Vellore Institute of Technology, Chennai

School of Computer Science and Engineering

Internet of Things Laboratory – BECE351E

LAB MANUAL

ACADEMIC YEAR 2023-2024

5TH SEMESTER

FALL INTER SEMESTER

Submitted by

Student Name : Mainak Chattopadhyay

Roll. No: 21BAI1217

Faculty In-Charge

Dr. M Manigandan

Assistant Professor

School of Electronics and Engineering

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

CERTIFICATE

Name: Mainak Chattopadhyay **Class: CH2022232500452**

Roll.No: 21BAI1217 **Lab: L13+L14**

This is certified to be the bonafide work of the student in the <u>Internet of Things</u> Laboratory during the Academic Year 2023-2024, 5th Semester FALL- INTER SEMESTER.

Faculty In-Charge

Dr. M Manigandan

Assistant Professor
SENSE

VIT

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

INDEX

Experiment No	Title	Date	Page No
1A	Two LED Blinking interfacing with Arduino Uno R3 using Simulation TinkerCAD.	May 3, 2023	4
1B	Gas Sensor Detector with Arduino UNO in Tinker CAD	May 3, 2023	9
2	Soil Mositure Monitoring with Buzzer Using Arduino Uno–TinkerCad and Real Time Implementation	May 10, 2023	14
3	Environmental monitoring Arduino Uno with DHT 11 Sensor using Node-Red	May 17, 2023	21
4	Smart Street Light using TinkerCad and Arduino	May 17, 2023	27
5	SMART Car Parking System using TinkerCad and Arduino	May 24, 2023	36
6	SMART Traffic Light Control System using TinkerCAD and Arduino	May 31, 2023	42
7	Environmental monitoring using ESP32 with DHT22 and ThingSpeak	June 7, 2023	55
8	Smart home automation using Node MCU (ESP32) with Relays and ThingSpeak Cloud Computing (Wokwi and ThingSpeak Simulation)	June 14, 2023	64
9	Time Environmental monitoring using Arduino Uno Wi-Fi Rev 2 with DHT11 and ThingSpeak Cloud Computing (Hardware and Software)	June 21, 2023	77
10	Real-Time Soil moisture monitoring using Arduino Uno WiFi Rev 2 and ThingSpeak Cloud Computing (Hardware and Software)	June 28, 2023	85

BECE351E – INTERNET OF THINGS

EXPT NO: 1

DATE: 26/04/23

NAME: Mainak Chattopadhyay

REG. NO.: 21BAI1217

Gas Sensor Detector with Arduino UNO in Tinker CAD

AIM

To perform and analyze the working of the gas sensor detector with Arduino UNO in Tinker CAD.

BLOCKS/COMPONENTS REQUIRED

Tinker CAD Block Modules

1. Arduino UNO
2. Gas Sensor

ALGORITHM

Step-1: Open TinkerCAD.

Step-2: In the Design Suite, Drag and drop the required blocks and its associated components into their respective places.

Step-3: Connect as per the circuit diagram given in the block diagram. Follow the image and place each part as shown. Notice that the breadboard is used to connect the components together. It contains a series of columns and two rails on each side. As you click from pin to pin, you'll create wires, each of which can be color-coded for easy identification.

Step-4: In code section, appropriate Arduino codes are fed. Create the next block, an "if" statement, which is a type of control block that makes a decision.

Step-5: Once you're done, select "Start Simulation" on the toolbar to turn on your Arduino Uno.

Step-6: Troubleshooting: If your program doesn't behave as expected, check your wiring and programming. Ensure that all pins are properly connected and that each block is written correctly.

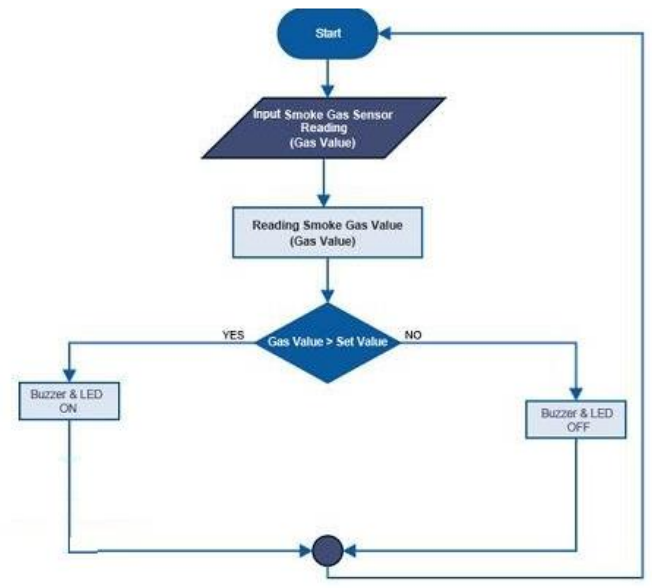
THEORY ABOUT GAS SENSOR DETECTOR

The gas sensor has a built-in potentiometer that allows you to adjust the sensor digital output (D0) threshold. This threshold sets the value above which the digital pin will output a HIGH signal. The voltage that the sensor outputs changes accordingly to the smoke/gas level that exists in the atmosphere. The sensor outputs a voltage that is proportional to the concentration of smoke/gas. In other words, the relationship between voltage and gas concentration is the following:

- The greater the gas concentration, the greater the output voltage
- The lower the gas concentration, the lower the output voltage

The output can be an analog signal (A0) that can be read with an analog input of the Arduino or a digital output (D0) that can be read with a digital input of the Arduino.

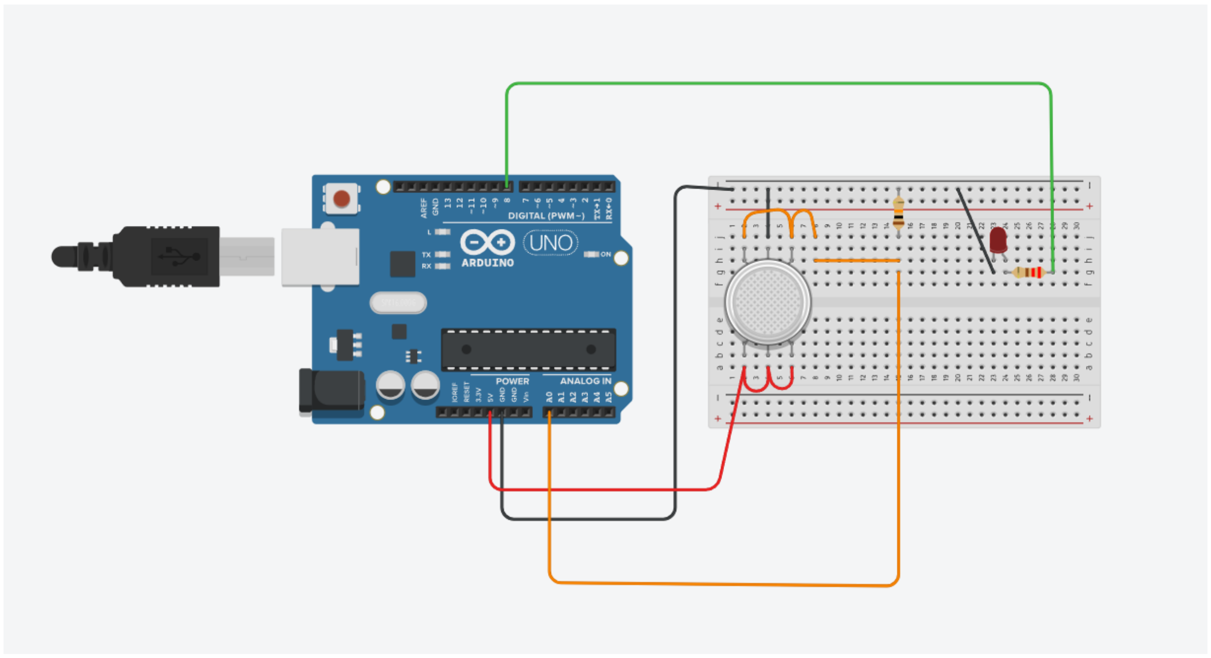
FLOW CHART


Start
Input Smoke Gas Sensor Reading (Gas Value)
Reading Smoke Gas Value (Gas Value)
Gas Value > Set Value
YES
NO
Buzzer & LED ON
Buzzer & LED OFF

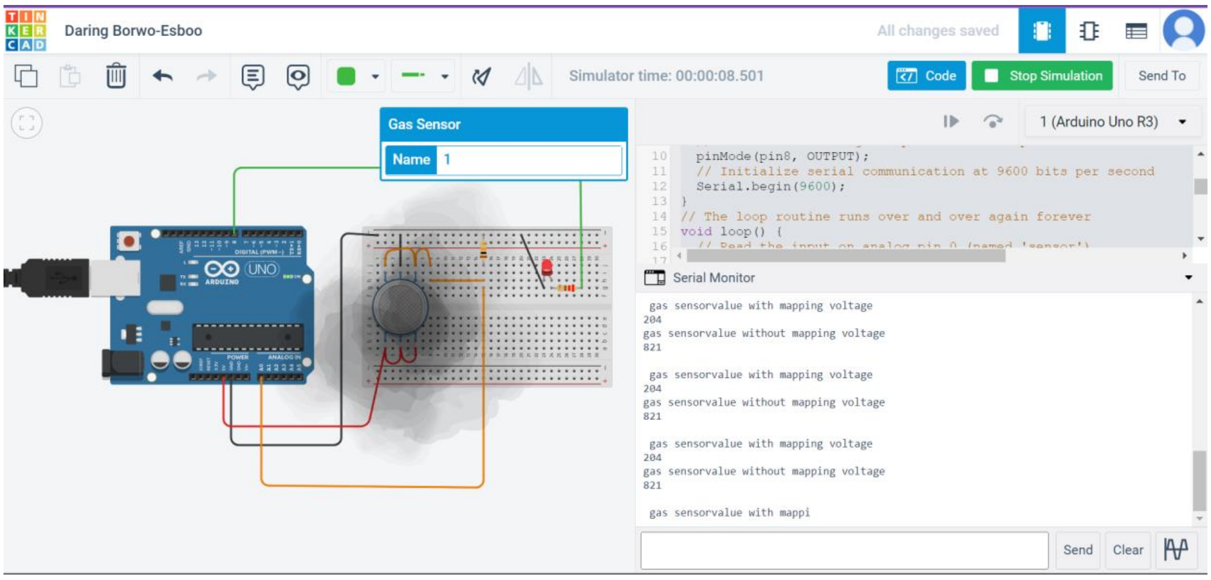
ARDUINO PROGRAM

```
// Digital pin 8 will be called 'pin8'

int pin8 = 8;

// Analog pin 0 will be called 'sensor'

int sensor = A0;

// Set the initial sensorValue to 0

int sensorValue = 0;

// The setup routine runs once when you press reset

void setup() {

  // Initialize the digital pin 8 as an output

  pinMode(pin8, OUTPUT);

  // 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 (named 'sensor')

  sensorValue = analogRead(sensor);

  // Print out the value you read

  unsigned int outputValue = map(sensorValue, 0, 1023, 0, 255); // map the 10-bit data to 8-bit data

  Serial.println("gas sensorvalue without mapping voltage");

  Serial.println(sensorValue);

  Serial.println("\n gas sensorvalue with mapping voltage");

  Serial.println(outputValue);

    // If sensorValue is greater than 500

  if (outputValue > 170) {

    // Activate digital output pin 8 - the LED will light up

    digitalWrite(pin8, HIGH);
  }
  else {
    // Deactivate digital output pin 8 - the LED will not light up
    digitalWrite(pin8, LOW);
  }
}
```

<u>SNAP SHOT OF THE CIRCUIT</u>



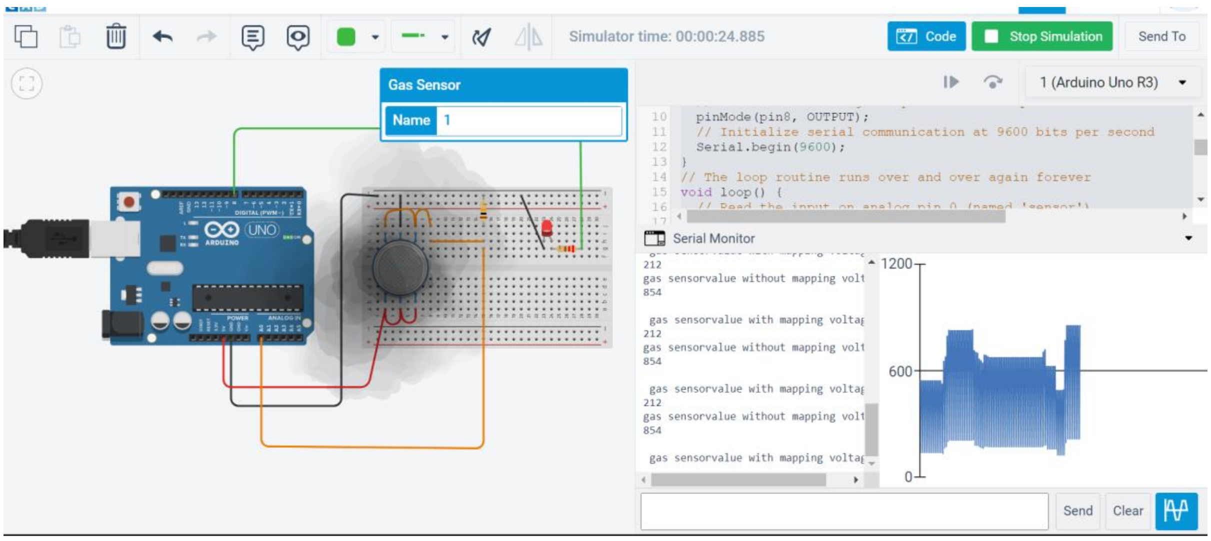
SNAP SHOT OF THE OUTPUT



```
  pinMode(pin8, OUTPUT);
  // 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 (named 'sensor')
```

```
 gas sensorvalue with mapping voltage
204
gas sensorvalue without mapping voltage
821

 gas sensorvalue with mapping voltage
204
gas sensorvalue without mapping voltage
821

 gas sensorvalue with mapping voltage
204
gas sensorvalue without mapping voltage
821

 gas sensorvalue with mappi
```

Changing the distance of smoke-



```
  pinMode(pin8, OUTPUT);
  // 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 (named 'sensor')
```

```
212
gas sensorvalue without mapping volt
854

 gas sensorvalue with mapping voltag
212
gas sensorvalue without mapping volt
854

 gas sensorvalue with mapping voltag
212
gas sensorvalue without mapping volt
854

 gas sensorvalue with mapping voltag
```

RESULT

The working of the gas sensor detector with Arduino UNO in Tinker CAD is analyzed and the variations in the gas detection with respect to distance is observed.

BECE351E – INTERNET OF THINGS

EXPT NO: 2

DATE: 10/05/23

NAME: Mainak Chattopadhyay

REG. NO.: 21BAI1217

SOIL MOSITURE MONITORING WITH BUZZER USING ARDUINO UNO– Tinker CAD and

AIM

To perform and analyze the working of the soil moisture monitoring

(i) Using sensor with Buzzer and LED a breadboard connected with Arduino UNO in Tinker CAD.
(ii) Using Arduino Uno Hardware

BLOCKS/COMPONENTS REQUIRED

(i) Software:

Tinker CAD Block Modules

1. Arduino UNO
2. Soil Moisture Sensor

(ii) Hardware:

1. Arduino UNO
2. Soil Moisture Sensor
3. Sensor Interface

THEORY ABOUT SOIL MOISTURE SENSOR

Soil moisture is basically the content of water present in the soil. This can be measured using a soil moisture sensor which consists of two conducting probes that act as a probe. The fork-shaped probe with two exposed conductors acts as a variable resistor (similar to a potentiometer) whose resistance varies with the soil's moisture content. This resistance varies inversely with soil moisture:

The more water in the soil, the better the conductivity and the lower the resistance.

The less water in the soil, the lower the conductivity and thus the higher the resistance.

The sensor produces an output voltage according to the resistance, which by measuring we can determine the soil moisture level. It can measure the moisture content in the soil based on the change in resistance between the two conducting plates. The resistance between the two conducting plates varies in an inverse manner with the amount of moisture present in the soil.

(i) USING TINKER CAD:

ALGORITHM

Step-1: Open TinkerCAD

Step-2: In the Design Suite, Drag and drop the required blocks and its associated components into their respective places.

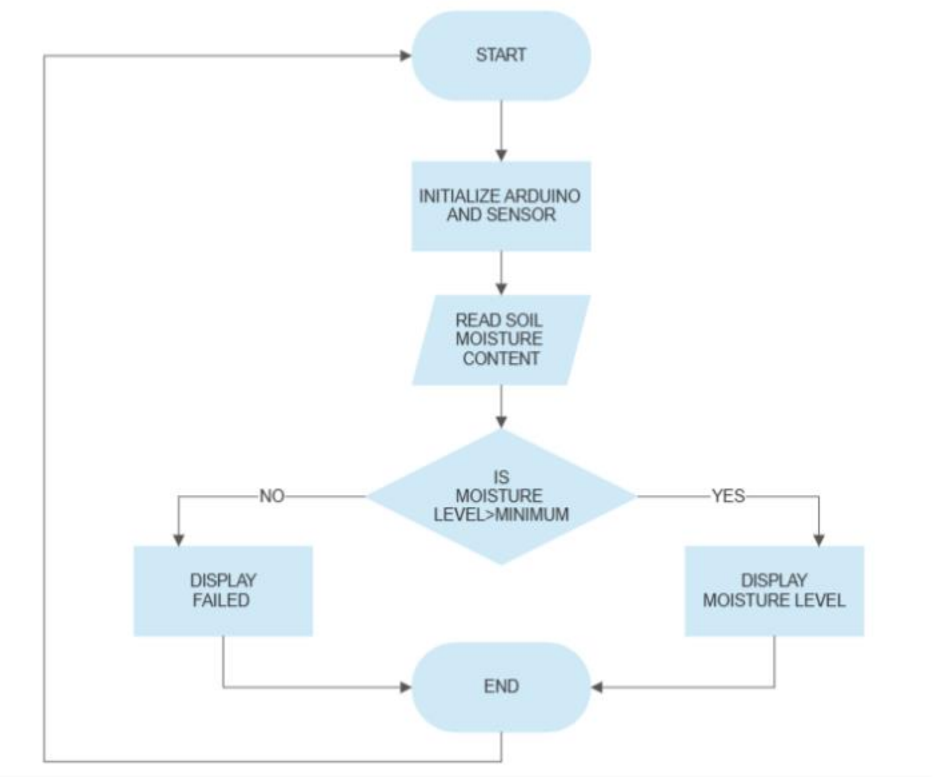
Step-3: Connect as per the circuit diagram given in the block diagram. Follow the image and place each part as shown. Notice that the breadboard is used to connect the components together. It contains a series of columns and two rails on each side. As you click from pin to pin, you'll create wires, each of which can be color-coded for easy identification.

Step-4: In code section, appropriate Arduino codes are fed. Create the next block, an "if" statement, which is a type of control block that makes a decision.

Step-5: Once you're done, select "Start Simulation" on the toolbar to turn on your Arduino Uno.

Step-6: Troubleshooting: If your program doesn't behave as expected, check your wiring and programming. Ensure that all pins are properly connected and that each block is written correctly.

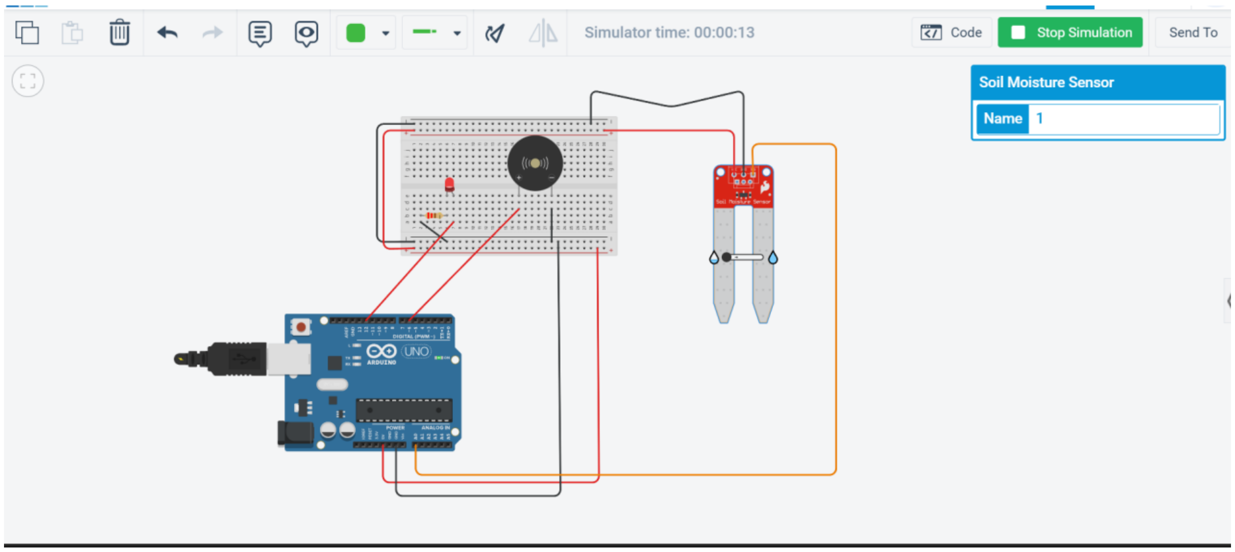
FLOW CHART

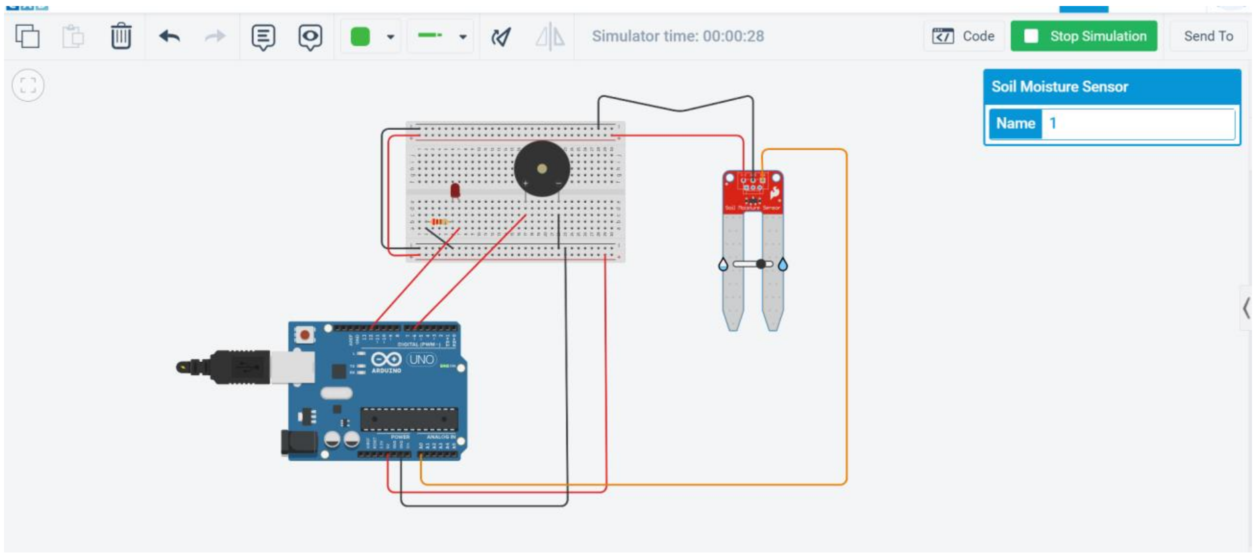

START
INITIALIZE ARDUINO AND SENSOR
READ SOIL MOISTURE CONTENT
IS MOISTURE LEVEL>MINIMUM
NO
YES
DISPLAY FAILED
DISPLAY MOISTURE LEVEL
END

ARDUINO PROGRAM

```
// C++ code

//

int moisture_data = 0;

void setup()

{

  pinMode(A0, OUTPUT);

  Serial.begin(9600);

  pinMode(12, OUTPUT);

  pinMode(6, OUTPUT);
}


void loop()
{
  moisture_data = analogRead(A0);
  Serial.println(moisture_data);
  if(moisture_data <21){
            digitalWrite(12,HIGH);
    digitalWrite(6,HIGH);
  }
  else{
            digitalWrite(12,LOW);
    digitalWrite(6,LOW);
  }
  delay(10);
}
```

<u>SNAP SHOT OF THE OUTPUT</u>



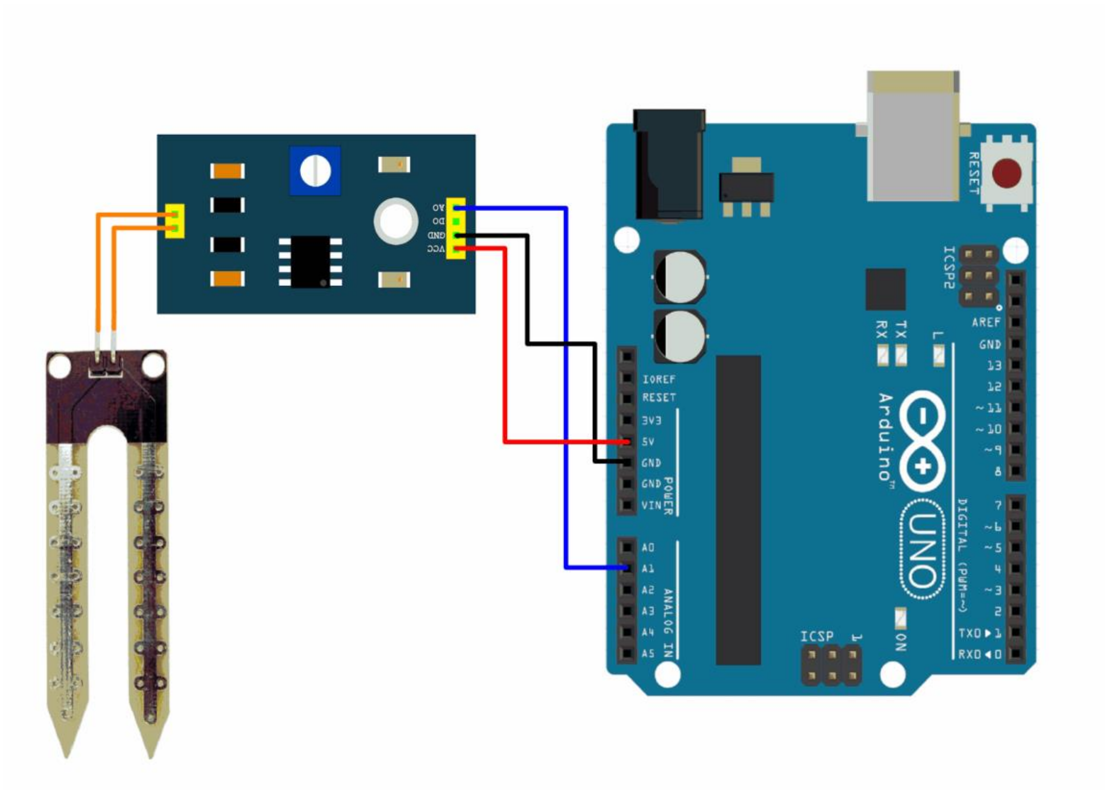


INFERENCE

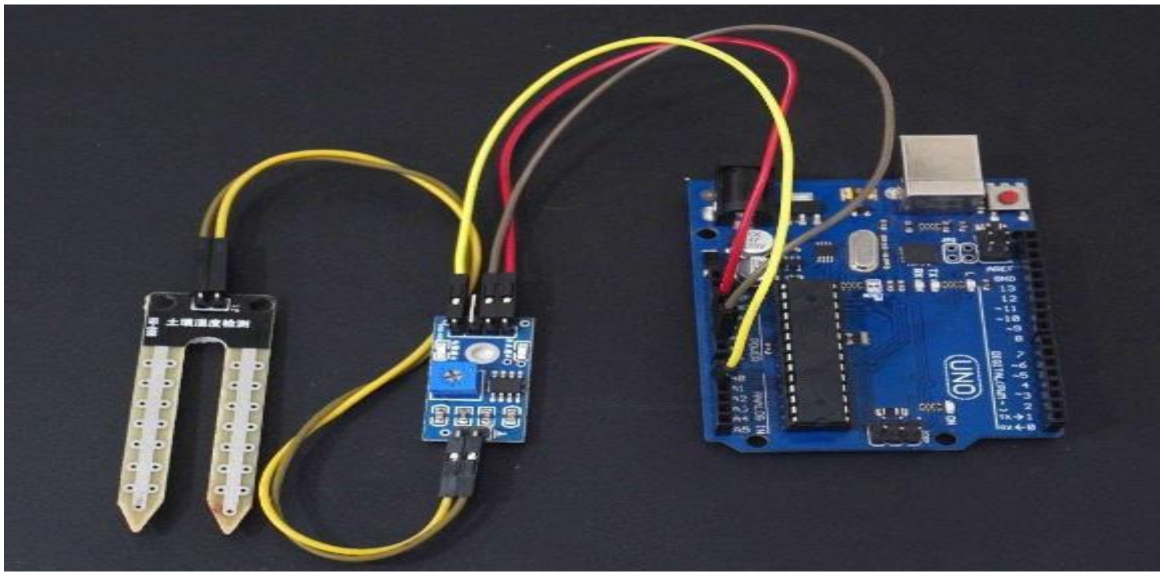
I got to know how soil sensor works and how the circuitry works for the same in simulation

(ii) USING ARDUINO UNO HARWARE:

HARDWARE CIRCUIT BLOCK DIAGRAM



HARDWARE CONNECTIONS



MEASUREMENT OF SOIL MOISTURE USING ARDUINO UNO

Analog output of soil moisture sensor is processed using ADC.

The moisture content in terms of percentage is displayed on the serial monitor.

The output of the soil moisture sensor changes in the range of ADC value from 0 to 1023.

This can be represented as moisture value in terms of percentage using formula given below.

$$AnalogOutput = \frac{ADCValue}{1023} \quad (1)$$

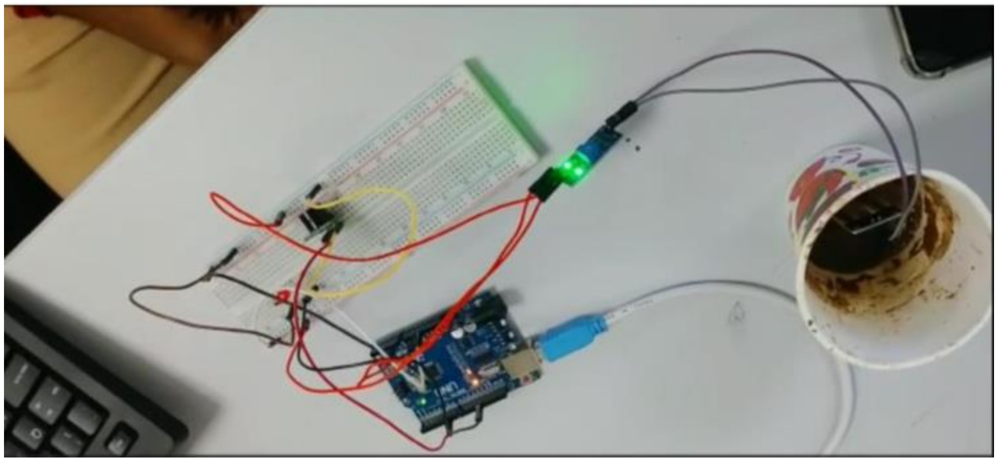
Moisture in percentage = 100 – (Analog output * 100)

For zero moisture, we get maximum value of 10-bit ADC, i.e. 1023. This, in turn, gives 0% moisture.

ARDUINO PROGRAM

```
const int sensor_pin = A1;      /* Soil moisture sensor O/P pin */

void setup() {

  Serial.begin(9600);   /* Define baud rate for serial communication */

}

void loop() {

  float moisture_percentage;

  int sensor_analog;

  sensor_analog = analogRead(sensor_pin);

  moisture_percentage = ( 100 - ( (sensor_analog/1023.00) * 100 ) );

  Serial.print("Moisture Percentage = ");

  Serial.print(moisture_percentage);

  Serial.print("%\n\n");
  delay(1000);
}
```

SNAP SHOT OF THE OUTPUT



INFERENCE

I got to know how soil sensor works and how the circuitry works for the same in hardware experimentation.

RESULT

The working of the soil moisture monitoring is performed

(i) using sensor with Buzzer and LED a breadboard connected with Arduino UNO in Tinker CAD is analysed and the variations in the moisture content is observed.
(ii) Using the sensor and Arduino Board and the variations in the moisture content is observed.

BECE351E – INTERNET OF THINGS

EXPT NO: 3

DATE: 17/05/23

NAME: Mainak Chattopadhyay

REG. NO.: 21BAI1217

Environmental monitoring Arduino Uno with DHT 11 Sensor using Node-Red

AIM

The aim of the experiment is to create an environmental monitoring system using an Arduino Uno and DHT11 sensor, combined with Node-RED. The system will measure temperature and humidity levels and display the data in real-time on a Node-RED dashboard. The objective is to monitor and visualize environmental conditions, which can be useful for various applications such as home automation, greenhouse monitoring, or general environmental sensing.

BLOCKS/COMPONENTS REQUIRED

(i) Software:

Node-RED

(ii) Hardware:

1. Arduino UNO
2. DHT11 Sensor
3. Sensor Interface

THEORY ABOUT DHT11 SENSOR

The DHT11 measures *relative humidity*. Relative humidity is the amount of water vapor in air vs. the saturation point of water vapor in air. At the saturation point, water vapor starts to condense and accumulate on surfaces forming dew. The saturation point changes with air temperature. Cold air can hold less water vapor before it becomes saturated, and hot air can hold more water vapor before it becomes saturated.
The formula to calculate relative humidity is:

$$RH = (\frac{\rho_w}{\rho_s})\ x\ 100\%$$

$RH :\ Relative\ Humidity$
$\rho_w :\ Density\ of\ water\ vapor$
$\rho_s :\ Density\ of\ water\ vapor\ at\ saturation$

Ranges and accuracy of the **DHT11:**
•Humidity Range: 20-90% RH
•Humidity Accuracy: ±5% RH
•Temperature Range: 0-50 °C
•Temperature Accuracy: ±2% °C
•Operating Voltage: 3V to 5.5V

Relative humidity is expressed as a percentage.
At 100% RH, condensation occurs, and at 0% RH, the air is completely dry.

ALGORITHM

Set up the Arduino Uno and DHT11 sensor:

Connect the DHT11 sensor to the Arduino Uno. The sensor has three pins: VCC, data, and GND. Connect VCC to 5V, data to a digital pin (e.g., pin 2), and GND to GND on the Arduino Uno.
Upload the DHT11 sensor library to your Arduino Uno. You can find the library and installation instructions in the Arduino Library Manager.
Write a sketch that reads data from the DHT11 sensor and sends it to the Arduino serial port

Set up Node-RED:

Install Node-RED on your computer or a Raspberry Pi. You can find installation instructions on the Node-RED website.
Launch the Node-RED flow editor by accessing the web interface.
Install the necessary Node-RED nodes. In this case, you'll need the node-red-node-serialport node for serial communication and the node-red-dashboard node for creating a user interface.
Restart Node-RED to ensure the installed nodes are loaded.

Create the Node-RED flow:

Drag an "inject" node from the Node-RED palette and configure it to trigger at a desired interval (e.g., every 5 seconds).
Connect the output of the inject node to a "serial" node.
Double-click on the serial node and configure it to communicate with the Arduino Uno serial port. Make sure the port, baud rate, and other settings match your setup.
Add a "function" node and wire it to the output of the serial node. Double-click on the function node.

FLOW CHART

1. Start: Begin the flowchart.

2. Set up Arduino Uno and DHT11 sensor: This step involves connecting the DHT11 sensor to the Arduino Uno using jumper wires. Ensure the proper connections for VCC, data, and GND.

3. Upload sketch to Arduino Uno: Upload the Arduino sketch to the Arduino Uno that reads data from the DHT11 sensor and sends it to the serial port.

4. Set up Node-RED: Install Node-RED on a computer or Raspberry Pi and launch the Node-RED flow editor.

5. Install required nodes: Install the necessary Node-RED nodes, such as node-red-node-serialport for serial communication and node-red-dashboard for creating the dashboard interface.

6. Create flow in Node-RED: Build the flow in Node-RED with the following components:

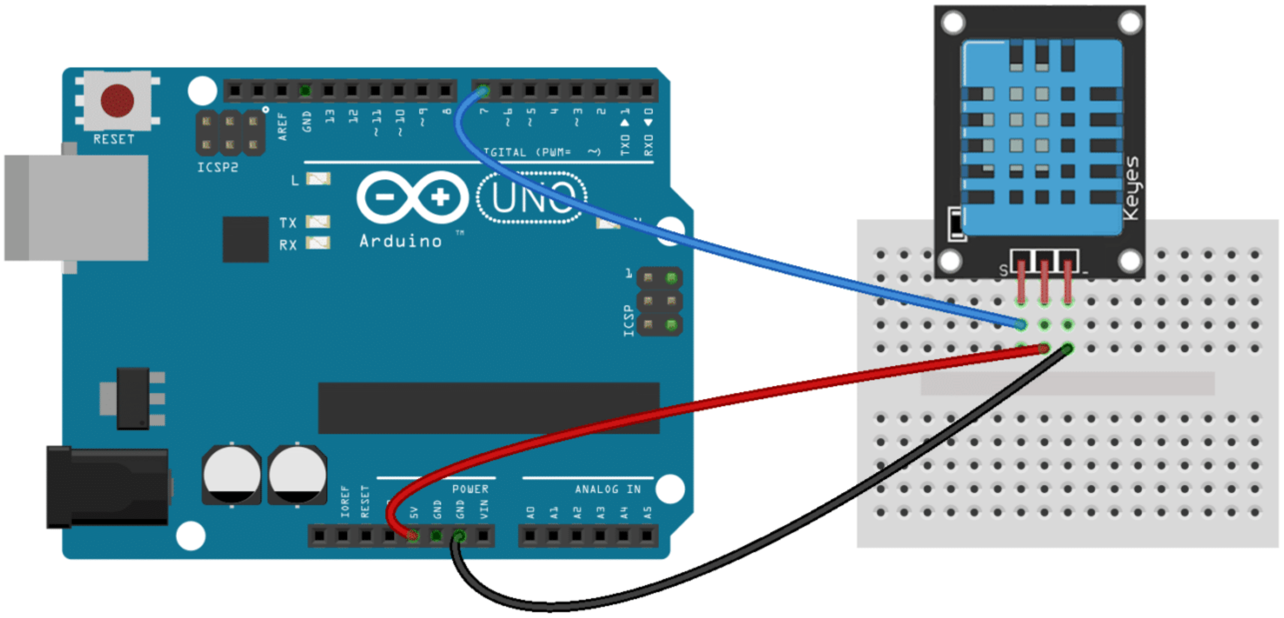
7. Inject node: Trigger the flow at a specific interval.
8. Serial node: Configure the node to communicate with the Arduino Uno's serial port.
9. Function node: Parse the incoming serial data to extract temperature and humidity values.
10. Dashboard nodes: Create the user interface to display the temperature and humidity data.
11. Deploy the flow: Deploy the Node-RED flow to make it active and start receiving data from the Arduino Uno.

12. End: End the flowchart.

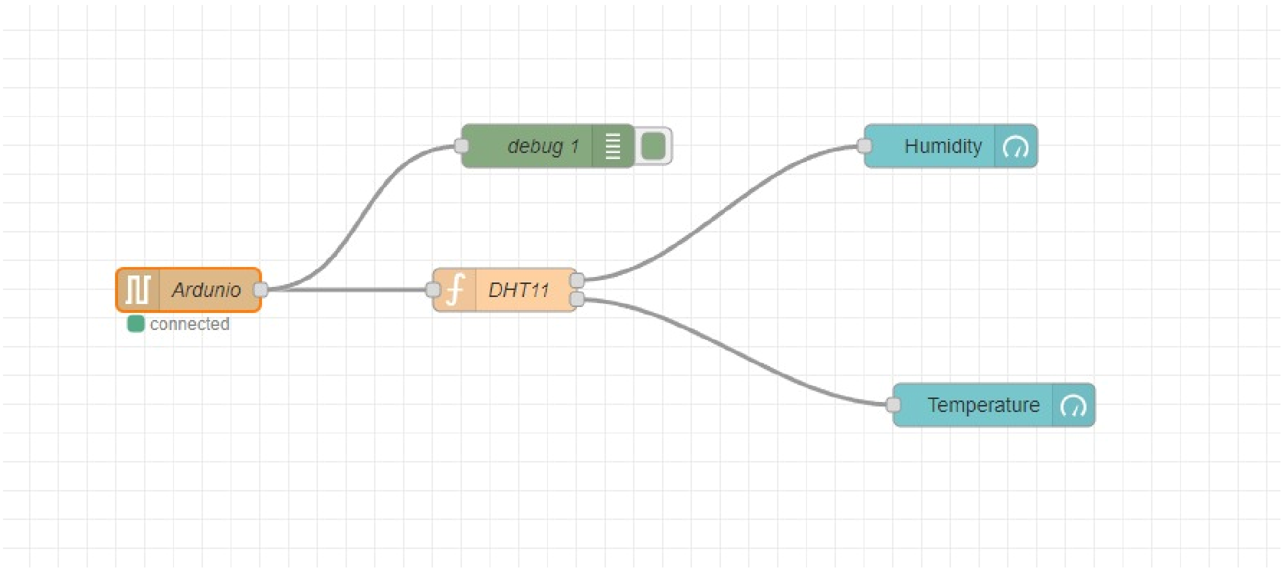
ARDUINO PROGRAM

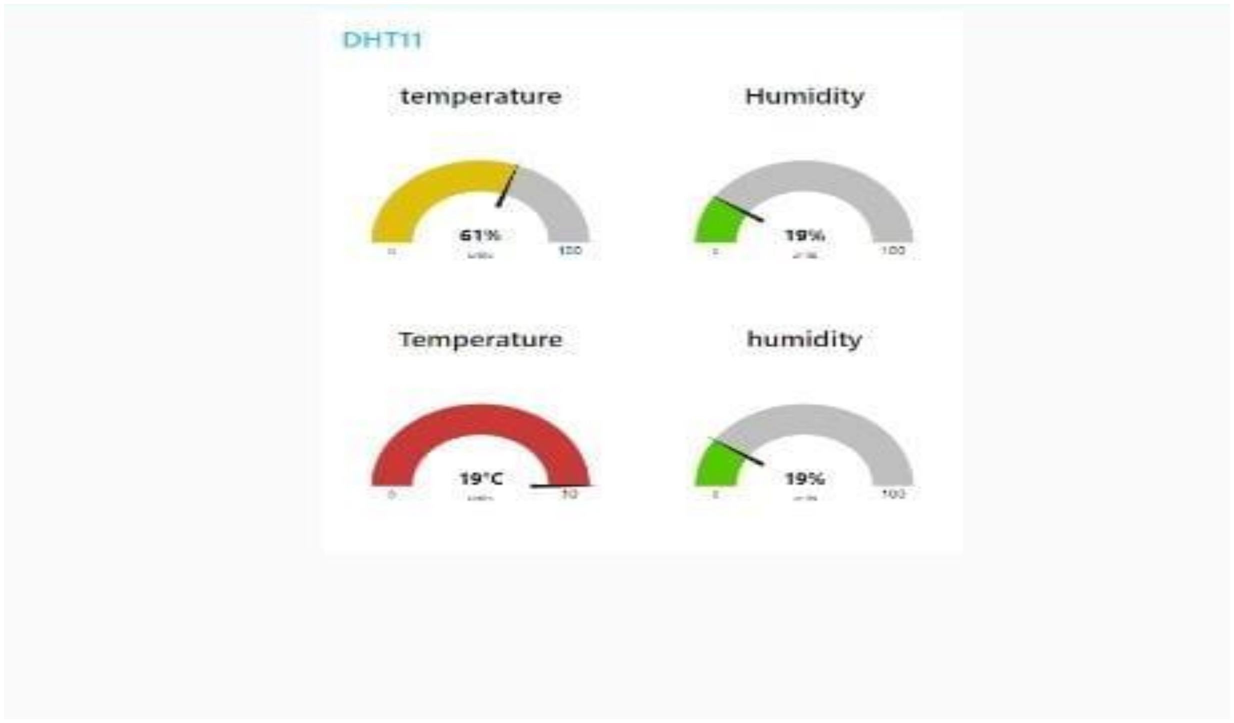
```
// Read DHT11 sensor and send serially to PC
#include <DHT.h> // Include Adafruit DHT11 Sensors Library #define DHTPIN 7 // DHT11
Output Pin connection
#define DHTTYPE DHT11 // DHT Type is DHT11 DHT
dht(DHTPIN, DHTTYPE); // Initialize DHT sensor
void setup () {
dht.begin();
Serial.begin(9600); // To see data on serial monitor
}
void loop (){
float H = dht.readHumidity(); //Read Humidity
float T = dht.readTemperature(); // Read temperature as Celsius // Check if any reads failed and if
exit
if (isnan(H) || isnan(T))
{ Serial.println("Failed to read from DHT sensor!"); return; } // Combine Humidity and
Temperature into single string String dhtData = String(H) + "," + String(T);
Serial.println(dhtData);
delay(2000); // Wait two seconds between measurements
}
```

USING ARDUINO UNO HARWARE:



SNAP SHOT OF THE NODE RED OUTPUT





INFERENCE

In this experiment I got to know to set up the inference of environmental monitoring using Arduino Uno with a DHT11 sensor and Node-RED.

RESULT

The working of the DHT 11 is performed

The Arduino Uno is connected to the DHT11 sensor, which measures temperature and humidity. The Arduino reads the sensor data and sends it to the Node-RED server.

In Node-RED, we would create a flow to receive the data from the Arduino Uno. This can be done using the serial input node, which listens to the serial port on the server where the Arduino is connected.

Once the data is received in Node-RED, we can perform various processing steps depending on your requirements. For example, we can convert the raw sensor data into meaningful units, filter or smooth the data, or calculate additional parameters.

BECE351E – INTERNET OF THINGS

EXPT NO: 4

DATE: May 16, 2023

NAME: Mainak Chattopadhyay

REG. NO.: 21BAI1217

Smart Street Light using TinkerCAD and Arduino

AIM

To perform and analyze the working of the Smart Street Light Control System

(i) Using sensor with Buzzer and LED a breadboard connected with Arduino UNO in Tinker CAD.

BLOCKS/COMPONENTS REQUIRED

(i) Software:

Tinker CAD Block Modules

1. Arduino UNO
2. LEDs
3. Ultrasonic Sensors

(ii) Hardware:

1. Arduino UNO
2. Ultrasonic Sensor
3. Sensor Interface

THEORY ABOUT SMART STREET LIGHT CONTROL SYSTEM AND ULTRASONIC SENSOR

A smart street light control system utilizing an ultrasonic sensor is a theoretical concept that aims to enhance the efficiency and functionality of street lighting. An ultrasonic sensor is a device that emits high-frequency sound waves and detects their reflections to measure distances. In the context of a smart street light control system, an ultrasonic sensor can be used to detect the presence of vehicles, pedestrians, or objects in the vicinity of the street light. The ultrasonic sensor is placed near the street light to monitor the surrounding area. When a vehicle or pedestrian enters the sensor's detection range, it sends a signal indicating the presence of an object. The ultrasonic sensor communicates the presence data to a central control system. This communication can be wired or wireless, depending on the implementation. The central control system receives the data from the ultrasonic sensor and processes it to determine appropriate actions for the street light. It uses algorithms and decision-making logic to analyze the input and make decisions based on predefined rules or user-defined settings.

Based on the input from the ultrasonic sensor, the control system can dynamically adjust the lighting of the street light. For example, if no presence is detected for a certain period, the control system may dim or turn off the street light to save energy. Conversely, when a vehicle or pedestrian is detected, the system can increase the brightness to ensure adequate illumination.

The smart street light control system aims to optimize energy consumption by adjusting lighting levels according to real-time requirements. This helps to reduce energy waste and lower operational costs.

The control system can also collect data on usage patterns, traffic flow, and other relevant information. This data can be analyzed to generate insights, identify trends, and optimize the overall street lighting infrastructure.

(i) USING TINKER CAD:

ALGORITHM

Step-1: Open TinkerCAD

Step-2: In the Design Suite, Drag and drop the required blocks and its associated components into their respective places.

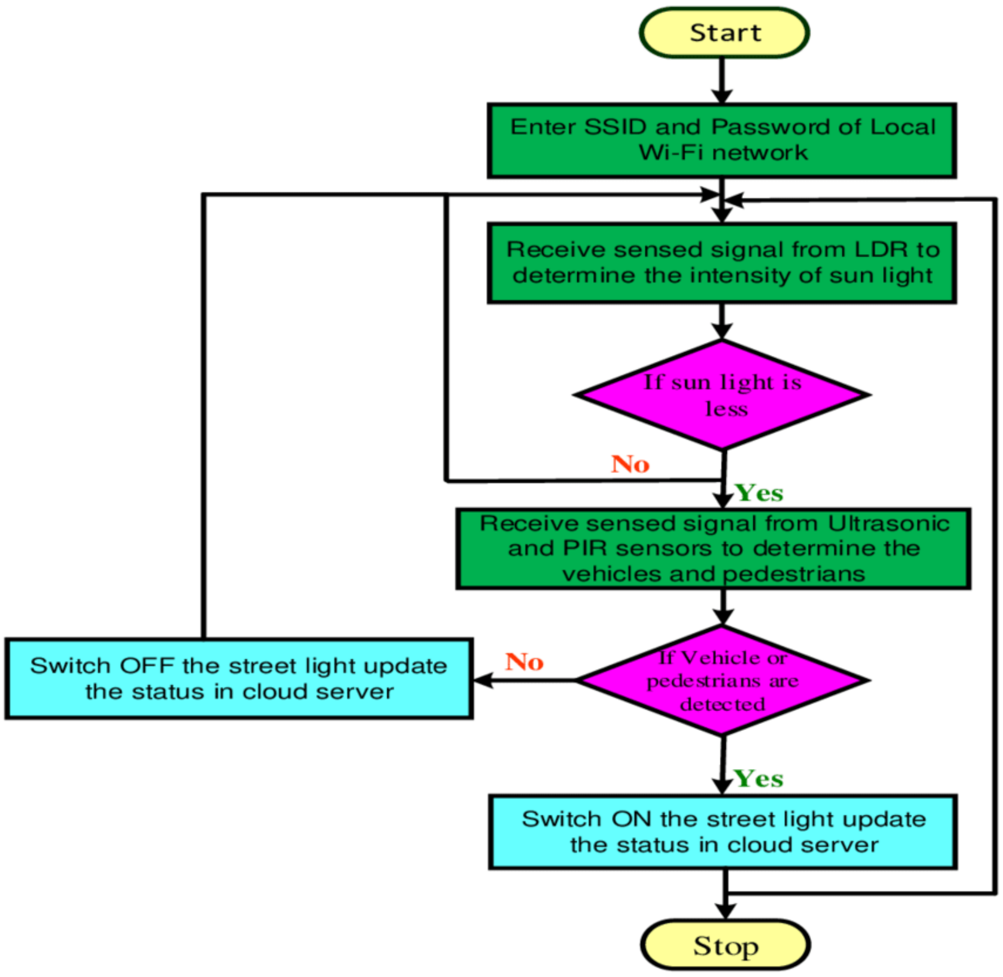
Step-3: Connect as per the circuit diagram given in the block diagram. Follow the image and place each part as shown. Notice that the breadboard is used to connect the components together. It contains a series of columns and two rails on each side. As you click from pin to pin, you'll create wires, each of which can be color-coded for easy identification.

Step-4: In code section, appropriate Arduino codes are fed. Create the next block, an "if" statement, which is a type of control block that makes a decision.

Step-5: Once you're done, select "Start Simulation" on the toolbar to turn on your Arduino Uno.

Step-6: Troubleshooting: If your program doesn't behave as expected, check your wiring and programming. Ensure that all pins are properly connected and that each block is written correctly.

FLOW CHART

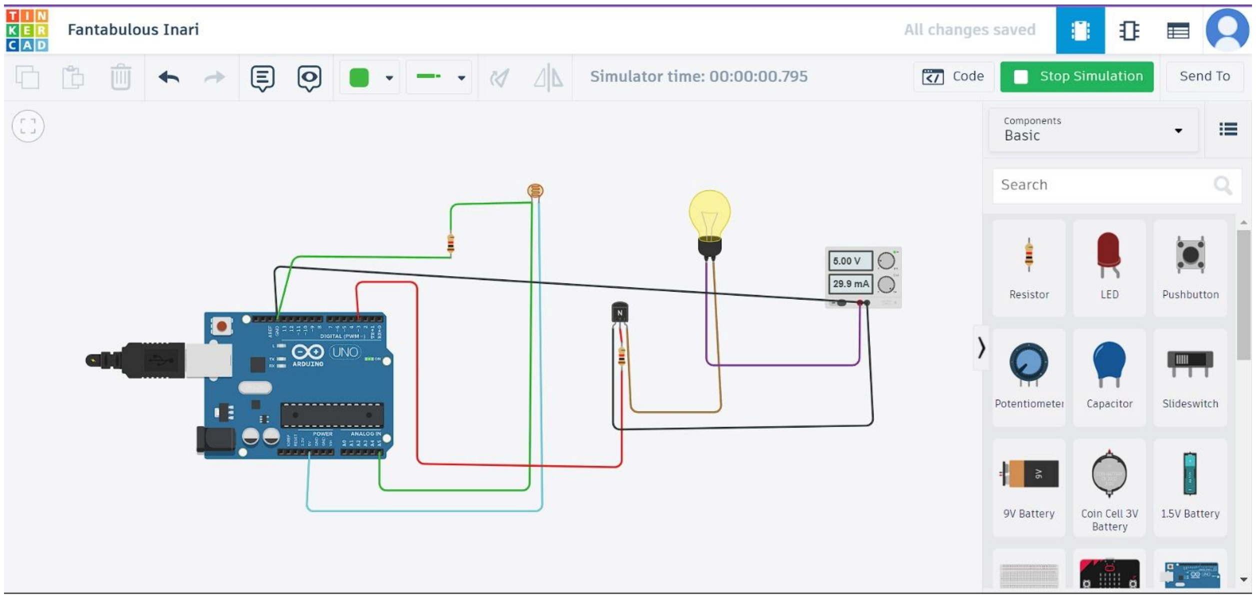


Start
Enter SSID and Password of Local Wi-Fi network
Receive sensed signal from LDR to determine the intensity of sun light
If sun light is less
No
Yes
Receive sensed signal from Ultrasonic and PIR sensors to determine the vehicles and pedestrians
If Vehicle or pedestrians are detected
No
Switch OFF the street light update the status in cloud server
Yes
Switch ON the street light update the status in cloud server
Stop

ARDUINO PROGRAM

```
int IR1 = 2;
int IR2 = 3;
int LED1 = 5;
int LED2 = 6;
int LDR = A3;
void setup()
{
Serial.begin(9600);
pinMode(LED1, OUTPUT);
pinMode(LED2, OUTPUT);
pinMode(IR1, INPUT);
pinMode(IR2, INPUT);
pinMode(LDR, INPUT);
}

void loop() {
intLDRValue = analogRead(LDR);
Serial.print("sensor = ");
Serial.print(LDRValue);
delay (500);
digitalWrite(LED1, LOW);
digitalWrite(LED2, LOW);
Serial.println("It's Bright Outside; Lights status: OFF");
if (LDRValue< 600 &&digitalRead(IR1) == HIGH)
   {
digitalWrite(LED1, HIGH);
Serial.println("It's Dark Outside; LED1 Lights status: ON");
   }

if (LDRValue> 600 &&digitalRead(IR2) == HIGH)
   {
digitalWrite(LED2, HIGH);

Serial.println("It's Dark Outside; LED2 Lights status: ON");
   }
}
```

SNAP SHOT OF THE OUTPUT



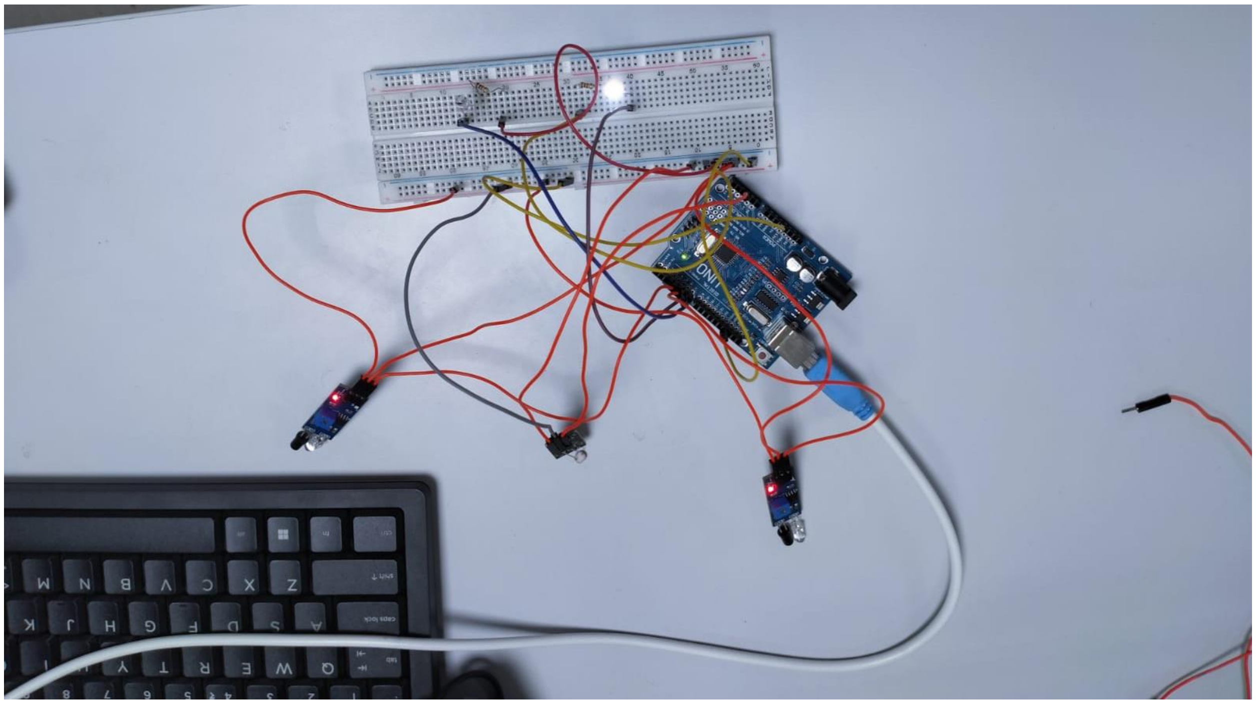
INFERENCE

The working of the Ultrasonic sensor detector and implementing Smart Street Control System with Arduino UNO in Tinker CAD is analyzed and the variations in the LED output is observed with respect to Ultrasonic sensor readings

(ii) USING ARDUINO UNO HARDWARE:

HARDWARE CIRCUIT BLOCK DIAGRAM

HARDWARE CONNECTIONS



MEASUREMENT OF TRAFFIC LIGHT OUTPUT USING ARDUINO UNO

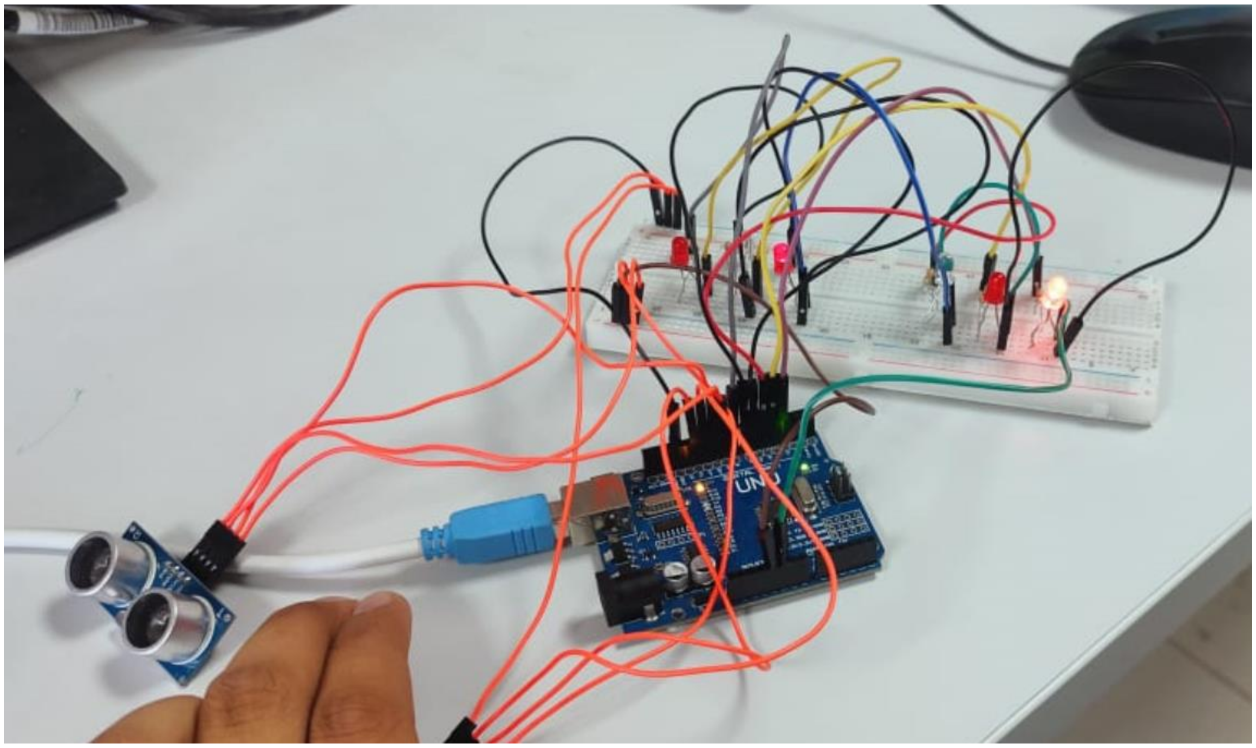
The distances are measured based on the lightPin and the echoPin readings

Now for a specific condition which are satisfied by the distance we can print the LED output.

ARDUINO PROGRAM

```
const int triggerPin = 2;    // Ultrasonic sensor trigger pin
const int echoPin = 3;       // Ultrasonic sensor echo pin
const int lightPin = 13;     // Street light control pin


void setup() {
  pinMode(triggerPin, OUTPUT);
  pinMode(echoPin, INPUT);
  pinMode(lightPin, OUTPUT);
  Serial.begin(9600);
}

void loop() {
  // Trigger ultrasonic sensor
  digitalWrite(triggerPin, LOW);
  delayMicroseconds(2);
  digitalWrite(triggerPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(triggerPin, LOW);

  // Measure the duration of the ultrasonic pulse
  long duration = pulseIn(echoPin, HIGH);

  // Calculate distance based on the speed of sound
  float distance = duration * 0.034 / 2;

  Serial.print("Distance: ");
  Serial.print(distance);
  Serial.println(" cm");

  // Adjust street light based on distance
  if (distance > 50) {
    // If no object detected within 50 cm, turn on the street light
    digitalWrite(lightPin, HIGH);
  } else {
    // If an object is detected within 50 cm, turn off the street light
    digitalWrite(lightPin, LOW);
  }

  delay(1000); // Delay between measurements
}
```

SNAP SHOT OF THE OUTPUT



INFERENCE

The working of the Ultrasonic sensor detector and implementing Smart Street Light Control System with Arduino UNO in hardware connection is analyzed and the variations in the LED output is observed with respect to Ultrasonic sensor readings. I learnt how to connect various components on the breadboard and interface them with Arduino UNO.

RESULT

The working of the Smart Street Light Control System is performed

(i) using Ultrasonic sensor with LED a breadboard connected with Arduino UNO in Tinker CAD is analyzed and the variations in the moisture content is observed.

(ii) Using the Ultrasonic sensor and Arduino Board and the variations in the LED output is observed.

BECE351E – INTERNET OF THINGS

EXP-NO: 5

DATE: 24/05/2023

NAME:

Mainak Chattopadhyay

REG-NO: 21BAI1217

SMART Car Parking System using TinkerCAD and Arduino

AIM:

The aim of this experiment is to design and implement a SMART Car Parking System using TinkerCAD and Arduino, with the objective of creating an automated and efficient parking solution that maximizes space utilization, minimizes human error, and improves overall parking management.

BLOCKS/COMPONENTS REQUIRED:

Software:

TinkerCAD

Hardware:

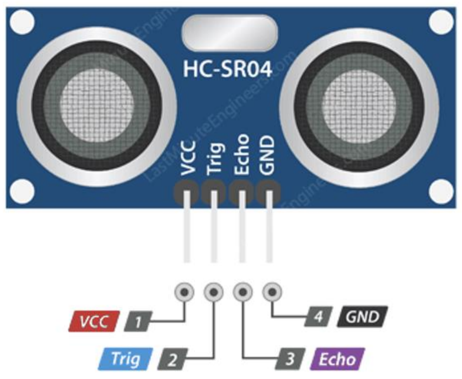
1. LED 3
2. Resistors: 3
3. Ultrasonic sensors

THEORY:

HC-SR04 ultrasonic sensor

•When is sensor is powered on an ultrasonic sound wave(at higher wavelength, humans cannot hear) is emitted through one of its transducers, and waiting for the sound to bounce off at some present object, the echo is captured by the second transducer.

•The distance is proportional to the time it takes for the echo to arrive.
•The sensor sends an ultrasonic sound wave through the trigger or trigger, bounces off the object and the receiver or echo detects the wave.
•By measuring the time taken for the wave from emission till it gets back we can know the distance.



Trigger Pin: pin is used to trigger ultrasonic sound pulses. By setting this pin to HIGH for 10µs, the sensor initiates an ultrasonic burst.

Echo: pin goes high when the ultrasonic burst is transmitted and remains high until the sensor receives an echo, after which it goes low. By measuring

ALGORITHM:

1. Initialize the Arduino and ultrasonic sensors.
2. Define the pin connections for the ultrasonic sensors.
3. Create variables to store the distance measured by each sensor and the threshold distance for occupancy detection.
4. Set up the display module to show the status of each parking space.
5. Set up the barrier control mechanism (if applicable).
6. In a loop:

   a. Read the distance from each ultrasonic sensor.
   b. Compare the measured distance with the threshold distance to determine if a parking space is occupied or vacant.
   c. Update the display module to indicate the status of each parking space.

d. If a parking space is detected as vacant, open the barrier (if applicable).
e. If a parking space is detected as occupied, close the barrier (if applicable).
f. Repeat the loop.

7. Add any additional functionalities or features as required, such as a timer for parking duration, payment processing, or integration with a mobile application.

FLOWCHART:

1. Start
2. Initialize Arduino and ultrasonic sensors
3. Define pin connections for the ultrasonic sensors
4. Set up display module
5. Set up barrier control mechanism (if applicable)
6. Loop:

   - Read distance from each ultrasonic sensor

   - Compare measured distance with threshold distance

   - If distance < threshold, parking space occupied

     - Update display module: Occupied status

     - Close barrier (if applicable)

   - If distance >= threshold, parking space vacant

     - Update display module: Vacant status

     - Open barrier (if applicable)

   - Repeat loop

7. Additional functionalities or features as required
8. End

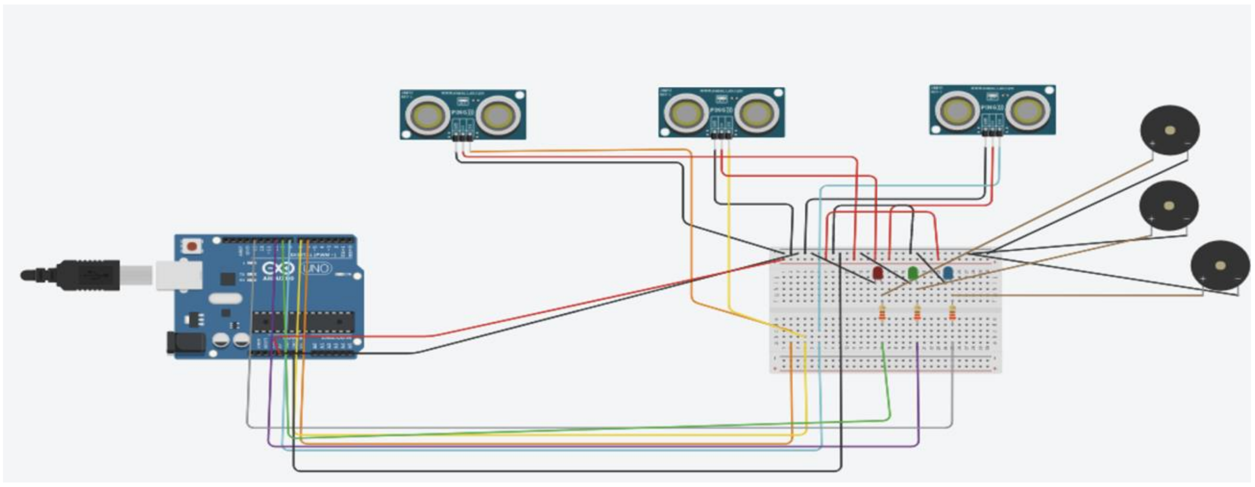
ARDUINO PROGRAM

```
const int trigPin = 2;
const int echoPin = 3;
const int buzzer = 6;
// defines variables
long duration;
int distance;
int safetyDistance;

void setup() {
pinMode(trigPin, OUTPUT); // Sets the
trigPin as an Output
pinMode(echoPin, INPUT); // Sets the
echoPin as an Input
pinMode(buzzer, OUTPUT);
safetyDistance = ultrasonic();
Serial.begin(9600); // Starts the serial
communication
}
```

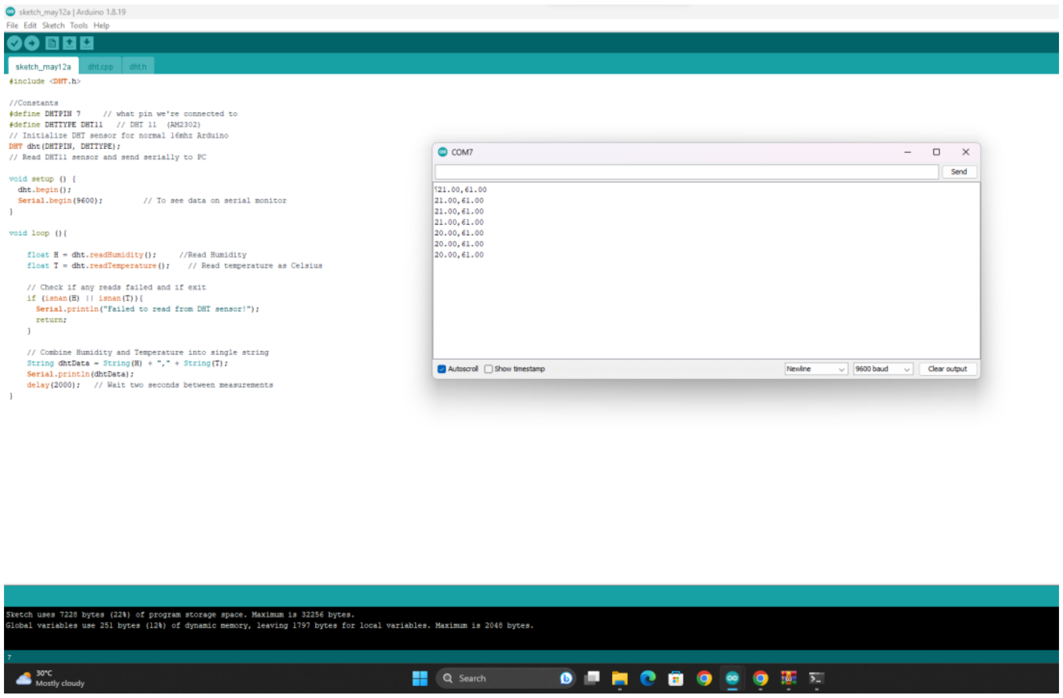
```
void loop() {
int dist = ultrasonic();
if (dist <= safetyDistance){
 tone(buzzer, 2500);
 delay(500);
 tone (buzzer, 2500);
}
else{
 noTone (buzzer);
}
// Prints the distance on the
Serial Monitor
Serial.print("Distance: ");
Serial.println(dist);
}
```

```
int ultrasonic(){
// Clears the trigPin
digitalWrite(trigPin, LOW);
delayMicroseconds(2);
// Sets the trigPin on HIGH state
for 10 micro seconds
digitalWrite(trigPin, HIGH);
delayMicroseconds(10);
digitalWrite(trigPin, LOW);
// Reads the echoPin, returns the
sound wave travel time in
microseconds
duration = pulseIn(echoPin,
HIGH);
// Calculating the distance
distance= duration*0.034/2;
return distance;
}
```

USING ARDUINO UNO HARWARE:



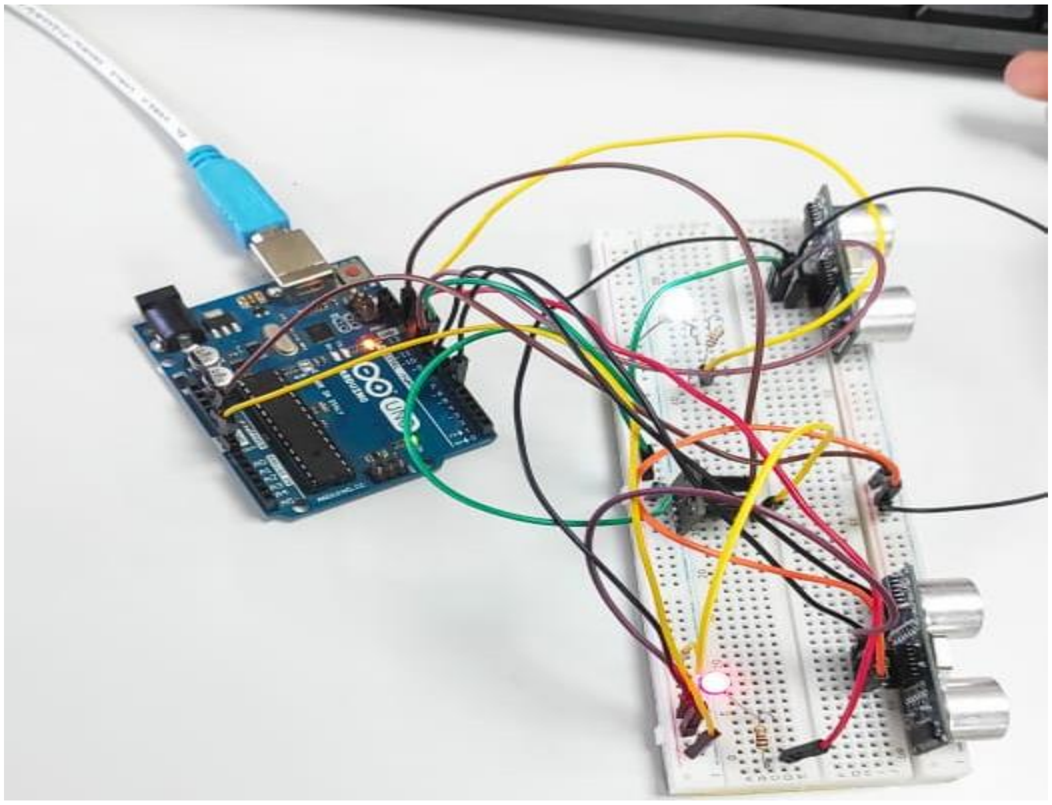
SNAP SHOT OF THE OUTPUT:



```
#include <DHT.h>

//Constants
#define DHTPIN 7     // what pin we're connected to
#define DHTTYPE DHT11   // DHT 11  (AM2302)
// Initialize DHT sensor for normal 16mhz Arduino
DHT dht(DHTPIN, DHTTYPE);
// Read DHT11 sensor and send serially to PC

void setup () {
  dht.begin();
  Serial.begin(9600);         // To see data on serial monitor
}

void loop (){

    float H = dht.readHumidity();     //Read Humidity
    float T = dht.readTemperature();    // Read temperature as Celsius

    // Check if any reads failed and if exit
    if (isnan(H) || isnan(T)){
      Serial.println("Failed to read from DHT sensor!");
      return;
    }

    // Combine Humidity and Temperature into single string
    String dhtData = String(H) + "," + String(T);
    Serial.println(dhtData);
    delay(2000);   // Wait two seconds between measurements
}
```

```
21.00,61.00
21.00,61.00
21.00,61.00
21.00,61.00
20.00,61.00
20.00,61.00
20.00,61.00
```

```
Sketch uses 7228 bytes (22%) of program storage space. Maximum is 32256 bytes.
Global variables use 251 bytes (12%) of dynamic memory, leaving 1797 bytes for local variables. Maximum is 2048 bytes.
```



INFERENCE:
The SMART Car Parking System experiment using TinkerCAD and Arduino with ultrasonic sensors provides an efficient and automated solution for parking space management. By utilizing ultrasonic sensors, the system can accurately detect the presence of vehicles in parking spaces and determine their occupancy status. The integration of Arduino control, data processing, and display modules allows users to easily locate available parking spots in real-time.

RESULT:

The SMART Car Parking System successfully detects and indicates the occupancy status of each parking space.
As vehicles enter or leave parking spaces, the system accurately updates the display module to reflect the current availability. When a parking space is vacant, the barrier (if applicable) opens automatically, allowing vehicles to enter.
Conversely, when a parking space is occupied, the barrier remains closed to prevent other vehicles from entering. Users can easily identify and navigate to available parking spaces, streamlining the parking process and minimizing congestion.
Overall, the experiment demonstrates the effectiveness of using ultrasonic sensors,

Arduino,and TinkerCAD in creating a SMART Car Parking System that improves parking management and optimizes space utilization.

BECE351E – INTERNET OF THINGS

EXPT NO: 6

DATE: May 31, 2023

NAME: Mainak Chattopadhyay

REG. NO.: 21BAI1217

SMART Traffic Light Control System using TinkerCAD and Arduino

AIM

To perform and analyze the working of the Traffic Light Control System

(i) Using sensor with Buzzer and LED a breadboard connected with Arduino UNO in Tinker CAD.

BLOCKS/COMPONENTS REQUIRED

(i) Software:

Tinker CAD Block Modules

1. Arduino UNO
2. LEDs
3. Ultrasonic Sensors

(ii) Hardware:

1. Arduino UNO
2. Ultrasonic Sensor
3. Sensor Interface

THEORY ABOUT TRAFFIC LIGHT CONTROL SYSTEM AND ULTRASONIC SENSOR

The working of a traffic light control system involves a combination of hardware and software components that work together to manage the flow of traffic at intersections. a control algorithm within the system determines the appropriate timing and sequencing of the traffic lights. The algorithm aims to optimize traffic flow and minimize delays while ensuring safety and fairness for all directions of traffic. In situations where multiple intersections are connected or coordinated, the traffic light control system may incorporate coordination strategies. This coordination allows for the synchronization of signals along a traffic corridor, optimizing the movement of vehicles through multiple intersections.

Ultrasonic sensors are commonly used for object detection and ranging. They work based on the principle of sound waves and their reflections. When the sound waves hit an object, a portion of the sound is reflected back towards the sensor. The rest of the sound may be absorbed, transmitted through the object, or scattered in various directions. Based on the calculated distance, the sensor can determine if an object is present within its detection range.

(i) USING TINKER CAD:

ALGORITHM

Step-1: Open TinkerCAD

Step-2: In the Design Suite, Drag and drop the required blocks and its associated components into their respective places.

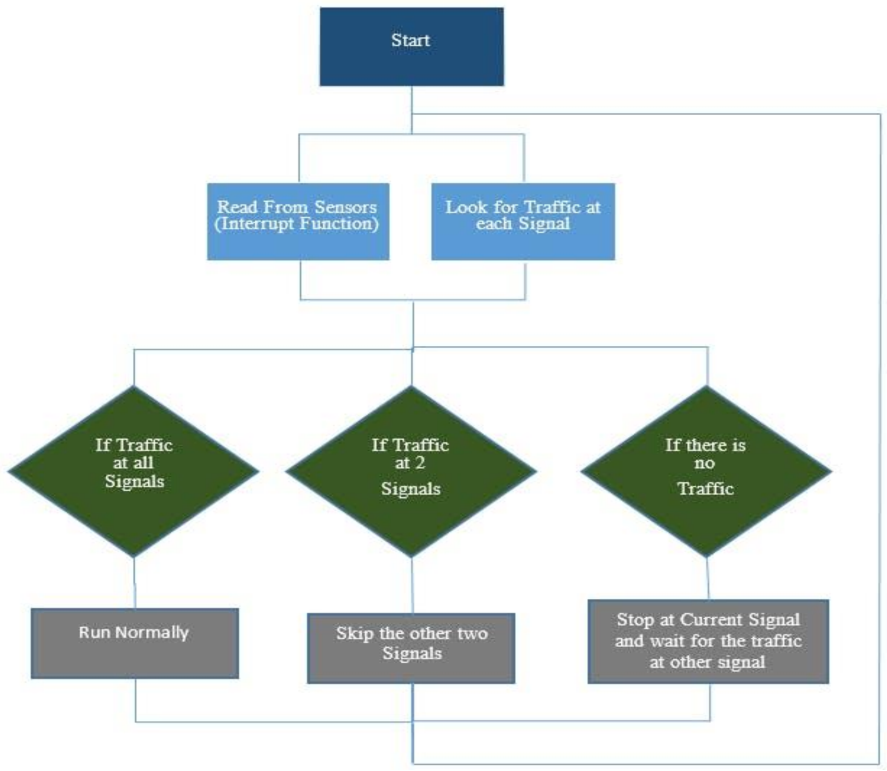
Step-3: Connect as per the circuit diagram given in the block diagram. Follow the image and place each part as shown. Notice that the breadboard is used to connect the components together. It contains a series of columns and two rails on each side. As you click from pin to pin, you'll create wires, each of which can be color-coded for easy identification.

Step-4: In code section, appropriate Arduino codes are fed. Create the next block, an "if" statement, which is a type of control block that makes a decision.

Step-5: Once you're done, select "Start Simulation" on the toolbar to turn on your Arduino Uno.

Step-6: Troubleshooting: If your program doesn't behave as expected, check your wiring and programming. Ensure that all pins are properly connected and that each block is written correctly.

FLOW CHART


Start
Read From Sensors (Interrupt Function)
Look for Traffic at each Signal
If Traffic at all Signals
If Traffic at 2 Signals
If there is no Traffic
Run Normally
Skip the other two Signals
Stop at Current Signal and wait for the traffic at other signal

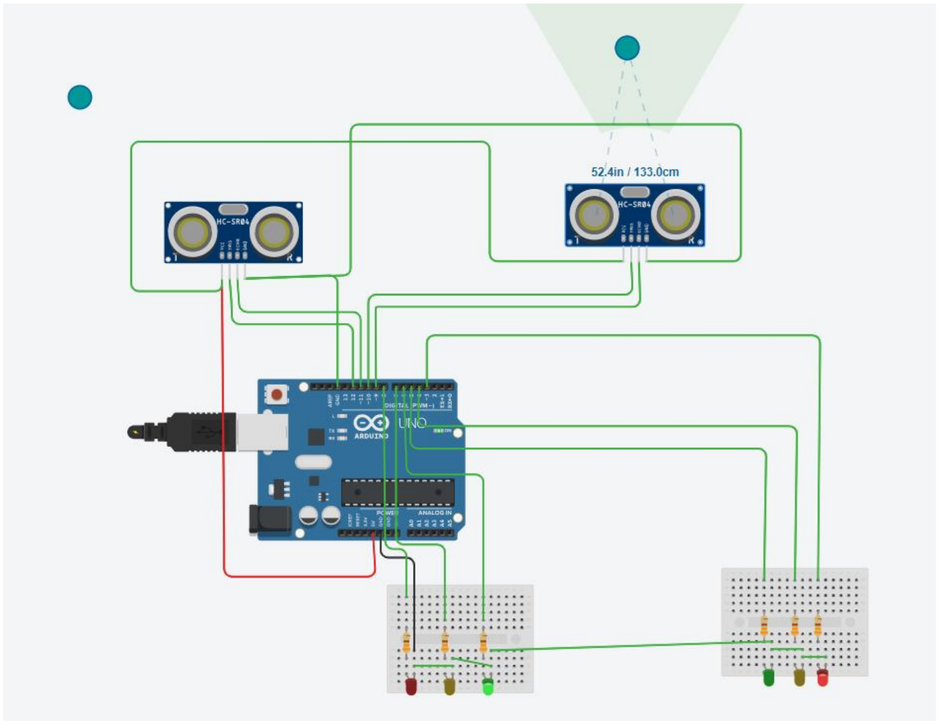
ARDUINO PROGRAM

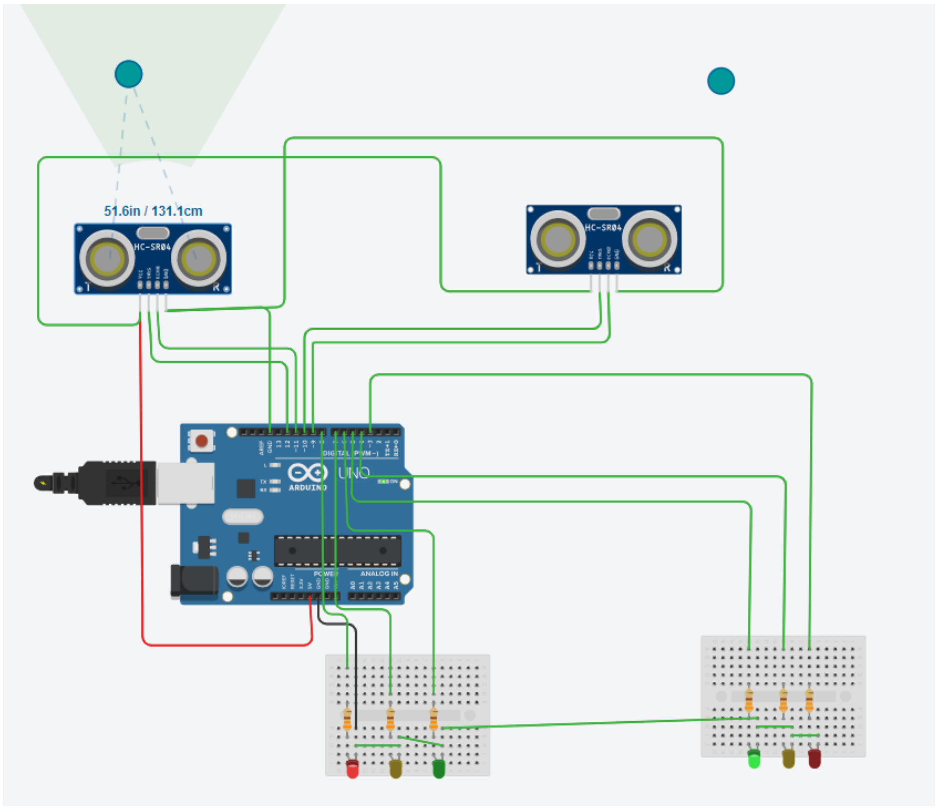
```
int red1=8;
int green1=6;
int yellow1=7;
int red2=3;
int green2=5;
int yellow2=4;
const int pingPin = 10; // Trigger Pin of Ultrasonic Sensor
const int echoPin = 9; // Echo Pin of Ultrasonic Sensor
const int pingPin2 = 12; // Trigger Pin of Ultrasonic Sensor
const int echoPin2 = 11; // Echo Pin of Ultrasonic Sensor

void setup() {
  // put your setup code here, to run once:
```

```
pinMode(red1,OUTPUT);
pinMode(red2,OUTPUT);
pinMode(yellow1,OUTPUT);
pinMode(yellow2,OUTPUT);
pinMode(green1,OUTPUT);
pinMode(green2,OUTPUT);
}

void loop() {
  // put your main code here, to run repeatedly:
  int distance1,distance2;
  distance1=calculatedistance(pingPin , echoPin);
  distance2=calculatedistance(pingPin2 , echoPin2);
  if(distance1>=distance2){
     digitalWrite(red1,LOW);
digitalWrite(green2,LOW);
digitalWrite(red2,LOW);
digitalWrite(green1,LOW);
digitalWrite(yellow1,HIGH);
digitalWrite(yellow2,HIGH);
delay(200);
    while(distance1>distance2){
   distance1=calculatedistance(pingPin , echoPin);
  distance2=calculatedistance(pingPin2 , echoPin2);
digitalWrite(red1,HIGH);
digitalWrite(green2,HIGH);
digitalWrite(red2,LOW);
digitalWrite(green1,LOW);
digitalWrite(yellow1,LOW);
digitalWrite(yellow2,LOW);
    }


  }




if(distance2>distance1){
    digitalWrite(red1,LOW);
digitalWrite(green2,LOW);
digitalWrite(red2,LOW);
digitalWrite(green1,LOW);
digitalWrite(yellow1,HIGH);
digitalWrite(yellow2,HIGH);
delay(200);
  while(distance2>distance1){
      distance1=calculatedistance(pingPin , echoPin);
  distance2=calculatedistance(pingPin2 , echoPin2);
digitalWrite(red1,LOW);
digitalWrite(green2,LOW);
digitalWrite(red2,HIGH);
digitalWrite(green1,HIGH);
digitalWrite(yellow1,LOW);
digitalWrite(yellow2,LOW);
  }
```

```
}

}


long microsecondsToCentimeters(long microseconds)
{
   return microseconds / 29 / 2;
}

int calculatedistance(int pingPin , int echoPin){
    long duration, inches, cm,meter;
  pinMode(pingPin, OUTPUT);
  digitalWrite(pingPin, LOW);
  delayMicroseconds(2);
  digitalWrite(pingPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(pingPin, LOW);

  pinMode(echoPin, INPUT);
  duration = pulseIn(echoPin, HIGH);

  cm = microsecondsToCentimeters(duration);
  meter = cm/100;
  return meter;
}
```

SNAP SHOT OF THE OUTPUT



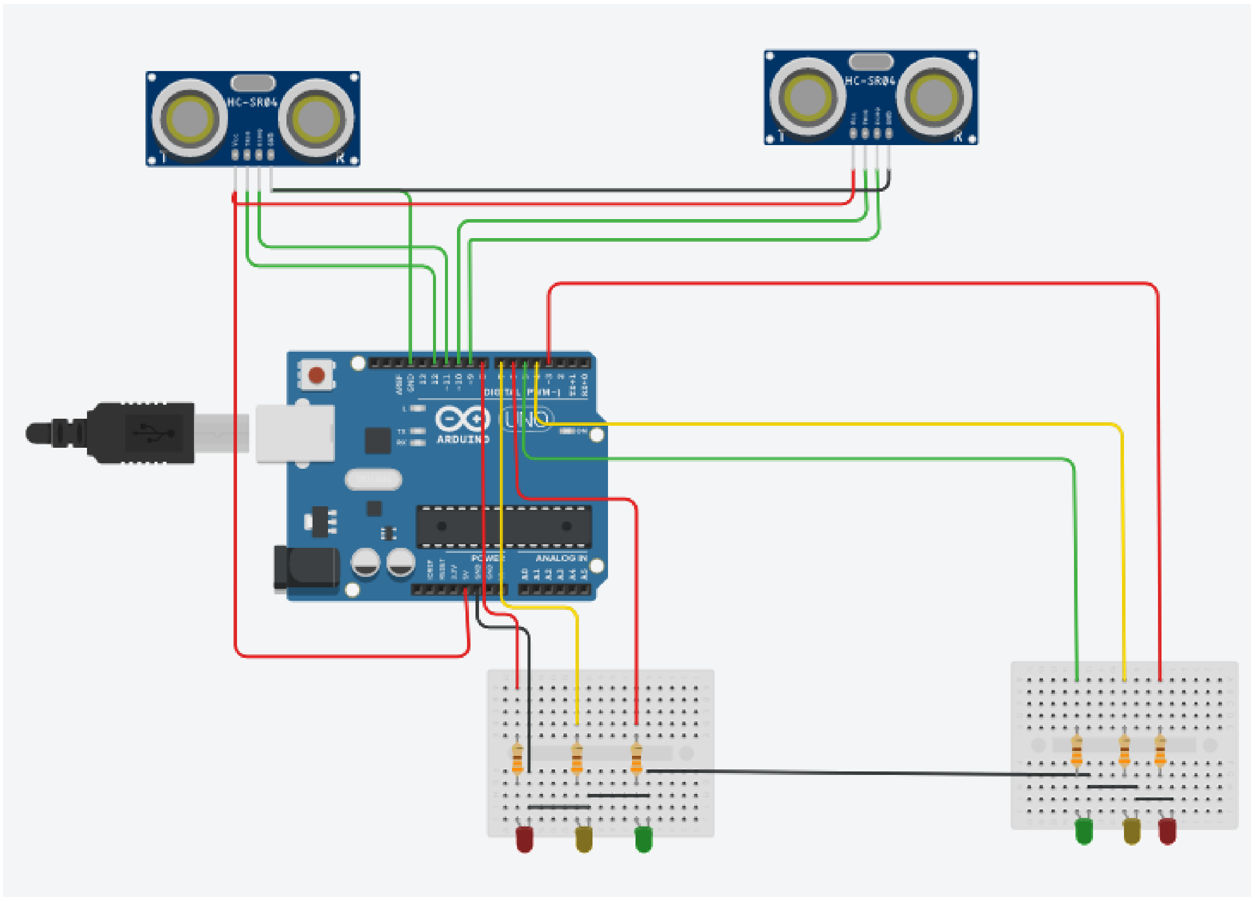


INFERENCE

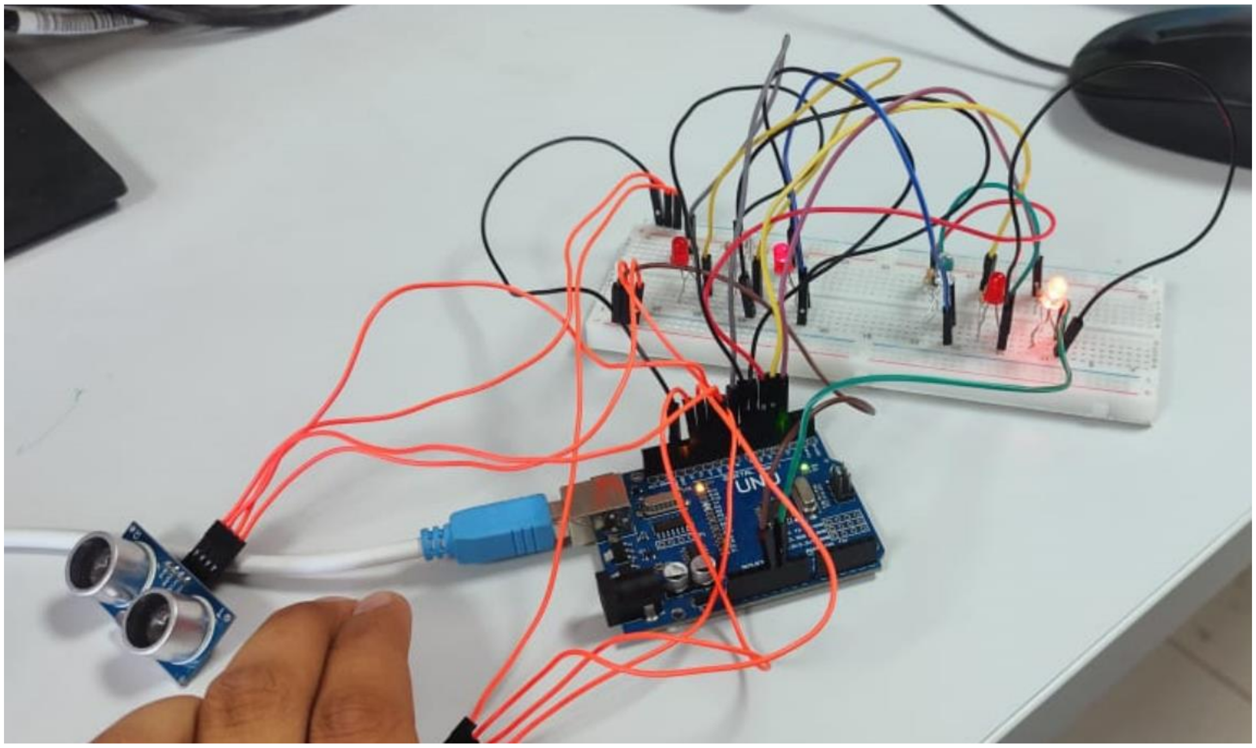
The working of the Ultrasonic sensor detector and implementing Traffic Control System with Arduino UNO in Tinker CAD is analyzed and the variations in the LED output is observed with respect to Ultrasonic sensor readings

(ii) USING ARDUINO UNO HARWARE:

HARDWARE CIRCUIT BLOCK DIAGRAM



HARDWARE CONNECTIONS



MEASUREMENT OF TRAFFIC LIGHT OUTPUT USING ARDUINO UNO

The distances are measured based on the pingPin and the echoPin readings

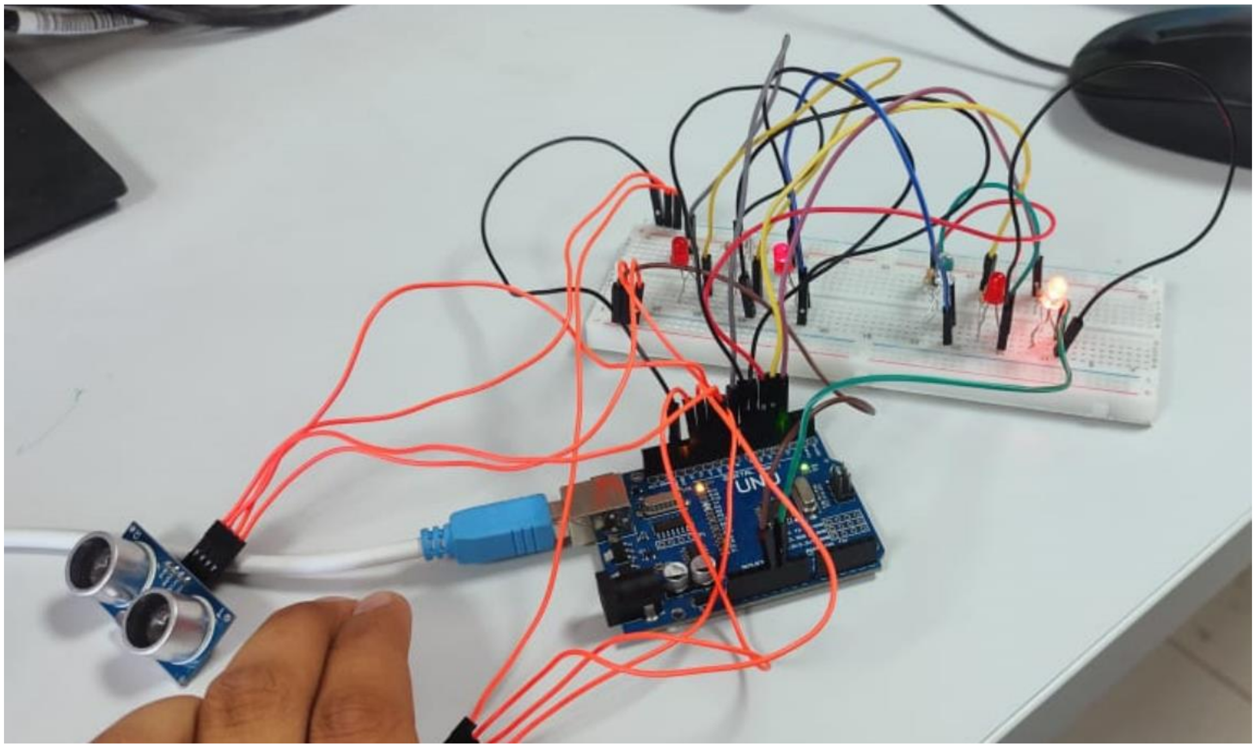
Now for a specific condition which are satisfied by the distance we can print the LED output.

ARDUINO PROGRAM

```
int red1=8;
int green1=6;
int yellow1=7;
int red2=3;
int green2=5;
int yellow2=4;
const int pingPin = 10; // Trigger Pin of Ultrasonic Sensor
const int echoPin = 9; // Echo Pin of Ultrasonic Sensor
const int pingPin2 = 12; // Trigger Pin of Ultrasonic Sensor
const int echoPin2 = 11; // Echo Pin of Ultrasonic Sensor

void setup() {
  // put your setup code here, to run once:
pinMode(red1,OUTPUT);
pinMode(red2,OUTPUT);
pinMode(yellow1,OUTPUT);
pinMode(yellow2,OUTPUT);
pinMode(green1,OUTPUT);
pinMode(green2,OUTPUT);
}

void loop() {
  // put your main code here, to run repeatedly:
  int distance1,distance2;
  distance1=calculatedistance(pingPin , echoPin);
  distance2=calculatedistance(pingPin2 , echoPin2);
  if(distance1>=distance2){
     digitalWrite(red1,LOW);
digitalWrite(green2,LOW);
digitalWrite(red2,LOW);
digitalWrite(green1,LOW);
digitalWrite(yellow1,HIGH);
digitalWrite(yellow2,HIGH);
delay(200);
    while(distance1>distance2){
   distance1=calculatedistance(pingPin , echoPin);
  distance2=calculatedistance(pingPin2 , echoPin2);
```

```
digitalWrite(red1,HIGH);
digitalWrite(green2,HIGH);
digitalWrite(red2,LOW);
digitalWrite(green1,LOW);
digitalWrite(yellow1,LOW);
digitalWrite(yellow2,LOW);
    }

  }



if(distance2>distance1){
    digitalWrite(red1,LOW);
digitalWrite(green2,LOW);
digitalWrite(red2,LOW);
digitalWrite(green1,LOW);
digitalWrite(yellow1,HIGH);
digitalWrite(yellow2,HIGH);
delay(200);
  while(distance2>distance1){
      distance1=calculatedistance(pingPin , echoPin);
  distance2=calculatedistance(pingPin2 , echoPin2);
digitalWrite(red1,LOW);
digitalWrite(green2,LOW);
digitalWrite(red2,HIGH);
digitalWrite(green1,HIGH);
digitalWrite(yellow1,LOW);
digitalWrite(yellow2,LOW);
  }



}


}



long microsecondsToCentimeters(long microseconds)
{
   return microseconds / 29 / 2;
}

int calculatedistance(int pingPin , int echoPin){
    long duration, inches, cm,meter;
  pinMode(pingPin, OUTPUT);
  digitalWrite(pingPin, LOW);
  delayMicroseconds(2);
  digitalWrite(pingPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(pingPin, LOW);

  pinMode(echoPin, INPUT);
  duration = pulseIn(echoPin, HIGH);

  cm = microsecondsToCentimeters(duration);
  meter = cm/100;
  return meter;
}
```

SNAP SHOT OF THE OUTPUT



INFERENCE

The working of the Ultrasonic sensor detector and implementing Traffic Control System with Arduino UNO in hardware connection is analyzed and the variations in the LED output is observed with respect to Ultrasonic sensor readings. I learnt how to connect various components on the breadboard and interface them with Arduino UNO.

RESULT

The working of the Traffic Control System is performed

(i) using Ultrasonic sensor with LED a breadboard connected with Arduino UNO in Tinker CAD is analyzed and the variations in the moisture content is observed.

(ii) Using the Ultrasonic sensor and Arduino Board and the variations in the LED output is observed.

Vellore Institute of Technology
Chennai Campus

Programme: BTech CSE with AI and ML
BECE351E - Internet of Things (IoT) Lab
Lab Sheet 7

Environmental monitoring using ESP32 with DHT22 and Thingspeak

Name: Mainak Chattopadhyay

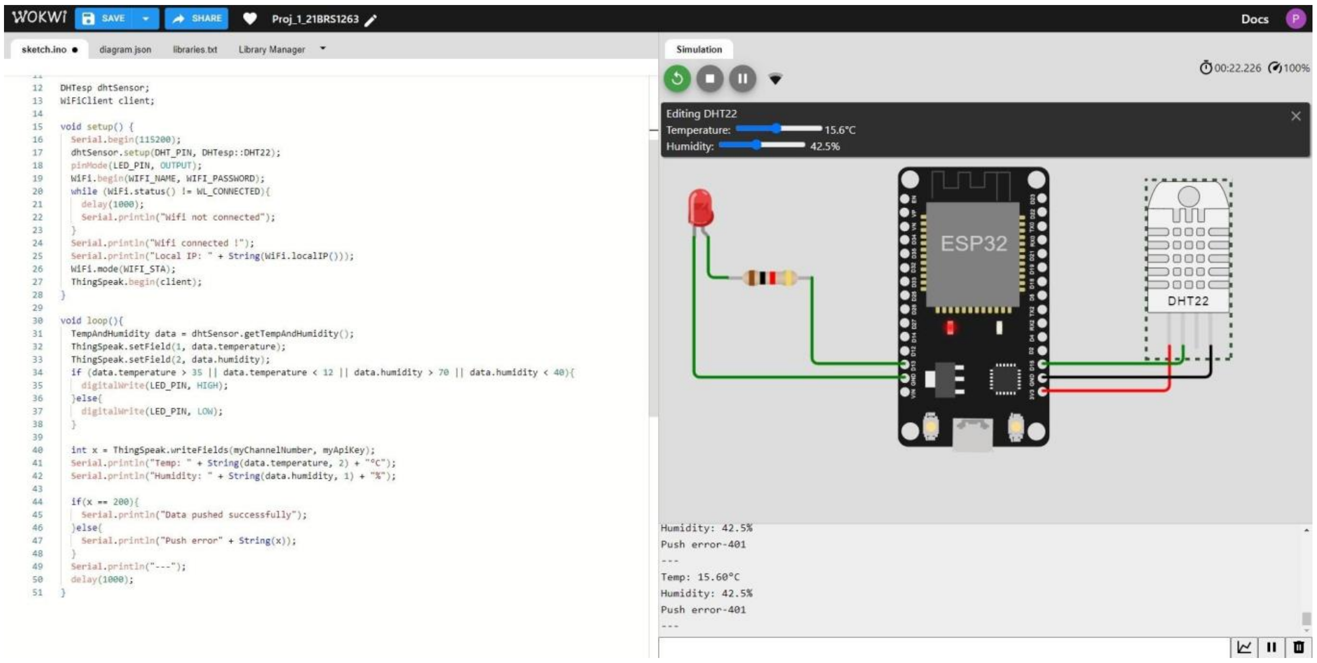
Roll Number: 21BAI1217
Date of the Lab Class: 08 June 2023

1. Introduction:
(a) The goal of this experiment is to monitor environmental conditions such as temperature and humidity using an ESP32 microcontroller, a DHT22 sensor, and sending the data to the cloud platform ThingSpeak for visualization and analysis.

(b) The required components for this experiment are:
- ESP32 development board: It provides the processing power and WiFi connectivity.
- DHT22 sensor: It measures temperature and humidity.
- Breadboard and jumper wires: Used to create the circuit connections.
- USB cable: Provides power to the ESP32 board and enables programming.
- Computer with Arduino IDE: Used for writing and uploading the code to the ESP32.

2. Circuit Diagram & Explanation:
(a) The steps involved in designing the circuit diagram are as follows:
1. Connect the VCC pin of the DHT22 sensor to the 3.3V pin of the ESP32.
2. Connect the GND pin of the DHT22 sensor to the GND pin of the ESP32.
3. Connect the Data pin of the DHT22 sensor to any digital pin of the ESP32 (e.g., GPIO 4).
4. Connect the VIN pin of the ESP32 to the USB power source.
5. Connect the GND pin of the ESP32 to the GND rail on the breadboard.
6. Connect the 3.3V pin of the ESP32 to the power rail on the breadboard.

(b) The circuit functions as follows:
The DHT22 sensor measures temperature and humidity. It communicates with the ESP32 microcontroller through a single-wire protocol. The ESP32 reads the sensor data and uses its built-in WiFi capability to connect to the ThingSpeak platform. The ESP32 then sends the sensor data to ThingSpeak using HTTP requests.



```
DHTesp dhtSensor;
WiFiClient client;

void setup() {
  Serial.begin(115200);
  dhtSensor.setup(DHT_PIN, DHTesp::DHT22);
  pinMode(LED_PIN, OUTPUT);
  WiFi.begin(WIFI_NAME, WIFI_PASSWORD);
  while (WiFi.status() != WL_CONNECTED){
    delay(1000);
    Serial.println("Wifi not connected");
  }
  Serial.println("Wifi connected !");
  Serial.println("Local IP: " + String(WiFi.localIP()));
  WiFi.mode(WIFI_STA);
  ThingSpeak.begin(client);
}

void loop(){
  TempAndHumidity data = dhtSensor.getTempAndHumidity();
  ThingSpeak.setField(1, data.temperature);
  ThingSpeak.setField(2, data.humidity);
  if (data.temperature > 35 || data.temperature < 12 || data.humidity > 70 || data.humidity < 40){
    digitalWrite(LED_PIN, HIGH);
  }else{
    digitalWrite(LED_PIN, LOW);
  }

  int x = ThingSpeak.writeFields(myChannelNumber, myApiKey);
  Serial.println("Temp: " + String(data.temperature, 2) + "°C");
  Serial.println("Humidity: " + String(data.humidity, 1) + "%");

  if(x == 200){
    Serial.println("Data pushed successfully");
  }else{
    Serial.println("Push error" + String(x));
  }
  Serial.println("---");
  delay(1000);
}
```

```
Humidity: 42.5%
Push error-401
---
Temp: 15.60°C
Humidity: 42.5%
Push error-401
---
```

(c) Code that can be used to run the circuit:

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

#define WIFI_SSID "your_wifi_ssid"
#define WIFI_PASSWORD "your_wifi_password"
#define DHT_PIN 4
#define DHT_TYPE DHT22

DHT dht(DHT_PIN, DHT_TYPE);
const char* server = "api.thingspeak.com";
const char* apiKey = "your_thingspeak_api_key";

void setup() {
  Serial.begin(115200);
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);

  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
  Serial.println("Connected to WiFi");
  dht.begin();
}

void loop() {
  float temperature = dht.readTemperature();
  float humidity = dht.readHumidity();
  if (isnan(temperature) || isnan(humidity)) {
    Serial.println("Failed to read data from DHT sensor");
    return;
  }

  Serial.print("Temperature: ");
  Serial.print(temperature);
  Serial.print(" °C\tHumidity: ");
  Serial.print(humidity);
  Serial.println(" %");
  String url = "/update?api_key=" + String(apiKey) + "&field1=" +
String(temperature) + "&field2=" + String(humidity);

  WiFiClient client;
  if (client.connect(server, 80)) {
    Serial.println("Sending data to ThingSpeak");
    client.print(String("GET ") + url + " HTTP/1.1\r\n" +
                 "Host: " + server + "\r\n" +
                 "Connection: close\r\n\r\n");
    delay(500);
    client.stop();
  }

  delay(5000);
}
```

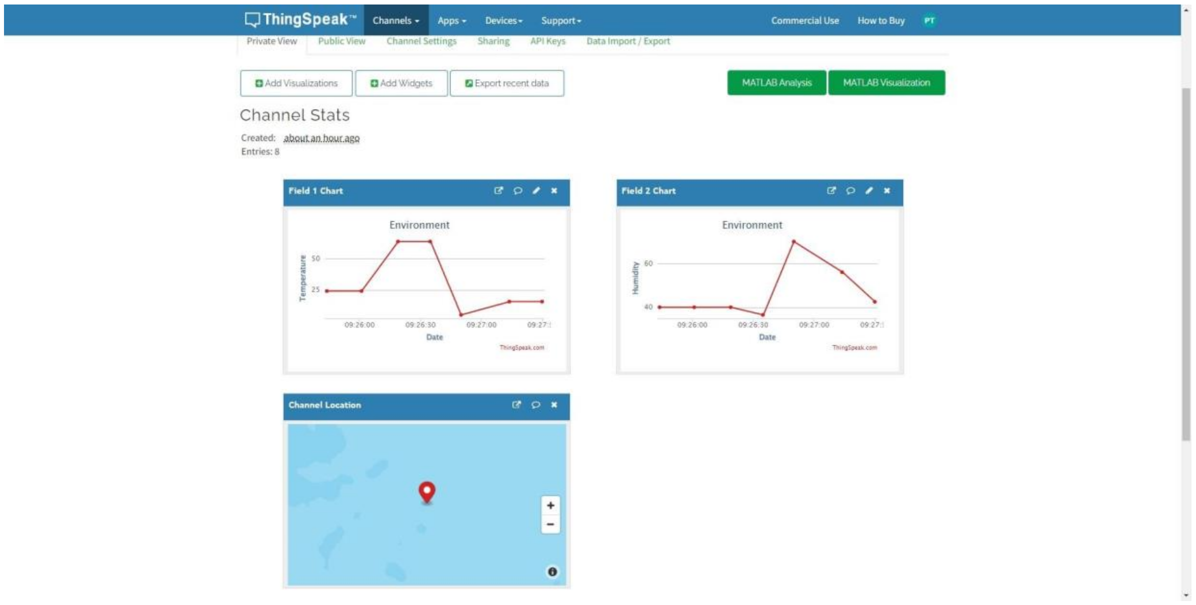
3. Output of the experiment:

(a) To run the simulation in Wokwi , follow these steps:

1. Open the Wokwi website and create a new project.
2. Drag and drop an ESP32 module, DHT22 sensor, and breadboard onto the canvas.
3. Connect the components as per the circuit diagram explained earlier.
4. Open the Arduino IDE, copy the code mentioned above, and paste it into the Arduino editor in Wokwi.
5. Click on the "Upload" button to compile and upload the code to the simulated ESP32.
6. The simulation will start running, and you will see the temperature and humidity readings in the serial monitor.

(b) To assess the output displayed in ThingSpeak:

1. Log in to your ThingSpeak account or create a new one.
2. Create a new channel and add two fields for temperature and humidity.
3. Obtain the API key for your ThingSpeak channel.
4. Modify the code's `apiKey` variable with your ThingSpeak API key.
5. Once the circuit is running and connected to WiFi, the ESP32 will send the temperature and humidity data to ThingSpeak.
6. In your ThingSpeak channel, you can view and analyze the data in real-time



Experiment-8

Smart home automation using NodeMCU (ESP32) with Relays and Thingspeak Cloud Computing (Wokwi and Thingspeak Simulation)

APPARATUS/COMPONENTS REQUIRED

1. ThingSpeak
2. Wokwi
3. 1 x ESP32
4. 1 x DHT22
5. 16 x Jump wires
6. 4 x Relay's

THEORY

1. NodeMCU (ESP32) with Relays: The NodeMCU board serves as the main hardware platform for smart home automation. It is connected to relays, which act as switches to control electrical appliances and devices.

2. Wi-Fi Connectivity: The NodeMCU establishes a Wi-Fi connection, enabling it to connect to the internet and communicate with the Thingspeak Cloud platform.

3. User Interface: Develop a user interface on the Wokwi simulation platform, allowing users to control and monitor their smart home devices. This interface can include buttons, sliders, or other interactive elements to trigger actions or change settings.

4. Thingspeak Cloud Platform: Use the Thingspeak Cloud platform to manage the smart home automation system. Create a Thingspeak channel to store and analyze data and set up APIs to send and receive commands from the NodeMCU.

5. Control Appliances: Configure the NodeMCU to receive commands from the user interface on Wokwi. Based on the received commands, the NodeMCU triggers the corresponding relays to turn on or off the respective appliances.

6. Data Logging and Analysis: Use Thingspeak to log data related to appliance usage, such as energy consumption or device status. This data can be analyzed to identify patterns, optimize energy usage, or generate insights.

7. Remote Monitoring: Through the Wokwi and Thingspeak integration, users can remotely monitor the status of their appliances, such as whether lights are on or off, or the temperature of a room. This monitoring can be done through the user interface on Wokwi or through the Thingspeak mobile application.

8. Automation and Scheduling: Implement automation and scheduling capabilities within the system. Users can define rules or schedules for their devices to automatically turn on or off based on specific conditions or time triggers.

9. Notifications: Set up notifications through Thingspeak to alert users about events or changes in their smart home system. For example, users can receive a

notification when a specific appliance has been left on for too long.

10. Expandability: The system can be expanded by adding more NodeMCU boards and relays to control additional appliances or devices in different areas of the home.

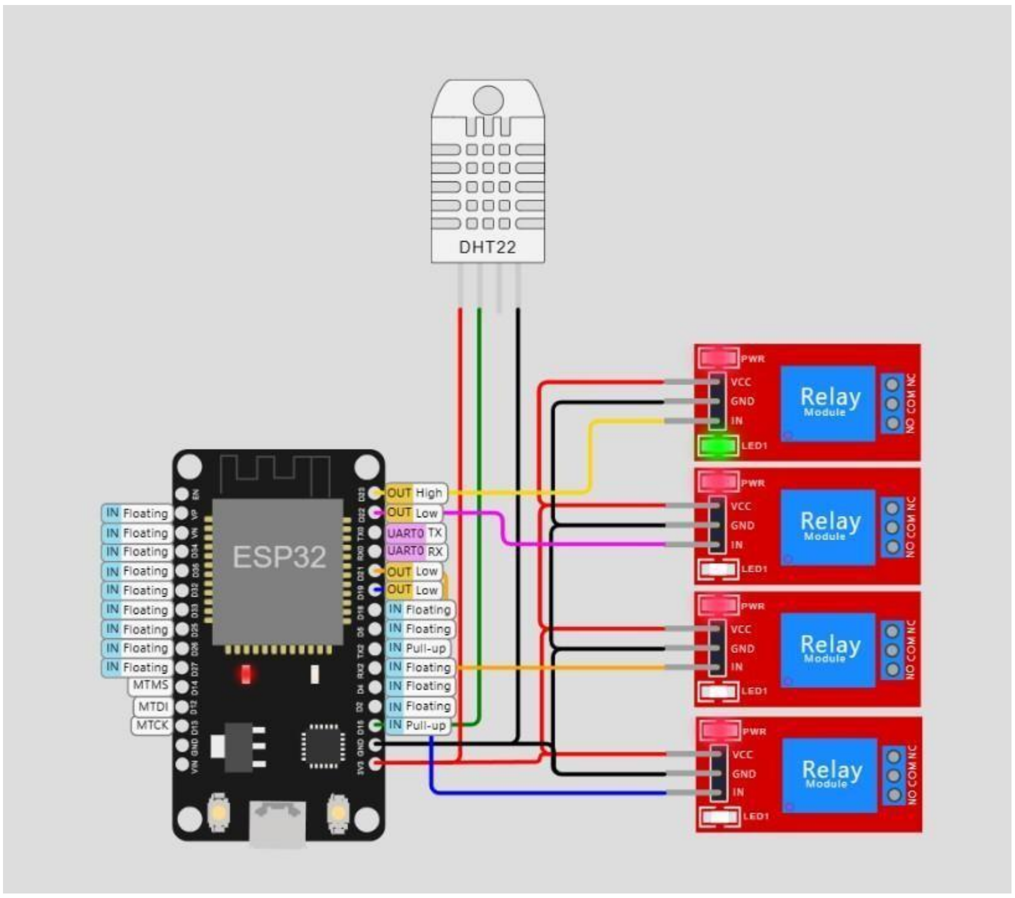
ALGORITHM

1. Initialize: Set up the NodeMCU board, connect the relays, and establish Wi-Fi connectivity.
2. Connect to Thingspeak: Establish a connection with the Thingspeak Cloud platform using HTTP or MQTT protocols.
3. User Interface: Create a user interface on the Wokwi simulation platform, allowing users to interact with their smart home devices.
4. Receive Commands: Listen for commands from the user interface on Wokwi, such as turning on/off specific appliances or adjusting settings.
5. Control Relays: Based on the received commands, trigger the corresponding relays to turn on/off the respective appliances or devices.
6. Send Status Updates: Send the current status of appliances (on/off) or other relevant data to the Thingspeak Cloud platform for logging and monitoring purposes.
7. Automation and Scheduling: Implement automation and scheduling capabilities, allowing users to define rules or schedules for their devices to automatically turn on/off based on specific conditions or time triggers.
8. Data Logging and Analysis: Use the Thingspeak Cloud platform to log data related to appliance usage, energy consumption, or device status. Analyze the data to generate insights and optimize energy usage.
9. Remote Monitoring: Enable users to remotely monitor the status of their appliances or devices through the user interface on Wokwi or via the
10. Thingspeak mobile application.
11. Notifications: Set up notifications through Thingspeak to alert users about
12. events or changes in their smart home system, such as appliances being left on for an extended period.
13. Expandability: Allow for the addition of more NodeMCU boards and relays to control additional appliances or devices in different areas of the home.

CODE -

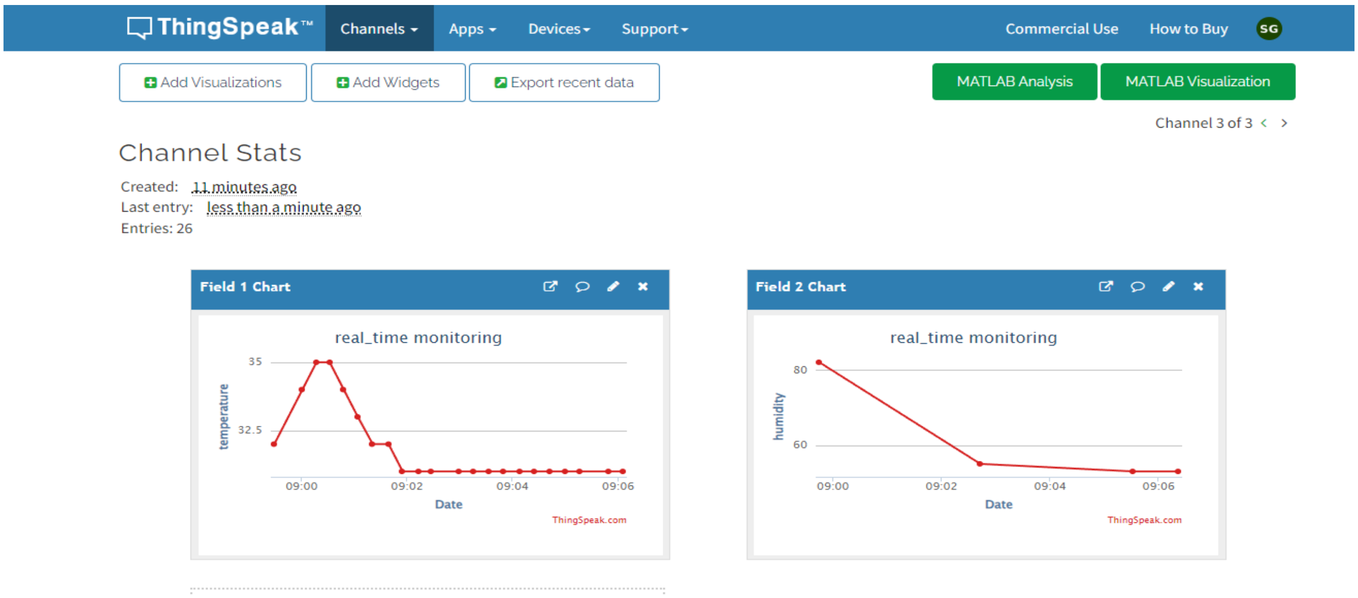
```
#include <WiFiNINA.h>
 #include <DHT.h>
 #include "secrets.h"
 #include "ThingSpeak.h" // always include thingspeak header file after other header files and
custom macros
 #define DHTPIN 8
 #define DHTTYPE DHT11
 DHT dht(DHTPIN, DHTTYPE);
 char ssid[] = "Parthiba";   // your network SSID (name)
 char pass[] = "abcd1234";     // your network password
 int keyIndex = 0;                 // your network key Index number (needed only for WEP)
 WiFiClient  client;
 uint8_t temperature, humidity;
 unsigned long myChannelNumber = 2206591;
 const char * myWriteAPIKey = "MDC7UA0KNX7O2G2V";



 void setup() {
   Serial.begin(115200);  // Initialize serial
   dht.begin();
   while (!Serial) {
         ; // wait for serial port to connect. Needed for Leonardo native USB port only
   }
   Serial.println();
   Serial.println();
   Serial.print("Connecting to ");
   Serial.println(ssid);
   // check for the WiFi module:
   if (WiFi.status() == WL_NO_MODULE) {
         Serial.println("Communication with WiFi module failed!");
         // don't continue
         while (true);
   }

   ThingSpeak.begin(client);  //Initialize ThingSpeak
 }



void loop() {
   // Connect or reconnect to WiFi
   if(WiFi.status() != WL_CONNECTED){
         Serial.print("Attempting to connect to SSID: ");
         Serial.println(ssid);
         while(WiFi.status() != WL_CONNECTED){
         WiFi.begin(ssid, pass); // Connect to WPA/WPA2 network. Change this line if
using open or WEP network
         Serial.print(".");
         delay(5000);
         }
         Serial.println("\nConnected.");
   }
 temperature = dht.readTemperature();
   humidity = dht.readHumidity();
   Serial.print("Temperature Value is :");
   Serial.print(temperature);
   Serial.println("C");
   Serial.print("Humidity Value is :");
   Serial.print(humidity);
   Serial.println("%");
   // Write to ThingSpeak. There are up to 8 fields in a channel, allowing you to store up to 8
different
   // pieces of information in a channel. Here, we write to field 1.
             ThingSpeak.writeField(myChannelNumber, 1, temperature, myWriteAPIKey);
             ThingSpeak.writeField(myChannelNumber, 2, humidity, myWriteAPIKey);
             // set the status
    delay(1000); // Wait 20 seconds to update the channel again
 }
```

CIRCUIT SCHEMATIC



OUTPUT

```
rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0030,len:1156
load:0x40078000,len:11456
ho 0 tail 12 room 4
load:0x40080400,len:2972
entry 0x400805dc
Attempting to connect to SSID: Wokwi
.WiFi Connected
Welcome at Smart Home
Problem reading channel. HTTP error code 400
weather   temp:  24.00    humidity:  40.00
Ch1: 0
Ch2: 0
Ch3: 0
Ch4: 0
Channel update successful.

Problem reading channel. HTTP error code 400
weather   temp:  71.40    humidity:  72.50
Ch1: 1
Ch2: 0
Ch3: 0
Ch4: 0
```



Channel Stats

Created: 11 minutes ago
Last entry: less than a minute ago
Entries: 26

Vellore Institute of Technology
Chennai Campus

Programme: BTech CSE with AI and ML
BECE351E - Internet of Things (IoT) Lab
Lab Sheet 9

Real-Time Environmental monitoring using Arduino Uno WiFi Rev 2 with DHT11 and ThingSpeak Cloud Computing

Name : Mainak Chattopadhyay Roll
Number: 21BAI1217
Date of the Lab Class: 15 June 2023

1. Introduction:

(a) The objective of this experiment is to perform environmental monitoring using Arduino WiFi Rev 2 and the DHT11 sensor, and then visualize the data on the Thingspeak platform. The goal is to collect temperature and humidity readings from the DHT11 sensor and transmit them wirelessly to a cloud-based platform for analysis and visualization.

(b) The required components to execute this experiment are:

- Arduino WiFi Rev 2 board: It provides wireless connectivity to send data to the internet.
- DHT11 sensor: It measures temperature and humidity levels in the surrounding environment.
- Jumper wires: Used to establish connections between the components.
- Breadboard: Provides a platform to assemble the circuit.
- USB cable: Connects the Arduino board to the computer for programming and power.

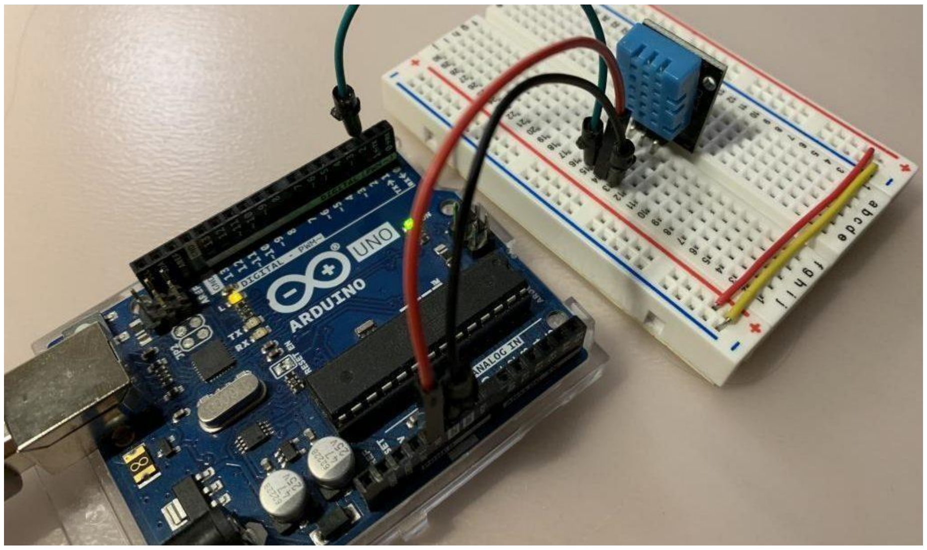
2. Circuit Diagram & Explanation:

(a) To design the circuit diagram, follow these steps:

1. Connect the VCC pin of the DHT11 sensor to the 5V pin on the Arduino.
2. Connect the GND pin of the DHT11 sensor to the GND pin on the Arduino.
3. Connect the DATA pin of the DHT11 sensor to any digital pin (e.g., pin 2) on the Arduino.
4. Connect the Arduino WiFi Rev 2 board to your computer using the USB cable.

(b) The circuit functions as follows:

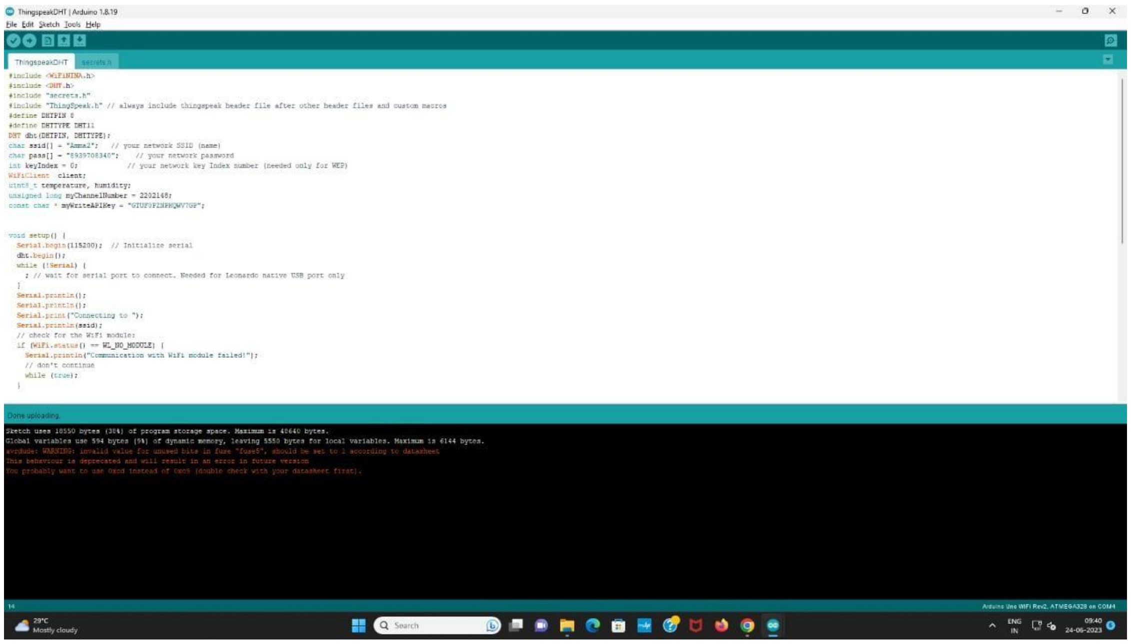
1. The DHT11 sensor measures temperature and humidity levels in the surroundings.
2. The Arduino WiFi Rev 2 board reads the data from the DHT11 sensor through the digital pin.
3. The Arduino board establishes a wireless connection to the internet using its built-in WiFi module.
4. The collected temperature and humidity data are sent to the Thingspeak platform via HTTP requests over the internet.



(c) Here's an example code snippet that can be utilized to run the circuit:

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

#define DHTPIN 2
#define DHTTYPE DHT11

char ssid[] = "YourWiFiSSID";       // Replace with your network credentials
char password[] = "YourWiFiPassword";
char server[] = "api.thingspeak.com";

WiFiClient client;
DHT dht(DHTPIN, DHTTYPE);

void setup() {
  Serial.begin(9600);
  delay(10);

  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
}

void loop() {
  float humidity = dht.readHumidity();
  float temperature = dht.readTemperature();

  if (isnan(humidity) || isnan(temperature)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }

  if (client.connect(server, 80)) {
    String postStr = "field1=" + String(temperature) + "&field2=" +
String(humidity);
    client.print("POST /update HTTP/1.1\n");
    client.print("Host: " + String(server) + "\n");
    client.print("Connection: close\n");
    client.print("X-THINGSPEAKAPIKEY: YOUR_API_KEY\n");
    client.print("Content-Type: application/x-www-form-urlencoded\n");
    client.print("Content-Length: ");
    client.print(postStr.length());
    client.print("\n\n");
    client.print(postStr);

    Serial.println("Data sent to Thingspeak");
  }

  client.stop();
  delay(30000); // Send data every 30 seconds
}
```

Replace "YourWiFiSSID" and "YourWiFiPassword" with your own WiFi network credentials. Also, replace "YOUR_API_KEY" with the API key provided by Thingspeak.



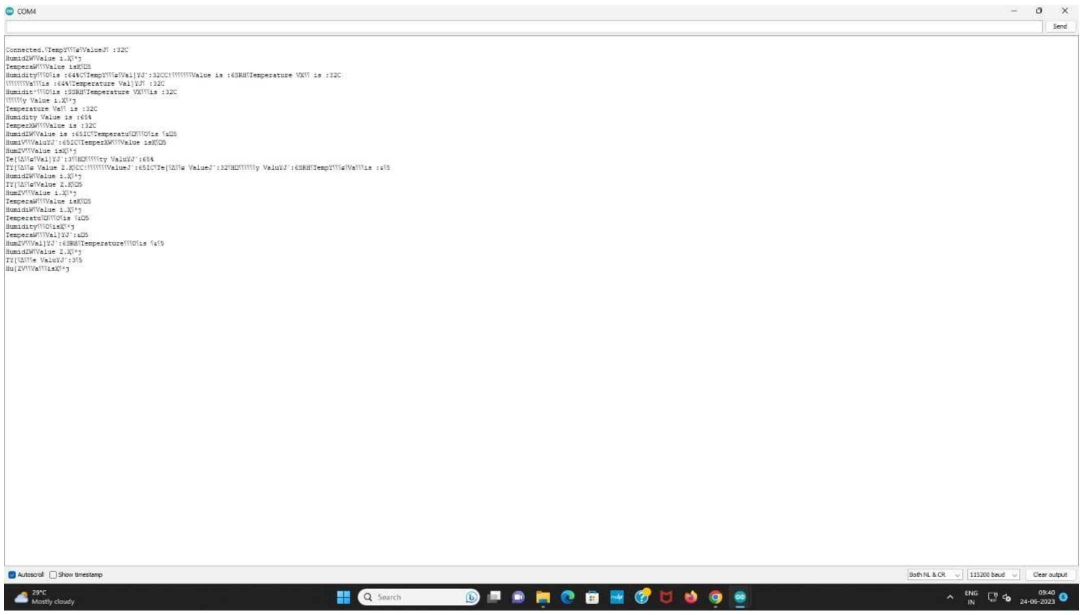
```
#include <WiFiNINA.h>
#include <DHT.h>
#include "secrets.h"
#include "ThingSpeak.h" // always include thingspeak header file after other header files and custom macros
#define DHTPIN 8
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);
char ssid[] = "Amma2";   // your network SSID (name)
char pass[] = "6939708340";    // your network password
int keyIndex = 0;            // your network key Index number (needed only for WEP)
WiFiClient  client;
uint8_t temperature, humidity;
unsigned long myChannelNumber = 2202148;
const char * myWriteAPIKey = "GTUF0PZNPHQWV7GP";


void setup() {
  Serial.begin(115200);  // Initialize serial
  dht.begin();
  while (!Serial) {
    ; // wait for serial port to connect. Needed for Leonardo native USB port only
  }
  Serial.println();
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);
  // check for the WiFi module:
  if (WiFi.status() == WL_NO_MODULE) {
    Serial.println("Communication with WiFi module failed!");
    // don't continue
    while (true);
  }
```

```
Done uploading.
Sketch uses 18550 bytes (38%) of program storage space. Maximum is 48640 bytes.
Global variables use 594 bytes (9%) of dynamic memory, leaving 5550 bytes for local variables. Maximum is 6144 bytes.
avrdude: WARNING: invalid value for unused bits in fuse "fuse5", should be set to 1 according to datasheet
This behaviour is deprecated and will result in an error in future version
You probably want to use 0xcd instead of 0xc9 (double check with your datasheet first).
```

3. Output of the experiment:

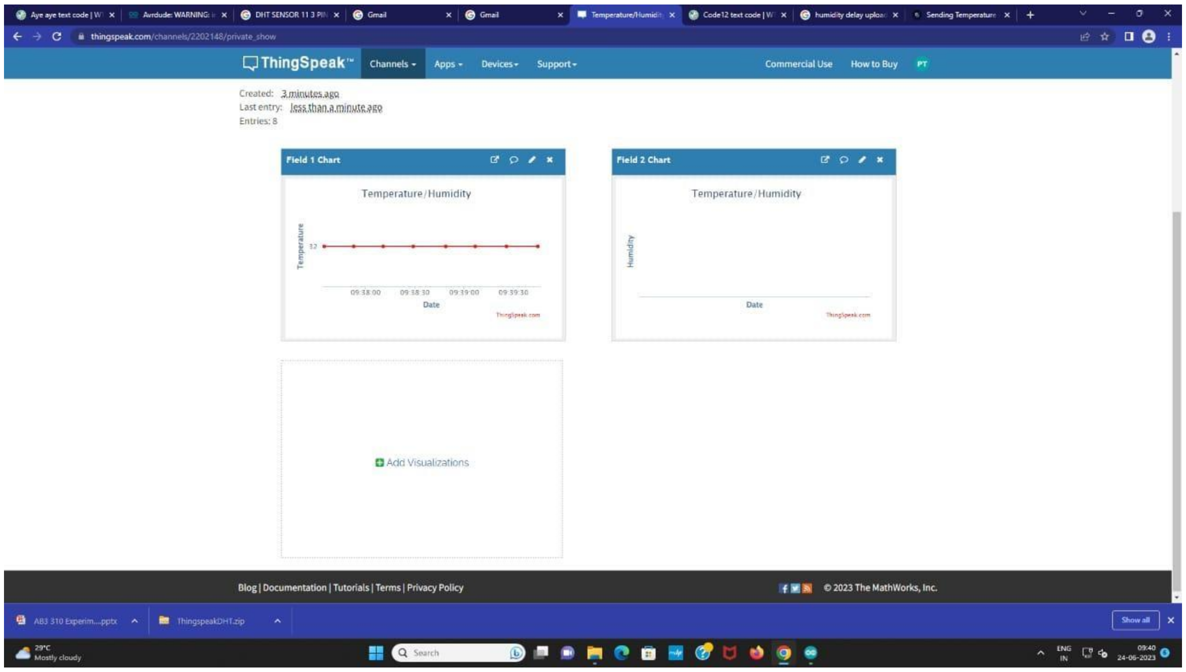
(a) To run the simulation, follow these steps:

1. Upload the code to the Arduino WiFi Rev 2 board.
2. Make sure the board is connected to your computer via USB.
3. Open the Serial Monitor in the Arduino IDE to view the output.



```
Connected.TempY≈ValueJ :32C
HumidZWValue i.X*j
TemperaWValue isK05
HumidityOis :64%CTempY≈Val]YJ':32CC!Value is :65RHTemperature VX is :32C
Vais :64%Temperature Val]YJ :32C
Humidit'Ois :55RHTemperature VXis :32C
y Value i.X*j
Temperature Va is :32C
Humidity Value is :65%
TemperXWValue is :32C
HumidZWValue is :65ICTemperatuOis ≈05
HumiVValuYJ':65ICTemperXWValue isK05
HumZVValue isX*j
Te[A≈Val]YJ':3HDty ValuYJ':65%
TY[A≈ Value Z.KCC!ValueJ':65ICTe[A≈ ValueJ':32HDy ValuYJ':65RHTempY≈Vais :≈5
HumidZWValue i.X*j
TY[A≈Value Z.K05
HumZVValue i.X*j
TemperaWValue isK05
HumidiWValue i.X*j
TemperatuOis ≈05
HumidityOisX*j
TemperaWVal]YJ':≈05
HumZVVal]YJ':65RHTemperatureOis ≈5
HumidZWValue Z.X*j
TY[Ae ValuYJ':35
Hu[ZVVaisX*j
```

(b) The output is assessed by observing the data displayed on the Thingspeak platform. After successfully running the circuit and sending data to Thingspeak, you can log in to your Thingspeak account and navigate to the specific channel associated with your experiment. There, you will find temperature and humidity readings being recorded in real-time and displayed as graphs or numerical data, depending on your preferred visualization settings.



Created: 3 minutes ago
Last entry: less than a minute ago
Entries: 8

BECE351E – INTERNET OF THINGS

EXPT NO: 10

DATE: June 28, 2023

NAME: Mainak Chattopadhyay

REG. NO.: 21BAI1217

Real-Time Soil moisture monitoring using Arduino Uno WiFi Rev 2 and ThingSpeak CloudComputing (Hardware and Software)

Aim

The goal of this experiment is to run Real-Time Soil moisture monitoring such as using an Arduino Uno Wifi Rev 2 microcontroller, a DHT22 sensor, and sending the data to the cloud platform ThingSpeak for visualization and analysis.

Apparatus/Components Required

1. Thingspeak
2. System
3. 1 x Arduino UNO board WIFI Rev 2.
4. 8 x Jump wires
5. 1 x Power Supply
6. 1 x Soil Moisture Sensor

Theory

Real-time soil moisture monitoring using Arduino Uno WiFi Rev 2 and ThingSpeak Cloud Computing offers several advantages in terms of bothhardware and software:

- **Hardware Advantages:**

1. Arduino Uno WiFi Rev 2: The Arduino Uno WiFi Rev 2 board is a versatile microcontroller with built-in Wi-Fi capabilities. It combines the functionality of Arduino Uno with wireless connectivity, eliminating the need for additional Wi-Fi modules orshields.

2. Wireless Connectivity: The Wi-Fi capability of the Arduino Uno WiFi Rev 2 enables wireless communication between the sensornodes and the cloud platform. This eliminates the need for physical connections, making the system more flexible and scalable.

3. Cost-effective Solution: Arduino Uno WiFi Rev 2 is an affordableand widely available microcontroller board. It offers value for money without compromising on features and performance,

   a. making it a cost-effective choice for soil moisture monitoringprojects.

4. Easy Integration: Arduino boards, including the Uno WiFi Rev 2, have a large community support and extensive documentation. This makes it easy to find resources, libraries, and example codesto integrate sensors and develop the required functionality.

- **Software Advantages:**

1. ThingSpeak Cloud Platform: ThingSpeak is a cloud-based IoTplatform specifically designed for collecting, analyzing, and visualizing data from IoT devices. It provides an intuitive webinterface that allows users to easily manage their data and perform real-time analytics.
2. Real-Time Data Monitoring: With ThingSpeak, you can monitor soil moisture levels in real time from anywhere with an internet connection. The data from Arduino Uno WiFi Rev 2 is uploaded toThingSpeak, where it can be viewed through customizable dashboards and charts.
3. Data Visualization and Analysis: ThingSpeak provides powerful data visualization tools that enable you to create custom charts, graphs, and visualizations based on your soil moisture data. Youcan gain insights
4. into trends, patterns, and correlations, helping you make informeddecisions regarding irrigation and plant health.
5. Alerts and Notifications: ThingSpeak allows you to set up alerts based on predefined thresholds or conditions. You can receivenotifications via email, SMS, or other channels when the soil
6. moisture levels go beyond the desired range. This enables timelyintervention and prevents damage to crops due to under or
7. overwatering.
8. Compatibility with MATLAB: ThingSpeak has seamless integration with MATLAB, a powerful data analysis and modeling tool. You canuse MATLAB Analytics on ThingSpeak to perform advanced analytics, apply machine learning algorithms, and develop
9. predictive models for better understanding and management ofsoil moisture data.

Algorithm

1. Initialize the Arduino Uno WiFi Rev 2 board and the soil moisture sensor.
2. Establish a Wi-Fi connection to your local network using the built-in Wi-Fi capabilities of Arduino Uno WiFi Rev 2.
3. Connect to the ThingSpeak cloud platform by providing the appropriate APIkey and server information.
4. Set up a loop to continuously perform the following steps:
5. Read the soil moisture level from the sensor connected to the Arduino UnoWiFi Rev 2 board.
6. Convert the raw sensor reading into meaningful moisture data.
7. Upload the moisture data to the designated field in the ThingSpeak channelusing the established Wi-Fi connection.
8. Optional: Implement any necessary error handling or data validation.
9. Delay for a specific interval to control the frequency of data updates (e.g.,every few minutes).
10. Monitor the serial output or debug logs of the Arduino Uno WiFi Rev 2 board to ensure the sensor readings and Wi-Fi connectivity are functioningcorrectly.
11. Access the ThingSpeak platform to view the real-time soil moisture databeing sent from the Arduino board.
12. Configure visualizations, charts, alerts, or other analytics within the ThingSpeak platform to derive insights from the collected data.
13. Optionally, implement additional features such as setting threshold alerts, integrating with external systems, or controlling irrigation based on the moisture data.

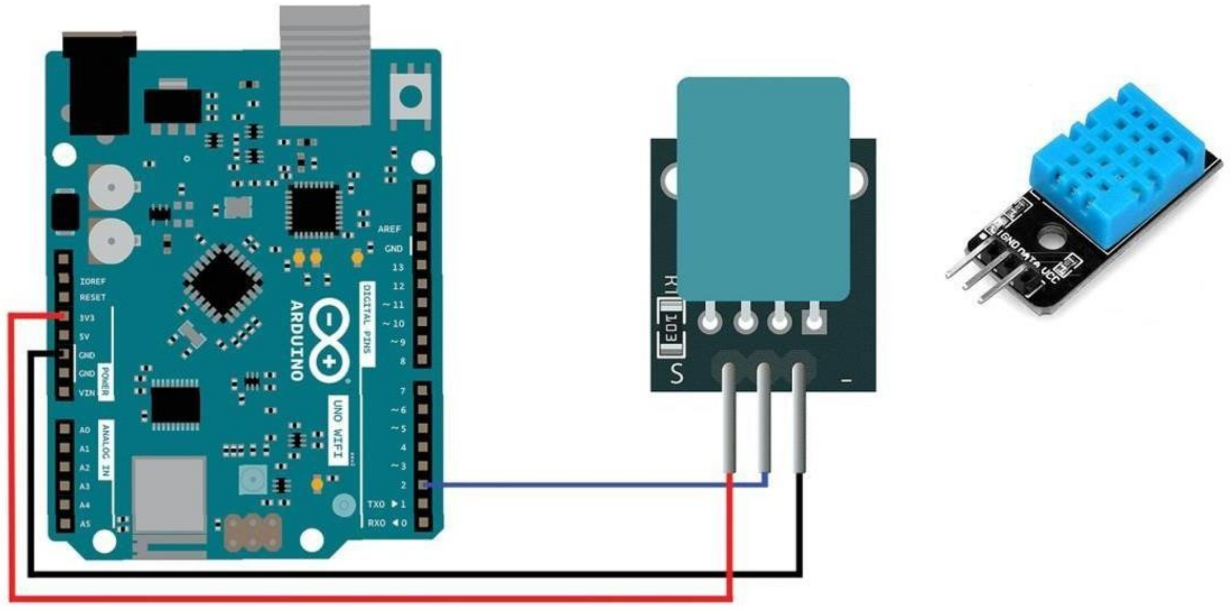
Program/Serial Output

```
#include
<WiFiNINA.h>
#include
"secrets.h"

#include "ThingSpeak.h" // always include thingspeak header file after other header files and
custommacros

int moisture_data = 0;


char ssid[] = "Dp"; // your network SSID (name)

char pass[] = "Dp090919"; // your network password

int keyIndex = 0; // your network key Index number (needed only
for WEP)WiFiClient client;
```

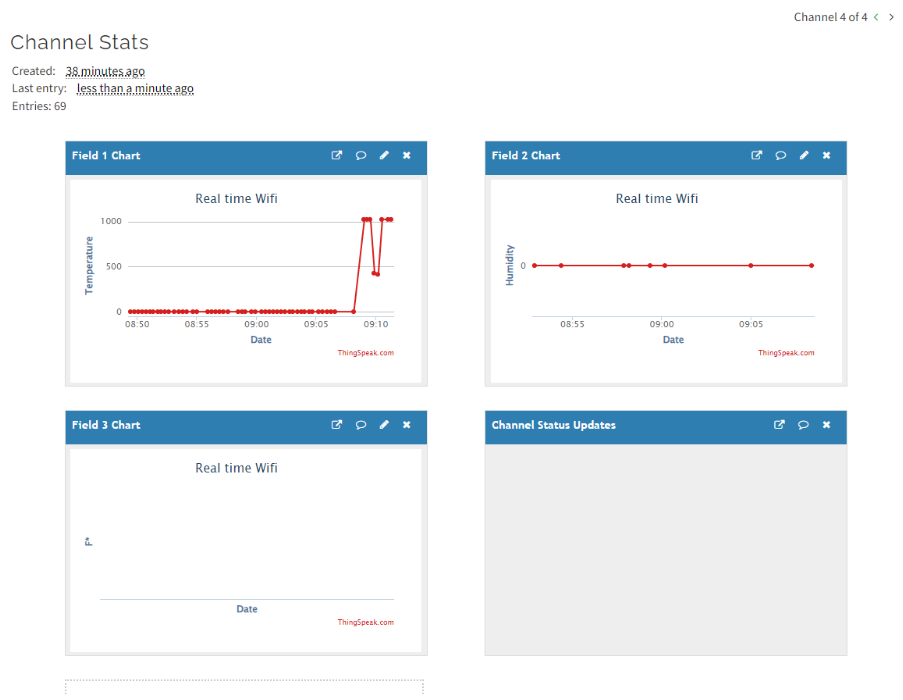
```
unsigned long myChannelNumber = 2206557;

const char * myWriteAPIKey = "ATKY2Z0UVK4RHZYJ";


void setup() {

Serial.begin(115200); //
Initialize serialpinMode(A0,
INPUT);


while (!Serial) {

; // wait for serial port to connect. Needed for Leonardo native USB port only

}

Serial.println();
Serial.println();
Serial.print("Connectin
g to ");
Serial.println(ssid);

// check for the WiFi module:

if (WiFi.status() == WL_NO_MODULE) {
Serial.println("Communication with WiFi module
failed!");

// don't
continue
while
(true);

}


ThingSpeak.begin(client); //Initialize ThingSpeak

}

void loop() {

// Connect or reconnect to WiFi
if(WiFi.status() !=
WL_CONNECTED){
Serial.print("Attempting to connect to
SSID: ");Serial.println(ssid);

while(WiFi.status() != WL_CONNECTED){
```

```
WiFi.begin(ssid, pass); // Connect to WPA/WPA2 network. Change this line if using open
or WEPnetwork

Serial.pri
nt(".");
delay(50
00);

}

Serial.println("\nConnected.");


}


moisture_data = analogRead(A0);


Serial.print("Soil Moisture data
Value is :");
Serial.print(moisture_data);


// Write to ThingSpeak. There are up to 8 fields in a channel, allowing you to store up to 8
different

// pieces of information in a channel. Here, we write to field 1.
ThingSpeak.writeField(myChannelNumber, 1, moisture_data, myWriteAPIKey);


// set the status

delay(1000); // Wait 20 seconds to update the channel again

}
```

Circuit schematic



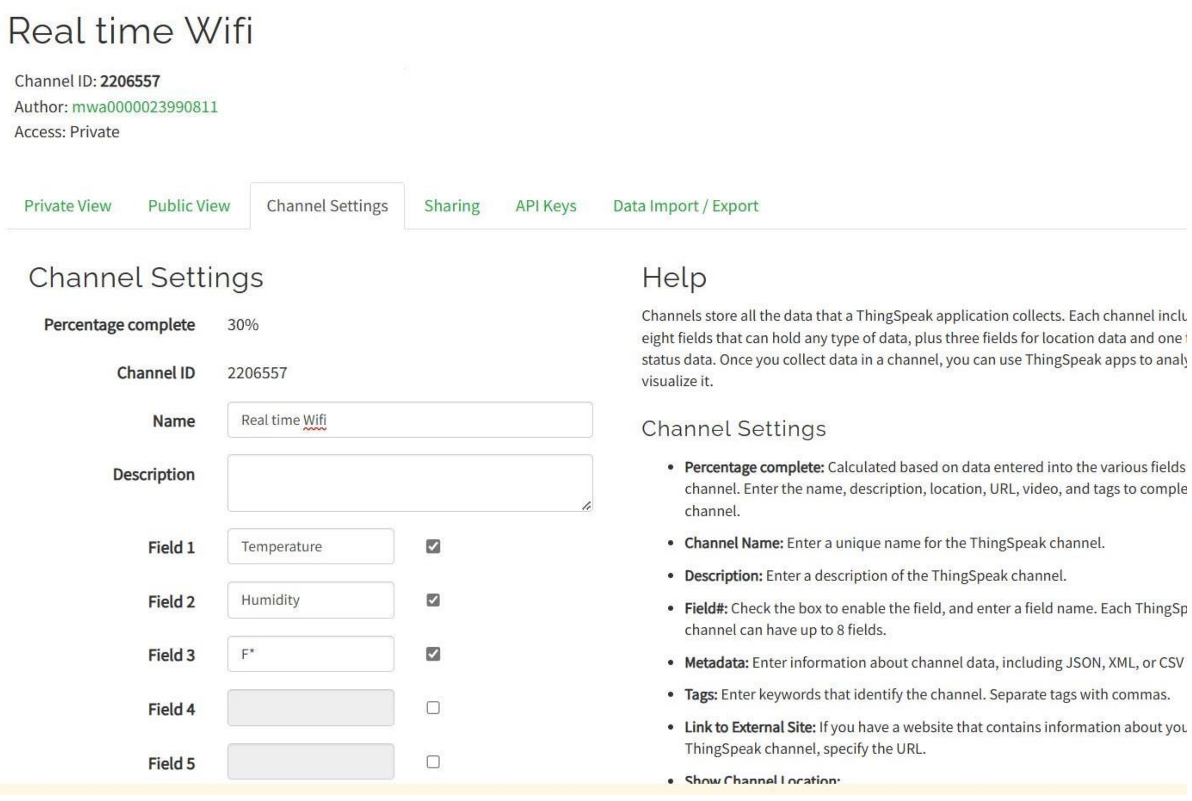
OUTPUT



Channel 4 of 4

Channel Stats

Created: 38 minutes ago
Last entry: less than a minute ago
Entries: 69



Real time Wifi

Channel ID: **2206557**
Author: mwa0000023990811
Access: Private

Channel Settings

Percentage complete	30%	
Channel ID	2206557	
Name	Real time Wifi	
Description		
Field 1	Temperature	✓
Field 2	Humidity	✓
Field 3	F*	✓
Field 4		☐
Field 5		☐

Help

Channels store all the data that a ThingSpeak application collects. Each channel inclu eight fields that can hold any type of data, plus three fields for location data and one status data. Once you collect data in a channel, you can use ThingSpeak apps to analy visualize it.

Channel Settings

- **Percentage complete:** Calculated based on data entered into the various fields channel. Enter the name, description, location, URL, video, and tags to comple channel.
- **Channel Name:** Enter a unique name for the ThingSpeak channel.
- **Description:** Enter a description of the ThingSpeak channel.
- **Field#:** Check the box to enable the field, and enter a field name. Each ThingSp channel can have up to 8 fields.
- **Metadata:** Enter information about channel data, including JSON, XML, or CSV
- **Tags:** Enter keywords that identify the channel. Separate tags with commas.
- **Link to External Site:** If you have a website that contains information about you ThingSpeak channel, specify the URL.
- **Show Channel Location:**

Result

Real-time soil moisture monitoring using Arduino Uno WiFi Rev 2 and ThingSpeakCloud Computing is a viable solution for tracking and analyzing soil moisture

levels. It combines the capabilities of the Arduino Uno WiFi Rev 2 board with thecloud-based data storage and visualization provided by ThingSpeak.

The hardware components required for this setup include the Arduino Uno WiFiRev 2 board and a soil moisture sensor. The built-in Wi-Fi capabilities of the

Arduino Uno WiFi Rev 2 enable easy connectivity to your local network.

The software side involves initializing the Arduino board, establishing a Wi-Fi connection, and connecting to the ThingSpeak platform using the provided API key and server information. A continuous loop reads the soil moisture level from the sensor, converts the raw sensor reading into meaningful data, and uploads it to the designated field in the ThingSpeak channel through the Wi-Fi connection.You can also implement error handling, data validation, and control the update frequency.

By monitoring the serial output or debug logs, you can ensure that the systemis functioning correctly. Accessing the ThingSpeak platform allows you to view the real-time soil moisture data sent from the Arduino board. ThingSpeak

provides various features for data visualization, analytics, and alert generation,enabling you to derive insights from the collected data.

Additionally, you have the flexibility to customize and enhance the system according to your specific needs. This includes configuring visualizations, settingthreshold alerts, integrating with external systems, or controlling irrigation basedon the moisture data.

Overall, the combination of Arduino Uno WiFi Rev 2 and ThingSpeak Cloud Computing offers a practical and scalable solution for real-time soil moisturemonitoring, enabling efficient management of agricultural or gardening applications.