1. Year and Semester : 2013 SPRING

2. Course Number : CS-492

3. Course Title : Kinect Programming

4. Work Number : LA-04

5. Work Name : Arduino Laboratory Report

6. Work Version : Version 1

7. Long Date : Sunday, 25, March, 2013

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

Circuit #8

Overview:

This circuit taught me how to read analog inputs in from the analog-in pins mostly. It essentially uses the analog input from a potentiometer (a twisty knob) in order to dynamically set the delay in-between the LED's switch to on/off.

Modifications:

For modifications, I first thought that it was weird that we were told to use a different set of pins for the analog input (when before we could use the digital pins for analog output.) So I tried to make the potentiometer work while it was connected to a digital input (and I thought pin # with a tilde prefix may allow analog input like it does with analog output.) The booklet says that the analog-in inputs take in a 0-5 voltage and convert that to a number between 0 and 1024 (while the digital pins convert the 0-5 voltage into either LOW or HIGH.) The digital pins allow for digital/analog outputs, but they only allow for digital (HIGH/LOW) inputs (because the outputs are simulated as analog.)

Here’s the code that I added when I wanted to try the digital pins for analog input:

```
int sensorPin = 3;
void setup()
{
  pinMode(sensorPin, INPUT);
}


void loop()
{
  sensorValue = digitalRead(sensorPin);
}
```

Then I moved on to the “Making It Better” section and it suggested hooking a servo up to the potentiometer. So I instead tried hooking a motor up to the Arduino so that we can control the speed of the motor with the potentiometer (sadly I was unable to get it working.) The circuit was hooked up so that the potentiometer and motor are all wired independently. The code is where the potentiometer and motor were supposed to be connected (read from the potentiometer and write to the motor.) And I did eventually find out what the problem was (I used the temperature sensor instead of the transistor.) I was really excited to see that I got that working finally (it was after I moved on to circuit 9 that I realized I could have used the wrong part.)

Here’s the code that I thought would work for it though:

```
int sensorPin = 0;

int motorPin = 9;

void setup()

{

  pinMode(motorPin, OUTPUT);

}

void loop()

{

  //The .../4 -1 converts 1-1024 to 0-255 for the motor.

  motorOnThenOffWithSpeed((analogRead(sensorPin)/4) -1);

}

void motorOnThenOffWithSpeed(int onSpeed)

{

  analogWrite(motorPin, onSpeed);

  delay(1000);

  analogWrite(motorPin, 50);

  delay(500);

}
```

Circuit 9

Overview:

This circuit instead of taking in user-input, takes in input from the environment and adjusts the intensity of a led respectively. I thought it was pretty odd that the led gets dimmer as more light is detected by the photo resistor though (later I realized that this is probably for night light-type things that need to run when no light is present.) I would have thought that the photo resistor outputs its highest value when it is exposed to a lot of light (but it's the other way around.) But essentially the photo resistor outputs a value between 0 and 900 to an analog pin depending on the intensity of the light it detects (0 for bright light, 900 for no light.) Then the led takes in a value between 0 and 255 from a simulated analog output pin. And all that makes it able to alter the intensity of the led depending on the lighting around it.

Modifications:

When moving onto the "Making It Better" section, I first tried out the minor code alteration that allows the circuit to work like a night light (I just added a threshold condition and digital writes instead of analog writes.)

Here’s the code alteration for the night light:

```
void loop()

{

 int threshold = 300;

 if (analogRead(lightPin) > threshold)

 {

   digitalWrite(ledPin, HIGH);

 }

 else

 {

   digitalWrite(ledPin, LOW);

 }

}
```

Then I also did the light controlled servo thingy suggested in the “Making It Better” section. The booklet said to use some code from a knob example that uses a servo and alter it so that it reads from the photo resistor’s pin instead of whatever it read from. But it also said that the value read from the photo resistor would need mapped to a different range in order for the servo to use its full range of motion. Naturally I figured since the beginning bit of code mapped the input from 0-900 to something else, this bit of code also needed to do that. Sadly I was mistaken and now realize that the constrain() function was used for a reason in the beginning code. I tried several combinations of values to map from with no success. I tried printing the value of outputted by the photo resistor, but while my code ran, it did not print to the console (later I realized that I had to open the Serial Monitor to see the printed information.)

Here's the code I used for the photo resistor and servo`:

```
#include <Servo.h>
Servo myservo;  // create servo object to control a servo
int lightPin = 0;
int val;
void setup()
{
  Serial.begin(9600);
  myservo.attach(11);
}
void loop()
{
  val = analogRead(lightPin);            // reads the value of the potentiometer (value between 0 and
1023)
  Serial.println(val);
  val = map(val, 0, 900, 0, 180);     // scale it to use it with the servo (value between 0 and 180)
  myservo.write(val);                  // sets the servo position according to the scaled value
  delay(15);                           // waits for the servo to get there
}
```

Circuit 10

Overview:

This circuit was really quick to do, but it finally taught me how to use Serial in order to debug a program with print statements (which was actually quite simple.) This circuit essentially used a temperature sensor to take in the temperature from its surrounding environment and output a value between 0 and 1023 into an analog input pin. Then the code read this value and converted it into a floating point value between 0 and 5 (which represented the voltage.) Then I believe the voltage was converted into the temperature in Celsius units (the booklet's explanation comments confused me, but I analyzed the values the program printed.) And the Serial thingy actually outputs to the Serial Monitor instead of the console (which was what I did wrong when trying to print in the last circuit.)

Modifications:

The “Making It Better” section of our booklet gave suggestions for simple modifications. One modification involves just converting the temperature's units from Celsius to Fahrenheit (which just used a simple mathematical conversion: F = C*1.8 + 32.)

It was also suggested that using a more detailed route to printing the results of the temperature sensor is better for debugging purposes (label the outputs.) And from that, I learned that the print() and println() functions differ in that print() does not add a newline character to the end of the message, whilst println() does.

There was also a note in the “Making It Better” section that if a lot of data needs outputted to the Serial Monitor, it is possible to speed up the outputting process to 12 times its original speed. But I didn't find this necessary for what we were doing.

Circuit #11

Overview:

This circuit is similar to circuit #3 in that it uses a transistor in order to control something, except this circuit uses the transistor to turn on/off a relay. And our relay specifically switches which pins get a positive charge and which get a negative charge. There are two pins that are positively charged and another two that are negatively charged at all times. The transistor just makes the relay oscillate the positive and negative charge between those pins.

Modifications:

I pretty much just hooked up our motor so that it is connected to the relay and the ground (with a diode bridging between the relay and ground.) Then the motor (with the previous code) would just turn on and run for a couple seconds, then turn off for a couple seconds.

In the “Making it Better” section of our booklet, there is a circuit schematic that when built will oscillate the direction of the motor’s spinning. It looks like the motor is connected to both com pins (instead of the positive connections.) There are also negative connections to NC and FNO. And there are positive connections to FNC and NO. So while FNC and NC are connected to the coms, the motor will be connected to positive/negative (positive to the motor’s negative wire and negative to the motor’s positive wire.) And while FNO and NO are connected to the coms, the motor will be connected to negative/positive (negative to the motor’s negative wire and positive to the motor’s positive wire.) That was indeed satisfying to see a circuit that I altered from a schematic work.

Circuit #12

Overview:

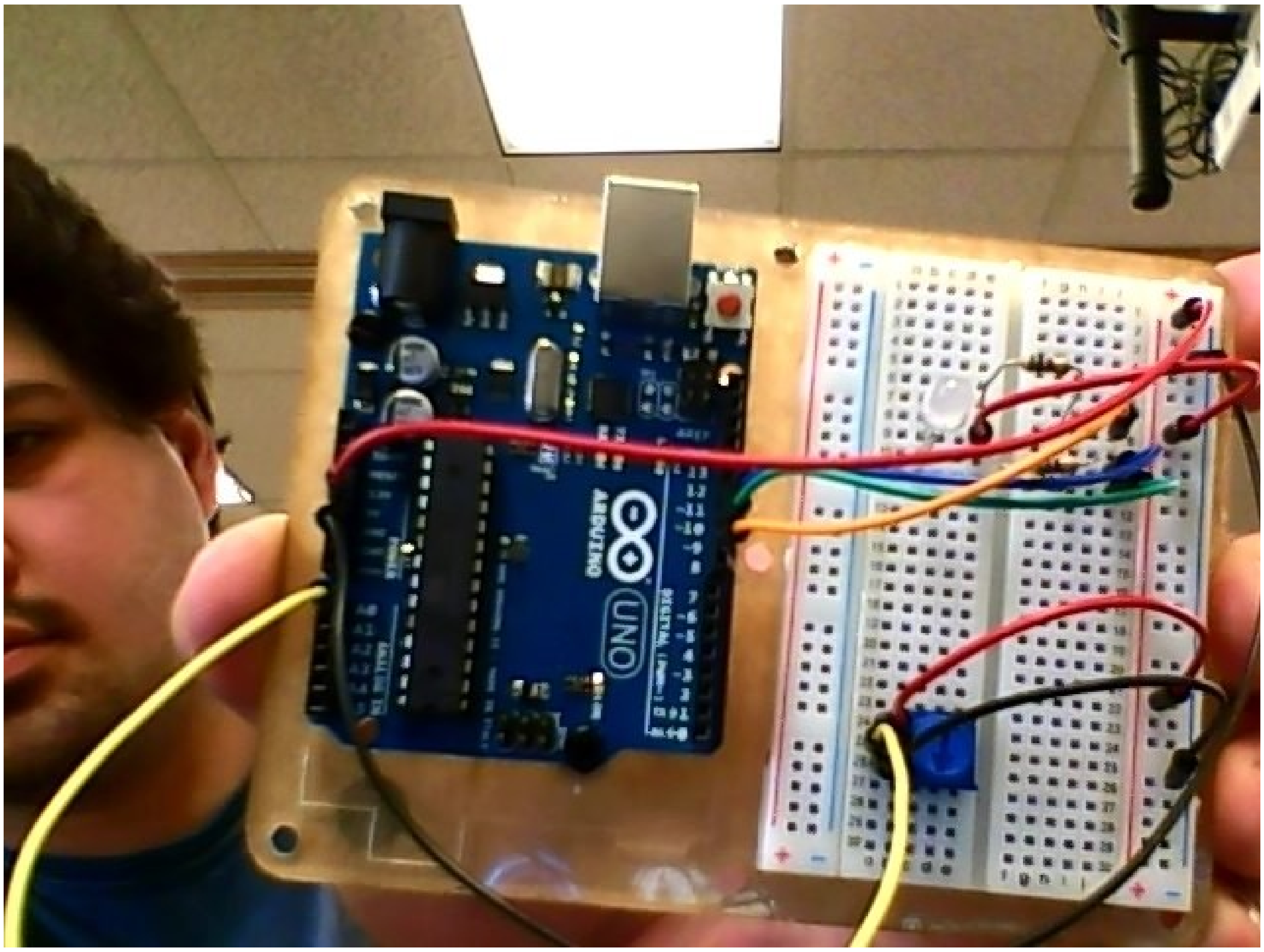
This circuit works a lot like the red/green LEDs (because of the RGB LED,) but instead there are four leads for the RGB LED (three leads go to pins that represent the red/green/blue and the other lead needs a positive charge.) With this RGB LED, we are able to alter the color that is emitted. A HIGH signal going to one of the RGB leads makes red/green/blue apart of the color that is emitted (and a LOW signal makes that color not shown.) So to make more colors than just red, green or blue, we can just show a combination of those colors (which mixes them together to create a different color.) For example, cyan can be emitted by sending the green and blue leads a HIGH signal, while sending a LOW signal to the red lead (green and blue mixed together creates cyan.)

Modification:

At first, the code would just strictly emit cyan from the RGB LED. But in the code, there is a randomColor() function that picks and displays a random color (it was commented out in the loop() function.) So I changed the loop() function so that it would call randomColor() and then delay for a second.

Then I downloaded some code that controls the RGB LED using analog instead of digital writes (ardx.org/RGBANA.) The code at first only sets the RGB LED to magenta, but that's because the other examples are commented out. There is an example that can set the color to any rgb-value. There is also an example that uses a function that fades between two colors. And with this code, I altered it so that it uses a potentiometer to cycle through some of the pretty analog colors (Figure 1 is a picture of the altered circuit.)

Figure 1 – The potentiometer was setup independently to that of the RGB LED, they just are connected within the code.



The following code is what I added into the code I downloaded from ardx.org/RGBANA:

```
//This is an added global variable.
const byte* COLORS[] = {ORANGE, INDIGO, VIOLET, YELLOW, CYAN, MAGENTA,
WHITE, PINK};

//This is the whole loop() function.
void loop()
{
  int colorIndx = analogRead(0);        //The potentiometer is connected to pin #0.
  colorIndx = map(colorIndx, 0, 1015, 0, 7);  //There are 8 colors total.
  setColor(ledAnalogOne, COLORS[colorIndx]);
}
```

If you'll look at the above code, the analogRead() reads the value that the potentiometer is outputting. Then that number (which is between 0 and 1023) is converted so that the number is between 0 and 7 (because there are 8 elements in the global array I added.) And finally that number (that's between 0 and 7) is used to select the color (from the global array) that the RGB LED is to be set to.