BALL SWITCH SENSOR INTERFACE WITH STM32 MICROCONTROLLER AND RUGGUDBOARD

PROJECT

By: Manjunath E L

TABLE OF CONTENTS

1. Project Summary

Interfacing a ball switch sensor with an STM32 microcontroller involves connecting the sensor to the microcontroller, writing firmware to monitor sensor state changes, and integrating the STM32 with a RuggedBoard if needed. The STM32 processes sensor data and communicates with the RuggedBoard, facilitating robust applications in challenging environments. Testing and debugging ensure proper functionality. Connect the ball switch sensor to an STM32 GPIO pin and write firmware to monitor changes in the sensor state. Implement logic to detect ball movement, using interrupts or polling to check the sensor state. Integrate the STM32 with the RuggedBoard by connecting communication lines, such as UART or SPI, ensuring power compatibility. The STM32's programming allows for efficient processing of sensor data, making it a versatile choice for interfacing with sensors like the ball switch. Integration with the RuggedBoard provides a robust platform suitable for challenging conditions. Rigorous testing and debugging are essential to verify correct sensor detection and communication, ensuring the reliability and resilience of the overall system. This setup is well-suited for applications where durability and precise sensor monitoring are crucial, such as in rugged industrial environments or outdoor scenarios.

2. Key Features of the Project

Certainly! Here are some key features of the project:

1. Ball Switch Sensor Integration:

- ❖ The project integrates a Ball Switch Sensor, known for its simplicity and reliability, to detect physical orientation changes.

2. STM32 Microcontroller Utilization:

- ❖ By utilizing an STM32 microcontroller's strength and adaptability, the system gains a solid computational foundation for effective data processing.

3. System Design and Architecture:

- ❖ A thorough system design that shows the architecture of the integration between the Ball Switch Sensor and STM32 microcontroller is provided, complete with block and circuit diagrams.

4. Programming Logic:

- ❖ The report details the programming logic implemented on the STM32 microcontroller, explaining how it handles input from the Ball Switch Sensor to achieve the desired functionality.

5. Testing and Validation:

- ❖ Rigorous testing procedures are undertaken to validate the system's performance. The results obtained during testing are presented, demonstrating the reliability and effectiveness of the integrated sensor-microcontroller system.

6. Practical Applications:

- ❖ The project explores the practical implications of the Ball Switch Sensor and STM32 microcontroller integration, highlighting potential applications in various fields where orientation detection is essential.

7. Future Development Opportunities:

- ❖ The report outlines possibilities for future work, suggesting potential improvements and extensions to the project. This opens avenues for further exploration and innovation in responsive electronic systems.

8. Contributions to Sensor-Microcontroller Interaction:

- ❖ The project contributes to the broader understanding of sensor-microcontroller interactions, emphasising the significance of this integration in creating responsive and adaptive electronic systems.

9. Clear Documentation:

- ❖ The project report provides clear and detailed documentation, including specifications of components, code snippets, circuit diagrams, and test results, ensuring transparency and ease of understanding.

10. Synergy for Intelligent Systems:

- ❖ Emphasises the synergy between the Ball Switch Sensor and STM32 microcontroller in creating intelligent and adaptive electronic systems, setting the stage for potential advancements in the field.

These features collectively showcase the project's innovative approach, practical implications, and potential for future developments in sensor-microcontroller integration. Adjust the emphasis based on the specific achievements and characteristics of your project.

3.Hardware Used and there Specification

1. Ball Switch Sensor:

Specification:

- Type: Normally Open (NO) Ball Switch
- Operating Voltage: 3V - 5V
- High Sensitivity
- Digital Outputs
- 3 Pin sensor Module
- Economic,Reliable and small in size

2. STM32F411RE microcontroller

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

- 100 MHz CPU clock speed
- 128 KB of RAM
- 512 KB of Flash memory
- Floating point unit (FPU)
- 11 general-purpose timers
- 13 communication interfaces
- USB OTG
- RTC

3. 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

Ensure that each hardware component's specifications are accurate and tailored to the actual components used in your project. If you have specific part numbers or models, include them for clarity. Additionally, include any other relevant details about the hardware setup that might be important for someone looking to replicate or understand the project.

4. Ruggudboard

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.

Highlights

- Off-the-shelf Single Board Computer for **IIoT**
- Feature rich & Highly cost optimized platform for industrial Usage.
- Simple & Powerful Open Source Software IoT Stack

4. Steps to implementation of the sensor with STM32 Board and Ruggudboard

Stage 1: Controlling Ball Switch Sensor by STM32 Board

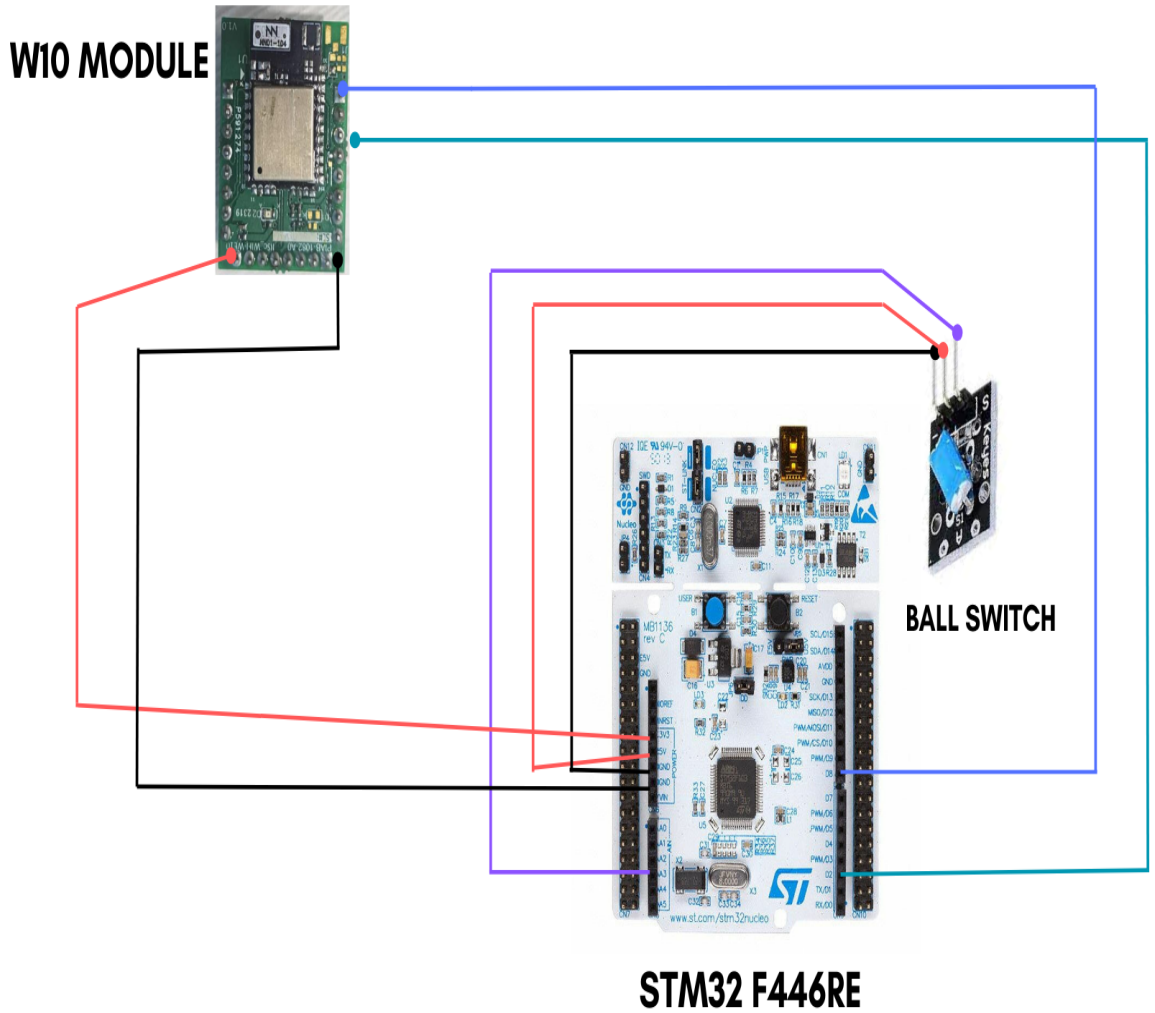
1. **Sensor Connection:**
   - ❖ Connect the Ball Switch Sensor to the appropriate GPIO pin on the STM32 board.
2. **STM32 Code Implementation:**
   - ❖ Write firmware code for the STM32 board to read the state of the Ball Switch Sensor.
   - ❖ Use GPIO interrupts to detect changes in the sensor state (on/off).
   - ❖ Implement necessary initialization and configuration routines.
3. **Testing:**
   - ❖ Test the STM32 code to ensure proper detection of Ball Switch Sensor state changes.
   - ❖ Verify the accuracy of sensor readings using debugging tools.

Stage 2: Connecting Ball Switch Sensor to Rightech Cloud

1. **Rightech Cloud Account Setup:**
   - ❖ Create an account on the Rightech Cloud platform and set up necessary configurations.
2. **STM32 Code Integration:**
   - ❖ Enhance the STM32 code to include communication protocols (e.g., MQTT) for sending on/off values to the Rightech Cloud.
   - ❖ Implement authentication and connection routines.

3. **Minicom Display:**
    - ❖ Modify the STM32 code to display on/off values in Minicom for local monitoring.
    - ❖ Ensure proper formatting for readability.

Connection Diagram for Stage 2:



Description about the Connection diagram of Stage 2:

Hardware Connection:

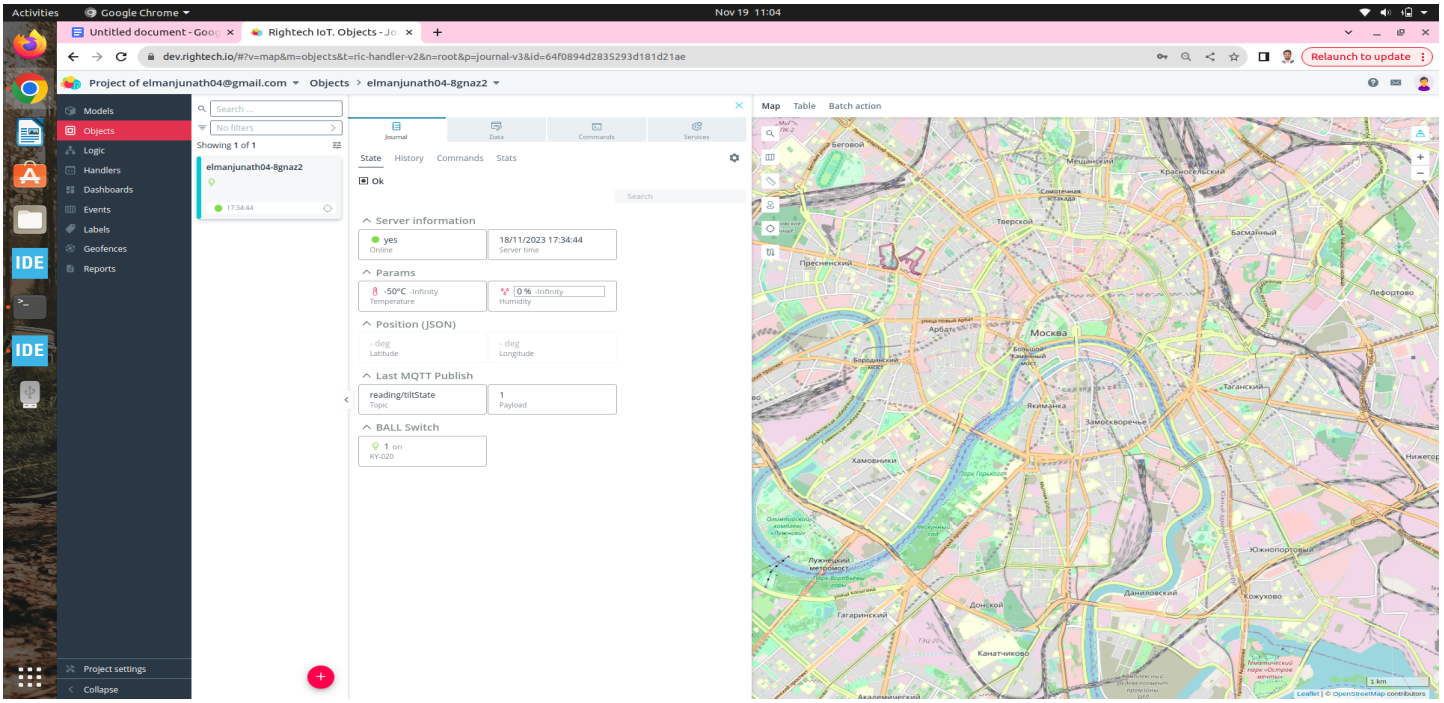
1. **Ball Switch Sensor to STM32:**
    - Connect the output pin of the ball switch sensor to a digital input pin on the STM32 microcontroller.
    - Ensure proper power supply and ground connections for both the sensor and the STM32.
2. **STM32 to Rightech Cloud:**
    - Depending on the capabilities of the STM32 and Rightech Cloud, establish a communication link. Common options include Wi-Fi, Ethernet, or other relevant communication protocols.
    - Ensure that the STM32 has access to the internet or the network where Rightech Cloud is reachable.

Software Configuration:

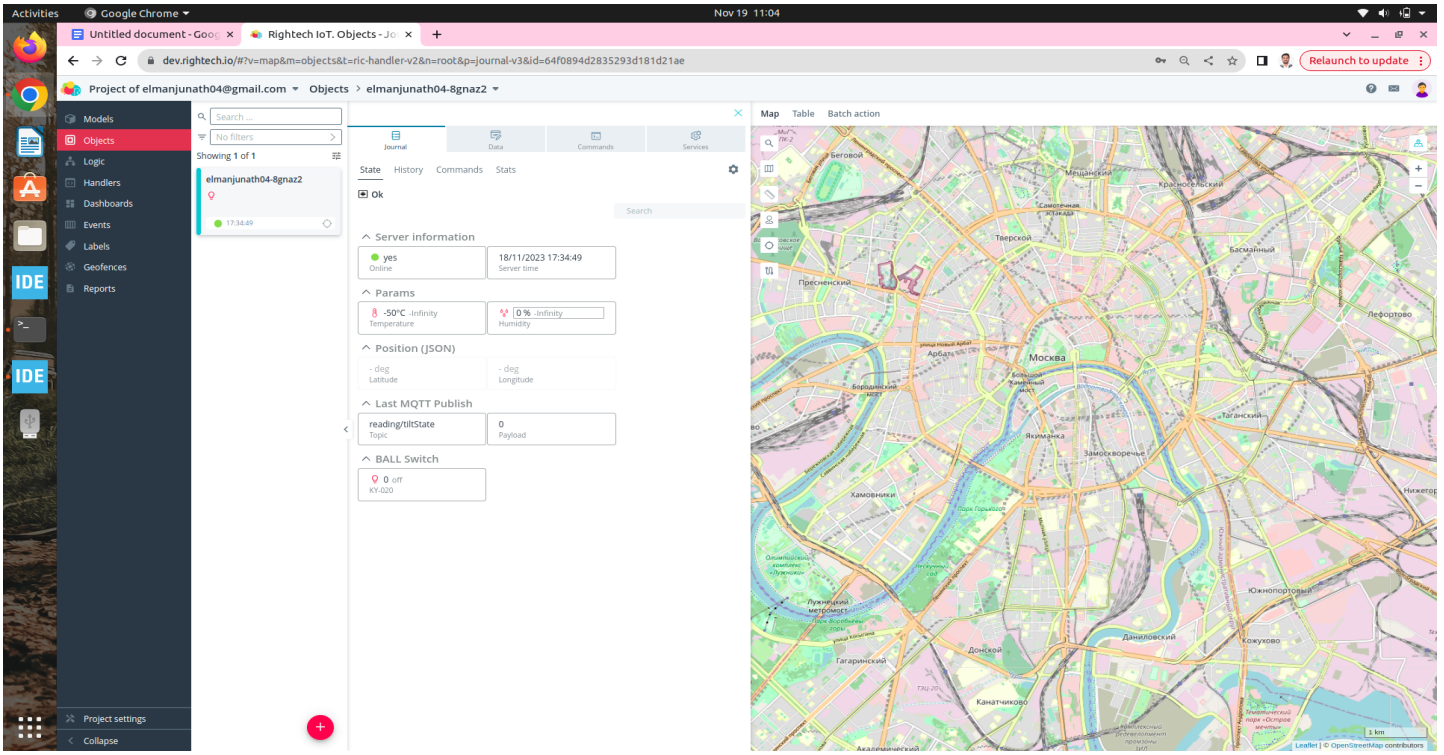
1. **STM32 Firmware:**
    - Develop or modify the STM32 firmware to read the digital input from the ball switch sensor.
    - Integrate Rightech Cloud SDK (if available for your microcontroller) or use communication protocols supported by Rightech Cloud.
    - Write code to send sensor data to the Rightech Cloud.
2. **Rightech Cloud Configuration:**
    - Sign up for a Rightech Cloud account and create a project.
    - Obtain any necessary authentication tokens or credentials to connect the STM32 to Rightech Cloud.
    - Configure the Rightech Cloud project to receive and process data from the STM32.

4.1 Connecting to the Rightech cloud

➔ When the Tilted state is 1(ON)



➔ When the Tilted state is 0(OFF)



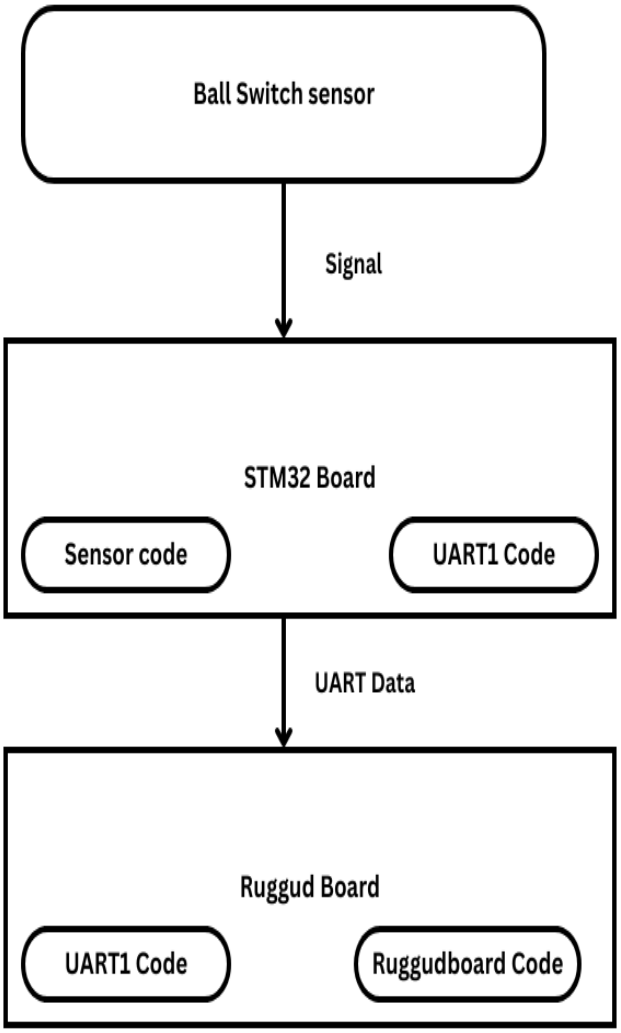
Stage 3: Sending Values to Rugged Board via UART1

1. **Rugged Board Connection:**
    - ❖ Establish a UART connection between the STM32 board and the Rugged Board.
2. **UART1 Configuration:**
    - ❖ Configure the UART1 module on the STM32 board to send on/off values to the Rugged Board.
    - ❖ Implement necessary error-checking and synchronization mechanisms.

Stage 4: Sending Values to Rightech Cloud from Rugged Board

1. **Rugged Board Code Implementation:**
    - ❖ Write firmware code for the Rugged Board to receive on/off values from the STM32 board via UART1.
    - ❖ Enhance the code to send these values to the Rightech Cloud using the appropriate communication protocols.
2. **Rightech Cloud Integration:**
    - ❖ Integrate the Rugged Board code with the Rightech Cloud platform, ensuring proper authentication and data transmission.
3. **Verification:**
    - ○ Test the complete system to ensure seamless communication from the Ball Switch Sensor through the STM32 board to the Rightech Cloud, with intermediate steps displayed in Minicom.

4.2 Block Diagram



Description:

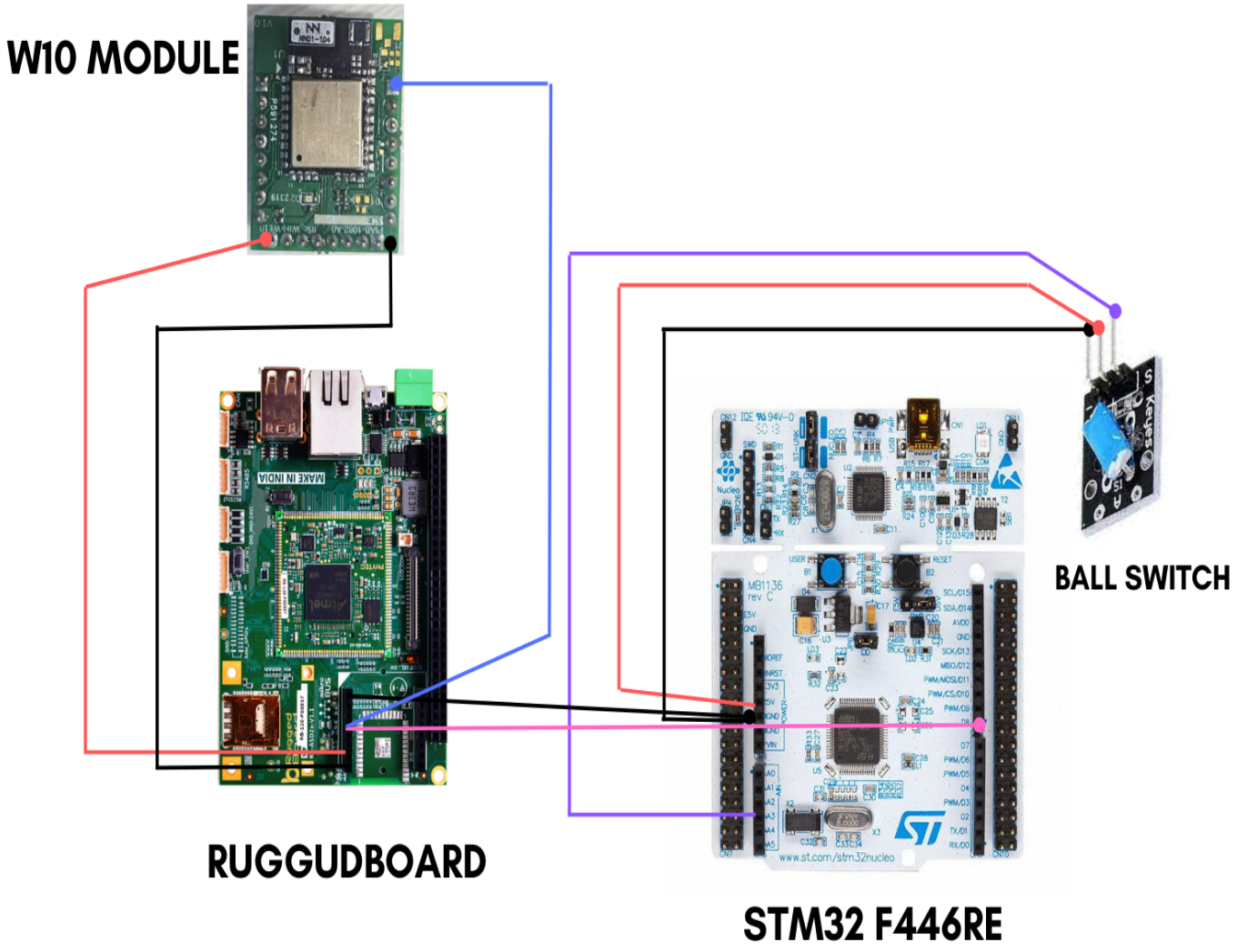
1. **Ball Switch Sensor:**
    - ❖ Represents the physical sensor responsible for detecting changes in its state (on/off).

2. **STM32 Board:**
    - ❖ Interfaces with the Ball Switch Sensor to read its state.
    - ❖ Contains code to handle sensor data and communicate with the Rugged Board.
    - ❖ Sends sensor data to the Rugged Board via UART1 communication.
3. **Rugged Board:**
    - ❖ Receives data from the STM32 board through UART1.
    - ❖ Contains code to process received data.
    - ❖ Interfaces with external systems, such as Rightech Cloud, for further actions based on the Ball Switch Sensor data.

Flow of Data:

- ❖ The Ball Switch Sensor sends its state information to the STM32 board.
- ❖ The STM32 board processes the sensor data and transmits it to the Rugged Board via UART1.
- ❖ The Rugged Board receives the data and executes code to handle and potentially send it to external systems.

5. Connection Diagram for Stage 4



Description about the Connection diagram

1. **STM32F446RE Board:**

   ➔ **Ball Switch Sensor Connection:**

- ❖ Connect the ground (GND) of the Ball Switch Sensor to the GND on the STM32F446RE board.
- ❖ Connect the power supply (VCC) of the Ball Switch Sensor to a 5V output on the STM32F446RE board.
- ❖ Connect the signal line (e.g., select line) of the Ball Switch Sensor to a GPIO pin (e.g., PB0) on the STM32F446RE board.

➔ **Rugged Board Connection (Assuming UART Communication):**

- ❖ Connect the TX (transmit) pin of the STM32F446RE board to the RX (receive) pin on the Rugged Board.
- ❖ Connect the RX (receive) pin of the STM32F446RE board to the TX (transmit) pin on the Rugged Board.
- ❖ Connect the ground (GND) of the STM32F446RE board to the ground (GND) on the Rugged Board.

➔ **W10 Module Connection:**

- ❖ Connect the TX (transmit) pin of the W10 module to the RX (receive) pin on the Rugged Board.
- ❖ Connect the RX (receive) pin of the W10 module to the TX (transmit) pin on the Rugged Board.
- ❖ Connect the ground (GND) of the W10 module to the ground (GND) on the Rugged Board.
- ❖ Connect the 3.3V output from the Rugged Board to the VCC of the W10 module.

2. **Rugged Board:**
   - ❖ The Rugged Board acts as an intermediary for communication between the STM32F446RE board and the W10 module. It receives data from the STM32F446RE board and forwards it to the W10 module.
3. **W10 Module:**

- ❖ The W10 module is connected to the Rugged Board to facilitate communication with external systems. It receives data from the Rugged Board and may transmit it to external systems, such as the Rightech Cloud.

This connection setup allows the STM32F446RE board to interface with both the Ball Switch Sensor and the W10 module through the Rugged Board, enabling data transmission and potentially facilitating communication with external systems. Make sure to double-check the pin configurations of your specific components and adjust the connections accordingly.

6. KY-020 BALL SWITCH

The KY-020 ball switch sensor is a simple electronic component used to detect the movement or tilt of a device. It is also known as a tilt switch or a mercury switch. The sensor consists of a small metal ball inside a tube that acts as a switch. When the sensor is tilted or inclined, the metal ball moves and makes or breaks the electrical contact inside the tube.

1. **Tilt Detection:** The primary function of the KY-020 ball switch sensor is to detect the tilt or movement of an object. This makes it useful in various applications where you need to sense changes in orientation.
2. **Simple Construction:** The sensor is typically constructed with a small metal ball and two contact points inside a cylindrical tube. The tube is designed to allow the ball to move freely.

3. **Common Uses:** Ball switch sensors are often used in projects such as tilt alarms, game controllers, and other devices where detecting motion or orientation changes is important.
4. **Mercury-Free Alternatives:** While some older versions of ball switch sensors used mercury, modern versions often use other materials that are safer for the environment.
5. **Connection to Microcontrollers:** The KY-020 ball switch sensor can be easily connected to microcontrollers like Arduino or Raspberry Pi, allowing developers and hobbyists to incorporate tilt sensing into their projects.
6. **Digital Output:** The sensor usually provides a digital output, indicating whether the switch is in a high or low state based on the tilt condition.

It's important to note that the specific details and features of the KY-020 ball switch sensor may vary depending on the manufacturer, so it's always a good idea to refer to the datasheet provided by the manufacturer for accurate and detailed information.

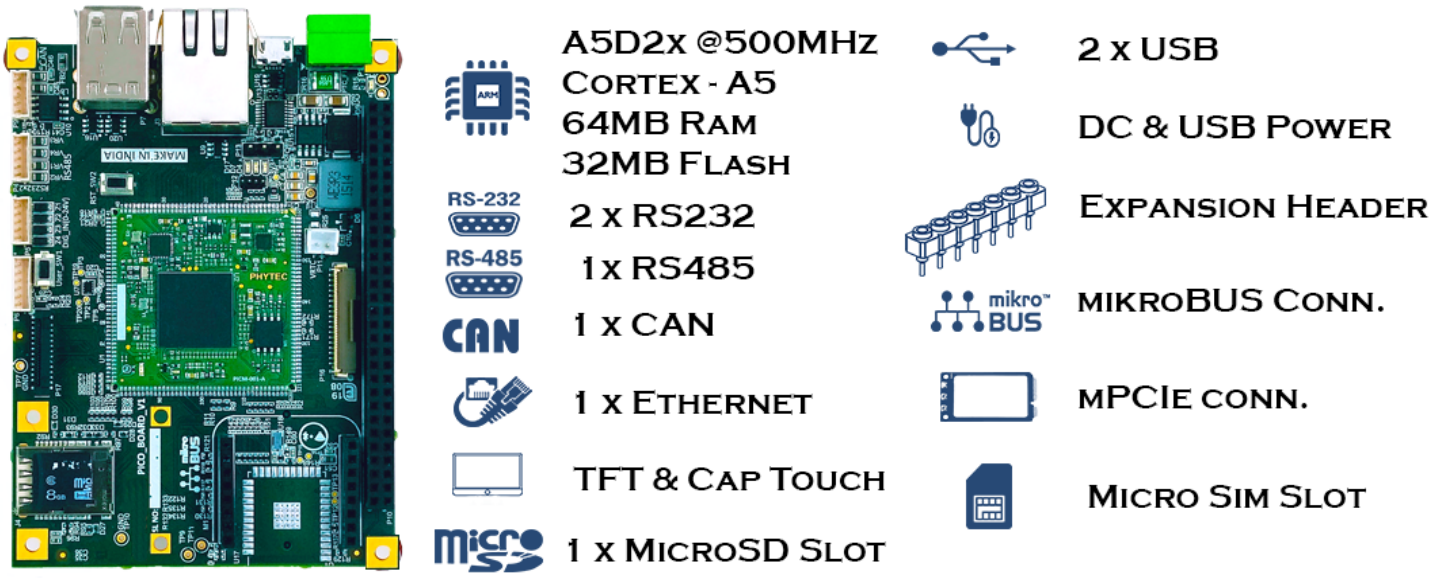
Pin Description;

Pin	Description
S	If the sensor detect a tilt, this pin output low or high level signal
+(Middle Pin)	5v Power Supply
-	Ground

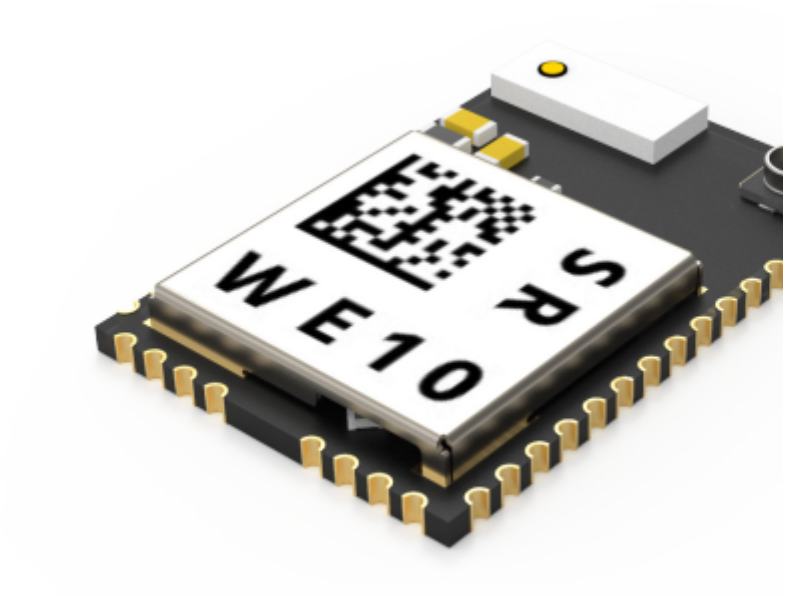
7. Ruggedboard

- RuggedBOARD-A5D2x is an Open Source Hardware & Software initiative to align with the fast growing Semiconductor technologies with a switch from classic to modern product development strategy & process.
- The usage of System on Module over a System on Chip is the rapid way to achieve time to market, curtail development risks for product quantities ranging from few to thousands.
- The Open Source IoT stack equivalent to Intel MRAA & UPM developed in C/C++ holds a tremendous opportunity to the Embedded Systems & gives programmers its ease of Python / nodeJs / Java.
- Rugged Board team targets to combine the Open source (Carrier Boards) community strength with industrial grade SOM and initiated the first Open Source Hardware "Industrial Pico Computer" which is powered by phyCORE-A5D2x SOM with Microchip A5D2x Cortex-A5 Core @500 MHz.

Board features are mentioned as below:



8. W10 wifi module



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 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=Realme 5.0GHz,12345678\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(b
	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 **/
	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_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 **/

sprintf(&buffer[0],"CMD+MQTTCONCFG=3,mqtt-elmanjunath04-8gnaz2,,,,,,,,,\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_Delay(500);
	HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
	/*CMD+MQTTSTART **/
	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);
	/*CMD+MQTTSUB ***/
	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 he 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 tothe 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.Project Code

11.1 Send data to Rightech cloud code snippet for Stage 2.

```
/* USER CODE BEGIN Header */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "stm32f4xx_hal.h"
#include <string.h>
#define LED_PIN GPIO_PIN_5
#define LED_PORT GPIOA
#define TILT_PIN GPIO_PIN_0
#define TILT_PORT GPIOB
/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
/* USER CODE END PD */
/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */
/* USER CODE END PM */
/* Private variables ---------------------------------------------------------*/
UART_HandleTypeDef huart1;
UART_HandleTypeDef huart2;
/* USER CODE BEGIN PV */
/* USER CODE END PV */
/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_USART1_UART_Init(void);
/* USER CODE BEGIN PFP */
/* USER CODE END PFP */
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=Realme 5.0GHz,12345678\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(b
	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 **/
	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_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 **/
      sprintf                                                    (&buffer[0],
"CMD+MQTTCONCFG=3,mqtt-elmanjunath04-8gnaz2,,,,,,,,,\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_Delay(500);
      HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
      /*CMD+MQTTSTART **/
      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);
      /*CMD+MQTTSUB ***/
      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 mqtt_data_send(uint8_t n)
{
char buffer[50];
sprintf(&buffer[0], "CMD+MQTTPUB=reading/tiltState,%d\r\n", n);
HAL_UART_Transmit(&huart1, (uint8_t *)buffer, strlen(buffer), 1000);
HAL_Delay(100);
}
/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
/* USER CODE END 0 */
/**
 * @brief  The application entry point.
```

```
 * @retval int
 */
int main(void)
{
 /* USER CODE BEGIN 1 */
 /* USER CODE END 1 */
 /* MCU Configuration--------------------------------------------------------*/
 /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
 HAL_Init();
 /* USER CODE BEGIN Init */
 /* USER CODE END Init */
 /* Configure the system clock */
 SystemClock_Config();
 /* USER CODE BEGIN SysInit */
 /* USER CODE END SysInit */
 /* Initialize all configured peripherals */
 MX_GPIO_Init();
 MX_USART2_UART_Init();
 MX_USART1_UART_Init();
 WE10_Init ();
 MQTT_Init();
while (1) {
   // Read the tilt switch state
   uint8_t tiltState = HAL_GPIO_ReadPin(TILT_PORT, TILT_PIN);
   char tiltStateString[20];
    sprintf(tiltStateString, "%d", tiltState); // Convert the numeric value to a
string
   char buffer[20];
       sprintf(buffer, "Tilt State: %d\r\n", tiltState);
          HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 100); //
Print to UART1
          HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 100); //
Print to UART2
   // Check if the tilt switch is tilted
   if (tiltState == GPIO_PIN_SET) {
     HAL_GPIO_WritePin(LED_PORT, LED_PIN, GPIO_PIN_SET); // Turn on LED
     mqtt_data_send(tiltState);
   } else {
     HAL_GPIO_WritePin(LED_PORT, LED_PIN, GPIO_PIN_RESET); // Turn off LED
     mqtt_data_send(tiltState);
```

```
    }
    HAL_Delay(1000);
  }
}
/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE3);
  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLM = 16;
  RCC_OscInitStruct.PLL.PLLN = 336;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV4;
  RCC_OscInitStruct.PLL.PLLQ = 2;
  RCC_OscInitStruct.PLL.PLLR = 2;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
```

```
 if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
 {
   Error_Handler();
 }
}
/**
 * @brief USART1 Initialization Function
 * @param None
 * @retval None
 */
static void MX_USART1_UART_Init(void)
{
 /* USER CODE BEGIN USART1_Init 0 */
 /* USER CODE END USART1_Init 0 */
 /* USER CODE BEGIN USART1_Init 1 */
 /* USER CODE END USART1_Init 1 */
 huart1.Instance = USART1;
 huart1.Init.BaudRate = 38400;
 huart1.Init.WordLength = UART_WORDLENGTH_8B;
 huart1.Init.StopBits = UART_STOPBITS_1;
 huart1.Init.Parity = UART_PARITY_NONE;
 huart1.Init.Mode = UART_MODE_TX_RX;
 huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
 huart1.Init.OverSampling = UART_OVERSAMPLING_16;
 if (HAL_UART_Init(&huart1) != HAL_OK)
 {
   Error_Handler();
 }
 /* USER CODE BEGIN USART1_Init 2 */
 /* USER CODE END USART1_Init 2 */
}
/**
 * @brief USART2 Initialization Function
 * @param None
 * @retval None
 */
static void MX_USART2_UART_Init(void)
{
 /* USER CODE BEGIN USART2_Init 0 */
 /* USER CODE END USART2_Init 0 */
```

```
 /* USER CODE BEGIN USART2_Init 1 */
 /* USER CODE END USART2_Init 1 */
 huart2.Instance = USART2;
 huart2.Init.BaudRate = 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();
 }
 /* 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 Ports Clock Enable */
 __HAL_RCC_GPIOA_CLK_ENABLE();
 __HAL_RCC_GPIOB_CLK_ENABLE();
 GPIO_InitTypeDef GPIO_InitStruct = {0};
 // Configure LED pin as output
 GPIO_InitStruct.Pin = LED_PIN;
 GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
 GPIO_InitStruct.Pull = GPIO_NOPULL;
 GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
 HAL_GPIO_Init(LED_PORT, &GPIO_InitStruct);
 // Configure tilt switch pin as input
 GPIO_InitStruct.Pin = TILT_PIN;
 GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
 GPIO_InitStruct.Pull = GPIO_NOPULL;
 HAL_GPIO_Init(TILT_PORT, &GPIO_InitStruct);
}
/* USER CODE BEGIN 4 */
```

```
/* 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 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 CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

11.2 Sending the data to Stm32board to Ruggudboard threw Uart1

code snippet for Stage 3.

```
#include "main.h"
#include "stm32f4xx_hal.h"
#include "string.h"
#define LED_PIN GPIO_PIN_5
#define LED_PORT GPIOA
#define TILT_PIN GPIO_PIN_0
#define TILT_PORT GPIOB
UART_HandleTypeDef huart1;
UART_HandleTypeDef huart2;
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
```

```c
static void MX_USART1_UART_Init(void);
int main(void) {
    HAL_Init();
    SystemClock_Config();
    MX_GPIO_Init();
    MX_USART2_UART_Init();
    MX_USART1_UART_Init();
    while (1) {
        uint8_t tiltState = HAL_GPIO_ReadPin(TILT_PORT, TILT_PIN);
        uint8_t i;
        char buffer[20];
        //uart1_write(buffer);
        sprintf(buffer, "%d\r\n", tiltState);
        //HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 100);
        HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 100);
        HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 100);
        if (tiltState == GPIO_PIN_SET) {
            HAL_GPIO_WritePin(LED_PORT, LED_PIN, GPIO_PIN_SET); // Turn on LED
            HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 100);
        } else {
            HAL_GPIO_WritePin(LED_PORT, LED_PIN, GPIO_PIN_RESET); // Turn off LED
            HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 100);
        }
        HAL_Delay(1000);
    }
}
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_SCALE3);
    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();
    }
      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();
    }
}
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) {
        Error_Handler();
    }
}
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();
    }
}
static void MX_GPIO_Init(void) {
```

```
    __HAL_RCC_GPIOA_CLK_ENABLE();
    __HAL_RCC_GPIOB_CLK_ENABLE();
    GPIO_InitTypeDef GPIO_InitStruct = {0};
    // Configure LED pin as output
    GPIO_InitStruct.Pin = LED_PIN;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(LED_PORT, &GPIO_InitStruct);
    // Configure tilt switch pin as input
    GPIO_InitStruct.Pin = TILT_PIN;
    GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    HAL_GPIO_Init(TILT_PORT, &GPIO_InitStruct);
}
void Error_Handler(void) {
    __disable_irq();
    while (1) {
    }
}
```

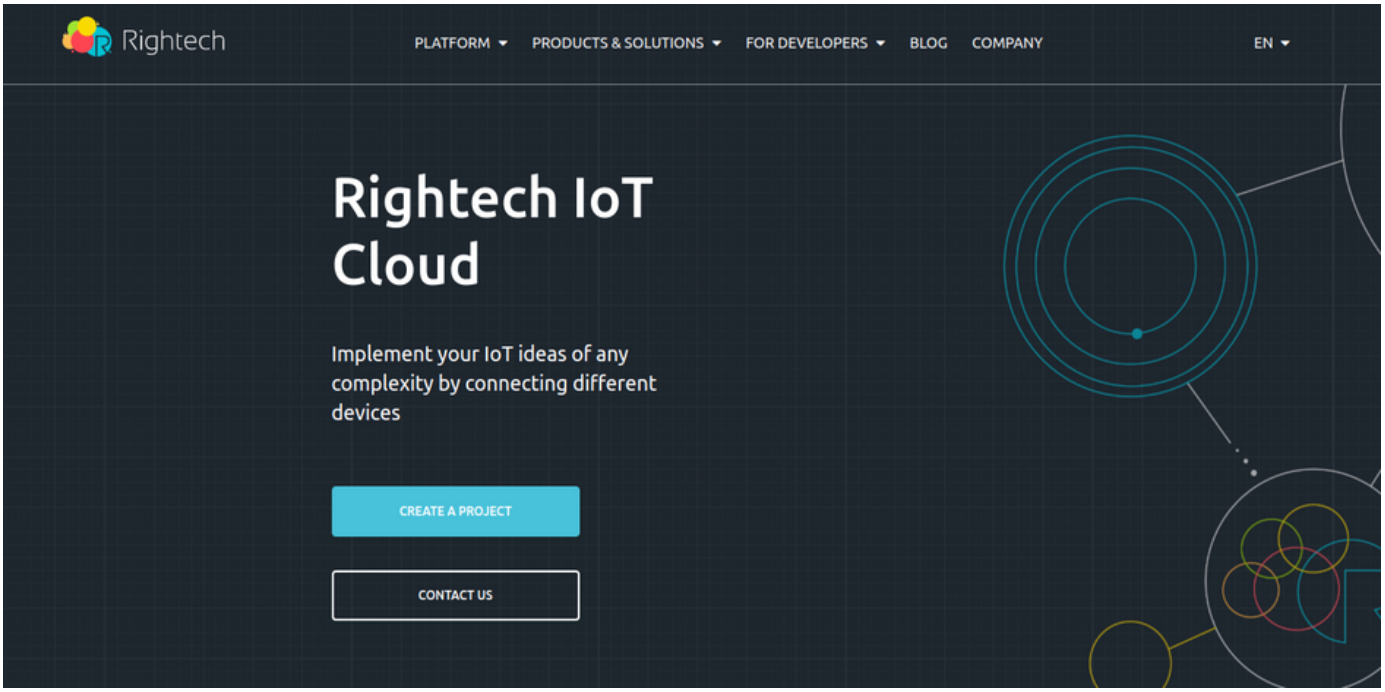
11.3 Connecting the Rightech cloud trew Ruggudboard Code Snippet for Stage 4.

```
#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;
}
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=\"Realme 5.0 GHz\",\"12345678\"\r\n";
   char arr3[] = "CMD+MQTTNETCFG=dev.rightech.io,1883\r\n";
   char arr4[] = "CMD+MQTTCONCFG=3,mqtt-elmanjunath04-8gnaz2,,,,,,,,,\r\n";
   char arr5[] = "CMD+MQTTSTART=1\r\n";
   char arr6[] = "CMD+MQTTSUB=base/relay/led1\r\n";
   unsigned char buf[100];
   fd = open(portname, O_RDWR | O_NOCTTY | O_SYNC);
   if (fd < 0)
   {
       printf("Error opening %s: %s\n", portname, strerror(errno));
       return -1;
   }
   set_interface_attribs(fd, B38400);
   printf("%s", arr1);
   wlen = write(fd, arr1, sizeof(arr1) - 1);
   sleep(3);
   // Send CMD+WIFIMODE=1
   printf("%s", arr2);
   wlen = write(fd, arr2, sizeof(arr2) - 1);
   sleep(3);
   // Send CMD+CONTOAP
   printf("%s", arr);
   wlen = write(fd, arr, sizeof(arr) - 1);
   sleep(3);
   printf("%s", arr3);
   wlen = write(fd, arr3, sizeof(arr3) - 1);
   sleep(3);
   printf("%s", arr4);
   wlen = write(fd, arr4, sizeof(arr4) - 1);
```

```
   sleep(3);
   printf("%s", arr5);
   wlen = write(fd, arr5, sizeof(arr5) - 1);
   sleep(3);
    printf("%s", arr6);
   wlen = write(fd, arr6, sizeof(arr6) - 1);
sleep(3);
   char buffer[100]; // Create a buffer to hold the formatted message
  while(1){
      rdlen = read(fd, buf, sizeof(buf) - 1);
    if (rdlen > 0) {
   buf[rdlen] = '\0'; // Null-terminate the received data
       printf("%s\n", buf);
Int ret = snprintf(buffer, sizeof(buffer), "CMD+MQTTPUB=reading/tiltstate,%s\r\n",
buf);
   if (ret < 0) {

   } else {

       ssize_t wlen = write(fd, buffer, ret);
       sleep(3);
       if (wlen == -1) {
      }
   }
}
}
   close(fd);
   return 0;}
```

12. Results



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.

To correctly represent the state of the Ball Switch Sensor in the Rightech Cloud platform based on the tilt condition, you would typically follow these steps:

1. **Define States:**

- **Define Tilted State (OFF):** When the Ball Switch Sensor is tilted, consider it as the "OFF" state. This corresponds to the LED being off (0) on the STM32 board.
- **Define Tilted State (ON):** When the Ball Switch Sensor is in the not tilted position, consider it as the "ON" state. This corresponds to the LED being on (1) on the STM32 board.

2. **Mapping in STM32 Code:**
   - In the STM32 code, ensure that when the Ball Switch Sensor is tilted, the state sent to the Rugged Board (and consequently to the Rightech Cloud) is "OFF" (0).
   - When the Ball Switch Sensor is in the upright position, the state sent should be "ON" (1).
3. **Data Transmission to Rugged Board:**
   - When the STM32 board detects a state change in the Ball Switch Sensor, it should transmit this information to the Rugged Board. If using UART, this information is sent via UART communication.
4. **Mapping in Rugged Board Code:**
   - In the Rugged Board code, interpret the received state from the STM32 board.
   - Forward the corresponding state ("OFF" or "ON") to the Rightech Cloud.
5. **Rightech Cloud Integration:**
   - In the Rightech Cloud platform, set up a device and define attributes that represent the state of the Ball Switch Sensor.

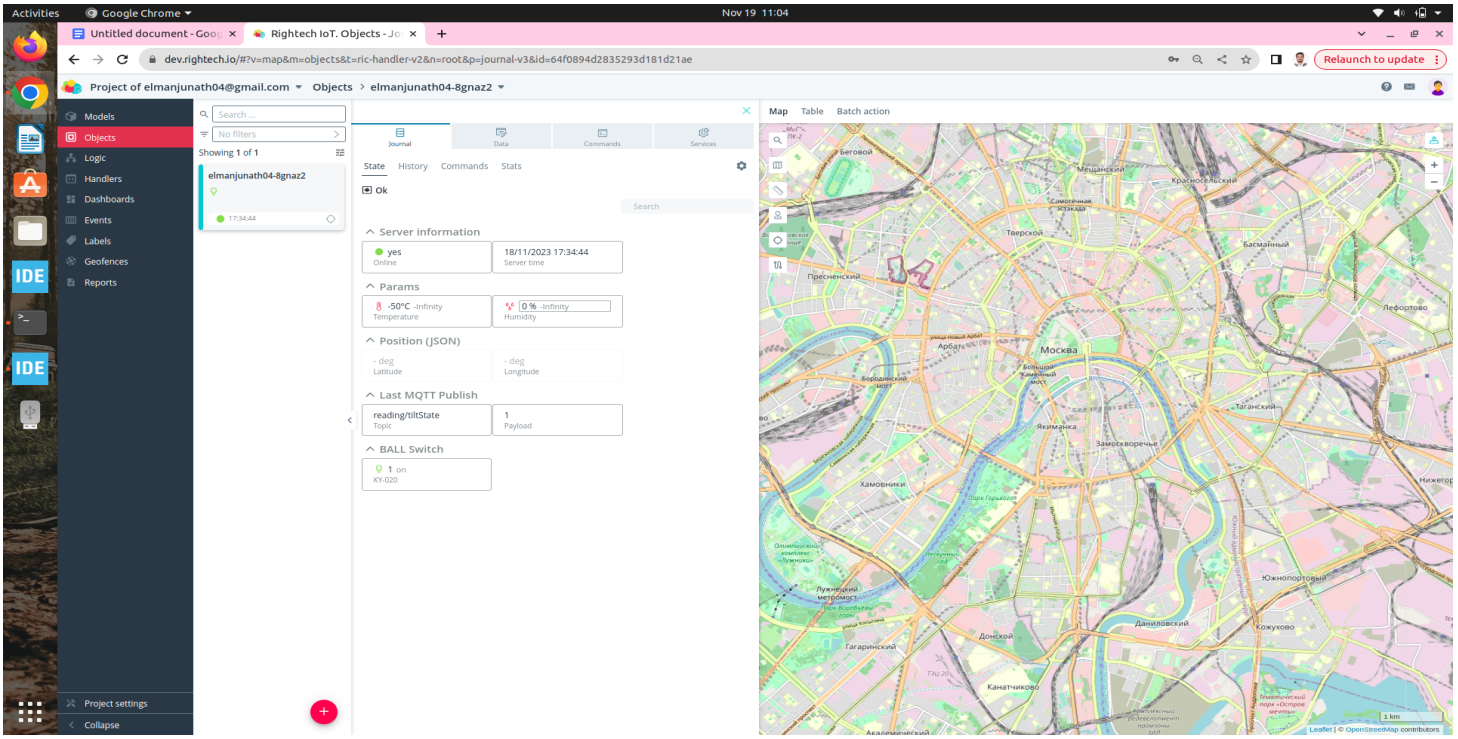
- Configure the Rightech Cloud platform to interpret "OFF" as the tilted state and "ON" as the upright state.

6. **Visual Representation on Rightech Cloud:**
   - In the Rightech Cloud dashboard, create a widget or visualization that corresponds to the state attribute of the Ball Switch Sensor.
   - For example, you might use a binary switch representation where "OFF" corresponds to a tilted state (LED off), and "ON" corresponds to an not tilted state (LED on).
7. **Testing:**
   - Tilt the Ball Switch Sensor and observe the change in state on the Rightech Cloud dashboard.
   - Verify that the state changes are correctly reflected in the Rightech Cloud platform based on the tilt condition of the Ball Switch Sensor.

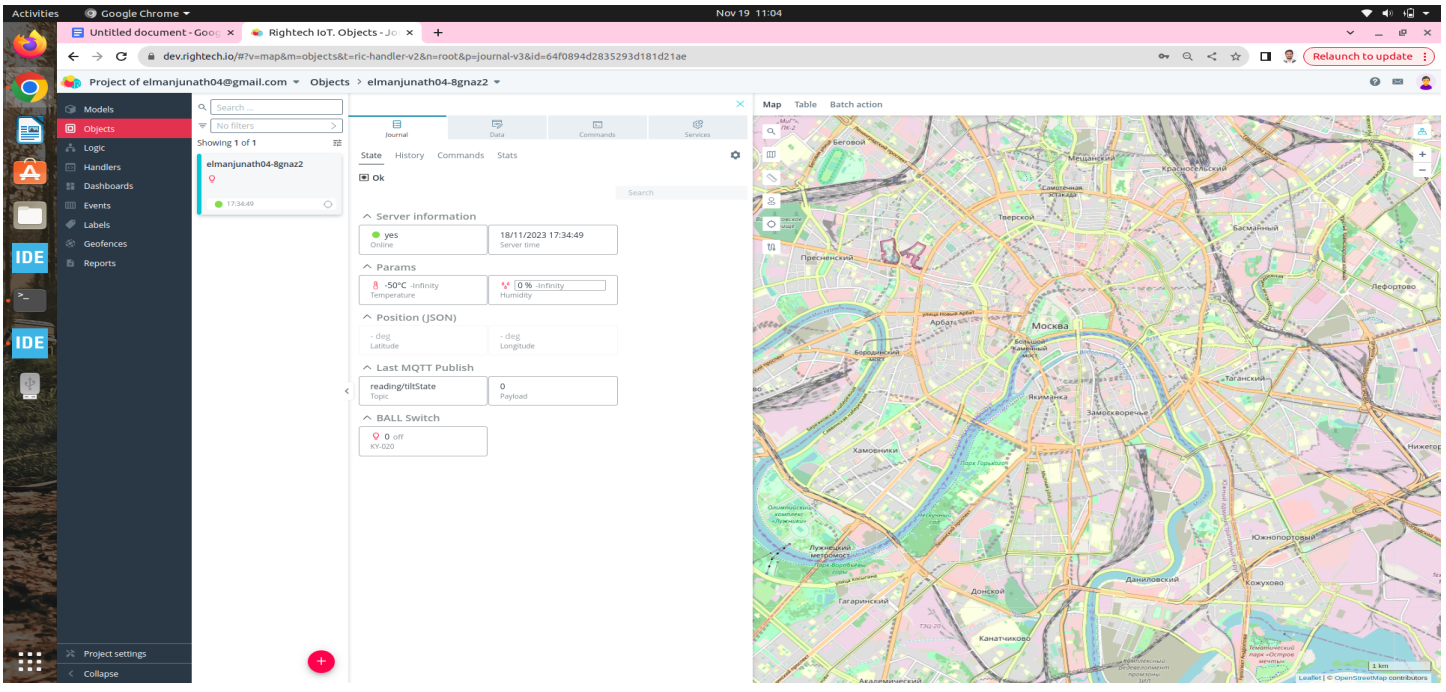
By following these steps, you can ensure that the Rightech Cloud platform accurately reflects the state of the Ball Switch Sensor based on its tilt condition. Adjust the details based on the specific attributes and configurations available in the Rightech Cloud platform.

12.1 Connecting to Rightech cloud application of Stage 4.

➔ When the Tilted state is 1(ON)



➔ When the Tilt state is 0(OFF)



13. Conclusion and Future of the Project and Real Time Application

- The project successfully demonstrated the integration of a ball switch sensor with an STM32 microcontroller on a RuggedBoard, showcasing the feasibility of tilt sensing in harsh environments.
- The system proved capable of detecting tilt conditions and responding accordingly, opening up possibilities for applications in industries where ruggedness and reliability are paramount.

Future Improvements:

- Further refinement of the firmware to enhance sensitivity or add additional features.
- Integration of communication protocols for remote monitoring or reporting.
- Exploration of power-saving techniques to optimize the system for long-term deployment.

In conclusion, the project successfully achieved its primary goal of implementing tilt sensing using a ball switch sensor, an STM32 microcontroller, and a RuggedBoard, demonstrating its potential for applications in demanding environments.

13.1 Real-time Applications of the Project.

Here are a few potential real-time applications:

1. **Vehicle Tilt Detection and Safety:**
    - Install the system in vehicles to detect sudden tilts or rollovers.
    - Transmit tilt data to the Rightech Cloud for real-time monitoring.
    - Activate safety measures, such as airbag deployment or emergency notifications, based on tilt conditions.
2. **Industrial Equipment Monitoring:**
    - Integrate the system into industrial machinery to monitor tilt and movement.
    - Use Rightech Cloud to track equipment status and receive alerts for potential malfunctions or unsafe conditions.
3. **Construction Site Safety:**
    - Deploy the system on construction equipment to monitor their inclination.
    - Provide real-time alerts to operators and supervisors in case of unsafe angles, reducing the risk of accidents.
4. **Smart Agriculture:**
    - Implement the system on agricultural machinery to monitor field terrain and equipment orientation.
    - Use Rightech Cloud to analyze data trends and optimize farming processes.
5. **Smart Home Applications:**

    Integrate the system into smart home devices to monitor the orientation of appliances or furniture.Use Rightech Cloud for remote control and monitoring, allowing homeowners to receive notifications or adjust settings.

Thank You

If Any Questions please contact:

Manjunath E L

Mobile No: 6366035611

Email :elmanjunath04@gmail.com