1. **Title - Write a program for sending alert messages to the user for controlling and interacting with your environment.**

To create a program for sending alert messages to control and interact with your environment in IIoT (Industrial Internet of Things), we'll develop a simple Python script that monitors a specific condition (e.g., temperature threshold) and sends alerts using email notifications. This example assumes you have a sensor providing data to your system and want to alert when a certain condition is met

Requirements:

Python (3.6 or higher recommended)

smtplib library (for sending emails)

Access to an email server (e.g., Gmail)

Explanation:

Email Configuration: Replace sender\_email, receiver\_email, and sender\_password with your own email credentials and the recipient's email address. read\_temperature() Function: Simulates reading temperature data. Replace this with actual sensor data retrieval logic in a real-world application. send\_alert() Function: Establishes a connection to the SMTP server (Gmail in this case), logs in with your email credentials, and sends an email with the specified subject and body. Main Program (main()): Continuously monitors the temperature every minute. If the temperature exceeds a threshold (30.0°C in this example), an alert email is sent using the send\_alert() function

#define TEMP\_PIN A0 // Pin where the TMP36 sensor is connected

#define BUZZER\_PIN 8 // Buzzer pin

const float TEMPERATURE\_THRESHOLD = 23.0; // Temperature threshold in Celsius

void setup() {

// Initialize the buzzer pin as an output

pinMode(BUZZER\_PIN, OUTPUT);

// Start the Serial Monitor for debugging

Serial.begin(9600);

}

void loop() {

// Read the temperature from the TMP36 sensor

int tempReading = analogRead(TEMP\_PIN);

float voltage = tempReading \* (5.0 / 1023.0);

float temperatureC = (voltage - 0.5) \* 100.0;

// Print the temperature to the Serial Monitor

Serial.print("Temperature: ");

Serial.print(temperatureC);

Serial.println(" C");

// Check if the temperature exceeds the threshold

if (temperatureC > TEMPERATURE\_THRESHOLD) {

// Turn on the buzzer

digitalWrite(BUZZER\_PIN, HIGH);

// Print an alert message to the Serial Monitor

Serial.println("ALERT: Temperature is too high!");

} else {

// Turn off the buzzer

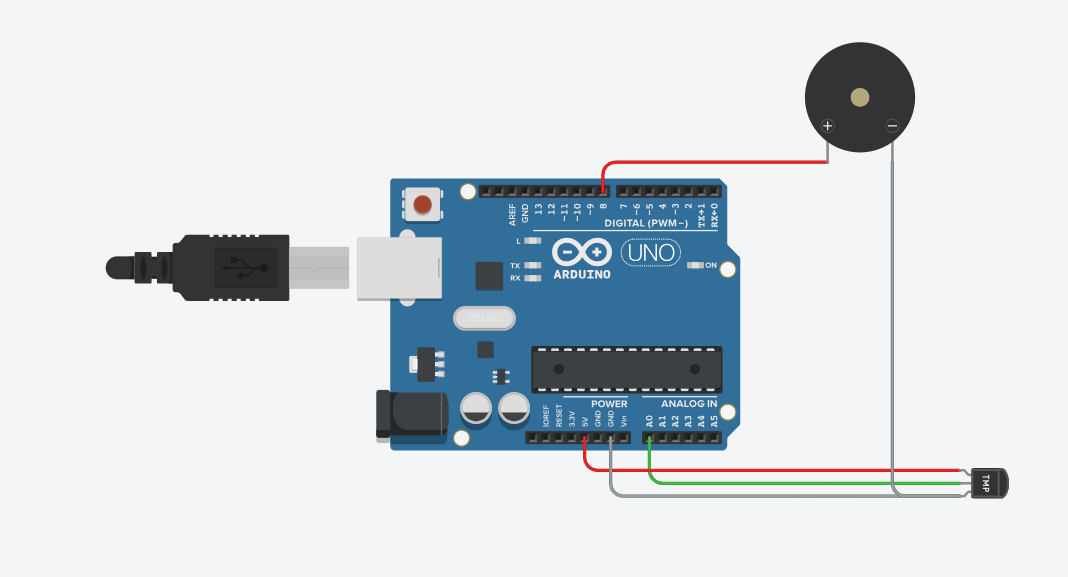
digitalWrite(BUZZER\_PIN, LOW);

}

// Wait for a short period before the next loop

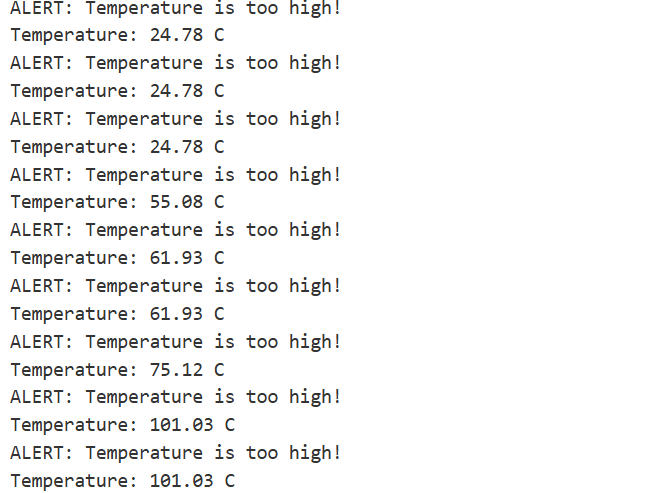
 delay(1000);

}

****

https://www.tinkercad.com/things/c5NG5jgeVYH-brave-gaaris-allis/editel?returnTo=https%3A%2F%2Fwww.tinkercad.com%2Fdashboard%2Fdesigns%2Fcircuits&sharecode=jof6jgo9sihUaoBekn3S0O2fIl1jY26sFyXB2HCNGKI

**Output:**

****

1. **Title - Write an Arduino/ Raspberry pi program for interfacing with PIR sensor Experiment.**

Problem Statement - Design an Arduino program to interface with a PIR (Passive Infrared)sensor on Tinkercad. Simulate the detection of motion using the PIR sensor and visualize theresults through serial communication or a virtual LED.

Prerequisite – C/C++ programming, basic understanding of conditionals, loops, and Tinkercad environment

Software Requirements - Tinkercad simulation platform, internet connection, computer or compatible device.

Hardware Requirements – Arduino UNO R3, Breadboard Small, Jumpre Wires, Resistor, PIR Sensor.

Learning Objectives – Learn to Sensor interfacing, event detection, visualization, Arduino programming, Tinkercad simulation.

Outcomes - After Completion of this assignment students are able to Motion detection simulation, data visualization, Arduino proficiency, IoT understanding.

Theory - The Passive Infrared (PIR) sensor is a fundamental component used for motion detection by measuring infrared radiation changes. It is widely utilized in applications ranging from security systems to automated lighting. In this experiment, we will interface a PIR sensor with an Arduino microcontroller to detect motion and provide a visual indication using an LED. The setup will also involve a resistor to ensure proper current flow through the LED. This experiment demonstrates the practical integration of these components and the basics of digital input-output operations with the Arduino.

Working Principle of PIR Sensor The PIR sensor operates based on the principle of detecting infrared radiation. All objects emit infrared radiation as a function of their temperature. The PIR sensor includes a pyroelectric material that generates a voltage when exposed to infrared radiation. This voltage change is processed internally and converted into a digital signal.

-Detection Range and Angle: PIR sensors have a specified field of view and detection range, usually determined by the lens and design of the sensor. The typical range is between 3 to 12 meters, and the field of view is approximately 120 degrees.

-Signal Output: The sensor outputs a HIGH signal (1) when motion is detected and a LOW signal (0) when no motion is detected.

Components Overview

1.PIR Sensor: A PIR sensor detects motion by sensing changes in infrared radiation. It

typically has three pins:

-VCC: Power supply (5V).

-GND: Ground connection.

-OUT: Digital output indicating motion detection (HIGH or LOW).

2.Arduino Microcontroller: An Arduino board will read the digitalsignal from the PIR sensor and control the LED based on the sensor's output. 3.LED: A Light Emitting Diode (LED) will visually indicate motion detection. 4.Resistor: A resistor is used to limit the current flowing through the LED to prevent damage. For a standard LED, a 220-ohm resistor is typically used. 5.Breadboard: A breadboard is used to build the circuit by connecting the components without soldering.

Circuit Design and Wiring 1.Wiring the PIR Sensor: Connect the VCC pin of the PIR sensor to the 5V pin on the Arduino. Connect the GND pin of the PIR sensor to the GND pin on the Arduino. Connect the OUT pin of the PIR sensor to a digital input pin on the Arduino (e.g., pin 2). 2.Connecting the LED: Connect the anode (longer leg) of the LED to a digital output pin on the Arduino (e.g., pin 13). Connect the cathode (shorter leg) of the LED to one end of a 220-ohm resistor.

Connect the other end of the resistor to GND on the Arduino. 3.Breadboard Layout: Use the breadboard to make connections more organized and ensure stable connections. Insert the PIR sensor, LED, and resistor into the breadboard and make connections according to the wiring instructions above. Testing and Observations 1.Build the Circuit: Assemble the circuit on a breadboard following the wiring instructions. 2.Upload and Run the Code: Upload the provided code to the Arduino using the Arduino IDE and start the program. 3.Observe the LED and Serial Monitor:

When motion is detected within the PIR sensor’s field of view, the LED should light up, and the serial monitor should display "Motion detected!" When no motion is detected, the LED will turn off, and the serial monitor will display "No motion detected."

Conclusion – This experiment demonstrates the integration of a PIR sensor with an Arduino to create a basic motion detection system. The use of a PIR sensor, LED, resistor, and Arduino provides a comprehensive understanding of digital input-output operations and practical circuit design.

// Define pin numbers

const int pirPin = 2; // PIR sensor input pin

const int ledPin = 13; // LED output pin (built-in on many Arduino boards)

void setup() {

pinMode(pirPin, INPUT); // Set PIR pin as input

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

Serial.begin(9600); // Initialize serial communication for debugging

}

void loop() {

int pirState = digitalRead(pirPin); // Read PIR sensor state

if (pirState == HIGH) { // If motion is detected

digitalWrite(ledPin, HIGH); // Turn on LED

Serial.println("Motion detected!");

} else {

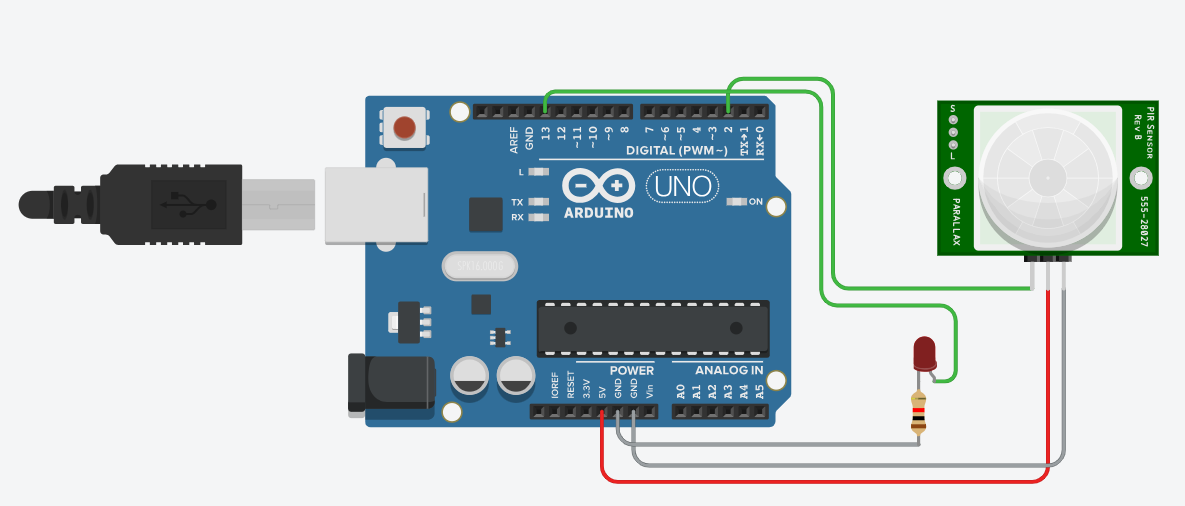
digitalWrite(ledPin, LOW); // Turn off LED

Serial.println("No motion");

}

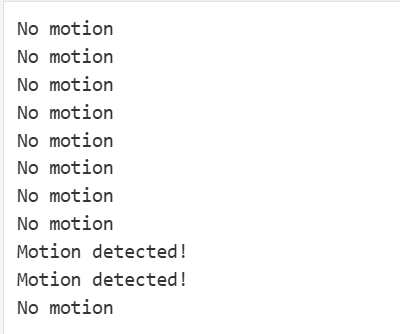
delay(500); // Wait for half a second before rechecking

}

****

**https://www.tinkercad.com/things/5QvFtJrEK9N-terrific-hillar-jaiks/editel?returnTo=https%3A%2F%2Fwww.tinkercad.com%2Fdashboard%2Fdesigns%2Fcircuits&sharecode=e7ONVJMyF3JTtU50\_vsp-gpr2wDClsy6KE571amp2P4**

**Output:**

****

1. **Title - Write a program for developing an IIoT application for energy monitoring and optimization.**

Developing an IIoT (Industrial Internet of Things) application for energy monitoring and optimization involves integrating sensors, data processing, analysis, and control mechanisms to enhance energy efficiency and reduce costs in industrial settings. Below is a structured approach to create such an application using Python for data handling and visualization:

Requirements:

1.Python (3.6 or higher recommended)

2. MQTT library (for MQTT communication)

3. MongoDB library (for data storage and retrieval)

4. Dash library (for creating interactive web-based dashboards)

Components of the IIoT Application:

1. Data Acquisition from Sensors

In this example, we'll simulate sensor data for energy consumption metrics (e.g., power usage, voltage, current). Replace this with actual sensor data retrieval logic in a real-world setup

1. MQTT Integration for Data Transmission MQTT facilitates communication between sensors, devices, and the IIoT application. Install the paho-mqttlibrary for Python
2. Data Storage in MongoDB Store sensor data in MongoDB for historical analysis and optimization. Install the pymongolibrary for Python.
3. Visualization with Dash Dash is used to create a web-based dashboard for real-time monitoring and analysis of energy data. Install the dash and dash\_core\_components libraries for Python.

Explanation: • Data Acquisition: Simulated sensor data (get\_sensor\_data()) for power usage, voltage, and current is periodically published to an MQTT topic (IIoT/energy/data) using the paho-mqtt library. • MQTT Integration: paho-mqtt library connects to an MQTT broker (mqtt.eclipse.org) and publishes sensor data. • Data Storage: Sensor data is stored in MongoDB Atlas using the pymongolibrary. The store\_in\_mongodb() function inserts sensor data into the energy\_data collection. • Visualization: Dash creates a real-time interactive dashboard (app.layout) that displays energy consumption metrics (power\_usage, voltage, current) retrieved from MongoDB. The update\_graph()function updates the graph every 5 seconds with new data retrieved from MongoDB.

Running the Application: 1. Save the scripts (send\_sensor\_data.py for publishing sensor data, dashboard.py for Dash visualization).2. Replace placeholders (mongo\_username, mongo\_password, mongo\_cluster) with your MongoDB Atlas credentials. 3. Start the MQTT data publisher (python send\_sensor\_data.py) and Dash dashboard (python dashboard.py) scripts. 4. Access the Dash application in your web browser (http://127.0.0.1:8050/) to monitor real-time energy consumption metrics

float x, y, z, temp;

void setup()

{

pinMode(5, OUTPUT);

pinMode(6, OUTPUT);

pinMode(A5, INPUT);

pinMode(A4, INPUT);

Serial.begin(9600);

}

void loop()

{

// Read values from analog inputs

y = analogRead(A5);

z = analogRead(A4);

// Print values to the Serial Monitor for debugging

Serial.print("y: ");

Serial.println(y);

Serial.print("z: ");

Serial.println(z);

// Calculate temperature based on reading from A4

temp = (double)z / 1024.0;

temp = temp \* 5.0;

temp = temp - 0.5;

temp = temp \* 100.0;

Serial.print("Temperature: ");

Serial.println(temp);

// Logic to control pins based on y and temp values

if ((y < 550) && (temp > 30))

{

digitalWrite(5, HIGH);

digitalWrite(6, HIGH);

}

else if ((y < 550) && (temp <= 30))

{

digitalWrite(5, HIGH);

digitalWrite(6, LOW);

}

else if ((y >= 550) && (temp > 30))

{

digitalWrite(5, LOW);

digitalWrite(6, HIGH);

}

else if ((y >= 550) && (temp <= 30))

{

digitalWrite(5, LOW);

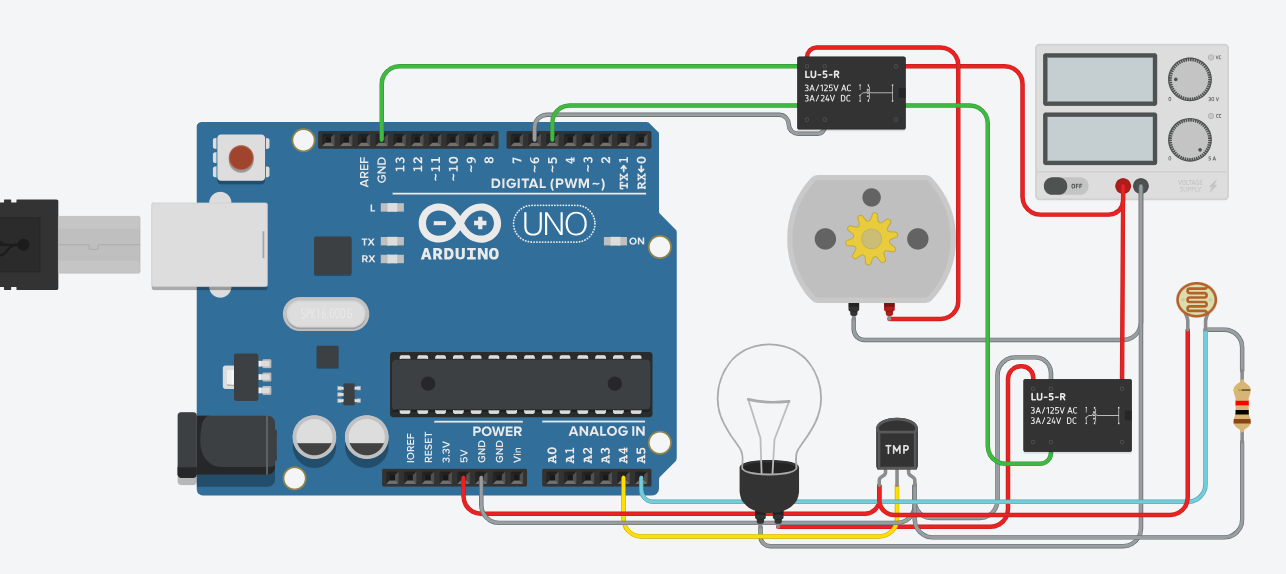
digitalWrite(6, LOW);

}

// Add a short delay to avoid excessive Serial output

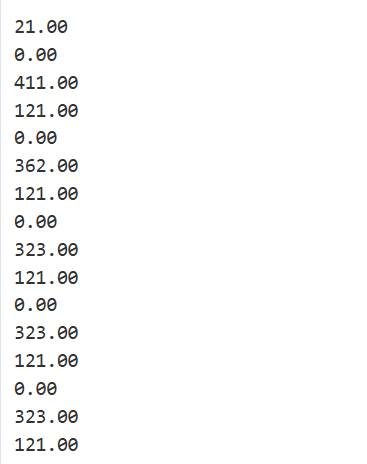
  delay(500);

}

****

[**https://www.tinkercad.com/things/ky1dlhoBYgV-incredible-blad-bojo/editel?returnTo=https%3A%2F%2Fwww.tinkercad.com%2Fdashboard%2Fdesigns%2Fcircuits&sharecode=QuMx9owIfe6QuotwZa3O\_qCVTph3Q06wzKI3NQzgpQ4**](https://www.tinkercad.com/things/ky1dlhoBYgV-incredible-blad-bojo/editel?returnTo=https%3A%2F%2Fwww.tinkercad.com%2Fdashboard%2Fdesigns%2Fcircuits&sharecode=QuMx9owIfe6QuotwZa3O_qCVTph3Q06wzKI3NQzgpQ4)

**Output:**

****

1. **Title - Write a program for implementing security measures in an IIoT system.**

Problem Statement - Design and develop security measures for an IIoT system on Tinkercad, including authentication, authorization, encryption, data integrity, and monitoring.

Prerequisite – C/C++ programming, basic understanding of conditionals, loops, and Tinkercad environment.

Software Requirements - Tinkercad simulation platform, internet connection, computer or compatible device.

Hardware Requirements - Arduino Uno R3, Resistor, LED, Breadboard Small, Gas sensor, Potentiometer, Piezo, TMP Sensor, Jumper Wires, and LCD 16/2.

Learning Objectives - Implement authentication and authorization protocols. Develop data encryption techniques. Design data integrity verification methods. Create audit and monitoring systems.

Outcomes - After Completion of this assignment students are able to Secure authentication Role-based authorization, Data encryption, Integrity verification, and Activity monitoring

Theory - The Industrial Internet of Things (IIoT) represents a significant advancement in connecting and managing a wide range of devices and sensorsto enhance operational efficiency and data collection. However, with these advancements come substantial security concerns, as interconnected systems are vulnerable to unauthorized access, data breaches, and other cyber threats. Implementing robust security measures is essential for protecting these systems. This experiment demonstrates how to design and implement security measures in an IIoT system using Arduino, breadboard, various sensors (gas and temperature), resistors, LEDs, LCDs, a potentiometer, a piezo buzzer, and jumper wires, all within the Tinkercad simulation environment.

Components and Their Functions 1.Arduino Microcontroller: Function: Acts as the central processing unit of the IIoT system. It handles data from sensors, processes security measures, and controls outputs based on programmed security protocols. 2.Breadboard: Function: Provides a platform for prototyping the circuit without soldering. It allows for easy connections and modifications of the components.

3.Gas Sensor: Function: Monitors the presence of gases in the environment. This sensor can be used to detect potential hazards or unauthorized presence based on gas levels, providing an additional layer of security. 4.Temperature Sensor (TMP): Function: Measures the ambient temperature. In security contexts, it can detect environmental changes that might indicate tampering or system failure.

5.Resistors: Function: Control the flow of current through various components, protecting sensitive parts from damage and ensuring proper operation. 6.LEDs: Function: Provide visual feedback for various system statuses, such as authentication success, alert signals, or system errors. 7.LCD: Function: Displays real-time information, such as system status, alerts, or authentication messages. It helps in monitoring and interacting with the system. 8.Potentiometer: Function: Acts as a variable resistor that can be adjusted to simulate different conditions or settings, such as sensitivity levels for sensors.

9.Piezo Buzzer: Function: Generates sound alerts for notifications or warnings, such as unauthorized access attempts or system errors. 10. Jumper Wires: Function: Connect various components on the breadboard and to the Arduino, ensuring proper circuit connectivity. System Design and Security Measures 1.Authentication and Authorization: Implementation: Authentication can be implemented using a simple password mechanism or code. The Arduino can be programmed to check a predefined password input against a stored value.

Components Used: The LCD displays prompts and status messages, while LEDs indicate authentication success or failure. 2.Data Encryption: Implementation: Although full encryption may be complex, a simplified encryption scheme can be used to demonstrate basic principles. For example, XOR-based encryption can be applied to sensor data before transmission. Components Used: Arduino handles the encryption/decryption process, ensuring that sensor data remains secure. 3.Data Integrity Verification: Implementation: Use checksums or simple hash functions to verify the integrity of transmitted data. This can ensure that data has not been altered during transmission. Components Used: The Arduino processes and verifies data from sensors, comparing checksums before taking action. 4.Audit and Monitoring: Implementation: Logs system activity, such as sensor readings and authentication attempts, to provide a record of events. This helps in identifying and investigating potential security issues. Components Used: The LCD and LEDs display real-time monitoring data, while the piezo buzzer alerts users to critical events. 5.Alert Mechanisms: Implementation: Set up visual and auditory alerts to notify users of security breaches, system failures, or unauthorized access attempts. Components Used: LEDs and the piezo buzzer provide immediate feedback and alerts. Circuit Design and Implementation

1.Building the Circuit: Breadboard Setup: Arrange the Arduino, sensors, and other components on the breadboard. Connect the gas sensor, temperature sensor, and potentiometer to the Arduino using jumper wires. Wiring: Connect the sensors' output pins to Arduino analog or digital input pins. Connect the LCD, LEDs, and piezo buzzer to appropriate output pins on the Arduino. Power Supply: Ensure that the power supply to the Arduino and components is stable and within the required voltage range. 2.Programming the Arduino: Authentication Code: Write code to handle user authentication, checking input values against predefined credentials and updating the LCD and LED status accordingly. Encryption and Decryption: Implement a simple encryption algorithm for sensor data, ensuring that data integrity is maintained. Data Integrity: Write functions to calculate and verify checksums or hashes for transmitted data.

Logging and Alerts: Program the Arduino to log system activities and provide alerts via the LCD and piezo buzzer. Computer Laboratory II B.E.AI&DS-SEM VII [A.Y.2024-25] Department of Artificial Intelligence and Data Science ,SVCET,RAJURI 3.Testing and Calibration: Component Testing: Test each component individually to ensure proper operation. Check sensor readings, authentication processes, and alert mechanisms. Calibration: Adjust the potentiometer to calibrate sensor sensitivity levels. Verify that dataencryption and integrity verification are functioning correctly. 4.Security Analysis: Evaluate Security Measures: Assess the effectiveness of implemented security features. Identify any potential vulnerabilities or areas for improvement. Document Findings: Prepare a report detailing the security measures, implementation process, test results, and any observed issues

Conclusion - This experiment demonstrates the implementation of security measures in an IIoT system using Arduino and various components within Tinkercad. By integrating authentication, authorization, encryption, data integrity verification, and monitoring, the system provides a comprehensive approach to securing IIoT operations

#include <LiquidCrystal.h>

int gas;

int wait = 100;

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

void setup() {

lcd.begin(16, 2); // Initialize the 16x2 LCD

pinMode(9, OUTPUT); // Set pin 9 as an output (for alarm/buzzer)

pinMode(A0, INPUT); // Set A0 as an input (for gas sensor)

}

void loop() {

gas = analogRead(A0); // Read gas sensor value from A0

lcd.clear(); // Clear the LCD screen

lcd.setCursor(0, 0);

lcd.print("Gas level = ");

lcd.print(gas);

if (gas > 680) { // If gas level exceeds threshold

digitalWrite(9, HIGH); // Turn on the alarm/buzzer

lcd.setCursor(0, 1);

lcd.print("Oh no a gas leak");

} else {

digitalWrite(9, LOW); // Turn off the alarm/buzzer

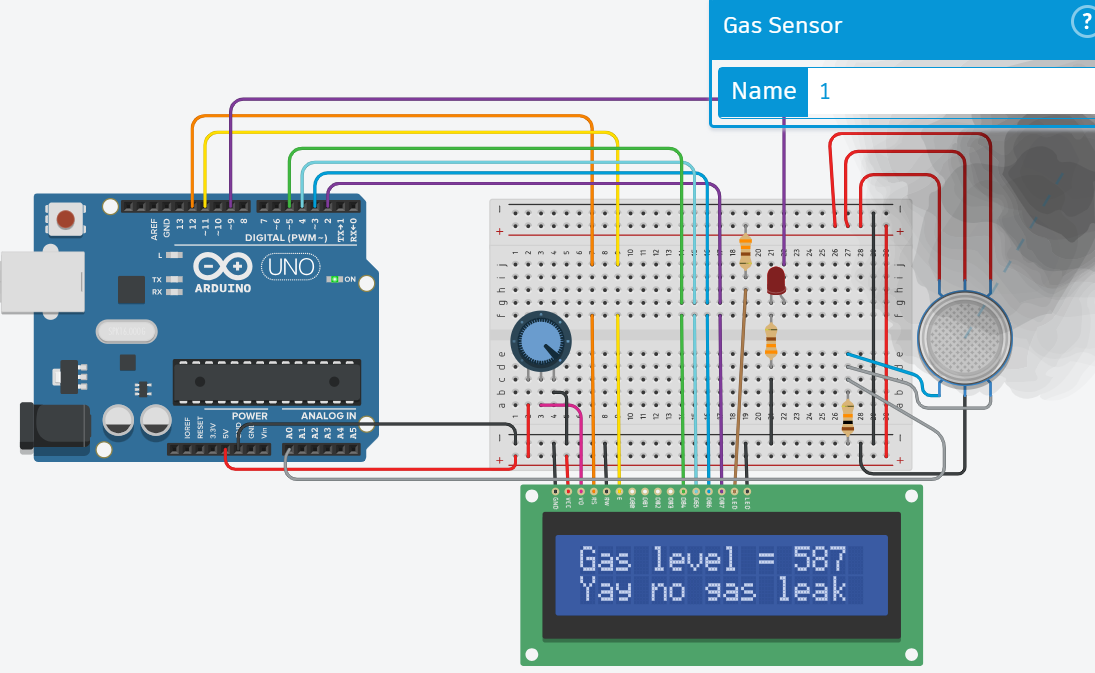
lcd.setCursor(0, 1);

lcd.print("Yay no gas leak");

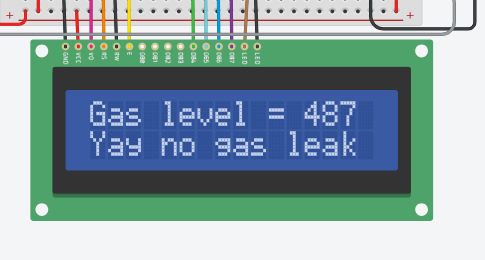
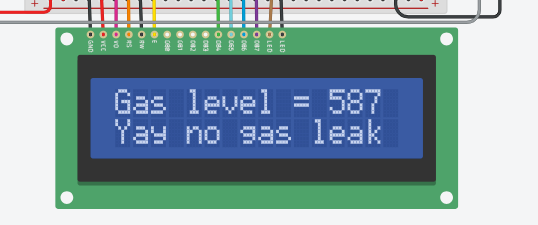
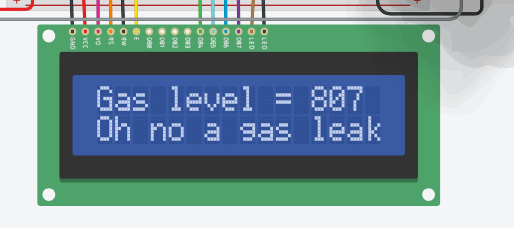
}

delay(wait); // Delay before updating the reading

}



**https://www.tinkercad.com/things/ahxdQ4Ld9oy-exquisite-blad-fulffy/editel?returnTo=https%3A%2F%2Fwww.tinkercad.com%2Fdashboard%2Fdesigns%2Fcircuits&sharecode=Kn0uFgXXN4T5lq2LK3uBpga8eGoCZ0yx8Kk2sLFRs7U**

**Output:**

1. **Write a program for sending sensor data to the cloud and storing it in a database**

In an IIoT (Industrial Internet of Things) environment, sending sensor data to the cloud and storing it in a database is a common requirement for data analysis, monitoring, and decision-making. Below is a Python program that demonstrates how to send simulated sensor data to a cloud platform (using MQTT for messaging) and store it in a cloud-based database (MongoDB Atlas as an example).

Requirements: 1. Python (3.6 or higher recommended) 2. paho-mqtt library (for MQTT communication) 3. pymongo library (for MongoDB interaction) 4. Access to an MQTT broker (e.g., Mosquitto) and MongoDB Atlas account

Steps to Create the Program:

1. Install Required Libraries Ensure you have Python installed, then install the necessary libraries: pip install paho-mqtt pymongo

2. Setting up MongoDB Atlas • Create a MongoDB Atlas cluster and database. • Whitelist your IP address to allow access to the cluster. • Create a MongoDB user with appropriate permissions for the database.

3. Python Program

Explanation: • MQTT Configuration: Replace broker\_address with the address of your MQTT broker. The program connects to the broker (mqtt.eclipse.org in this example) and publishes sensor data (sensor\_data) to the topic IIoT/sensor/data. • MongoDB Atlas Configuration: Replace mongo\_username, mongo\_password, and mongo\_cluster with your MongoDB Atlas credentials and cluster details. Data is stored in the IIoT database and sensor\_data collection. • get\_sensor\_data() Function: Simulates sensor data (temperature and humidity in this case) with a timestamp. • MQTT Callbacks: on\_connect and on\_publish handle connection to the MQTT broker and publishing of data. • send\_to\_mongodb() Function: Connects to MongoDB Atlas using pymongo and inserts sensor data into the specified collection. • Main Program (main()): Runs an infinite loop to continuously generate sensor data, publish it via MQTT, and store it in MongoDB Atlas every 5 seconds (adjust as needed).

Running the Program:

1. Save the script (e.g., send\_sensor\_data.py) and run it using Python (python send\_sensor\_data.py). 2. Monitor the console output for messages confirming data publication and insertion into MongoDB Atlas.

3. Check your MongoDB Atlas database to verify that sensor data is being stored correctly.

String ssid = "Simulator Wifi"; // SSID to connect to

String password = ""; // No password for the virtual WiFi

String host = "api.thingspeak.com"; // ThingSpeak API host

const int httpPort = 80; // HTTP port

String uri = "/update?api\_key=HD64O90IN8CFLR1Q&field1="; // Base URI for ThingSpeak API

// Function to setup ESP8266

int setupESP8266(void) {

Serial.begin(115200); // Serial connection to ESP8266

Serial.println("AT"); // Test AT command to check connection

delay(10);

if (!Serial.find("OK")) return 1;

// Connect to WiFi

Serial.println("AT+CWJAP=\"" + ssid + "\",\"" + password + "\"");

delay(10);

if (!Serial.find("OK")) return 2;

// Open TCP connection to the host

Serial.println("AT+CIPSTART=\"TCP\",\"" + host + "\"," + httpPort);

delay(50);

if (!Serial.find("OK")) return 3;

return 0;

}

// Function to send temperature data to ThingSpeak

void anydata(void) {

int temp = map(analogRead(A0), 20, 358, -40, 125); // Map analog value to temperature

// Construct HTTP request with temperature value appended to the URI

String httpPacket = "GET " + uri + String(temp) + " HTTP/1.1\r\nHost: " + host + "\r\nConnection: close\r\n\r\n";

int length = httpPacket.length();

// Send the message length to ESP8266

Serial.print("AT+CIPSEND=");

Serial.println(length);

delay(10); // Wait for ESP8266 response

if (!Serial.find(">")) return;

// Send HTTP request

Serial.print(httpPacket);

delay(10); // Wait for ESP8266 response

if (!Serial.find("SEND OK\r\n")) return;

}

// Setup function

void setup() {

int result = setupESP8266();

if (result != 0) {

Serial.println("ESP8266 setup failed with error code: " + String(result));

while (1); // Halt further execution if setup fails

}

Serial.println("ESP8266 setup complete.");

}

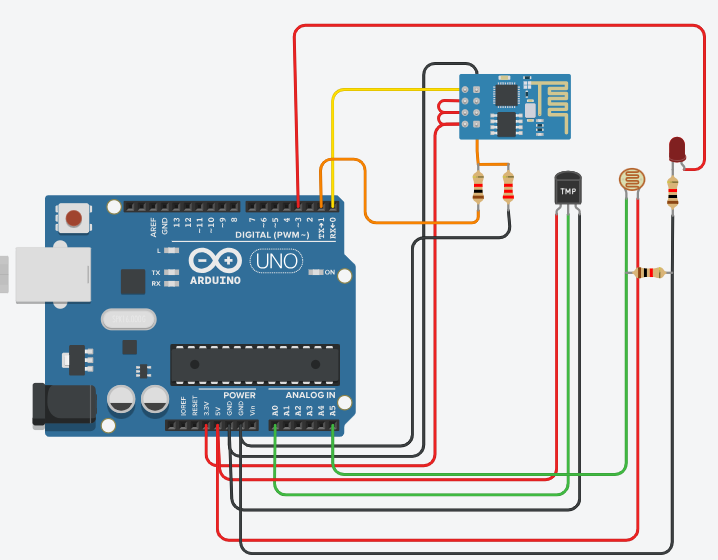
// Loop function

void loop() {

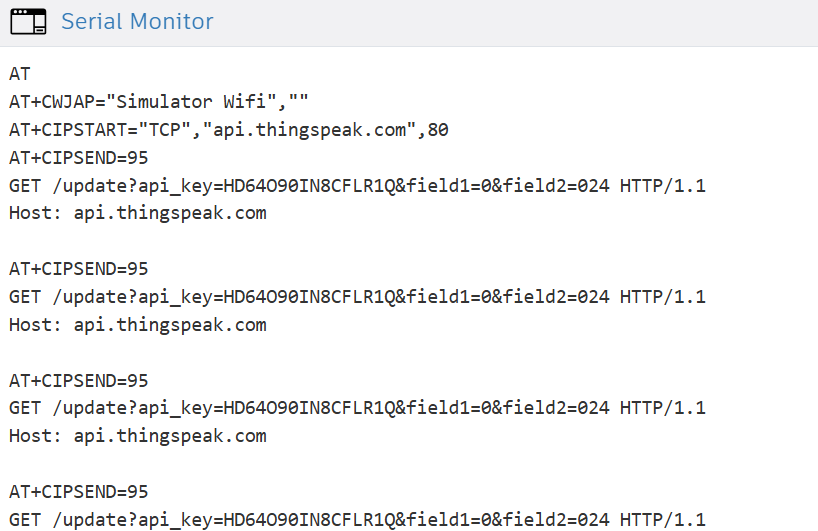
anydata(); // Send data to ThingSpeak

delay(2000); // Wait for 2 seconds before sending data again

}



https://www.tinkercad.com/things/ep49J2Bp1r6-copy-of-wifi-module-esp8266-/editel?returnTo=https%3A%2F%2Fwww.tinkercad.com%2Fdashboard%2Fdesigns%2Fcircuits&sharecode=8vq4ZgQBB1w27wkIQ4UGF9x3wIybi7OkT0-4rcDd6Jc

**Output:**

1. **Write a program for building a small-scale IIoT network using wireless communication protocols.**

To build a small-scale IIoT network using wireless communication protocols, we'll create a simple example using Python and some popular libraries for IoT and networking. For this example, we'll simulate two devices communicating wirelessly over WiFi using MQTT (Message Queuing Telemetry Transport) protocol. MQTT is widely used in IoT scenarios due to its lightweight nature and efficient publishsubscribe messaging model.

Requirements: Python (3.6 or higher recommended) paho-mqtt library (for MQTT communication) Two devices (simulated as two Python scripts running on your computer)

Steps: 1. Install Required Libraries First, you need to install the paho-mqtt library. You can install it using pip: pip install paho-mqtt 2. Implementing the Publisher (Device 1) Create a Python script (publisher.py) for Device 1. This device will publish data to a topic on an MQTT broker (server).

3.Implementing the Subscriber (Device 2) Create another Python script (subscriber.py) for Device 2. This device will subscribe to the same topic and receive data published by Device 1.

Running the Program: Open two terminal windows (or two Python IDEs) for Device 1 (publisher.py) and Device 2 (subscriber.py). Run publisher.py in one terminal or IDE. Run subscriber.py in the other terminal or IDE.

Explanation: Publisher (publisher.py): Simulates sensor data (temperature in this case) using get\_sensor\_data() function. Connects to an MQTT broker (test.mosquitto.org in this example) and publishes the sensor data to topic IIoT/temperature every 5 seconds. Subscriber (subscriber.py): Connects to the same MQTT broker and subscribes to topic IIoT/temperature. Receives and prints the sensor data published by the publisher.

#include <IRremote.hpp>

const int rcvPin=3;

IRrecv irrecv(rcvPin);

decode\_results results;

void setup()

{

Serial.begin(9600);

irrecv.enableIRIn(); // Start the receiver

pinMode(5, OUTPUT);

}

void loop() {

if(IrReceiver.decode()) {

auto value= IrReceiver.decodedIRData.decodedRawData;

//switch(results.value)

switch(value)

{

case 4010852096:

Serial.println("1"); // Button 1

digitalWrite(5,HIGH);

break;

case 3994140416: // Template

Serial.println("2"); // Button

digitalWrite(5,LOW);

break;

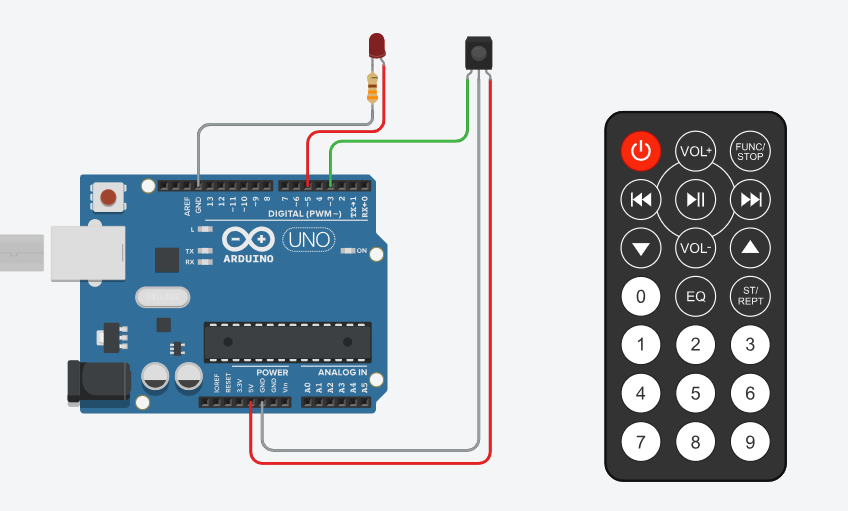
default: Serial.println(value);

}

IrReceiver.resume(); // Receive the next value

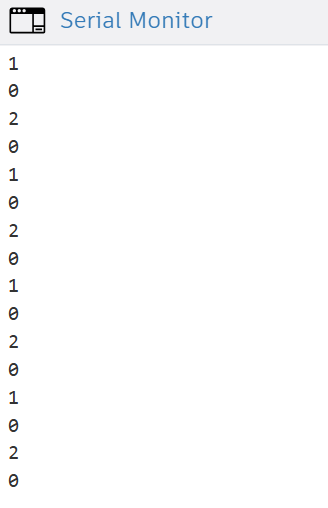
  }

}

**Circuit**

**https://www.tinkercad.com/things/0Z32x2xbfZP-copy-of-ir-remote-example-using-arduino-irremote-3x-version/editel?returnTo=%2Fthings%2F0Z32x2xbfZP-copy-of-ir-remote-example-using-arduino-irremote-3x-version&sharecode=LS3CTW0sJ5eNX1SVJ0us4bQjPJPMO4Ja0PXx2lEbYPo**

**Output:**

****