



NETAJI SUBHAS UNIVERSITY OF TECHNOLOGY, DELHI

LAB File

Internet of Things

EAEPC18

ECAM-1

Submitted to:

Prof. Gangadirappa

Submitted by:

Hardik Chhabra

2020UEA6504

INDEX

S.No.	Title	Date	Sign
1			
2			
3			
4			
5			
6			
7			

EXPERIMENT-1

Aim: Study the interfacing of different sensors with NodeMCU, Raspberry Pi, and Arduino

Theory:

Microcontrollers like NodeMCU, Raspberry Pi, and Arduino are widely used in electronics and IoT (Internet of Things) to interface various sensors. Each platform has its unique characteristics, capabilities, and use cases when working with sensors.

NodeMCU:

NodeMCU is based on the ESP8266 Wi-Fi module and is primarily used for IoT applications. It offers built-in Wi-Fi connectivity, making it a popular choice for wireless sensor applications.

Advantages of NodeMCU:

- **Wireless Connectivity:** NodeMCU's built-in Wi-Fi capability allows it to connect to the internet and cloud services, making it suitable for remote monitoring and control.
- **Low Power Consumption:** NodeMCU is designed for low-power operation, making it energy-efficient for battery-powered applications.
- **Extensive Community and Libraries:** NodeMCU benefits from a large and active community, resulting in many resources and libraries for sensor interfacing.

Raspberry Pi:

Raspberry Pi is a single-board computer with a full Linux environment, making it a versatile platform for sensor interfacing and data processing.

Advantages of Raspberry Pi:

- **Computing Power:** Raspberry Pi has significantly more processing power than microcontrollers, allowing for complex data analysis and running multiple applications simultaneously.
- **Operating System:** Raspberry Pi runs a Linux-based OS, enabling advanced programming and multitasking capabilities.
- **GPIO Pins:** Raspberry Pi has GPIO (General Purpose Input/Output) pins that can be used for sensor interfacing, making it flexible for various sensor types.

Arduino:

Arduino is a popular microcontroller platform known for its simplicity and ease of use. It comes in various models, such as Arduino Uno, Nano, and Mega, suitable for different applications.

Advantages of Arduino:

- **Real-time Control:** Arduino is excellent for real-time applications and control systems due to its fast execution and precise timing.
- **Dedicated IDE:** The Arduino IDE provides a user-friendly environment for programming and uploading code to the microcontroller.
- **Vast Sensor Support:** Arduino has a wide range of sensors and shields available, simplifying sensor integration.

Sensor Interfacing:

When interfacing sensors with these platforms, there are common steps involved:

1. **Physical Connection:** Connect the sensor's pins to the appropriate pins or ports on the microcontroller. This often requires understanding the voltage levels and communication protocols (e.g., I2C, SPI) the sensor uses.
2. **Programming:** Write code that reads data from the sensor. This code varies depending on the platform and the programming language (e.g., Arduino uses C/C++, Raspberry Pi uses Python).
3. **Data Processing:** Process the sensor data as needed. This may involve data calibration, filtering, or conversion to meaningful units.
4. **Communication:** Transmit the sensor data to a display, storage, or communication medium, such as a screen, cloud service, or database.

Comparison:

- NodeMCU is ideal for IoT applications and remote sensing due to its built-in Wi-Fi. It's suitable for applications where wireless connectivity is crucial.
- Raspberry Pi excels in applications that require data processing, data analysis, or running multiple software applications alongside sensor interfacing.
- Arduino is well-suited for real-time control and simpler sensor applications, especially when precise timing is required.

Conclusion:

The choice of NodeMCU, Raspberry Pi, or Arduino for sensor interfacing depends on the specific requirements of the project, such as power consumption, data processing capabilities, and the need for wireless connectivity. Each platform offers its own strengths and trade-offs, making them suitable for a wide range of sensor-based applications.

EXPERIMENT-2

Aim: Study and implement LED Blink for microcontroller

Apparatus:

MSP430 microcontroller (e.g., MSP430G2553), USB-to-UART adapter, Breadboard, LED, 220-330Ω resistor, Jumper wires, Computer with MSP430 IDE installed

Theory:

The MSP430 microcontroller family, developed by Texas Instruments, is known for its low power consumption and versatility. It is widely used in various embedded systems and IoT applications. The specific model used in this experiment, the MSP430G2553, features GPIO (General Purpose Input/Output) pins that can be configured as digital outputs for controlling external devices like LEDs.

Code:

```
% Hardik Chhabra 2020UEA6504
#include <msp430.h>
int main(void) {
    WDTCTL = WDTPW | WDTHOLD;
    P1DIR = 00000001;
    P1OUT = 00000001;
    unsigned int i;
    while (1) {
        P1OUT = 00000000;
        for (i = 0; i<50000; i++);
        P1OUT = 00000001;
        for (i = 0; i<50000; i++);
    }
    return 0;
}
```

Output:

- The LED connected to P1.0 should blink ON and OFF continuously.
- The blink rate can be adjusted by changing the value in the for loop. Smaller values make the LED blink faster, while larger values make it blink slower.

Conclusion:

In this experiment, we have successfully programmed the MSP430 microcontroller to control an LED, making it blink at a specific rate. This basic project forms the foundation for more complex MSP430-based projects involving various sensors and actuators.

EXPERIMENT - 3

AIM: Data logging on ThingSpeak.

SOFTWARE USED: ThingSpeak website

THEORY:

ThingSpeak is an open source IoT platform with MATLAB analytics. It allows you to collect data from sensors, Arduino, Raspberry Pi, and other devices and store it in the cloud.

You can then analyze this data using MATLAB or other tools, and visualize it using charts and graphs.

ThingSpeak works by collecting data from IoT devices and storing it in channels. Each channel can have up to eight fields, which can be used to store different types of data. You can then access this data using ThingSpeak's API or web interface.

Benefits:

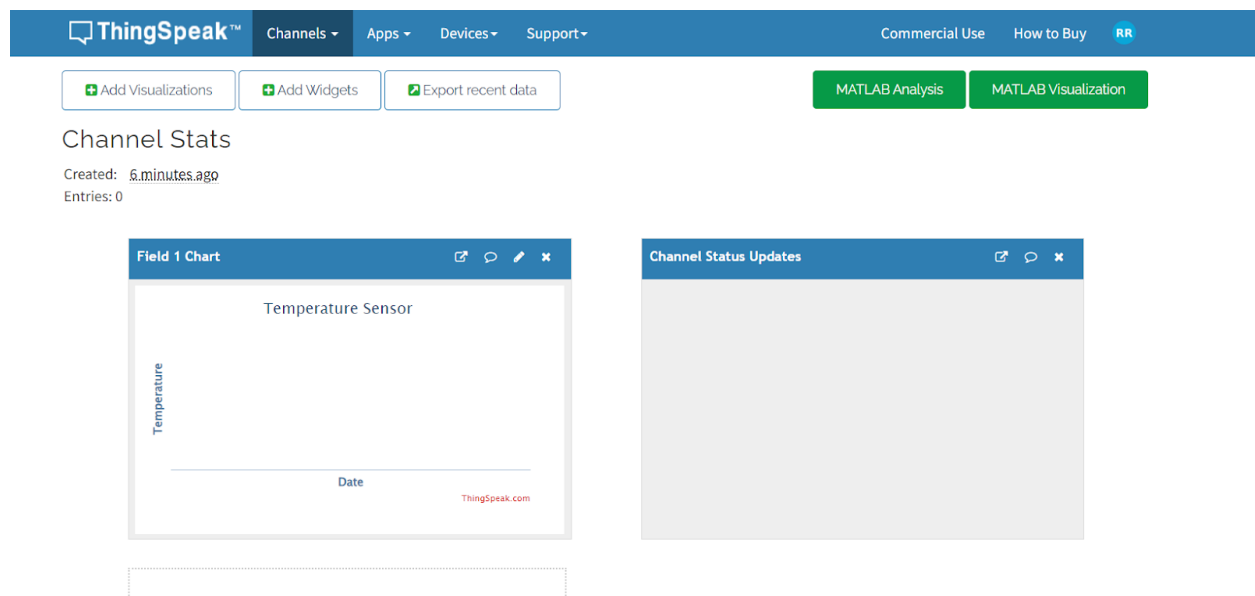
- **Simplicity:** ThingSpeak is easy to use and does not require any coding experience.
- **Scalability:** ThingSpeak can handle a large volume of data from a variety of devices.
- **Security:** ThingSpeak uses HTTPS to encrypt data transmissions.
- **Visualization:** ThingSpeak provides built-in tools for visualizing data.
- **MATLAB analytics:** ThingSpeak integrates with MATLAB, which provides powerful tools for data analysis and visualization.

PROCEDURE:

How to use ThingSpeak for data logging

To use ThingSpeak for data logging, you will need to:

1. Create a ThingSpeak account: You can create a ThingSpeak account for free at <https://thingspeak.com/>.
2. Create a channel: Once you have an account, you can create a new channel. Each channel can have up to eight fields.
3. Connect your device to ThingSpeak: You will need to connect your IoT device to ThingSpeak in order to send data to your channel. This may involve using a microcontroller board, such as an Arduino or Raspberry Pi, or using a cloud-based service, such as IFTTT.
4. Send data to ThingSpeak: Once your device is connected, you can start sending data to your ThingSpeak channel. This can be done using ThingSpeak's API or web interface.
5. Visualize and analyze data: Once you have data in your ThingSpeak channel, you can visualize it using ThingSpeak's built-in tools or export it to MATLAB for further analysis.



EXPERIMENT - 4

AIM: Temperature Sensor data on ThingSpeak Cloud.

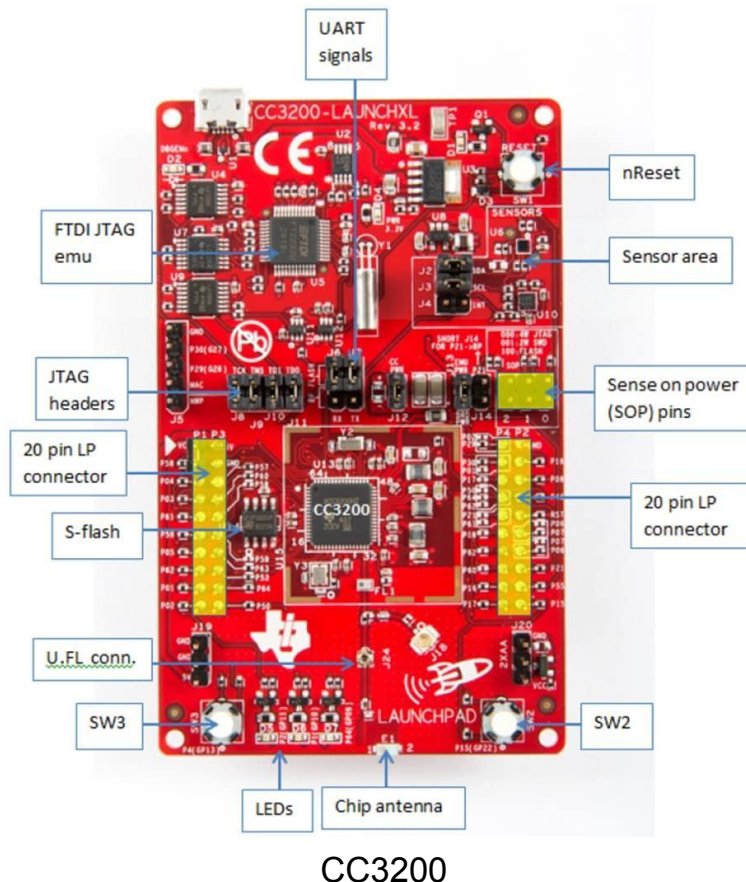
MATERIALS REQUIRED: TI CC3200, Energia IDE, ThingSpeak

THEORY:

The CC3200 is a low-power microcontroller with Wi-Fi and Bluetooth connectivity, making it ideal for IoT applications. It can be used to collect sensor data, such as temperature, humidity, and pressure, and send it to the cloud for analysis and visualization.

ThingSpeak is a popular IoT platform that allows you to collect, store, and visualize data from a variety of devices. It provides a simple way to send data from the CC3200 to the cloud and visualize it using charts and graphs.

TMP006 temperature sensor is used.



CODE:

```
// Include external libraries
#include <SPI.h>
#include <WiFi.h>
//Required for BMA222 Accelerometer sensor
#include <BMA222.h>
//Required for TMP006 Temperature sensor
#include <Wire.h>
#include "Adafruit_TMP006.h"

// Wireless network parameters
char ssid[] = "ABC";          // Your wireless network name also called SSID
char password[] = "Password"; // Your wireless network password

char thingSpeakAddress[] = "api.thingspeak.com";
char apiKey[] = "39DH3IUHR3";

const unsigned long postingInterval = 15L * 1000L; // delay between updates, in
milliseconds

#define POSTSTRING "field1=%s&field2=%5.2f&field3=%5.2f"

//Initialise dataString char Array
char dataString[]="field1=5c313e0555cf&field2=00.00&field3=00.00";
char deviceId[] = "xxxxxxxxxxxxx";

// Create MAC address and WiFiIPStack objects
MACAddress mac;

WiFiClient client;

// Define external sensors, inputs and outputs.
BMA222 accSensor;          // Three axis acceleration sensor
Adafruit_TMP006 tmp006(0x41);

// Setup function runs once when microprocessor is powered up
void setup() {
```

```

uint8_t macOctets[6];
Serial.begin(115200);      //Initialise serial port for local monitoring on the
Serial Monitor via USB
Serial.print("Attempting to connect to Network named: ");
Serial.println(ssid);

// Connect to WPA/WPA2 network. Change this line if using open or WEP
network:
WiFi.begin(ssid, password);
while ( WiFi.status() != WL_CONNECTED) {
  Serial.print(".");      // print dots while we wait to connect
  delay(300);
}

Serial.println("\nYou're connected to the network");
Serial.println("Waiting for an ip address");

while (WiFi.localIP() == INADDR_NONE) {
  Serial.print(".");      // print dots while we wait for an ip addresss
  delay(300);
}

// We are connected and have an IP address.
Serial.print("\nIP Address obtained: ");
Serial.println(WiFi.localIP());

mac = WiFi.macAddress(macOctets);
Serial.print("MAC Address: ");
Serial.println(mac);

// Use MAC Address as deviceId
sprintf(deviceId, "%02x%02x%02x%02x%02x%02x", macOctets[0],
macOctets[1], macOctets[2], macOctets[3], macOctets[4], macOctets[5]);
Serial.print("deviceId: ");
Serial.println(deviceId);

if (!tmp006.begin()) {

```

```

    Serial.println("No sensor found");
    while (1);
}

// start the accel sensor
accSensor.begin();
uint8_t chipID = accSensor.chipID();
Serial.print("ChipID: ");
Serial.println(chipID);
}

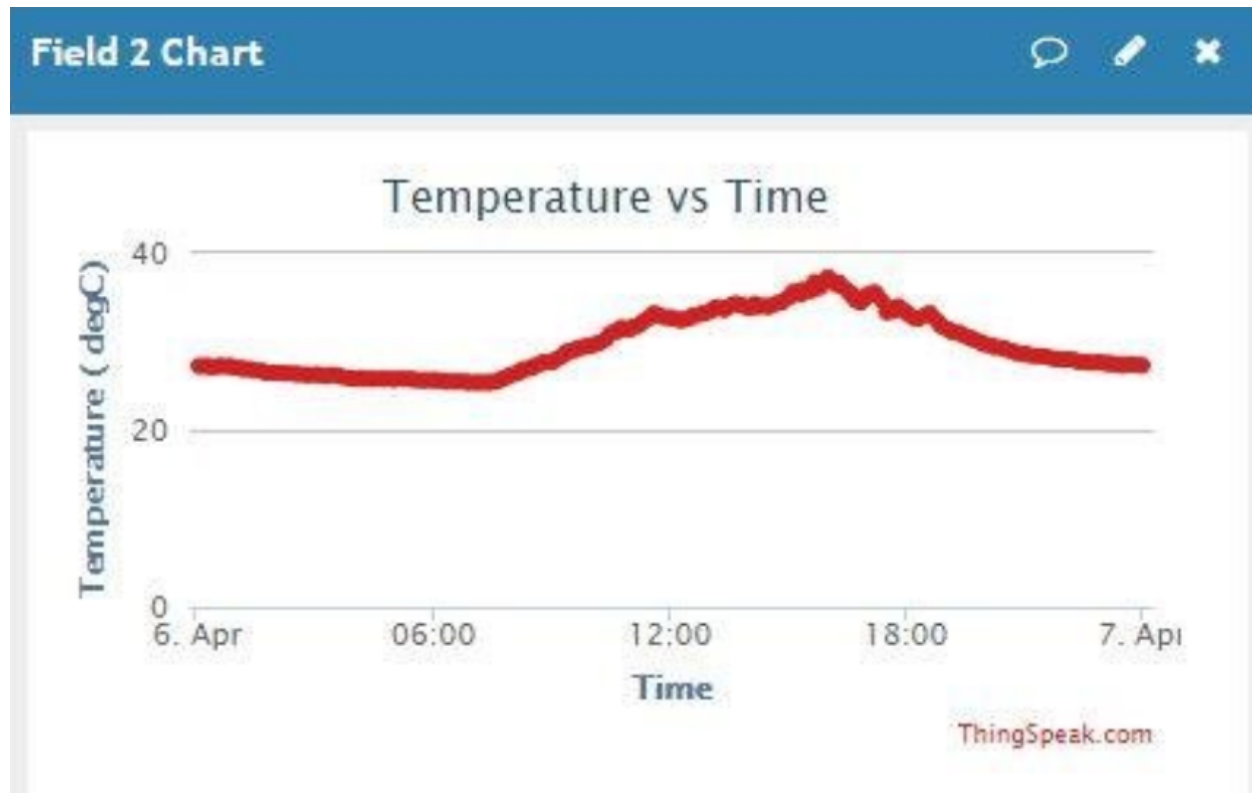
// Main loop. Runs continuously
void loop() {
    Serial.println("Preparing to post");
    if (client.connect(thingSpeakAddress,80)){
        Serial.println("Connected to server");

        sprintf(dataString,POSTSTRING,deviceId,tmp006.readObjTempC(),tmp006.read
        DieTempC());
        //Other sensor/data sources
        // accSensor.readXData(),
        accSensor.readYData(),accSensor.readZData(),-WiFi.RSSI()
        Serial.print("Posting: ");
        Serial.println(dataString);
        client.print("POST /update HTTP/1.1\n");
        client.print("Host: api.thingspeak.com\n");
        client.print("Connection: close\n");
        client.print("X-THINGSPEAKAPIKEY: ");
        client.print(apiKey);
        client.print("\n");
        client.print("Content-Type: application/x-www-form-urlencoded\n");
        client.print("Content-Length: ");
        client.print(sizeof(dataString)-1);
        client.print("\n\n");
        client.print(dataString);
        client.stop();
    }
}

```

```
else {  
  // if you couldn't make a connection:  
  Serial.println("connection failed");  
}  
delay(postingInterval);  
}
```

OUTPUT:



EXPERIMENT - 5

AIM: Accelerometer Sensor data on ThingSpeak Cloud

MATERIALS REQUIRED: TI CC3200, ThingSpeak

THEORY:

Accelerometers measure acceleration, which can be used to determine motion, tilt, and orientation. This data can be used for a variety of applications, such as fall detection, gesture recognition, and navigation.

ThingSpeak is a popular IoT platform that allows you to collect, store, and visualize data from your devices. It provides a simple and user-friendly interface for managing your data and creating charts and graphs.

Bosch-BMA222 accelerometer sensor is used.

CODE:

```
// Include external libraries
#include <SPI.h>
#include <WiFi.h>
//Required for BMA222 Accelerometer sensor
#include <BMA222.h>
//Required for TMP006 Temperature sensor
#include <Wire.h>
#include "Adafruit_TMP006.h"

// Wireless network parameters
char ssid[] = "ABC";          // Your wireless network name also called SSID
char password[] = "Password"; // Your wireless network password

char thingSpeakAddress[] = "api.thingspeak.com";
char apiKey[] = "235RF3E33T";

const unsigned long postingInterval = 15L * 1000L; // delay between updates, in
milliseconds
```

```

#define POSTSTRING "field1=%s&field2=%5.2f&field3=%5.2f"

//Initialise dataString char Array
char dataString[]="field1=5c313e0555cf&field2=00.00&field3=00.00";
char deviceId[] = "xxxxxxxxxxxx";

// Create MAC address and WiFiIPStack objects
MACAddress mac;

WiFiClient client;

// Define external sensors, inputs and outputs.
BMA222 accSensor;          // Three axis acceleration sensor
Adafruit_TMP006 tmp006(0x41);

// Setup function runs once when microprocessor is powered up
void setup() {
  uint8_t macOctets[6];
  Serial.begin(115200);      //Initialise serial port for local monitoring on the
  Serial Monitor via USB
  Serial.print("Attempting to connect to Network named: ");
  Serial.println(ssid);

  // Connect to WPA/WPA2 network. Change this line if using open or WEP
  network:
  WiFi.begin(ssid, password);
  while ( WiFi.status() != WL_CONNECTED) {
    Serial.print(".");      // print dots while we wait to connect
    delay(300);
  }

  Serial.println("\nYou're connected to the network");
  Serial.println("Waiting for an ip address");

  while (WiFi.localIP() == INADDR_NONE) {
    Serial.print(".");      // print dots while we wait for an ip addresss
    delay(300);
  }
}

```

```

}

// We are connected and have an IP address.
Serial.print("\nIP Address obtained: ");
Serial.println(WiFi.localIP());

mac = WiFi.macAddress(macOctets);
Serial.print("MAC Address: ");
Serial.println(mac);

// Use MAC Address as deviceId
sprintf(deviceId, "%02x%02x%02x%02x%02x%02x", macOctets[0],
macOctets[1], macOctets[2], macOctets[3], macOctets[4], macOctets[5]);
Serial.print("deviceId: ");
Serial.println(deviceId);

if (!tmp006.begin()) {
  Serial.println("No sensor found");
  while (1);
}

// start the accel sensor
accSensor.begin();
uint8_t chipID = accSensor.chipID();
Serial.print("ChipID: ");
Serial.println(chipID);
}

// Main loop. Runs continuously
void loop() {
  Serial.println("Preparing to post");
  if (client.connect(thingSpeakAddress,80)){
    Serial.println("Connected to server");

    sprintf(dataString,POSTSTRING,deviceId,tmp006.readObjTempC(),tmp006.read
DieTempC());
    //Other sensor/data sources

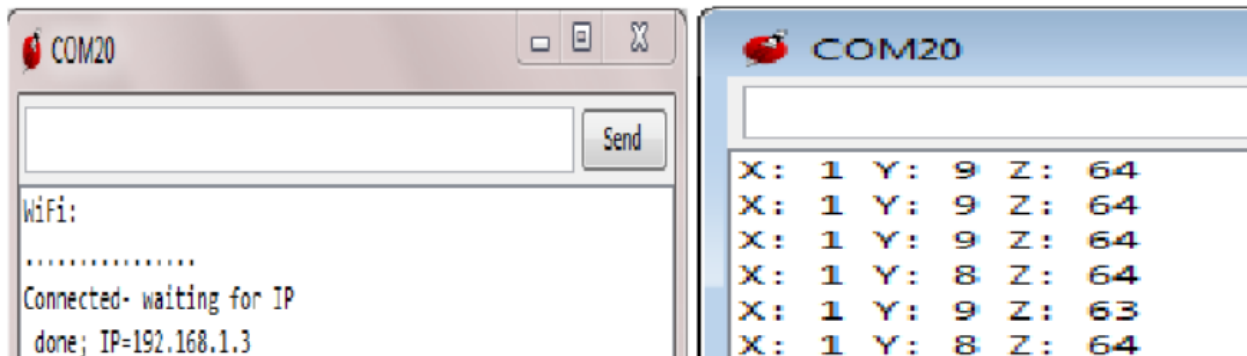
```

```

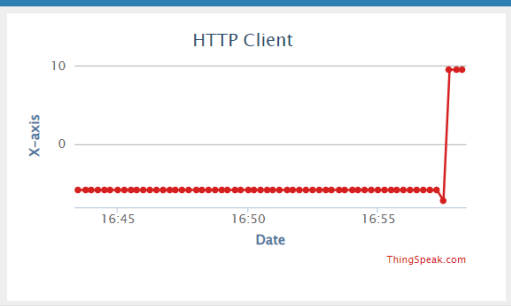
    accSensor.readXData(),
accSensor.readYData(),accSensor.readZData(),-WiFi.RSSI()
    Serial.print("Posting: ");
    Serial.println(dataString);
    client.print("POST /update HTTP/1.1\n");
    client.print("Host: api.thingspeak.com\n");
    client.print("Connection: close\n");
    client.print("X-THINGSPEAKAPIKEY: ");
    client.print(apiKey);
    client.print("\n");
    client.print("Content-Type: application/x-www-form-urlencoded\n");
    client.print("Content-Length: ");
    client.print(sizeof(dataString)-1);
    client.print("\n\n");
    client.print(dataString);
    client.stop();
}
else {
    // if you couldn't make a connection:
    Serial.println("connection failed");
}
delay(postingInterval);
}

```

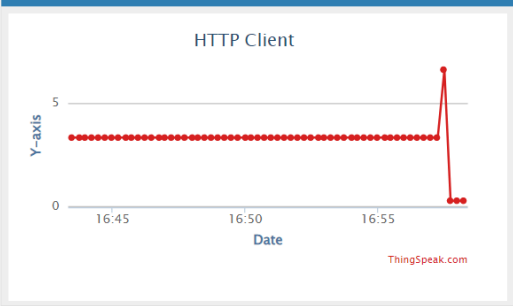
OUTPUT:



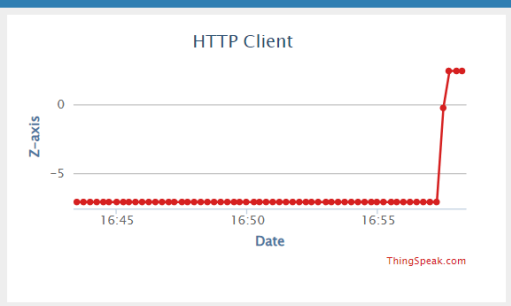
Field 1 Chart



Field 2 Chart



Field 3 Chart



EXPERIMENT-6

Aim: Develop an IoT system to monitor CO gas. Upload data to ThinkSpeak Cloud for efficient storage. Implement Machine Learning to predict RH based on CO levels.

Apparatus:

MSP430 microcontroller (e.g., MSP430G2553), USB-to-UART adapter, Breadboard, Gas Sensors, Jumper wires, Computer with MSP430 IDE installed

Theory:

The experiment integrated an MSP430 microcontroller in an IoT setup to monitor real-time CO gas levels. Collected data was transmitted to ThinkSpeak Cloud for cloud storage. The primary focus was utilizing machine learning, specifically Decision Tree Regression, to regress RH based on CO concentrations.

- **IoT Integration:**

The MSP430 microcontroller facilitated continuous CO gas data collection, offering real-time insights. ThinkSpeak Cloud served as the cloud platform, ensuring seamless data transmission and centralized storage.

- **Machine Learning Model:**

Implemented a Decision Tree Regression model to predict RH, establishing a correlation between CO levels and RH, essential for environmental monitoring.

Code:

```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

# Load the AirQuality dataset (replace 'your_dataset.csv' with the actual file
name)
# Assuming the dataset has a header row
data = pd.read_csv('your_dataset.csv')

# Select features and target variable
features = ['CO(GT)']
target = 'RH'

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data[features],
data[target], test_size=0.2, random_state=42)

# Decision Tree Regression
decision_tree_model = DecisionTreeRegressor()
```

```

decision_tree_model.fit(X_train, y_train)
decision_tree_predictions = decision_tree_model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, decision_tree_predictions)
print("Decision Tree Regression Mean Squared Error:", mse)

# Plot the results
plt.scatter(X_test, y_test, color='black', label='Actual data')
plt.scatter(X_test, decision_tree_predictions, color='red', label='Decision
Tree Predictions')
plt.xlabel('CO(GT)')
plt.ylabel('RH')
plt.legend()
plt.show()

```

Output:

The screenshot shows a Jupyter Notebook window with the following content:

In [37]:

```

1 df.drop('Date',axis=1,inplace=True)
2 df.drop('Time',axis=1,inplace=True)
3 df.drop('NMHC(GT)',axis=1,inplace=True)
4 df.head()

```

Out[37]:

	CO(GT)	PT08.S1(CO)	C6H6(GT)	PT08.S2(NMHC)	NOx(GT)	PT08.S3(NOx)	NO2(GT)	PT08.S4(NO2)	PT08.S5(O3)	T	RH	A
0	2.6	1360.00	11.881723	1045.50	166.0	1056.25	113.0	1692.00	1267.50	13.60	48.875001	0.7577
1	2.0	1292.25	9.397165	954.75	103.0	1173.75	92.0	1558.75	972.25	13.30	47.700000	0.72548
2	2.2	1402.00	8.997817	939.25	131.0	1140.00	114.0	1554.50	1074.00	11.90	53.975000	0.75023
3	2.2	1375.50	9.228796	948.25	172.0	1092.00	122.0	1583.75	1203.25	11.00	60.000000	0.78671
4	1.6	1272.25	6.518224	835.50	131.0	1205.00	116.0	1490.00	1110.00	11.15	59.575001	0.78879

In [13]:

```

1 #df.replace(to_replace=-200, value=np.NaN, inplace=True)

```

In [38]:

```

1 from sklearn.model_selection import train_test_split
2 x=df[["CO(GT)","PT08.S1(CO)","C6H6(GT)","PT08.S2(NMHC)","NOx(GT)","PT08.S3(NOx)","NO2(GT)","PT08.S4(NO2)","PT08.S5(O3)"]]
3 y=df[["RH","A"]]
4 x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)

```

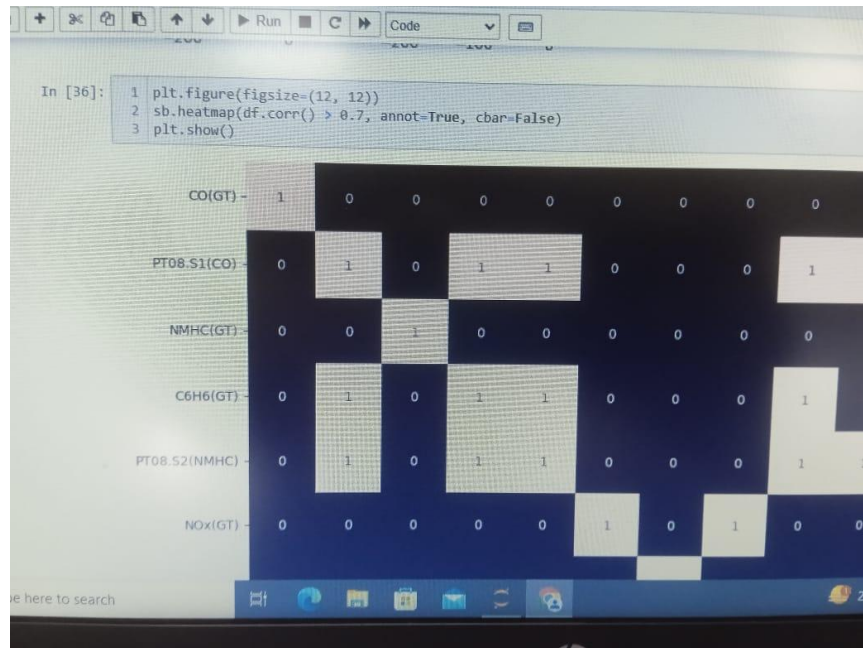
In [40]:

```

1 from sklearn.linear_model import LinearRegression
2 reg=LinearRegression()
3 reg.fit(x_train,y_train)

```

The bottom of the image shows a Windows taskbar with a search bar, several application icons, and a system tray displaying "23°C Smoke".



In [33]:

```
1 df.describe().T
```

Out[33]:

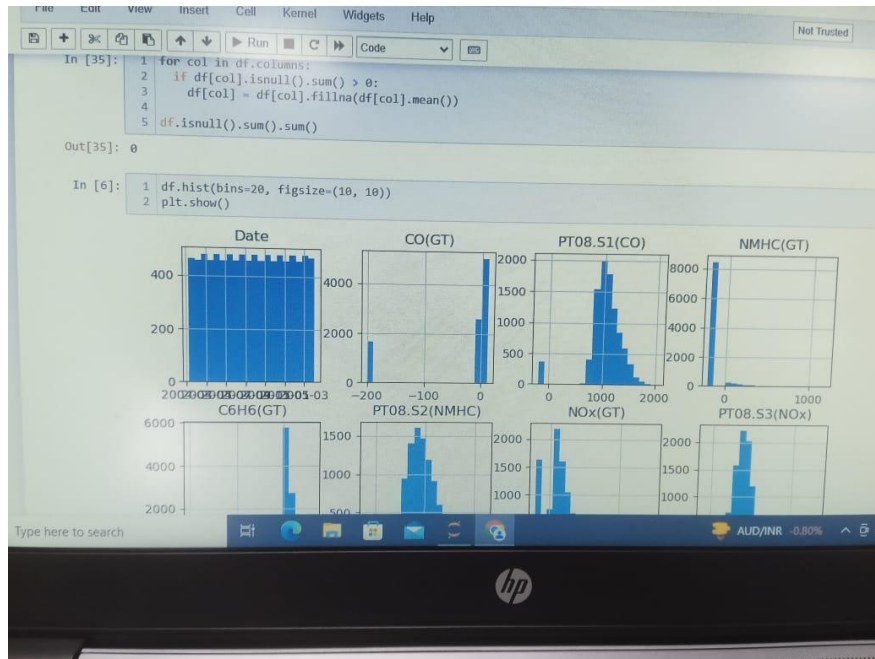
	count	mean	std	min	25%	50%	75%	max
CO(GT)	9357.0	-34.207524	77.657170	-200.0	0.600000	1.500000	2.600000	11.900000
PT08.S1(CO)	9357.0	1048.869652	329.817015	-200.0	921.000000	1052.500000	1221.250000	2039.750000
NMHC(GT)	9357.0	-159.090093	139.789093	-200.0	-200.000000	-200.000000	-200.000000	1189.000000
C6H6(GT)	9357.0	1.865576	41.380154	-200.0	4.004958	7.886653	13.636091	63.741476
PT08.S2(NMHC)	9357.0	894.475963	342.315902	-200.0	711.000000	894.500000	1104.750000	2214.000000
NOx(GT)	9357.0	168.604200	257.424561	-200.0	50.000000	141.000000	284.200000	1479.000000
PT08.S3(NOx)	9357.0	794.872333	321.977031	-200.0	637.000000	794.250000	960.250000	2682.750000
NO2(GT)	9357.0	58.135898	126.931428	-200.0	53.000000	96.000000	133.000000	339.700000
PT08.S4(NO2)	9357.0	1391.363266	467.192382	-200.0	1184.750000	1445.500000	1662.000000	2775.000000
PT08.S5(O3)	9357.0	974.951534	456.922728	-200.0	699.750000	942.000000	1255.250000	2522.750000
T	9357.0	9.776600	43.203438	-200.0	10.950000	17.200000	24.075000	44.600000
RH	9357.0	39.483611	51.215645	-200.0	34.050000	48.550000	61.875000	88.725000
AH	9357.0	-6.837604	38.976670	-200.0	0.692275	0.976823	1.296223	2.231036

In [34]:

```
1 df.isnull().sum()
```

Out[34]:

Date	0
Time	0
CO(GT)	0
PT08.S1(CO)	0



Conclusion:

The experiment successfully demonstrated the synergy between IoT, cloud technology, and machine learning for environmental monitoring. The use of the MSP430 microcontroller, coupled with ThinkSpeak Cloud, provided a robust infrastructure for real-time data collection and cloud-based storage. The Decision Tree Regression model added value by predicting RH based on CO gas concentrations, paving the way for future advancements in environmental monitoring systems. The experiment's outcomes contribute to the development of scalable and intelligent solutions for addressing air quality challenges.

EXPERIMENT-7

Aim: Develop an IoT system to monitor gasses in the atmosphere. Upload data to ThinkSpeak Cloud for efficient storage. Implement Machine Learning to predict AH and RH based on gas levels.

Apparatus:

MSP430 microcontroller (e.g., MSP430G2553), USB-to-UART adapter, Breadboard, Gas Sensors, Jumper wires, Computer with MSP430 IDE installed

Theory:

The experiment integrated an IoT setup with the MSP430 microcontroller for real-time environmental data collection, focusing on gases and key factors like temperature (T) and humidity (RH). The objective was to predict Absolute Humidity (AH) and Relative Humidity (RH) using Linear Regression and Decision Tree Regression.

- **Data and Preprocessing:**

The MSP430 continuously collected and preprocessed data, addressing missing values to ensure dataset integrity.

- **Feature Selection:**

Selected CO, C6H6, NOx, T, and RH as relevant features based on their impact on air quality.

- **Machine Learning Models:**

Implemented Linear Regression and Decision Tree Regression for predicting AH and RH.

- **Evaluation Metrics:**

Used mean squared error to assess model accuracy.

- **IoT and Cloud:**

Integrated MSP430 into an IoT setup, uploading data to ThinkSpeak Cloud for centralized storage.

Code:

```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error

# Load the AirQuality dataset (replace 'your_dataset.csv' with the actual file name)
# Assuming the dataset has a header row
data = pd.read_csv('dataset.csv')
```

```

# Define features and target variables
features = ['CO(GT)', 'PT08.S1(CO)', 'NMHC(GT)', 'C6H6(GT)', 'PT08.S2(NMHC)',
            'NOx(GT)', 'PT08.S3(NOx)', 'NO2(GT)', 'PT08.S4(NO2)',
            'PT08.S5(O3)', 'T']

targets = ['AH', 'RH']

# Split the dataset into training and testing sets
X_train, X_test, y_train_ah, y_test_ah, y_train_rh, y_test_rh =
train_test_split(
    data[features], data['AH'], data['RH'], test_size=0.2, random_state=42
)

# Standardize the features (if needed)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Linear Regression for AH
linear_model_ah = LinearRegression()
linear_model_ah.fit(X_train_scaled, y_train_ah)
linear_predictions_ah = linear_model_ah.predict(X_test_scaled)

# Decision Tree Regression for AH
decision_tree_model_ah = DecisionTreeRegressor()
decision_tree_model_ah.fit(X_train, y_train_ah)
decision_tree_predictions_ah = decision_tree_model_ah.predict(X_test)

# Linear Regression for RH
linear_model_rh = LinearRegression()
linear_model_rh.fit(X_train_scaled, y_train_rh)
linear_predictions_rh = linear_model_rh.predict(X_test_scaled)

# Decision Tree Regression for RH
decision_tree_model_rh = DecisionTreeRegressor()
decision_tree_model_rh.fit(X_train, y_train_rh)
decision_tree_predictions_rh = decision_tree_model_rh.predict(X_test)

# Evaluate models for AH
linear_mse_ah = mean_squared_error(y_test_ah, linear_predictions_ah)
decision_tree_mse_ah = mean_squared_error(y_test_ah,
decision_tree_predictions_ah)

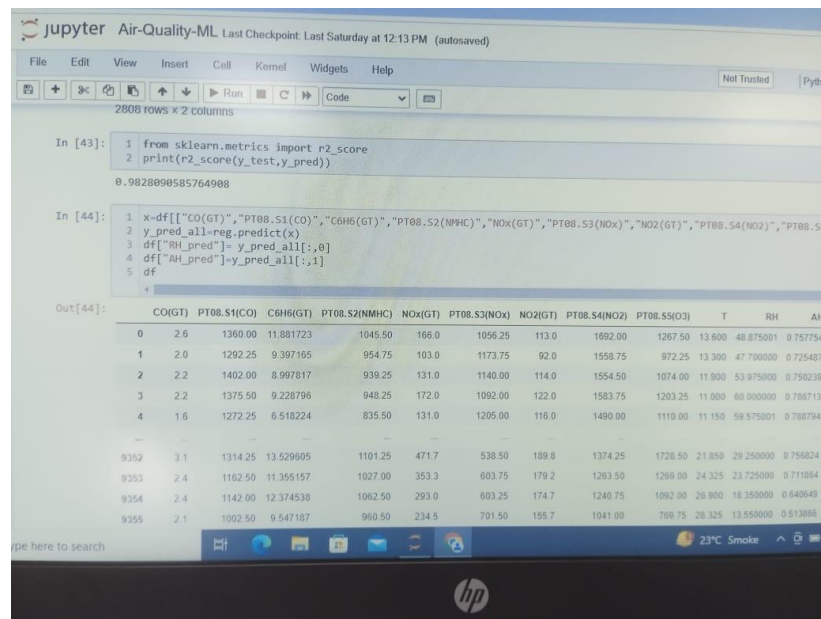
# Evaluate models for RH
linear_mse_rh = mean_squared_error(y_test_rh, linear_predictions_rh)
decision_tree_mse_rh = mean_squared_error(y_test_rh,
decision_tree_predictions_rh)

```

```
# Display results
print("Linear Regression Mean Squared Error for AH:", linear_mse_ah)
print("Decision Tree Regression Mean Squared Error for AH:",
decision_tree_mse_ah)

print("Linear Regression Mean Squared Error for RH:", linear_mse_rh)
print("Decision Tree Regression Mean Squared Error for RH:",
decision_tree_mse_rh)
```

Output:



The screenshot shows a Jupyter Notebook titled "Air-Quality-ML" with a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar. The notebook contains two code cells. The first cell (In [43]) imports `r2_score` from `sklearn.metrics` and prints the `r2_score` for the test set, resulting in `0.9828898585764988`. The second cell (In [44]) performs a regression analysis, creating a DataFrame `df` with columns for various air quality metrics and predicted values. The output of this cell is a DataFrame with 2808 rows and 2 columns, showing the first few rows and a summary of the data.

	CO(GT)	PT08.S1(CO)	CGH6(GT)	PT08.S2(NMHC)	NOx(GT)	PT08.S3(NOx)	NO2(GT)	PT08.S4(NO2)	PT08.S5(O3)	T	RH	AH
0	2.6	1360.00	11.881723	1045.50	166.0	1056.25	113.0	1892.00	1267.50	13.600	48.875001	0.757754
1	2.0	1292.25	9.397165	954.75	103.0	1173.75	92.0	1558.75	972.25	13.300	47.790000	0.725487
2	2.2	1402.00	8.997817	939.25	131.0	1140.00	114.0	1554.50	1074.00	11.900	53.975000	0.750230
3	2.2	1375.50	9.228796	948.25	172.0	1092.00	122.0	1583.75	1203.25	11.000	60.000000	0.788713
4	1.6	1272.25	6.518224	835.50	131.0	1205.00	116.0	1480.00	1110.00	11.150	58.575001	0.788794
...
9352	3.1	1314.25	13.529605	1101.25	471.7	538.50	189.8	1374.25	1728.50	21.850	29.250000	0.759824
9353	2.4	1162.50	11.355157	1027.00	353.3	603.75	179.2	1263.50	1269.00	24.325	23.725000	0.711864
9354	2.4	1142.00	12.374538	1062.50	293.0	603.25	174.7	1240.75	1062.00	26.900	18.350000	0.640649
9355	2.1	1002.50	9.547187	960.50	234.5	701.50	155.7	1041.00	788.75	28.325	13.550000	0.513866


```
File Edit View Insert Cell Kernel Widgets Help
+ - * < > Run Code

In [41]: 1 y_pred=reg.predict(x_test)
         2 y_pred

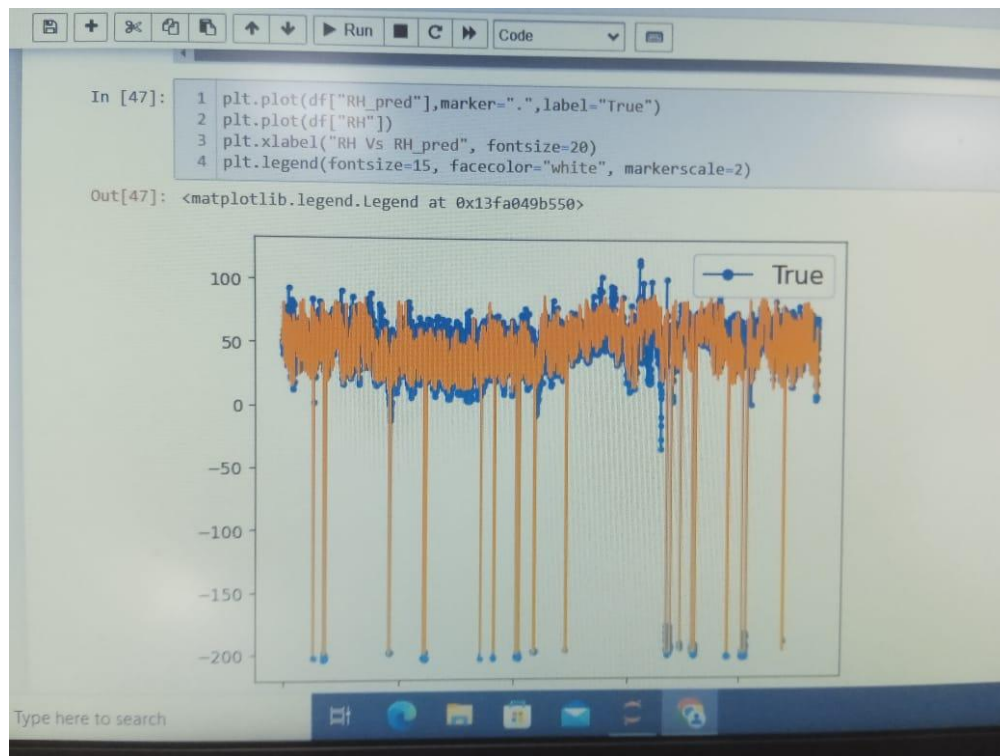
Out[41]: array([[62.39829141,  0.83965506],
                [24.50695682,  2.15261541],
                [59.78717496,  0.93507527],
                ...,
                [34.45557916, -0.20920832],
                [31.14606398,  0.59204125],
                [65.99010457,  0.84834172]])

In [42]: 1 y_test

Out[42]:
```

	RH	AH
6681	70.550000	0.760741
3954	18.225000	1.037593
5552	78.699999	1.316927
9247	68.600000	1.106785
3114	15.033333	0.972838
...
1770	28.150000	0.978503
6537	69.674999	0.995395
573	27.500000	0.857609
475	19.950000	0.535189
1084	70.674999	1.177251

Type here to search



Conclusion:

In conclusion, this experiment successfully demonstrated the integration of an MSP430 microcontroller in an IoT setup for monitoring various gases. The combination of machine learning, ThinkSpeak Cloud for cloud storage, and real-time data collection lays the foundation for a comprehensive environmental monitoring system. Further advancements and refinements can contribute to more accurate predictions and actionable insights for improving air quality.