BLE (Bluetooth Low Energy)

Bluetooth Low Energy (BLE) is a low power wireless technology used to connect devices to each other, highly targeted for applications in heartbeat sensors, smartphones, smartwatch fitness, beacons, security, and home entertainment industries. Unlike the **Bluetooth** that is always on, **BLE** remains in sleep mode except when a connection is initiated, making the device low power consumption. Devices that work with BLE can have two different functions on a connection: **central device** or **peripheral device**.

Central devices: Usually, central devices receive data. Example: tablets, smartphones, computers, etc.

Peripheral devices: These are sensors and low power devices that connect to the central device.

In this tutorial, the **SiP HTLRBL32L** Microcontroller will act as a peripheral device that will send and receive data from a central device, which in our case is a Smartphone. The **SiP HTLRBL32L** is ready for applications using Bluetooth® Low Energy 5.2 providing excellent performance with minimum power consumption, enabling applications with years of battery life and flexibility for the IoT (Internet of Things) ecosystem.

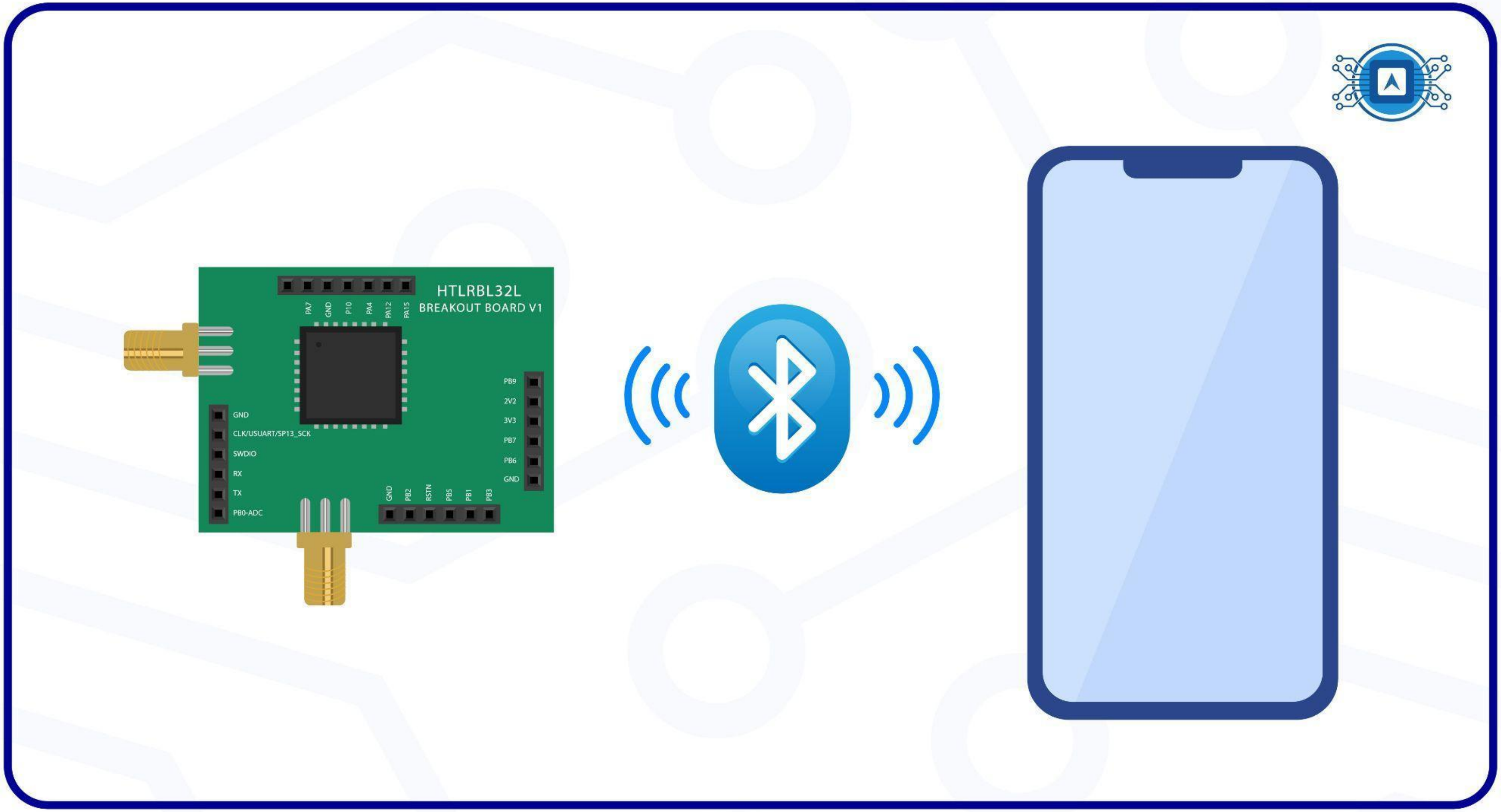

Image 1: BLE with HTRBL32L. Source: *The author.*

Necessary tools:

Hardware - Components

- HTLRBL32L board;
- FTDi module for connecting the board to the computer
- Protoboard;
- Jumpers;
- SmartPhone with BLE.

SOFTWARE:

- **Wise Studio** IDE to compile the code;
- **Termite** to visualize the board's serial;
- **RF-flasher** software to write the firmware to the board;
- Bluetooth test app (GATTBrowser);
- Git installed

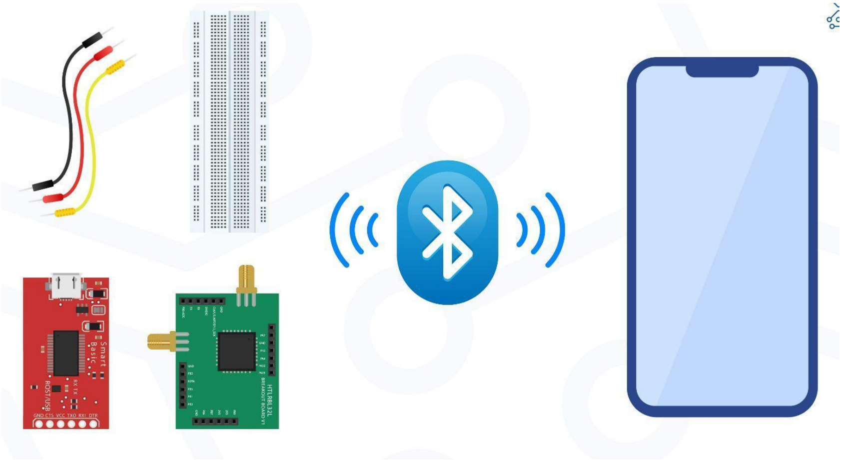

Image 2: Components. Source: *The author.*

1. Protocol

There are basically two important protocols in the communication between two **BLE** devices: GAP and GATT.

1.1 GAP Protocol:

GAP **(Generic Access Profile)** defines how BLE-enabled devices can become available and how these two devices can communicate directly with each other.

1.2 GATT Protocol:

GATT is an acronym for **Generic ATTribute Profile** and defines the specifications for how two BLE devices transfer data from one side to the other, using the concepts of **Service** and **Characteristic**. In this protocol, the central devices act as clients and any peripheral device is the server.

In a simple way, the discovery of the devices is done using the **GAP** protocol. After discovery, the communication between devices is completed through the **GATT** protocol.

2. BLE Characteristic

A **Characteristic** in the context of BLE represents information that a server wants to expose to a client. For example, the heartbeat "**Characteristic**" represents the heartbeat monitoring in BPM of a device that can be read by a client. The **Characteristic** contains other attributes, such as:

➔ **Value:** Characteristic's data value.
➔ **Declaration:** Characteristic's properties (location, read, write, notify, etc.);
➔ **Description:** ASCII string that describes the Characteristic
➔ **Service:** And a group of Characteristics.
➔ **UUID (Universally Unique Identifier):** is the unique identification code of a specific Service. It can be 16 or 128 bits depending on the service.

3. Using the Wise Studio IDE.

The tutorial will use the cloning of the "**PushButton_Bluetooth**" project as an example, available in the [repository](). After the cloning procedure, we will use the **[Wise Studio]() IDE** to compile the code. To perform this procedure, open the "**File**" tab in **Wise Studio**, click on "**Open projects from file System**". Next click on "**Directory**", select the downloaded or cloned folder and then click **Finish**. Next, open the "**Includes**" folder and find the "**HT_ble_api.h**" library to view the **UUID** information, where we set the **Service** and **Characteristic** properties as shown in image 3.

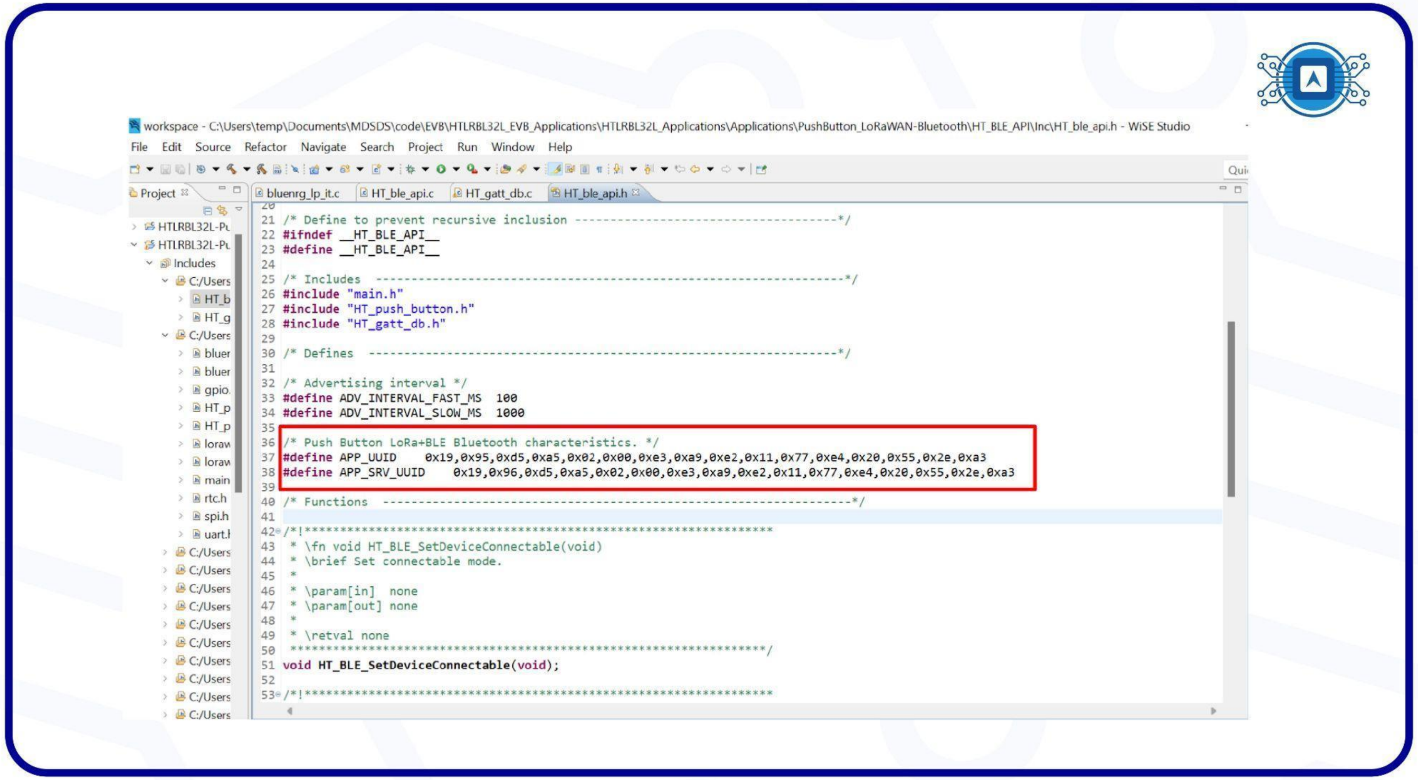

Image 3: BLE with Wise Studio. Source: *Screenshot by the author*.

2.1 Build Project

To compile all the code and consequently build the binary code, right-click on the main project folder and click "**Build Project**", as shown in image 4.

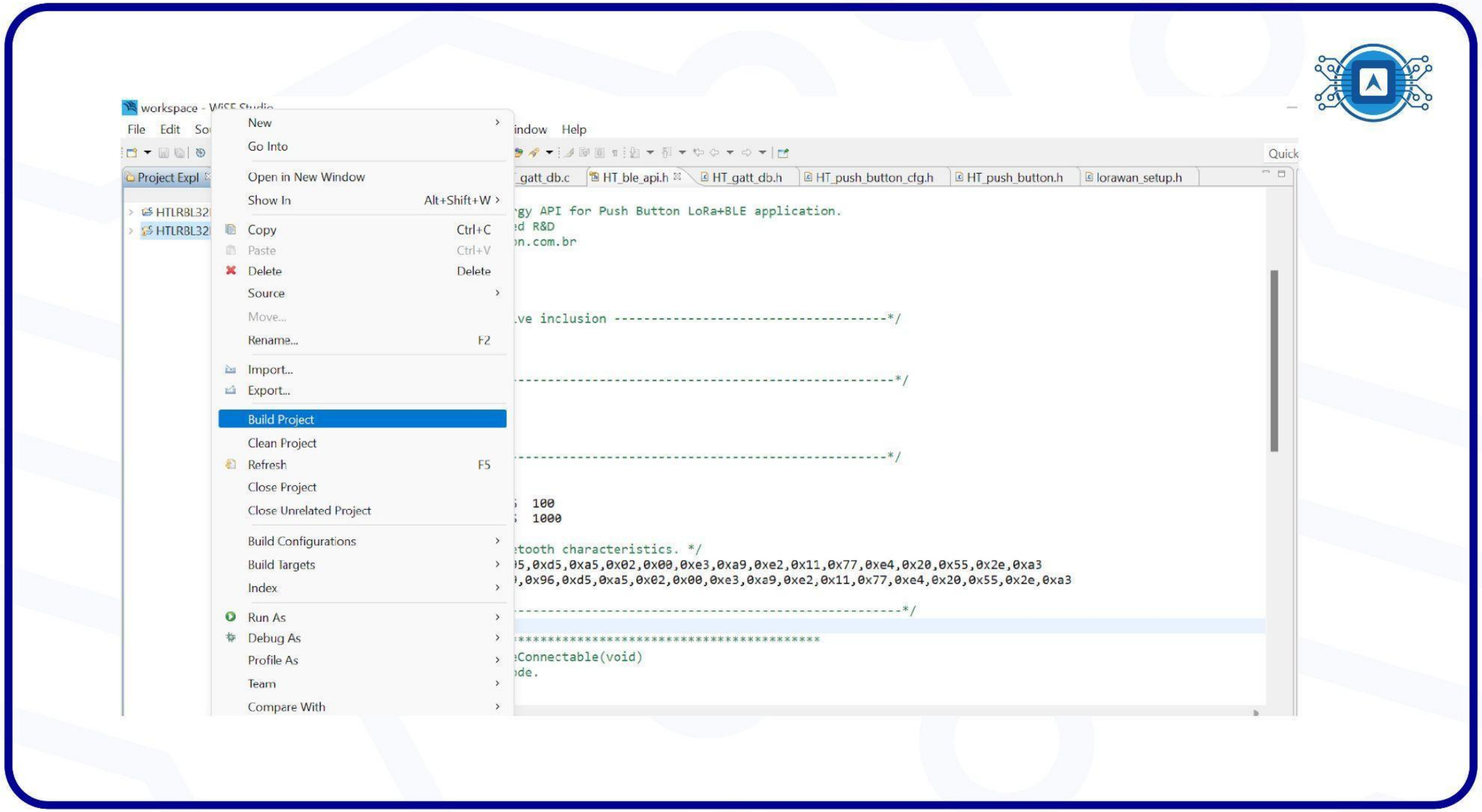

Image 4: Generating the binary. Source: *Screenshot by the author*.

3. Circuit assembly to enter record mode.

The next step is to assemble the board according to the electronic diagram shown in picture 5. The microcontroller's **GPIO PA10** connector should be at high level, that is, connected to 3.3v on the board itself, activating the UART bootloader.

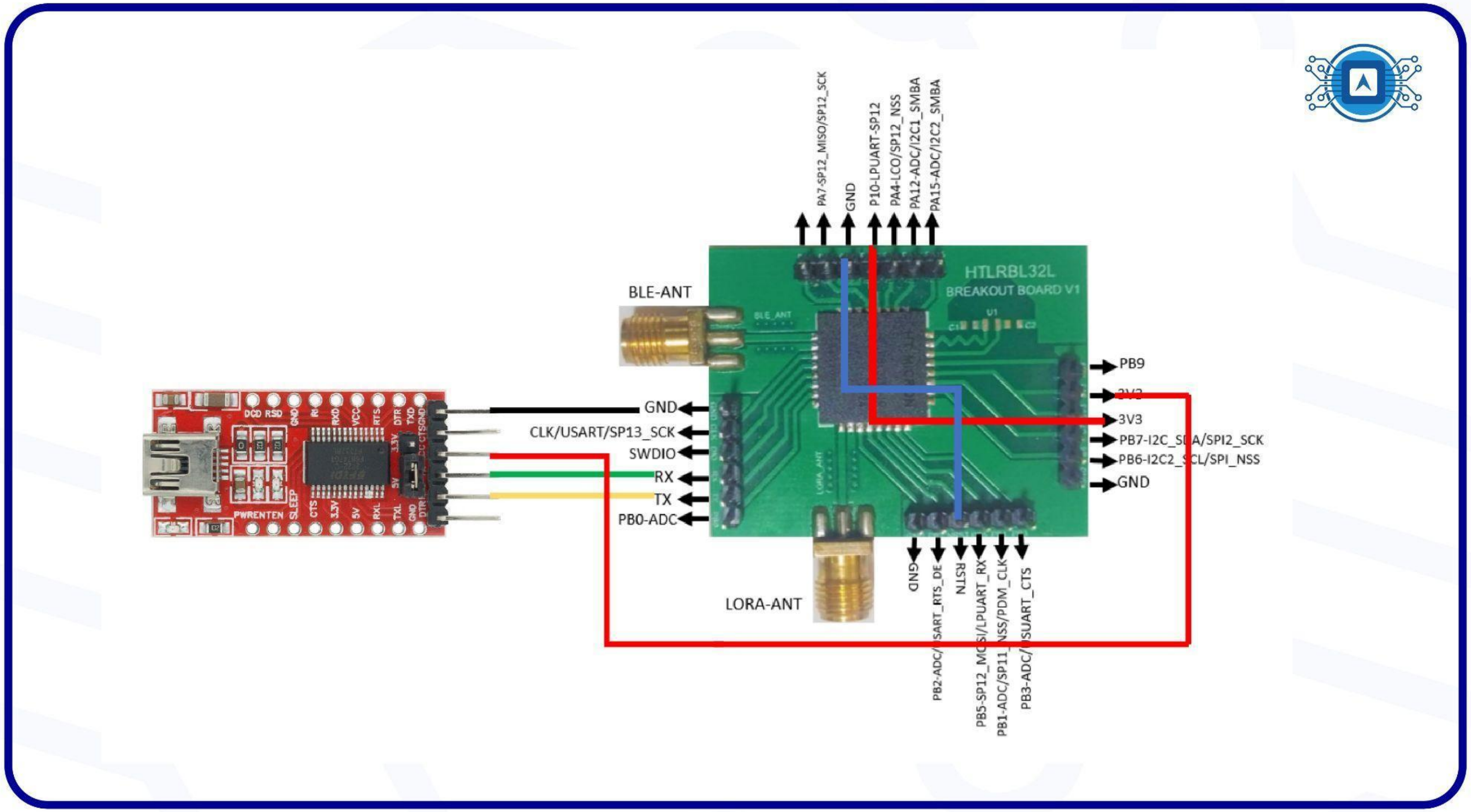

Image 5: Recording schematic. Source: *Screenshot by the author*.

4. Write the Code on the board.

The firmware will be written using the **RF-flasher** software. The procedure for writing the firmware using RF-flasher is in the text **Firmware Recording and Running Tests**.

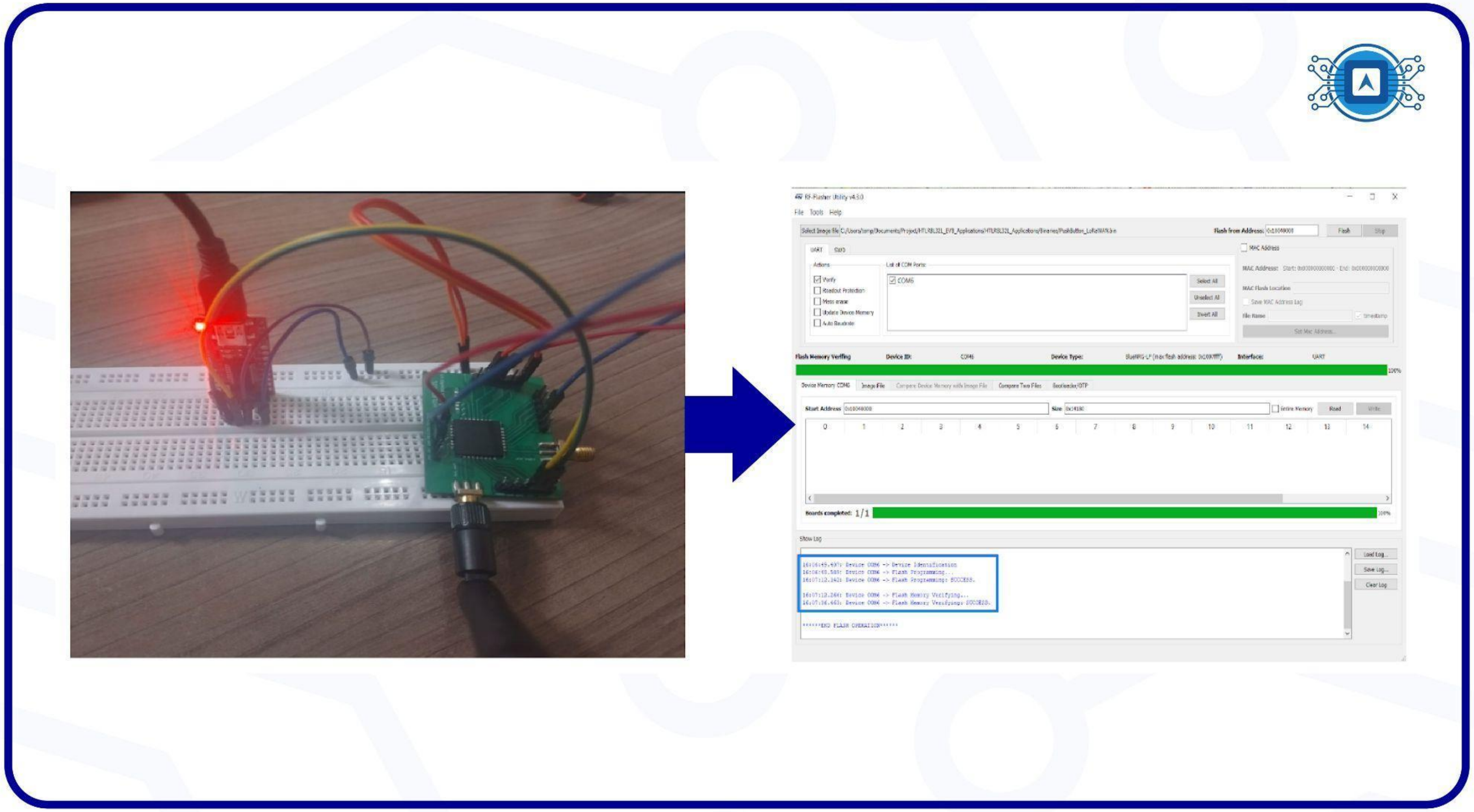

Image 6: Flashing the Firmware. Source: *The author.*

5. GATTBrowser - APP.

To perform the data communication test with the Smartphone, we need an application that can intermediate this communication. Therefore, we will use **GATTBrowser**, a free application that uses the GATT communication protocol. On your Smartphone, go to “**playStore**” and install the “**GATTBrowser**” application. Then open the app and activate the Bluetooth on your Smartphone, it will scan it to find the **BLE** devices nearby. In our case it will find the **HTLRBL32L** microcontroller, which will be identified as **PushButton**. In this scenario, the Smartphone acts as the client and the HTLRBL32L microcontroller, which has the BLE communication protocol, is the server.

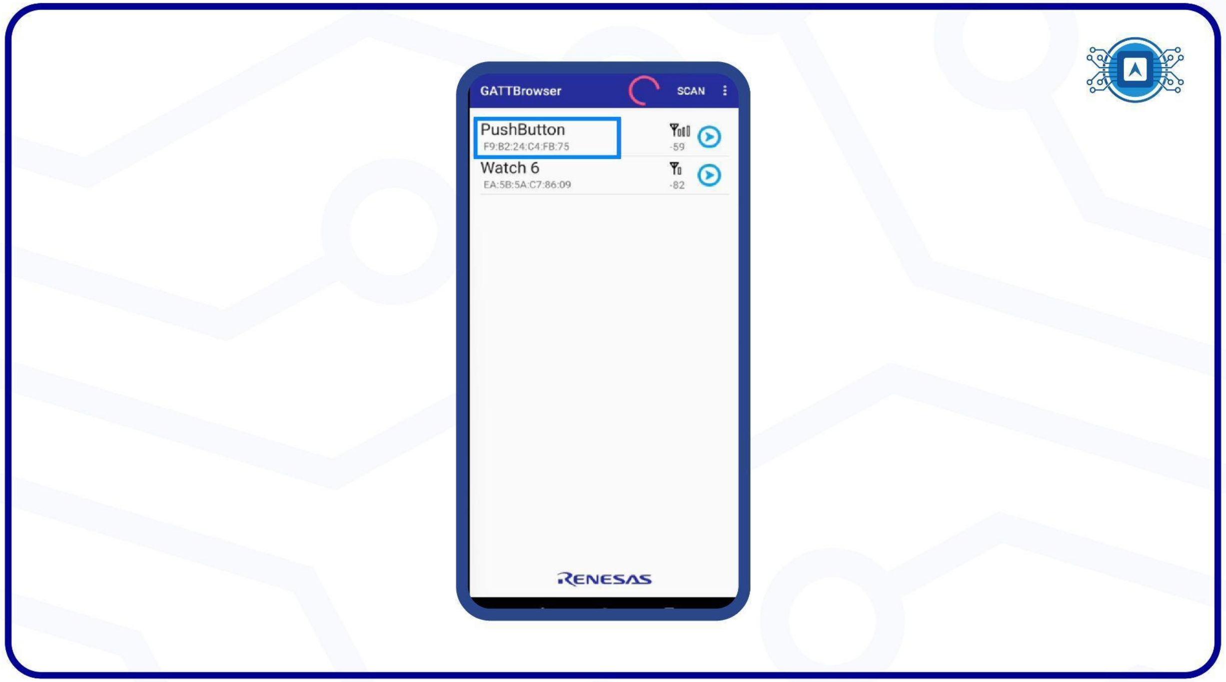

Image 7: PushButton. Source: *The author*.

6. Connecting the Smartphone with SiP HTLRBL32L

Before we proceed, open your serial terminal (**Termite**) to view the microcontroller data. After this procedure, click on the button next to the "**PushButton**" in the **GATTBrowser** app and the smartphone will start connecting to the **HTLRBL32L** microcontroller. It can be seen through Termite, that the **microcontroller** has successfully made this connection with the SmartPhone, as shown in image 8.

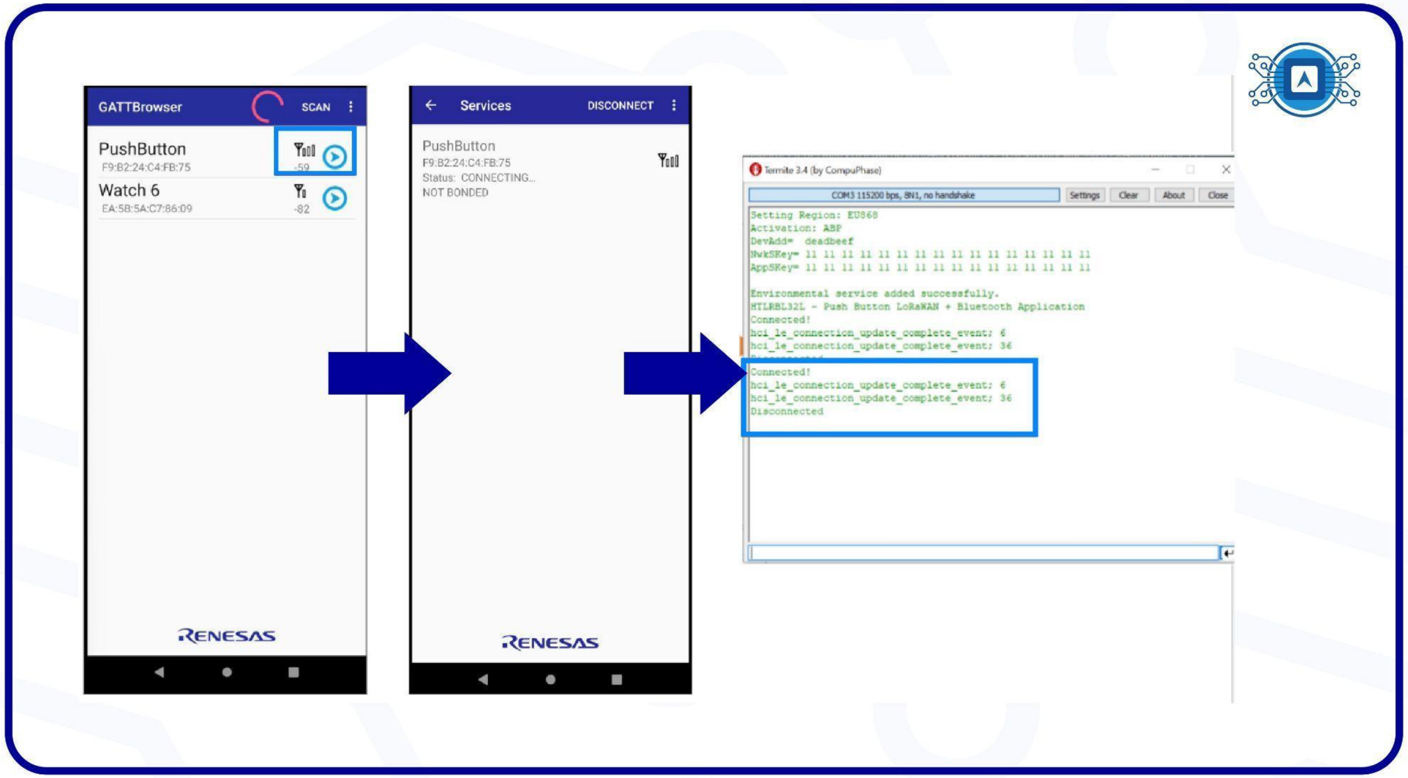

Image 8: Connecting the Smartphone with SiP HTLRBL32L. Source: *The author.*

Once connected, it will display the list of **Service** and their **Characteristic**. We can see the **Service UUID** in the App.

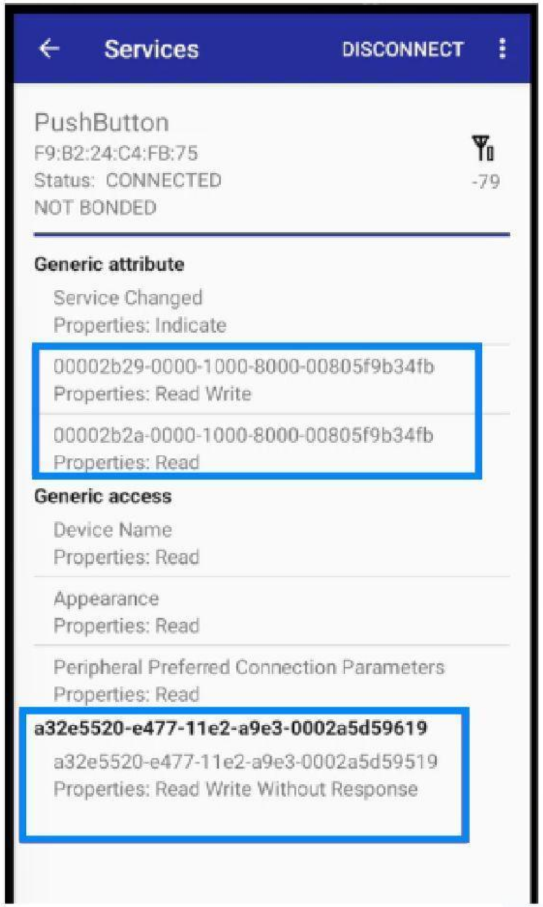

Image 9: Service UUID. Source: *The author.*

7. Cellular data communication test

In the **READ** button the user will receive information from the microcontroller, in the **Write** button the user will send information to the microcontroller, as shown in image 10.

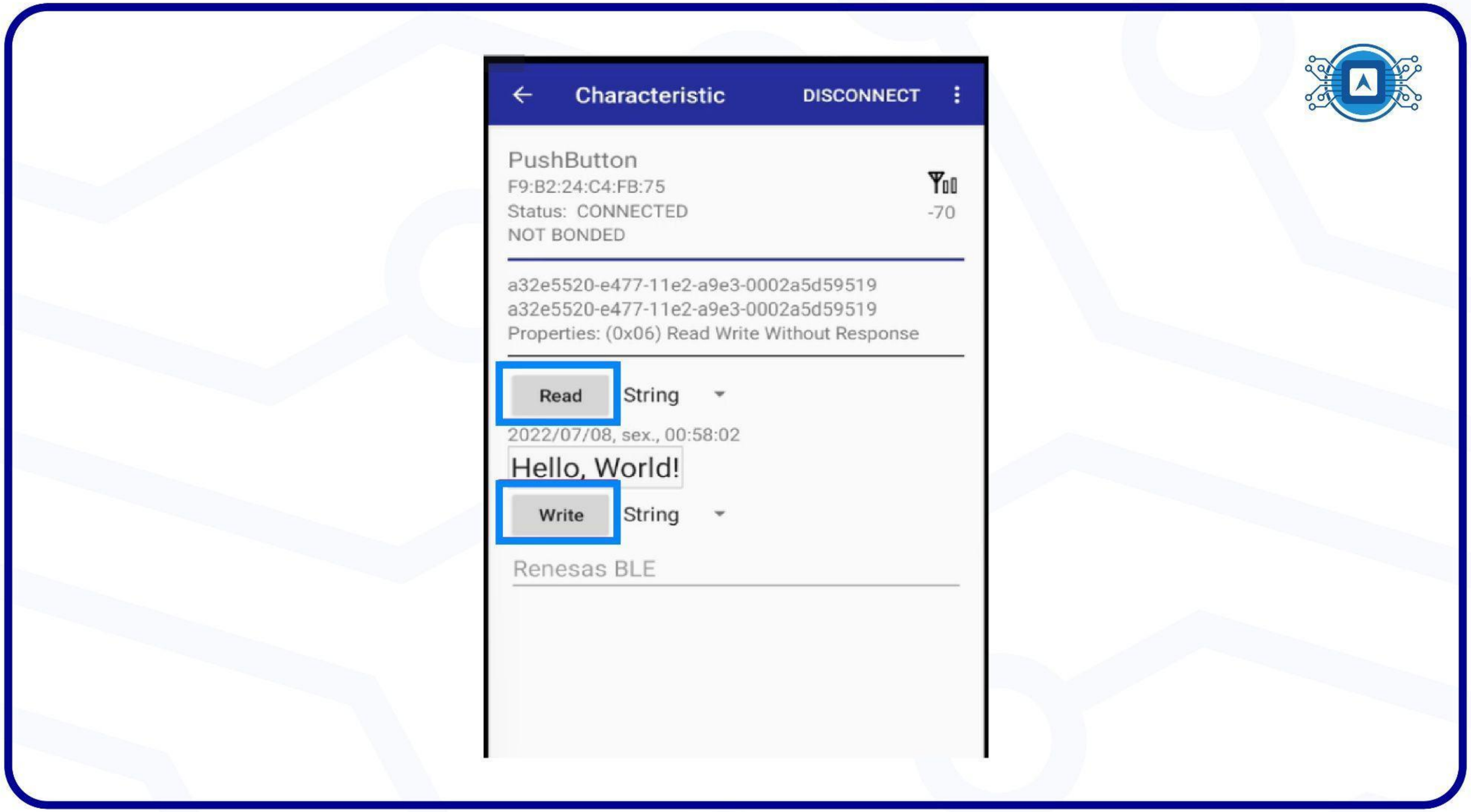

Image 10: Mobile phone data communication test. Source: *The author.*

7.1 Receiving data from the microcontroller.

Start the tests by receiving string information from the microcontroller. To perform this procedure, press the **READ** button and the user will receive the following message on his Smartphone, “**Hello, World!**” As shown in image 11.

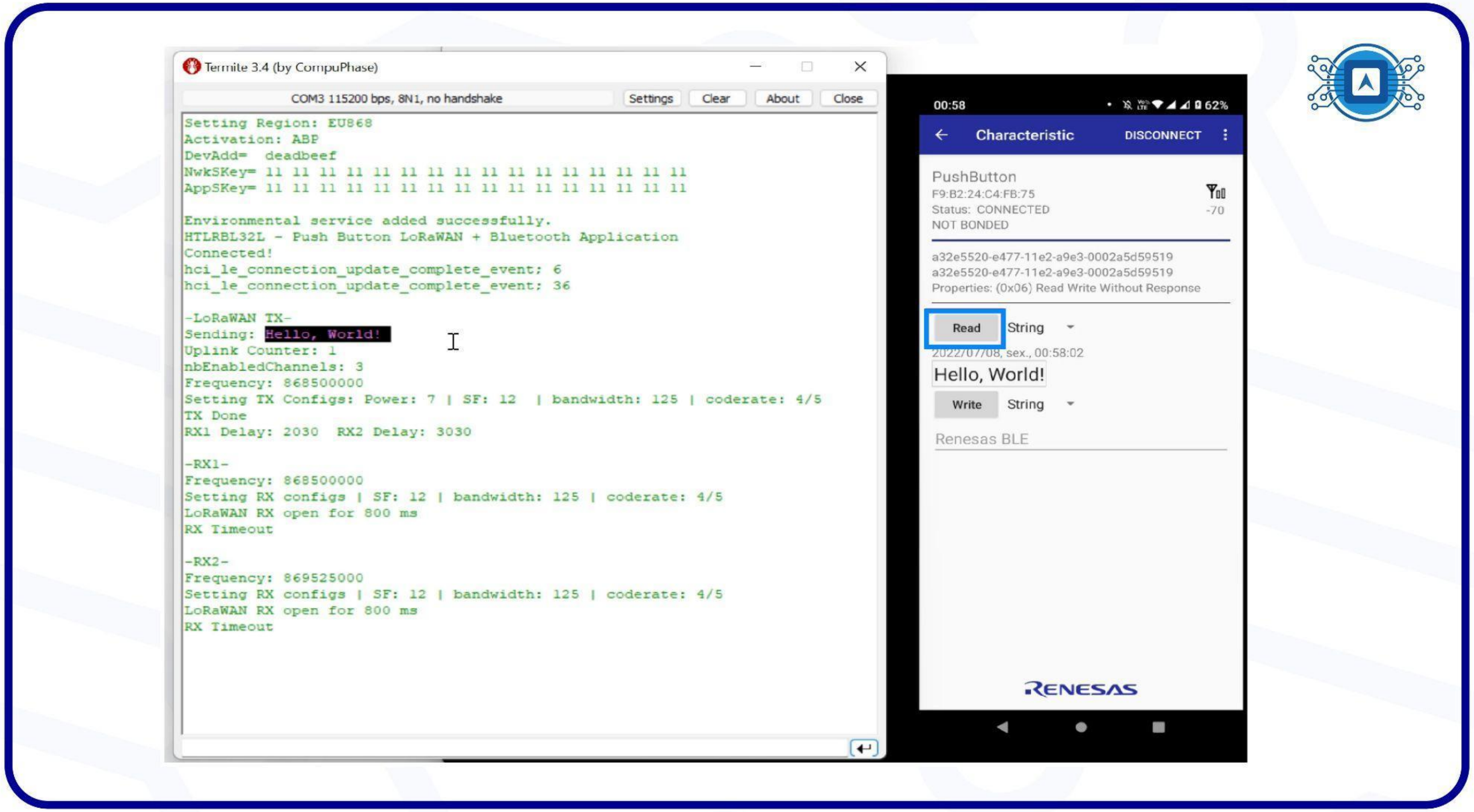

Image 11: READ communication test. Source: *The author*.

7.2 Sending data to the Microcontroller.

In the Write button the user will send information to the microcontroller. We can type any sentence, for example: “Manaus-Am”, the microcontroller will receive it through the BLE protocol.

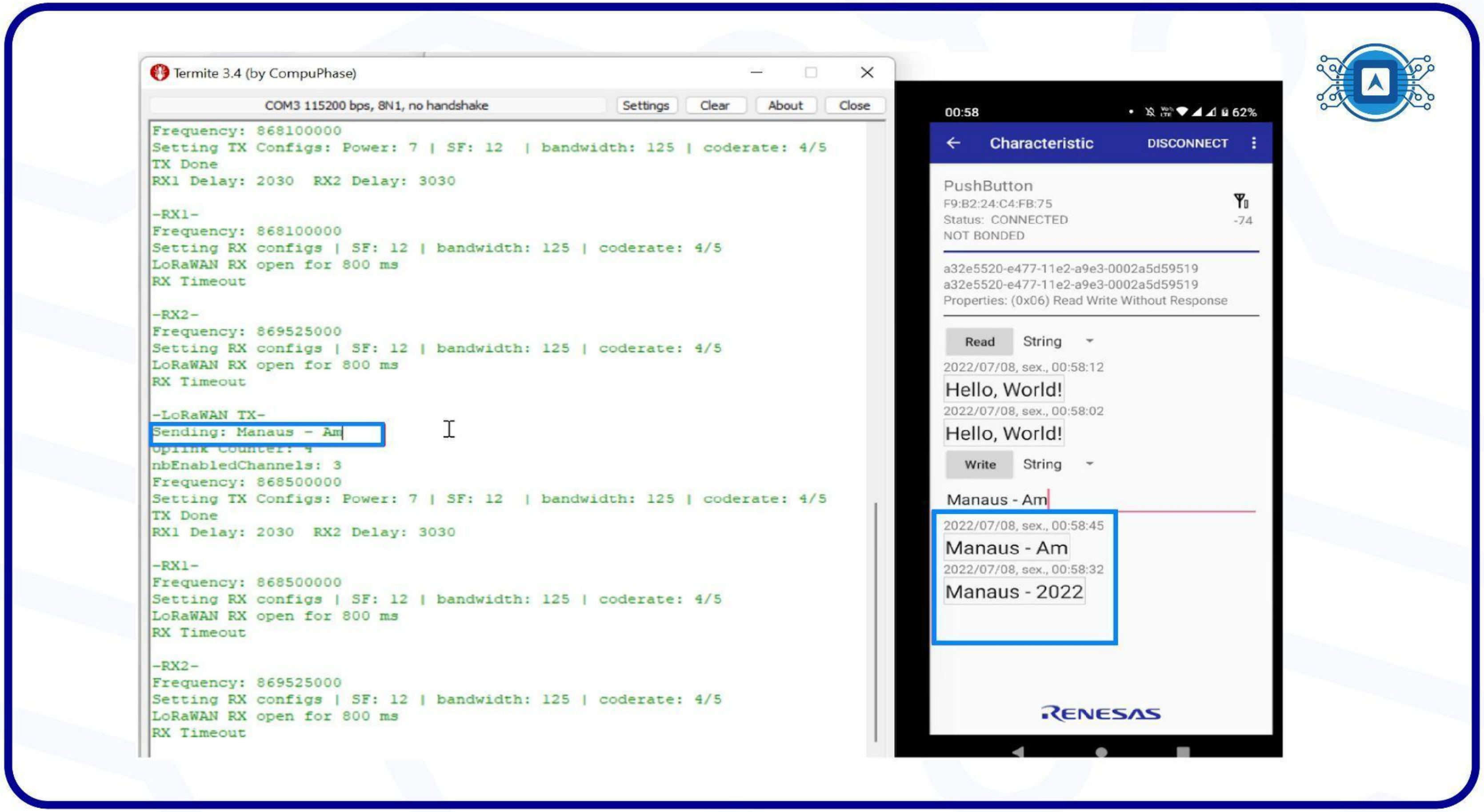

Image 12: Writing communication test. Source: *The author.*

In this tutorial the basics of Bluetooth Low Energy were presented, starting with a practical example using the **HTLRBL32L** microcontroller. The idea is to use BLE to send or receive sensor readings from other devices.

References

BLUETOOTH. **Intro to Bluetooth Generic Access Profile (GAP)**. 2022. Available at: <https://www.bluetooth.com/bluetooth-resources/intro-to-bluetooth-generic-access-profile-gap/>. Accessed on july 28th 2022.

ELECTRONICSHUB. **How to use BLE in ESP32? ESP32 BLE (Bluetooth Low Energy) Tutorial**. 2021. Available at: <https://www.electronicshub.org/esp32-ble-tutorial/>. Accessed on july 28th 2022.

NOVELBITS. **Basics bluetooth low energy.** 2022. Available at: **<** https://novelbits.io/basics-bluetooth-low-energy/ **>.** Accessed on july 28th 2022.