Lab 3 – Introduction to Arduino

Introduction to Arduino:

The Arduino (shown in Figure 1) is a relatively inexpensive, yet versatile open-source microcontroller. It is designed to facilitate interaction with the physical world via sensors while being able to perform calculations and various functions. The Arduino can be connected to a computer via a USB cable and programmed using a simplified version of the C programming language, and it has both analog and digital pins from or to which it can read or write values. The maximum voltage that it is able to supply is 5V; thus, a “HIGH” digital pin corresponds to 5V, while a “LOW” digital pin corresponds to 0V. There are many “shields” and sensors that are designed for interaction with the Arduino, or microcontrollers in general. The Arduino can read values from sensors or other inputs, and it can also write values to other components based on computations given by the program.

Arduino is fast becoming one of the most popular microcontrollers on the market. Its ease of use, extensive software library and most importantly, its low cost (~$30 for a basic set compared to $150 for other microcontrollers) have come to make it as popular as it is today. Many projects using the Arduino can be found on www.hackaday.com.

In order to start having fun with the Arduino, free software can be found at: http://arduino.cc/en/Main/Software for Macs, Windows and Linux operating systems. This website also provides tons of easy tutorials for you to start. Tutorials can be found at: http://arduino.cc/en/Tutorial/HomePage

In this lab, you will become familiar with the Arduino and some of its applications. You will learn how to use the Arduino to interface between hardware (your circuits) and software (the code). You will also learn how to use a voltage divider to convert a change in resistance (from a sensor) to a change in voltage which the microcontroller can read.

Goals:

- Learn how to program the Arduino and use basic functions
  - delay, pinMode, digitalWrite, digitalRead, analogRead, etc.
- Learn how to use a voltage divider
- Control digital outputs based on analog/digital input readings
Procedure:

1. Becoming familiar with the Arduino and the Integrated Development Environment (IDE)

In this section, you’ll learn how to use the Arduino to flash an LED connected to digital pin 13.

i. Go to Start → Programs → ESE Lab Software and open “Arduino 0022” (or Arduino 1.0 or Arduino 1.0.1) to launch the Arduino IDE.

ii. Copy/Write the code in the IDE window. For this tutorial, go to File → Examples → Basics → Blink, or copy the following code into the IDE window.

```cpp
/*/ 
Blink 
Turns on an LED on for one second, then off for one second, repeatedly. 
This example code is in the public domain. */

void setup() { 
  // initialize the digital pin as an output. 
  // Pin 13 has an LED connected on most Arduino boards: 
  pinMode(13, OUTPUT);
}

void loop() { 
  digitalWrite(13, HIGH); // set the LED on 
  delay(1000); // wait for a second 
  digitalWrite(13, LOW); // set the LED off 
  delay(1000); // wait for a second 
}
```

Notice that text following two slashes // or between /* */ is grayed-out and is not read by the Arduino; these are known as comments, and they serve to help the reader understand the code.

iii. Go to the Tools menu of the Arduino IDE (Figure 2), and select the Serial Port (COM Port) to which the Arduino Board is connected to. This should be the last one (not COM1 or COM3). Make sure that the Arduino is plugged into one of the USB ports; otherwise you will only see COM1 and COM3.

Figure 2: Select the Serial port using Arduino IDE
iv. Go to Tools → Board. If your board says “Arduino UNO,” make sure the first option (Arduino Uno) is selected. If your board says “Arduino Duemilanove,” select the second option (Arduino Duemilanove or Nano w/ ATmega328).

v. Use the Upload icon of the Arduino IDE to compile and download the program to the Arduino Board. Figure 3 shows the Upload icon in the Arduino 0022 IDE. In the newer versions (1.0 and 1.0.1), the Upload icon is the second button from the left, next to the Compile button.

vi. If your upload was successful (Figure 4) you will get the “Done Uploading” message.

vii. If you get any other error message, contact your TA for help!

viii. Look at your Arduino board. You should see a yellow LED (labeled ‘L’) flashing on and off at a frequency of 0.5 Hz. This LED is wired up directly to digital pin 13. In the setup function, the pinMode function is used to specify that pin 13 is being used as an output. Within the loop function of this code, pin 13 is written HIGH (pin 13 becomes a 5-V...
source) using the `digitalWrite` function. Next, the program waits for one second using the `delay()` function, which takes an argument in milliseconds. Then, pin 13 is written LOW (0V). The program again waits (does nothing) for a second. The sequence of events within `loop` will be repeated forever as long as the Arduino is supplied with power, while the events in `setup` occur only once when the program starts running.

ix. Now modify the code so that the flashing rate is 1 Hz (1 cycle per second).

2. Interfacing the breadboard shield with the Arduino

One of the great things about the Arduino is that there are specialized boards, or shields, that fit right on top of the Arduino which allow you to easily interface additional hardware with the microcontroller. One such shield that you will use a lot is the breadboard shield, which gives you a "workspace" on top of the Arduino to build circuits that you can interface with the digital and analog pins.

i. Plug the breadboard shield on top of the Arduino.

ii. Build the circuits in Figure 5. Digital pins 10 and 12 will be used as a programmable power source; that is, you will program them to output either 5V or 0V.

iii. Modify the code that you used in the previous section so that the two LEDs flash in an alternating pattern, i.e. when one is on, the other is off. Show a TA!

3. Implementing a half-adder

One of the great things about the microcontroller is that you can program it to execute any digital logic function. Recall the half-adder circuit that you designed in Lab 2. The truth table for this circuit is shown in Table 1.

![Diagram of LED circuits controlled by digital pins]

Table 1: Truth table for half-adder

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>S</th>
<th>Cout</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
The half adder has two inputs, A and B, and two outputs, S and Cout. You can easily represent the state of the two outputs with two LEDs (keep the two LED circuits from Part 2) using the `digitalWrite` function. Likewise, the Arduino has a built-in function to read the state of any of the digital I/O pins: `digitalRead(pin)`. If the voltage on the pin specified by the argument is greater than 3V, the function returns HIGH (or 1). If the voltage is lower than 2V, the function returns LOW (or 0).

The hardware setup is slightly less trivial, however. It might seem obvious to use a switch as an input to control the output of voltage. The output should be 5V when the switch is pressed and 0V when the switch is open. The circuit setup shown in Figure 6 might seem like an intuitive way to achieve this.

![Figure 6: Incorrect switching circuit](image)

However, the circuit in Figure 6 is not a valid switching circuit. Assume that if the switch is closed, the digital pin receives 5V. Now consider what would happen when the switch is opened. To what voltage does the digital pin go? This circuit has no way of drawing the pin voltage down to ground when the switch is open. The voltage at the digital pin is said to be floating.

Now consider the circuit represented in Figure 7. What is wrong with this configuration?

![Figure 7: Incorrect switching circuit (getting closer)](image)

When the switch is open, the digital pin is grounded (0V) and therefore will read a low signal. When the switch is closed, however, the 5-V supply voltage is shorted to ground (which is bad). Therefore, we need to add a resistor between the switch and ground to avoid a short circuit between 5V and ground. This resistor also serves to pull the voltage on the digital pin down to ground by providing a path to drain current through. For this reason, it is called a pull-down resistor. A correct switching circuit is shown in Figure 8.
When the switch is pressed, the digital pin sees 5V. Now, when the switch is opened, the voltage difference over the resistor forces a current to flow from the digital pin to ground. Since the digital pin input is not a power source (it is not supplying a constant voltage), the voltage is quickly depleted and the potential at the digital pin becomes 0V. This configuration is called active-high because the output voltage is HIGH when the switch is closed. (In the post-lab assignment, you will design an active-low switch.)

i. Keep two LED circuits from Part 2 connected to pins 10 and 12.

ii. Build two active-high switches (Figure 8, where Vin is 5V and R1 is 1-kΩ) connected to pins 5 and 7 (or any other two digital pins that you’d like).

iii. Copy the following sketch into the Arduino IDE and click “Upload.”

```cpp
// Example sketch for reading digital inputs

int buttonA = 5;  // change value if you used a different digital I/O pin
int buttonB = 7;

void setup(){
  pinMode(buttonA, INPUT);
  pinMode(buttonB, INPUT);
  pinMode(10, OUTPUT);
  pinMode(12, OUTPUT);
}

void loop(){
  int inputA = digitalRead(buttonA); // read voltage on pin, store value in variable
  if(inputA == 1){ // if button is pressed
    digitalWrite(10, HIGH); // turn on LED on pin 10
  }
  else{ // else (if button is not pressed)
    digitalWrite(10, LOW); // turn off LED on pin 10
  }
}

iv. Read through the code to figure out what each line does. Note that this program uses conditional statements (if...else). First, the voltage at digital pin 5 is read, and the
value (either 0 or 1) is stored in a variable, inputA (you can call this almost anything you want). If a 1 was stored in the variable, then 5V is provided at digital pin 10; else, 0V is provided at pin 10.

v. Modify the program to implement a half-adder. To implement the half-adder function given by the truth table in Table 1, you’ll need to use an if...else if...else if...else code structure to control the outputs for each of the four input cases. Go to the following reference articles to read about the if...else if...else control statements and comparison operators:

vi. Verify your program by testing all four input combinations and observing the state of the LEDs. Show a TA!

Aside: Microcontrollers can be very useful for implementing digital logic functions such as the half-adder. The microcontroller performs these functions by executing commands in a sequential fashion. However, if you take ESE 171 (the lab portion of ESE 170 – Digital Logic and Design), you will build many different digital logic systems not with a microcontroller, but with a field-programmable gate array (FPGA). An FPGA is like a programmable breadboard for digital logic. One of the advantages of the FPGA for digital design applications is that signals travel through the logic gates in parallel, whereas each gate must be executed sequentially in a microcontroller. Whereas you tell a microcontroller what to do, you tell an FPGA what to be.

4. Interfacing a light sensor with the Arduino

As you can imagine, most sensors do not just output 0’s and 1’s (these would be very bad sensors). Rather, sensors operate in the analog realm. Recall the photoresistor that you used in Lab 1. The photoresistor is essentially a light sensor. When it is exposed to light, its resistance decreases, and when it is covered up, its resistance increases. If the Arduino runs on a microprocessor that can only understand two different voltage levels, how are we going to be able to interface our light sensor with the Arduino?

Luckily, the Arduino has built-in analog-to-digital converters (ADCs). The ADC maps an analog input voltage between 0 and 5V to a 10-bit binary value (0 to 1023, in decimal form). This conversion allows the microprocessor to interpret a binary representation of the analog signal.

However, there is one other problem. The Arduino can only read voltages, and our light sensor converts light intensity to a level of resistance, not voltage. How can we convert a change in resistance to a change in voltage which the Arduino can read? This is where the voltage divider comes into play! A voltage divider circuit is just two resistors in series connected between a power supply and ground, as shown in Figure 9.
If R1 is connected to the voltage supply and R2 is connected to ground then the voltage at the junction between the two resistors is given by:

\[ V_{\text{out}} = V_{\text{in}} \left( \frac{R_2}{R_1 + R_2} \right) \]

i. Build the circuit on the left in Figure 10, using a photoresistor as R1 and a 1-kΩ resistor for R2. Connect Vout to analog pin 0 using a jumper wire.

ii. Use the DMM to measure the output of the voltage divider. You can connect the black wire to the ground (GND) pin and the red wire to the spot at which you want to probe the voltage (the junction between the resistors).

Now you will use the Arduino’s **analogRead** function to convert the output voltage to a 10-bit value. You will also learn how to use the **Serial** library functions to print text and variable values to the **Serial Monitor**.
iii. Upload the following code to the Arduino.

```cpp
// Example sketch for reading values from ADC and printing to Serial Monitor

int photopin = 0; // variable to store analog pin number

void setup() {
    Serial.begin(9600);   // initialize serial communication at 9600 baud (bits per second)
    pinMode(photopin, INPUT);    // initialize photopin as input
}

void loop() {
    int pinValue = analogRead(photopin); // read and store value from ADC
    Serial.print("Value= ");  // print text to Serial Monitor
    Serial.println(pinValue);  // print value to Serial Monitor, followed by return
    delay(500);  // To make the values more readable when looping
}
```

iv. Read through the code and try to understand what it’s doing. Utilize the Arduino reference manuals:

v. In the Arduino IDE, open the serial monitor by pressing the “Serial Monitor” button, or by pressing Ctrl+Shift+M.

vi. Adjust the amount of light on the photoresistor to observe the change in values displayed on the serial monitor. How do these numbers correlate to your readings on the DMM?

vii. Cover the photoresistor with your finger. Does this increase or decrease the output voltage? Why?

viii. Now build the circuit on the right in Figure 4 by switching R1 with R2. How does the output range change? Does the voltage increase or decrease when you cover the photoresistor with your finger?

ix. Which configuration do you think is better? Try different resistors.

x. Modify the code so that an LED turns on only when you cover the photoresistor with your finger. The LED should remain off otherwise.

5. Using a potentiometer to control digital outputs

Perhaps an easier way to control your output voltage is by creating a voltage divider using a potentiometer. Recall that a potentiometer is a three-terminal variable resistor. While the resistance between the outer terminals is always constant, you can shift the internal resistance back and forth by rotating the screw on the potentiometer, as demonstrated in Figure 11.
You can imagine the potentiometer as being two resistors $R_1$ and $R_2$ in series whose total resistance $R_1 + R_2$ is always equal to the nominal value of the potentiometer. With this in mind, you can use the potentiometer as a voltage divider.

i. Construct the circuit shown in Figure 12.

Copy the following code into the window and upload to the Arduino:

```c
/*
Blink with variable rate
Turns an LED on and off repeatedly at a rate varied by a potentiometer.
*/

int analogPin = 0;  // analog pin used to connect the potentiometer
int val;     // variable used to store the value from the analog pin

void setup() {
  pinMode(12, OUTPUT);    // initialize the digital pin as an output
}

void loop() {
  val = analogRead(analogPin);  // set val equal to the value read from analog pin 0
  digitalWrite(12, HIGH);     // set the LED on
  delay(val);                 // wait for val milliseconds
  digitalWrite(12, LOW);      // set the LED off
  delay(val);                 // wait for val milliseconds
}
```

![Figure 11: Shifting of resistance in potentiometer](http://www.allaboutcircuits.com/vol_1/chpt_6/1.html)

![Figure 12 - Circuit schematic for part 5](image)
ii. Take a moment to read and understand the code. Note that the variables *analogPin* and *val* can have almost any name you’d like them to have; they are just used to store values.

iii. The potentiometer acts as a voltage divider in this circuit. When you rotate the pot, the voltage on analog pin 0 varies from 0 to 5V. The *analogRead* function reads the voltage on the specified pin to a value between 0 and 1023, with 1023 corresponding to 5V.

iv. When the potentiometer is at 50% (rotated halfway), what is the voltage on analog pin 0? What is the value of the variable *val*?

v. Write a program that does the following *(show a TA)*:
   a. Turns LED 1 on and LED 2 off when the pot is rotated between 25% and 50%
   b. Turns LED 1 off and LED 2 on when the pot is rotated between 50% and 75%
   c. Flashes LED 1 at 5Hz and turns LED 2 off when the pot is rotated less than 25%
   d. Turns LED 1 off and flashes LED 2 at 5Hz when the pot is rotated more than 75%