
IoT-based Temperature and Humidity Monitoring System

PRESENTED BY

SHRUTHI K

Table of Contents

1. PROJECT SUMMARY

2. KEY FEATURES OF PROJECT

3. HARDWARE USED AND THEIR SPECIFICATIONS

4.AHT 25 SENSOR

4.1 SLAVE ADDRESS AND CONFIRATIONS

4.2 SIGNAL CONVERSION

5.W10 WIFI

6.RUGGED BOARD

7.WHY INTERRUPT BASED UART

8.PROJECT STAGES

8.1 STAGE-1

8.2 STAGE-2

8.3 STAGE-3

8.4 STAGE-4

9.FURTHER SCOPE

10.REAL TIME APPLICATION

11. STAGE-2 AND STAGE-4 CODE

12. CONCLUSION

1. Project Summary

The Internet of Things (IoT) is a rapidly growing field that has the potential to revolutionize the way we interact with the world around us. One of the most promising applications of IoT is in the area of environmental monitoring. By deploying IoT devices in various environments, we can collect real-time data on environmental conditions such as temperature, and humidity. This data can then be used to track changes in environmental conditions over time, identify potential problems, and take corrective action.

This project proposes the development of an IoT-based temperature and humidity monitoring system using STM32, AHT25, and W10 WiFi modules. The system will use an STM32 microcontroller to acquire sensor data from an AHT25 temperature and humidity sensor. The sensor data will then be published to MQTT using the W10 WiFi module, where it can be accessed by a web application or other IoT devices. The web application will allow users to visualize the sensor data and track changes in environmental conditions over time.

The system will be deployed in a real-world environment to test its performance. The deployment results will be used to assess the system's feasibility and identify any potential improvements.

The proposed system has the potential to be a valuable tool for tracking and monitoring environmental conditions. The system is relatively low-cost and easy to deploy, making it a feasible option for a wide range of applications. The system is also scalable, making it possible to deploy it in a variety of environments.

The results of this project will contribute to the body of knowledge on IoT-based environmental monitoring systems. The project will also provide a valuable case study for other researchers and developers who are interested in developing IoT-based ecological monitoring systems.

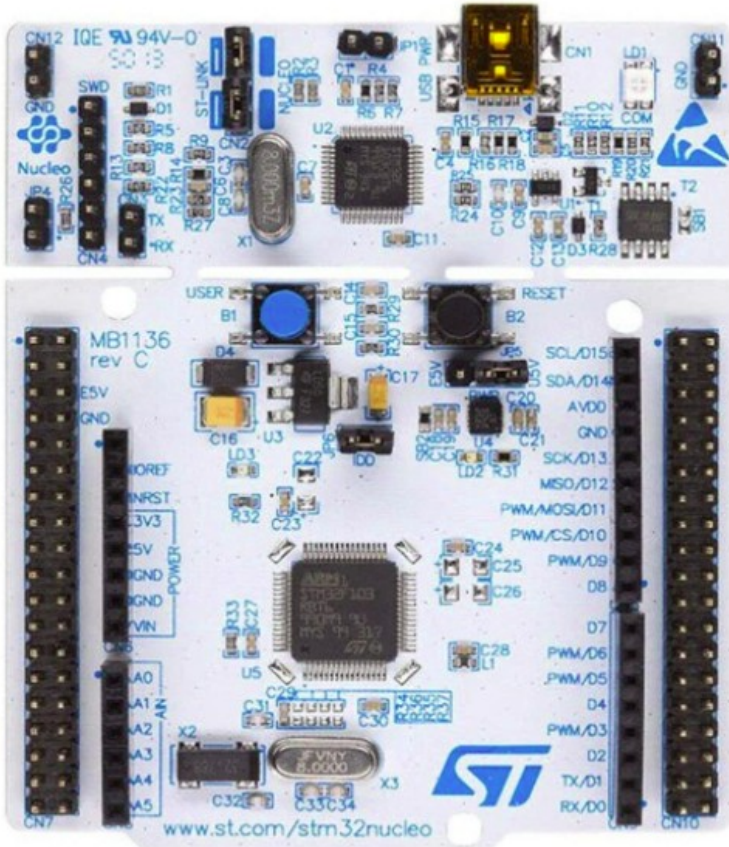
2. key features of the project

- Sensor Initialization with State Machine and Application state Machine maintained.
 - UART Communication is based on Interrupt.
 - Common function for UART receive data parsing.
 - Timer based delay.
 - STM32 Application is Master and configured W10 (Wi-Fi) module accordingly with AT commands.
 - FreeRTOS Task with priority implimentation.
 - QUEUE are used for inter proces communication.
 - Data is pushed to the MQTT server.
 - Data is in JASON format.
 - Temperature and Humidity data is pushed in every 1 Min.
-

3. Hardware Used and there Specification

1. STM32F411RE microcontroller

The STM32F411RE is a high-performance microcontroller based on the ARM Cortex-M4 processor. It has a number of features that make it well-suited for a variety of applications, including:



100 MHz CPU clock speed

128 KB of RAM

512 KB of Flash memory

Floating point unit (FPU)

11 general-purpose timers

13 communication interfaces

USB OTG RTC

RUGGEDBOARD

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. Rugged Board 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

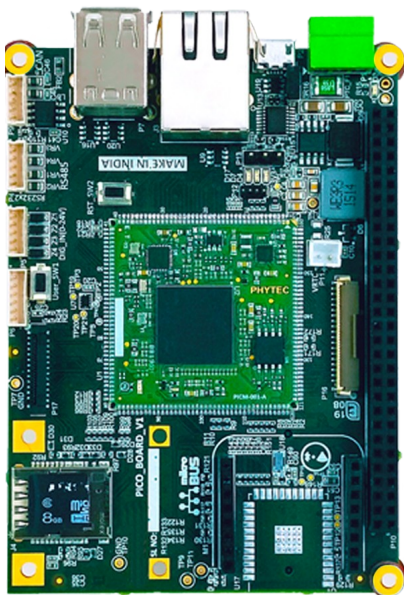
Power

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



A5D2x @500MHz
CORTEX - A5
64MB RAM
32MB FLASH

RS-232



2 x RS232

RS-485



1 x RS485

CAN



1 x CAN



1 x ETHERNET



TFT & CAP TOUCH



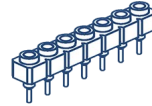
1 x MICROSD SLOT



2 x USB



DC & USB POWER



EXPANSION HEADER



mikroBUS CONN.



mPCIe CONN.



MICRO SIM SLOT

4.AHT 25 sensor

Interface definition

Pin	Name	Paraphrase
1	VDD	Connect power (2.2-5.5V)
2	SDA	Serial data, two-way
3	GND	Power to
4	SCL	Serial clock, two-way



AHT25 Pin Distribution (Top View)

4. AHT25 Humidity and Temperature Module

- Relative humidity and temperature output
- Superior sensor performance, typical accuracy RH: $\pm 2\%$, T: $\pm 0.3^{\circ}\text{C}$
- Fully calibrated and processed digital output, I²C protocol
- Wide voltage support 2.2 to 5.5V DC

-
- Excellent long-term stability
 - Fast-response and anti-interference capability.

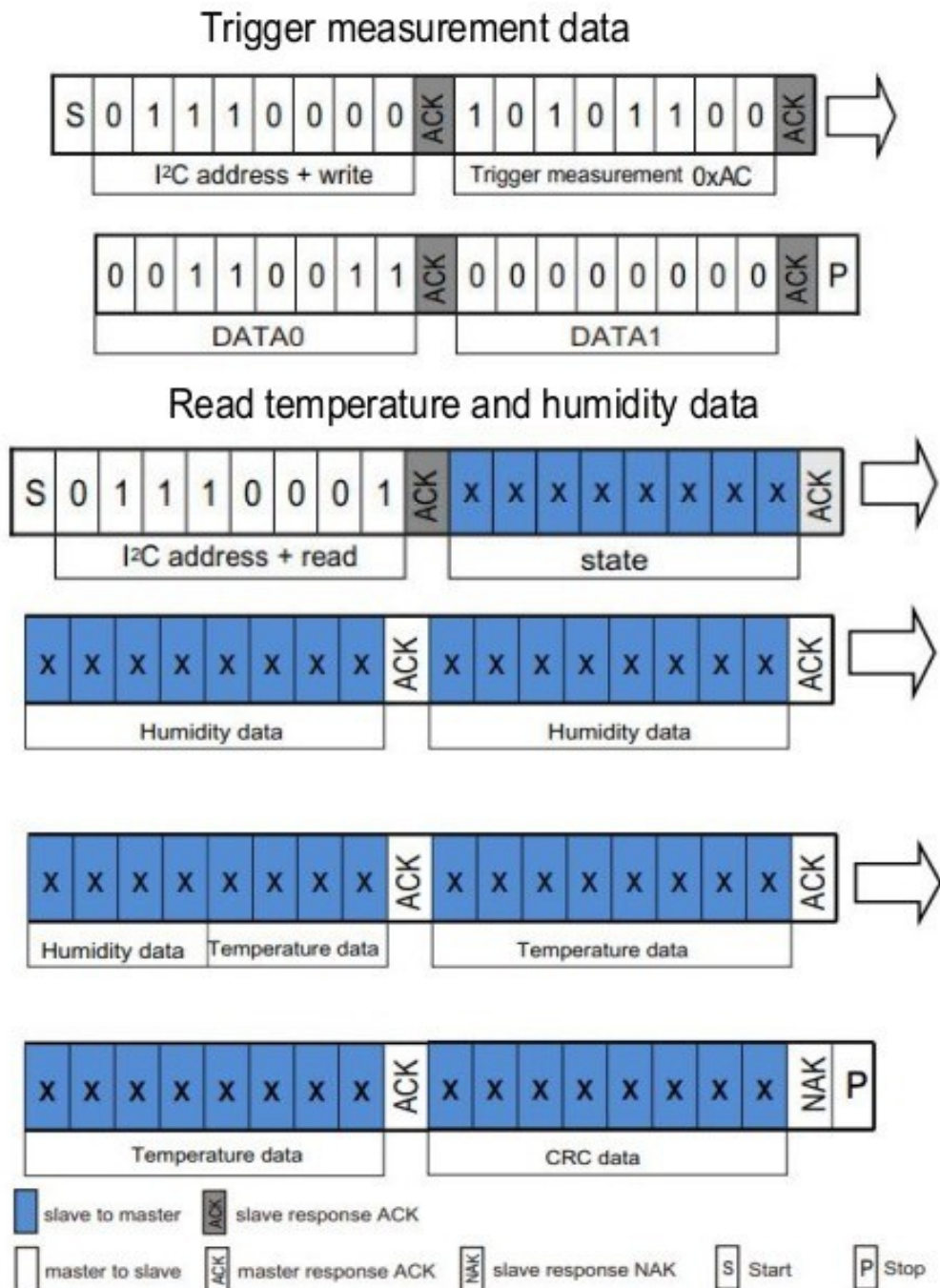
Note:

1. Ensure that the host MCU's power supply voltage aligns with the sensor's requirements during circuit operation.
2. Enhance system reliability by implementing control over the sensor's power supply.
3. During the initial power-up of the sensor, prioritize providing power to the sensor's VDD. Set SCL and SDA high levels after a 5ms delay.

To prevent signal conflicts, restrict the microprocessor (MCU) to driving SDA and SCL only at low levels. Employ an external pull-up resistor, such as a 4.7k Ω resistor, to raise the signal to a high level.

4.1 SLAVE ADDRESS AND CONFIGURATIONS

- ❖ **AHT25 read address 0x70**
- ❖ **AHT25 write address 0x71**
- ❖ **AHT25 start measure address 0xAC**



4.2 SIGNAL CONVERSION

Relative Humidity Conversion:

The relative humidity RH can be calculated according to the relative humidity signal S RH output by SDA through the following formula:

$$RH[\%] = \left(\frac{S_{RH}}{2^{20}} \right) * 100\%$$

Temperature Conversion:

The temperature T can be calculated by substituting the temperature output signal S T into the following formula:

(The result is expressed in temperature °C):

$$T[^\circ C] = \left(\frac{S_T}{2^{20}} \right) * 200 - 50$$

5. W10 WiFi



The W10 WiFi module is a low-cost, easy-to-use WiFi module that can be used to connect IoT devices to the internet. The module has a built-in TCP/IP stack, so it can be easily connected to a variety of IoT platforms. The module also has a number of other features, such as:

- 100mW transmit power

- 11Mbps data rate

- 802.11 b/g/n compatibility

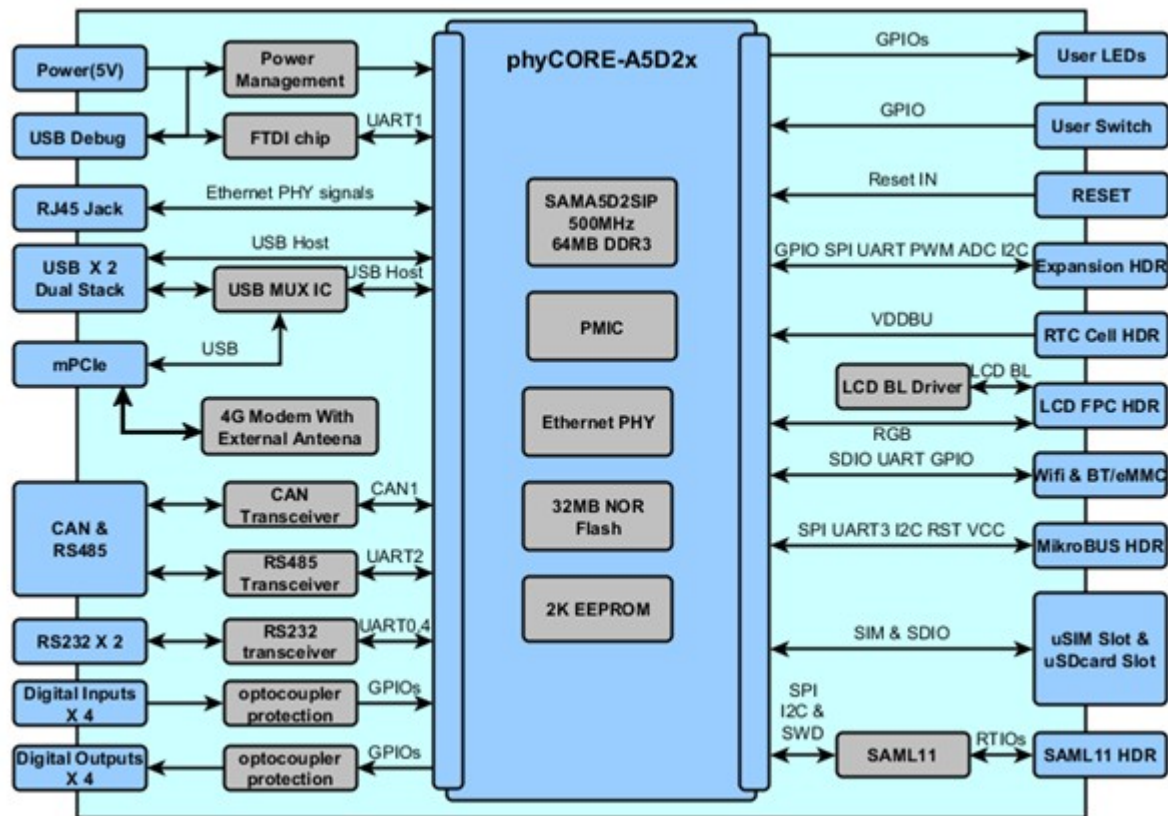
- Integrated antenna

6. RUGGED BOARD

The RuggedBoard for phyCORE-A5D2x is a SIP (System in Peripheral) which is a low-cost, feature-rich software development platform supporting the Microchip's A5D2x microprocessor. Moreover, due to the numerous standard interfaces the RuggedBoard A5D2x can serve as bedrock for your application. At the core of the RuggedBoard is the phyCORE-A5D2x System On Module (SOM) in a direct solder form factor, containing the processor,

Flash, power regulation, supervision, transceivers, and other core functions required to support the A5D2x processor. Surrounding the SOM is the RuggedBoard carrier board, adding power input, buttons, connectors, signal breakout, Ethernet and mikro-BUS connectivity amongst other things.

This RuggedBoard offers an ultra-low cost Single Board Computer for the A5D2x processor, while maintaining most of the advantages of the SOM concept. Adding the phyCORE-A5D2x SOM into your own design is as simple as ordering the connector version and making use of our RuggedBoard Carrier Board reference schematics.



The RuggedBoard has the following features

- 1 x Ethernet
- 2 x RS-232
- 1 x RS-485 (Isolated)
- 1 x CAN
- 4x DIN (Isolated)
- 4x DOUT
- 1 x LVDS Display
- 1 x Micro SD
- 1 x SIM
- 2 x USB 2.0
- 1 x mikroBUS
- 1 x 60 PIN Expansion Headers

7. Why interrupt based uart communication

The advantages of interrupt-based UART communication over other methods:

- **Improved efficiency:** The CPU is not constantly polling the UART status, so it can be used for other tasks. This can improve the efficiency of the system and reduce power consumption.
- **Reduced latency:** Interrupt-based communication can be faster than polling, as the CPU is only interrupted when data is ready to be sent or received. This can improve the responsiveness of the system.
- **Better multitasking:** Interrupt-based communication allows the CPU to handle multiple UART events simultaneously. This can be useful for applications that require frequent and asynchronous serial communication.

Here are some of the disadvantages of interrupt-based UART communication:

- **More complex code:** Interrupt-based communication is more complex to implement than polling. This is because the programmer needs to write code to handle the interrupts.
- **More overhead:** Interrupt-based communication can have more overhead than polling. This is because the CPU needs to save and restore its state when it is interrupted.

Overall, interrupt-based UART communication is a more efficient and responsive way to communicate with serial devices. However, it is more complex to implement and can have more overhead.

Here are some examples of applications where interrupt-based UART communication would be a good choice:

- **Sensor networks:** Sensor networks often need to communicate with each other or with a central server. Interrupt-based communication can be used to improve the efficiency and responsiveness of these networks.
-

-
- **Wireless modules:** Wireless modules often use UART to communicate with the host microcontroller. Interrupt-based communication can be used to improve the performance of these modules.
 - **Real-time systems:** Real-time systems often need to communicate with other devices in a timely manner. Interrupt-based communication can be used to ensure that these communications are not missed.

STAGE

STAGE:1

Interfacing AHT25 sensor to STM32 and reading the values from the sensor and displaying it on live expression and minicom.

Task 1: Connect the AHT25 sensor to the STM32 microcontroller.

Task 2: Write code to read the temperature and humidity values from the AHT25 sensor.

Task 3: Display the temperature and humidity values on a live expression and minicom.

STAGE:2

Publishing the sensor data to MQTT via a WE10 Wi-Fi module.

Task 1: Connect the WE10 WiFi module to the STM32 microcontroller.

Task 2: Write code to publish the temperature and humidity values to an MQTT broker.

Task 3: Verify that the sensor data is being published to the MQTT broker.

STAGE:3

The Ruggedboard with STM32F446RE is used to transmit the data and get the AHT25 value in the ruggedboard minicom.

Task 1: Connect the Rugged Board to the STM32 microcontroller.

Task 2: Write code to transfer the temperature and humidity values for STM32 microcontroller.

Task 3: Write code to receive the temperature and humidity values for Rugged Board.

Task 4: Display the temperature and humidity values on the minicom.

STAGE:4

The Ruggedboard with STM32F446RE is used to transmit the data and get the AHT25 value in the ruggedboard minicom and pass the value into the Right-tech by using W10 module is connected with the Rugged board.

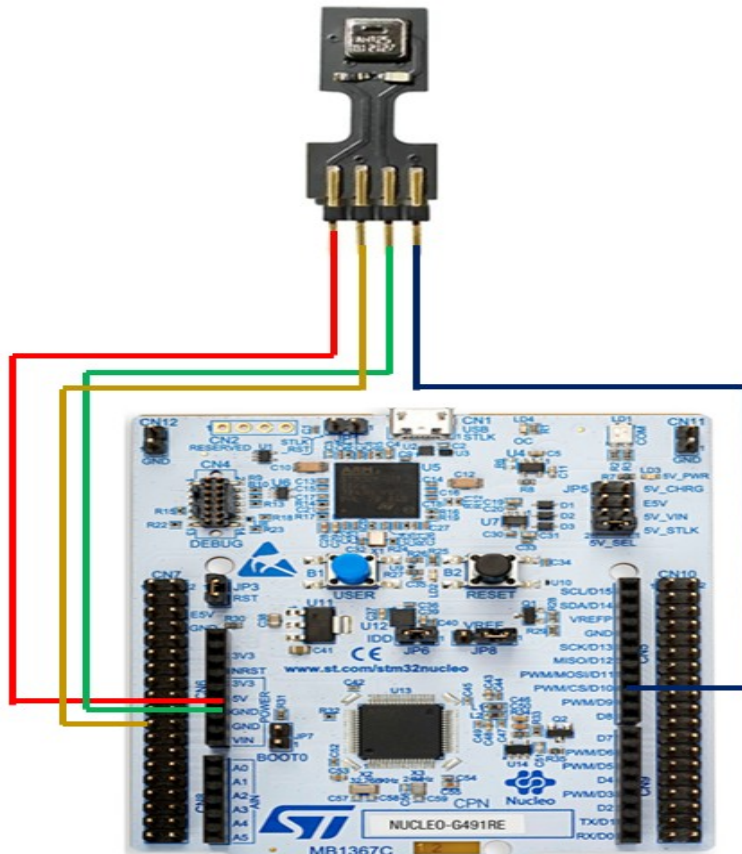
Task 1: Integrate the code from stages 1, 2, and 3 into a single application.

Task 2: Test the integrated application to ensure that it is working correctly.
Task 3: Deploy the application to the STM32 microcontroller.

8. PROJECT STAGES

8.1 STAGE-1

Stage 1: Interfacing AHT25 sensor to STM32 and reading the values from the sensor and displaying it on live expression and minicom.



Task 2: Write code to read the temperature and humidity values from the AHT25 sensor.

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

    HAL_I2C_Master_Transmit(&hi2c1, AHT25_ADDR, &cmd, 1, HAL_MAX_DELAY);
    HAL_Delay(100);
    HAL_I2C_Master_Receive(&hi2c1, 0x71, data, 6, HAL_MAX_DELAY);

    *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;
}
```

The code first declares two variables, data and cmd. The data variable will be used to store the raw sensor data, and the cmd variable will be used to send the measurement command to the sensor. The next few lines of code send the measurement command to the sensor and then wait 100 milliseconds for the sensor to complete the measurement.

The next line of code reads the raw sensor data into the data variable. The last two lines of code convert the raw sensor data into floating-point values and store them in the temperature and humidity variables.

The read_sensor_values() function is a simple example of how to read sensor data from an AHT25 sensor using the HAL I2C driver. The function takes two pointers to floating-point variables as input, and it stores the temperature and humidity values in these variables. The function returns void.

```
*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;
```

The first line of code converts the raw humidity data from the sensor into a floating-point value. The raw humidity data is stored in the data[1], data[2], and data[3] bytes of the data array. The << and | operators are used to combine the three bytes into a single 32-bit integer. The / operator is then used to divide the integer by 1048576, which is the maximum value that can be stored in a 32-bit integer. The * operator is then used to multiply the result by 100, which converts the value into a percentage.

The second line of code converts the raw temperature data from the sensor into a floating-point value. The raw temperature data is stored in the data[3], data[4], and data[5] bytes of the data array. The & operator is used to mask out the lower 4 bits of the data[3] byte. The << and | operators are then used to combine the three bytes into a single 32-bit integer. The / operator is then used to divide the integer by 1048576, which is the maximum value that can be stored in a 32-bit integer. The * operator is then used to multiply the result by 200, which converts the value into degrees Celsius. The - 50.0 expression is then used to subtract 50 degrees Celsius from the result, which converts the value into degrees Fahrenheit.

Task 3: Display the temperature and humidity values on a live expression and minicom.

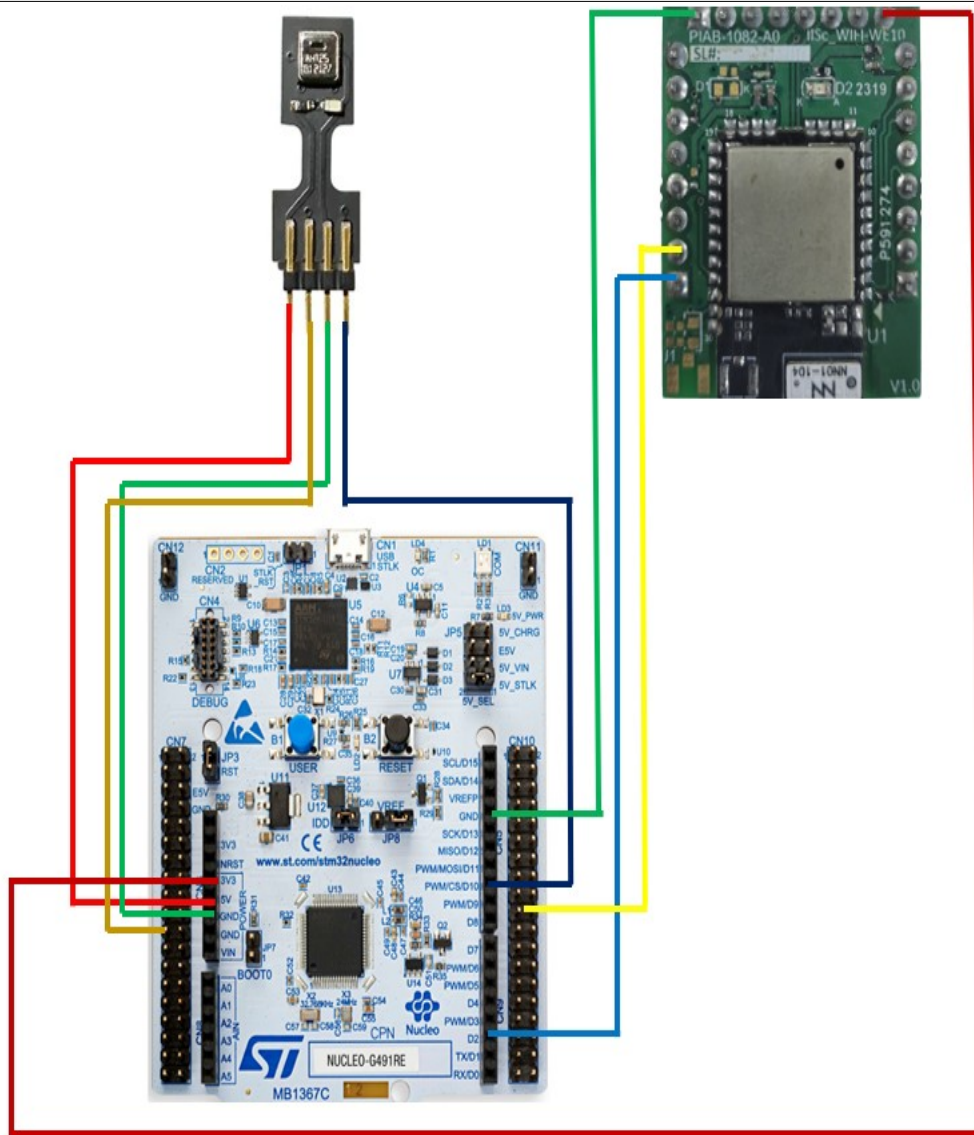
Expression	Type	Value
temperature	float	29.2657852
humidity	float	63.1189346

```
Temperature=27.15 Humdity=65.96
Temperature=27.16 Humdity=65.83
Temperature=27.20 Humdity=65.73
Temperature=27.23 Humdity=65.70
Temperature=29.21, Humidity=62.55
Temperature=27.96, Humidity=59.99
```

STAGE-2

Stage 2: Publishing the sensor data to MQTT via a WE10 Wi-Fi module.

Task 1: Connect the WE10 WiFi module to the STM32 microcontroller.



PIN CONFIGURATION

PIN	PIN NUMBER	COMPONENTS
VCC 5V	VCC	W10 AND AHT25
ground	GND	W10 AND AHT25
SCL	PB8	AHT25
SDK	PB9	AHT25
TX	PA9	W10
RX	PA10	W10

Task 2: Write code to publish the temperature and humidity values to an MQTT broker.

Code snippet of WiFi module initialization

```
void MQTT_Init()
```

```
char buffer[128];
```

```
/******CMD+MQTTNETCFG *****/
```

```
    //memset(&buffer[0],0x00,strlen(buffer));
    sprintf (&buffer[0], "CMD+MQTTNETCFG=dev.rightech.io,1883\r\n");
    HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 10000);
    HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 10000);
    //memset(&buffer[0],0x00,strlen(buffer));
    //HAL_Delay(500);
    HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 10000);
    HAL_Delay(500);
    HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 10000);
```

```
/******CMD+MQTTCONCFG---->LED *****/
```

```
    //memset(&buffer[0],0x00,strlen(buffer));
    sprintf (&buffer[0], "CMD+MQTTCONCFG=3,mqtt-shruthi7352-okag7k,,,,,,,,,\r\n");
    HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
    HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
    //memset(&buffer[0],0x00,strlen(buffer));
    //HAL_Delay(500);
    HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
    HAL_Delay(500);
    HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
```

```
/******CMD+MQTTSTART *****/
```

```
    //memset(&buffer[0],0x00,strlen(buffer));
    sprintf (&buffer[0], "CMD+MQTTSTART=1\r\n");
    HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
    HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
    // memset(&buffer[0],0x00,strlen(buffer));
    HAL_Delay(5000);
    HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
    HAL_Delay(500);
    HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
```

```
/******CMD+MQTTSUB *****/
```

```
    //memset(&buffer[0],0x00,strlen(buffer));
    sprintf (&buffer[0], "CMD+MQTTSUB=base/relay/led1\r\n");
    HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
    HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
    HAL_Delay(500);
```

```
HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
HAL_UART_Transmit(&huart2,(uint8_t*)buffer, strlen(buffer), 1000);
```

- The code first declares a buffer of 128 characters. The buffer will be used to store the commands that are sent to the WE10 module.
- The next few lines of code send the CMD+RESET command to the WE10 module. This command resets the module to its default state.
- The next line of code sends the CMD+WIFIMODE=1 command to the WE10 module. This command sets the module to operate in Wi-Fi mode.
- The next line of code sends the CMD+CONTOAP=SSID, PASSWD command to the WE10 module. This command configures the module to connect to the Wi-Fi network with the specified SSID and password.
- The next line of code sends the CMD?WIFI command to the WE10 module. This command queries the module for its WiFi status. The last line of code waits for 2000 milliseconds and then receives a response from the WE10 module. The response is stored in the buffer.
- The WE10_Init() function is a simple example of how to initialize a WE10 module and connect it to a Wi-Fi network. The function takes no arguments and it returns void.

Code snippet for MQTT initialization

```
void MQTT_Init()
```

```
char buffer[128];
```

```
/******CMD+MQTTNETCFG *****/
```

```
    //memset(&buffer[0],0x00,strlen(buffer));
    sprintf (&buffer[0], "CMD+MQTTNETCFG=dev.rightech.io,1883\r\n");
    HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 10000);
    HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 10000);
    //memset(&buffer[0],0x00,strlen(buffer));
    //HAL_Delay(500);
    HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 10000);
    HAL_Delay(500);
    HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 10000);
```

```
/******CMD+MQTTCONCFG---->LED *****/
```

```
    //memset(&buffer[0],0x00,strlen(buffer));
    sprintf (&buffer[0], "CMD+MQTTCONCFG=3,mqtt-shruthi7352-okag7k,,,,,,,,,\r\n");
    HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
    HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
```

```

//memset(&buffer[0],0x00,strlen(buffer));
//HAL_Delay(500);
HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
HAL_Delay(500);
HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);

/*****CMD+MQTTSTART *****/

//memset(&buffer[0],0x00,strlen(buffer));
sprintf (&buffer[0], "CMD+MQTTSTART=1\r\n");
HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
// memset(&buffer[0],0x00,strlen(buffer));
HAL_Delay(5000);
HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
HAL_Delay(500);
HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);

/*****CMD+MQTTSUB *****/




//memset(&buffer[0],0x00,strlen(buffer));
sprintf (&buffer[0], "CMD+MQTTSUB=base/relay/led1\r\n");
HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
HAL_Delay(500);
HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
}

```

- The code you provided is used initialize a WE10 module and connect it to an MQTT broker. The code first declares a buffer of 128 characters. The buffer will be used to store the commands that are sent to the WE10 module.
 - The next few lines of code send the CMD+MQTTNETCFG command to the WE10 module. This command configures the module to connect to the MQTT broker at dev.rightech.io on port 1883.
 - The CMD+MQTTCONCFG command configures the module to connect to the MQTT broker as a client with the username mqtt-shruthi7352-okag7k and no password.
 - The CMD+MQTTSTART command starts the MQTT client and connects to the broker.
 - The CMD+MQTTSUB command subscribes the client to the topic base/state/temperature and base/state/humidity.
-

-
- The MQTT_Init() function is a simple example of how to initialize a WE10 module and connect it to an MQTT broker. The function takes no arguments and it returns void.
 - Here is a more detailed explanation of the code:
 - The configuration and initiation of MQTT settings for the WE10 module involve several commands. The CMD+MQTTNETCFG command facilitates the configuration of MQTT parameters, with the first parameter specifying the hostname or IP address of the MQTT broker, and the second parameter denoting the port number. Subsequently, the CMD+MQTTCONCFG command is employed to set up the MQTT client, where the first parameter designates the username and the second parameter signifies the password of the MQTT client.
 - Once the configuration is in place, the CMD+MQTTSTART command is executed to initiate the MQTT client on the WE10 module. This command establishes a connection between the client and the MQTT broker. Additionally, the CMD+MQTTSUB command is utilized for subscribing the MQTT client to a particular topic, with the first parameter specifying the desired topic. This comprehensive set of commands ensures the proper configuration and functioning of the MQTT capabilities of the WE10 module.

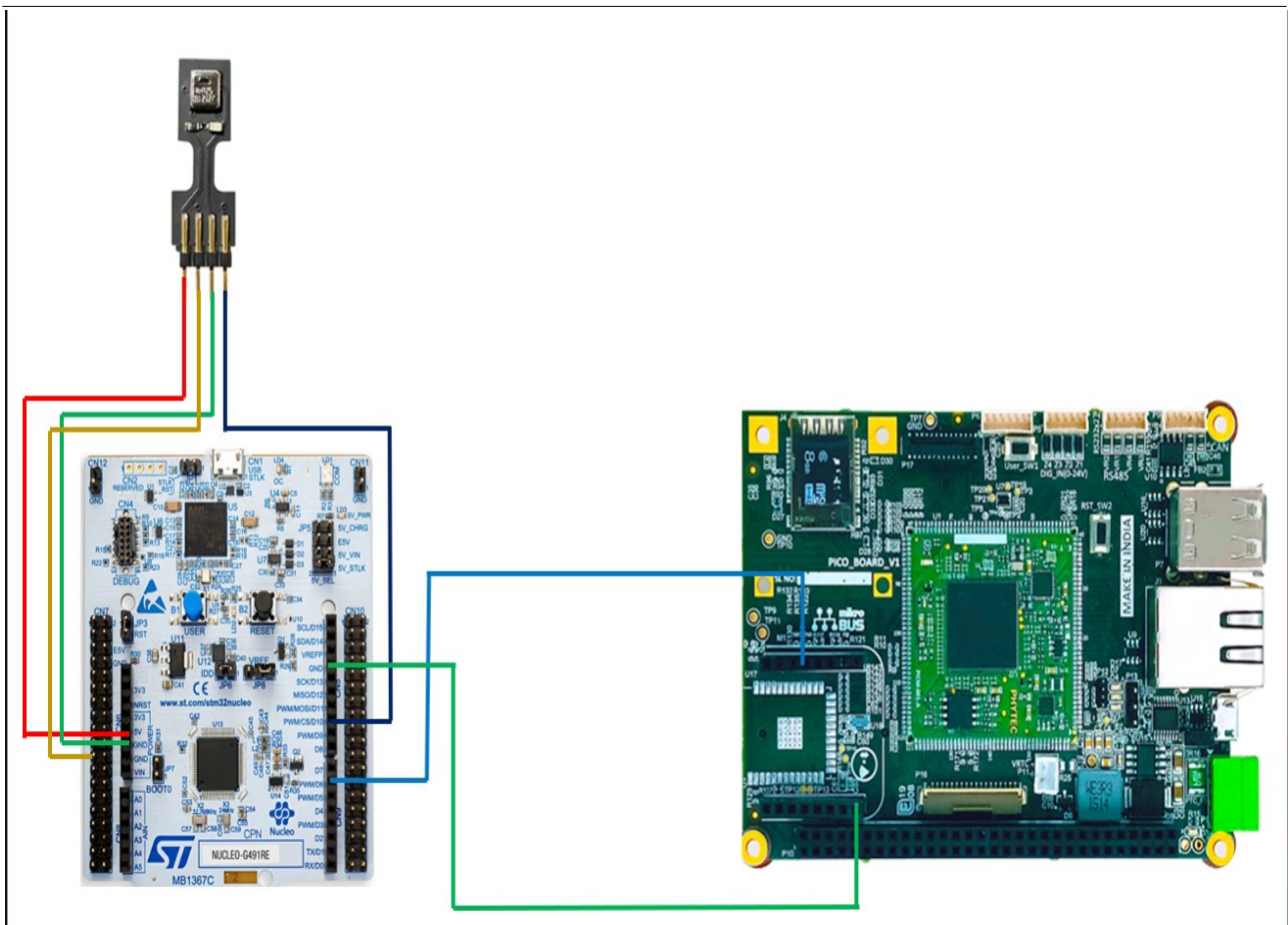
Task 3: Verify that the sensor data is being published to the MQTT broker.

 yes Online	20/11/2023 19:56:28 Server time
^ Params	
 26.88°C Min Temperature	 79.87 % Mid Humidity
^ Position (JSON)	
- deg Latitude	- deg Longitude
^ Last MQTT Publish	
base/state/humidity Topic	79.87 Payload

STAGE-3

In Stage 3, the sensor data is transferred to the Rugged Board, and the minicom is used to display the sensor data.

Task 1: Connect the Rugged Board to the STM32 microcontroller.



In stm32f411re with AHT25 :

PIN	PIN NUMBER	COMPONENTS
VCC 5V	VCC	AHT25
GROUND	GND	AHT25
SCL	PC6	AHT25
SDA	PB9	AHT25
TX	PA9	RuggedBoard

In Ruggedboard with W10:

PIN	PIN NUMBER	COMPONENTS
VCC 5V	VCC	W10
GROUND	GND	W10
TX	TX	W10
RX	RX	Stm32f411re

Task 2: Write code to transfer the temperature and humidity values for STM32 microcontroller.

```
while (1)
{
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
    read_sensor_values(&temperature, &humidity);

    sprintf(buffer, "Temperature=%.2f\r\n", temperature);
    HAL_UART_Transmit(&huart2, (uint8_t *)buffer,
strlen(buffer), HAL_MAX_DELAY);
    HAL_UART_Transmit(&huart6, (uint8_t *)buffer,
strlen(buffer), HAL_MAX_DELAY);
    HAL_Delay(1000);
}
```

Creating the Data String:

The line of code, `sprintf(buffer, "%.2f,%.2f\r\n", temperature, humidity);`, is responsible for formatting temperature and humidity values into a string suitable for transmission over UART. The `sprintf()` function takes three parameters:

- Buffer: A pointer to the buffer where the formatted string will be stored.
- `"%.2f,%.2f\r\n"`: A format string specifying how temperature and humidity values should be formatted.
- Temperature, humidity: The actual temperature and humidity values to be formatted.

The `"%.2f"` format specifier dictates that values should be formatted as floating-point numbers with two decimal places, while `"\r\n"` denotes a carriage return and a newline character.

Transmitting Data via UART2 and UART6:

The subsequent two lines of code transmit the formatted data string to UART2 and UART3, respectively. The `HAL_UART_Transmit()` function is employed with four arguments:

- `huart`: A pointer to the UART handle.
 - `(uint8_t *)buffer`: A pointer to the data buffer intended for transmission.
 - `strlen(buffer)`: The length of the data buffer.
 - `HAL_MAX_DELAY`: The maximum delay time allowed for the transmission.
- Following the transmission, there's a line of code introducing a 30-second delay, likely implemented to prevent overwhelming the UART ports or to allow sufficient time for receiving devices to process the data. This coding pattern is commonly employed for sending sensor data to external devices or logging it into a file.
-

Task 3: Write code to receive the temperature and humidity values for Rugged Board.

```
//char stm_data_rev[1024];
bytes_read_stm = mraa_uart_read(uart, stm_data_rev, sizeof(stm_data_rev)-1);

sleep(10);
if (bytes_read_stm > 0)
{
    // Process and handle received data here
    stm_data_rev[bytes_read_stm] = '\0';
    printf("STM received data(temperature,humidity): %s\n\n", stm_data_rev);
}
//char stm_data_rev[1024]={ "27.99,63.37\r\n"};
const char *delimiter = ",";

char *temperature = strtok(stm_data_rev, delimiter);
char *humidity = strtok(NULL, "\0");
```

❖ Reading Data from UART Port:

`bytes_read_stm = mraa_uart_read(uart, stm_data_rev, sizeof(stm_data_rev)-1);`

This line of code reads data from the UART port connected to the STM32 microcontroller. The `mraa_uart_read()` function takes three arguments:

- `uart`: A pointer to the UART object
- `stm_data_rev`: A buffer to store the received data
- `sizeof(stm_data_rev)-1`: The size of the buffer

The function returns the number of bytes that were read.

● Processing Received Data:

```
if (bytes_read_stm > 0)
{
    // Process and handle received data here
    stm_data_rev[bytes_read_stm] = '\0';
    printf("STM received data(temperature,humidity): %s\n\n", stm_data_rev);
}
```

This section of code verifies whether there is any incoming data from the UART port. If data is present, the code proceeds to process and manage the received data. The statement `stm_data_rev[bytes_read_stm] = '\0'` is included to append a null terminator to the data buffer. This step is essential to ensure the proper functionality of the `printf()` function.

● Declaring Delimiter:

```
const char *delimiter = ",";
```

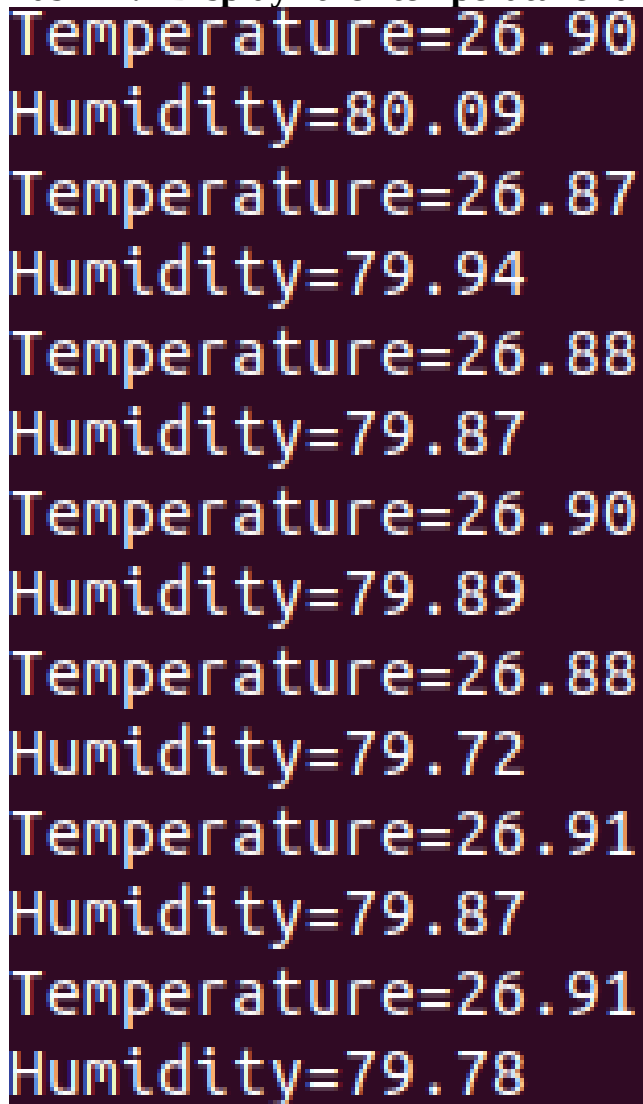
This line of code defines a constant variable named "delimiter" and assigns it the value. The purpose of this delimiter is to facilitate the separation of the received data into two distinct parts: temperature and humidity.

- **Splitting Received Data:**

```
char *temperature = strtok(stm_data_rev, delimiter);  
char *humidity = strtok(NULL, "\\0");
```

In these two lines of code, the `strtok()` function is employed to divide the received data into two segments. In the initial call to `strtok()`, the data buffer and the delimiter are provided as arguments, resulting in a pointer to the first token, representing the temperature. The subsequent call to `strtok()` utilizes a `NULL` pointer as an argument, indicating that it will use the delimiter from the preceding call to separate the remaining data. This second call yields a pointer to the second token, representing the humidity.

Task 4: Display the temperature and humidity values on the minicom.



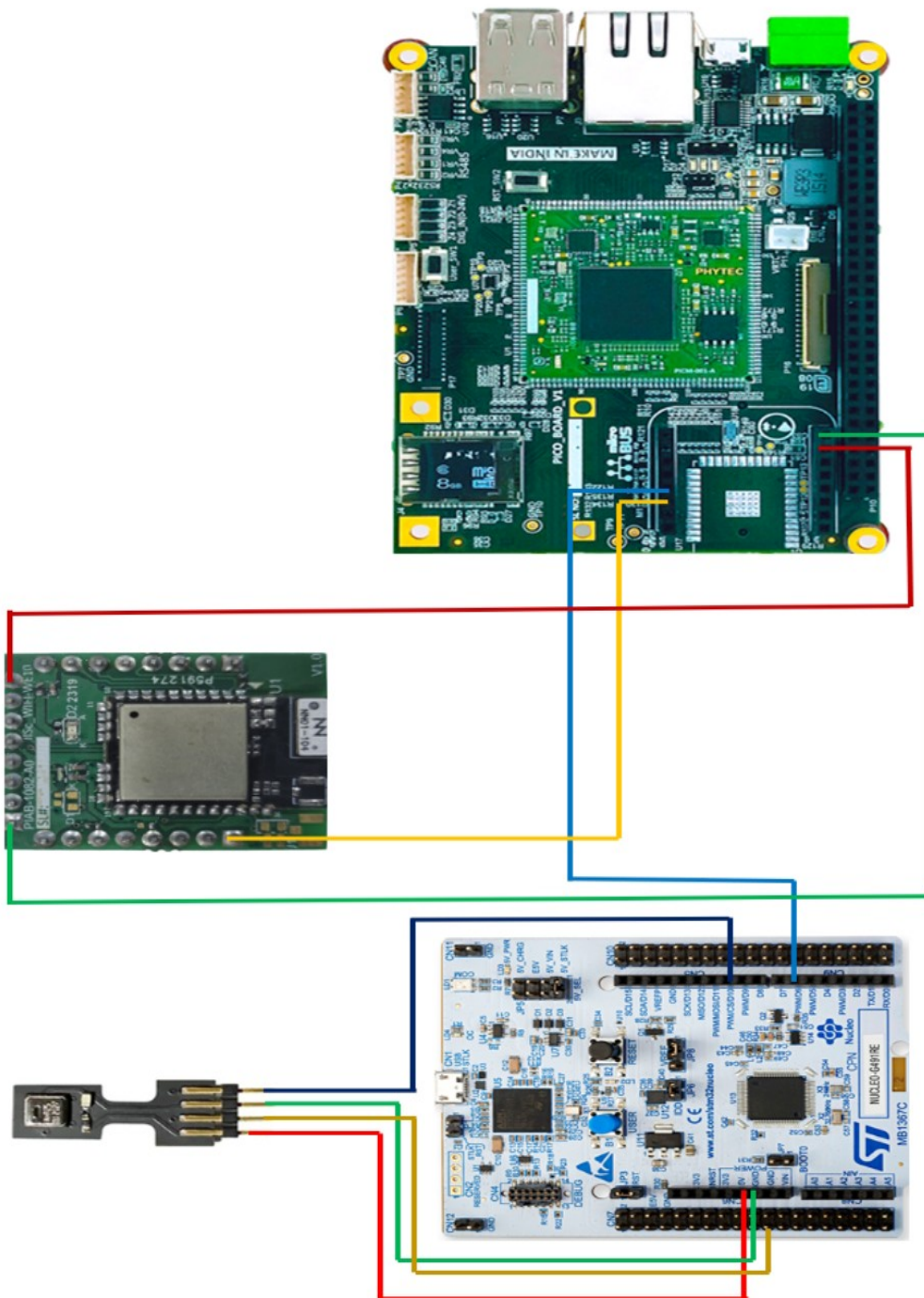
The image shows a terminal window with a dark background and light-colored text. It displays a series of temperature and humidity readings, each on a new line. The text is as follows:

```
Temperature=26.90  
Humidity=80.09  
Temperature=26.87  
Humidity=79.94  
Temperature=26.88  
Humidity=79.87  
Temperature=26.90  
Humidity=79.89  
Temperature=26.88  
Humidity=79.72  
Temperature=26.91  
Humidity=79.87  
Temperature=26.91  
Humidity=79.78
```

8.4 STAGE-4

Combining stages 1, 2, and 3

Task 1: Establish a connection between the AHT25 sensor and the STM32 microcontroller, and interconnect both the WE10 module and the STM32 microcontroller with the Rugged Board.



Task 2:

-
- Write code to transfer the temperature and humidity values for STM32 microcontroller.
 - Write code to receive data from STM32 and publish the data to MQTT broker using Rugged Board.

 yes
Online

20/11/2023 19:56:28
Server time

^ Params

 26.88°C Min
Temperature

 79.87 % Mid
Humidity

^ Position (JSON)

- deg
Latitude

- deg
Longitude

^ Last MQTT Publish

base/state/humidity
Topic

79.87
Payload

9. Further Scope

Scope 1: Integration of Additional Environmental Sensors

Expand the system's capabilities by integrating additional environmental sensors such as air quality sensors, soil moisture sensors, or light sensors. This would provide a more comprehensive understanding of the monitored environment and enhance the system's applicability across various scenarios.

Scope 2: Implementing Predictive Analytics

Develop predictive analytics features to forecast potential environmental changes based on historical data. By implementing machine learning algorithms, the system could offer insights into trends, enabling users to proactively address environmental issues before they escalate.

Scope 3: Enhancing Security Measures

Strengthen the security aspects of the IoT-based monitoring system. Implement encryption protocols and authentication mechanisms to ensure the confidentiality and integrity of the transmitted data. Additionally, explore the incorporation of secure over-the-air (OTA) updates for system patches and improvements.

10. REAL TIME APPLICATION'S

1. Smart Agriculture Monitoring:

The IoT-based temperature and humidity monitoring system can be employed in agriculture to provide real-time environmental data. Farmers can use the data to optimize irrigation schedules, assess crop health, and make informed decisions about planting and harvesting.

2. Warehouse Climate Control:

Implementing the system in warehouses allows for continuous monitoring of temperature and humidity levels. This real-time data is crucial for preserving the quality of stored goods, especially in industries where climate-sensitive products, such as pharmaceuticals or food items, are stored.

3. Energy-Efficient HVAC Systems:

The system can be integrated into heating, ventilation, and air conditioning (HVAC) systems in buildings. Real-time monitoring of environmental conditions enables more efficient control of HVAC systems, adjusting temperature and humidity levels based on actual needs, thus optimizing energy consumption.

4. Eco-friendly Urban Planning:

Deploying the IoT monitoring system in urban areas can contribute to sustainable urban planning. By continuously collecting data on temperature and humidity, city planners can gain insights into microclimates, helping to design green spaces, optimize energy usage, and enhance overall urban environmental quality.

11. STAGE-2 and 4 Code

11.1 STAGE-2 CODE

```
/* USER CODE BEGIN Header */
/**
 * *****
 *
 * @file      : main.c
 * @brief     : Main program body
 * *****
 *
 * @attention
 *
 * Copyright (c) 2023 STMicroelectronics.
 * All rights reserved.
 *
 * This software is licensed under terms that can be found in the LICENSE file
 * in the root directory of this software component.
 * If no LICENSE file comes with this software, it is provided AS-IS.
 *
 * *****
 *
 */
/* USER CODE END Header */
/* Includes -----*/
#include "main.h"
#include "stdio.h"
#include "string.h"
#define AHT25_ADDR 0x70
#define AHT25_INIT_CMD 0xE1
#define AHT25_MEASURE_CMD 0xAC

/* Private includes -----*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private typedef -----*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define -----*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro -----*/
/* USER CODE BEGIN PM */
```

```
/* USER CODE END PM */

/* Private variables -----*/
I2C_HandleTypeDef hi2c1;

UART_HandleTypeDef huart1;
UART_HandleTypeDef huart2;

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_I2C1_Init(void);
static void MX_USART1_UART_Init(void);
void WE10_Init (char *SSID, char *PASSWD);
void MQTT_Init();
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code -----*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/**
 * @brief The application entry point.
 * @retval int
 */
float temperature, humidity;

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

    HAL_I2C_Master_Transmit(&hi2c1, AHT25_ADDR,&cmd, 1, HAL_MAX_DELAY);
    HAL_Delay(100);
    HAL_I2C_Master_Receive(&hi2c1,0x71, data, 6, HAL_MAX_DELAY);

    *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;
}

void mqtt_data_send()
{
char buffer[50];

sprintf (&buffer[0], "CMD+MQTTPUB=base/state/temperature,%.2f\r\n",temperature);
HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
HAL_Delay(100);
sprintf (&buffer[0], "CMD+MQTTPUB=base/state/humidity,%.2f\r\n",humidity);
HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
}
int main(void)
{
/* USER CODE BEGIN 1 */
    char buffer[50];
/* USER CODE END 1 */

/* MCU Configuration-----*/

/* Reset of all peripherals, Initializes the Flash interface and the Systick. */
HAL_Init();

/* USER CODE BEGIN Init */

/* USER CODE END Init */

/* Configure the system clock */
SystemClock_Config();

/* USER CODE BEGIN SysInit */

/* USER CODE END SysInit */

/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_USART2_UART_Init();
MX_I2C1_Init();
MX_USART1_UART_Init();
WE10_Init("pavi","pavi0205");
MQTT_Init();
/* USER CODE BEGIN 2 */

/* USER CODE END 2 */

/* Infinite loop */
```

```
/* USER CODE BEGIN WHILE */
while (1)
{
    /* USER CODE END WHILE */

    read_sensor_values(&temperature,&humidity);
    sprintf(buffer, "Temperature=%.2f°C\r\n", temperature);
    HAL_UART_Transmit(&haurt2, (uint8_t *)buffer, strlen(buffer),1000);
    mqtt_data_send();
    HAL_Delay(1000);

    sprintf(buffer, "Humidity=%.2f%%\r\n", humidity);
    HAL_UART_Transmit(&haurt2, (uint8_t *)buffer, strlen(buffer),1000);
    HAL_Delay(1000);
    // printf("temperature:%f\n humidity:%f\n",temperature,humidity);
    /* USER CODE BEGIN 3 */
}
/* USER CODE END 3 */
}
```

```
void WE10_Init (char *SSID, char *PASSWD)
```

```
{
    char buffer[128];
    /****** CMD+RESET *****/
    //memset(&buffer[0],0x00,strlen(buffer));
    sprintf (&buffer[0], "CMD+RESET\r\n");
    HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
    HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);

    HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
    HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);

    /****** CMD+WIFIMODE=1 *****/
    //memset(&buffer[0],0x00,strlen(buffer));
    sprintf (&buffer[0], "CMD+WIFIMODE=1\r\n");
    HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
    HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);

    HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
    HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);

    /****** CMD+CONTOAP=SSID,PASSWD *****/
    //memset(&buffer[0],0x00,strlen(buffer));
    sprintf (&buffer[0], "CMD+CONTOAP=pavi,pavi0205");
    HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 10000);
    HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 10000);
    //memset(&buffer[0],0x00,strlen(buffer));
}
```

```

    HAL_Delay(2000);
    HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 10000);
    HAL_Delay(500);
    HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 10000);

    /***** CMD?WIFI *****/
    //memset(&buffer[0],0x00,strlen(buffer));
    sprintf (&buffer[0], "CMD?WIFI\r\n");
    HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 10000);
    HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 10000);
//    memset(&buffer[0],0x00,strlen(buffer));
//    HAL_Delay(500);
    HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 10000);
    HAL_Delay(500);
    HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 10000);

}

void MQTT_Init()
{

    char buffer[128];

    /*****CMD+MQTTNETCFG *****/
    //memset(&buffer[0],0x00,strlen(buffer));
    sprintf (&buffer[0], "CMD+MQTTNETCFG=dev.rightech.io,1883\r\n");
    HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 10000);
    HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 10000);
    //memset(&buffer[0],0x00,strlen(buffer));
    //HAL_Delay(500);
    HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 10000);
    HAL_Delay(500);
    HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 10000);

    /*****CMD+MQTTCONCFG---->LED *****/
    //memset(&buffer[0],0x00,strlen(buffer));
    sprintf (&buffer[0], "CMD+MQTTCONCFG=3,mqtt-shruthi7352-okag7k,,,,,,,,,\r\n");
    HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
    HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
    //memset(&buffer[0],0x00,strlen(buffer));
    //HAL_Delay(500);
    HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
    HAL_Delay(500);
    HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);

```

```

/*****CMD+MQTTSTART *****/
//memset(&buffer[0],0x00,strlen(buffer));
sprintf (&buffer[0], "CMD+MQTTSTART=1\r\n");
HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
//
memset(&buffer[0],0x00,strlen(buffer));
HAL_Delay(5000);
HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
HAL_Delay(500);
HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);

/*****CMD+MQTTSUB *****/
//memset(&buffer[0],0x00,strlen(buffer));
sprintf (&buffer[0], "CMD+MQTTSUB=base/relay/led1\r\n");
HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
HAL_Delay(500);
HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
}

```

11.2 STAGE-4 CODE

```
/* USER CODE BEGIN Header */
/**
 * @file           : main.c
 * @brief          : Main program body
 *
 * @attention
 *
 * Copyright (c) 2023 STMicroelectronics.
 * All rights reserved.
 *
 * This software is licensed under terms that can be found in the LICENSE file
 * in the root directory of this software component.
 * If no LICENSE file comes with this software, it is provided AS-IS.
 */
/* USER CODE END Header */
/* Includes -----*/
#include "main.h"

/* Private includes -----*/
/* USER CODE BEGIN Includes */
#include <string.h>
#include <stdio.h>
```

```

/* USER CODE END Includes */

/* Private typedef -----
*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define -----
*/
/* USER CODE BEGIN PD */
#define AHT25_ADDR 0x70 // Address of the AHT25 sensor
// AHT25 commands
#define AHT25_INIT_CMD 0xE1
#define AHT25_MEASURE_CMD 0xAC
/* USER CODE END PD */

/* Private macro -----
*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables -----
*/
I2C_HandleTypeDef hi2c1;

UART_HandleTypeDef huart2;
UART_HandleTypeDef huart6;

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----
*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_I2C1_Init(void);
static void MX_USART6_UART_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code -----
*/
/* USER CODE BEGIN 0 */
float temperature, humidity;
char buffer[100];
//char buffer[100];
//float temperature, humidity;

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

    HAL_I2C_Master_Transmit(&hi2c1, AHT25_ADDR, &cmd, 1, HAL_MAX_DELAY);
    HAL_Delay(100);
    HAL_I2C_Master_Receive(&hi2c1, AHT25_ADDR, data, 6, HAL_MAX_DELAY);

```

```

    *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;
}
/* USER CODE END 0 */

/**
 * @brief The application entry point.
 * @retval int
 */
int main(void)
{
    /* USER CODE BEGIN 1 */

    /* USER CODE END 1 */

    /* MCU Configuration----- */

    /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
    HAL_Init();

    /* USER CODE BEGIN Init */

    /* USER CODE END Init */

    /* Configure the system clock */
    SystemClock_Config();

    /* USER CODE BEGIN SysInit */

    /* USER CODE END SysInit */

    /* Initialize all configured peripherals */
    MX_GPIO_Init();
    MX_USART2_UART_Init();
    MX_I2C1_Init();
    MX_USART6_UART_Init();
    /* USER CODE BEGIN 2 */

    /* USER CODE END 2 */

    /* Infinite loop */
    /* USER CODE BEGIN WHILE */
    while (1)
    {
        /* USER CODE END WHILE */

        /* USER CODE BEGIN 3 */
        read_sensor_values(&temperature, &humidity);

        sprintf(buffer, "Temperature=%.2f\r\n", temperature);
        HAL_UART_Transmit(&huart2, (uint8_t *)buffer,
strlen(buffer), HAL_MAX_DELAY);
        HAL_UART_Transmit(&huart6, (uint8_t *)buffer,
strlen(buffer), HAL_MAX_DELAY);
        HAL_Delay(1000);

        sprintf(buffer, "Humidity=%.2f\r\n", humidity);

```

```
        HAL_UART_Transmit(&huart2, (uint8_t *)buffer,
strlen(buffer), HAL_MAX_DELAY);
        HAL_UART_Transmit(&huart6, (uint8_t *)buffer,
strlen(buffer), HAL_MAX_DELAY);
        HAL_Delay(1000);
    }
/* USER CODE END 3 */
}
```

12.CONCLUSION

In conclusion, this project introduces a robust IoT-based temperature and humidity monitoring system leveraging STM32, AHT25, W10 Wi-Fi modules, and a Rugged Board. By seamlessly acquiring, transferring, and publishing real-time environmental data, the system enables efficient monitoring and analysis. The deployment of this system in real-world scenarios not only validates its performance but also highlights its practicality, cost-effectiveness, and scalability across diverse environments. The project's outcomes contribute valuable insights to the realm of IoT-based environmental monitoring systems, offering a promising tool for tracking and managing environmental conditions.
