**IoT-based Distance Measurement System**

**PRESENTED BY**

**HARISH KUMAR A M.E(VLSI Design)**

**NOVEMBER-2023**

**TABLE OF CONTENT**

1.PROJECT SUMMARY

2.KEY FEATURES OF THE PROJECT

3.HARDWARE USED AND THERE SPECIFICATION

4.Why interrupt based uart communication

5.connection diagram

6.HC-SR04

7.RUGGED BOARD

8.W10 WiFi

9.code snippet to wifi module initialization and connection with interrupt-based uart commands

10.code snippet to MQTT initialization and connection with interrupt-based uart commands

11. Send\_Task Function

12.PROJECT CODE

13.RIGHT TECH OUTPUT

**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 distance measuring between the object. By deploying IoT devices in various measuring, we can collect real-time data by measuring distance. This data can then be used to track distance between the object over time, identify potential problems, and take corrective action.

This project proposes the development of an IoT-based distance monitoring system using STM32, HC-SR04, and W10 WiFi modules. The system will use an STM32 microcontroller to acquire sensor data from an HC-SR04 distance 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.

Also, the stm32f446re with HC-SR04 data is transmitted into ruggedboard and connect the two boards.When the W10 module is connected with the ruggedboard it will send the data into the MQTT server from the ruggedboard.

The proposed system has the potential to be a valuable tool for tracking and monitoring distance between the objects. 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 distance measuring systems. The project will also provide a valuable case study for other researchers and developers who are interested in developing IoT-based ecological distance measuring systems.

**2.key features of the project**

Sensor Initialization with Timers.

UART Communication is based on Interrupt.

Common function for UART receive data parsing.

Timer based delay.

STM32 Application is Master and configured with W10 (Wi-Fi) module accordingly with AT commands.

Data is pushed to the MQTT server.

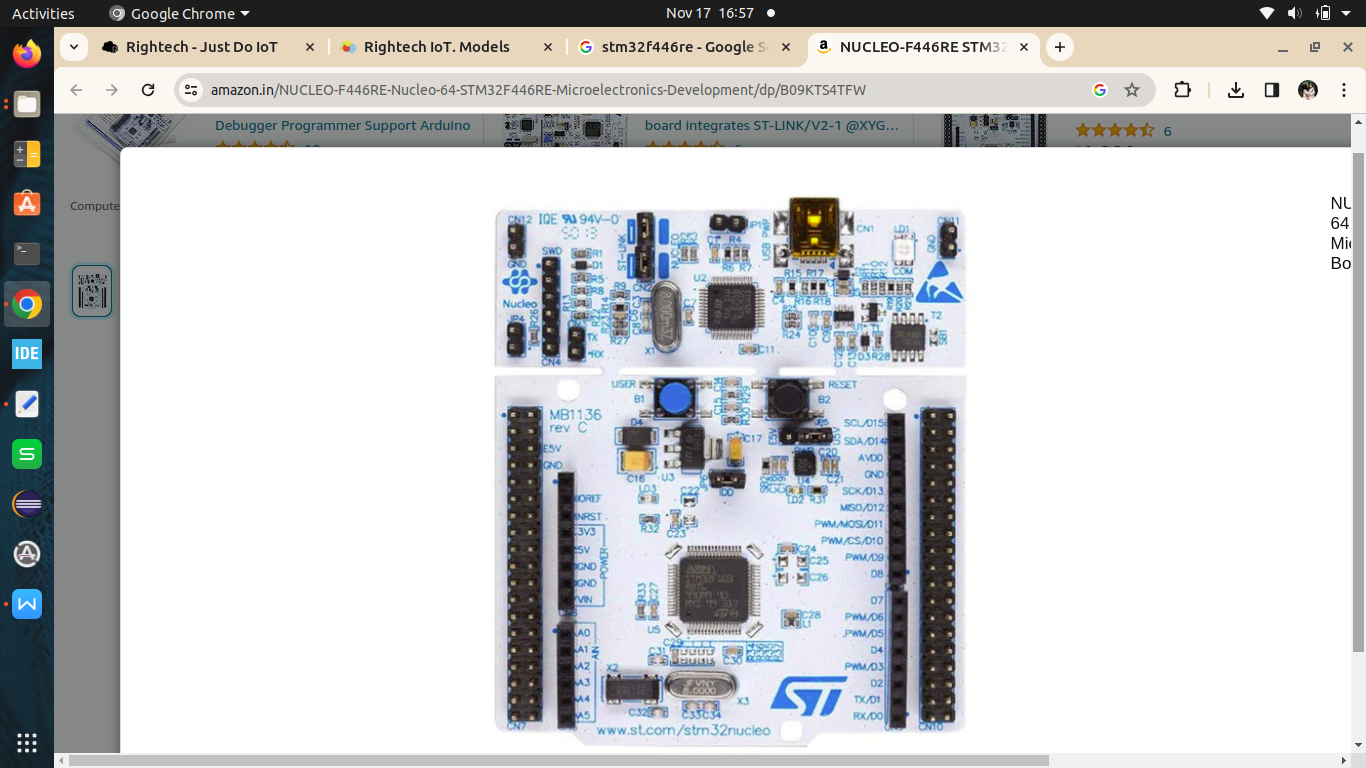
When the data is transmitted into ruggedboard from stm332f446re communicate between the two board and tranmit the data.

Distance data is pushed in every time when object is interrupt.

**3.Hardware Used and there Specification**

**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

13 communication interfaces USB OTG RTC

**RUGGEDBOARD**

RuggedBoard - A5D2x is an 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.

**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

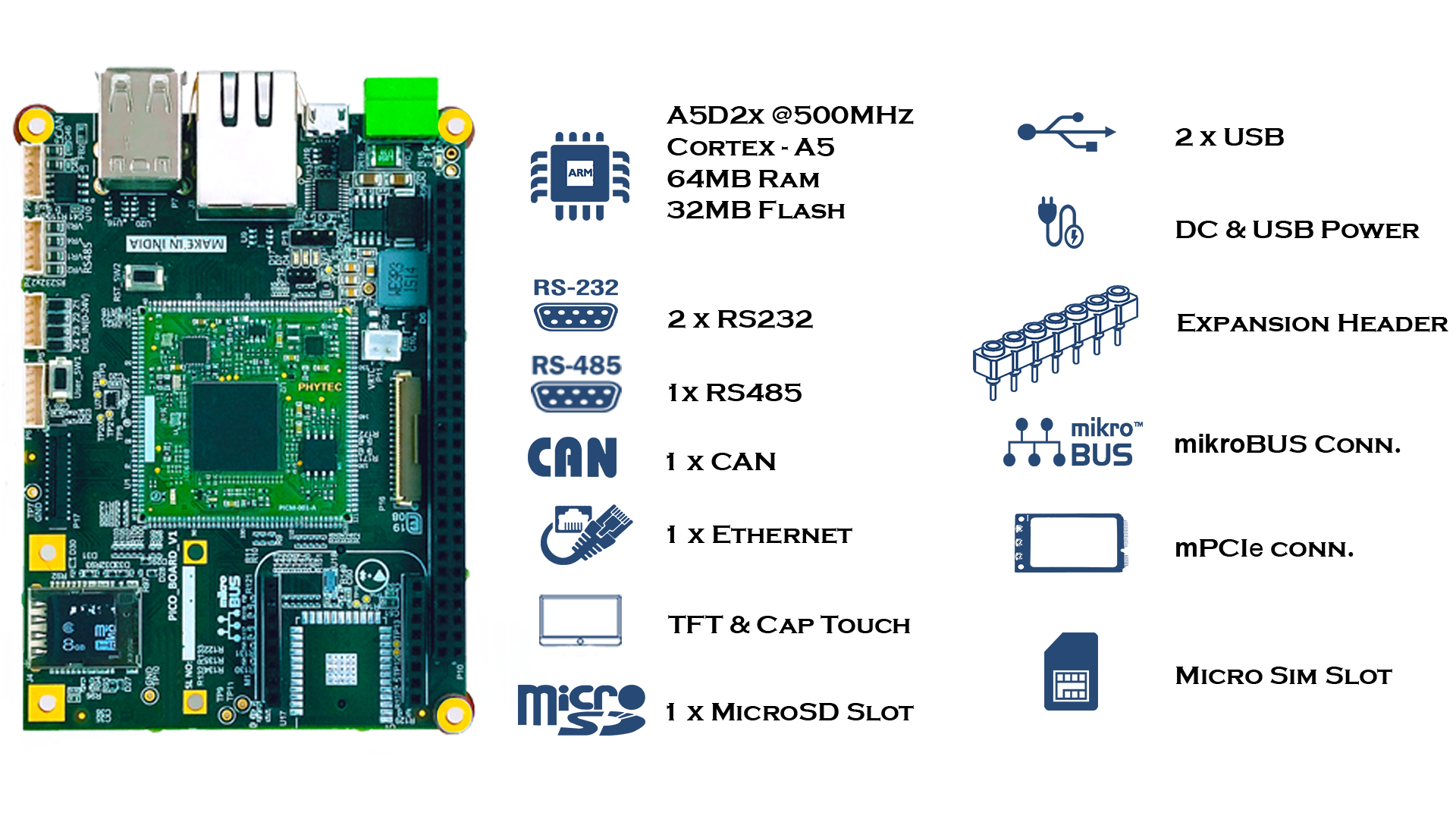
Power

Input Power DC +5V or Micro USB Supply

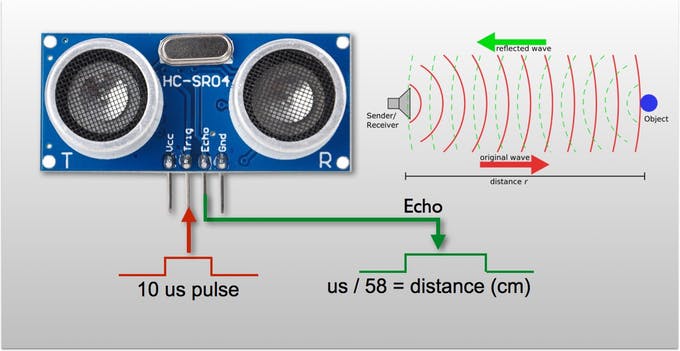
Temperature Range - 40°to + 85°C

Optional Accessories

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



**HC-SR04 sensor**



Power Supply :+5V DC

Quiescent Current : <2mA

Working Current: 15mA

Effectual Angle: <15°

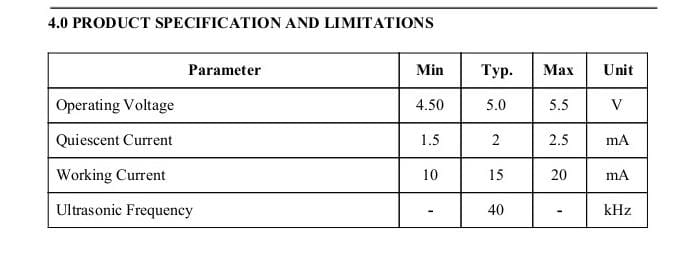
Ranging Distance : 2cm – 400 cm/1" -13ft

Resolution : 0.3 cm

Measuring Angle: 30 degree

Trigger Input Pulse width: 10uS

Dimension: 45mm x 20mm x 15mm



**OPERATION**:

The timing diagram of HC-SR04 is shown. To start measurement, Trig of SR04 must receive a

pulse of high (5V) for at least 10us, this will initiate the sensor will transmit out 8 cycle of ultrasonic

burst at 40kHz and wait for the reflected ultrasonic burst. When the sensor detected ultrasonic from

receiver, it will set the Echo pin to high (5V) and delay for a period (width) which proportion to

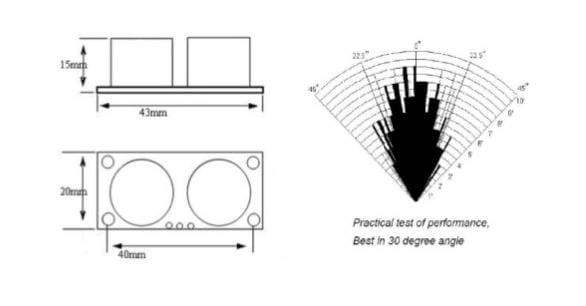
distance. To obtain the distance, measure the width (Ton) of Echo pin.

Time = Width of Echo pulse, in uS (micro second)

● Distance in centimeters = Time / 58

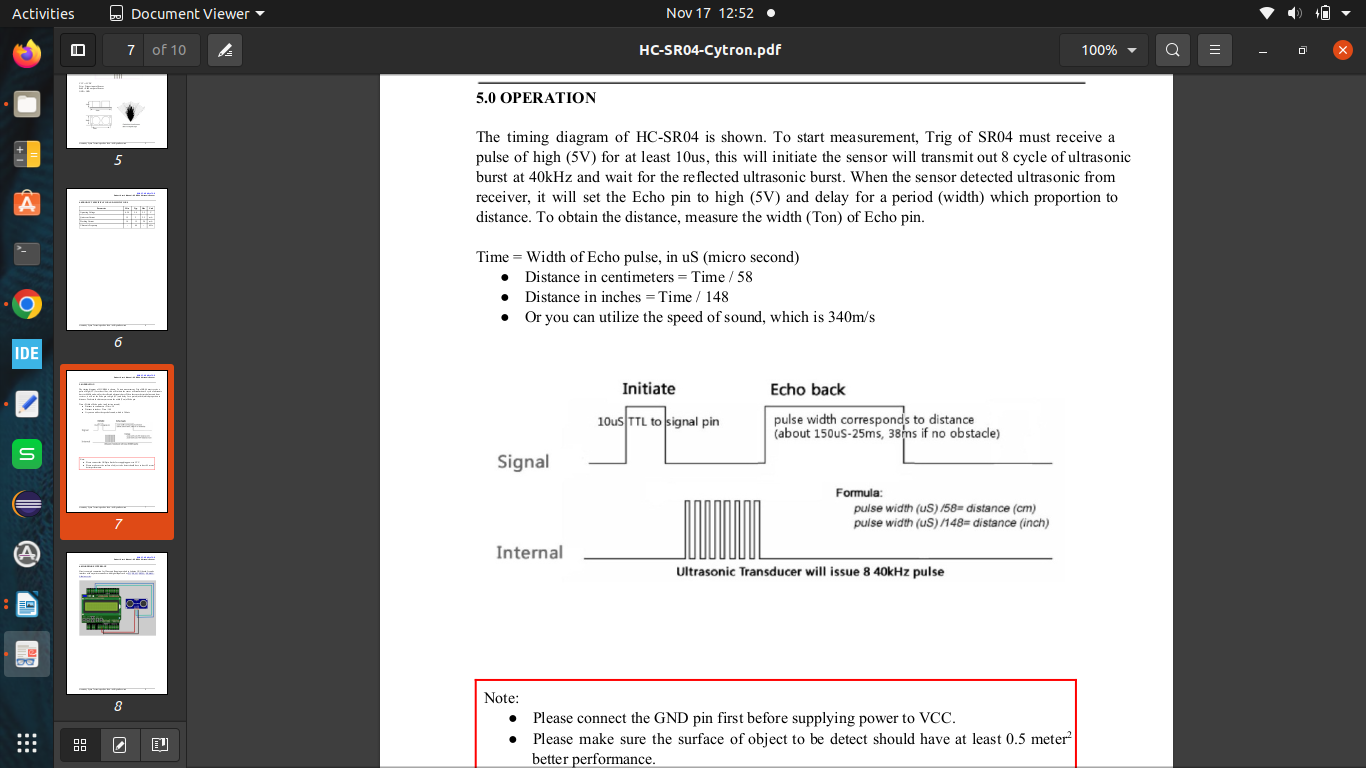
● Distance in inches = Time / 148

● Or you can utilize the speed of sound, which is 340m/s



**Timing diagram**

The Timing diagram is shown below. You only need to supply a short 10uSpulse to the trigger input to start the ranging, and then the module will send outan 8 cycle burst of ultrasound at 40 kHz andraise its echo. The Echo is adistance object that is pulse width and the range in proportion .You cancalculate the range through the time interval between sending trigger signal andreceiving echo signal. Formula: uS / 58 = centimeters or uS / 148 =inch; or: therange = high level time \* velocity(340M/S) / 2; we suggest to use over 60msmeasurement cycle, in order to prevent trigger signal to the echo signal.

****

**4.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**

The STM32F446RE with HC-SR04 is used to communicate with the module and board. It is used to measure the distance between the object.

**STAGE:2**

The STM32F446RE with HC-SR04,W10 is used to transmit the data into the MQTT server to Publish the data in the Right-tech.

**STAGE:3**

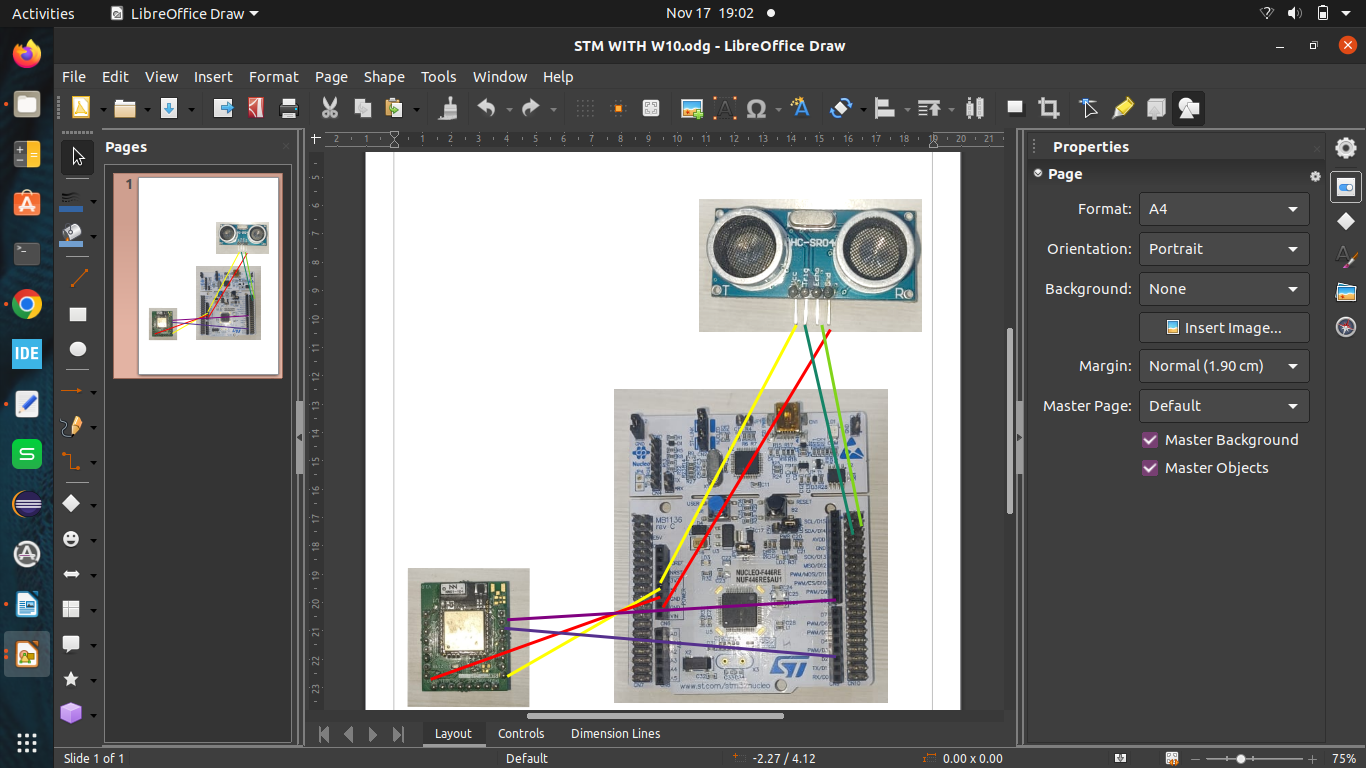
The Ruggedboard with STM32F446RE is used to transmit the data and get the HC-SR04 value in the ruggedboard minicom.

**STAGE:4**

The Ruggedboard with STM32F446RE is used to transmit the data and get the HC-SR04 value in the ruggedboard minicom and pass the value into the Right-tech by using W10 module is connected with the Rugged board.

**5.connection diagram**

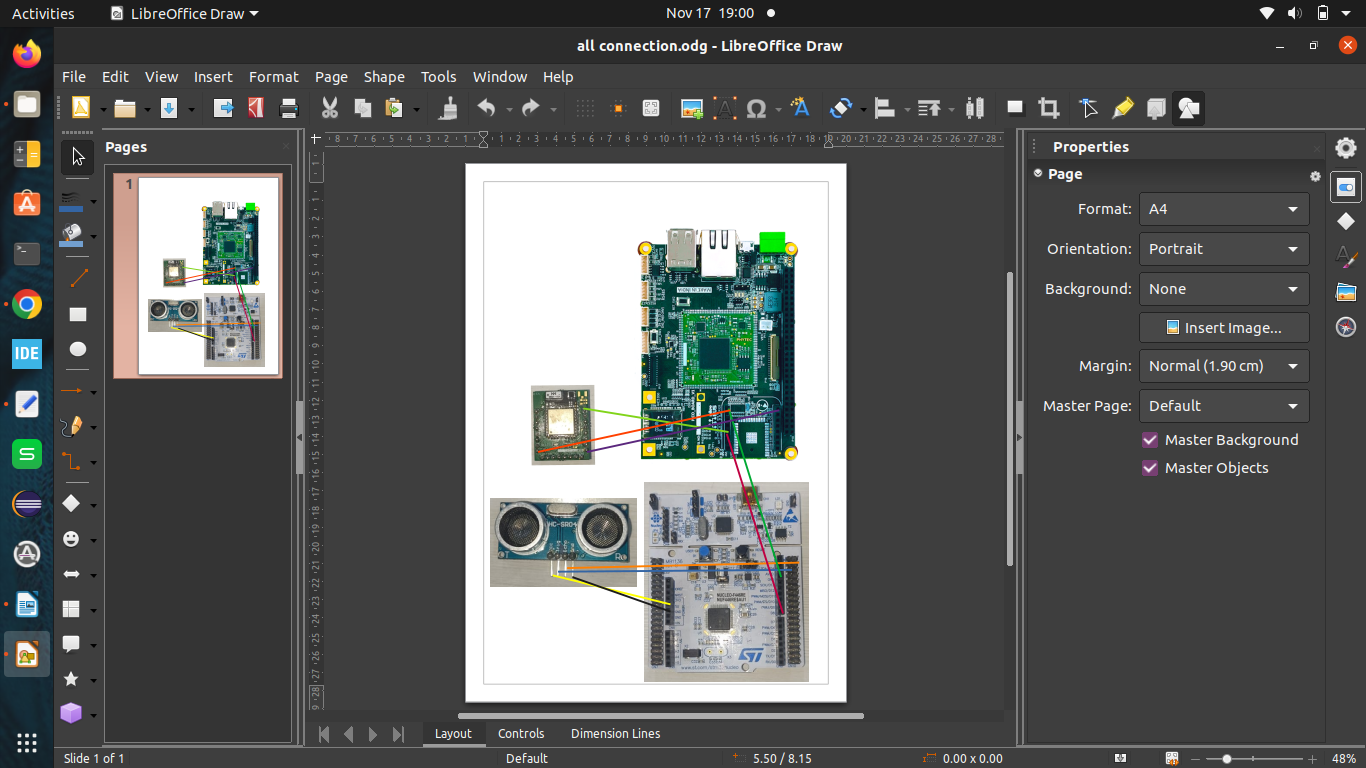
**STAGE:2**



PIN CONNECTION:

|  |  |  |
| --- | --- | --- |
| PIN | PIN NUMBER | COMPONENTS |
| VCC 5V | VCC | W10 AND HC-SR04 |
| GROUND | GND | W10 AND HC-SR04 |
| SCL | PC6 | HC-SR04 |
| SDA | PB9 | HC-SR04 |
| Tx | PA9 | W10 |
| Rx | PA10 | W10 |

**STAGE:4**

****

**In stm32f446re with hc-sr04:**

|  |  |  |
| --- | --- | --- |
| PIN | PIN NUMBER | COMPONENTS |
| VCC 5V | VCC | HC-SR04 |
| GROUND | GND | HC-SR04 |
| SCL | PC6 | HC-SR04 |
| SDA | PB9 | HC-SR04 |
| Tx | PA9 | Ruggedboard |

**In Ruggedboard with w10:**

|  |  |  |
| --- | --- | --- |
| PIN | PIN NUMBER | COMPONENTS |
| VCC 5V | VCC | W10 |
| GROUND | GND | W10 |
| Tx | Tx | W10 |
| Rx | Rx | stm32f446re |

**6.HC-SR04**

The HC-SR04 ultrasonic sensor uses sonar to determine distance to an object like bats or dolphins

do. It offers excellent non-contact range detection with high accuracy and stable readings in an

easy-touse package. From 2cm to 400 cm or 1” to 13 feet. It operation is not affected by sunlight

or black material like Sharp rangefinders are (although acoustically soft materials like cloth can be

difficult to detect). It comes complete with ultrasonic transmitter and receiver module.

**Features:**

● Power Supply :+5V DC

● Quiescent Current : <2mA

● Working Current: 15mA

● Effectual Angle: <15°

● Ranging Distance : 2cm – 400 cm/1" -3ft

● Resolution : 0.3 cm

● Measuring Angle: 30 degree

● Trigger Input Pulse width: 10uS

● Dimension: 45mm x 20mm x 15mm

**OPERATION**

The timing diagram of HC-SR04 is shown. To start measurement, Trig of SR04 must receive a

pulse of high (5V) for at least 10us, this will initiate the sensor will transmit out 8 cycle of ultrasonic

burst at 40kHz and wait for the reflected ultrasonic burst. When the sensor detected ultrasonic from

receiver, it will set the Echo pin to high (5V) and delay for a period (width) which proportion to

distance. To obtain the distance, measure the width (Ton) of Echo pin.

Time = Width of Echo pulse, in uS (micro second)

● Distance in centimeters = Time / 58

● Distance in inches = Time / 148

● Or you can utilize the speed of sound, which is 340m/s

Note:

● Please connect the GND pin first before supplying power to VCC.

● Please make sure the surface of object to be detect should have at least 0.5 meter 2

better performance.

**HARDWARE INTERFACE**

Here is example connection for Ultrasonic Ranging module to Arduino UNO board. It can be

interface with any microcontroller with digital input such as PIC, SK40C, SK28A, SKds40A,

Arduino series.

**7.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

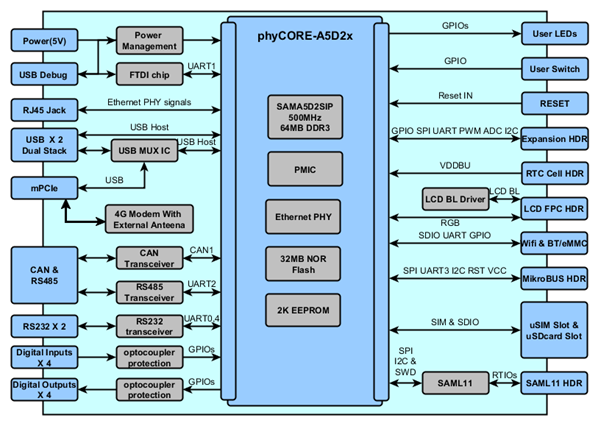
other things.

This RuggedBoardoffers 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

**8.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.

**9.Code snippet to wifi module initialization and connection with interrupt-based uart commands**

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=hari,9486890765\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);

}

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 WiFi 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 WiFi 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 WiFi network. The function takes no arguments and it returns void.

**10.code snippet to MQTT initialization and connection with interrupt-based uart commands**

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-harishkumarslm-qcangb,,,,,,,,,\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 a 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-arifm4348- ud8eo8 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/relay/led1.

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 CMD+MQTTNETCFG command is used to configure the MQTT parameters of the WE10 module. The first parameter is the hostname or IP address of the MQTT broker. The second parameter is the port number of the MQTT broker.

The CMD+MQTTCONCFG command is used to configure the MQTT client of the WE10 module. The first parameter is the username of the MQTT client. The second parameter is the password of the MQTT client.

The CMD+MQTTSTART command is used to start the MQTT client of the WE10 module. This command connects the client to the MQTT broker.

The CMD+MQTTSUB command is used to subscribe the MQTT client to a topic. The first parameter is the topic that the client wants to subscribe to.

**11.Send\_Task Function**

void mqtt\_data\_send()

{

char buffer[50];

sprintf (&buffer[0], "CMD+MQTTPUB=base/state/distance,%.2f\r\n",dis1);

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

HAL\_UART\_Transmit(&huart2, (uint8\_t\*)buffer, strlen(buffer), 1000);

HAL\_Delay(100);

}

This code is a function written in C that sends MQTT (Message Queuing Telemetry Transport) data using UART (Universal Asynchronous Receiver-Transmitter) communication.This declares a function named mqtt\_data\_send with no input parameters and no return value (void).

A character array (buffer) named buffer is declared with a size of 50 characters. This buffer will be used to store the formatted MQTT message.The sprintf function is used to format a string with the MQTT message. The message includes a command ("CMD+MQTTPUB"), a topic ("base/state/distance"), and a floating-point value (dis1) with two decimal places.

The formatted message stored in the buffer is transmitted via UART. The HAL\_UART\_Transmit function is used for this purpose. The message is sent to two UART interfaces (huart1 and huart2), and the third argument specifies the timeout duration (1000 milliseconds in this case).

A delay of 100 milliseconds is introduced using HAL\_Delay. This delay allows time for the UART transmissions to complete before the function exits.In summary, this code snippet is part of a larger program, and its purpose is to format and transmit an MQTT message containing distance information (dis1) over two UART interfaces (huart1 and huart2). The delay at the end ensures that there is sufficient time for the transmissions to complete before moving on.

The code you provided defines the Send\_Task function. The Send\_Task is a task that will be executed by the code.

The Send\_Task function first declares a variable of type data called DatatoSend.

The temp member of the data structure is used to store the temperature reading, and the humidity member of the data structure is used to store the humidity reading.

The Send\_Task function then enters an infinite loop. In each iteration of the loop, the Send\_Task function reads the temperature and humidity readings from the sensors, stores the readings in the DatatoSend structure, and then puts the DatatoSend structure on the myQueueTemp message queue. The osMessageQueuePut() function is used to put a message on a message queue. The first parameter is the handle of the message queue, the second parameter is a pointer to the message, the third parameter is the priority of the message, and the fourth parameter is the timeout value.

**SendTask:**

The SendTask is a task that will be created by the code.The osThreadId\_t SendTaskHandle variable is used to store the handle of the SendTask. The osThreadAttr\_t SendTask\_attributes structure defines the attributes of the SendTask.

The SendTask\_attributes structure has three members: name: The name of the task.stack\_size The size of the stack that will be allocated to the task. priority: The priority of the task.

In this case, the name of the task is "SendTask", the stack\_size is 128 \* 4 bytes, and the priority is osPriorityNormal.

The osThreadAttr\_t structure is used to configure the attributes of a task. The name member is used to set the name of the task. The stack\_size member is used to set the size of the stack that will be allocated to the task. The priority member is used to set the priority of the task.

**RecieveTask:**

The RecieveTask is a task that will be created by the code.

The osThreadId\_t RecieveTaskHandle variable is used to store the handle of the RecieveTask. The osThreadAttr\_t RecieveTask\_attributes structure defines the attributes of the RecieveTask.

The RecieveTask\_attributes structure has three members: name: The name of the task.stack\_size The size of the stack that will be allocated to the task. priority: The priority of the task.

In this case, the name of the task is "RecieveTask", the stack\_size is 128\* 4 bytes, and the priority is osPriorityLow.

The osThreadAttr\_t structure is used to configure the attributes of a task. The name member is used to set the name of the task. The stack\_size member is used to set the size of the stack that will be allocated to the task. The priority member is used to set the priority of the task.

**12.PROJECT CODE**

**STAGE 2:(stm32f446re with w10 module)**

/\* 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"

/\* 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 \*/

uint32\_t echo1,echo2;

float dis1,dis2;

uint32\_t last\_measurement\_time = 0;

/\* USER CODE END PD \*/

/\* Private macro -------------------------------------------------------------\*/

/\* USER CODE BEGIN PM \*/

/\* USER CODE END PM \*/

/\* Private variables ---------------------------------------------------------\*/

TIM\_HandleTypeDef htim8;

TIM\_HandleTypeDef htim11;

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\_TIM8\_Init(void);

static void MX\_TIM11\_Init(void);

/\* USER CODE BEGIN PFP \*/

/\* USER CODE END PFP \*/

/\* Private user code ---------------------------------------------------------\*/

/\* USER CODE BEGIN 0 \*/

/\* USER CODE END 0 \*/

void mqtt\_data\_send()

{

char buffer[50];

sprintf (&buffer[0], "CMD+MQTTPUB=base/state/distance,%.2f\r\n",dis1);

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

HAL\_UART\_Transmit(&huart2, (uint8\_t\*)buffer, strlen(buffer), 1000);

HAL\_Delay(100);

}

/\*\*

\* @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\_TIM8\_Init();

MX\_TIM11\_Init();

WE10\_Init ();

MQTT\_Init();

/\* USER CODE BEGIN 2 \*/

HAL\_TIM\_Base\_Start(&htim11);

HAL\_TIM\_Base\_Start(&htim11);

HAL\_TIM\_PWM\_Start(&htim11,TIM\_CHANNEL\_1);

HAL\_TIM\_IC\_Start(&htim8,TIM\_CHANNEL\_1);

HAL\_TIM\_IC\_Start(&htim8,TIM\_CHANNEL\_2);

TIM11->CCR1 = 3;

/\* USER CODE END 2 \*/

/\* Infinite loop \*/

/\* USER CODE BEGIN WHILE \*/

while (1)

{

/\* USER CODE END WHILE \*/

/\* USER CODE BEGIN 3 \*/

if (HAL\_GetTick() - last\_measurement\_time >= 1000)

{

last\_measurement\_time = HAL\_GetTick();

echo1 = HAL\_TIM\_ReadCapturedValue(&htim8, TIM\_CHANNEL\_1);

dis1 = echo1 / 58.0f;

printf("Echo1: %u Dis1: %.2f cm\r\n", echo1, dis1);

char buffer[50];

sprintf(buffer, "Distance: %.2f cm Echo: %u\r\n", dis1 , echo1);

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

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

HAL\_Delay(1000); // Adjust this delay as needed for your application

}

}

/\* USER CODE END 3 \*/

}

/\*\*

\* @brief System Clock Configuration

\* @retval None

\*/

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=hari,9486890765\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-harishkumarslm-qcangb,,,,,,,,,\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);

}

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 TIM8 Initialization Function

\* @param None

\* @retval None

\*/

static void MX\_TIM8\_Init(void)

{

/\* USER CODE BEGIN TIM8\_Init 0 \*/

/\* USER CODE END TIM8\_Init 0 \*/

TIM\_SlaveConfigTypeDef sSlaveConfig = {0};

TIM\_MasterConfigTypeDef sMasterConfig = {0};

TIM\_IC\_InitTypeDef sConfigIC = {0};

/\* USER CODE BEGIN TIM8\_Init 1 \*/

/\* USER CODE END TIM8\_Init 1 \*/

htim8.Instance = TIM8;

htim8.Init.Prescaler = 0;

htim8.Init.CounterMode = TIM\_COUNTERMODE\_UP;

htim8.Init.Period = 65535;

htim8.Init.ClockDivision = TIM\_CLOCKDIVISION\_DIV1;

htim8.Init.RepetitionCounter = 0;

htim8.Init.AutoReloadPreload = TIM\_AUTORELOAD\_PRELOAD\_DISABLE;

if (HAL\_TIM\_Base\_Init(&htim8) != HAL\_OK)

{

Error\_Handler();

}

if (HAL\_TIM\_IC\_Init(&htim8) != HAL\_OK)

{

Error\_Handler();

}

sSlaveConfig.SlaveMode = TIM\_SLAVEMODE\_RESET;

sSlaveConfig.InputTrigger = TIM\_TS\_TI2FP2;

sSlaveConfig.TriggerPolarity = TIM\_INPUTCHANNELPOLARITY\_RISING;

sSlaveConfig.TriggerFilter = 0;

if (HAL\_TIM\_SlaveConfigSynchro(&htim8, &sSlaveConfig) != HAL\_OK)

{

Error\_Handler();

}

sMasterConfig.MasterOutputTrigger = TIM\_TRGO\_RESET;

sMasterConfig.MasterSlaveMode = TIM\_MASTERSLAVEMODE\_DISABLE;

if (HAL\_TIMEx\_MasterConfigSynchronization(&htim8, &sMasterConfig) != HAL\_OK)

{

Error\_Handler();

}

sConfigIC.ICPolarity = TIM\_INPUTCHANNELPOLARITY\_RISING;

sConfigIC.ICSelection = TIM\_ICSELECTION\_DIRECTTI;

sConfigIC.ICPrescaler = TIM\_ICPSC\_DIV1;

sConfigIC.ICFilter = 0;

if (HAL\_TIM\_IC\_ConfigChannel(&htim8, &sConfigIC, TIM\_CHANNEL\_1) != HAL\_OK)

{

Error\_Handler();

}

if (HAL\_TIM\_IC\_ConfigChannel(&htim8, &sConfigIC, TIM\_CHANNEL\_2) != HAL\_OK)

{

Error\_Handler();

}

/\* USER CODE BEGIN TIM8\_Init 2 \*/

/\* USER CODE END TIM8\_Init 2 \*/

}

/\*\*

\* @brief TIM11 Initialization Function

\* @param None

\* @retval None

\*/

static void MX\_TIM11\_Init(void)

{

/\* USER CODE BEGIN TIM11\_Init 0 \*/

/\* USER CODE END TIM11\_Init 0 \*/

TIM\_OC\_InitTypeDef sConfigOC = {0};

/\* USER CODE BEGIN TIM11\_Init 1 \*/

/\* USER CODE END TIM11\_Init 1 \*/

htim11.Instance = TIM11;

htim11.Init.Prescaler = 199;

htim11.Init.CounterMode = TIM\_COUNTERMODE\_UP;

htim11.Init.Period = 39999;

htim11.Init.ClockDivision = TIM\_CLOCKDIVISION\_DIV1;

htim11.Init.AutoReloadPreload = TIM\_AUTORELOAD\_PRELOAD\_DISABLE;

if (HAL\_TIM\_Base\_Init(&htim11) != HAL\_OK)

{

Error\_Handler();

}

if (HAL\_TIM\_PWM\_Init(&htim11) != HAL\_OK)

{

Error\_Handler();

}

sConfigOC.OCMode = TIM\_OCMODE\_PWM1;

sConfigOC.Pulse = 0;

sConfigOC.OCPolarity = TIM\_OCPOLARITY\_HIGH;

sConfigOC.OCFastMode = TIM\_OCFAST\_DISABLE;

if (HAL\_TIM\_PWM\_ConfigChannel(&htim11, &sConfigOC, TIM\_CHANNEL\_1) != HAL\_OK)

{

Error\_Handler();

}

/\* USER CODE BEGIN TIM11\_Init 2 \*/

/\* USER CODE END TIM11\_Init 2 \*/

HAL\_TIM\_MspPostInit(&htim11);

}

/\*\*

\* @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 = 115200;

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 \*/

}

/\*\*

\* @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 \*/

/\* 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 \*/

**STAGE:3**

#include<stdio.h>

#include<string.h>

#include<errno.h>

#include<stdlib.h>

void error(const char \*msg) {

perror(msg);

exit(1);

}

int main() {

const char \*portname = "/dev/ttyS3"; // Replace with the actual UART device file

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

if (fd < 0) {

error("Error opening UART");

}

struct termios tty;

if (tcgetattr(fd, &tty) < 0) {

error("Error from tcgetattr");

}

cfsetospeed(&tty, B115200); // Set the baud rate

cfsetispeed(&tty, B115200);

tty.c\_cflag |= (CLOCAL | CREAD); // Ignore modem control lines, enable receiver

tty.c\_cflag &= ~CSIZE; // Clear data size bits

tty.c\_cflag |= CS8; // 8-bit data

tty.c\_cflag &= ~PARENB; // No parity bit

tty.c\_cflag &= ~CSTOPB; // 1 stop bit

tty.c\_cflag &= ~CRTSCTS; // No hardware flow control

tty.c\_lflag = 0; // Non-canonical mode

tty.c\_cc[VMIN] = 1; // Minimum number of characters to read

tty.c\_cc[VTIME] = 1; // Time to wait for data (in tenths of a second)

if (tcsetattr(fd, TCSANOW, &tty) != 0) {

error("Error from tcsetattr");

}

while(1)

{

char buf[50];

memset(buf, 0, sizeof(buf));

int n = read(fd, buf, sizeof(buf));

if (n < 0) {

error("Error reading");

}

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

}

close(fd);

return 0;

}

**STAGE:4(rugged board with w10)**

#include <errno.h>

#include <fcntl.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <termios.h>

#include <unistd.h>

int set\_interface\_attribs(int fd, int speed)

{

struct termios tty;

if (tcgetattr(fd, &tty) < 0)

{

printf("Error from tcgetattr: %s\n", strerror(errno));

return -1;

}

cfsetispeed(&tty, (speed\_t)speed);

tty.c\_cflag |= (CLOCAL | CREAD); /\* ignore modem controls \*/

tty.c\_cflag &= ~CSIZE;

tty.c\_cflag |= CS8; /\* 8-bit characters \*/

tty.c\_cflag &= ~PARENB; /\* no parity bit \*/

tty.c\_cflag &= ~CSTOPB; /\* only need 1 stop bit \*/

tty.c\_cflag &= ~CRTSCTS; /\* no hardware flowcontrol \*/

tty.c\_iflag = IGNPAR;

tty.c\_lflag = 0;

tty.c\_cc[VMIN] = 1;

tty.c\_cc[VTIME] = 1;

if (tcsetattr(fd, TCSANOW, &tty) != 0)

{

printf("Error from tcsetattr: %s\n", strerror(errno));

return -1;

}

return 0;

}

r t

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=\"hari\",\"9865453376\"\r\n";

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

char arr4[] = "CMD+MQTTCONCFG=3,mqtt-harishkumarslm-qcangb,,,,,,,,,\r\n";

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

char arr6[] = "CMD+MQTTSUB=base/state/centimeter\r\n";

// char arr7[] = "CMD+MQTTPUB=base/state/centimeter\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);

//rdlen = read(fd, buf, sizeof(buf));

//buf[rdlen]='\0';

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

// Send CMD+WIFIMODE=1

printf("%s", arr2);

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

sleep(3);

//rdlen = read(fd, buf, sizeof(buf));

//buf[rdlen]='\0';

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

// Send CMD+CONTOAP

printf("%s", arr);

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

sleep(3);

//rdlen = read(fd, buf, sizeof(buf));

//buf[rdlen]='\0';

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

printf("%s", arr3);

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

sleep(3);

//rdlen = read(fd, buf, sizeof(buf));

//buf[rdlen]='\0';

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

printf("%s", arr4);

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

sleep(3);

//rdlen = read(fd, buf, sizeof(buf));

//buf[rdlen]='\0';

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

printf("%s", arr5);

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

sleep(3);

//rdlen = read(fd, buf, sizeof(buf));

//buf[rdlen]='\0';

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

//while(1){

printf("%s", arr6);

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

sleep(3);

//rdlen = read(fd, buf, sizeof(buf));

//buf[rdlen]='\0';

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

// printf("%s", arr7);

// wlen = write(fd, arr7, sizeof(arr7) - 1);

// sleep(3);

// char buff[10]="c";

// rdlen = read(fd, buf, sizeof(buf) - 1); // Read data into the buffer

// if (rdlen > 0) {

// buf[rdlen-3] = '\0'; // Null-terminate the received data

// printf("Received data: %s\n", buf); }

// wlen = write(fd , buf, sizeof(buf));

// wlen = write(fd ,"CMD+MQTTPUB=base/state/centimeter,%s\r\n", buf);

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

//char buf[100]="42"; // Create a buffer to store the value read

//int fd; // Your file descriptor

//set\_interface\_attribs(fd, B9600);

// Read data into the 'buf' buffer

while(1){

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

if (rdlen > 0) {

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

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

// Format the data from 'buf' into 'buffer'

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

if (ret < 0) {

// Handle the error if snprintf fails

} else {

// Open the file descriptor 'fd' if not already opened

// Write the formatted message to the file descriptor

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

sleep(3);

if (wlen == -1) {

// Handle the write error if needed

}

}

}

}

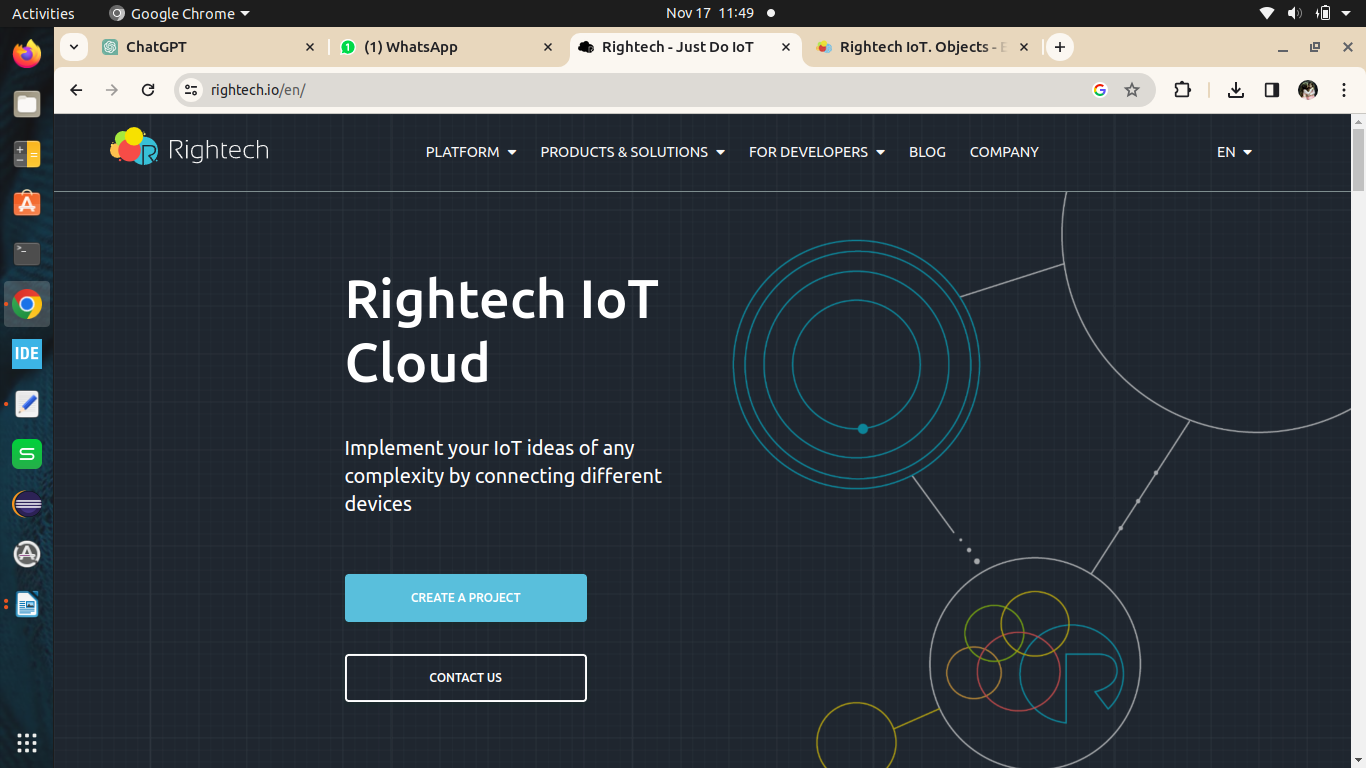
close(fd);

return 0;

}

**13.RIGHT TECH OUTPUT**

Rightech IoT Cloud is a tool for developers. RIC is independent of specific equipment and protocols, which makes it easier for developers to combine different devices under one solution.Platform tools allow developers to create IoT solutions without extra code and reuse 90% of that solution to launch similar cases.

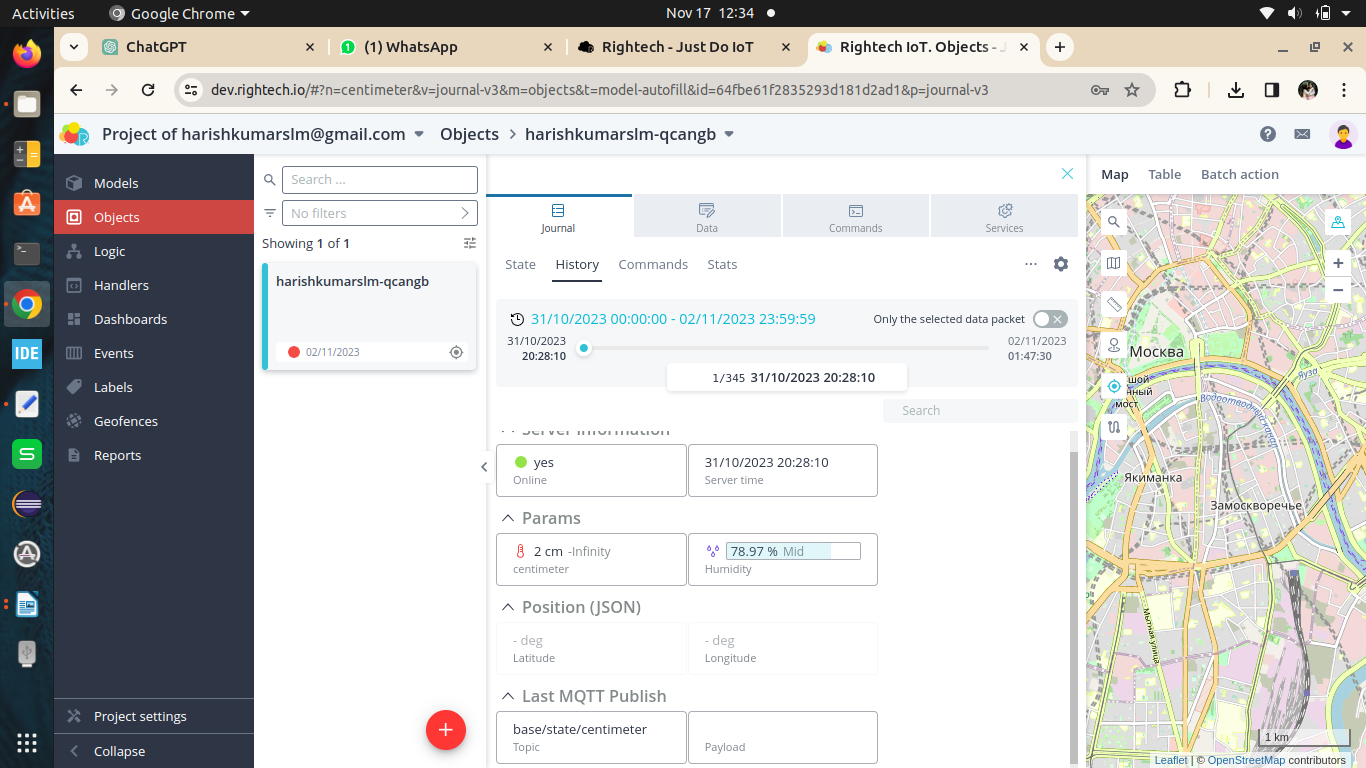


We can download the stored data from Rightech IoT in different formats and those are

GPX, or GPS Exchange Format, is an XML schema designed as a common GPS data format for software applications.

CSV (comma-separated values) file is a text file that has a specific format that allows data to be saved in a table-structured format.

JSON (JavaScript Object Notation, pronounced /ˈdʒeɪsən/; also /ˈdʒeɪˌsɒn/) is an open standard file format and data interchange formatthat uses human-readable text to store and transmit data objects consisting of attribute–value pairs and arrays (or other serializable values).I have downloaded it in JSON format which is given below.



{

"id": "centimeter",

"name": "centimeter",

"active": true,

"type": "argument",

"source": "state",

"dataType": "number",

"unit": "length-centimeter",

"reference": "base/state/centimeter",

"display": {},

"factor": 1,

"linear": true,

"children": [],

"levels": {

"type": "range",

"value": [

{

"color": "#f16059",

"name": "-Infinity",

"value": {

"a": "-Infinity",

"b": 10

}

},

{

"color": "#a2ce4b",

"name": "Min",

"value": {

"a": 10,

"b": 30

}

},

{

"color": "#ffae62",

"name": "Mid",

"value": {

"a": 30,

"b": 50

}

},

{

"color": "#f16059",

"name": "Max",

"value": {

"a": 50,

"b": "Infinity"

}

}

],

"svg": "<svg xmlns=\"http://www.w3.org/2000/svg\" x=\"0px\" y=\"0px\"\nviewBox=\"0 0 16 16\"\n><g id=\"surface1\"><path style=\" \" d=\"M 7.511719 1 C 6.140625 1 5.011719 2.128906 5.011719 3.5 L 5.011719 9.121094 C 4.414063 9.746094 4 10.5625 4 11.5 C 4 13.425781 5.574219 15 7.5 15 C 9.425781 15 11 13.425781 11 11.5 C 11 10.574219 10.597656 9.765625 10.011719 9.140625 L 10.011719 3.5 C 10.011719 2.128906 8.882813 1 7.511719 1 Z M 7.511719 2 C 8.339844 2 9.011719 2.671875 9.011719 3.5 L 9.011719 4 L 7 4 L 7 5 L 9.011719 5 L 9.011719 6 L 7 6 L 7 7 L 9.011719 7 L 9.011719 8 L 7 8 L 7 9 L 9.011719 9 L 9.011719 9.289063 C 9.011719 9.429688 9.070313 9.5625 9.175781 9.65625 C 9.683594 10.117188 10 10.765625 10 11.5 C 10 12.886719 8.886719 14 7.5 14 C 6.113281 14 5 12.886719 5 11.5 C 5 10.757813 5.328125 10.105469 5.84375 9.640625 C 5.949219 9.546875 6.011719 9.414063 6.011719 9.269531 L 6.011719 3.5 C 6.011719 2.671875 6.683594 2 7.511719 2 Z \"></path></g></svg>"

},

"usage": null

},