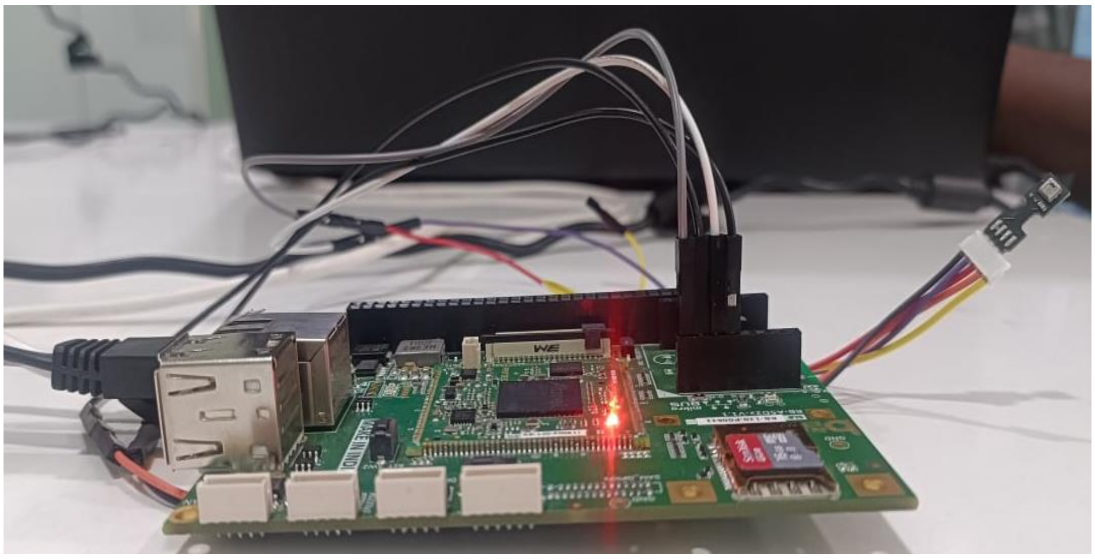
# AHT25 with W10 Module to Connect and Publish the Data in THINGSBOARD MQTT



**PRESENTED BY**

**HARISH KUMAR A M.E(VLSI Design)**
**MOHAN KUMAR D**
**VEERESH PATEL**

## TABLE OF CONTENT

## 1.Project Summary

AHT25 sensor is used to read the temperature and humidity values from the environment by using MCU like RUGGEDBOARD and send the values into mqtt broker like THINGSBOARD.

## 2.Key Features of the Project:

MQTT Configuration:

The paho library is used for MQTT communication.
The ACCESS_TOKEN variable holds the ThingsBoard device access token.
The MQTT broker's host (broker) and port (port) are specified.
I2C Configuration:

The script uses the mraa library to interface with I2C.
I2C_BUS and AHT25_ADDR define the I2C bus number and the address of the AHT25 sensor, respectively.
An I2C object (I2C) is created and configured to communicate with the AHT25 sensor.
AHT25 Sensor Commands:

Constants (AHT25_INIT_CMD and AHT25_MEASURE_CMD) represent initialization and measurement commands for the AHT25 sensor.
Sensor Reading Function:

The read_sensor_values function sends the measurement command to the AHT25 sensor, waits for the measurement to complete, and then reads and calculates the temperature and humidity values.
MQTT Callback:

The on_publish function is a callback function that gets executed when data is successfully published to ThingsBoard. It prints a message indicating that the data has been published.
Main Function:

The main function sets up the MQTT client, connects to the ThingsBoard MQTT broker, and initializes the AHT25 sensor.
It then enters a loop where it continuously reads sensor values, formats the data into a JSON payload, and publishes the payload to the specified topic on ThingsBoard.
There are sleep delays in the loop to control the rate of data publication.
Execution:

The script is designed to run indefinitely, continuously monitoring the AHT25 sensor and publishing the temperature and humidity data to ThingsBoard.
Usage:

Replace the placeholder access token ('4Iq0PdNohIDjA9cZSyhH') with the actual access token of your ThingsBoard device.
Adjust the MQTT broker's host (broker), port (port), and the delays in the loop as needed for your application.
This script essentially creates a simple environmental monitoring system, where the AHT25 sensor is used to measure temperature and humidity, and the data is sent to ThingsBoard for visualization and analysis

**3.Hardware used and there Specification**
Rugged Board - A5D2x is an Single Board Computer providing as easy migration path from Micro controller to Microprocessor. Rugged Board is enabled with industry Standard Yocto Build Embedded Linux platform and open source libraries for industrial application development. RuggedBoard is an Open source Industrial single board computer powered by ARM

Cortex-A5 SoC @500 MHz, implemented with the finest platform for rapid prototyping. The usage of System On Module over a System On Chip is the most rapid way to achieve time to market, curtail development risks for product quantities ranging from a few hundred to thousands.
RuggedBoard- A5D2x consists of Multiple Interfaces such as Ethernet, RS232, CAN, RS485, Digital Input and Digital Output with optically isolated, Standard MikroBus header for Add-On Sensors, Actuators and Multiple Wireless Modules such as ZigBee, LoRa, Bluetooth etc. mPCIe connector with USB interface used for Cloud Connectivity modules 3G, 4G, NB-IoT, WiFi. Expansion header with GPIO, UART, I2C, SPI, PWR etc.

**RuggedBoard - A5D2x Specification:**

System On Module

| | |
|---|---|
| SOC | Microchip ATSAMA5d2x Cortex-A5 |
| Frequency | 500MHz |
| RAM | 64 MB DDR3 |
| Flash | 32 MB NOR flash |
| SD Card | SD Card Upto 32 GB |

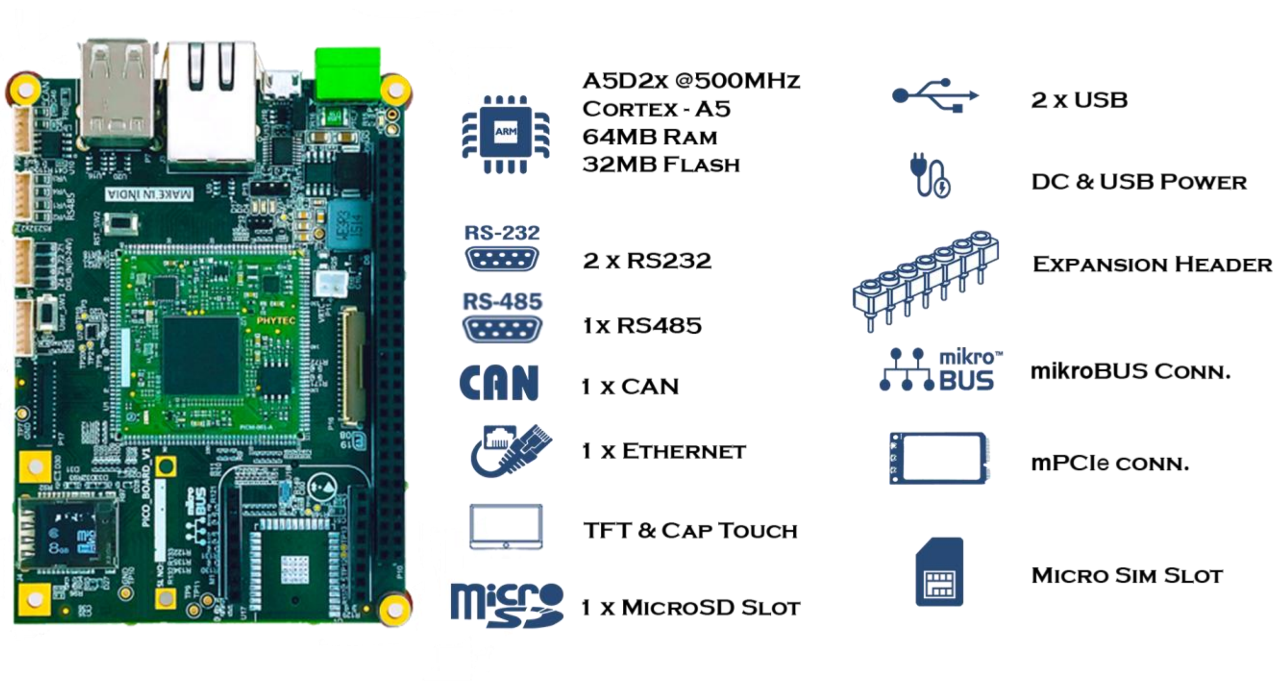
Industrial Interface

| | |
|---|---|
| RS232 | 2x RS232 |
| USB | 2 x USB*(1x Muxed with mPCIe) |
| Digital Input | 4x DIN (Isolated ~ 24V) |
| Digital Output | 4x DOUT (Isolated ~ 24V) |
| RS485 | 1xRs485 |
| CAN | 1xCAN |

Internet Access

| | |
|---|---|
| Ethernet | 1 x Ethernet 10/100 |
| Wi-Fi/BT | Optional on Board Wi-Fi/BT |
| SIM Card | 1 x SIM Slot (for mPCIe Based GSM Module) |

Add-On Module Interfaces
Mikro-BUS  Standard Mikro-BUS
mPCIe     1 x mPCIe* (Internally USB Signals is used)
Expansion Header  SPI, I2C, UART, PWM, GPIO,ADC
 Power
Input Power DC +5V or Micro USB Supply
Temperature Range     - 40°to + 85°C
Optional Accessories
Accessories Set Micro USB Cable, Ethernet Cable, Power Adapter 5V/3A



AHT25 SENSOR:

1.Product Overview

The AHT25 temperature and humidity sensor is equipped with a newly designed ASIC dedicated chip, an improved MEMS semiconductor capacitive humidity sensor element and a standard temperature sensor element, and its performance has reached the industry's advanced level. The improved new generation temperature and humidity sensor AHT25 has more stable performance in harsh environments and can maintain good accuracy in a larger measurement range. AHT25 uses standard-pitch plug-in connectors, which can be easily replaced in application. Every sensor has been rigorously calibrated and tested. Due to the improvement and miniaturization of the sensor, its cost performance has become higher.

## 2.Application Scenarios

The AHT25 can be widely used in consumer electronics, medical, automotive, industrial, meteorological and other fields, such as: HVAC, dehumidifiers and refrigerators and other home appliances, testing and testing equipment and other related temperature and humidity testing and control products.

## 3.Product Features

- •Fully calibrated
- •Digital output, I²C interface
- •Excellent long-term stability
- •Quick response and strong anti-interference ability

| | |
|---|---|
| Supply voltage | DC : 2.2 - 5.5V |
| Measuring range (humidity) | 0 ~ 100% RH |
| Measuring range (temperature) | -40 ~ + 80 ℃ |
| Humidity accuracy | ± 2 % RH ( 25 ℃ ) |
| Temperature accuracy | ± 0.3 ℃ |
| Resolution | temperature: 0.01℃ Humidity: 0.024%RH |
| Output signal | $I^2C$ signal |

## 4.Connection diagram

In Ruggedboard with AHT25:

| PIN | PIN NUMBER | COMPONENTS |
|---|---|---|
| VCC 5V | VCC | AHT25(+5V) |
| GROUND | GND | AHT25(GND) |
| PD21 | MRAA 32 | SDA(I2C-0) |
| PD22 | MRAA 31 | SCL(I2C -0) |

## 5.Project Code and explanation

STAGE 1:

Communicate AHT25 sensor with RUGGEDBOARD to receive the temperature and humidity values using mraa and sysfs code.

Code1:(MRAA)

```
#include "stdio.h"
#include "string.h"
#include "mraa/i2c.h"
#include "mraa/uart.h"
#include "unistd.h"

#define AHT25_ADDR 0x38 // Address of the AHT25 sensor
#define I2C_BUS 0

#define AHT25_INIT_CMD 0xE1
#define AHT25_MEASURE_CMD 0xAC

mraa_i2c_context i2c;
mraa_uart_context uart;

void read_sensor_values(float *temperature, float *humidity)
{
   uint8_t data[6];
   uint8_t cmd = AHT25_MEASURE_CMD;
   mraa_i2c_write_byte(i2c,AHT25_MEASURE_CMD);

   // Delay for measurement
   usleep(1000);  // Delay for 100 milliseconds

   mraa_i2c_read_bytes_data(i2c, 0x00,data, 6);

   *humidity = ((float)((data[1] << 12) | (data[2] << 4) |
(data[3] >> 4))) / 1048576.0 * 100.0;
   *temperature = ((float)(((data[3] & 0x0F) << 16) | (data[4] <<
8) | data[5])) / 1048576.0 * 200.0 - 50.0;
}

int main()
{
   // Initialize I2C
   i2c = mraa_i2c_init(0); // Use the default I2C bus
   mraa_i2c_frequency(i2c, MRAA_I2C_STD);
   mraa_i2c_address(i2c, AHT25_ADDR);
   mraa_i2c_write_byte(i2c, AHT25_INIT_CMD);


   // Initialize UART
   uart = mraa_uart_init(0); // Use the default UART
   mraa_uart_set_baudrate(uart, 115200);

mraa_uart_set_mode(uart,8,MRAA_UART_PARITY_NONE,1
);
   mraa_uart_set_flowcontrol(uart,0,0);

   while (1)
   {
    float temperature, humidity;
      read_sensor_values(&temperature, &humidity);



      printf( "Temperature=%.2f\r\n", temperature);
 //   mraa_uart_write(uart, temperature);
    usleep(500000);
      printf( "Humidity=%.2f\r\n", humidity);
//    mraa_uart_write(uart, humidit );
      usleep(500000);
   }

   // Cleanup
   mraa_i2c_stop(i2c);
   mraa_uart_stop(uart);


   return 0;
}
```

Code Explanation:

These are the include statements for various header files. They are necessary for using standard input/output functions (stdio.h), string manipulation functions (string.h), I2C and UART libraries from the MRAA (Intel IoT Platform) library (mraa/i2c.h and mraa/uart.h), and for providing the usleep function for microsecond delays (unistd.h).

These are preprocessor macros defining the I2C address of the AHT25 sensor (AHT25_ADDR), the I2C bus number (I2C_BUS), and two command codes (AHT25_INIT_CMD and AHT25_MEASURE_CMD) for initializing the sensor and requesting measurement data.

These are declarations of MRAA I2C and UART contexts, which will be used to interact with the I2C bus and UART interface.

This function, read_sensor_values, is responsible for reading the temperature and humidity values from the AHT25 sensor. It writes the measurement command to the sensor, waits for a delay (100 milliseconds), then reads 6 bytes of data from the sensor. The received bytes are then processed to calculate the temperature and humidity values, which are stored in the provided pointers.

In the main function, the I2C and UART interfaces are initialized. The program then enters an infinite loop where it repeatedly reads sensor values, prints them to the console, and sleeps for a short duration. Note that the UART write lines are commented out, so the sensor data is only printed to the console. The program continues this loop indefinitely until manually terminated. Finally, cleanup actions for the I2C and UART interfaces are performed before exiting the program.

Code2:

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>
#include <stdint.h>  // Add this line for uint8_t type
#include <sys/ioctl.h>  // Add this line for ioctl function
#include <linux/i2c-dev.h>  // Add this line for I2C_SLAVE
constant

#define AHT25_ADDR 0x38 // Address of the AHT25 sensor
#define I2C_BUS "/dev/i2c-0"

#define AHT25_INIT_CMD 0xE1
#define AHT25_MEASURE_CMD 0xAC

int i2c_fd;

void write_i2c_register(uint8_t reg, uint8_t value) {
   uint8_t buf[2];
   buf[0] = reg;
   buf[1] = value;
   write(i2c_fd, buf, 2);
}

void read_i2c_registers(uint8_t reg, uint8_t *data, int length) {
   write(i2c_fd, &reg, 1);
   read(i2c_fd, data, length);
}

void read_sensor_values(float *temperature, float *humidity) {
   uint8_t data[6];
   uint8_t cmd = AHT25_MEASURE_CMD;

   write_i2c_register(0x00, cmd);

   // Delay for measurement
```

```
    usleep(1000);  // Delay for 100 milliseconds

    read_i2c_registers(0x00, data, 6);

    *humidity = ((float)((data[1] << 12) | (data[2] << 4) |
(data[3] >> 4))) / 1048576.0 * 100.0;
    *temperature = ((float)(((data[3] & 0x0F) << 16) | (data[4] <<
8) | data[5])) / 1048576.0 * 200.0 - 50.0;
}

int main() {
    char buffer[128];
    float temperature, humidity;

    // Initialize I2C
    i2c_fd = open(I2C_BUS, O_RDWR);
    if (i2c_fd < 0) {
        perror("Error opening I2C bus");
        exit(1);
    }

    if (ioctl(i2c_fd, I2C_SLAVE, AHT25_ADDR) < 0) {
        perror("Error setting I2C address");
        close(i2c_fd);
        exit(1);
    }

    write_i2c_register(0x00, AHT25_INIT_CMD);

    while (1) {
        read_sensor_values(&temperature, &humidity);

        printf("Temperature=%.2f\r\n", temperature);
        usleep(1000000);
        printf("Humidity=%.2f\r\n", humidity);
        usleep(1000000);
    }
    close(i2c_fd);

    return 0;
}
```

Code Explanation:
These are the standard C libraries used for input/output, memory allocation, file control, string manipulation, and I2C communication.
These are constants used in the program. AHT25_ADDR is the I2C address of the AHT25 sensor, I2C_BUS is the I2C bus device file, and the other two constants are command codes used for sensor initialization and measurement.i2c_fd is a file descriptor used to identify the I2C bus.This function writes a value to a specified I2C register.This function reads data from consecutive I2C registers starting from the specified register.This function initiates a measurement on the AHT25 sensor, waits for a delay, and then reads the measured temperature and humidity values.The main function initializes the I2C bus, sets the I2C address of the AHT25 sensor, and performs the sensor initialization. It then enters an infinite loop, repeatedly reading and printing temperature and humidity values with a delay.Open the I2C bus and set the I2C address for communication.This writes the initialization command to the AHT25 sensor.The program enters an infinite loop where it reads and prints the temperature and humidity values with a delay of one second between each reading.Closes the I2C bus before exiting the program.This code is designed for Linux systems, and it uses the ioctl and open functions for I2C communication. The sensor readings are obtained by sending specific commands to the AHT25 sensor and interpreting the data received.

STAGE2:

Transfer the data to the THINGSBOARD by using c language code and python code.In python code ,we used paho-mqtt library to transfer the data to the mqtt server.

Code1:(python)

```
import paho.mqtt.client as paho
import time
import json
import mraa

# Replace 'oDfyF7380QGAv49715CV' with the actual access
token of your device
ACCESS_TOKEN = '4Iq0PdNohIDjA9cZSyhH'

broker = "phyclouds.com"  # host name
port = 1884  # data listening port

# I2C bus and device addresses
I2C_BUS = 1
AHT25_ADDR = 0x38

# I2C device object
I2C = mraa.I2c(I2C_BUS)
I2C.address(AHT25_ADDR)

# AHT25 commands
AHT25_INIT_CMD = 0xE1
AHT25_MEASURE_CMD = 0xAC

# Read the temperature and humidity values from the AHT25
sensor
def read_sensor_values():
    I2C.writeByte(AHT25_MEASURE_CMD)
    time.sleep(0.85)  # Some delay for measurement

    data = I2C.read(6)
    humidity = ((data[1] << 12) | (data[2] << 4) | (data[3] >> 4)) /
1048576.0 * 100.0
    temperature = ((data[3] & 0x0F) << 16 | (data[4] << 8) |
data[5]) / 1048576.0 * 200.0 - 50.0

    return temperature, humidity

def on_publish(client, userdata, result):
    print("Data published to ThingsBoard\n")

def main():
    client = paho.Client("Env_Monitoring")
    client.on_publish = on_publish
    client.username_pw_set(ACCESS_TOKEN)
    client.connect(broker, port, keepalive=60)

    # Initialize the AHT25 sensor
    I2C.writeByte(AHT25_INIT_CMD)
    time.sleep(0.1)  # Some delay for initialization

    while True:
        # Read sensor values
        temperature, humidity = read_sensor_values()

        payload = {"humidity": humidity, "temperature":
temperature}
        payload_str = json.dumps(payload)

        time.sleep(1)  # Adjust this delay as needed for your
application

        # Publish data to ThingsBoard
        result = client.publish("v1/devices/me/telemetry",
payload_str)
```

```
        time.sleep(1)  # Adjust this delay as needed for your
application

if __name__ == "__main__":
    main()
```

Code Explanation:

These lines import necessary modules: paho.mqtt.client for MQTT communication, time for handling time-related functions, json for working with JSON data, and mraa for interfacing with I2C.Specifies the MQTT broker's host name and port number.Specifies the MQTT broker's host name and port number.Initializes the I2C interface using the mraa library and sets the address for communication with the AHT25 sensor.Defines the initialization and measurement commands for the AHT25 sensor.Defines a function read_sensor_values to read temperature and humidity from the AHT25 sensor.Callback function to handle the publish event. It prints a message when data is successfully published to ThingsBoard.Main function that sets up the MQTT client, assigns the callback function for publish events, sets the access token, and connects to the MQTT broker.Initializes the AHT25 sensor with a small delay for stability.In a continuous loop, it reads sensor values, formats them into a JSON payload, and publishes the data to ThingsBoard using MQTT. It includes delays to control the data publishing frequency.Executes the main function when the script is run as the main program.

CODE 2:

```
#include <stdio.h>
#include <stdlib.h>
```

```
#include <string.h>
#include <fcntl.h>
#include <termios.h>
#include <unistd.h>
#include <math.h>
#include "MQTTClient.h"
#include "mraa/i2c.h"
#include "mraa/uart.h"
#include <signal.h>

// MQTT Settings
#define ACCESS_TOKEN "4Iq0PdNohIDjA9cZSyhH"
#define MQTT_ADDRESS "tcp://phyclouds.com:1884"
#define MQTT_CLIENTID "iiscSmartSwitch"
#define MQTT_TOPIC "v1/devices/me/telemetry"
#define MQTT_QOS 1

// Global MQTT client variable
MQTTClient client;
MQTTClient_connectOptions conn_opts =
MQTTClient_connectOptions_initializer;

// I2C and UART Initialization
mraa_i2c_context i2c;
mraa_uart_context uart;

// AHT25 Sensor Constants
#define AHT25_ADDR 0x38 // Address of the AHT25 sensor
#define AHT25_INIT_CMD 0xE1
#define AHT25_MEASURE_CMD 0xAC

// Function to publish data to MQTT
void publishToMQTT(char *topic, char *payload)
{
    MQTTClient_message pubmsg =
MQTTClient_message_initializer;
    pubmsg.payload = payload;
    pubmsg.payloadlen = strlen(payload);
    pubmsg.qos = 1;
    pubmsg.retained = 0;
    MQTTClient_deliveryToken token;

    MQTTClient_publishMessage(client, topic, &pubmsg,
&token);
    MQTTClient_waitForCompletion(client, token, 1000);

    int rc = MQTTClient_waitForCompletion(client, token,
10000);

    if (rc == MQTTCLIENT_SUCCESS)
    {
        // Print information about the published data
        printf("Published MQTT data - Topic: %s,
Payload: %s\r\n", topic, payload);
    }
    else
    {
        fprintf(stderr, "Failed to publish file data. Return
code: %d\n", rc);
    }
}

// Function to handle program termination signals
void handle_termination(int signum)
{
    printf("\nReceived termination signal. Cleaning up
resources...\n");

    // Cleanup resources
    MQTTClient_destroy(&client);
    mraa_i2c_stop(i2c);
    mraa_uart_stop(uart);

    exit(0);
```

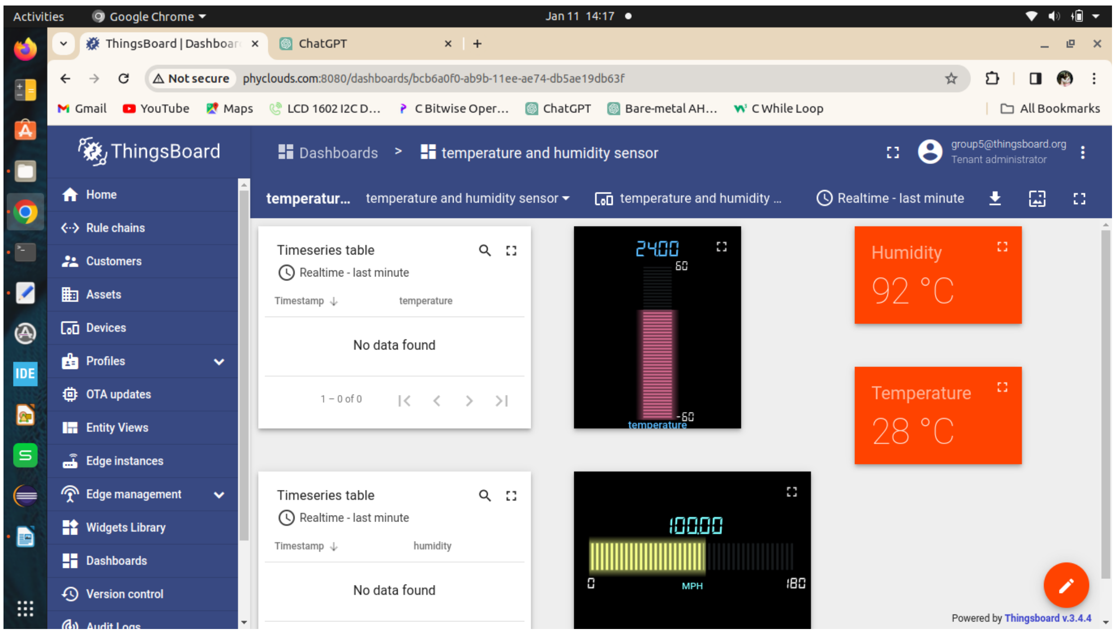
```
}

// Function to read sensor values
void read_sensor_values(float *temperature, float *humidity)
{
   uint8_t data[6];
   mraa_i2c_write_byte(i2c, AHT25_MEASURE_CMD);

   // Delay for measurement
   usleep(1000); // Delay for 100 milliseconds

   mraa_i2c_read_bytes_data(i2c, 0x00, data, 6);

   *humidity = ((float)((data[1] << 12) | (data[2] << 4) |
(data[3] >> 4))) / 1048576.0 * 100.0;
   *temperature = ((float)(((data[3] & 0x0F) << 16) | (data[4] <<
8) | data[5])) / 1048576.0 * 200.0 - 50.0;
}

int main()
{

  // Set up signal handler for program termination signals
   signal(SIGINT, handle_termination);
   // MQTT initialization
   MQTTClient_create(&client, MQTT_ADDRESS,
MQTT_CLIENTID, MQTTCLIENT_PERSISTENCE_NONE,
NULL);
   conn_opts.keepAliveInterval = 20;
   conn_opts.cleansession = 1;
   conn_opts.username = ACCESS_TOKEN;

   if (MQTTClient_connect(client, &conn_opts) !=
MQTTCLIENT_SUCCESS)
   {
      fprintf(stderr, "Failed to connect to MQTT server\n");
      return -1;
```

```
    }

    // Initialize I2C
    i2c = mraa_i2c_init(0); // Use the default I2C bus#include
<unistd.h>
    mraa_i2c_frequency(i2c, MRAA_I2C_STD);
    mraa_i2c_address(i2c, AHT25_ADDR);
    mraa_i2c_write_byte(i2c, AHT25_INIT_CMD);

    // Initialize UART
    uart = mraa_uart_init(0); // Use the default UART
    mraa_uart_set_baudrate(uart, 115200);
    mraa_uart_set_mode(uart, 8,
MRAA_UART_PARITY_NONE, 1);
    mraa_uart_set_flowcontrol(uart, 0, 0);

    while (1)
    {
        float temperature, humidity;
        read_sensor_values(&temperature, &humidity);

        // Convert sensor data to JSON string
        char mqtt_payload[500];
        snprintf(mqtt_payload, sizeof(mqtt_payload),
"{\"Humidity\": %.2f, \"Temperature\": %.2f}", humidity,
temperature);

        // Publish data to MQTT
        publishToMQTT(MQTT_TOPIC, mqtt_payload);

         // Print information about the published data
        printf("Data published to MQTT - Topic: %s,
Payload: %s\r\n", MQTT_TOPIC, mqtt_payload);

        // Sleep for 7 seconds
        sleep(7);
    }
    // Cleanup resources
    MQTTClient_destroy(&client);
    mraa_i2c_stop(i2c);
    mraa_uart_stop(uart);

    return 0;
}
```

Code Explanation:

These lines import necessary modules: mqtt.client for MQTT communication, time for handling time-related functions, json for working with JSON data, and mraa for interfacing with I2C.Specifies the MQTT broker's host name and port number.Specifies the MQTT broker's host name and port number.Initializes the I2C interface using the mraa library and sets the address for communication with the AHT25 sensor.Defines the initialization and measurement commands for the AHT25 sensor.Defines a function read_sensor_values to read temperature and humidity from the AHT25 sensor.Callback function to handle the publish event. It prints a message when data is successfully published to ThingsBoard.Main function that sets up the MQTT client, assigns the callback function for publish events, sets the access token, and connects to the MQTT broker.Initializes the AHT25 sensor with a small delay for stability.In a continuous loop, it reads sensor values, formats them into a JSON payload, and publishes the data to ThingsBoard using MQTT. It includes delays to control the data publishing frequency.Executes the main function when the script is run as the main program.

## 6.Output


ThingsBoard
Dashboards > temperature and humidity sensor
Home
Rule chains
Customers
Assets
Devices
Profiles
OTA updates
Entity Views
Edge instances
Edge management
Widgets Library
Dashboards
Version control
Timeseries table
Realtime - last minute
Timestamp
temperature
No data found
1 – 0 of 0
24.00
60
-60
temperature
Humidity
92 °C
Temperature
28 °C
Timeseries table
Realtime - last minute
Timestamp
humidity
No data found
100.00
0
MPH
180
Powered by Thingsboard v.3.4.4