Daffodil International University

Department of Computer Science and Engineering

Lab Manual

Version: 2017.04

Course Code: CSE 423
Course Title: Embedded Systems

Table of Contents

Session 1: Introduction to Arduino Programming

Intended Learning Outcome:

a. Get familiar with AVR-C Language
b. Write the very first code for Arduino Board
c. Gathering Idea about different syntax

Expected skills:

a. Basic knowledge on Programming (C/C++) b. Basic knowledge on Hardware

Tools Required:

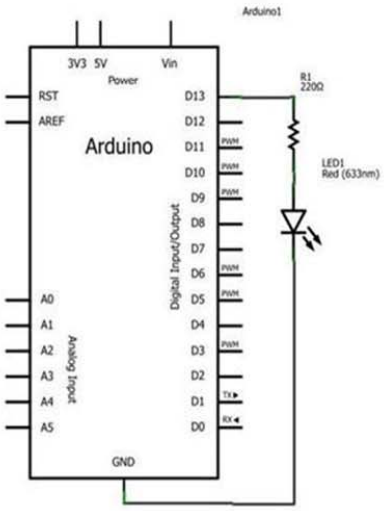
a. Arduino IDE
b. Arduino Uno Board
c. LED
d. 220 Ω Resistor

Session Detail:

This session is intended to show the simplest thing you can do with an Arduino to see physical output. It blinks an LED. To build the circuit, attach a 220-ohm resistor to pin 13. Then attach the long leg of an LED (the positive leg, called the anode) to the resistor. Attach the short leg (the negative leg, called the cathode) to ground. Then plug your Arduino board into your computer, start the Arduino program, and enter the code below.

Most Arduino boards already have an LED attached to pin 13 on the board itself. If you run this example with no hardware attached, you should see that LED blink. In our board you can see this blinking effect from on board LED.

Circuit Diagram:



Code:

In the program below, the first thing you do is to initialize pin 13 as an output pin with the line
pinMode (13, OUTPUT);
In the main loop, you turn the LED on with the line:
digitalWrite (13, HIGH);
This supplies 5 volts to pin 13. That creates a voltage difference across the pins of the LED, and lights it up. Then you turn it off with the line:
digitalWrite (13, LOW);
That takes pin 13 back to 0 volts, and turns the LED off. In between the on and the off, you want enough time for a person to see the change, so the delay() commands tell the Arduino to do nothing for 1000 milliseconds, or one second. When you use the delay () command, nothing else happens for that amount of time.

```
 /*
  Blink  Turns on an LED on for one second, then off for one second, repeatedly.
 */
 // Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int led = 13;
// the setup routine runs once when you press reset:
void setup()
{
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}


// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(led, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);               // wait for a second
  digitalWrite(led, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);               // wait for a second
}
```

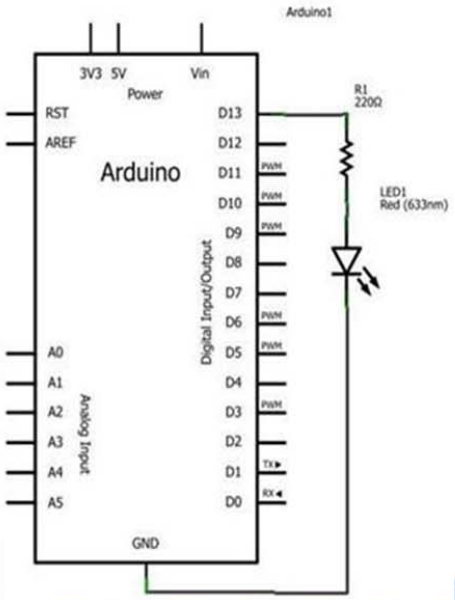
Post Lab Exercise:
You must try it at your home with different delay times.
Further Readings:
You can go through the https://circuits.io/ website for further practice via online.

Session 2: Arduino Programming (Blink without Delay)
Intended Learning Outcome:
a. Get familiar with AVR-C Language
b. Write the modified code for Arduino Uno Board based on 1st Lab
c. Gather Idea about different syntax
Expected skills:
a. Basic knowledge on Programming (C/C++) b. Basic knowledge on Hardware

Tools Required:
a. Arduino IDE
b. Arduino Uno Board
c. LED
d. 220 Ω Resistor

Session Detail:
Sometimes you need to do two things at once. For example you might want to blink an LED (or some other time-sensitive function) while reading a button press or other input. In this case, you can't use delay(), or you'd stop everything else the program while the LED blinked. The program might miss the button press if it happens during the delay(). This sketch demonstrates how to blink the LED without using delay(). It keeps track of the last time the Arduino turned the LED on or off. Then, each time through loop(), it checks if a long enough interval has passed. If it has, it toggles the LED on or off.
To build the circuit, grab a LED and attach it's long leg (positive) to pin 13. Attach the short, negative leg to ground (GND). Then plug your Arduino board into your computer, start the Arduino program, and enter the code below.

Circuit Diagram:



Code:

The code below uses the millis() function, a command that returns the number of milliseconds since the Arduino board started running its current program, to blink an LED.

```
 /* Blink without Delay
 Turns on and off a light emitting diode(LED) connected to a digital pin, without using the delay() function.  This means that other code can run at the same time without being interrupted by the LED code.
*/
// constants won't change. Used here to
// set pin numbers:
const int ledPin =  13;      // the number of the LED pin
// Variables will change:
int ledState = LOW;           // ledState used to set the LED
long previousMillis = 0;       // will store last time LED was updated
// the follow variables is a long because the time, measured in miliseconds,
// will quickly become a bigger number than can be stored in an int.
long interval = 1000;          // interval at which to blink (milliseconds)
void setup() {
  // set the digital pin as output:
```

```
  pinMode(ledPin, OUTPUT);
Serial.begin (9600);
}
void loop()
{
  // here is where you'd put code that needs to be running all the time.

  // check to see if it's time to blink the LED; that is, if the
  // difference between the current time and last time you blinked
  // the LED is bigger than the interval at which you want to
  // blink the LED.
 unsigned long currentMillis = millis();
 Serial.println(currentMillis);
  if(currentMillis - previousMillis > interval) {
    // save the last time you blinked the LED
    previousMillis = currentMillis;

    // if the LED is off turn it on and vice-versa:
    if (ledState == LOW)
      ledState = HIGH;
    else
      ledState = LOW;

    // set the LED with the ledState of the variable:
    digitalWrite(ledPin, ledState);
  }
}
```

Post Lab Exercise:
You must try to understand function of millis () at home..

Further Readings:
You can go through the https://circuits.io/ website for further practice via online.

Session 3: Button Interfacing

Intended Learning Outcome:

a. Get familiar with AVR-C Language
b. Write the modified code for Arduino Uno Board based on 2nd Lab
c. Gather Idea about different syntax

Expected skills:

a. Basic knowledge on Programming (C/C++) b. Basic knowledge on Hardware

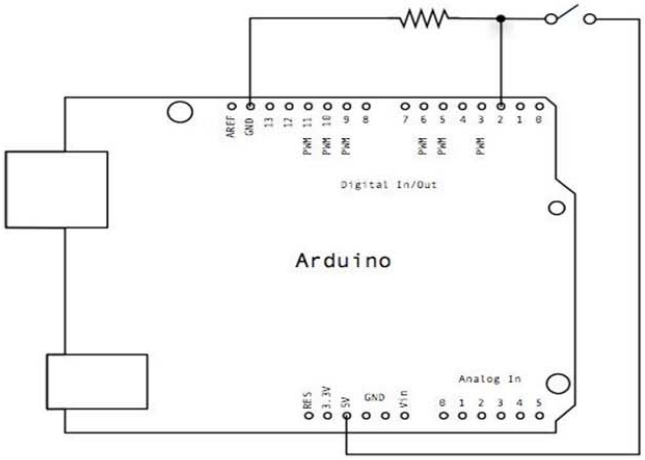
Tools Required:

a. Arduino IDE
b. Arduino Uno Board
c. Push button or switch
d. 10 kΩ resistor
e. Breadboard
f. Jumper Cable

Session Detail:

Connect three wires to the Arduino board. The first two, red and black, connect to the two long vertical rows on the side of the breadboard to provide access to the 5 volt supply and ground. The third wire goes from digital pin 2 to one leg of the pushbutton. That same leg of the button connects through a pull-down resistor (here 10 kΩ) to ground. The other leg of the button connects to the 5 volt supply. When the pushbutton is open (unpressed) there is no connection between the two legs of the pushbutton, so the pin is connected to ground (through the pull-down resistor) and we read a LOW. When the button is closed (pressed), it makes a connection between its two legs, connecting the pin to 5 volts, so that we read a HIGH.

Circuit Diagram:



Code:

```
/*
  Button
Turns on and off a light emitting diode (LED) connected to digital   pin 13, when
pressing a pushbutton attached to pin 2.
 */
// constants won't change. They're used here to
// set pin numbers:
const int buttonPin = 2;     // the number of the pushbutton pin
const int ledPin =  13;      // the number of the LED pin

// variables will change:
int buttonState = 0;         // variable for reading the pushbutton status

void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the pushbutton pin as an input:
  pinMode(buttonPin, INPUT);
}

void loop(){
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);

  // check if the pushbutton is pressed.
  // if it is, the buttonState is HIGH:
  if (buttonState == HIGH) {
    // turn LED on:
    digitalWrite(ledPin, HIGH);
  }
  else {
    // turn LED off:
    digitalWrite(ledPin, LOW);
```

```
 }
}
```

Post Lab Exercise:
Do the same exercise with the help of Proteus.
Further Readings:
You can go through the https://circuits.io/ website for further practice via online.

Session 4: Button State Change Detection (Edge Detection)

Intended Learning Outcome:
a. Get familiar with AVR-C Language
b. Write the modified code for Arduino Uno Board based on 3rd Lab
c. Gather Idea about different syntax
Expected skills:
a. Basic knowledge on Programming (C/C++) b. Basic knowledge on Hardware

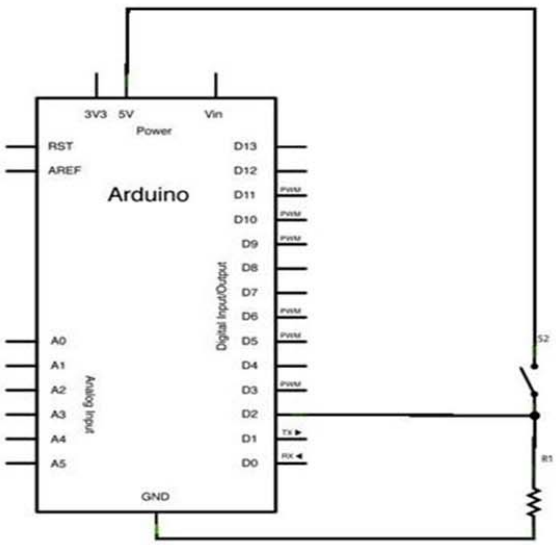
Tools Required:
a. Arduino IDE
b. Arduino Uno Board
c. Push button or switch
d. 10 KΩ resistor
e. Breadboard
f. Jumper Cable

Session Detail:
Connect three wires to the Arduino board. The first goes from one leg of the pushbutton through a pull-down resistor (here 10 kΩ) to ground. The second goes from the corresponding leg of the pushbutton to the 5 volt supply. The third connects to a digital i/o pin (here pin 2) which reads the button's state.

When the pushbutton is open (unpressed) there is no connection between the two legs of the pushbutton, so the pin is connected to ground (through the pull-down resistor) and we read a LOW. When the button is closed (pressed), it makes a connection between its two legs, connecting the pin to voltage, so that we read a HIGH. (The pin is still connected to ground, but the resistor resists the flow of current, so the path of least resistance is to +5V.)

If you disconnect the digital i/o pin from everything, the LED may blink erratically. This is because the input is "floating" - that is, not connected to either voltage or ground. It will more or less randomly return either HIGH or LOW. That's why you need a pull-down resistor in the circuit.

Circuit Diagram:



Code:

The sketch below continually reads the button's state. It then compares the button's state to its state the last time through the main loop. If the current button state is different from the last button state and the current button state is high, then the button changed from off to on. The sketch then increments a button push counter.

The sketch also checks the button push counter's value, and if it's an even multiple of four, it turns the LED on pin 13 ON. Otherwise, it turns it off.

```
/*
  State change detection (edge detection)
 This example shows how to detect when a button or button changes from off to on and on to off.
 */

// this constant won't change:
const int  buttonPin = 2;    // the pin that the pushbutton is attached to
const int ledPin = 13;       // the pin that the LED is attached to

// Variables will change:
int buttonPushCounter = 0;   // counter for the number of button presses
int buttonState = 0;         // current state of the button
int lastButtonState = 0;     // previous state of the button

void setup() {
  // initialize the button pin as a input:
  pinMode(buttonPin, INPUT);
  // initialize the LED as an output:
  pinMode(ledPin, OUTPUT);
  // initialize serial communication:
  Serial.begin(9600);
```

```
}

void loop() {
  // read the pushbutton input pin:
  buttonState = digitalRead(buttonPin);

  // compare the buttonState to its previous state
  if (buttonState != lastButtonState) {
    // if the state has changed, increment the counter
    if (buttonState == HIGH) {
      // if the current state is HIGH then the button
      // wend from off to on:
      buttonPushCounter++;
      Serial.println("on");
      Serial.print("number of button pushes:  ");
      Serial.println(buttonPushCounter);
    }
    else {
      // if the current state is LOW then the button
      // wend from on to off:
      Serial.println("off");
    }
  }
  // save the current state as the last state,
  //for next time through the loop
  lastButtonState = buttonState;


  // turns on the LED every four button pushes by
  // checking the modulo of the button push counter.
  // the modulo function gives you the remainder of
  // the division of two numbers:
  if (buttonPushCounter % 4 == 0) {
    digitalWrite(ledPin, HIGH);
  } else {
   digitalWrite(ledPin, LOW);
  }
```

}

Post Lab Exercise:
Simulate the same exercise with the help of Proteus.

Further Readings:
You can go through the https://www.arduino.cc/en/Reference/HomePage for further readings.

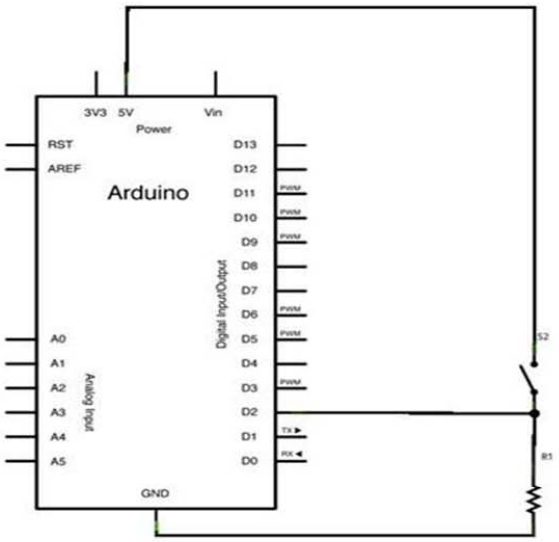
Session 5: De-Bounce Problem

Intended Learning Outcome:
a. Get familiar with AVR-C Language
b. Write the modified code for Arduino Uno Board based on 4th Lab
c. Gather Idea about different syntax
Expected skills:
a. Basic knowledge on Programming (C/C++) b. Basic knowledge on Hardware

Tools Required:
a. Arduino IDE
b. Arduino Uno Board
c. Push button or switch
d. 10 kΩ resistor
e. Breadboard
f. Jumper Cable

Session Detail:
This session demonstrates the use of a pushbutton as a switch: each time you press the button, the LED (or whatever) is turned on (if it's off) or off (if on). It also debounces the input, which means checking twice in a short period of time to make sure it's definitely pressed. Without debouncing, pressing the button once can appear to the code as multiple presses. Makes use of the millis() function to keep track of the time when the button is pressed.

Circuit Diagram:



Code:

```
/*
 Debounce
 Each time the input pin goes from LOW to HIGH (e.g. because of a push-button
press), the output pin is toggled from LOW to HIGH or HIGH to LOW.  There's a
minimum delay between toggles to debounce the circuit (i.e. to ignore noise).
 */
```

```
// constants won't change. They're used here to
// set pin numbers:
const int buttonPin = 2;     // the number of the pushbutton pin
const int ledPin =  13;      // the number of the LED pin

// Variables will change:
int ledState = HIGH;         // the current state of the output pin
int buttonState;             // the current reading from the input pin
int lastButtonState = LOW;   // the previous reading from the input pin

// the following variables are long's because the time, measured in miliseconds,
// will quickly become a bigger number than can be stored in an int.
long lastDebounceTime = 0;  // the last time the output pin was toggled
long debounceDelay = 50;    // the debounce time; increase if the output flickers

void setup() {
  pinMode(buttonPin, INPUT);
  pinMode(ledPin, OUTPUT);
}

void loop() {
  // read the state of the switch into a local variable:
  int reading = digitalRead(buttonPin);

  // check to see if you just pressed the button
  // (i.e. the input went from LOW to HIGH),  and you've waited
  // long enough since the last press to ignore any noise:

  // If the switch changed, due to noise or pressing:
  if (reading != lastButtonState) {
    // reset the debouncing timer
    lastDebounceTime = millis();
  }

  if ((millis() - lastDebounceTime) > debounceDelay) {
    // whatever the reading is at, it's been there for longer
```

```
    // than the debounce delay, so take it as the actual current state:
    buttonState = reading;
  }

  // set the LED using the state of the button:
  digitalWrite(ledPin, buttonState);

  // save the reading.  Next time through the loop,
  // it'll be the lastButtonState:
  lastButtonState = reading;
}
```

Post Lab Exercise:
You must try the session in online platform or with the board.
Further Readings:
You can go through the https://circuits.io/ website for further practice via online.

Session 6: Digital Read Serial (Monitor the state of a switch pc via USB)

Intended Learning Outcome:
a. Get familiar with AVR-C Language
b. Write the modified code for Arduino Uno Board based on 5th Lab
c. Gather Idea about different syntax
Expected skills:
a. Basic knowledge on Programming (C/C++) b. Basic knowledge on Hardware

Tools Required:
a. Arduino IDE
b. Arduino Uno Board
c. Push button or switch
d. 10 kΩ resistor
e. Breadboard
f. Jumper Cable

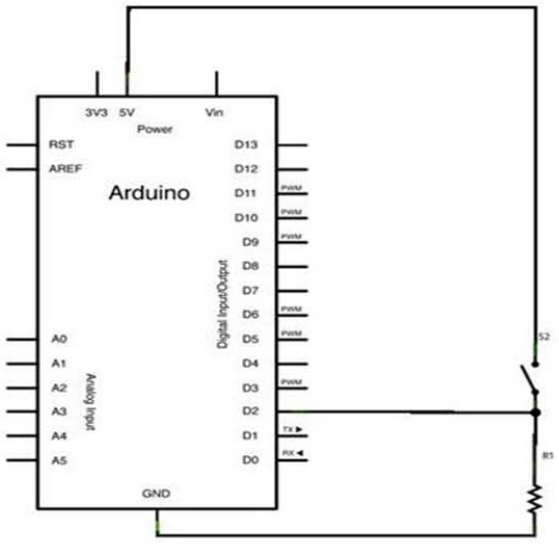
Session Detail:
Connect three wires to the Arduino board. The first two, red and black, connect to the two long vertical rows on the side of the breadboard to provide access to the

5 volt supply and ground. The third wire goes from digital pin 2 to one leg of the pushbutton. That same leg of the button connects through a pull-down resistor (here 10 kΩ) to ground. The other leg of the button connects to the 5 volt supply.

Pushbuttons or switches connect two points in a circuit when you press them. When the pushbutton is open (unpressed) there is no connection between the two legs of the pushbutton, so the pin is connected to ground (through the pull-down resistor) and reads as LOW, or 0. When the button is closed (pressed), it makes a connection between its two legs, connecting the pin to 5 volts, so that the pin reads as HIGH, or 1.

If you disconnect the digital i/o pin from everything, the LED may blink erratically. This is because the input is "floating" - that is, it doesn't have a solid connection to voltage or ground, and it will randomly return either HIGH or LOW. That's why you need a pull-down resistor in the circuit.

Circuit Diagram:



Code:

In the program below, the very first thing that you do will in the setup function is to begin serial communications, at 9600 bits of data per second, between your Arduino and your computer with the line:

Serial.begin (9600);

Next, initialize digital pin 2, the pin that will read the output from your button, as an input:

```
pinMode(2,INPUT);
```

Now that your setup has been completed, move into the main loop of your code. When your button is pressed, 5 volts will freely flow through your circuit, and when it is not pressed, the input pin will be connected to ground through the 10 kΩ resistor. This is a digital input, meaning that the switch can only be in either an on state (seen by your Arduino as a "1", or HIGH) or an off state (seen by your Arduino as a "0", or LOW), with nothing in between.

The first thing you need to do in the main loop of your program is to establish a variable to hold the information coming in from your switch. Since the information coming in from the switch will be either a "1" or a "0", you can use an int datatype. Call this variable sensorValue, and set it to equal whatever is being read on digital pin 2. You can accomplish all this with just one line of code:

```
int sensorValue = digitalRead(2);
```

Once the Arduino has read the input, make it print this information back to the computer as a decimal value. You can do this with the command Serial.println() in our last line of code:

```
Serial.println(sensorValue);
```

Now, when you open your Serial Monitor in the Arduino environment, you will see a stream of "0"s if your switch is open, or "1"s if your switch is closed.

```
/*
  DigitalReadSerial
 Reads a digital input on pin 2, prints the result to the serial monitor
 */

// digital pin 2 has a pushbutton attached to it. Give it a name:
int pushButton = 2;
```

```
// the setup routine runs once when you press reset:
void setup() {
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
  // make the pushbutton's pin an input:
  pinMode(pushButton, INPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  // read the input pin:
  int buttonState = digitalRead(pushButton);
  // print out the state of the button:
  Serial.println(buttonState);
  delay(1);        // delay in between reads for stability
}
```

Post Lab Exercise:
You must try the session in online platform or with the board.

Further Readings:
You can go through the https://www.arduino.cc/en/Reference/HomePage for further readings.

Session 7: Input Pullup Serial
Intended Learning Outcome:
a. Get familiar with AVR-C Language
b. Write the modified code for Arduino Uno Board based on 5th Lab
c. Gather Idea about different syntax
Expected skills:
a. Basic knowledge on Programming (C/C++) b. Basic knowledge on Hardware

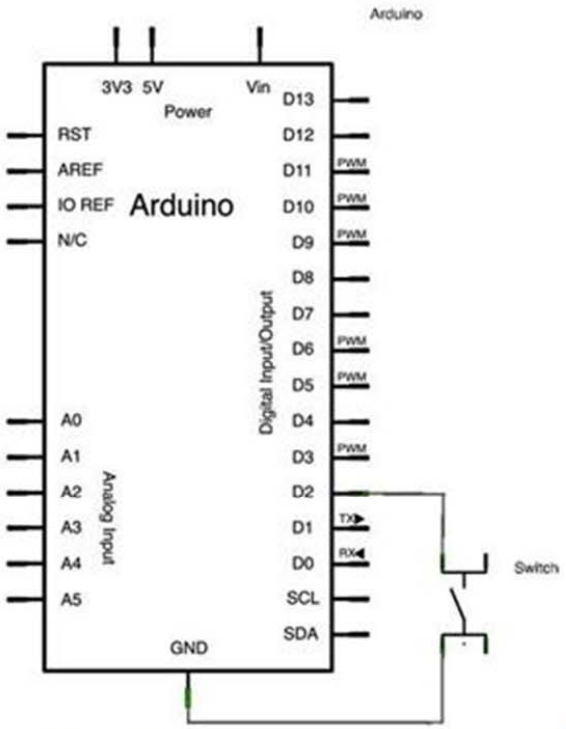
Tools Required:
a. Arduino IDE
b. Arduino Uno Board
c. Push button or switch
d. 10 kΩ resistor
e. Breadboard
f. Jumper Cable

Session Detail:
Connect three wires to the Arduino board. The first two, red and black, connect to the two long vertical rows on the side of the breadboard to provide access to the 5 volt supply and ground. The third wire goes from digital pin 2 to one leg of the pushbutton. That same leg of the button connects through a pull-down resistor (here 10 kΩ) to ground. The other leg of the button connects to the 5 volt supply.

Pushbuttons or switches connect two points in a circuit when you press them. When the pushbutton is open (unpressed) there is no connection between the two legs of the pushbutton, so the pin is connected to ground (through the pull-down resistor) and reads as LOW, or 0. When the button is closed (pressed), it makes a connection between its two legs, connecting the pin to 5 volts, so that the pin reads as HIGH, or 1.

If you disconnect the digital i/o pin from everything, the LED may blink erratically. This is because the input is "floating" - that is, it doesn't have a solid connection to voltage or ground, and it will randomly return either HIGH or LOW. That's why you need a pull-down resistor in the circuit.

Circuit Diagram:



Code:

In the program below, the very first thing that you do will in the setup function is to begin serial communications, at 9600 bits of data per second, between your Arduino and your computer with the line:

Serial.begin(9600);

Next, initialize digital pin 2 as an input with the internal pull-up resistor enabled:
pinMode(2, INPUT);
digitalWrite(2, HIGH);

The following line make pin 13, with the onboard LED, an output :
pinMode(13, OUTPUT);

Now that your setup has been completed, move into the main loop of your code. When your button not is pressed, the internal pull-up resistor connects to 5 volts. This causes the Arduino to report "1" or HIGH. When the button is pressed, the Arduino pin is pulled to ground, causing the Arduino report a "0", or LOW.

The first thing you need to do in the main loop of your program is to establish a variable to hold the information coming in from your switch. Since the information coming in from the switch will be either a "1" or a "0", you can use an int datatype. Call this variable sensorValue, and set it to equal whatever is being read on digital pin 2. You can accomplish all this with just one line of code:

```
int sensorValue = digitalRead(2);
```

Once the Arduino has read the input, make it print this information back to the computer as a decimal (DEC) value. You can do this with the command Serial.println() in our last line of code:

```
Serial.println(sensorValue, DEC);
```

Now, when you open your Serial Monitor in the Arduino environment, you will see a stream of "0"s if your switch is closed, or "1"s if your switch is open.

The LED on pin 13 will illuminate when the switch is HIGH, and turn off when LOW.

```
/*
 Input Pullup Serial
 This example demonstrates the use of pinMode(INPUT_PULLUP). It reads a
 digital input on pin 2 and prints the results to the serial monitor.
 */
void setup()
{
 //start serial connection
 Serial.begin(9600);
 //configure pin2 as an input and enable the internal pull-up resistor
 pinMode(2, INPUT);
 digitalWrite(2, HIGH);
 pinMode(13, OUTPUT);
}
```

```
void loop(){
  //read the pushbutton value into a variable
  int sensorVal = digitalRead(2);
  //print out the value of the pushbutton
  Serial.println(sensorVal);

  // Keep in mind the pullup means the pushbutton's
  // logic is inverted. It goes HIGH when it's open,
  // and LOW when it's pressed. Turn on pin 13 when the
  // button's pressed, and off when it's not:
  if (sensorVal == HIGH) {
    digitalWrite(13, LOW);
  }
  else {
    digitalWrite(13, HIGH);
  }
}
```

Post Lab Exercise:
You must try the session in online platform or with the board.

Further Readings:
You can go through the https://www.arduino.cc/en/Reference/HomePage for further readings.

Session 8: Analog Read Voltage (This example shows you how to read an analog input).
Intended Learning Outcome:
a. Get familiar with AVR-C Language
b. Write the modified code for Arduino Uno Board based on 7th Lab
c. Gather Idea about different syntax
Expected skills:
a. Basic knowledge on Programming (C/C++) b. Basic knowledge on Hardware

Tools Required:
a. Arduino IDE

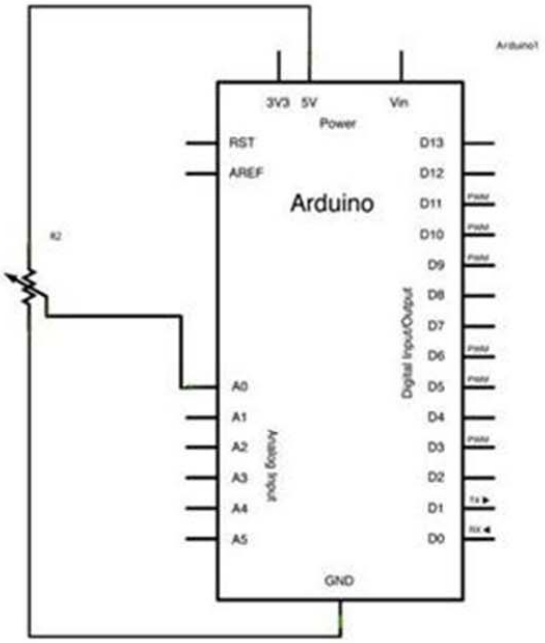
b. Arduino Uno Board
c. Variable resistor
d. Breadboard
e. Jumper Cable

Session Detail:

Connect the three wires from the potentiometer to your Arduino board. The first goes to ground from one of the outer pins of the potentiometer. The second goes from 5 volts to the other outer pin of the potentiometer. The third goes from analog input 2 to the middle pin of the potentiometer.

By turning the shaft of the potentiometer, you change the amount of resistance on either side of the wiper which is connected to the center pin of the potentiometer. This changes the voltage at the center pin. When the resistance between the center and the side connected to 5 volts is close to zero (and the resistance on the other side is close to 10 kΩ), the voltage at the center pin nears 5 volts. When the resistances are reversed, the voltage at the center pin nears 0 volts, or ground. This voltage is the analog voltage that you're reading as an input. The Arduino has a circuit inside called an analog-to-digital converter that reads this changing voltage and converts it to a number between 0 and 1023. When the shaft is turned all the way in one direction, there are 0 volts going to the pin, and the input value is 0. When the shaft is turned all the way in the opposite direction, there are 5 volts going to the pin and the input value is 1023. In between, analogRead() returns a number between 0 and 1023 that is proportional to the amount of voltage being applied to the pin.

Circuit Diagram:



Code:

In the program below, the very first thing that you do will in the setup function is to begin serial communications, at 9600 bits of data per second, between your Arduino and your computer with the line:

```
Serial.begin(9600);
```

Next, in the main loop of your code, you need to establish a variable to store the resistance value (which will be between 0 and 1023, perfect for an int datatype) coming in from your potentiometer:

```
int sensorValue = analogRead(A0);
```

To change the values from 0-1023 to a range that corresponds to the voltage the pin is reading, you'll need to create another variable, a float, and do a little math. To scale the numbers between 0.0 and 5.0, divide 5.0 by 1023.0 and multiply that by sensorValue :

```
float voltage= sensorValue * (5.0 / 1023.0);
```

Finally, you need to print this information to your serial window as. You can do this with the command Serial.println() in your last line of code:

Serial.println(voltage)

Now, when you open your Serial Monitor in the Arduino development environment (by clicking the button directly to the right of the "Upload" button in the header of the program), you should see a steady stream of numbers ranging from 0.0 - 5.0. As you turn the pot, the values will change, corresponding to the voltage coming into pin A0.

```
/*
  ReadAnalogVoltage
  Reads an analog input on pin 0, converts it to voltage, and prints the result to the serial monitor.
  Attach the center pin of a potentiometer to pin A0, and the outside pins to +5V and ground.

 */

// the setup routine runs once when you press reset:
void setup() {
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
}

// the loop routine runs over and over again forever:
void loop() {
  // read the input on analog pin 0:
  int sensorValue = analogRead(A0);
  // Convert the analog reading (which goes from 0 - 1023) to a voltage (0 - 5V):
  float voltage = sensorValue * (5.0 / 1023.0);
  // print out the value you read:
  Serial.println(voltage);
}
```

Post Lab Exercise:

You must try the session in online platform or with the board.

Further Readings:

You can go through the https://circuits.io/ website for further practice via online.

Session 9: Analog In, Out Serial

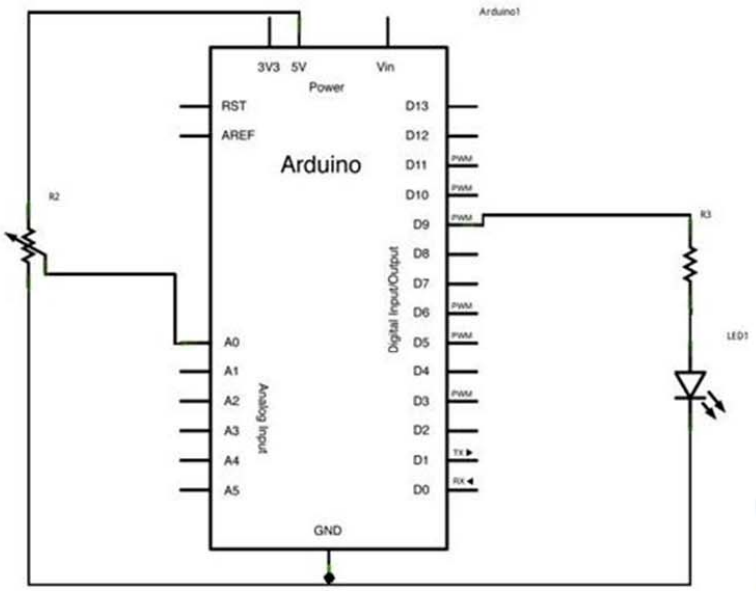
Intended Learning Outcome:
a. Get familiar with AVR-C Language
b. Write the modified code for Arduino Uno Board based on 7th Lab
c. Gather Idea about different syntax

Expected skills:
a. Basic knowledge on Programming (C/C++) b. Basic knowledge on Hardware

Tools Required:
a. Arduino IDE
b. Arduino Uno Board
c. Potentiometer
d. Breadboard
e. LED
f. 220 Ω resistor
g. Jumper Cable

Session Detail:
Connect one pin from your pot to 5V, the center pin to analog pin 0, and the remaining pin to ground. Next, connect a 220 ohm current limiting resistor to digital pin 9, with an LED in series. The long, positive leg (the anode) of the LED should be connected to the output from the resistor, with the shorter, negative leg (the cathode) connected to ground.

Circuit Diagram:



Code:

In the program below, after declaring two pin assignments (analog 0 for your potentiometer and digital 9 for your LED) and two variables, sensorValue and outputValue, the only thing that you do will in the setup function is to begin serial communication.

Next, in the main loop of the code, sensorValue is assigned to store the raw analog value coming in from the potentiometer. Because the Arduino has an analogRead resolution of 0-1023, and an analogWrite resolution of only 0-255, this raw data from the potentiometer needs to be scaled before using it to dim the LED.

In order to scale this value, use a function called map()

outputValue = map(sensorValue, 0, 1023, 0, 255);

outputValue is assigned to equal the scaled value from the potentiometer. map() accepts five arguments: The value to be mapped, the low range and high range of the raw data, and the low and high values for that data to be scaled too. In this

case, the sensor data is mapped down from its original range of 0 to 1023 to 0 to 255.

The newly mapped sensor data is then output to the analogOutPin dimming or brightening the LED as the potentiometer is turned. Finally, both the raw and scaled sensor values are sent to the Arduino serial window in a steady stream of data.

```
 /*
  Analog input, analog output, serial output
Reads an analog input pin, maps the result to a range from 0 to 255 and uses the
result to set the pulsewidth modulation (PWM) of an output pin. Also prints the
results to the serial monitor.
 */

// to the pins used:
const int analogInPin = A0;  // Analog input pin that the potentiometer is attached
to
const int analogOutPin = 9; // Analog output pin that the LED is attached to

int sensorValue = 0;        // value read from the pot
int outputValue = 0;        // value output to the PWM (analog out)

void setup() {
  // initialize serial communications at 9600 bps:
  Serial.begin(9600);
}
void loop() {
  // read the analog in value:
  sensorValue = analogRead(analogInPin);
  // map it to the range of the analog out:
  outputValue = map(sensorValue, 0, 1023, 0, 255);
  // change the analog out value:
  analogWrite(analogOutPin, outputValue);

  // print the results to the serial monitor:
  Serial.print("sensor = " );
```

```
  Serial.print(sensorValue);
  Serial.print("\t output = ");
  Serial.println(outputValue);

  // wait 2 milliseconds before the next loop
  // for the analog-to-digital converter to settle
  // after the last reading:
  delay(2);
}
```

Post Lab Exercise:
You must try the session in online platform or with the board.

Further Readings:
You can go through the https://www.arduino.cc/en/Reference/HomePage for further readings.

Session 10: Fading (Demonstrates the use of the analogWrite() function in fading an LED off and on)

Intended Learning Outcome:

a. Get familiar with AVR-C Language
b. Write the modified code for Arduino Uno Board based on 7th Lab
c. Gather Idea about different syntax

Expected skills:

a. Basic knowledge on Programming (C/C++) b. Basic knowledge on Hardware

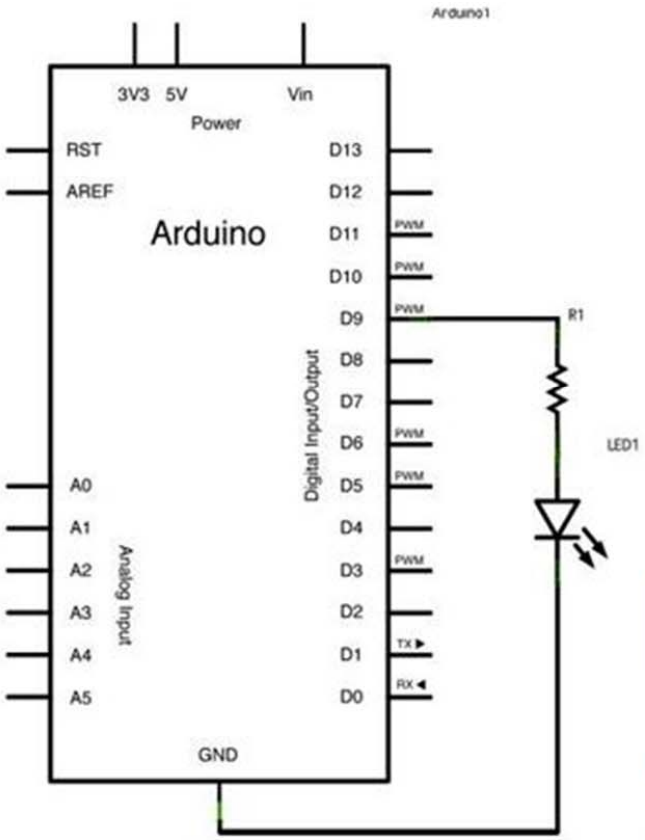
Tools Required:

a. Arduino IDE
b. Arduino Uno Board
c. Breadboard
d. LED
e. 220 Ω resistor
f. Jumper Cable

Session Detail:

Connect the anode (the longer, positive leg) of your LED to digital output pin 9 on your Arduino through a 220-ohm resistor. Connect the cathode (the shorter, negative leg) directly to ground.

Circuit Diagram:



Code:

After declaring pin 9 to be your ledPin, there is nothing to do in the setup() function of your code.

The analogWrite() function that you will be using in the main loop of your code requires two arguments: One telling the function which pin to write to, and one indicating what PWM value to write.

In order to fade your LED off and on, gradually increase your PWM value from 0 (all the way off) to 255 (all the way on), and then back to 0 once again to complete the cycle. In the sketch below, the PWM value is set using a variable called brightness. Each time through the loop, it increases by the value of the variable fadeAmount.

If brightness is at either extreme of its value (either 0 or 255), then fadeAmount is changed to its negative. In other words, if fadeAmount is 5, then it is set to -5. If

it's 55, then it's set to 5. The next time through the loop, this change causes brightness to change direction as well.

analogWrite() can change the PWM value very fast, so the delay at the end of the sketch controls the speed of the fade. Try changing the value of the delay and see how it changes the program.

```
/*
 Fade
 This example shows how to fade an LED on pin 9
 using the analogWrite() function.

 */

int led = 9;          // the pin that the LED is attached to
int brightness = 0;    // how bright the LED is
int fadeAmount = 5;    // how many points to fade the LED by

// the setup routine runs once when you press reset:
void setup()  {
  // declare pin 9 to be an output:
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop()  {
  // set the brightness of pin 9:
  analogWrite(led, brightness);

  // change the brightness for next time through the loop:
  brightness = brightness + fadeAmount;

  // reverse the direction of the fading at the ends of the fade:
  if (brightness == 0 || brightness == 255) {
    fadeAmount = -fadeAmount ;
  }
  // wait for 30 milliseconds to see the dimming effect
```

```
  delay(30);
}
```

Post Lab Exercise:
You must try the session in online platform or with the board.

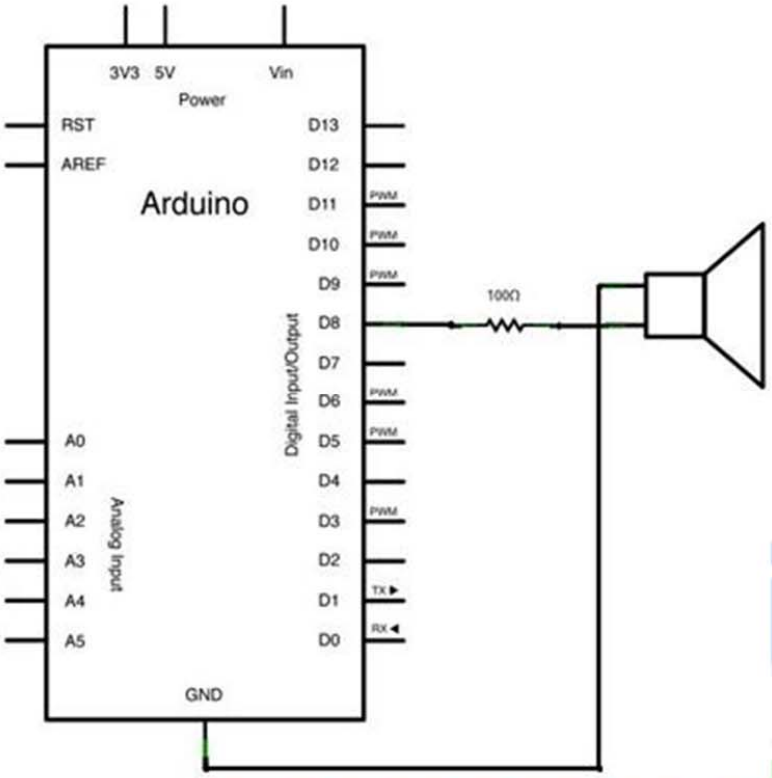
Further Readings:
You can go through the https://circuits.io/ website for further practice via online.

Session 11: Play a Melody (Play a Melody using the tone() function)
Intended Learning Outcome:
a. Get familiar with AVR-C Language
b. Write the modified code for Arduino Uno Board based on 7th Lab
c. Gather Idea about different syntax
Expected skills:
a. Basic knowledge on Programming (C/C++) b. Basic knowledge on Hardware

Tools Required:
a. Arduino IDE
b. Arduino Uno Board
c. Breadboard
d. 8 Ω small speaker
e. 100 Ω resistor
f. Jumper Cable

Session Detail:
Connect one terminal of your speaker to digital pin 8 through a 100 ohm resistor. Connect the other terminal to ground.

Circuit Diagram:



Code:

The code below uses an extra file, pitches.h. This file contains all the pitch values for typical notes. For example, NOTE_C4 is middle C. NOTE_FS4 is F sharp, and so forth. This note table was originally written by Brett Hagman, on whose work the tone() command was based. You may find it useful for whenever you want to make musical notes.

The main sketch is as follows:

```
/*
 Melody

 Plays a melody
 */
```

```
#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

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

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

void setup() {
  // iterate over the notes of the melody:
  for (int thisNote = 0; thisNote < 8; thisNote++) {

    // 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[thisNote];
    tone(8, melody[thisNote],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;
```

```
    delay(pauseBetweenNotes);
    // stop the tone playing:
    noTone(8);
  }
}
```

Post Lab Exercise:
You must try the session in online platform or with the board.

Further Readings:
You can go through the https://circuits.io/ website for further practice via online.