**ELEC 291 Section 20C**

Lab 1

L2C

Team 2A

*Student name Student number Contribution percentage*

Andy Ruan 36863141 33.3%

Kevin Wong 32105132 33.3%

Clarence Su 36387132 33.3%

Contribution Summary:

Andy Ruan wrote and compiled most of the Arduino code along with setting up the communication among teammates in order to write this and future lab reports.

Kevin Wong worked on the building the circuits and writing/organisation of the majority of this lab report.

Clarence Su worked on the building the circuits and the fritzing model used in this lab report.

Note that every member contributed to the discussion of every part of the lab, even the ones that they did not particularly focus on.

**Introduction and Motivations**

This report outlines the process in which Team 2A used to complete lab 1 of ELEC 291-20C. It is organized in the following manner: First, it will provide an introduction and motivations for this lab. The second part of this report will describe the lab and the sub parts of it. The third part will be the conclusions from the lab. The fourth part will provide references. Appendix I will include an image of the fritzing breadboard schematic. Appendix II will discuss the Arduino code used in the lab. Lastly, Appendix III will discuss the answers to the three application notes.

As this lab was used as an introduction to the course, our group was not concerned with the most efficient method of distributing work but rather that all of our fundamentals were at an appropriate level. Therefore, we were motivated to have each member understand each part of this lab fully before proceeding. This included the code, the hardware (Arduino, DIP switch, LEDs, resistors, analog sensors, etc.), and the fritzing schematics. Allowing for each member of the team to learn all the parts of the lab is critical for the success of future labs and projects because they build upon the skills learned in this lab.

**Lab Description**

Lab 1 was divided into five parts:

1. Getting Started with Arduino and Arduino IDE
2. Interfacing Arduino with LED
3. Controlling LEDs using switches
4. Using photocell and PWM
5. Using serial monitor

These five parts together resulted in the team building a complete circuit that uses a switch to control a RGB LED. Additionally, the circuit uses a photocell to sense the amount of light in the environment in order to control if a single-colour LED would light up in sufficient darkness.

In part one, the team learned the basics of Arduino syntax and basics of the Arduino IDE, an environment for programming to the Arduino. In part two, the team learned to create a basic circuit that would trigger a RGB LED to flash red, green, blue in different patterns. In part three, the team added a switch that would cause the LED to flash in a certain pattern depending on which switches were on or off. In part four, the team added a photocell to a free analog pin with PWM to determine the intensity of light in the surroundings in order to trigger a single LED to flash. Lastly, the team used serial output to display some parameters to the serial monitor.

Following the lab procedure, our team was able to create a working circuit that completed all parts of the lab. The team read the basics to breadboarding to connect different parts of the circuit together. In the step for RGB LEDs the team decided to use three resistors for each RGB cathode instead of one common resistor. The main reason is because having one resistor for each LED, may lead to different voltages across each LED hence the brightness might be lower for one LED and higher for another. It is best to use three resistors in this case ensuring that all LEDs have the same voltage [1].

For testing, the team decided to manually change the inputs to the circuit and determine if the output was the anticipated one. For the RGB LED part of the lab, the team tested the switch by turning on different switches and determining if it matched the color pattern from the Arduino code. For the photocell part of the lab, the team tested the photocell by blocking as much light as possible to see if the single LED would light up. Lastly, for the serial monitor part, we began by experimenting with what simple Serial.println() statements would print out on the monitor for different parameters. After, we organized the printed data into a more human readable format, similar to the example in the lab handout.

One problem the team had while working on the lab was that the RGB LED was not working as expected and the green light displayed when we tapped on one of the wires of the circuit. After debugging for a few minutes, the problem was fixed. It happened to be human error when wiring the circuit because a wire was connected to the wrong digital pin. A good lesson from this error is to make a sketch on Fritzing before actually wiring the actual circuit so that if there are any errors in the circuit, comparing to the Fritzing diagram will quickly determine the problem.

Another problem that the team had while working on the lab was that the photocell seemed to not detect any light. After examining the output of the photocell on the serial monitor, it was determined that the threshold to turn the LED on had been set too low for the environment we were in (sunlight through the window). The team learned that reading the documentation thoroughly for a component is critical before starting to work on it otherwise it would take some time debugging and figuring out if the component is not working our overall circuit is not working.

The last problem was rewiring the circuit multiple times. Due to the small breadboard space and the lack of planning ahead when wiring the circuit, our team rewired the circuit several times in order to fit new components such as single LED and photocell. As with the first problem, having a fritzing breadboard schematic before wiring would save the team time from rewiring.

**Conclusions**

This lab was a great stepping stone for students who did not have experience working with Arduinos and basic circuits. From this lab, students learned how to build a simple circuit with an Arduino to make LEDs blink and could transfer those skills to make a more complex circuit in subsequent labs. Additionally, students learned the basics of Arduino code and resources to find examples of code for more complex projects. Moreover students learned to communicate technical details of labs to other people using a written report and Fritzing software.

Our team learned an important lesson from this lab. Students should build a circuit on the Fritzing software before wiring the circuit on a breadboard. This is important because there are often human errors that occur in wiring circuits. With Fritzing, students are visually clear on which part of the breadboard they need to wire in order to make a working circuit. Prototyping the circuit on Fritzing allows for efficient use of time because it reduces the amount of time spent debugging when the circuit does not work.

**References**

[1] http://forum.arduino.cc/index.php?topic=65746.0

[2] http://www.explainthatstuff.com/amplifiers.html

**Appendix I**

Below is an image fritzing breadboard schematic for Lab 1.

12557893_914001305373977_327251185_o.jpg

Figure 1. Arduino and breadboard schematic diagram

**Appendix II**

Below is the code written for Lab 1.

/\*

\* Code used in lab 1 of ELEC 291-20C

\*/

// CONSTANTS

const int ONBOARDLEDPIN = 13;

const int REDLEDPIN = 10;

const int GREENLEDPIN = 11;

const int BLUELEDPIN = 12;

const int SWITCH8 = 9;

const int SWITCH7 = 8;

const int PHOTOPIN = 0;

const int PHOTOLEDPIN = 6;

const int PHOTOTHRESHOLD = 800;

const long INTERVAL = 2000;

// VARIABLES

unsigned long prevMilli = 0;

int onBoardLEDState = LOW;

int sw8Status, sw7Status, swLEDState, photoCellStatus, photoCellState; // used for Serial.print()

void setup() {

Serial.begin(9600); // begin serial comms

pinMode(ONBOARDLEDPIN, OUTPUT);

pinMode(REDLEDPIN, OUTPUT);

pinMode(GREENLEDPIN, OUTPUT);

pinMode(BLUELEDPIN, OUTPUT);

pinMode(PHOTOLEDPIN, OUTPUT);

// other pins implicitly treated as INPUT

}

void loop() {

// read the two switches used on the DIP switch and the value of the photocell

sw8Status = digitalRead(SWITCH8);

sw7Status = digitalRead(SWITCH7);

photoCellStatus = analogRead(PHOTOPIN);

// Both switches OFF => RGB OFF

if ( sw8Status == LOW && sw7Status == LOW ) {

digitalWrite(REDLEDPIN, HIGH);

digitalWrite(GREENLEDPIN, HIGH);

digitalWrite(BLUELEDPIN, HIGH);

swLEDState = LOW;

}

// sw8 ON, sw7 OFF => Red->Green->Blue pattern

else if ( sw8Status == HIGH && sw7Status == LOW ) {

digitalWrite(REDLEDPIN, LOW);

delay(1000);

digitalWrite(REDLEDPIN, HIGH);

digitalWrite(GREENLEDPIN, LOW);

delay(1000);

digitalWrite(GREENLEDPIN, HIGH);

digitalWrite(BLUELEDPIN, LOW);

delay(1000);

digitalWrite(BLUELEDPIN, HIGH);

swLEDState = HIGH;

}

// sw8 OFF, sw7 ON => Green

else if ( sw8Status == LOW && sw7Status == HIGH ) {

digitalWrite(REDLEDPIN, HIGH);

digitalWrite(GREENLEDPIN, LOW);

digitalWrite(BLUELEDPIN, HIGH);

swLEDState = HIGH;

}

// both ON => Blue

else {

digitalWrite(REDLEDPIN, HIGH);

digitalWrite(GREENLEDPIN, HIGH);

digitalWrite(BLUELEDPIN, LOW);

swLEDState = HIGH;

}

// single-colour LED turns on at low light

if ( photoCellStatus >= PHOTOTHRESHOLD ) {

digitalWrite(PHOTOLEDPIN, HIGH);

photoCellState = LOW;

}

else {

digitalWrite(PHOTOLEDPIN, LOW);

photoCellState = HIGH;

}

// blinks the LED on the Arduino every 2s without using delay();

// checks every loop() if INTERVAL ms has passed

unsigned long currMilli = millis();

if ( currMilli - prevMilli >= INTERVAL ) {

prevMilli = currMilli;

if ( onBoardLEDState == LOW ) {

onBoardLEDState = HIGH;

}

else {

onBoardLEDState = LOW;

}

digitalWrite(ONBOARDLEDPIN, onBoardLEDState);

}

// prints the status of multiple elements on a line

// Switches: sw8 sw7

Serial.print("Switches: ");

Serial.print(sw8Status);

Serial.print(" ");

Serial.print(sw7Status);

// LEDs: Arduino RGB Light-sens

Serial.print("\tLEDs: ");

Serial.print(onBoardLEDState);

Serial.print(" ");

Serial.print(swLEDState);

Serial.print(" ");

Serial.print(photoCellState);

// value read from the photocell

Serial.print("\tPhotocell value: ");

Serial.print(photoCellStatus);

Serial.print("\n");

delay(200); // 200 ms delay between loops as requested

}

**Appendix III**

**1. Capacitor use:**

When a capacitor is added between a circuit’s power and ground, it can bypass high frequency signals such as noise that we do not want. This bypassing effectively stabilizes the voltage being delivered from the power supply to the terminals of an electronic component. An example of a bypass capacitor being used on the Arduino Uno is shown in the next figure.



Figure 2. A bypass capacitor(circled in black) used on the Arduino Uno’s ATMEGA8 chip

**2. Transistor/op-amp use:**

Amplification is needed when a signal from the output pin needs to be stronger. This is extremely useful when dealing with audio devices because it can amplify the original sounds where people can hear. An example would be hearing aids [2].

**3. Datasheet (TL072-CN):**

1)The maximum range of the power supply voltage: [-18, 18], with respect to the zero reference level (ground).

2) The range of the allowed input voltage: [-Vcc, Vcc] or [-15, 15] depending on the supply voltage. Input voltage must never exceed the magnitude of the supply voltage or 15 volts, whichever is less.

3) IC pins are the VCC+ and VCC-. Pin4 is VCC- and Pin8 is VCC+ (refer to Figure 3).

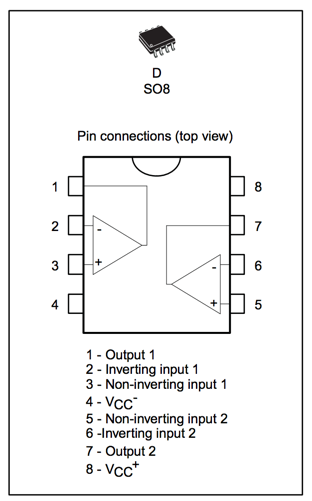


Figure 3: Pin connections of the TL072-CN