KON435E - INDUSTRIAL DATA COMMUNICATIONS

HOMEWORK - I

GROUP 6

BATUHAN AVCI – 040190555

ERDEN BAHADIR - 040190774

ENES CAN GÜVEN – 040190776

BERKANT BALKAN - 040190789

DELIVERY DATE: 19/11/2022

LECTURER: ASSOC. PROF. ALİ FUAT ERGENÇ

CONTENTS

1. GENERAL INFORMATIONS

In this application, simple Modbus RTU Protocol implemented for two ESP32 (one master – one slave). Our scenario includes the measurement of temperature and humudity and controlling relays for provided conditions. Master sends a query to the slave for the measurement of temperature and humidity. Then slave responds to the sended query. By the checking of respond Master sends relay on/off commands with a query depending on the measurement values and the set values. (heater relay on/off, humidifier relay on/off). Slave takes action for the commands and responds with the necessary response.

1.1. About Modbus Protocol

What is Modbus?

Modbus is a serial communication protocol developed by Modicon published by Modicon in 1979 for use with its programmable logic controllers (PLCs). In simple terms, it is a method used for transmitting information over serial lines between electronic devices. The device requesting the information is called the Modbus Master and the devices supplying information are Modbus Slaves. In a standard Modbus network, there is one Master and up to 247 Slaves, each with a unique Slave Address from 1 to 247. The Master can also write information to the Slaves.

What is Modbus RTU?

Modbus RTU is an open, serial (RS-232 or RS-485) protocol derived from the Master/Slave architecture. It is a widely accepted protocol due to its ease of use and reliability. Modbus RTU is widely used within Building Management Systems (BMS) and Industrial Automation Systems (IAS). This wide acceptance is due in large part to MODBUS RTU's ease of use.

MODBUS RTU messages are a simple 16-bit CRC (Cyclic-Redundant Checksum). The simplicity of these messages is to ensure reliability. Due to this simplicity, the basic 16-bit MODBUS RTU register structure can be used to pack in floating point, tables, ASCII text, queues, and other unrelated data.

1.2. CRC for MODBUS RTU

CRC is fundamental theory of cyclic error correcting codes. By the usage of systematic representation of cyclic codes, which encodes the data by adding a constant length check value for detecting of errors during communication period. It is really important to check the errors caused in transmission of the message in many communication channels including magnetic and optical storage devices.

Calculation performed both in master and slave during the transmission where they evaluate fixed length binary sequence as CRC value for each packet of data to be sent or stored. At the end of calculation, they appends the check value to the data to create a codeword. Each time calculated values are compared and if they don't match, the data block contains an error. In consideration of situation, master can request another query to be performed by slave by the help of CRC error analysis.

In our implementation, we created our own MODBUS RTU and CRC functions to provide a healthy dataflow between ESP32s amongst the RS485 modules. In MODBUS RTU CRC part of the data is indicated by 2 bytes. Below code sample of our team is for CRC:

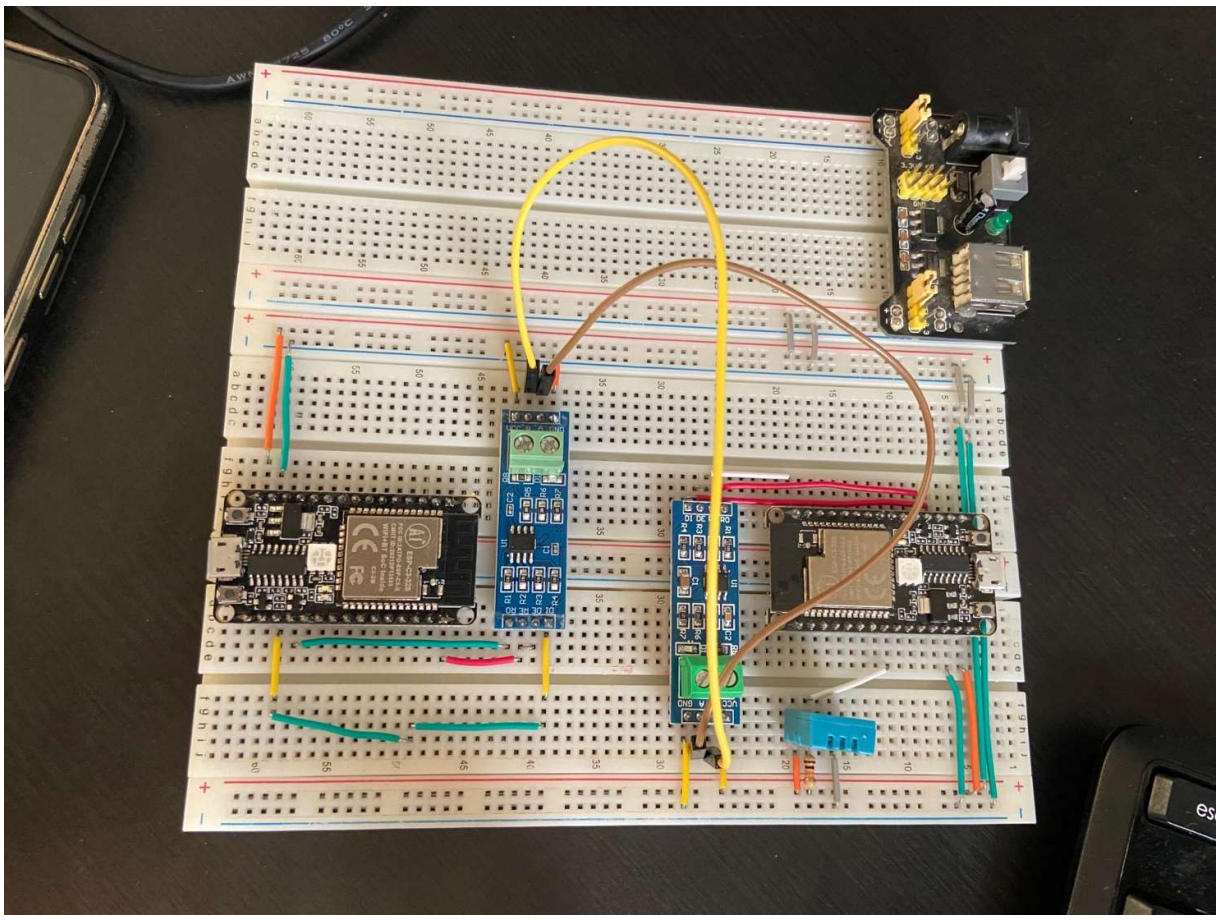
```
unsigned int calculateCRC(unsigned char bufferSize, unsigned char frame[])
  {
    unsigned int temp, temp2, flag;
    temp = 0xFFFF;
    for (unsigned char i = 0; i < bufferSize; i++) {
      temp = temp ^ frame[i];
      for (unsigned char j = 1; j <= 8; j++) {
        flag = temp & 0x0001;
        temp >>= 1;
        if (flag)
          temp ^= 0xA001;
      }
    }
    // Reverse byte order.
    temp2 = temp >> 8;
    temp = (temp << 8) | temp2;
    temp &= 0xFFFF;
   return temp;}
```

General CRC calculation at sender side can be summarized like below:

- Calculation of the length of the divisor descripted by L
- Appending L-1 bits to the original message
- Performing binary division
- Providing the reminder as CRC

2. DESIGN PROCESS

2.1. Circuit Design



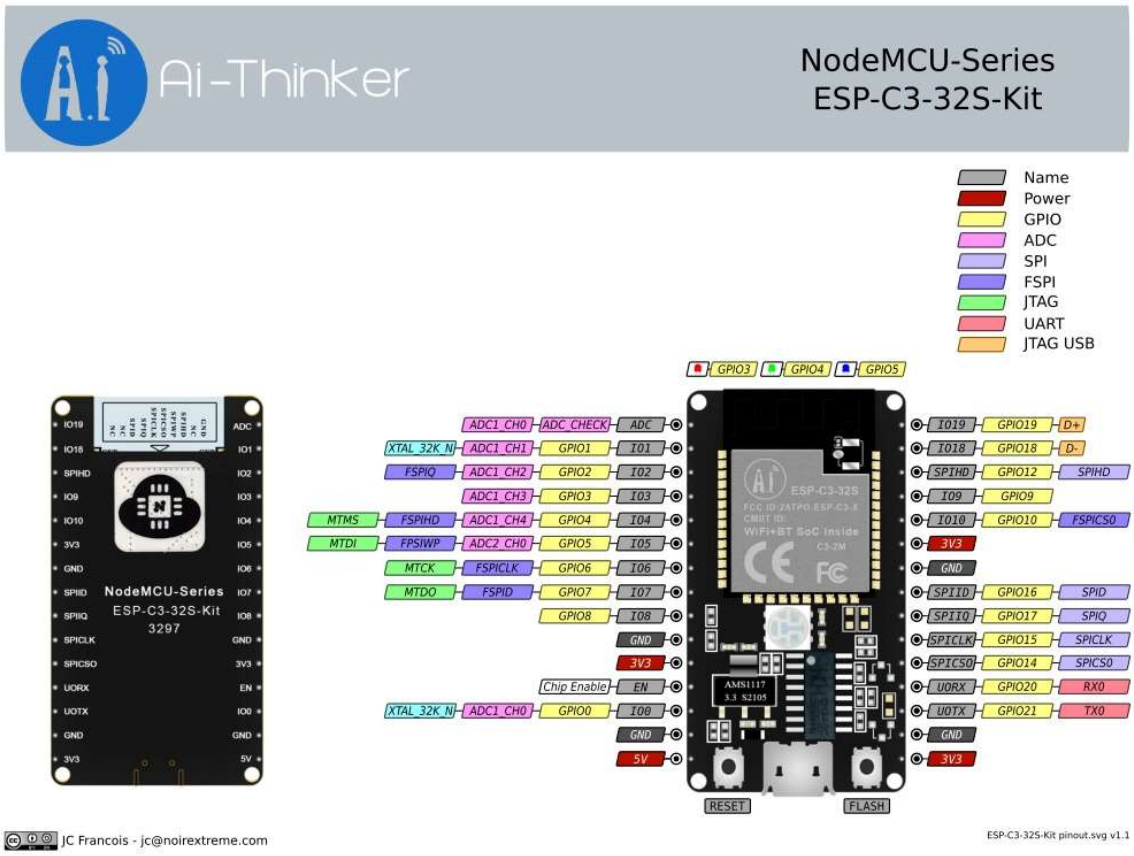
Master ESP32

Port	Connection
Tx0	RS485di
Rx	RS485RO
GPIO19	ESP32 Master – Tx Enable Pin

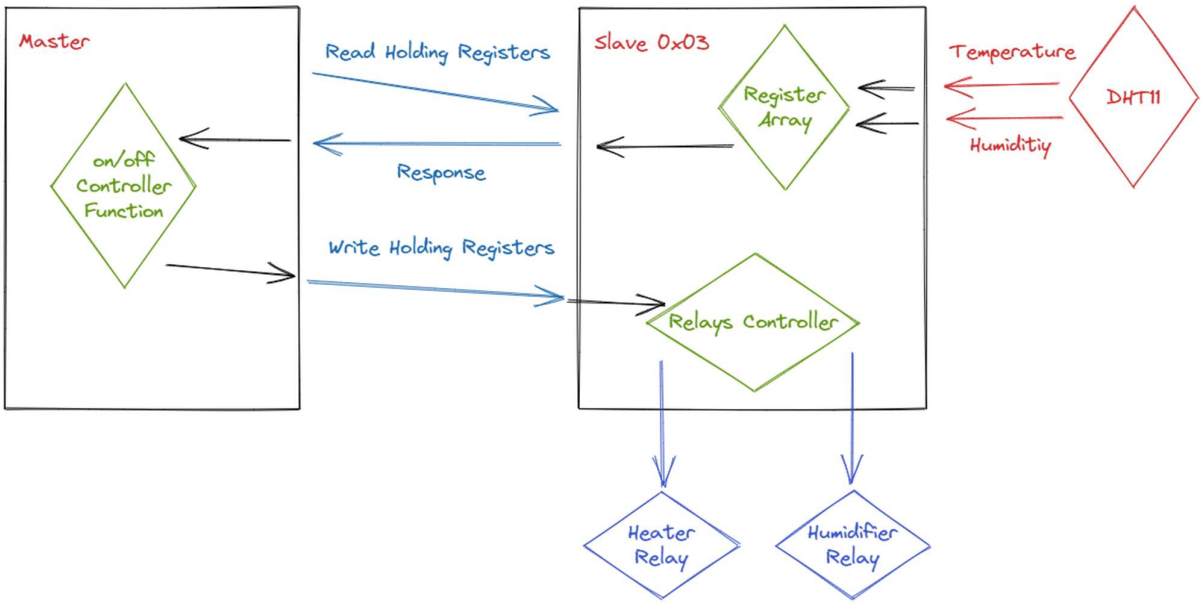
Slave ESP32

Port	Connection
Tx0	RS485di
Rx	RS485RO
GPIO19	ESP32 Master – Tx Enable Pin
GPIO 8	DHT11 Data
GPIO 3 (Red Builtin LED)	Relay Heater
GPIO 3 (Blue Builtin LED)	Relay Humidifier

RS485 ANB pins connected with each other. 5.5 Vin and Gnd pins connected breadboard power supplier. ESP32 pin out is shown below.



2.2. Coding



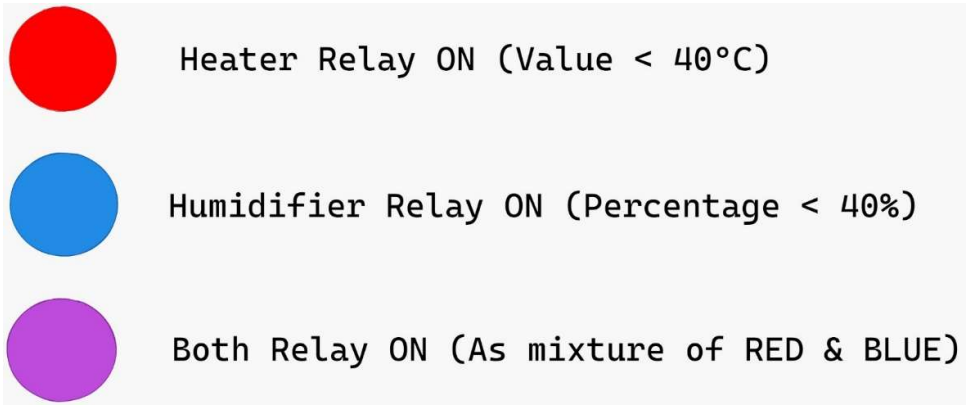
In figure, Master and Slave ESP32 code flow is shown. In figure, arrows state the Modbus RTU Package. The package structures shown below. Also Arduino codes added to end of this report. Functions and variables detailed in code as command lines.

Read Holding Register – Master -> Slave:

Read Holding Register Response – Master -> Slave:

Write Multiple Holding Registers:

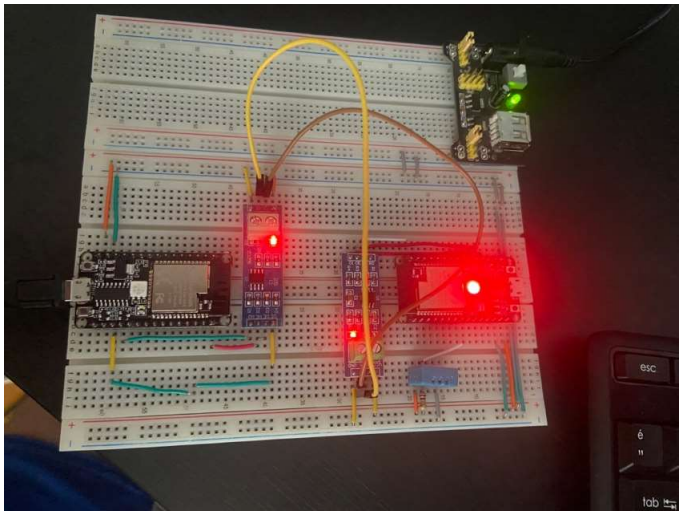
3. EXPECTATIONS



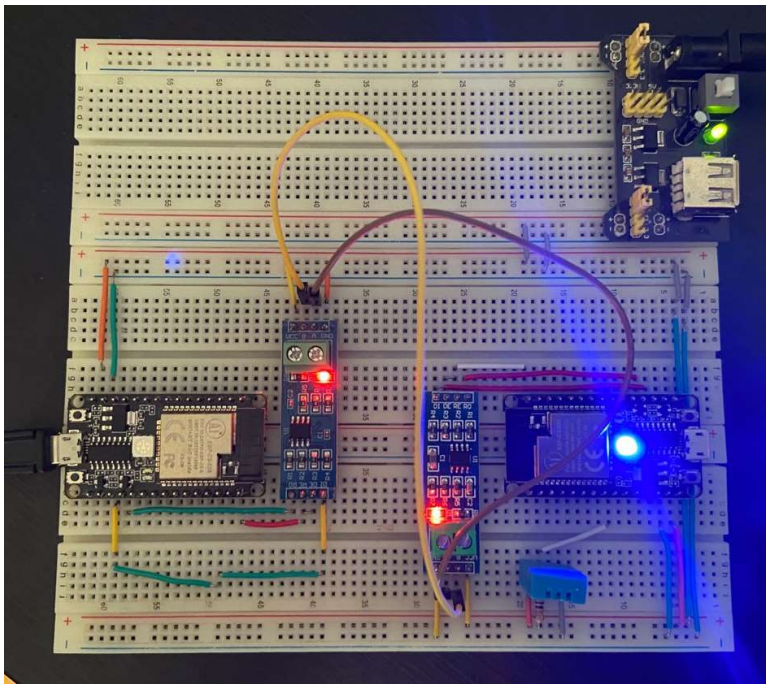
In this application, Relays are simulated by RGB LEDs. As stated above figure, heater stated by ‘’Red LED’’, humidifier stated by ‘’Blue LED’’ and both relays are ON, LED indicates ‘’Purple’’.

4. APPLICATION RESULTS AND APPLICATION VIDEO’S LINK

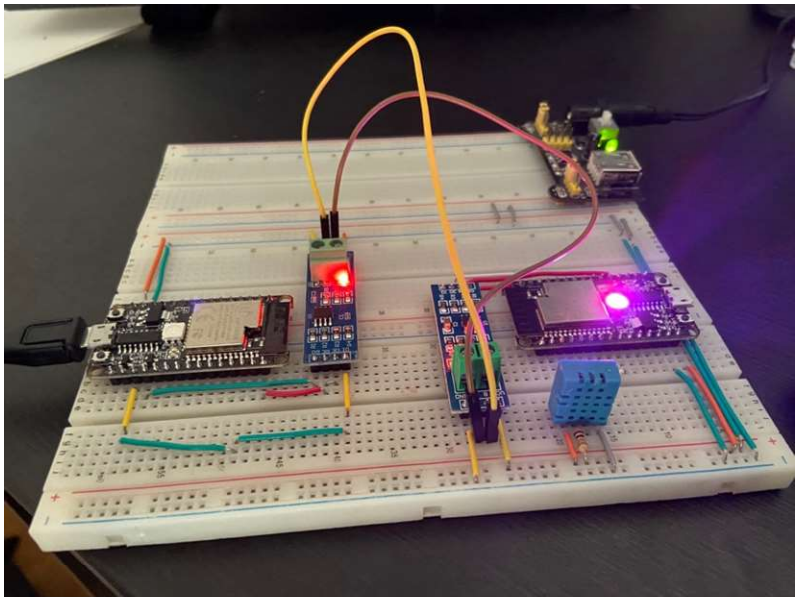
When Heater Value is less than 40 C°:



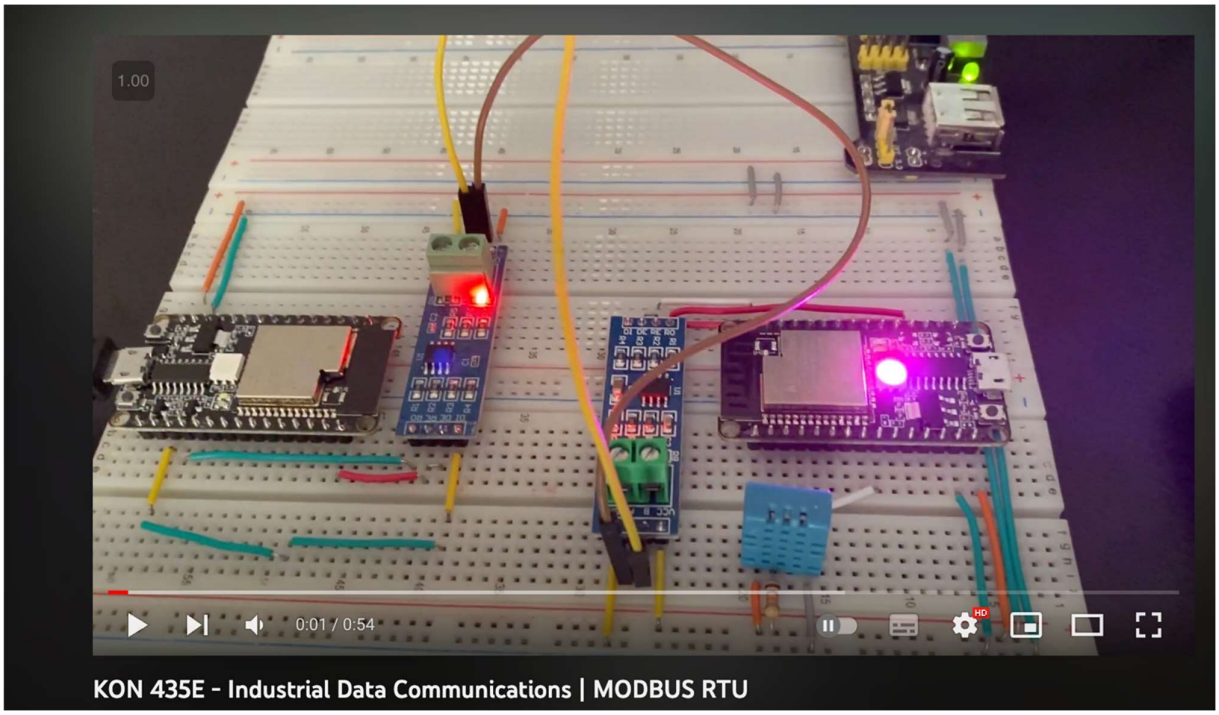
When Humilifier percentage is less than %40:



When Both Conditions are provided:



And Application Video's Link:



https://youtu.be/63SXL3AwyQA

In the video we used person breathe as humidifier and the hair dryer performs as dropper for humudity and booster for temperature. By the help of this conditions we created different scenarios. At first both temperature and humidity conditions is below desired value. It can be thinked like bothe relays are on. Then we increased the humidity level and the blue led is off meaning like humidifier relay is off, and only the temperature relay is on indicated with red led. Then we used hair dryer to drop the humidity level and boost the temperature level.

Master ESP32

```
#define READ_HOLDING_REGISTERS 3
#define tx_enable 19

////////////////////// Port information ///////////////////
#define baud 115200

unsigned int T1_5; // inter character time out in microseconds
unsigned int T3_5; // frame delay in microseconds

  unsigned char frame[24];  //frame to be send for READ_HOLDING_REGISTERS (to slave)
  unsigned char frame_rcv[24]; //frame to be received for READ_HOLDING_REGISTERS (from slave)
  unsigned char  pwh_frame[24]; // frame to be send for WRITE_MULTIPLE_HOLDING_REGISTERS (to slave)

  float sicaklik;
  float nem;


  struct Packet {

    unsigned char id;
    unsigned char function;
    unsigned char address;
    unsigned int no_of_registers;
    unsigned int* register_array;

  };



  Packet packet; //packet for read measurement registers

  Packet pwh; // packet for write heaters and humidifier registers

  unsigned int regs[9];
  unsigned int pwh_regs[9];

  void setup() {

    packet.id = 0x03;              // Slave ID
    packet.function = 0x03;        // READ_HOLDING_REGISTERS function code
    packet.address = 0;            // Address of HOLDING_REGISTERS
    packet.no_of_registers = 2;    // Number of HOLDING_REGISTERS
    packet.register_array = regs; // HOLDING_REGISTERS value array

    pwh.id = 0x03;                 // Slave ID
    pwh.function = 0x06;           // WRITE_MULTIPLE_HOLDING_REGISTERS function code
    pwh.address = 0;               // Address of HOLDING_REGISTERS
    pwh.no_of_registers = 2;       // Number of HOLDING_REGISTERS
    pwh.register_array = pwh_regs;// HOLDING_REGISTERS value array

```

```
53.     Serial.begin(baud);
54.     T1_5 = 15000000 / baud;
55.     T3_5 = 35000000 / baud;
56.
57.     pinMode(tx_enable, OUTPUT);
58.     digitalWrite(tx_enable, LOW);
59.     pinMode(3, OUTPUT);
60.
61.     digitalWrite(3, HIGH);
62.     delay(1000);
63.     digitalWrite(3, LOW);
64.
65.   }
66.
67.   void loop() {
68.     unsigned int crc16;
69.
70.     frame[0] = packet.id;
71.     frame[1] = packet.function;
72.     frame[2] = packet.address >> 8;           // address Hi
73.     frame[3] = packet.address & 0xFF;         // address Lo
74.     frame[4] = packet.no_of_registers >> 8;   // no_of_registers Hi
75.     frame[5] = packet.no_of_registers & 0xFF; // no_of_registers Lo
76.     crc16 = calculateCRC(6, frame);           // calculate CRC
77.     frame[6] = crc16 >> 8;                    // crc16 Hi
78.     frame[7] = crc16 & 0xFF;                  // crc16 Lo
79.
80.     pwh_frame[0] = pwh.id;
81.     pwh_frame[1] = pwh.function;
82.     unsigned char noOfBytes = pwh.no_of_registers * 2;
83.     pwh_frame[2] = pwh.address >> 8;              // address Hi
84.     pwh_frame[3] = pwh.address & 0xFF;            // address Lo
85.     pwh_frame[4] = pwh.no_of_registers >> 8;      // no_of_registers Hi
86.     pwh_frame[5] = pwh.no_of_registers & 0xFF;
87.     pwh_frame[6] = pwh.register_array[0] >> 8;    // Temperature data Hi
88.     pwh_frame[7] = pwh.register_array[0] & 0xFF;  //Humidity data Lo
89.
90.     pwh_frame[8] = pwh.register_array[1] >> 8;    // Humidity data Hi
91.     pwh_frame[9] = pwh.register_array[1] & 0xFF;  //Humidity data Lo
92.
93.     crc16 = calculateCRC(8, pwh_frame);
94.     pwh_frame[10] = crc16 >> 8;                   //crc16 Hi
95.     pwh_frame[11] = crc16 & 0xFF;                 // crc16 Lo
96.
97.
98.     digitalWrite(tx_enable, HIGH);                // Enable tx
99.     for (unsigned char i = 0; i < 8; i++)
100.      Serial.write(frame[i]);
101.    Serial.flush();
102.    delayMicroseconds(T1_5);
103.    digitalWrite(tx_enable, LOW);                  // Enable tx
104.
105.    delay(100);
106.
107.    unsigned char buffer_r = 0;
108.    while (Serial.available()) {
109.      frame_rcv[buffer_r] = Serial.read();
110.      buffer_r++;
```

```
111.      delayMicroseconds(T1_5);
112.    }
113.
114.    if (buffer_r > 6) {                              // buffer size at
    least 6
115.      if (frame_rcv[0] == 0x03) {                    // check slave ID
116.        if (frame_rcv[1] == 0x03) {                  // check for funtion
    code
117.          unsigned char index = 3;
118.          for (unsigned char i = 0; i < packet.no_of_registers; i++) {
119.
120.            packet.register_array[i] = (frame_rcv[index] << 8) |
    frame_rcv[index + 1];
121.            index += 2;
122.          }
123.        }
124.      }
125.    }
126.
127.    digitalWrite(tx_enable, HIGH);
128.    for (unsigned char i = 0; i < 12; i++)
129.      Serial.write(pwh_frame[i]);
130.    Serial.flush();
131.    delayMicroseconds(T1_5);
132.    digitalWrite(tx_enable, LOW);
133.    delay(100);
134.
135.
136.    sicaklik = packet.register_array[0];
137.    nem = packet.register_array[1];
138.    Serial.println(sicaklik);
139.    Serial.println(nem);
140.    Serial.println(frame_rcv[1]);
141.    if (sicaklik <= 40) {
142.      pwh.register_array[0] = 1;
143.    }
144.    else {
145.      pwh.register_array[0] = 0;
146.    }
147.
148.    if (nem <= 40) {
149.      pwh.register_array[1] = 1;
150.    }
151.    else {
152.      pwh.register_array[1] = 0;
153.    }
154.    delay(1000);
155.  }
156.
157.  unsigned int calculateCRC(unsigned char bufferSize, unsigned char
    frame[])
158.  {
159.    unsigned int temp, temp2, flag;
160.    temp = 0xFFFF;
161.    for (unsigned char i = 0; i < bufferSize; i++) {
162.      temp = temp ^ frame[i];
163.      for (unsigned char j = 1; j <= 8; j++) {
164.        flag = temp & 0x0001;
```

```
165.        temp >>= 1;
166.        if (flag)
167.          temp ^= 0xA001;
168.      }
169.    }
170.    // Reverse byte order.
171.    temp2 = temp >> 8;
172.    temp = (temp << 8) | temp2;
173.    temp &= 0xFFFF;
174.    return temp; // the returned value is already swopped - crcLo byte is
   first & crcHi byte is last
175. }
```

Slave ESP32

```
#include "DHT.h"
#define DHTPIN 8
#define DHTTYPE DHT11
#define READ_HOLDING_REGISTERS 0x03
#define WRITE_MULTIPLE_HOLDING_REGISTERS 0X10
DHT dht(DHTPIN, DHTTYPE);
unsigned long currTime;
unsigned long prevTime;
float t = 0;
float h = 0;
//#define HEATER_LED 12
//#define HUMIDIFIER_LED 18
#define tx_enable 19
unsigned char frame[24];
unsigned char frame_snd[24];
unsigned char pwh_frame_snd[24];
unsigned int T1_5;
unsigned int T3_5;// inter character time out in microseconds
#define baud 115200
unsigned char slaveID = 3;
//unsigned int sicaklik;
float sicaklik = 0;
float mData[2];
unsigned int heater_state = 0;
unsigned int states[2];
unsigned int humidifier_state = 0;
struct Packet {

  unsigned char id;
  unsigned char function;
  unsigned char address;
  unsigned int no_of_registers;
  unsigned int* register_array;

};

Packet packet;


void setup() {
  Serial.begin(baud);
  T1_5 = 15000000 / baud;
  T3_5 = 35000000 / baud;
  pinMode(tx_enable, OUTPUT);

  digitalWrite(tx_enable, HIGH);
  delay(1000);
  digitalWrite(tx_enable, LOW);

  pinMode(3, OUTPUT);
  pinMode(5, OUTPUT);
  pinMode(4, OUTPUT);

  dht.begin();

  digitalWrite(4, HIGH);
```

```
  delay(1000);
  digitalWrite(4, LOW);


}

void loop() {
  //  sicaklik = analogRead(analogPin);
  currTime = millis();
  if (currTime - prevTime >= 2000) {
    mData[0] = dht.readTemperature();
    mData[1] = dht.readHumidity();
  }
  unsigned char buffer_ = 0;
  unsigned char overflow = 0;

  while (Serial.available()) {
    //    Serial.println("asdas"); //buraya giriyor
    frame[buffer_] = Serial.read();
    buffer_++;
    delayMicroseconds(T1_5);

  }


  if (buffer_ > 6) {
    //    Serial.println("asdas");
    //    Serial.println(frame[0]);
    //    Serial.println(frame[1]);

    packet.id = frame[0];
    packet.function = frame[1];
    unsigned char startingAddress = ((frame[2] << 8) | frame[3]);
    packet.address = ((frame[2] << 8) | frame[3]);
    //    packet.no_of_registers =  ((frame[4] << 8) | frame[5]);
    //    unsigned int maxData = packet.address + packet.no_of_registers;
    unsigned char index;
    //    unsigned char noOfBytes = packet.no_of_registers * 2;

    if (packet.id == slaveID) {
            packet.no_of_registers =  ((frame[4] << 8) | frame[5]);
            unsigned int maxData = packet.address + packet.no_of_registers;
            unsigned char noOfBytes = packet.no_of_registers * 2;

      if (packet.function == READ_HOLDING_REGISTERS) { //frame 1 yapalim
//        packet.no_of_registers =  ((frame[4] << 8) | frame[5]);
//        unsigned int maxData = packet.address + packet.no_of_registers;
//        unsigned char noOfBytes = packet.no_of_registers * 2;
        unsigned int crc_calculated = calculateCRC(buffer_ - 2, frame);
        //        Serial.println(crc_calculated);
        unsigned int crc_received = ((frame[buffer_ - 2] << 8) | frame[buffer_ -
1]);
        //        Serial.println(crc_received);

        if (crc_calculated == crc_received) {
          digitalWrite(4, HIGH);

          frame_snd[0] = slaveID;
```

```
        frame_snd[1] = READ_HOLDING_REGISTERS;
        frame_snd[2] = noOfBytes;
        unsigned char responseFrameSize = 5 + noOfBytes;
        unsigned char address = 3; // PDU starts at the 4th byte
        unsigned int temp;

        for (index = startingAddress; index < maxData; index++) {

          temp = mData[index]; // elemanin yazdigi data

          frame_snd[address] = temp >> 8;
          address++;
          frame_snd[address] = temp & 0xFF;
          address++;
        }
        frame_snd[responseFrameSize - 2] = 0xFF;
        frame_snd[responseFrameSize - 1] = 0xFF;

        digitalWrite(tx_enable, HIGH);
        for (unsigned char i = 0; i < responseFrameSize; i++) {
          Serial.write(frame_snd[i]);
        }
        Serial.flush();
        delayMicroseconds(T3_5);
        digitalWrite(tx_enable, LOW);

      }



    }
    else if (packet.function == WRITE_MULTIPLE_HOLDING_REGISTERS) {

      pwh_frame_snd[0] = slaveID;
      pwh_frame_snd[1] = WRITE_MULTIPLE_HOLDING_REGISTERS;
      unsigned char address = 6; // PDU starts at the 4th byte
      heater_state = ((frame[6] << 8) | frame[7]);
      humidifier_state = ((frame[8] << 8) | frame[9]);
      Serial.print(humidifier_state);

    }
  }

}
if (heater_state == 1) {
  digitalWrite(3, HIGH);
}
else {
  digitalWrite(3, LOW);
}

if (humidifier_state == 1) {
  digitalWrite(5, HIGH);
}
else {
  digitalWrite(5, LOW);
}
```

```
  digitalWrite(4, LOW);

}

unsigned int calculateCRC(unsigned char bufferSize, unsigned char frame[])
{
  unsigned int temp, temp2, flag;
  temp = 0xFFFF;
  for (unsigned char i = 0; i < bufferSize; i++) {
    temp = temp ^ frame[i];
    for (unsigned char j = 1; j <= 8; j++) {
      flag = temp & 0x0001;
      temp >>= 1;
      if (flag)
        temp ^= 0xA001;
    }
  }
  // Reverse byte order.
  temp2 = temp >> 8;
  temp = (temp << 8) | temp2;
  temp &= 0xFFFF;
  return temp; // the returned value is already swopped - crcLo byte is first &
crcHi byte is last
}
```

REFERENCES

KON 435E – Industrial Data Communications lecture notes.

CRC Generating and Checking, Thomas Schmidt, Microchip Technology Inc. 2000.

Peterson, W. W. and Brown, D.T. “Cyclic Codes for Error Detection.” In Proceedings of the IRE, January 1961, 228–235

Tanenbaum, Andrew S. Computer Networks, Second Edition. Prentice Hall, 1988.