OPEN IOT HARDWARE
EMBEDDED PROJECTS REPORT

Submitted by : ROHITH. S

Intern - IoT HARDWARE

1. LED Array Patterns

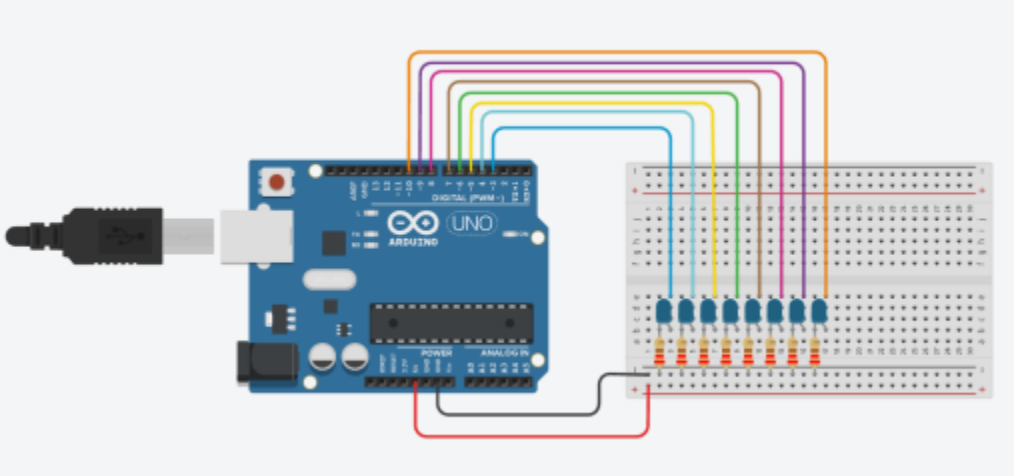
AIM

Interface an LED Array to the Arduino UNO board and obtain different LED Array patterns.

COMPONENTS REQUIRED

1. Arduino Uno
2. Breadboard
3. Jumper wires
4. LEDs
5. 330 Ohm resistors

CIRCUIT DIAGRAM



CODE

```
// Array of pin numbers for LEDs
const int k[] = {33, 35, 37, 39, 41, 43, 45, 47};

// Function to blink all LEDs
void blinkall() {
    for (int i = 0; i < 8; i++) {
        digitalWrite(k[i], HIGH); // Turn on LED
        delay(250);               // Wait for 250 milliseconds
    }
    for (int i = 0; i < 8; i++) {
        digitalWrite(k[i], LOW); // Turn off LED
        delay(250);              // Wait for 250 milliseconds
    }
}

// Function to alternate blinking of LEDs
void altblink() {
    // Blink even-indexed LEDs
    for (int i = 0; i < 8; i += 2) {
        digitalWrite(k[i], HIGH); // Turn on LED
        delay(250);               // Wait for 250 milliseconds
        digitalWrite(k[i], LOW);  // Turn off LED
    }
    delay(500); // Wait before the next sequence

    // Blink odd-indexed LEDs
    for (int i = 1; i < 8; i += 2) {
        digitalWrite(k[i], HIGH); // Turn on LED
        delay(250);               // Wait for 250 milliseconds
        digitalWrite(k[i], LOW);  // Turn off LED
    }
    delay(500); // Wait before the next sequence
}

void setup() {
    // Initialize pins as OUTPUT
    for (int i = 3; i < 10; i++) {
        pinMode(i, OUTPUT);
    }
}
```

```
void loop() {
    blinkall();   // Call function to blink all LEDs
    delay(2000);  // Wait for 2 seconds before next action
    altblink();   // Call function to alternate blinking of LEDs
    delay(2000);  // Wait for 2 seconds before repeating the loop
}
```

RESULT

Obtained different array patterns.

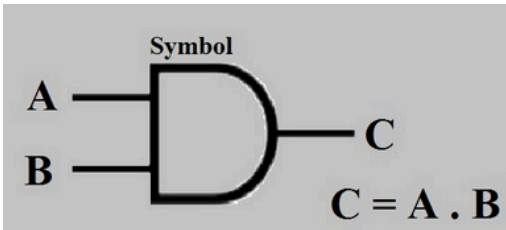
2. Logic Gates

AIM

To simulate the behavior of an AND logic gate.

DESCRIPTION

It turns a light on or off based on one or two inputs, effectively showing the truth table for a given logic gate. A logic gate is a device performing a Boolean logic operation on one or more binary inputs and then outputs a single binary output. In a circuit, logic gates work based on a combination of digital signals coming from their inputs. Most logic gates have two inputs and one output, and they are based on Boolean algebra. At any given moment, every terminal is in one of the two binary conditions: true or false. False represents 0, and true represents 1. There are seven basic logic gates: AND, OR, XOR, NOT, NAND, NOR, and XNOR. The AND gate is named so because if 0 is false and 1 is true, the gate acts in the same way as the logical "and" operator.



Truth Table

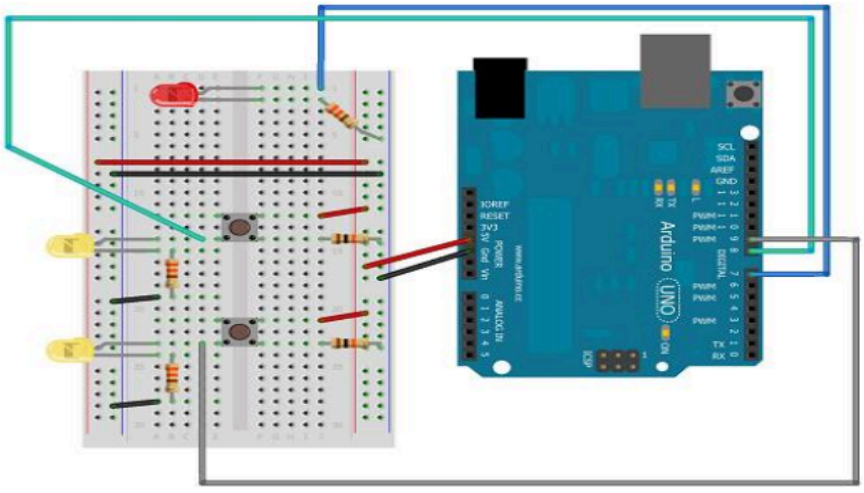
INPUT		OUTPUT
A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

COMPONENTS REQUIRED:

1. Arduino Board
2. 3 x LED (2 Green, 1 Red)
3. 2 x 330-ohm resistors
4. 10k ohm resistor
5. 2 push buttons
6. Jumper wires

CIRCUIT DIAGRAM

The push buttons are connected with 10K resistors to GND. An LED is placed on the same row as the connections to the input pins (8, 9). This gives us a push-to-make configuration for the push buttons. It also means that we don't have to write any code to make the two input indicator LEDs light up. You can test the button connections early on. The Yellow LEDs in this diagram should light when the buttons are pressed and be off when not.



CODE :

```
// Pin Definitions
int pinOut = 49; // Output pin
int pinA = 41;   // Input pin A
int pinB = 37;   // Input pin B


void setup() {
    pinMode(pinOut, OUTPUT); // Set output pin mode
    pinMode(pinA, INPUT);    // Set input pin A mode
    pinMode(pinB, INPUT);    // Set input pin B mode
}
void loop() {
    boolean pinAState = digitalRead(pinA); // Read state of input pin A
    boolean pinBState = digitalRead(pinB); // Read state of input pin B
    boolean pinOutState;                   // Variable to store output state


    pinOutState = pinAState & pinBState;    // Perform AND operation on the
states of pin A and B
    digitalWrite(pinOut, pinOutState);      // Write the result to the output
pin
}
```

RESULT

The output is obtained from an AND gate integrated with an arduino.

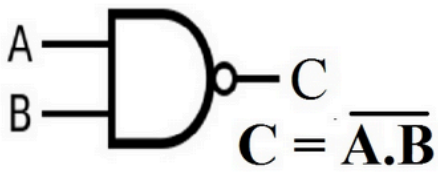
3. NAND GATE

AIM

Implementing a NAND gate using an Arduino Uno.

DESCRIPTION

A NAND gate is a fundamental logic gate in digital electronics that performs the logical operation of negation and conjunction. In this setup, we will utilize the Arduino Uno along with LEDs and push buttons to physically demonstrate the functionality of a NAND gate. The project aims to provide a practical understanding of logic gates and microcontroller interfacing, offering a hands-on learning experience for enthusiasts and beginners in electronics and programming. By following this guide, readers will learn how to wire the circuit, program the Arduino, and observe the behavior of the NAND gate in action.



Truth Table

INPUT		OUTPUT
A	B	A NAND B
0	0	1
0	1	1
1	0	1
1	1	0

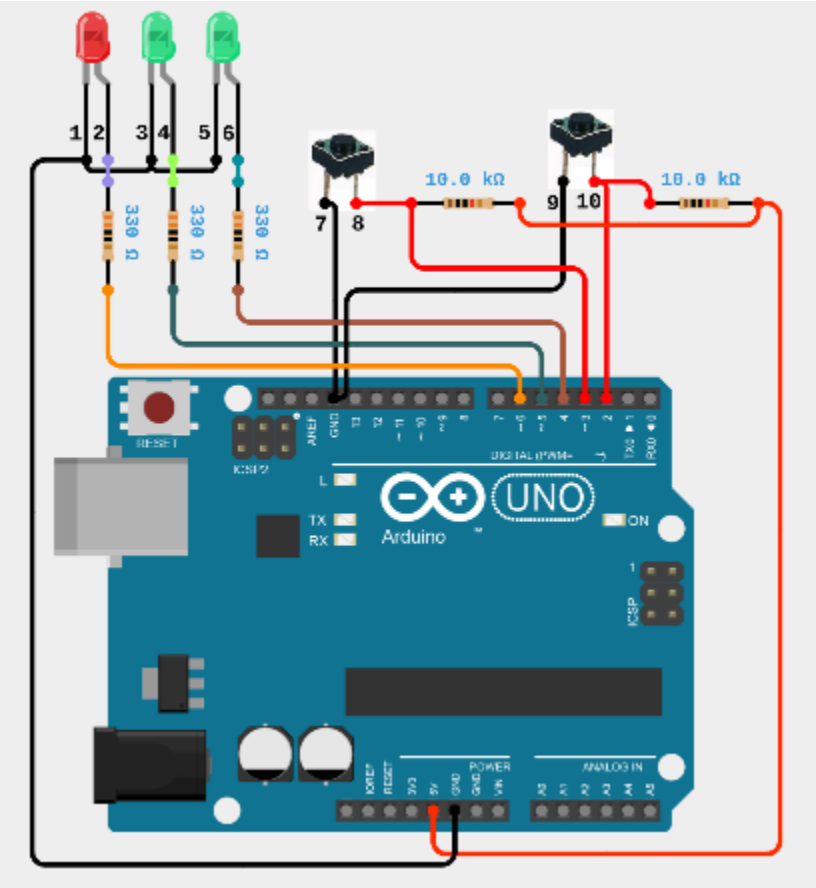
COMPONENTS REQUIRED:

1. Arduino Board

2. 3 x LED (2 Green, 1 Red)
3. 2 x 330-ohm resistors
4. 10k ohm resistor
5. 2 push buttons
6. Jumper wires

CIRCUIT DIAGRAM

The circuit diagram for implementing a NAND gate using Arduino Uno is essential for visualizing how to connect the components effectively.



CODE

```
// Pin Definitions
int b1 = 49; // Button 1 pin
int b2 = 47; // Button 2 pin
int val1 = 0; // Variable to store the state of button 1
int val2 = 0; // Variable to store the state of button 2

int led1 = 33; // LED 1 pin
int led2 = 31; // LED 2 pin
int led3 = 22; // LED 3 pin

bool s1 = false; // State of button 1
bool s2 = false; // State of button 2
bool result = false; // Result of NAND operation

// Function to perform NAND operation
bool nand(bool input1, bool input2) {
    return !(input1 && input2); // Return NAND result
}

void setup() {
    pinMode(b1, INPUT);   // Set button 1 pin as input
    pinMode(b2, INPUT);   // Set button 2 pin as input
    pinMode(led1, OUTPUT); // Set LED 1 pin as output
    pinMode(led2, OUTPUT); // Set LED 2 pin as output
    pinMode(led3, OUTPUT); // Set LED 3 pin as output
}

void loop() {
    val1 = digitalRead(b1); // Read the state of button 1
    delay(150);              // Debounce delay

    if (val1 == 0) {
        s1 = !s1;           // Toggle state of s1 if button 1 is pressed
    }

    val2 = digitalRead(b2); // Read the state of button 2
    delay(150);              // Debounce delay

    if (val2 == 0) {
        s2 = !s2;           // Toggle state of s2 if button 2 is pressed
    }
```

```
    // Control LED based on the state of s1
    if (s1) {
        digitalWrite(led1, HIGH); // Turn on LED 1 if s1 is true
    } else {
        digitalWrite(led1, LOW);  // Turn off LED 1 if s1 is false
    }

    // Control LED based on the state of s2
    if (s2) {
        digitalWrite(led2, HIGH); // Turn on LED 2 if s2 is true
    } else {
        digitalWrite(led2, LOW);  // Turn off LED 2 if s2 is false
    }

    result = nand(s1, s2);        // Calculate NAND result based on s1 and s2

    if (result) {
        digitalWrite(led3, HIGH); // Turn on LED 3 if result is true (NAND)
    } else {
        digitalWrite(led3, LOW);  // Turn off LED 3 if result is false (NAND)
    }
}
```

RESULT

The output is obtained from a NAND gate integrated with an arduino.

4. Reed Switch

AIM

To obtain the Interrupt function of an Arduino Uno board.

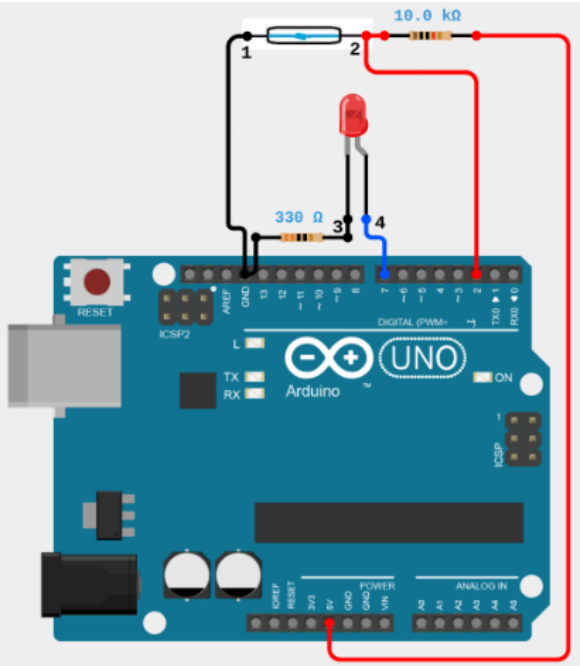
DESCRIPTION

This setup illustrates the integration of a reed switch with an Arduino board. A reed switch is an electric switch actuated by magnetism. Inside the reed switch, there are two thin metal strips (called reeds) that come together to complete an electrical circuit when a magnet is nearby. When the magnet moves away, the metal strips separate, breaking the circuit and turning the switch off. This transition controls the LED state, turning it on when the magnetic field is present and off when it's absent.

COMPONENTS REQUIRED

1. Arduino UNO
2. Reed switch
3. Resistors (330 ohm)
4. LED

CIRCUIT DIAGRAM



CODE

```
// Pin Definitions
int reedSwitch = 53; // Pin connected to the reed switch
int LED = 51;        // Pin connected to the LED

// Interrupt Service Routine for turning on the LED
void isr1() {
    digitalWrite(LED, HIGH); // Turn on the LED
}

// Interrupt Service Routine for turning off the LED
void isr2() {
    digitalWrite(LED, LOW);  // Turn off the LED
}

void setup() {
    pinMode(reedSwitch, INPUT); // Set reed switch pin as input
    pinMode(LED, OUTPUT);       // Set LED pin as output
```

```
}

void loop() {
    int s = digitalRead(reedSwitch); // Read the state of the reed switch
    attachInterrupt(s, isr1, FALLING); // Attach interrupt for falling edge
    attachInterrupt(s, isr2, RISING);  // Attach interrupt for rising edge
}
```

RESULT

The implemented circuit utilizing the reed switch and LED demonstrated the expected behavior using interrupts. When a magnetic field approached and triggered the reed switch to change its state from closed to open, the LED turned on. Conversely, when the magnetic field was removed and the reed switch returned to its closed state, the LED turned off. This behavior persisted reliably, demonstrating the successful integration of the reed switch with the Arduino Uno for magnetic field detection.

5. Servo Motor

Aim

Integrate a servo motor with an Arduino board.

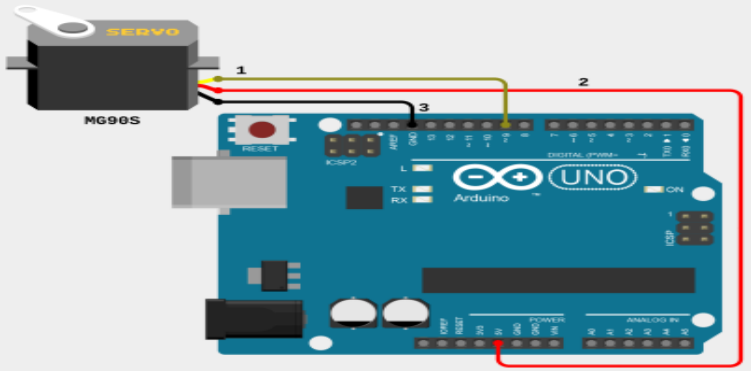
Description

The objective is to control the servo motor's position, enabling it to move from 0 to 180 degrees and back repeatedly, with a delay of 15 milliseconds between movements. Servo motors are widely used in various applications requiring precise control over angular position, making them ideal for robotics, automation, and other motion-related projects. For servo motors, PWM (Pulse Width Modulation) pins are utilized, and the 'Servo' library manages PWM signal generation internally. When using the 'Servo' library, you can attach the servo to any digital pin, and the library will generate the necessary PWM signal.

Components Required

1. Arduino Uno
2. MG90S Servo Motor

Circuit Diagram



Library

- Servo library

Steps

1. Open Arduino IDE.
2. Go to Sketch.
3. Select Include Library.
4. Select Manage Libraries.
5. Install Servo library.

Code

```
#include <Servo.h>
int spin = 6; // Pin connected to the servo
int ppin = 3; // Pin connected to the button
int count = 0; // Counter for rotations
Servo servo; // Create servo object

int lastButtonState = LOW; // Previous button state
int angle = 0; // Current angle of the servo
void setup() {
    Serial.begin(9600); // Initialize serial communication
    servo.attach(spin); // Attach the servo to pin 'spin'
    pinMode(ppin, INPUT); // Set button pin as input
    pinMode(spin, OUTPUT); // Set servo pin as output}

void loop() {
    int state = digitalRead(ppin); // Read button state
    delay(15); // Short delay for stability

    if (state == HIGH && lastButtonState == LOW) {
        angle += 90; // Increment angle by 90 degrees
        Serial.println("Rotation of 90 degrees");
```

```
        servo.write(angle); // Move servo to new angle
        count++;
        delay(15);}

    if (count == 2) {
        angle += 180; // Increment angle by another 180 degrees
        Serial.println("Rotation of another 90 degrees");
        servo.write(angle); // Move servo to new angle
        count = 0;
        delay(15);
    } else {
        Serial.println("No rotation");
    }

    lastButtonState = state; // Update last button state
    delay(1000); // Delay before next loop iteration
}
```

Result

The servo motor smoothly rotates from 0 to 180 degrees and then back to 0 degrees with a delay of 15 milliseconds between movements, creating a continuous back-and-forth motion.

6. Light Dependent Resistor

Aim

Integrate an LDR sensor with an Arduino Uno board and observe the light intensity.

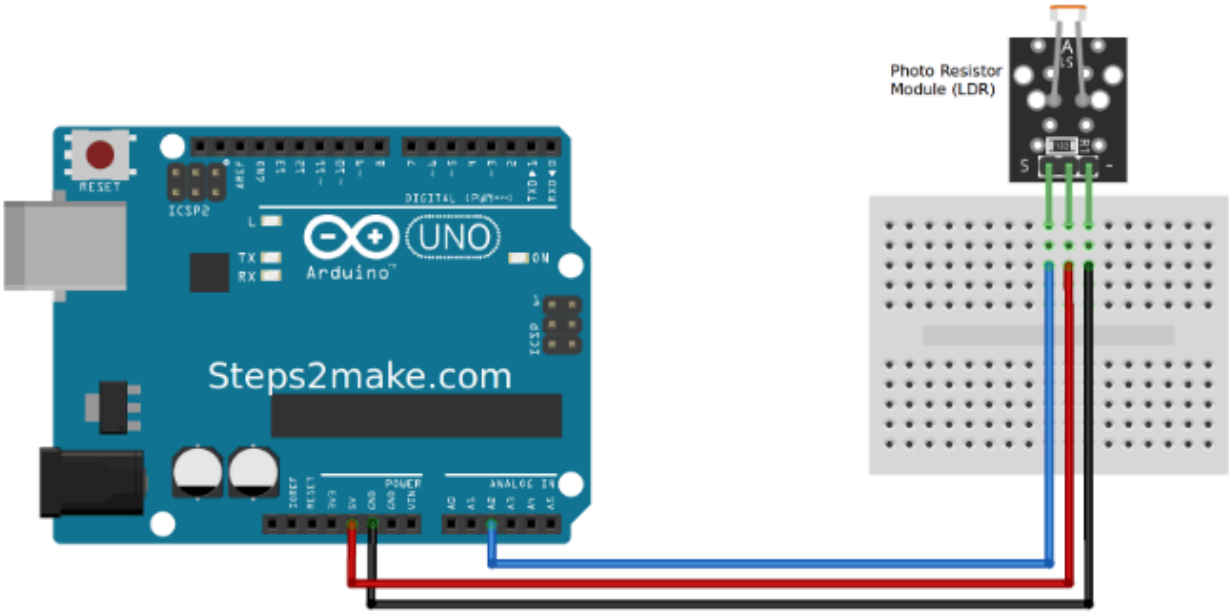
Description

A Light Dependent Resistor (LDR) is a passive electronic sensor used to detect light. It consists of two conductors separated by an insulator, which becomes more conductive when exposed to high levels of light intensity, forming a variable resistor in the circuit. This system senses the intensity of light in its environment. The LDR typically has four pins, and the ones used in this project are GND, 5V, and Analog input.

Components Required

1. Arduino Board
2. LDR sensor
3. Jumper wires

Circuit Diagram



Code

```
void setup() {
    pinMode(2, INPUT);          // Set pin 2 as an input
    Serial.begin(9600);        // Initialize serial communication at
9600 bps
}

void loop() {
    int v = digitalRead(2);    // Read the digital value from pin 2

    if (v == 0) {               // Check if the reading is LOW
```

```
        Serial.print("HIGH READING IS ");
        Serial.println(v);     // Print the reading to the Serial
Monitor
    }
    else if (v == 1) {         // Check if the reading is HIGH
        Serial.print("LOW READING IS ");
        Serial.println(v);     // Print the reading to the Serial
Monitor
    }

    delay(100);                // Optional: Add a delay to reduce
serial output frequency
}
```

Result

The light intensity obtained using an LDR sensor can be observed through the serial monitor of the Arduino IDE. The readings will vary based on the ambient light conditions, allowing for real-time monitoring of light intensity. This project demonstrates how to utilize an LDR sensor to measure light levels and can be expanded for applications such as automatic lighting systems or environmental monitoring.

7. Ultrasonic Distance Sensor

Aim

Integrate ultrasonic sensors with an Arduino Uno board.

Description

Ultrasonic sensors are widely used for applications such as distance measurement and object detection. The HC-SR04 ultrasonic sensor is one of the most commonly used modules for distance measurement with Arduino. In this project, the behavior of three LEDs is controlled based on the measured distance:

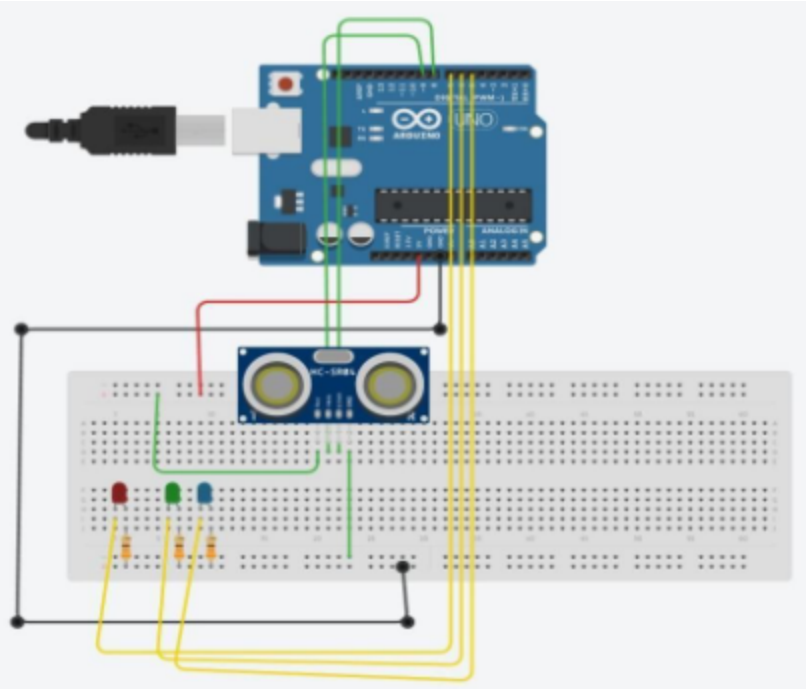
- If the distance is less than 10 cm, a red LED will turn on.
- If the distance is greater than 10 cm but less than 100 cm, a green LED will turn on.
- If the distance is greater than 100 cm, a blue LED will turn on.

The HC-SR04 sensor operates by emitting ultrasonic waves and calculating the time it takes for the waves to bounce back after hitting an object. This time measurement can be used to calculate the distance to the object in front of the sensor.

Components Required

1. Arduino Uno
2. HC-SR04 Ultrasonic Sensor
3. LEDs (Red, Green, Blue)
4. Resistor (220 Ω)
5. Breadboard
6. Jumper Wires

Circuit Diagram



Code

```
void setup() {
    Serial.begin(9600);       // Initialize serial communication at
9600 bps
    pinMode(6, OUTPUT);       // Set pin 6 as output (Trigger pin)
    pinMode(5, INPUT);        // Set pin 5 as input (Echo pin)
    pinMode(4, OUTPUT);       // Set pin 4 as output (LED or indicator)
}

void loop() {
```

```
    digitalWrite(6, HIGH);    // Send a HIGH signal to trigger the
ultrasonic sensor
    delayMicroseconds(10);     // Wait for 10 microseconds
    digitalWrite(6, LOW);     // Set the trigger pin LOW
    delayMicroseconds(10);     // Wait for 10 microseconds

    float dur = pulseIn(5, HIGH); // Measure the duration of the pulse
from the Echo pin
    float dis = dur * 0.017;       // Calculate distance in centimeters

    Serial.println(dis);           // Print the distance to the Serial
Monitor

    if (dis < 50) {                // If distance is less than 50 cm
        digitalWrite(4, HIGH);     // Turn on the output (LED)
    } else if (dis >= 50) {        // If distance is greater than or
equal to 50 cm
        digitalWrite(4, LOW);      // Turn off the output (LED)
    }
}
```

Result

The integration of the HC-SR04 sensor with the Arduino was successful. The output behavior was observed as follows:

- When an object was detected at a distance less than 10 cm, the red LED turned on.
- When the object was between 10 cm and 100 cm away, the green LED illuminated.
- When the object was more than 100 cm away, the blue LED lit up.

This project effectively demonstrates how to use an ultrasonic sensor for distance measurement and control multiple outputs based on varying conditions.

8. Flex Sensor

Aim

Integrate a flex sensor with an Arduino Uno to measure the degree of bend and display it on a 1602 LCD screen.

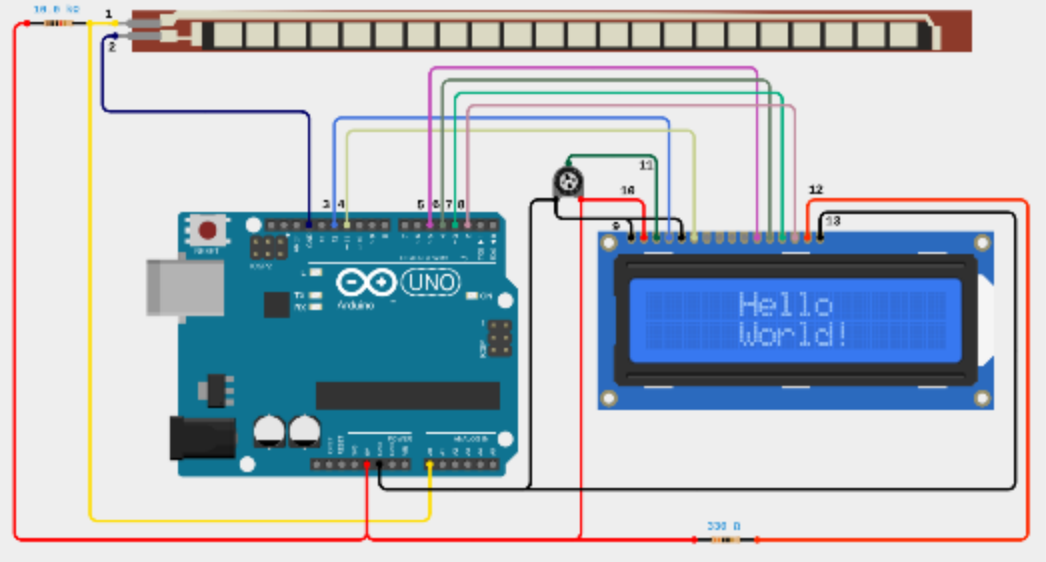
Description

A flex sensor detects changes in resistance based on the degree of bend, providing a variable input to the Arduino Uno. The Arduino processes this input and displays the corresponding degree of bend on the LCD screen in real-time. The flex sensor operates as a variable resistor, where its resistance increases as it bends. This change in resistance is used to calculate the angle of bend.

Components Required

1. Arduino Uno
2. Flex Sensor
3. 1602A LCD Display
4. Potentiometer
5. Resistors (10k Ohm, 330 Ohm)

Circuit Diagram



Library

- **LiquidCrystal library**

Steps to Include the Library

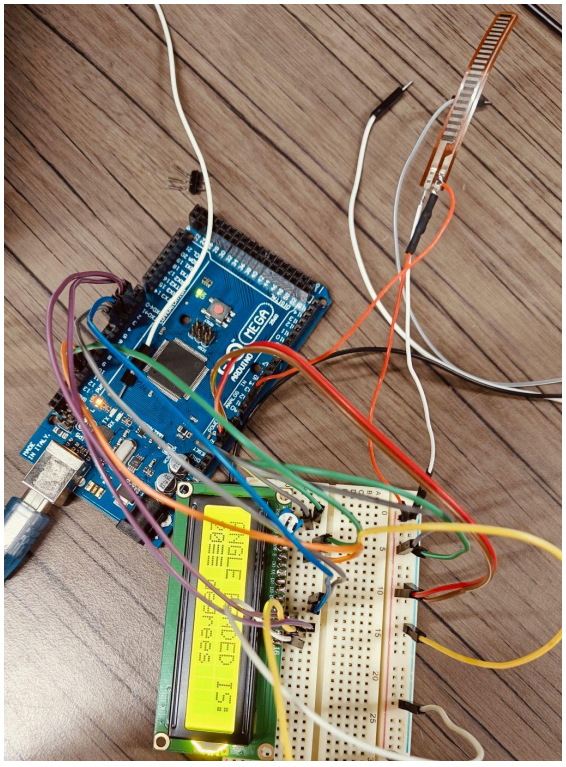
1. Open Arduino IDE.
2. Go to Sketch.
3. Select Include Library.
4. Select Manage Libraries.
5. Install LiquidCrystal library.

Code

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

```
int flex_sensor = A0; // Pin connected to the flex sensor


// Initialize the library by associating any needed LCD interface pin
with the Arduino pin number it is connected to
const int rs = 12, en = 11, d4 = 5, d5 = 4, d6 = 3, d7 = 2;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);


int val = 0;
int angle = 0;


void setup() {
    // Set up the LCD's number of columns and rows:
    lcd.begin(16, 2);
    // Print a message to the LCD.
}


void loop() {
    lcd.clear();
    lcd.print("ANGLE BENDED IS:");
```

```
    int val = analogRead(flex_sensor); // Read the value from the flex
sensor

    int angle = map(val, 795, 920, 0, 45); // Map the analog value to
an angle between 0 and 45 degrees


    // Set the cursor to column 0, line 1 (note: line 1 is the second
row):

    lcd.setCursor(0, 1);

    lcd.print(angle); // Display the angle on the LCD


    delay(500); // Wait for half a second before next reading

}
```



Result

The integration of the Arduino Uno with the flex sensor and 1602 LCD display successfully allows for the measurement and display of the degree of bend detected by the sensor. As the flex sensor bends, its resistance changes, which is then processed by the Arduino. The corresponding degree of bend is calculated and displayed on the 1602 LCD screen in real-time. The LCD provides a clear visual indication of the flex sensor's output, allowing users to monitor the degree of bend with clarity. This project demonstrates how to effectively use a flex sensor in conjunction with an LCD for real-time monitoring applications such as wearable technology or robotic control systems.

9. Capacitive Touch Sensor

Aim

Integrate a gesture sensor with an Arduino Uno board.

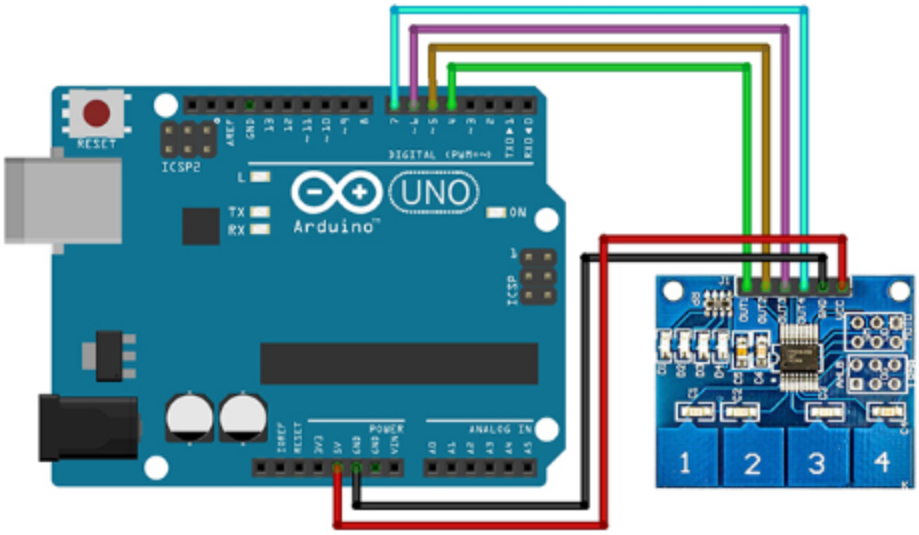
Description

The TTP224 is a capacitive touch IC that allows you to convert your PCB into a touch pad. This feature is useful for replacing traditional buttons or adding touch functionality to your projects. The TTP224 operates with a current range of 2.5 µA to 9.0 µA and supports an operating voltage of 2.4V to 5.5V.

Components Required

1. Arduino Board
2. TTP224 Sensor
3. Jumper wires

Circuit Diagram

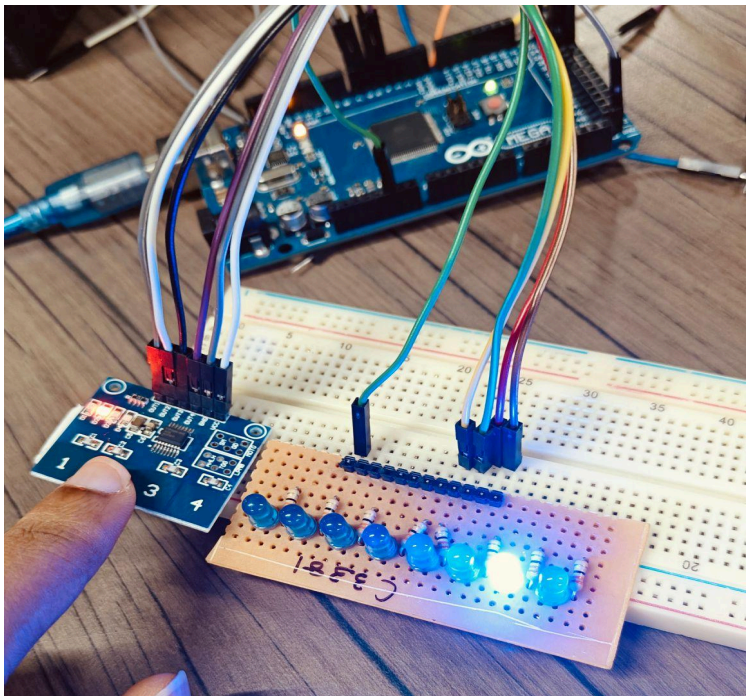


Code

```
// Pin Definitions
int a = 7; // Input pin 1
int b = 6; // Input pin 2
int c = 5; // Input pin 3
int d = 4; // Input pin 4

int l = 30; // Output pin 1
int m = 31; // Output pin 2
int n = 32; // Output pin 3
int o = 33; // Output pin 4

void setup() {
    Serial.begin(9600);     // Initialize serial communication at 9600
bps
```

```
    pinMode(a, INPUT);      // Set input pin a
    pinMode(b, INPUT);      // Set input pin b
    pinMode(c, INPUT);      // Set input pin c
    pinMode(d, INPUT);      // Set input pin d
    pinMode(l, OUTPUT);     // Set output pin l
    pinMode(m, OUTPUT);     // Set output pin m
    pinMode(n, OUTPUT);     // Set output pin n
    pinMode(o, OUTPUT);     // Set output pin o
}

void loop() {
    int k[] = {a, b, c, d}; // Array of input pins
    int t[] = {l, m, n, o}; // Array of output pins

    for (int i = 0; i < 4; i++) {
        switch (digitalRead(k[i])) { // Read the state of each input
pin
            case HIGH:                // If the input is HIGH (1)
                digitalWrite(t[i], HIGH); // Turn on the corresponding
output
                break;
            case LOW:                 // If the input is LOW (0)
                digitalWrite(t[i], LOW);  // Turn off the corresponding
output     break;
default:
                digitalWrite(t[i], LOW);   // Default case to ensure
output is LOW
        }
    }
```

```
}
```



Result

The integration of the TTP224 sensor with the Arduino Uno was successful, allowing for the detection of touch inputs. Each output from the TTP224 corresponds to a touch event on its respective pad, which is read by the Arduino and displayed in the Serial Monitor. This project demonstrates how to use a capacitive touch sensor as an alternative to traditional buttons, providing a modern interface for user interaction in various applications.

10. Dust Sensor

Aim

Integrate the dust sensor with an Arduino Uno board and find the atmospheric dust quantity.

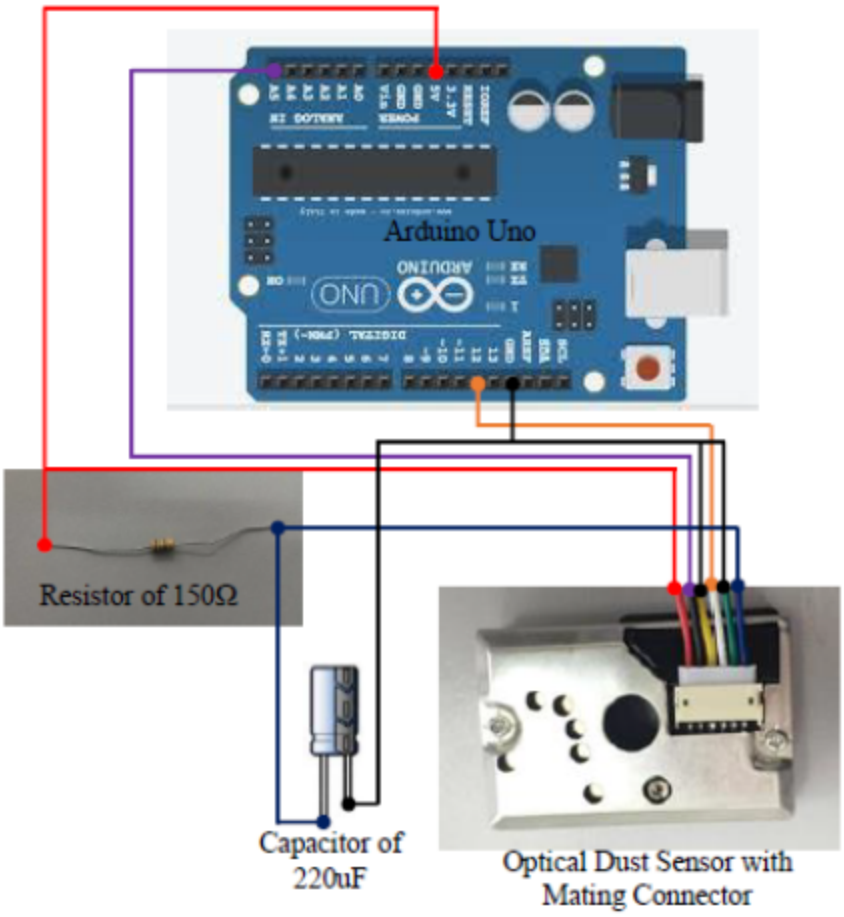
Description

Sharp's GP2Y1010AU0F is an optical air quality sensor, also known as an optical dust sensor, designed to sense dust particles in the air. It consists of an infrared emitting diode and a phototransistor arranged diagonally to detect the reflected light from dust particles. This sensor is particularly effective at detecting very fine particles, such as cigarette smoke, and is commonly used in air purifier systems. To interface with this sensor, you need to connect to its 6-pin, 1.5mm pitch connector using a mating connector.

Components Required

1. GP2Y1010AU0F dust sensor
2. Arduino Uno
3. Breadboard
4. Resistors (150 Ohm)
5. Capacitor (220 µF)

Circuit Diagram



Code

```
int measurePin = A0;          // Pin connected to the dust sensor's output

int ledPower = 34;            // Pin to control the LED in the dust sensor
```

```
unsigned int samplingTime = 280; // Time for sampling signal from the sensor (in microseconds)

unsigned int deltaTime = 40;     // Delay for reading the signal (in microseconds)

unsigned int sleepTime = 9680;   // Time to keep the LED off before next measurement (in microseconds)


float voMeasured = 0;          // Variable to store measured voltage

float calcVoltage = 0;         // Variable to calculate voltage

float dustDensity = 0;         // Variable to store calculated dust density


void setup() {

    Serial.begin(9600);        // Initialize serial communication at 9600 bps

    pinMode(ledPower, OUTPUT); // Set LED control pin as output

}


void loop() {

    digitalWrite(ledPower, LOW);          // Turn on the LED for measurement

    delayMicroseconds(samplingTime);      // Wait for sampling time

    voMeasured = analogRead(measurePin);  // Read the analog value from the sensor

    delayMicroseconds(deltaTime);          // Wait for delta time
```

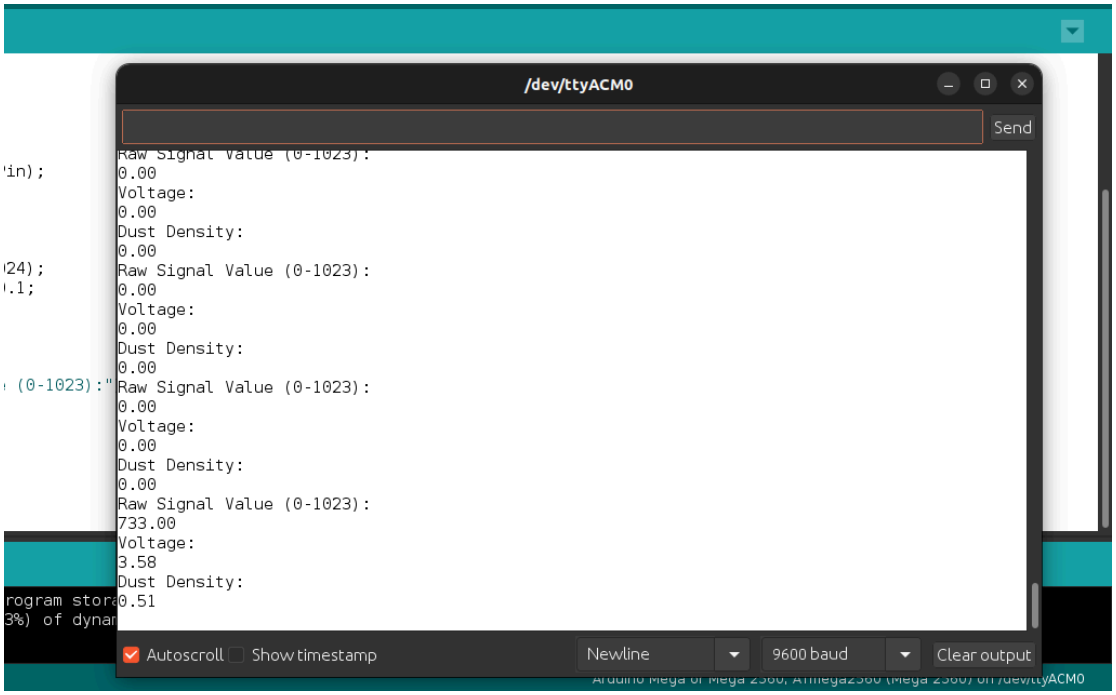
```
    digitalWrite(ledPower, HIGH);          // Turn off the LED
    delayMicroseconds(sleepTime);          // Wait for sleep time

    calcVoltage = voMeasured * (5.0 / 1024); // Convert ADC value to
voltage
    dustDensity = 0.17 * calcVoltage - 0.1;   // Calculate dust density

    if (dustDensity < 0) {
        dustDensity = 0.00;                  // Ensure dust density is
not negative
    }

    Serial.println("Raw Signal Value (0-1023):");
    Serial.println(voMeasured);              // Print raw signal value
    Serial.println("Voltage:");
    Serial.println(calcVoltage);             // Print calculated
voltage
    Serial.println("Dust Density:");
    Serial.println(dustDensity);             // Print calculated dust
density

    delay(1000);                             // Wait before next
measurement
}
```

Result

The integration of the GP2Y1010AU0F dust sensor with the Arduino Uno was successful, allowing for real-time measurement of atmospheric dust quantity. The output values displayed on the Serial Monitor include:

- Raw signal value (ADC reading)
- Calculated voltage from the sensor output
- Dust density in micrograms per cubic meter (μg/m^3)

As a result, users can monitor air quality effectively by assessing dust levels in their environment, which is crucial for health and well-being. This project demonstrates how to utilize a dust sensor in various applications, such as air quality monitoring systems or environmental assessments.



11. SHT45 Temperature and Humidity Sensor

Aim

Integrate the SHT45 temperature and humidity sensor to measure indoor temperature and humidity levels.

Description

The SHT45 is a fourth-generation digital sensor platform designed for measuring relative humidity (RH) and temperature with high accuracy. It utilizes an I2C interface for communication and is suitable for various applications, including environmental monitoring and HVAC systems. Key features of the SHT45 include:

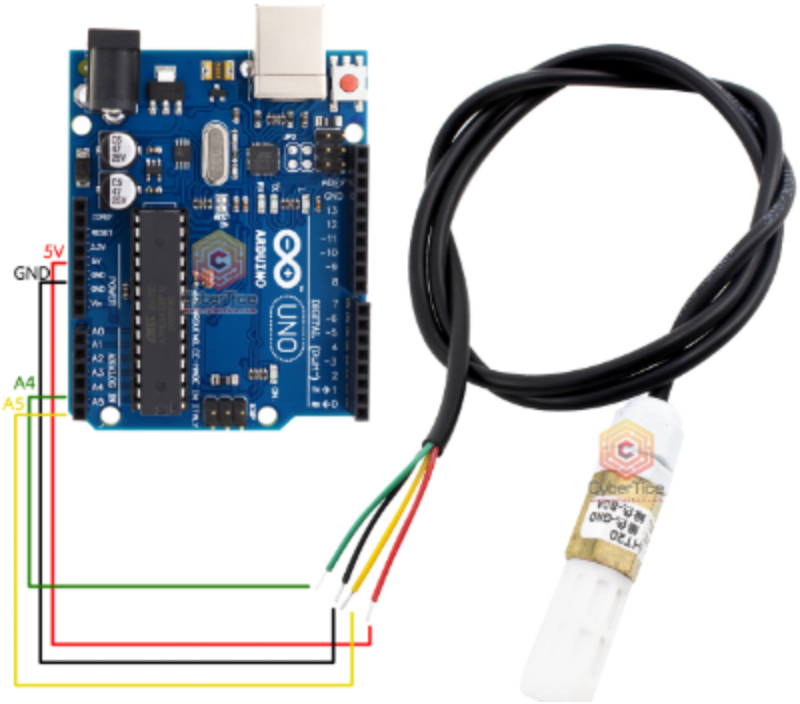
- **Operating Voltage**: 1.08 V to 3.6 V
- **Communication Interface**: I2C
- **Accuracy**: ±1.0% RH / ±0.1°C
- **Measuring Range**: 0 to 100% RH / -40 to 125°C
- **Average Current Consumption**: 2.2 µA during measurements
- **Idle Current**: 80 nA

The SHT45 sensor also includes an integrated heater that can be activated through I2C commands, making it effective for applications in high-humidity environments where condensation may occur.

Components Required

1. Arduino Board
2. SHT45 Sensor
3. Jumper wires

Circuit Diagram



Library

- **Adafruit SHT4x library**

Steps to Include the Library

1. Open Arduino IDE.
2. Go to Sketch.
3. Select Include Library.
4. Select Manage Libraries.

5. Install the Adafruit SHT4x library.

Code

cpp

```cpp
#include "Adafruit_SHT4x.h"


// Create an instance of the sensor
Adafruit_SHT4x sht4 = Adafruit_SHT4x();


void setup() {
    Serial.begin(115200);
    while (!Serial) delay(10); // Wait for Serial Monitor to open


    Serial.println("SHT45 Sensor Test");


    if (!sht4.begin()) {
        Serial.println("Couldn't find SHT45");
        while (1) delay(10); // Halt if sensor not found
    }
    Serial.println("SHT45 found!");
}
```

```
void loop() {
    float humidity = sht4.readHumidity(); // Read humidity
    float temperature = sht4.readTemperature(); // Read temperature

    Serial.print("Temperature: ");
    Serial.print(temperature, 1); // Print temperature with one decimal
place
    Serial.print(" °C, Humidity: ");
    Serial.print(humidity, 1); // Print humidity with one decimal place
    Serial.println(" %");

    delay(1000); // Wait before next reading
}
```

Result

The integration of the SHT45 sensor with the Arduino allows for accurate measurement of indoor temperature and humidity. The readings are displayed in real-time on the Serial Monitor, providing valuable data for monitoring environmental conditions. The SHT45 sensor's high accuracy and low power consumption make it suitable for battery-operated devices and applications requiring precise climate control. Its ability to operate effectively in condensing environments further enhances its utility in various settings, including smart home systems and industrial applications. In summary, the SHT45 represents a significant advancement over previous models like the SHT20, offering improved accuracy, lower power requirements, and enhanced functionality through its integrated heater feature.

12. Digital Clock Using LCD Display and RTC Module

Aim

Create a digital clock using an RTC (Real-Time Clock) module and Arduino Uno, displaying the date and time on an LCD.

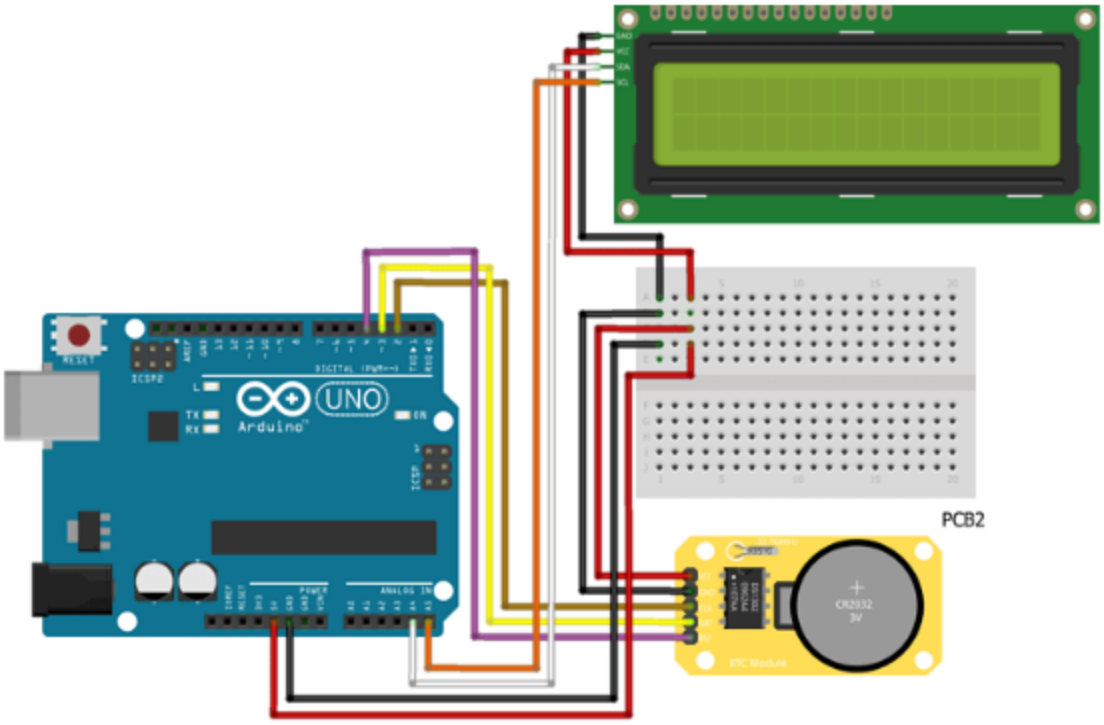
Description

This project focuses on integrating an Arduino microcontroller with an LCD display and a real-time clock (RTC) module to visualize time and date information accurately in real-time. RTC modules are essential in various electronic systems that require precise timekeeping, such as data logging and scheduling, providing a reliable solution independent of external factors.

Components Required

1. LCD Display (16x2)
2. RTC Module (DS1307)
3. Connecting wires
4. Arduino board
5. Resistor (220 Ohm)

Circuit Diagram



Libraries Required

- **LiquidCrystal library**
- **DS1307RTC library**
- **Time library**
- **Wire library** (inbuilt)

Steps to Include the Libraries

1. Open Arduino IDE.
2. Go to Sketch.
3. Select Include Library.

4. Select Manage Libraries.
5. Install the LiquidCrystal library, DS1307RTC library, and Time library.

Code

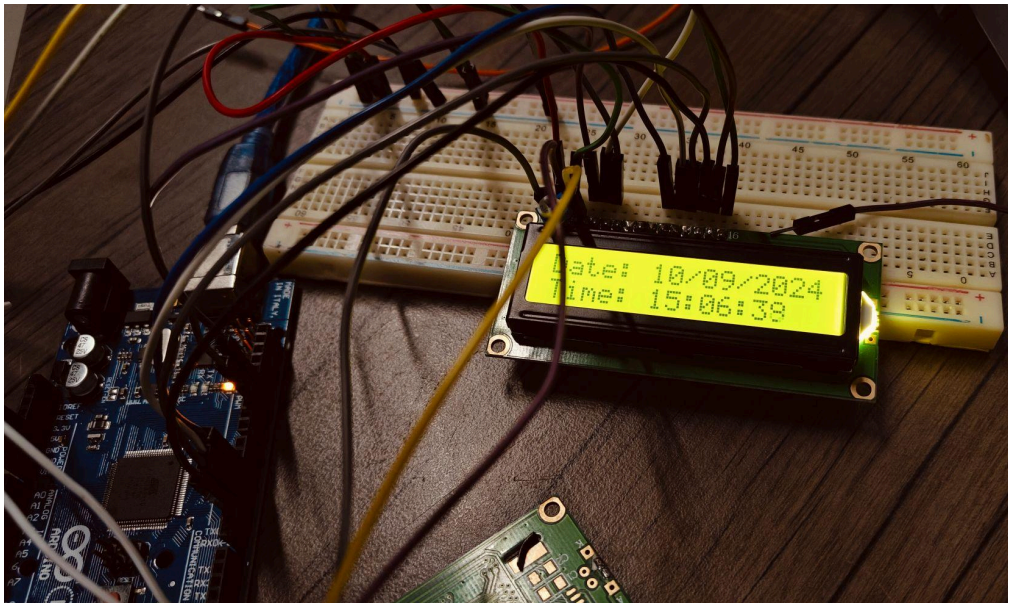
```
#include <Wire.h>

#include <TimeLib.h>

#include <DS1307RTC.h>

#include <LiquidCrystal.h>


// Initialize the LCD with the specified pins

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);


void setup() {

    lcd.begin(16, 2);            // Set up the LCD's number of columns
and rows

    Serial.begin(9600);          // Initialize serial communication at
9600 bps

    while (!Serial);             // Wait for serial port to connect

    delay(200);

    Serial.println("DS1307RTC Read Test");

    Serial.println("--------------------");

}
```

```
void loop() {
    tmElements_t tm;

    // Read the current time from the RTC
    if (RTC.read(tm)) {
        Serial.print("Time = ");

        // Print time in HH:MM:SS format
        print2digits(tm.Hour);
        Serial.write(':');
        print2digits(tm.Minute);
        Serial.write(':');
        print2digits(tm.Second);
        Serial.print(", Date (D/M/Y) = ");
        Serial.print(tm.Day);
        Serial.write('/');
        Serial.print(tm.Month);
        Serial.write('/');
        Serial.print(tmYearToCalendar(tm.Year));
        Serial.println();
```

```
        char dateBuffer[20];
        char timeBuffer[20];

        // Format the date string
        sprintf(dateBuffer, "Date: %02d/%02d/%04d", tm.Day, tm.Month,
tmYearToCalendar(tm.Year));

        // Format the time string
        sprintf(timeBuffer, "Time: %02d:%02d:%02d", tm.Hour, tm.Minute,
tm.Second);

        // Clear the LCD
        lcd.clear();

        // Set the cursor to column 0, line 0 and print the date
        lcd.setCursor(0, 0);
        lcd.print(dateBuffer);

        // Set the cursor to column 0, line 1 and print the time
        lcd.setCursor(0, 1);
        lcd.print(timeBuffer);
```

```
    } else {
        // Handle errors in reading from the RTC
        if (RTC.chipPresent()) {
            Serial.println("The DS1307 is stopped. Please run the
SetTime example to initialize the time and begin running.");
            Serial.println();
        } else {
            Serial.println("DS1307 read error! Please check the
circuitry.");
            Serial.println();
        }
        delay(9000); // Wait before retrying if there's an error
    }
    delay(1000); // Update every second
}

void print2digits(int number) {
    if (number >= 0 && number < 10) {
        Serial.write('0'); // Print leading zero for single-digit
numbers
    }
    Serial.print(number); // Print the number
```

```
}
```



Result

The integration of the RTC module (DS1307) with the Arduino Uno and LCD display successfully allows for real-time display of indoor temperature and humidity. The date and time are displayed on the LCD screen in a user-friendly format. The project demonstrates how to use an RTC module effectively for accurate timekeeping in various applications such as clocks, data logging systems, and scheduling tasks. The LCD provides a clear visual representation of the current date and time.

Additional Notes

- Ensure that the RTC module is powered correctly; otherwise, it may not retain the set date and time when powered off.
- The code includes a commented-out section for setting the initial date and time on the RTC; uncomment this line after adjusting it to your desired date and time.

- This project can be expanded by adding features such as alarms or timers based on the current time displayed by the RTC module.

How to Setup Chirp Stack

Chirp Stack URL:

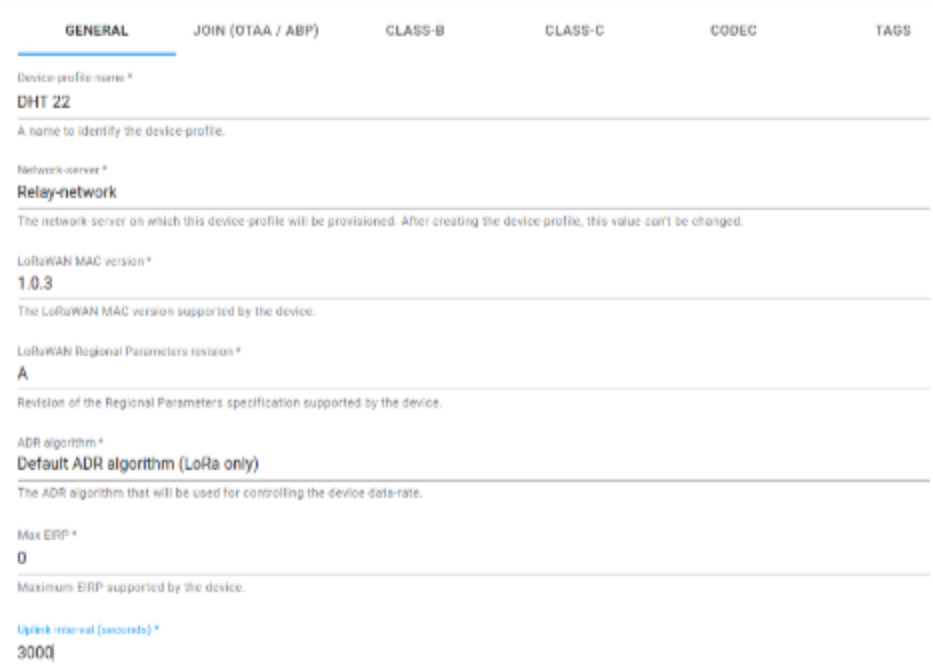
User name :****

Password :*****

➢ Open chirp stack on browser using the above user name and Password

Create a Device profile

➢ Device profile -> Create -> (fill up the option of device profile as in the below window)

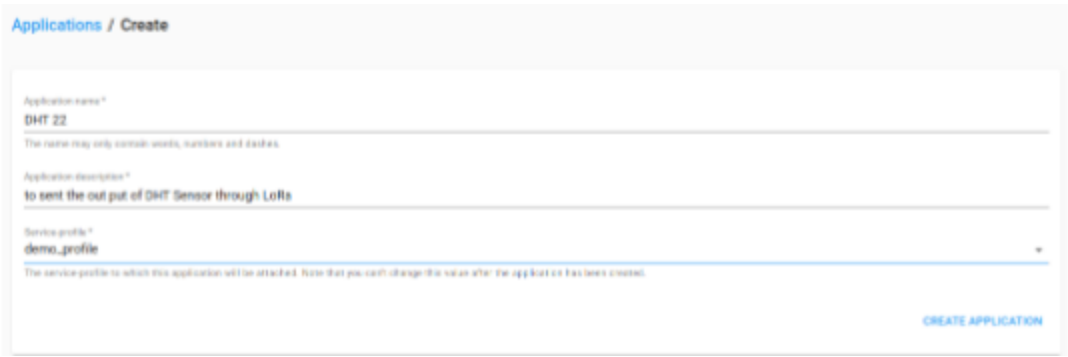


➢ we can enter the decoding codec on the CODEC Section

➢ Then click on Create Device Profile (On bottom)

Create Application

➢ Application -> Create (fill up the option of application as in the below window)



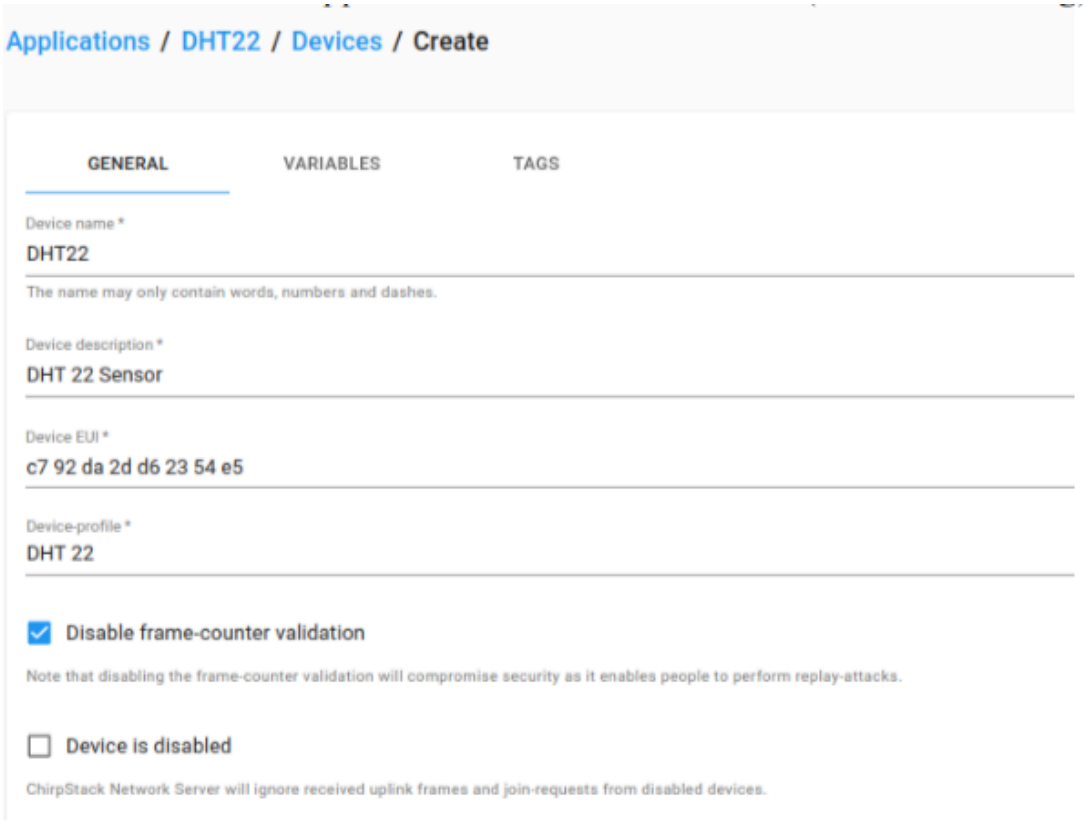
➢ Then click on Create Device Profile (On bottom)

Create Device

➢ Application -> Select our Application -> Devices -> Create (Fill the following)

➢ Then click on Create Device Profile (On bottom)

➢ Click on activation and Generate Device address, Network session Key and Application session key, as in the below window



(We need to put these generated Adders and Keys on the program to get data)

➢ Click on Re Activate Device

How to edit codec of decode data

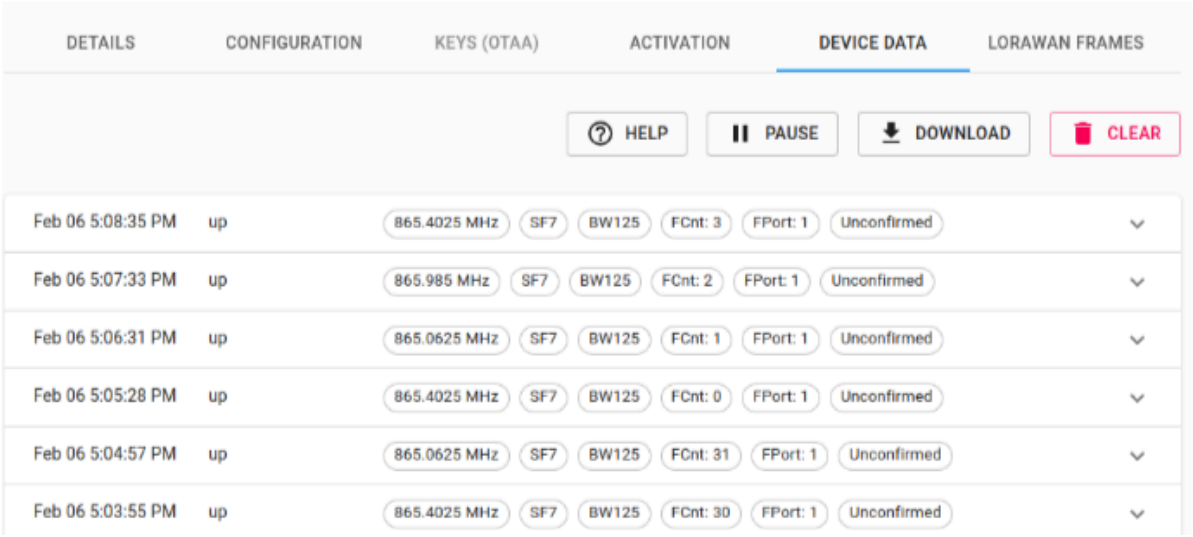
➢ Application -> Select our application name -> Click on our device profile -> Code -. Select custom JavaScript codec function from payload codec

➢ Then Enter the codec (Refer working with bytes web site to get codec hints)

➢ Click on Update device profile

How to check data received on Chirp Stack

➢ Application -> Select our application name -> click on our device name ->Device Data



We can see the receiving data packets here

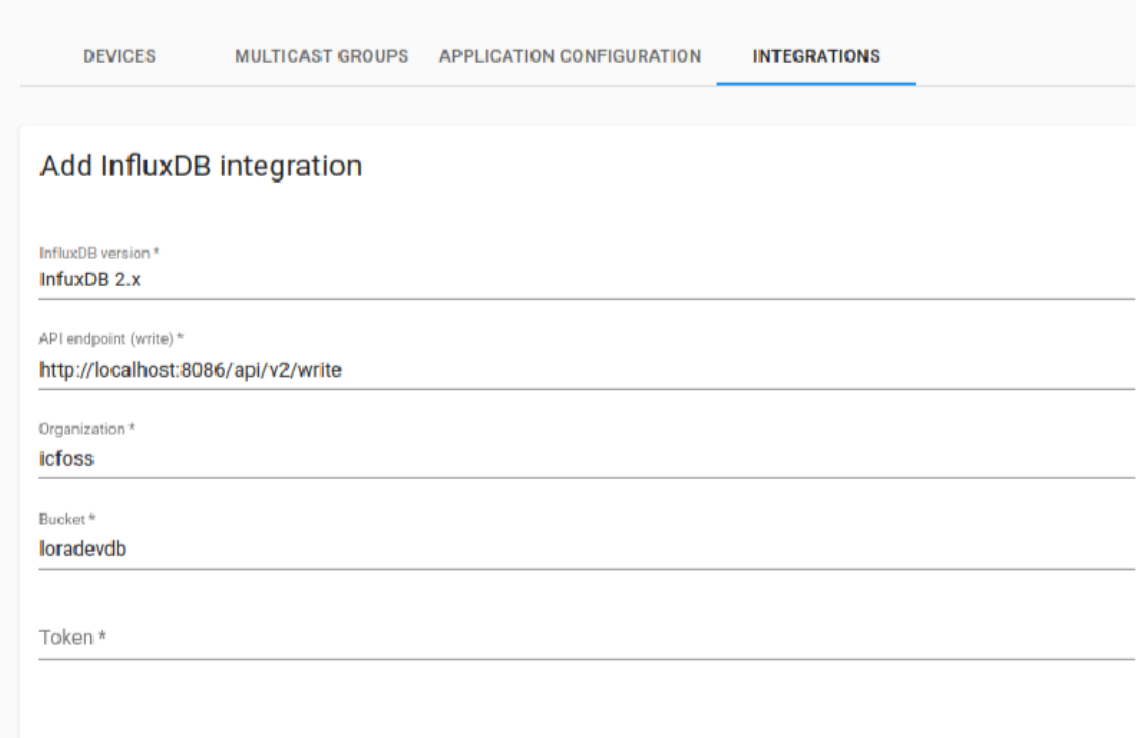
➢ Click on any packet to see the decoded Data

Here data is 25.7

How to Visualize data received on Chirp Stack

➢ Application -> select our Application Name -> Integrations -> Influx DB -> Add

fill up the integration details as follows



(We want to generate the Token from Influx DB)**

➢ Click on add Integration

Influx DB URL:

User name : ***

Password : *****

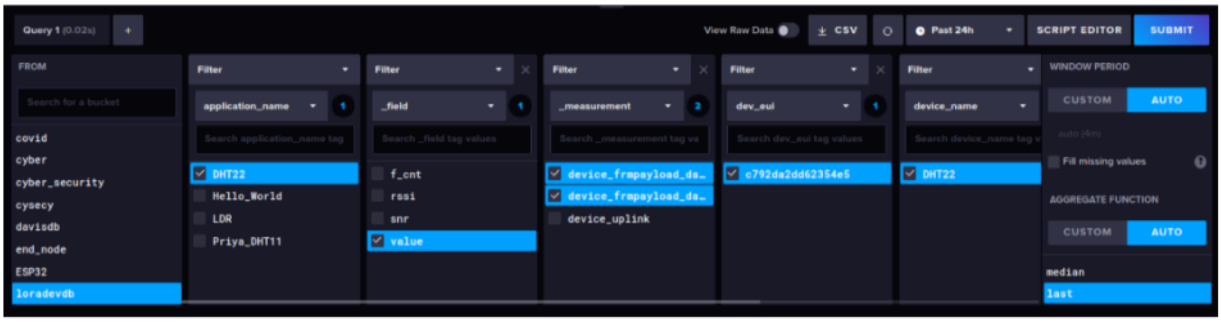
Open Influx DB and sign in using this username and password

➢ Click on the arrow mark icon on the left -> API tokens -> generate API Token -> custom API token -> add a description -> select our bucket -> allow permission for read and write

➢ Generate API Token (Copy the token and Paste it on Chirp Stack)**

Then,

➢ Bucket -> select our bucket -> filter our device and Data from the following window



➢ click on submit ->Script Editor -> copy the Script***

then open Grafana

Grafana URL: https://visualizedev.icfoss.org

User name : ***

Password :****

Open Grafana and sign in using this username and password

➢ Click on settings icon -> configuration -> data source -> Select our data source -> Save & test

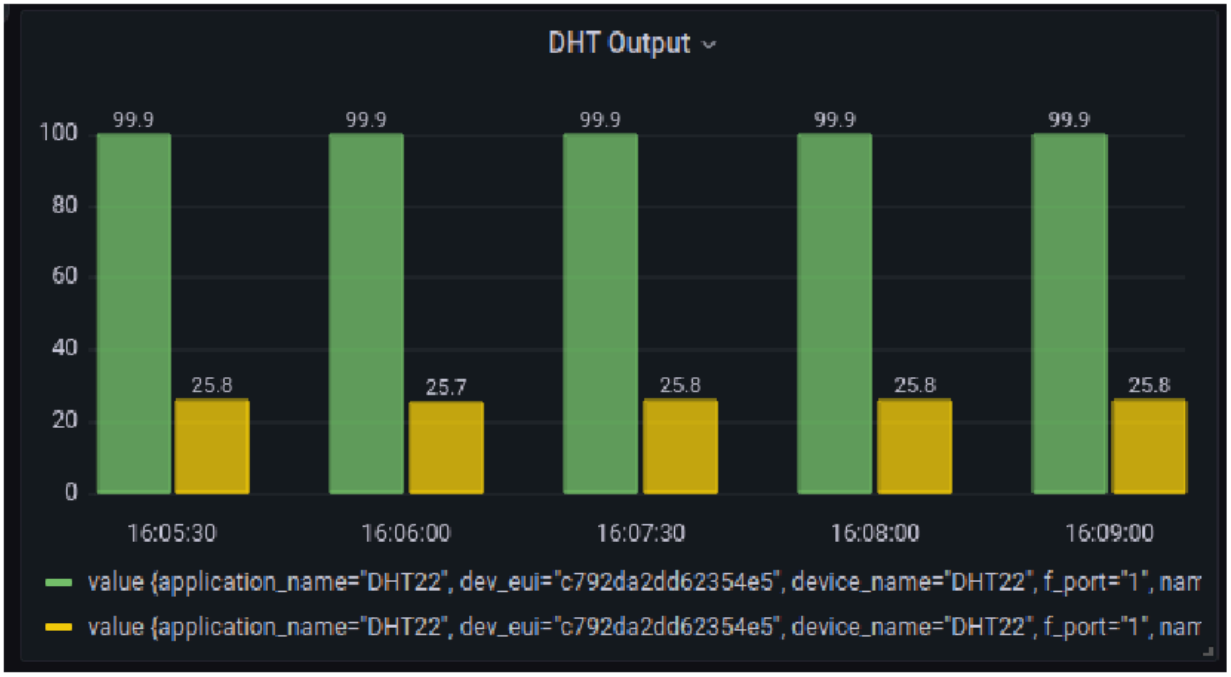
then,

➢ Click on Dash board -> new dash board -> add new panel -> select our data source

➢ Then paste the Script*** copied and select graph type and edit the panel tittle – Apply

Then we can see the graphical visualization of the data received at Chirp Stack

Example,



Example of 8-bit Encoding

To illustrate how to encode an 8-bit value in LoRa:

- **Data to Encode**: Let's say we want to send the hexadecimal values `0x07`, `0x56`, `0x45`, and `0xA3`.
- **Binary Representation**:
  - `0x07` → `00000111`
  - `0x56` → `01010110`
  - `0x45` → `01000101`
  - `0xA3` → `10100011`
-
- **Payload Construction**: These binary values are grouped together into a payload that will be transmitted as a series of chirps.

Encoder Function

cpp

```cpp
// Function to encode temperature and pressure into a payload
void encodeData(uint8_t *payload, float temperature, float pressure) {
    // Scale temperature to one decimal place (e.g., 25.3°C becomes 253)
    uint16_t temp = (uint16_t)(temperature * 10);
    // Convert pressure to a 32-bit unsigned integer
    uint32_t pressu = (uint32_t)(pressure);

    // Fill the payload array with encoded values
    payload[0] = temp >> 8;       // High byte of temperature
    payload[1] = temp & 0xff;     // Low byte of temperature
    payload[2] = (pressu >> 24) & 0xff; // High byte of pressure (byte 3)
    payload[3] = (pressu >> 16) & 0xff; // Byte 2 of pressure
    payload[4] = (pressu >> 8) & 0xff;  // Byte 1 of pressure
    payload[5] = pressu & 0xff;        // Low byte of pressure
}
```

Decoder Function

cpp

```cpp
// Function to decode the payload back into temperature and pressure values
void decodeData(uint8_t *payload, float *temperature, float *pressure)
{
```

```
    // Combine high and low bytes for temperature
    uint16_t temp = (payload[0] << 8) | payload[1];
    // Combine bytes for pressure
    uint32_t pressu = (payload[2] << 24) | (payload[3] << 16) |
(payload[4] << 8) | payload[5];


    // Convert back to float with one decimal place for temperature
    *temperature = temp / 10.0;
    // Convert back to float for pressure
    *pressure = (float)pressu;
}
```

13. Interfacing of Arduino and RFM95 LoRa Module

Aim

Integrate the Arduino Uno board with the RFM95 LoRa module to send data through LoRaWAN and receive it on The Things Network (ChirpStack).

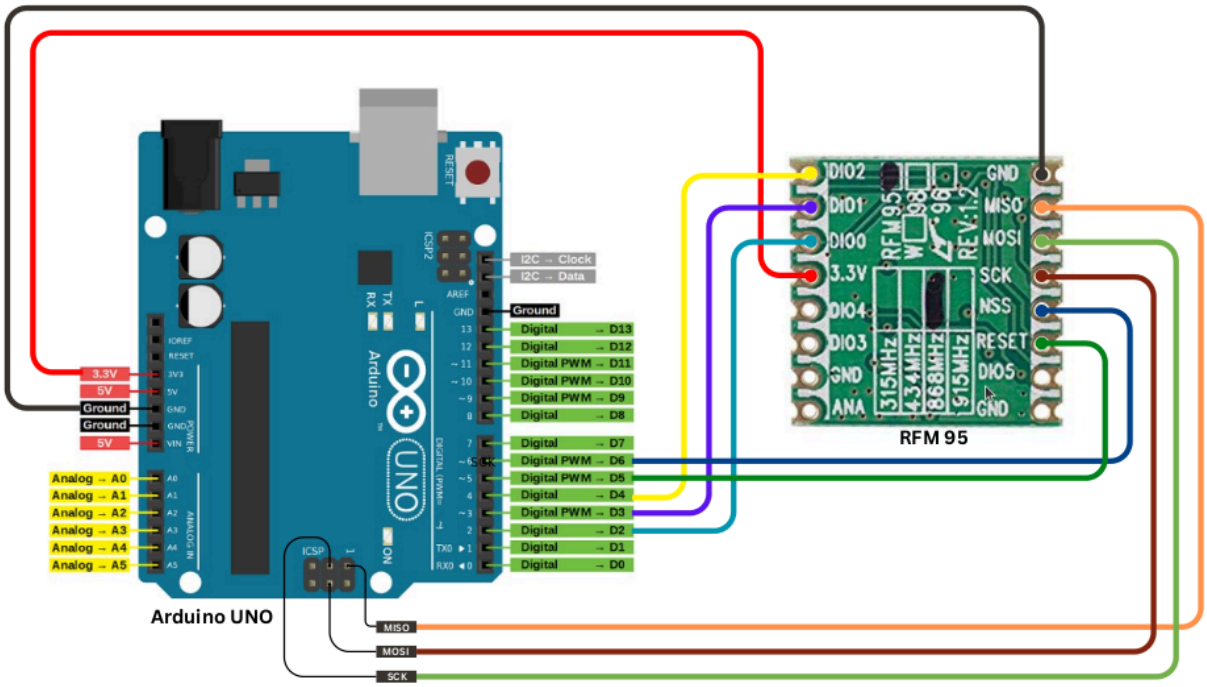
Description

Integrating an Arduino Uno with the RFM95 LoRa (Long Range) module enables wireless communication over long distances with low power consumption, making it ideal for IoT (Internet of Things) applications. The RFM95 module operates on the LoRa protocol, which allows for robust and secure data transmission. This project aims to demonstrate how to set up a simple communication system using the RFM95 module and Arduino, sending data packets that can be received by a LoRaWAN gateway.

Components Required

1. Arduino UNO Board
2. RFM95 Module
3. Connecting wires

Circuit Diagram



Libraries Required

- **MCCI LoRaWAN LMIC library**

Steps to Include the Library

1. Open Arduino IDE.
2. Go to Sketch.
3. Select Include Library.
4. Select Manage Libraries.
5. Install the MCCI LoRaWAN LMIC library.

Code

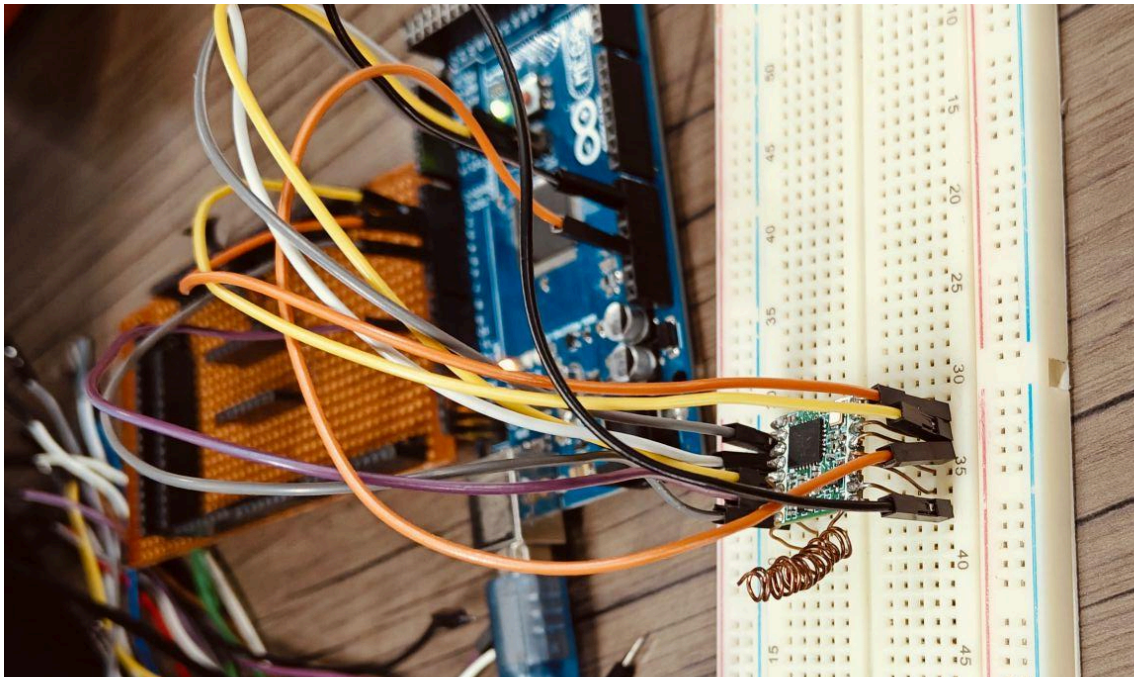
```
#include <lmic.h>

#include <hal/hal.h>

#include <SPI.h>


// LoRaWAN NwkSKey, network session key

static const PROGMEM u1_t NWKSKEY[16] = {0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};

// LoRaWAN AppSKey, application session key

static const PROGMEM u1_t APPSKEY[16] = {0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};

// LoRaWAN end-device address (DevAddr)

static const u4_t DEVADDR = 0x00000000;


static uint8_t mydata[] = "Hello, world!";

static osjob_t sendjob;

const unsigned TX_INTERVAL = 60;


const lmic_pinmap lmic_pins = {.nss = 10, .rxtx = LMIC_UNUSED_PIN, .rst
= 5, .dio = {2, 3, LMIC_UNUSED_PIN}};
```

```
void onEvent(ev_t ev) {
    Serial.print(os_getTime()); Serial.print(": ");
    switch(ev) {
        case EV_SCAN_TIMEOUT: Serial.println(F("EV_SCAN_TIMEOUT"));
break;
        case EV_JOINED: Serial.println(F("EV_JOINED")); break;
        case EV_TXCOMPLETE:
            Serial.println(F("EV_TXCOMPLETE (includes waiting for RX
windows)"));
            if (LMIC.txrxFlags & TXRX_ACK) Serial.println(F("Received
ack"));
            if (LMIC.dataLen) {
                Serial.print(F("Received "));
                Serial.println(LMIC.dataLen);
                Serial.println(F(" bytes of payload"));
            }
            os_setTimedCallback(&sendjob,
os_getTime()+sec2osticks(TX_INTERVAL), do_send);
            break;
        default: Serial.print(F("Unknown event: "));
Serial.println((unsigned) ev); break;
    }
}
```

```
void do_send(osjob_t* j) {
    if (LMIC.opmode & OP_TXRXPEND) {
        Serial.println(F("OP_TXRXPEND, not sending"));
    } else {
        LMIC_setTxData2(1, mydata, sizeof(mydata)-1, 0);
        Serial.println(F("Packet queued"));
    }
}

void setup() {
    while (!Serial); // Wait for serial port to connect
    Serial.begin(115200);
    delay(100);
    Serial.println(F("Starting"));

    os_init();
    LMIC_reset();

    // Set static session parameters
    uint8_t appskey[sizeof(APPSKEY)];
```

```
    uint8_t nwkskey[sizeof(NWKSKEY)];
    memcpy_P(appskey, APPSKEY, sizeof(APPSKEY));
    memcpy_P(nwkskey, NWKSKEY, sizeof(NWKSKEY));
    LMIC_setSession(0x13, DEVADDR, nwkskey, appskey);

    // Set up channels
    LMIC_setupChannel(0, 865062500, DR_RANGE_MAP(DR_SF12, DR_SF7),
BAND_MILLI);
    LMIC_setupChannel(1, 865402500, DR_RANGE_MAP(DR_SF12, DR_SF7),
BAND_MILLI);
    LMIC_setupChannel(2, 865985000, DR_RANGE_MAP(DR_SF12, DR_SF7),
BAND_MILLI);

    // Disable link check validation
    LMIC_setLinkCheckMode(0);

    // Set data rate and transmit power for uplink
    LMIC.dn2Dr = DR_SF9;
    LMIC_setDrTxpow(DR_SF7,14);

    do_send(&sendjob); // Start sending data
}
```

```
void loop() {
    os_runloop_once(); // Run the LMIC event loop
}
```



Result

The integration of the Arduino UNO and RFM95 module was successful. Data was sent through LoRaWAN and successfully received on ChirpStack (The Things Network). This project demonstrates how to establish a simple wireless communication system using LoRa technology.

14. Integration of ULPLoRa Board with DHT11 SENSOR

Aim

Integrate the ULPLoRa board with the DHT11 sensor for temperature and humidity monitoring and visualization in IoT applications.

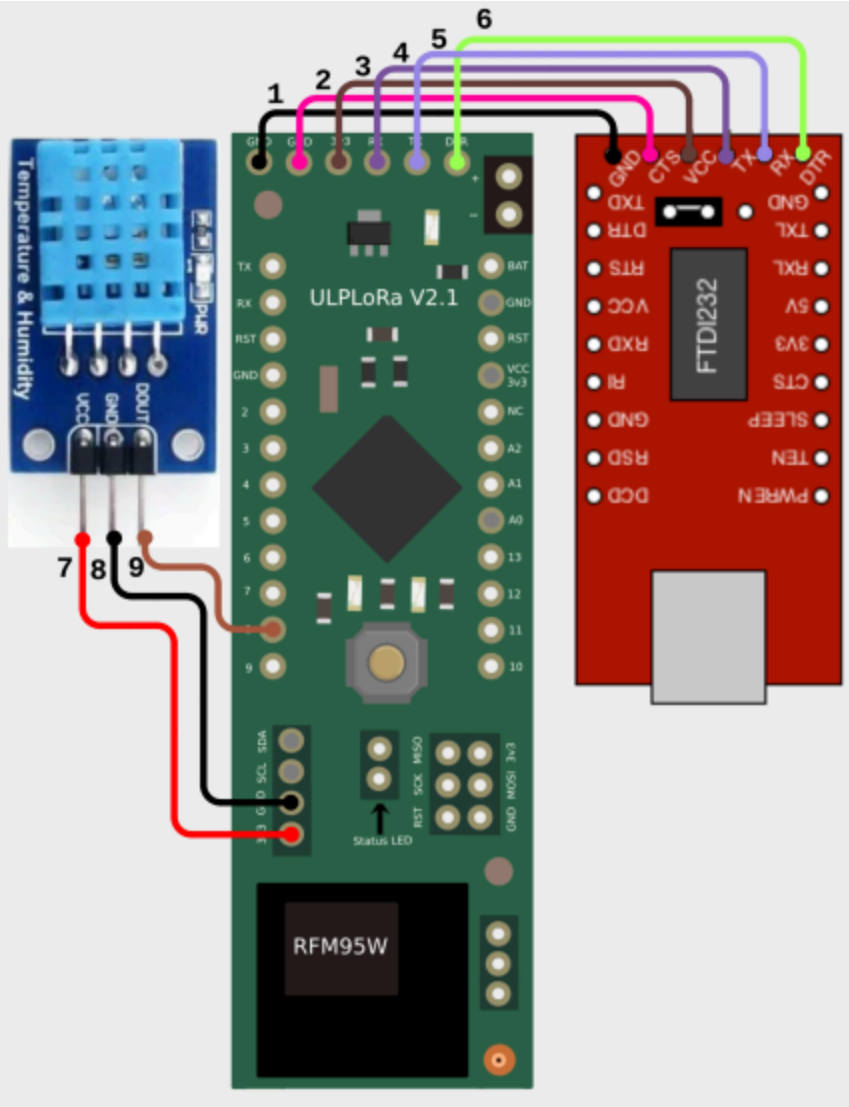
Description

The ULPLoRa board combines an Arduino Pro Mini with an RFM95W LoRa module, providing a flexible platform for wireless communication. The DHT11 sensor offers accurate temperature and humidity measurements. This project aims to establish a robust system capable of capturing temperature and humidity data, wirelessly transmitting it via LoRa to the ChirpStack server, storing it in InfluxDB, and visualizing it using Grafana.

Components Required

1. ULPLoRa Board
2. DHT11 Sensor
3. FTDI FT232RL USB to Serial Adapter

Circuit Diagram



Libraries Required

- **DHT sensor library (DFRobot_DHT11 Version 1.0.0)**
- **MCCI LoRaWAN LMIC library** (for LoRa communication)

Steps to Include Libraries

1. Open Arduino IDE.
2. Go to Sketch.

3. Select Include Library.
4. Select Manage Libraries.
5. Install the DHT sensor library (DFRobot_DHT11 Version 1.0.0).
6. Download and install the MCCI LoRaWAN LMIC library from the provided link.

Code

```
#include <lmic.h>

#include <hal/hal.h>

#include <SPI.h>

#include <DFRobot_DHT11.h>


DFRobot_DHT11 DHT;

#define DHT11_PIN 11


static const PROGMEM u1_t NWKSKEY[16] = {0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};

static const u1_t PROGMEM APPSKEY[16] = {0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};

static const u4_t DEVADDR = 0x00000000;


void os_getArtEui(u1_t* buf) { }

void os_getDevEui(u1_t* buf) { }
```

```
void os_getDevKey(u1_t* buf) { }


static uint8_t mydata[4];

static osjob_t sendjob;

const unsigned TX_INTERVAL = 60;


const lmic_pinmap lmic_pins = {

    .nss = 10,

    .rxtx = LMIC_UNUSED_PIN,

    .rst = 5,

    .dio = {2, 3, LMIC_UNUSED_PIN},

};


void onEvent(ev_t ev) {

    Serial.print(os_getTime()); Serial.print(": ");

    switch(ev) {

        case EV_SCAN_TIMEOUT: Serial.println(F("EV_SCAN_TIMEOUT"));
break;

        case EV_BEACON_FOUND: Serial.println(F("EV_BEACON_FOUND"));
break;

        case EV_BEACON_MISSED: Serial.println(F("EV_BEACON_MISSED"));
break;
```

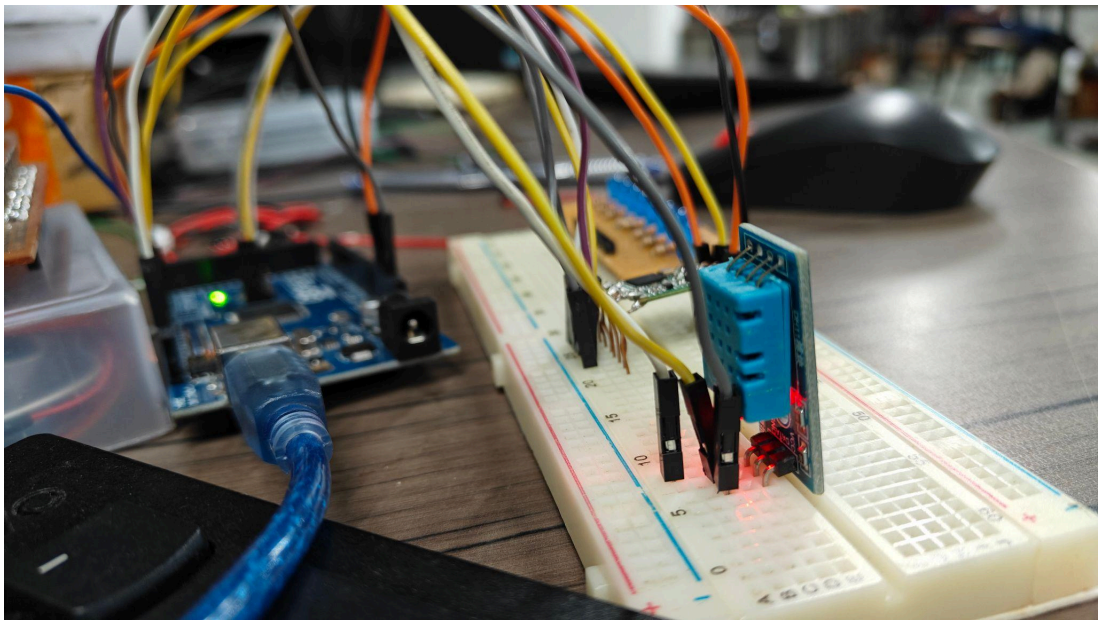
```
        case EV_BEACON_TRACKED: Serial.println(F("EV_BEACON_TRACKED"));
break;

        case EV_JOINING: Serial.println(F("EV_JOINING")); break;

        case EV_JOINED: Serial.println(F("EV_JOINED")); break;

        case EV_JOIN_FAILED: Serial.println(F("EV_JOIN_FAILED"));
break;

        case EV_REJOIN_FAILED: Serial.println(F("EV_REJOIN_FAILED"));
break;

        case EV_TXCOMPLETE:

            Serial.println(F("EV_TXCOMPLETE (includes waiting for RX
windows)"));

            if (LMIC.txrxFlags & TXRX_ACK) Serial.println(F("Received
ack"));

            if (LMIC.dataLen) {

                Serial.println(F("Received "));

                Serial.println(LMIC.dataLen);

                Serial.println(F(" bytes of payload"));

            }

            break;

        case EV_LOST_TSYNC: Serial.println(F("EV_LOST_TSYNC")); break;

        case EV_RESET: Serial.println(F("EV_RESET")); break;

        case EV_RXCOMPLETE: Serial.println(F("EV_RXCOMPLETE")); break;

        case EV_LINK_DEAD: Serial.println(F("EV_LINK_DEAD")); break;
```

```
        case EV_LINK_ALIVE: Serial.println(F("EV_LINK_ALIVE")); break;
        case EV_TXSTART: Serial.println(F("EV_TXSTART")); break;
        case EV_TXCANCELED: Serial.println(F("EV_TXCANCELED")); break;
        case EV_JOIN_TXCOMPLETE: Serial.println(F("EV_JOIN_TXCOMPLETE:
no JoinAccept")); break;
        default: Serial.print(F("Unknown event: "));
Serial.println((unsigned) ev); break;
    }
}


void do_send(osjob_t* j) {
    if (LMIC.opmode & OP_TXRXPEND) {
        Serial.println(F("OP_TXRXPEND, not sending"));
    } else {
        DHT.read(DHT11_PIN);
        Serial.print(" temp : ");
        Serial.print(DHT.temperature);
        Serial.print("   hum : ");
        Serial.print(DHT.humidity);
        Serial.println();

        delay(1000);
```

```
        uint8_t payload[4];
        uint16_t temp = (uint16_t)(DHT.temperature * 100);
        uint16_t hum = (uint16_t)(DHT.humidity * 100);

        payload[0] = temp >> 8;
        payload[1] = temp & 0xff;
        payload[2] = hum >> 8;
        payload[3] = hum & 0xff;

        LMIC_setTxData2(1, payload, sizeof(payload), 0);
        Serial.println(F("Packet queued"));
    }
    os_setTimedCallback(&sendjob, os_getTime() +
sec2osticks(TX_INTERVAL), do_send);
}

void setup() {
    while (!Serial);
    Serial.begin(115200);
    delay(100);
```

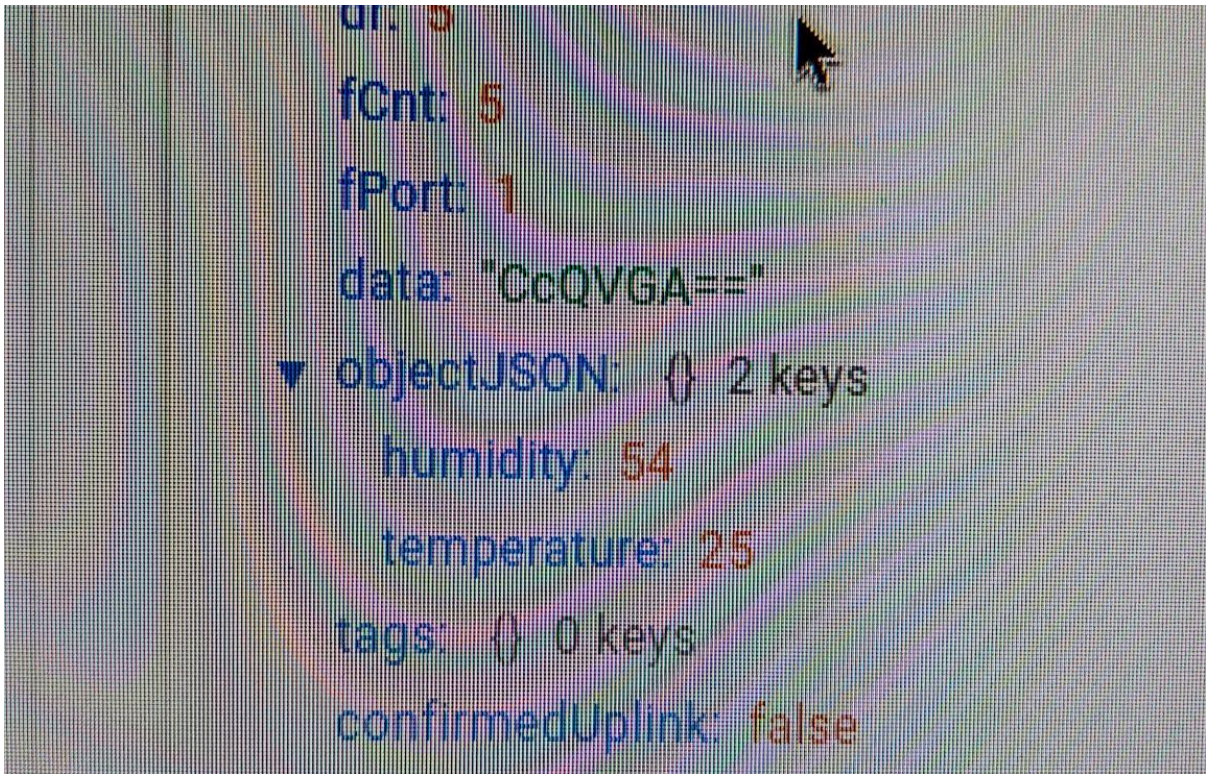
```
    Serial.println(F("Starting"));


    os_init();
    LMIC_reset();


    #ifdef PROGMEM
    uint8_t appskey[sizeof(APPSKEY)], nwkskey[sizeof(NWKSKEY)];
    memcpy_P(appskey, APPSKEY, sizeof(APPSKEY));
    memcpy_P(nwkskey, NWKSKEY, sizeof(NWKSKEY));
    LMIC_setSession(0x13, DEVADDR, nwkskey, appskey);
    #else
    LMIC_setSession(0x13, DEVADDR, NWKSKEY, APPSKEY);
    #endif


    #if defined(CFG_in866)
    LMIC_setupChannel(0, 865062500, DR_RANGE_MAP(DR_SF12, DR_SF7),
BAND_MILLI);
    LMIC_setupChannel(1, 865402500, DR_RANGE_MAP(DR_SF12, DR_SF7),
BAND_MILLI);
    LMIC_setupChannel(2, 865985000, DR_RANGE_MAP(DR_SF12, DR_SF7),
BAND_MILLI);
    #else
```

```
    # error Region not supported
    #endif

    LMIC_setLinkCheckMode(0);
    LMIC.dn2Dr = DR_SF9;
    LMIC_setDrTxpow(DR_SF7,14);

    do_send(&sendjob);
}

void loop() {
    unsigned long now = millis();
    if ((now & 512) != 0) digitalWrite(13, HIGH);
    else digitalWrite(13, LOW);

    os_runloop_once();
}
```



Result

The integration of the ULPLoRa board with the DHT11 sensor successfully enables temperature and humidity monitoring within IoT applications. The data is captured by the DHT11 sensor, transmitted via LoRaWAN using the MCCI LoRaWAN LMIC library, stored in InfluxDB, and visualized using Grafana. This setup empowers users to efficiently gather valuable environmental data, enhancing their ability to monitor conditions remotely and make informed decisions based on real-time information.


fCnt: 5
fPort: 1
data: "CcQVGA=="
▼ objectJSON: {} 2 keys
humidity: 54
temperature: 25
tags: {} 0 keys
confirmedUplink: false

15. Integration of ULPLoRa Board with BMP180 Sensor

Aim

Integrate the ULPLoRa board with the BMP180 sensor for pressure monitoring and visualization in IoT applications.

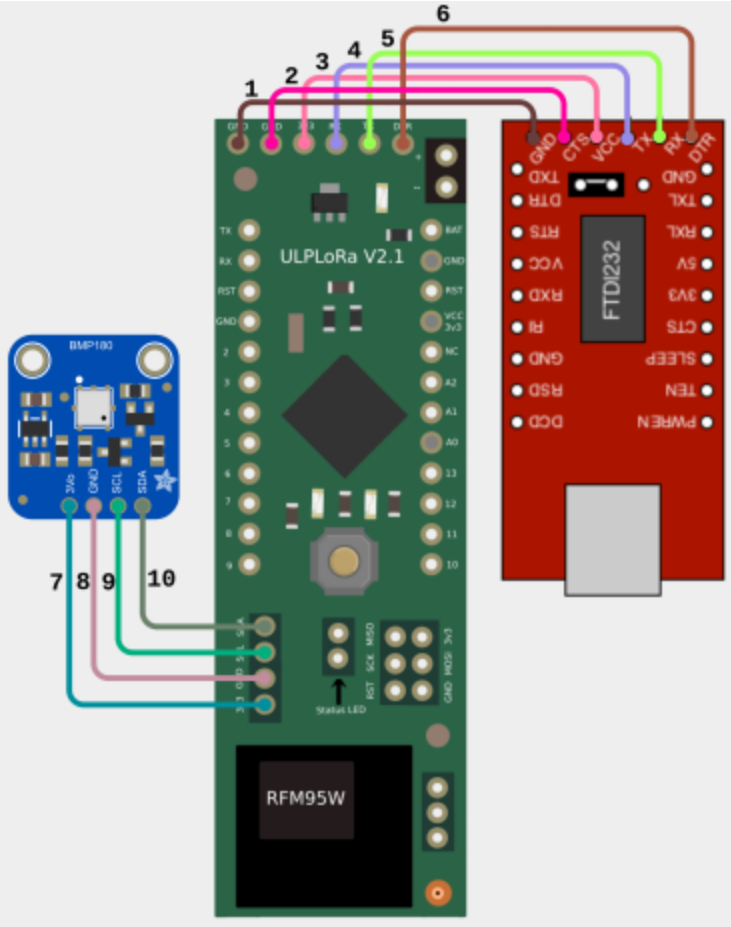
Description

The ULPLoRa board combines an Arduino Pro Mini with an RFM95W LoRa module, providing a versatile platform for wireless communication. The BMP180 sensor offers precise atmospheric pressure measurements. The primary objective of this project is to establish a robust system capable of capturing pressure data, wirelessly transmitting it via LoRa to the ChirpStack server, storing it in InfluxDB, and subsequently visualizing it using Grafana.

Components Required

1. ULPLoRa Board
2. BMP180 Sensor
3. FTDI FT232RL USB to Serial Adapter

Circuit Diagram



Libraries Required

- **BMP180 library**: Download from [BMP180 GitHub]
- **MCCI LoRaWAN LMIC library**: Download from [MCCI LoRaWAN LMIC GitLab]

Code

```
#include <lmic.h>
#include <hal/hal.h>
#include <SPI.h>
```

```
#include <Wire.h>
#include <BMP180.h>

BMP180 myBMP(BMP180_ULTRAHIGHRES);

#ifdef COMPILE_REGRESSION_TEST
# define CFG_in866 1
#else
# define FILLMEIN
#endif

static const PROGMEM u1_t NWKSKEY[16] = {0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
static const u1_t PROGMEM APPSKEY[16] = {0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
static const u4_t DEVADDR = 0x00000000;

void os_getArtEui (u1_t* buf) { }
void os_getDevEui (u1_t* buf) { }
void os_getDevKey (u1_t* buf) { }

static uint8_t mydata[4];
static osjob_t sendjob;
const unsigned TX_INTERVAL = 60;

const lmic_pinmap lmic_pins = {
    .nss = 10,
    .rxtx = LMIC_UNUSED_PIN,
```

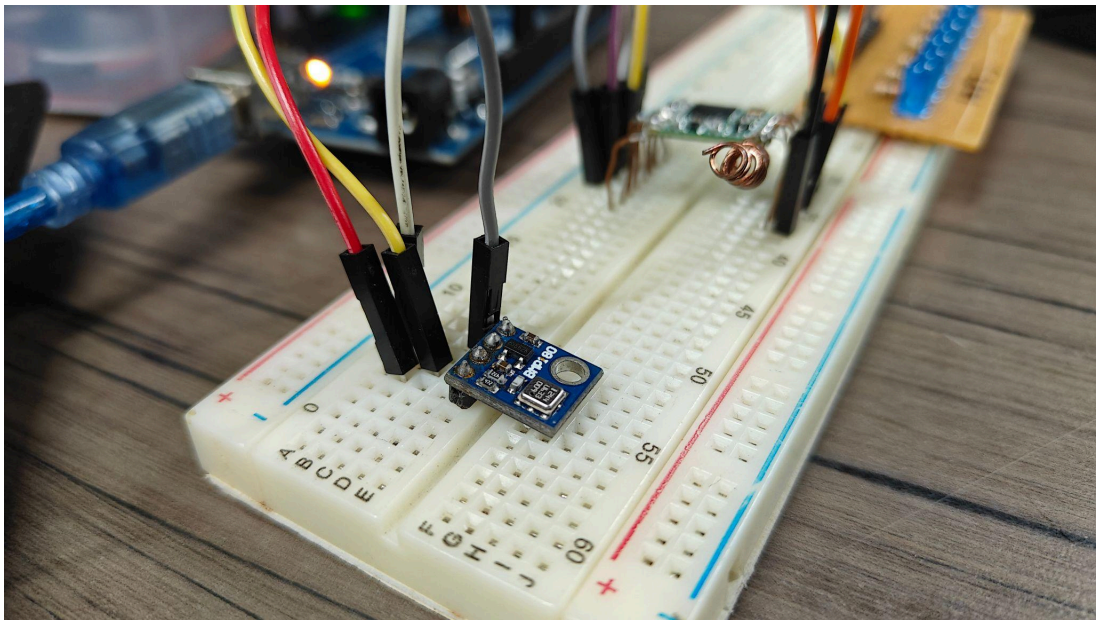
```
    .rst = 5,
    .dio = {2, 3, LMIC_UNUSED_PIN}
};


void onEvent (ev_t ev) {
    Serial.print(os_getTime()); Serial.print(": ");
    switch(ev) {
        case EV_SCAN_TIMEOUT: Serial.println(F("EV_SCAN_TIMEOUT"));
break;
        case EV_BEACON_FOUND: Serial.println(F("EV_BEACON_FOUND"));
break;
        case EV_BEACON_MISSED: Serial.println(F("EV_BEACON_MISSED"));
break;
        case EV_BEACON_TRACKED: Serial.println(F("EV_BEACON_TRACKED"));
break;
        case EV_JOINING: Serial.println(F("EV_JOINING")); break;
        case EV_JOINED: Serial.println(F("EV_JOINED")); break;
        case EV_JOIN_FAILED: Serial.println(F("EV_JOIN_FAILED"));
break;
        case EV_REJOIN_FAILED: Serial.println(F("EV_REJOIN_FAILED"));
break;
        case EV_TXCOMPLETE:
            Serial.println(F("EV_TXCOMPLETE (includes waiting for RX
windows)"));
            if (LMIC.txrxFlags & TXRX_ACK) Serial.println(F("Received
ack"));
            if (LMIC.dataLen) {
                Serial.print(F("Received "));
Serial.print(LMIC.dataLen); Serial.println(F(" bytes of payload"));
            }
            break;
```

```
        case EV_LOST_TSYNC: Serial.println(F("EV_LOST_TSYNC")); break;
        case EV_RESET: Serial.println(F("EV_RESET")); break;
        case EV_RXCOMPLETE: Serial.println(F("EV_RXCOMPLETE")); break;
        case EV_LINK_DEAD: Serial.println(F("EV_LINK_DEAD")); break;
        case EV_LINK_ALIVE: Serial.println(F("EV_LINK_ALIVE")); break;
        case EV_TXSTART: Serial.println(F("EV_TXSTART")); break;
        case EV_TXCANCELED: Serial.println(F("EV_TXCANCELED")); break;
        case EV_RXSTART: break;
        case EV_JOIN_TXCOMPLETE: Serial.println(F("EV_JOIN_TXCOMPLETE:
no JoinAccept")); break;
        default: Serial.print(F("Unknown event: "));
Serial.println((unsigned) ev); break;   }}
void do_send(osjob_t* j){
    if (LMIC.opmode & OP_TXRXPEND) Serial.println(F("OP_TXRXPEND, not
sending"));
    else {
        float temperatur = myBMP.getTemperature();
        float pressur = myBMP.getPressure();
        myBMP.begin();
        Serial.print(F("Temperature.......: "));
Serial.print(myBMP.getTemperature(), 1); Serial.println(F(" +-1.0C"));
        Serial.print(F("Pressure..........: "));
Serial.print(myBMP.getPressure()); Serial.println(F(" +-100Pa"));
        delay(1000);


        uint8_t payload[6];
        uint16_t temp = (uint16_t)(temperatur * 10);
        uint32_t pressu = (uint32_t)(pressur);
```

```
        payload[0] = temp >> 8;
        payload[1] = temp & 0xff;
        payload[2] = (pressu >> 24) & 0xff;
        payload[3] = (pressu >> 16) & 0xff;
        payload[4] = (pressu >> 8) & 0xff;
        payload[5] = pressu & 0xff;

        LMIC_setTxData2(1, payload, sizeof(payload), 0);
        Serial.println(F("Packet queued"));
    }
    os_setTimedCallback(&sendjob, os_getTime() +
sec2osticks(TX_INTERVAL), do_send);
}

void setup() {
    while (!Serial);
    Serial.begin(115200);
    delay(100);
    Serial.println(F("Starting"));
    #ifdef VCC_ENABLE
    pinMode(VCC_ENABLE, OUTPUT);
    digitalWrite(VCC_ENABLE, HIGH);
    delay(1000);
    #endif

    os_init();
    LMIC_reset();
```

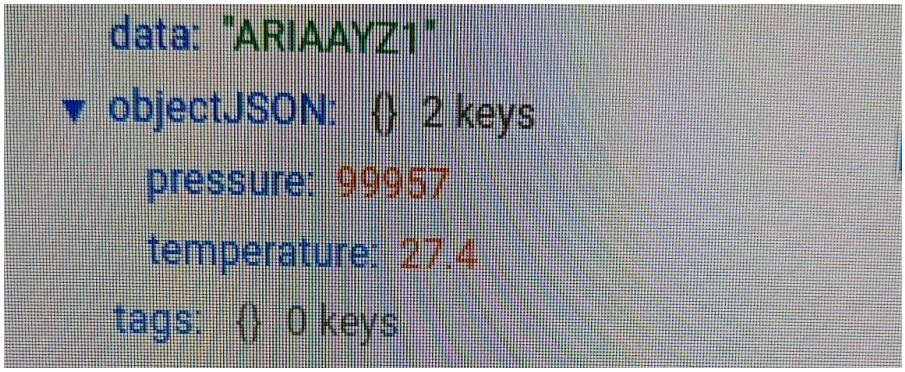
```
    #ifdef PROGMEM
    uint8_t appskey[sizeof(APPSKEY)], nwkskey[sizeof(NWKSKEY)];
    memcpy_P(appskey, APPSKEY, sizeof(APPSKEY));
    memcpy_P(nwkskey, NWKSKEY, sizeof(NWKSKEY));

    LMIC_setSession(0x13, DEVADDR, nwkskey, appskey);
    #else
    LMIC_setSession(0x13, DEVADDR, NWKSKEY, APPSKEY);
    #endif

    #if defined(CFG_eu868)
    LMIC_setupChannel(0, 868100000, DR_RANGE_MAP(DR_SF12, DR_SF7),
BAND_CENTI);
    LMIC_setupChannel(1, 868300000, DR_RANGE_MAP(DR_SF12, DR_SF7B),
BAND_CENTI);
    LMIC_setupChannel(2, 868500000, DR_RANGE_MAP(DR_SF12, DR_SF7),
BAND_CENTI);
    LMIC_setupChannel(3, 867100000, DR_RANGE_MAP(DR_SF12, DR_SF7),
BAND_CENTI);
    LMIC_setupChannel(4, 867300000, DR_RANGE_MAP(DR_SF12, DR_SF7),
BAND_CENTI);
    LMIC_setupChannel(5, 867500000, DR_RANGE_MAP(DR_SF12,... DR_SF7),
BAND_CENTI);

    LMIC_setupChannel(6, 867700000, DR_RANGE_MAP(DR_SF12, DR_SF7),
BAND_CENTI);

    LMIC_setupChannel(7, 867900000, DR_RANGE_MAP(DR_SF12, DR_SF7),
BAND_CENTI);
    LMIC_setupChannel(8, 868800000, DR_RANGE_MAP(DR_FSK , DR_FSK),
BAND_MILLI); // g2-band
```

```
        #elif defined(CFG_us915) || defined(CFG_au915)
   LMIC_selectSubBand(1);
   #elif defined(CFG_as923)
   #elif defined(CFG_kr920)
   #elif defined(CFG_in866)
   LMIC_setupChannel(0 ,865062500 ,DR_RANGE_MAP(DR_SF12
,DR_SF7),BAND_MILLI);
   LMIC_setupChannel(1 ,865402500 ,DR_RANGE_MAP(DR_SF12
,DR_SF7),BAND_MILLI);
   LMIC_setupChannel(2 ,865985000 ,DR_RANGE_MAP(DR_SF12
,DR_SF7),BAND_MILLI);
   #else
   # error Region not supported
   #endif


   LMIC_setLinkCheckMode(0);


   LMIC.dn2Dr = DR_SF9;
   LMIC_setDrTxpow(DR_SF7 ,14);
do_send(&sendjob);}
void loop() {
   unsigned long now = millis();
   if ((now &512)!=0) digitalWrite(13,HIGH);
   else digitalWrite(13 ,LOW);
    os_runloop_once();}
```



Result

The integration of the ULPLoRa board with the BMP180 sensor successfully enables pressure monitoring within IoT applications. The data is captured by the BMP180 sensor, transmitted via LoRaWAN using the MCCI LoRaWAN LMIC library, stored in InfluxDB, and visualized using Grafana. This setup empowers users to efficiently gather valuable environmental data regarding atmospheric pressure and temperature, enhancing their ability to monitor conditions remotely and make informed decisions based on real-time information.



CONCLUSION

In this project report, I explored the integration of various sensors and their functionalities, providing a comprehensive understanding of their operational principles and applications in real-world scenarios. Through hands-on experimentation and implementation, I successfully demonstrated the behavior of fundamental logic gates, including AND and NAND gates, using Arduino platforms. This practical approach not only reinforced my theoretical knowledge but also enhanced my skills in circuit design and programming. Furthermore, I delved into the integration of these projects with ChirpStack, InfluxDB, and Grafana for data visualization. This integration allowed me to collect, store, and analyze data from my sensors effectively. By leveraging ChirpStack's capabilities for LoRaWAN communication, I established a robust framework for remote sensor management. InfluxDB served as a powerful time-series database, enabling efficient data storage and retrieval, while Grafana provided an intuitive interface for visualizing my data in real-time. Throughout this journey, I also explored several embedded libraries that facilitated the development of my projects. These libraries simplified complex tasks, allowing me to focus on the core functionalities of my applications. My experience with these tools has equipped me with valuable skills in embedded systems programming and data management. In conclusion, this project not only deepened my understanding of sensor technologies and logic gates but also highlighted the importance of data integration and visualization in modern electronics. The knowledge I gained from this report lays a strong foundation for future explorations in IoT (Internet of Things) applications and embedded systems development. I look forward to applying these insights in more advanced projects that harness the power of sensors and data analytics to create innovative solutions.