# 音乐播放和混音

## 一、 声音是什么?

实作过很多章节之后，本章开始来谈些不一样的主题，声音就是一个，很多时后，美妙的声音能感动人心，能让 EduCake 发出动人的声音想必是件有趣的事情。

首先，得先了解，声音是一种需要透过介质(空气、水等等)传递的波动，在不同介质里面还会有不同的速度，传入耳朵，震动耳膜，然后我们才有办法感知声音的来源和内容。一般来说，人类耳朵可以听到声音的频率约是在 20Hz~ 20000Hz(赫兹) 之间，超过这个范围就称为超音波，而低于这一范围的称为次声波；而狗就可以听到 40～50000Hz 的声音，所以有时候明明没听到声音狗还是大叫就是这样，因为他们听到人类听不到的声音了，用超音波模块简单的测试就可以发现这个状况。其中 Hz 是一种描述声音频率的单位，如下图 1

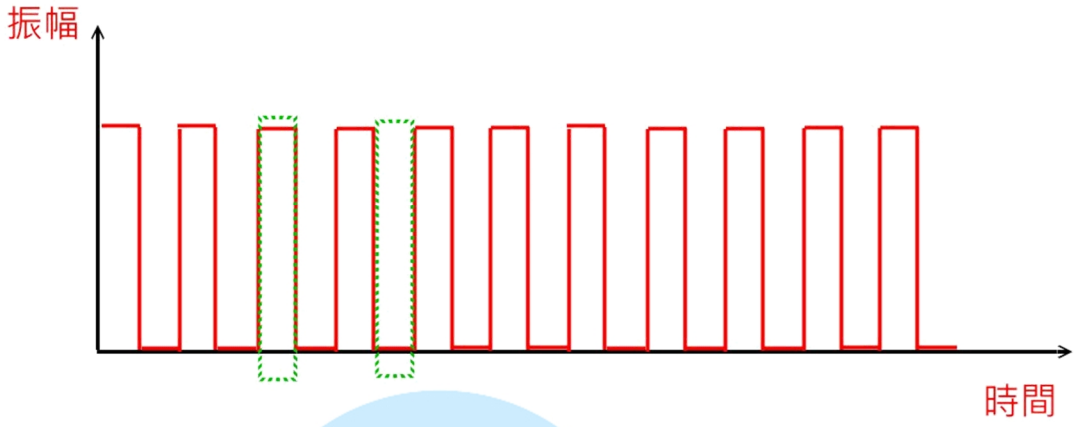


图 1. 一个简单的声音波动图

图中的声音只是一种简单的高低的方波型构成，有点像是之前谈过的 PWM，这个图的波只有两种，高态和低态，不管哪一种，都算是一次震动，只是差别震动幅度高低而已，频率的定义就是: 一秒钟里面发生多少次这样的震动，EX: 300Hz 代表一秒钟有三百次震动次数，频率越高，代表震动的次数越多，我们的耳朵听到的声音就会越尖锐，相对的频率越低听到的声音就会越低沉。而震动的幅度代表了声音的强度，震动的幅度越大，我们就会听到越大的声音。这就好像我们用手拍水面，轻轻拍，水波很低缓；但用力拍，水就会波动剧烈是一样的。

再来就是音色(声音的特色)，如图 2，同样的音高和同样的声音强度的情况下，音色会让一段声音明显被分辨出是不同乐器或是不同的人声，这主要是因为发出声音的材质不同，虽然发出一样频率的声音，但该材质的致密度、刚性、弹性等等的物理性质不同使得发出的声音里面会有额外的”特色杂音”混在其中，形成特色频率。就好像唱歌，虽然同样都唱出 Do、Re、Me、Fa、So…的音，但男生唱出来混有男生特有的低沉音在内，女声唱出混有女生特有的高音在内，都会使得一段同样旋律的声音出来有明显不同；各种大提琴、箫、吉他、钢琴等等也是因为这样。这些特色音里面的这些特色杂音，因为通常含盖了中低高等音频，甚至有些几乎要超过人耳能听到的范围，所有的这些声音若是以特定比例混合起来，虽然同时都发出一样声调，可以因为彼此音色的特性，

使的整体的声音变好听，这也是为何乐团演奏总喜欢一堆各式乐器混和去搭配的原因，整体演奏的感觉好很多。

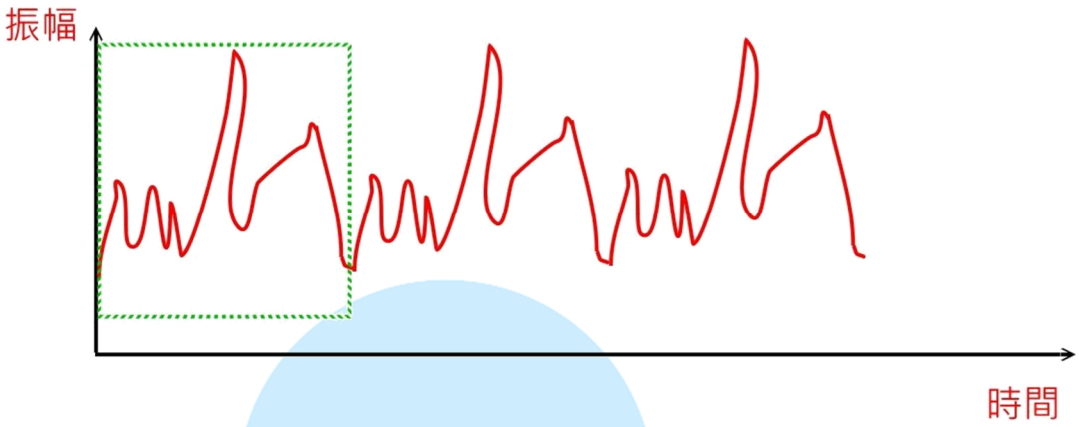


图 2. 声音的特色

有了频率、振幅、音色，其实我们就可以把很多美妙的声音重现出来。控制振幅，也就是音量大小(对控制来说就是改变电流大小或是 PWM)，可以模拟出声音远近的变化感觉，而变化的速度，还可以让人耳有速度的感觉，就好像警车或是救护车，我们可以听声音就知道他们接近或远离的速度是一样的，不同的是，这里面除了音量的变化还带有都普勒效应的频率变化。也可以利用左右两个喇叭来配合，更能营造出立体的环场和左右方向感觉的声音。甚至反推这个过程就可以去作声音的辨识，甚至语音的辨识了，不过后面这些不是本章讨论范围，往后再谈。

## 二、 发出声音

知道声音的特性以后，就可以利用 EduCake 的内建功能来实作出这些声音，首先需要了解两个指令 tone( )、notone( )。

tone( ) 这个指令会在指定的 digital pin 脚上产生一个指定频率的方波(工作周期是 50% 的 PWM 信号)，用来仿真出不同频率的声音，可以指定持续发出声音的时间或者一直持续直到呼叫 noTone()，这个 pin 脚上就可以接普通的小蜂鸣器(这种音质通常很难听，只适合用来发出哔哔之类声音)或者喇叭来发出一段声音，后者声音的质量明显比较好。另外须注意同一时间只能产生一段声音，如果已经有某个 pin 脚上在播放声音，则想要同时在不同 pin 脚呼叫 tone() 是无效的，所以若想要在多个 pin 脚上播放不同的声音，需要在呼叫前先使用 noTone()关闭此 pin 脚的声音，再用 tone( )到别的 pin 脚去播放，这个动作对于制作像是左右声道的喇叭轮流播放特别重要。tone( )的使用方法主要有以下两种。

tone(pin, frequency)

tone(pin, frequency, duration)

pin: 指定要產生音頻的 pin 腳

frequency: 聲音頻率(Hz)，unsigned int，須注意人耳只能聽到 20~20000 左右的範圍

duration: 音頻的持續時間，單位為毫秒，unsigned long 型別 (此參數為非必需)

首先把电路先接好如图 3:

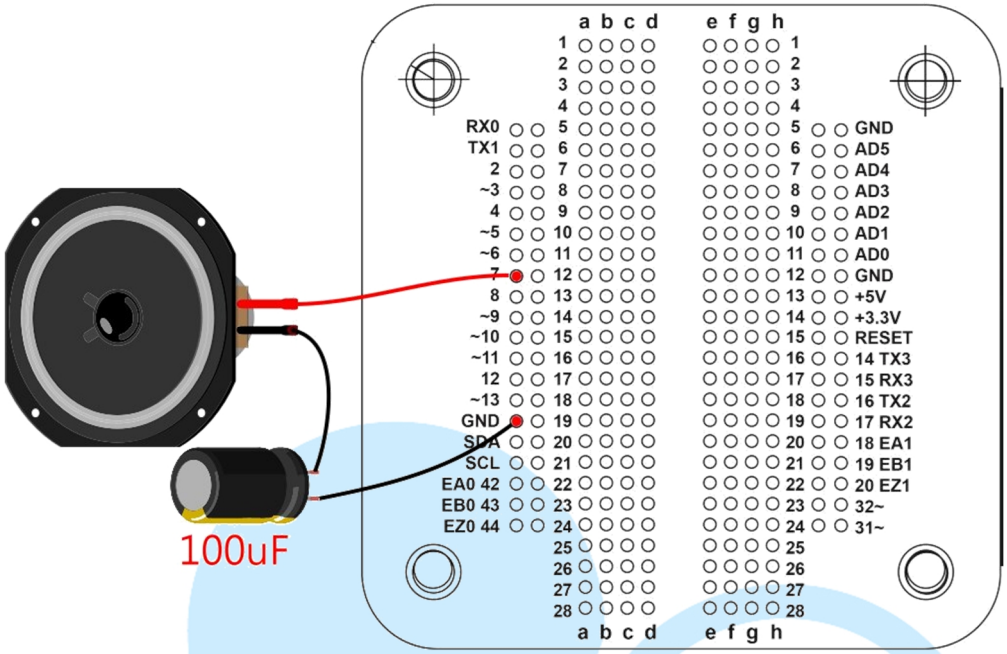


图 3. 单喇叭接线图

电路中那颗 100uF 的电解电容主要是滤波的效果，有些人则是直接只使用电阻，主要是怕喇叭的电阻过低，控制板的电流会太大，但会稍微降低音量，到底是搭配大小电容的串联、并联，或是加电阻上去调整，或是改变采用的喇叭尺寸的大小，或是用晶体管接电流放大电路去让声音更大声更好控制等等，这部分就留给读者自行去测试，各有不同的效果喔。若喇叭有接共振腔还会有不同效果，像有人会接大音箱或是小音箱，或是最近流行鸚鵡螺型状的音箱都是为了比较好的音质效果而采用。而喇叭另外一条线接到 pin 7，到时候就可以使用 tone()指令来对 pin 7 作声音的控制，这样就可以实作出一组喇叭电路。简易的测试程序代码如下

```
void setup()
{    }
void loop()
{
   tone(7, 500); //  令 pin7  不斷發出 500Hz 的聲音
   delay(500); //  這個 delay 等於讓上面指令持續發出 0.5 秒聲音
   noTone(7); //  關閉 pin 7 的聲音
   delay(500);
}
```

若要改两组喇叭模拟左右声道，那电路也要改一下，这次使用电阻来调整音量，电路如图 4

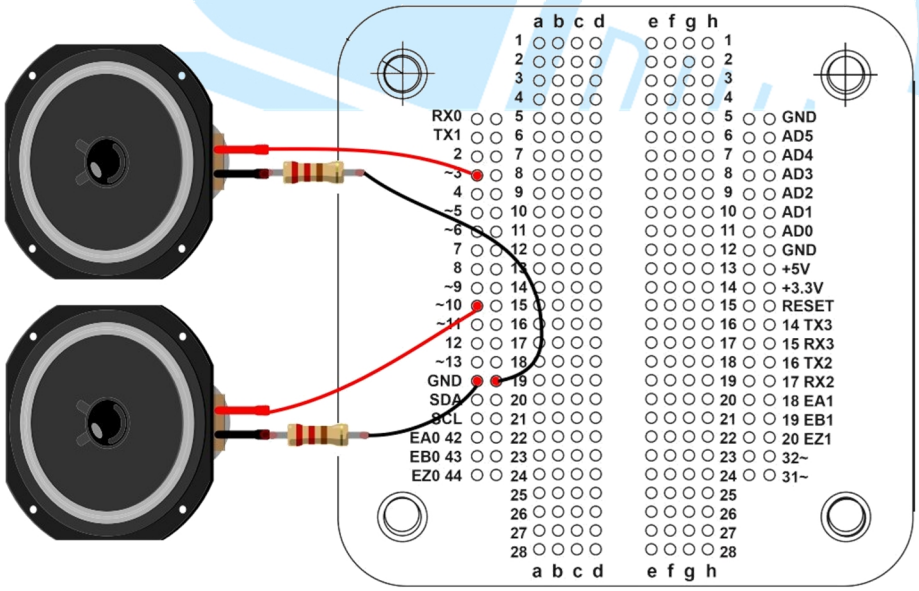


图 4. 双喇叭接线图

搭配的程序代码就可以写成这样

```
int left_sound=10;
int right_sound=3;
void setup()
{    }
void loop()
{
  tone(left_sound, 500); //  令左邊喇叭不斷發出 500Hz 的聲音
  delay(500); //  這個 delay 等於讓上面指令持續發出 0.5 秒聲音
  noTone(left_sound); //  關閉左邊喇叭
  tone(right_sound, 500); //  接著立刻打開右邊喇叭不斷發出 500Hz
的聲音
  delay(500); //  這個 delay 等於讓上面指令持續發出 0.5 秒聲音
  noTone(right_sound); //  關閉右邊喇叭
}
```

其中那个电阻变成可变电阻的话，就可以调整音量大小，若是用之前的章节谈过的 serial 通讯，那就可以变成使用计算机的 UI + 鼠标拖拉式的来改变音量了，应用的方式是很多的，就看读者怎么搭配前面谈过的东西。

## 三、 音符和节拍

接下来想要播放音乐的话，必须先了解音符和节拍这两件事情。节拍简单讲就是速度快慢，通常讲一分钟 180 拍，就是指一分钟可以播放 180 个音符，所以很明显节拍数越大，代表播放速度越快，相对就越慢，就可以利用控制节拍来控制播放速度，参照之前讲的 tone 和 notone 中间使用的那个 delay()指令，里面的数字其实就可以用来控制节拍和播放速度。

音符比较麻烦，一般以钢琴键白色的部分(黑色的按键先不管)简单来分就是低音(Do、Re、Me、Fa、So、La、Se)、中音、高音，各七个音共 21 音。频率约是如下:

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

中音部分: 523、587、659、698、784、880、988

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

这些音符的频率是从维基百科查询到的(http://zh.wikipedia.org/wiki/音符)，音符的相关知识很多，但这里就不使用过多篇幅来研究，有兴趣的读者可以去更深入的研究看看。有了音符频率表，再来就可以用程序来播出来看看，先来播看看低音的 Do~Se:

```
int frequency[]={
  262,294,330,349,392,440,494}; // 把低七音的頻率都先放到陣列裡面

void setup()
{
}
void loop()
{
  int a;
  for (a=0;a<7;a++)
  {
    tone(7, frequency[a]); // 從 pin7 發出聲音
    delay(500); // 每個音播放 0.5 秒
    noTone(7);
  }
}
```

这样就可以把低音的 Do~Se 播出来听听看是什么感觉了，用大喇叭，小喇叭或是迷你的蜂鸣器，听到的声音都不会一样，读者可以自行试试看。若是需要从低音到高音都一口气播出来试试看，那就把里面那个 frequency 的宣告改这样

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

然后后面那个 for 循环改这样

```
for (a=0;a<7;a++)   改成   for (a=0;a<21;a++)
```

这样就可以一次听到所有的低中高音的标准音符频率，有了频率表，就可以开始来编辑歌谱让 EduCake 唱歌了。

## 四、 循环由高至低、警车声、弦波音

知道音符的播放后，我们还需要效果音，效果音其实还是各种频率的声音，只是可能连续、断续、忽高忽低、起伏、随机数、甚至是以某种数学曲线计算去播放，可以作出各种不同的有趣效果，底下就一个一个来看。

前面有谈过人耳能听到的声音介于 20~20000Hz 之间，这里就写个程序把这范围的声音播放出来听听看:

```
void setup()
{
}
void loop()
{
  int a;
  for(a=20;a<20000;a+=10) // 频率 20~20000，每隔 10Hz 播出频率
  {
    tone(7,a);
    delay(5);
  }
  noTone(7);
}
```

根据笔者测试，普通那种计算机用 6cm 小喇叭，大约到 12000Hz 以上耳朵就听不太到了，要换比较小尺寸的喇叭才听的到很尖锐很高的声音。再来试试看哔哔的声音:

```
void setup()
{
}
void loop()
{
    tone(7 ,900); //  調整裡面的 900 可以讓嗶嗶聲有不同感覺
    delay(200); //  控制嗶嗶聲的快慢
    noTone(7);
    delay(300);
}
```

可以发现哔哔声其实是一开一关的声音构成，里面的频率和 delay 的时间控制都可以玩出很多种不同的哔哔变化。把这个声音用循环包起来像上一个程序一样令频率变化，并加上加速度的感觉，可以制作类似高处掉下的滑音。

```
void setup()
{
}
void loop()
{
  int a;
  double v=0,s=3500; //  假設從 3500 米的高度掉下，初速為零，重力加速度 10
  while (s>50) //  掉到 50 米的高度就停
  {
    tone(7,s); //  每次直接把目前高度當作頻率輸出
    delay(30);
    noTone(7);
    s-=v; //  每次掉下 V 的距離
    v+=10; // V 每次都因為重力加速度增加 10
  }
}
```

测试结果可以听出来这声音还满好玩的，而程序代码里面的 tone(7,s); 这里面的频率如果替换成 tone(7,3500-s);，还会有不一样的有趣效果，本来由高到低的滑音会变成由低到高，调整程序代码里面的 V+=10 那个代表加速度的数字也会让声音变化率有特别的效果，当然，delay()里面的数字也可以调整看看，都很有趣。这里若加入使用三角函数的成分，还会有不一样的效果:

```
void setup()
{
}
void loop()
{
  double a;
  for(a=0;a<100;a+=0.15)
  {
    tone(7, 1000+500*sin(a));
    //一圓周=2 pi =2x3.14=6.28 ，所以 100 約跑了 16 圈
    // sin(a) 會在+-1 之間變化 ，乘上 500 再加上一千以後，
    // 他會在 500~1500 之間作 sin 的弦波變化
    delay(50);
  }
  noTone(7);
}
```

感觉是不是比滑音又更有趣了?不同的弦波函数如 cos()、tan()等等，都会有不同表现喔。而生活中常听到的警车救护车声音，其实仔细分析也是这种忽高忽低的声音构成，以下来试试看这个片段:

```
int spd =10; // 改變這個能直接影響循環快慢
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);
}
```

程序代码里面的 delay 和循环里面的数字调整一下，就可以模拟出各种警车或是救护车的声音啰，不过，通常我们实际听到的和这个有点不同，因为警车或是救护车除了高低起伏的声音以外，还会搭配高速的接近或是远离我们，这会有都普勒效应，频率会有稍微的改变，所以若想模拟警车声音接近的话，就得让整体频率慢慢变高；远离则是慢慢变低，就会有类似效果。相关的实际公式就请参照高中物理课本，里面会有详细说明和公式，或是查询这个维基网页:

http://zh.wikipedia.org/wiki/都卜勒效应

以下这段程序代码用来随机数生成不可预期的声音，很有电影里面机器人讲话的感觉

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

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

里面的 random(100,2000) 会产生 100~1999 之间的随机数频率，改变这两个数值或是改变 delay()里面的时间都会有很奇特的声音出现。差不多所有的一般声音都可以藉由以上这些小技巧去测试合成，也让静悄悄的单晶世界有更多乐趣。

## 五、 放一首歌，歌谱分析

上面谈过各种音频控制，这里我们就实际来让 EduCake 唱出一首歌来试试看。首先，我们先要有一个歌谱，这到 GOOGLE 上面到处抓都会有，只是歌曲几乎都要注意版权问题，所以虽然可以抓到很多歌，却不能直接拿来使用。找很久最后发现一些很久很久以前的童谣就比较没有版权问题，可以直接使用来说明或是自己玩，但这若是要贩卖恐怕还是不行的。

以下就以两只老虎这首经典童谣来说明相关的细节，如下图 5。因为这也不是音乐课，我们不需要看懂整张图里面所有的细节，只要几个重点看懂就好。第一点就是图中的歌词上面都会跟着数字，像是”两只老虎”上面跟着 1231，这些 1234567 数字对应了歌谱的 Do、Re、Me、Fa、So、La、Se，只要照着这些数字对应的音符去敲钢琴或是乐器，就能谈出两只老虎这条有趣的童谣。第二点就是升降 KEY 问题，前面谈过 Do~Se 七个音，还有分低音、中音和高音，歌谱里面也需要有这些标住才能展现歌词的高扬或是低沉的变化，注意第二行和第三行的歌词里面就有这些标记，第二行的”一只没有耳朵”上面跟的 565431，其中的 5654 底下有底线，这就说要升阶，中音要变高音才对；第三行的”真奇怪”那个”奇”字上面的 5，下方有一点，那个点的意思就是要降级，中音要变低音。第三点，第一行和第二行里面的”跑的快”后面都有接着”-”，这是代表前一个音要多播放一拍，一般每个字是一拍，有这个符号就可以让某个字两拍，拖长音的意思，所以若是想要更长的音就两个，再长就三个，依此类推。

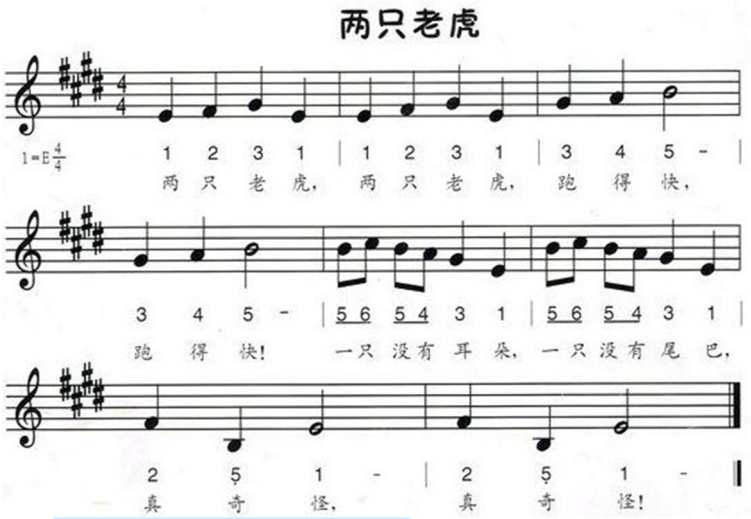


图 5. 两只老虎五线谱

有了歌谱再来就要把歌谱想办法转换成真正的程序代码，由上面的歌谱和解说信息，我们可以大概的整理出每个歌词里面的字都需要音频(高低音)和节拍(长短)，并将他们宣告成程序可以使用的数组如下

```
byte tigerTone[]= //記錄所有音符的頻率對應索引值
{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[]=// 記錄每一個音符需要播放多長拍數，程式裡面再看要定義每分鐘
幾拍，就可以控制撥放速度
{1,1,1,1,1,1,1,1,1,2,1,1,2, 1,1,1,1,1,1,1,1,1,1,1,1,1,1, 2,1,1,2};
```

因为前面已经使用完整的 frequency[]数组记录低中高音的频率，共存放 21 个数值，这里的 tigerTone 数组里面只要纪录对应的号码就可以顺利播放，其中 0~6 对应低音的 Do~Se、7~13 对应中音的 Do~Se、14~20 对应高音的 Do~Se，组合起来的程序代码如下:

```
    const int speaker=13; //  喇叭改到 13 號腳位輸出，為了後面的整合
作準備
    int frequency[]={   //  把低中高音的音頻先建立表格
      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, 1,1,1,1,1,1,1,1,1,1,1,1,1,1, 2,1,1,2};
    const int playLen=sizeof(tigerTone); //  播放長度先計算出來，需注
意這是實際占記憶體長度

    void setup()
    {
    }
    void loop()
    {
      int a;
      for(a=0;a< playLen -1;a++)
      {
        tone(speaker,frequency[tigerTone[a]]); //從預設的頻率表裡面索
引目前要播放的
        delay(300* tigerBeat[a]); //  每個音符的延遲時間，延的越久這個
音就播越久
        //  約是一秒播三個音符，每分鐘 180 個
        noTone(speaker); //關閉聲音，等待下一個音符，也可以最後在關
      }
      delay(3000);
    }
```

这样作就能完整播放一首歌了，播完以后，最后那个 delay 指令会等待三秒钟又会继续在播一次。如果把这整个流程都包装起来，就可以利用 SD 卡储存大量的音乐，用按钮来选择要播哪一首，搭液晶屏幕甚至触控面板，就是一台点歌器了，后面我们会有章节实际完整的来作这个有趣的东西。

## 六、 多首同播

前面讲到点歌器的概念，这边就来实际作看看简单的版本。首先，得先有几首歌，然后还要有对应的按钮来选歌，或是很多首歌的话也可以利用 4*4 之类的数字键盘来选歌。要能选择歌曲播放，首先得先要有多首歌谱，为了方便说明起见，以及配合后面的功能，我们安排四首歌:两只老虎、小星星、小蜜蜂、小毛驴这些经典童谣。先来建立每一首歌的频率对应表格和播放节拍表

```
int frequency[]={   // 把低中高音的音頻先建立表格
  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};// 兩隻老虎
byte tigerBeat[]={1,1,1,1,1,1,1,1,1,1,2,1,1,2,
1,1,1,1,1,1,1,1,1,1,1,1,1,1, 2,1,1,2};
int   tigerLen =sizeof(tigerTone) ;   // 順便把這首歌的長度計算出
來

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};   // 小蜜蜂
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[]={7,7,11,11,12,12,11,10,10,9,9,8,8,7,11,11,10,10,9,9,8,11,11,1
0,
    10,9,9,8,7,7,11,11,12,12,11,10,10,9,9,8,8,7}; //小星星
    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};//小毛驢
    byte
donBeat[]={ 1,1,1,1,1,1,1,1,1,1,1,1,2,1,1,1,1,1,1,1,1,1,1,1,1,2,1,1,1,1,1,1,1,
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) ;
```

建立好这些以后就可以开始用程序来播放，也可以把这些信息写入档案放到 SD 卡里面，就可以利用 SD 卡的超大容量放置很多首歌来提供随选播放了。

之前的程序只能播放一首歌，但现在我们有四首歌了，程序的编排也要针对多首歌的使用来重新思考。

首先我们想要用一些按钮来对应每首歌，按 1 就播放两只老虎、按 2 就播放小蜜蜂、按 3 就播放小星星...依此类推。但因为每首歌长度并不相同、按钮要按下就要重头开始播放对应的歌，而且后面还需要能够有类似暂停、修改音乐播放的内容或是升降音等等功能，所以需要把程序代码重新安排成类似这样:

```
int play_no=-1; //  記錄現在是在播放哪一首，-1 代表沒有在播放
int play_pos=0; //  目前播放到歌曲的哪個位置
int is_play=0; //  是否有在播放

int bb[4]={2,3,4,5}; //四個按鍵的腳位定義在 digital 的 2~5
int bb_state[4]; //  記錄每個按鍵的狀態

void setup()
{
  int a;
  Serial.begin(9600);
  for(a=0;a<4;a++)
    pinMode(bb[a], INPUT_PULLUP);
}

void loop()
{
  int a;
  for(a=0;a<4;a++) //  先掃描這些按鍵是否有被按下
   {
   bb_state[a]=digitalRead(bb[a]);   //把按鍵狀態存到陣列中
   }

  //  接下來就判斷每個按鍵的狀態來作處理，這裡假設使用者一次只按一個按鍵
  //  一次只能播一首歌，當然也可以安排兩鍵一起按下，一次兩首歌混在一起播放
```

```
// 但這樣就聽不太出來到底在播什麼，所以先不要這樣作
 if (bb_state[0]==0) // 第一個按鍵被按下，要播第 1 首歌
 {
    Serial.println(beeLen); // 印出相關的簡單訊息到監視畫面
    play_no=1;     // 設定第 1 首歌要播
    play_pos=0;   // 位置歸零，從頭開始播放
    is_play=1;   // 1 代表可以播，0 代表不播放
    Serial.println("bee playing...");
 }
 else if (bb_state[1]==0) // 第二個按鍵被按下，要播第 2 首歌
 {
    Serial.println(starLen);
    play_no=2;     // 設定第 2 首歌要播
    play_pos=0;   // 位置歸零，從頭開始播放
    is_play=1;   // 1 代表可以播，0 代表不播放
    Serial.println("star playing...");
 }
 else if (bb_state[2]==0) // 第三個按鍵被按下，要播第 3 首歌
 {
    Serial.println(tigerLen);
    play_no=3;     // 設定第 3 首歌要播
    play_pos=0;   // 位置歸零，從頭開始播放
    is_play=1;   // 1 代表可以播，0 代表不播放
    Serial.println("tiger playing...");
 }
 else if (bb_state[3]==0) // 第四個按鍵被按下，要播第 4 首歌
 {
    Serial.println(donLen);
    play_no=4;     // 設定第 4 首歌要播
    play_pos=0;   // 位置歸零，從頭開始播放
    is_play=1;   // 1 代表可以播，0 代表不播放
    Serial.println("don playing...");
 }
```

```
  playSong(); //  呼叫歌曲播放函數
   delay(5);
}

// 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)
{
   tone(speaker,frequency[toneNo]);
   delay(300* beatNo);
   noTone(speaker);
}

// 這個函數專門處理播放的相關流程和問題
void playSong()
{
   if (is_play==1) //   1 代表現在可以播放歌曲，0 代表不播放
   {
      switch(play_no)   // 針對 play_no 的內容來決定要播放哪一首歌
      {
      case 1://beeTone
         if   (play_pos>=beeLen) // 判斷目前播放的位置是否已經到最後位置了
         {
            is_play=0; // 如果已經到歌曲最後位置，設定停止播放
            return ; //離開函數
         }
         //若還沒播完，就播放目前位置的音符
         play (beeTone[play_pos],beeBeat[play_pos]);
         play_pos++; //然後把播放的位置+1 ，指向下一個音符位置
         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;
        }
    }
}
```

这样就可以按下 1~4 号按钮，对应播放上面设定好的童谣，并且歌曲播放完毕会自动停止，播放过程中，还可以随时按下其他的按钮来切换播放不同的歌曲，是不是很有趣呢？这里还可以思考看看改用不同的方式，让歌曲的种类和选择方式变很多样化，像是使用红外线遥控器来摇控，这需要结合红外线

接收传感器材行。也可以使用连续旋转的旋转编码器，类似汽车音响那样，用转的来选择歌曲或是频道，也可以修改程序，多一些按钮可以设定像是播放顺序的设定；或是播完自动轮播；或是某首歌曲设定不断重复播放...之类的进接用户自定义功能，这都有助于让音乐播放器更好玩更贴近使用者的需求。

## 七、 控制速度、混音、调音量、升降 KEY 等等

能够播放多首歌曲以后，我们就可以开始来加入一些特效，让整个播放器的功能能够更丰富有趣。在一般 PUB 或是演唱会现场，常会看到有键盘手或是混音调音人员在按着很复杂的大按钮盘，上面可能一堆按钮或是一堆转盘旋钮之类的东西，可以让人很轻易的在歌曲或是舞曲中加入各种特效声音像是动物叫声、物品撞击声、金属音、摇滚节奏等等，不管是哪一种，其实原理都是一样的，一首音乐就像上面讲的是一堆音符构成，若我们把想要混入的特效音也编成音符，加入到正在播放中的音乐里面，就可以结合音乐和特效音。加入的方法是，直接把音乐和特效音的频率结合起来，可能是直接相加除以 2；或是两者乘上一个权重以后相加，例如音乐占 7 成，特效音 3 成之类的，这样还可以有个特效音权重设定的功能可以加入。但在我们这个范例里面这种作法的效果都不会好，因为分辨率太低了，前面的范例都一秒才播放三个音左右，等于取样频率 3Hz/sec，一般的音乐早就都 22KHz/sec 或以上了，这就好像照相机的分辨率越高画质越好一样，取样频率越高也代表声音的音质会越好，但这又牵涉到复杂的音频处理和音乐取得问题，这边先跳过，我们先从简单的实作来处理。

首先是控制速度，这可以轻易的由 void play (byte toneNo, byte beatNo) 这个函数下手，里面的 beatNo 就是控制音乐播放速度的关键，调整个这数字的大小就可以直接控制音乐的快慢，想象播放超快的两只老虎，或是超慢的小毛驴，其实很有趣的，这只要针对这个变量宣告一个 play_spd 来控制就可以了。

再来是混音，前面讲过取样频率不高的时候效果不好，我们这里就用不同方式，改作特效音，提供一个高音的弹跳声音和一个低音的弹跳声音，给使用者按下的时候直接插入播放，一样可以有不同效果呈现。弹跳音的作法就如同前面讲过的类似救护车的滑音或是随机数音，只是把循环的频率压在低频可能是 200Hz 附近、高频压在 1200Hz 附近，也是有不错效果。

调整音量的部分，其实就是输出功率的控制，但因为这里是直接由脚位输出，功率很低，要调音量不容易作。若真的想处理就是要找一些音频功率放大芯片来放大，搭可变电阻就可以简单的处理音量。或是也可以采用现成的音频放大模块，如图 6

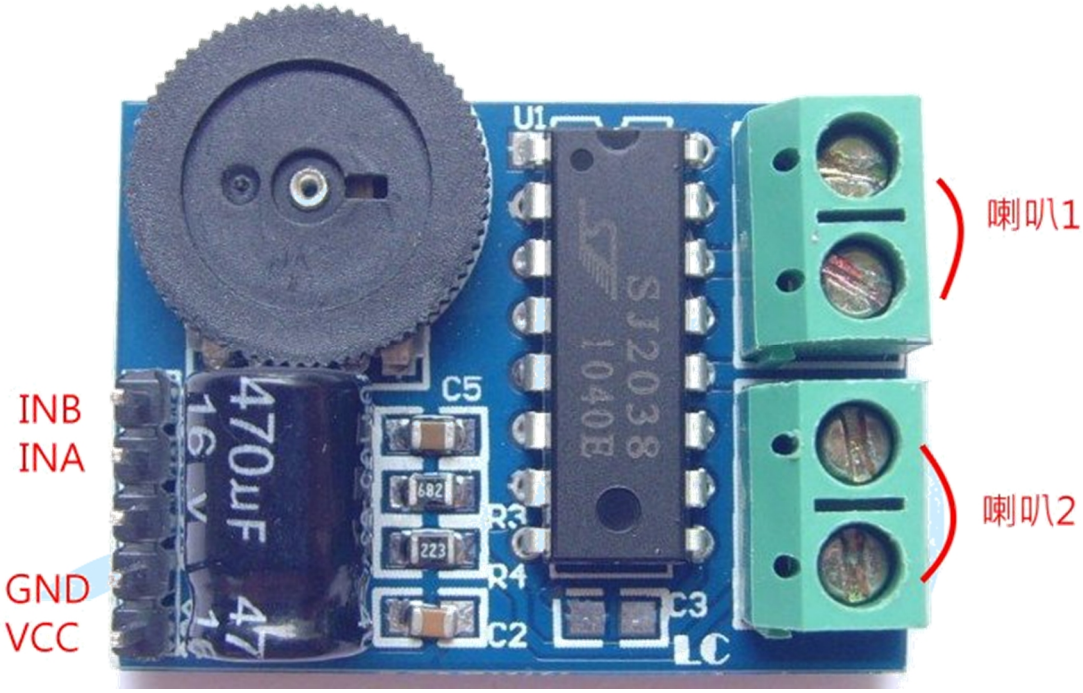


图 6. 音频放大模块图

想要升降播放的 KEY，那就得要另外宣告变量来控制，这里有两种作法可以参考，一种是直接把频率索引表里面的数值调升或是调降，会有很明显的变化，但有个问题是 0 这个索引值已经对应到频率表里面最低的一个频率，没有调降空间了，若是遇到这个频率就调不了。同样的 20 这个索引值已经对应到最高的频率，一样没有调高空间。所以另外一个作法是直接宣告一个变量 play_keys 来作处理，这变量可以默认为 0，代表频率没有变化，但让这个变量可以在+-300~500 之间或是更大的范围内受到按钮调整，然后播放频率的时候直接把这个变量加上去，就可以直接把接下来的每一个音符都作声音频率的调升/调降。完整电路图如图 7

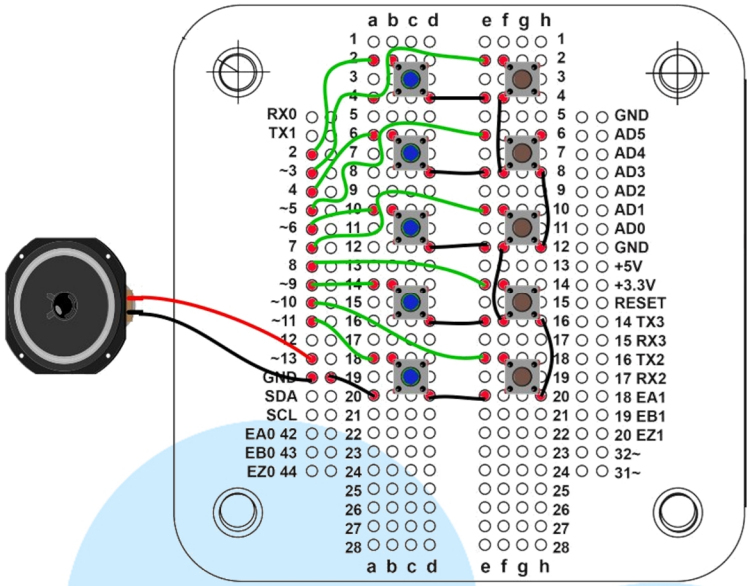


图 7. 完整线路图

完整程序代码如下

```
int speaker_pin=13;

int frequency[]={   // 把低中高音的音頻先建立表格
   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};// 兩隻老虎
byte tigerBeat[]={1,1,1,1,1,1,1,1,1,1,2,1,1,2, 1,1,1,1,1,1,1,1,1,1,1,1,1,1,
2,1,1,2};
int   tigerLen =sizeof(tigerTone) ;   // 順便把這首歌的長度計算出來

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};   // 小蜜蜂
```

```
    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}; //小星星
    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};//小毛驢
    byte
donBeat[]={ 1,1,1,1,1,1,1,1,1,1,1,1,2,1,1,1,1,1,1,1,1,1,1,1,1,2,1,1,1,1,1,1,1,1,
    1,1,1,1,1,2,1,1,1,1,1,1,1,1,1,1,1,1,1,4};
    int donLen=sizeof(donTone) ;

    int play_spd =0; //  速度控制
    int play_no=-1; //  記錄現在是在播放哪一首，-1 代表沒有在播放
    int play_pos=0; //  目前播放到歌曲的哪個位置
    int is_play=0; //  是否有在播放
    int play_keys=0; //  音頻升降控制-300~+600

    int bb[10]={
      2,3,4,5,6,7,8,9,10,11}; //  十顆按鈕對應到:1~4 號歌曲、升降、速度和
2 種特效音
    int bb_state[10]; //對應十顆按鈕的按下狀態
```

```
void setup()
{
  int a;
  Serial.begin(9600);
  for(a=0;a<10;a++)
    pinMode(bb[a], INPUT_PULLUP); //利用內部的上拉電阻來設定
按鈕
}

void loop()
{
  int a;

  for(a=0;a<10;a++) //  先掃描全部的按鈕狀態
      bb_state[a]=digitalRead(bb[a]);

  for(a=0;a<10;a++) //  輸出所有按鈕狀態到監控視窗
  {
    Serial.print( bb_state[a]);
    Serial.print(", ");
  }
  Serial.println("");

  if (bb_state[0]==0) //播第 1 首歌
  {
    Serial.println(beeLen);//  印出相關的簡單訊息到監視畫面
    play_spd =0; //重設速度控制變數
    play_no=1;   //  設定第 1 首歌要播
    play_pos=0;   //  位置歸零，從頭開始播放
    play_keys=0; //  把升降 KEY 先歸零
    is_play=1; // 1 代表可以播，0 代表不播放
    Serial.println("bee playing...");
  }
```

```
        else if (bb_state[1]==0) //播第 2 首歌
          {
            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) //播第 3 首歌
          {
            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) //播第 4 首歌
          {
            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) //  升 KEY
          {
            if ( play_keys<600)
```

```
        play_keys+=50;
            else
                play_keys=600; //最高到 600，當然可以設定更高，但過高會
太尖銳聽不太到
        }
        else if   (bb_state[5]==0) //  降 KEY
        {
            if (play_keys<-300)
                play_keys=-300;
            else
                play_keys-=50;
        }
        else if   (bb_state[6]==0)//  速度降低
        {
            play_spd +=100;
            if (play_spd >1000)
                play_spd =1000;
        }
        else if   (bb_state[7]==0)//  速度提高
        {
            play_spd -=50;
            if (play_spd <-200)
                play_spd =-200;
        }
        else if   (bb_state[8]==0)//  特效音 1  ，時間約 0.6 秒
        {
            for(a=0;a<5;a++)   //  產生低音的碰碰聲
            {
                tone(speaker, 150);
                delay(50);
                tone(speaker, 250);
                delay(50);
            }
```

```
    }
    else if   (bb_state[9]==0)// 特效音 2 ，時間約 0.6 秒
    {
      for(a=1000;a<1600;a+=20)   //製造高音的滑音效果
      {
        tone(speaker, a);
        delay(20);
      }
    }

    playSong();// 呼叫歌曲播放函數
    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);
  }

  // 這個函數專門處理播放的相關流程和問題
  void playSong()
  {
    if (is_play==1) //   1 代表現在可以播放歌曲，0 代表不播放
    {
      switch(play_no)   // 針對 play_no 的內容來決定要播放哪一首歌
      {
      case 1://beeTone
```

```
        if   (play_pos>=beeLen) //  判斷目前播放的位置是否已經到最後
位置了
            {
              is_play=0; //  如果已經到歌曲最後位置，設定停止播放
              return ; //離開函數
            }
            //若還沒播完，就播放目前位置的音符
            play (beeTone[play_pos],beeBeat[play_pos]);
            play_pos++; //然後把播放的位置+1  ，指向下一個音符位置
            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;
            }
        }
    }
```