**BỘ CÔNG THƯƠNG**

**TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP TP. HCM**



**CHUYÊN ĐỀ IOT**

**BÀI BÁO CÁO THỰC HÀNH BUỔI 7**

**NHÓM 6**

Giảng viên : **PHẠM QUANG TRÍ**

Sinh viên :

Hoàng Đỗ Duy Hải- 21130351

Đoàn Mạnh Tú - 21007261

Dương Minh Toàn - 21069251

Yêu cầu 1:

**Mục lục:**

[I.Yêu cầu 2: Sơ đồ nguyên lý. 4](#_Toc179408580)

[II.Yêu cầu 3: Lưu đồ giải thuật. 5](#_Toc179408581)

[III.Yêu cầu 4: Mã nguồn chương trình. 8](#_Toc179408583)

[IV.Yêu cầu 5: Giải thích mã nguồn. 17](#_Toc179408584)

[V.Yêu cầu 6: Minh chứng kết quả thực nghiệm. 32](#_Toc179408585)

[VI.Yêu cầu 7: Video.: 33](#_Toc179408586)

**Bảng phân nhiệm vụ từng thành viên****:**

|  |  |
| --- | --- |
| **Họ và tên** | **Nhiệm vụ** |
| Hoàng Đỗ Duy Hải | Vẽ sơ đồ nguyên lý, làm code |
| Đoàn Mạnh Tú | Làm word, lưu đồ giải thuật |
| Dương Minh Toàn | Làm video |

**Yêu cầu bài:**

Bài tập mức độ 3 (10 điểm):

Hệ thống IoT bao gồm 2 thành phần: mô-đun Raspberry “Master” và mô-đun Raspberry “Slave”.

Master: In giá trị nhiệt độ, độ ẩm nhận được từ Slave ra màn hình Raspberry.

Gửi giá trị nhiệt độ, độ ẩm trung bình trong 20 giây nhận được từ Slave lên Server ThingSpeak mỗi 20 giây. Gửi tín hiệu điều khiển 3 LED (độc lập và tùy biến).

Slave: Đọc giá trị nhiệt độ, độ ẩm từ cảm biến nhiệt độ, độ ẩm và hiển thị các giá trị này lên LCD 16x2. Đồng thời gửi giá trị này đến Master mỗi 1 giây. Điều khiển 3LED theo yêu cầu của Master.

Lưu ý: Nhóm bắt buộc phải tự xây dựng được FRAME truyền đáp ứng yêu cầu trên. Trong FRAME truyền phải có CRC. FRAME truyền phải được thiết kế có tính logic, bao quát được tất cả các trường hợp. Giải thích chi tiết ý nghĩa các thông tin trong FRAME truyền vào báo cáo thực hành và video thuyết trình. Bắt buộc nhóm 1, 3, 5, 7, 9 và 11 sử dụng giao thức UDP; nhóm 2, 4, 6, 8, 10 và 12 sử dụng giao thức TCP/IP để trao đổi dữ liệu.

Bài tập mức độ 3+ (13 điểm):

Từ mức độ 3:

Bổ sung thêm 1 mô-đun Raspbery X có chức năng như sau:

Đọc giá trị cảm biến khoảng cách, biến trở và gửi về cho

Master mỗi 1 giây. Điều khiển 3 LED theo yêu cầu của

Master.

Master: Thêm chức năng in giá trị cảm biến mới nhận được

từ Raspberry X ra màn hình Raspberry mỗi khi nhận được tín

hiệu. Gửi dữ liệu trung bình của giá trị cảm biến mới này lên

Server ThingSpeak mỗi 20 giây. Gửi tín hiệu điều khiển 3

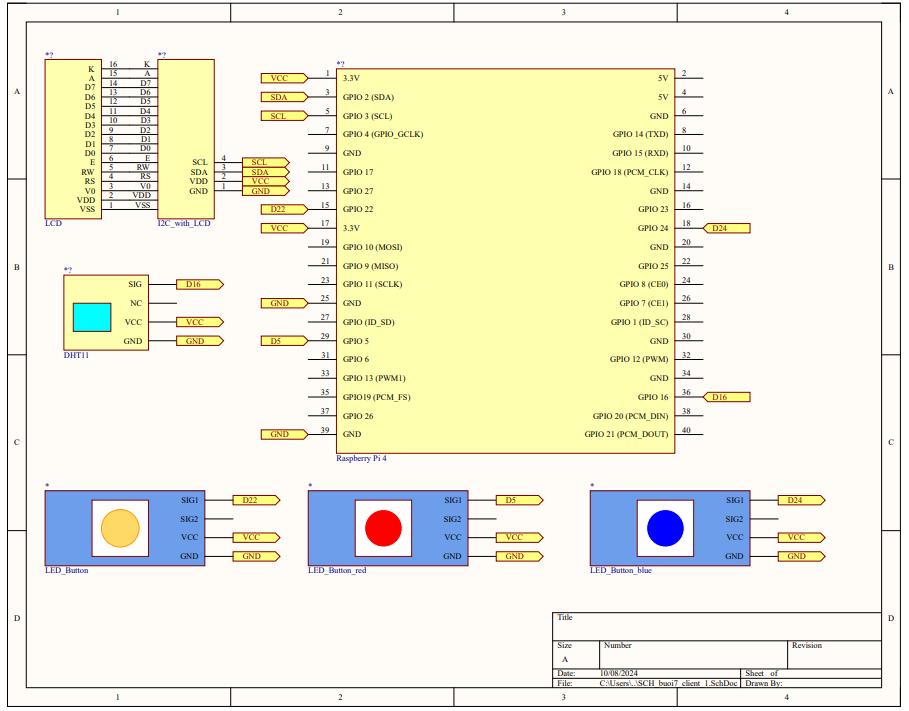
LED mới (độc lập và tùy biến) cho Raspberry X. Chương trình

trên Master phải hoạt động bình thường nếu 1 trong 2 hoặc

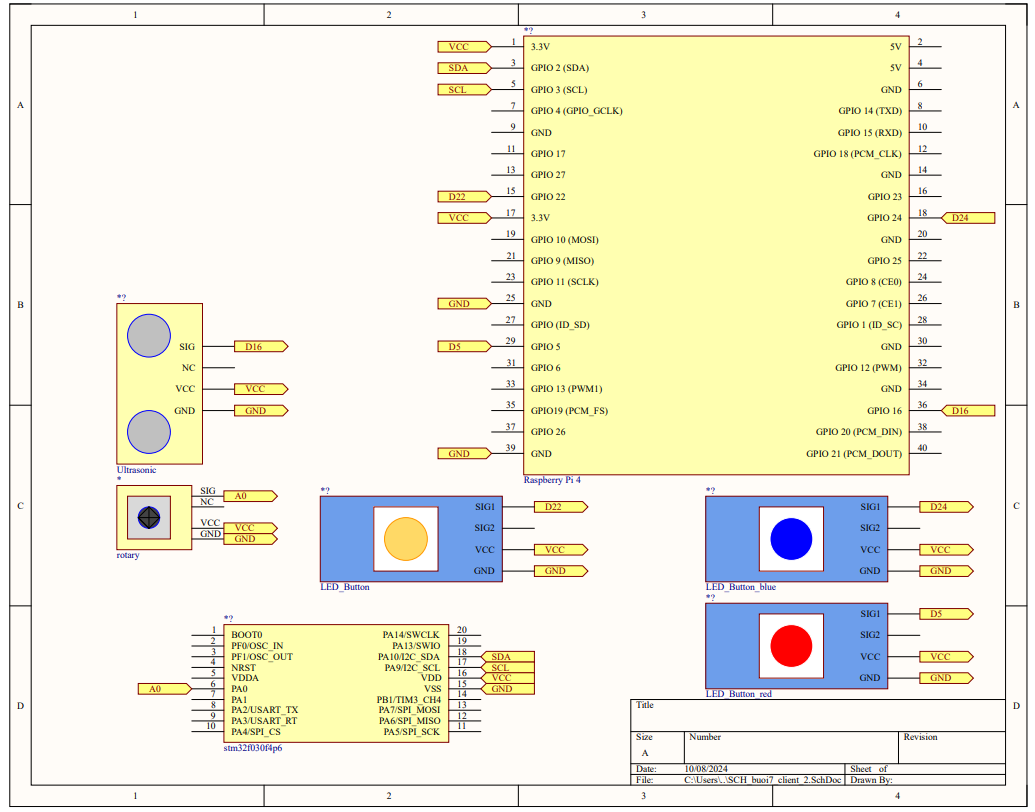
cả 2 Raspberry Slave và X không hoạt động

# **I.Yêu cầu 2: Sơ đồ nguyên lý.**

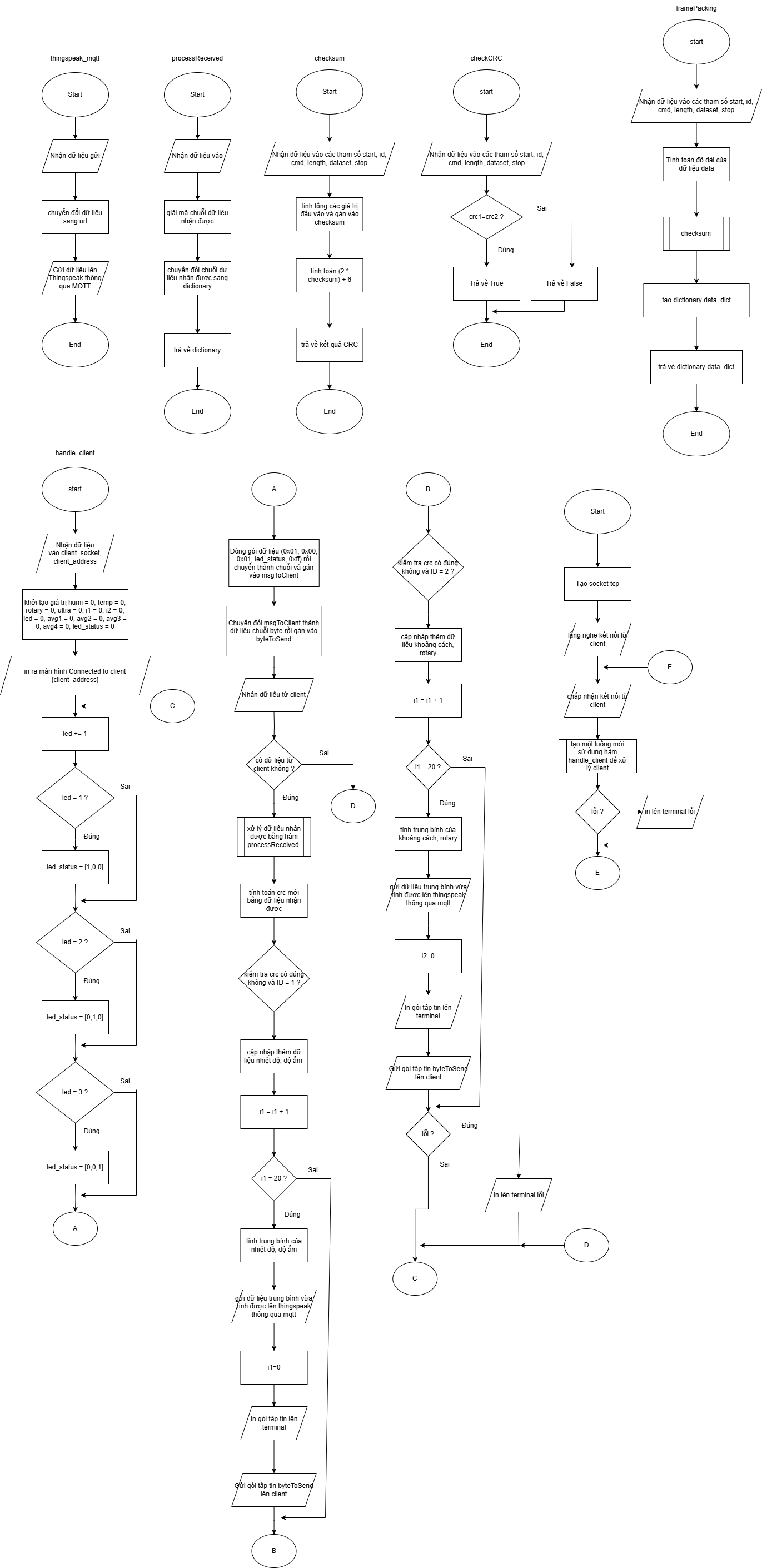
Slave 1:



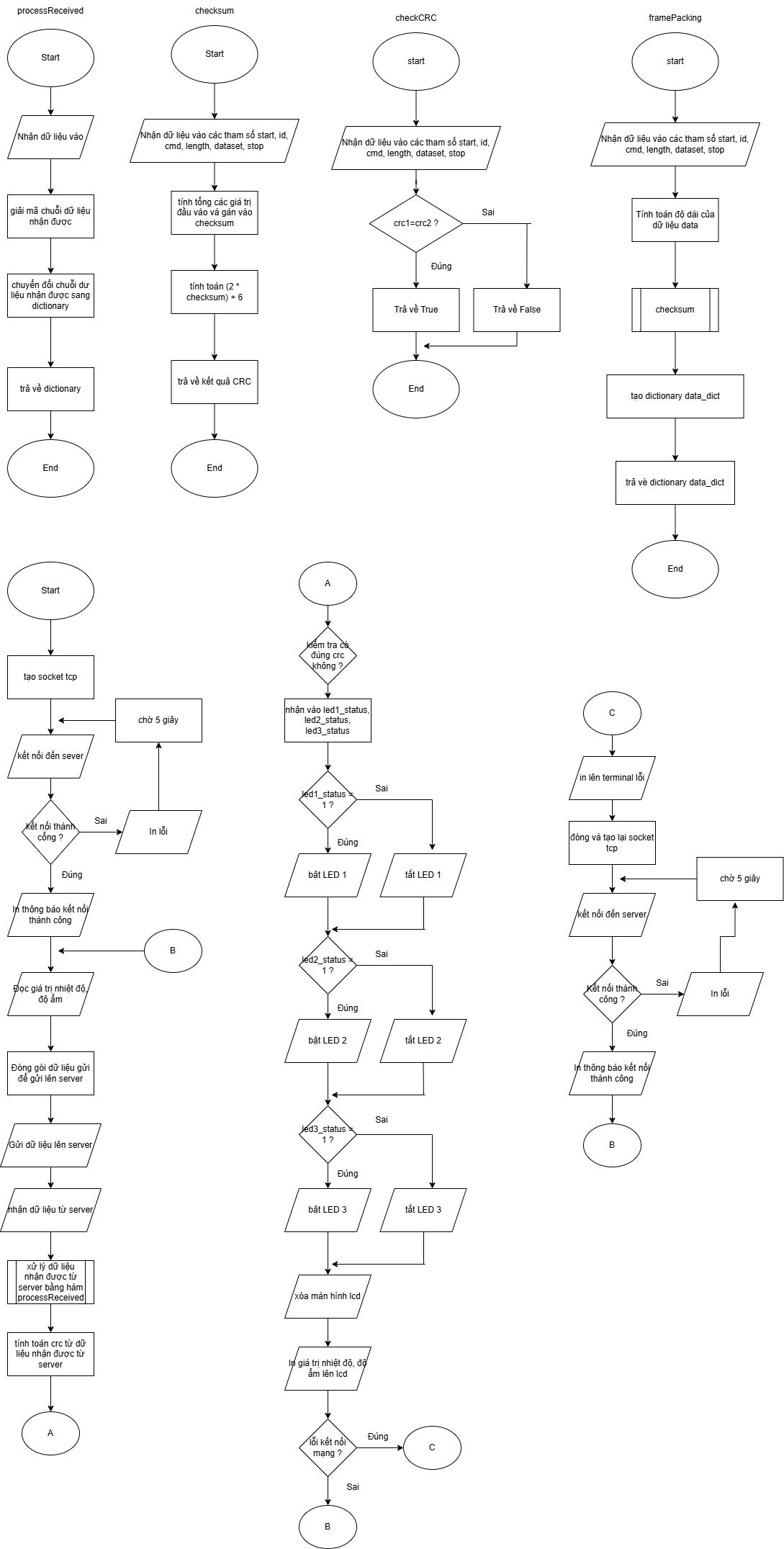
Slave 2:



# **II.Yêu cầu 3: Lưu đồ giải thuật.**

Master: ****

Slave 1:



Slave 2:

# 

# **III.Yêu cầu 4: Mã nguồn chương trình.**

Master:

import socket

import ast

import threading

import paho.mqtt.client as mqtt

from time import sleep

localIP = "0.0.0.0"

localPort = 11000

bufferSize = 1024

# Tạo socket TCP một lần duy nhất

TCP\_Server = socket.socket(family=socket.AF\_INET, type=socket.SOCK\_STREAM)

TCP\_Server.bind((localIP, localPort))

TCP\_Server.listen(5)  # Lắng nghe kết nối đến (cho phép tối đa 5 kết nối trong hàng đợi)

print("TCP server up and listening")

# MQTT

client = mqtt.Client("EhYPCgI1EDY7NgwMMAYPFxA")

client.username\_pw\_set(username="EhYPCgI1EDY7NgwMMAYPFxA",

                       password="TActyP2L+kNIoac/5j2y7FGZ")

client.connect("mqtt3.thingspeak.com", 1883, 60)

def thingspeak\_mqtt(data5, data6, field\_1, field\_2):

    channel\_ID = "2648891"

    payload = "field%s=%s&field%s=%s&status=MQTTPUBLISH" % (field\_1, data5, field\_2, data6)

    client.publish("channels/%s/publish" % channel\_ID, payload)

def processReceived(dataReceived):

    input\_string = dataReceived.decode("utf-8")

    data\_dict = ast.literal\_eval(input\_string)

    return data\_dict

def checksum(start, id, cmd, length, datasets, stop):

    checksum = 0

    checksum += (start + id + cmd + length + sum(data for data in datasets) + stop)

    return (2 \* checksum) + 6

def framePacking(start, id, cmd, data, stop):

    data\_dict = {

        "Start": start,

        "ID": id,

        "CMD": cmd,

        "Length": len(data),

        "Data": data,

        "CRC": checksum(start, id, cmd, len(data), data, stop),

        "Stop": stop

    }

    return data\_dict

def checkCRC(crc1, crc2):

    return crc1 == crc2

# Trạng thái của 3 đèn LED

led\_status = [1, 1, 1]

def handle\_client(client\_socket, client\_address):

    humi = 0

    temp = 0

    rotary = 0

    ultra = 0

    i1 = 0

    i2 = 0

    led = 0

    avg1 = 0

    avg2 = 0

    avg3 = 0

    avg4 = 0

    # Khai báo led\_status bên trong hàm handle\_client để đảm bảo mỗi luồng có một biến led\_status riêng

    led\_status = 0

    print(f"Connected to client {client\_address}")

    try:

        while True:

            led += 1

            if led == 1:

                led\_status = [1, 0, 0]

            elif led == 2:

                led\_status = [0, 1, 0]

            elif led == 3:

                led\_status = [0, 0, 1]

                led = 0

            msgToClient = str(framePacking(0x01, 0x00, 0x01, led\_status, 0xff))

            bytesToSend = str.encode(msgToClient)

            # Nhận dữ liệu từ client

            ClientMsg\_mgs = client\_socket.recv(bufferSize)

            if not ClientMsg\_mgs:

                break  # Client đã ngắt kết nối

            data\_dict = processReceived(ClientMsg\_mgs)

            msg = "Message from Client: {}".format(data\_dict)

            crc2 = checksum(data\_dict['Start'], data\_dict['ID'], data\_dict['CMD'], len(data\_dict['Data']),

                            data\_dict['Data'], data\_dict['Stop'])

            if checkCRC(data\_dict['CRC'], crc2) and (data\_dict['ID'] == 1):

                adr = "Client IP + Port: {}".format(client\_address)

                humi += data\_dict['Data'][0]

                temp += data\_dict['Data'][1]

                i1 += 1

                if i1 == 20:

                    avg1 = humi / 20

                    avg2 = temp / 20

                    thingspeak\_mqtt(avg1, avg2, 1, 2)

                    print("SEND DATA TO THINGSPEAK ---- HUMI & TEMP")

                    i1 = 0

                    avg1 = 0

                    avg2 = 0

                print("Client\_1 :")

                print(data\_dict)

                print(adr)

                print(i1)

                client\_socket.send(bytesToSend)

                print("-----------------------------------------------")

            if checkCRC(data\_dict['CRC'], crc2) and (data\_dict['ID'] == 2):

                adr = "Client IP + Port: {}".format(client\_address)

                ultra += data\_dict['Data'][0]

                rotary += data\_dict['Data'][1]

                i2 += 1

                if i2 == 20:

                    avg3 = ultra / 20

                    avg4 = rotary / 20

                    thingspeak\_mqtt(avg3, avg4, 3, 4)

                    print("SEND DATA TO THINGSPEAK ---- ULTRA & ROTARY")

                    avg3 = 0

                    avg4 = 0

                    i2 = 0

                print("Client\_2 :")

                print(data\_dict)

                print(adr)

                print(i2)

                client\_socket.send(bytesToSend)

                print("-----------------------------------------------")

    except Exception as e:

        print(f"Lỗi: {e}")

    finally:

        client\_socket.close()

        print("Client disconnected")

while True:

    try:

        # Chấp nhận kết nối từ client

        print("Waiting for a client to connect...")

        client\_socket, client\_address = TCP\_Server.accept()

        # Tạo luồng mới để xử lý client

        client\_thread = threading.Thread(target=handle\_client, args=(client\_socket, client\_address))

        client\_thread.start()

    except Exception as e:

        print(f"Lỗi: {e}")

    sleep(1)

Slave 1:

import socket

from time import sleep

from gpiozero import LED

from seeed\_dht import DHT

from grove.display.jhd1802 import JHD1802

import ast

# Khai báo 3 LED trên các chân GPIO

led1 = LED(5)  # LED 1 ở chân GPIO 5

led2 = LED(22)  # LED 2 ở chân GPIO 22

led3 = LED(24)  # LED 3 ở chân GPIO 24

lcd = JHD1802()

sensor = DHT("11", 16)

ServerAddressPort = ("192.168.187.89", 11000)

bufferSize = 1024

i = -1

# Tạo socket TCP một lần duy nhất

TCP\_Client = socket.socket(family=socket.AF\_INET, type=socket.SOCK\_STREAM)

def processReceived(dataReceived):

    input\_string = dataReceived.decode("utf-8")

    data\_dict = ast.literal\_eval(input\_string)

    return data\_dict

def checksum(start, id, cmd, length, datasets, stop):

    checksum = 0

    checksum += (start + id + cmd + length + sum(data for data in datasets) + stop)

    return (2 \* checksum) + 6

def framePacking(start, id, cmd, data, stop):

    data\_dict = {

        "Start": start,

        "ID": id,

        "CMD": cmd,

        "Length": len(data),

        "Data": data,

        "CRC": checksum(start, id, cmd, len(data), data, stop),

        "Stop": stop

    }

    return data\_dict

def checkCRC(crc1, crc2):

    return crc1 == crc2

# Kết nối TCP đến server

while True:

    try:

        TCP\_Client.connect(ServerAddressPort)

        print("Connected to server")

        break

    except Exception as e:

        print(f"Cannot connect to server, retrying...: {e}")

        sleep(5)

while True:

    try:

        i += 1

        humi, temp = sensor.read()

        print("Nhiet do = ", temp)

        print("Do Am = ", humi)

        # Đóng gói dữ liệu gửi lên server (nhiệt độ và độ ẩm)

        msgToServer = str(framePacking(0x01, 0x01, 0x04, [humi, temp], 0xff))

        bytesToSend = str.encode(msgToServer)

        # Gửi dữ liệu lên server qua TCP

        TCP\_Client.send(bytesToSend)

        print("TCP Client connected and sending")

        # Nhận phản hồi từ server

        ServerMsg\_msg = TCP\_Client.recv(bufferSize)

        msg = "Message from Server: {}".format(ServerMsg\_msg)

        # Xử lý dữ liệu nhận được từ server

        data = processReceived(ServerMsg\_msg)

        # Tính toán và kiểm tra checksum (CRC)

        crc2 = checksum(data['Start'], data['ID'], data['CMD'], len(data['Data']), data['Data'], data['Stop'])

        print("crc2 = ", crc2)

        if checkCRC(data['CRC'], crc2):

            print(data)

            led1\_status = data['Data'][0]  # Trạng thái của LED 1

            led2\_status = data['Data'][1]  # Trạng thái của LED 2

            led3\_status = data['Data'][2]  # Trạng thái của LED 3

            # Bật/tắt LED 1

            if led1\_status == 1:

                led1.on()

            else:

                led1.off()

            # Bật/tắt LED 2

            if led2\_status == 1:

                led2.on()

            else:

                led2.off()

            # Bật/tắt LED 3

            if led3\_status == 1:

                led3.on()

            else:

                led3.off()

        # Hiển thị dữ liệu lên LCD

        lcd.clear()

        lcd.setCursor(0, 1)

        lcd.write(' ND:{0:2}C'.format(temp))

        lcd.setCursor(0, 8)

        lcd.write('DA:{0:2}%'.format(humi))

        print(msg)

        print(i)

    except (socket.error, Exception) as e:

        print(f"Lỗi: {e}")

        print("Retrying to connect to server...")

        # Đóng socket hiện tại và tạo lại

        TCP\_Client.close()

        TCP\_Client = socket.socket(family=socket.AF\_INET, type=socket.SOCK\_STREAM)

        # Kết nối lại tới server

        while True:

            try:

                TCP\_Client.connect(ServerAddressPort)

                print("Reconnected to server")

                break

            except Exception as e:

                print(f"Cannot reconnect to server, retrying...: {e}")

                sleep(5)

    print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")

    sleep(1)

# Đóng kết nối TCP sau khi hoàn tất

TCP\_Client.close()

Slave 2:

import socket

from time import sleep

from gpiozero import LED

from grove.grove\_ultrasonic\_ranger import GroveUltrasonicRanger

from grove.adc import ADC

import ast  # Thêm import ast

SENSOR = ADC()

sensor = GroveUltrasonicRanger(16)

# Khai báo 3 LED trên các chân GPIO

led1 = LED(5)  # LED 1 ở chân GPIO 5

led2 = LED(22)  # LED 2 ở chân GPIO 22

led3 = LED(24)  # LED 3 ở chân GPIO 24

ServerAddressPort = ("192.168.187.89", 11000)

bufferSize = 1024

i = -1

# Tạo socket TCP một lần duy nhất

TCP\_Client = socket.socket(family=socket.AF\_INET, type=socket.SOCK\_STREAM)

def processReceived(dataReceived):

    input\_string = dataReceived.decode("utf-8")

    data\_dict = ast.literal\_eval(input\_string)

    return data\_dict

def checksum(start, id, cmd, length, datasets, stop):

    checksum = 0

    checksum += (start + id + cmd + length + sum(data for data in datasets) + stop)

    return (2 \* checksum) + 6

def framePacking(start, id, cmd, data, stop):

    data\_dict = {

        "Start": start,

        "ID": id,

        "CMD": cmd,

        "Length": len(data),

        "Data": data,

        "CRC": checksum(start, id, cmd, len(data), data, stop),

        "Stop": stop

    }

    return data\_dict

def checkCRC(crc1, crc2):

    return crc1 == crc2

# Kết nối TCP đến server

while True:

    try:

        TCP\_Client.connect(ServerAddressPort)

        print("Connected to server")

        break

    except Exception as e:

        print(f"Cannot connect to server, retrying...: {e}")

        sleep(5)

while True:

    try:

        i += 1

        distance = round(sensor.get\_distance(), 1)

        rotary = SENSOR.read\_voltage(0)

        print("Khoang Cach = ", distance)

        print("Bien Tro = ", rotary)

        # Đóng gói dữ liệu gửi lên server (khoảng cách và biến trở)

        msgToServer = str(framePacking(0x01, 0x02, 0x03, [distance, rotary], 0xff))

        bytesToSend = str.encode(msgToServer)

        # Gửi dữ liệu lên server qua TCP

        TCP\_Client.send(bytesToSend)

        print("TCP Client connected and sending")

        # Nhận phản hồi từ server

        ServerMsg\_msg = TCP\_Client.recv(bufferSize)

        msg = "Message from Server: {}".format(ServerMsg\_msg)

        # Xử lý dữ liệu nhận được từ server

        data = processReceived(ServerMsg\_msg)

        # Tính toán và kiểm tra checksum (CRC)

        crc2 = checksum(data['Start'], data['ID'], data['CMD'], len(data['Data']), data['Data'], data['Stop'])

        print("crc2 = ", crc2)

        if checkCRC(data['CRC'], crc2):

            print(data)

            led1\_status = data['Data'][0]  # Trạng thái của LED 1

            led2\_status = data['Data'][1]  # Trạng thái của LED 2

            led3\_status = data['Data'][2]  # Trạng thái của LED 3

            # Bật/tắt LED 1

            if led1\_status == 1:

                led1.on()

            else:

                led1.off()

            # Bật/tắt LED 2

            if led2\_status == 1:

                led2.on()

            else:

                led2.off()

            # Bật/tắt LED 3

            if led3\_status == 1:

                led3.on()

            else:

                led3.off()

        print(msg)

        print(i)

    except (socket.error, Exception) as e:

        print(f"Lỗi: {e}")

        print("Retrying to connect to server...")

        # Đóng socket hiện tại và tạo lại

        TCP\_Client.close()

        TCP\_Client = socket.socket(family=socket.AF\_INET, type=socket.SOCK\_STREAM)

        # Kết nối lại tới server

        while True:

            try:

                TCP\_Client.connect(ServerAddressPort)

                print("Reconnected to server")

                break

            except Exception as e:

                print(f"Cannot reconnect to server, retrying...: {e}")

                sleep(5)

    print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")

    sleep(1)

# Đóng kết nối TCP sau khi hoàn tất

TCP\_Client.close()

# **IV.Yêu cầu 5: Giải thích mã nguồn.**

Master:

1. Các thư viện

import socket

import ast

import threading

import paho.mqtt.client as mqtt

from time import sleep

socket: Thư viện để tạo và quản lý các kết nối mạng.

ast: Thư viện để phân tích cú pháp (parsing) chuỗi Python thành đối tượng Python.

threading: Thư viện để quản lý đa luồng (multi-threading).

paho.mqtt.client: Thư viện để kết nối và giao tiếp với các broker MQTT.

time: Thư viện để sử dụng hàm sleep, cho phép dừng chương trình trong một khoảng thời gian nhất định.

2. Thiết lập địa chỉ IP và cổng cho server TCP

localIP = "0.0.0.0"

localPort = 11000

bufferSize = 1024

localIP: Địa chỉ IP của server, sử dụng "0.0.0.0" để lắng nghe tất cả các địa chỉ IP.

localPort: Cổng mà server sẽ lắng nghe (11000).

bufferSize: Kích thước bộ đệm cho dữ liệu nhận từ client.

3. Tạo socket TCP

TCP\_Server = socket.socket(family=socket.AF\_INET, type=socket.SOCK\_STREAM)

TCP\_Server.bind((localIP, localPort))

TCP\_Server.listen(5)

socket.socket: Tạo một socket TCP.

bind: Gán địa chỉ IP và cổng cho socket.

listen: Bắt đầu lắng nghe các kết nối đến (tối đa 5 kết nối trong hàng đợi).

4. Thiết lập kết nối MQTT

client = mqtt.Client("EhYPCgI1EDY7NgwMMAYPFxA")

client.username\_pw\_set(username="EhYPCgI1EDY7NgwMMAYPFxA",

                       password="TActyP2L+kNIoac/5j2y7FGZ")

client.connect("mqtt3.thingspeak.com", 1883, 60)

mqtt.Client: Tạo một đối tượng client MQTT.

username\_pw\_set: Đặt tên người dùng và mật khẩu cho client.

connect: Kết nối đến broker MQTT.

5. Các hàm

a.Hàm thingspeak\_mqtt

def thingspeak\_mqtt(data5, data6, field\_1, field\_2):

    channel\_ID = "2648891"

    payload = "field%s=%s&field%s=%s&status=MQTTPUBLISH" % (field\_1, data5, field\_2, data6)

    client.publish("channels/%s/publish" % channel\_ID, payload)

Gửi dữ liệu đến ThingSpeak thông qua MQTT.

Tạo payload với dữ liệu cần gửi và sau đó gọi phương thức publish của client MQTT.

b. Hàm processReceived

def processReceived(dataReceived):

    input\_string = dataReceived.decode("utf-8")

    data\_dict = ast.literal\_eval(input\_string)

    return data\_dict

Chuyển đổi dữ liệu nhận được từ byte sang chuỗi và sau đó thành một dictionary.

c. Hàm checksum

def checksum(start, id, cmd, length, datasets, stop):

    checksum = 0

    checksum += (start + id + cmd + length + sum(data for data in datasets) + stop)

    return (2 \* checksum) + 6

Tính toán giá trị checksum (CRC) cho dữ liệu để kiểm tra tính toàn vẹn của dữ liệu.

d. Hàm framePacking

def framePacking(start, id, cmd, data, stop):

    data\_dict = {

        "Start": start,

        "ID": id,

        "CMD": cmd,

        "Length": len(data),

        "Data": data,

        "CRC": checksum(start, id, cmd, len(data), data, stop),

        "Stop": stop

    }

    return data\_dict

Đóng gói dữ liệu vào một dictionary với các trường như Start, ID, CMD, Length, Data, CRC, và Stop.

e. Hàm checkCRC

def checkCRC(crc1, crc2):

    return crc1 == crc2

Kiểm tra xem giá trị CRC nhận được có khớp với giá trị CRC tính toán hay không.

6. Trạng thái của đèn LED

led\_status = [1, 1, 1]

Biến này giữ trạng thái của ba đèn LED.

7. Hàm xử lý client

def handle\_client(client\_socket, client\_address):

    humi = 0

    temp = 0

    rotary = 0

    ultra = 0

    i1 = 0

    i2 = 0

    led = 0

    avg1 = 0

    avg2 = 0

    avg3 = 0

    avg4 = 0

    # Khai báo led\_status bên trong hàm handle\_client để đảm bảo mỗi luồng có một biến led\_status riêng

    led\_status = 0

    print(f"Connected to client {client\_address}")

    try:

        while True:

            led += 1

            if led == 1:

                led\_status = [1, 0, 0]

            elif led == 2:

                led\_status = [0, 1, 0]

            elif led == 3:

                led\_status = [0, 0, 1]

                led = 0

            msgToClient = str(framePacking(0x01, 0x00, 0x01, led\_status, 0xff))

            bytesToSend = str.encode(msgToClient)

            # Nhận dữ liệu từ client

            ClientMsg\_mgs = client\_socket.recv(bufferSize)

            if not ClientMsg\_mgs:

                break  # Client đã ngắt kết nối

            data\_dict = processReceived(ClientMsg\_mgs)

            msg = "Message from Client: {}".format(data\_dict)

            crc2 = checksum(data\_dict['Start'], data\_dict['ID'], data\_dict['CMD'], len(data\_dict['Data']),

                            data\_dict['Data'], data\_dict['Stop'])

            if checkCRC(data\_dict['CRC'], crc2) and (data\_dict['ID'] == 1):

                adr = "Client IP + Port: {}".format(client\_address)

                humi += data\_dict['Data'][0]

                temp += data\_dict['Data'][1]

                i1 += 1

                if i1 == 20:

                    avg1 = humi / 20

                    avg2 = temp / 20

                    thingspeak\_mqtt(avg1, avg2, 1, 2)

                    print("SEND DATA TO THINGSPEAK ---- HUMI & TEMP")

                    i1 = 0

                    avg1 = 0

                    avg2 = 0

                print("Client\_1 :")

                print(data\_dict)

                print(adr)

                print(i1)

                client\_socket.send(bytesToSend)

                print("-----------------------------------------------")

            if checkCRC(data\_dict['CRC'], crc2) and (data\_dict['ID'] == 2):

                adr = "Client IP + Port: {}".format(client\_address)

                ultra += data\_dict['Data'][0]

                rotary += data\_dict['Data'][1]

                i2 += 1

                if i2 == 20:

                    avg3 = ultra / 20

                    avg4 = rotary / 20

                    thingspeak\_mqtt(avg3, avg4, 3, 4)

                    print("SEND DATA TO THINGSPEAK ---- ULTRA & ROTARY")

                    avg3 = 0

                    avg4 = 0

                    i2 = 0

                print("Client\_2 :")

                print(data\_dict)

                print(adr)

                print(i2)

                client\_socket.send(bytesToSend)

                print("-----------------------------------------------")

    except Exception as e:

        print(f"Lỗi: {e}")

    finally:

        client\_socket.close()

        print("Client disconnected")

handle\_client: Hàm chính xử lý mỗi kết nối client.

Bên trong hàm, nó liên tục nhận dữ liệu từ client, kiểm tra tính hợp lệ của dữ liệu, và xử lý dữ liệu theo ID. Nếu ID là 1, nó xử lý thông tin độ ẩm và nhiệt độ. Nếu ID là 2, nó xử lý thông tin khoảng cách siêu âm và dữ liệu rotary.

Khi dữ liệu hợp lệ được nhận, nó sẽ gửi lại thông tin trạng thái LED đến client.

8. Vòng lặp chính

while True:

    try:

        # Chấp nhận kết nối từ client

        print("Waiting for a client to connect...")

        client\_socket, client\_address = TCP\_Server.accept()

        # Tạo luồng mới để xử lý client

        client\_thread = threading.Thread(target=handle\_client, args=(client\_socket, client\_address))

        client\_thread.start()

    except Exception as e:

        print(f"Lỗi: {e}")

    sleep(1)

Vòng lặp này chờ các kết nối từ client. Khi một client kết nối, nó tạo một luồng mới để xử lý client đó bằng hàm handle\_client và tiếp tục chờ cho các kết nối tiếp theo.

Slave 1:

1. Các thư viện

import socket

from time import sleep

from gpiozero import LED

from seeed\_dht import DHT

from grove.display.jhd1802 import JHD1802

import ast

socket: Thư viện để tạo và quản lý kết nối mạng TCP.

time: Thư viện để sử dụng hàm sleep, cho phép dừng chương trình trong một khoảng thời gian nhất định.

gpiozero: Thư viện để điều khiển các thiết bị GPIO như đèn LED.

seeed\_dht: Thư viện để đọc dữ liệu từ cảm biến DHT (nhiệt độ và độ ẩm).

grove.display.jhd1802: Thư viện để điều khiển màn hình LCD JHD1802.

ast: Thư viện để chuyển đổi chuỗi thành đối tượng Python.

2. Khai báo các thiết bị và cấu hình

led1 = LED(5)

led2 = LED(22)

led3 = LED(24)

lcd = JHD1802()

sensor = DHT("11", 16)

ServerAddressPort = ("192.168.187.89", 11000)

bufferSize = 1024

i = -1

led1, led2, led3: Khởi tạo ba đối tượng LED tương ứng với các chân GPIO.

lcd: Khởi tạo màn hình LCD.

sensor: Khởi tạo cảm biến DHT (được chỉ định loại DHT11 ở chân GPIO 16).

ServerAddressPort: Địa chỉ IP và cổng của server mà client sẽ kết nối.

bufferSize: Kích thước bộ đệm cho dữ liệu nhận từ server.

i: Biến để đếm số lần vòng lặp.

3. Tạo socket TCP

TCP\_Client = socket.socket(family=socket.AF\_INET, type=socket.SOCK\_STREAM)

socket.socket: Tạo một socket TCP để giao tiếp với server.

4. Các hàm

a. Hàm processReceived

def processReceived(dataReceived):

    input\_string = dataReceived.decode("utf-8")

    data\_dict = ast.literal\_eval(input\_string)

    return data\_dict

Chuyển đổi dữ liệu nhận được từ byte sang chuỗi và sau đó thành một dictionary.

b. Hàm checksum

def checksum(start, id, cmd, length, datasets, stop):

    checksum = 0

    checksum += (start + id + cmd + length + sum(data for data in datasets) + stop)

    return (2 \* checksum) + 6

Tính toán giá trị checksum (CRC) cho dữ liệu để kiểm tra tính toàn vẹn của dữ liệu.

c. Hàm framePacking

def framePacking(start, id, cmd, data, stop):

    data\_dict = {

        "Start": start,

        "ID": id,

        "CMD": cmd,

        "Length": len(data),

        "Data": data,

        "CRC": checksum(start, id, cmd, len(data), data, stop),

        "Stop": stop

    }

    return data\_dict

Đóng gói dữ liệu vào một dictionary với các trường như Start, ID, CMD, Length, Data, CRC, và Stop.

d. Hàm checkCRC

def checkCRC(crc1, crc2):

    return crc1 == crc2

Kiểm tra xem giá trị CRC nhận được có khớp với giá trị CRC tính toán hay không.

5. Kết nối TCP đến server

while True:

    try:

        TCP\_Client.connect(ServerAddressPort)

        print("Connected to server")

        break

    except Exception as e:

        print(f"Cannot connect to server, retrying...: {e}")

        sleep(5)

Vòng lặp này cố gắng kết nối đến server. Nếu kết nối không thành công, nó sẽ in thông báo lỗi và thử lại sau 5 giây. Khi kết nối thành công, nó sẽ in ra thông báo và thoát khỏi vòng lặp.

6. Vòng lặp chính để gửi và nhận dữ liệu

while True:

    try:

        i += 1

        humi, temp = sensor.read()

        print("Nhiet do = ", temp)

        print("Do Am = ", humi)

        # Đóng gói dữ liệu gửi lên server (nhiệt độ và độ ẩm)

        msgToServer = str(framePacking(0x01, 0x01, 0x04, [humi, temp], 0xff))

        bytesToSend = str.encode(msgToServer)

        # Gửi dữ liệu lên server qua TCP

        TCP\_Client.send(bytesToSend)

        print("TCP Client connected and sending")

        # Nhận phản hồi từ server

        ServerMsg\_msg = TCP\_Client.recv(bufferSize)

        msg = "Message from Server: {}".format(ServerMsg\_msg)

        # Xử lý dữ liệu nhận được từ server

        data = processReceived(ServerMsg\_msg)

        # Tính toán và kiểm tra checksum (CRC)

        crc2 = checksum(data['Start'], data['ID'], data['CMD'], len(data['Data']), data['Data'], data['Stop'])

        print("crc2 = ", crc2)

        if checkCRC(data['CRC'], crc2):

            print(data)

            led1\_status = data['Data'][0]  # Trạng thái của LED 1

            led2\_status = data['Data'][1]  # Trạng thái của LED 2

            led3\_status = data['Data'][2]  # Trạng thái của LED 3

            # Bật/tắt LED 1

            if led1\_status == 1:

                led1.on()

            else:

                led1.off()

            # Bật/tắt LED 2

            if led2\_status == 1:

                led2.on()

            else:

                led2.off()

            # Bật/tắt LED 3

            if led3\_status == 1:

                led3.on()

            else:

                led3.off()

        # Hiển thị dữ liệu lên LCD

        lcd.clear()

        lcd.setCursor(0, 1)

        lcd.write(' ND:{0:2}C'.format(temp))

        lcd.setCursor(0, 8)

        lcd.write('DA:{0:2}%'.format(humi))

        print(msg)

        print(i)

    except (socket.error, Exception) as e:

        print(f"Lỗi: {e}")

        print("Retrying to connect to server...")

        # Đóng socket hiện tại và tạo lại

        TCP\_Client.close()

        TCP\_Client = socket.socket(family=socket.AF\_INET, type=socket.SOCK\_STREAM)

        # Kết nối lại tới server

        while True:

            try:

                TCP\_Client.connect(ServerAddressPort)

                print("Reconnected to server")

                break

            except Exception as e:

                print(f"Cannot reconnect to server, retrying...: {e}")

                sleep(5)

    print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")

    sleep(1)

Vòng lặp chính: Mỗi vòng lặp sẽ:

Tăng giá trị của i để đếm số vòng lặp.

Đọc dữ liệu nhiệt độ và độ ẩm từ cảm biến DHT.

In ra nhiệt độ và độ ẩm để kiểm tra.

Đóng gói dữ liệu (độ ẩm và nhiệt độ) để gửi lên server.

Gửi dữ liệu đến server qua kết nối TCP.

Nhận phản hồi từ server.

Xử lý dữ liệu nhận được từ server (phân tích checksum và kiểm tra).

Nếu checksum hợp lệ, nó sẽ bật/tắt các LED theo trạng thái được chỉ định trong dữ liệu từ server.

Hiển thị nhiệt độ và độ ẩm lên màn hình LCD.

Nếu có lỗi trong kết nối hoặc nhận dữ liệu, chương trình sẽ in thông báo lỗi, đóng socket hiện tại, và cố gắng kết nối lại với server.

Slave 2:

1. Các thư viện

import socket

from time import sleep

from gpiozero import LED

from grove.grove\_ultrasonic\_ranger import GroveUltrasonicRanger

from grove.adc import ADC

import ast

socket: Thư viện để tạo và quản lý kết nối mạng TCP.

time: Sử dụng hàm sleep để dừng chương trình trong một khoảng thời gian nhất định.

gpiozero: Thư viện điều khiển các thiết bị GPIO như đèn LED.

grove.grove\_ultrasonic\_ranger: Để sử dụng cảm biến siêu âm đo khoảng cách.

grove.adc: Thư viện để đọc giá trị từ cảm biến biến trở (Analog-to-Digital Converter - ADC).

ast: Thư viện để phân tích cú pháp chuỗi thành các đối tượng Python.

2. Khai báo các thiết bị và cấu hình

SENSOR = ADC()

sensor = GroveUltrasonicRanger(16)

led1 = LED(5)

led2 = LED(22)

led3 = LED(24)

ServerAddressPort = ("192.168.187.89", 11000)

bufferSize = 1024

i = -1

SENSOR: Biến để tương tác với module ADC, giúp đọc giá trị từ cảm biến biến trở.

sensor: Biến khởi tạo cảm biến siêu âm, gắn vào chân GPIO 16.

led1, led2, led3: Khai báo 3 đèn LED tương ứng với các chân GPIO.

ServerAddressPort: Địa chỉ IP và cổng của server mà client sẽ kết nối.

bufferSize: Kích thước bộ đệm để nhận dữ liệu từ server.

i: Biến đếm số vòng lặp.

3. Tạo socket TCP

TCP\_Client = socket.socket(family=socket.AF\_INET, type=socket.SOCK\_STREAM)

socket.socket: Tạo một socket TCP để giao tiếp với server.

4. Các hàm

a. Hàm processReceived

def processReceived(dataReceived):

    input\_string = dataReceived.decode("utf-8")

    data\_dict = ast.literal\_eval(input\_string)

    return data\_dict

Chuyển đổi dữ liệu nhận được từ byte sang chuỗi và sau đó thành một dictionary.

b. Hàm checksum

def checksum(start, id, cmd, length, datasets, stop):

    checksum = 0

    checksum += (start + id + cmd + length + sum(data for data in datasets) + stop)

    return (2 \* checksum) + 6

Tính toán giá trị checksum (CRC) cho dữ liệu để kiểm tra tính toàn vẹn của dữ liệu.

c. Hàm framePacking

def framePacking(start, id, cmd, data, stop):

    data\_dict = {

        "Start": start,

        "ID": id,

        "CMD": cmd,

        "Length": len(data),

        "Data": data,

        "CRC": checksum(start, id, cmd, len(data), data, stop),

        "Stop": stop

    }

    return data\_dict

Đóng gói dữ liệu vào một dictionary với các trường như Start, ID, CMD, Length, Data, CRC, và Stop.

d. Hàm checkCRC

def checkCRC(crc1, crc2):

    return crc1 == crc2

Kiểm tra xem giá trị CRC nhận được có khớp với giá trị CRC tính toán hay không.

5. Kết nối TCP đến server

while True:

    try:

        TCP\_Client.connect(ServerAddressPort)

        print("Connected to server")

        break

    except Exception as e:

        print(f"Cannot connect to server, retrying...: {e}")

        sleep(5)

Vòng lặp này cố gắng kết nối đến server. Nếu kết nối không thành công, nó sẽ in thông báo lỗi và thử lại sau 5 giây. Khi kết nối thành công, nó sẽ in ra thông báo và thoát khỏi vòng lặp.

6. Vòng lặp chính để gửi và nhận dữ liệu

while True:

while True:

    try:

        i += 1

        distance = round(sensor.get\_distance(), 1)

        rotary = SENSOR.read\_voltage(0)

        print("Khoang Cach = ", distance)

        print("Bien Tro = ", rotary)

        # Đóng gói dữ liệu gửi lên server (khoảng cách và biến trở)

        msgToServer = str(framePacking(0x01, 0x02, 0x03, [distance, rotary], 0xff))

        bytesToSend = str.encode(msgToServer)

        # Gửi dữ liệu lên server qua TCP

        TCP\_Client.send(bytesToSend)

        print("TCP Client connected and sending")

        # Nhận phản hồi từ server

        ServerMsg\_msg = TCP\_Client.recv(bufferSize)

        msg = "Message from Server: {}".format(ServerMsg\_msg)

        # Xử lý dữ liệu nhận được từ server

        data = processReceived(ServerMsg\_msg)

        # Tính toán và kiểm tra checksum (CRC)

        crc2 = checksum(data['Start'], data['ID'], data['CMD'], len(data['Data']), data['Data'], data['Stop'])

        print("crc2 = ", crc2)

        if checkCRC(data['CRC'], crc2):

            print(data)

            led1\_status = data['Data'][0]  # Trạng thái của LED 1

            led2\_status = data['Data'][1]  # Trạng thái của LED 2

            led3\_status = data['Data'][2]  # Trạng thái của LED 3

            # Bật/tắt LED 1

            if led1\_status == 1:

                led1.on()

            else:

                led1.off()

            # Bật/tắt LED 2

            if led2\_status == 1:

                led2.on()

            else:

                led2.off()

            # Bật/tắt LED 3

            if led3\_status == 1:

                led3.on()

            else:

                led3.off()

        print(msg)

        print(i)

    except (socket.error, Exception) as e:

        print(f"Lỗi: {e}")

        print("Retrying to connect to server...")

        # Đóng socket hiện tại và tạo lại

        TCP\_Client.close()

        TCP\_Client = socket.socket(family=socket.AF\_INET, type=socket.SOCK\_STREAM)

        # Kết nối lại tới server

        while True:

            try:

                TCP\_Client.connect(ServerAddressPort)

                print("Reconnected to server")

                break

            except Exception as e:

                print(f"Cannot reconnect to server, retrying...: {e}")

                sleep(5)

    print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")

    sleep(1)

Vòng lặp chính: Mỗi vòng lặp sẽ:

Tăng giá trị của i để đếm số vòng lặp.

Đọc dữ liệu nhiệt độ và độ ẩm từ cảm biến DHT.

In ra nhiệt độ và độ ẩm để kiểm tra.

Đóng gói dữ liệu (khoảng cách và rotary) để gửi lên server.

Gửi dữ liệu đến server qua kết nối TCP.

Nhận phản hồi từ server.

Xử lý dữ liệu nhận được từ server (phân tích checksum và kiểm tra).

Nếu checksum hợp lệ, nó sẽ bật/tắt các LED theo trạng thái được chỉ định trong dữ liệu từ server.

Nếu có lỗi trong kết nối hoặc nhận dữ liệu, chương trình sẽ in thông báo lỗi, đóng socket hiện tại, và cố gắng kết nối lại với server.

Giải thích frame truyền:

Frame gửi dữ liệu nhiệt độ và độ ẩm

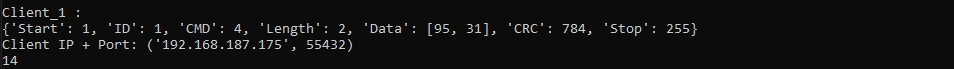
Ví dụ: Nhiệt độ 31 độ C và Độ ẩm 95%

Data[]:0x5F (95%) 0x1F (31C)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Start byte** | **ID byte** | **CMD byte** | **Length byte** | **Data[]** | **CRC byte** | **Stop byte** |
| 0x01 | 0x01 | 0x04 | 0x02 | 0x5F 0x1F | CRC | 0xFF |

CRC byte được tính dựa trên giá trị nhiệt độ và độ ẩm đã được đóng gói.

Terminal:



Frame gửi dữ liệu khoảng cách và biến trở.

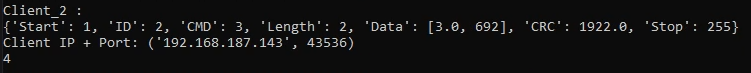
Ví dụ: Khoảng cách 3cm và biến trở là 692V

Data[]: 0x03 (3cm) 0x2B4 (692V)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Start byte** | **ID byte** | **CMD byte** | **Length byte** | **Data[]** | **CRC byte** | **Stop byte** |
| 0x01 | 0x02 | 0x03 | 0x02 | 0x03 0x2B4 | CRC | 0xFF |

CRC byte được tính dựa trên giá trị khoảng cách và biến trở đã được đóng gói.

Terminal:

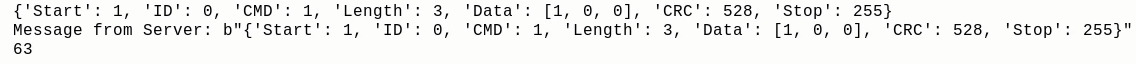


**Frame điều khiển LED 1 bật, LED 2 tắt, LED 3 tắt**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Start byte** | **ID byte** | **CMD byte** | **Length byte** | **Data[]** | **CRC byte** | **Stop byte** |
| 0x01 | 0x00 | 0x01 | 0x03 | 0x01 0x00 0x00 | CRC | 0xFF |

CRC byte được tính dựa trên giá trị LED 1, 2, 3 đã được đóng gói.

Terminal:



**Frame điều khiển LED 1 tắt, LED 2 bật, LED 3 tắt**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Start byte** | **ID byte** | **CMD byte** | **Length byte** | **Data[]** | **CRC byte** | **Stop byte** |
| 0x01 | 0x00 | 0x01 | 0x03 | 0x00 0x01 0x00 | CRC | 0xFF |

CRC byte được tính dựa trên giá trị LED 1, 2, 3 đã được đóng gói.

Terminal:



**Frame điều khiển LED 1 tắt, LED 2 tắt, LED 3 bật**

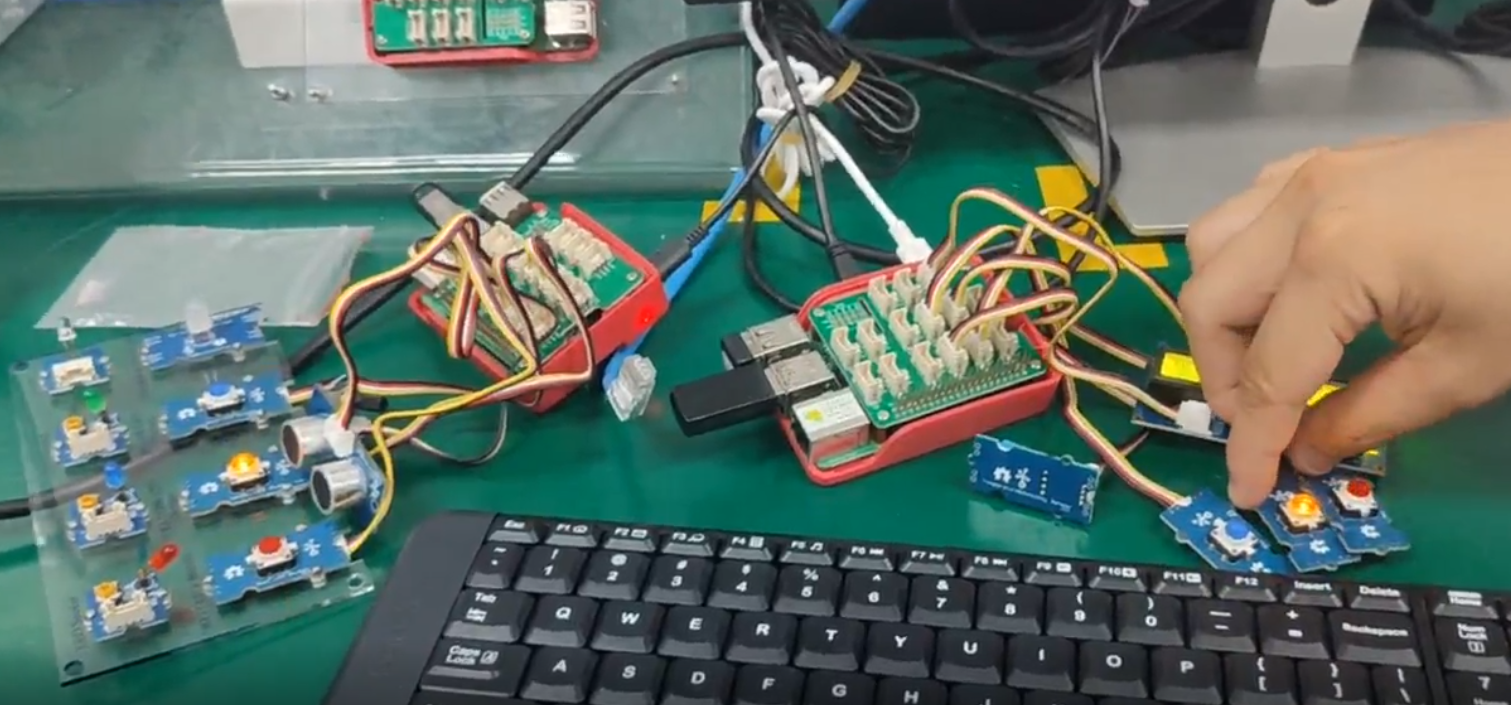
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Start byte** | **ID byte** | **CMD byte** | **Length byte** | **Data[]** | **CRC byte** | **Stop byte** |
| 0x01 | 0x00 | 0x01 | 0x03 | 0x00 0x00 0x01 | CRC | 0xFF |

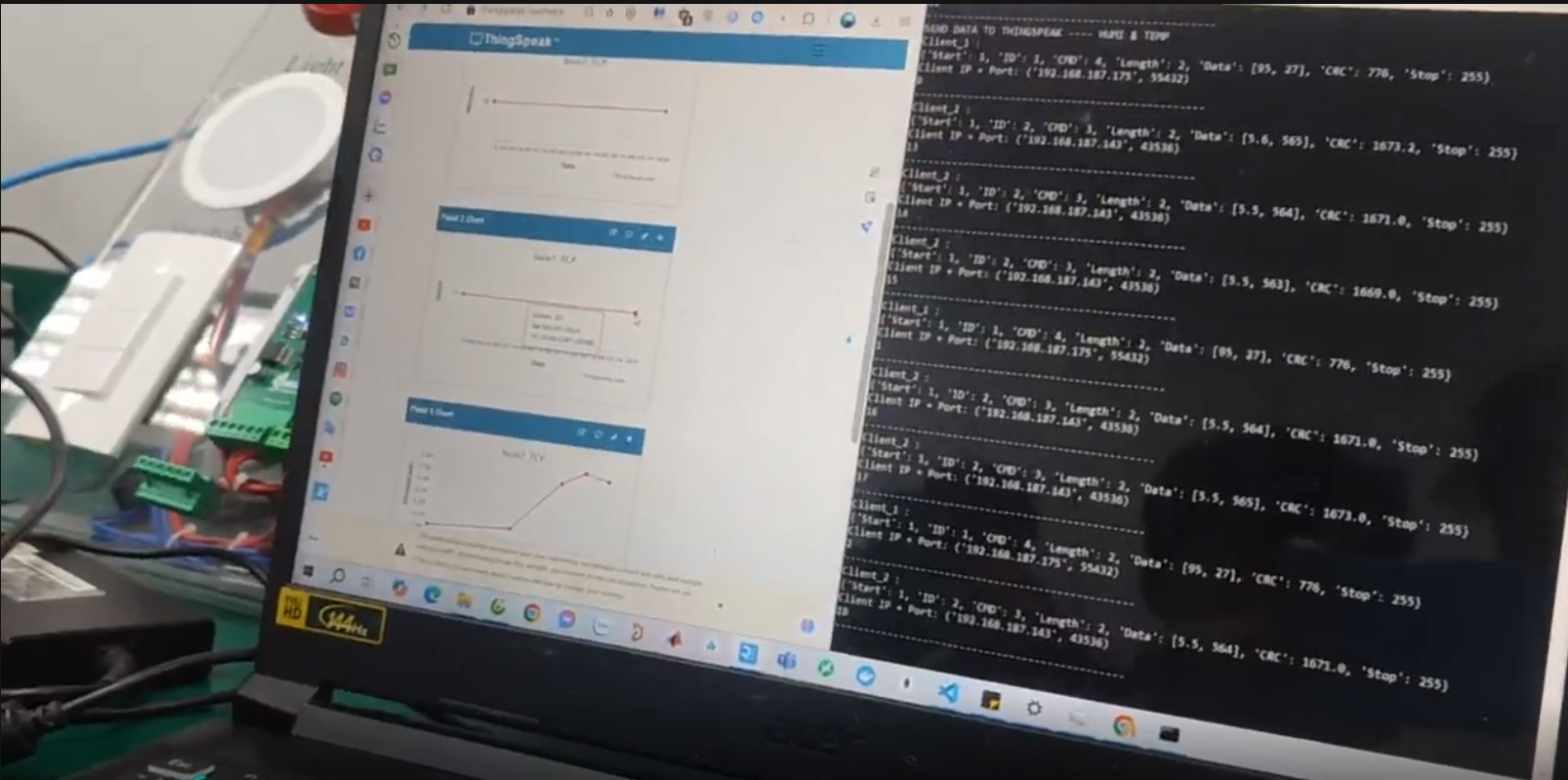
CRC byte được tính dựa trên giá trị LED 1, 2, 3 đã được đóng gói.

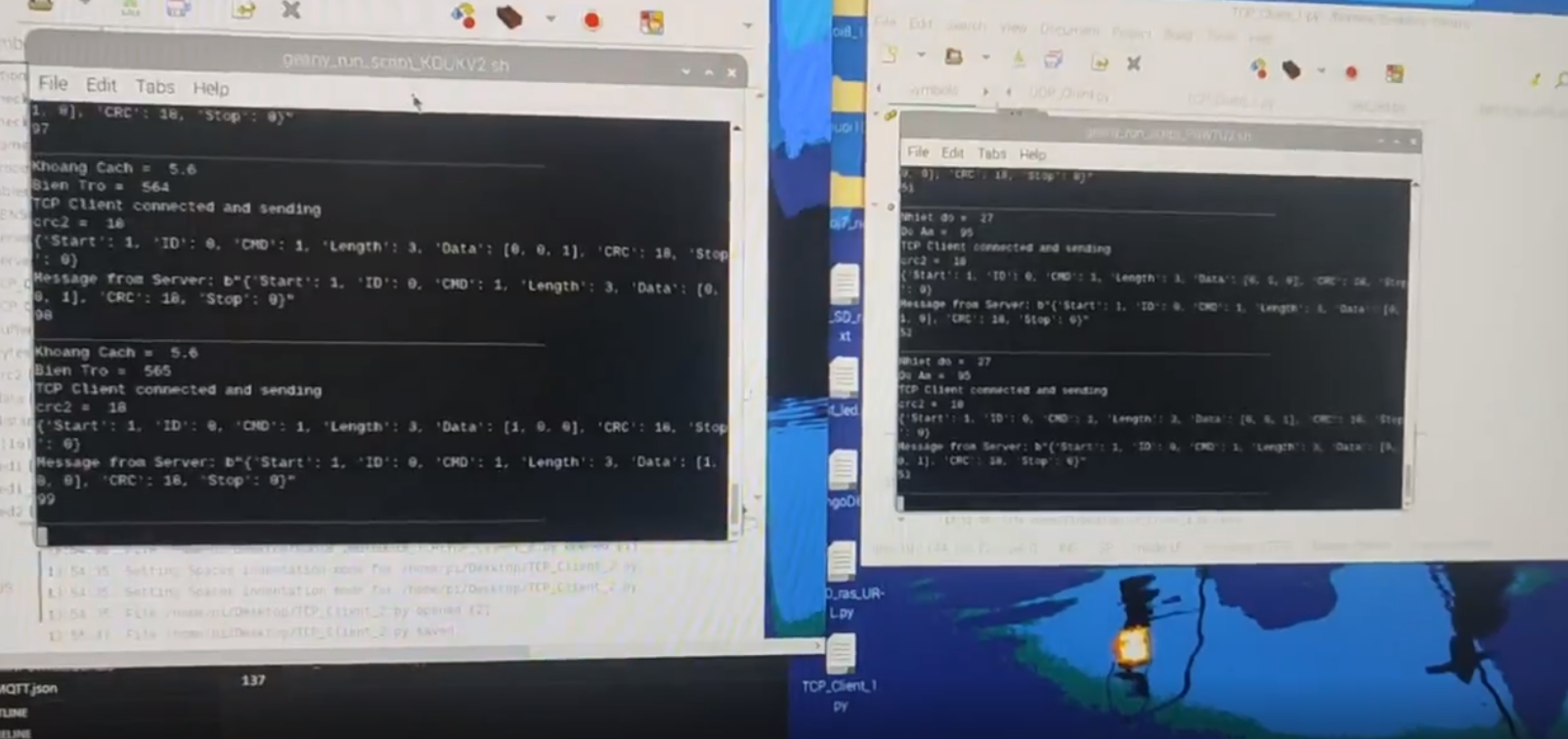
Terminal:

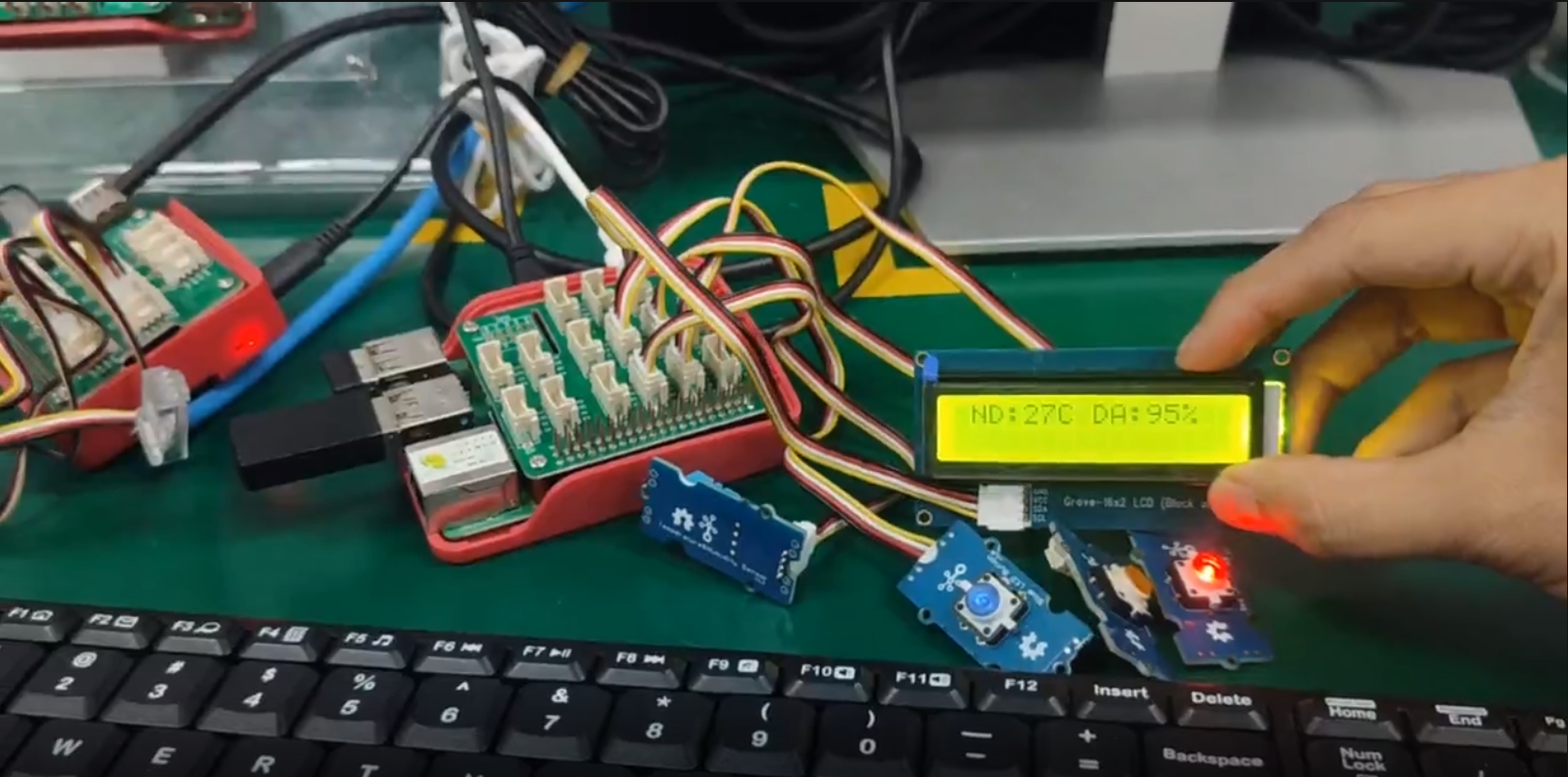


# **V.Yêu cầu 6: Minh chứng kết quả thực nghiệm.**









# **VI.Yêu cầu 7: Video.:**

https://www.youtube.com/watch?feature=shared&v=RBK6-SQvFGA