LEARNING DOCUMENT FOR MRUDHU ELECTRONICS ARDUINO UNO R3 KIT

Chapter 1 - Blink

Description:

Turn an LED on and off every second.

This example shows the simplest thing you can do with an Arduino to see physical output: it blinks the on-board LED.

Hardware Required

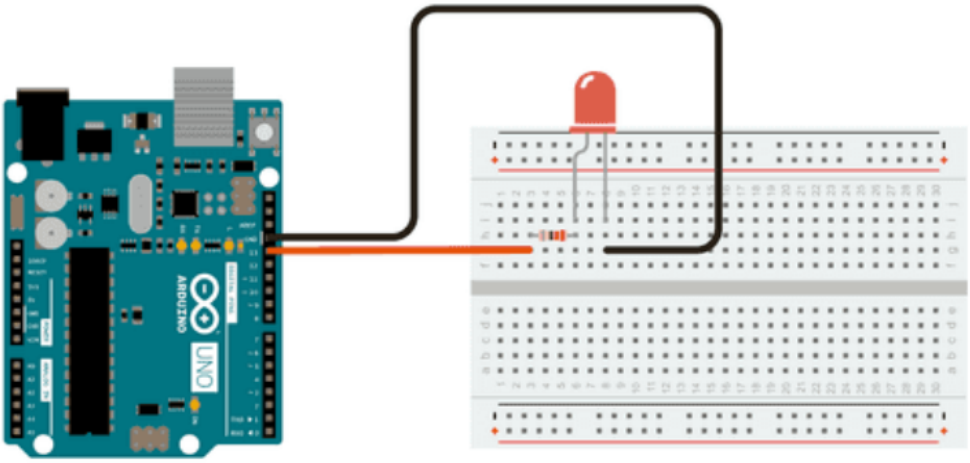
- Arduino Board
- LED (optional)
- 220 ohm resistor (optional)

Circuit:

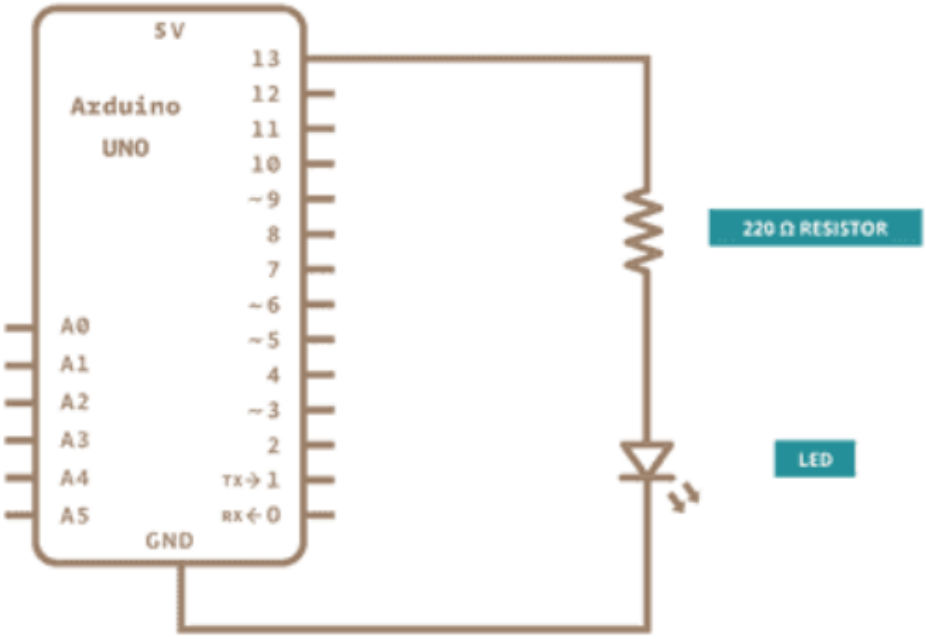
This example uses the built-in LED that most Arduino boards have. This LED is connected to a digital pin and its number may vary from board type to board type. To make your life easier, we have a constant that is specified in every board descriptor file. This constant is LED_BUILTIN and allows you to control the built-in LED easily.
Pin -- D13
If you want to light an external LED with this sketch, you need to build this circuit, where you connect one end of the resistor to the digital pin correspondent to the LED_BUILTIN constant. Connect the long leg of the LED (the positive leg, called the anode) to the other end of the resistor. Connect the short leg of the LED (the negative leg, called the cathode) to the GND. In the diagram below we show an UNO board that has D13 as the LED_BUILTIN value.

The resistor is essential for safe operation as it limits the current flowing through the LED, preventing damage to both the LED and the Arduino's output pin. You can choose the resistor value based on the desired current using Ohm's Law (V = IR) where V is the voltage of your board (5V or 3.3V) minus the forward voltage for the LED you are using (typical for red would be 1.8 to 2.2 volts). In this case, using a 220-ohm resistor with an Arduino UNO R3 (a 5V board) limits the current to a safe level for both the LED and the Arduino pin. Adjusting the resistor value allows you to control the LED's brightness while ensuring safe operation. For 5V boards you can expect the LED to be visible to a resistor value of up to 1K Ohm.

Connection:



Schematic:



Code Explanation:

After you build the circuit plug your Arduino board into your computer, start the Arduino Software (IDE) and enter the code below. You may also load it from the menu File/Examples/01.Basics/Blink . The first thing you do is to initialize LED_BUILTIN pin as an output pin with the line

pinMode(LED_BUILTIN, OUTPUT);

In the main loop, you turn the LED on with the line:

digitalWrite(LED_BUILTIN, HIGH);

This supplies 5 volts to the LED anode. That creates a voltage difference across the pins of the LED, and lights it up. Then you turn it off with the line:

digitalWrite(LED_BUILTIN, LOW);

That takes the LED_BUILTIN pin 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 board to do nothing for 1000 milliseconds, or one second. When you use the delay() command, nothing else happens for that amount of time. Once you've understood the basic examples, check out the BlinkWithoutDelay example to learn how to create a delay while doing other things.

Once you've understood this example, check out the DigitalReadSerial example to learn how read a switch connected to the board.

Code:

```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);  // change state of the LED by setting the pin
to the HIGH voltage level
  delay(1000);                      // wait for a second
  digitalWrite(LED_BUILTIN, LOW);   // change state of the LED by setting the pin
to the LOW voltage level
  delay(1000);                      // wait for a second
}
```


Chapter 2 - Analog Read Serial

Description:

Read a potentiometer, print its state out to the Arduino Serial Monitor.

This example shows you how to read analog input from the physical world using a potentiometer. A potentiometer is a simple mechanical device that provides a varying amount of resistance when its shaft is turned. By passing voltage through a potentiometer and into an analog input on your board, it is possible to measure the amount of resistance produced by a potentiometer (or pot for short) as an analog value. In this example you will monitor the state of your potentiometer after establishing serial communication between your Arduino and your computer running the Arduino Software (IDE).

Hardware Required

- Arduino Board
- 10k ohm Potentiometer

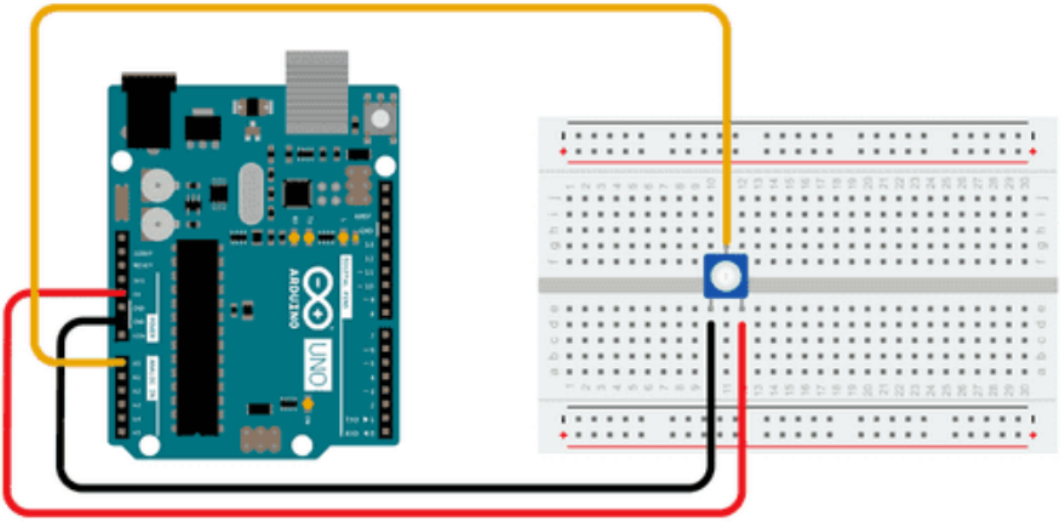
Circuit

Connect the three wires from the potentiometer to your board. The first goes from one of the outer pins of the potentiometer to ground. The second goes from the other outer pin of the potentiometer to 5 volts. The third goes from the middle pin of the potentiometer to the analog pin A0.

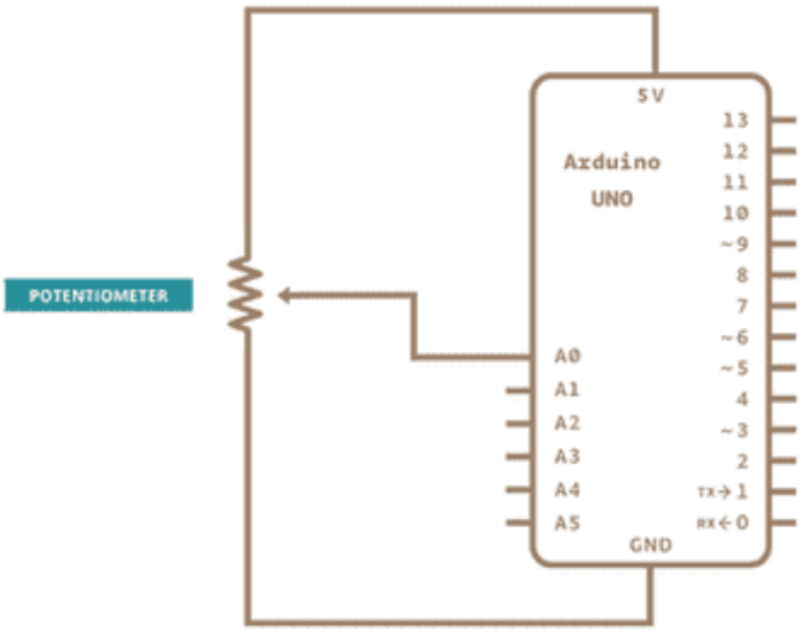
By turning the shaft of the potentiometer, you change the amount of resistance on either side of the wiper, which is connected to the center pin of the potentiometer. This changes the voltage at the center pin. When the resistance between the center and the side connected to 5 volts is close to zero (and the resistance on the other side is close to 10k ohm), 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 boards have a circuit inside called an analog-to-digital converter or ADC 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.

Connection:



Schematic:



Code Explanation:

In the sketch below, the only thing that you do in the setup function is to begin serial communications, at 9600 bits of data per second, between your board and your computer with the command:

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 intdatatype) coming in from your potentiometer:

int sensorValue = analogRead(A0);

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

Serial.println(sensorValue);

Now, when you open your Serial Monitor in the Arduino Software (IDE) (by clicking the icon that looks like a lens, on the right, in the green top bar or using the keyboard shortcut Ctrl+Shift+M), you should see a steady stream of numbers ranging from 0-1023, correlating to the position of the pot. As you turn your potentiometer, these numbers will respond almost instantly.

Code:

```
// 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);
  // print out the value you read:
  Serial.println(sensorValue);
  delay(1);  // delay in between reads for stability
}
```


Chapter 3 - Fading a LED

Description:

Demonstrates the use of analog output to fade an LED.

This example 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.

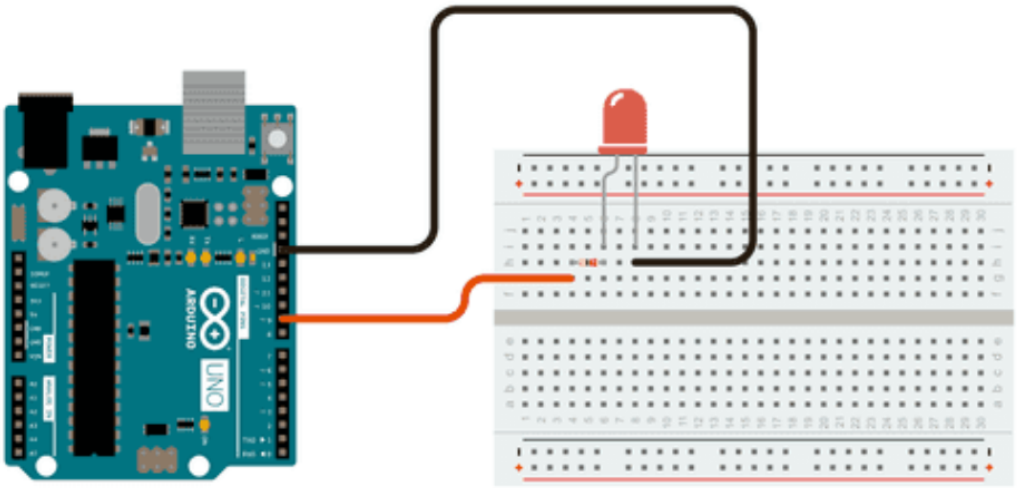
Hardware Required:

- Arduino Board
- LED
- 220 ohm resistor
- hook-up wires
- breadboard

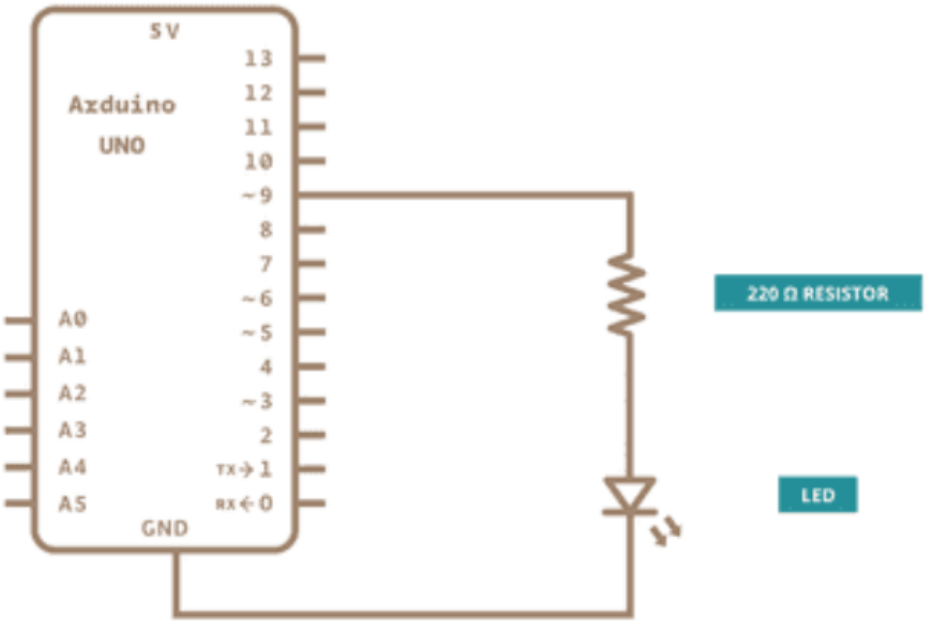
Circuit:

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

Connection:



Schematic:



Code Explanation:

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 -5, 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 fading effect.

Code:

```
int led = 9;        // the PWM pin 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);
}
```


Chapter 4 - Digital Read Serial

Description:

Read a switch, print the state out to the Arduino Serial Monitor.

This example shows you how to monitor the state of a switch by establishing serial communication between your Arduino and your computer over USB.

Hardware Required:

- Arduino Board
- A momentary switch, button, or toggle switch
- 10k ohm resistor
- hook-up wires/Jumper wires
- Breadboard

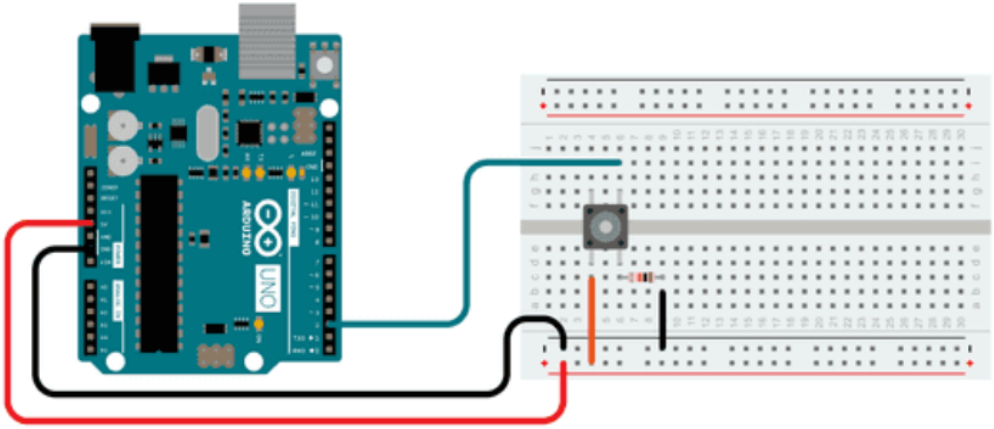
Circuit:

Connect three wires to the 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 10k ohm) to ground. The other leg of the button connects to the 5 volt supply.

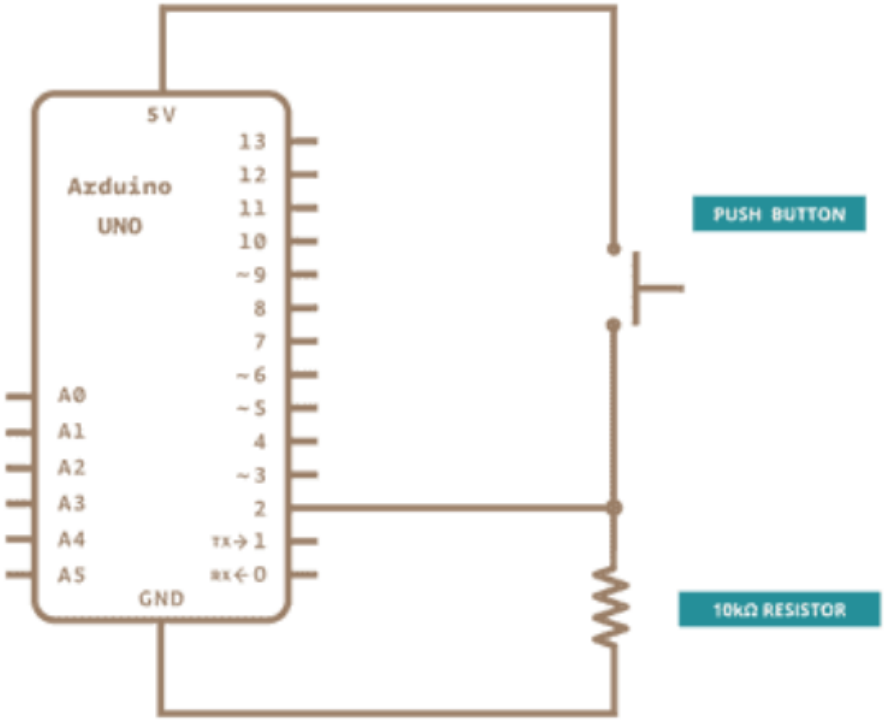
Pushbuttons or switches connect two points in a circuit when you press them. When the pushbutton is open (unpressed) there is no connection between the two legs of the pushbutton, so the pin is connected to ground (through the pull-down resistor) and reads as LOW, or 0. When the button is closed (pressed),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

Connection:



Schematic:



Code Explanation:

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 board 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 10k ohm 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 intdatatype. 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 board 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 Software (IDE), you will see a stream of "0"s if your switch is open, or "1"s if your switch is closed.

Code:

```
// 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
}
```


Chapter 5 - Ping Ultrasonic Range Finder

Description:

Detect objects with an ultrasonic range finder.

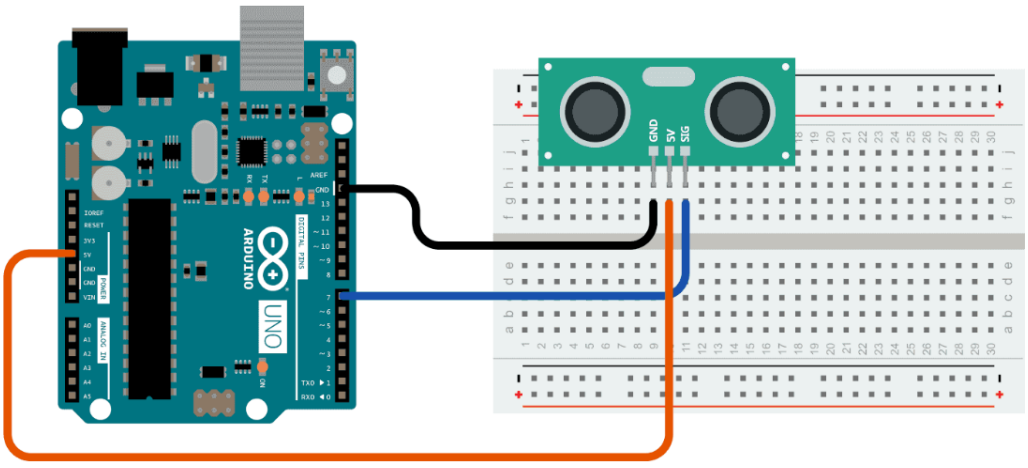
The HCSR04 is an ultrasonic range finder. It detects the distance of the closest object in front of the sensor (from 3 cm up to 400 cm). It works by sending out a burst of ultrasound and listening for the echo when it bounces off of an object. It pings the obstacles with ultrasound. The Arduino board sends a short pulse to trigger the detection, then listens for a pulse on the same pin using the pulseIn() function. The duration of this second pulse is equal to the time taken by the ultrasound to travel to the object and back to the sensor. Using the speed of sound, this time can be converted to distance.

Hardware Required

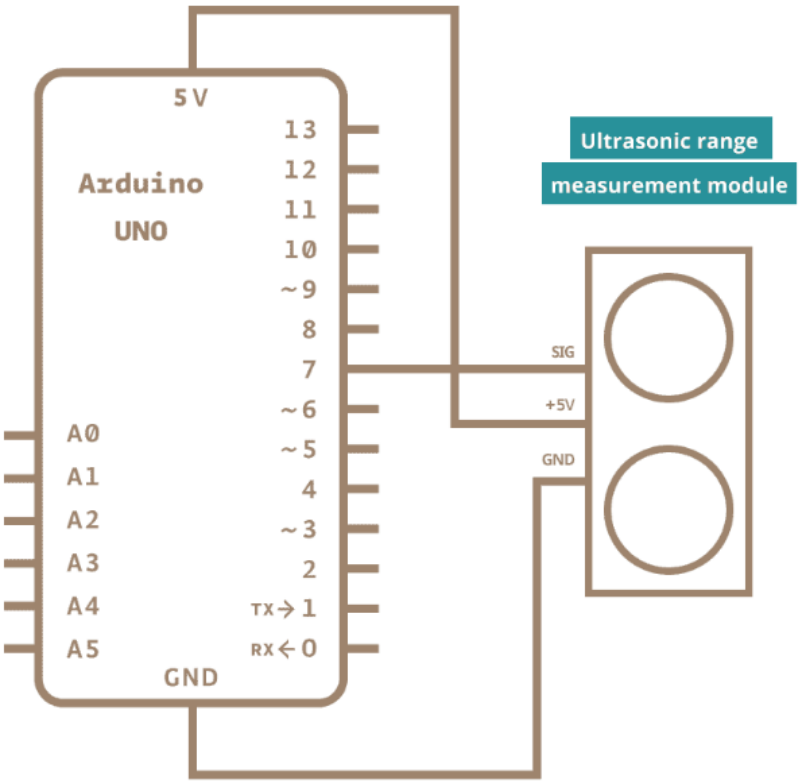
- Arduino Board
- Ultrasonic Range Finder
- hook-up wires

Circuit:

The 5V pin of the SEN136B5B is connected to the 5V pin on the board, the GND pin is connected to the GND pin, and the SIG (signal) pin is connected to digital pin 7 on the board.



Schematic:


5V
Arduino
UNO
13
12
11
10
~9
8
7
~6
~5
4
~3
2
TX→1
RX←0
A0
A1
A2
A3
A4
A5
GND
Ultrasonic range
measurement module
SIG
+5V
GND

Code1:

```
const int pingPin = 7;
void setup() {
  // initialize serial communication:
  Serial.begin(9600);
}
void loop() {
  // establish variables for duration of the ping, and the distance result
  // in inches and centimeters:
  long duration, inches, cm;
  // The PING))) is triggered by a HIGH pulse of 2 or more microseconds.
  // Give a short LOW pulse beforehand to ensure a clean HIGH pulse:
  pinMode(pingPin, OUTPUT);
  digitalWrite(pingPin, LOW);
  delayMicroseconds(2);
  digitalWrite(pingPin, HIGH);
  delayMicroseconds(5);
  digitalWrite(pingPin, LOW);
  // The same pin is used to read the signal from the PING))): a HIGH pulse
  // whose duration is the time (in microseconds) from the sending of the ping
  // to the reception of its echo off of an object.
  pinMode(pingPin, INPUT);
  duration = pulseIn(pingPin, HIGH);
  // convert the time into a distance
  inches = microsecondsToInches(duration);
  cm = microsecondsToCentimeters(duration);
  Serial.print(inches);
  Serial.print("in, ");
  Serial.print(cm);
  Serial.print("cm");
  Serial.println();
  delay(100);
}
long microsecondsToInches(long microseconds) {
  // According to Parallax's datasheet for the PING))), there are 73.746
  // microseconds per inch (i.e. sound travels at 1130 feet per second).
  // This gives the distance travelled by the ping, outbound and return,
  // so we divide by 2 to get the distance of the obstacle.
  // See: https://www.parallax.com/package/ping-ultrasonic-distance-sensor-downloads/
  return microseconds / 74 / 2;
}
long microsecondsToCentimeters(long microseconds) {
  // The speed of sound is 340 m/s or 29 microseconds per centimeter.
  // The ping travels out and back, so to find the distance of the object we
  // take half of the distance travelled.
  return microseconds / 29 / 2;}
```


Code2:

```
#include "Arduino.h"
class Ultrasonic
{
   public:
   Ultrasonic(int pin);
   void DistanceMeasure(void);
   double microsecondsToCentimeters(void);
   double microsecondsToInches(void);
   private:
   int this_pin;//pin number of Arduino that is connected with SIG pin of Ultrasonic Ranger.
   long duration;// the Pulse time received;
};
Ultrasonic::Ultrasonic(int pin)
{
   this_pin = pin;
}
/*Begin the detection and get the pulse back signal*/
void Ultrasonic::DistanceMeasure(void)
{
   pinMode(this_pin, OUTPUT);
   digitalWrite(this_pin, LOW);
   delayMicroseconds(2);
   digitalWrite(this_pin, HIGH);
   delayMicroseconds(5);
   digitalWrite(this_pin,LOW);
   pinMode(this_pin,INPUT);
   duration = pulseIn(this_pin,HIGH);
}
/*The measured distance from the range 0 to 400 Centimeters*/
double Ultrasonic::microsecondsToCentimeters(void)
{   return duration/29.0/2.0; }
/*The measured distance from the range 0 to 157 Inches*/
double Ultrasonic::microsecondsToInches(void)
{   return duration/74.0/2.0; }
Ultrasonic ultrasonic(2);
void setup()
{
   Serial.begin(9600);
}
void loop()
{
   double RangeInInches;
   double RangeInCentimeters;
   ultrasonic.DistanceMeasure();// get the current signal time;
   RangeInInches = ultrasonic.microsecondsToInches();//convert the time to inches;
   RangeInCentimeters = ultrasonic.microsecondsToCentimeters();//convert the time to centimeters
   Serial.println("The distance to obstacles in front is: ");
   Serial.print(RangeInInches);//0~157 inches
   Serial.println(" inch");
   Serial.print(RangeInCentimeters);//0~400cm
   Serial.println(" cm");
   delay(1000);
}
```


Chapter 6 - Arduino UNO Controlled Servo Gate with Ultrasonic Sensor and LED Indicators

Description:

The circuit in question is designed around an Arduino UNO microcontroller and includes a variety of peripheral components. The primary function of this circuit is to utilize an HC-SR04 Ultrasonic Sensor to detect the presence of an object within a certain distance threshold and to respond by controlling a Servomotor SG90 and two LEDs. The Arduino UNO controls the logic and processing of sensor data, drives the servomotor, and manages the state of the LEDs based on the proximity of an object.

Hardware Required:

- Arduino Uno
- LED 2 Pin Red
- Servomotor SG90
- HC-SR90 Ultrasonic Distance Sensor

Circuit:

Arduino UNO

- **5V**: Powers the HC-SR04 Ultrasonic Sensor and Servomotor SG90.
- **GND**: Common ground for the HC-SR04 Ultrasonic Sensor, Servomotor SG90, and both red LEDs.
- **D12**: Connected to the anode of one red LED.
- **D11**: Connected to the anode of the other red LED.
- **D10**: Connected to the ECHO pin of the HC-SR04 Ultrasonic Sensor.
- **D9**: Connected to the TRIG pin of the HC-SR04 Ultrasonic Sensor.
- **D6**: Connected to the SIG pin of the Servomotor SG90.

LED: Two Pin (red)

- **Anode**: Connected to either D12 or D11 on the Arduino UNO.
- **Cathode**: Connected to GND on the Arduino UNO.

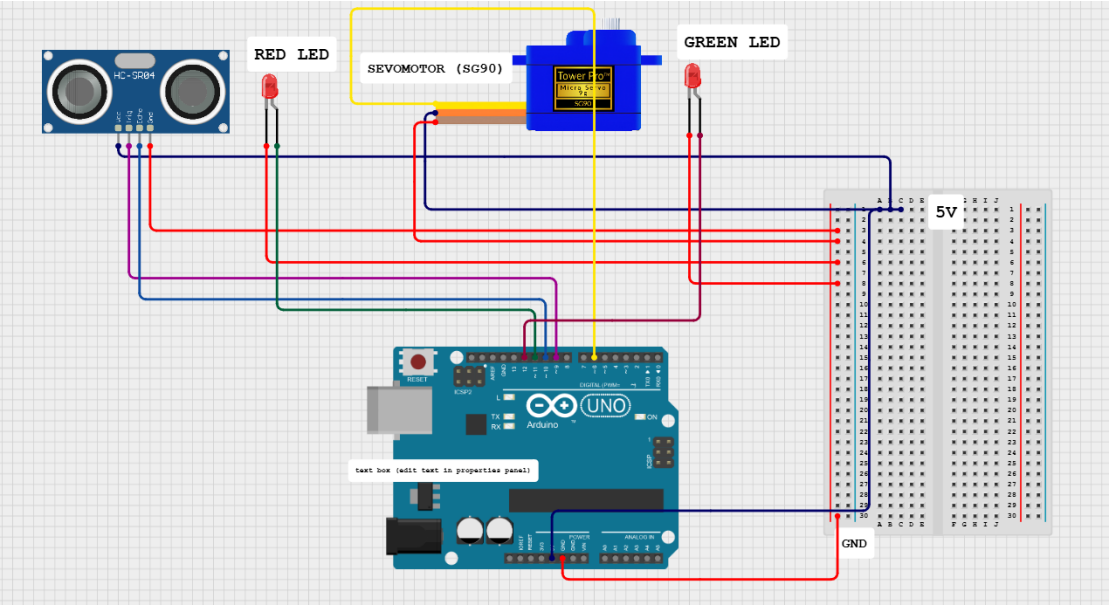
Servomotor SG90

- **SIG**: Connected to D6 on the Arduino UNO.
- **VCC**: Connected to 5V on the Arduino UNO.
- **GND**: Connected to GND on the Arduino UNO.

HC-SR04 Ultrasonic Sensor

- **VCC**: Connected to 5V on the Arduino UNO.
- **TRIG**: Connected to D9 on the Arduino UNO.
- **ECHO**: Connected to D10 on the Arduino UNO.
- **GND**: Connected to GND on the Arduino UNO.

Schematic:



Code:

```
#include <Servo.h>

const int trigPin = 9;
const int echoPin = 10;
const int redLEDPin = 12;
const int greenLEDPin = 11;
const int servoPin = 6;

Servo gateServo;

const int distanceThreshold = 10;

void setup() {
  Serial.begin(9600);

  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  pinMode(redLEDPin, OUTPUT);
  pinMode(greenLEDPin, OUTPUT);

  gateServo.attach(servoPin);

  digitalWrite(redLEDPin, HIGH);
  digitalWrite(greenLEDPin, LOW);
  gateServo.write(0);
}

void loop() {

  long duration, distance;
  digitalWrite(trigPin, LOW);
  delay(1000);
```

```
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  duration = pulseIn(echoPin, HIGH);
  distance = (duration / 2) / 29.1;

  if (distance < distanceThreshold) {
    digitalWrite(redLEDPin, LOW);
    digitalWrite(greenLEDPin, HIGH);
    gateServo.write(90);
  } else {
    digitalWrite(redLEDPin, HIGH);
    digitalWrite(greenLEDPin, LOW);
    gateServo.write(0);
  }

  delay(10);
}
```

Chapter 7 - Arduino UNO with DHT11 Temperature and Humidity Sensor

Description:

This circuit integrates a DHT11 Humidity and Temperature Sensor with an Arduino UNO microcontroller. The purpose of the circuit is to measure ambient temperature and humidity, and output the readings to a serial monitor. A 10k Ohm resistor is used as a pull-up for the data line of the DHT11 sensor.

Hardware Required:

DHT11 Humidity and Temperature Sensor

- **Description**: A sensor that measures ambient humidity and temperature.
- **Pins**: VDD, DATA, NULL, GND

Arduino UNO

- **Description**: A microcontroller board based on the ATmega328P.
- **Pins**: UNUSED, IOREF, Reset, 3.3V, 5V, GND, Vin, A0-A5, SCL, SDA, AREF, D0-D13

Resistor

- **Description**: A passive two-terminal electrical component that implements electrical resistance as a circuit element.
- **Value**: 10,000 Ohms (10k Ohm)

Circuit:

DHT11 Humidity and Temperature Sensor

- **VDD**: Connected to the 5V output of the Arduino UNO.
- **DATA**: Connected to digital pin D2 of the Arduino UNO through a 10k Ohm resistor.
- **GND**: Connected to the ground (GND) pin of the Arduino UNO.

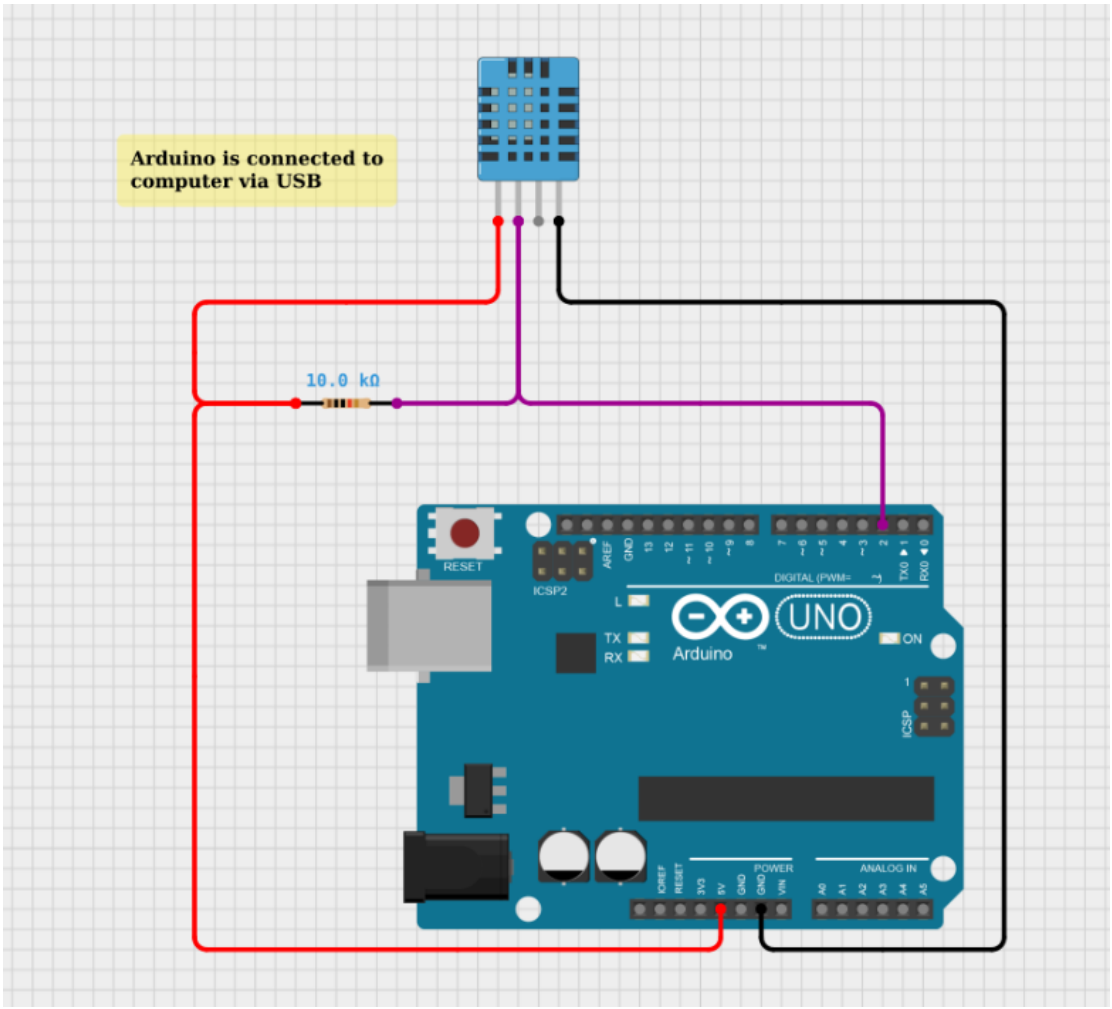
Arduino UNO

- **5V**: Provides power to the DHT11 sensor.
- **D2**: Receives data from the DHT11 sensor.
- **GND**: Common ground for the circuit.

Resistor (10k Ohm)

- **One end**: Connected to the DATA pin of the DHT11 sensor.
- **Other end**: Connected to digital pin D2 of the Arduino UNO.

Schematic:



Code:

```
#include <Adafruit_Sensor.h>
#include <DHT.h>
#include <DHT_U.h>

#define DHTPIN 2     // Digital pin connected to the DHT sensor

// Uncomment the type of sensor in use:
#define DHTTYPE    DHT11     // DHT 11

DHT_Unified dht(DHTPIN, DHTTYPE);

uint32_t delayMS;

void setup() {
  Serial.begin(9600);
  // Initialize device.
  dht.begin();
  Serial.println(F("DHTxx Unified Sensor Example"));
  // Print temperature sensor details.
  sensor_t sensor;
  dht.temperature().getSensor(&sensor);
```

```
  Serial.println(F("------------------------------------"));
  Serial.println(F("Temperature Sensor"));
  Serial.print  (F("Sensor Type: ")); Serial.println(sensor.name);
  Serial.print  (F("Driver Ver:  ")); Serial.println(sensor.version);
  Serial.print  (F("Unique ID:   ")); Serial.println(sensor.sensor_id);
  Serial.print  (F("Max Value:   ")); Serial.print(sensor.max_value); Serial.println(F("°C"));
  Serial.print  (F("Min Value:   ")); Serial.print(sensor.min_value); Serial.println(F("°C"));
  Serial.print  (F("Resolution:  ")); Serial.print(sensor.resolution); Serial.println(F("°C"));
  Serial.println(F("------------------------------------"));
  // Print humidity sensor details.
  dht.humidity().getSensor(&sensor);
  Serial.println(F("Humidity Sensor"));
  Serial.print  (F("Sensor Type: ")); Serial.println(sensor.name);
  Serial.print  (F("Driver Ver:  ")); Serial.println(sensor.version);
  Serial.print  (F("Unique ID:   ")); Serial.println(sensor.sensor_id);
  Serial.print  (F("Max Value:   ")); Serial.print(sensor.max_value); Serial.println(F("%"));
  Serial.print  (F("Min Value:   ")); Serial.print(sensor.min_value); Serial.println(F("%"));
  Serial.print  (F("Resolution:  ")); Serial.print(sensor.resolution); Serial.println(F("%"));
  Serial.println(F("------------------------------------"));
  // Set delay between sensor readings based on sensor details.
  delayMS = sensor.min_delay / 1000;
}

void loop() {
  // Delay between measurements.
  delay(delayMS);
  // Get temperature event and print its value.
  sensors_event_t event;
  dht.temperature().getEvent(&event);
  if (isnan(event.temperature)) {
    Serial.println(F("Error reading temperature!"));
  }
  else {
    Serial.print(F("Temperature: "));
    Serial.print(event.temperature);
    Serial.println(F("°C"));
  }
  // Get humidity event and print its value.
  dht.humidity().getEvent(&event);
  if (isnan(event.relative_humidity)) {
    Serial.println(F("Error reading humidity!"));
  }
  else {
    Serial.print(F("Humidity: "));
    Serial.print(event.relative_humidity);
    Serial.println(F("%"));
  }
}
```

Note: This code is designed to be uploaded to the Arduino UNO microcontroller. It requires the Adafruit DHT Sensor Library and the Adafruit Unified Sensor Library, which can be installed through the Arduino Library Manager. The serial monitor baud rate should be set to 9600 to match the `Serial.begin(9600);` configuration in the code.

Chapter 8 - Arduino-Based Temperature and Humidity Monitor with DHT11 and I2C LCD

Description: This circuit is designed to measure temperature and humidity using a DHT11 sensor and display the readings on a 16x2 I2C LCD. The circuit is controlled by an Arduino UNO, which reads data from the DHT11 sensor and updates the LCD display accordingly. A resistor is used to pull up the data line of the DHT11 sensor.

Hardware Required:

DHT11 Humidity and Temperature Sensor

- **Description**: Measures temperature and humidity.
- **Pins**: VDD, DATA, NULL, GND

Resistor

- **Description**: Provides a pull-up resistance for the DHT11 sensor data line.
- **Pins**: pin1, pin2
- **Properties**:
    - **Resistance**: 200 Ohms

Arduino UNO

- **Description**: Microcontroller board used to control the circuit.
- **Pins**: UNUSED, IOREF, Reset, 3.3V, 5V, GND, Vin, A0, A1, A2, A3, A4, A5, SCL, SDA, AREF, D13, D12, D11, D10, D9, D8, D7, D6, D5, D4, D3, D2, D1, D0

16x2 I2C LCD

- **Description**: Displays temperature and humidity readings.
- **Pins**: GND, VCC, SDA, SCL

Circuite connection:

DHT11 Humidity and Temperature Sensor

- **VDD**: Connected to 5V on Arduino UNO
- **DATA**: Connected to pin2 of the Resistor
- **GND**: Connected to GND on Arduino UNO

Resistor

- **pin1**: Connected to 5V on Arduino UNO
- **pin2**: Connected to DATA pin of DHT11 sensor and D9 on Arduino UNO

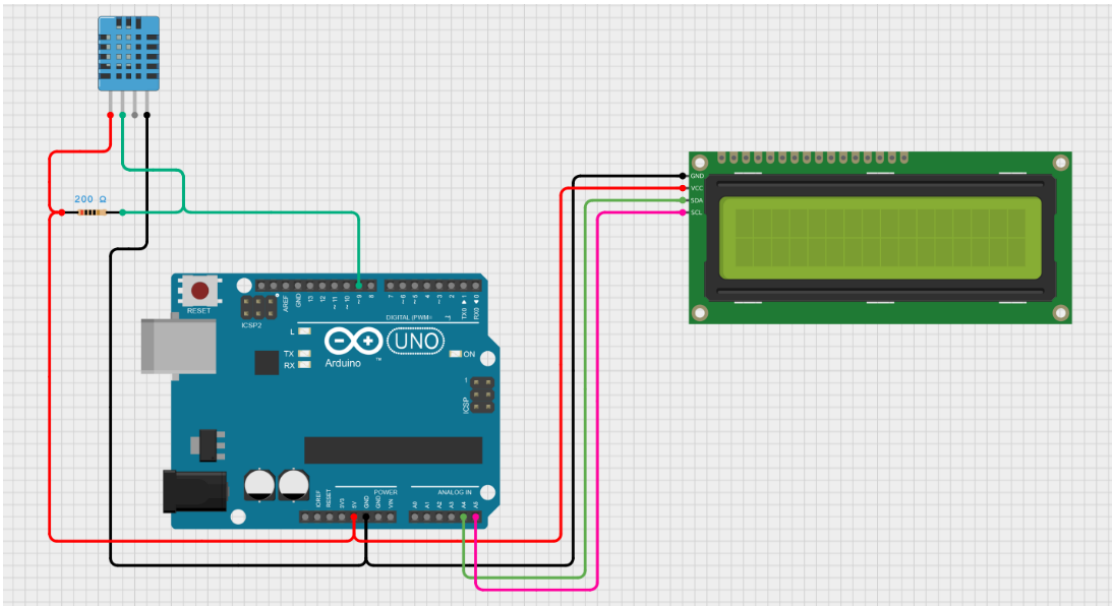
Arduino UNO

- **5V**: Connected to VDD of DHT11 sensor and pin1 of the Resistor
- **GND**: Connected to GND of DHT11 sensor and GND of 16x2 I2C LCD
- **A4**: Connected to SDA of 16x2 I2C LCD
- **A5**: Connected to SCL of 16x2 I2C LCD
- **D9**: Connected to pin2 of the Resistor

16x2 I2C LCD

- **GND**: Connected to GND on Arduino UNO
- **VCC**: Connected to 5V on Arduino UNO
- **SDA**: Connected to A4 on Arduino UNO
- **SCL**: Connected to A5 on Arduino UNO

Schematic:



Code:

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <DHT.h>

// Initialize the LCD with the I2C address (usually 0x27 or 0x3F)
LiquidCrystal_I2C lcd(0x27, 16, 2);

// Define the DHT sensor pin and type
#define DHTPIN 9
#define DHTTYPE DHT11

// Initialize the DHT sensor
DHT dht(DHTPIN, DHTTYPE);
```

```
unsigned long startTime;
float previousTemperature = NAN;

void setup() {
  // Initialize the LCD
  lcd.begin(16, 2);
  lcd.backlight();

  // Print a message to the LCD
  lcd.print("Initializing...");

  delay(500);

  // Initialize the start time
  startTime = millis();

  // Initialize the DHT sensor
  dht.begin();

  pinMode(6, OUTPUT);
  pinMode(7, OUTPUT);

  digitalWrite(6, LOW);
  digitalWrite(7, HIGH);
}

void loop() {
  // Calculate the elapsed time in milliseconds
  unsigned long elapsedTime = millis() - startTime;

  // Convert elapsed time to seconds with one decimal place
  float elapsedTimeSec = elapsedTime / 1000.0;

  // Read temperature as Celsius
  float temperature = dht.readTemperature();

  // Check if any reads failed and exit early (to try again).
  if (isnan(temperature)) {
    lcd.setCursor(0, 0);
    lcd.print("Error reading");
    return;
  }

  // Only update the display if the temperature has changed
  if (temperature != previousTemperature) {
    // Clear the first row
    lcd.setCursor(0, 0);
    lcd.print("                "); // Clear the first row by printing spaces
```



```
    // Set the cursor to the first row, first column
    lcd.setCursor(0, 0);
    lcd.print("Temp: ");

    // Clear the second row
    lcd.setCursor(0, 1);
    lcd.print("                "); // Clear the second row by printing spaces

    // Set the cursor to the second row, first column
    lcd.setCursor(0, 1);

    // Print the temperature to the LCD
    lcd.print(temperature);
    lcd.print(" C");

    // Update the previous temperature
    previousTemperature = temperature;
  }

  // Add a small delay to avoid flickering
  delay(50);
}
```

Note: This code initializes the DHT11 sensor and the 16x2 I2C LCD. It reads the temperature from the DHT11 sensor and displays it on the LCD. The display is updated only when the temperature changes to avoid unnecessary flickering.

Chapter 9 - Arduino UNO LED Control with Resistor

Description: This circuit consists of an Arduino UNO microcontroller, a red LED, and a 220 Ohm resistor. The LED is connected to the Arduino UNO through the resistor, allowing the microcontroller to control the LED.

Hardware Required:

Arduino UNO

- **Description:** A microcontroller board based on the ATmega328P.
- **Pins:** D11
- **Purpose in Circuit:** Acts as the main controller for the circuit.

LED: Two Pin (red)

- **Description:** A red light-emitting diode.
- **Pins:** cathode, anode
- **Purpose in Circuit:** Provides visual feedback by emitting light when powered.

Resistor

- **Description:** A 220 Ohm resistor.
- **Pins:** pin1, pin2
- **Properties:**
  - **Resistance:** 220 Ohms
- **Purpose in Circuit:** Limits the current flowing through the LED to prevent damage.

Circuites:

Arduino UNO

- **GND:** Connected to pin1 of the Resistor.
- **D11:** Connected to the anode of the LED.

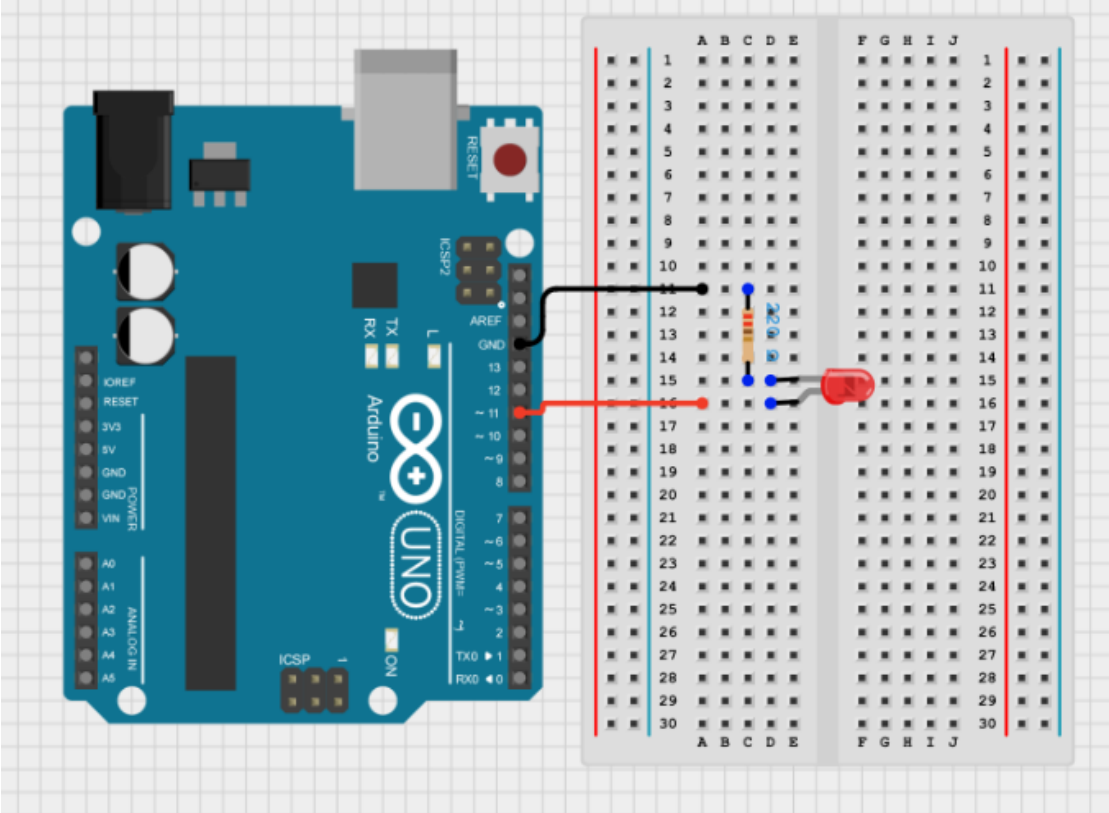
LED: Two Pin (red)

- **anode:** Connected to D11 of the Arduino UNO.
- **cathode:** Connected to pin2 of the Resistor.

Resistor

- **pin1:** Connected to GND of the Arduino UNO.
- **pin2:** Connected to the cathode of the LED.

Schematic::


RESET
ICSP2
AREF
GND
TX
RX
L
IOREF
RESET
3V3
5V
GND
GND
VIN
POWER
A0
A1
A2
A3
A4
A5
ANALOG IN
Arduino
UNO
ICSP
ON
DIGITAL (PWM~)
TX0 1
RX0 0
A B C D E
F G H I J
220

Code:

Chapter 10 - Battery-Powered Light-Activated Alarm with BC547 Transistor and Photocell

Description: This document provides a detailed overview of a simple electronic circuit that includes a 9V battery, a BC547 transistor, a photocell (LDR), a resistor, a buzzer, and a red LED. The circuit is designed to activate the buzzer and LED based on the light intensity detected by the photocell.

Hardware required:

9V Battery

1. **Description:** Provides the power supply for the circuit.
2. **Pins:** +, -

BC547 Transistor

1. **Description:** A general-purpose NPN transistor used for switching and amplification.
2. **Pins:** Collector, Base, Emitter

Photocell (LDR)

1. **Description:** A light-dependent resistor that changes its resistance based on the light intensity.
2. **Pins:** pin 0, pin 1

Resistor

1. **Description:** Limits the current flow in the circuit.
2. **Pins:** pin1, pin2
3. **Properties:**
    1. **Resistance:** 10,000 Ohms

Buzzer

1. **Description:** Produces sound when activated.
2. **Pins:** PIN, GND

LED: Two Pin (red)

1. **Description:** Emits red light when current flows through it.
2. **Pins:** cathode, anode
3.

Circuite Connection:

9V Battery

+ Pin:

- Connected to the Collector of the BC547 Transistor
- Connected to pin 0 of the Photocell (LDR)
- Connected to the PIN of the Buzzer
- Connected to pin1 of the Resistor
- Connected to the cathode of the LED

- Pin:

- Connected to the Emitter of the BC547 Transistor
- Connected to pin 1 of the Photocell (LDR)
- Connected to the GND of the Buzzer
- Connected to pin2 of the Resistor
- Connected to the anode of the LED

BC547 Transistor

Collector:

- Connected to the + pin of the 9V Battery
- Connected to pin 0 of the Photocell (LDR)
- Connected to the PIN of the Buzzer
- Connected to pin1 of the Resistor
- Connected to the cathode of the LED

Emitter:

- Connected to the - pin of the 9V Battery
- Connected to pin 1 of the Photocell (LDR)
- Connected to the GND of the Buzzer
- Connected to pin2 of the Resistor
- Connected to the anode of the LED

Photocell (LDR)

pin 0:

- Connected to the + pin of the 9V Battery
- Connected to the Collector of the BC547 Transistor
- Connected to the PIN of the Buzzer
- Connected to pin1 of the Resistor
- Connected to the cathode of the LED

pin 1:

- Connected to the - pin of the 9V Battery
- Connected to the Emitter of the BC547 Transistor
- Connected to the GND of the Buzzer
- Connected to pin2 of the Resistor
- Connected to the anode of the LED

Resistor

pin1:

- Connected to the + pin of the 9V Battery
- Connected to the Collector of the BC547 Transistor
- Connected to pin 0 of the Photocell (LDR)
- Connected to the PIN of the Buzzer
- Connected to the cathode of the LED

pin2:

- Connected to the - pin of the 9V Battery
- Connected to the Emitter of the BC547 Transistor
- Connected to pin 1 of the Photocell (LDR)
- Connected to the GND of the Buzzer
- Connected to the anode of the LED

Buzzer

PIN

- Connected to the + pin of the 9V Battery
- Connected to the Collector of the BC547 Transistor
- Connected to pin 0 of the Photocell (LDR)
- Connected to pin1 of the Resistor
- Connected to the cathode of the LED

GND:

- Connected to the - pin of the 9V Battery
- Connected to the Emitter of the BC547 Transistor
- Connected to pin 1 of the Photocell (LDR)
- Connected to pin2 of the Resistor
- Connected to the anode of the LED

LED: Two Pin (red)

cathode:

- Connected to the + pin of the 9V Battery
- Connected to the Collector of the BC547 Transistor
- Connected to pin 0 of the Photocell (LDR)

- Connected to the PIN of the Buzzer
- Connected to pin1 of the Resistor

Anode:

- Connected to the - pin of the 9V Battery
- Connected to the Emitter of the BC547 Transistor
- Connected to pin 1 of the Photocell (LDR)
- Connected to the GND of the Buzzer
- Connected to pin2 of the Resistor

Schematic: