Hands-On Guide: Data Acquisition for IoT in the Energy Sector

Hands-On 1: Setting up a Basic IoT Data Acquisition System

Objective

To set up a basic data acquisition system using sensors and a microcontroller, collect energy data, and send it to the cloud for monitoring.

Materials

- Microcontroller (e.g., ESP32 or Arduino)
- Energy sensor (e.g., ACS712 or PZEM-004T)
- Breadboard and jumper wires
- Internet connection
- Cloud platform (e.g., ThingSpeak, Blynk, or MQTT broker)
- Power source

Steps

1. Connect the Sensor to the Microcontroller
Follow the wiring guidelines for your sensor (e.g., ACS712 to analog pin A0 on an Arduino or ESP32).

2. Programming the Microcontroller
Upload the following code to read data from the sensor and display it in the Serial Monitor:

Listing 1: Code for Reading ACS712 Sensor

```
const int analogInPin = A0;
int sensorValue = 0;

void setup() {
```

```
  Serial.begin(9600);
}

void loop() {
  sensorValue = analogRead(analogInPin);
  float voltage = (sensorValue * 5.0) / 1023.0;
  float current = (voltage - 2.5) / 0.185;
  Serial.print("Current:");
  Serial.println(current);
  delay(1000);
}
```

3. Connect to the Cloud

Set up an account on a cloud platform such as ThingSpeak. Modify your microcontroller code to send data to the cloud. Example for ESP32:

Listing 2: ESP32 Code for ThingSpeak

```
#include <WiFi.h>
#include "ThingSpeak.h"

const char* ssid = "yourSSID";
const char* password = "yourPASSWORD";
WiFiClient client;

unsigned long channelNumber = yourChannelNumber;
const char* writeAPIKey = "yourAPIKey";

void setup() {
  Serial.begin(115200);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
  ThingSpeak.begin(client);
}

void loop() {
  float current = readCurrent(); // Function to read sensor data
  ThingSpeak.setField(1, current);
  int response = ThingSpeak.writeFields(channelNumber, writeAPIKey);
  if (response == 200) {
    Serial.println("Data sent successfully.");
  } else {
    Serial.println("Failed to send data.");
  }
```

```
  delay(20000); // Send data every 20 seconds
}
```

4. Monitoring the Data

Log in to your cloud platform to visualize the real-time data sent from the ESP32.

Hands-On 2: Edge Computing with ESP32 for Real-Time Processing

Objective

Implement edge computing to process energy data locally on the ESP32 before sending critical alerts to the cloud.

Materials

- ESP32 microcontroller
- Energy sensor (e.g., ACS712, PZEM-004T)
- Breadboard and jumper wires
- LED or buzzer for alerts
- Internet connection

Steps

1. Set Up the Sensor and ESP32
Connect the sensor to ESP32 similarly to Hands-On 1.

2. Writing Edge Computing Code
The following code processes the energy data locally and triggers an alert when the current exceeds a threshold:

Listing 3: Edge Computing Code for ESP32

```
const int sensorPin = A0;
int sensorValue = 0;
float currentThreshold = 10.0; // Set threshold current in amps
const int ledPin = 13; // LED or buzzer for alert

void setup() {
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
}

void loop() {
  sensorValue = analogRead(sensorPin);
  float voltage = (sensorValue * 5.0) / 1023.0;
  float current = (voltage - 2.5) / 0.185; // Convert to current

  if (current > currentThreshold) {
    digitalWrite(ledPin, HIGH); // Trigger alert
    Serial.println("ALERT: Current exceeds threshold!");
```

```
  } else {
    digitalWrite(ledPin, LOW);
  }

  delay(1000);
}
```

Hands-On 3: Data Analysis and Visualization on Cloud

Objective

Analyze and visualize energy data acquired from an IoT system using cloud tools like ThingSpeak, Blynk, or Google Sheets.

Steps

1. Login to Cloud Platform
Access the data stored from the previous hands-on session.

2. Visualizing Real-Time Data
Use the platform's built-in tools to create charts and graphs for real-time energy data visualization.

3. Performing Data Analysis
Export the data to Google Sheets or Excel for further analysis.

4. Advanced Analysis with Python
Use Python to analyze the data. Here's an example using `pandas` and `matplotlib`:

Listing 4: Python Code for Data Analysis

```
import pandas as pd
import matplotlib.pyplot as plt

# Load the data from CSV
data = pd.read_csv('energy_data.csv')

# Plot current over time
plt.figure(figsize=(10, 5))
plt.plot(data['Time'], data['Current (A)'])
plt.xlabel('Time')
plt.ylabel('Current (A)')
plt.title('Energy Consumption Over Time')
plt.legend()
plt.show()
```

Hands-On 4: Implementing MQTT for Efficient Data Transmission

Objective

Use MQTT protocol to transmit energy data between IoT devices and a server efficiently.

Materials

- ESP32 microcontroller
- Energy sensor
- MQTT broker (e.g., HiveMQ, Mosquitto)
- Internet connection

Steps

1. Setting Up the MQTT Broker
Set up an MQTT broker (e.g., HiveMQ or Mosquitto). You can use a public broker or install a local instance.

2. ESP32 MQTT Client Setup
Install the PubSubClient library for MQTT communication. The following code sends sensor data to the MQTT broker:

Listing 5: ESP32 Code for MQTT Communication

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

const char* ssid = "yourSSID";
const char* password = "yourPASSWORD";
const char* mqttServer = "broker.hivemq.com";
const int mqttPort = 1883;

WiFiClient espClient;
PubSubClient client(espClient);

void setup() {
  Serial.begin(115200);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
  client.setServer(mqttServer, mqttPort);
```

```
}

void loop() {
  if (!client.connected()) {
    reconnect();
  }
  float current = readCurrent(); // Function to read sensor data
  String payload = String(current);
  client.publish("energy/data", payload.c_str());
  delay(1000); // Publish every second
}
```

3. Testing MQTT Communication

Use an MQTT client (e.g., MQTT.fx or HiveMQ) to subscribe to the topic "energy/data" and visualize the data.