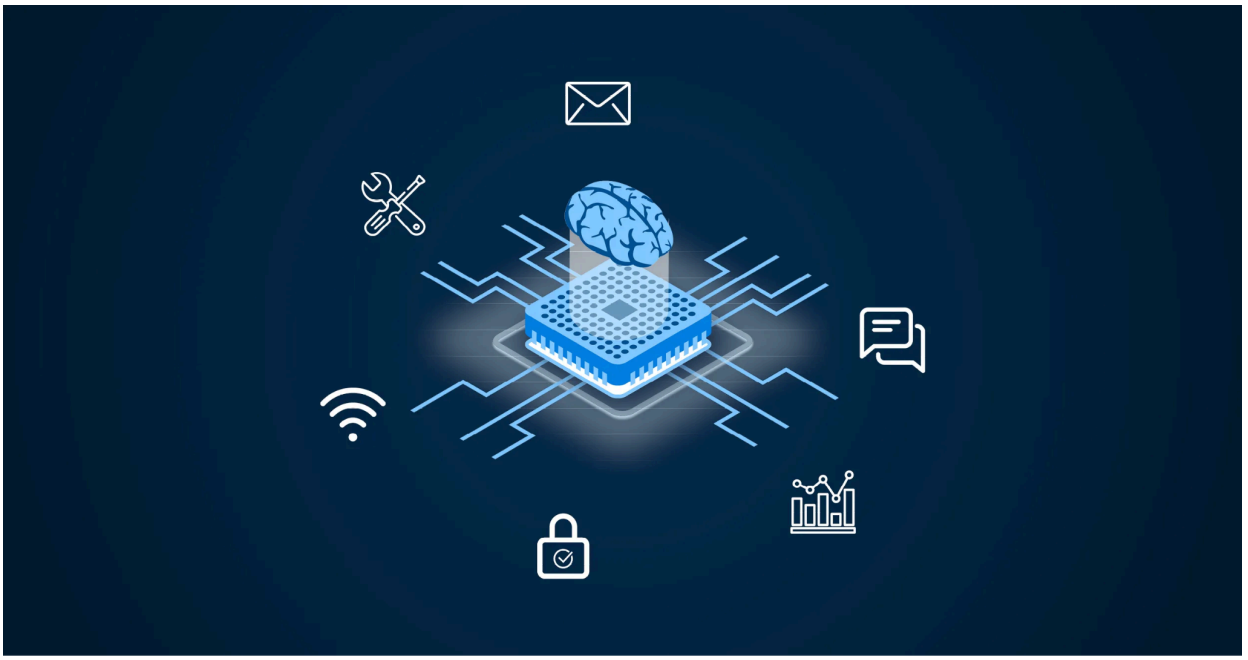
BE33 MODULE INTERFACED WITH STM32 MICROCONTROLLER AND RUGGEDBOARD AND SENDING DATA TO CLOUD



Presented by:

Manjunath E L

Pradeep.S

Pavithra.R

TABLE OF CONTENTS

1.PROJECT SUMMARY

2. KEY FEATURES OF THE PROJECT

3.HARDWARE USED AND THERE SPECIFICATIONS

3.1 BE33 BLUETOOTH MODULE

3.2 USB to TTL CONVERTER

3.3 RUGGED BOARD

3.4 STM32 MICROCONTROLLER

3.5 AHT25 TEMPERATURE SENSOR

3.6 W10 WIFI MODULE

4. STAGES TO IMPLEMENT BE33 MODULE

Stage 1: Sending data threw USB to TTL Converter using BE33 MODULE

Stage 2: Board to Board Communication using BE33 module and sending data from one board to another(RBA5D2X).

Stage 3: Sending data from STM32 microcontroller to RBA5D2X over the BE33 Module.

Stage 4: Sending AHT25 temperature sensor values from STM32 microcontroller to RBA5D2X.

Stage 5: Sending data to Rightech cloud Platform

5. PROJECT CODES

5.1 Code snippet for the BE33 Module for STM32

5.2 Code Snippet for the stage 2 (RBA5D2X TO RBA5D2X)

5.3 Code snippet for the Stage3 for STM32 Microcontroller.

5.4 Code snippet for the Stage3 for RBA5D2X

5.5 Code snippet for the Stage 4 for STM32 Microcontroller.

5.5.1 Code snippet for the Stage 4 for RBA5D2X

5.6 Code snippet for the Stage 5 for RBA5D2X to send the data to cloud.

6.RESULT AND CONCLUSION

1.PROJECT SUMMARY :

The project involves the integration of a BE33 Bluetooth module with a TTL (Transistor-Transistor Logic) interface, a rugged board (A5D2x), and an STM32 microcontroller to establish a robust and versatile wireless communication system. The BE33 Bluetooth module serves as the wireless communication bridge, enabling seamless data exchange between the rugged board and external devices. The TTL interface facilitates the connection between the BE33 module and the rugged board, allowing for reliable UART-based communication. The rugged board, A5D2x, with its durable and ruggedized design, enhances the project's suitability for deployment in challenging environments. The STM32 microcontroller acts as the central processing unit, managing the communication between the Bluetooth module and the rugged board while providing additional processing capabilities. This integrated system is designed to support various applications, such as IoT deployments and industrial automation, where wireless communication, ruggedness, and microcontroller processing are critical. The project aims to offer a comprehensive solution for wireless data transfer and control, leveraging the capabilities of the BE33 Bluetooth module, TTL interface, rugged board, and STM32 microcontroller in a synergistic manner.

This comprehensive project leverages the BE33 Bluetooth module, TTL interface, rugged board A5D2x, and an STM32 microcontroller to create a robust and adaptable wireless communication system. The BE33 Bluetooth module facilitates seamless wireless connectivity, allowing for efficient data exchange between the rugged board and external devices. The integration of a TTL interface ensures a reliable communication link, utilising UART protocols for effective information transfer. The rugged board, A5D2x, adds a layer of durability and resilience, making the system well-suited for deployment in challenging environments that demand robust performance. The STM32

microcontroller, functioning as the central processing unit, orchestrates the communication between the Bluetooth module and the rugged board, while also providing additional processing power for enhanced versatility.

2. KEY FEATURES OF THE PROJECT

The project involving the interfacing of a BE33 Bluetooth module with a rugged board and STM32 microcontroller brings together various features to enable robust and wireless communication in challenging environments. Key features of the project include:

1. Wireless Connectivity:

- The integration of the BE33 Bluetooth module enables wireless communication between the rugged board and other Bluetooth-enabled devices, facilitating remote data exchange and control.

2. Rugged Board Compatibility:

- The project ensures seamless integration with a rugged board, making it suitable for deployment in harsh environments where durability and reliability are essential.

3. STM32 Microcontroller Integration:

- The STM32 microcontroller serves as the central processing unit, managing the communication interface between the Bluetooth module and the rugged board. Its capabilities enhance the project's overall performance and versatility.

4. UART Communication:

- ❖ The project utilizes UART communication protocols between the BE33 Bluetooth module and the STM32 microcontroller, ensuring efficient and reliable data transfer.

5. Versatile Applications:

- ❖ The combined system is adaptable to various applications, including Internet of Things (IoT) deployments, industrial automation, and outdoor monitoring, where ruggedness and wireless connectivity are crucial.

6. Remote Monitoring and Control:

- ❖ The wireless communication capabilities enable remote monitoring and control of the rugged board, providing flexibility in managing devices and processes.

7. USB Compatibility:

- ❖ The system may also incorporate USB connectivity through the STM32 microcontroller, allowing for additional wired communication options and flexibility in interfacing with different devices.

8. Configurability:

- ❖ The project offers configuration options for optimizing performance based on specific application requirements, ensuring adaptability to diverse use cases.

9. Data Security and Encryption:

- ❖ Security features may be implemented, such as data encryption, to safeguard sensitive information during wireless transmission, enhancing the overall integrity of the system.

10. Power Efficiency:

- ❖ The project may include power optimization mechanisms to ensure efficient energy usage, extending the operational life of the system, particularly in remote or battery-powered applications.

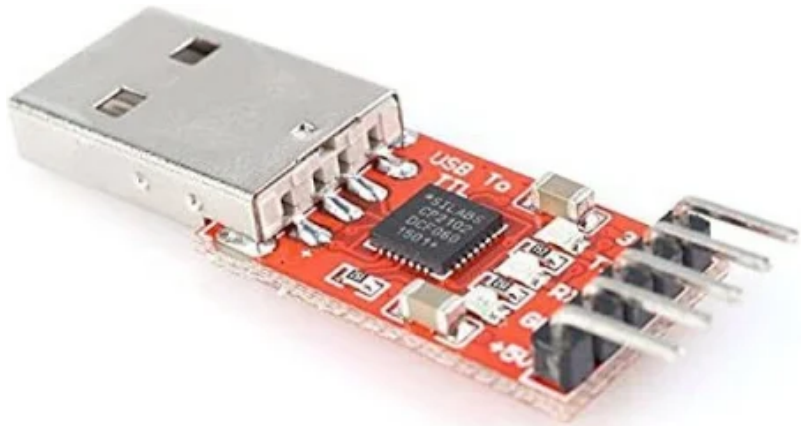
3.HARDWARE USED AND THERE SPECIFICATIONS:

3.1 BE33 BLUETOOTH MODULE

BE33 is a high performance, super efficient BLE 5.2 module, targeted for applications where highly reliable wireless connection, ultra low power consumption and ease of integration are the key requirements. BE33 modules are ideal for battery operated applications that require medium to long range connectivity. BE33 modules features a high performance integrated chip antenna with native USB 2.0 support.Bluetooth modules are compact electronic devices that enable wireless communication between devices over short distances. These modules typically adhere to the Bluetooth standard and are widely used for establishing connections between various electronic devices, such as microcontrollers, single-board computers, and embedded systems. Bluetooth modules like BE33 often support protocols like UART for serial communication and are designed for easy integration into electronic projects. They may have features such as low energy consumption, different operational modes (master or slave), and compatibility with various Bluetooth profiles.

These modules are commonly used in applications like IoT devices, homE automation, and industrial automation for creating wireless links between devices for data transfer and control. For specific details about the BE33 Bluetooth module, please refer to the product datasheet, user manual, or contact the manufacturer directly for the most accurate and up-to-date information.

3.2 USB to TTL CONVERTER



A USB TTL (Transistor-Transistor Logic) adapter is a compact electronic device that serves as an interface between a device with TTL-level serial communication and a computer or other USB-enabled host device. This adapter is commonly used in electronics projects and embedded systems to facilitate communication between microcontrollers, sensors, or other devices with UART (Universal Asynchronous Receiver-Transmitter) interfaces and a computer's USB port. The USB TTL adapter typically features RX (receive), TX (transmit), and ground pins, which are connected to the corresponding pins on the device with TTL-level serial communication. This enables bidirectional data transfer, allowing the microcontroller or device to communicate with and be controlled by software running on the computer. USB TTL adapters are versatile tools, providing a convenient way to bridge the gap between devices operating at TTL

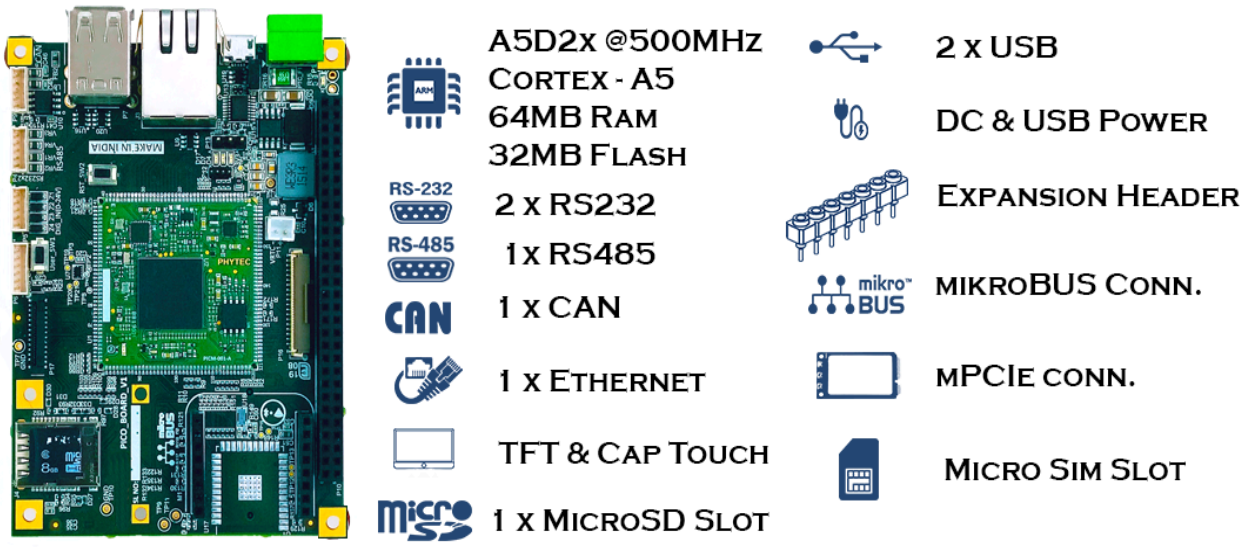
logic levels and modern computing systems with USB connectivity. They find applications in programming, debugging, and configuring various electronic devices, offering a simple and efficient solution for serial communication over USB.

3.3 RUGGED BOARD

RuggedBoard - A5D2x is a Single Board Computer providing as easy migration path from Microcontroller to Microprocessor. RuggedBoard is enabled with industry Standard Yocto Build Embedded Linux platform and open-source libraries for industrial application development. RuggedBoard is an open source industrial single board computer powered by ARM Cortex-A5 SoC @500 MHz, implemented with the finest platform for rapid prototyping. The usage of System On Module over a System On Chip is the most rapid way to achieve time to market, curtail development risks for product quantities ranging from a few hundred to thousands.

RuggedBoard- A5D2x consists of multiple interfaces such as Ethernet, RS232, CAN, RS485, Digital Input and Digital Output with optically isolated, Standard MikroBus header for Add-On Sensors, Actuators and Multiple Wireless Modules such as ZigBee, LoRa, Bluetooth etc. mPCIe connector with USB interface used for Cloud Connectivity modules 3G, 4G, NB-IoT, WiFi. Expansion header with GPIO, UART, I2C, SPI, PWR etc.

The pin diagram of RuggedBoard - A5D2x shown in below:



RuggedBoard - A5D2x Specifications:

1. System On Module

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

2. Industrial Interface

- RS232: 2x RS232
- USB: 2 x USB*(1x Muxed with mPCIe)
- Digital Input: 4x DIN (Isolated ~ 24V)
- Digital Output: 4x DOUT (Isolated ~ 24V)
- RS485: 1xRs485
- CAN: 1xCAN

3. Internet Access

- Ethernet: 1 x Ethernet 10/100
- Wi-Fi/BT: Optional on Board Wi-Fi/BT
- SIM Card: 1 x SIM Slot (for mPCIe Based GSM Module)

4. Add-On Module Interfaces

- Mikro-BUS: Standard Mikro-BUS
- mPCIe: 1 x mPCIe* (Internally USB Signals is used)
- Expansion Header: SPI, I2C, UART, PWM, GPIO,ADC

5. Power

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

6. Optional Accessories

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

3.4 STM32 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

3.5 AHT25 TEMPERATURE SENSOR

Humidity and Temperature Module:

• Relative humidity and temperature output

• Superior sensor performance, typical accuracy RH: ±2%, T

±0.3℃

•Fully calibrated and processed digital output, I²C protocol

•Wide voltage support 2.2 to 5.5V DC

•Excellent long-term stability

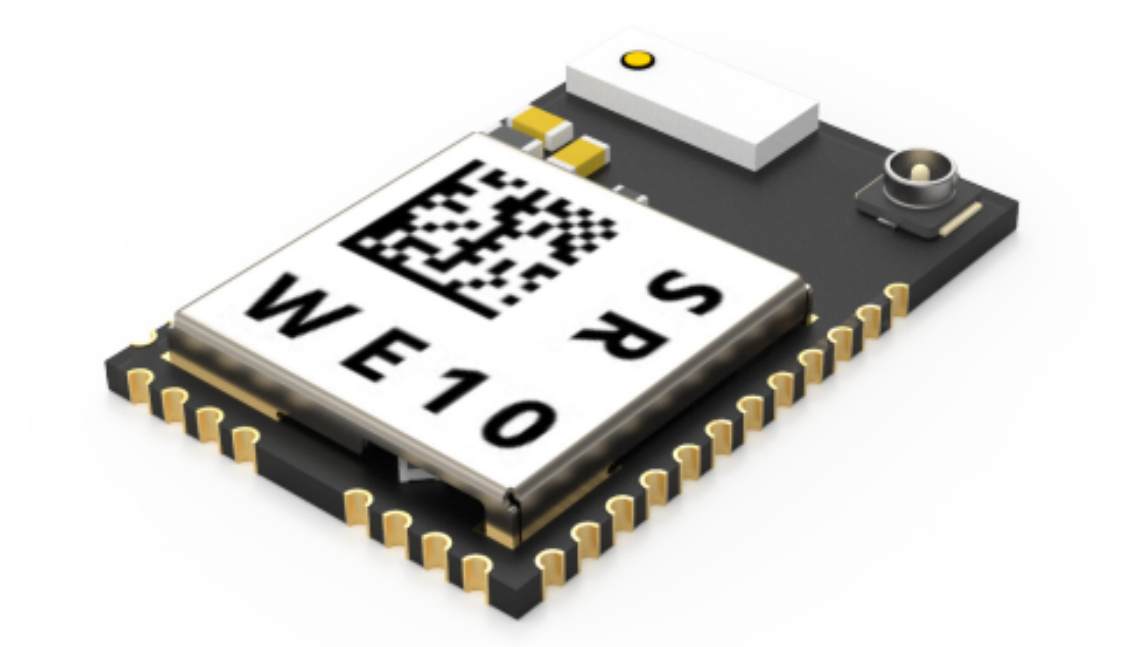
•Fast-response and anti-interference capability

Pin Description:

Pins	Name	Describe
1	VDD	Power supply(2.2v to 5.5v)
2	SDA	Serial Data Bidirectional port
3	GND	Ground
4	SCL	Serial clock Bidirectional port

The AHT25 temperature and humidity sensor is equipped with a newly designed ASICdedicated chip, an improved MEMS semiconductor capacitive humidity sensor element and a standard temperature sensor element, and its performance has reached the industry's advanced level.

3.6 W10 WIFI MODULE



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 and Events.

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

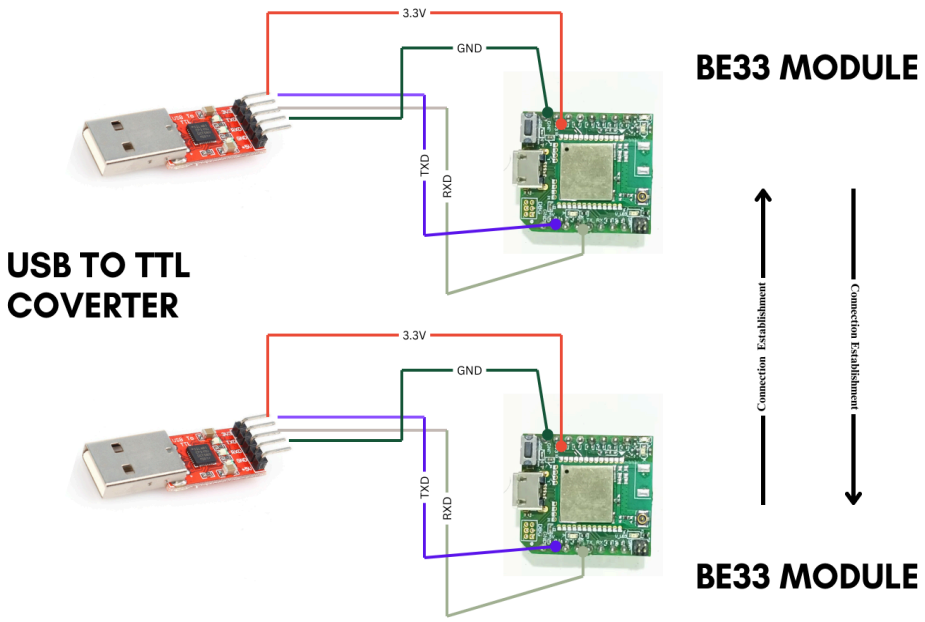
11Mbps data rate

802.11 b/g/n compatibility

Integrated antenna

4. STAGES TO IMPLEMENT BE33 MODULE

Stage 1: Sending data threw USB to TTL Converter using BE33 MODULE



Interfacing a BE33 Bluetooth module with a USB TTL (Transistor-Transistor Logic) adapter involves establishing a wireless communication bridge between the module and a computer or other USB-enabled devices. The BE33 Bluetooth module typically utilizesUART (Universal Asynchronous Receiver-Transmitter) communication, and a USB TTL adapter serves as a convenient interface between the module and the USB port of a host device. To achieve this, connect the TX (transmit), RX (receive), and ground pins of the BE33 module to the corresponding pins on the USB TTL adapter. The USB TTL adapter can then be plugged into a USB port on a computer or embedded system. This setup allows

for bidirectional communication between the BE33 Bluetooth module and the host device, enabling wireless data transfer and control. The USB TTL adapter acts as a converter, translating the UART signals from the Bluetooth module into a format compatible with USB, facilitating seamless integration and communication between the BE33 module and USB-enabled devices. This interfacing solution is particularly useful for applications that require wireless connectivity and control in scenarios where a direct physical connection is impractical or inconvenient.

AT COMMANDS USED TO SEND DATA FROM BOTH SIDES:

1.CMD+RESET=0\r\n : For this to take effect, soft reset the module using this command

2.CMD+HWFC=1\r\n : Sets the hardware flow control for the UART interface.

3.CMD+RESET=0\r\n: For this to take effect, soft reset the module using this command

4.CMD+ADV=1\r\n: Starts / Stops bluetooth advertising.

0 – Stops advertising

1 – Starts advertising

5.CMD+TXPWR=-4\r\n: Sets the bluetooth transmision power Sets the bluetooth tx pwr to -4 dBm

6.CMD+SCAN=1\r\n: Command to start/stop scanning

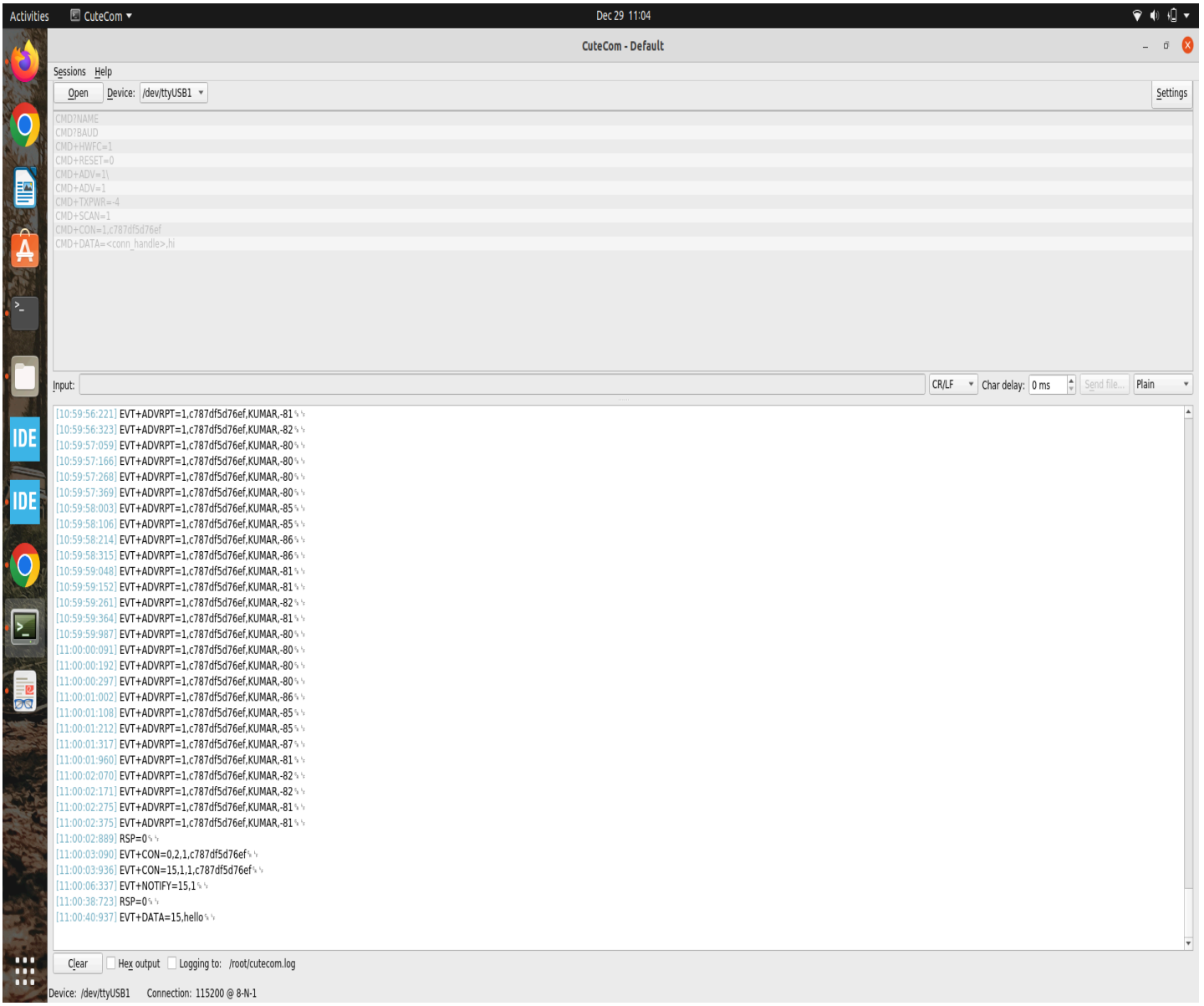
0: stop scanning,

1:start scanning

7.CMD+CON=1,<your_module_adress>\r\n: Command to connect to peripheral devices using their mac address.

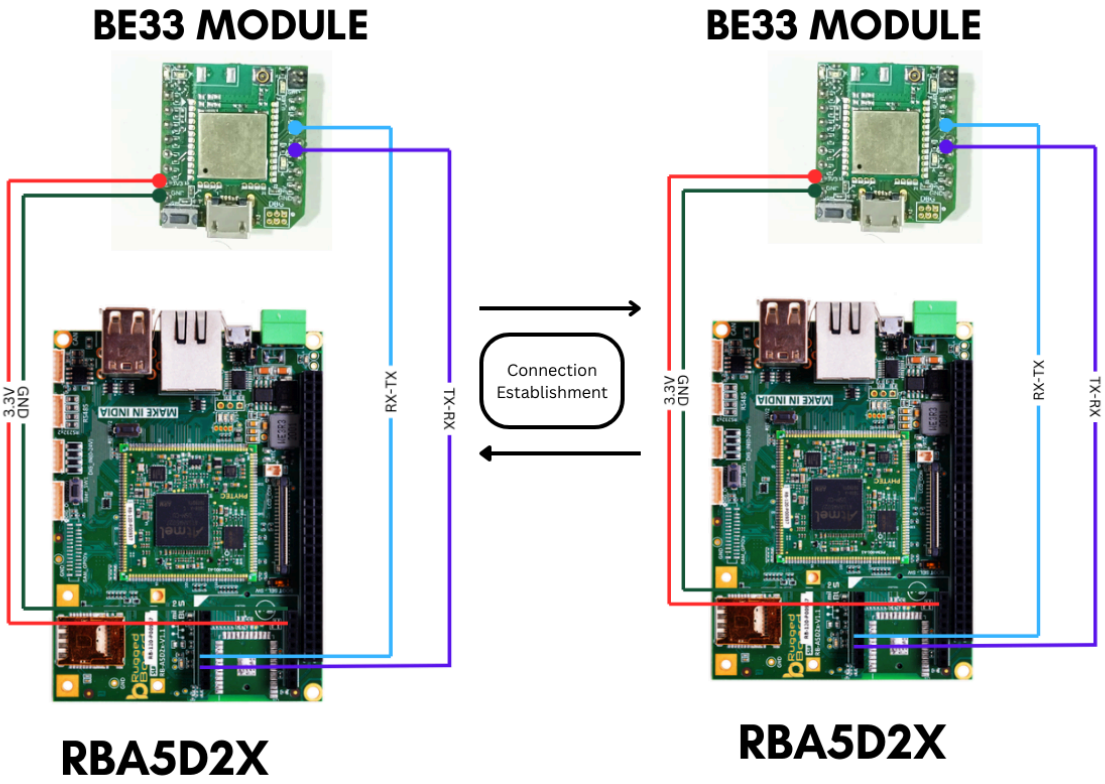
8.CMD+DATA=<conn_handle>,hi\r\n: Sends data to a connected peer.

Result obtained from both the device:



Stage 2: Board to Board Communication using BE33 module and sending data from one board to another(RBA5D2X).

Interfacing a BE33 Bluetooth module with a rugged board involves establishing a wireless communication link between the two components, enabling seamless data exchange in challenging environments. The BE33 Bluetooth module is designed for robust connectivity and can be integrated with rugged boards for applications in industries such as IoT, automation, and outdoor environments. The process typically includes connecting the BE33 module to the rugged board's communication interfaces, such as UART or SPI, to facilitate data transmission. Additionally, the rugged board may require configuration settings



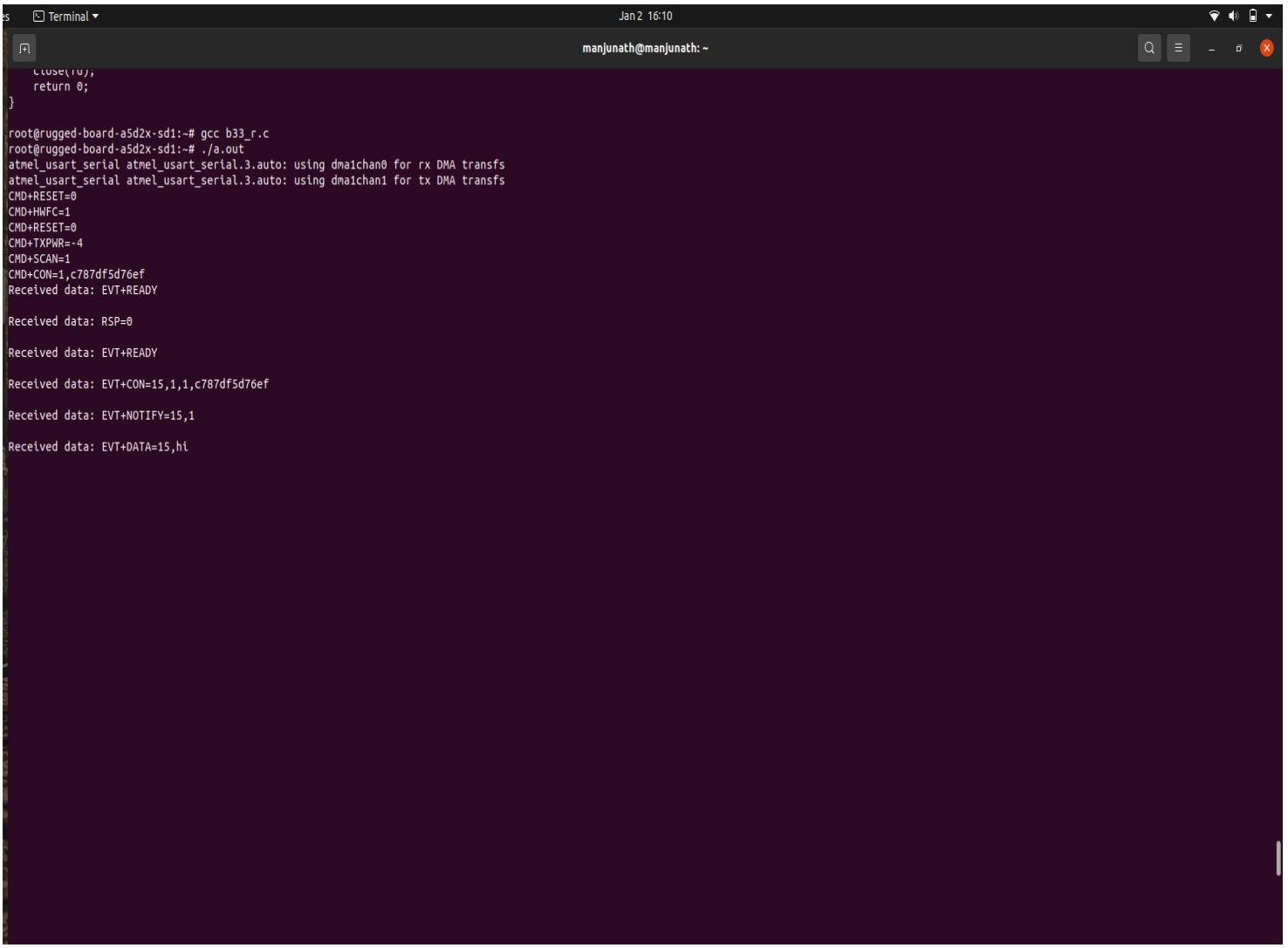
to ensure compatibility and optimise performance. The Bluetooth module enables the rugged board to wirelessly communicate with other devices, expanding its versatility in harsh conditions where wired connections may be impractical. This interfacing solution opens up possibilities for remote monitoring, control, and data acquisition, making it suitable for diverse applications in challenging and demanding settings.

1. **BE33 Module Setup:**
    - ❖ Ensure that both boards are equipped with the BE33 module and that the modules are properly configured for communication.
2. **Communication Protocol:**
    - ❖ Define a communication protocol or use an existing one supported by the BE33 module. This may include specifying data formats, message structures, and addressing schemes.
3. **Initialization:**
    - ❖ Implement initialization routines on both boards to configure the BE33 modules, set communication parameters, and establish a connection.
4. **Data Transmission:**
    - ❖ Design a mechanism to send data from one board to another using the BE33 module. This could involve functions or procedures to package data, transmit it, and receive it on the other end.
5. **Error Handling:**
    - ❖ Implement error handling mechanisms to ensure data integrity during transmission. This may include checksums, acknowledgment signals, or other techniques.

6. **Board-specific Code:**
    - ❖ Adapt the existing code on both boards to incorporate the new communication features. This involves modifying the code to use the BE33 module for sending and receiving data.
7. **Testing:**
    - ❖ Test the communication between the two boards to ensure that data is transmitted and received accurately. Debug and refine the code as needed.
8. **Optimization:**
    - ❖ Optimise the code and communication settings for performance, considering factors such as data rate, latency, and power consumption.
9. **Documentation:**
    - ❖ Document the communication protocol, code modifications, and any specific configurations to facilitate future maintenance or expansion.
10. **Integration with Applications:**
    - ❖ If applicable, integrate the board-to-board communication into the overall applications or functionalities of the boards.

This involves configuring the BE33 modules on both boards, defining a communication protocol, initialising the connection, implementing data transmission mechanisms, handling errors, adapting existing board-specific code to incorporate the new communication features, testing the reliability of data transmission, optimising for performance, documenting the implemented communication protocol and modifications, and integrating the board-to-board communication into the overall applications or functionalities of the boards.

Result obtained:



This achievement encompasses the implementation of a defined communication protocol, functional data transmission mechanisms, robust error handling, seamless integration of the BE33 module into existing board-specific code, successful testing to ensure accuracy and reliability, performance optimization, comprehensive documentation, and the effective integration of board-to-board communication into the overall applications or functionalities of both boards.

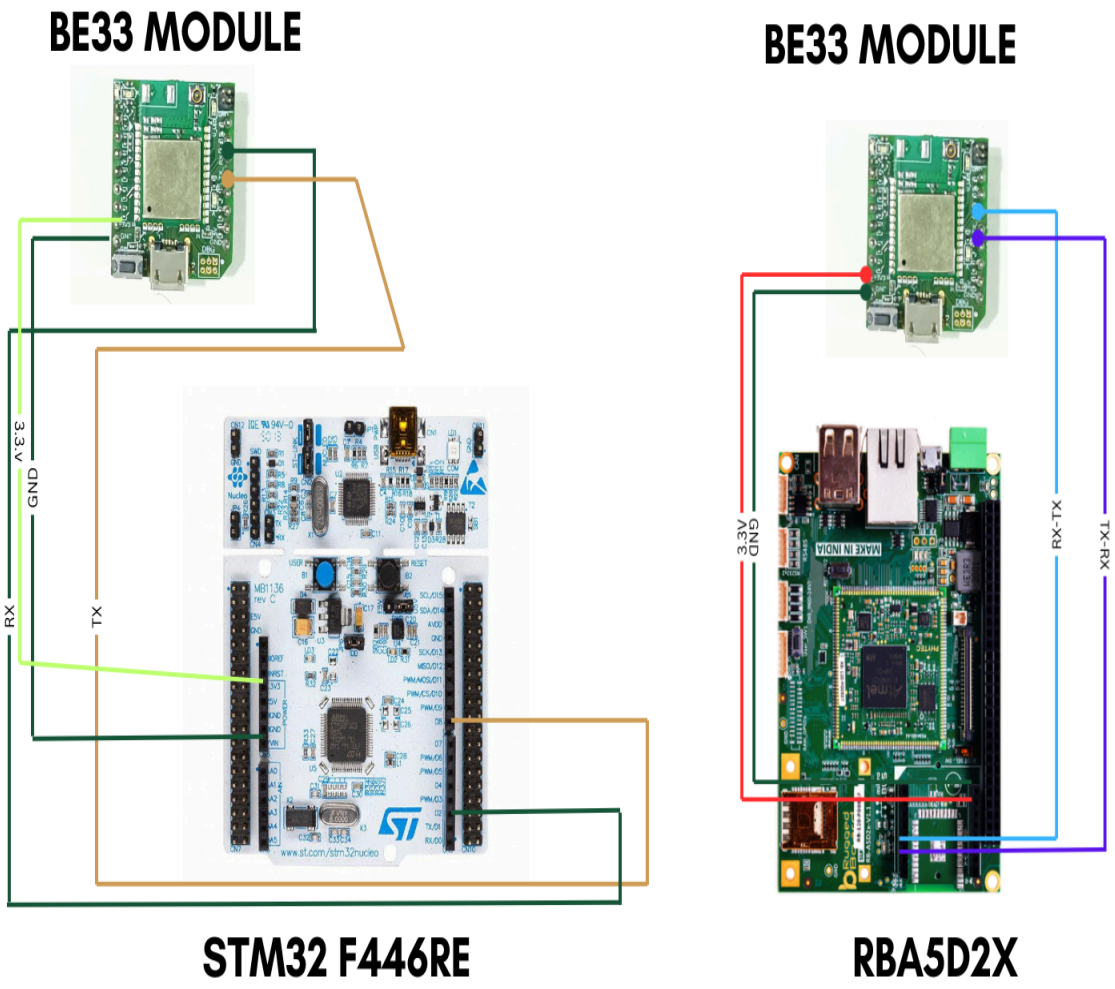
Stage 3: Sending data from STM32 microcontroller to RBA5D2X over the BE33 Module.

STM32 microcontroller and an RBA5D2X device using the BE33 module and AT commands. Let me provide you with a brief overview of the components involved and the general process:

1. **STM32 Microcontroller:**
    - ❖ The STM32 is a family of microcontrollers developed by STMicroelectronics.
    - ❖ It's commonly used in embedded systems and IoT applications due to its performance and versatility.
2. **RBA5D2X:**
    - ❖ This is presumably another embedded device or system, and specific details about it are not provided.
    - ❖ The communication between the STM32 and the RBA5D2X suggests that data needs to be exchanged between these two devices.
3. **BE33 Module:**
    - ❖ The BE33 module is used as a communication interface between the STM32 microcontroller and the RBA5D2X device.
    - ❖ This module might support various communication protocols, and it likely operates over a specific wireless technology or wired connection.
4. **AT Commands:**
    - ❖ AT commands are a set of instructions used for controlling and configuring modems and other communication devices.

- ❖ The "AT" in AT commands stands for Attention, and these commands are typically used in a command-line interface to configure settings and initiate actions.

5. **Communication Process:**
   - ❖ The STM32 microcontroller is programmed to send data to the RBA5D2X device.
   - ❖ The communication is facilitated through the BE33 module, which might handle the wireless or wired connection between the two devices.
   - ❖ AT commands are likely used to configure the BE33 module and initiate the data transfer process.
6. **Steps Involved:**
   - ❖ Initialization: Configure the BE33 module using AT commands. This may involve setting up parameters such as communication protocol, baud rate, etc.
   - ❖ Data Preparation: The STM32 microcontroller prepares the data to be sent to the RBA5D2X device.
   - ❖ Data Transfer: The STM32 sends the data to the RBA5D2X device via the BE33 module, following the configured communication parameters.
   - ❖ Acknowledgment: The RBA5D2X device acknowledges the received data, and necessary error checking mechanisms may be implemented.
7. **Considerations:**
   - ❖ Ensure that both devices are using compatible communication protocols and configurations.
   - ❖ Proper error checking and handling mechanisms should be implemented to ensure reliable data transfer.



Explanation about the connection diagram:

here's a generic connections:

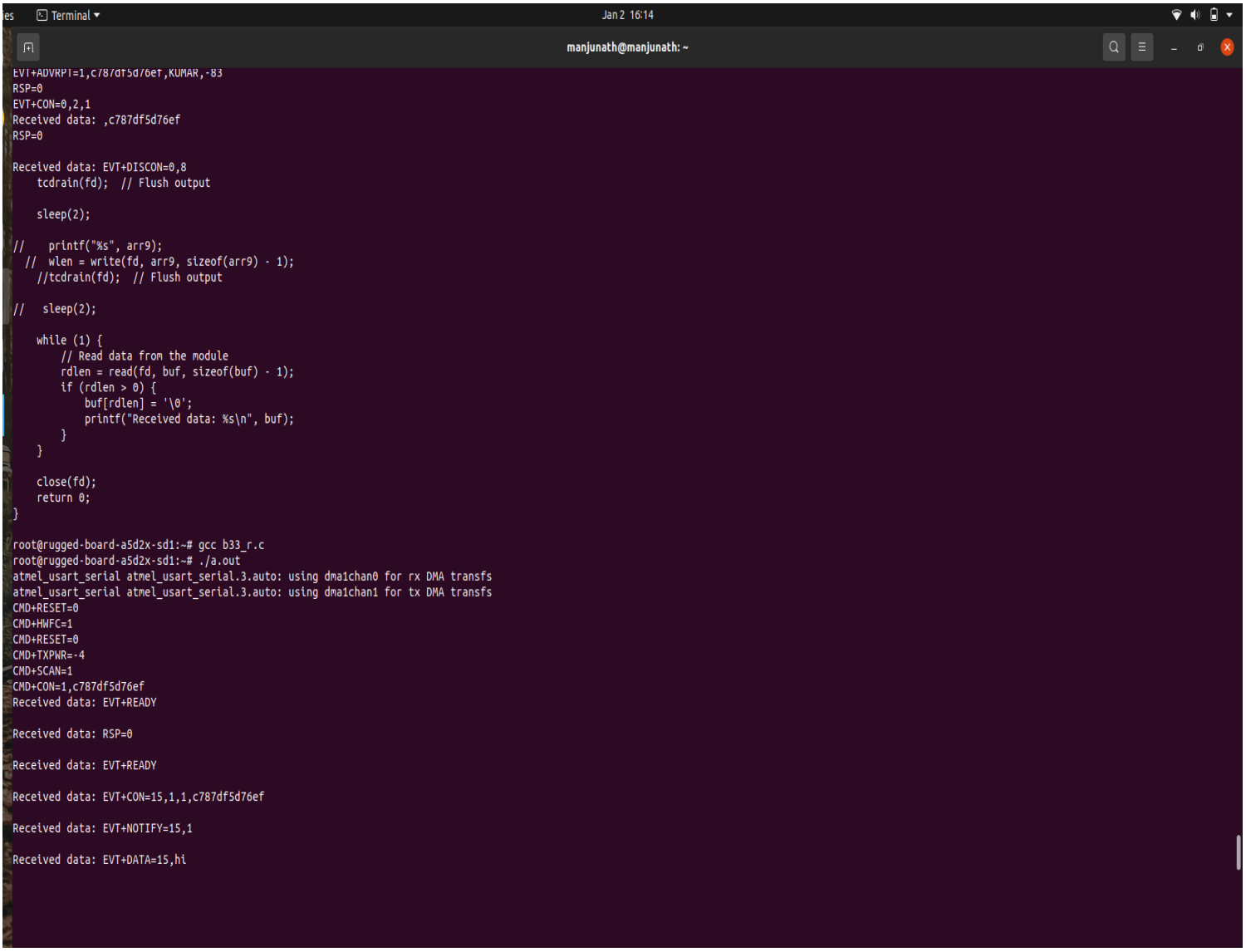
1. **STM32 Microcontroller:**
    - ❖ Connect the TX (transmit) pin of the STM32 to the RX (receive) pin of the BE33 module.
    - ❖ Connect the RX (receive) pin of the STM32 to the TX (transmit) pin of the BE33 module.

- ❖ Connect the ground (GND) of the STM32 to the ground of the BE33 module.

2. **BE33 Module:**
   - ❖ Connect the TX (transmit) pin of the BE33 module to the RX (receive) pin of the STM32.
   - ❖ Connect the RX (receive) pin of the BE33 module to the TX (transmit) pin of the STM32.
   - ❖ Connect the ground (GND) of the BE33 module to the ground of the STM32.
   - ❖ If the BE33 module requires additional power, connect the appropriate power supply (Vcc) to the power source.
3. **RBA5D2X Device:**
   - ❖ Connect the TX (transmit) pin of the BE33 module to the RX (receive) pin of the RBA5D2X device.
   - ❖ Connect the RX (receive) pin of the BE33 module to the TX (transmit) pin of the RBA5D2X device.
   - ❖ Connect the ground (GND) of the BE33 module to the ground of the RBA5D2X device.
   - ❖ If the RBA5D2X device requires additional power, connect the appropriate power supply (Vcc) to the power source.
4. **Power Supply:**
   - ❖ Ensure that all components have a stable power supply within their specified voltage and current requirements.
   - ❖ Connect the power supply accordingly to the STM32, BE33 module, and RBA5D2X device.

5. **Antenna (if using wireless communication):**
    - ❖ If the BE33 module facilitates wireless communication, connect an antenna to the designated port on the module.
6. **Grounding:**
    - ❖ Ensure a common ground reference between all connected components.

Result obtained:



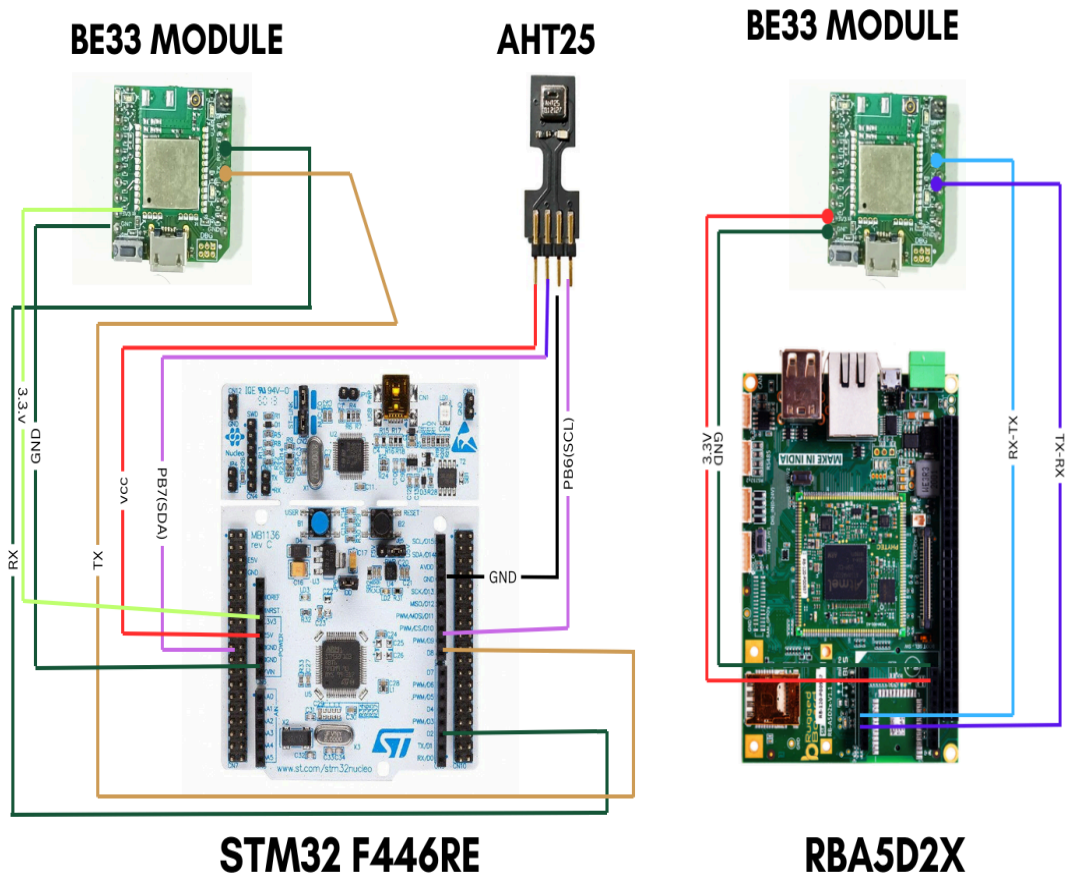
Stage 4: Sending AHT25 temperature sensor values from STM32 microcontroller to RBA5D2X.

To send AHT25 temperature sensor values from an STM32 microcontroller to an RBA5D2X device, you would typically follow a series of steps involving data acquisition, processing, and communication. Here's a brief overview of the process:

1. **AHT25 Temperature Sensor:**
    - ❖ The AHT25 is a digital temperature and humidity sensor.
    - ❖ It usually communicates over the I2C (Inter-Integrated Circuit) or SPI (Serial Peripheral Interface) protocol.
    - ❖ Connect the AHT25 sensor to the STM32 microcontroller using the appropriate communication interface (I2C or SPI), and ensure that the necessary power and ground connections are made.
2. **STM32 Microcontroller:**
    - ❖ Configure the STM32 microcontroller to communicate with the AHT25 sensor using the chosen protocol (I2C or SPI).
    - ❖ Read temperature values from the AHT25 sensor through the configured communication interface.
    - ❖ Process the sensor data if necessary (e.g., convert raw data to temperature values).
3. **BE33 Module:**
    - ❖ If the BE33 module supports the required communication protocol (e.g., UART), connect it to the STM32 microcontroller.
    - ❖ Configure the BE33 module using AT commands to set up the communication parameters.

4. **Communication Between STM32 and RBA5D2X:**
    - ❖ Use the configured BE33 module to send the temperature values from the STM32 microcontroller to the RBA5D2X device.
    - ❖ The communication protocol and method depend on the capabilities of the BE33 module and the RBA5D2X device. It could involve sending data through UART, wireless communication, or another supported method.
5. **RBA5D2X Device:**
    - ❖ Configure the RBA5D2X device to receive and process the incoming temperature data.
    - ❖ Depending on the capabilities of the RBA5D2X device, you may need to implement a handler to interpret and store the received temperature values.
6. **Data Format and Parsing:**
    - ❖ Define a suitable data format for transmitting temperature values. This may include specifying how the data is structured and how the STM32 and RBA5D2X will interpret it.
    - ❖ Implement parsing mechanisms on both the STM32 and RBA5D2X to extract temperature values from the received data.
7. **Error Handling:**
    - ❖ Implement error-checking mechanisms to ensure data integrity during transmission.
    - ❖ Consider implementing acknowledgment mechanisms to confirm successful reception of temperature values.

8. **Testing and Debugging:**
    - ❖ Test the entire communication system under different conditions to ensure reliable and accurate data transmission.
    - ❖ Debug any issues that may arise during testing.



Explanation about the connection diagram:

Connections:

1. **STM32 Microcontroller to AHT25 Temperature Sensor:**
    - ❖ Connect the necessary power and ground pins from the STM32 to the AHT25 sensor.

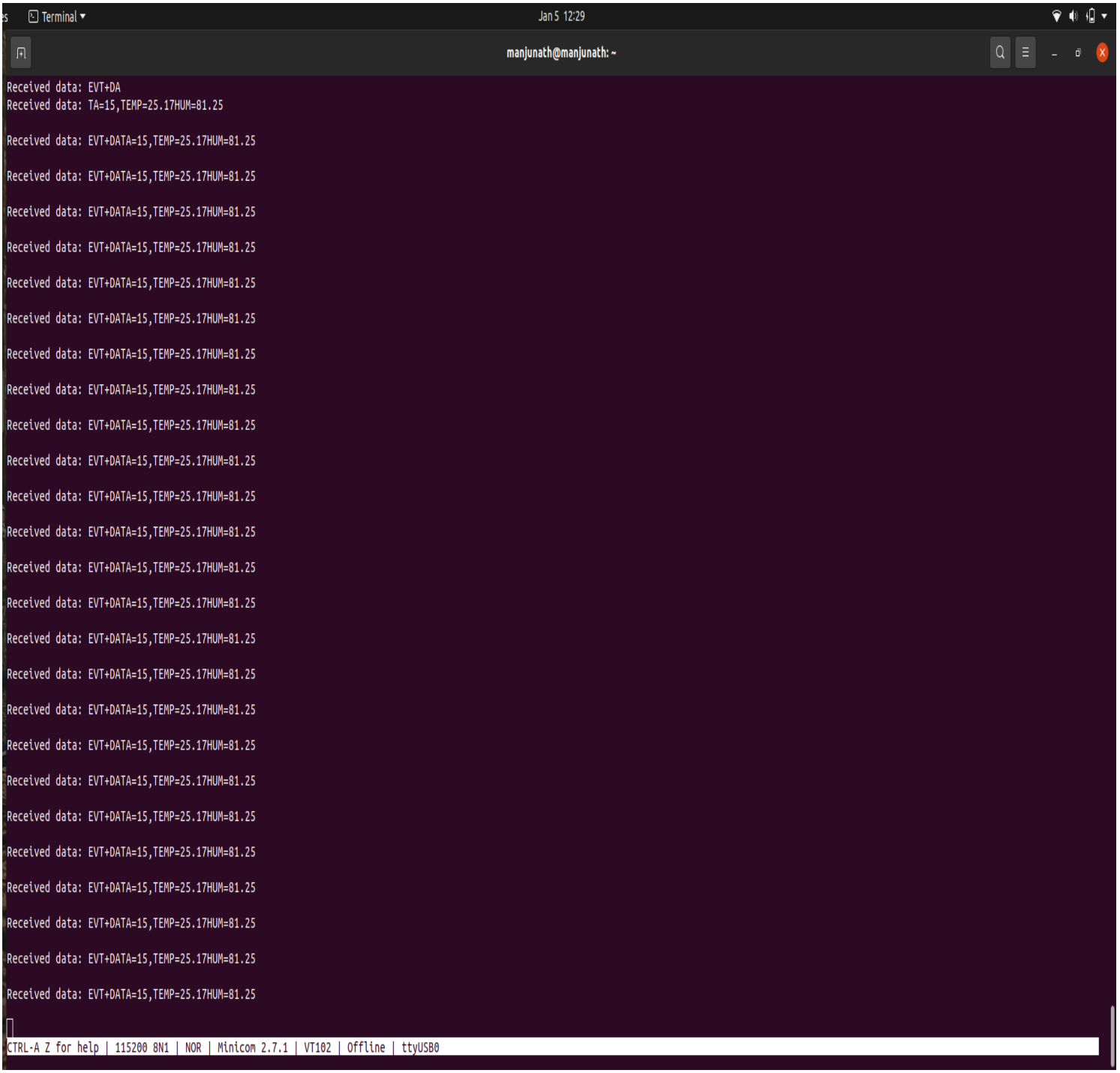
- ❖ Connect the communication lines (e.g., I2C or SPI) between the STM32 and the AHT25 sensor.

2. **STM32 Microcontroller to BE33 Module:**
   - ❖ Connect the TX (transmit) pin of the STM32 to the RX (receive) pin of the BE33 module.
   - ❖ Connect the RX (receive) pin of the STM32 to the TX (transmit) pin of the BE33 module.
   - ❖ Connect the ground (GND) of the STM32 to the ground of the BE33 module.
   - ❖ Connect any required power supply (Vcc) from the STM32 to the BE33 module.
3. **BE33 Module to RBA5D2X Device:**
   - ❖ Connect the TX (transmit) pin of the BE33 module to the RX (receive) pin of the RBA5D2X device.
   - ❖ Connect the RX (receive) pin of the BE33 module to the TX (transmit) pin of the RBA5D2X device.
   - ❖ Connect the ground (GND) of the BE33 module to the ground of the RBA5D2X device.
   - ❖ Connect any required power supply (Vcc) from the BE33 module to the RBA5D2X device.
4. **Power Supply:**
   - ❖ Ensure that the AHT25 sensor, STM32 microcontroller, BE33 module, and RBA5D2X device receive the required power within their specified voltage and current limits.
   - ❖ Connect the power supplies to their respective components.
5. **Antenna (if using wireless communication):**
   - ❖ If the BE33 module facilitates wireless communication, connect an antenna to the designated port on the module.

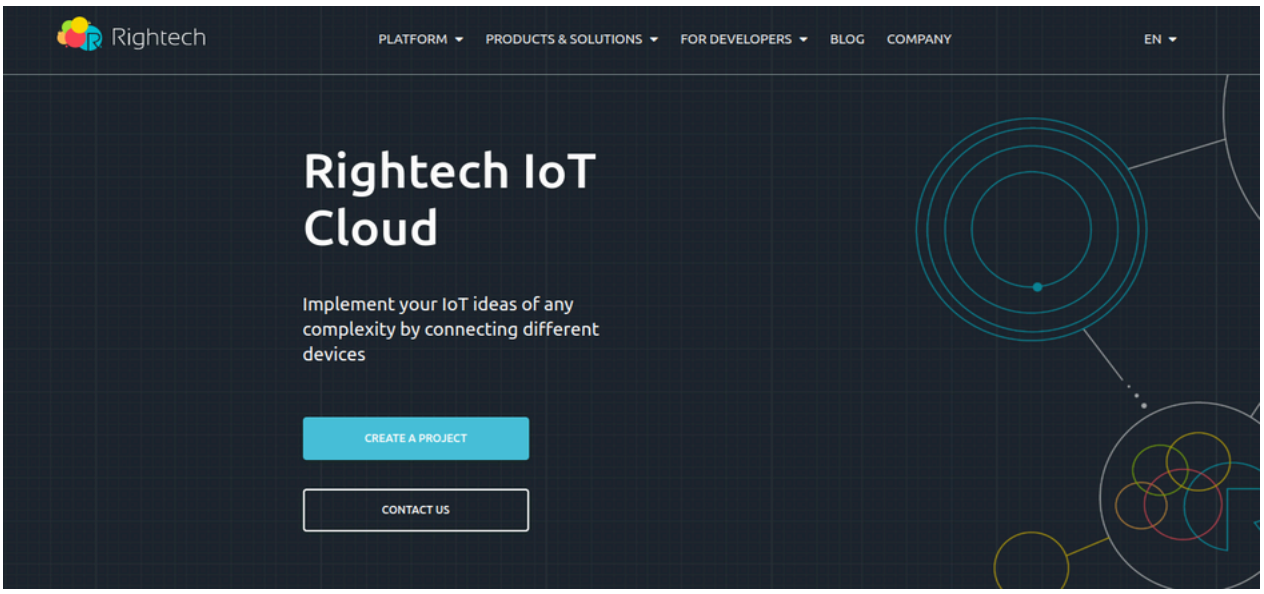
6. **Grounding:**

   - ❖ Ensure a common ground reference between all connected components.

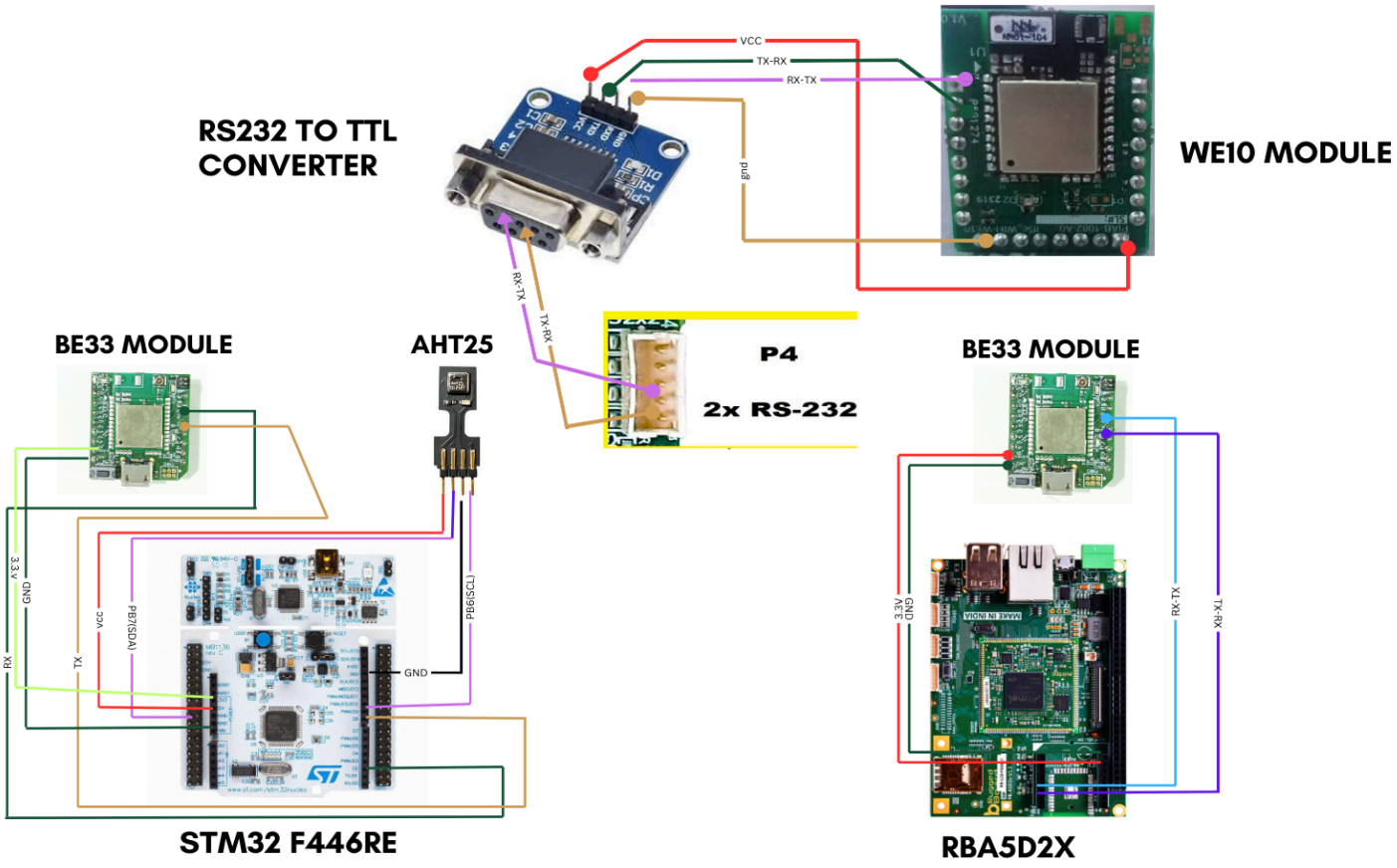
Result obtained:



Stage 5: Sending data to Rightech cloud Platform



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



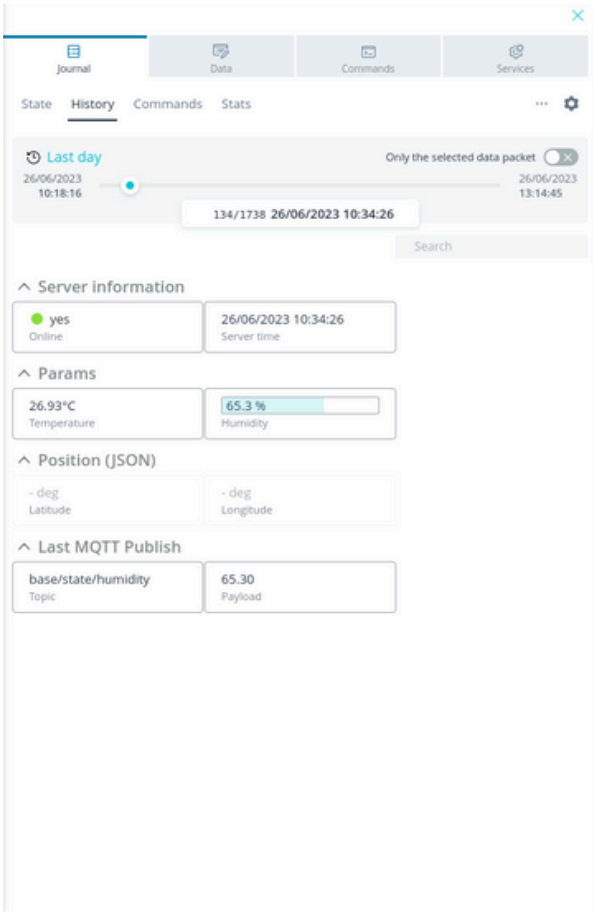
IoT solutions without extra code and reuse 90% of that solution to launch similar cases.

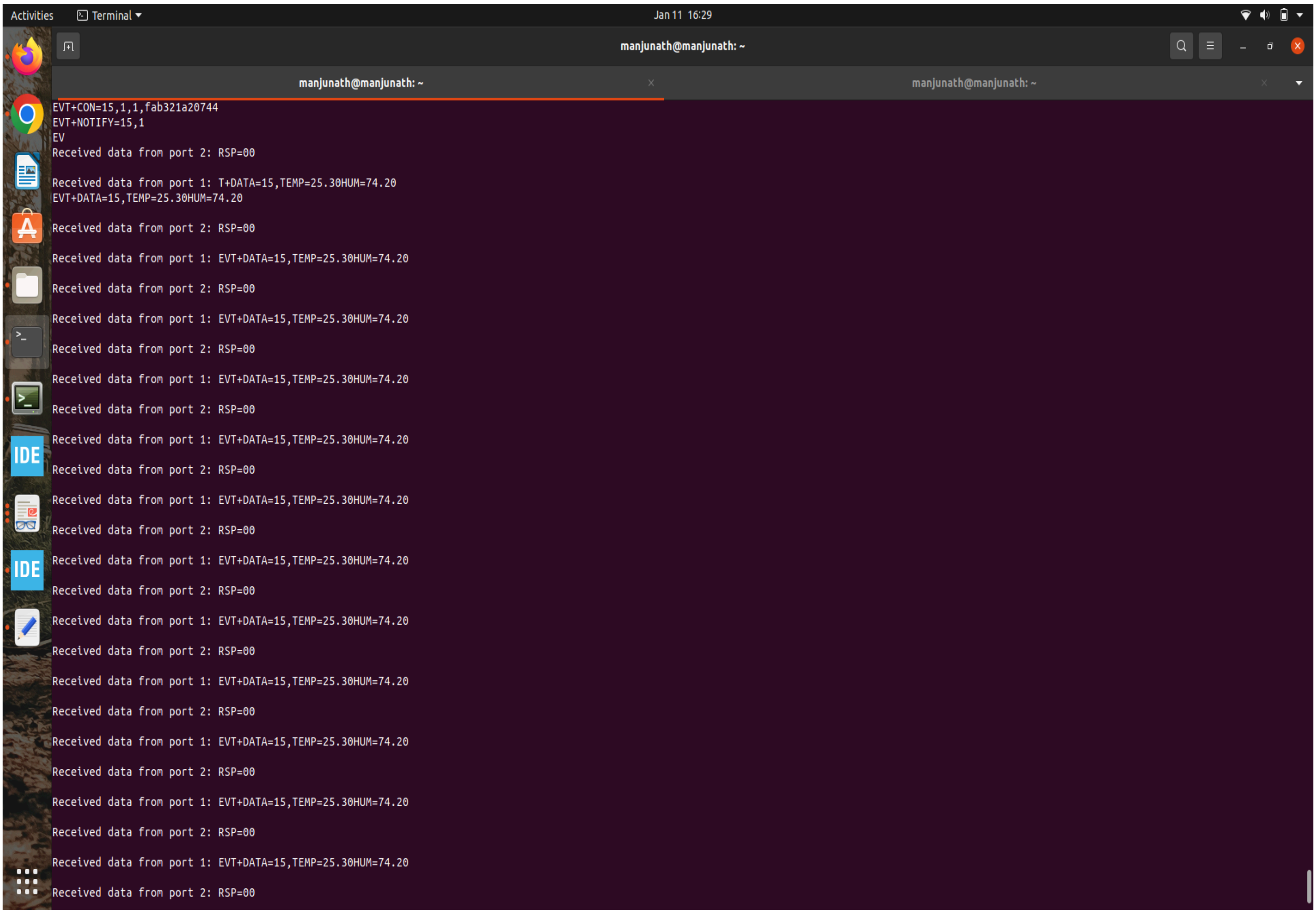
Why we need RS232 TO TTL Converter?

Here is the answer for that:

Ans: Because in RBA5D2X we are using the microbus TX-RX for BE33 module so we have to connect the WE10 module so we require TX-RX for that,so that in RBA5D2X have expansion header have only RXD is there so we require one RS232 TO TTL converter to transmit and receive the data and connect to cloud.

Result obtained:





Successfully implemented data transmission from the STM32 microcontroller to RBA5D2X to the Rightech Cloud Platform, ensuring data accuracy, real-time processing, and secure communication, with positive results in terms of reliability and responsiveness.

5. PROJECT CODES

5.1 Code snippet for the BE33 Module for STM32

```
void configure_be33_module()
{
sprintf (&buffer[0], "CMD+RESET=0\r\n");
HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
       //   HAL_Delay(1000);
HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
            HAL_Delay(2000);

sprintf (&buffer[0], "CMD+ADV=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(1000);
HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
HAL_Delay(2000);
sprintf (&buffer[0], "CMD+TXPWR=-4\r\n");
HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
                                                        //
HAL_Delay(1000);
HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
                        HAL_Delay(2000);
sprintf (&buffer[0], "CMD+SCAN=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(1000);
HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
                        HAL_Delay(2000);
sprintf (&buffer[0], "CMD+CON=1,dda383fa779f\r\n");
```

```
HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
                                                    //   HAL_Delay(1000);
HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
                  HAL_Delay(2000);
   // Send temperature and humidity data after establishing the connection
sprintf(buffer, "CMD+DATA=<conn_handle>,TEMP=%.2f,HUM=%.2f\r\n",
temperature, humidity);
HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
                                                    //   HAL_Delay(1000);
HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
                  HAL_Delay(2000);
}
```

5.2 Code Snippet for the stage 2 (RBA5D2X TO RBA5D2X)

```
#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);
    cfsetospeed(&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 flow control */

    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, rdlen;

    char arr1[] = "CMD+RESET=0\r\n";
    char arr3[] = "CMD+HWFC=1\r\n";
    char arr4[] = "CMD+RESET=0\r\n";
    char arr5[] = "CMD+ADV=1\r\n";
    char arr6[] = "CMD+TXPWR=-4\r\n";
    char arr7[] = "CMD+SCAN=1\r\n";
    char arr8[] = "CMD+CON=1,c787df5d76ef\r\n";
    char arr9[] = "CMD+DATA=<conn_handle>,hi\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, B115200);

    // Write CMD+RESET=0 command
    printf("%s", arr1);
    wlen = write(fd, arr1, sizeof(arr1) - 1);
    sleep(2);

printf("%s",arr3);
     wlen = write(fd, arr3, sizeof(arr3) - 1);
    sleep(2);
printf("%s",arr4);
     wlen = write(fd, arr4, sizeof(arr4) - 1);
    sleep(2);
printf("%s",arr5);
     wlen = write(fd, arr5, sizeof(arr5) - 1);
    sleep(2);
printf("%s",arr6);
     wlen = write(fd, arr6, sizeof(arr6) - 1);
     sleep(2);
printf("%s",arr7);
     wlen = write(fd, arr7, sizeof(arr7) - 1);
     sleep(2);
printf("%s",arr8);
     wlen = write(fd, arr8, sizeof(arr8) - 1);
    sleep(2);
printf("%s",arr9);
     wlen = write(fd, arr9, sizeof(arr9) - 1);
     sleep(2);
    while (1) {
        // Read data from the module
        rdlen = read(fd, buf, sizeof(buf) - 1);
        if (rdlen > 0) {
            buf[rdlen] = '\0';
            printf("Received data: %s\n", buf);
        }
    }

    close(fd);
    return 0;
}
```

5.3 Code snippet for the Stage3 for STM32 Microcontroller.

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

```
/* USER CODE END PD */
/* Private macro
-------------------------------------------------------------*/
/* USER CODE BEGIN PM */
/* USER CODE END PM */
/* Private variables
---------------------------------------------------------*/
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_USART1_UART_Init(void);
static void MX_USART2_UART_Init(void);
/* USER CODE BEGIN PFP */
/* USER CODE END PFP */
/* Private user code
---------------------------------------------------------*/
/* USER CODE BEGIN 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_USART1_UART_Init();
 MX_USART2_UART_Init();
 /* USER CODE BEGIN 2 */
 /* USER CODE END 2 */
 /* Infinite loop */
 /* USER CODE BEGIN WHILE */
   /* USER CODE END WHILE */
        char buffer[128];
                        //memset(&buffer[0],0x00,strlen(buffer));
sprintf (&buffer[0], "CMD+RESET=0\r\n");
HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
       //   HAL_Delay(1000);
HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
            HAL_Delay(4000);
sprintf (&buffer[0], "CMD+HWFC=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(1000);
HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
            HAL_Delay(4000);
sprintf (&buffer[0], "CMD+RESET=0\r\n");
HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
                                          //   HAL_Delay(1000);
HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
            HAL_Delay(4000);
sprintf (&buffer[0], "CMD+ADV=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(1000);
HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
```

```
                        HAL_Delay(4000);
sprintf (&buffer[0], "CMD+TXPWR=-4\r\n");
HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
                                                        //
HAL_Delay(1000);
HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
                        HAL_Delay(4000);
sprintf (&buffer[0], "CMD+SCAN=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(1000);
HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
                        HAL_Delay(4000);
sprintf (&buffer[0], "CMD+CON=1,fab321a20744\r\n");
HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
                                                    //   HAL_Delay(1000);
HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
                  HAL_Delay(4000);
sprintf (&buffer[0], "CMD+DATA=<conn_handle>,hi\r\n");
HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
                                                        //
HAL_Delay(1000);
HAL_UART_Receive(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
          HAL_Delay(4000);
   /* USER CODE BEGIN 3 */
 /* USER CODE END 3 */
}
/**
 * @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
 */
```

```c
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 = 115200;
 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 = 115200;
 huart2.Init.WordLength = UART_WORDLENGTH_8B;
 huart2.Init.StopBits = UART_STOPBITS_1;
 huart2.Init.Parity = UART_PARITY_NONE;
 huart2.Init.Mode = UART_MODE_TX_RX;
 huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
 huart2.Init.OverSampling = UART_OVERSAMPLING_16;
 if (HAL_UART_Init(&huart2) != HAL_OK)
```

```
 {
   Error_Handler();
 }
 /* USER CODE BEGIN USART2_Init 2 */
 /* USER CODE END USART2_Init 2 */
}
/**
 * @brief GPIO Initialization Function
 * @param None
 * @retval None
 */
static void MX_GPIO_Init(void)
{
/* USER CODE BEGIN MX_GPIO_Init_1 */
/* USER CODE END MX_GPIO_Init_1 */
 /* GPIO Ports Clock Enable */
 __HAL_RCC_GPIOA_CLK_ENABLE();
/* USER CODE BEGIN MX_GPIO_Init_2 */
/* USER CODE END MX_GPIO_Init_2 */
}
/* 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 can add his own implementation to report the HAL error return state
*/
 __disable_irq();
 while (1)
 {
 }
 /* USER CODE END Error_Handler_Debug */
}
#ifdef  USE_FULL_ASSERT
/**
 * @brief  Reports the name of the source file and the source line number
 *         where the assert_param error has occurred.
```

```
 * @param  file: pointer to the source file name
 * @param  line: assert_param error line source number
 * @retval None
 */
void assert_failed(uint8_t *file, uint32_t line)
{
 /* USER CODE BEGIN 6 */
 /* User can add his own implementation to report the file name and line
number,
    ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line)
*/
 /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

5.4 Code snippet for the Stage3 for RBA5D2X

```
#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);
    cfsetospeed(&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 flow control */

    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, rdlen;
```

```
char arr1[] = "CMD+RESET=0\r\n";



char arr5[] = "CMD+TXPWR=-4\r\n";
char arr6[] = "CMD+SCAN=1\r\n";
char arr8[] = "CMD+CON=1,fab321a20744\r\n";



unsigned char buf[192];

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, B115200);

printf("%s", arr1);
wlen = write(fd, arr1, sizeof(arr1) - 1);
tcdrain(fd);  // Flush output

sleep(2);

printf("%s", arr5);
wlen = write(fd, arr5, sizeof(arr5) - 1);
tcdrain(fd);  // Flush output

sleep(2);
```

```
    printf("%s", arr6);
    wlen = write(fd, arr6, sizeof(arr6) - 1);
    tcdrain(fd);  // Flush output
    sleep(2);

    printf("%s", arr8);
    wlen = write(fd, arr8, sizeof(arr8) - 1);
    tcdrain(fd);  // Flush output                                  /.?
    sleep(2);

    while (1) {
        // Read data from the module

        rdlen = read(fd, buf, sizeof(buf) - 1);
        if (rdlen > 0) {
            buf[rdlen] = '\0';
            printf("Received data: %s\n", buf,rdlen);
        }
    }

    close(fd);
    return 0;
}
```

5.5 Code snippet for the Stage 4 for STM32 Microcontroller.

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

```
/* Private macro
-------------------------------------------------------------*/
/* USER CODE BEGIN PM */
/* USER CODE END PM */
/* Private variables
-------------------------------------------------------------*/
I2C_HandleTypeDef hi2c1;
UART_HandleTypeDef huart1;
UART_HandleTypeDef huart2;
#define AHT25_ADDR 0x70 // Address of the AHT25 sensor
// AHT25 commands
#define AHT25_INIT_CMD 0xE1
#define AHT25_MEASURE_CMD 0xAC
float temperature, humidity;
char buffer[192];
/* USER CODE BEGIN PV */
/* USER CODE END PV */
/* Private function prototypes
-----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_I2C1_Init(void);
static void MX_USART1_UART_Init(void);
static void MX_USART2_UART_Init(void);
/* USER CODE BEGIN PFP */
void read_sensor_values(float *temperature, float *humidity);
void configure_be33_module(void);
/* USER CODE END PFP */
/* Private user code
-------------------------------------------------------------*/
/* USER CODE BEGIN 0 */
void configure_be33_module()
{
sprintf (&buffer[0], "CMD+RESET=0\r\n");
HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
       //   HAL_Delay(1000);
HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
HAL_Delay(2000);
```

```
sprintf (&buffer[0], "CMD+ADV=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(1000);
HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
                        HAL_Delay(2000);
sprintf (&buffer[0], "CMD+TXPWR=-4\r\n");
HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
                                                          //
HAL_Delay(1000);
HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
                        HAL_Delay(2000);
sprintf (&buffer[0], "CMD+SCAN=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(1000);
HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
                        HAL_Delay(2000);
sprintf (&buffer[0], "CMD+CON=1,dda383fa779f\r\n");
HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
                                                    //   HAL_Delay(1000);
HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
                  HAL_Delay(2000);
   // Send temperature and humidity data after establishing the connection
sprintf(buffer, "CMD+DATA=<conn_handle>,TEMP=%.2f,HUM=%.2f\r\n",
temperature, humidity);
HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
                                                    //   HAL_Delay(1000);
HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
                  HAL_Delay(2000);
}
```

```
void read_sensor_values(float *temperature, float *humidity)
{
   uint8_t data[6];
   uint8_t cmd = AHT25_MEASURE_CMD;
   HAL_I2C_Master_Transmit(&hi2c1, AHT25_ADDR, &cmd, 1, HAL_MAX_DELAY);
   HAL_Delay(100);
   HAL_I2C_Master_Receive(&hi2c1, 0x71, data, 6, HAL_MAX_DELAY);
   *humidity = ((float)((data[1] << 12) | (data[2] << 4) | (data[3] >> 4)))
/ 1048576.0 * 100.0;
   *temperature = ((float)(((data[3] & 0x0F) << 16) | (data[4] << 8) |
data[5])) / 1048576.0 * 200.0 - 50.0;
}
/* 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_I2C1_Init();
   MX_USART1_UART_Init();
   MX_USART2_UART_Init();
   /* USER CODE BEGIN 2 */
   /* USER CODE END 2 */
   /* Infinite loop */
   /* USER CODE BEGIN WHILE */
```

```
   /* USER CODE END WHILE */
        read_sensor_values(&temperature, &humidity);
        // Configure the B33 module and send data
        configure_be33_module();
        while(1)
        {
        sprintf(buffer, "CMD+DATA=<conn_handle>,TEMP=%.2f,HUM=%.2f\r\n",
temperature, humidity);
      HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
      HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
                                                          //
HAL_Delay(1000);
        HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
        HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
                        HAL_Delay(2000);
}
        // Add any additional functionality or delay if needed
   /* USER CODE BEGIN 3 */
   /* USER CODE END 3 */
}
/**
 * @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 I2C1 Initialization Function
 * @param None
 * @retval None
 */
static void MX_I2C1_Init(void)
{
  /* USER CODE BEGIN I2C1_Init 0 */
  /* USER CODE END I2C1_Init 0 */
  /* USER CODE BEGIN I2C1_Init 1 */
  /* USER CODE END I2C1_Init 1 */
  hi2c1.Instance = I2C1;
  hi2c1.Init.ClockSpeed = 100000;
  hi2c1.Init.DutyCycle = I2C_DUTYCYCLE_2;
  hi2c1.Init.OwnAddress1 = 0;
  hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c1.Init.OwnAddress2 = 0;
```

```
    hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
    hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
    if (HAL_I2C_Init(&hi2c1) != HAL_OK)
    {
        Error_Handler();
    }
    /* USER CODE BEGIN I2C1_Init 2 */
    /* USER CODE END I2C1_Init 2 */
}
/**
 * @brief USART1 Initialization Function
 * @param None
 * @retval None
 */
static void MX_USART1_UART_Init(void)
{
    /* USER CODE BEGIN USART1_Init 0 */
    /* USER CODE END USART1_Init 0 */
    /* USER CODE BEGIN USART1_Init 1 */
    /* USER CODE END USART1_Init 1 */
    huart1.Instance = USART1;
    huart1.Init.BaudRate = 115200;
    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 = 115200;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
      Error_Handler();
  }
  /* USER CODE BEGIN USART2_Init 2 */
  /* USER CODE END USART2_Init 2 */
}
/**
 * @brief GPIO Initialization Function
 * @param None
 * @retval None
 */
static void MX_GPIO_Init(void)
{
  /* USER CODE BEGIN GPIO_Init 1 */
  /* USER CODE END GPIO_Init 1 */
  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOB_CLK_ENABLE();
  /* USER CODE BEGIN GPIO_Init 2 */
  /* USER CODE END GPIO_Init 2 */
}
/* 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 can add his own implementation to report the HAL error return
state */
   __disable_irq();
   while (1)
   {
   }
   /* USER CODE END Error_Handler_Debug */
}
#ifdef USE_FULL_ASSERT
/**
 * @brief  Reports the name of the source file and the source line number
 *         where the assert_param error has occurred.
 * @param  file: pointer to the source file name
 * @param  line:
void assert_failed(uint8_t *file, uint32_t line)
{
   /* USER CODE BEGIN 6 */
   /* User can add his own implementation to report the file name and line
number,
    ex: printf("Wrong parameters value: file %s on line %d\r\n", file,
line) */
   /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

5.6 Code snippet for the Stage 4 for RBA5D2X

```
#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);
    cfsetospeed(&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 flow control */

    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, rdlen;

    char arr1[] = "CMD+RESET=0\r\n";



    char arr5[] = "CMD+TXPWR=-4\r\n";
    char arr6[] = "CMD+SCAN=1\r\n";
    char arr8[] = "CMD+CON=1,fab321a20744\r\n";



    unsigned char buf[192];

    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, B115200);

    printf("%s", arr1);
    wlen = write(fd, arr1, sizeof(arr1) - 1);
```

```
 tcdrain(fd);  // Flush output

 sleep(2);

 printf("%s", arr5);
 wlen = write(fd, arr5, sizeof(arr5) - 1);
 tcdrain(fd);  // Flush output

 sleep(2);

printf("%s", arr6);
 wlen = write(fd, arr6, sizeof(arr6) - 1);
 tcdrain(fd);  // Flush output
 sleep(2);

 printf("%s", arr8);
 wlen = write(fd, arr8, sizeof(arr8) - 1);
 tcdrain(fd);  // Flush output                                /.?
 sleep(2);

 while (1) {
    // Read data from the module

    rdlen = read(fd, buf, sizeof(buf) - 1);
    if (rdlen > 0) {
        buf[rdlen] = '\0';
        printf("Received data: %s\n", buf,rdlen);
    }
 }
```

```
    close(fd);
    return 0;
}
```

5.7 Code snippet for the Stage 5 for RBA5D2X to send the data to cloud.

```
#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);
    cfsetospeed(&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 flow control */

    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;
}

void send_first_set_of_commands(int fd) {
    char arr1[] = "CMD+RESET=0\r\n";
    char arr4[] = "CMD+ADV=1\r\n";
    char arr5[] = "CMD+TXPWR=-4\r\n";
    char arr6[] = "CMD+SCAN=1\r\n";
//    char arr8[] = "CMD+CON=1,fab321a20744\r\n";

    printf("%s", arr1);
    write(fd, arr1, sizeof(arr1) - 1);
```

```
    tcdrain(fd);  // Flush output
    sleep(2);

    printf("%s", arr4);
    write(fd, arr4, sizeof(arr4) - 1);
    tcdrain(fd);  // Flush output
    sleep(2);

    printf("%s", arr5);
    write(fd, arr5, sizeof(arr5) - 1);
    tcdrain(fd);  // Flush output
    sleep(2);

    printf("%s", arr6);
    write(fd, arr6, sizeof(arr6) - 1);
    tcdrain(fd);  // Flush output
    sleep(2);

//    printf("%s", arr8);
  //  write(fd, arr8, sizeof(arr8) - 1);
    //tcdrain(fd);  // Flush output
    //sleep(2);
}

void send_second_set_of_commands(int fd) {
    char arr1_new[] = "CMD+RESET\r\n";
    char arr2[] = "CMD+WIFIMODE=1\r\n";
    char arr3[] = "CMD+CONTOAP=\"Realme 5.0GHz\",\"12345678\"\r\n";
    char arr4[] = "CMD+MQTTNETCFG=dev.rightech.io:1883\r\n";
```

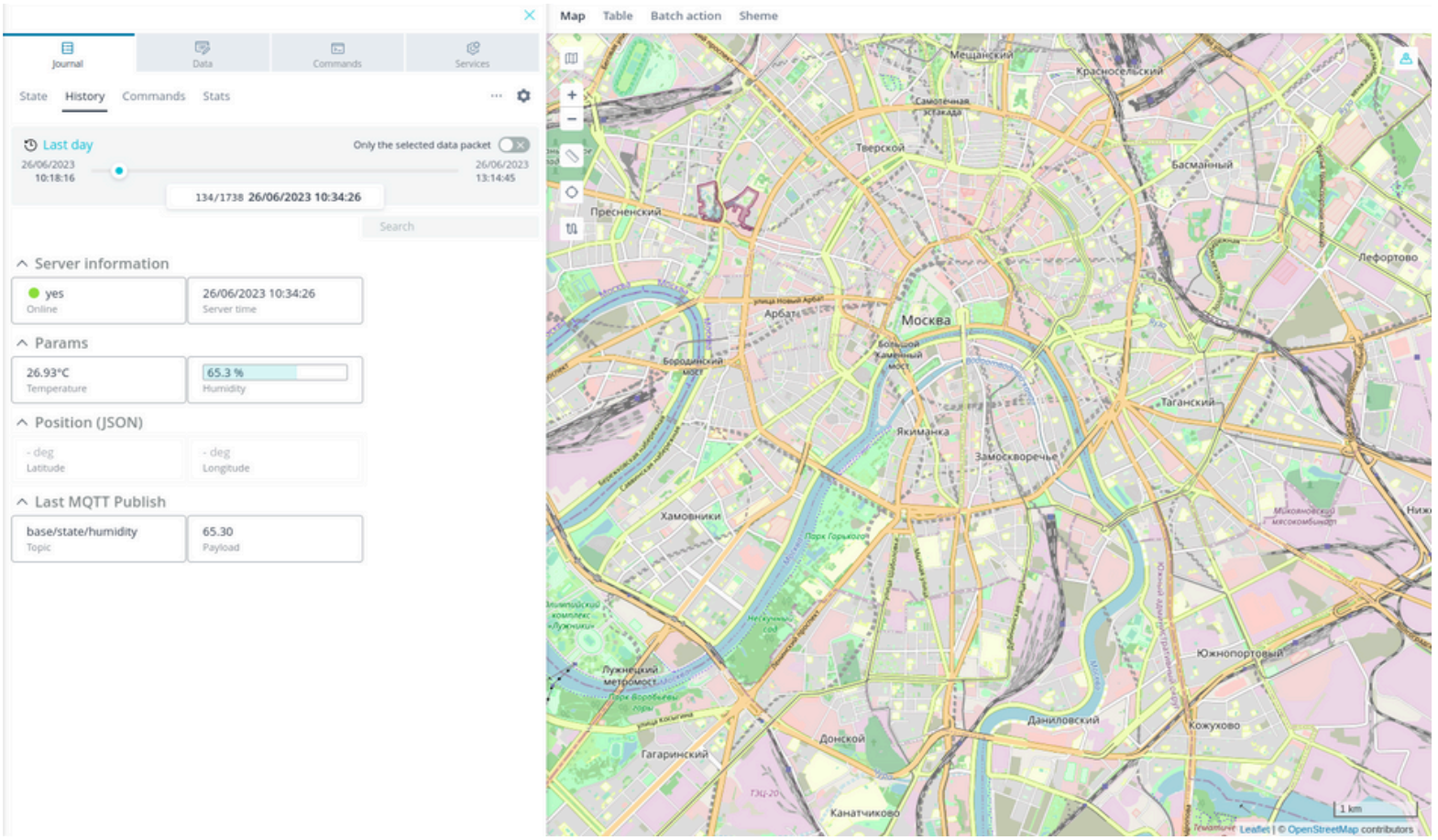
```
    char arr5[] =
"CMD+MQTTCONCFG=3,mqtt-elmanjunath04-8gnaz2,,,,,,,,,\r\n";
    char arr6[] = "CMD+MQTTSTART=1\r\n";
    char arr7[] =  "CMD+MQTTSUB=base/state/temperature\r\n";
    char arr8[] = "CMD+CON=1,fab321a20744\r\n";

    // Add other commands for the second set as needed

    printf("%s", arr1_new);
    write(fd, arr1_new, sizeof(arr1_new) - 1);
    tcdrain(fd);  // Flush output
    sleep(2);

    printf("%s", arr2);
    write(fd, arr2, sizeof(arr2) - 1);
    tcdrain(fd);  // Flush output
    sleep(2);
printf("%s", arr3);
    write(fd, arr3, sizeof(arr3) - 1);
    tcdrain(fd);  // Flush output
    sleep(2);
printf("%s", arr4);
    write(fd, arr4, sizeof(arr4) - 1);
    tcdrain(fd);  // Flush output
    sleep(2);
printf("%s", arr5);
    write(fd, arr5, sizeof(arr6) - 1);
    tcdrain(fd);  // Flush output
    sleep(2);
```

```
printf("%s", arr6);
    write(fd, arr6, sizeof(arr6) - 1);
    tcdrain(fd);  // Flush output
    sleep(2);
printf("%s", arr7);
    write(fd, arr7, sizeof(arr7) - 1);
    tcdrain(fd);  // Flush output
    sleep(2);
 printf("%s", arr8);
    write(fd, arr8, sizeof(arr8) - 1);
    tcdrain(fd);  // Flush output
    sleep(2);

    // Add other commands for the second set as needed
}

int main() {
    char *portname1 = "/dev/ttyS3";  // Port for the first set of commands
    char *portname2 = "/dev/ttyS4";  // Port for the second set of commands

    int fd1, fd2;
    int rdlen;
    unsigned char buf[192];

    // Open the first serial port for the first set of commands
    fd1 = open(portname1, O_RDWR | O_NOCTTY | O_SYNC);
    if (fd1 < 0) {
        printf("Error opening %s: %s\n", portname1, strerror(errno));
        return -1;
```

```
}

// Open the second serial port for the second set of commands
fd2 = open(portname2, O_RDWR | O_NOCTTY | O_SYNC);
if (fd2 < 0) {
    printf("Error opening %s: %s\n", portname2, strerror(errno));
    close(fd1);
    return -1;
}

// Set serial interface attributes for the first set of commands
set_interface_attribs(fd1, B115200);

send_first_set_of_commands(fd1);

// Set serial interface attributes for the second set of commands
set_interface_attribs(fd2, B115200);

send_second_set_of_commands(fd2);

while (1) {
    // Your existing code for reading and processing data from both ports
    rdlen = read(fd1, buf, sizeof(buf) - 1);
    if (rdlen > 0) {
        buf[rdlen] = '\0';
        printf("Received data from port 1: %s\n", buf);
    }

    rdlen = read(fd2, buf, sizeof(buf) - 1);
```

```
        if (rdlen > 0) {
            buf[rdlen] = '\0';
            printf("Received data from port 2: %s\n", buf);
int ret = snprintf(buf, sizeof(buf),
"CMD+MQTTPUB=base/state/temperature,%s\r\n;

    if (ret < 0) {

    } else {

        ssize_t wlen = write(fd2, buf, ret);
        sleep(3);
        if (wlen == -1) {
        }
}
}
}

    // Close both ports
    close(fd1);
    close(fd2);

    return 0;
}
```

RESULT AND CONCLUSION



Results:

1. Temperature Data Transmission:

- ❖ Verify that temperature data from the AHT25 sensor is successfully acquired by the STM32 microcontroller.
- ❖ Confirm that the STM32 successfully communicates with the BE33 module and sends temperature data to the RBA5D2X device.

2. Communication Reliability:

- ❖ Evaluate the reliability of data transmission between the STM32 and

the RBA5D2X device through the BE33 module.

- Check for any errors, interruptions, or data corruption during transmission.

3. Sensor Accuracy:

- Assess the accuracy of temperature readings obtained from the AHT25 sensor.
- Compare the sensor readings with a known reference to validate accuracy.

4. System Stability:

- Ensure the stability and robustness of the entire system under various conditions and environmental factors.

The results of sending data to the Rightech Cloud Platform will depend on various factors, including the successful implementation of the data transmission process, the reliability of the communication link, and the responsiveness of the Rightech Cloud services. Here are potential results and considerations for this stage:

5.Successful Data Transmission:

- If everything is implemented correctly, data should be successfully transmitted from the STM32 microcontroller to the Rightech Cloud Platform.

6.Data Accuracy and Integrity:

- Check whether the data received on the Rightech Cloud matches the data sent from the STM32 microcontroller. Ensure that the integrity of the data is maintained throughout the transmission.

7.Real-time or Batch Processing:

- Depending on your project requirements, evaluate whether the data is processed in real-time or in batches on the Rightech Cloud Platform. This may affect the timeliness of data availability for analysis.

Conclusion:

1. Achievements:

- Summarize the successful aspects of the project, such as the successful communication between devices, accurate temperature readings, and overall system reliability.

2. Challenges and Solutions:

- Discuss any challenges encountered during the project and the solutions implemented to overcome them.

3. Improvements:

- Suggest any potential improvements or optimizations for future iterations of the project.

4. Lessons Learned:

- Share insights and lessons learned during the project development, including any unexpected issues and how they were addressed.

5. Future Work: Outline potential future developments or enhancements for the project, such as additional features, scalability, or compatibility with other devices.

Real-Time Application Scenarios:

1. **Smart Agriculture:**
    - Deploy sensors on farmland or in greenhouses.
    - Monitor temperature conditions in real time.
    - Optimize irrigation and ventilation systems based on the collected data.
2. **Cold Chain Logistics:**
    - Install sensors in refrigerated transport vehicles or storage facilities.
    - Ensure the temperature remains within the desired range for transporting perishable goods.
3. **Industrial Processes:**
    - Integrate sensors into industrial processes where temperature control is critical.
    - Implement real-time monitoring to prevent equipment overheating.
4. **Building Automation:**
    - Use sensors to monitor and control room temperature in smart buildings.
    - Optimise HVAC systems based on real-time data.
5. **Healthcare:**
    - Implement temperature monitoring in medical storage facilities.
    - Ensure vaccines and medicines are stored at the appropriate temperatures

Thank you