ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

ĐỒ ÁN THIẾT KẾ LUẬN LÝ - CO3093

Báo cáo cuối kì:

"Phát triển device driver cho DHT20, LCD 16x2, LED RGB và Module Wifi ESP32"

Giảng viên hướng dẫn: Nguyễn Thành Lộc

Sinh viên thực hiện: Lê Đức Cường - 2210423
Hoàng Thanh Chí Bảo - 2210205
Nguyễn Hạo Duy - 2210512

Thành phố Hồ Chí Minh, tháng 12 năm 2024

Mục lục

Danh sách hình vẽ

1 Tổng quan đề tài

1.1 Giới thiệu đề tài

Trong thời đại công nghệ hiện nay, IoT (Internet of Things) đang trở thành xu hướng chủ đạo, với các ứng dụng ngày càng mở rộng trong nhiều lĩnh vực như nhà thông minh, giám sát môi trường, tự động hóa công nghiệp và nông nghiệp thông minh. Điểm chung của các ứng dụng IoT này là sự cần thiết phải tích hợp nhiều thiết bị ngoại vi khác nhau để thu thập dữ liệu, hiển thị thông tin, xử lý tín hiệu và giao tiếp mạng. Các thiết bị như cảm biến, màn hình hiển thị, đèn LED, và module WiFi đóng vai trò cốt lõi trong việc xây dựng hệ thống. Tuy nhiên, việc khai thác hiệu quả và tối ưu hóa khả năng của các thiết bị này phụ thuộc vào khả năng lập trình và phát triển device driver phù hợp.

Device driver là một thành phần phần mềm trung gian giữa phần cứng và phần mềm ứng dụng, chịu trách nhiệm giao tiếp và điều khiển trực tiếp các thiết bị ngoại vi. Một driver được phát triển tốt không chỉ giúp đơn giản hóa quá trình lập trình mà còn đảm bảo hiệu suất, tính ổn định và khả năng mở rộng của hệ thống. Trong đề tài này, chúng em tập trung phát triển các driver cho các thiết bị phổ biến trong hệ thống IoT bao gồm:

+ Cảm biến DHT20: Là một cảm biến nhiệt độ và độ ẩm chính xác cao, sử dụng giao thức I2C, phù hợp để giám sát môi trường trong thời gian thực.

+ LCD 16x2: Màn hình hiển thị thông dụng sử dụng giao tiếp I2C hoặc GPIO, cho phép trình bày thông tin trực quan trong các ứng dụng nhúng.

+ LED báo hiệu trạng thái: Đèn LED gồm 3 màu xanh đỏ vàng, phù hợp để hiển thị các trạng thái đèn phù hợp với nhiệt độ, cảnh báo nhiệt độ cao,...

+ Module WiFi ESP32: Một module WiFi đa năng với khả năng hỗ trợ giao thức TCP/IP, MQTT và khả năng xử lý mạnh mẽ, đáp ứng yêu cầu giao tiếp mạng của các ứng dụng IoT.

+ Động cơ quạt mini: Module động cơ quạt sử dụng kỹ thuật điều chế độ rộng xung (PWM) để điều chỉnh tốc độ quạt.

1.2 Mục tiêu đề tài

Nghiên cứu giao thức truyền thông của từng thiết bị (I2C, PWM, UART) để hiểu rõ cơ chế hoạt động và yêu cầu kỹ thuật.

Phát triển các driver tương thích, hỗ trợ đầy đủ tính năng của từng thiết bị và tối ưu hóa hiệu năng.

Tích hợp các driver trên nền tảng vi điều khiển STM32 với môi trường phát triển STM32CubeIDE.

Xây dựng web để kiểm tra và minh họa khả năng hoạt động của các driver, như hiển thị thông tin cảm biến trên LCD qua giao diện mạng ESP32

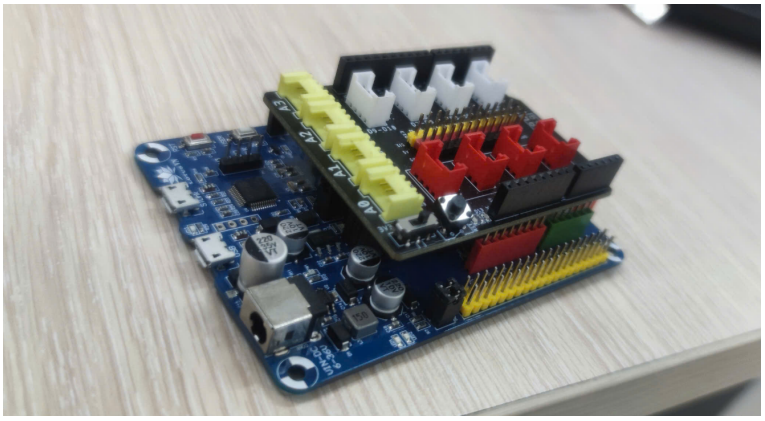
2 Thông tin thiết bị sử dụng

2.1 STM32F103RB

STM32F103RB là một vi điều khiển thuộc dòng STM32 của STMicroelectronics, được xây dựng trên kiến trúc ARM Cortex-M3. Đây là một trong những vi điều khiển phổ biến trong các ứng dụng nhúng nhờ vào hiệu suất cao, khả năng tiết kiệm năng lượng và nhiều tính năng phong phú.

2.1.1 Thông số kỹ thuật

- Kiến trúc: ARM Cortex-M3
- Tốc độ xung nhịp tối đa: 72 MHz
- Bộ nhớ Flash: 128 KB
- Bộ nhớ RAM: 20 KB
- Số chân: 64 chân (LQFP hoặc BGA)
- Điện áp hoạt động: 2.0V đến 3.6V



Hình 1: Board mạch STM32 mở rộng ở H6-812

2.1.2 Các tính năng nổi bật

- Giao thức UART, SPI, I2C, CAN, USB, và nhiều giao thức khác. Hỗ trợ giao tiếp với nhiều thiết bị ngoại vi và cảm biến.
- 12-bit ADC với tối đa 16 kênh đầu vào. Tốc độ chuyển đổi nhanh, cho phép thu thập dữ liệu từ cảm biến một cách hiệu quả.
- Nhiều bộ đếm thời gian (timers) cho phép điều khiển thời gian và tạo xung PWM.

2.1.3 Ứng dụng

- Hệ thống tự động hóa: Điều khiển thiết bị, cảm biến và thu thập dữ liệu.
- Thiết bị IoT: Kết nối và truyền dữ liệu qua mạng không dây hoặc có dây.
- Thiết bị y tế: Theo dõi sức khỏe, thiết bị đo lường.
- Robot và điều khiển động cơ: Điều khiển động cơ DC, servo, hoặc bước.
- Thiết bị tiêu dùng: Các sản phẩm điện tử như máy chơi game, thiết bị gia dụng thông minh.

2.1.4 Lập trình và các tài nguyên hỗ trợ

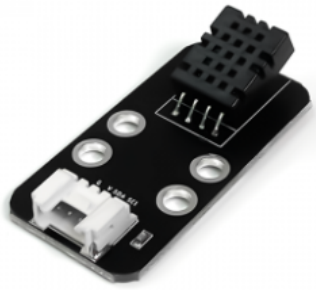
- STM32F103RB có thể lập trình bằng nhiều ngôn ngữ như C, C++ thông qua các IDE như STM32CubeIDE, Keil uVision, hoặc IAR Embedded Workbench.
- STMicroelectronics cung cấp tài liệu kỹ thuật, datasheet và hướng dẫn sử dụng cho STM32F103RB, giúp lập trình viên dễ dàng tìm hiểu và phát triển ứng dụng.

2.2 Cảm biến nhiệt độ độ ẩm DHT20

Cảm biến nhiệt độ và độ ẩm DHT20 sử dụng giao thức đầu ra là I2C. Cảm biến có độ chính xác cao, giá thành thấp, thích hợp với các ứng dụng cần đo nhiệt độ,độ ẩm của môi trường.

2.2.1 Thông số kỹ thuật

- Điện áp đầu vào: 3.3V
- Đo phạm vi độ ẩm: 0 - 100% RH và dải nhiệt độ đo: -40 $\tilde{+}$ 80 ℃
- Độ chính xác độ ẩm: ± 2% RH (25 ℃) và độ chính xác nhiệt độ: ± 0,5 ℃
- Tín hiệu đầu ra: Tín hiệu I2C



Hình 2: Hình ảnh module DHT20

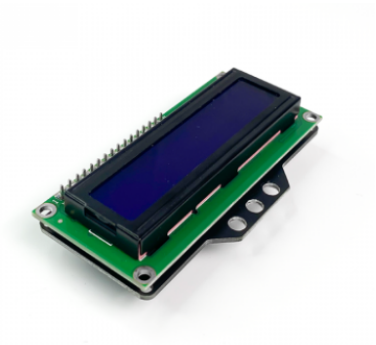
2.2.2 Ứng dụng

Cảm biến DHT20 được ứng dụng rộng rãi trong các dự án tự động hóa, như ghi nhận dữ liệu về nhiệt độ và độ ẩm trong môi trường. Nó có thể được sử dụng trong các hệ thống HVAC, máy hút ẩm, và các thiết bị theo dõi môi trường khác. Dưới đây là một số ứng dụng cụ thể của DHT20:

- Hệ thống điều hòa không khí (HVAC):
  - Giúp theo dõi và điều chỉnh nhiệt độ và độ ẩm trong không gian sống.
  - Tối ưu hóa hiệu suất của hệ thống HVAC bằng cách cung cấp dữ liệu chính xác về điều kiện môi trường.
- Máy hút ẩm:
  - DHT20 có thể được sử dụng để điều khiển máy hút ẩm, giúp duy trì độ ẩm trong không gian kín.
  - Cảm biến cung cấp thông tin cần thiết để máy hút ẩm hoạt động hiệu quả và tiết kiệm năng lượng.
- Ngoài ra, DTH20 còn sử dụng trong các thiết bị IOT, các dự án nghiên cứu về khí hậu, môi trường và các lĩnh vực khoa học khác.

2.3 Màn hình LCD 16x2

Màn hình LCD sử dụng driver HD44780, có khả năng hiển thị 2 dòng với mỗi dòng 16 ký tự, màn hình có độ bền cao, rất phổ biến, nhiều code mẫu và dễ dàng sử dụng hơn nếu đi kèm mạch chuyển tiếp I2C.



Hình 3: Màn hình LCD 16x2

2.3.1 Thông số kỹ thuật

- Điện áp hoạt động: 3.3 V.
- Địa chỉ I2C: 0x27
- Màu: Xanh lá

- Kích thước lỗ bắt ốc: 3x M3
- Kích thước của mạch: 80mm x 42mm x 19m
- Trọng lượng 38g

2.3.2 Ứng dụng

- Hiển thị thông tin: Nhiệt độ, độ ẩm, thời gian, trạng thái hệ thống.
- Menu điều khiển: Tạo giao diện menu cơ bản, giúp người dùng chọn các tùy chọn bằng nút bấm.
- Thông báo lỗi hoặc trạng thái: Hiển thị lỗi trong hệ thống, trạng thái hoạt động của thiết bị.
- Đồng hồ thời gian thực (RTC): Hiển thị giờ, phút, giây từ module RTC.
- Tích hợp vào hệ thống đo đạc: Ví dụ như hệ thống đo và hiển thị nhiệt độ đa nhiệm,...

2.4 ESP32

2.4.1 Thông số kỹ thuật

- Cổng nạp: Type C hoặc Micro USB (tùy chọn).
- Nguồn: 5V (USB), 3.3V I/O.
- Hiệu năng: Lên đến 600 DMIPS, tần số 240 MHz.
- Kết nối: WiFi 802.11 b/g/n (2.4 GHz, 150 Mbps), Bluetooth 4.2 BLE + BR/EDR.
- Bộ nhớ: 448 KB ROM, 520 KB SRAM, hỗ trợ flash ngoài.
- Chip USB-Serial: CP2102.
- GPIO: 24 chân, ADC 12-bit (18 kênh).
- Bảo mật: AES, SHA-2, RSA, ECC, RNG.

Hình 4: Hình ảnh module ESP32

2.4.2 Ứng dụng

ESP32 có thể được lập trình thông qua Arduino IDE, cho phép người dùng thực hiện các dự án từ cơ bản đến nâng cao, như giám sát môi trường và điều khiển thiết bị. ESP32 có khả năng thiết lập web server, cho phép người dùng truy cập và điều khiển thiết bị qua trình duyệt web. Ngoài ra, nó có thể kết nối với nhiều thiết bị khác nhau, mở rộng khả năng tương tác và điều khiển dựa vào tính năng Bluetooth chế độ kép.

2.5 Động cơ quạt mini

2.5.1 Thông số kỹ thuật

- Điện áp hoạt động: 3.3V
- Tín hiệu điều khiển: 2 pins
- Kích thước của mạch: 24mm x 48mm x 16mm

Hình 5: Hình ảnh module động cơ quạt mini

2.5.2 Ứng dụng

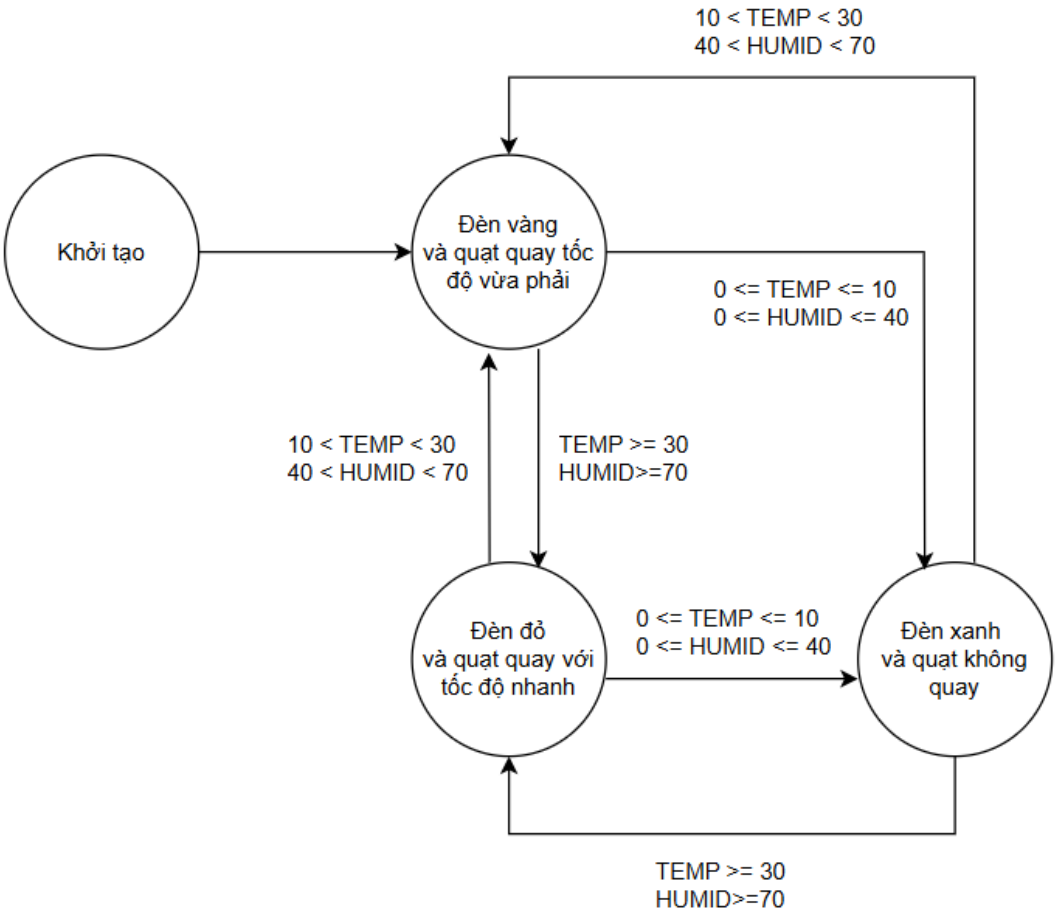
Có thể kết hợp module động cơ này với những module cảm biến khác như: điều khiển bật/tắt quạt theo thời gian, đọc giá trị nhiệt độ để bật/tắt quạt,...

2.6 LED báo hiệu trạng thái

Hình 6: Hình ảnh LED trạng thái 3 màu

3 Thiết kế

3.1 Máy trạng thái báo hiệu tín hiệu đèn và tốc độ quạt



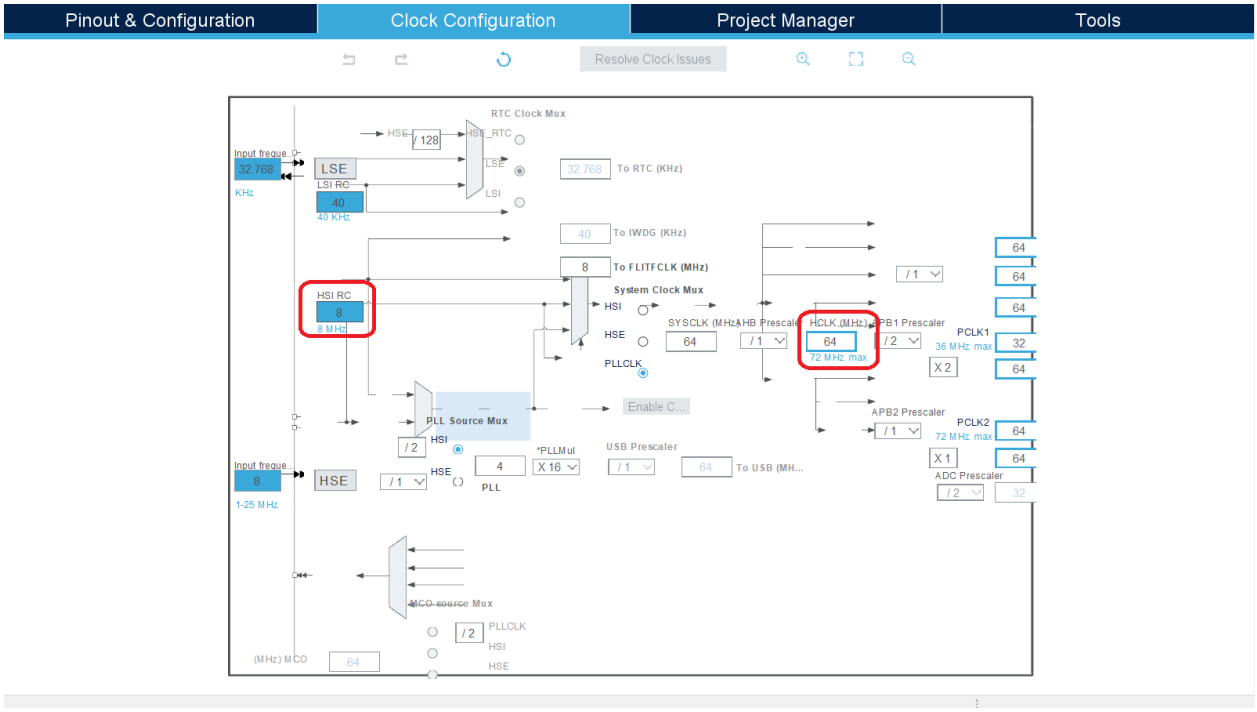
Hình 7: Máy trạng thái biểu diễn LED báo hiệu và tốc độ quạt

Để thuận tiện cho quá trình mô phỏng, đèn báo trạng thái và quạt được thể hiện rõ hơn, nhóm chúng em đã cài đặt lại như sau:

- Đèn xanh: Dưới 27°C
- Đèn vàng: từ 27 - 28°C
- Đèn đỏ: Trên 28°C

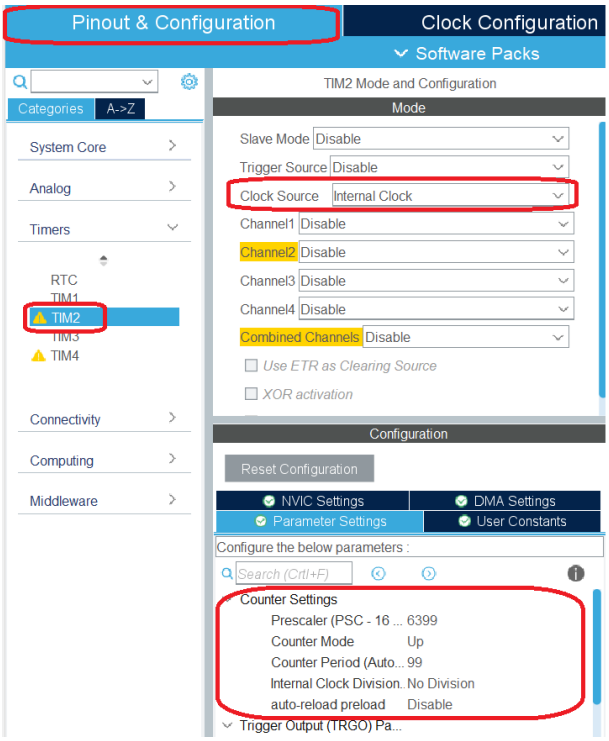
3.2 Timer Interrupt Setup

Bước 1: Kiểm tra nguồn xung nhịp của hệ thống trên tab Clock Configuration (từ tệp *.ioc).



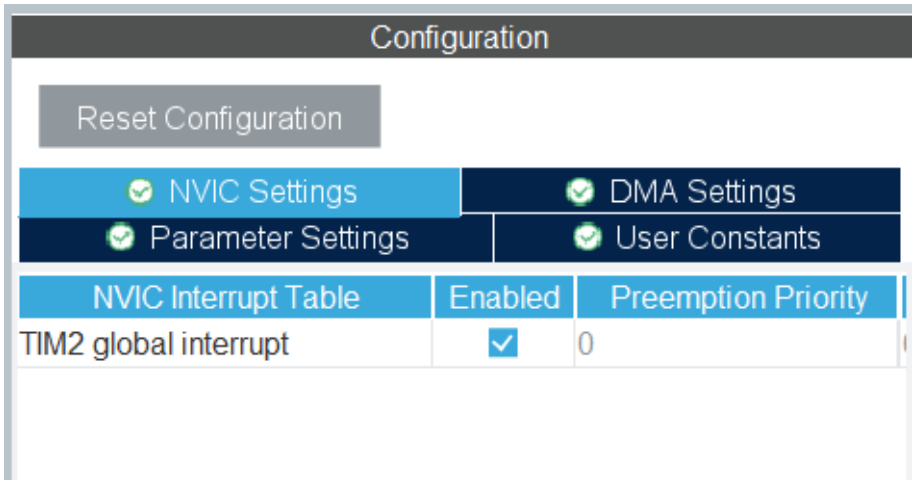
Hình 8: Cài đặt Clock Configuration cho hệ thống

Bước 2: Cấu hình thời gian trong Parameter Settings như sau:



Hình 9: Cấu hình cho Timer2

Bước 3: Enable timer interrupt



Hình 10: Enable timer interrupt

Tính toán Prescaller với Counter Period:

- Mục tiêu là đặt một ngắt timer có chu kỳ 10ms.
- Nguồn clock là **64MHz** (ở hình trên), bằng cách đặt **Prescaller** là 6399, nguồn Clock đầu vào cho Timer là: $\frac{\mathbf{64}}{\mathbf{(6399+1)}} = \mathbf{100000}$ Hz
- Ngắt sẽ được kích hoạt khi bộ đếm của timer đếm từ 0 đến 99, nghĩa là tần số được chia thêm cho 100, tương ứng với 100Hz.
- Tần số ngắt của timer là **100 Hz**, nghĩa là chu kỳ là $\frac{\mathbf{1}}{\mathbf{100}} = \mathbf{10}$ ms.

Bước 4: Tạo 1 file software_timer.c để tạo và quản lý các bộ đếm thời gian timer và 1 file software_timer.h để khai báo biến và các hàm.

```c
#define timer_cycle 10

int timer_counter=0;
int timer_flag=0;
int timer_counter_fan=0;
int timer_flag_fan=0;
void setTimer(int duration)
{
  timer_counter = duration/timer_cycle;
  timer_flag = 0;
}
void setTimer_fan(int duration)
{
  timer_counter_fan = duration/timer_cycle;
  timer_flag_fan = 0;
}
void timeRun()
{
  if(timer_counter > 0){
    timer_counter--;
    if(timer_counter <= 0){
      timer_flag = 1;
    }
  }
  if(timer_counter_fan > 0){
    timer_counter_fan--;
    if(timer_counter_fan <= 0){
      timer_flag_fan = 1;
    }
  }
}
```

Bước 5: Ở file main.c, thêm hàm ngắt, hàm này được gọi sau mỗi 10ms:

```c
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  timeRun();
}
```

3.3 Scheduler

Scheduler giúp tổ chức các công việc trong hệ thống theo cách tối ưu hóa hiệu suất. Một **scheduler** hiệu quả không chỉ giảm thiểu tài nguyên hệ thống mà còn đảm bảo khả năng đáp ứng thời gian thực. Nhóm chúng em đã thiết kế Scheduler cụ thể 2 hàm SCH_Dispatch và SCH_Update đơn giản độ phức tạp chỉ với O(1). Lý do chúng em thiết kế 2 hàm độ phức tạp với O(1) vì:

- Nếu thiết kế hàm SCH_Dispatch và SCH_Update với độ phức tạp O(n): Điều này gây tốn thời gian thực hiện vì cả hai hàm đều sử dụng vòng lặp. Cụ thể, hàm SCH_Update được gọi trong ngắt 10ms, còn hàm SCH_Dispatch được gọi trong vòng lặp while(1).
- Chúng em đã vận dụng kiến thức được học ở môn Vi xử lý - Vi điều khiển và Cấu trúc dữ liệu & Giải thuật để tránh gây tốn thời gian cho mỗi vòng lặp liên tục. Do đó, nhóm chúng em đã thiết kế các hàm có độ phức tạp O(1) và chuyển độ phức tạp sang các hàm khác nhằm giảm thiểu thời gian thực hiện trong mỗi lần lặp.

```
SCH_Task tasks[SCH_TASKNUMBER];

void SCH_Init(void) {
    for (uint8_t i = 0; i < SCH_TASKNUMBER; i ++) {
        tasks[i].functionPointer = 0;
        tasks[i].id = SCH_TASKNUMBER - i;
        tasks[i].delay = 0;
        tasks[i].period = 0;
        tasks[i].flag = 0;
    }
}

void SCH_Update(void) {
    if (tasks[0].functionPointer == 0) return;
    if (tasks[0].delay > 0) {
  if (tasks[0].delay > TIMER_TICK) {
    tasks[0].delay -= TIMER_TICK;
  }
    else {
  tasks[0].delay = 0;
        }
    }
    if (tasks[0].delay == 0) {
  tasks[0].flag = 1;
    }
}

void SCH_Dispatch(void) {
```

```
    if (tasks[0].flag == 0) return;
    (*tasks[0].functionPointer)();
    if (tasks[0].period > 0) {
        SCH_RefreshTask();
    }
    else {
        SCH_DeleteTask(tasks[0].id);
    }
}

uint8_t SCH_AddTask(void (*functionPointer)(void), uint32_t delay, uint32_t period) {
    if (tasks[SCH_TASKNUMBER - 1].functionPointer != 0) return 0;
    uint8_t currentID = tasks[SCH_TASKNUMBER - 1].id;
    uint32_t currentDelay = 0;
    for (uint8_t i = 0; i < SCH_TASKNUMBER; i ++) {
        currentDelay += tasks[i].delay;
        if (currentDelay > delay || tasks[i].functionPointer == 0) {
            for (uint8_t j = SCH_TASKNUMBER - 1; j > i; j --) {
                tasks[j] = tasks[j - 1];
            }
            tasks[i].functionPointer = functionPointer;
            tasks[i].id = currentID;
            tasks[i].period = period;
            tasks[i].flag = 0;
            if (currentDelay > delay) {
                int newDelay = currentDelay - delay;
                tasks[i].delay = tasks[i + 1].delay - newDelay;
                if (tasks[i].delay == 0) {
                    tasks[i].flag = 1;
                }
                tasks[i + 1].delay = newDelay;
                if (tasks[i + 1].delay == 0) {
                    tasks[i + 1].flag = 1;
                }
            }
            else {
                tasks[i].delay = delay - currentDelay;
                if (tasks[i].delay == 0) {
                    tasks[i].flag = 1;
                }
            }
            return tasks[i].id;
        }
    }
```

```
73      return 0;
74  }
75
76  unsigned char SCH_DeleteTask(uint8_t id) {
77      for (uint8_t i = 0; i < SCH_TASKNUMBER; i ++) {
78        if (tasks[i].functionPointer == 0) return 0;
79          if (tasks[i].id == id) {
80              uint8_t currentID = tasks[i].id;
81              if (tasks[i + 1].functionPointer != 0) {
82                  tasks[i + 1].delay += tasks[i].delay;
83              }
84              for (uint8_t j = i; j < SCH_TASKNUMBER - 1; j ++) {
85                  tasks[j] = tasks[j + 1];
86              }
87              tasks[SCH_TASKNUMBER - 1].functionPointer = 0;
88              tasks[SCH_TASKNUMBER - 1].id = currentID;
89              tasks[SCH_TASKNUMBER - 1].delay = 0;
90              tasks[SCH_TASKNUMBER - 1].period = 0;
91              tasks[SCH_TASKNUMBER - 1].flag = 0;
92              return tasks[SCH_TASKNUMBER - 1].functionPointer == 0;
93          }
94      }
95      return 0;
96  }
97
98  unsigned char SCH_RefreshTask(void) {
99      if (tasks[0].functionPointer == 0) return 0;
100     SCH_Task currentTask = tasks[0];
101     uint32_t currentDelay = 0;
102     for (uint8_t i = 0; i < SCH_TASKNUMBER; i ++) {
103         if (i + 1 == SCH_TASKNUMBER || tasks[i + 1].functionPointer == NULL) {
104             tasks[i].functionPointer = currentTask.functionPointer;
105             tasks[i].id = currentTask.id;
106             tasks[i].period = currentTask.period;
107             tasks[i].flag = 0;
108             tasks[i].delay = currentTask.period - currentDelay;
109             if (tasks[i].delay == 0) {
110                 tasks[i].flag = 1;
111             }
112             return 1;
113         }
114         currentDelay += tasks[i + 1].delay;
115         if (currentDelay > currentTask.period) {
116             tasks[i].functionPointer = currentTask.functionPointer;
```

```
17              tasks[i].id = currentTask.id;
18              tasks[i].period = currentTask.period;
19              tasks[i].flag = 0;
20              int newDelay = currentDelay - currentTask.period;
21              tasks[i].delay = tasks[i + 1].delay - newDelay;
22              if (tasks[i].delay == 0) {
23                  tasks[i].flag = 1;
24              }
25              tasks[i + 1].delay -= tasks[i].delay;
26              if (tasks[i + 1].delay == 0) {
27                  tasks[i + 1].flag = 1;
28              }
29              return 1;
30          }
31          else {
32              tasks[i] = tasks[i + 1];
33          }
34      }
35      return 0;
36  }
```

Listing 1: Mã nguồn file scheduler.c

Phân tích:

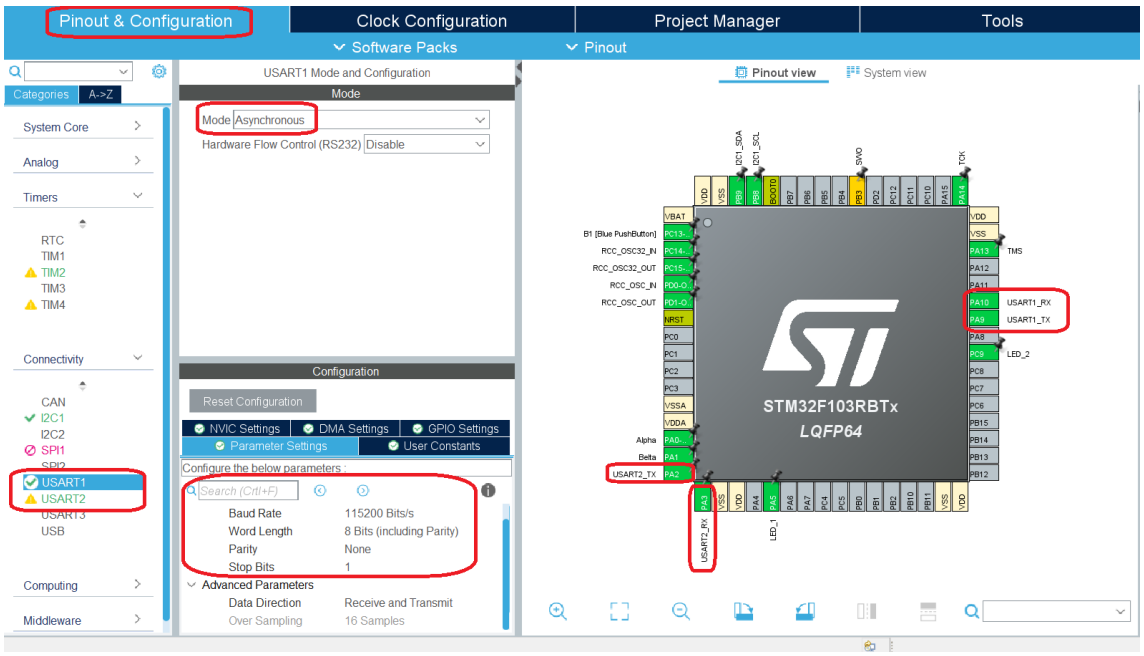
- Cấu trúc Dữ liệu
  - `functionPointer`: con trỏ tới hàm thực thi tác vụ.
  - `id`: ID của tác vụ.
  - `delay`: thời gian còn lại trước khi tác vụ có thể thực thi.
  - `period`: chu kỳ thực thi (nếu có).
  - `flag`: cờ cho biết tác vụ đã sẵn sàng để thực thi hay chưa.
- Các hàm chính trong scheduler
  - `SCH_Init`: Hàm khởi tạo hệ thống lập lịch. Hàm này thiết lập tất cả các con trỏ hàm về 0, ID, delay, period, và flag về 0 cho tất cả các tác vụ.
  - `SCH_Update`: Hàm cập nhật trạng thái của tác vụ đầu tiên. Nếu delay lớn hơn 0, hàm sẽ giảm delay xuống theo giá trị của TIMER_TICK. Khi delay bằng 0, flag sẽ được đặt thành 1.
  - `SCH_Dispatch`: Hàm kiểm tra flag của tác vụ đầu tiên. Nếu flag bằng 1, hàm sẽ gọi hàm được chỉ định bởi `functionPointer`. Nếu tác vụ có chu kỳ (period $>$ 0), hàm sẽ gọi `SCH_RefreshTask`; ngược lại, nó sẽ gọi `SCH_DeleteTask` để xóa tác vụ.

- `SCH_AddTask`: Hàm dùng để thêm một tác vụ mới vào danh sách tác vụ. Nó kiểm tra xem có còn chỗ trống không và duyệt qua danh sách để xác định vị trí chèn tác vụ mới, cập nhật delay và flag cho các tác vụ liên quan.
- `SCH_DeleteTask`: Hàm xóa một tác vụ theo ID. Nó cập nhật delay cho tác vụ tiếp theo nếu tồn tại và di chuyển các tác vụ còn lại lên để lấp đầy chỗ trống.
- `SCH_RefreshTask`: Hàm cập nhật tác vụ đầu tiên với thông tin từ danh sách. Nó tính toán delay mới cho tác vụ và cập nhật flag nếu cần.

- Độ phức tạp của thuật toán đối với các hàm trong scheduler
  - `SCH_Init`: **O(n)**, với n là số lượng tác vụ, vì hàm này duyệt qua tất cả các tác vụ để khởi tạo.
  - `SCH_Update`: **O(1)**, vì chỉ cập nhật tác vụ đầu tiên.
  - `SCH_Dispatch`: **O(1)**, vì nó chỉ kiểm tra và gọi hàm của tác vụ đầu tiên.
  - `SCH_AddTask`: **O(n)**, trong trường hợp xấu nhất, hàm này có thể cần duyệt qua tất cả các tác vụ để tìm vị trí chèn.
  - `SCH_DeleteTask`: **O(n)**, vì cần duyệt qua danh sách để tìm tác vụ cần xóa.
  - `SCH_RefreshTask`: **O(n)**, vì cần duyệt qua các tác vụ để cập nhật thông tin cho tác vụ đầu tiên.

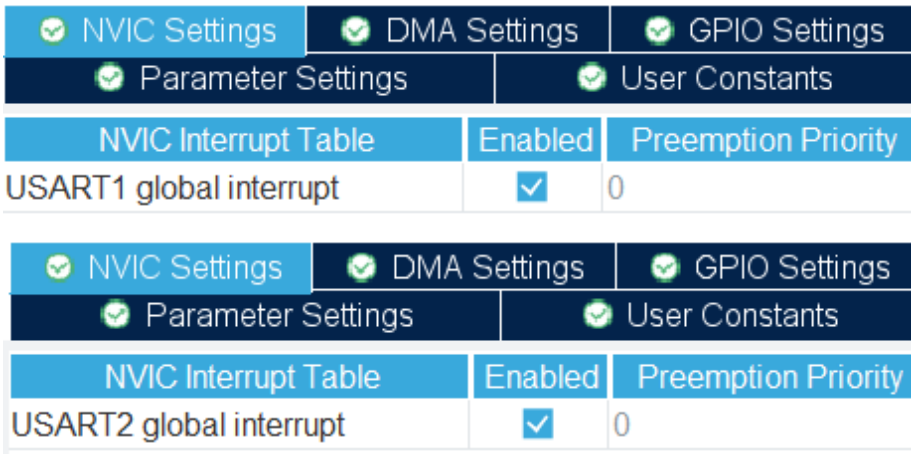
3.4 Giao tiếp UART với ESP32

Bước 1: Chúng ta cấu hình cho lần lượt UART1 và UART2



Hình 11: Cấu hình UART cho cả UART1 VÀ UART2

Bước 2: Chúng ta kích hoạt ngắt UART

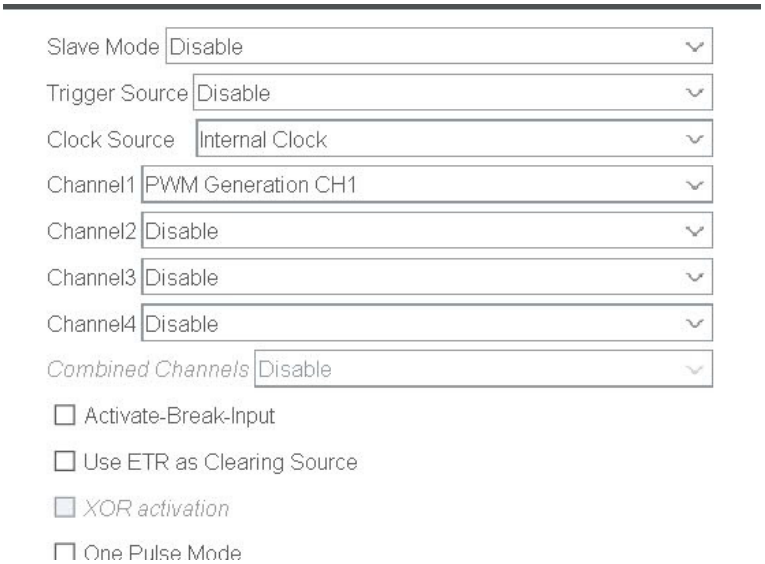


Hình 12: Enable UART interrupt

Bước 3: Ở module ESP32, thì chúng ta sử dụng công cụ ArduinoIDE để lập trình, include thư viện cần thiết "driver/uart.h" để điều khiển giao tiếp UART. Sau đó #define các chân RXD và TXD tương ứng như đã trình bày ở phần chi tiết nối dây 4.5.1.

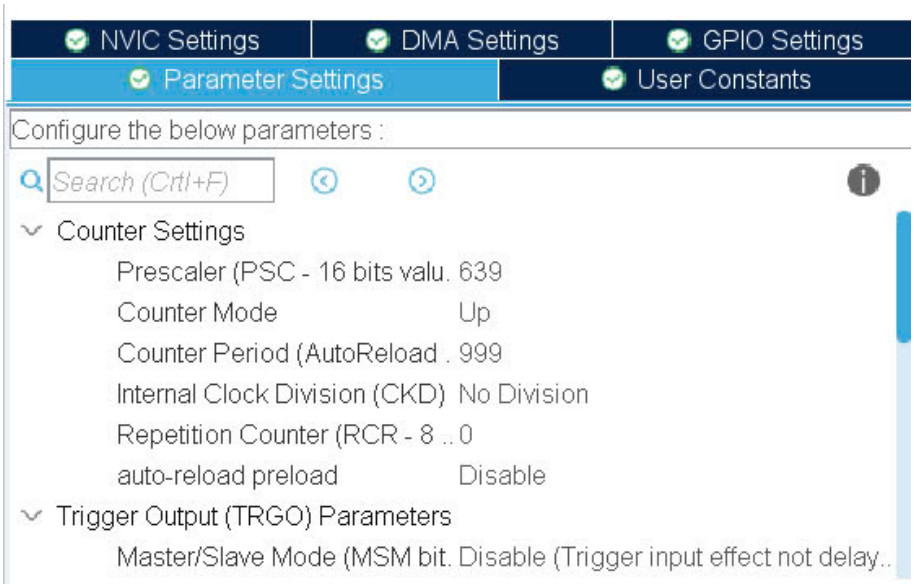
3.5 PWM điều khiển quạt

Bước 1: Chúng ta cấu hình PMW channel 1



Hình 13: Cấu hình PMW Channel 1 (Chân PA8)

Bước 2: Cấu hình thời gian trong Parameter Settings như sau:



Hình 14: Cấu hình thời gian trong Parameter Settings của quạt

Bước 3: Cấu hình xung cho quạt

Hình 15: Cấu hình xung cho quạt có Duty Cycle 20%

Bước 4: Để thay đổi Pulse của quạt, ta sẽ dùng hàm HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, VAR) với VAR = 0 thì Dutycycle = 0. Nếu VAR = 199 thì Duty cycle = 20%, nếu VAR = 499 thì Duty cycle = 50%. Nếu Duty cycle của quạt càng cao thì tốc độ quay của quạt càng cao. Nếu Duty cycle = 100% thì quạt quay với tốc độ tối đa.

3.6 Web Server

Chức năng chính của website trên là hiện thị Nhiệt độ(C và F) cùng với Độ ẩm trong cùng một đơn vị thời gian bằng dạng biểu đồ tròn hoặc là dạng số thực. Lấy tham số max lần lượt là 100°C, 373°F, 100% để làm tham chiếu tính % cho biểu đồ tròn.

Giao diện bao gồm ba vòng tròn đại diện cho các thông số:

- Nhiệt độ (°C): Hiển thị nhiệt độ hiện tại cùng với mức tối đa cho phép.
- Nhiệt độ (°F): Tương tự như trên, nhưng sử dụng đơn vị Fahrenheit.
- Độ ẩm (%): Hiển thị độ ẩm hiện tại và mức tối đa.

Sử dụng công cụ Arduino IDE để lập trình và điều khiển module ESP32. ESP32 là một vi điều khiển mạnh mẽ với khả năng kết nối Wi-Fi và Bluetooth, cho phép thu thập dữ liệu từ Module STM32 thông qua giao thức UART và gửi chúng lên website .

HTLM được sử dụng để xây dựng cấu trúc cơ bản của giao diện web. HTML cung cấp các thẻ và phần tử cần thiết để hiển thị thông tin về nhiệt độ và độ ẩm một cách rõ ràng và dễ hiểu. Bên cạnh đó, CSS dùng để định dạng và tạo kiểu cho giao diện web. JavaScript được sử dụng để xử lý dữ liệu nhận được từ ESP32 và cập nhật giao diện người dùng theo thời gian thực. JavaScript cho phép tương tác động với các phần tử HTML, tạo ra các hiệu ứng và cải thiện tính năng của website, như làm mới dữ liệu mà không cần tải lại trang.

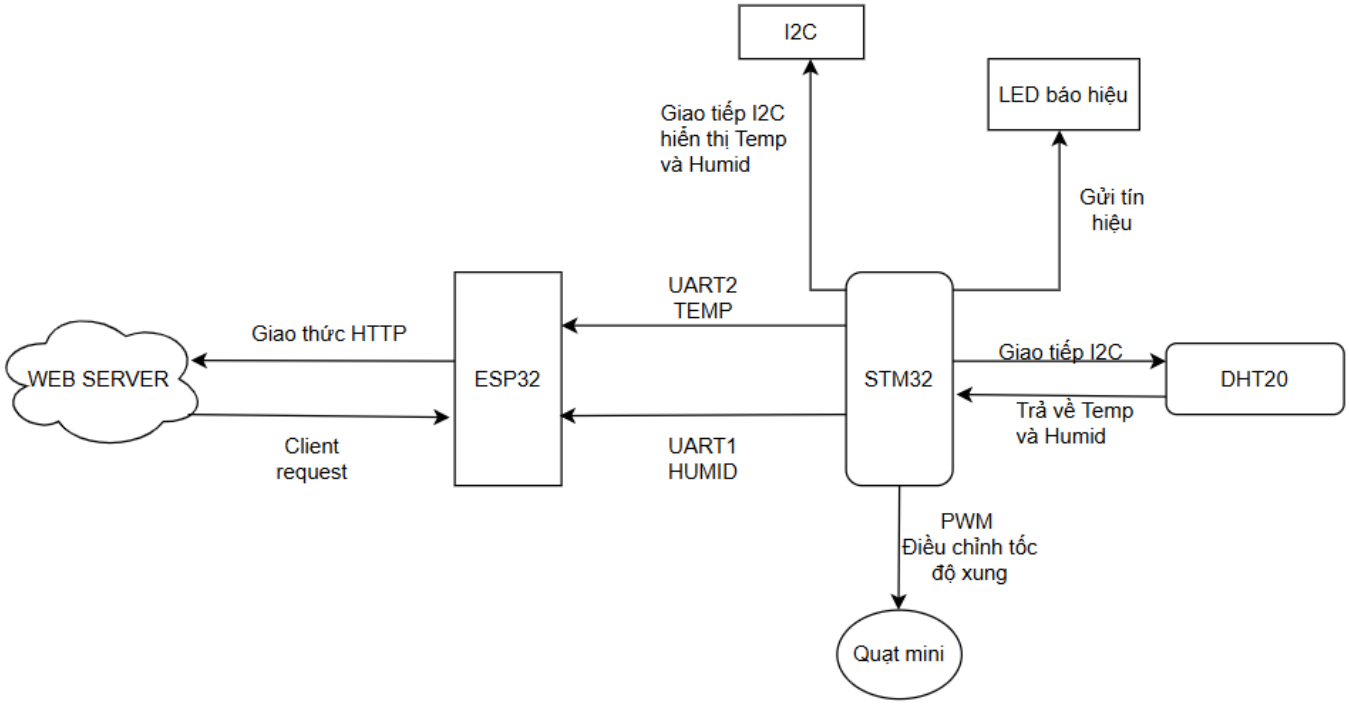
Giao diện web này là một ví dụ điển hình về cách trình bày thông tin một cách trực quan và dễ hiểu. Nó phản ánh chính xác dữ liệu thu được từ cảm biến DHT20 về nhiệt độ và độ ẩm trong môi trường. Từ đó, người dùng có thể nhận định rõ hơn về các điều kiện thời tiết hiện tại. Hơn nữa, đồ thị cung cấp cái nhìn trực quan để so sánh các giá trị theo thời gian, giúp người dùng dễ dàng theo dõi và phân tích sự biến đổi của nhiệt độ và độ ẩm.

4 Hiện thực

4.1 Số lượng phần cứng sử dụng

STT	Tên phần cứng	Số lượng
1	STM32F103RB	1
2	ESP32	1
3	DHT20	1
4	LED trạng thái	1
5	Quạt mini	1

4.2 Sơ đồ giao tiếp giữa các module



Hình 16: Sơ đồ giao tiếp giữa các module

Mô tả sơ đồ hệ thống:

- Đo đạc thông số môi trường: Cảm biến DHT20 thực hiện đo nhiệt độ và độ ẩm, sau đó gửi dữ liệu qua giao tiếp I2C đến STM32.

- Xử lý dữ liệu tại STM32:

  - STM32 nhận dữ liệu từ DHT20 qua giao tiếp I2C.
  - Dữ liệu được xử lý để:
    - Gửi thông số nhiệt độ qua UART2 và độ ẩm qua UART1 đến ESP32.
    - Kiểm tra ngưỡng nhiệt độ và độ ẩm để điều khiển quạt mini qua tín hiệu PWM.
    - Thay đổi trạng thái LED báo hiệu nếu cần thiết.
    - Hiển thị thông số lên màn hình I2C.

- Truyền dữ liệu đến ESP32:

  - STM32 truyền dữ liệu nhiệt độ và độ ẩm qua UART đến ESP32.
  - ESP32 đóng vai trò chuyển tiếp dữ liệu này lên Web Server thông qua giao thức HTTP.

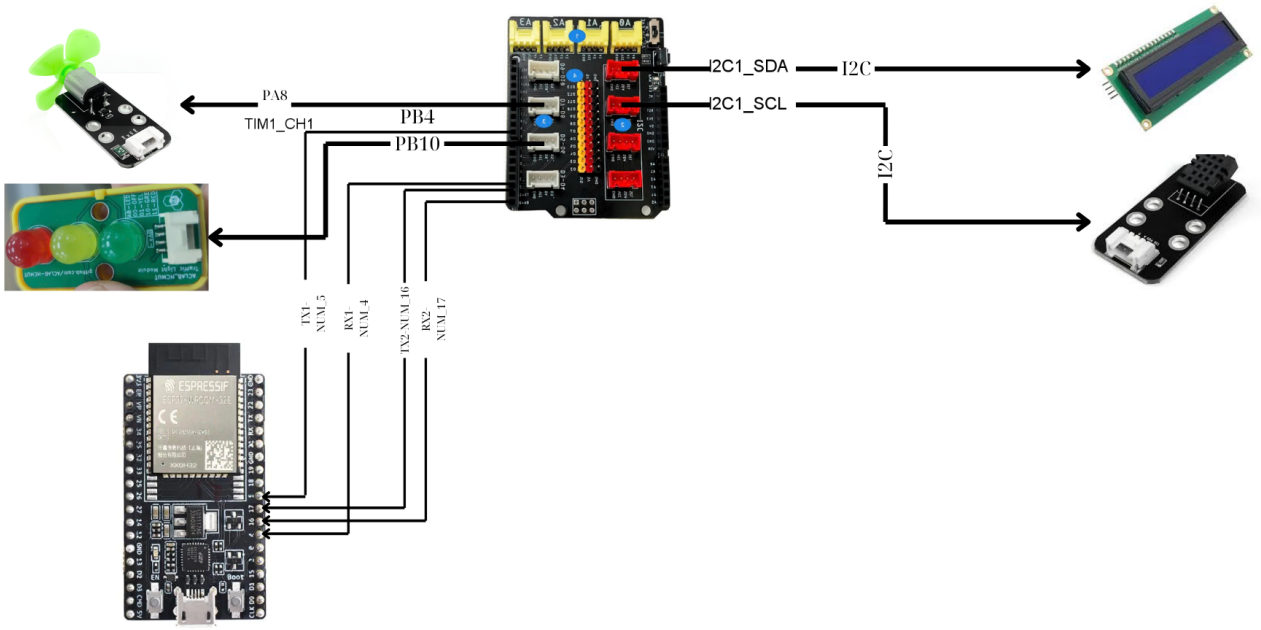
- Hiển thị qua Web Server:

  - Web Server nhận dữ liệu từ ESP32 và hiển thị lên giao diện Web.

- Điều khiển quạt và LED:

  - STM32 điều chỉnh tốc độ quạt qua tín hiệu PWM dựa trên điều kiện nhiệt độ, độ ẩm.
  - LED báo hiệu trạng thái hoạt động hoặc cảnh báo các tình huống đặc biệt.

4.3 Sơ đồ nối dây toàn bộ hệ thống



Hình 17: Sơ đồ nối dây

4.4 Hiện thực cảm biến DHT20 và hiển thị ra LCD

4.4.1 Mã nguồn cảm biến DHT20

```
extern I2C_HandleTypeDef hi2c1;

#define SLAVE_ADDRESS_DHT20 (0x38 << 1)

uint16_t value_x10[2] = {0, 0};
char temp[20], humid[20];

void dht20_init(){
  //Set register when call a wrong reset
  uint8_t init[3];
  init[0] = 0xA8;
  init[1] = 0x00;
  init[2] = 0x00;
  HAL_I2C_Master_Transmit(&hi2c1, SLAVE_ADDRESS_DHT20, (uint8_t*) init, 3, 0xFF);
  setTimer(10);
  init[0] = 0xBE;
  init[1] = 0x08;
  init[2] = 0x00;
  HAL_I2C_Master_Transmit(&hi2c1, SLAVE_ADDRESS_DHT20, (uint8_t*) init, 3, 0xFF);
  setTimer(10);
}

void dht20_reset(uint8_t regist){
  //reset register
  uint8_t reset[3], reply[3];
  reset[0] = regist;
  reset[1] = 0x00;
  reset[2] = 0x00;
  HAL_I2C_Master_Transmit(&hi2c1, SLAVE_ADDRESS_DHT20, (uint8_t*) reset, 3, 0xFF);
  setTimer(10);

  HAL_I2C_Master_Receive(&hi2c1, SLAVE_ADDRESS_DHT20 | 0x01, (uint8_t*) reply, 3, 0xFF)
    ;
  reset[0] = 0xB0 | regist;
  reset[1] = reply[1];
  reset[2] = reply[2];
  setTimer(10);
  HAL_I2C_Master_Transmit(&hi2c1, SLAVE_ADDRESS_DHT20, (uint8_t*) reset, 3, 0xFF);
}

void dht20_start(){
```

```
41   //query the DHT20
42   uint8_t status[1];
43   HAL_I2C_Master_Receive(&hi2c1, SLAVE_ADDRESS_DHT20 | 0x01, (uint8_t*) status, 1, 0xFF
       );
44   if((status[0] & 0x18) != 0x18){
45     dht20_reset(0x1B);
46     dht20_reset(0x1C);
47     dht20_reset(0x1E);
48     HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, RESET);
49   }
50   if ((status[0] & 0x18) == 0x18){
51     HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, SET);
52   }
53
54   uint8_t data[3] = {0xAC, 0x33, 0x00};
55   //HAL_I2C_Master_Transmit(&hi2c1, SLAVE_ADDRESS_DHT20, (uint8_t*) trigger, 1, 0xFF);
56   HAL_I2C_Master_Transmit(&hi2c1, SLAVE_ADDRESS_DHT20, (uint8_t*) data, 3, 0xFF);
57   setTimer(80);
58 }
59
60 void dht20_read(uint16_t* value){
61   dht20_start();
62   uint8_t data[7];
63   uint32_t Temper = 0, Humid = 0;
64   HAL_I2C_Master_Receive(&hi2c1, SLAVE_ADDRESS_DHT20 | 0x01, (uint8_t*) data, 7, 0xFF);
65   //Humid
66   Humid = (Humid | data[1]) << 8;
67   Humid = (Humid | data[2]) << 8;
68   Humid = Humid | data[3];
69   Humid = Humid >> 4;
70     Humid = (Humid * 100 * 10 / 1024 / 1024);
71     value[0] = Humid;
72   //Temperature
73     Temper = (Temper | data[3]) << 8;
74     Temper = (Temper | data[4]) << 8;
75     Temper = Temper | data[5];
76     Temper = Temper & 0xfffff;
77     Temper = Temper*200*10/1024/1024 - 500;
78   value[1] = Temper;
79
80 }
81 void dht20_output(){
82   dht20_read(value_x10);
83   //Display on the LCD
```

```
84 //  sprintf(temp, "TEMP:  %d.%d %cC",value_x10[1]/10,value_x10[1]%10 ,0b11011111);
85 //  sprintf(humid,"HUMID: %01d.%d %%   ",value_x10[0]/10,value_x10[0]%10);
86     sprintf(temp, "%d.%d ",value_x10[1]/10,value_x10[1]%10 );
87     sprintf(humid,"%01d.%d    ",value_x10[0]/10,value_x10[0]%10);
88   lcd_show_value();
89 }
```

Listing 2: Mã nguồn file dht20.c

Mô tả:

- dht20_read(uint16_t* value): Đọc dữ liệu từ cảm biến và tính toán giá trị độ ẩm và nhiệt độ. Kết quả được lưu vào mảng value[]:
  - value[0]: Độ ẩm (% x10).
  - value[1]: Nhiệt độ (°C x10).
- dht20_output(): Gọi hàm dht20_read() để lấy dữ liệu từ cảm biến. Và xử lý dữ liệu thành chuỗi temp và humid để hiển thị lên LCD thông qua lcd_show_value().

4.4.2 Mã nguồn LCD

```
1  #include "I2C-LCD.h"
2  #include "dht20.h"
3
4  extern I2C_HandleTypeDef hi2c1;  // change your handler here accordingly
5
6  #define SLAVE_ADDRESS_LCD (0x21 << 1) // change this according to ur setup
7
8  void lcd_send_cmd (char cmd)
9  {
10   char data_u, data_l;
11   uint8_t data_t[4];
12   data_u = (cmd&0xf0);
13   data_l = ((cmd<<4)&0xf0);
14   data_t[0] = data_u|0x0C;  //en=1, rs=0
15   data_t[1] = data_u|0x08;  //en=0, rs=0
16   data_t[2] = data_l|0x0C;  //en=1, rs=0
17   data_t[3] = data_l|0x08;  //en=0, rs=0
18   HAL_I2C_Master_Transmit (&hi2c1, SLAVE_ADDRESS_LCD,(uint8_t *) data_t, 4, 100);
19 }
20
21 void lcd_send_data (char data)
22 {
23   char data_u, data_l;
```

```
  uint8_t data_t[4];
  data_u = (data&0xf0);
  data_l = ((data<<4)&0xf0);
  data_t[0] = data_u|0x0D;  //en=1, rs=0
  data_t[1] = data_u|0x09;  //en=0, rs=0
  data_t[2] = data_l|0x0D;  //en=1, rs=0
  data_t[3] = data_l|0x09;  //en=0, rs=0
  HAL_I2C_Master_Transmit (&hi2c1, SLAVE_ADDRESS_LCD,(uint8_t *) data_t, 4, 100);
}

void lcd_init (void) {
  lcd_send_cmd (0x33); /* set 4-bits interface */
  lcd_send_cmd (0x32);
  HAL_Delay(50);
  lcd_send_cmd (0x28); /* start to set LCD function */
  HAL_Delay(50);
  lcd_send_cmd (0x01); /* clear display */
  HAL_Delay(50);
  lcd_send_cmd (0x06); /* set entry mode */
  HAL_Delay(50);
  lcd_send_cmd (0x0c); /* set display to on */
  HAL_Delay(50);
  lcd_send_cmd (0x02); /* move cursor to home and set data address to 0 */
  HAL_Delay(50);
  lcd_send_cmd (0x80);
}

void lcd_send_string (char *str)
{
  while (*str) lcd_send_data (*str++);
}

void lcd_clear_display (void)
{
  lcd_send_cmd (0x01); //clear display
}

void lcd_goto_XY (int row, int col)
{
  uint8_t pos_Addr;
  if(row == 1)
  {
    pos_Addr = 0x80 + row - 1 + col;
  }
```

```
68    else
69    {
70      pos_Addr = 0xC0+ col;
71    }
72    lcd_send_cmd(pos_Addr);
73  }
74
75  void lcd_show_value(){
76      lcd_goto_XY(0, 0);
77      lcd_send_string("HUMID: ");
78      lcd_goto_XY(0, 8);
79      lcd_send_string(humid);
80      lcd_goto_XY(1, 0);
81      lcd_send_string("TEMP: ");
82      lcd_goto_XY(1, 8);
83      lcd_send_string(temp);
84  }
```

Listing 3: Mã nguồn file i2c_lcd.c

Mô tả:

- hi2c1: Biến được dùng để giao tiếp với LCD thông qua hàm HAL_I2C_Master_Transmit.
- SLAVE_ADDRESS_LCD: Địa chỉ I2C của module LCD, được định nghĩa là (0x21 « 1).
- temp, humid: Chuỗi ký tự lưu giá trị nhiệt độ (temp) và độ ẩm (humid) để hiển thị trên màn hình LCD.
- lcd_send_cmd(char cmd): gửi lệnh điều khiển đến LCD qua giao thức I2C. Các lệnh như: xóa màn hình, di chuyển con trỏ, cấu hình chế độ hiển thị.
- lcd_send_data(char data): Gửi một ký tự dữ liệu để hiển thị trên LCD.
- lcd_show_value(void): Hiển thị giá trị nhiệt độ và độ ẩm trên LCD. Gọi các hàm con như lcd_goto_XY() và lcd_send_string() để định vị và in chuỗi.

4.5 Hiện thực máy trạng thái LED báo hiệu và quạt

```
int status=INIT;
int status_fan=0;
int tempInt;
int humidInt;
int temp1=0;
void init_variables() {
  tempInt=atoi(temp);
  humidInt=atoi(humid);
}
TIM_HandleTypeDef htim1;
void fsm_system() {
    switch (status) {
        case INIT:
          dht20_output();
            status = LED_YELLOW;
            break;

        case LED_RED:
            if (timer_flag == 1) {
              dht20_output();
                if (0 < tempInt && tempInt < 27) {
                    status = LED_GREEN;
                } else if (27 <= tempInt && tempInt < 28) {
                    status = LED_YELLOW;
                }
            if(timer_flag_fan==1)
            {
               __HAL_TIM_SET_COMPARE(&htim1,TIM_CHANNEL_1,999);
               HAL_Delay(2);
            }
                 HAL_GPIO_WritePin(A_GPIO_Port, A_Pin, SET);
                 HAL_GPIO_WritePin(B_GPIO_Port, B_Pin, SET);
                 setTimer(500);
            }
            break;

        case LED_GREEN:
            if (timer_flag == 1) {
              dht20_output();
                if (tempInt >= 28) {
                    status = LED_RED;
                } else if (27 <= tempInt && tempInt < 28) {
```

```
                    status = LED_YELLOW;
                }
            if(timer_flag_fan==1)
            {
               __HAL_TIM_SET_COMPARE(&htim1,TIM_CHANNEL_1,0);
               HAL_Delay(2);
            }
                HAL_GPIO_WritePin(A_GPIO_Port, A_Pin, SET);
                HAL_GPIO_WritePin(B_GPIO_Port, B_Pin, RESET);
                setTimer(500);
            }
            break;

        case LED_YELLOW:
            if (timer_flag == 1) {
              dht20_output();
                if (0 < tempInt && tempInt < 27) {
                    status = LED_GREEN;
                } else if (tempInt >= 28) {
                    status = LED_RED;
                }
            if(timer_flag_fan==1) {
               __HAL_TIM_SET_COMPARE(&htim1,TIM_CHANNEL_1,499);
               HAL_Delay(2);
            }
                HAL_GPIO_WritePin(A_GPIO_Port, A_Pin, RESET);
                HAL_GPIO_WritePin(B_GPIO_Port, B_Pin, SET);
                setTimer(500);

            }
            break;
        default:
          break;
    }
}
```

Mô tả:

- LED: Hiển thị mức độ nhiệt độ (Đỏ = Cao, Xanh = Thấp, Vàng = Trung bình).
- Quạt: Điều chỉnh tốc độ tương ứng với mức nhiệt độ.
- init_variables(): Khởi tạo các biến tempInt và humidInt bằng cách chuyển đổi chuỗi nhiệt độ và độ ẩm (temp, humid) thành số nguyên.
- fsm_system(): Hàm chính điều khiển trạng thái của hệ thống.

4.6 Hiện thực giao tiếp UART từ STM32 với ESP32, đưa dữ liệu lên Web Server

4.6.1 Chi tiết nối dây

Để gửi biến nhiệt độ, ở chân PA2 (UART2_TX), PA3 (UART2_RX) của STM32 ta sẽ nối với lần lượt RXD_PIN (GIPO_NUM_5), TXD_PIN (GIPO_NUM_4) của ESP32.

Để gửi biến độ ẩm, ở chân PA9 (UART1_TX), PA10 (UART1_RX) của STM32 ta sẽ nối với lần lượt RXD2_PIN (GIPO_NUM_5), TXD2_PIN (GIPO_NUM_4) của ESP32.

4.6.2 Mã nguồn UART

```
HAL_UART_Transmit(&huart2, temp, sizeof(temp), 1000);
HAL_UART_Transmit(&huart1, humid, sizeof(humid), 1000);
```

```
  SCH\_AddTask(uart\_task, 0, 500);
```

Hàm uart_task với mục đích thêm vào task trong scheduler để thực hiện cụ thể được thực hiện lần đầu 0s còn các lần sau là 0.5s. Hai lệnh HAL_UART_Transmit sẽ truyền dữ liệu của temp và humid qua giao thức UART.

Mã nguồn nhận UART ESP32:

```
#define TXD_PIN (GPIO_NUM_17)
#define RXD_PIN (GPIO_NUM_16)

#define TXD2_PIN (GPIO_NUM_4)  // UART1 TX
#define RXD2_PIN (GPIO_NUM_5)  // UART1 RX
```

Chân 16 và 17 sẽ nhận dữ liệu UART từ STM32 cụ thể nó sẽ nhận dữ liệu temp. Chân 4 và 5 sẽ nhận dữ liệu UART từ STM32 cụ thể nó sẽ nhận dữ liệu humid.

```
Serial.begin(115200);

 // Config UART2
    const uart_port_t uart_num = UART_NUM_2;
    uart_config_t uart_config = {
        .baud_rate = 115200,
        .data_bits = UART_DATA_8_BITS,
        .parity = UART_PARITY_DISABLE,
        .stop_bits = UART_STOP_BITS_1,
        .flow_ctrl = UART_HW_FLOWCTRL_CTS_RTS,
        .rx_flow_ctrl_thresh = 122,
    };
    ESP_ERROR_CHECK(uart_param_config(uart_num, &uart_config));
    ESP_ERROR_CHECK(uart_set_pin(UART_NUM_2, TXD_PIN, RXD_PIN, UART_PIN_NO_CHANGE,
    UART_PIN_NO_CHANGE));
    const int uart_buffer_size = (1024 * 2);
```

```
    QueueHandle_t uart_queue;
    ESP_ERROR_CHECK(uart_driver_install(UART_NUM_2, uart_buffer_size, uart_buffer_size,
     10, &uart_queue, 0));

    // Config UART1
    const uart_port_t uart_num1 = UART_NUM_1;

    ESP_ERROR_CHECK(uart_param_config(uart_num1, &uart_config));
    ESP_ERROR_CHECK(uart_set_pin(UART_NUM_1, TXD2_PIN, RXD2_PIN, UART_PIN_NO_CHANGE,
    UART_PIN_NO_CHANGE));
    ESP_ERROR_CHECK(uart_driver_install(UART_NUM_1, uart_buffer_size, uart_buffer_size,
     10, &uart_queue, 0));

uint8_t data[128];
    int length = 0;
    ESP_ERROR_CHECK(uart_get_buffered_data_len(uart_num, (size_t*)&length));
    length = uart_read_bytes(uart_num, data, length, 100);
    if (length > 0) {
        char * test = (char*) data;
     value = strtof(test, nullptr);

    }

    // Read data from UART1
    const uart_port_t uart_num1 = UART_NUM_1;
    uint8_t data1[128];
    int length1 = 0;
    ESP_ERROR_CHECK(uart_get_buffered_data_len(uart_num1, (size_t*)&length1));
    length1 = uart_read_bytes(uart_num1, data1, length1, 100);
    if (length1 > 0) {
        char * test1 = (char*) data1;
     value1 = strtof(test1, nullptr);
    }
```

Giá trị temp và humid sẽ được lưu trong 2 biến value và value1 vì dữ liệu từ STM32 truyền qua là kiểu char [20] sau đó chúng em chuyển dữ liệu kiểu char sang kiểu float bằng hàm strtof(). Các hàm còn lại dùng để config và check lỗi dữ liệu UART truyền qua.

Code Web server:

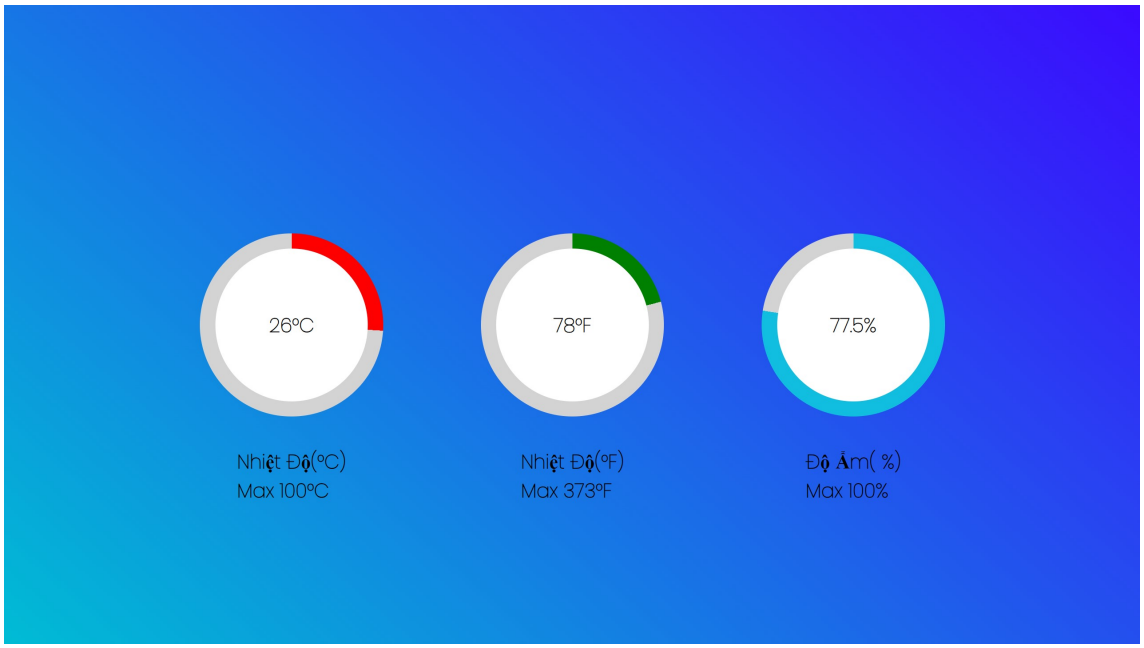
```
while (client.connected() && currentTime - previousTime <= timeoutTime) {
  currentTime = millis();
  if (client.available()) {
    char c = client.read();
    Serial.write(c);
    header += c;

    if (header.indexOf("GET /data") >= 0) {
      client.println("HTTP/1.1 200 OK");
      client.println("Content-Type: application/json");
      client.println("Connection: close");
      client.println();
      client.print("{\"tempC\":");
      client.print(value);
      client.print(", \"tempF\":");
      client.print((value * 9 / 5) + 32);
      client.print(", \"humidity\":");
      client.print(value1);
      client.println("}");
      break;
    }

    if (c == '\n') {
      if (currentLine.length() == 0) {
        client.println("HTTP/1.1 200 OK");
        client.println("Content-type:text/html");
        client.println("Connection: close");
        client.println();
        //  G i   m   HTML
        client.println("<!DOCTYPE html>");
        client.println("<html lang=\"en\">");
        client.println("<head>");
        client.println("  <meta charset=\"UTF-8\">");
        client.println("  <meta name=\"viewport\" content=\"width=device-width, initial
    -scale=1.0\">");
        client.println("  <link href=\"https://f...content-available-to-author-only...s
    .com/css2?family=Poppins:wght@100&display=swap\" rel=\"stylesheet\">");
        client.println("  <style>");
        client.println("    * { padding: 0; margin: 0; font-family: 'Poppins', sans-
    serif; }");
        client.println("    body { height: 100vh; background-image: linear-gradient(41
    deg, #00bcd4, #3b0aff); display: flex; justify-content: center; align-items: center
    ; }");
```

```
39          client.println("    :root { --size: 300px; --bord: 50px; --a: 60%; --b: 60%; --
        c: 60%; }");
40          client.println("    .DoC { width: var(--size); height: var(--size); margin: 1em
         auto; border-radius: 50%; background: conic-gradient(red var(--value), lightgrey
        var(--value)); position: relative; display: flex; justify-content: center; align-
        items: center; margin: 80px; }");
41          client.println("    .DoC::after { content: ''; position: absolute; left: 50%;
        top: 50%; transform: translate(-50%, -50%); width: calc(100% - var(--bord)); height
        : calc(100% - var(--bord)); background: white; border-radius: inherit; }");
42          client.println("     .DoF { width: var(--size); height: var(--size); margin: 1
        em auto; border-radius: 50%; background: conic-gradient(green var(--value),
        lightgrey var(--value)); position: relative; display: flex; justify-content: center
        ; align-items: center; margin: 80px; }");
43          client.println("    .DoF::after { content: ''; position: absolute; left: 50%;
        top: 50%; transform: translate(-50%, -50%); width: calc(100% - var(--bord)); height
        : calc(100% - var(--bord)); background: white; border-radius: inherit; }");
44          client.println("          .DoAm { width: var(--size); height: var(--size);
        margin: 1em auto; border-radius: 50%; background: conic-gradient(rgb(18, 190, 224)
        var(--value), lightgrey var(--value)); position: relative; display: flex; justify-
        content: center; align-items: center; margin: 80px; }");
45          client.println("    .DoAm::after { content: ''; position: absolute; left: 50%;
        top: 50%; transform: translate(-50%, -50%); width: calc(100% - var(--bord)); height
        : calc(100% - var(--bord)); background: white; border-radius: inherit; }");
46          client.println("    .inner-head { display: flex; }");
47          client.println("    .text { position: absolute; top: 350px; }");
48          client.println("    p { position: relative; z-index: 1; font-size: 2em; font-
        weight: 900; }");
49          client.println("    .DoC { --value: var(--a); }");
50          client.println("    .DoF { --value: var(--b); }");
51          client.println("    .DoAm { --value: var(--c); }");
52          client.println("    .b-skills { padding-top: 46px; text-align: center; }");
53          client.println("    .b-skills:last-child { margin-bottom: -30px; }");
54          client.println("    .b-skills h2 { margin-bottom: 50px; font-weight: 900; text-
        transform: uppercase; }");
55          client.println("    .skill-item { max-width: 250px; width: 100%; margin-bottom:
         30px; color: black; position: relative; }");
56          client.println("    .chart-container { position: relative; width: 100%; padding
        -top: 100%; margin-bottom: 27px; }");
57          client.println("    .skill-item .chart, .skill-item .chart canvas { position:
        absolute; top: 0; left: 0; width: 100% !important; height: 100% !important; }");
58          client.println("    .skill-item .chart:before { content: ''; width: 0; height:
        100%; display: inline-block; vertical-align: middle;}");
59          client.println("    .skill-item .percent { display: inline-block; vertical-
        align: middle; line-height: 1; font-size: 40px; font-weight: 900; position:
```

```
relative; z-index: 2; }");
60          client.println("    .skill-item .percent:after { content: attr(data-after);
font-size: 20px; }");
61          client.println("    p { font-weight: 900; }");
62          client.println("  </style>");
63          client.println("</head>");
64          client.println("<body>");
65          client.println("  <div class=\"inner-head\">");
66          client.println("    <div class=\"chart DoC\">");
67          client.println("      <p id='tempC'></p>");
68          client.println("      <p class=\"text\"> N h i t       (&deg;C)<br>Max 100&deg;C
</p>");
69          client.println("    </div>");
70          client.println("    <div class=\"chart DoF\">");
71          client.println("      <p id='tempF'></p>");
72          client.println("      <p class=\"text\"> N h i t       (&deg;F)<br>Max 373&deg;F
</p>");
73          client.println("    </div>");
74          client.println("    <div class=\"chart DoAm\">");
75          client.println("      <p id='humidity'></p>");
76          client.println("      <p class=\"text\">        m  ( %)<br>Max 100%</p>");
77          client.println("    </div>");
78          client.println("  </div>");
79          client.println("  <script>");
80          client.println("    function updateValues() {");
81          client.println("      fetch('/data')");
82          client.println("        .then(response => response.json())");
83          client.println("        .then(data => {");
84          client.println("          document.getElementById('tempC').innerText = data.
tempC + ' C ';");
85          client.println("          document.getElementById('tempF').innerText = Math.
floor((data.tempC * 9/5) + 32) + ' F ';");
86          client.println("          document.getElementById('humidity').innerText = data.
humidity + '%';");
87          client.println("          document.documentElement.style.setProperty('--a',
data.tempC + '%');");
88          client.println("          document.documentElement.style.setProperty('--b',
Math.floor((data.tempC * 9/5) + 32) / 3.73 + '%');");
89          client.println("          document.documentElement.style.setProperty('--c',
data.humidity + '%');");
90          client.println("        });");
91          client.println("    }");
92          client.println("    setInterval(updateValues, 500);");
93          client.println("  </script>");
```

```
94          client.println("</body>");
95          client.println("</html>");
96          client.println();
97          break;
98        } else {
99          currentLine = "";
100       }
101     } else if (c != '\r') {
102       currentLine += c;
103     }
104   }
105 }
106     header = "";
107     client.stop();
108     Serial.println("Client disconnected.");
109     Serial.println("");
110   }
111 }
```

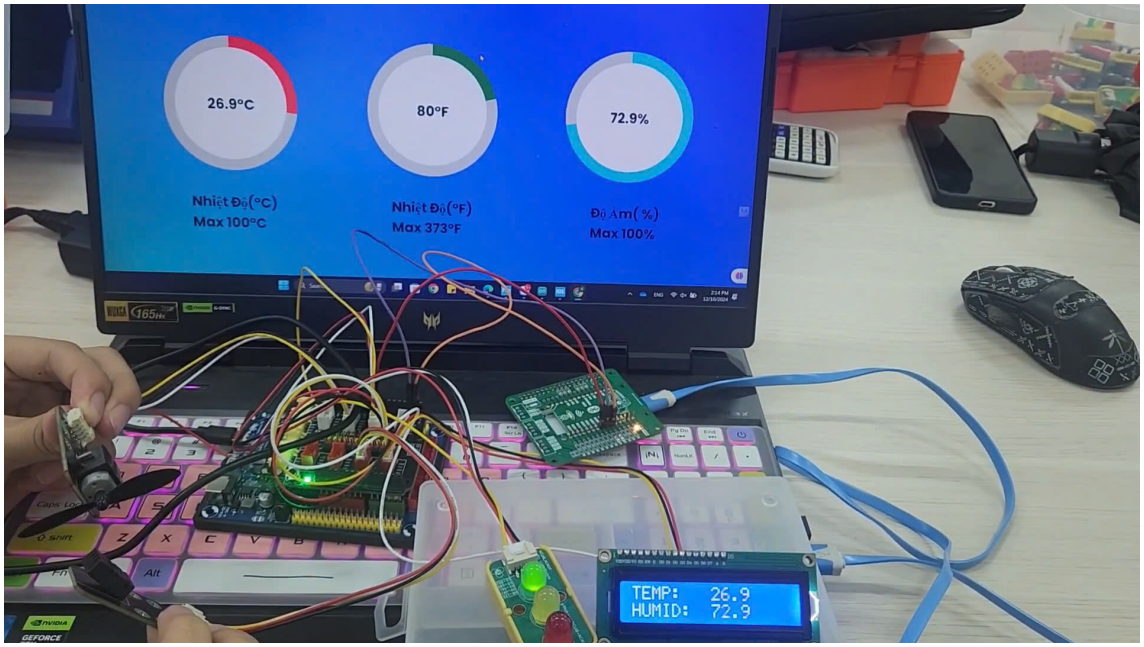
- Kết nối Client: Mã bắt đầu bằng cách kiểm tra xem client có còn kết nối hay không và thời gian đã trôi qua có nằm trong giới hạn cho phép không.
- Đọc Dữ liệu: Nếu có dữ liệu từ client, chương trình sẽ đọc từng ký tự và lưu trữ chúng vào một biến để xử lý.
- Xử lý Yêu cầu:
  - Nếu yêu cầu là GET /data, chương trình sẽ gửi một phản hồi với dữ liệu JSON chứa các thông số như nhiệt độ (C và F) và độ ẩm.
  - Nếu nhận được dấu hiệu yêu cầu trang chính (khi gặp ký tự newline), chương trình sẽ gửi một trang HTML để hiển thị dữ liệu cảm biến một cách trực quan.
- HTML và CSS: Trang HTML được tạo ra có chứa các phần tử hiển thị nhiệt độ và độ ẩm dưới dạng biểu đồ tròn. CSS được sử dụng để định dạng giao diện.
- Cập nhật Dữ liệu: Một hàm JavaScript sẽ được gọi định kỳ để lấy dữ liệu mới từ server và cập nhật trang mà không cần tải lại trang.
- Kết thúc Kết nối: Sau khi hoàn tất việc xử lý yêu cầu, chương trình sẽ đóng kết nối với client và reset các biến cần thiết.



Hình 18: Giao diện trang web sau khi hoàn thành

4.7 Hiện thực trên mạch thật

Chúng ta nối dây như sơ đồ đã được mô tả ở phần trước và thực hiện nạp code. Kết qảu như hình dưới đây:



Hình 19: Ảnh hiện thực trên mạch thật

5 Video thuyết trình và demo

Video thuyết trình và demo của nhóm chúng em: https://youtu.be/Wy_bLwYr1zo?si=JuLFcX4sUbJa5E91

6 Mã nguồn của đồ án

Link github của nhóm chúng em: https://github.com/leduccuonghcmut/DA241

7 Kết luận

Trong đề tài này, nhóm chúng em thiết kế và phát triển device driver giám sát và điều khiển nhiệt độ, độ ẩm với sự giao tiếp chính giữa STM32 và ESP32. Hệ thống sử dụng cảm biến DHT20 để thu thập dữ liệu nhiệt độ và độ ẩm, truyền dữ liệu qua giao tiếp UART, và hiển thị thông tin cũng như điều khiển quạt thông qua ESP32 kết nối với web server qua giao thức HTTP.

Qua quá trình demo trên mạch thật, nhóm chúng em đã kiểm chứng hệ thống hoạt động ổn định và đáp ứng đầy đủ các chức năng yêu cầu. Cụ thể, dữ liệu nhiệt độ và độ ẩm được cảm biến DHT20 thu thập chính xác, hiển thị rõ ràng trên màn hình LCD 16x2, đồng thời dữ liệu này được truyền qua ESP32 và đưa lên web server. Hệ thống cũng thực hiện tốt việc điều khiển quạt và LED báo hiệu trạng thái dựa trên các ngưỡng nhiệt độ được thiết lập.

Tuy nhiên, nhóm chúng em nhận thấy vẫn còn một số điểm cần cải thiện, như tối ưu hiệu suất driver và mở rộng khả năng kết nối với các thiết bị ngoại vi khác. Đây sẽ là định hướng để nhóm phát triển thêm trong tương lai.

Nhóm chúng em xin chân thành cảm ơn thầy!

Tài liệu tham khảo

[1] Slide bài giảng môn học CO3009 - Vi xử lý và vi điều khiển.

[2] Lập trình STM32 trên STM32CubeIDE, thầy Lê Trọng Nhân. Truy cập từ:
https://www.youtube.com/watch?v=ZuBKAAFxjgk&list=PLyD_mbw_VznOxKMBg5CjJ6OWvDzPNR8F4

[3] STM32F103RB Datasheet, Truy cập từ: https://s.net.vn/61vK

[4] ESP-AT Command Firmware, Espressif Systems. Truy cập từ:
https://www.espressif.com/en/products/sdks/esp-at/overview