



# Bethel324-control-system

## Team members

- 
- 21300756 채영민
  - 21700105 김석진

## Table of Contents

---

[Team members](#)

[Table of Contents](#)

[Introduction](#)

[Prior knowledge](#)

[Features](#)

[System overview](#)

[System description](#)

[Team composition](#)

[Links](#)

[References](#)

## Introduction

저희는 벤엘관 324호에 살고 있습니다. 기숙사 생활에는 고정적으로 반복되는 몇 가지 일들이 있고, 이러한 일들이 저희를 피곤하게 만듭니다. 방을 비울 때는 방문을 잠그고, 밤 11시가 되면 소등하고, 자기 전에 편안한 잠을 위해 가습기를 텁니다. 이러한 일들을 자동으로 혹은 스마트폰 터치 하나로 할 수 있다면 얼마나 편할지 기대하는 마음으로 Bethel324-control-system을 구현하기로 했습니다.

이 프로젝트에서 사용되는 통신 프로토콜은 TCP 기반의 HTTP 프로토콜, TCP 기반의 Modbus-TCP 프로토콜, RS485 기반의 Modbus-RTU 프로토콜이 있습니다. Modbus 프로토콜이 생소한 분들을 위해 아래 별도의 설명을 첨부했습니다.

## Prior knowledge

### Modbus protocol

1. Modbus란?

- OSI model 7계층인 application layer messaging protocol
- 1979년부터 산업 표준으로 쓰이고 있다.

- TCP/IP에서는 MODBUS의 port는 502로 reserve했다.
- 'function code'에 의해 명시된 서비스를 제공하는 request/reply protocol
- MODBUS는 application protocol이기 때문에 그 아래 구현 방법은 다양하다.
  - TCP/IP over Ethernet. See MODBUS Messaging Implementation Guide V1.0a.
  - Asynchronous serial transmission over a variety of media (wire : EIA/TIA-232-E, EIA-422, EIA/TIA-485-A; fiber, radio, etc.)
  - MODBUS PLUS, a high speed token passing network.

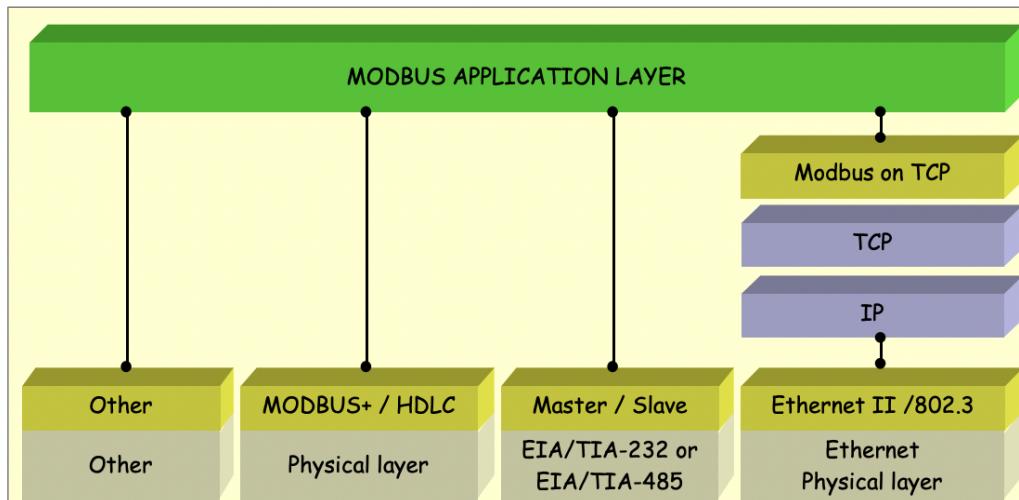


Fig 1. Various Modbus Implementations

## 2. General description

### 1. Protocol description

MODBUS의 PDU(Protocol Data Unit)과 ADU(Application Data Unit)는 아래와 같다. ADU는 client에서 MODBUS transaction을 시작하기 위해 만들고 function code에는 server가 수행할 action이 명시된다.

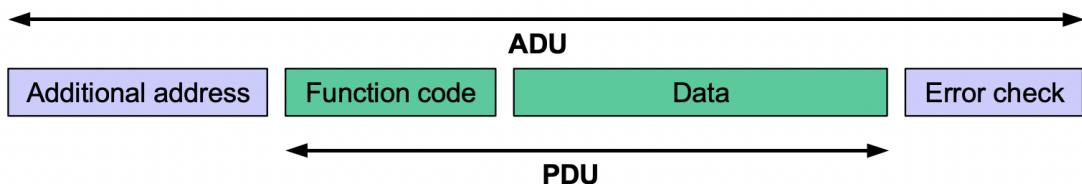


Figure 3: General MODBUS frame

Fig 2. General MODBUS frame

- Function code

Function code는 1 byte이고 valid code는 [1, 255]임. 이 중 [128, 255]는 exception response를 위해 reserved된다. 몇몇 function code에는 multiple action을 정의하기 위해

sub-function code가 추가된다.

- Data

data field는 server가 action을 수행하기 위해 필요한 추가적인 정보가 들어간다. 하지만 몇몇 function code의 경우 필요하지 않기 때문에 data field의 길이가 0 즉, 존재하지 않기도 한다.

- Error

Error가 발생하지 않은 경우 server는 단순히 echo로 response한다.

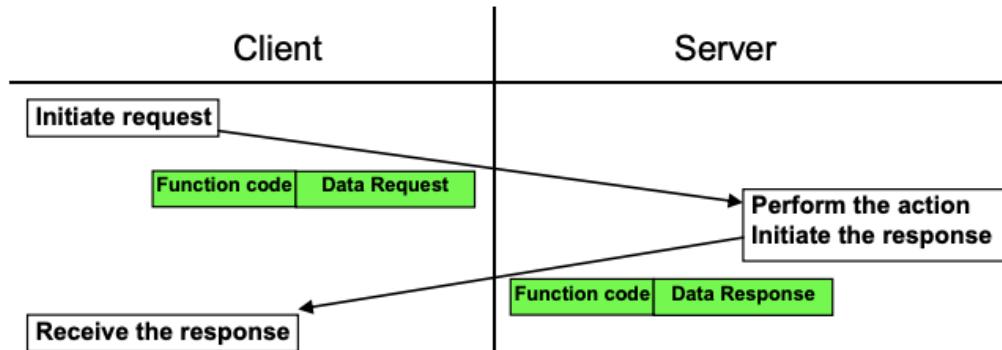


Fig 3. MODBUS normal conversation

Error가 발생한 경우 server는 exception function code와 exception code를 response한다. 이때 exception function code란 original function code의 MSB를 1로 set한 것이다.

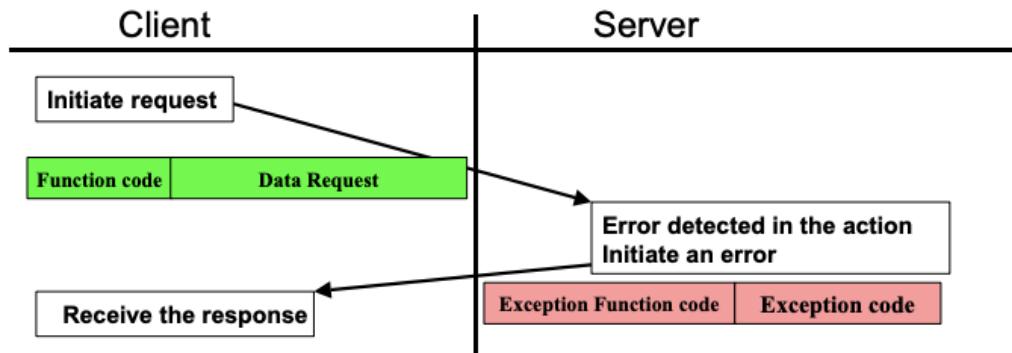


Fig 4. MODBUS abnormal conversation

- size of PDU

MODBUS PDU의 크기는 serial line network의 첫 번째 MODBUS 구현에서 상속된 크기 제약에 의해 제한된다. (max. RS485 ADU = 256 bytes)

아래는 몇 가지 예시들이다.

1. serial line communication에서 MODBUS PDU 크기 = 256 - server address(1 byte) - CRC(2 bytes) = 253 bytes
2. RS232 / RS485 ADU = 253 bytes + Server address (1 byte) + CRC (2 bytes) = 256 bytes
3. TCP MODBUS ADU = 253 bytes + MBAP (7 bytes) = 260 bytes.

- 3 types of PDU

MODBUS의 PDU는 아래와 같이 3종류로 나뉘고 각 PDU는 다음과 같이 정의된다.

1. MODBUS Request PDU

```
mb_req_pdu = {function_code, request_data},      where
            function_code = [1 byte] MODBUS function code,
            request_data = [n bytes] This field is function code dependent and usually
                                contains information such as variable references,
                                variable counts, data offsets, sub-function codes etc.
```

Fig 5. Definition of MODBUS Request PDU

2. MODBUS Response PDU

```
mb_req_pdu = {function_code, request_data},      where
            function_code = [1 byte] MODBUS function code,
            request_data = [n bytes] This field is function code dependent and usually
                                contains information such as variable references,
                                variable counts, data offsets, sub-function codes etc.
```

Fig 6. Definition of MOBDUS Response PDU

3. MODBUS Exception Response PDU

```
mb_excep_rsp_pdu = {exception-function_code, request_data},      where
            exception-function_code = [1 byte] MODBUS function code + 0x80
            exception_code = [1 byte] MODBUS Exception Code Defined in table
                                "MODBUS Exception Codes" (see section 7 ).
```

Fig 7. Definition of MODBUS Exception Response PDu

2. Data encoding

MODBUS는 Big-Endian 방식을 사용한다. 따라서 아래 예시처럼 1 byte 이상의 data를 전송할 때 MSB가 먼저 전송된다.

<u>Register size</u>	<u>value</u>	
16 - bits	0x1234	the first byte sent is 0x12 then 0x34

Fig 8. MODBUS data encoding

### 3. MODBUS Data model

MODBUS 데이터 모델은 아래와 같이 구별되는 특성을 가지고 있다.

Primary tables	Object type	Type of	Comments
Discretes Input	Single bit	Read-Only	This type of data can be provided by an I/O system.
Coils	Single bit	Read-Write	This type of data can be alterable by an application program.
Input Registers	16-bit word	Read-Only	This type of data can be provided by an I/O system
Holding Registers	16-bit word	Read-Write	This type of data can be alterable by an application program.

Fig 9. MODBUS Data model

### 3. Function code categories

MODBUS Function code는 아래와 같은 3가지 종류가 있다.

#### 1. Public function code

- well-defined function code
- unique하다.
- defined public assigned function code뿐만 아니라 future use를 위해 reserve된 unassigned function codes 둘 다 포함한다.

#### 2. User-defined function code

- User-defined function code의 두 가지 범위가 있다. i.e. [65, 72], [100, 110]
- user가 선택해서 function code를 implement 할 수 있다.
- unique 보장 못 한다.

#### 3. Reserved function code

legacy product들을 위해 몇몇 회사에 의해 현재까지 쓰이고 public use를 위해서는 사용가능하지 않은 function code들이다.

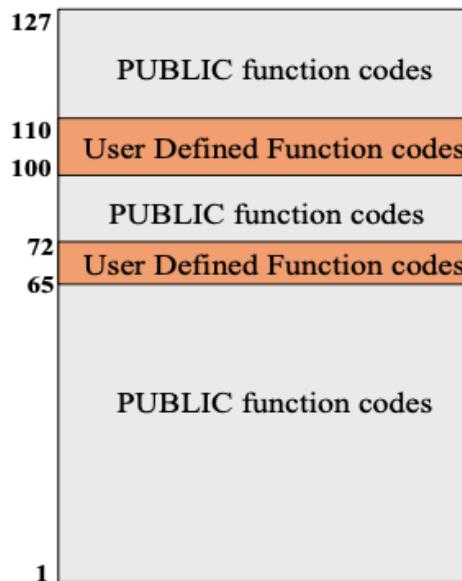


Fig 10. MODBUS Function code categories

## Features ⭐

로컬 네트워크 상에서 웹 브라우저를 통해 아래와 같은 기능들을 수행할 수 있다.

1. LED 상태 (on/off) 확인 및 제어
2. 가습기 상태 (on/off) 확인 및 제어
3. 문 잠김 상태 (lock/ unlock) 확인 및 제어
4. 습도 (0~100%) 확인

## System overview 🏠

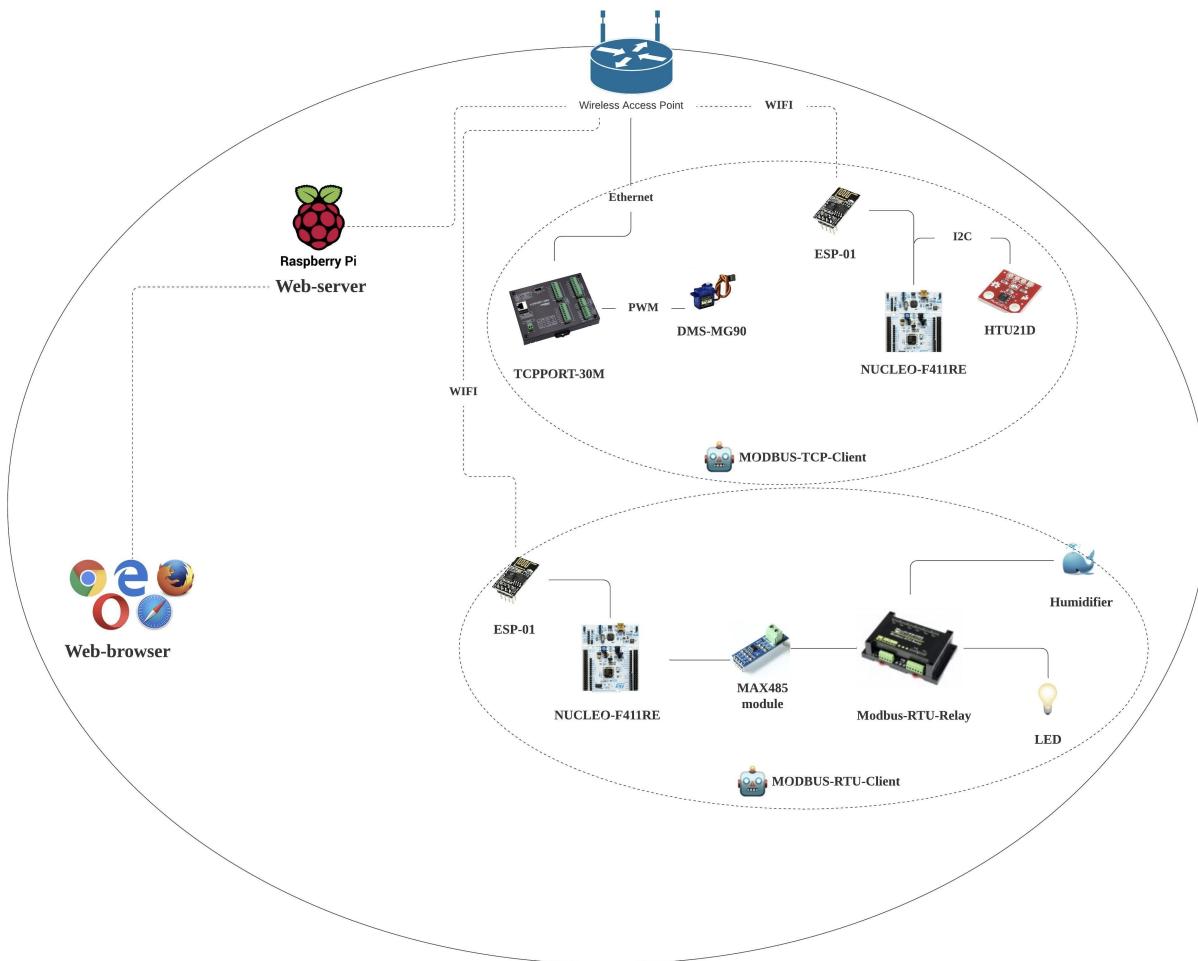


Fig 11. System overview

Bethel324-control-system은 LAN상에서 웹 브라우저를 통해 LED, 가습기, doorlock에 대한 상태를 확인하고 제어하는 시스템이고 Web Server, Modbus-RTU-Client, Modbus-TCP-Client로 구성되어 있다.

Web Server는 외부 사용자의 제어 명령을 Modbus-RTU-Client 혹은 Modbus-TCP-Client로 넘겨주는 기능과 현재 센서, 액츄에이터들의 상태를 외부 사용자의 웹페이지에 출력해주는 기능을 한다. Web Server는 하드웨어로 Raspberry Pi 4를 사용하며, Python의 Flask를 이용하여 구현했다.

Modbus-RTU-Client는 LED와 Humidifier를 제어하는 기능을 담당한다. Web Server에서 LED와 Humidifier에 대한 제어 명령이 오면, 그 명령을 받아 처리한다. 내부적으로 Nucleo-F411RE와 Modbus-RTU-Relay로 구성되어 있으며, Nucleo-F411RE가 ESP-01 모듈을 통해 WiFi로 Web Server와 직접 통신한다. Nucleo-F411RE와 Modbus-RTU-Relay는 RS485 통신으로 연결되고, Modbus RTU Protocol Frame으로 통신한다. Nucleo-F411RE는 자체적으로 RS485 통신이 없으므로 MAX485 모듈이 UART 통신을 RS485 통신으로 변환해준다. Nucleo-F411RE에는 ARM 사에서 개발 중인 Mbed-os를 기반으로 구현된 Firmware가 업로드 되어있다.

Modbus-TCP-Client는 습도 측정과 doorlock을 제어하는 기능을 담당한다. Web Server에서 Doorlock에 대한 제어와 습도 측정에 대한 명령이 들어오면 그것을 받아 처리한 뒤 결과를 Web Server로 보내준다. 내부적으로 Nucleo-F411RE와 Modbus-TCP PORT-30M으로 구성되어 있으며, Modbus-TCP PORT-30M이 Web Server와 직접 Modbus TCP Protocol Frame으로 통신한다. Doorlock(DMS-MG90)은 TCP PORT의 PWM out에서

PWM 신호를 제어할 수 있기 때문에 직접 제어된다. 하지만 온습도 센서(HTU21D)는 TCPPORT-30M에서 I2C 인터페이스를 제공하지 않기 때문에 직접 제어되지 못 한다. 이를 해결하기 위해 Nucleo-F411RE에 연결하여 측정된 습도를 ESP-01 모듈을 통한 wifi 통신으로 TCPPORT-30M에 전송한다. Nucleo-F411RE에는 RTU 측과 동일하게 ARM 사에서 개발 중인 Mbed-os를 기반으로 구현된 Firmware가 업로드 되어있다.

## Software overview

---

1. Modbus-RTU-Client
  - mbed-os-6.11.0
2. Modbus-TCP-Client
  - mbed-os-6.11.0
3. Web server
  - Linux raspberrypi 5.4.72-v7l+
  - Flask 1.0.2
  - Python 3.7.3

## Hardware overview

---

1. Modbus-RTU-Client 
  1. [MODBUS RTU RELAY](#)
  2. [MAX485 Module](#)
  3. [STM32 Nucleo-64 board](#)
  4. [ESP-01](#)
2. Modbus-TCP-Client 
  1. [TCPPORT-30M](#)
  2. [DMS-MG90](#)
  3. [STM32 Nucleo-64 board](#)
  4. [ESP-01](#)
  5. [HTU21D](#)
3. Web server 
  1. [Raspberry Pi 4 Model B](#)

## System description

---

# MODBUS-RTU-Client



## 1. MAX485 모듈을 이용한 NUCLEO-F411RE와 Modbus-RTU-Relay간 통신

Modbus-RTU-Relay는 physical layer에서는 RS485 프로토콜을 사용하고, application layer에서는 MODBUS-RTU 프로토콜을 사용한다.

RS485 인터페이스를 구현하기 위해 NUCLEO-F411RE와 Modbus-RTU-Relay 사이에 MAX485 모듈로 연결했다.

그리고 Modbus-RTU-Relay는 serial 통신을 하는데 default format이 '9600, N, 8, 1'이다. 이를 구현하기 위해 Mbed OS에서 제공하는 UART driver 중 하나인 UnbufferedSerial을 사용했다.

## 2. LED 상태 확인 및 제어

LED는 Modbus-RTU-Relay의 relay 0에 연결되어 있다. 따라서, LED를 켜고 끄기 위해서는 relay 0를 각각 열고 닫으면 된다. 이를 위한 modbus 명령어는 아래와 같다.

- LED ON : 0x01, 0x05, 0x00, 0x00, 0xFF, 0x00, 0x8C, 0x3A
- LED OFF : 0x01, 0x05, 0x00, 0x00, 0x00, 0x00, 0xCD, 0xCA

LED ON을 해석하면 id가 0x01인 slave에게 0x05인 function code를 명령하는 것을 알 수 있다. 0x05는 'write single coil'로 뒤따라오는 Output Address Hi, Output Address Lo, Output Value Hi, Output Value Lo과 함께 해석하면 Coil 0에 0xFF00 즉, ON을 write하는 것을 알 수 있다.

## 3. Humidifier 상태 확인 및 제어

Humidifier는 Modbus-RTU-Relay의 relay 1에 연결되어 있다. 따라서, Humidifier를 켜고 끄기 위해서는 relay 1를 각각 열고 닫으면 된다. 이를 위한 modbus 명령어는 아래와 같다.

- Humidifier ON : 0x01, 0x05, 0x00, 0x01, 0xFF, 0x00, 0xDD, 0xFA
- Humidifier OFF : 0x01, 0x05, 0x00, 0x01, 0x00, 0x00, 0x9C, 0x0A

Humidifier ON을 해석하면 id가 0x01인 slave에게 0x05인 function code를 명령하는 것을 알 수 있다. 0x05는 'write single coil'로 뒤따라오는 Output Address Hi, Output Address Lo, Output Value Hi, Output Value Lo과 함께 해석하면 Coil 1에 0xFF00 즉, ON을 write하는 것을 알 수 있다.

## 4. NUCLEO-F411RE와 web-server 간 통신

NUCLEO-F411RE는 ESP-01을 통해 wifi로 LAN에 접속하여 web-server와 HTTP 통신을 한다. 이때 NUCLEO-F411RE는 2초마다 web-server에게 어떤 명령을 수행해야 하는지 데이터를 전송하고 web-server는 이에 대한 response로 HTTP body에 1byte짜리 명령을 보낸다.

명령으로 온 1byte 중 LSB는 humidifier에 대한 제어 명령이고 LSB 바로 왼쪽에 있는 bit는 LED에 대한 제어 명령이다. 이때 bit가 1이면 on이고, 0이면 off를 의미한다. 따라서 web-server가 0x01을 보내면

humidifier는 켜고, LED는 끄게 된다. NUCLEO-F411RE의 전력 사용량을 최적화하기 위해 가장 최근에 받은 명령어와 현재 받은 명령어가 다를 경우에만 Modbus-RTU-Relay에 제어 명령을 보내도록 구현했다.

또한 간혹 통신 문제로 웹서버로부터 HTTP body가 포함되지 않은 짧은 길이의 데이터를 전송받는데 이러한 예외상황을 처리하기 위해 전송받은 데이터가 153 바이트보다 짧으면 2초를 멈춘 뒤 다시 서버에게 명령어를 전송하도록 구현했다.

## 5. 코드

```
#include "mbed.h"

#define SERVER_IP "192.168.0.100"
#define SERVER_PORT 5000
#define SERVER_RES_SIZE 153

UnbufferedSerial modbus(PC_6, PC_7, 9600);
DigitalOut enable(ARDUINO_UNO_D4);

UnbufferedSerial pc(CONSOLE_TX, CONSOLE_RX, 115200);

WiFiInterface *wifi;
TCPSocket socket;

char txBuf_pc[1024];
char txBuf_server[] = "GET /status HTTP/1.1\r\n\r\n";
char rxBuf_server[1024];

char LED_on[] = {0x01, 0x05, 0x00, 0x00, 0xFF, 0x00, 0x8C, 0x3A};
char LED_off[] = {0x01, 0x05, 0x00, 0x00, 0x00, 0x00, 0xCD, 0xCA};
char HUM_on[] = {0x01, 0x05, 0x00, 0xFF, 0x00, 0xDD, 0xFA};
char HUM_off[] = {0x01, 0x05, 0x00, 0x01, 0x00, 0x00, 0x9C, 0x0A};

int main()
{
    SocketAddress sockAddr;
    SocketAddress serverAddr(SERVER_IP, SERVER_PORT);
    nsapi_error_t err;
    char last_command = 0xff;

    /* Connect to wifi */
    wifi = WiFiInterface::get_default_instance();
    if(!wifi)
    {
        sprintf(txBuf_pc, "ERROR: No WiFiInterface found.\r\n");
        pc.write(txBuf_pc, strlen(txBuf_pc));
        while(1){};
    }

    sprintf(txBuf_pc, "Connecting to %s...\r\n", MBED_CONF_APP_WIFI_SSID);
    pc.write(txBuf_pc, strlen(txBuf_pc));

    err = wifi->connect(MBED_CONF_APP_WIFI_SSID,
                         MBED_CONF_APP_WIFI_PASSWORD, NSAPI_SECURITY_WPA_WPA2);

    if(err != NSAPI_ERROR_OK)
    {
        sprintf(txBuf_pc, "Connection error!!\r\n");
        pc.write(txBuf_pc, strlen(txBuf_pc));
        return -1;
    }
}
```

```

sprintf(txBuf_pc, "Success!!\r\n");
pc.write(txBuf_pc, strlen(txBuf_pc));

wifi->get_ip_address(&sockAddr);
sprintf(txBuf_pc, "IP : %s\r\n", sockAddr.get_ip_address());
pc.write(txBuf_pc, strlen(txBuf_pc));

enable = 1;
while(true)
{
    // Open a TCP socket on the network interface
    socket.open(wifi);

    err = socket.connect(serverAddr);
    if(err != NSAPI_ERROR_OK)
    {
        sprintf(txBuf_pc, "Error connecting : %d\r\n", err);
        pc.write(txBuf_pc, strlen(txBuf_pc));
        socket.close();
        return -1;
    }

    // send GET message to server
    socket.send((const char*)txBuf_server, strlen(txBuf_server));
    sprintf(txBuf_pc, "Send : %s\r\n", txBuf_server);
    pc.write(txBuf_pc, strlen(txBuf_pc));

    // recv from the server
    nsapi_size_or_error_t size = socket.recv(rxBuf_server, 1024);

    if(size <= 0)
    {
        if(size == NSAPI_ERROR_WOULD_BLOCK) continue;

        sprintf(txBuf_pc, "Error while receiving data from TCP socket(%d)\r\n", size);
        pc.write(txBuf_pc, strlen(txBuf_pc));
        return -1;
    }
    rxBuf_server[size] = '\0';

    sprintf(txBuf_pc, "\r\nRX data : (%d)%s\r\n", size, rxBuf_server);
    pc.write(txBuf_pc, strlen(txBuf_pc));

    // close socket
    socket.close();

    // trouble-shooting
    if(size < SERVER_RES_SIZE)
    {
        ThisThread::sleep_for(chrono::seconds(2));
        continue;
    }

    // parse the command
    char command = rxBuf_server[size - 1] - '0';
    sprintf(txBuf_pc, "command : %c\r\n", command);
    pc.write(txBuf_pc, strlen(txBuf_pc));

    if(last_command != command)
    {

        if(command & 0x02)      modbus.write(LED_on, 8);
        else                  modbus.write(LED_off, 8);
    }
}

```

```

        ThisThread::sleep_for(chrono::milliseconds(100));

        if(command & 0x01)      modbus.write(HUM_on, 8);
        else                  modbus.write(HUM_off, 8);
    }
    last_command = command;

    ThisThread::sleep_for(chrono::seconds(2));
}
}

```

## MODBUS-TCP-Client

### 1. TCPPORT-30M에 연결된 DMS-MG90 제어

TCPPORT-30M은 PWM 신호를 보낼 수 있는 PWM OUT 디바이스를 가지고 있다. PWM OUT은 pulse width가 100ms로 고정되어 있고 modbus tcp 명령어를 통해 0~100% 사이 duty cycle을 제어할 수 있다.

그리고 문을 열고 닫기 위해 사용하는 servo motor인 DMS-MG90은 약 1ms pulse를 보내면 -90도 회전하고 약 2ms pulse를 보내면 90도 회전한다. 따라서 문을 열기 위해서는 duty cycle을 1%로 하라는 modbus tcp 명령어를 보내고, 문을 닫으라고 하기 위해서는 duty cycle을 2%로 하라는 modbus tcp 명령어를 보내면 된다. 이 명령어들은 web-server로부터 전송받는다.

### 2. HTU21D로부터 읽어들인 상대습도 TCPPORT-30M에 전송

HTU21D는 내부적으로 I2C 인터페이스를 사용하기 때문에 TCPPORT-30M과 직접적으로 연결할 수 없다. 따라서 I2C 통신이 가능한 NUCLEO-F411RE를 TCPPORT-30M과 HTU21D 사이에 두어 읽어들인 상대습도를 전송하는 역할을 하도록 구성했다.

NUCLEO-F411RE는 ESP-01을 통해 wifi로 local network에 접속하여 10초마다 HTU21D로부터 읽어들인 상대습도를 TCPPORT-30M의 OUTPUT PORT라는 디바이스에 write하는 modbus-tcp 명령어를 보내는데 그 이유는 다음과 같다.

TCPPORT-30M에 OUTPUT PORT 디바이스는 8개의 포트를 가지고 있고 각 포트들은 0과 1만 read, write가 가능한 coil 메모리 영역을 속한다. 그리고 HTU21D로부터 읽어들인 상대습도는 0~100 사이 값을 가지므로 7개의 bit로 표현 가능하다. 따라서 OUTPUT PORT는 1byte로 표현가능한 상대습도를 저장하기에 충분하다고 판단했다.

### 3. TCPPORT-30M의 1대 1 통신 특성으로 인한 web-server와 NUCLEO-F411RE 간 연결 조정 알고리즘

TCPPORT-30M은 Ethernet을 통해 local network에 접속한다. 그래서 한 기기와 이미 통신이 establish되었다면 다른 기기는 TCPPORT-30M과 통신하지 못한다.

TCPPORT-30M은 상대습도를 전송하는 NUCLEO-F411RE 뿐만 아니라 web-server와도 통신해야 하기 때문에 두 기기 모두 통신할 수 있도록 web-server와 NUCLEO-F411RE 간 연결 조정 알고리즘을 구현할 필요가 있었다.

NUCLEO-F411RE에서 TCPPORT-30M에 통신하기 위해 소켓을 open하고 connect할 때 이미 TCPPORT-30M이 웹서버와 통신하고 있다면 에러가 발생한다. 이 경우에는 이미 만들어진 socket을

close하고 2초를 기다린 뒤 다시 통신을 시도하도록 구현했다.

뿐만 아니라 NUCLEO-F411RE 내부적으로 문제가 발생했을 때 이미 만들어진 소켓을 close하지 않으면 웹서버가 통신할 수 없다. 따라서 문제가 발생할 때마다 이미 만들어진 socket을 close하고 2초를 기다린 뒤 다시 통신을 시도하도록 구현했다.

마지막으로 간혹 통신 문제로 TCPPORT-30M에서 보낸 response를 receive되지 않는 문제가 있는데 이를 해결하기 위해 socket의 10초짜리 time out을 설정하여 10초 안에 response를 받지 못하면 socket을 close하고 2초를 기다린 뒤 다시 통신하도록 구현했다.

#### 4. 코드

```
#include "mbed.h"
#include "HTU21D.h"

// IP and PORT# of MODBUS TCPPORT-30M
#define SERVER_IP "192.168.0.200"
#define SERVER_PORT 502
#define MAX_BUF_SIZE 1024

HTU21D htu21d(ARDUINO_UNO_D14, ARDUINO_UNO_D15);
UnbufferedSerial pc(CONSOLE_TX, CONSOLE_RX, 115200);

WiFiInterface *wifi;
TCPSocket socket;

char txBuf_pc[MAX_BUF_SIZE];
char txBuf_server[MAX_BUF_SIZE];
char rxBuf_server[MAX_BUF_SIZE];

char write_humi[] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x08, 0x01, 0x0F, 0x00, 0x10, 0x00, 0x08, 0x01, 0x00};

int main()
{
    SocketAddress sockAddr;
    SocketAddress serverAddr(SERVER_IP, SERVER_PORT);
    nsapi_error_t err;

    float humi;
    int i_humi, len;

    /* Connect to wifi */
    wifi = WiFiInterface::get_default_instance();
    if(!wifi)
    {
        sprintf(txBuf_pc, "ERROR: No WiFiInterface found.\r\n");
        pc.write(txBuf_pc, strlen(txBuf_pc));
        while(1){};
    }

    sprintf(txBuf_pc, "Connecting to %s...\r\n", MBED_CONF_APP_WIFI_SSID);
    pc.write(txBuf_pc, strlen(txBuf_pc));

    err = wifi->connect(MBED_CONF_APP_WIFI_SSID,
                         MBED_CONF_APP_WIFI_PASSWORD, NSAPI_SECURITY_WPA_WPA2);

    if(err != NSAPI_ERROR_OK)
    {
        sprintf(txBuf_pc, "Connection error!!\r\n");
        pc.write(txBuf_pc, strlen(txBuf_pc));
    }
}
```

```

        return -1;
    }

    sprintf(txBuf_pc, "Success!!\r\n");
    pc.write(txBuf_pc, strlen(txBuf_pc));

    wifi->get_ip_address(&sockAddr);
    sprintf(txBuf_pc, "IP : %s\r\n", sockAddr.get_ip_address());
    pc.write(txBuf_pc, strlen(txBuf_pc));

    socket.set_timeout(10 *1000); // Time out : 10s
    while(true)
    {
        // Open a TCP socket on the network interface
        socket.open(wifi);

        err = socket.connect(serverAddr);
        if(err != NSAPI_ERROR_OK)
        {
            sprintf(txBuf_pc, "Error connecting : %d\r\n", err);
            pc.write(txBuf_pc, strlen(txBuf_pc));

            socket.close();
            ThisThread::sleep_for(chrono::seconds(2));
            continue;
        }

        // read humidity
        if((humi = htu21d.read_humi()) == -1)
        {
            sprintf(txBuf_pc, "ERROR OCCURED WHILE READING HUMIDITY\r\n");
            pc.write(txBuf_pc, strlen(txBuf_pc));

            socket.close();
            ThisThread::sleep_for(chrono::seconds(2));
            continue;
        }
        else if(htu21d.crc_check())
        {
            sprintf(txBuf_pc, "CRC ERROR OCCURED WHILE READING HUMIDITY\r\n");
            pc.write(txBuf_pc, strlen(txBuf_pc));

            socket.close();
            ThisThread::sleep_for(chrono::seconds(2));
            continue;
        }

        i_humi = (int) humi;
        sprintf(txBuf_pc, "Relative Humidity = %d [%%]\r\n", i_humi);
        pc.write(txBuf_pc, strlen(txBuf_pc));

        // send write_humi
        len = sizeof(write_humi)/sizeof(write_humi[0]);
        write_humi[len - 1] = (i_humi & 0xff);

        socket.send((const char*)write_humi, 14);
        sprintf(txBuf_pc, "Send : %d %d\r\n"
                , write_humi[0], write_humi[1], write_humi[2], write_humi[3]
                , write_humi[4], write_humi[5], write_humi[6], write_humi[7]
                , write_humi[8], write_humi[9], write_humi[10], write_humi[11]
                , write_humi[12], write_humi[13]);
        pc.write(txBuf_pc, strlen(txBuf_pc));

        // recv from the server
        nsapi_size_or_error_t size = socket.recv(rxBuf_server, 1024);

```

```

        if(size <= 0)
        {
            if(size == NSAPI_ERROR_WOULD_BLOCK)
            {
                socket.close();
                ThisThread::sleep_for(chrono::seconds(2));
                continue;
            }

            sprintf(txBuf_pc, "Error while receiving data from TCP socket(%d)\r\n", size);
            pc.write(txBuf_pc, strlen(txBuf_pc));

            socket.close();
            ThisThread::sleep_for(chrono::seconds(2));
            continue;
        }
        rxBuf_server[size] = '\0';

        sprintf(txBuf_pc, "Recv : %d %d\r\n"
            , rxBuf_server[0], rxBuf_server[1], rxBuf_server[2], rxBuf_server[3]
            , rxBuf_server[4], rxBuf_server[5], rxBuf_server[6], rxBuf_server[7]
            , rxBuf_server[8], rxBuf_server[9], rxBuf_server[10], rxBuf_server[11]
            );
        pc.write(txBuf_pc, strlen(txBuf_pc));

        // close socket
        socket.close();

        ThisThread::sleep_for(chrono::seconds(10));
    }

    wifi->disconnect();

    while(true){};
}

```

## Web Server

### 1. Back End

- Modbus-RTU-Client와의 통신

- Description

HTTP 프로토콜을 사용하며, LED와 Humidifier의 On/Off에 대한 통신을 한다. 해당 URL(<http://webserver/status>)로 Request를 받으면 그에 대한 Response를 보낸다.

- LED, Humidifier On/Off

- Receive

Modbus-RTU-Client는 LED와 Humidifier의 On/Off에 대한 여부를 물기 위해 2초마다 Web Server로 Request한다.

- Transmit

Web Server는 저장된 제어명령을 0~3의 정수로 변환하여 Response를 보낸다.

0: LED Off, Humidifier Off

- 1: LED Off, Humidifier On
- 2: LED On, Humidifier Off
- 3: LED On, Humidifier On

- Code

```
@app.route('/status')
def status():
    global led, humidifier
    ret = 0
    if led:
        ret += 2
    if humidifier:
        ret += 1
    return str(ret)
```

- Modbus-TCP PORT-30M와의 통신

- Description

Modbus-TCP 프로토콜을 사용하며, doorlock을 On/Off 하는 명령을 주거나, humidity를 읽어온다.

- doorlock On/Off

- Transmit

doorlock On을 명령할 때는 PWM의 Duty 비를 1%로 설정하도록 송신한다. 아래와 같이 Modbus-TCP 프로토콜 패킷을 보낸다.



[Modbus-TCP Request 패킷]

Function Code: 0x06 (Write Single Registers)

Start Address: 0x00 0x04

Data (PWM Duty): 0x00 0x01

Packet: 0x00 0x00 0x00 0x00 0x00 0x06 0x01 0x06 0x00 0x04 0x00  
0x01

doorlock Off를 명령할 때는 PWM의 Duty비를 2%로 설정하도록 송신한다. 아래와 같이 Modbus-TCP 프로토콜 패킷을 보낸다.



#### [Modbus-TCP Request 패킷]

Function Code: 0x06 (Write Single Registers)

Start Address: 0x00 0x04

Data (PWM Duty): 0x00 0x02

Packet: 0x00 0x00 0x00 0x00 0x00 0x06 0x01 0x06 0x00 0x04 0x00  
0x02

- Code

아래 코드는 현재 도어락을 toggle 시키는 함수이며, 내부적으로 사용된 modbus\_send()는 직접 구현한 함수로 TCPPORT 장비와 TCP 연결을 열고 해당 패킷을 보낸 뒤 연결을 닫는 일을 수행한다.

```
def toggle_door():
    door_on = b"\x00\x00\x00\x00\x00\x06\x01\x06\x00\x04\x00\x01"
    door_off = b"\x00\x00\x00\x00\x00\x06\x01\x06\x00\x04\x00\x02"

    global door
    door = not door
    if door:
        modbus_send(door_on)
    else:
        modbus_send(door_off)
```

- Read humidity

- Transmit

humidity에 대한 정보는 Modbus-TCP-client에 의해서 Coil 메모리 16~23번지에 7bit의 값으로 업데이트 되고 있고, 그것을 읽기 위한 Modbus-TCP 패킷을 아래와 같이 보내면 된다.



#### [Modbus-TCP Request 패킷]

Function Code: 0x01 (Read Single Coil)

Start Address: 0x00 0x10

bit length: 0x00 0x07

Packet: 0x00 0x00 0x00 0x00 0x00 0x06 0x01 0x01 0x00 0x10 0x00  
0x07

- Receive

위와 같이 Read Single Coil 패킷을 보내면 아래와 같은 Response 패킷이 온다.

💡 [Modbus-TCP Response 패킷 예시(습도 69%)]  
 Function Code: 0x01 (Read Single Coil)  
 Data: 0x00 0x45 (69)  
 Packet: 0x00 0x00 0x00 0x00 0x00 0x05 0x01 0x01 0x02 0x00 0x45

- code

아래 코드는 humidity를 읽어오기 위해 Modbus-TCP Request를 보내고, 받은 Response의 맨 마지막 byte를 현재 습도로 업데이트 하는 코드이다.

```

def update_humidity():
    global humidity
    humidity_read = b"\x00\x00\x00\x00\x00\x06\x01\x01\x00\x10\x00\x07"
    response = modbus_send(humidity_read)
    if response is not -1:
        humidity = response[-1]
  
```

## 2. Front End

- UI image

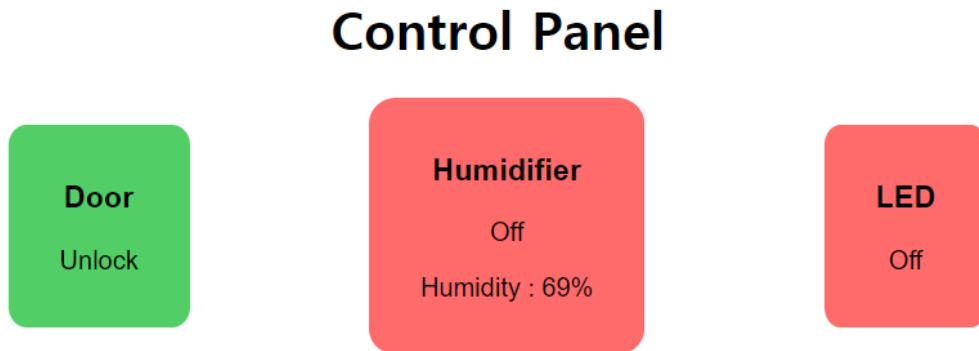


Fig 12. Web UI

- URL을 통해 제공되는 기능

Control Panel은 유저에게 다음 4가지 기능을 URL 형식으로 제공한다.

- 1) Doorlock On/Off 제어 및 현재 상태 보기

URL: /door

해당 URL 요청 시, Doorlock이 toggle 되며, 최종 상태가 반환된다.

- 2) Humidifier On/Off 제어 및 현재 상태 보기

URL: /humidifier

해당 URL 요청시, humidifier가 toggle 되며, 최종 상태가 반환된다.

3) LED On/Off 제어 및 현재 상태 보기

URL: /led

해당 URL 요청시, led가 toggle 되며, 최종 상태가 반환된다.

4) 현재 Humidity 보기

URL: /humidity

해당 URL 요청시, 서버의 humidity 정보가 update 되며, update 된 습도가 반환된다.

- 편리한 UI/ UX

- Ajax를 통한 비동기 통신

Ajax를 통해 웹브라우저가 update 될 때, 페이지를 전부 새로고침 하는 것이 아니라 필요한 부분만 update 된다.

- Waiting 버튼

사용자가 버튼을 클릭하면 Ajax를 통해 서버로부터 응답이 오기 전까지 버튼은 waiting 상태가 된다. 이때 버튼은 노란색으로 변하고 클릭해도 클릭이 되지 않는다. 이를 통해 불필요한 버튼 클릭을 막을 수 있고 사용자가 직관적으로 현재 상태를 알 수 있다. 따라서 사용자 경험을 향상시킬 수 있다.

- URL 기능을 button으로 사용

제공되는 URL 기능을 button 클릭으로 간단하게 사용 가능하다.

- 습도 자동 update

Ajax를 통해 웹브라우저가 습도를 10초마다 서버로부터 제공받은 습도로 update 한다.

## Team composition



- 채영민 : Back End, HW 환경 설정 및 구성 등
- 김석진 : Front End, MODBUS-RTU-Client, MODBUS-TCP-Client 등

## Links



- Youtube : [한동대학교 21-1학기 임베디드 파이널 프로젝트 : Bethel324 control system 시현 영상](#)
- Github : <https://github.com/SukJinKim/Bethel324-control-system>

## References

---

### MODBUS 프로토콜 관련

1. MODBUS PROTOCOL SPECS : <https://www.modbus.org/specs.php>
2. MODBUS PROTOCOL SPECIFICATION V1.1b3 :  
[https://www.modbus.org/docs/Modbus\\_Application\\_Protocol\\_V1\\_1b3.pdf](https://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf)
3. MODBUS over Serial Line Specification and Implementation Guide V1.02 :  
[https://www.modbus.org/docs/Modbus\\_over\\_serial\\_line\\_V1\\_02.pdf](https://www.modbus.org/docs/Modbus_over_serial_line_V1_02.pdf)
4. MODBUS MESSAGING ON TCP/IP IMPLEMENTATION GUIDE V1.0b :  
[https://www.modbus.org/docs/Modbus\\_Messaging\\_Implementation\\_Guide\\_V1\\_0b.pdf](https://www.modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf)

### MODBUS RTU Relay 관련

1. Description : <https://www.waveshare.com/modbus-rtu-relay.htm>
2. Protocol manual : [https://www.waveshare.com/wiki/Protocol\\_Manual\\_of\\_Modbus\\_RTU\\_Relay](https://www.waveshare.com/wiki/Protocol_Manual_of_Modbus_RTU_Relay)

### MAX-485 관련

1. MAX-485 datasheet : <http://www.openimpulse.com/blog/wp-content/uploads/wpsc/downloadables/MAX485-Datasheet.pdf>

### TCPPORT-30M 관련

1. TCPPORT-30M official doc : <http://comfilewiki.co.kr/ko/doku.php?id=tcpport:index>