1. Year and Semester : 2013 SPRING

2. Course Number : CS-492

3. Course Title : Kinect Programming

4. Work Number : LA-03

5. Work Name : Arduino Laboratory Report

6. Work Version : Version 1

7. Long Date : Sunday, 3, February, 2013

8. Author(s) Name(s) : Jake Waffle

Circuit #4

This circuit essentially powers a mini-servo and allows the uploaded code to rotate it. It requires the use of a class from a built-in Servo.h file. And that class defines a way to connect the servo to a pin# and rotate the servo to a position between 0 and 180.

When modifying the code, I tried out the example code in the “Making It Better” portion of our arduino booklet. And it basically showed how to rotate the servo without using the Servo class from the Servo.h file. The pin the servo is connected to is setup using the pinMode() function. Then when the pin outputs HIGH with a digitalWrite(), the servo rotates until the pin is set to LOW again (that’s what I thought it was doing from looking at the code anyway.) And of course there is a delay() function after all of that is done. But when I uploaded the code, the servo would just rotate until it hit a certain degree (like the original code did in a sense) and then it would stop.

This is the code from the “Making It Better” section:

```
int servoPin = 9;
void setup()
{
  pinMode(servoPin, OUTPUT);
}
void loop()
{
  int pulseTime = 1500;
  digitalWrite(servoPin, HIGH);
  delayMicroseconds(pulseTime);
  digitalWrite(servoPin, LOW);
  delay(25);
}
```

Circuit #5

This circuit makes use of a shift register chip to signal eight LEDs to turn on. And the idea behind the chip is that we want to pass a decimal number to the data pin (I think the shiftOut() function we use converts the number to binary and passes the bits individually.) Then the chip will send a HIGH/LOW signal to the pins that correspond with the 1's/0's in the binary number we passed to the data pin. Aside for the chip part, we've dealt with the LEDs before in a couple of circuits already. But the code cycles through 0-256 and sends those numbers to the data pin. And that essentially loops through all combinations of LED on/off status' for the eight LEDs.

For modifications, I completed the code in the "Making It Better" section of our booklet. This code essentially will individually turn on each LED. The parts that I added were just a global integer variable to hold the current number and a function that would add/subtract to/from the global integer depending on the LED that we want to turn on/off (the function also called updateLEDs() using the global integer as a parameter too.) I also had to write a function to compute the power.

Here's the code that I added:

```
int byteNum = 0;
int pow(int value, int exponent)
{
  int total = value;
```

```
  for(int i = 0; i < exponent-1; i++)
  {
    total *= value;
  }
  return total;
}

void changeLED(int value, boolean on)
{
  int byteNum = 0;
  if (on)
  {
    byteNum += pow(2, value);
  }
  else
  {
    byteNum -= pow(2, value);
  }
  updateLEDs(byteNum);
}
```

Circuit #6

This circuit is really simple, it just consists of a piezo element that is connected to a ground and a pin# on the Arduino. And with this piezo element we can output music. But in reality the piezo element is just making a series of clicking noises at a specified frequency in order to imitate a musical note. Looking at the code supplied by the book, the different tones are denoted by the delay time between the pin's switch from LOW to HIGH and HIGH to LOW. So the delay directly affects the square wave's frequency given off by the piezo element (I think anyways, the frequency is denoted by the oscillations per unit of time.)

An issue I ran into after running the code that our booklet supplies had to do with the fact that there are two 'c' tones defined in the notes. So when the piezo played a 'c' tone, it would play both of the 'c' tones instead of just one of them (because the code loops through all of the possible notes to find any matching the note that is supposed to be played.) That really stumped me for a bit, but I realized that the piezo was incorrect only when it played the 'c' tones.

Then when going to modify the code, I chose to just program in a different song to play (there doesn't seem like much else that can be done.) And to do that, I went to my piano to figure out a simple song that I know that also doesn't use any sharps or flats. The one I came up with was Epona's song from “The Legend of Zelda Ocarina of Time” (which ended up requiring an E# and a D from the next octave up.)

These are the global variables I adjusted to change the song:

```
int length = 32;

char notes[] = "cagcagcagageeEeabccdc";

int beats[] = {1,1,2,1,1,2,1,1,2,2,3,1,1,1,1,1,1,1,2,2,2,2};
```

And I also had to alter the variables within the playNote() function (this is where I added in the E# and the high octave D, which I just guessed numbers for):

```
char names[] = {'E', 'e', 'f', 'g', 'a', 'b', 'c', 'd'};

int tones[] = {1600, 1519, 1432, 1275, 1136, 1014, 956, 850};
```

Circuit #7

This circuit is the first one that makes use of a pin for input purposes. And even though there are two buttons in the circuit to begin with, only one of them is being utilized (it triggers the LED to be turned on when pressed and turned off when not pressed.) I've been focusing on how the pull up resistor, ground and button all go together to give input to the input pin mostly. The resistor transfers a HIGH signal to the input pin when the button isn't pressed (the schematics show a line through the button that isn't aligned when unpressed, so the ground isn't connected to the pin at that point.) But when the button is pressed, the ground overpowers the HIGH signal and transfers a LOW signal to the input pin (that's confusing how when the positive and ground are connected, only the ground matters.) I think this happens, because the ground drains the connected bar of its voltage (when people touch electricity and are grounded, the electricity goes through them to the ground.)

The modification to this first just made it so that the one button would turn the led on and the other would turn it off (the logic in the code just changed a little bit, not too drastic though.) Then when setting the circuit up to slowly fade the led on and off I noticed something that really didn't occur to me beforehand (I didn't switch the led's pin from 13 to 9 and that didn't let the fading work at all.) That realization was that there are two different kinds of pins. Both kinds of pins allow digital/analog writes, but only the pin numbers with a tilde prefix are allowed to send specific inputs ranging from 0-255 (rather than just the normal HIGH/LOW.) And the code for the fading pretty much just changed the buttons purpose. So that they'll alter a

variable instead of directly affecting the led's pin. From there the variable gets constrained to the range of 0 to 255 and an analog write sends that value to the led.