Music Player and Mixer

1. Introduction to Audio Function

Before getting into the fun stuffs, using EduCake to transmit sound and play nice music, let' s take a look at audio function to better understand audio and sound wave.

First, we need to understand that sound is a vibration that propagates and transmits through a medium, such as air or water. Sound travels at different speed through different type of medium. As sound wave reaches human ear, the sound vibrates in varying frequencies causes our ear' s membrane to sense and detect the sound, part of human' s hearing mechanism.

In general, human is capable of hearing sound between the 20Hz to 20,000Hz range and cannot hear sound wave outside of this range. Animal such as dog is more sensitive to sound that human and able to hear sound between the 40Hz to 50,000Hz range. We can perform experiment using ultrasound module that can emit sound wave above the 20,000Hz range where it' s not detectable by human and audible to dog.

Hz (short for Hertz) is used as the measurement for sound wave frequency, as shown in figure-1 below:

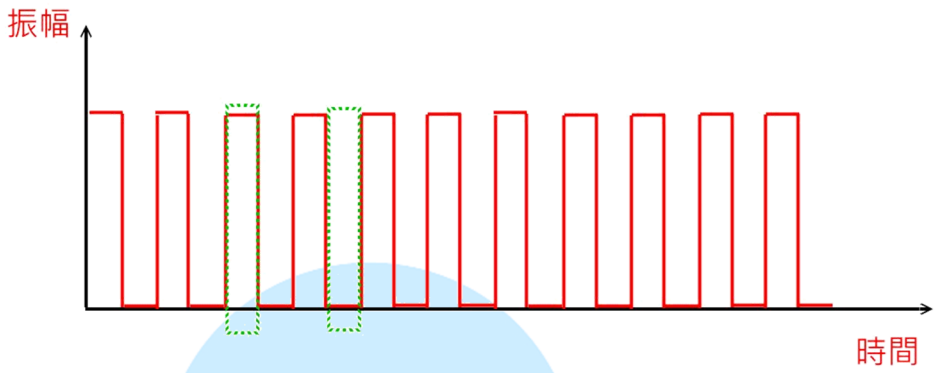

Figure-1: A simple sound wave

In figure-1, the graph represent a simple sound wave, similar to PWM signal covered in the earlier chapter, where each signal transition from high-to-low or low-to-high represent a single vibration with different in amplitude. Sound frequency is calculated by counting the number of signal transition within one second, where 300Hz represent a sound wave that vibrate 300 times per second. To human ear, sound wave at higher frequency is sharper than low frequency. The sound wave amplitude is relative to the noise level. Sound wave with higher amplitude is louder.

Another characteristic for sound is timbre, also known as tone color or tone quality, as represent by the graph in figure-2. In music, timbre is what make a musical instrument sounds different from another, such as Piano versus violin, even they are both playing the same frequency and amplitude (same note and volume). Timbre is a sound characteristic that human use to identify whether the sound is generated by a Piano or Violin, which is quite distinct. While different type of musical instrument can generate musical note based on the same frequency at the same amplitude, the sound density and rigidity are different from different type of musical instrument. By mixing sound from different instrument, musician is able to produce amazing music.

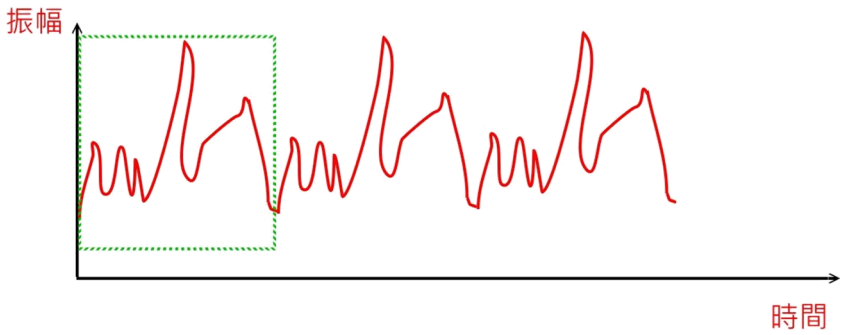

Figure-2: Timbre

By controlling sound wave’ s timbre, amplitude and varying density, we can generate beautiful sound. The sound wave’ s amplitude, which affect audio volume, can be controlled by increasing or decreasing electrical current similar to PWM, and can create audio sensation that make it seems like the sound wave is coming from a distance. With two different channel of sound waves, it’ s possible to generate 3D and stereo audio sensation, where you can hear moving object such as automobile, moving from left to right, or gradually moving away and vice versa. In additional to controlling sound wave’ s output, the audio input function can be adopted to perform voice recognition and converts sound wave into input and command for an application.

This application note is intended to provide basic information about audio function and does not cover the advanced function mentioned in the previous paragraph. Perhaps, we can cover some of the more advanced audio function in future application note.

2. Audio Output Function

In this section, we will work through sample exercise to generate audio output using the EduCake. Before getting to the exercise, let' s take a look at the tone() and notone() functions.

The tone() function is used to generate square wave at a specified frequency (a 50% duty cycle PWM signal) on a designated digital pin and attach a buzzer or speaker to the pin to generate different tone by changing the frequency. The tone() function can be called with parameters to generate tone different frequency, for a predetermined duration or until the noTone() function is called. Only one tone can be generated at a given time. If a tone is already playing on a pin, calling the tone() function does not have any effect. If the tone is playing on the same pin, the call will set its frequency.

To generate different tone on multiple pins, you need to call the noTone() function prior to calling the tone() function.

The tone() function usage:

```
tone(pin, frequency)

tone(pin, frequency, duration)

pin:  Designated pin to generate the signal output

frequency:  Output frequency, as an unsigned integer.
(Human can hear tone within the 20 ~ 20,000 Hz range)

duration:  Output duration in millisecond, as an unsigned
      long variable. (Optional)
```

First, construct the circuit as shown in Figure-3:

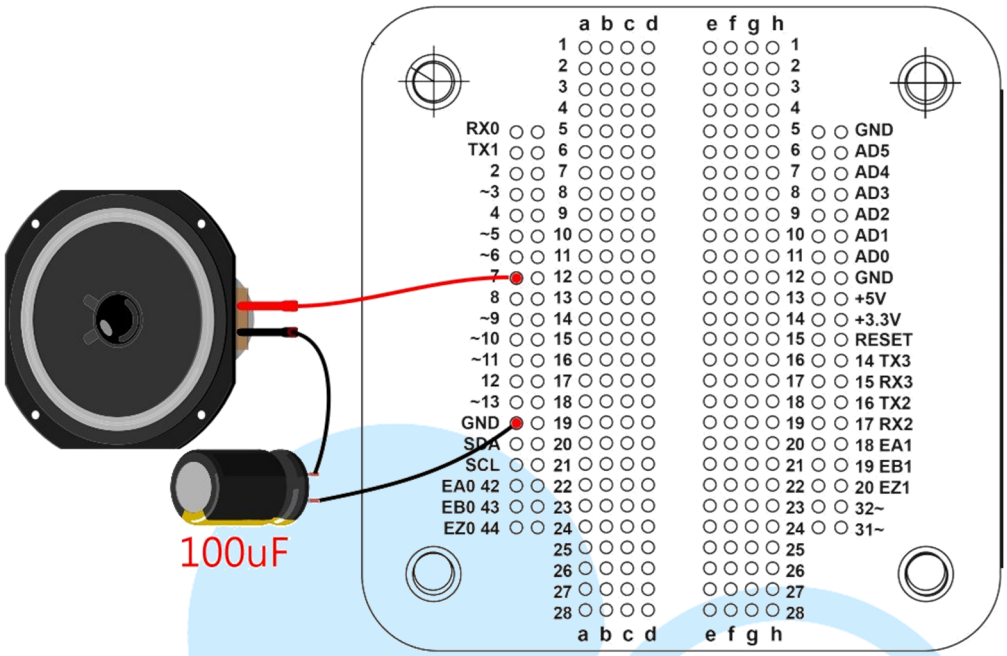

Figure-3. Single speaker circuit

In the above circuit, the 100μF capacitor is used as filter. It' s also common for some people to use a resistor instead. Since the resistance for the buzzer or speaker is low, placing a resistor in series help lower the current flow to avoid damaging the board. However, the additional resistor will lower the output volume. You can experiment different methods to control audio output, such as connecting different size capacitor in series or parallel, different resistance in series, use different type or size of buzzer/speaker or use transistor to amplify current to increase audio volume and etc.

In today' s market, there are different option and technology you can use to generate audio with different quality. Using vibration resonance speaker, you can turn just about any surface into a speaker. Or, you can use speaker design based on the popular Nautilus speaker design to output high quality audio. With a buzzer/speaker attached to Pin #7, as shown in Figure-3, you can use the following codes to generate audio output:

```
void setup()
{   }
void loop()
{
  tone(7, 500); // Output 500Hz tone using Pin-7
  delay(500); // Delay 500ms, while Pin-7 output continue
  noTone(7); // Turn off Pin-7 tone output
  delay(500);
}
```

Next, we will alter the circuit to simulate left and right audio channels, using resister to control audio volume, as shown in Figure-4:

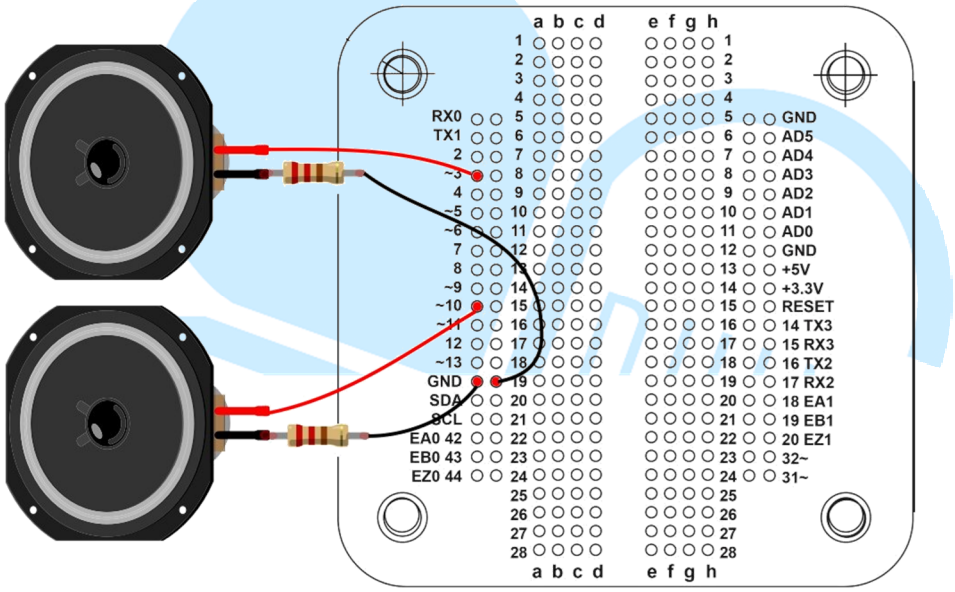

Figure-4. Dual channels speaker circuit

Modify the code to generate audio output to both speaker, alternating between left-channel and right-channel speaker, as shown below:

```
int left_sound=10;
int right_sound=3;
void setup()
{   }
void loop()
{
  tone(left_sound, 500); // Output 500Hz tone to left channel
  delay(500); // Delay 500ms, while left-channel output
continue
  noTone(left_sound); // turn off left-channel output
  tone(right_sound, 500); // Output 500Hz tone to right channel
  delay(500); // Delay 500ms, while right-channel output
continue
  noTone(right_sound); // turn off right-channel output
}
```

In the above circuit, Figure-4, you can control audio output volume by changing the circuitry with adjustable resistance. With appropriate electronic components and codes to control the circuit, you can dynamically control audio output remotely from a PC, through the serial interface, and create an application on the PC with UI and slider control, which enable you to control audio output using the mouse.

3. Musical Note and Beat

To play with music, we need to have some basic understanding about musical note and beat, where a musical note specify the frequency and the beat specify the pace (speed) to play the musical notes. In the musical world, "180 beat per minute" represent playing 180 different musical notes within a minute, which is referred to as the tempo.

The higher value for beat the faster pace the music is playing. For music playing at a slower pace, the value for the beat is lower. The beat can be used to control the pace, or tempo, to play musical notes. In the earlier examples, the time delay between calling the tone() and notone() function control how fast the next tone is played after the current tone, which is the beat or tempo.

Musical note is a bit more complicated. There are 7 notes, C, D, E, F, G, A, B and 5 half-notes, C#, D#, F#, G# and A#, which make up a total of 12 musical notes. These musical notes can be generated at lower or higher pitch in different frequency range. Using piano key as reference, which include musical notes from 9 different octave (octave 0 to 8), following is a list of frequency for the 7 musical notes in 3 different octave:

- The following frequencies generate C, D, E, F, G, A, B musical notes equivalent to octave 4 on a full size piano:

  262, 294, 330, 349, 392, 440, 494

- The following frequencies generate C, D, E, F, G, A, B musical notes equivalent to octave 5 on a full size piano:

  523, 587, 659, 784, 880, 988

- The following frequencies generate C, D, E, F, G, A, B musical notes equivalent to octave 6 on a full size piano:

  1046, 1175, 1318, 1397, 1568, 1760, 1976

Musical note at a higher octave has a higher pitch sound. It' s not within this application note' s objective to talk about music theory. For more information, refer to the following URL:

http://en.wikipedia.org/wiki/Note

The following codes output the 7 musical notes at lower pitch:

```
// Assign the 7 frequency associate with musical notes
// to an array
int frequency[]={262,294,330,349,392,440,494};

void setup()
{
}
void loop()
{
  int a;
  for (a=0;a<7;a++)
  {
    tone(7, frequency[a]); // Generate output through
Pin-7
    delay(500); // Delay 500 ms
    noTone(7);
  }
}
```

While the sound output is different when the output is attached to a small speaker, larger speaker or a buzzer, you can hear distinctively the 7 musical notes.

To hear the 7 musical notes at different pitch, change the frequency array as follow:

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

And, change the for loop to the following:

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

With the above modification, the 7 musical notes in three different pitch (total 21 notes) will be generated. Following the musical notes sequences and beat from a music sheet and map the notes to a music frequency table, you can use the EduCake to play music.

4. High-Low Tone and Siren

In the previous section, we talked about generating tones and musical notes. In this section, let's talk about sound effects.

Using EduCake, we can generate different sound effects that fluctuate between different frequencies, continuously or intermittently, switching between higher and lower pitch or generating random tones programmatically.

The audible audio range for human is between the 20 ~ 20,000Hz frequency range. The codes in the following listing is designed to generate tone within this range:

```
void setup()
{
}
void loop()
{
  int a;

  // Generate tone at 10 Hz increment
  // Starting from 20 Hz, within the 20 - 20,000 range
  for(a=20;a<20000;a+=10)
  {
    tone(7,a);
    delay(5);
  }
  noTone(7);
}
```

Based on the experiment we have done, using a common 6 cm PC speaker, the tone is not audible above 12,000 Hz. To output audible higher pitch tone at higher frequency, we need to use a smaller speaker.

Next, let's look at the following code that output beep sound:

```
void setup()
{
}
void loop()
{
   tone(7 ,900); // use a frequency other than 900 to
output
                 // different sound
    delay(200); // Delay time is used to control beeping
speed
    noTone(7);
    delay(300);
}
```

From the above code, we can see the beeping sound is generated by switching tone on and off. By varying the delay time, we can generate beeping sound with different effects. By encapsulating the above codes in a loop, with additional code to create a accelerated sound effect, we can create sound that simulate something falling, as follow:

```
void setup()
{
}
void loop()
{
  int a;
  // Assuming an object is falling from 3500 meter
  // The initial traveling speed is zero
  // using 10 as the gravity acceleration value
  double v=0,s=3500;
  while (s>50) // Falling object stop at 50 meters
  {
   tone(7,s); // Using the height from the falling object
as
              //  frequency for output, represent by s
    delay(30);
    noTone(7);
    s-=v; // Update the current height
    v+=10; // gravity acceleration adjustment
  }
}
```

By changing some of the parameters used in the above code, we can generate interesting sound effect, such as changing the "tone (7, s);" function call to "tone (7, 3500-s);", which will generate sound effect that simulate object moving upward. Varying the V+=10 gravity

acceleration and delay() parameters also trigger changes to the sound effect. We can generate even more interesting sound effect by adding trigonometry components into the code, as follow:

```
void setup()
{
}
void loop()
{
  double a;
  for(a=0;a<100;a+=0.15)
  {
    tone(7, 1000+500*sin(a));
    //Circumference=2 pi =2x3.14=6.28, approximate 100
   // Result from sin(a) function fluctuate between +1
to -1
   // After multiply by 500 with 1000 added
    // The frequency alternate within the 500~1500
sine-wave
    delay(50);
  }
  noTone(7);
}
```

With the added trigonometry component, the sound effect is quite different from the falling/rising sound effect in the previous section. Different Sine-wave function trigger different sound effects, such as cos(), tan() and etc. These high-low sound effects are similar to the emergency siren used by police, ambulance and fire truck, which we routinely hear as part of the daily living in any major metropolitan city.

Let’ s take a look at the following code fragment:

```
int spd =10; // spd value directly affect the looping speed
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);
}
```

By changing the delay and looping parameters in the above code fragment, it can be modified to simulate high-low tone similar to the police and ambulance siren. In real life, the actual siren we hear typically travel rapidly from far to near or moving away rapidly, which has Doppler Effect and change the siren' s frequency slightly.

To simulate real life siren traveling from far to near, we need to gradually increase the siren frequencies (both high and low). To simulate siren moving away, we need to gradually decrease the siren frequencies.

To learn more about Doppler Effect, visit the following URL:

http://simple.wikipedia.org/wiki/Doppler_effect

The code below uses random number as frequency and generate unpredictable sound, which can sound like the robotic voice we hear in the movie.

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

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

In the above code, the "random (100, 2000)" function generate a random number within the 100 to 1999 range, which is used as the frequency for the tone() function. Changing these 2 values and delay time for the delay() function will change the above code to generate some strange sound. Using the above technique, it's possible to generate just about any sound.

5. Playing a Song and Analyzing Music Sheet

In the previous section, we talked about different audio output. In this section, we use an EduCake to play an actual song. To play a song, we need the music sheet for the song. Since many of the songs have copyright and cannot be used without permission from the copyright owner, we need to find song that is in the public domain, not affected by copyright. To avoid copyright issue, we will use an old Chinese folk song for the exercise in this section.

It' s not the intention for this application note to teach music and we don' t need to have full understanding about music theory to understand the sample in this section, just need to know how to read the musical note on the music sheet. As shown in Figure-5, there rows of musical notation represented by numeric number, with number ranging from 1 through 7, which correspond to Do, Re, Me, Fa, So, La Se or C, D, E, F, G, A, B. Using a full size piano as reference, which has 88 key that produce musical notes that span 9 different octave, octave-0 through octave-8, the equivalent numerical notes 1, 2, 3, 4, 5, 6 and 7 represent C4, D4, E4, F4, G4, A4 and B4. Numerical notes with a dot below the number are one octave lower and numerical notes with underline are one octave higher. Numerical note with a dash after represent the note is play at an extended duration, where each dash is equivalent to one beat.

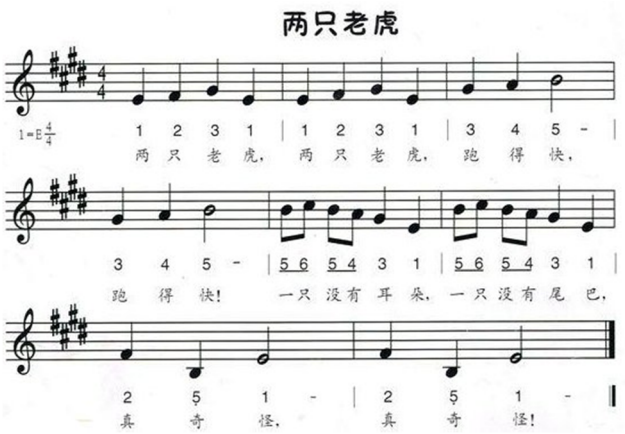

Figure-5. Music sheet for an old Chinese folk song

With the above music sheet, we need to convert the musical notes into digital format in order to play the song using EduCake.

Based on the musical note and associated frequency discussed in earlier section, the digital equivalent for musical note sequence from the song and the beat for each of the notes are replicated into two separate arrays, as shown in the following code listing:

```
// Index for corresponding frequency for each musical notes
in
// sequence from the song sheet
byte tigerTone[]=
{7,8,9,7,7,8,9,7,9,10,11, 9,10,11,18,19,18,17,9,7,
18,19,18,17,9,7,8,4,7,8,4,7};

// Corresponding beat for each of the musical notes,
// from the music sheet, which control the duration
// to play each of the musical notes
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};
```

In earlier exercise, we created the frequency[] array, which contain frequencies for musical notes in 3 different octave, total of 21 notes. Using the value from the tigerTone() array as the array index for the frequency[] array, we can programmatically control EduCake to play the song, using the following code:

```
const int speaker=13; // Use Pin-13 to general audio
output

// array contains frequencies for 21 musical notes
int frequency[]={
  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};
```

```
// Length of the array containing musical notes for the
song
const int playLen=sizeof(tigerTone);

void setup()
{
}
void loop()
{
  int a;
  for(a=0;a< playLen -1;a++)
  {
   //  As the coding looping through the tigerTone[] array,
   //  Play the tone from the frequency[] array
   //  Using the value from tigerTone[a] as the index
   tone(speaker,frequency[tigerTone[a]]);

   // Using the delay() function to control play duration
   //  for each note
   // Base on the beat from the corresponding tigerBeat[]
array
   delay(300* tigerBeat[a]);
   // The 300 msec above represent time duration for 1 beat
   //  where there are 180 beats in a min,
   //  Which is approximately 300 msec for each beat
   noTone(speaker); // Turn off tone playback and
                    //  wait for next musical note
  }
  delay(3000);
}
```

The above code play the song form the music sheet continuously, as shown in Figure-5. After finish playing the song, it wait for 3 seconds and then loop back and play the song again.

The code above can be expanded to turn the EduCake into a music playback device, with multiple push button to select songs, simple LCD display to show operating status where the complete solution can be packaged onto an SD storage card with storage space for the program and collection of music files. By adding LCD touchscreen to the solution, it' s possible to turn the solution into a song selector user interface for karaoke sing along system.

6. Multiple Songs Playback

In the previous section, we talked about the song selector interface for karaoke sing along system. In this section, we will talk about a simple implementation of song selector application, using button to select song for playback.

For the multi songs selection exercise in this section, we will use 4 Chinese folk songs, "Two tigers", "Little star", "Little bee" and "Little donkey". We will create two arrays for each of the 4 songs, one for the musical note sequence and the other for the corresponding beat for each of the musical notes, as follow:

```
// Array containing frequencies for the 7 music notes in
3
// different pitch (octave), total of 21 notes

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

// Musical note sequence for the song: Two tigers
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};

// corresponding beat for the song:  Two tigers
byte tigerBeat[]={1,1,1,1,1,1,1,1,1,1,2,1,1,2,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1, 2,1,1,2};

// Length for the Two tigers song
int  tigerLen =sizeof(tigerTone) ;

// Musical note sequence for the song: Little bee
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};
// corresponding beat for the song:  Little bee
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};
```

```
// Length for the Little bee song
int beeLen=sizeof(beeTone) ;

// Musical note sequence for the song: Little star
byte
starTone[]={7,7,11,11,12,12,11,10,10,9,9,8,8,7,11,11,1
0,10,9,9,8,11,11,10,
10,9,9,8,7,7,11,11,12,12,11,10,10,9,9,8,8,7};

// corresponding beat for the song: Little star
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};

// Length for the Little star song
int starLen=sizeof(starTone) ;

// Musical note sequence for the song: Little donkey
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};

// corresponding beat for the song: Little donkey
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};

// Length for the Little donkey song
int donLen=sizeof(donTone) ;
```

With the above arrays that contain the digital equivalent of the 4 songs, we can programmatically play these songs. Using an SD storage card that can store large number of digital data, we can store these songs in electronic format on to the SD storage card and create an application to play back these songs from the SD card.

In the following exercise, we will use 4 push buttons map to the 4 songs. When a button is pressed, the program will play back the corresponding song, as shown in the following code listing:

```
int play_no=-1; // Variable used to identify the song currently
                //  playing, -1 represent music is not playing

int play_pos=0; // Variable used to identify the progress or
                //  position of the notes currently playing.

int is_play=0; // Variable to identify whether music is playing

int bb[4]={2,3,4,5}; // Pin # attached to the 4 buttons
int bb_state[4]; // Variable to indicate the button status

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

void loop()
{
  int a;

  // Check to see whether any of the buttons is pressed
  for(a=0;a<4;a++)
   {
   bb_state[a]=digitalRead(bb[a]);  //Log button status to array
   }

// Next, the code check the button status, assuming 1 button is
// pressed and play the corresponding song
// When 2 buttons are pressed at the same time,
// the two corresponding songs can play back at the same time.
// However, when 2 songs are play at the same time, it's just
// noises that does not reflect any one of the song, which we
// are not going to do here.

  if (bb_state[0]==0) // Button 1 pressed, play the 1st song
  {
    // Output relevant info to the serial monitor
    Serial.println(beeLen);

    play_no=1;   // Set to play the 1st song
    play_pos=0;  // Set to play from the beginning
    is_play=1;  // Set to 1 to play, set to 0 to stop playing
    Serial.println("bee playing...");
  }
  else if (bb_state[1]==0) // Button 2 pressed, play the 2nd song
  {
    // Output relevant info to the serial monitor
    Serial.println(starLen);

    play_no=2;   // Set to play the 2nd song
```

```
    play_pos=0;  // Set to play from the beginning
      is_play=1;  // Set to 1 to play, set to 0 to stop playing
      Serial.println("star playing...");
    }
    else if (bb_state[2]==0) // Button 3 pressed, play the 3rd song
    {
      Serial.println(tigerLen);
      play_no=3;   // Set to play the 3rd song
      play_pos=0;  // Set to play from the beginning
      is_play=1;  // Set to 1 to play, set to 0 to stop playing
      Serial.println("tiger playing...");
    }
    else if (bb_state[3]==0) // Button 4 pressed, play the 4th song
    {
      Serial.println(donLen);
      play_no=4;   // Set to play the 4th song
      play_pos=0;  // Set to play from the beginning
      is_play=1;  // Set to 1 to play, set to 0 to stop playing
      Serial.println("don playing...");
    }

  playSong(); // Function to play the song
    delay(5);
  }

  // This function play one single note at specified beat
  void play (byte toneNo, byte  beatNo)
  {
    tone(speaker,frequency[toneNo]);
    delay(300* beatNo);
    noTone(speaker);
  }

  // This function is called to play the song
  void playSong()
  {
    if (is_play==1) // Check to see whether the song is set to play
    {
      switch(play_no)  // Switch statement to decide the song to play
      {
      case 1://beeTone
        if  (play_pos>=beeLen) // check to see whether end of song
        {
          is_play=0; // end of song detected, set to stop playing
          return ; //
        }
        //Not end of song, play current musical note
        play (beeTone[play_pos],beeBeat[play_pos]);
        play_pos++; //Increment to the next musical note position
        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;
      }
    }
  }
```

With above code, when one of the 4 buttons is pressed, it will playback the corresponding song from the beginning and stop when reaching the end of the song. While playing, you can press a different button and change to play a different song. With additional code and hardware, you can expand the above code to provide more interesting feature, such as automatically play the next song in sequence after the current song is finished, using infrared interface to remotely control music playback, press a button to keep on repeating the current song and etc.

7. Change Playback Speed, Pitch & Audio Mix

Continue to the basic and multiple song playback technique covered in the previous section, there are many other techniques we can use to create fun and interesting audio function.

At many music and entertainment performance stage, we often see an audio control center with expensive sound synthesizer and audio control equipment, with lots of buttons and different type of controls, operate by a sound technician. During a performance, the audio control center is responsible for playing appropriate music when the performance is idle, control the volume and quality of the audio during the performance, inject and mix different type sound effect, with the music, as part of the performance to deliver the intended sound effects, such as animal sound, sound that simulate collision between different things, metallic sound and etc.

While the low-cost Arduino grade hardware, including the 86Duino EduCake, is not designed to deliver high resolution audio playback and sound effects as the professional sound equipment, with the right electronic components and application code, you can control playback speed, change the pitch of the sound and mix multiple audio during playback using the EduCake.

There are different techniques to mix multiple audio at different frequencies, by adding the two frequencies then divide by 2, combining the two frequencies and multiple by a certain factor, adding one audio source at higher volume with another audio source at lower volume, and etc. When mixing multiple audio streams, you can achieve better result at higher sampling rate. For the samples in the previous section, the audio playback about 3 notes per second, equivalent to 3 Hz/sec sampling rate. Music we listen to typically is in the range of 22K Hz/Sec sampling rate or even higher. Dealing with different type of audio with different quality is a complex subject, and is not the scope for this application note to cover.

Within the "void play(byte toneNo, byte beatNo)" function, the beatNo variable can be adjusted to controls playback speed. For the sample for this section, the play_spd variable is used to control playback speed.

With regard to audio volume, you can control the volume by varying the output power. Since the output current from the I/O pin on the EduCake is low, it's range is limited to change audio volume that is noticeable to human hearing. We need an audio amplifier module, as shown in Figure-6, to produce higher audio volume.

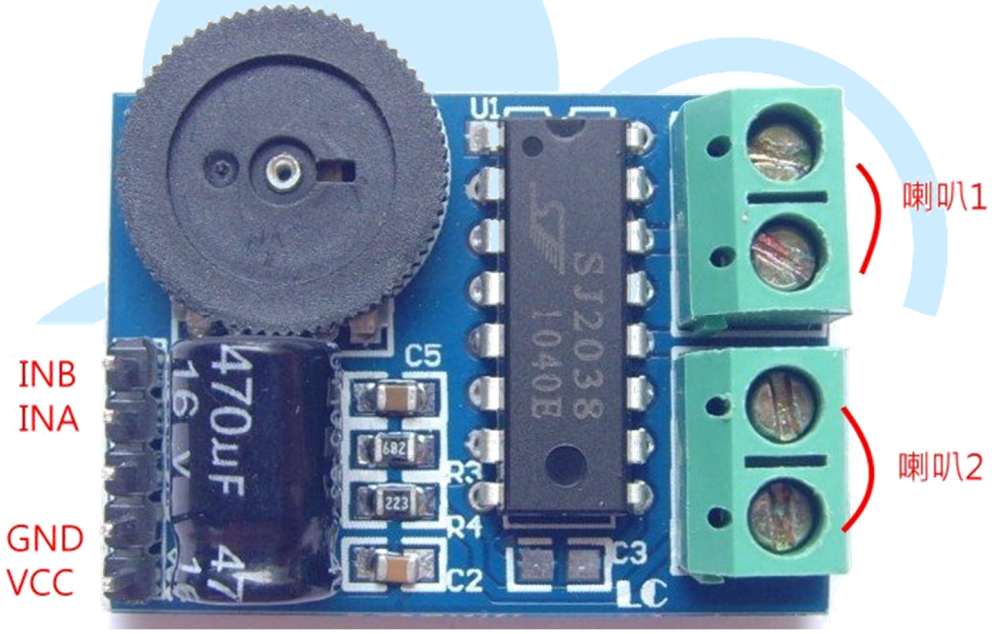

Figure-6. Audio amplifier module

To change the audio's pitch during playback, we need to incorporate a variable to change the pitch. There are different methods to alter the pitch during playback. For the exercise in this section, we can use the following two methods:

1. Change all the frequencies in the array that contains the 21 frequencies that represent 7 musical notes in 3 different pitch, at the same proportion to raise or lower the pitch for all 21 musical notes.

2. Use the play_keys variable to adjust the pitch during playback, by adding or subtracting the play_keys variable' s value with the playback frequency, where the play_keys value is within the -300 to 600 range.

Use the circuitry as shown in Figure-7 to work through the exercise in the section.

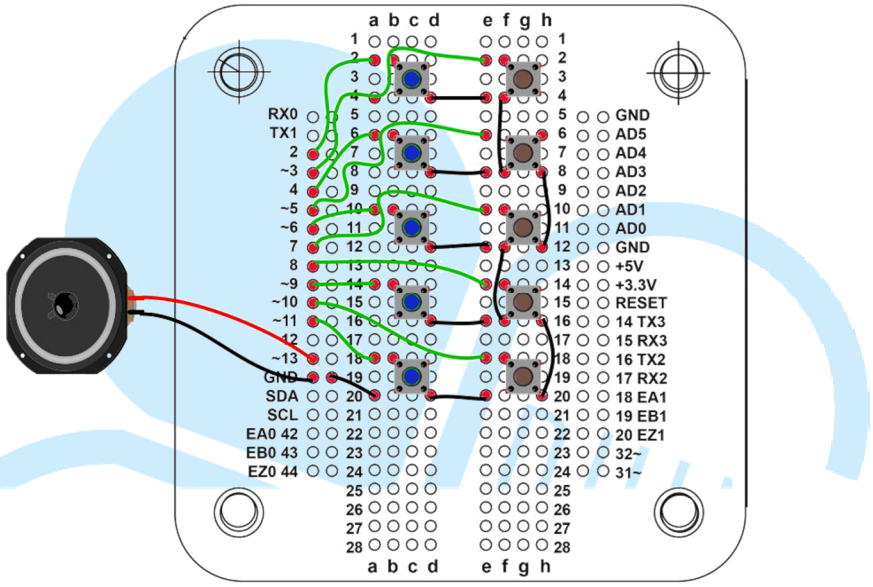

Figure-7. Circuitry for the exercise.

Following is the code listing for the exercise:

```
int speaker_pin=13; // Configure Pin-13 as audio output

// Array contains frequencies for the 7 musical notes
// in 3 different octave, total 21 musical notes
int frequency[]={
  262,294,330,349,392,440,494,
  523,587,659,698,784,880,988,
  1046,1175,1318,1397,1568,1760,1976};

// Musical note sequence for song:  Two tigers
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};

// corresponding beat for song:  Two tigers
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};

// Length of song:  Two tigers
int  tigerLen =sizeof(tigerTone) ;

// Musical note sequence for song:  Little bee
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};

// corresponding beat for song:  Little bee
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};

// Length of song:  Little bee
int beeLen=sizeof(beeTone) ;
// Musical note sequence for song:  Little star
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};

// corresponding beat for song:  Little star
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};
// Length of song:  Little star
int starLen=sizeof(starTone) ;

// Musical note sequence for song:  Little donkey
byte
```

```
donTone[]={7,7,7,9,11,11,11,11,12,12,12,13,11,10,10,12,1
2,
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};

// corresponding beat for song:  Little donkey
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};

// Length of song:  Little donkey
int donLen=sizeof(donTone) ;

int play_spd =0; // Variable to control playback speed
int play_no=-1; // Variable to identify the song currently
playing
                //  play_no = -1 represent not playing
int play_pos=0; // Variable to identify current playback
position
int is_play=0; // Variable to indicate whether playback is
active
int play_keys=0; // Variable to adjust sound pitch during
playback
                 // range for play_keys variable is
-300~+600

// Array to configure Pin# assigned to each of the 10 buttons
// button 1 to 4 to playback song 1 through 4
// button 5 and 6 to control pitch during playback
// button 7 and 8 to control playback speed
// button 9 and 10 to inject sound effects
int bb[10]={2,3,4,5,6,7,8,9,10,11};

// Status indicator for each of the 10 buttons
int bb_state[10];


void setup()
{
  int a;
  Serial.begin(9600);
  for(a=0;a<10;a++)
    // Use internal Pull-up resistor to configure and
    // initialize each of the 10 buttons
    pinMode(bb[a], INPUT_PULLUP);
}
```

```
void loop()
{
  int a;

  // Scan and record current status for the 10 buttons
  for(a=0;a<10;a++)
     bb_state[a]=digitalRead(bb[a]);

  // Output current status for the buttons to serial
monitor
  for(a=0;a<10;a++)
  {
    Serial.print( bb_state[a]);
    Serial.print(", ");
  }
  Serial.println("");

  if (bb_state[0]==0) //Play the 1st song
  {
   Serial.println(beeLen);// Output length of the current
                          // song via serial monitor
    play_spd =0; // initialize playback speed
    play_no=1;  // Variable to indicate 1st song is playing
    play_pos=0;  // Set to play from the beginning
    play_keys=0; // Initialize pitch control variable
    is_play=1; // Set to "1" to play and "0" to stop playing
    Serial.println("bee playing...");
  }

  else if (bb_state[1]==0) // Play the 2nd song
  {
    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) // Play the 3rd song
  {
    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) // Play the 4th song
{
  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) // Raise audio pitch
{
 if ( play_keys<600)
   play_keys+=50;
 else
   // 600 is the maximum value to deviate the pitch
   play_keys=600;
}
else if  (bb_state[5]==0) // Lower audio pitch
{
  if (play_keys<-300)
    play_keys=-300;
  else
    play_keys-=50;
}

else if  (bb_state[6]==0)// decrease playback speed
{
  play_spd +=100;
  if (play_spd >1000)
    play_spd =1000;
}

else if  (bb_state[7]==0)// increase playback speed
{
  play_spd -=50;
  if (play_spd <-200)
    play_spd =-200;
}

// inject 1st sound effect, about 0.6 second
else if  (bb_state[8]==0)
{
  for(a=0;a<5;a++)  // generate low frequency sound
  {
    tone(speaker, 150);
    delay(50);
    tone(speaker, 250);
    delay(50);
```

```
    }
   }

  // inject 2nd sound effect, about 0.6 second
  else if  (bb_state[9]==0)
  {
    for(a=1000;a<1600;a+=20) // Generate high frequency sound
    {
      tone(speaker, a);
      delay(20);
    }
  }

  playSong();// Function to playback song
  delay(2);
}

// Function to playback a single musical note
//
void play (byte toneNo, byte  beatNo)
{
  // Combine tone frequency with the pitch adjustment value
  int pk=play_keys+frequency[toneNo];

  tone(speaker, pk);

  // play_spd value added to the delay to control playback
speed
  delay(300* beatNo+ play_spd);
  noTone(speaker);
}

// Function to playback sequence of musical notes in a song
void playSong()
{
  if (is_play==1) //  stop playing when is_play not equal to
1
  {
    // Switch statement to select song for playback
    switch(play_no)
    {

   case 1:  // Play the song Little bee
      if  (play_pos>=beeLen) // Check whether is end of song
     {
       is_play=0; // End of song, set is_play to 0 to stop
       return ;
     }
```

```
    // Not end of song, continue to play
       play (beeTone[play_pos],beeBeat[play_pos]);
       play_pos++; // Increase the current play position to next
       break;

  case 2:  // Play the song Little star
       if  (play_pos>=starLen)
       {
         is_play=0;
         return ;
       }
       play (starTone[play_pos],starBeat[play_pos]);
       play_pos++;
       break;

     case 3:  // Play the song Two tigers
       if  (play_pos>=tigerLen)
       {
         is_play=0;
         return ;
       }
       play (tigerTone[play_pos],tigerBeat[play_pos]);
       play_pos++;
       break;

     case 4:  // Play the song Little donkey
       if  (play_pos>=donLen)
       {
         is_play=0;
         return ;
       }
       play (donTone[play_pos],donBeat[play_pos]);
       play_pos++;
       break;
     }
   }
 }
```