7);
    delay(300);
}
```

音の変化と、内部の機器の起動音等の違いにお気づきになられただろうか。ブーンという独特の音が、始まりの合図だ。

音の変化は一定の周波率に基づいている。興味のある方は、その違いを、色々と試してみるのも、これまで通り、楽しいものに違いない。

時間が有るなら、ぜひやって欲しいと筆者は思う。

測試結果可以聽出來這聲音還滿好玩的，而程式碼裡面的 tone(7,s)；這裡面的頻率如果替換成 tone(7,3500-s)；，還會有不一樣的有趣效果，本來由高到低的滑音會變成由低到高，調整程式碼裡面的 V+=10 那個代表加速度的數字也會讓聲音變化率有特別的效果，當然，delay()裡面的數字也可以調整看看，都很有趣。這裡若加入使用三角函數的成分，還會有不一樣的效果:

```
void setup()
{
}
void loop()
{
  int a;
  double v=0,s=3500; // 3500m から落下したと仮定する。初期速度０，重力加速度 10
  while (s>50) // 50m高度に達すると停止
  {
    tone(7,s); // 現在の周波率通知
    delay(30);
    noTone(7);
    s-=v; // V の距離で停止
    v+=10; // V 時の重力加速度時に 10 増加する
  }
}
```

音が滑らかになったのにお気づきだろうか？面白いだろう?異なる弦波音の周波率を例えば cos()、tan()等にすると，その度に異なる変化が見られることと思う。日々の我々の生活の中で耳にする、救急車やパトカーのドッ

プラー効果音の仕組みも、実は、この様な原理を利用して、構成されているのである。

```
int spd =10; // 音の発生時の速度を遅くさせる
void setup()
{
}
void loop()
{
  int a;
  for(a=420;a<1300;a+= spd)
  {
    tone(7,a);
    delay(20);
  }
  for(a=1300;a>500;a-= spd)
  {
    tone(7,a);
    delay(30);
  }
  noTone(7);
}
```

上記プログラミングの中の delay の数字を若干調整してみよう。そうすれば、パトカーや救急車の様なドップラー効果音を発生させ、それを体感することが出来るようになる。

しかしながら、通常、私たちが実際耳にするものとは若干違いが有る。なぜなら、救急車やパトカーはドップラー効果音を発生する以外に、一定の異なる速度によって、私達から遠ざかって行くからである。その為、私たちが今回、疑似的に発生させる音と、私たちが日常生活の中で耳にするドップラー効果音を聞き比べてみた場合、読者が感じる印象や感覚が、完全に一致するとは言えない。これは最初に断わっておく。

この実験をするに当たり、もっと詳しく知りたい読者がいたら、下記のサイトを参照願いたい。

http://zh.wikipedia.org/wiki/ドップラー効果音

以下のプログラミングでランダムに発生させる音が与える効果は、映画に登場するロボット達があたかも会話しているかの様な印象を我々に与える。

```
void setup()
{
  randomSeed(analogRead(0));
}

void loop()
{
  tone(7, random(100,2000));
  delay(100);
  noTone(7);
}
```

上記に挙げた random(100,2000)は 100～1999 の間にランダムに音を発生する。この二つの数値を変更してみよう。また異なる効果を我々に与えるはずだ。異なる効果、異なる音、異なる印象に異なる周波数。どうだろう？読者諸氏も、きっと、美しい音の世界にすっかり魅せられてしまったのではないだろうか？

五、　音楽を実際に流し、五線譜を調べてみよう

これまでに述べた様々な事を、今度は、実際に、 EduCake に音を一曲歌わせることで、実際に体験してみよう。まず，五線譜を一つ取り出して，GOOGLE を利用して、その五線譜を使用するに当たり、様々な版権上の規制や禁止が無いか確認しよう。もし、その歌がそれに該当するのであれば、使用をやめること。もし、そうだったとしても、気にすることはない。歌は、この様な版権上の規制が、非常に多いのだ。

その点、童謡には、この様な規制が少ない。そこで私たちは今回、下記に挙げる童謡を使用したいと思う。この歌は非常に有名なので、読者も、きっと耳にしたことが有ると思う。使用する五線譜は、図 5 の通りである。

今回は学生時代の音楽の授業ではないので、詳しく五線譜を読み込む必要はない。ただ、一度ざっと目を通して下されば充分である。

”二匹のトラ”の歌詞の上部には 1231 を配する。これら 1234567 の数字は五線譜上の Do、Re、Me、Fa、So、La、Se に相当する。これらの数字を配することで、私たちは二匹のトラという童謡を楽しむことが出来るようになるのだ。もう一つ、大事なのは KEY の上げ下げである。先に挙げた Do~Se をただ配するだけでは、音の高低は生まれない。その為、音楽を味わうには、私たちはまた再度上部に 565431 を加える必要がある。この中の 5654 は五線譜上における低音を表す。

そして、もう一つ大切なものは音のリズムだ。これら、上記に挙げた二つ、音符とその音の高低、リズムの三つが絶妙に組み合わさることによって、私たちは初めて、音楽と言う音の生み出す、その妙なる調べを深く味わうことが出来るのだ。

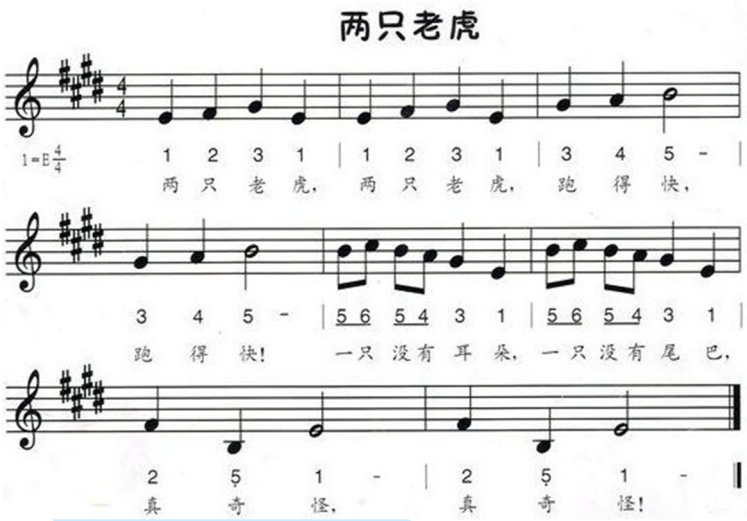

图 5. 二匹のトラの五線譜

五線譜上の音符を、下記に挙げるように、それに適応するプログラミングに変換して行こう。

五線譜上の歌詞に、必要なプログラミングを当てはめることが出来たなら、次は、リズム等を加えていく。

必要なプログラミングは、下記の通りだ。ご参照願いたい。

```
    byte tigerTone[]= //音符に必要な周波数を記録
    {7, 8, 9, 7, 7, 8, 9, 7, 9, 10, 11,  9, 10, 11, 18, 19, 18, 17, 9, 7,
18, 19, 18, 17, 9, 7, 8, 4, 7, 8, 4, 7};

    byte tigerBeat[]=// 音の長短、一分間に必要とされる音のリズムの長短を制御
    {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 2,  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
2, 1, 1, 2};
```

先に完成した frequency[]の配列の高低部分に，合計 21 個の数値がおかれて有る為，ここでの tigerTone 陣配列では番号に沿って数字を配列していくだけでよい。その中の 0～6 は低音 Do～Se に、7～13 は中音の Do～Se に、14～20 は高音の Do～Se に相当し，その組み合わせのプログラミングは下記の通り：

```
    const int speaker=13; // スピーカーの 13 ピンへの出力を改める，これは後の作業の為である
    int frequency[]={  // 低中音域の設定
      262, 294, 330, 349, 392, 440, 494,
```

```
        523, 587, 659, 698, 784, 880, 988,
        1046, 1175, 1318, 1397, 1568, 1760, 1976};
    byte tigerTone[]=
    {7, 8, 9, 7, 7, 8, 9, 7, 9, 10, 11,  9, 10, 11, 18, 19, 18, 17, 9, 7,
18, 19, 18, 17, 9, 7, 8, 4, 1, 8, 4, 1};
    byte tigerBeat[]=
    {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 2,  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
2, 1, 1, 2};
    const int playLen=sizeof(tigerTone); //放送音の長短を計算する,
実際の音の長さに注意すること

    void setup()
    {
    }
    void loop()
    {
      int a;
      for(a=0;a< playLen -1;a++)
      {
       tone(speaker,frequency[tigerTone[a]]); //あらかじめ設定され
た周波数にのっとって放送
       delay(300* tigerBeat[a]); // 描く音のずれ, 長さを計算
       //おおよそ一秒後子に 3 個の数字, 毎分 180 個
       noTone(speaker); //スイッチ音である, 音符を一つ加えること
      }
      delay(3000);
    }
```

この様にして音の配列が完成したら、一度音を流してみよう。それが終わったら，最後の delay 指令が出てから三秒後、音を流してみよう。もし、これらの過程がすべて完了すれば， SD カードを利用して、大量の音楽を保存することも可能となってくるのだ。ボタンをクリックして好きな音楽を選択して流し，液晶スクリーンで画像を見る。そんな楽しみ方もできるのだ。

どうだろうか。続く後章では、より面白く、興味深い様々な楽しみ方を、色々と紹介していきたいと思う。

六、　複数音楽同時放送

先に歌の概念について述べたが、ここでは簡単な見本を実際に制作してみよう。まず、いくつかの歌を選び、その五線譜に相応するキーを、4*4 のキーボードを用いて当てはめていこう。

様々な歌が有るが、ここでは私たちは以下の四つの歌を用いて実作してみようと思う。それとはつまり：二匹のトラ、キラキラ星、小さなミツバチ、小さなロバと言う、大変有名な四つの童謡である。

最初に、それぞれの歌の周波数及びリズムを構成しよう。

```
    int frequency[]={  // 中高音のリズムを先に設定
      262, 294, 330, 349, 392, 440, 494,
      523, 587, 659, 698, 784, 880, 988,
      1046, 1175, 1318, 1397, 1568, 1760, 1976};

    byte tigerTone[]={7, 8, 9, 7, 7, 8, 9, 7, 9, 10, 11,
9, 10, 11, 18, 19, 18, 17, 9, 7,  18, 19, 18,
    17, 9, 7, 8, 4, 1, 8, 4, 1};// 二匹のトラ
    byte tigerBeat[]={1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 2,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,  2, 1, 1, 2};
    int  tigerLen =sizeof(tigerTone) ;  // この歌の長さを算出する

    byte
beeTone[]={ 11, 9, 9, 10, 8, 8, 7, 8, 9, 10, 11, 11, 11, 11, 9, 9, 10, 8, 8, 7, 9, 11, 1
1, 9, 8, 8, 8, 8, 8, 9,
    10, 9, 9, 9, 9, 9, 10, 11, 11, 9, 9, 10, 8, 8, 7, 9, 11, 11, 7};  // 小さなミツ
バチ
    byte
beeBeat[]={ 1, 1, 2, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 1, 1, 2, 1, 1, 2, 1, 1, 1, 1, 4, 1, 1, 1,
1, 1, 1, 2, 1, 1,
    1, 1, 1, 1, 2, 1, 1, 2, 1, 1, 2, 1, 1, 1, 1, 4};
    int beeLen=sizeof(beeTone) ;



    byte
starTone[]={7, 7, 11, 11, 12, 12, 11, 10, 10, 9, 9, 8, 8, 7, 11, 11, 10, 10, 9, 9, 8, 1
1, 11, 10,
    10, 9, 9, 8, 7, 7, 11, 11, 12, 12, 11, 10, 10, 9, 9, 8, 8, 7}; //キラキラ星
    byte
starBeat[]={1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1,
2, 1, 1, 1, 1,
```

```
    1,1,2,1,1,1,1,1,1,2};
    int starLen=sizeof(starTone) ;

    byte
donTone[]={7,7,7,9,11,11,11,11,12,12,12,13,11,10,10,12,12,9,9,9,9,
8,8,8,8,
    11,11,7,7,7,9,11,11,11,11,12,12,12,13,11,10,10,10,12,9,9,9,9,9
,8,8,8,9,7};//小さなロバ
    byte
donBeat[]={ 1,1,1,1,1,1,1,1,1,1,1,1,2,1,1,1,1,1,1,1,1,1,1,1,2,1,
1,1,1,1,1,1,1,
    1,1,1,1,1,2,1,1,1,1,1,1,1,1,1,1,1,1,1,4};
    int donLen=sizeof(donTone) ;
```

これらのプログラミングが無事完了したら、それらのデータを SD カードの中に保存しておこう。SD カードの中には非常にたくさんの歌曲を保存することができ、非常に便利である。

私たちは今、四曲の童謡をプログラミングしたが、同様の方法を用い、様々な歌のデータを入力していくことができるので、ぜひ、試してみていただきたい。

まず、1 に二匹のトラを、2 に小さなミツバチを、3 にキラキラ星... といった具合に次々に設定していく。

しかしながら、各歌の長短が異なる為、キーボードを押してすぐ音楽が流れるようにするには、一時停止の機能を用い、ワンクッションおいてから、各歌が流れるように設定をした方がよい。以下に挙げるのは、その際のプログラミング方法である。

```
    // このように設定してもし聞き取りづらくなった場合は、設定を中止
すること
      if (bb_state[0]==0) // 第一番目のキーを、第一番目の曲に設定す
る
      {
        Serial.println(beeLen); // 関連する簡単なデータをモニター上
に映す
        play_no=1;    // 第 1 番目の曲放送設定
        play_pos=0;   // 位置を 0 にし、頭から放送するように設定
```

```
          is_play=1;  // 1 を放送, 0 を非放送に設定
          Serial.println("bee playing...");
        }
        else if (bb_state[1]==0) // 第 2 番目のキーを, 第 2 番目の歌に
設定
        {
          Serial.println(starLen);
          play_no=2;   // 第 2 番目の歌の放送設定
          play_pos=0;  //位置を 0 にし、頭から放送するように設定
          is_play=1;  // 1 を放送, 0 を非放送に設定
          Serial.println("star playing...");
        }
        else if (bb_state[2]==0) //第 3 番目のキーを, 第 3 番目の歌に設
定
        {
          Serial.println(tigerLen);
          play_no=3;   //第 3 番目の曲放送設定
          play_pos=0;  //位置を 0 にし、頭から放送するように設定
          is_play=1;  //  1 を放送, 0 を非放送に設定
          Serial.println("tiger playing...");
        }
        else if (bb_state[3]==0) //第 4 番目のキーを, 第 4 番目の歌に設
定
        {
          Serial.println(donLen);
          play_no=4;   //第 4 番目の曲放送設定
          play_pos=0;  //位置を 0 にし、頭から放送するように設定
          is_play=1;  //  1 を放送, 0 を非放送に設定
          Serial.println("don playing...");
        }
      playSong(); // 歌曲の放送関数
        delay(5);
      }

      // この関数は単音の放送を処理し, 各音符は一定の周波数と放送持続
時間を持つ
      void play (byte toneNo, byte  beatNo)
      {
        tone(speaker,frequency[toneNo]);
        delay(300* beatNo);
        noTone(speaker);
      }

      // この関数は主に放送にか関わる工程及びその問題を処理する
```

```
    void playSong()
    {
      if (is_play==1) //  1 を放送, 0 を非放送に設定
      {
        switch(play_no)  // play_no の内容に対し、どの歌曲を放送するか決定
        {
        case 1://beeTone
          if  (play_pos>=beeLen) // 目下放送している歌曲の位置が最後まで来ているかどうかを判断

          {
            is_play=0; // 歌曲がもし最後まで来ていたら、停止する
            return ; //関数から離れる
          }まだ完了していなかったら、目前の音符を放送する
          play (beeTone[play_pos],beeBeat[play_pos]);
          play_pos++; //その後放送位置に+1 をし、続く音符の位置を指定
         break;
        case 2://starTone
          if  (play_pos>=starLen)
    {
            is_play=0;
            return ;
          }
          play (starTone[play_pos],starBeat[play_pos]);
          play_pos++;
          break;
        case 3://tigerTone
          if  (play_pos>=tigerLen)
          {
            is_play=0;
            return ;
          }
          play (tigerTone[play_pos],tigerBeat[play_pos]);
          play_pos++;
          break;
        case 4://donTone
          if  (play_pos>=donLen)
          {
            is_play=0;
            return ;
          }
          play (donTone[play_pos],donBeat[play_pos]);
```

```
            play_pos++;
            break;
          }
        }
      }
```

このようにして 1～4 のキーを設定してゆき、各童謡の設定を完了させよう。各歌曲の放送が完了すると自動停止する。放送中、随時その他の、目下放送していない歌曲と、目下放送している歌曲を入れ替えることが可能だ。どうだ、とても興味深いだろう？

ここでは異なる方式を試みてみたいと思う。すなわち、歌曲の種類と選択方式を多様に変化させるという事である。例えば、赤外線を用いてリモコンをコントロールするように、ここでは赤外線を応用して用いたいと思う。

また、連続旋回する旋盤コードを用いて、車の様な音を発生させることも可能である。

その他、プログラミングに修正を加えることで、放送する楽曲の順列を変更したり、自動リピートや、ランダム放送などなど... と言った多種多様な機能を用い、私たちは、より深く、音楽を楽しむことが可能なのだ。

七、　速度制御、混音、音量調整、KEY 升降等

たくさんの歌曲を入れることが出来たら、特殊効果を加え、放送機器をより興味深いものにしてみよう。

一般的なパブもしくはコンサートにおいて、鍵盤を持った人もしくはミキサーが、とても複雑な、上部にボタン等が付いた機器を用いて、人々を容易に、歌曲の中若しくはダンスミュージックの中に招き入れるが、またそこに動物の鳴き声や物のぶつかる音、金属音、ロックミュージックなどの効果音を加えることもある。

たとえ、その原理がどれも一様で、音楽が先に述べた音符の構成のようであったとしても、もし、私たちがミックスしたい効果音が音符で構成されていたとし、目下放送中の音楽に加えてみるとしたら、それも、特殊効果音という事ができるのだ。

混入方法は、直接音楽と特殊効果音の周波率を結合する場合には、その増加除法は 2 である；もしくは両者に一定の負荷を加えたのちに増加させる場合、例えば、音楽が全体の 7 割を占め、特殊効果音が 3 割を占めるといった場合においては、特殊効果音を設定の上、音楽に混入することが可能である。しかしながら私たちの先例においては、このやり方はあまり良いやり方とは言えない。なぜなら、解析度があまりにも低いので、先に挙げた先例では一秒間に三個程度の音しか放送することが出来ないからだ。3Hz/sec の周波数を得てから、一般的な音楽は 22KHz/sec 以上であるが、カメラの解析度が高くなればそれだけ画質が向上するのと同じように、取得した周波数が高くなれば、それだけ音質も向上してゆく。ただし、複雑なオーディオ処理や音楽取得問題に話が及ぶとなると、話が複雑となってしまうので、ここでｈは一旦置いておくこととして、まず先に、簡単な実作処理についてお話することとする。

まず最初に速度制御を行う。`void play (byte toneNo, byte beatNo)` という関数から取り掛かるとする。その中の beatNo は音楽の放送速度制御に関わっており、この数字の大きさにより、音の速度が変わる。もし、非常にアップテンポの二匹のトラや、非常にスローテンポの小さなロバを放送した

いとしたら、実際、この様に思うことは非常に興味深いことだが、その変数を play_spd に通知し、制御すれば可能である。

次に述べるのは混音についてである。取得する周波数が低ければその効果も高くはないと先に述べたが、私たちはここで異なる方式を用いて、特殊効果音を改作し、高音で弾がはねるような音と低音で弾がはねる音を作り、利用者がキーを押せば直接挿入し放送することが可能としたいと思う。

弾の跳ねる音のやり方は、先に述べた救急車の音のやり方と同様であるが、また、周波数を低周波で 200Hz 程度にし、高周波で 1200Hz 程度にすると、非常に良い効果が得られる。

音量部分を調節することは、つまり出力効率のコントロールであるといえる。しかしながら、ここでは直接にピンから出力しているので、その効率は低く、音量の調節も容易なものではない。

もし本当にそう処理したいのであれば、オーディオ効率を選んで大きなチップを使用することで、変電圧機器が簡単に音量を処理することが可能となる。或いは既成のオーディオモジュールを用いることも可能である。その場合は図 6 の様になる。

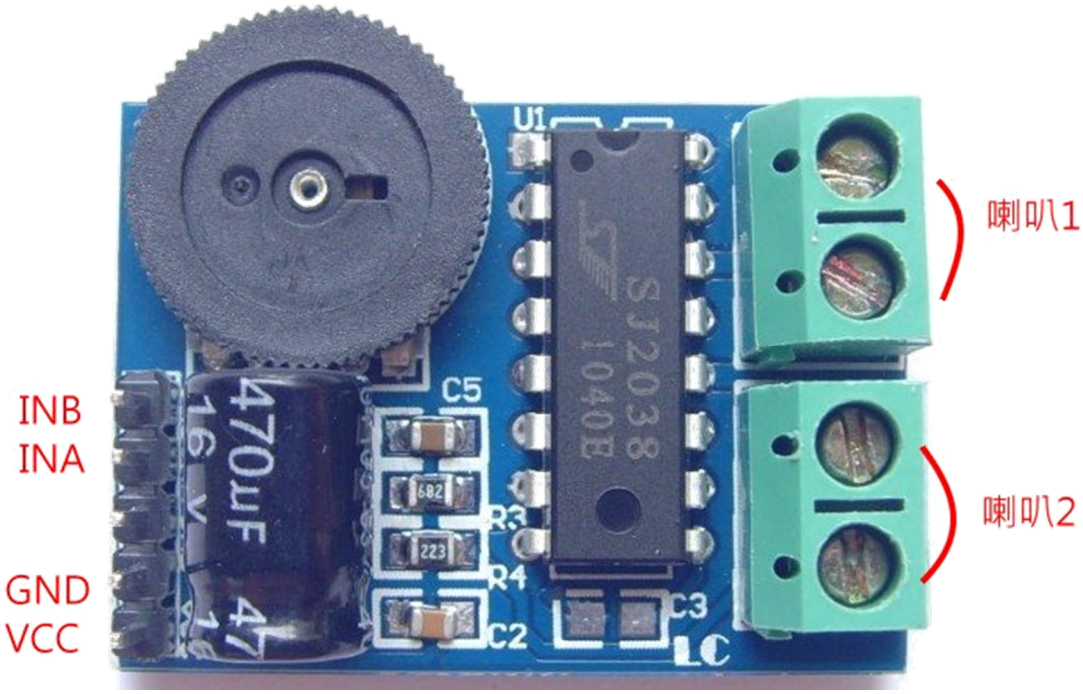

図 6. オーディオにモジュールを搭載した図

放送するキーの上げ下げを行いたいなら、別途変数を通知して制御する必要がある。ここでは二つの方法を紹介するので、ご参考願いたい。一つ目は直接周波数を内部数値の上がり下がりや増加減少から求める方法である。明らかな変化が有ったとしても、問題の数値 0、これは索引の値が達すべき周波数における最低数値の週は率に対し、調整の余地が無い、或いは調整不能の周波率であることを表す。同様に 20 の場合、到達すべき周波率内において、最も高い周波率に達しており、これ以上高くすることができないことを示している。

もう一つの方法は直接変数 play_keys を通知し処理を行う方法である。この変数はあらかじめ 0 と設定しておいてもよく、周波数が変化していないことを表している。しかしながら、この変数は+-300～500 の間において或いはより大きな範囲内において、変更することが可能である。その後、周波率を放送する際、直接この変数を加えてゆくことで、直接、後に続く各音符を、周波数の上がり下がりに合わせることが可能である。その回路図は図 7 の通り。

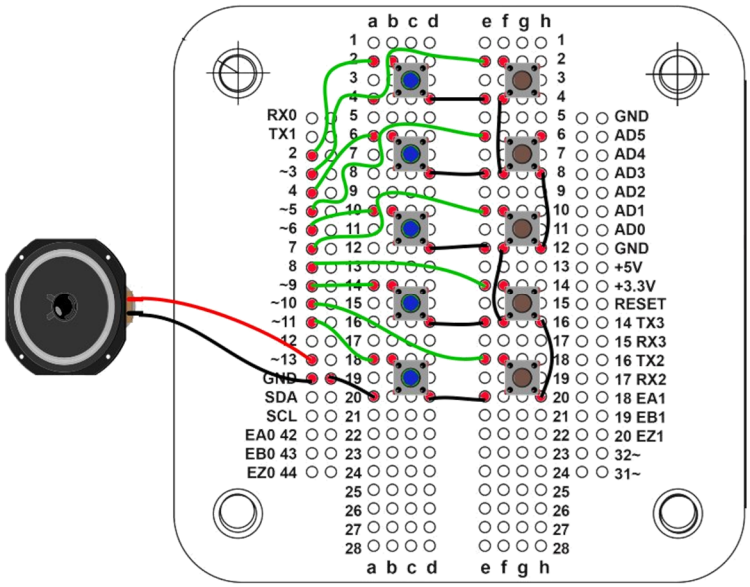

図 7. 回路図

コードは以下の通り。

```
    int speaker_pin=13;

    int frequency[]={  //低中高音のオーディオを先にフォームの設定を
行う
```

```
        262, 294, 330, 349, 392, 440, 494,
        523, 587, 659, 698, 784, 880, 988,
        1046, 1175, 1318, 1397, 1568, 1760, 1976};

    byte tigerTone[]={7, 8, 9, 7, 7, 8, 9, 7, 9, 10, 11,
9, 10, 11, 18, 19, 18, 17, 9, 7,  18, 19, 18,
    17, 9, 7, 8, 4, 1, 8, 4, 1};// 二匹のトラ
    byte tigerBeat[]={1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 2,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,  2, 1, 1, 2};
    int  tigerLen =sizeof(tigerTone) ;  // 続いてこの歌の長さを算出
する
    byte beeTone[]={
    11, 9, 9, 10, 8, 8, 7, 8, 9, 10, 11, 11, 11, 11, 9, 9, 10, 8, 8, 7, 9, 11, 11, 9, 8, 8, 8
, 8, 8, 9,
    10, 9, 9, 9, 9, 9, 10, 11, 11, 9, 9, 10, 8, 8, 7, 9, 11, 11, 7};  // 小さなミツバ
チ


    byte
beeBeat[]={ 1, 1, 2, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 1, 1, 2, 1, 1, 2, 1, 1, 1, 1, 4, 1, 1, 1, 1,
1, 1, 2, 1, 1,
    1, 1, 1, 1, 2, 1, 1, 2, 1, 1, 2, 1, 1, 1, 1, 4};
    int beeLen=sizeof(beeTone) ;

    byte
starTone[]={7, 7, 11, 11, 12, 12, 11, 10, 10, 9, 9, 8, 8, 7, 11, 11, 10, 10, 9, 9, 8, 11,
11, 10,
    10, 9, 9, 8, 7, 7, 11, 11, 12, 12, 11, 10, 10, 9, 9, 8, 8, 7}; //キラキラ星
    byte
starBeat[]={1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2,
1, 1, 1, 1,
    1, 1, 2, 1, 1, 1, 1, 1, 1, 2};
    int starLen=sizeof(starTone) ;

    byte
donTone[]={7, 7, 7, 9, 11, 11, 11, 11, 12, 12, 12, 13, 11, 10, 10, 12, 12, 9, 9, 9, 9, 8,
8, 8, 8,
    11, 11, 7, 7, 7, 9, 11, 11, 11, 11, 12, 12, 12, 13, 11, 10, 10, 10, 12, 9, 9, 9, 9, 9,
8, 8, 8, 9, 7};//小さなロバ
    byte
donBeat[]={ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1,
1, 1, 1, 1, 1, 1, 1,
    1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 4};
    int donLen=sizeof(donTone) ;
```

```
      int play_spd =0; // 速度制御
      int play_no=-1; // 現在どの歌が放送されているか記録し、-1 は何も
 放送されていない状態を表す

int play_pos=0; // 目下歌がどの位置まで放送しているか
      int is_play=0; // 放送しているかの有無
      int play_keys=0; //オーディオ上げ下げを-300~+600 に制御
      int bb[10]={
        2,3,4,5,6,7,8,9,10,11}; // 10 個のボタンを対応させる：1~4 番の
 歌曲、上げ下げ、速度と 2 種の特殊効果音
int bb_state[10]; //10 個のボタンの状態に対応
      void setup()
      {
        int a;
        Serial.begin(9600);
        for(a=0;a<10;a++)
          pinMode(bb[a], INPUT_PULLUP); //内部の上位電圧を利用してボタ
 ンの設定を行う
      }

      void loop()
      {
        int a;

        for(a=0;a<10;a++) // 先に全てのボタンの状態をスキャンする

       bb_state[a]=digitalRead(bb[a]);

        for(a=0;a<10;a++) // 全てのボタンの状態をモニタリングウィンド
 ウに出力する

 {
          Serial.print( bb_state[a]);
          Serial.print(", ");
        }
        Serial.println("");

        if (bb_state[0]==0) //第一番目の歌を放送
        {
          Serial.println(beeLen);// 相関する簡単なデータをモニタリン
 グウィンドウに出力する
         play_spd =0; //速度制御変数の再設定
          play_no=1;  //第 1 番目の歌を放送することを設定
          play_pos=0;  // カウンターリセット、歌の頭から放送
```

```
      play_keys=0; // 昇降キーを先にカウンターリセット

      is_play=1; //1 は放送許可を、0 は放送しないことを示す
      Serial.println("bee playing...");
    }



else if (bb_state[1]==0) //第 2 番目の歌の放送
  {
    Serial.println(starLen);
    play_spd =0;
    play_no=2;
    play_pos=0;
    play_keys=0;
    is_play=1;
    Serial.println("star playing...");
  }
  else if (bb_state[2]==0) //第 3 番目の歌の放送
  {
    Serial.println(tigerLen);
    play_spd =0;
    play_no=3;
    play_pos=0;
    play_keys=0;
    is_play=1;
    Serial.println("tiger playing...");
  }
  else if (bb_state[3]==0) //第 4 番目の歌の放送
  {
    Serial.println(donLen);
    play_spd =0;
    play_no=4;
    play_pos=0;
    play_keys=0;
    is_play=1;
    Serial.println("don playing...");
  }
  else if  (bb_state[4]==0) //キーを上げる
  {
    if ( play_keys<600)
```

```
      play_keys+=50;
         else
           play_keys=600; //最も高い時 600 に達し、当然ながら、より高く設定することが可能であるが、あまり高すぎると音が鋭くなりすぎ聞き取りづらくなる

      }
       else if  (bb_state[5]==0) // キーを下す
       {
         if (play_keys<-300)
           play_keys=-300;
         else
           play_keys-=50;
       }
       else if  (bb_state[6]==0)// 速度を下げる
         play_spd +=100;
         if (play_spd >1000)
           play_spd =1000;
       }
       else if  (bb_state[7]==0)// 速度を上げる
       {
         play_spd -=50;
         if (play_spd <-200)
           play_spd =-200;
       }
       else if  (bb_state[8]==0)// 特殊効果音 1、時間約 0.6 秒
       {
         for(a=0;a<5;a++)  // 低音のバンパー音を発生
         {
           tone(speaker, 150);
           delay(50);
           tone(speaker, 250);
           delay(50);



        }
       else if  (bb_state[9]==0)// 特殊効果音 2、時間約 0.6 秒
       {
         for(a=1000;a<1600;a+=20)  //高音のグリッサンド効果を作成
         {
           tone(speaker, a);
```

```
            delay(20);
          }
        }

        playSong();// 歌曲を呼び出し関数を入れる
        delay(2);
      }

      // 這個函數處理播放一個單音的動作, 每個音符會有一個頻率和想要播放的持續時間
      void play (byte toneNo, byte  beatNo)
      {
        int pk=play_keys+frequency[toneNo]; // 先把頻率和升降 key 結合
        tone(speaker, pk);
        // delay 裡面把速度控制變數加入, 即可改變播放速率
        delay(300* beatNo+ play_spd);
        noTone(speaker);
      }

      // この関数は主に放送する関連プロセスと問題を処理する
      void playSong()
      {
        if (is_play==1) //  1 は現在歌曲が放送可能であることを表し、0 は不可能であることを示す
        {
          switch(play_no)  // play_no の内容に対しどの歌を放送するか決定する
          {
          case 1://beeTone


      if  (play_pos>=beeLen) // 目下放送している曲の位置がすで最後まで達しているかどうか判断する

       {
              is_play=0; // もし既に歌曲が最後まで達していたら、放送停止の設定をする
              return ; //関数から離れる
            }
            //もしまだ放送が終わっていなかったら、目下の位置に当たる音符を放送する
            play (beeTone[play_pos],beeBeat[play_pos]);
            play_pos++; //その後放送中の位置に+1 し、音符位置をポイントする
```

```
        break;
        case 2://starTone
          if  (play_pos>=starLen)
          {
            is_play=0;
            return ;
          }
          play (starTone[play_pos],starBeat[play_pos]);
          play_pos++;
          break;
        case 3://tigerTone
          if  (play_pos>=tigerLen)
          {
            is_play=0;
            return ;
          }
          play (tigerTone[play_pos],tigerBeat[play_pos]);
          play_pos++;
          break;
        case 4://donTone
          if  (play_pos>=donLen)
          {
            is_play=0;
            return ;


}
          play (donTone[play_pos],donBeat[play_pos]);
          play_pos++;
          break;
        }
      }
    }
```