VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Huy Nguyen

# INDUSTRIAL 16-CHANNEL RELAY MODULE UTILIZING MODBUS RTU PROTOCOL

Technology and Communication
2023

## ACKNOWLEDGEMENTS

Firstly, I would like to thank Mr. Smail Menani for his great help to give pros and cons for my thesis project and for guiding not only the thesis but also my study at school. I was able to complete my thesis and get a valuable experience such as technical skills and project plan which is great preparation for my future career path.

Secondly, I would like to express my appreciation to Mr. Thong who is my mentor, for giving me the precious opportunity to work at TMA Embedded Team. That gave me great guidance and a chance to work in a professional working environment with a friendly team who supports me when I struggle.

Finally, I want to thank my family and friends from the bottom of my heart. They always inspired me so much and nothing can describe my gratitude to them. I want to spread my gratitude to all my teachers and friends at VAMK and in Finland, who have supported me during my whole studying time and living far from my home.

Huy Nguyen,
Vaasa, March 20th 2023

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Information Technology

## ABSTRACT

| | |
|---|---|
| Author | Huy Nguyen |
| Title | Industrial 16-channel Relay Module Utilizing Modbus RTU Protocol |
| Year | 2023 |
| Language | English |
| Pages | 34 + 1 Appendices |
| Name of Supervisor | Smail Menani |

The purpose of this thesis aims to develop a system to remotely monitor and control the assets of a water company.

The used technology to implement the project is Modbus RTU, a personal computer with a USB to RS485 adapter to communicate with the device, a server to allow the user to collect the required data to be sent to the device through LoRaWAN networks and ChirpStack Server.

The main objective of the thesis is successfully attained and resulted in the development of an embedded device enabling customers to connect and remotely monitor and control the water company assets.

Keywords Industrial, Mobus RTU, LoRaWan

## CONTENTS

ACKNOWLEDGEMENTS

ABSTRACT

1 INTRODUCTION 9

2 TECHNOLOGIES USED 10

2.1 C/C++ Language 10

2.3 Python Language 10

2.4 Microcontroller 10

2.5 MODBUS RTU 11

2.6 RS485 12

3 DEVICE DEFINITION 13

3.1 DEVICE REQUIREMENT 13

3.1.1 MODBUS FUNCTION 13

3.1.2 LORAWAN 14

3.2 APPLICATION FEATURES 14

3.3 APPLICATION SPECIFICATIONS 14

4 DEVICE FUNCTION 17

4.1 READ THE STATE OF THE RELAYS (0x01) 18

4.2 READ THE STATE OF THE INPUTS (0x02) 19

4.3 READ THE ADDRESS AND VERSION (0x03) 21

4.4 WRITE A SINGLE AND ALL RELAYS (0x05) 23

4.5 SET BAUD RATE AND ADDRESS (0x06) 26

5 IMPLEMENTATION & TESTING 27

5.1 IMPLEMENTATION 27

5.2 TESTING 30

6 CONCLUSIONS 31

REFERENCES 33

**LIST OF ABBREVIATIONS**

| | |
|---|---|
| MCU | Microcontroller unit |
| CRC | A Cyclic redundancy check |
| OTA | Over-the-Air |
| LPWAN | A low power wide area network |

**LIST OF FIGURES AND TABLES**


| | | |
|---|---|---|
| **Figure 1.** | Modbus RTU Relay | p. 9 |
| **Figure 2.** | STM32 32-bit Arm | p. 11 |
| **Figure 3.** | How does Modbus RTU protocol work | p. 11 |
| **Figure 4.** | MCU with Modbus RTU RS and 485 protocol | p. 12 |
| **Figure 5.** | Functions code and their description | p. 13 |
| **Figure 6.** | A device with LoRaWAN node | p. 14 |
| **Figure 7.** | RS485 Communication | p. 15 |
| **Figure 8.** | Relay Connection | p. 15 |
| **Figure 9.** | A device in a 3D projection | p. 16 |
| **Figure 10.** | A device communicates with a server | p. 16 |
| **Figure 11.** | A structure consisting message and device | p. 17 |
| **Figure 12.** | Separate slave messages into 3 types | p. 17 |
| **Figure 13.** | Reading the state of the relays code and result | p. 18 |
| **Figure 14.** | Reading the state of inputs code and result | p. 20 |
| **Figure 15.** | Reading device address code and result | p. 21 |
| **Figure 16.** | Reading software version code and result | p. 22 |
| **Figure 17.** | Controlling relay code and result | p. 24 |
| **Figure 18.** | Setting baud rate code | p. 25 |

**Figure 19.** Setting information with Modbus Application p. 27

**Figure 20.** Control and list all information on relays p. 28

**Figure 21.** Website and Thingsboard server p. 29

**Figure 22.** Website and Thingsboard server p. 30

**Figure 23.** Results when test checking states of input p. 31

**LIST OF APPENDICES**

**APPENDIX 1.** Code and flowchart of the device

# 1 INTRODUCTION

Nowadays, the development of technology has changed the way we live by reducing the time spent on finding information, using home automation so that lights, temperature, security, and other functions can be adjusted through a device, and making life more convenient. In the industrial field, employees can save a lot of time with technology such as controlling devices through websites or applications.

The thesis describes an industrial 16-channel relay module controlled using the Modbus RTU protocol with the RS485 interface. A device will be handled by TMA Solutions, and used by the water supply company. The purpose of the device is to control 16 water pumps that turn on/off and 16 floats in a water tank to prevent water overflow. The water pumps can be controlled remotely with a mobile phone, a tablet or through a web application.

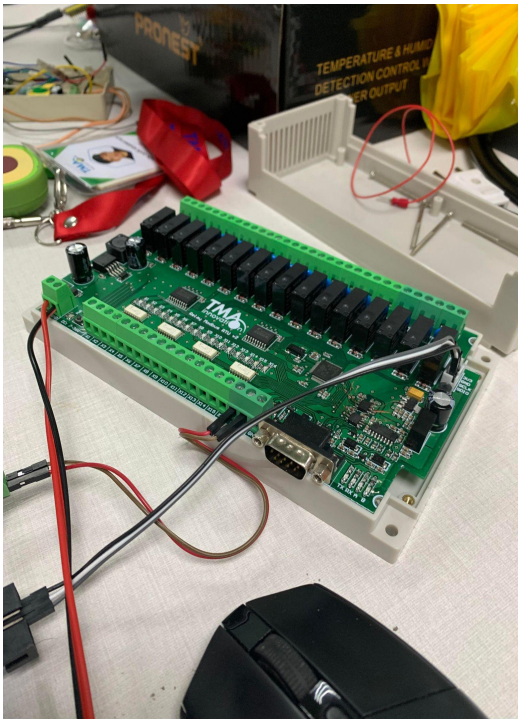



**Figure 1.** Modbus RTU Relay

# 2 TECHNOLOGIES USED

The technologies used in this thesis are C/C++, Python, Modbus protocol, and the high-performance STM32F4 MCU. They are briefly described in this chapter.

## 2.1 C/C++ Language

C language is a general-purpose programming language and a very popular language even though it is old. Gradually, C language became so popular that it is used in a wide variety of applications ranging from Embedded Systems to Super Computer.[1]

C++ language is a cross-platform language that can be used to create high-performance applications and is one of the world's most popular programming languages as an extension to the C language.[2]

## 2.2 Python Language

Today, Python is a popular programming language and is used for web development, software development, mathematics, and system scripting. Python can be used in embedded, small, or minimal hardware devices, depending on how limited the actual device is. Python can be most powerful when used as a communication intermediary between the user and the embedded systems that work together. [3]

## 2.3 Microcontroller

A microcontroller is a compact integrated circuit designed to govern a specific operation in an embedded system and it includes a processor, memory, and input/output peripherals on a single chip. In this project, Arduino was used for testing and Stm32 for the final product.[4]

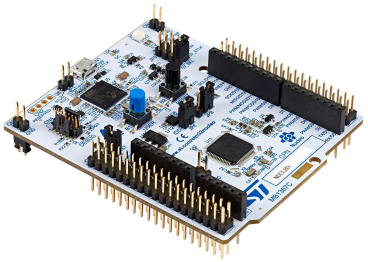


**Figure 2**. STM32 32-bit Arm

STM32 is a very common type of microcontroller used in numerous types of devices providing facilities for linking with other types of microcontrollers. STM32 is mainly used as a product for professional developers which requires certain professional knowledge but at the same time, it is relatively complicated to write code to realize functions.[5]

## 2.4 Modbus RTU Protocol

The Modbus RTU protocol is a means of communication that lows data exchange between PLCs and computers. Electronic devices can exchange information over serial lines using the Modbus RTU protocol

The Modbus RTU protocol is a master/slave protocol, one master controls the Modbus data transactions with multiple slaves that respond to the master's requests to read from or write data to the slaves.[6]

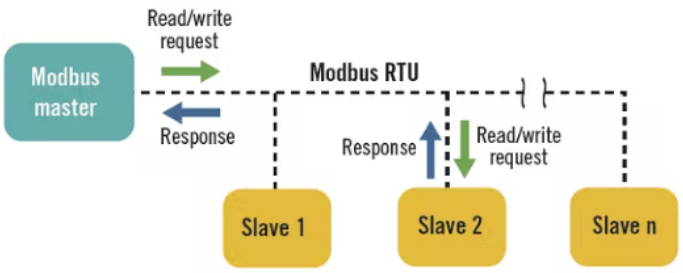



**Figure 3.** Operation of the Modbus RTU protocol work

RS485/232 communication can be used with Modbus

## 2.5 RS485 Communication

Industrial and instrumentation applications require the transmission of data between multiple systems often over very long distances. RS485 bus standard is one of the most widely used physical layer bus designs in the aforementioned application. RS485 has some advantages, such as long-distance links up to 1200m, and multiple drivers and receivers can be connected on the same bus. This type of bus is mainly used in the following applications: process control networks, industrial automation, remote terminals, and building automation such as heating, ventilation, and air conditioning.[7]

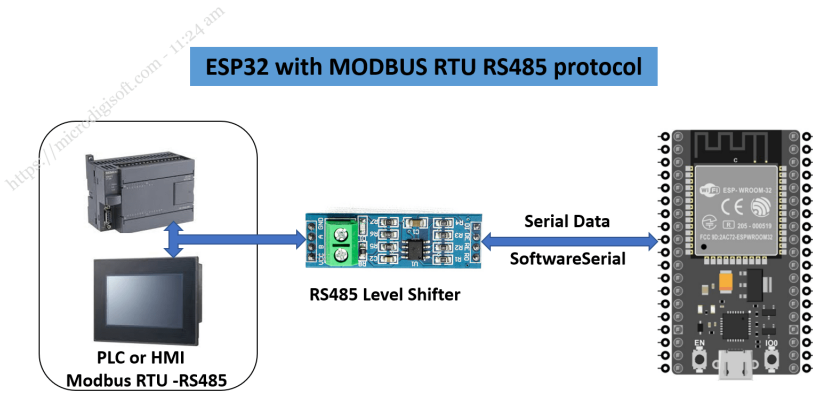



**Figure 4**. MCU with Modbus RTU and RS485 protocol

Modbus is not the same as RS485, the reason is that both protocols are related concepts and work together to work successfully.

# 3 DEVICE DEFINITIONS

The clients and the project team had a thorough discussion to establish the project scope, features, functionalities, preliminary design, and timeline, which are described in this chapter.

## 3.1 Device Requirements

After meeting with the customers, a list of functional requirements for using Modbus RTU and LoraWan to push the data to the Thingsboard server or company website

### 3.1.1 Modbus Function

| Function code | Description |
|---|---|
| 01 | Read the state of the relay |
| 03 | Read the address and version |
| 05 | Write a single relay |
| 06 | Set baud rate and address |
| 02 | Read the state of the input |

**Figure 5.** Functions code and their description

- The function code 01 is used to read the status of all relays on a remote device, with the relay address ranging from 1 to 16. [8]
- The function code 02 is used to read the status of all inputs on a remote device, with the input address ranging from 10001 to 10016. [8]
- The function code 03 is used to read a block of holding registers to get the address and version of the device on a remote device. [8]

- The function code 05 is used to set the state of an individual relay on a remote device to either ON or OFF. [8]
- The function code 06 is used to write a single holding register on a remote device to change the baud rate or address of a device. [8]

### 3.1.2 LoraWan

LoRaWan technology is a wireless communication system that covers long distances and uses minimal power, making it a desirable option for Industrial Internet of Things (IoT) applications due to it is appealing features.

**Figure 6.** A device with LoRaWAN node

## 3.2 Application Features

Multi-devices can be connected to the RS485 bus. The addresses of the device can range from 1 to 255. Each connected device can be controlled by sending the required command.

The device has an anti-reverse circuit that prevents circuit damage due to incorrect connection. The circuit is made from a rail-shaped ABS shell because it is easy to install, and safe to use.

### 3.3 System Specifications

- Power supply voltage: 5V.
- Communication interface: RS458 and the range address of RS458: 1~255.
- Communication protocol: Standard Modbus RTU protocol
- Baud rate: 4800, 9600, 19200, 38400, 57600, 115200, 128000, 25600, and default communication format: 115200, N, 16, 1.
- Relay channels: 16
- LED indicators:
    - STA: MCU indicator, keep flashing when the MCU is normally working.
    - TXD: TX indicator, lights up when sending data.
    - RXD: RX indicator, lights up when receiving data.

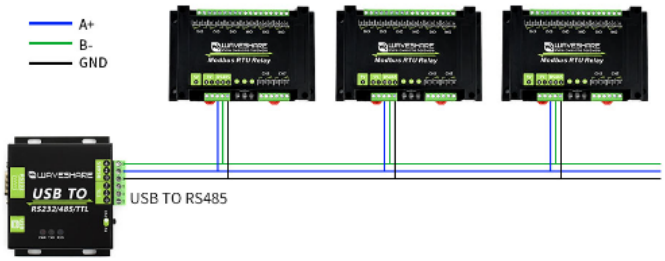



**Figure 7.** RS485 Communication

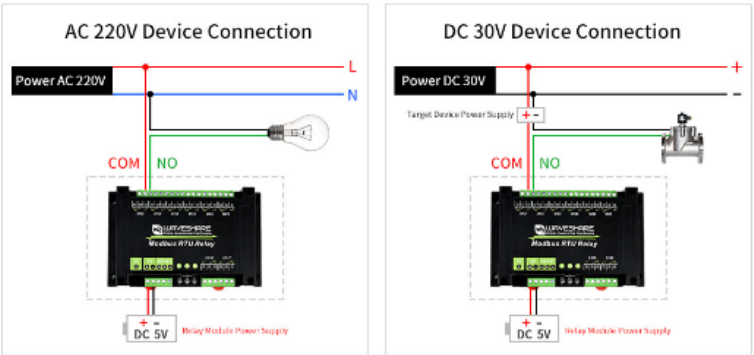



**Figure 8.** Relay Connection

The device directly controls a 220VAC home application or device below 30VDC , such as industrial control, smart home, intelligent agriculture, breeding farming

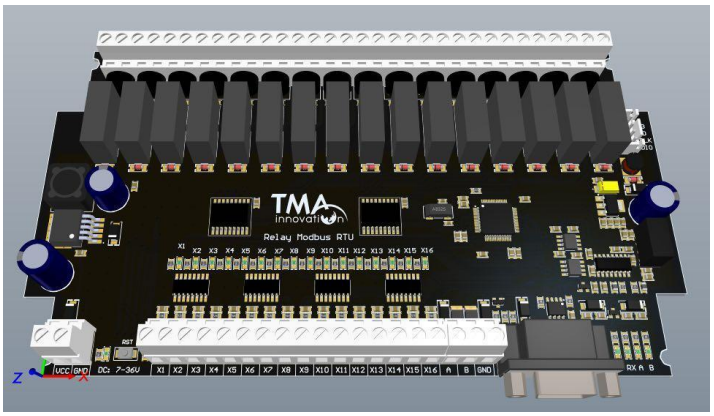



**Figure 9**. A device in a 3D projection

In this project, 16 relays (the black boxes which are at the top edge of the board in Figure 8) control 16 water pumps or applications, and 16 inputs (the white interface at the bottom edge of the board) read the status of 16 water tanks.

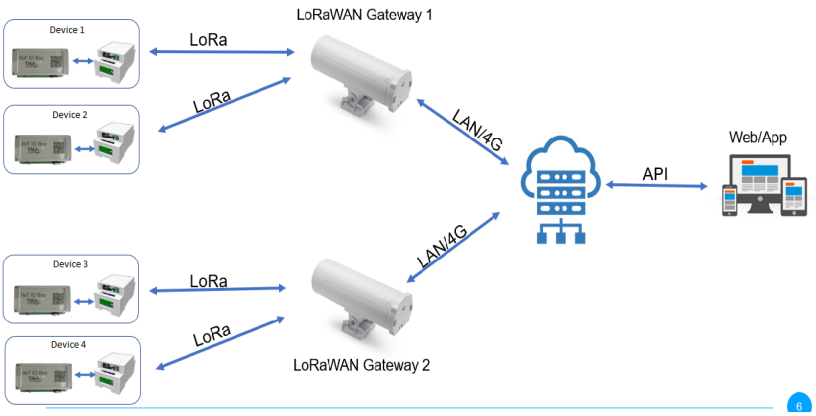



**Figure 10**. A device communicates with a server

Customers prefer an easy-to-use interface to control their assets through a website. The control commands issued by the customer through the website are sent to the server, which in turn controls a LoRaWAN box node. Once the server receives the control commands, it sends the commands to the appropriate node, which controls a device and executes the desired action.

# 4 DEVICE FUNCTION

The developed device has an individual ID based on the linked list structure when every device and input link to each other through the next pointer, and the slave message struct is used to store information about each device structure. [10]

```
struct Device{
    uint8_t data;
    uint16_t address;
    uint8_t id;
    struct Device *next;
};
typedef struct Device modbusDigReg;
modbusDigReg *_digReg;
modbusDigReg *_lastReg;

struct Slave{
    uint8_t *msg;
    uint16_t len;
    uint16_t crc;

    modbusDigReg *_device;
};
typedef struct Slave modbusSlave;
```

**Figure 11**. A structure consisting message and a device

After sending the command line to execute, the slave messages are separated into three types: funcT, which stores function code; byte1 which stores the registered address or small function codes; and byte2 which stores the data device. After processing the slave message, the memory of the message has to be deallocated to prevent memory leaks.

```
void run(modbusSlave *slave){
    uint8_t funcT;
    uint16_t byte1, byte2, byte3;
    sendData(slave);
    funcT = slave->msg[1];
    byte1 = slave->msg[2] <<8 | slave->msg[3];
    byte2 = slave->msg[4] <<8 | slave->msg[5];
    if(funcT == WRITE_MULTIAO){
      if(byte2<maxwriteregs){;
        int st=0;
        for(int i=0;i<byte2;i++){
          byte3 = (slave->msg[7+st]<<8) | slave->msg[8+st];
          setValue(slave->_device, byte1 + 40001 + i, byte3);
          st+=2;
        }
      }
      else{
        return;
      }
    }
    free(slave->msg);
    slave->len=0;
```

**Figure 12.** Separate slave message into 3 type

Default settings are:

- Device address: 1
- Relay address: 1~16
- Input address: 10001~10016

## 4.1 Reading the State of Relays Command (0x01)

The sending command is 01 01 00 00 00 08 3D CC to check how many relays are on. The command is made up as follows:

- 01: device address
- 01: function code command
- 00 00: the initially registered address of a controlled relay
- 00 08: check 8 the registered address from the initial address
- 3D CC: The CRC checksum of the first six bytes

After receiving the command line and separating data into 3 values, the device has to dynamically allocate the memory of the slave message by using malloc() because of deallocating the memory previously. Because the address of a device is from 0 to 15, then 1 has to be added to the address on getValue() function and the first 8 addresses are stored in message[3] and the next 8 addresses are stored in message[4]. Based on byte1 and byte2, a device sets the data device at the address and puts data into a slave message to transmit through Realterm.

**Figure 13**. Reading state of relays code and result

The receiving command is 01 01 01 00 51 88, constituting of

- 01: device address
- 01: function code command
- 01: number of bytes returned
- 00: all relays are off
- 51 88: The CRC checksum of the first four bytes

Receiving command: 01 01 01 41 91 B8

- 01: device address
- 01: function code command
- 01: number of bytes returned
- 41: relay 0 and 6 are open (0x41: 0100 0001)
- 91 B8: The CRC checksum of the first four bytes

### 4.2 Reading the State of Inputs Command (0x02)

Sending command: 01 02 00 00 00 08 79 CC to check how many inputs connect to the device. The command is made up as follows:

- 01: device address
- 02: function code command
- 00 00: the initially registered address of an input
- 00 08: check 8 the registered address from the initial address
- 79 CC: The CRC checksum of the first six bytes

After receiving the command line and separating data into three values, the device has to dynamically allocate the memory of the slave message by using malloc() because of deallocating the memory previously. Because the address of a device is from 10000 to 10015 then 10001 has to be added to the address on

getValue() function and the first 8 addresses are stored in message[3] and the next 8 addresses are stored in message[4]. Because the function (0x02) is nearly the same as the function(0x01) but it is different from the default value from relays then I have to reverse the algorithm. Based on byte1 and byte2, a device sets the data device at the address and puts data into a slave message to transmit through Realterm.

**Figure 14.** Reading state of inputs code and result

Receiving command: 01 02 01 FF E1 C8

- 01: device address
- 02: function code command
- 01: number of bytes returned
- FF: all inputs connect to a device
- E1 C8: The CRC checksum of the first four bytes

Receiving command: 01 02 01 F3 E1 CD

- 01: device address
- 02: function code command
- 01: number of bytes returned
- F3: inputs 10002 and 10003 do not connect to the device (0xF3: 1111 0011)
- E1 CD: The CRC checksum of the first four bytes

## 4.3 Reading Device Address and Version (0x03)

The function code 03 can be categorized into 2 types: reading device address (4000) and reading software version (8000).

- Reading device address command (0x4000)

Sending command: 00 03 40 00 00 01 90 1B to read device address. The command is made up as follows:

- 00: the broadcast address
- 03: function code command
- 40 00: read the device address function
- 00 01: set device address to 0x01
- 90 1B: The CRC checksum of the first six bytes

After receiving the command line and separating data into three values, the device has to dynamically allocate the memory of the slave message by using malloc() because of deallocating the memory previously. Based on byte1 and byte2, a device executes a small function code by byte1 and sets data values by byte2, and puts data into a slave message to transmit through Realterm.

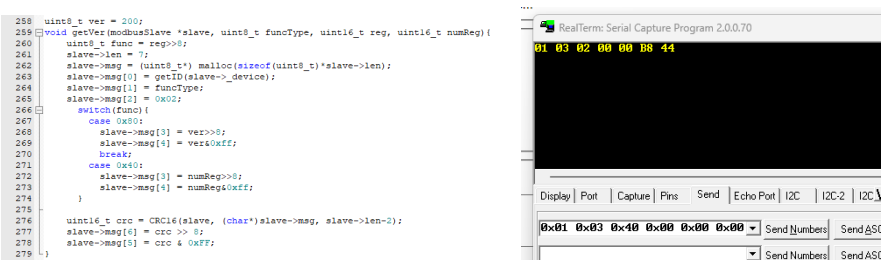



**Figure 15**. Reading device address code and result

Receiving command: 01 03 02 00 01 79 84

- 01: device address

- 03: function code command
- 02: number of bytes returned
- 00 01: set the device address as 0x0001
- 79 84: The CRC checksum of the first four bytes

- Reading software version command (0x8000)

Sending command: 01 03 80 00 00 01 8F CA to read the software version. The command is made up as follows:

- 01: the device address
- 03: function code command
- 80 00: read the software version function
- 00 01: the number of registers
- 8F CA: The CRC checksum of the first six bytes

After receiving the command line and separating data into three values, the device has to dynamically allocate the memory of the slave message by using malloc() because of deallocating the memory previously. The device executes function code based on byte1 and has a global value to store the value of the version. Data value converts into hex and puts all data into a slave message to transmit through Realterm.

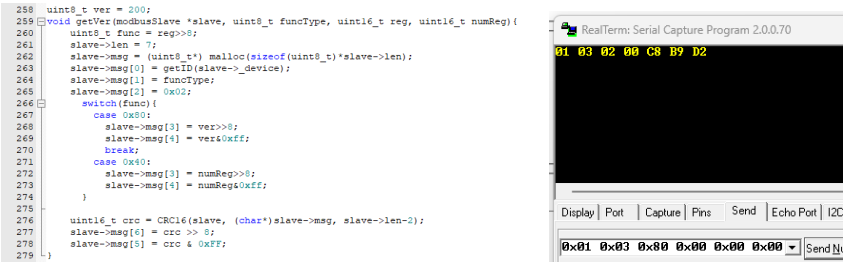



**Figure 16**. Reading software version code and result

Receiving command: 01 03 02 00 C8 F0 B8

- 01: device address
- 03: function code command
- 02: number of bytes returned
- 00 C8: convert to decimal (0x00C8 = 200 means version 2.00)
- F0 B8: The CRC checksum of the first four bytes

### 4.4 Controlling Relay Command (0x05)

Sending command: 01 05 00 03 FF 00 7C 3A to turn on relay 3. The command is made up as follows:

- 01: device address
- 05: function code command
- 00 03: the registered address of a controlled relay
- FF 00: turn on the relay
- 7C 3A: The CRC checksum of the first six bytes

Sending command: 01 05 00 03 00 00 3D CA to turn off relay 3. The command is made up as follows:

- 01: device address
- 05: function code command
- 00 03: the registered address of a controlled relay
- 00 00: turn off the relay
- 3D CA: The CRC checksum of the first six bytes

After receiving the command line and separating data into three values, the device has to dynamically allocate the memory of the slave message by using malloc() because of deallocating the memory previously. A device sets the data device at the address based on byte1 and byte2 and puts data into a slave message to transmit through Realterm.

```c
void getStatus(modbusSlave *slave, uint8_t funcType, uint16_t reg, uint16_t val){
  slave->len=8;
  slave->msg = (uint8_t*) malloc(sizeof(uint8_t)*slave->len);

  if(funcType==WRITE_DO) {
    setValue(slave->_device,reg+1,val>>8);
    slave->msg[1] = WRITE_DO;

  }
//  else{
//    setValue(slave->_device,reg+40001,(val>>8));
//    slave->msg[1] = WRITE_AO;
//  }
  if(slave->msg[1] == WRITE_DO){
      slave->msg[0] = getID(slave->_device);
      slave->msg[2] = reg>>8;
      slave->msg[3] = reg&0xff;
      slave->msg[4] = val>>8;
      slave->msg[5] = val&0xff;
      uint16_t crc = CRC16(slave, (char*)slave->msg, slave->len-2);
      slave->msg[7] = crc >> 8;
      slave->msg[6] = crc & 0xFF;
      switch(slave->msg[3]){
        case 0x00:
        HAL_GPIO_WritePin(GPIOA,GPIO_PIN_5,slave->msg[4]&1);
        break;
```

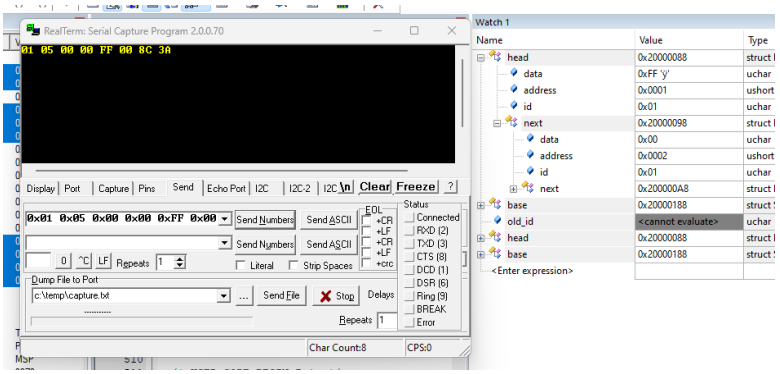



**Figure 17**. Controlling relay code and result

## 4.5 Setting Device Address and Baud Rate (0x06)

The function code 05 can be categorized into 2 types: setting baud rate (2000) and setting device address (4000)

- Setting baud rate command (0x2000)

Sending command: 01 06 20 00 00 05 43 D8 to set the baud rate

  - 01: the device address
  - 06: function code command
  - 20 00: set the baud rate function
  - 00: no parity check

- 05: baud rate value (following the baud rate value: 0x00: 4800, 0x01: 9600, 0x02: 19200, 0x03: 38400, 0x04: 57600, 0x05: 115200, 0x06: 128000, 0x07: 256000)
- 43 D8: The CRC checksum of the first six bytes

After receiving the command line and separating data into three values, the device has to dynamically allocate the memory of the slave message by using malloc() because of deallocating the memory previously. Next, the baud rate value is arranged into an array, and based on byte1 and byte2, a device executes a small function code on byte1 and gets data value on byte2. Data value converts into the int type to get the index on the array and a device has to deinitialize USART to set a new baud rate value based on the data value and initialize USART again. After that data is put into a slave message to transmit through Realterm.

```
uint16_t new_baud[8] = {48,96,192,384,576,1152,1280,2560};
void setBaud_Add(modbusSlave *slave, uint8_t funcType, uint16_t reg, uint16_t val){
    slave->len = 8;
    slave->msg = (uint8_t*) malloc(sizeof(uint8_t)*slave->len);
    slave->msg[0] = getID(slave->_device);
    slave->msg[1] = funcType;
    slave->msg[2] = reg>>8;
    slave->msg[3] = reg&0xff;
    slave->msg[4] = val>>8;
    slave->msg[5] = val&0xff;
    uint8_t new_id = slave->msg[5];
    switch(slave->msg[2]){
      case 0x20:
        HAL_UART_Abort_IT(&huart2);
        HAL_UART_DeInit(&huart2);
        huart2.Init.BaudRate = new_baud[(int)val]*100;
        if(HAL_UART_Init(&huart2)!=HAL_OK){
          Error_Handler();
        }
        break;
      case 0x40:
        while(slave->_device!=NULL){
          setID(slave->_device, new_id);
          slave->_device = slave->_device->next;
        }
        break;
    }
    uint16_t crc = CRC16(slave, (char*)slave->msg, slave->len-2);
    slave->msg[7] = crc >> 8;
    slave->msg[6] = crc & 0xFF;
}
```

**Figure 18**. Setting baud rate code

Receiving command: 01 06 20 00 05 43 D8

- 01: the device address
- 06: function code command
- 20 00: set the baud rate function
- 00: no parity check
- 05: baud rate value

    - 43 D8: The CRC checksum of the first six bytes

- **Setting device address command (0x4000)**

Sending command: 00 06 40 00 00 01 5C 1B to set device address

    - 00: the broadcast address
    - 06: function code command
    - 40 00: set device address function
    - 00 01: set device address as 0x0001
    - 5C 1B: The CRC checksum of the first six bytes

Receiving command: 00 06 40 00 00 01 5C 1B

    - 00: the broadcast address
    - 06: function code command
    - 40 00: set device address function
    - 00 01: set device address as 0x0001
    - 5C 1B: The CRC checksum of the first six bytes

# 5 IMPLEMENTATION & TESTING

## 5.1 Implementation

This chapter introduces in detail how to use the developed device. Before releasing a device for a customer, our team developed the frontend interface that can be accessed via a laptop or a tablet, allowing the customer to easily control and interact with the device.

In this project, the ModbusPoll application was used to test and control many features, setting the serial port, baud rate, data bits, parity, and stop bit before executing.

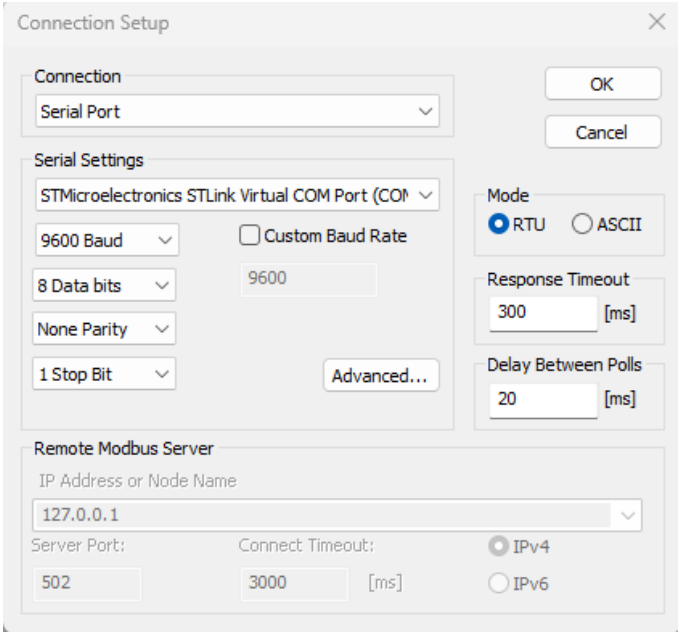



**Figure 19**. Setting information with Modbus Application

Suppose a customer wants to activate relay 5. If they were to use Realterm, they would need to manually write an 8-byte message and transmit it. However, by using the ModbusPoll application, the customer can save a lot of time by only selecting the desired function, address, and value. After that, the application

automatically generates the appropriate command. The customer can use function code 02 to display all the relays in a table format.

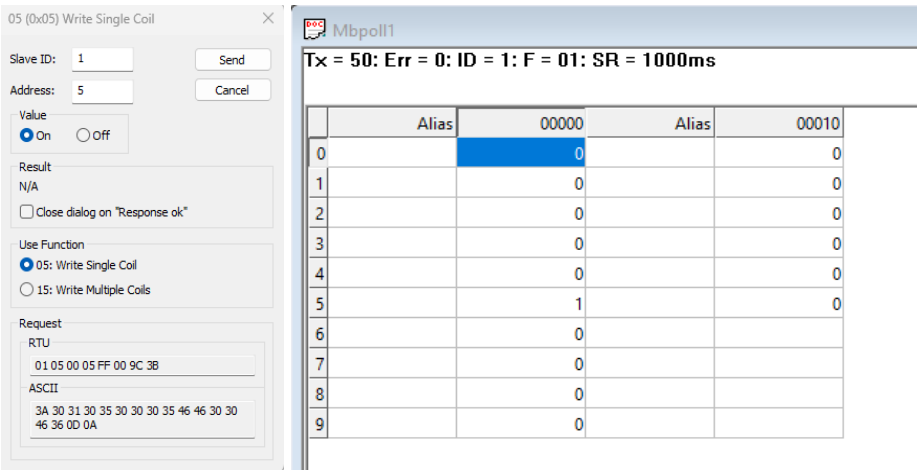
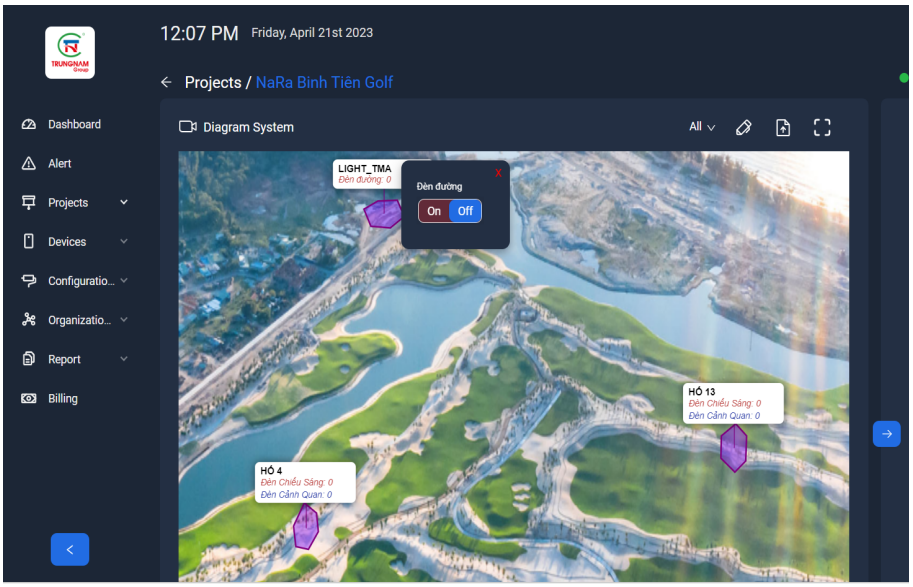



**Figure 20**. Control and list all information on relays

In addition to using ModbusPoll, the customer also has the option to control the device through a website. By default, the initial state of light is OFF on the website and the corresponding data for the light on the server is set to zero (0).



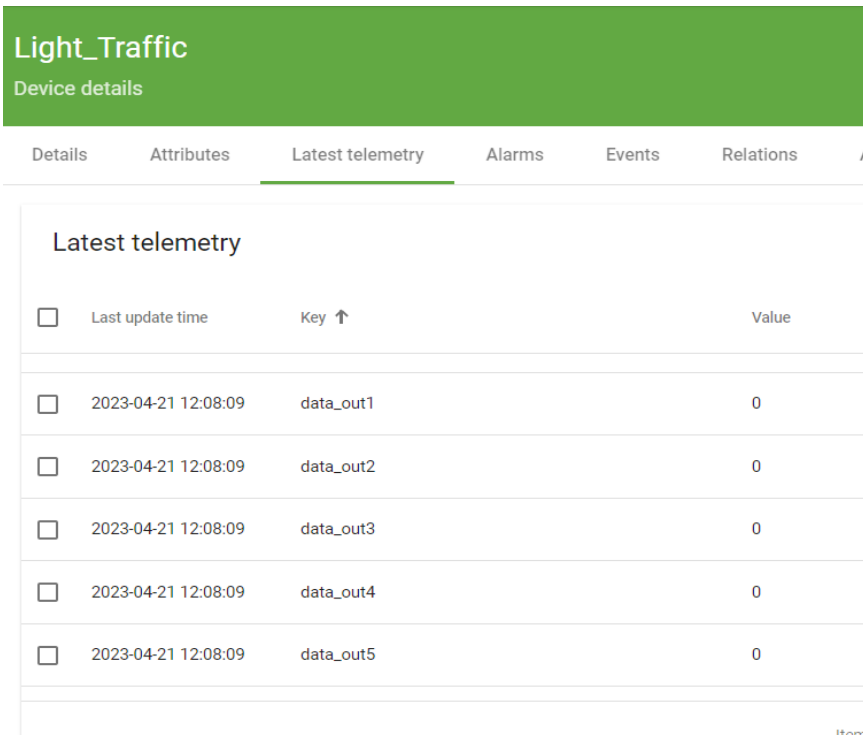
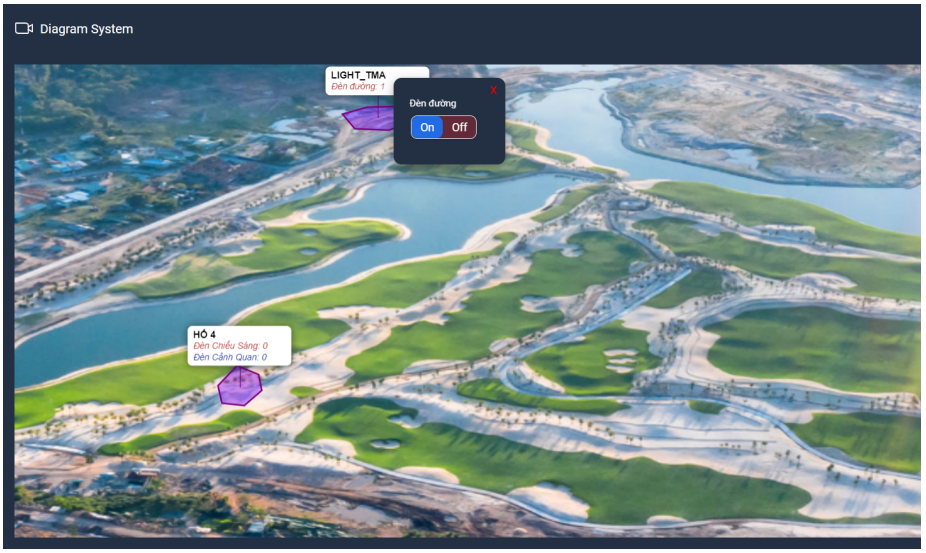



**Figure 21**. Website and Thingsboard server

The server records the state of the light as one (1) after the light is switched ON from the OFF state. Additionally, customers can observe the input and device connected to the device to ensure everything is working properly.



| | | | |
|---|---|---|---|
| ☐ | 2023-04-21 12:26:29 | data_out1 | 1 |
| ☐ | 2023-04-21 12:26:29 | data_out2 | 0 |
| ☐ | 2023-04-21 12:26:29 | data_out3 | 0 |
| ☐ | 2023-04-21 12:26:29 | data_out4 | 0 |
| ☐ | 2023-04-21 12:26:29 | data_out5 | 0 |

**Figure 22**. Website and Thingsboard server

## 5.2 Testing

The device was tested by one manual tester who is working in TMA Solutions company, responsible for testing all features and bugs on the device before releasing it to customers. The Python language was used to test the device and it required to download all libraries and set up the correct method, baud rate, port, byte size, time out, parity and stop bits to perform the test.

If all floats are  connected to the device, the result prints all True but if one of 16 floats is not connected  by accident , the result prints False.

```
client1.connect()
time.sleep(1)
# client1.write_coil(1, 0, unit=1) #turn on or off relay
# rq=client1.write_register(8192, 5000, unit=1)
while True:
    float_state_rr = client1.read_discrete_inputs(0, 16, unit=1)
    print("")
    print("State of input:")
    for i in range(0,16,1):
        float_value = float_state_rr.bits[i]
        print(float_value)
    time.sleep(2)
```

```
State of input:
True
False
True
True
True
True
True
True
True
True
True
True
True
True
True
True
```

```
State of input:
True
True
True
True
True
True
True
True
True
True
True
True
True
True
True
True
```

**Figure 23.** Results when test checking states of input

# 6 CONCLUSIONS

The system which uses the Modbus RTU protocol and an IoT ticket platform, was successfully developed and deployed on the device. The device can receive data from the server to execute the function which is required by the user. The customer received the device on time and provided favorable feedback about it.

By implementing a different approach to configuring port outputs, the system can be improved. This method involves grouping necessary outputs into a single array and employing a new algorithm to execute the code, thereby reducing the reliance on multiple if-else conditions. Currently, the command function for configuring the device address is unavailable, and the solution involves establishing a broadcast device for storing or switching between multiple devices. Additionally, to conserve energy, the device can be set to sleep mode when not in use.

As it exists currently, the device can be used as an industrial solution instead of a wireless network. The customer has expressed interest in integrating additional capabilities that use Over-The-Air (OTA) technology, enabling remote firmware updates from a distance.

## REFERENCES

/1/ C language – Overview. Accessed 21/04/2023. https://www.tutorialspoint.com/cprogramming/c_overview.htm

/2/ Avila, Risto. 2022. C++ for Embedded: Advantages, Disadvantages, and Myths. Accessed 21/04/2023.https://www.qt.io/embedded-development-talk/c-for-embedded-advantages-disadvantages-and-myths.

/3/ Radcliffe, Tom. 2016. Python vs. C/C++ in embedded systems. Accessed 21/03/2023. https://opensource.com/life/16/8/python-vs-cc-embedded-systems.

/4/ What is Arduino. Accessed 22/03/2023. https://www.arduino.cc/en/Guide/Introduction

/5/ Introduction to STM32 Microcontroller. Accessed 22/03/2023. https://www.theengineeringknowledge.com/introduction-to-stm32-microcontroller/

/6/ Cyburt, Bruce. 2012. Introduction to Modbus. Accessed 25/03/2023

https://www.automation.com/en-us/articles/2012-1/introduction-to-modbus

/7/ Lammert Bies. 2021. RS485 serial information. Accessed 25/03/2023

https://www.lammertbies.nl/comm/info/rs-485

/8/ Panisetti Prudhviraj. 2020. What are the function codes of Modbus (RTU) and their requests and responses? Accessed 26/03/2023

https://medium.com/analytics-vidhya/what-are-the-function-codes-of-modbus-rtu-and-their-requests-and-responses-8c33a467aed3

/9/ Modbus RTU protocol Overview. Accessed 26/03/2023

https://dghcorp.com/modbus-overview/

/10/ Marco Lieblang. Are there Classes in the C Programming Language? Accessed 02/04/2023

https://moderncprogramming.com/are-there-classes-in-the-c-programming-language/#:~:text=There%20are%20no%20classes%20in,the%20object%20oriented%20programming%20world.

**APPENDIX 1**

Link GitHub includes code and flowchart of device: https://github.com/huynguyen2310/Industrial-16-channel-relay

The flowchart will describe how a device work and the code will show algorithms to execute.