iDesign Studio ~ Lab 3: Output via buzzer

written by Shani Mensing '15

In this lab, we will learn about the **tone()** function supported by the Arduino platform. We will start by writing a program that causes the SquareWear board to play a tone through its onboard buzzer.

**A little bit about code...**

- **Comments** are used by programmers to explain what code should be doing; they are ignored by the computer/microcontroller.
  - **Single line comments** are specified by placing // at the beginning of the line, as in:
    ```
    // this is a comment
    ```
  - **Multi-line comments** are specified by starting with a /* and ending with a */, as in:
    ```
    /* This is the start
    of a multiple line comment.
    And this is the end */
    ```
- A **constant** can be declared to hold a specific value, such as a pin for output
  - examples:
    ```
    #define RED 8
    #define BUZZER_PIN 9
    ```
- The Arduino uses a Java-like programming language. You will see **statements**, punctuated by a semi-colon ;
- A **variable** is used to specify a value that may change
  - We can **declare** a variable with its **type** and name, as in:
    ```
    // an integer variable that will count the number of button presses
    int numberOfButtonPresses;
    ```
  - We can **assign** a value into the variable with the = operator, as in:
    ```
    // start off with no button presses
    numberOfButtonPresses = 0;
    ```
  - We can **both declare and assign** a value in a single statement, as in:
    ```
    // an integer variable that will count the number of button presses
    // starting with the value 0
    int numberOfButtonPresses = 0;
    ```
- A **method** is used to encapsulate behavior and is composed of 2 parts:
  - the **method declaration**, with a **return type** and **parameters**, as in
    ```
    // a method that returns an integer and requires two parameters
    int add( int x, int y )
    ```
  - the **method definition**, full of statements follows the declaration in {} in the code below, the **definition** is in bold
    ```
    // a method that returns an integer and requires two parameters
    int add( int x, int y )
    {
      // create a variable, assigning the result of adding the two parameters
      int sum = x + y;
      // output the computed sum
      return sum;
    }
    ```
- Methods (written by us or others) can be used by **invoking** or **calling** the method, as in the statement below; the **method invocation** is in bold:
  ```
  // invoke the method add, passing 3 and 4 as the parameters
  // save the value returned by the method in a variable
  int sumOf3and4 = add( 3, 4 );
  ```
Step 1

- Open the Arduino app and create a new sketch from the File menu -> New.
- We will use the tone function, described here: [http://arduino.cc/en/Reference/Tone](http://arduino.cc/en/Reference/Tone)
- Paste the following code into the sketch

```c
/*
Single tone

Plays a single tone on SquareWear

* use buzzer on digital pin 9
*/

// the pin we will use for output
// specific to SquareWear
#define BUZZER_PIN 9

/**
* The code in this special method is executed
* once when the microcontroller is turned on
* (or the program is uploaded).
**/
void setup()
{
  // play a single note
  // first parameter is the pin number to output (the buzzer)
  // second is the note frequency
  // third is how long to play the note (in milliseconds)
  tone( BUZZER_PIN, 262, 1000 );
}

/**
* The code in this special method is constantly executed
* after setup has occurred.
**/
void loop()
{
  // it will be annoying to constantly hear a tone,
  // so let's do nothing!
}
```

- Save as `singleTone`
- Now upload to your SquareWear – what happens? Try turning it off and on also.
Step 2

We will now modify the program above to respond to button presses.

- Create a new sketch from the File menu -> New.
- Paste the following code into the sketch

```c
/*
* Single tone from button
Plays a single tone on SquareWear in response to a button press
* use buzzer on digital pin 9 for output
* and button on pin for input
*/

// the pin we will use for output
#define BUZZER_PIN 9

// the pin for button input
#define BUTTON_PIN 4

void setup()
{
  // specify that we will use the buzzer for output
  pinMode(BUZZER_PIN, OUTPUT);
  digitalWrite(BUTTON_PIN, HIGH);

  // specifies that we will use the button as input
  pinMode(BUTTON_PIN, INPUT);
  digitalWrite(BUTTON_PIN, HIGH);
}

/*
* The code in this special method is executed
* once when the microcontroller is turned on
* (or the program is uploaded).
*/
void loop()
{
  if (digitalRead(BUTTON_PIN) == LOW)
  {
    // if so, play the note
    playNote();
  }
  else
  {
    // in case a note was playing.
    // stop playing the note
    noTone( BUZZER_PIN );
  }

  /* Define a method that will play a single note.
  */
void playNote()
{
  // play a single note
  // first parameter is the pin number to output (the buzzer)
  // second parameter is the note frequency
  // without the third parameter, just keep playing the note
  // tone( BUZZER_PIN, 262 );
}
```

- Save as `singleToneButton`
- Now upload to your SquareWear – what happens? Try pressing the button 😊
Step 3
What about notes that are more intuitive?

- Create a new sketch from the File menu -> New.
- Paste the following code into the sketch

```c
/*
Melody
Plays a melody on SquareWear

* use buzzer on digital pin 9

created 21 Jan 2010
modified 30 Aug 2011
by Tom Igoe
modified 24 Sep 2013
by Audrey St. John

From example code in the public domain.
http://arduino.cc/en/Tutorial/Tone */

// the pin we will use for output
// specific to SquareWear
#define BUZZER_PIN 9

// include more intuitive notes
#include "pitches.h"

// notes in the melody:
int melody[] =
{ NOTE_C4, NOTE_G3,NOTE_G3,NOTE_A3,NOTE_G3,0, NOTE_B3, NOTE_C4};

// the number of notes to play
int melodyLength = 8;

// note durations: 4 = quarter note, 8 = eighth note, etc.:
int noteDurations[] = { 4, 8, 8, 4,4,4,4 };

/**
* The code in this special method is executed
* once when the microcontroller is turned on
* (or the program is uploaded).
*/

void setup()
{
  // for each note in the melody:
  for (int noteIndex = 0; noteIndex < melodyLength; noteIndex++)
  {
    // to calculate the note duration, take one second
    // divided by the note type.
    // e.g. quarter note = 1000 / 4, eighth note = 1000/8, etc.
    int noteDuration = 1000/noteDurations[noteIndex];

    // play the note for the calculated number of milliseconds
    tone(BUZZER_PIN, melody[noteIndex], noteDuration);

    // to distinguish the notes, set a minimum time between them.
    // the note's duration * 30% seems to work well:
    int pauseBetweenNotes = noteDuration * 1.30;

    // wait for calculated milliseconds
delay(pauseBetweenNotes);
  }
}

/**
* The code in this special method is constantly executed
* after setup has occurred.
*/

void loop()
{
  // it will be annoying to constantly hear the melody,
  // so let's do nothing!
}

/**
* Save as shortMelody
*/
```
• Now create a new tab by using the little arrow on the top of your sketch window

Now create a new tab by using the little arrow on the top of your sketch window

• and call it Pitches.h

• Paste in the following code which defines more intuitive notes

```c
//****************************************************************************
// Public Constants
//****************************************************************************
#define NOTE_B0 31
#define NOTE_C1 33
#define NOTE_CS1 35
#define NOTE_D1 37
#define NOTE_DS1 39
#define NOTE_E1 41
#define NOTE_F1 44
#define NOTE_FS1 46
#define NOTE_G1 49
#define NOTE_GS1 52
#define NOTE_A1 55
#define NOTE_AS1 58
#define NOTE_B1 62
#define NOTE_C2 65
#define NOTE_CS2 69
#define NOTE_D2 73
#define NOTE_DS2 78
#define NOTE_E2 82
#define NOTE_F2 87
#define NOTE_FS2 93
#define NOTE_G2 98
#define NOTE_GS2 104
#define NOTE_A2 110
#define NOTE_AS2 117
#define NOTE_B2 123
#define NOTE_C3 131
#define NOTE_CS3 139
#define NOTE_D3 147
#define NOTE_DS3 156
#define NOTE_E3 165
#define NOTE_F3 175
#define NOTE_FS3 185
#define NOTE_G3 196
#define NOTE_GS3 208
#define NOTE_A3 220
#define NOTE_AS3 233
#define NOTE_B3 247
#define NOTE_C4 262
#define NOTE_CS4 277
#define NOTE_D4 294
#define NOTE_DS4 311
#define NOTE_E4 330
#define NOTE_F4 349
#define NOTE_FS4 370
#define NOTE_G4 392
#define NOTE_GS4 415
#define NOTE_A4 440
#define NOTE_AS4 466
#define NOTE_B4 494
#define NOTE_C5 523
#define NOTE_CS5 554
#define NOTE_D5 587
#define NOTE_DS5 622
#define NOTE_E5 659
#define NOTE_F5 698
#define NOTE_FS5 740
#define NOTE_G5 784
#define NOTE_GS5 831
#define NOTE_A5 880
#define NOTE_AS5 932
#define NOTE_B5 988
#define NOTE_C6 1047
#define NOTE_CS6 1109
#define NOTE_D6 1175
#define NOTE_DS6 1245
#define NOTE_E6 1319
#define NOTE_F6 1397
#define NOTE_FS6 1480
#define NOTE_G6 1568
#define NOTE_GS6 1661
#define NOTE_A6 1760
#define NOTE_AS6 1865
#define NOTE_B6 1976
#define NOTE_C7 2093
#define NOTE_CS7 2217
#define NOTE_D7 2349
#define NOTE_DS7 2489
#define NOTE_E7 2637
#define NOTE_F7 2794
#define NOTE_FS7 2960
#define NOTE_G7 3136
#define NOTE_GS7 3322
#define NOTE_A7 3520
#define NOTE_AS7 3729
#define NOTE_B7 3951
#define NOTE_C8 4186
#define NOTE_CS8 4435
#define NOTE_D8 4699
#define NOTE_DS8 4978
#define NOTE_E8 523
#define NOTE_F8 554
#define NOTE_FS8 587
#define NOTE_G8 622
#define NOTE_GS8 659
#define NOTE_A8 698
#define NOTE_AS8 740
#define NOTE_B8 784
#define NOTE_C9 831
#define NOTE_CS9 880
#define NOTE_D9 932
#define NOTE_DS9 988
#define NOTE_E9 1047
#define NOTE_F9 1109
#define NOTE_FS9 1175
#define NOTE_G9 1245
#define NOTE_GS9 1319
#define NOTE_A9 1397
#define NOTE_AS9 1480
#define NOTE_B9 1568
#define NOTE_C10 1661
#define NOTE_CS10 1760
#define NOTE_D10 1865
#define NOTE_DS10 1976
#define NOTE_E10 2093
#define NOTE_F10 2217
#define NOTE_FS10 2349
#define NOTE_G10 2489
#define NOTE_GS10 2637
#define NOTE_A10 2794
#define NOTE_AS10 2960
#define NOTE_B10 3136
#define NOTE_C11 3322
#define NOTE_CS11 3520
#define NOTE_D11 3729
#define NOTE_DS11 3951
#define NOTE_E11 4186
#define NOTE_F11 4435
#define NOTE_FS11 4699
#define NOTE_G11 4978
#define NOTE_GS11 523
#define NOTE_A11 554
#define NOTE_AS11 587
```
Step 4 – your turn!

Using what you’ve learned from Parts 1 and 2, make a new sketch based off of the code in Part 3 that plays the melody in response to a button press.

- Create a new sketch
- Start by pasting in the code from Part 3
- Save as shortMelodyButton
- Make a new tab called Pitches.h
  o Paste in the code from Part 3’s Pitches.h
- Now modify the code in shortMelodyButton to follow Part 2 for button presses
Step 5

The following code simplifies playing a song with common notes and plays the first part of twinkle twinkle little star.

```cpp
/*
Melody
Plays a melody on SquareWear
* use buzzer on digital pin 9
Written by Shani Manning, edited by Audrey St. John
From example code in the public domain:
http://arduino.cc/en/Tutorial/Tone
*/

// the pin we will use for output
// specific to SquareWear
#define BUZZER_PIN 9

// notes for the song. A space represents a rest
char song[] = "ccggaag ffeeddc ";

// the number of notes in the song
int songLength = 15;

// beats per note
int beats[] = { 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 4};

// speed of song
int tempo = 300;

// convenience notes means we don't need Pitches.h
// names of the notes
char names[] = { 'c', 'd', 'e', 'f', 'g', 'a', 'b', 'C' };

// corresponding tones for C4-
// B4, C5 (looked up values from Pitches.h)
int tones[] = {262, 295, 330, 349, 392, 440, 494, 523};

// number of notes
int numberConvenienceNotes = 8;

// play the song using the variables: song, songLength, beats
void playSong()
{
    // for each note in the melody:
    for (int noteIndex = 0; noteIndex < songLength; noteIndex++)
    {
        // if it's a space
        if (song[noteIndex] == ' ')
        {
            // rest
            delay( beats[noteIndex] * tempo/5);
        }
        else
        {
            // play the note at that index for the specified time
            playNote(song[noteIndex], beats[noteIndex] * tempo);
        }
        // pause between notes
        delay(tempo);
    }
}

// play the tone corresponding to the note name
void playNote(char noteName, int duration)
{
    // loop through the names
    for (int i = 0; i < numberConvenienceNotes; i++)
    {
        // if we found the right name
        if (names[i] == noteName)
        {
            // play the tone at the same index
            tone( BUZZER_PIN, tones[i], duration);
            // don't bother looking through the rest of the names
            break;
        }
    }
}

/**
* The code in this special method is executed
* once when the microcontroller is turned on
* (or the program is uploaded).
**/
void setup()
{
    // play the song at the beginning
    playSong();
}

/**
* The code in this special method is constantly executed
* after setup has occurred
**/
void loop()
{
    // it will be annoying to constantly hear the melody,
    // so let's do nothing!
}
```
Modify the code to complete the song with the following notes.

<table>
<thead>
<tr>
<th>Treble Clef</th>
<th>Bass Clef</th>
</tr>
</thead>
<tbody>
<tr>
<td>C C G G A A G</td>
<td>5 5 1 1 1 1 1</td>
</tr>
<tr>
<td>Twin-kle twin-kle lit-tle star</td>
<td></td>
</tr>
<tr>
<td>4 4 3 3 2 2 1</td>
<td></td>
</tr>
<tr>
<td>F F E E D D C</td>
<td></td>
</tr>
<tr>
<td>2 2 3 3 4 4 5</td>
<td></td>
</tr>
<tr>
<td>how I won-der what you are</td>
<td></td>
</tr>
<tr>
<td>5 5 4 4 3 3 2</td>
<td></td>
</tr>
<tr>
<td>G G F F E E D</td>
<td></td>
</tr>
<tr>
<td>1 1 2 2 3 3 4</td>
<td></td>
</tr>
<tr>
<td>up a-beve the world so high</td>
<td></td>
</tr>
<tr>
<td>5 5 4 4 3 3 2</td>
<td></td>
</tr>
<tr>
<td>G G F F E E D</td>
<td></td>
</tr>
<tr>
<td>1 1 2 2 3 3 4</td>
<td></td>
</tr>
<tr>
<td>like a dia-mond in the sky</td>
<td></td>
</tr>
<tr>
<td>1 1 5 5 5 5 5</td>
<td></td>
</tr>
<tr>
<td>C C G G A A G</td>
<td></td>
</tr>
<tr>
<td>5 5 1 1 1 1 1</td>
<td></td>
</tr>
<tr>
<td>Twin-kle twin-kle lit-tle star</td>
<td></td>
</tr>
<tr>
<td>4 4 3 3 2 2 1</td>
<td></td>
</tr>
<tr>
<td>F F E E D D C</td>
<td></td>
</tr>
<tr>
<td>2 2 3 3 4 4 5</td>
<td></td>
</tr>
<tr>
<td>how I won-der what you are</td>
<td></td>
</tr>
</tbody>
</table>
**Project – due Wednesday, October 2 at 1:15pm**

Design and create your own soft music box, such as a bag that plays a song when you open it.

*Additional ideas (not required):* add LEDs, change the song according to the temperature, randomly choose a song...

- Using what you've learned during lab, create an Arduino sketch with your own song. It must have at least 50 notes, but don't overdo it (you may run out of memory!).
- Then investigate the light sensor:
  - Create a new sketch
  - Paste in the following code

```c
// ******************************
// Bright light
// Audrey St. John
// If there is enough light, red led is on;
// otherwise, red led is off
// Internal connections:
// - red pin: 8
// - light_sensor: A0
// ******************************

#define RED 8
#define LIGHT_SENSOR_PIN A0
#define minLightValue 400

void setup()
{
  pinMode(RED, OUTPUT);
  digitalWrite(RED, LOW);
}

boolean brightEnough()
{
  int light_value = analogRead(LIGHT_SENSOR_PIN);
  if (light_value >= minLightValue)
    return true;
  else
    return false;
}

void loop()
{
  if (brightEnough())
    digitalWrite(RED, HIGH);
  else
    digitalWrite(RED, LOW);
}
```

- Save it as `brightLight`
- Now upload to your SquareWear – the red light should go on and off depending on if there is enough light at the photoresistor
- Modify your song Arduino sketch to play the song only when there is enough light
- Upload your program onto your SquareWear and incorporate into your design!
- Submit your final code via Moodle's Lab 4, either by uploading the Arduino file itself (with extension .ino) or by pasting into a Word document.