**A Project on**

# Programming with Arduino

By

**Durjoy Dutta Chaudhury**

# Table of Contents

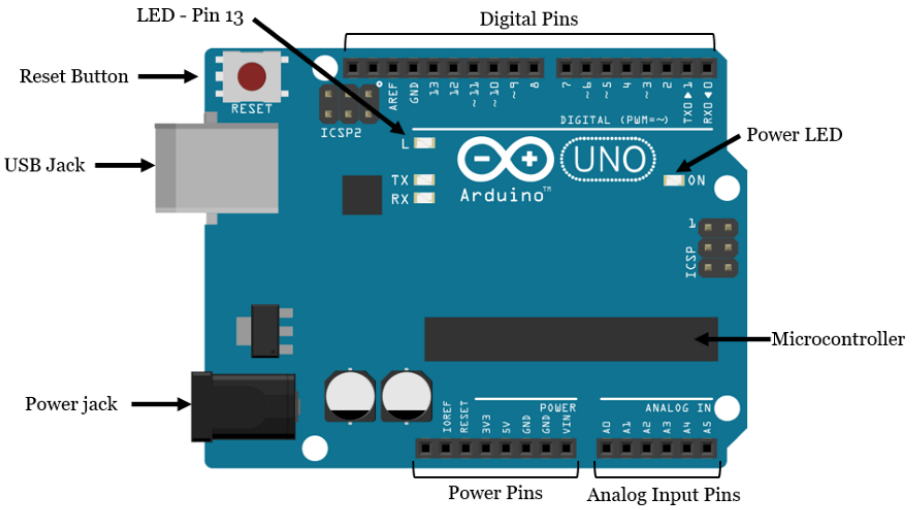
# 1 Introduction

## 1.1 Arduino

Arduino comprises of both a physical programmable circuit board (commonly known as a microcontroller) and a programming software, or IDE (Integrated Development Environment) that can be run on a PC, used to compose and transfer PC code to the circuit board. It can be done by using the Arduino programming language (based on Wiring), and the Arduino Software (IDE), based on Processing. Unlike other programmable circuit boards, the Arduino does not require a different equipment (called a software engineer) to upload code to the circuit board, one can essentially utilize a USB link. Also, the Arduino IDE utilizes a rearranged rendition of C++, making it simpler to figure out how to program. In a word, Arduino make the functions of the micro-controller into a more accessible package. The Uno is one of the more prevalent boards in the Arduino family and an extraordinary option for the beginners.

## 1.2 Components of the Arduino UNO R3 Board

There are different types of Arduino boards for different purposes. But all the boards have the majority of the following components in common. We are going to use an Arduino UNO R3 board for all of our projects.




Starting clockwise from the top center:

- Analog Reference pin (orange)
- Digital Ground (light green)
- Digital Pins 2-13 (green)

- Digital Pins 0-1/Serial In/Out - TX/RX (dark green
- Reset Button - S1 (dark blue)
- In-circuit Serial Programmer (blue-green)
- Analog In Pins 0-5 (light blue)
- Power and Ground Pins (power: orange, grounds: light orange)
- External Power Supply In (9-12VDC) - X1 (pink)
- Toggles External Power and USB Power (place jumper on two pins closest to desired supply) - SV1 (purple)
- USB (used for uploading sketches to the board and for serial communication between the board and the computer; can be used to power the board) (yellow)

- These pins cannot be used for digital i/o (digitalRead and digitalWrite) if serial communication is also being used (e.g. Serial.begin).

### 1.2.1 Digital Pins

The digital pins on an Arduino board can be used for general purpose input and output via the pinMode(), digitalRead(), and digitalWrite() commands. Each pin has an internal pull-up resistor which can be turned on and off using digitalWrite() (w/ a value of HIGH or LOW, respectively) when the pin is configured as an input.

- Serial: 0 (RX) and 1 (TX). Used to receive (RX) and transmit (TX) TTL serial data.[1]
- External Interrupts: 2 and 3. These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value.
- PWM: 3, 5, 6, 9, 10, and 11. Provide 8-bit PWM output with the analogWrite() function. On boards with an ATmega8, PWM output is available only on pins 9, 10, and 11

### 1.2.2 Analog Pins

The analog input pins support 10-bit analog-to-digital conversion (ADC) using the analogRead() function. Most of the analog inputs can also be used as digital pins: analog input 0 as digital pin 14 through analog input 5 as digital pin 19.

### 1.2.3 Power Pins

- **9V:** The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). Different boards accept different input voltages ranges.

- **5V:** The regulated power supply used to power the microcontroller and other components on the board. This can come either from VIN via an on-board regulator, or be supplied by USB or another regulated 5V supply.
- **3.3V:** (Diecimila-only) A 3.3 volt supply generated by the on-board FTDI chip.
- **GND:** Ground pins.

# 2 Blinking & Fading LED

## 2.1 Blinking LED

LEDs are small, powerful lights that are used in many different applications. To start, we will work on blinking an LED, the Hello World of microcontrollers. It is as simple as turning a light on and off. Establishing this important baseline will give you a solid foundation as we work towards experiments that are more complex.

### 2.1.1 Experiment URL

Use this link to get the **Tinkercad** simulation of this experiment.

### 2.1.2 Objectives

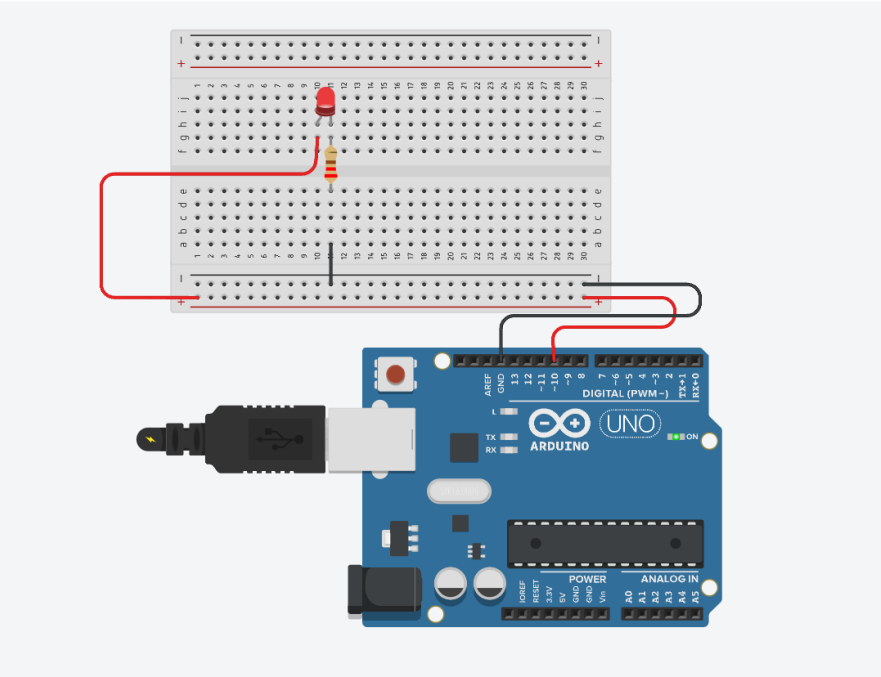
In this lesson, we will program the Arduino's GPIO output high level and low level (0V), and then make the LED which is connected to the Arduino's GPIO flicker with a certain frequency

### 2.1.3 Necessary Components

We will need the following components −

- 1 x Breadboard
- 1 x Arduino Uno R3
- 1 x LED
- 1 x 330 Ω Resistor
- 2 × Jumper

### 2.1.4 Circuit Diagram




### 2.1.5 Arduino Code

```
1     /* Blinking LED    */
2
3 int  ledPin = 10; //Initialize digital pin 10 set as LED
    output pin
4 int gap = 500; // Wait for 500 millisecond(s)
5
6 void setup()
7 {
8   pinMode(ledPin, OUTPUT);
9 }
10
11 void loop()
12 {
13   // turn the LED on (HIGH is the voltage level)
14   digitalWrite(ledPin, HIGH);
15   delay(gap);
16   // turn the LED off by making the voltage LOW
17   digitalWrite(ledPin, LOW);
18   delay(gap);
19 }
20
```

## 2.2 Fading LED

### 2.2.1 Experiment URL

Use this link to get the **Tinkercad** simulation of this experiment.

### 2.2.2 Objectives

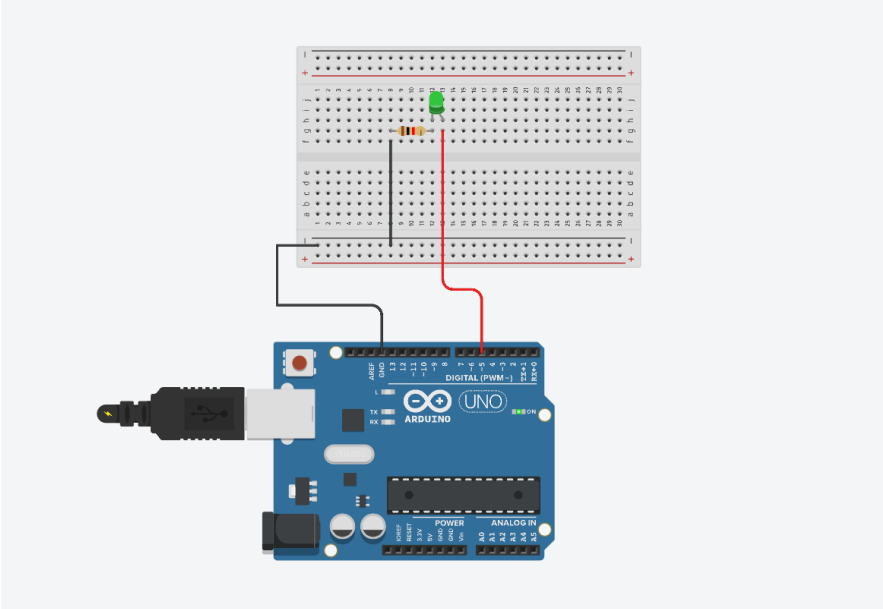
This experiment demonstrates the use of the analogWrite() function in fading an LED off and on. AnalogWrite uses pulse width modulation (PWM), turning a digital pin on and off very quickly with different ratio between on and off, to create a fading effect.

### 2.2.3 Necessary Components

We will need the following components –

- 1 x Breadboard
- 1 x Arduino Uno R3
- 1 x LED
- 1 x 330 Ω Resistor
- 2 × Jumper

### 2.2.4 Schematic



### 2.2.5 Arduino Code

```
/*Fading LED*/

const int pinLED = 5; //declaring PWM pin 5 as LED output
int brightness = 0 ; //declaring LED initial condition
int fadeAmount = 5; // how many points to fade the LED by

void setup()
{
  pinMode(pinLED, OUTPUT); //initializing PWM pin 5 as LED
   output
  Serial.begin(9600);
}

void loop(){

  for (brightness = 0; brightness <= 255; brightness +=
   fadeAmount){
  analogWrite(pinLED,brightness);
  delay(50); //interval for the brightenig effect
  }

  delay(500);  //wait for 500ms to run the loop again

  for (brightness = 255; brightness >= 0; brightness -=
   fadeAmount){
  analogWrite(pinLED,brightness);
  delay(50); //interval for the dimming effect
  }
  delay(500); //wait for 500ms to run the loop again
}
```

# 3 Measurement of voltages (Below 5V and above)

## 3.1 Experiment URL

Use this link to get the **Tinkercad** simulation of this experiment.

## 3.2 Objectives

We are going to make a simple digital voltmeter using the Arduino Uno R3, which can safely measure input dc voltages in 0 to 30V range. The Arduino board can be powered from a standard 9V battery pack or you can just use your desktop's USB port, as usual.

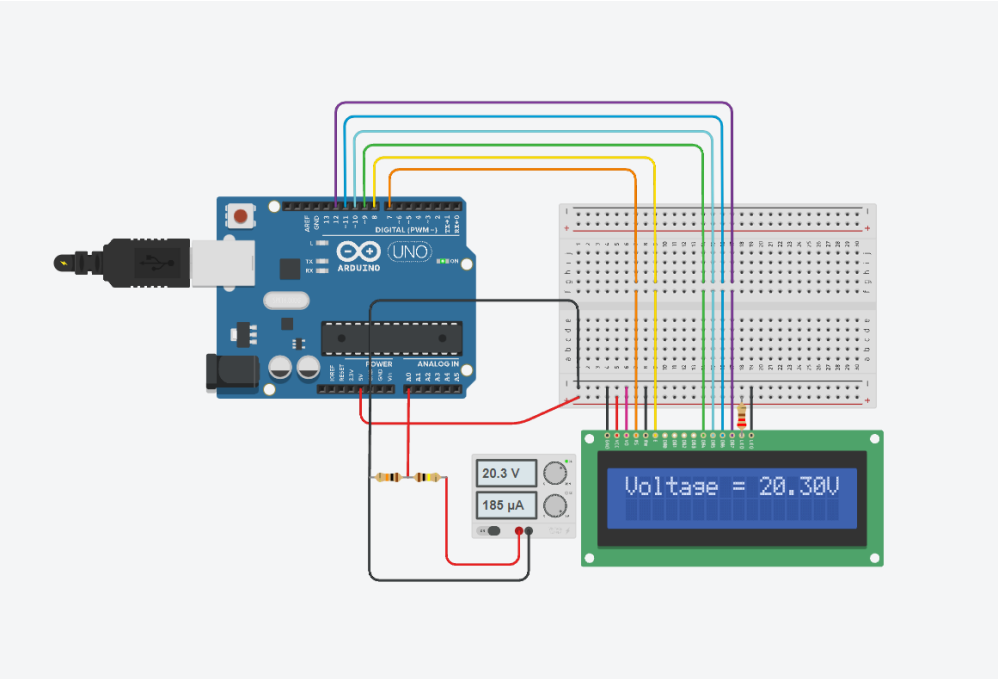
As you may well know, Arduino's analog inputs can be used to measure DC voltage between 0 and 5V (when using the standard 5V analog reference voltage) and this range can be increased by using two resistors to create a voltage divider. The voltage divider decreases the voltage being measured to within the range of the Arduino analog inputs. Code in the Arduino sketch is then used to compute the actual voltage being measured.

## 3.3 Necessary Components

We will need the following components −

- 1 x Arduino Uno R3
- 1 x Breadboard
- 1 x 30V Power Supply
- 1 x 1kΩ Resistor
- 1 x 5kΩ Resistor
- 1 x 16x2 LCD screen
- 1 x USB A-B cable
- Jumper Wires

## 3.4 Circuit Diagram




## 3.5 Formula for calculating voltage:

$$V_0 = (Val * 5.0)/1023.00 \tag{1}$$

Here in these formula Val is the value that is read by Arduino as analog input, which is further multiplied by the voltage that is been supplied by Arduino and thus to get the Vout it is divided by the cycle of time that is covered after every bit to get the value.

$$V_{in} = V_0 \ / \ \frac{R2}{(R1 + R2)} \tag{2}$$

Using this formula we can convert the high voltages into the voltage range of Arduino (0 - 5V) using the voltage divider and measure any voltage using suitable resistor combinations.

NOTE: Here there is no restriction on using the specified amount of resistors, one can vary it according to the availability of the resistors.

## 3.6 Arduino Code

```
/*Analog voltage measurement using Arduino*/

#include <LiquidCrystal.h>

//initialize the library by associating needed LCD interface pin
const int rs = 7, en = 8, d4 = 9, d5 = 10, d6 = 11, d7 = 12;

LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

int analogIn = A0;
float v0 = 0.0;
float vin = 0.0;
float R1 = 5000.0; // resistance of R1 (5K)
float R2 = 1000.0; // resistance of R2 (1K)
int value;
void setup()
  {
   pinMode(analogIn, INPUT);
   lcd.begin(16, 2);
   lcd.setCursor(4,0);
   lcd.print("Voltmeter");
   delay(1000);
    }

void loop()
  {
  value = analogRead(analogIn);  //read the value at analog input pin
  v0 = (value * 5.0) / 1023.0; // voltage value as read by A0
  vin = v0 / (R2/(R1+R2)); // voltage reverted back to original

  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Voltage = ");
  lcd.print(vin);
  lcd.print("V");
  delay(100);
}

```

# 4 Interfacing a seven segment display

## 4.1 Experiment URL

Use this link to get the **Tinkercad** simulation of this experiment.

## 4.2 Objectives

## 4.3 Necessary Components

We will need the following components –

- 1 x seven segment display (common cathode)
- 1 x Arduino UNO R3
- 1 x breadboard
- 1 x 1kΩ resistors (1/4 W)
- jumper wires

## 4.4 Circuit Diagram

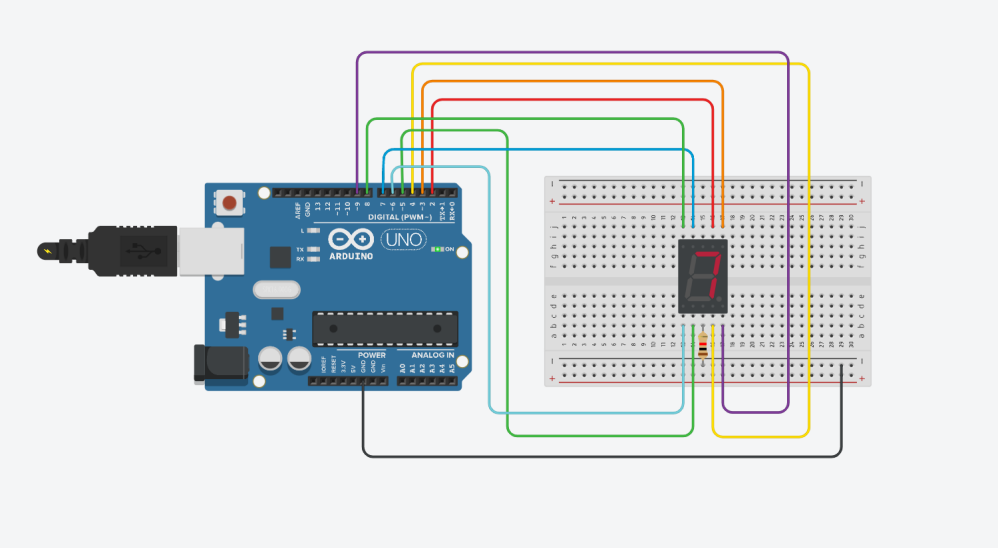



Figure 1: Interfacing seven segment display with Arduino

## 4.5 Arduino Code

```
/*Interfacing a seven segment display using Arduino*/
int i;
int pinA = 2;
int pinB = 3;
int pinC = 4;
int pinD = 5;
int pinE = 6;
int pinF = 7;
int pinG = 8;
int pinDP = 9;

void setup()
{
  for ( i = 2; i <= 9; i++){
  pinMode(i, OUTPUT);
}
}

void loop()
{
  // display 0
  digitalWrite(2, HIGH);
  digitalWrite(3, HIGH);
  digitalWrite(4, HIGH);
  digitalWrite(5, HIGH);
  digitalWrite(6, HIGH);
  digitalWrite(7, HIGH);
  digitalWrite(8, LOW);
  digitalWrite(9, LOW);
  delay(1000); // Wait for 1000 millisecond(s)
  // display 1
  digitalWrite(2, LOW);
  digitalWrite(3, HIGH);
  digitalWrite(4, HIGH);
  digitalWrite(5, LOW);
  digitalWrite(6, LOW);
  digitalWrite(7, LOW);
  digitalWrite(8, LOW);
  digitalWrite(9, LOW);
  delay(1000); // Wait for 1000 millisecond(s)
  // display 2
  digitalWrite(2, HIGH);
  digitalWrite(3, HIGH);
  digitalWrite(4, LOW);
  digitalWrite(5, HIGH);
  digitalWrite(6, HIGH);
  digitalWrite(7, LOW);
  digitalWrite(8, HIGH);
```

```
49    digitalWrite(9, LOW);
50    delay(1000); // Wait for 1000 millisecond(s)
51    // display 3
52    digitalWrite(2, HIGH);
53    digitalWrite(3, HIGH);
54    digitalWrite(4, HIGH);
55    digitalWrite(5, HIGH);
56    digitalWrite(6, LOW);
57    digitalWrite(7, LOW);
58    digitalWrite(8, HIGH);
59    digitalWrite(9, LOW);
60    delay(1000); // Wait for 1000 millisecond(s)
61    // display 4
62    digitalWrite(2, LOW);
63    digitalWrite(3, HIGH);
64    digitalWrite(4, HIGH);
65    digitalWrite(5, LOW);
66    digitalWrite(6, LOW);
67    digitalWrite(7, HIGH);
68    digitalWrite(8, HIGH);
69    digitalWrite(9, LOW);
70    delay(1000); // Wait for 1000 millisecond(s)
71    // display 5
72    digitalWrite(2, HIGH);
73    digitalWrite(3, LOW);
74    digitalWrite(4, HIGH);
75    digitalWrite(5, HIGH);
76    digitalWrite(6, LOW);
77    digitalWrite(7, HIGH);
78    digitalWrite(8, HIGH);
79    digitalWrite(9, LOW);
80    delay(1000); // Wait for 1000 millisecond(s)
81    // display 6
82    digitalWrite(2, HIGH);
83    digitalWrite(3, LOW);
84    digitalWrite(4, HIGH);
85    digitalWrite(5, HIGH);
86    digitalWrite(6, HIGH);
87    digitalWrite(7, HIGH);
88    digitalWrite(8, HIGH);
89    digitalWrite(9, LOW);
90    delay(1000); // Wait for 1000 millisecond(s)
91    // display 7
92    digitalWrite(2, HIGH);
93    digitalWrite(3, HIGH);
94    digitalWrite(4, HIGH);
95    digitalWrite(5, LOW);
96    digitalWrite(6, LOW);
97    digitalWrite(7, LOW);
98    digitalWrite(8, LOW);
```

```
 99    digitalWrite(9, LOW);
100    delay(1000); // Wait for 1000 millisecond(s)
101    // display 8
102    digitalWrite(2, HIGH);
103    digitalWrite(3, HIGH);
104    digitalWrite(4, HIGH);
105    digitalWrite(5, HIGH);
106    digitalWrite(6, HIGH);
107    digitalWrite(7, HIGH);
108    digitalWrite(8, HIGH);
109    digitalWrite(9, LOW);
110    delay(1000); // Wait for 1000 millisecond(s)
111    // display 9
112    digitalWrite(2, HIGH);
113    digitalWrite(3, HIGH);
114    digitalWrite(4, HIGH);
115    digitalWrite(5, LOW);
116    digitalWrite(6, LOW);
117    digitalWrite(7, HIGH);
118    digitalWrite(8, HIGH);
119    digitalWrite(9, LOW);
120    delay(1000); // Wait for 1000 millisecond(s)
121 }
```

# 5 DHT22 Temperature and Humidity Sensor

## 5.1 Experiment URL

Use this link to get the **Wokwi** simulation of this experiment.

## 5.2 Objectives

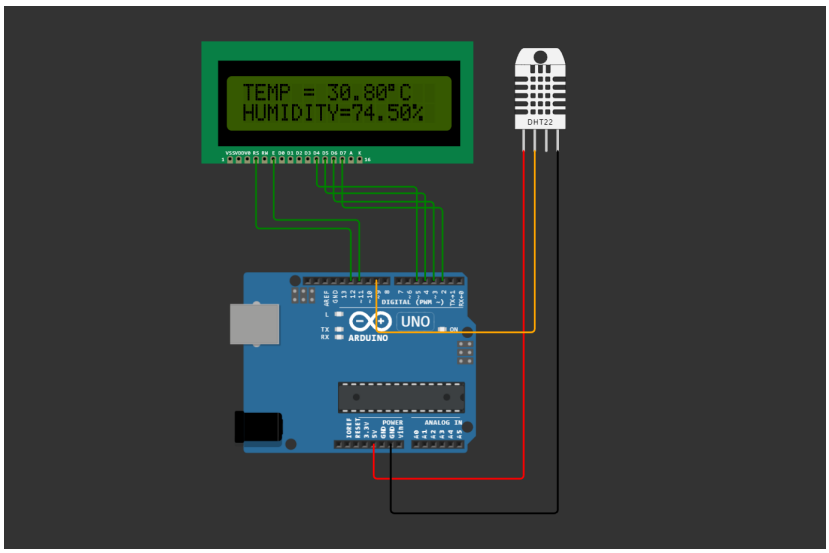
DHT22 is a Humidity and Temperature Sensor, which generates calibrated digital output. DHT22 can be interface with any microcontroller like Arduino, Raspberry Pi, etc. and get instantaneous results. DHT22 is a low cost humidity and temperature sensor which provides high reliability and long term stability. In this project, we will build a small circuit to interface Arduino with DHT22 Temperature and Humidity Sensor. One of the main applications of connecting DTH22 sensor with Arduino is weather monitoring.

## 5.3 Necessary Components

We will need the following components −

- 1 x Arduino Uno R3
- 1 x Breadboard
- 1 x DHT22 temperature and humidity sensor
- 1 x 10k Ω potentiometer
- 1 x 16x2 LCD screen
- 1 x USB A-B cable
- Jumper Wires

## 5.4 Circuit Diagram




## 5.5 Arduino Code

```
#include <LiquidCrystal.h>
#include <DHT.h>

#define DHTPIN  9
#define DHTTYPE DHT22


LiquidCrystal lcd(12, 11, 5, 4, 3, 2); // initialize the
    library with the numbers of the interface pins
DHT dht(DHTPIN, DHTTYPE); //initialize DHT sensor pin


void setup() {
  dht.begin();
  lcd.begin(16, 2); // set up the LCD's number of columns and
     rows

  lcd.setCursor(0,0);

  lcd.print("Initializing....");
  delay(1000);

}

void loop() {
  float temp = dht.readTemperature();
//float tempF = dht.readTemperature(true);
  float hum = dht.readHumidity();

```

```
28    lcd.clear();
29
30    lcd.setCursor(0,1);
31    lcd.print("HUMIDITY=");
32    lcd.print(hum);                  //Displaying humidity
33    lcd.print("%");
34
35    lcd.setCursor(0,0);
36    lcd.print("TEMP = ");
37    lcd.print(temp);                  //Displaying temperature in C
38    lcd.print((char)223);
39    lcd.print("C");
40    delay(2000);
41 }
42
43
```

# 6 Data logger for studying charging and discharging of RC circuit

## 6.1 Experiment URL

Use this link to get the **Tinkercad** simulation of this experiment.

## 6.2 Objectives

If a capacitor of capacitance C (in farads), initially charged to a potential $V_0$ (volts) is connected across a resistor R (in ohms), a time-dependent current will flow according to Ohm's law. This situation is shown by the RC (resistor-capacitor) circuit below when the switch is closed. As the current flows, the charge q is depleted, reducing the potential across the capacitor, which in turn reduces the current. This process creates an exponentially increasing voltage, modeled by

$$V_C(t) = V_0(1e^{-\frac{t}{R.C}}) \tag{3}$$

The rate of the decrease is determined by the product RC, known as the time constant of the circuit. A large time constant means that the capacitor will discharge slowly. The same time constant RC describes the rate of charging as well as the rate of discharging. The graph shows how the voltage across the capacitor $V_C$ and the voltage across the resistor $V_R$ vary with time when charging. The relationships are found by integrating the expression for the voltage on a capacitor. The voltage over the resistor while charging is given by:

$$V_R(t) = V_0(e^{-\frac{t}{R.C}}) \tag{4}$$

We will be using a square wave voltage supply to provide the 'switching' between open and closed for our RC circuit. We will then examine the plots in LoggerPro of the voltages over both the capacitor and the resistor for the charging and discharging of the RC circuit. We will compare the curve fits to the theoretical predictions for the voltage and the time constant for the circuit.

The main objective of this Arduino investigation of the RC circuit is the experimental verification of the formulas for the time-varying voltage and electric current for the charging or discharging capacitor. A secondary objective is to provide an alternative to more expensive commercial laboratory materials and, at the same time, to expose the students to the underlying electronics and computer programming that would have otherwise been hidden from sight.

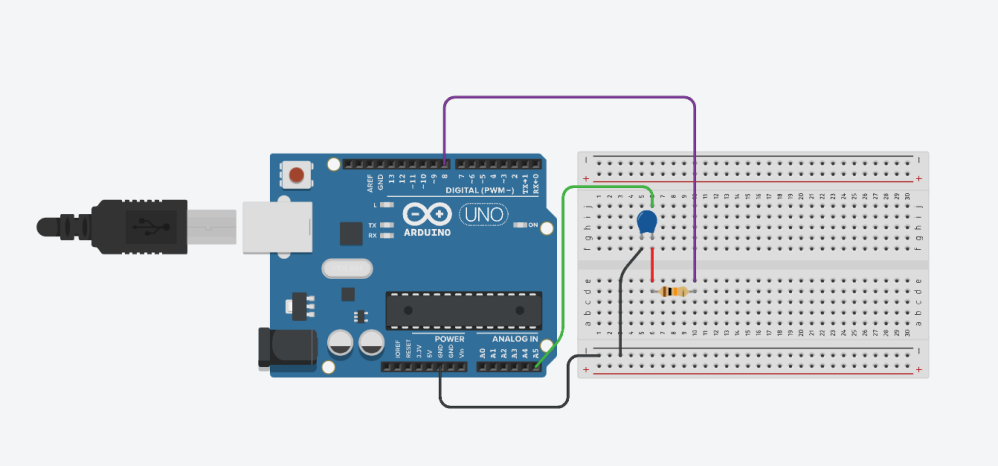
## 6.3 Necessary Components

We will need the following components –

- 1 x Arduino Uno R3

- 1 x Breadboard
- 1 x 100 $\mu$F capacitor
- 1 x 10k $\Omega$ resistor
- 1 x USB A-B cable
- Jumper Wires

## 6.4 Circuit Diagram




## 6.5 Arduino Code

```
1 /*DataLogger for charging and discharging of the RC Circuit
    */
2 int chargePin = 8;
3 int capPin = A5;
4
5 const float C = 100.0; //capacitance in uF
6 const float R = 10.0; //capacitance in kOhms
7 const float timeConstant = (R*C)/1000;
8
9 float startTimeCharge;
10 float startTimeDischarge;
11 float elapsedTimeCharge;
12 float elapsedTimeDischarge;
13 float rawInput;
14 float voltage;
15 float percentCharge;
16
17 void setup() {
18   Serial.begin(9600);
19   Serial.print("The time constant of the RC circuit is: ");
20   Serial.print(timeConstant);
```

```
21   Serial.println("s");
22
23   pinMode(chargePin, OUTPUT);
24   digitalWrite(chargePin, LOW);
25 }
26
27 void loop() {
28   rawInput = analogRead(capPin);
29   voltage = (rawInput/1024.0)*5;
30
31   /////////Charging////////////
32
33   digitalWrite(chargePin, HIGH);
34   startTimeCharge = millis(); //
35
36
37   while (rawInput < 1023.0) {
38   rawInput = analogRead(capPin);
39   voltage = (rawInput/1024.0)*5;
40   float elapsedTimeCharge = (millis() - startTimeCharge) /
      1000.0;  //
41   Serial.print(voltage, 3); // voltage in Volts
42   Serial.print("\t");
43   Serial.print(elapsedTimeCharge, 3); //time in seconds
44   Serial.print("\n");
45   delay(100);
46   }
47
48  /////////Discharging////////////
49
50   digitalWrite(chargePin, LOW);
51   startTimeDischarge = millis(); //
52
53
54   while (rawInput > 0.0) {
55   rawInput = analogRead(capPin);
56   voltage = (rawInput/1023.0)*5;
57   float elapsedTimeDischarge = (millis() - startTimeDischarge
      ) / 1000.0;
58   Serial.print(voltage, 3); // voltage in Volts
59   Serial.print("\t");
60   Serial.print(elapsedTimeDischarge, 3); //time in seconds
61   Serial.print("\n");
62   delay(100);
63   }
64 }
```

## 6.6 Charging and discharging plot

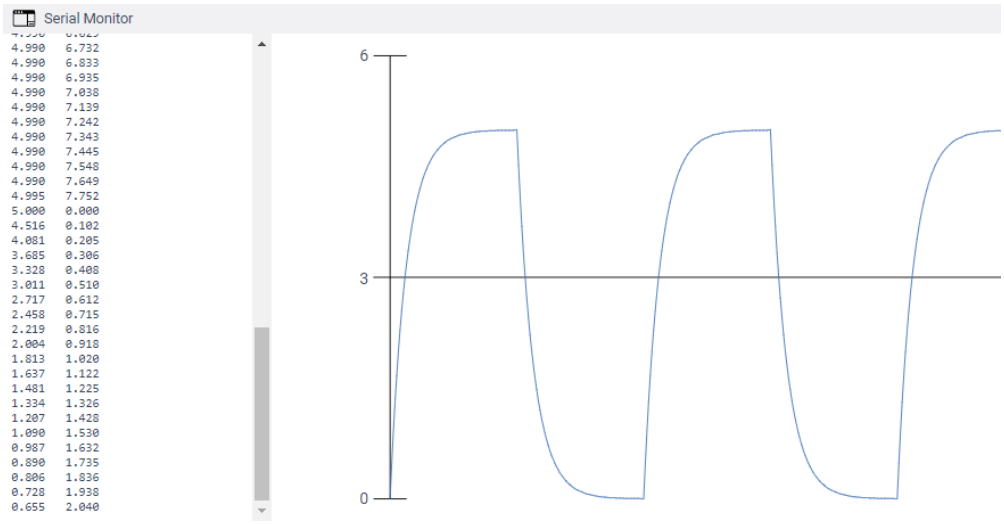



Figure 2: Data logging real time charging and discharging

We copied and pasted the two columns (voltage and time) of the serial monitor in a text file and plotted the data using **Python**'s plotting library **Matplotlib**.

```
import matplotlib.pyplot as plt
import numpy as np

voltage = []
time = []

#X, Y = np.loadtxt("capacitor_charging.txt", delimiter=',', unpack=True)
f = open("capacitor_discharging.txt","r")
i = []

for i in f:
    i = i.replace('\n', ' ')
    i = i.replace('\t', ' ')
    i = i.split()
    voltage.append(float(i[0]))
    time.append(float(i[1]))
print(voltage)
print(time)
plt.plot(time, voltage, "--", label = "voltage vs time")
plt.title('Capacitor Disharging')
plt.xlabel('Time (s)')
plt.ylabel('Voltage (V)')
plt.legend()
plt.show()
```

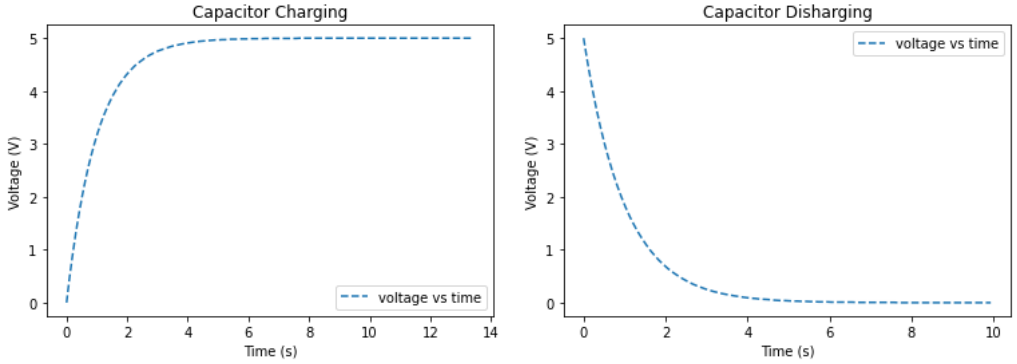



Figure 3: RC Charging and discharging plot