KY-031 Knock-sensor module

PRESENTED BY

S.PRADEEP KUMAR

NOVEMBER 2023

TABLE OF CONTENTS

1)Project overview

2)Hardware Components

2.1.TAP MODULE KY-031

2.2.STM32F446RE

2.3.Rugged board –a5d2x

2.4.WE10 module

3)Software Components

3.1.STM32 IDE tool

3.2.minicom

3.3.Right tech IOT cloud

4)PROJECT STAGES

5)Code snippet to wifi module initialization

6)code snippet to MQTT initialization

7)Send_Task Function

8)PROJECT CODE

9)MINICOM OUTPUT

10)RIGHT TECH OUTPUT

11)Applications

12)References

1.Project overview

The KY-031 is a small module designed for detecting taps or vibrations in the surrounding environment. It is commonly used in electronic projects and prototypes where sensing physical movement or impacts is essential.

In this project, the STM32F446RE microcontroller serves as the central processing unit, orchestrating the interaction with various components. The KY-031 tap module is employed as a sensor to detect taps or vibrations in the surrounding environment. This module is interfaced with the STM32F446RE, allowing it to capture tap events and respond accordingly.

The primary feature of tap detection is implemented through a dedicated algorithm. Upon detecting a tap, the STM32F446RE triggers the blinking of an LED, providing a visual indication of the detected event. The LED serves as a local indicator of tap occurrences, enhancing the user experience and making the system responsive to environmental stimuli.

To facilitate monitoring and debugging, the project incorporates UART communication with Minicom. When a tap is detected, relevant information is transmitted to Minicom, allowing real-time observation of system behavior. This UART communication pathway acts as a valuable debugging tool, aiding in the development and optimization of the tap detection algorithm.

Additionally, the project includes UART communication with Rightec, an interface that allows the STM32F446RE to transmit data to the Rightec environment. This integration enables the visualization of sensor data or tap events within the Rightec interface, providing a comprehensive and user-friendly representation of the system's behavior.

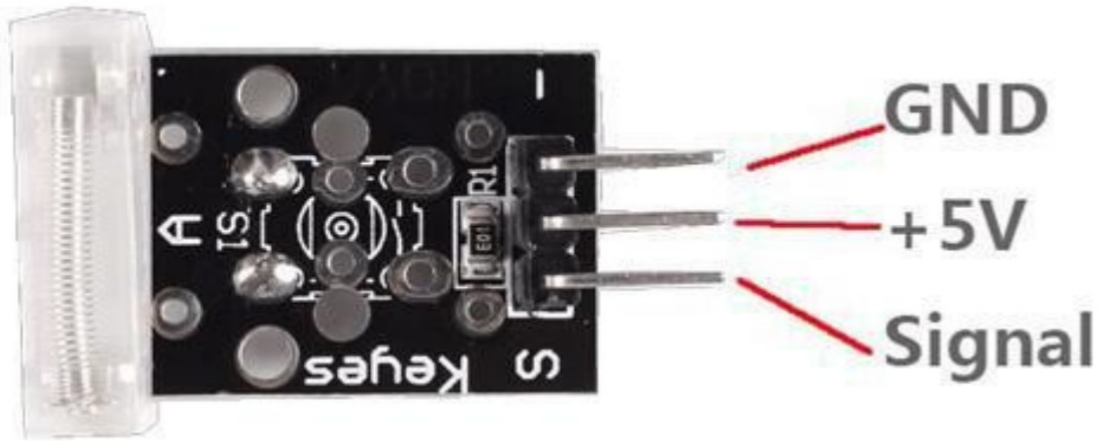
Furthermore, the project involves the transmission of strings to a rugged board via UART communication. These strings may contain relevant data or messages intended for processing by the rugged board, enhancing the project's versatility and potential applications.

In summary, the project successfully demonstrates the integration of the STM32F446RE microcontroller with diverse components, showcasing capabilities in tap detection, LED control, UART communication with Minicom and Rightec, and string transmission to a rugged board. The comprehensive functionality and integration make this project suitable for applications requiring sensor-based event detection, data transmission, and visualization in both local and remote environments.

2.Hardware Components

2.1.KY-031(TAP MODULE)
2.2.STM32F446RE
2.3.Rugged board –a5d2x
2.4.WE10 module

2.1.1.TAP MODULE KY-031



The KY-031 Tap Module is a sensor designed to detect taps or impacts and convert these mechanical stimuli into electrical signals. This type of sensor is commonly used in projects where the detection of tapping or knocking events is essential. The KY-031 variant is engineered to be a compact and versatile solution for applications that require a simple and effective tap detection mechanism.

In terms of its basic functionality, the KY-031 Tap Module employs a piezoelectric element or a vibration sensor as its core component. When the module experiences a tap or vibration, the piezoelectric element generates a small electrical charge in response to the mechanical stress. This charge is then converted into an electrical signal that can be processed by microcontrollers or other electronic systems.

The KY-031 Tap Module typically comes with pins for easy integration into a circuit. The output signal generated by the module can be used to trigger specific actions or responses in a connected electronic system. Additionally, these modules often include sensitivity adjustments, allowing users to fine-tune the sensor's responsiveness to taps of varying intensities.Due to its simplicity and ease of use, the KY-031 Tap Module finds applications in a range of projects. Common uses include implementing tap-controlled interfaces in electronic devices, creating interactive surfaces that respond to taps or knocks, or developing simple alarms triggered by impact events.

In summary, the KY-031 Tap Module is a compact and efficient sensor designed for tap and impact detection. Its straightforward design, coupled with adjustable sensitivity, makes it suitable for a variety of projects where the detection of tapping events is critical for proper system functioning. Whether integrated into interactive prototypes, electronic gadgets, or security systems, the KY-031 Tap Module adds a valuable element of touch or impact sensing to diverse applications.

2.1.2.Features of ky-031

The KY-031 Tap Module is a sensor designed for detecting taps or impacts and is known for its simplicity and effectiveness in various electronic projects. While specific features may vary depending on the manufacturer, here are some common features associated with the KY-031 Tap Module:

- **1. Piezoelectric Sensor:** The KY-031 Tap Module typically incorporates a piezoelectric element or a vibration sensor as its primary sensing component. This allows the module to convert mechanical vibrations, such as taps or knocks, into electrical signals.

- **2. Compact Design:** The module is designed to be compact, making it suitable for applications where space is limited. Its small form factor enables easy integration into different electronic systems.

- **3.Pin Configuration:** The KY-031 Tap Module commonly includes pins that facilitate straightforward connection to a microcontroller or other electronic devices. This user-friendly design enhances the ease of integration into various projects.

- **4.Adjustable Sensitivity:** Many KY-031 Tap Modules come with sensitivity adjustment features. This allows users to fine-tune the sensor's responsiveness to taps of different intensities, making it adaptable to a variety of scenarios.

- **5.Digital Output:** The module typically provides a digital output signal that changes state when a tap or impact is detected. This output can be easily interfaced with microcontrollers for further processing or to trigger specific actions in the connected electronic system.

- **6.Versatility:** Due to its simplicity and effectiveness, the KY-031 Tap Module finds applications in a wide range of projects. It is commonly used in tap-controlled interfaces, interactive surfaces, or projects requiring impact detection.

- **7.Low Power Consumption:** Many KY-031 modules are designed to operate with low power consumption, making them suitable for battery-powered applications where energy efficiency is crucial.

- **8.Cost-Effective:** The KY-031 Tap Module is often cost-effective, providing an affordable solution for projects that require tap or impact sensing capabilities without compromising functionality.

2.1.3.pin connections:

The KY-031 tap module typically has three pins, and their configuration is as follows:

- **1.VCC (Power):** Connect this pin to the positive voltage supply (e.g., 5V) of your circuit.

- **2.GND (Ground):** Connect this pin to the ground (0V) of your circuit.

- **3.Signal(Output):** Some versions of the KY-031 module may have an signal pin. This pin provides an signal corresponding to the intensity of the tap or knock. If you don't need the information, you can leave this pin unconnected.

Pin Name	Description
VIN	Module power supply 5V
GND	Ground
OUT	Signal Output

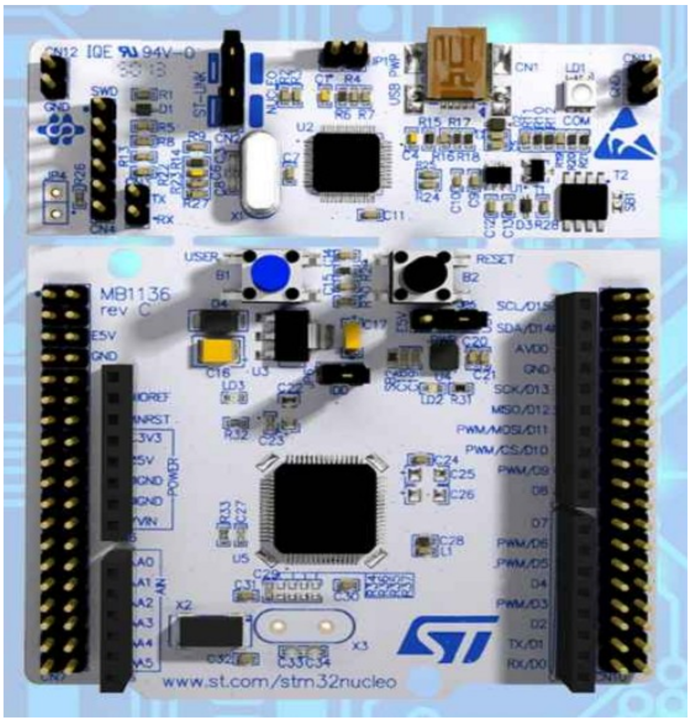
2.1.4Usage of tap module ky-031

The KY-031 module you are referring to is commonly known as a "Tap Sensor" or "Tap Module." This module detects taps or knocks and can be used in various electronic projects. Here are some typical uses for the KY-031 Tap Module:

- **1.Tapping Detection:** The primary purpose of the KY-031 is to detect taps or knocks. When you tap or knock on the module, it generates a signal that can be used to trigger an action or response in your circuit.
- **2.Interactive Projects:** The module can be used in interactive projects where a tap or knock serves as an input to control a device or initiate a specific function.
- **3.Gesture-Controlled Devices:** Incorporate the KY-031 in projects where gestures or taps are used to control electronic devices. For example, you might tap a surface to turn on a light or switch between modes.
- **4.Sound-Activated Systems:** Use the tap sensor in sound-activated systems where a tap serves as a trigger to activate or deactivate a device.
- **5.Alarm Systems:** You can use the KY-031 as part of an alarm system, where a tap or knock can be detected to trigger an alarm or alert.

- **6.Education and Prototyping:** The module is also useful for educational purposes, helping students understand basic sensor principles and serving as a prototyping tool for experimenting with different electronic circuits.

2.2.STM32F446RE microcontroller



• The STM32F446xC/E devices are based on the high-performance Arm® Cortex®-M4 32-bit RISC core operating at a frequency of up to 180 MHz. The Cortex-M4 core features a floating point unit (FPU) single precision supporting all Arm® single-precision data-processing instructions and data types. It also implements a full set of DSP instructions and a memory protection unit (MPU) that enhances application security.

• The STM32F446xC/E devices incorporate high-speed embedded memories (Flash memory up to 512 Kbytes, up to 128 Kbytes of SRAM), up to 4 Kbytes of backup SRAM, and an extensive range of enhanced I/Os and peripherals connected to two APB buses, two AHB buses and a 32-bit multi-AHB bus matrix.

• All devices offer three 12-bit ADCs, two DACs, a low-power RTC, twelve general-purpose 16-bit timers including two PWM timers for motor control, two general-purpose 32-bit timers.

• They also feature standard and advanced communication interfaces.

• Integration of services from STM32CubeMX:STM32 microcontroller, microprocessor, development platform and example project selectionPinout, clock, peripheral, and middleware configurationProject creation and generation of the initialization codeSoftware and middleware completed with enhanced STM32Cube Expansion Packages

• Based on Eclipse®/CDT™, with support for Eclipse® add-ons, GNU C/C++ for Arm® toolchain and GDB debugger

• STM32MP1 Series:Support for OpenSTLinux projects: LinuxSupport for Linux

• Additional advanced debug features including:CPU core, peripheral register, and memory

2.3Rugged board-a5d2x

Rugged Board - A5D2x is an Single Board Computer providing as easy migration path from Micro controller to Microprocessor. Rugged Board is enabled with industry Standard yocto Build Embedded Linux platform and open source libraries for industrial application development. RuggedBoard is an Open source Industrial single board computer powered by ARM Cortex-A5 SoC @500 MHz, implemented with the finest platform for rapid prototyping. The usage of System On Module over a System On Chip is the most rapid way to achieve time to market, curtail development risks for product quantities ranging from a few hundred to thousands. RuggedBoard- A5D2x consists of Multiple Interfaces such as Ethernet, RS232, CAN, RS485, Digital Input and Digital Output with optically isolated, Standard MikroBus header for Add-On Sensors, Actuators and Multiple Wireless Modules such as ZigBee, LoRa, Bluetooth etc. mPCIe connector with USB interface used for Cloud Connectivity modules 3G, 4G, NB-IoT, WiFi. Expansion header with GPIO, UART, I2C, SPI, PWR etc.

2.3.1RuggedBoard - 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

2.4.WE10 MODULE

2.4.1.Overview

- WE-XX series of WiFi modules are controlled by a simple, intutive serial ASCII command interface. There are three
- different types of data sets that are exchanged between the module and host controller; Commands, Responses andEvents.

2.4.2.Commands

- These are used to command the WiFi module to perform a particular function. All the commands follow the structure
- described.
- CMD+<command>=<parameter 1>,<parameter 2><CR><LF>
- CMD – ‘Op code’ to identify the data stream as a ‘Command’.
- <command> - Name of command @refer command list (table x)
- <parameters> - Parameters expected by the command @refer command list (table x)
- <CR><LF> - Every command ends with “<CR><LF>” characters.

- There are 2 types of commands; SET commands and GET commands.
- (a) SET commands: These commands are used to set a parameters for a particular function.
- eg. “CMD+NAME=test123<CR><LF>”. This command sets the name of the device to ‘test123’.
- (b) GET commands: These commands are used to get default/stored parameters from the module.
- eg. “CMD?NAME<CR><LF>” This command fetches the name of the module.

- WE-XX modules respond to all the commands with a status code.
- Total length of any ATCMD should not exceed 1600 bytes for WE10.
- Total length of any ATCMD should not exceed 3200 bytes for WE20D.

2.4.3.Response

- WE-XX modules send out data packets in response to commands sent by the host controller. All the reponses follow
- the structure described.
- RSP=<status>,<paramter 1>,<parameter 2><CR><LF>
- RSP – Op code to identify the data stream as a ‘Response’
- <status> - Execution status of a particular command that was sent by the host controller. <parameters> - Any parameters requested by the host controller @refer command list (table x)
- <CR><LF> - Every response ends with “<CR><LF>” characters.

2.4.4.Events

- WE-XX module generate events to notify the host controller of special conditions like “peer connection” , “disconnec-
- tion”, “data reception” etc. All the events follow the structure described
- EVT+<event>=<parameter 1>,<parameter 2><CR><LF>
- EVT – Op Code to identify the data stream as an ‘Event’
- <event> - Name of the event @refer event list (table x)
- <parameters> - Any parameters that would be needed by the host controller @refer event list (table x)
- <CR><LF> - Every event ends with “<CR><LF>” characters.
- © 2021, Celium Devices Private Limited2.
- Default UART Settings
- Baud rate 38400
- HWFC Disabled
- Parity None
- Data bits 8
- Stop bits 1

2.4.5.Commands:

2.4.5.1.System Commands:

- CMD+RESET command to soft reset module

- CMD+FACRESET Command to reset factory data, uart, ota and fast connect details set to default.

- CMD+UARTCONF Command to configure UART

- CMD+GPIO Command to control GPIO

2.4.5.2.WIFI b/g/n Commands:

- CMD+WIFIMODE Command to set the WiFi mode
- CMD+CONTOAP Command to connect to an Access Point (AP)
- CMD+DISCONN Command to disconnect from the AP
- CMD+SCANAP Command to scan for nearby Access Points (APs)
- CMD+SETAP Command to configure AP
- CMD+DHCP Command to set IP Type (DHCP or Static IP)
- CMD+STAIP Command to set static ip in STA mode
- CMD+GATEWAYIP Command to set the Gateway IP in AP mode
- CMD+AUTOCONAP Command to enable auto connection.
- CMD+MAC Command to set the mac address of the device

2.4.5.3.TCP/UDP/SSL commands:

- CMD+SETSERVER Command to set TCP/UDP/SSL server
- CMD+SETCLIENT Command to set TCP/UPD/SSL client
- CMD+CLOSESKT Command to close TCP/UDP/SSL connection

- CMD+SKTSENDDATA Command to send data via TCP/UDP/SSL connections
- CMD+SKTRCVDATA Command to get received data from TCP/UDP/SSL connections
- CMD+SKTAUTORCV Command to enable/disable auto receive data from SKT connections

2.4.5.4.HTTP Commands:

- CMD+HTTP Command to push data to a web server using HTTP/HTTPs protocol
- CMD+HTTPDATA Command to make multiple POST and PUT requests in single session
- CMD+HTTPCLOSE Command to close current HTTP session

2.4.5.5.MQTT Commands:

- CMD+MQTTCONCFG Command to set MQTT connection configuration
- CMD+MQTTNETCFG command to set MQTT network configuration
- CMD+MQTTPUB Command to publish data to broker
- CMD+MQTTSUB Command to subscribe to MQTT topic
- CMD+MQTTUNSUB Command to unsubscribe to mqtt topic

- CMD+MQTTSTART Command to start mqtt communication

- CMD+MQTTSSL Command to set MQTT SSL and CRT conifg

3.Software components:

3.1.STM32 IDE tool

3.2.minicom

3.3.Right tech IOT cloud

3.1.STM IDE TOOL

Overview

3.1.1.Integrated Development Environment (IDE):

• The IDE is the primary interface for software development. It includes code editing, debugging, and project management tools.

3.1.2 STM32CubeMX:

• STM32CubeMX is a graphical configuration tool that allows you to initialize the peripherals and configure the pin assignments for your STM32 microcontroller.

3.1.3 HAL (Hardware Abstraction Layer) Library:

• STM32CubeIDE integrates the HAL library, which provides a set of high-level functions to interact with the microcontroller peripherals.

3.1.4 CMSIS (Cortex Microcontroller Software Interface Standard):

• CMSIS provides a standardized interface to the core functions of a Cortex-M microcontroller, including the NVIC (Nested Vector Interrupt Controller) and SysTick.

3.1.5 Debugger:

• The debugger allows you to set breakpoints, inspect variables, and step through code during the debugging process.

3.1.6 Project Configuration:

• You can configure various project settings, including compiler options, linker scripts, and build configurations.

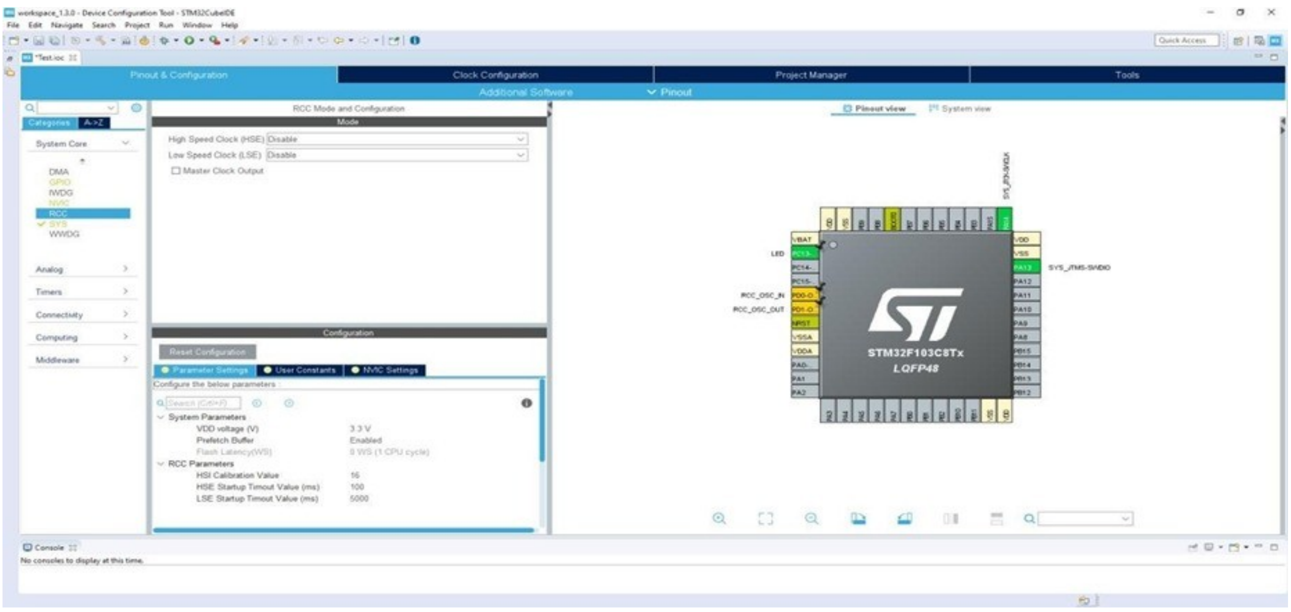
3.1.7 Console Output:

• The console provides feedback from the compiler and debugger, displaying messages, warnings, and errors.

3.1.8 Peripheral Configuration:

• STM32CubeMX generates initialization code based on your peripheral configurations, helping to set up the microcontroller's features.

3.1.9 Project Explorer:



RCC Mode and Configuration

Mode

High Speed Clock (HSE) Disable

Low Speed Clock (LSE) Disable

☐ Master Clock Output

Configuration

Reset Configuration

Parameter Settings | User Constants | NVIC Settings

Configure the below parameters :

```
System Parameters
    VDD voltage (V)               3.3 V
    Prefetch Buffer               Enabled
    Flash Latency(WS)             0 WS (1 CPU cycle)
RCC Parameters
    HSI Calibration Value         16
    HSE Startup Timout Value (ms) 100
    LSE Startup Timout Value (ms) 5000
```

• The Project Explorer organizes your project files and allows you to navigate through the source code and configuration files.

3.1.10 Workflow:

Project Initialization:

• Create a new project in STM32CubeIDE.

• Configure the microcontroller and peripherals using STM32CubeMX.

Code Development:

• Write and edit your C code in the IDE.

Build and Compilation:

• Compile your code to generate the binary file.

Debugging:

• Use the debugger to find and fix issues in your code.

Flash and Run:

• Flash the compiled code onto the STM32 microcontroller and run the application.

3.2. MINICOM

Minicom is a text-based serial communication program that is commonly used to connect to and communicate with devices over a serial port. It is often used 16 for debugging and configuring devices, especially in embedded systems and projects involving microcontrollers or other hardware components. Below is an explanation of how minicom can be used in a project:

3.2.1. Installation:

• Before using minicom, you need to install it on your system. You can typically install it using your system's package manager. For example, on a Debian-based system, you can use:

• bash

• sudo apt-get install minicom

3.2.2 Connecting to a Serial Port:

• Minicom is primarily used for serial communication, so you need to connect it to the serial port of the device you want to communicate with. Use the following command to open minicom:

• bash

• minicom -D /dev/ttyUSB0

• Here, /dev/ttyUSB0 is the path to the serial port. The actual port may vary depending on your system and the connected device.

3.2.3 Configuration:

• Once minicom is open, you may need to configure the serial port settings such as baud rate, data bits, stop bits, and parity. This is often necessary to match the settings of the device you are communicating with. You can access the configuration menu by pressing Ctrl-A followed by Z.

3.2.4. Interacting with the Device:

• After configuring the serial port, you can interact with the device. Minicom allows you to send commands and receive responses. This is particularly useful for debugging purposes and for configuring devices that have a serial console.

3.2.5. Exiting Minicom:

• To exit minicom, you can use the Ctrl-A followed by X shortcut. 17

3.2.6. File Transfer:

• Minicom also supports file transfer using protocols like Xmodem or Ymodem. This can be useful for updating firmware or transferring files between your computer and the connected device.

Example Workflow:

3.2.7. Connect to Device:

• Open minicom and connect to the serial port of your device.

3.2.8. Configure Settings:

• Configure the serial port settings to match the requirements of your device.

3.2.9. Interact and Debug:

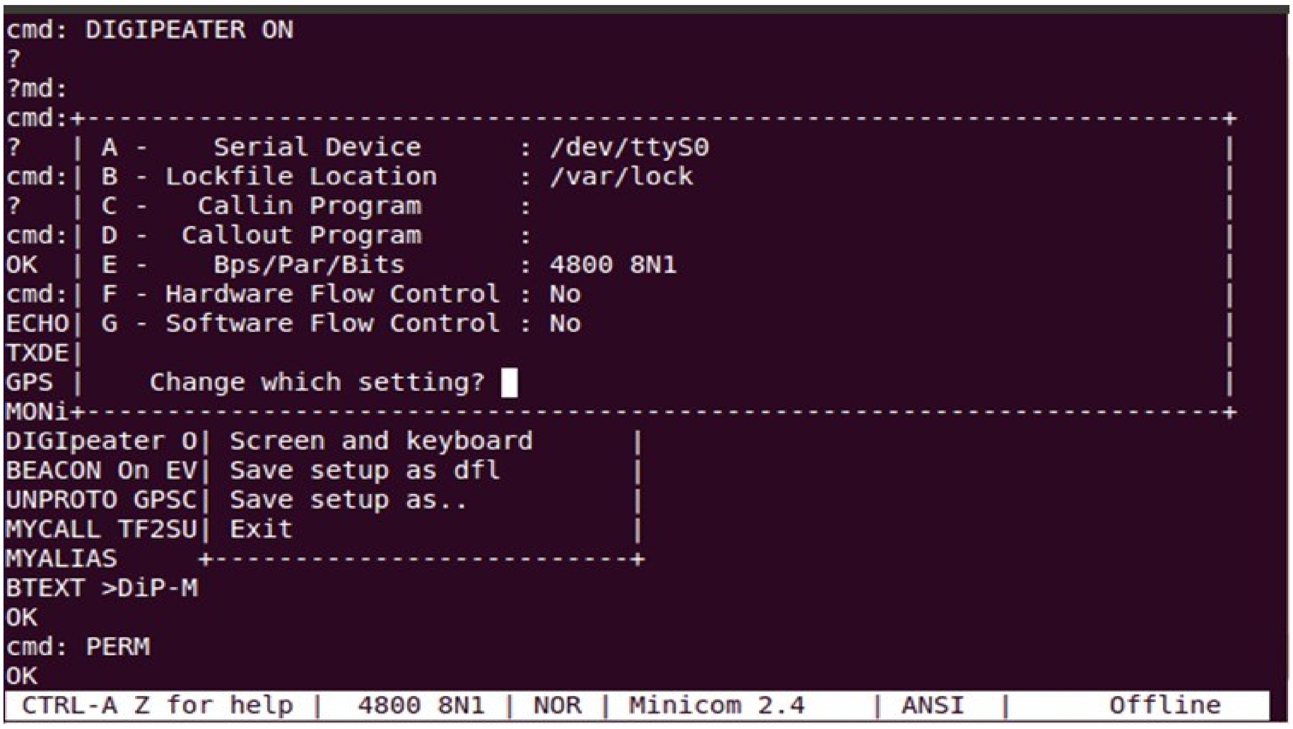
• Send commands and receive responses for debugging or configuring the device.

3.2.10.File Transfer (if needed):

• Use minicom's file transfer capabilities if you need to transfer files between your computer and the device.

3.2.11.Exit Minicom:

• When you are done, exit minicom.



```
cmd: DIGIPEATER ON
?
?md:
cmd:+-----------------------------------------------------------------------+
?   | A -    Serial Device      : /dev/ttyS0                               |
cmd:| B - Lockfile Location     : /var/lock                                |
?   | C -   Callin Program      :                                          |
cmd:| D -  Callout Program      :                                          |
OK  | E -    Bps/Par/Bits       : 4800 8N1                                 |
cmd:| F - Hardware Flow Control : No                                       |
ECHO| G - Software Flow Control : No                                       |
TXDE|                                                                      |
GPS |    Change which setting?                                             |
MONi+-----------------------------------------------------------------------+
DIGIpeater O| Screen and keyboard      |
BEACON On EV| Save setup as dfl        |
UNPROTO GPSC| Save setup as..          |
MYCALL TF2SU| Exit                     |
MYALIAS     +--------------------------+
BTEXT >DiP-M
OK
cmd: PERM
OK
 CTRL-A Z for help |  4800 8N1 | NOR | Minicom 2.4     | ANSI  |       Offline
```

3.3.RIGHTTECH IOT CLOUD :

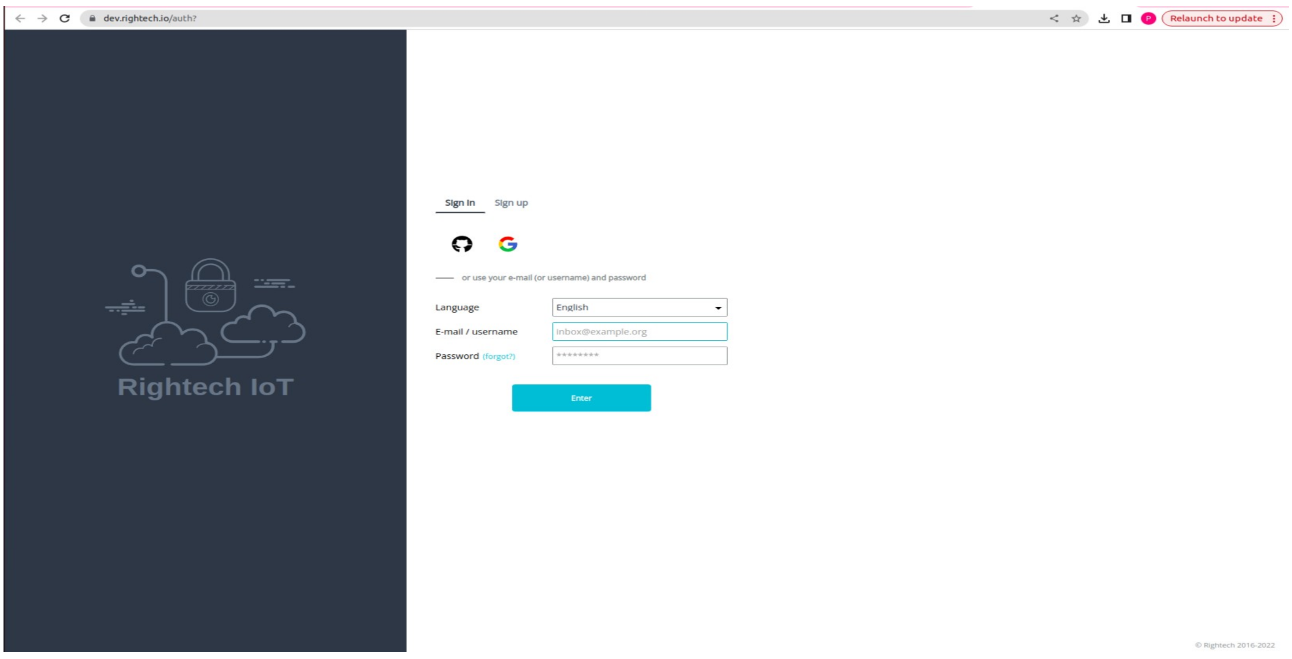
• Visit the "RightTech IoT" Website: Go to the official website or portal of "RightTech IoT."

• Registration: Look for a "Register" or "Sign Up" option on their website. Click on it to start the registration process.

• Fill Out Registration Form: Provide the required information, such as your name, email address, password, and any other details that the platform requests.

• Account Verification: Some platforms may require you to verify your email address by clicking on a verification link sent to your email. Complete the verification process if required.

• Log In: Once your registration is complete and your account is verified, log in to your "RightTech IoT" account using your credentials.



or use your e-mail (or username) and password

Language: English

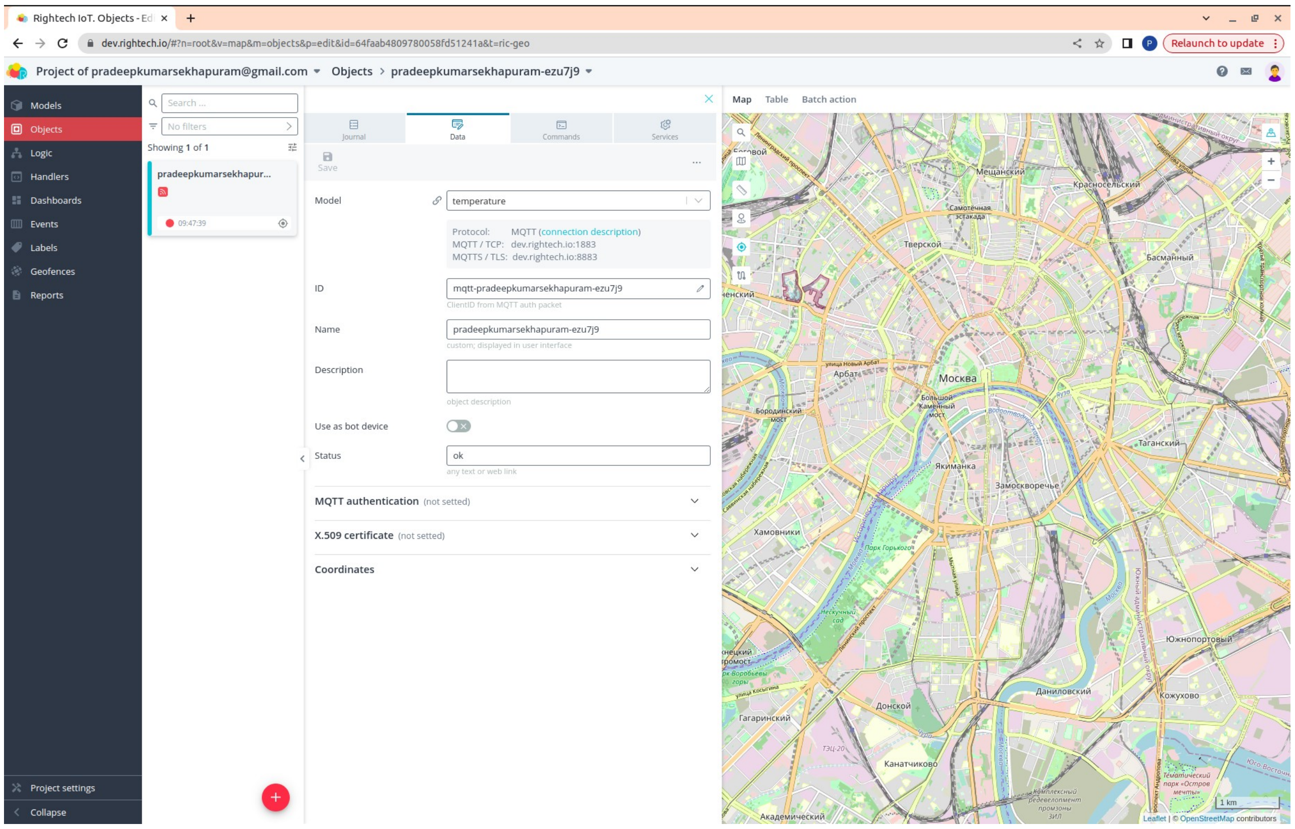
E-mail / username: inbox@example.org

Password (forgot?): ********

fig:Rightech IOT cloud login

• Explore the Platform: Navigate through the platform to understand its features, dashboard, and settings. You should look for an option related to creating or managing parameters, which are typically settings or values used to configure and control IoT devices or data.

• Create Parameters: Depending on the platform's interface and options, you may find a section where you can create parameters or configure settings for your IoT devices or applications. Follow the platform's instructions to create the parameters you need.


dev.rightech.io/#?n=root&v=map&m=objects&p=edit&id=64faab4809780058fd51241a&t=ric-geo

Project of pradeepkumarsekhapuram@gmail.com › Objects › pradeepkumarsekhapuram-ezu7j9

Model: temperature

Protocol: MQTT (connection description)
MQTT / TCP: dev.rightech.io:1883
MQTTS / TLS: dev.rightech.io:8883

ID: mqtt-pradeepkumarsekhapuram-ezu7j9
ClientID from MQTT auth packet

Name: pradeepkumarsekhapuram-ezu7j9
custom; displayed in user interface

Description:
object description

Use as bot device: off

Status: ok
any text or web link

MQTT authentication (not setted)

X.509 certificate (not setted)

Coordinates

fig:Rightech IOT cloud ID identification

4.PROJECT STAGES

STAGE:1

The STM32F446RE with KY-031 is used to communicate with the module and board. It is used to detect the tap.

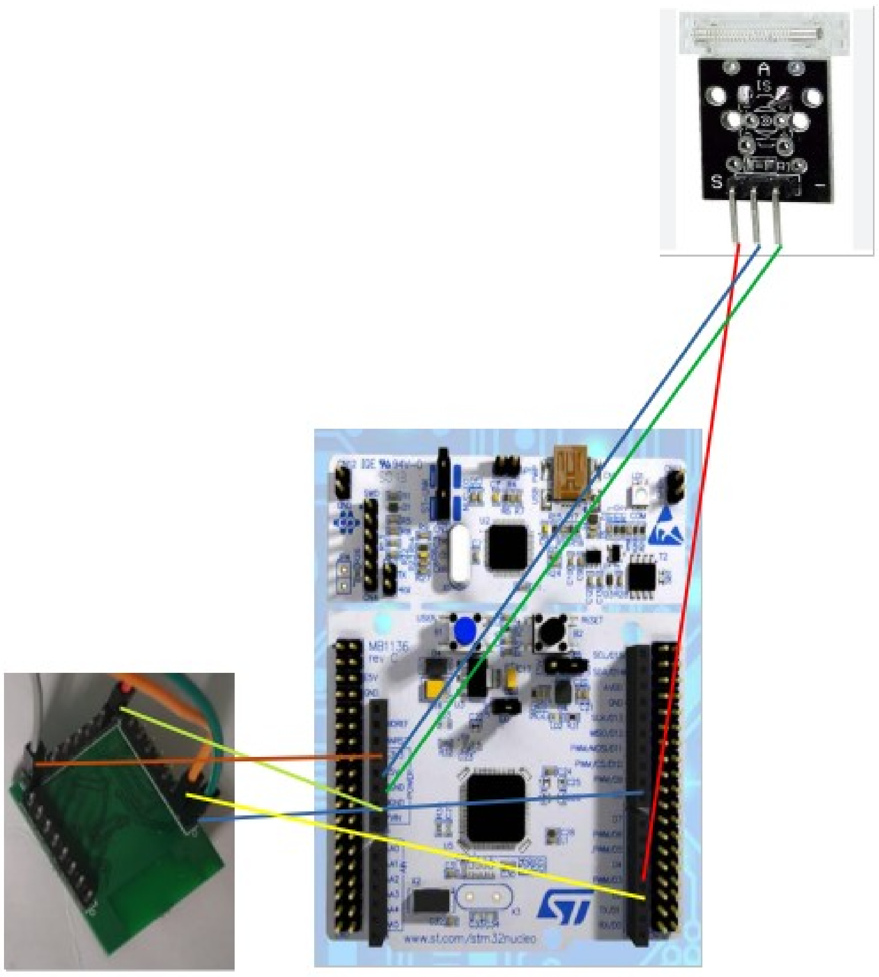
STAGE:2
The STM32F446RE with KY-031,W10 is used to transmit the data into the MQTT server to Publish the data in the Right-tech.

STAGE:3
The Ruggedboard with STM32F446RE is used to transmit the data and get the KY-031 value in the ruggedboard minicom.

STAGE:4
The Ruggedboard with STM32F446RE is used to transmit the data and get the KY-031 value in the ruggedboard minicom and pass the value into the Right-tech by using W10 module is connected with the Rugged board.

connection diagram

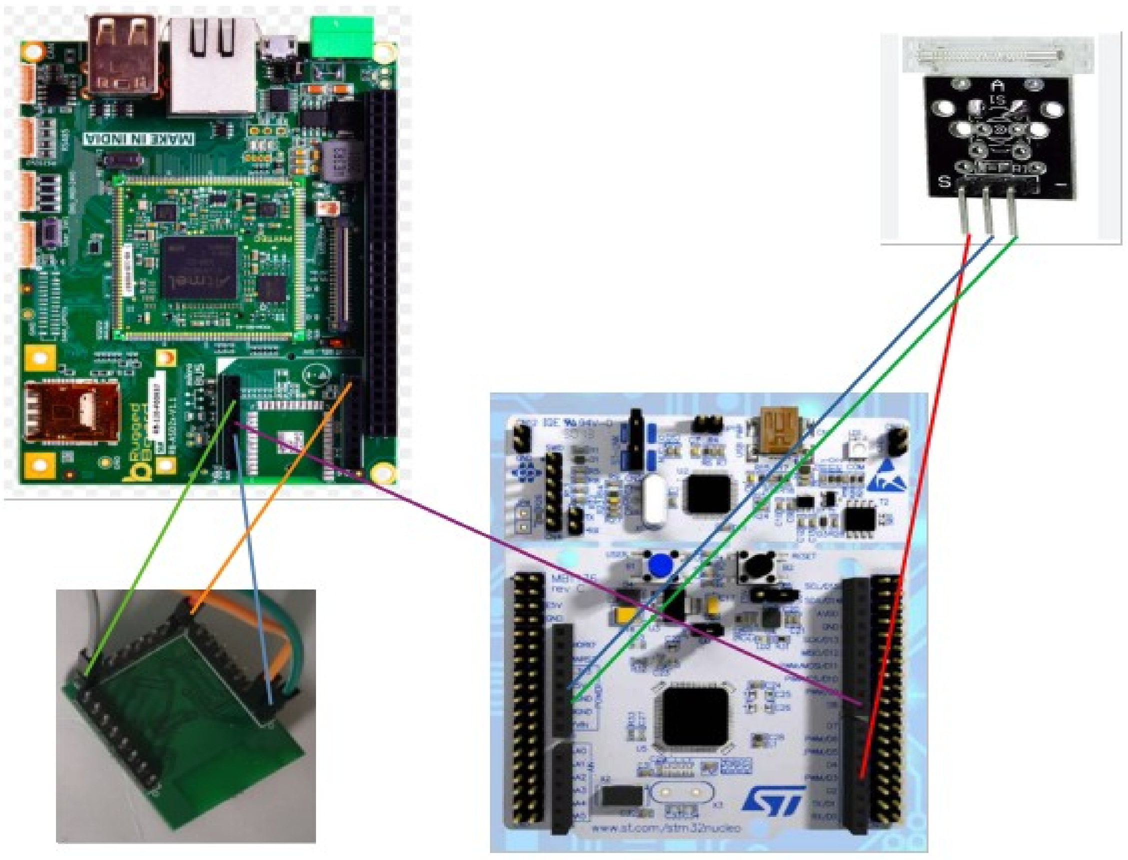
STAGE:2



pin connections

PIN	PIN NUMBER	COMPONENTS
VCC 5V	VCC	KY-031
GROUND	GND	W10 AND KY-031
SIGNAL	PB3	KY-031
VCC 3.3V	VCC	W10
Tx	PA9	W10
Rx	PA10	W10

STAGE:4



stm32f446re with ky-031:

PIN	PIN NUMBER	COMPONENTS
VCC 5V	VCC	KY-031
GROUND	GND	KY-031
SIGNAL	PB3	KY-031
Tx	PA9	Ruggedboard

In Ruggedboard with W10:

PIN	PIN NUMBER	COMPONENTS
VCC 3.3v	VCC	W10
GROUND	GND	W10
Tx	Tx	W10
Rx	Rx	stm32f446re(Tx)

5.Code snippet to wifi module initialization

```
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.

6.code snippet to 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-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.

7.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.

8.PROJECT CODE

STAGE 1:(STM32F446RE with KY-031)

```
#include "stm32f4xx_hal.h"
#include <stdbool.h>
#include <stdio.h>
#include <string.h>
#define LedPin GPIO_PIN_5
#define LedPort GPIOA
#define ShockPin GPIO_PIN_3
#define ShockPort GPIOB
UART_HandleTypeDef huart2;
UART_HandleTypeDef huart1;

void SystemClock_Config(void);
void Error_Handler(void);
bool tap_detected;
static void MX_USART2_UART_Init(void)
{
  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();
  }
}
```

```
int main(void)
{
    HAL_Init();
    SystemClock_Config();
    MX_USART2_UART_Init();
    HAL_InitTick(0);  // Initialize the HAL system tick

    // Initialize HAL library
    __HAL_RCC_GPIOA_CLK_ENABLE();
    __HAL_RCC_GPIOB_CLK_ENABLE();

    GPIO_InitTypeDef GPIO_InitStruct;
    GPIO_InitStruct.Pin = LedPin;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(LedPort, &GPIO_InitStruct);

    GPIO_InitStruct.Pin = ShockPin;
    GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
    GPIO_InitStruct.Pull = GPIO_PULLUP;
    HAL_GPIO_Init(ShockPort, &GPIO_InitStruct);
    char buffer[50];
    while (1)
      {
        /* USER CODE END WHILE */
      char buffer[100];

        /* USER CODE END WHILE */
          uint8_t val = HAL_GPIO_ReadPin(ShockPort,
ShockPin);

                if (val == GPIO_PIN_SET)
                {
                    HAL_GPIO_WritePin(LedPort, LedPin,
GPIO_PIN_RESET);
                    // Set the variable to 1 when the
sensor detects a signal
                    tap_detected = false;
                }
                else
                {
                    HAL_GPIO_WritePin(LedPort, LedPin,
GPIO_PIN_SET);
                    tap_detected = true;
                    HAL_Delay(2000);
                }

                if (tap_detected) {
                    sprintf(buffer, "Tap Detected: Yes\
r\n");
                    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);
                    sprintf(buffer, "Tap Detected: No\
r\n");
                    HAL_UART_Transmit(&huart1,
(uint8_t*)buffer, strlen(buffer), HAL_MAX_DELAY);
                    HAL_UART_Transmit(&huart2,
(uint8_t*)buffer, strlen(buffer), HAL_MAX_DELAY);

                }

      }
      /* USER CODE END 3 */
   }

void SystemClock_Config(void)
{
    RCC_OscInitTypeDef RCC_OscInitStruct = {0};
    RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

    __HAL_RCC_PWR_CLK_ENABLE();

__HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SC
ALE1);

    RCC_OscInitStruct.OscillatorType =
RCC_OSCILLATORTYPE_HSE;
    RCC_OscInitStruct.HSEState = RCC_HSE_ON;
    RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
    RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
    RCC_OscInitStruct.PLL.PLLM = 8;
    RCC_OscInitStruct.PLL.PLLN = 360;
    RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
    RCC_OscInitStruct.PLL.PLLQ = 7;
    if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
    {
        Error_Handler();
    }

    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_DIV4;
    RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;
    if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct,
FLASH_LATENCY_5) != HAL_OK)
    {
        Error_Handler();
    }
}

void Error_Handler(void)
{
    while (1)
    {
        // An error occurred, stay in this loop
    }
}
```

STAGE 2:(stm32f446re with w10 module)

```
#include "stm32f4xx_hal.h"
#include <stdbool.h>
#include <string.h>
#include <stdio.h>
#define LedPin GPIO_PIN_5
#define LedPort GPIOA
#define ShockPin GPIO_PIN_3
#define ShockPort GPIOB
```

```
UART_HandleTypeDef huart2;
UART_HandleTypeDef huart1;
void WE10_Init ();
void MQTT_Init();
void SystemClock_Config(void);
void Error_Handler(void);
bool tap_detected;
static void MX_USART1_UART_Init(void);
static void MX_USART2_UART_Init(void)
{
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 38400;
  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();
  }
}

void mqtt_data_send(uint16_t tap_detected)
{
    char buffer[50];
    sprintf (&buffer[0], "CMD+MQTTPUB=base/state/tap,%d\
r\n",tap_detected);
    HAL_UART_Transmit(&huart1, (uint8_t*)buffer,
strlen(buffer), 1000);
    HAL_Delay(100);

}
int main(void)
{
    HAL_Init();
    SystemClock_Config();
    MX_USART2_UART_Init();
    MX_USART1_UART_Init();
   HAL_InitTick(0);  // Initialize the HAL system tick
    WE10_Init();
    MQTT_Init();
    // Initialize HAL library
   __HAL_RCC_GPIOA_CLK_ENABLE();
    __HAL_RCC_GPIOB_CLK_ENABLE();

    GPIO_InitTypeDef GPIO_InitStruct;
    GPIO_InitStruct.Pin = LedPin;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(LedPort, &GPIO_InitStruct);

    GPIO_InitStruct.Pin = ShockPin;
    GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
    GPIO_InitStruct.Pull = GPIO_PULLUP;
    HAL_GPIO_Init(ShockPort, &GPIO_InitStruct);
    //char buffer[50];
    while (1)
    {
      /* USER CODE END WHILE */
      char buffer[100];

        /* USER CODE END WHILE */
          uint8_t val = HAL_GPIO_ReadPin(ShockPort,
ShockPin);

                if (val == GPIO_PIN_SET)
                {
                    HAL_GPIO_WritePin(LedPort, LedPin,
GPIO_PIN_RESET);
                    // Set the variable to 1 when the
sensor detects a signal
                    tap_detected = false;
                }
                else
                {
                    HAL_GPIO_WritePin(LedPort, LedPin,
GPIO_PIN_SET);
                    tap_detected = true;
                    HAL_Delay(2000);
                }

                if (tap_detected) {
                    sprintf(buffer, "Tap Detected: Yes\
r\n");
                    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);
                    sprintf(buffer, "Tap Detected: No\
r\n");
                    HAL_UART_Transmit(&huart1,
(uint8_t*)buffer, strlen(buffer), HAL_MAX_DELAY);
                    HAL_UART_Transmit(&huart2,
(uint8_t*)buffer, strlen(buffer), HAL_MAX_DELAY);

                }

    }
}

void SystemClock_Config(void)
{
    RCC_OscInitTypeDef RCC_OscInitStruct = {0};
    RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

    __HAL_RCC_PWR_CLK_ENABLE();

__HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SC
ALE1);

    RCC_OscInitStruct.OscillatorType =
RCC_OSCILLATORTYPE_HSE;
    RCC_OscInitStruct.HSEState = RCC_HSE_ON;
    RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
    RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
    RCC_OscInitStruct.PLL.PLLM = 8;
    RCC_OscInitStruct.PLL.PLLN = 360;
    RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
    RCC_OscInitStruct.PLL.PLLQ = 7;
    if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
    {
        Error_Handler();
    }
    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_DIV4;
    RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;
    if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct,
FLASH_LATENCY_5) != HAL_OK)
    {
        Error_Handler();
    }
}

void Error_Handler(void)
{
    while (1)
    {
        // An error occurred, stay in this loop
    }
}

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=SBCS.2.4.GHz,SBCS@1234\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];

	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);
	//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);


	sprintf (&buffer[0], "CMD+MQTTCONCFG=3,mqtt-
pradeepkumarsekhapuram-ezu7j9,,,,,,,,,\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_Delay(500);
	HAL_UART_Transmit(&huart2, (uint8_t*)buffer,
strlen(buffer), 1000);


	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);
	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);

    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);

}




static void MX_USART1_UART_Init(void)
{
  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.Mode = UART_MODE_TX_RX;
  huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart1.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart1) != HAL_OK)
  {
      while (1);
  }
}
```

STAGE:3(Ruggedboard with STM32F446RE)

#include<stdio.h>

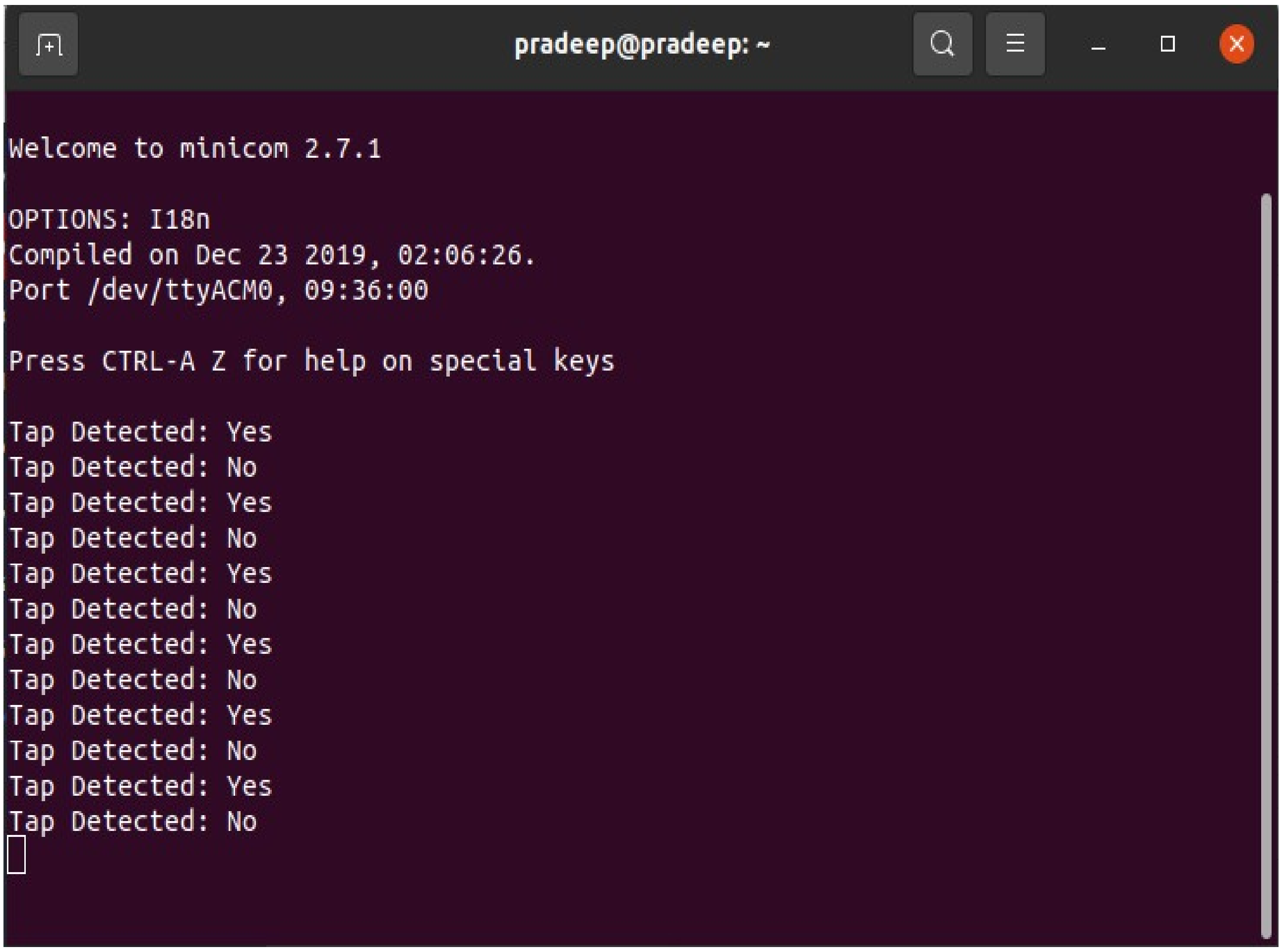
```
#include<string.h>
#include<errno.h>
#include<stdlib.h>

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=\"Pradeep\",\"pradeep123\"\r\n";
    char arr3[] = "CMD+MQTTNETCFG=dev.rightech.io,1883\r\n";
    char arr4[] = "CMD+MQTTCONCFG=3,mqtt-pradeepkumarsekhapuram-
ezu7j9,,,,,,,,,\r\n";
    char arr5[] = "CMD+MQTTSTART=1\r\n";
    char arr6[] = "CMD+MQTTSUB=base/state/tap\r\n";
//    char arr7[] = "CMD+MQTTPUB=base/state/tap\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/tap,%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;
}
```

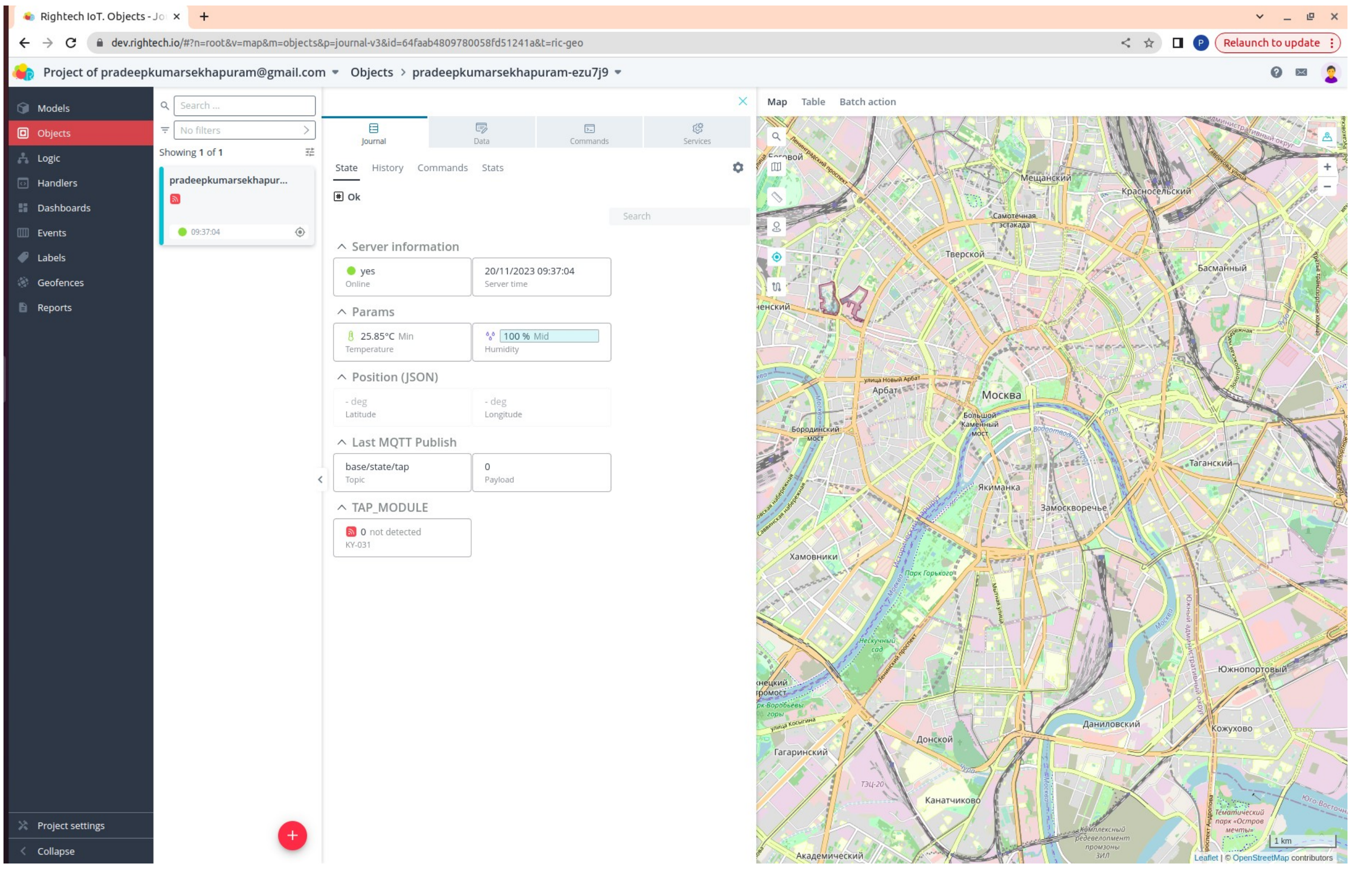
9.MINICOM OUTPUT



```
Welcome to minicom 2.7.1

OPTIONS: I18n
Compiled on Dec 23 2019, 02:06:26.
Port /dev/ttyACM0, 09:36:00

Press CTRL-A Z for help on special keys

Tap Detected: Yes
Tap Detected: No
Tap Detected: Yes
Tap Detected: No
Tap Detected: Yes
Tap Detected: No
Tap Detected: Yes
Tap Detected: No
Tap Detected: Yes
Tap Detected: No
Tap Detected: Yes
Tap Detected: No
```

When a tap is detected by the tap module KY-031, the LED in the microcontroller illuminates, indicating that the tap has been detected, and the message "Tap Detected: Yes" is displayed in Minicom. Otherwise, if no tap is detected, the message "Tap Detected: No" is shown in Minicom.

10.RIGHT TECH OUTPUT



Ok

Server information

yes Online	20/11/2023 09:37:04 Server time

Params

25.85°C Min Temperature	100 % Mid Humidity

Position (JSON)

- deg Latitude	- deg Longitude

Last MQTT Publish

base/state/tap Topic	0 Payload

TAP_MODULE

0 not detected KY-031

After establishing a connection to RightTec through the functions WE10_Init and MQTT_Init, and subsequently using mqtt_data_send in the code, tapping the sensor triggers the message "Tap Detected: 1." In contrast, if no tap is detected, the message "Tap Detected: 0" is displayed.

11.Applications

- Home Automation
- Electronic Drums or Percussion Instruments
- Interactive Art Installations
- Smart Furniture
- Gesture Control
- Health Monitoring Devices
- Industrial Automation

- Sports Equipment Monitoring
- Interactive Displays
- Custom Alarm Systems

12.References

https://arduinomodules.info/ky-031-knock-sensor-module/

https://datasheetspdf.com/pdf/1402041/Joy-IT/KY-031/1

https://sensorkit.joy-it.net/en/sensors/ky-031