

**KY-013 ANALOG TEMP SENSOR**

**Submitted**

**By**

**Pavithra Ravuru**

**November 20, 2023**

TABLE OF CONTENTS

[1 PROJECT SUMMARY 2](#_Toc151387480)

[2 KEY FEATURES 3](#_Toc151387481)

[3 HARDWARE SETUP AND SPECIFICATIONS 4](#_Toc151387482)

[3.1 STM32F446RE Microcontroller 4](#_Toc151387483)

[3.2 RuggedBoard 4](#_Toc151387484)

[3.3 HC-SR04 Sensor 6](#_Toc151387485)

[4 PROJECT METHODOLOGY 7](#_Toc151387486)

[4.1 Connection Diagram 8](#_Toc151387487)

[4.2 STM Code 9](#_Toc151387488)

[4.3 RuggedBoard Code 23](#_Toc151387489)

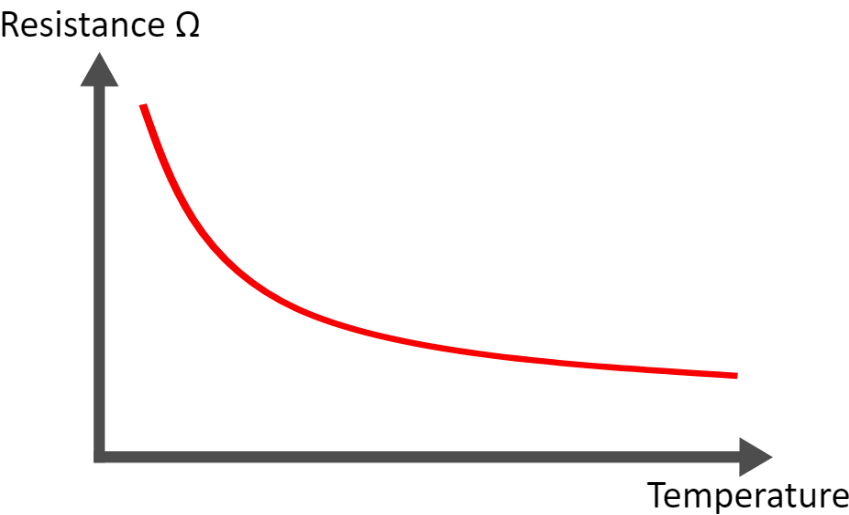
[5 PROJECT RESULTS 29](#_Toc151387490)

6 CONCLUSION................................................................................33

7 FUTURE SCOPE.............................................................................

# PROJECT SUMMARY

The KY-013 analog temperature sensor provides a voltage level that is directly proportional to the measured temperature. This module contains a Negative Temperature Coefficient (NTC) thermistor which can measure temperatures in the range of -55°C up to +125°C. The resistance value decreases at higher temperatures. This has a lower resistance value at higher temperatures. Based on the resulting resistance curve, the corresponding temperature can be calculated.



The change in resistance can be approximated mathematically, converted into a linear curve and the temperature coefficient (dependence of resistance change on temperature change) determined. Using these, the current temperature can then be calculated. The resistance can be determined with the aid of a voltage divider, in which a known voltage is divided over a known and an unknown (variable) resistance. Using this measured voltage, the resistance can then be calculated.

The basic pin structure of KY-013 Analog Temperature Sensor module is shown in below.

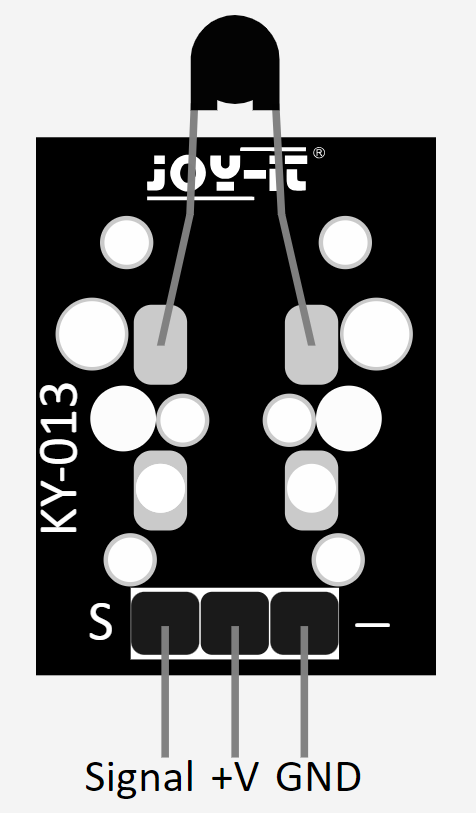


Figure 1: Pin Assignment of KY-013 Analog Temperature Sensor Module

This module typically includes a thermistor (temperature-sensitive resistor) and a resistor network on a small board. The module may come with a datasheet or documentation that provides information about the temperature-resistance relationship. This module produces an analog voltage signal that corresponds to the temperature. The output voltage can be read using an analog input on a microcontroller, Arduino, or other similar devices. Calibration may be required to convert the analog voltage readings into temperature values. This calibration is often specific to the characteristics of the particular thermistor used in the module. The module typically operates at a specific voltage, commonly 5V. The module usually has three pins: VCC (power), GND (ground), and OUT (analog output). Connect VCC to a 5V power source, GND to ground, and OUT to an analog input pin on your microcontroller or Arduino. The technical data of the module mentioned in below table.

Table 1: Technical Data

|  |  |
| --- | --- |
| Operating Voltage | 3.3-5V |
| Measuring Range | -55 °C to +125 °C |
| Measurement Accuracy | ± 0,5 °C |
| Known Resistance | 10 kΩ |
| Specific resistance of the NTC | 3950 Ω |

# KEY FEATURES

The key features associated with the KY-013 analog temperature sensor are:

* **Temperature Measurement:** The KY-013 is designed to measure temperature. It usually includes a thermistor as the temperature-sensing element.
* **Analog Output:** The sensor provides an analog voltage output that varies based on the temperature. The analog output allows for easy interfacing with analog-to-digital converters on microcontrollers.
* **Wide Operating Range:** The sensor typically has a wide operating temperature range, making it suitable for various applications.
* **Compatibility with Microcontrollers:** The KY-013 module is commonly used with microcontrollers such as Arduino. It can be easily connected to analog input pins for temperature readings.
* **Compact Design:** The KY-013 is often compact and comes in a module form, making it easy to integrate into electronic projects.
* **Simple Interface:** It usually has a straightforward interface, requiring minimal external components for basic operation.
* **Accuracy and Precision:** The accuracy and precision of temperature measurements can vary based on the specific sensor and its calibration.
* **Calibration Potentiometer:** Some versions of the KY-013 may include a potentiometer for calibration, allowing users to adjust the sensor's sensitivity to better suit their requirements.
* **Low Power Consumption:** The sensor typically has low power requirements, making it suitable for battery-powered applications.

# HARDWARE SETUP AND SPECIFICATIONS

## STM32F446RE Microcontroller

The STM32F446RE 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, and 13 communication interfaces etc.

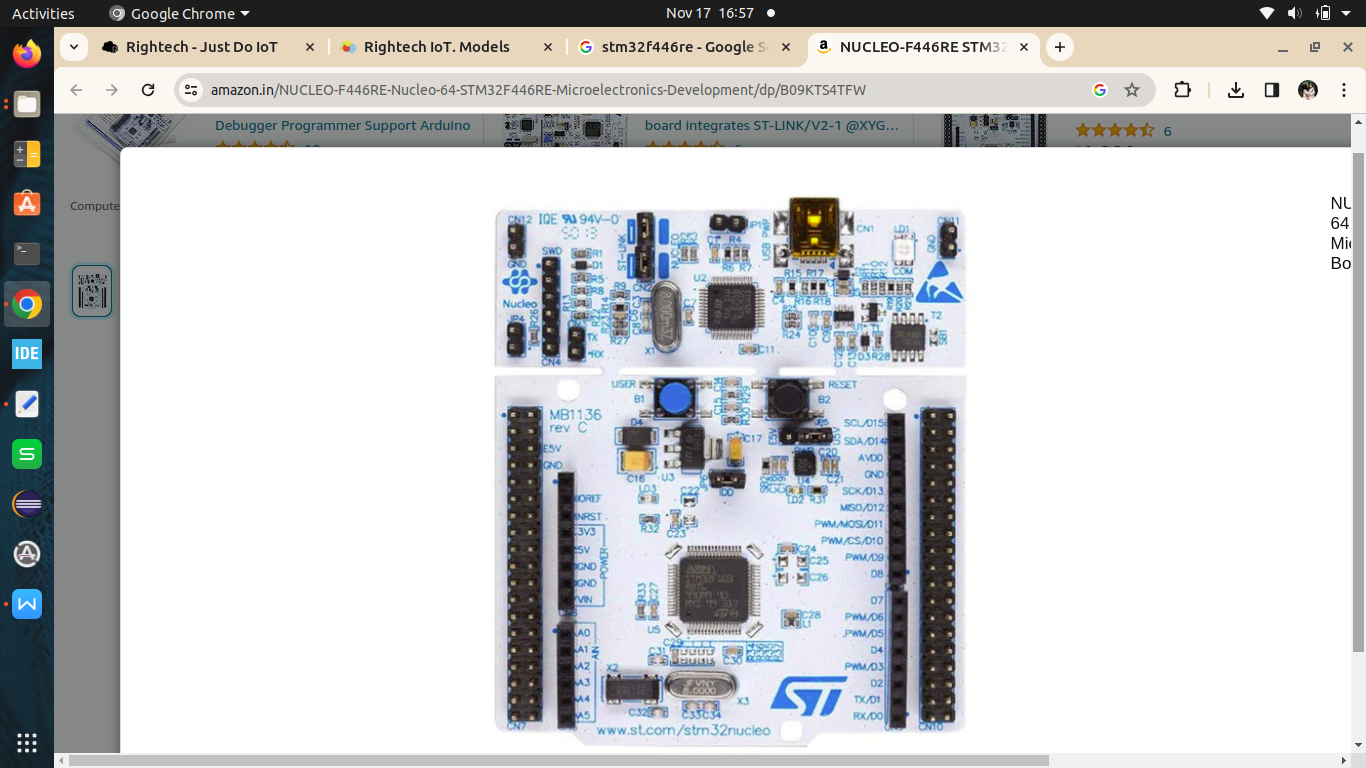


Figure 2: STM32F446RE Microcontroller

## 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.

## RuggedBoard

RuggedBoard - A5D2x is a Single Board Computer providing as easy migration path from Microcontroller to Microprocessor. RuggedBoard 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.

The pin diagram of RuggedBoard - A5D2x shown in below.

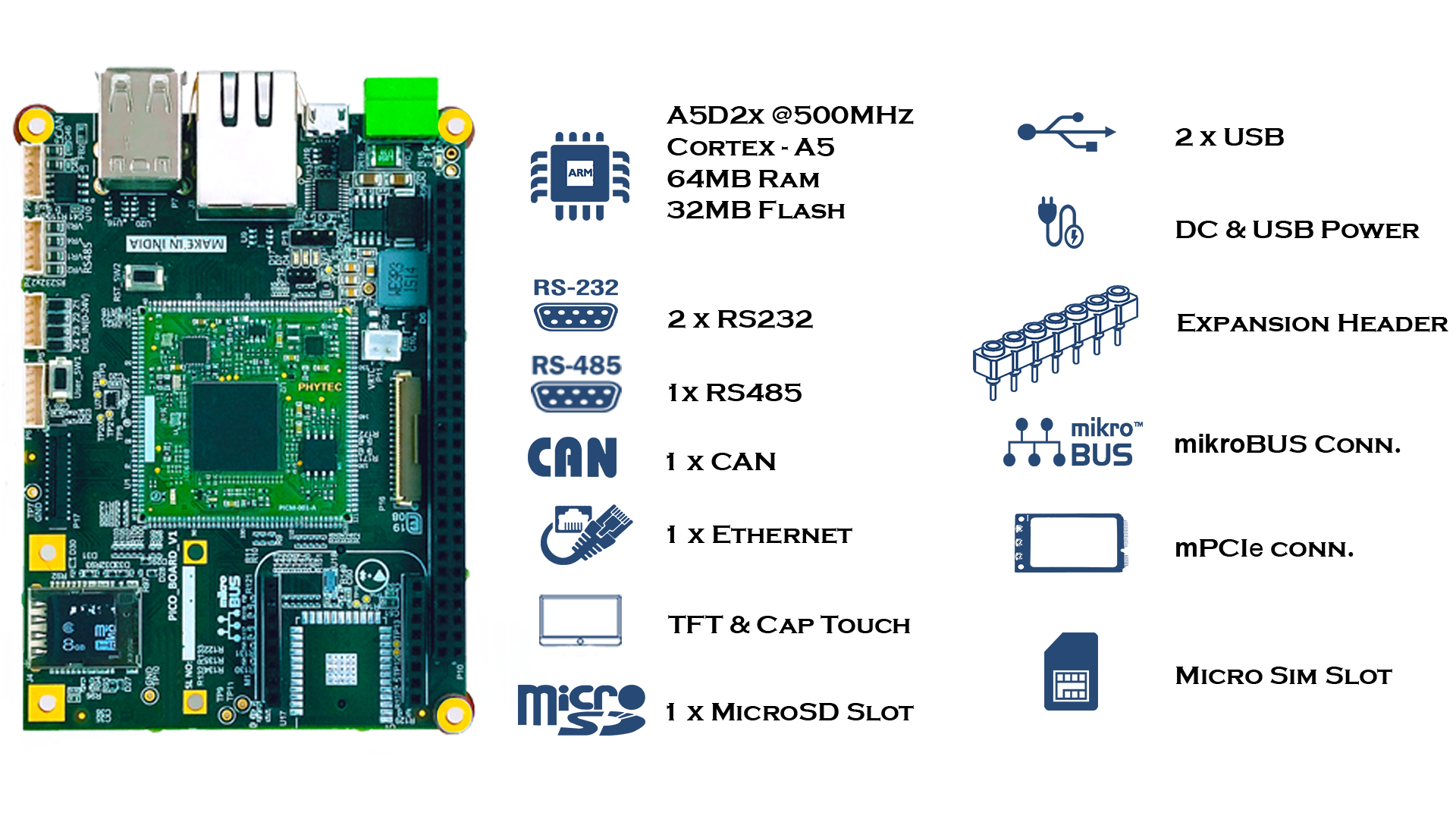


Figure 3: Pin Diagram of RuggedBoard - A5D2x

**RuggedBoard - A5D2x Specifications:**

**1. System On Module**

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

**2. 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

**3. 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)

**4. 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

**5. Power**

* Input Power: DC +5V or Micro USB Supply
* Temperature Range: - 40°to + 85°C

**6. Optional Accessories**

* Accessories Set Micro USB Cable, Ethernet Cable, Power Adapter 5V/3A

# PROJECT METHODOLOGY

The following stages were included in the project.

* **Stage 1:** The STM32F446RE with KY-013 analog temp sensor is used to measure the temperature.
* **Stage 2:** The STM32F446RE with KY-013, W10 is used to transmit the data into the MQTT server to Publish the data in the Right-tech.
* **Stage 3:** The RuggedBoard with STM32F446RE is used to transmit the data and get the temperature value in the RuggedBoard minicom.
* **Stage 4:** The Ruggedboard with STM32F446RE is used to transmit the data and get the temperature value in the RuggedBoard minicom and pass the value into the Right-tech by using W10 module is connected with the Rugged board.

## Connection Diagrams

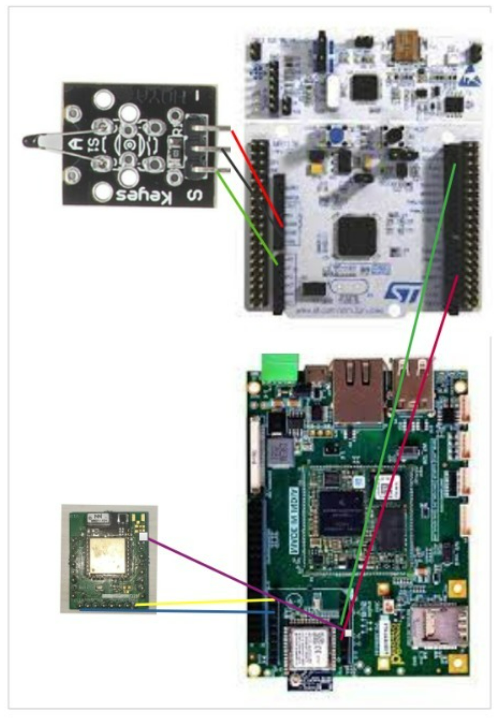


Figure 4: Connection Diagram for stage 4

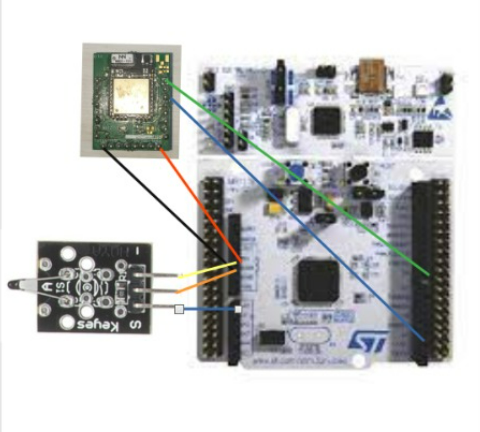


Figure 5: Connection Diagram for stage 2

## Codes

**Stage 1:**

while (1)

{

/\* USER CODE END WHILE \*/

/\* USER CODE BEGIN 3 \*/

// Start ADC conversion

HAL\_ADC\_Start(&hadc1);

if (HAL\_ADC\_PollForConversion(&hadc1,100) == HAL\_OK) {

Vo = HAL\_ADC\_GetValue(&hadc1);

}

HAL\_ADC\_Stop(&hadc1);

// Calculate temperature

R2 = R1 \* (1023.0 / (float)Vo - 1.0);

logR2 = log(R2);

K = 1.0 / (c1 + c2 \* logR2 + c3 \* logR2 \* logR2 \* logR2);

/ / Temperature in Kelvin

T = K - 273.15; // Convert Kelvin to Celsius

// Print temperature

printf("the current temperature in celsius:%.2C\r\n",T);

// You can use UART or other communication methods supported by STM32

char buffer[50];

sprintf(buffer, "Current temperature is: %.2f C\r\n", T);

HAL\_UART\_Transmit(&huart2, (uint8\_t\*)buffer, strlen(buffer), HAL\_MAX\_DELAY);

HAL\_UART\_Transmit(&huart1, (uint8\_t\*)buffer, strlen(buffer), HAL\_MAX\_DELAY);

// to send the temperature reading.

mqtt\_data\_send();

HAL\_Delay(1000); // Delay using HAL\_Delay

}

**Stage 2:**

while (1)

{

/\* USER CODE END WHILE \*/

/\* USER CODE BEGIN 3 \*/

// Start ADC conversion

HAL\_ADC\_Start(&hadc1);

if (HAL\_ADC\_PollForConversion(&hadc1,100) == HAL\_OK) {

Vo = HAL\_ADC\_GetValue(&hadc1);

}

HAL\_ADC\_Stop(&hadc1);

// Calculate temperature

R2 = R1 \* (1023.0 / (float)Vo - 1.0);

logR2 = log(R2);

K = 1.0 / (c1 + c2 \* logR2 + c3 \* logR2 \* logR2 \* logR2); // Temperature in Kelvin

T = K - 273.15; // Convert Kelvin to Celsius

// Print temperature

printf("the current temperature in celsius:%.2C\r\n",T);

// You can use UART or other communication methods supported by STM32

char buffer[50];

sprintf(buffer, "Current temperature is: %.2f C\r\n", T);

HAL\_UART\_Transmit(&huart2, (uint8\_t\*)buffer, strlen(buffer), HAL\_MAX\_DELAY);

HAL\_UART\_Transmit(&huart1, (uint8\_t\*)buffer, strlen(buffer), HAL\_MAX\_DELAY);

// to send the temperature reading.

mqtt\_data\_send();

HAL\_Delay(1000); // Delay using HAL\_Delay

}

**Stage 3:**

while (1)

{

/\* USER CODE END WHILE \*/

/\* USER CODE BEGIN 3 \*/

// Start ADC conversion

HAL\_ADC\_Start(&hadc1);

if (HAL\_ADC\_PollForConversion(&hadc1,100) == HAL\_OK) {

Vo = HAL\_ADC\_GetValue(&hadc1);

}

HAL\_ADC\_Stop(&hadc1);

// Calculate temperature

R2 = R1 \* (1023.0 / (float)Vo - 1.0);

logR2 = log(R2);

K = 1.0 / (c1 + c2 \* logR2 + c3 \* logR2 \* logR2 \* logR2); // Temperature in Kelvin

T = K - 273.15; // Convert Kelvin to Celsius

// Print temperature

printf("the current temperature in celsius:%.2C\r\n",T);

// You can use UART or other communication methods supported by STM32

char buffer[50];

sprintf(buffer, "Current temperature is: %.2f C\r\n", T);

HAL\_UART\_Transmit(&huart2, (uint8\_t\*)buffer, strlen(buffer), HAL\_MAX\_DELAY);

HAL\_UART\_Transmit(&huart1, (uint8\_t\*)buffer, strlen(buffer), HAL\_MAX\_DELAY);

// to send the temperature reading

HAL\_Delay(1000);

// Delay using HAL\_Delay

}

**Stage 4:**

int main()

{

char \*portname = "/dev/ttyS3";

int fd;

int wlen;

int rdlen;

int ret;

char res[5];

char arr1[] = "CMD+RESET\r\n";

char arr2[] = "CMD+WIFIMODE=1\r\n";

char arr[] = "CMD+CONTOAP=\"Pavi\",\"pavi0205\"\r\n";

char arr3[] = "CMD+MQTTNETCFG=dev.rightech.io,1883\r\n";

char arr4[] = "CMD+MQTTCONCFG=3,mqtt-ravurupavithra02-quhcys

,,,,,,,,,\r\n";

char arr5[] = "CMD+MQTTSTART=1\r\n";

char arr6[] = "CMD+MQTTSUB=base/relay/led1\r\n";

unsigned char buf[100];

fd = open(portname, O\_RDWR | O\_NOCTTY | O\_SYNC);

if (fd < 0)

{

printf("Error opening %s: %s\n", portname, strerror(errno));

return -1;

}

set\_interface\_attribs(fd, B38400);

printf("%s", arr1);

wlen = write(fd, arr1, sizeof(arr1) - 1);

sleep(3);

printf("%s", arr2);

wlen = write(fd, arr2, sizeof(arr2) - 1);

sleep(3);

printf("%s", arr);

wlen = write(fd, arr, sizeof(arr) - 1);

sleep(3);

printf("%s", arr3);

wlen = write(fd, arr3, sizeof(arr3) - 1);

sleep(3);

printf("%s", arr4);

wlen = write(fd, arr4, sizeof(arr4) - 1);

sleep(3);

printf("%s", arr5);

wlen = write(fd, arr5, sizeof(arr5) - 1);

sleep(3);

printf("%s", arr6);

wlen = write(fd, arr6, sizeof(arr6) - 1);

sleep(3);

char buffer[100]; // Create a buffer to hold the formatted message

while(1){

rdlen = read(fd, buf, sizeof(buf) - 1);

if (rdlen > 0) {

buf[rdlen] = '\0'; // Null-terminate the received data

printf("%s\n", buf);

int ret = snprintf(buffer, sizeof(buffer), "CMD+MQTTPUB=sensor/voltage,%s\r\n", buf);

if (ret < 0) {

} else {

ssize\_t wlen = write(fd, buffer, ret);

sleep(3);

if (wlen == -1) {

}

}

}

}

close(fd);

return 0;}

**Code Snippet to WiFi Module:**



**Code Snippet to MQTT Initialization:**

****

**Main 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"math.h"

#include<stdio.h>

#include <string.h>

int Vo;

float R1 = 10000.0; // Value of R1 on board

float logR2, R2, T ,K;

float c1 = 0.001129148, c2 = 0.000234125, c3 = 0.0000000876741; // Steinhart-Hart coefficients for thermistor

/\* 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 ---------------------------------------------------------\*/

ADC\_HandleTypeDef hadc1;

DMA\_HandleTypeDef hdma\_adc1;

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\_DMA\_Init(void);

static void MX\_USART2\_UART\_Init(void);

static void MX\_ADC1\_Init(void);

static void MX\_USART1\_UART\_Init(void);

void WE10\_Init (void);

void MQTT\_Init(void);

void mqtt\_data\_send(void);

/\* 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

\*/

void mqtt\_data\_send()

{

char buffer[50];

sprintf (&buffer[0], "CMD+MQTTPUB=base/state/temperature,%.2f\r\n",T);

HAL\_UART\_Transmit(&huart1, (uint8\_t\*)buffer, strlen(buffer), 1000);

HAL\_Delay(1000);

}

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\_DMA\_Init();

MX\_USART2\_UART\_Init();

MX\_ADC1\_Init();

MX\_USART1\_UART\_Init();

WE10\_Init ();

MQTT\_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 \*/

// Start ADC conversion

HAL\_ADC\_Start(&hadc1);

if (HAL\_ADC\_PollForConversion(&hadc1,100) == HAL\_OK) {

Vo = HAL\_ADC\_GetValue(&hadc1);

}

HAL\_ADC\_Stop(&hadc1);

// Calculate temperature

R2 = R1 \* (1023.0 / (float)Vo - 1.0);

logR2 = log(R2);

K = 1.0 / (c1 + c2 \* logR2 + c3 \* logR2 \* logR2 \* logR2); // Temperature in Kelvin

T = K - 273.15; // Convert Kelvin to Celsius

// Print temperature

printf("the current temperature in celsius:%.2C\r\n",T);

// You can use UART or other communication methods supported by STM32

char buffer[50];

sprintf(buffer, "Current temperature is: %.2f C\r\n", T);

HAL\_UART\_Transmit(&huart2, (uint8\_t\*)buffer, strlen(buffer), HAL\_MAX\_DELAY);

HAL\_UART\_Transmit(&huart1, (uint8\_t\*)buffer, strlen(buffer), HAL\_MAX\_DELAY);

// to send the temperature reading.

mqtt\_data\_send();

HAL\_Delay(1000); // Delay using HAL\_Delay

}

/\* USER CODE END 3 \*/

}

void WE10\_Init ()

{

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\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(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-ravurupavithra02-quhcys,,,,,,,,,\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);

}

/\*\*

\* @brief System Clock Configuration

\* @retval None

\*/

void SystemClock\_Config(void)

{

RCC\_OscInitTypeDef RCC\_OscInitStruct = {0};

RCC\_ClkInitTypeDef RCC\_ClkInitStruct = {0};

/\*\* Configure the main internal regulator output voltage

\*/

\_\_HAL\_RCC\_PWR\_CLK\_ENABLE();

\_\_HAL\_PWR\_VOLTAGESCALING\_CONFIG(PWR\_REGULATOR\_VOLTAGE\_SCALE3);

/\*\* Initializes the RCC Oscillators according to the specified parameters

\* in the RCC\_OscInitTypeDef structure.

\*/

RCC\_OscInitStruct.OscillatorType = RCC\_OSCILLATORTYPE\_HSI;

RCC\_OscInitStruct.HSIState = RCC\_HSI\_ON;

RCC\_OscInitStruct.HSICalibrationValue = RCC\_HSICALIBRATION\_DEFAULT;

RCC\_OscInitStruct.PLL.PLLState = RCC\_PLL\_ON;

RCC\_OscInitStruct.PLL.PLLSource = RCC\_PLLSOURCE\_HSI;

RCC\_OscInitStruct.PLL.PLLM = 16;

RCC\_OscInitStruct.PLL.PLLN = 336;

RCC\_OscInitStruct.PLL.PLLP = RCC\_PLLP\_DIV4;

RCC\_OscInitStruct.PLL.PLLQ = 2;

RCC\_OscInitStruct.PLL.PLLR = 2;

if (HAL\_RCC\_OscConfig(&RCC\_OscInitStruct) != HAL\_OK)

{

Error\_Handler();

}

/\*\* Initializes the CPU, AHB and APB buses clocks

\*/

RCC\_ClkInitStruct.ClockType = RCC\_CLOCKTYPE\_HCLK|RCC\_CLOCKTYPE\_SYSCLK

|RCC\_CLOCKTYPE\_PCLK1|RCC\_CLOCKTYPE\_PCLK2;

RCC\_ClkInitStruct.SYSCLKSource = RCC\_SYSCLKSOURCE\_PLLCLK;

RCC\_ClkInitStruct.AHBCLKDivider = RCC\_SYSCLK\_DIV1;

RCC\_ClkInitStruct.APB1CLKDivider = RCC\_HCLK\_DIV2;

RCC\_ClkInitStruct.APB2CLKDivider = RCC\_HCLK\_DIV1;

if (HAL\_RCC\_ClockConfig(&RCC\_ClkInitStruct, FLASH\_LATENCY\_2) != HAL\_OK)

{

Error\_Handler();

}

}

/\*\*

\* @brief ADC1 Initialization Function

\* @param None

\* @retval None

\*/

static void MX\_ADC1\_Init(void)

{

/\* USER CODE BEGIN ADC1\_Init 0 \*/

/\* USER CODE END ADC1\_Init 0 \*/

ADC\_AnalogWDGConfTypeDef AnalogWDGConfig = {0};

ADC\_ChannelConfTypeDef sConfig = {0};

/\* USER CODE BEGIN ADC1\_Init 1 \*/

/\* USER CODE END ADC1\_Init 1 \*/

/\*\* Configure the global features of the ADC (Clock, Resolution, Data Alignment and number of conversion)

\*/

hadc1.Instance = ADC1;

hadc1.Init.ClockPrescaler = ADC\_CLOCK\_SYNC\_PCLK\_DIV4;

hadc1.Init.Resolution = ADC\_RESOLUTION\_12B;

hadc1.Init.ScanConvMode = ENABLE;

hadc1.Init.ContinuousConvMode = ENABLE;

hadc1.Init.DiscontinuousConvMode = DISABLE;

hadc1.Init.ExternalTrigConvEdge = ADC\_EXTERNALTRIGCONVEDGE\_NONE;

hadc1.Init.ExternalTrigConv = ADC\_SOFTWARE\_START;

hadc1.Init.DataAlign = ADC\_DATAALIGN\_RIGHT;

hadc1.Init.NbrOfConversion = 1;

hadc1.Init.DMAContinuousRequests = ENABLE;

hadc1.Init.EOCSelection = ADC\_EOC\_SINGLE\_CONV;

if (HAL\_ADC\_Init(&hadc1) != HAL\_OK)

{

Error\_Handler();

}

/\*\* Configure the analog watchdog

\*/

AnalogWDGConfig.WatchdogMode = ADC\_ANALOGWATCHDOG\_SINGLE\_REG;

AnalogWDGConfig.HighThreshold = 0;

AnalogWDGConfig.LowThreshold = 0;

AnalogWDGConfig.Channel = ADC\_CHANNEL\_0;

AnalogWDGConfig.ITMode = DISABLE;

if (HAL\_ADC\_AnalogWDGConfig(&hadc1, &AnalogWDGConfig) != HAL\_OK)

{

Error\_Handler();

}

/\*\* Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.

\*/

sConfig.Channel = ADC\_CHANNEL\_0;

sConfig.Rank = 1;

sConfig.SamplingTime = ADC\_SAMPLETIME\_3CYCLES;

if (HAL\_ADC\_ConfigChannel(&hadc1, &sConfig) != HAL\_OK)

{

Error\_Handler();

}

/\* USER CODE BEGIN ADC1\_Init 2 \*/

/\* USER CODE END ADC1\_Init 2 \*/

}

/\*\*

\* @brief USART1 Initialization Function

\* @param None

\* @retval None

\*/

static void MX\_USART1\_UART\_Init(void)

{

/\* USER CODE BEGIN USART1\_Init 0 \*/

/\* USER CODE END USART1\_Init 0 \*/

/\* USER CODE BEGIN USART1\_Init 1 \*/

/\* USER CODE END USART1\_Init 1 \*/

huart1.Instance = USART1;

huart1.Init.BaudRate = 38400;

huart1.Init.WordLength = UART\_WORDLENGTH\_8B;

huart1.Init.StopBits = UART\_STOPBITS\_1;

huart1.Init.Parity = UART\_PARITY\_NONE;

huart1.Init.Mod7e = UART\_MODE\_TX\_RX;

huart1.Init.HwFlowCtl = UART\_HWCONTROL\_RTS\_CTS;

huart1.Init.OverSampling = UART\_OVERSAMPLING\_16;

if (HAL\_UART\_Init(&huart1) != HAL\_OK)

{

Error\_Handler();

}

/\* USER CODE BEGIN USART1\_Init 2 \*/

/\* USER CODE END USART1\_Init 2 \*/

}

/\*\*

\* @brief USART2 Initialization Function

\* @param None

\* @retval None

\*/

static void MX\_USART2\_UART\_Init(void)

{

/\* USER CODE BEGIN USART2\_Init 0 \*/

/\* USER CODE END USART2\_Init 0 \*/

/\* USER CODE BEGIN USART2\_Init 1 \*/

/\* USER CODE END USART2\_Init 1 \*/

huart2.Instance = USART2;

huart2.Init.BaudRate = 9600;

huart2.Init.WordLength = UART\_WORDLENGTH\_8B;

huart2.Init.StopBits = UART\_STOPBITS\_1;

huart2.Init.Parity = UART\_PARITY\_NONE;

huart2.Init.Mode = UART\_MODE\_TX\_RX;

huart2.Init.HwFlowCtl = UART\_HWCONTROL\_NONE;

huart2.Init.OverSampling = UART\_OVERSAMPLING\_16;

if (HAL\_UART\_Init(&huart2) != HAL\_OK)

{

Error\_Handler();

}

/\* USER CODE BEGIN USART2\_Init 2 \*/

/\* USER CODE END USART2\_Init 2 \*/

}

/\*\*

\* Enable DMA controller clock

\*/

static void MX\_DMA\_Init(void)

{

/\* DMA controller clock enable \*/

\_\_HAL\_RCC\_DMA2\_CLK\_ENABLE();

/\* DMA interrupt init \*/

/\* DMA2\_Stream0\_IRQn interrupt configuration \*/

HAL\_NVIC\_SetPriority(DMA2\_Stream0\_IRQn, 0, 0);

HAL\_NVIC\_EnableIRQ(DMA2\_Stream0\_IRQn);

}

/\*\*

\* @brief GPIO Initialization Function

\* @param None

\* @retval None

\*/

static void MX\_GPIO\_Init(void)

{

GPIO\_InitTypeDef GPIO\_InitStruct = {0};

/\* USER CODE BEGIN MX\_GPIO\_Init\_1 \*/

/\* USER CODE END MX\_GPIO\_Init\_1 \*/

/\* GPIO Ports Clock Enable \*/

\_\_HAL\_RCC\_GPIOC\_CLK\_ENABLE();

\_\_HAL\_RCC\_GPIOH\_CLK\_ENABLE();

\_\_HAL\_RCC\_GPIOA\_CLK\_ENABLE();

\_\_HAL\_RCC\_GPIOB\_CLK\_ENABLE();

/\*Configure GPIO pin Output Level \*/

HAL\_GPIO\_WritePin(LD2\_GPIO\_Port, LD2\_Pin, GPIO\_PIN\_RESET);

/\*Configure GPIO pin : B1\_Pin \*/

GPIO\_InitStruct.Pin = B1\_Pin;

GPIO\_InitStruct.Mode = GPIO\_MODE\_IT\_FALLING;

GPIO\_InitStruct.Pull = GPIO\_NOPULL;

HAL\_GPIO\_Init(B1\_GPIO\_Port, &GPIO\_InitStruct);

/\*Configure GPIO pin : LD2\_Pin \*/

GPIO\_InitStruct.Pin = LD2\_Pin;

GPIO\_InitStruct.Mode = GPIO\_MODE\_OUTPUT\_PP;

GPIO\_InitStruct.Pull = GPIO\_NOPULL;

GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_LOW;

HAL\_GPIO\_Init(LD2\_GPIO\_Port, &GPIO\_InitStruct);

/\* USER CODE BEGIN MX\_GPIO\_Init\_2 \*/

GPIO\_InitStruct.Pin = GPIO\_PIN\_1; // PA1

GPIO\_InitStruct.Mode = GPIO\_MODE\_ANALOG; // Analog mode

HAL\_GPIO\_Init(GPIOA, &GPIO\_InitStruct);

/\* USER CODE END MX\_GPIO\_Init\_2 \*/

}

/\* USER CODE BEGIN 4 \*/

int \_write(int file, char \*ptr, int len)

{

(void)file;

int DataIdx;

for (DataIdx = 0; DataIdx < len; DataIdx++)

{

ITM\_SendChar(\*ptr++);

}

return len;

}

/\* USER CODE END 4 \*/

/\*\*

\* @brief This function is executed in case of error occurrence.

\* @retval None

\*/

void Error\_Handler(void)

{

/\* USER CODE BEGIN Error\_Handler\_Debug \*/

/\* User can add his own implementation to report the HAL error return state \*/

\_\_disable\_irq();

while (1)

{

}

/\* USER CODE END Error\_Handler\_Debug \*/

}

#ifdef USE\_FULL\_ASSERT

/\*\*

\*

\* @brief Reports the name of the source file and the source line number

\* where the assert\_param error has occurred.

\* @param file: pointer to the source file name

\* @param line: assert\_param error line source number

\* @retval None

\*/

void assert\_failed(uint8\_t \*file, uint32\_t line)

{

/\* USER CODE BEGIN 6 \*/

/\* User can add his own implementation to report the file name and line number,

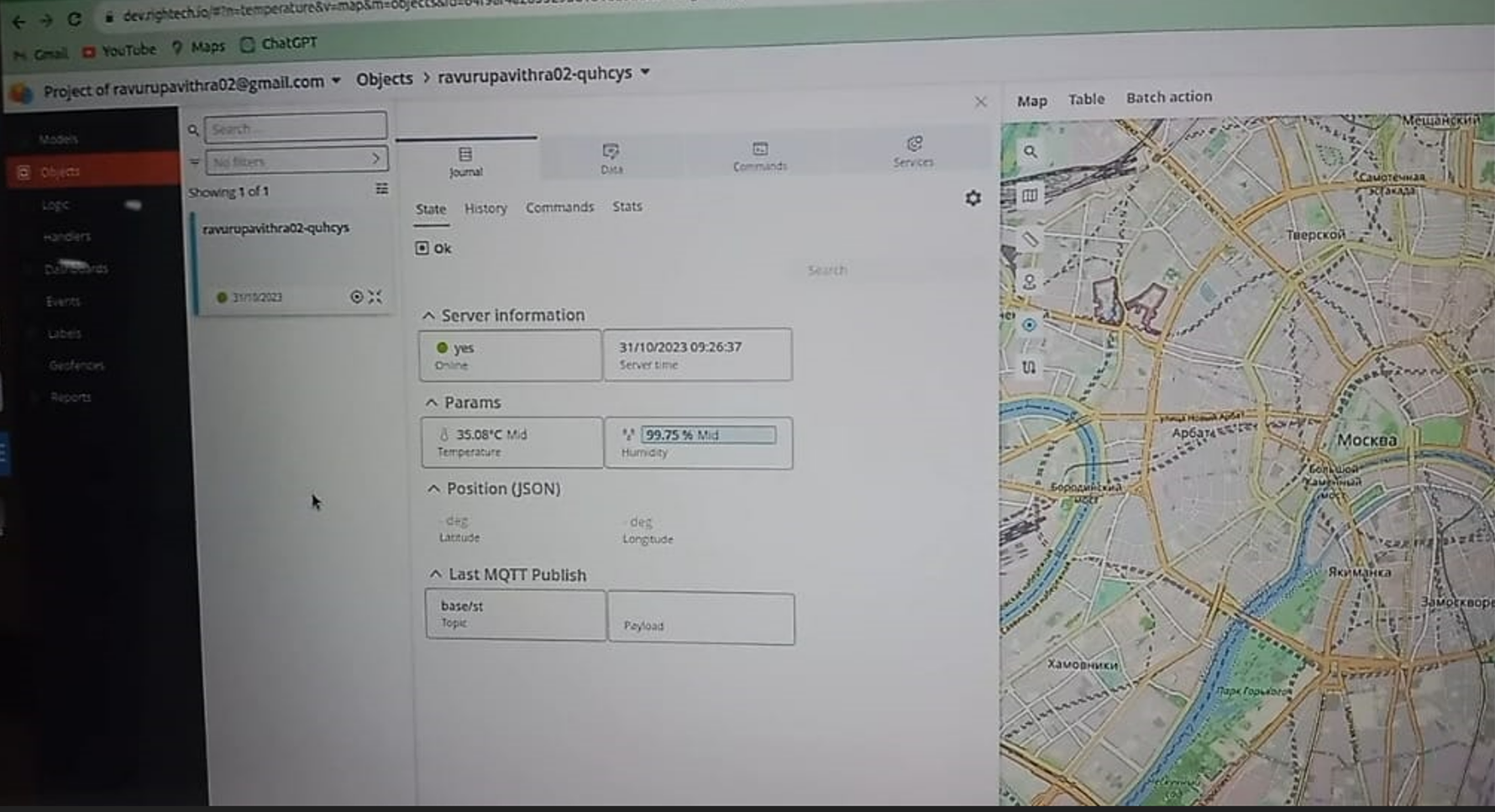
ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) \*/

/\* USER CODE END 6 \*/

}

#endif /\* USE\_FULL\_ASSERT \*/

# PROJECT RESULTS



# CONCLUSION

This project concludes a how to measure room temperature by using KY-103 Analog Temp Sensor. To do this a code based design approach was used. Firstly, STM32F446RE microcontroller connected to KY-013 senor, and W10 and transmit the data into the MQTT server to Publish the data in the Right-tech.The temperature values are shown in minicom by using KY-013 analog temp sensor and by using WE10 module we can transfer the temperature values to right tech cloud.

# FUTURE SCOPE

In the future works we can utilise, KY-013 Analog Temp Sensor to monitor the temperature in various outdoor environments such as agriculture or meteorology to track changes in the temperature. This will be used in thermostats to control the temperature of a environment, turning on heating or cooling systems as needed. The sensor will also used to monitor the temperature of electronic components and devices, shut them down or activate a cooling system if the temperature exceeds a safe limit.