Assignment No. 03

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

Problem Statement – Design and Development of an IIoT Application for Smart Home Energy Monitoring and Optimization.

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, Light Bulb, TMP Sensor, Jumpre Wires, Resistor, Dc Motor, Power Supply, and Relay.

Learning Objectives – 1.Understand IoT fundamentals and device integration.

2. Implement real-time data acquisition and visualization techniques.

3. Design user interfaces for effective interaction and feedback.

Outcomes - After Completion of this assignment students are able to Real-time energy monitoring, User-friendly interface, Smart device integration, and Secure data handling should be done.

Theory - The rapid advancement of technology has significantly impacted how we manage and optimize energy consumption in modern homes. The Industrial Internet of Things (IIoT) leverages interconnected devices and sensors to monitor, control, and optimize energy usage in real-time. This experiment focuses on developing a prototype IIoT application using Arduino, a light bulb, a power supply, a relay, a temperature sensor (TMP), a DC motor, and resistors. The goal is to create a system that monitors energy consumption, controls electrical devices, and optimizes their operation to improve efficiency.

Components and Their Functions

1. **Arduino Microcontroller:** The Arduino microcontroller is the central unit of the system. It processes input from sensors and controls outputs to devices like relays and motors. Arduino is chosen for its simplicity, versatility, and ease of integration with various sensors and actuators.

2. **Light Bulb:** The light bulb represents a typical home appliance. Its power consumption will be monitored and controlled to demonstrate energy management and optimization.

3. **Power Supply:** The power supply provides the necessary electrical power for the light bulb, Arduino, and other components. It ensures that the system operates within safe voltage and current limits.

3. **Relay:** The relay acts as a switch that controls the light bulb. It allows the Arduino to turn the light bulb on or off based on sensor readings and optimization algorithms.

4. **Temperature Sensor (TMP):** The TMP sensor measures the ambient temperature. Temperature data is crucial for understanding the thermal impact on energy consumption and for implementing optimization strategies.

5. **DC Motor:** The DC motor can simulate additional load or appliance. It represents another common household device whose power usage can be monitored and optimized.

6. **Resistor:** Resistors are used to control current flow and ensure safe operation of the components. They help in setting appropriate voltage levels for sensors and the Arduino.

System Design and Operation

The system design includes several key components and their interactions. Below is an overview of the design, operation, and implementation.

1. Data Acquisition:

The Arduino collects data from the temperature sensor to monitor the ambient temperature. It also measures power consumption indirectly through the relay and light bulb setup. By integrating a current sensor or power meter (which could be simulated by a resistor in this experiment), the Arduino can gauge the energy usage of the light bulb and DC motor.

2. Control Mechanism:

The Arduino controls the light bulb and DC motor via the relay. The relay acts as an intermediary switch that the Arduino can activate to turn the devices on or off. The relay's state is managed based on the data received from the temperature sensor and any optimization algorithms applied.

3. Optimization Algorithms:

The optimization algorithm can be designed to adjust the operation of the light bulb and DC motor based on real-time data. For instance, if the temperature is high, the algorithm might turn off the light bulb to reduce heat generation, thereby optimizing energy use. Alternatively, it could adjust the DC motor's operation to maintain efficient energy consumption.

4. User Interface:

Although the prototype might not include a graphical user interface, the Arduino can be programmed to output data to a serial monitor for real-time observation. For a complete system, a user interface could be developed to visualize data and control devices remotely.

5. Energy Monitoring and Reporting:

Energy monitoring involves tracking the power consumption of connected devices. The Arduino processes sensor data and relay states to compute energy usage. This information can be reported to users or logged for further analysis. In this setup, energy usage can be approximated based on the operation time and power rating of the light bulb and DC motor.

6. Safety Considerations:

Safety is crucial when working with electrical components. The power supply should be correctly rated for the components to prevent overloading. Relays should be selected based on their voltage and current ratings to handle the load of the light bulb and DC motor safely. Resistors must be used to limit current and protect sensitive components like sensors and the Arduino.

Implementation Steps

1. Component Setup:

Connect the power supply to the light bulb, relay, and DC motor, ensuring all connections are secure and correctly rated.

Attach the temperature sensor to the Arduino using appropriate pins and connections.

Integrate the relay with the Arduino to control the light bulb and motor.

Use resistors where necessary to protect components and ensure correct voltage levels.

2. Programming the Arduino:

Write the Arduino code to read temperature data from the TMP sensor.

Develop logic to control the relay based on temperature readings and optimization criteria.

Implement algorithms to monitor energy usage and adjust device operation accordingly.

Use serial communication to output data for real-time monitoring and analysis.

3. Testing and Calibration:

Test each component individually to ensure proper operation.

Calibrate the temperature sensor and verify that the relay controls the light bulb and motor accurately.

Validate the energy monitoring functionality and optimization algorithms.

4. Analysis and Optimization:

Analyze the data collected during tests to evaluate the performance of the optimization algorithms.

Adjust the algorithms as needed to improve energy efficiency and system response.

5. Documentation:

Document the system design, programming details, and test results.

Prepare a report on the energy monitoring and optimization performance, including any observed improvements in efficiency.

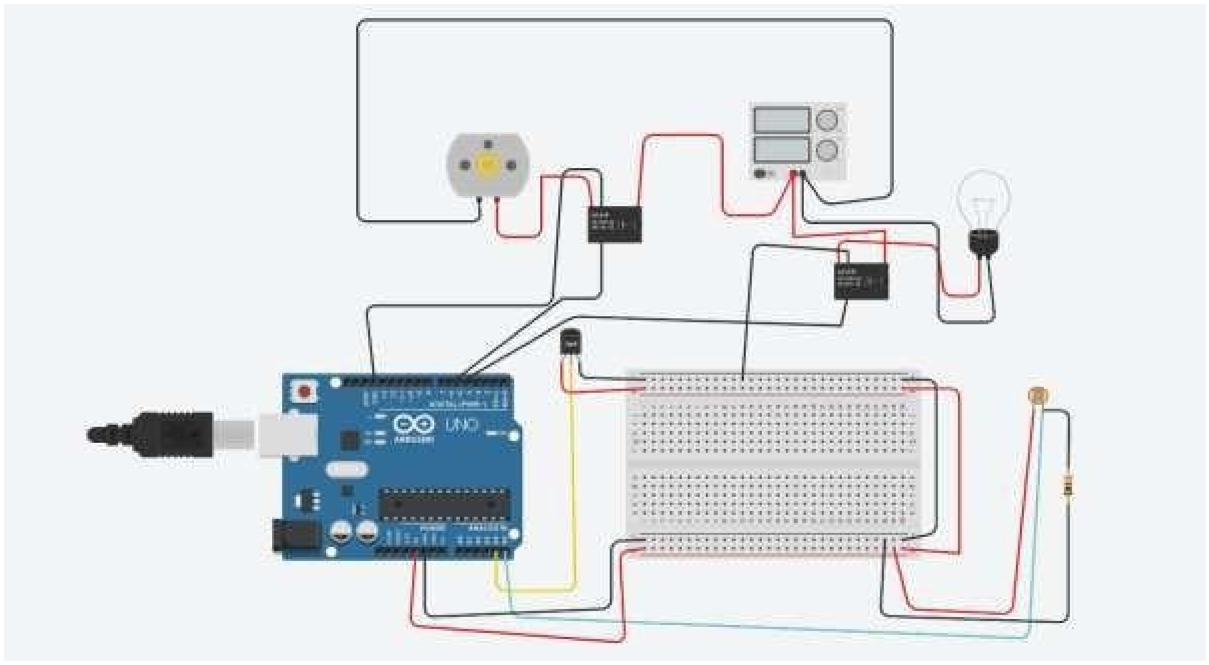
Source Code –

```
float x,y,z,temp;

void setup()

{

 // pinMode(8, INPUT);

  pinMode(5, OUTPUT);
  pinMode(6, OUTPUT);

  pinMode(A5,  INPUT);

pinMode(A4,    INPUT);

Serial.begin(9600);

}

void loop()

{

 // x= digitalRead(8);

  y= analogRead(A5);

  z= analogRead(A4);

  Serial.println(x);

  Serial.println(y);

  Serial.println(z);

  temp = (double)z / 1024;

  temp = temp * 5;

  temp = temp - 0.5;

  temp = temp * 100;

  //if ( (x>0) )

  //{
```

```
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);

}
```

```
 /*}
 else
 {
   digitalWrite(5, LOW);
   digitalWrite(6, LOW);
 }*/
}
```

Circuit Diagram -



Conclusion - This experiment provides a practical demonstration of IIoT applications for smart home energy management. By using Arduino and various components, the system enables real-time monitoring and optimization of energy consumption.

Assignment No. 04

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 sensors to 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.

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 data encryption and integrity verification are functioning correctly.

4.**Security Analysis:**

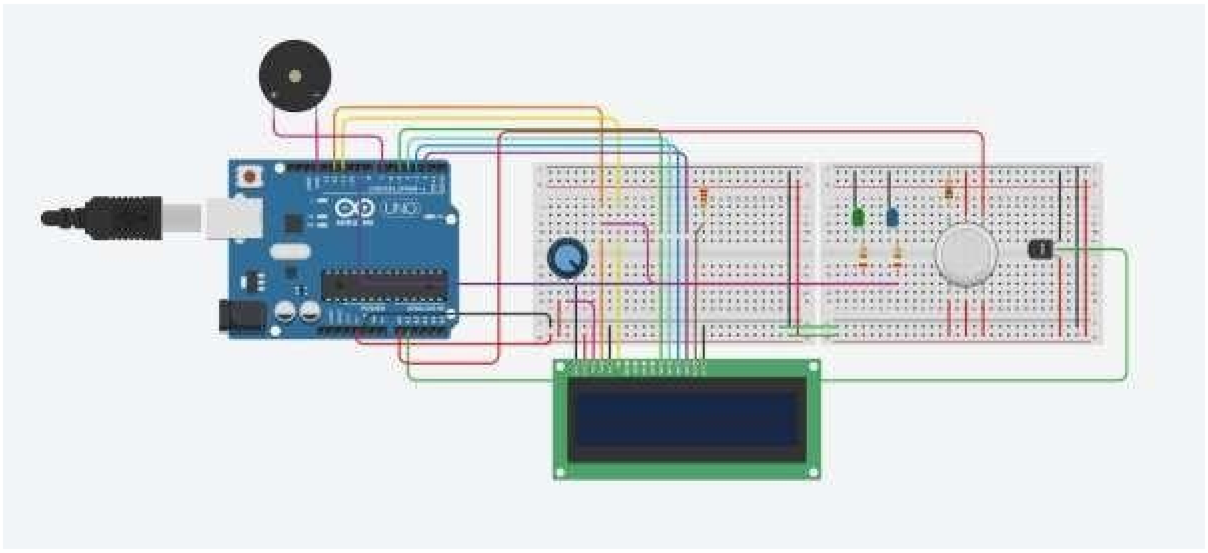
Evaluate Security Measures: Assess the effectiveness of implemented security features. Identify any potential vulnerabilities or areas for improvement.

Document Findings: Prepare a report detailing the security measures, implementation process, test results, and any observed issues.

Source Code –

```
#include <LiquidCrystal.h>

// Initialize the LCD with the pin numbers

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

int V_GasSen = 0;

int V_TempSens = 0;

void setup() {

  pinMode(A0, INPUT); // Gas sensor pin

  pinMode(A1, INPUT); // Temperature sensor pin

  pinMode(7, OUTPUT); // Buzzer pin

  pinMode(9, OUTPUT); // LED for gas detection

  pinMode(12, OUTPUT); // LED for temperature warning

 lcd.begin(16, 2); // Initialize the LCD with 16 columns and 2 rows

}
void loop() {
  // Read gas sensor value
  V_GasSen = analogRead(A0);
// Read temperature sensor value and calculate temperature
  V_TempSens = -40 + 0.488155 * (analogRead(A1) - 20);
// Display temperature and gas status on the LCD
  lcd.clear(); // Clear the LCD
  lcd.setCursor(0, 0); // Set cursor to the first row
 lcd.print("Temperature: "); // Print temperature label
 lcd.print(V_TempSens); // Print temperature value
 lcd.print(" C"); // Print temperature unit
 lcd.setCursor(0, 1); // Set cursor to the second row
 lcd.print("Gas: "); // Print gas label
  lcd.print(V_GasSen); // Print gas sensor value
 // Check for alerts
  if (V_GasSen >= 250) {
    tone(7, 523, 1000); // Play tone if gas is detected
    digitalWrite(9, HIGH); // Turn on the gas detection LED
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("ALERT: Gas Detected");

  } else {

    digitalWrite(9, LOW); // Turn off the gas detection LED

  }
  if (V_TempSens >= 70) {

    tone(7, 523, 1000); // Play tone if temperature exceeds the threshold

    digitalWrite(12, HIGH); // Turn on the temperature warning LED

    lcd.clear();

    lcd.setCursor(0, 0);

    lcd.print("ALERT: Temp High");

  } else {

    digitalWrite(12, LOW); // Turn off the temperature warning LED

  }
delay(1000); // Delay for one second

}
```

Circuit Diagram –



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.