ĐẠI HỌC BÁCH KHOA HÀ NỘI
TRƯỜNG ĐIỆN – ĐIỆN TỬ

BÁO CÁO BÀI TẬP LỚN

MÔN THIẾT KẾ HỆ THỐNG NHÚNG

NHÓM 9

Ngành KT Điều khiển & Tự động hóa

Thành viên nhóm: Bùi Quốc Doanh 20212715
Lê Thái Phát 20210667
Nguyễn Trung Dũng 20212723

Giảng viên hướng dẫn: TS. Lê Công Cường

Khoa: Tự động hóa
Trường: Điện – Điện tử
Học phần: Thiết kế hệ thống nhúng
Mã lớp: 158075

HÀ NỘI, 7/2025

MỤC LỤC

CHƯƠNG 1. GIỚI THIỆU CHUNG

1.1. Thông tin nhóm

Họ và tên: Bùi Quốc Doanh MSSV: 20212715 Phụ trách công việc: Tìm hiểu lí thuyết các mô hình đơn nhiệm và mô hình đa nhiệm Co-operative Scheduling từ đó xây dựng các mô hình tương ứng. Triển khai trên phần cứng thực tế và đánh giá kết quả.
Họ và tên: Nguyễn Trung Dũng MSSV: 20212723 Phụ trách công việc: Tìm hiểu lí thuyết mô hình đa nhiệm Pre-emptive Scheduling từ đó thiết kế, xây dựng mô hình đa nhiệm phù hợp với yêu cầu dự án. Triển khai trên phần cứng thực tế và đánh giá kết quả.
Họ và tên: Lê Thái Phát MSSV: 20210667 Phụ trách công việc: Tìm hiểu nguyên lí làm việc các module phần cứng, chuẩn bị phần cứng. Tìm hiểu nguyên lí các ngoại vi của STM32 sử dụng và tìm hiểu lí thuyết mô hình đơn nhiệm.

1.2. Yêu cầu bài tập lớn

Đề tài: Lập trình VĐK STM32 đọc cảm biến nhiệt độ, độ ẩm, điều khiển động cơ, hiện thị thông số lên LCD đồng thời truyền thông số lên máy tính. Cài đặt được thông số lập lịch và nhận lệnh điều khiển từ máy tính.

Mô tả đề tài: Bài tập lớn yêu cầu xây dựng một hệ thống nhúng trên nền vi điều khiển STM32 có khả năng thu thập dữ liệu nhiệt độ/độ ẩm từ cảm biến AHT10. Hệ thống cũng có nhiệm vụ điều khiển động cơ DC thông qua tín hiệu PWM để bật/tắt hoặc điều chỉnh tốc độ và chiều quay. Các thông số đo được sẽ hiển thị lên màn hình LCD, đồng thời truyền về máy tính thông qua giao tiếp UART. Ngoài ra, hệ thống hỗ trợ cài đặt thông số lập lịch cho các tác vụ và nhận lệnh điều khiển trực tiếp từ máy tính.

CHƯƠNG 2. GHÉP NỐI PHẦN CỨNG

2.1. Nguyên lý làm việc các module phần cứng

2.1.1. Khối vi điều khiển MCU

STM32 là một trong những dòng chip phổ biến của ST với nhiều dọ thông dụng như F0, F1, F2, F3… STM32F103C8T6 thuộc họ F1 với lõi ARM CortexM3. STM32F103C8T6 là vi điều khiển 32 bit, có tốc độ tối đa 72 MHz. Giá thành cũng khá ổn so với các loại vi điều khiển có chức năng tương tự. Mạch nạp cũng như công cụ lập trình khá đa dạng và dễ sử dụng.

- Chip: ARM 32-bit Cortex M3
- Điện áp hoạt động: 2.0 – 3.6 V
- Bộ nhớ: 64 Kb Flash, 20 Kb SRAM
- Chế độ tiết kiệm năng lượng: Sleep, Stop, Standby
- 2x12 bit, 1 µs ADC
- Kiểu chân: LQFP-48
- Watchdog: WWDG, IWDG
- 7 kênh DMA
- Timer: 4 bộ, 1 bộ nâng cao, 3 bộ đa dụng, 2 bộ Watchdog, 1 bộ Systick
- Kết nối: 3xUSART, 2xI2C, 2xSPI, USB 2.0, 1xCAN

Hình 2.1. *Chip STM32F103C8T6*

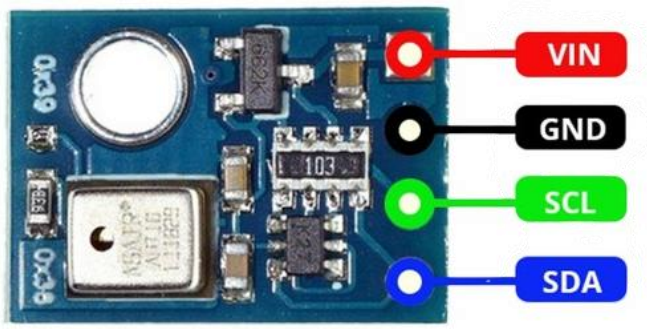
Hình 2.2. *Kit phát triển STM32F103C8T6*

2.1.2. Module cảm biến AHT10

a) Nguyên lí hoạt động

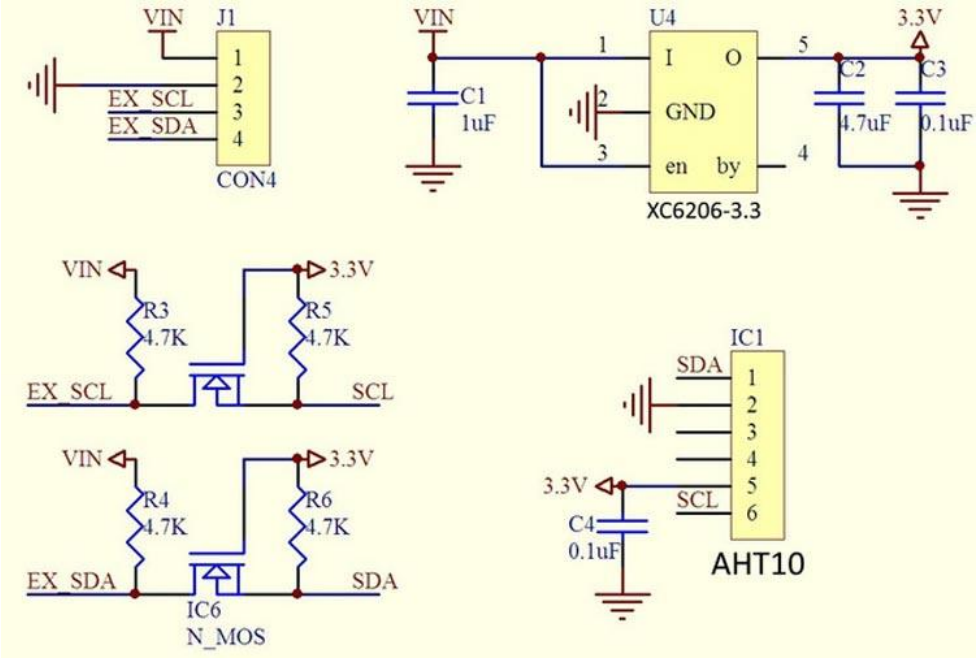
Cảm biến AHT10 hoạt động dựa trên nguyên lý điện dung, trong đó độ ẩm môi trường sẽ làm thay đổi giá trị điện dung của cảm biến. Sự thay đổi điện dung này được cảm biến chuyển đổi thành tín hiệu số thông qua bộ chuyển đổi ADC

tích hợp bên trong. Nhiệt độ được đo dựa trên nguyên lý bán dẫn với độ tuyến tính cao và độ nhạy tốt, cung cấp giá trị nhiệt độ dưới dạng số thông qua giao thức truyền thông I2C. Vi điều khiển STM32 sẽ giao tiếp với cảm biến qua I2C, gửi lệnh yêu cầu dữ liệu và nhận lại dữ liệu đo được, sau đó tính toán để hiển thị giá trị nhiệt độ và độ ẩm chính xác.



Hình 2.3. *Pinout module cảm biến nhiệt độ & độ ẩm AHT10*

b) Thông số kĩ thuật



Hình 2.4. *Hình 2.2. Sơ đồ nguyên lí của module AHT10*

- Điện áp hoạt động: 1.8V đến 3.6V (điển hình 3.3V).
- Dải đo độ ẩm: 0 - 100% RH, sai số ±2% RH.
- Dải đo nhiệt độ: -40°C đến 85°C, sai số ±0.3°C.
- Giao tiếp: I2C chuẩn, địa chỉ mặc định 0x38.
- Thời gian phản hồi dữ liệu: < 75ms.

2.1.3. Động cơ DC12V

Động cơ DC giảm tốc V1 1:48 hộp số kim loại có trục quay và bánh răng của hộp số được làm bằng kim loại cho tuổi thọ và độ bền cao hơn các loại bằng nhựa (các loại bằng nhựa khi chạy 1 thời gian sẽ bị tình trạng các bánh răng nhựa bị rơ,

kẹt khiến cho vận tốc động cơ thay đổi theo thời gian), thích hợp để lắp ráp các mô hình Robot, Cơ khí đơn giản.

Hình 2.5. *Hình 2.3. Động cơ DC12V giảm tốc V1*

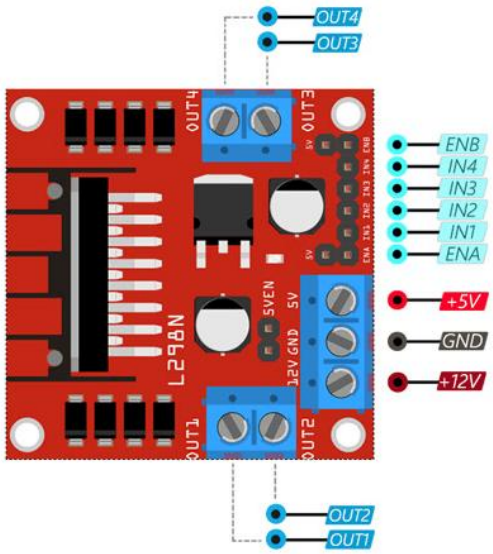
Thông số kĩ thuật:

- Điện áp hoạt động: 3V~ 9V DC (Hoạt động tốt nhất từ 6 – 8V)
- Dòng không tải: 70mA (250mA MAX)
- Mômen xoắn cực đại: 800gf cm min 1:48 (3V)
- Tốc độ không tải: 125 Vòng/ 1 Phút (3V) (Với bánh 66mm: 26m/1p) 208 Vòng/ 1 Phút (5V) (Với bánh 66mm: 44m/1p)

2.1.4. Module L298N điều khiển động cơ DC

a) Nguyên hoạt động

Module L298N là mạch điều khiển động cơ DC dựa trên IC điều khiển cầu H (H-Bridge) tích hợp. Nó cho phép điều khiển tốc độ và hướng quay của động cơ thông qua việc cung cấp tín hiệu PWM và tín hiệu logic từ vi điều khiển STM32. Nguyên lý làm việc của L298N dựa trên việc đóng mở các transistor bên trong IC để đảo chiều dòng điện cung cấp cho động cơ, từ đó điều chỉnh tốc độ và chiều quay động cơ. Khi tín hiệu PWM từ STM32 được cấp tới module driver, module sẽ điều chỉnh dòng điện cung cấp vào động cơ, từ đó thay đổi tốc độ quay của động cơ. Module này cũng hỗ trợ đảo chiều quay của động cơ bằng cách đổi trạng thái của các chân điều khiển logic.



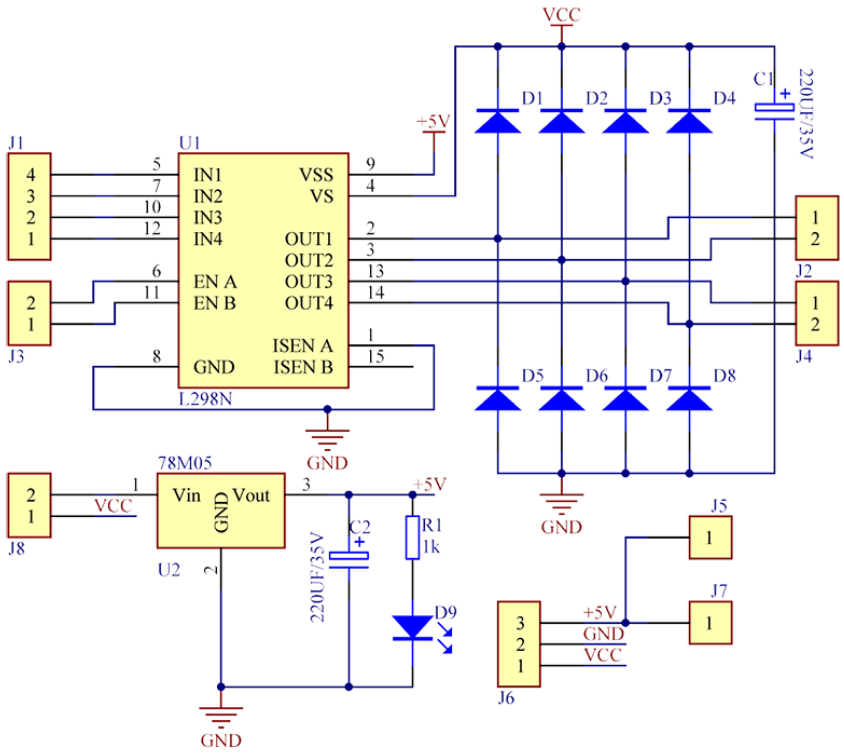
Hình 2.6. *Module điều khiển động cơ DC L298N*

Sử dụng IC L298 là một IC tích hợp nguyên khối gồm 2 mạch cầu H bên trong. Với điện áp làm tăng công suất nhỏ như động cơ DC loại vừa. Chức năng các chân của L298:

- 4 chân INPUT: IN1, IN2, IN3, IN4 được nối lần lượt với các chân 5, 7, 10, 12 của L298. Đây là các chân nhận tín hiệu điều khiển.
- 4 chân OUTUT: OUT1, OUT2, OUT3, OUT4 (tương ứng với các chân INPUT) được nối với các chân 2, 3, 13, 14 của L298. Các chân này sẽ được nối với động cơ.
- Hai chân ENA và ENB dùng để điều khiển mạch cầu H trong L298. Nếu ở mức logic “1” (nối với nguồn 5V) cho phép mạch cầu H hoạt động, nếu ở mức logic “0” thì mạch cầu H không hoạt động.

Với bài toán điều khiển chiều quay của động cơ chỉ cần lưu ý đến cách điều khiển chiều quay với L298:

- Khi ENA = 0: Động cơ không quay với mọi đầu vào.
- Khi ENA = 1, INT1 = 1, INT2 = 0: Động cơ quay thuận.
- Khi INT1 = 0, INT2 = 1: Động cơ quay nghịch. INT1 = INT2: Động cơ dùng ngay tức thì.
- Với ENB cũng tương tự với INT3, INT4.



Hình 2.7. *Sơ đồ nguyên lí module L298N*

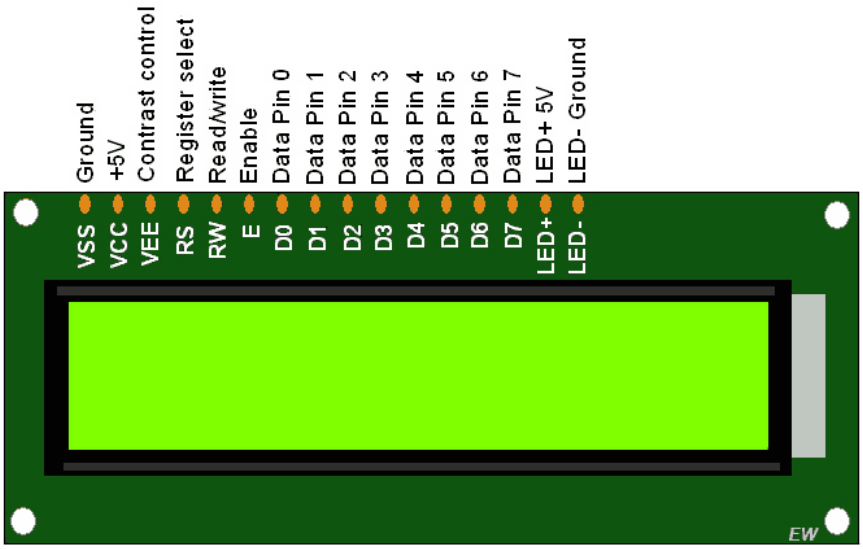
b) Thông số kĩ thuật

- Điện áp cấp nguồn cho động cơ: 5V - 35V.
- Dòng điện tối đa mỗi kênh: 2A (có thể lên đến 3A với tản nhiệt).
- Điện áp tín hiệu logic đầu vào: 5V.
- Tần số PWM điều khiển khuyến nghị: 1kHz - 10kHz.
- Nhiệt độ hoạt động: -25°C đến +130°C.

2.1.5. Màn hình LCD

a) Nguyên lí hoạt động

Màn hình LCD ký tự (HD44780): LCD ký tự HD44780 là loại màn hình tinh thể lỏng dùng để hiển thị ký tự, số và một số ký hiệu đặc biệt. Nó hoạt động dựa trên giao tiếp song song hoặc giao tiếp 4 bit để giao tiếp với vi điều khiển. Bên trong màn hình này gồm một bộ nhớ hiển thị DDRAM (Display Data RAM) chứa các ký tự cần hiển thị và bộ nhớ CGROM (Character Generator ROM) lưu trữ bộ ký tự cố định. Khi dữ liệu được gửi từ vi điều khiển thông qua các chân dữ liệu (D4-D7) cùng với tín hiệu điều khiển (RS, RW, E), màn hình sẽ thực hiện ghi dữ liệu vào DDRAM và chuyển đổi các dữ liệu này thành ký tự hiển thị. Tín hiệu Enable (E) đóng vai trò kích hoạt việc truyền dữ liệu từ vi điều khiển sang LCD.



Hình 2.8. *Màn hình LCD 1602*

b) Cấu tạo và chân kết nối

- Chân 1 (VSS): GND
- Chân 2 (VCC): +5V
- Chân 3 (Vo): Điều chỉnh độ tương phản (kết nối potentiometer 10kΩ)
- Chân 4 (RS): Chọn thanh ghi Command (0) hoặc Data (1)
- Chân 5 (RW): Chọn Read (1) hoặc Write (0)
- Chân 6 (E): Enable, xung để ghi dữ liệu
- Chân 7–14 (D0–D7): Dữ liệu 8-bit (khi dùng 4-bit chỉ sử dụng D4–D7)
- Chân 15 (LED+): Anode đèn nền (thường +5V qua điện trở)
- Chân 16 (LED–): Cathode đèn nền (GND)

c) Thông số kĩ thuật

- Điện áp hoạt động: 4.7V – 5.3V
- Dòng tiêu thụ (không đèn nền): ~1–2 mA
- Dòng tiêu thụ (đèn nền): ~15–20 mA
- Tần số PWM hỗ trợ: lên đến 100 Hz cho đèn nền
- Nhiệt độ hoạt động: 0°C – 50°C
- Kích thước module: ~80 × 36 × 12 mm
- Độ tương phản có thể điều chỉnh thông qua biến trở nối vào chân Vo

2.1.6. Module giao giao tiếp LCD ký tự 1602 sang I2C

a) Nguyên lí hoạt động

Module I2C chuyển đổi tín hiệu từ giao thức song song của LCD 1602 sang giao thức I2C. Chip PCF8574 hoạt động như một bộ mở rộng I/O, nhận dữ liệu từ vi điều khiển qua bus I2C và điều khiển các chân của LCD 1602. Dữ liệu được truyền qua hai đường SDA (Serial Data) và SCL (Serial Clock), giúp giảm đáng kể số lượng dây kết nối.

Quy trình hoạt động:

- Vi điều khiển gửi dữ liệu qua bus I2C đến chip PCF8574.

- PCF8574 chuyển đổi dữ liệu thành tín hiệu điều khiển LCD 1602.
- LCD hiển thị các ký tự hoặc thông tin theo yêu cầu.

Hình 2.9. *Module giao tiếp LCD ký tự sang I2C*

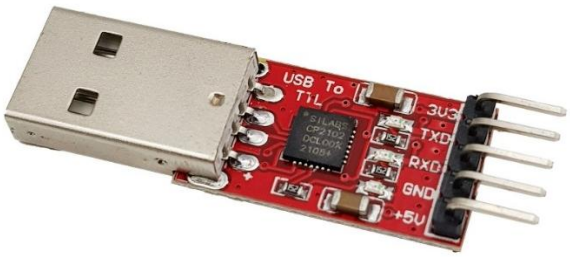
b) Thông số kĩ thuật

- Kích thước: 41.5 x 19 x 15.3mm
- Trọng lượng: 5g
- Màu PCB: Đen
- Điện áp cung cấp: 2,5-6V
- Hỗ trợ giao tiếp I2C
- Địa chỉ mặc định: 0X27 (có thể điều chỉnh bằng ngắn mạch chân A0/A1/A2)
- Tích hợp Jump chốt để cung cấp đèn cho LCD hoặc ngắt
- Tích hợp biến trở xoay điều chỉnh độ tương phản cho LCD

2.1.7. Module giao tiếp UART/USB (CP2102)

a) Nguyên lí hoạt động

- **Nhận dữ liệu UART:** CP2102 có khối UART nội, nhận tín hiệu TTL (TXD, RXD) từ STM32.
- **Chuyển đổi sang USB:** Dữ liệu UART sau đó được đưa vào bộ điều khiển USB (USB controller) tích hợp sẵn trong CP2102, đóng gói theo gói USB (USB packet) tuân theo chuẩn CDC (Communication Device Class).
- **Giao tiếp USB với máy tính:** Gói USB chứa dữ liệu UART được truyền qua cổng USB lên máy tính. Trình điều khiển (driver) CP2102 trên máy tính sẽ nhận gói và tạo ra một cổng ảo (COM port trên Windows hoặc /dev/ttyUSBx trên Linux).
- **Ngược lại:** Khi máy tính gửi dữ liệu qua cổng ảo USB, CP2102 giải gói USB, chuyển đổi trở lại tín hiệu UART TTL rồi xuất ra chân TXD đến STM32.
- **Quản lý luồng dữ liệu và ngắt:** CP2102 hỗ trợ tính năng FIFO và ngắt (interrupt) để báo hiệu cho vi điều khiển hay máy tính biết khi có dữ liệu sẵn sàng.



Hình 2.10. *Module USB/UART CP2102*

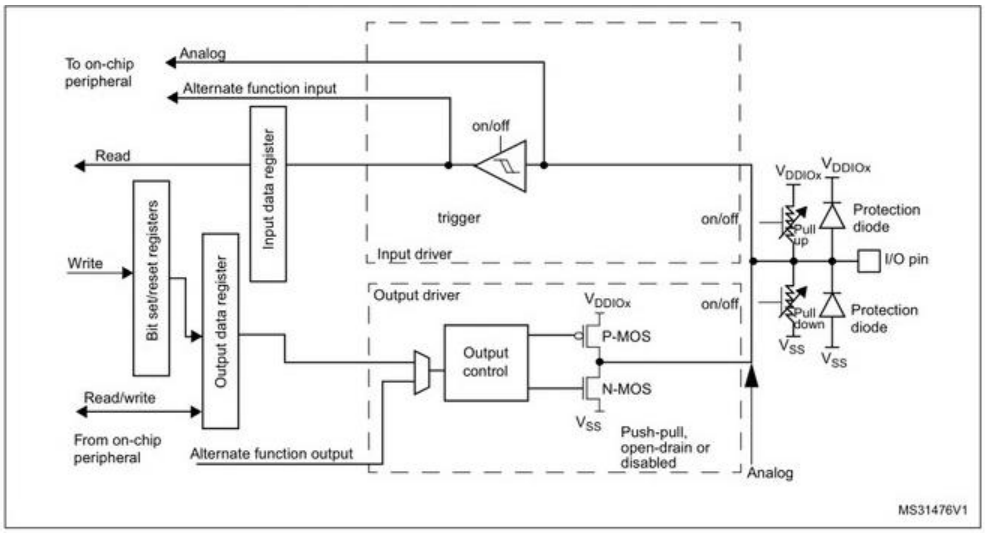
b) Thông số kĩ thuật

- 3.3V: Chân nguồn 3.3VDC (dòng cấp rất nhỏ tối đa 100mA), không sử dụng để cấp nguồn, thường chỉ sử dụng để thiết đặt mức tín hiệu Logic.
- TXD: chân truyền dữ liệu UART TTL (3.3VDC), dùng kết nối đến chân nhận RX của các module sử dụng mức tín hiệu TTL 3.3~5VDC.
- RXD: chân nhận dữ liệu UART TTL (3.3VDC), dùng kết nối đến chân nhận TX của các module sử dụng mức tín hiệu TTL 3.3~5VDC.
- GND: chân mass hoặc nối đất.
- 5V: Chân cấp nguồn 5VDC từ cổng USB, tối đa 500mA.

2.2. Các ngoại vi của STM32 được sử dụng

2.2.1. GPIO (General Purpose Input/Output - Đầu Vào/Ra Đa Năng)

Điều khiển các chân tín hiệu để giao tiếp với các thành phần như màn hình LCD, cảm biến. STM32 GPIO được cấu tạo như sau:



Hình 2.11. *Cấu trúc cơ bản của 1 cổng I/O*

STM32 GPIO bao gồm 2 khối cơ bản:

- Input Driver: Bao gồm thanh ghi Input Data (IDR), và 1 trigger. Tín hiệu Input ngoài việc được ghi vào IDR còn theo các đường Analog để vào bộ ADC, hoặc theo đường Alternate function input vào các ngoại vi khác
- Output Drive: Bao gồm thanh ghi Output Data (ODR), một khối output control để chọn tín hiệu ra là từ ODR hay từ các ngoại vi khác. Tiếp đến điều khiển 2 mosfet cho điện áp ra ở I/O pin.

Chức năng của STM32 GPIO bao gồm:

- Input pull up: Đầu vào có trở kéo lên (điện áp mặc định trên chân là Vcc).
- Input pull down: Đầu vào có trở kéo xuống (điện áp mặc định trên chân là 0V).
- Input floating: Đầu vào thả nổi, điện áp không cố định dao động từ 0V tới Vcc.
- Analog: Đầu vào tương tự, dùng để đo ADC.
- Ouput Push Pull: Đầu ra dạng đẩy kéo, tín hiệu sẽ chỉ có Vcc hoặc 0V tương ứng với Bit 1 và 0 ghi vào chân đó.
- Ouput Opendrain: Đầu ra dạng cực máng hở. Chỉ có thể kéo về 0V bằng cách ghi bit 0, khi ghi bit 1, chân IO sẽ có điện áp tương ứng với nguồn nối vào IO đó.
- Alternate function Push Pull: Đầu ra kểu đẩy kéo sử dụng trong các ngoại vi.
- Alternate function Open Drain: Đầu ra dạng cực máng hở, sử dụng trong các ngoại vi (thường gặp trong I2C).

2.2.2. Timer STM32 (PWM)

a) Giới thiệu về Timer STM32

Timer là ngoại vi quan trọng dùng để tạo tín hiệu thời gian, điều khiển PWM, bắt xung đầu vào hoặc tạo ngắt theo thời gian. Trên STM32, có ba nhóm timer chính:

Basic Timers (TIM6, TIM7)

- Chức năng chính: tạo ngắt định kỳ hoặc khởi tạo DAC.
- Không hỗ trợ Capture/Compare hay PWM.

General-Purpose Timers (TIM2–TIM5, TIM9–TIM14)

- Hỗ trợ đếm ngược, đếm xuôi, one-pulse mode, center-aligned mode.
- Có Capture/Compare cho PWM, Input Capture, Output Compare, Encoder Interface.

Advanced Timers (TIM1, TIM8)

- Thêm tính năng dead-time insertion, break input (bảo vệ quá dòng), complementary PWM cho điều khiển motor bridge.

b) Cấu trúc của 1 timer

- Prescaler (PSC): chia tần số xung clock (PCLK1 hoặc PCLK2) để điều chỉnh tốc độ đếm của CNT.
- Auto-Reload Register (ARR): giới hạn giá trị đếm cao nhất, khi CNT chạm ARR thì sẽ reset về 0 (hoặc giá trị ARR nếu đếm xuống), đồng thời phát ngắt (update event).
- Counter (CNT): bộ đếm chính, chạy theo giá trị PSC và ARR.
- Capture/Compare Register (CCRx): lưu giá trị để so sánh với CNT, dùng cho các chế độ PWM hoặc đo độ rộng xung.

c) Chế độ hoạt động

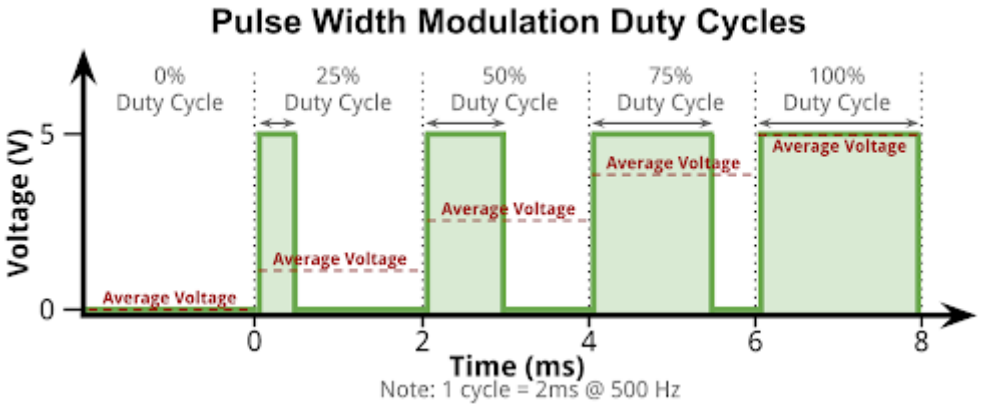
- Up-counting / Down-counting: CNT đếm lên từ 0→ARR hoặc đếm xuống từ ARR→0.
- Center-aligned mode: CNT đếm lên đến ARR rồi đếm xuống, cho sóng PWM đối xứng.
- One-pulse mode: sinh ra một xung duy nhất với độ rộng tính theo PSC và ARR.
- Encoder interface: sử dụng hai chân input (TI1, TI2) để đọc trực tiếp tín hiệu từ encoder quay.

d) Ứng dụng chính

- PWM generation: cấu hình CCRx để chia pha duty-cycle, điều khiển tốc độ động cơ.
- Input capture: đo chu kỳ và độ rộng xung vào bằng cách ghi CNT vào CCRx khi phát hiện cạnh tín hiệu.
- Output compare: tạo ngắt hoặc thay đổi trạng thái chân khi CNT bằng giá trị CCRx.
- Time base và ngắt định kỳ: dùng basic timer để tạo ngắt Hệ thống Tick hoặc đồng hồ thời gian thực.

e) Timer PWM

PWM hay Pulse Width Modulation là phương pháp điều chỉnh độ rộng của xung có chu kì cố định, nhằm tạo ra sự thay đổi điện áp tại đầu ra.



Hình 2.12. *Điều chế độ rộng xung PWM*

PWM ứng dụng nhiều trong việc điều khiển động cơ, các bộ nguồn xung boot, buck, nghịch lưu 1 pha, 3 pha…Trong mỗi Timer có 4 kênh độc lập phát PWM. Chu kì xung của PWM được quản lý bằng thanh ghi PSC và thanh ghi ARR. Duty Cycles được quản lý bằng thanh ghi CCR.

2.2.3. I2C (Inter- Integrated Circuit)

a) Giới thiệu giao thức I2C

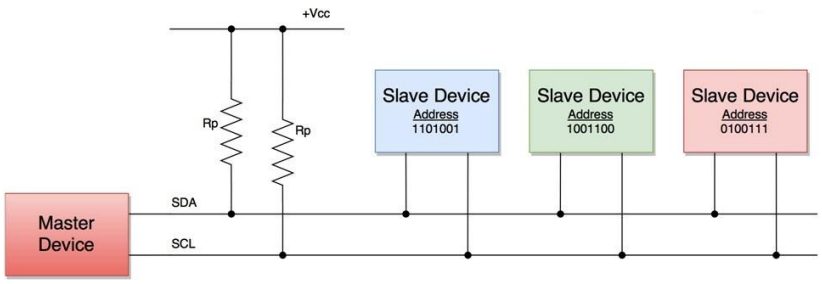
I2C viết tắt của Inter- Integrated Circuit là một phương thức giao tiếp được phát triển bởi hãng Philips Semiconductors. Dùng để truyền tín hiệu giữa vi xử lý và các IC trên các bus nối tiếp.

Đặc điểm:

- Tốc độ không cao
- Thường sử dụng onboard với đường truyền ngắn
- Nối được nhiều thiết bị trên cùng một bus
- Giao tiếp đồng bộ, sử dụng Clock từ master
- Sử dụng 7-bit hoặc 10-bit địa chỉ
- Chỉ sử dụng 2 chân tín hiệu SDA, SCL
- Có 2 tốc độ tiêu chuẩn là Standard mode (100 kb/s) và Low mode (10 kbit/s)

b) Kết nối vật lí của giao thức I2C

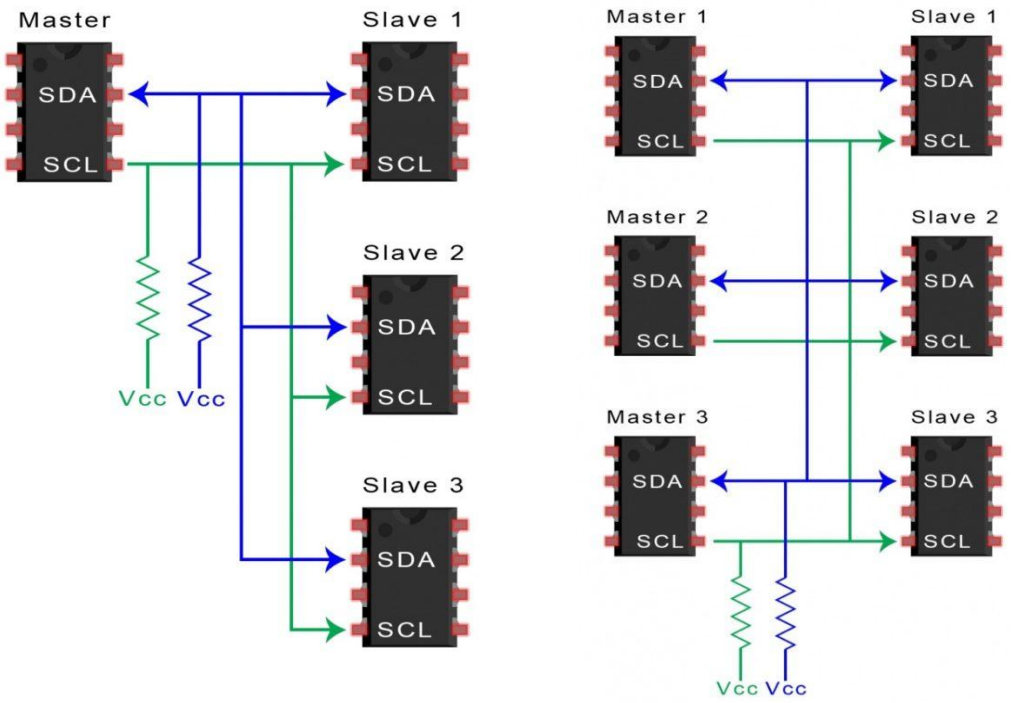
Bus I2C sử dụng 2 dây tín hiệu là SDA (Serial Data Line) và SCL (Serial Clock Line). Dữ liệu truyền trên SDA được đồng bộ với mỗi xung SCL. Đường SCL chỉ master mới có quyền điều khiển. Tất cả các thiết bị đều dùng chung 2 đường tín hiệu này.



Hình 2.13. *Kết nối vật lí của giao thức I2C*

Hai đường bus SDA và SCL hoạt động ở chế độ Open Drain hay cực máng hở. Nghĩa là tất cả các thiết bị trong mạng đều chỉ có thể lái 2 chân này về 0 chứ ko thể kéo lên 1. Để tránh việc sảy ra ngắn mạch khi thiết bị này kéo lên cao, thiết bị kia kéo xuống thấp. Để giữ mức logic là 1 ở trạng thái mặc định phải mắc thêm 2 điện trở treo lên Vcc (thường từ 1k – 4k7).

- Mỗi Bus I2C sẽ có 3 chế độ chính:
- Một Master, nhiều Slave
- Nhiều master, nhiều Slave
- Một Master, một Slave



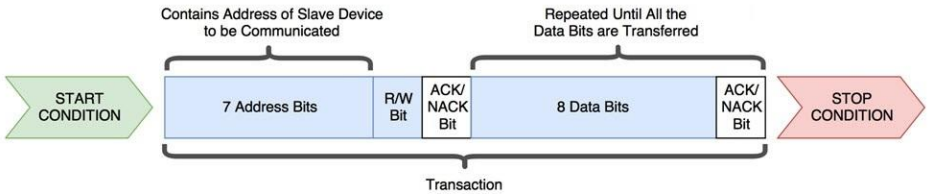
Hình 2.14. *Mô tả kết nối của các chế độ chính (1master/1slave và nhiều master/nhiều slave)*

Tại một thời điểm truyền nhận dữ liệu chỉ có một Master được hoạt động, điều khiển dây SCL và phát tín hiệu bắt đầu tới các Slave. Tất cả các thiết bị đáp ứng sự điều hướng của Master gọi là Slave. Giữa các Slave với nhau, phân biệt bằng 7bit địa chỉ.

c) Cách truyền dữ liệu của giao thức I2C

Giao thức (phương thức giao tiếp) là cách các thiết bị đã thống nhất với nhau khi sử dụng một chuẩn nào đó để truyền và nhận tín hiệu. Dữ liệu được truyền đi trên dây SDA được thực hiện như sau:

- Master thực hiện điều kiện bắt đầu I2C (Start Condition)
- Gửi địa chỉ 7 bit + 1bit Đọc/Ghi (R/W) để giao tiếp muốn đọc hoặc ghi dữ liệu tại Slave có địa chỉ trên
- Nhận phải hồi từ Bus, nếu có một bit ACK (Kéo SDA xuống thấp) Master sẽ gửi dữ liệu
- Nếu là đọc dữ liệu R/W bit = 1, chân SDA của master sẽ là input, đọc dữ liệu từ Slave gửi về. Nếu là ghi dữ liệu R/W = 0, chân SDA sẽ là output ghi dữ liệu vào Slave
- Truyền điều khiện kết thúc (Stop Condition)
- Mỗi lần giao tiếp có cấu trúc như sau:

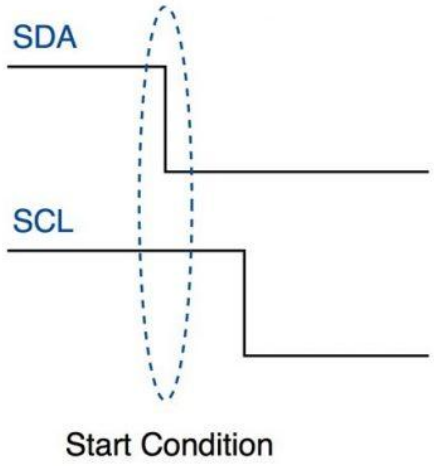


Hình 2.15. *Cấu trúc mỗi lần giao tiếp của I2C*

d) Start conditon (Điều kiện bắt đầu)

Bất cứ khi nào một thiết bị chủ / IC quyết định bắt đầu một giao dịch, nó sẽ chuyển mạch SDA từ mức điện áp cao xuống mức điện áp thấp trước khi đường SCL chuyển từ cao xuống thấp.

Khi điều kiện bắt đầu được gửi bởi thiết bị Master, tất cả các thiết bị Slave đều hoạt động ngay cả khi chúng ở chế độ ngủ (sleep mode) và đợi bit địa chỉ.



Hình 2.16. *Điều kiện bắt đầu để giao tiếp I2C*

Bit Read/Write

Bit này xác định hướng truyền dữ liệu. Nếu thiết bị Master / IC cần gửi dữ liệu đến thiết bị Slave, bit này được thiết lập là ‘0’. Nếu IC Master cần nhận dữ liệu từ thiết bị Slave, bit này được thiết lập là ‘1’.

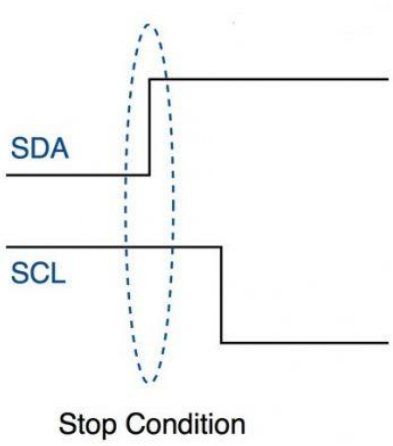
Bit ACK / NACK

ACK / NACK là viết tắt của Acknowledged/Not-Acknowledged. Nếu địa chỉ vật lý của bất kỳ thiết bị Slave nào trùng với địa chỉ được thiết bị Master phát, giá trị của bit này được set là ‘0’ bởi thiết bị Slave. Ngược lại, nó vẫn ở mức logic ‘1’ (mặc định).

Khối dữ liệu

Nó bao gồm 8 bit và chúng được thiết lập bởi bên gửi, với các bit dữ liệu cần truyền tới bên nhận. Khối này được theo sau bởi một bit ACK / NACK và được set thành ‘0’ bởi bên nhận nếu nó nhận thành công dữ liệu. Ngược lại, nó vẫn ở mức logic ‘1’. Sự kết hợp của khối dữ liệu theo sau bởi bit ACK / NACK được lặp lại cho đến quá trình truyền dữ liệu được hoàn tất.

e) Điều kiện kết thúc (Stop condition)



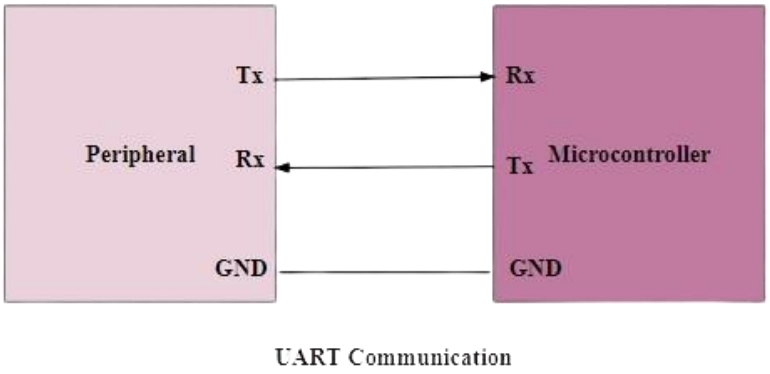
Hình 2.17. *Điều kiện kết thúc giao tiếp*

Sau khi các khung dữ liệu cần thiết được truyền qua đường SDA, thiết bị Master chuyển đường SDA từ mức điện áp thấp sang mức điện áp cao trước khi đường SCL chuyển từ cao xuống thấp.

2.2.4. UART

a) Khái quát về UART

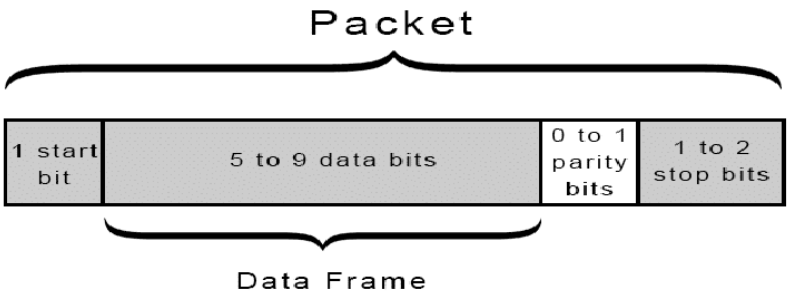
UART tiếng anh là Universal Asynchronous Reciver/Transmister một chuẩn giao tiếp không đồng bộ cho MCU và các thiết bị ngoại vi. Chuẩn UART là chuẩn giao tiếp điểm và điểm, nghĩa là trong mạng chỉ có hai thiết bị đóng vai trò là transmister hoặc reciver.



Hình 2.18. *Giao tiếp UART giữa ngoại vi và vi điều khiển*

b) Cách hoạt động của giao thức UART

- UART là giao thức truyền thông không đồng bộ, nghĩa là không có xung Clock, các thiết bị có thể hiểu được nhau nếu các Setting giống nhau
- UART là truyền thông song công(Full duplex) nghĩa là tại một thời điểm có thể truyền và nhận đồng thời.
- Trong đó quan trọng nhất là Baund rate (tốc độ Baund) là khoảng thời gian dành cho 1 bit được truyền. Phải được cài đặt giống nhau ở gửi và nhận.
- Sau đó là định dạng gói tin.
- Định dạng gói tin như sau:



Hình 2.19. *Định dạng 1 gói tin khi truyền nhận UART*

Start – Bit

Start-bit còn được gọi là bit đồng bộ hóa được đặt trước dữ liệu thực tế. Nói chung, một đường truyền dữ liệu không hoạt động được điều khiển ở mức điện áp cao. Để bắt đầu truyền dữ liệu, truyền UART kéo đường dữ liệu từ mức điện áp cao (1) xuống mức điện áp thấp (0). UART thu được thông báo sự chuyển đổi này từ mức cao sang mức thấp qua đường dữ liệu cũng như bắt đầu hiểu dữ liệu thực. Nói chung, chỉ có một start-bit.

Stop – Bit

Bit dừng được đặt ở phần cuối của gói dữ liệu. Thông thường, bit này dài 2 bit nhưng thường chỉ sử dụng 1 bit. Để dừng sóng, UART giữ đường dữ liệu ở mức điện áp cao.

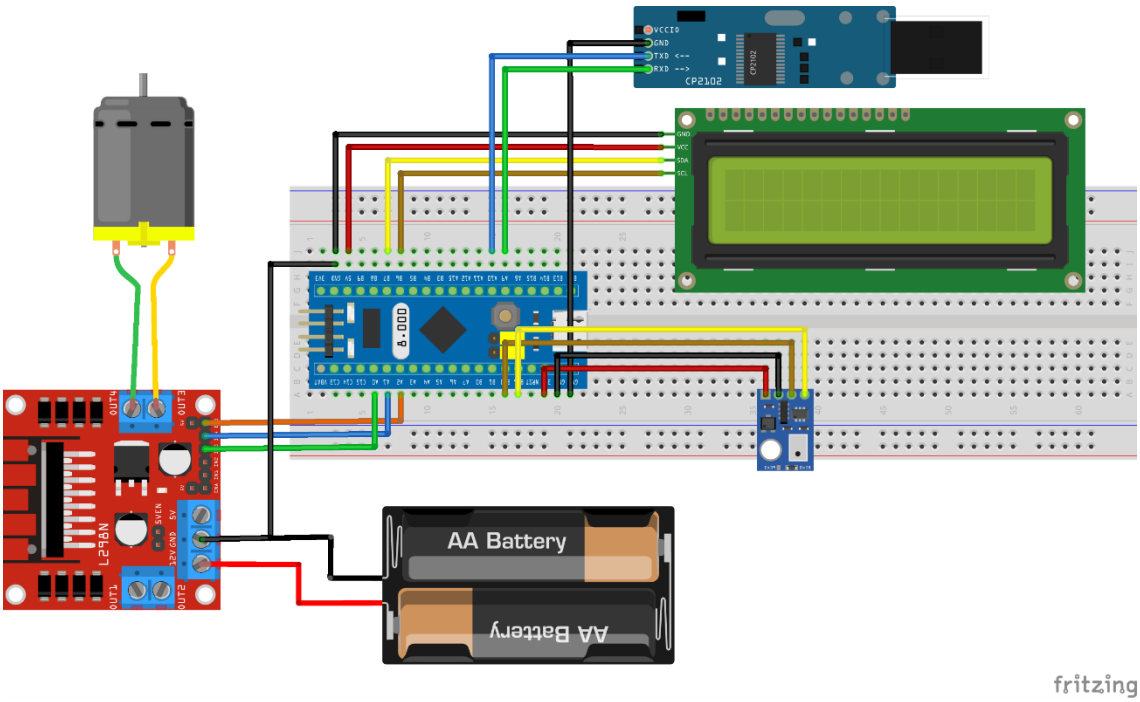
Partity Bit

Bit chẵn lẻ cho phép người nhận đảm bảo liệu dữ liệu được thu thập có đúng hay không. Đây là một hệ thống kiểm tra lỗi cấp thấp & bit chẵn lẻ có sẵn trong hai phạm vi như Chẵn lẻ – chẵn lẻ cũng như Chẵn lẻ – lẻ. Trên thực tế, bit này không được sử dụng rộng rãi nên không bắt buộc.

Data frame

Các bit dữ liệu bao gồm dữ liệu thực được truyền từ người gửi đến người nhận. Độ dài khung dữ liệu có thể nằm trong khoảng 5 & 8. Nếu bit chẵn lẻ không được sử dụng thì chiều dài khung dữ liệu có thể dài 9 bit. Nói chung, LSB của dữ liệu được truyền trước tiên sau đó nó rất hữu ích cho việc truyền.

2.3. Sơ đồ ghép nối giữa STM32 và các module phần cứng



Hình 2.20. *Sơ đồ ghép nối phần cứng*

Phần cứng	Kết nối chân
AHT10	VIN ↔ 3.3 GND ↔ G SCL ↔ B10 SDA ↔ B11
LCD 16x2	GND ↔ G VCC ↔ 5V SDA ↔ B7 SCL ↔ B6
CP2102	RXD ↔ A9 TXD ↔ A10 GND ↔ G
L298N	IN3 ↔ A0 IN4 ↔ A1 ENB ↔ A2

2.4. Nguyên lý giao tiếp giữa STM32 và các module phần cứng

2.4.1.

2.4.2. Giao tiếp I2C với cảm biến AHT10 và module giao tiếp kí tự LCD (PCF8574)

Nguyên lý:

- Giao tiếp I2C (Inter-Integrated Circuit) là giao tiếp nối tiếp đồng bộ có hai dây: SCL (clock) và SDA (data).
- STM32 đóng vai trò Master, còn cảm biến AHT10 và PCF8574 là Slave.
- Master sẽ gửi lệnh đọc dữ liệu tới Slave, rồi Slave gửi lại dữ liệu phản hồi.
- Đối với PCF8574 STM32 gửi lệnh và dữ liệu hiển thị theo trình tự định trước.

Ứng dụng:

- STM32 đọc dữ liệu nhiệt độ, độ ẩm từ cảm biến AHT10.
- Thư viện HAL của STM32 cung cấp hàm HAL_I2C_Master_Transmit() và HAL_I2C_Master_Receive() để giao tiếp.
- Hiển thị nhiệt độ, độ ẩm.
- Thư viện LCD thường sử dụng các hàm như lcd_gotoxy(), lcd_puts() để điều khiển.

2.4.3. Giao tiếp UART với máy tính (UART truyền dữ liệu và nhận lệnh)

Nguyên lý:

- Giao tiếp UART (Universal Asynchronous Receiver/Transmitter) là giao tiếp nối tiếp không đồng bộ.
- Chỉ cần hai chân: TX (truyền) và RX (nhận).
- Dữ liệu truyền theo dạng byte (8 bit hoặc nhiều hơn), có start/stop bit.

Ứng dụng:

- STM32 gửi thông tin đo được (nhiệt độ, độ ẩm) lên máy tính qua UART.
- Đồng thời nhận các lệnh điều khiển từ người dùng (qua phần mềm terminal).
- Sử dụng hàm HAL_UART_Transmit() và HAL_UART_Receive_IT().

2.4.4. Giao tiếp GPIO điều khiển động cơ qua driver cầu H

Nguyên lý:

- GPIO được dùng để điều khiển các tín hiệu số: bật/tắt (HIGH/LOW).
- Động cơ một chiều được điều khiển qua mạch cầu H (như L298N).
- Hai chân GPIO dùng để chọn chiều quay (IN1/IN2), một chân PWM điều tốc.

Ứng dụng:

- STM32 điều khiển chiều quay động cơ bằng HAL_GPIO_WritePin() và điều chỉnh tốc độ bằng PWM qua TIM2.

CHƯƠNG 3. THIẾT KẾ PHẦN MỀM

3.1. Phân tích yêu cầu xử lý đảm bảo tính thời gian thực cho các đối tượng

Bảng 3.1. *Phân tích yêu cầu xử lý*

Đối tượng	Yêu cầu thời gian thực
Nhiệt độ/độ ẩm	Đặc điểm chính: Nhiệt độ và độ ẩm thường thay đổi chậm, có dải giá trị xác định và cần độ chính xác tùy ứng dụng. Yêu cầu thời gian thực: + Tần suất lấy mẫu: Phụ thuộc vào tốc độ thay đổi môi trường và yêu cầu ứng dụng (từ vài giây/lần cho giám sát đến nhanh hơn cho điều khiển). + Độ trễ: Bao gồm độ trễ đọc cảm biến, xử lý dữ liệu, truyền thông (LCD, UART) và quan trọng nhất là độ trễ phản ứng của hệ thống điều khiển. + Xử lý lỗi & Độ tin cậy: Khả năng phát hiện, xử lý lỗi cảm biến và đảm bảo hoạt động liên tục, không bị treo.
Điều khiển động cơ	Đặc điểm chính: Động cơ có quán tính, cần điều khiển liên tục và chính xác (tốc độ, hướng qua PWM), đồng thời yêu cầu giám sát và xử lý lỗi khẩn cấp (quá tải, kẹt). Yêu cầu thời gian thực: + Độ trễ phản ứng lệnh: Phản hồi nhanh với lệnh người dùng (vài chục - vài trăm ms). + Tần suất cập nhật tín hiệu điều khiển: Cập nhật PWM liên tục, đủ nhanh để điều khiển mượt mà (vài trăm Hz - vài kHz). + Xử lý lỗi khẩn cấp: Phản ứng gần như tức thì (vài ms) để bảo vệ hệ thống.
Hiển thị màn hình	Đặc điểm chính: LCD cung cấp phản hồi trực quan, cần cập nhật định kỳ nhưng không quá nhanh (trừ khi có yếu tố động), và phải đảm bảo tính toàn vẹn dữ liệu hiển thị. Yêu cầu thời gian thực: + Độ trễ cập nhật: Vài trăm mili giây đến 1-2 giây cho dữ liệu thay đổi chậm; vài chục đến vài trăm mili giây cho phản hồi lệnh để người dùng cảm nhận sự "tức thì". + Tần suất làm mới: 500ms - 1000ms/lần cho màn hình văn bản là đủ. + Tính toàn vẹn hình ảnh: Tránh hiện tượng "rách hình" hoặc hiển thị dữ liệu lỗi do xung đột ghi.
Gửi dữ liệu lên máy tính	Đặc điểm chính: Giao tiếp nối tiếp theo tốc độ Baud Rate, sử dụng bộ đệm, thường là hai chiều và phục vụ giám sát, gỡ lỗi hoặc giao tiếp ứng dụng PC. Yêu cầu thời gian thực:

	+ Độ trễ báo cáo: Từ vài trăm ms đến vài giây cho báo cáo định kỳ; vài chục đến vài trăm ms cho cảnh báo hoặc phản hồi lệnh. + Tần suất gửi: Phụ thuộc vào tốc độ thay đổi dữ liệu và nhu cầu của PC, tránh gửi quá tải. + Độ tin cậy dữ liệu: Đảm bảo dữ liệu không bị mất, hỏng hoặc trộn lẫn. + Không làm tắc nghẽn hệ thống: Việc gửi dữ liệu không được cản trở các task quan trọng hơn.
Nhập lệnh điều khiển từ máy tính	Đặc điểm chính: Dữ liệu nhận tuần tự qua UART, cần được ghép thành lệnh hoàn chỉnh và xử lý tức thì Yêu cầu thời gian thực: + Độ trễ phản hồi lệnh: Rất thấp (vài chục - vài trăm ms) để người dùng cảm nhận tương tác tức thì. + Độ tin cậy: Đảm bảo không mất ký tự, không sai lệch dữ liệu, phân tích chính xác. + Xử lý lỗi cú pháp: Phát hiện và bỏ qua lệnh không hợp lệ mà không treo hệ thống. + Không làm tắc nghẽn: Quá trình nhận/phân tích không được gián đoạn các task quan trọng khác.

Bảng 3.2. *Các tác vụ chính*

Task	Mô tả
Xử lý lệnh từ máy tính	Phân tích, xử lý và thực thi lệnh nhận được từ máy tính
Điều khiển động cơ	Điều khiển các thông số cần thiết để điều khiển động cơ như tốc độ, chiều quay
Đọc cảm biến	Định kỳ đọc giá trị đo được từ cảm biến, lưu trữ cho các tác vụ khác sử dụng
Hiển thị LCD	Định kỳ hiển thị giá trị đo được từ cảm biến lên màn hình LCD
Gửi dữ liệu lên máy tính	Định kỳ gửi giá trị đo được từ cảm biến lên máy tính phục vụ giám sát

Bảng 3.3. *Tập lệnh điều khiển*

Command	Mô tả
!motor <on,off>	Bật/tắt động cơ
!rotate <cw,ccw>	Chọn chiều động cơ
!speed <0-100>	Điều chỉnh tốc độ quay của động cơ
!display <all,temp,humid>	Lựa chọn chế độ hiển thị

3.2. Phân tích và lập trình cho yêu cầu BTL theo 2 mô hình đơn nhiệm

3.2.1. Time-Trigger Cyclic Executive Scheduler

a) Giới thiệu mô hình Time-Trigger Cyclic Executive Scheduler

Khái niệm

Time-Trigger Cyclic Executive Scheduler (Trình lập lịch theo chu kỳ được kích hoạt bởi thời gian) là một mô hình điều khiển đơn nhiệm (non-preemptive) trong hệ thống nhúng, trong đó tất cả các tác vụ (task) được thực hiện theo một vòng lặp cố định, với chu kỳ xác định trước. Mỗi tác vụ sẽ được gọi tại một thời điểm cụ thể trong vòng lặp, thường được tính toán kỹ lưỡng để đảm bảo hệ thống hoạt động đúng thời gian yêu cầu.

Đặc điểm

- Không dùng hệ điều hành (No RTOS).
- Không có ngắt bởi lập lịch (no task preemption).
- Các tác vụ được lập trình như hàm và được gọi theo thứ tự cố định.
- Hệ thống chạy theo vòng lặp vô hạn (super loop).
- Tần suất gọi mỗi tác vụ được điều chỉnh bằng biến thời gian nội bộ hoặc bộ đếm.

Cấu trúc chương trình

```
int main(void) {
    system_init();
    while (1) {
        task_1();  // Gọi mỗi 10ms
        task_2();  // Gọi mỗi 100ms
        task_3();  // Gọi mỗi 1000ms
        wait_until_next_cycle();   // Delay hoặc đồng bộ thời gian
    }
}
```

Ưu điểm

- Đơn giản, dễ hiểu và dễ triển khai trên vi điều khiển không có hệ điều hành.
- Dễ kiểm soát thời gian chạy và độ trễ của các tác vụ.
- Tốn ít bộ nhớ, không cần quản lý tài nguyên phức tạp như trong hệ điều hành.

Nhược điểm

- Không phù hợp cho hệ thống phức tạp hoặc thời gian thực cứng.
- Không có khả năng ưu tiên – một tác vụ chạy lâu sẽ làm trễ các tác vụ khác.
- Khó xử lý các tác vụ mang tính ngắt quãng hoặc không đều.

Ứng dụng thực tế

- Hệ thống điều khiển thiết bị gia dụng, cảm biến đơn giản.
- Giao tiếp tuần tự UART đơn giản.
- Các dự án nhúng giáo dục, nghiên cứu yêu cầu thấp.

b) Xây dựng mô hình đơn nhiệm Time-Trigger Cyclic Executive Scheduler

Bảng 3.4. *Nhiệm vụ các task và thông số các task*

Task	Nhiệm vụ	Thông số	Phân tích
1	Đọc cảm biến nhiệt độ, độ ẩm (AHT10)	T_1 = 2000ms C_1 = 2ms D_1 = 1000ms	Nhiệt độ và độ ẩm thay đổi chậm, chu kỳ lấy mẫu 2s là hợp lý. Cảm biến AHT10 dùng giao tiếp I2C, thời gian đo & đọc dữ liệu mất khoảng 1.5-2ms. Sau khi lấy dữ liệu, hệ thống sẽ lưu trữ vào biến dùng chung.
2	Hiển thị dữ liệu lên LCD	T_2 = 500ms C_2 = 3ms D_2 = 100ms	LCD cần cập nhật liên tục để phản ánh trạng thái nhiệt độ, độ ẩm hiện tại. Thời gian hiển thị lên LCD khoảng 2–3ms nếu dùng giao tiếp I2C hoặc 4-bit. Thời gian phản hồi không cần quá nhanh, nên deadline đặt là 100ms.
3	Truyền dữ liệu UART	T_3 = 5000ms C_3 = 1ms D_3 = 500ms	Thông tin gửi qua UART 115200 baud chỉ mất khoảng 0.5ms/gói. Mỗi 5 giây gửi 1 bản tin nhiệt độ và độ ẩm để hiển thị hoặc giám sát trên PC.
4	Xử lý lệnh UART điều khiển	T_4 = 50ms C_4 = 1ms D_4 = 20ms	Tác vụ kiểm tra cờ command_ready để xử lý lệnh UART từ người dùng. Xử lý chuỗi lệnh ngắn mất dưới 1ms, cần phản hồi nhanh nên deadline đặt 20ms.
5	Điều khiển động cơ bằng PWM	T_5 = 100ms C_5 = 1ms D_5 = 50ms	Điều chỉnh tốc độ và chiều quay động cơ dựa vào giá trị điều khiển. Thời gian cập nhật PWM ngắn, chỉ vài dòng lệnh. Tần suất 100ms là hợp lý với hệ thống điều khiển đơn giản.

Bảng 3.5. *Danh sách các task và thông số T,C,D*

Task	Nhiệm vụ	Chu kỳ T	Thời gian thực thi C	Deadline D
T1	Đọc cảm biến AHT10	2000 ms	5 ms	1000 ms
T2	Hiển thị LCD	500 ms	3 ms	100 ms
T3	Gửi UART TX	5000 ms	2 ms	1000 ms
T4	Nhận UART RX	Bất đồng bộ (ISR)	N/A	N/A
T5	Điều khiển động cơ	100 ms	2 ms	100 ms

Task RX là ngắt (ISR) nên không xét trong chu kỳ chính P

$T = LCM(100ms, 500ms, 2000ms, 5000ms) = 10s$

Ta chọn f (frame thỏa mãn các điều kiện sau):

- Mỗi task chỉ được thực hiện tối đa 1 lần trong mỗi frame:
$$f \le T_i, \forall i \Leftrightarrow f \le \min\{100, 500, 2000, 5000\} = 100$$
- f phải lớn hơn hoặc bằng thời gian thực thi của mỗi task:
$$f \ge C_i, \forall i \Leftrightarrow f \ge \max\{2, 2, 3, 5\} = 5$$
- Điều kiện deadline:
$$2f - \gcd(T_i, f) \le D_i, \forall i$$

Sau khi tính toán thì chọn $f = 25$ với số frame là 400 sẽ thỏa mãn các điều kiện trên.

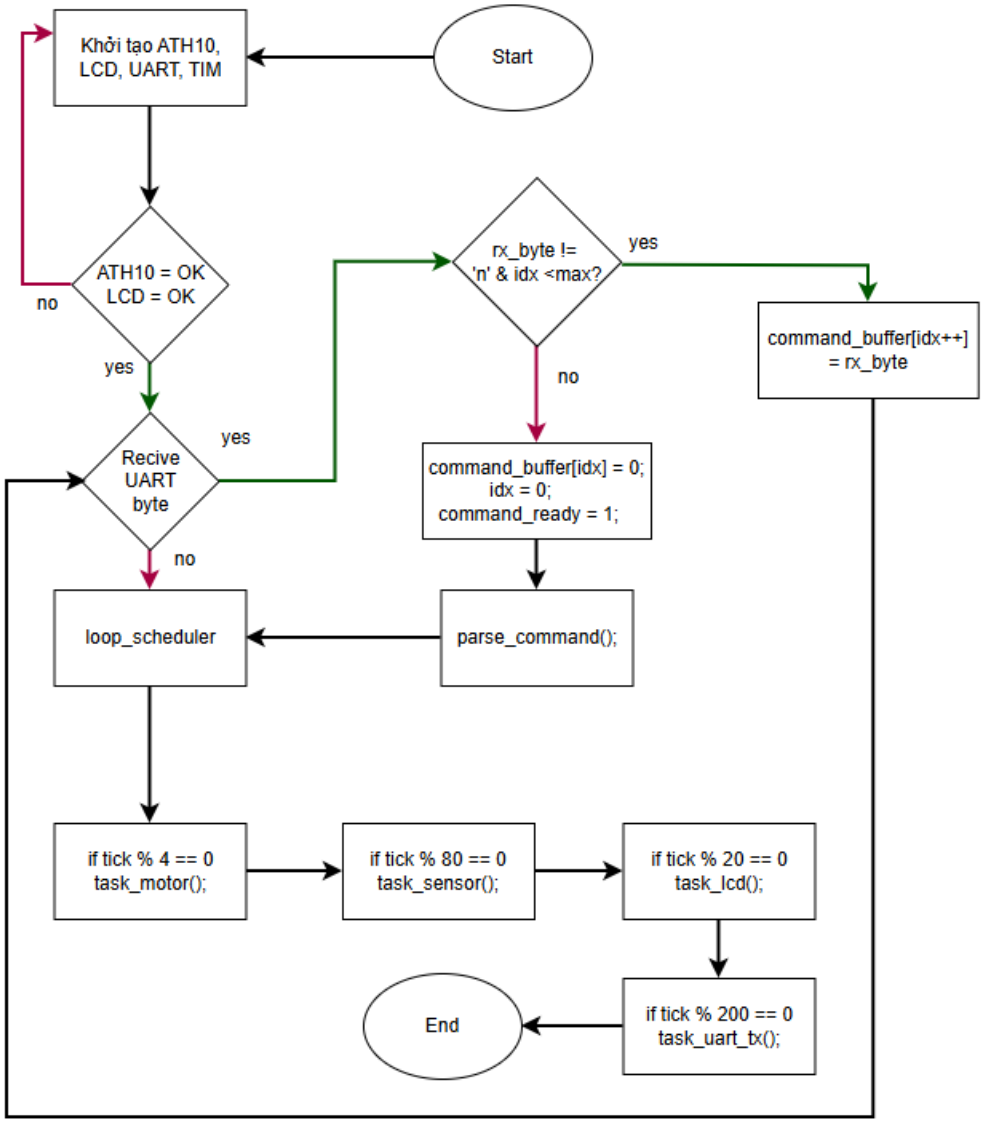
Bảng 3.6. *Bảng lập lịch thời gian chạy các task trong 1 chu kì*

Frame (ms)	Thời điểm (ms)	Task chạy
0	0	task_sensor(), task_lcd(), task_motor()
1	25	task_motor()
2	50	task_lcd(), task_motor()
3	75	task_motor()
4	100	task_motor()
5	125	task_lcd(), task_motor()
...	...	...
20	500	task_lcd(), task_motor()
40	1000	task_lcd(), task_motor()
80	2000	task_sensor(), task_lcd(), task_motor()
200	5000	task_uart_tx(), task_lcd(), task_motor()
320	8000	task_sensor(), task_lcd(), task_motor()
399	9975	task_motor()

Trong đó:

- task_motor() chạy **mỗi 100ms** → tức là **mỗi 4 frame**
- task_lcd() chạy mỗi 500ms → tức là **mỗi 20 frame**
- task_sensor() chạy mỗi 2000ms → mỗi **80 frame**
- task_uart_tx() chạy mỗi 5000ms → mỗi **200 frame**

c) Lưu đồ thuật toán và chương trình lập lịch theo Time-Trigger Cyclic Executive Scheduler



Hình 3.1. *Lưu đồ thuật toán chương trình*

Chương trình lập lịch:

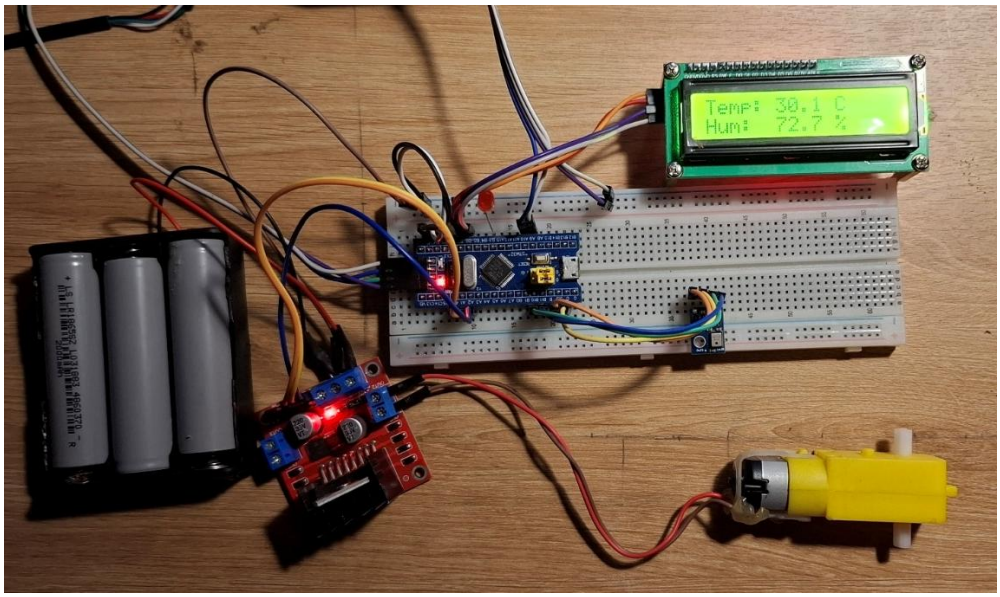
```
void loop_scheduler(void) {
    static uint16_t tick = 0;

    if (tick % 4 == 0)    task_motor();      // 100ms
    if (tick % 20 == 0)   task_lcd();        // 500ms
    if (tick % 80 == 0)   task_sensor();     // 2000ms
    if (tick % 400 == 0)  task_uart_tx();    // 10000ms
```

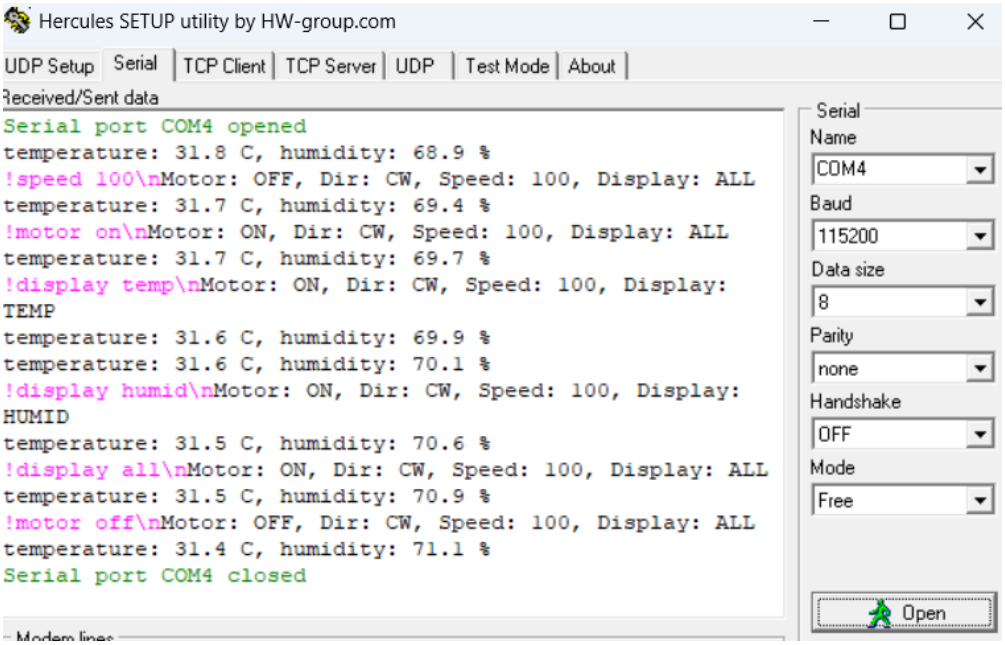
```
    if (command_ready) {
            parse_command(command_buffer);      //Gọi xử lý lệnh
            command_ready = 0;
            memset(command_buffer, 0, sizeof(command_buffer));
        }

    tick++;
    if (tick >= 400) tick = 0;
}
```

d) Kết quả thực nghiệm



Hình 3.2. *Phần cứng bao gồm kitSTM32f103 C8T6, ATH10, L298N, MotorDC 12V*



Hình 3.3. *Hiển thị dữ liệu nhiệt độ, độ ẩm và lệnh điều khiển*

e) Đánh giá

Bảng 3.7. *Kết quả hệ thống sau thực nghiệm*

Thành phần	Kết quả
Đọc cảm biến AHT10	Ổn định, đúng chu kỳ (2 giây / lần), giá trị chính xác và ổn định.
Hiển thị LCD	Hiển thị đúng nội dung, thay đổi theo lệnh UART (`!display`). Không bị nháy hoặc tràn dữ liệu.
UART điều khiển	Gửi các lệnh như `!motor on`, `!rotate ccw`, `!speed 75`,... đều phản hồi và cập nhật chính xác.
UART gửi dữ liệu	Gửi chuỗi `"temperature: ... humidity: ..."` đúng định dạng và chu kỳ (5 giây / lần).
Động cơ PWM	Chạy đúng chiều, tốc độ theo lệnh UART. Có thể dừng, đổi chiều, thay đổi tốc độ từ máy tính.
Lập lịch TTC	Các task thực thi đúng thời điểm nhờ tick+modulo, đảm bảo không trễ hẹn, không xung đột.

Qua quá trình triển khai, hệ thống đã đạt được sự ổn định và đáp ứng đầy đủ chức năng: đo và hiển thị nhiệt độ – độ ẩm, truyền dữ liệu qua UART, điều khiển động cơ và xử lý lệnh từ người dùng. Mô hình điều khiển theo lịch trình tuần tự định thời (TTC) cho phép hệ thống duy trì khả năng phản hồi tốt mà không cần đến hệ điều hành thời gian thực. Tổng thời gian thực thi các task trong 1 frame đều <25ms → **không bị trễ deadline**, đạt yêu cầu lý thuyết về TTC.

Tuy nhiên vẫn còn 1 số hạn chế

- Không phản hồi tức thời: Nếu người dùng gửi lệnh UART ngay sau khi parse_command() vừa chạy, phải chờ tối đa 25ms mới xử lý tiếp.
- Dữ liệu LCD có thể cập nhật chậm: Nếu AHT10 chưa cập nhật, LCD có thể hiển thị dữ liệu cũ. Tuy nhiên, không gây lỗi logic.
- Không mở rộng tốt cho ứng dụng phức tạp: TTC không phù hợp nếu thêm nhiều task bất đồng bộ hoặc yêu cầu thời gian thực cao (ví dụ: xử lý sensor tốc độ cao, camera, SPI tốc độ cao…).

3.2.2. Non-Preemptive Event-Triggered Scheduling

a) Giới thiệu mô hình đơn nhiệm Non-Preemptive Event-Triggered Scheduling

Định nghĩa:

(Lập lịch kích hoạt theo sự kiện, không ưu tiên) là mô hình điều khiển đơn nhiệm, nơi **chương trình chính (main loop)** không chạy tuần tự theo chu kỳ cố định (như Time-Triggered), mà thay vào đó chỉ chạy **khi có sự kiện xảy ra** (ví dụ: có lệnh UART mới, timeout, ngắt ngoài, hoặc cờ trạng thái được set từ ISR).

Đặc điểm mô hình:

- Không có ngắt thời gian định kỳ như HAL_Delay/tick
- Mỗi sự kiện xảy ra sẽ thiết lập một cờ
- Vòng lặp chính (main loop) kiểm tra các cờ, và xử lý tương ứng

- Không có ưu tiên, task nào có cờ thì thực hiện ngay và đến hết mới qua task khác

Ưu điểm:

- Đơn giản, dễ kiểm soát luồng chương trình
- Không dùng hệ điều hành hoặc FreeRTOS
- Có thể dễ dàng debug vì mọi thứ tuần tự

Nhược điểm:

- Không thích hợp nếu thời gian xử lý task dài (block các task khác)
- Không có ưu tiên xử lý, nếu UART đến khi đang xử lý LCD → bị chậm

b) Xây dựng mô hình hình đơn nhiệm Non-Preemptive Event-Triggered Scheduling

Ngắt Timer (HAL_TIM_PeriodElapsedCallback) được gọi mỗi 1ms hoặc 10ms tùy cấu hình timer.

Trong đó có:

- Biến đếm thời gian toàn cục: ms_ticks++
- Dựa vào ms_ticks, các cờ (event flag) được set:

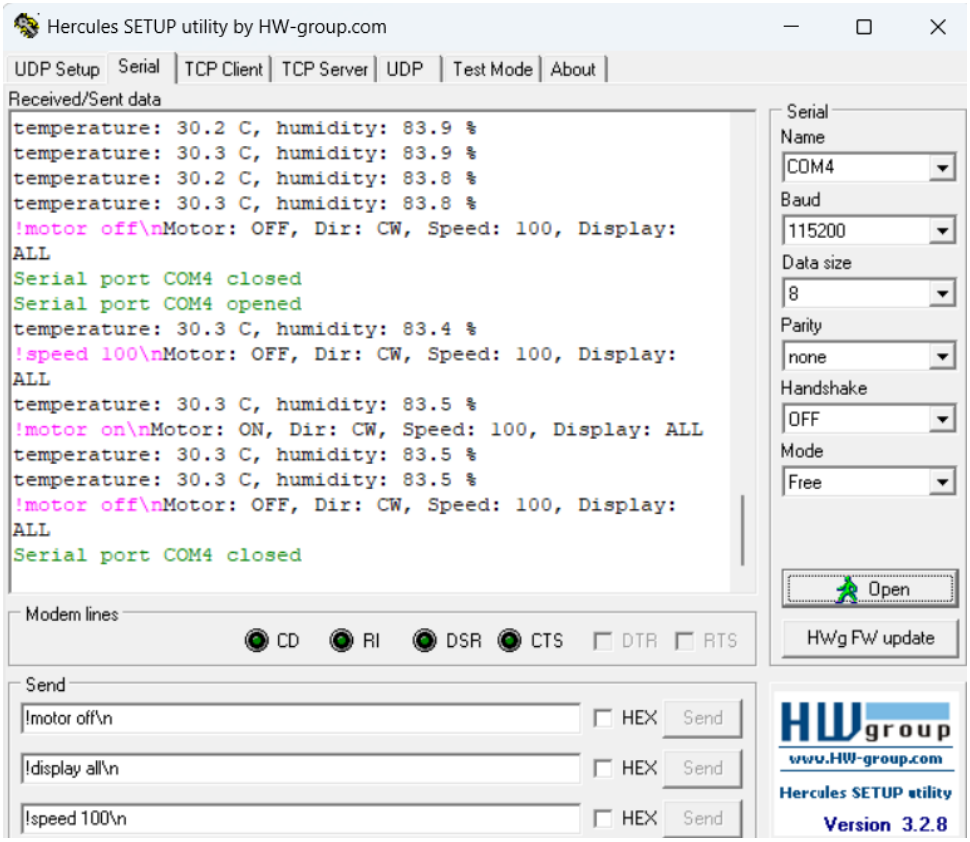
Tên flag	**Khoảng thời gian**	**Tác vụ liên quan**
`sensor_event`	mỗi 2000ms (2 giây)	Đọc dữ liệu từ cảm biến AHT10
`lcd_event`	mỗi 500ms	Hiển thị dữ liệu lên LCD
`uart_tx_event`	mỗi 5000ms	Gửi dữ liệu qua UART
`motor_event`	mỗi 100ms	Điều khiển động cơ theo lệnh

Chương trình tạo cờ sự kiện:

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  if (htim->Instance == TIM1)
  {
    ms_ticks ++;
    if (ms_ticks % 2000 == 0) sensor_event = 1;
    if (ms_ticks % 500 == 0)  lcd_event = 1;
    if (ms_ticks % 5000 == 0) uart_tx_event = 1;
    if (ms_ticks % 100 == 0)  motor_event = 1;

    if (ms_ticks >= 10000) ms_ticks = 0;  // Reset sau mỗi 10s để tránh tràn số
  }
}
```

c) Kết quả thực nghiệm và đánh giá



Hình 3.4. *Kết quả thực nghiệm với mô hình Non-Preemptive Event-Triggered Scheduling*

Đánh giá kết quả

Sau quá trình triển khai và thực nghiệm, hệ thống hoạt động ổn định theo đúng thiết kế với mô hình lập lịch kích hoạt theo sự kiện không có ngắt ngang (Non-Preemptive Event-Triggered Scheduling). Bộ định thời (TIM1) được cấu hình để tạo ngắt định kỳ mỗi 25ms nhằm tăng biến đếm ms_ticks. Dựa trên giá trị của biến đếm này, các cờ sự kiện (sensor_event, lcd_event, motor_event, uart_tx_event) được đặt ở các thời điểm phù hợp với chu kỳ yêu cầu của từng nhiệm vụ. Trong vòng lặp chính (while(1)), hệ thống kiểm tra các cờ sự kiện để gọi các hàm tác vụ tương ứng một cách tuần tự.

Hệ thống bao gồm các nhiệm vụ chính như đọc cảm biến nhiệt độ độ ẩm (AHT10), hiển thị thông tin lên LCD, điều khiển động cơ, và truyền dữ liệu qua UART. Kết quả thực nghiệm ghi nhận rằng tất cả các thành phần đều hoạt động ổn định và đúng theo kỳ vọng. Cảm biến AHT10 cho giá trị nhiệt độ và độ ẩm chính xác và được cập nhật mỗi 2 giây. LCD hiển thị dữ liệu một cách rõ ràng và thay đổi theo thời gian thực, với chu kỳ 500ms. Giao tiếp UART nhận lệnh điều khiển từ máy tính (thông qua phần mềm Hercules) và phản hồi chính xác, không bị trễ hoặc nhiễu. Các lệnh như !motor on, !speed 100, !rotate ccw, !display all được xử lý đúng, cập nhật trạng thái hệ thống và phản hồi chính xác. Tốc độ, chiều quay và trạng thái bật/tắt của động cơ được điều chỉnh chính xác theo lệnh.

Đặc biệt, việc sử dụng mô hình Non-Preemptive giúp hệ thống tránh được xung đột tài nguyên, đơn giản hóa việc lập trình và kiểm soát lỗi. Tuy nhiên, mô

hình này cũng yêu cầu các tác vụ phải nhẹ và nhanh chóng để không chặn các sự kiện khác xảy ra gần thời điểm nhau.

Tóm lại, kết quả thực nghiệm chứng minh rằng hệ thống hoạt động ổn định, đúng chức năng, và mô hình lập lịch được chọn hoàn toàn phù hợp với yêu cầu thời gian thực nhẹ của ứng dụng.

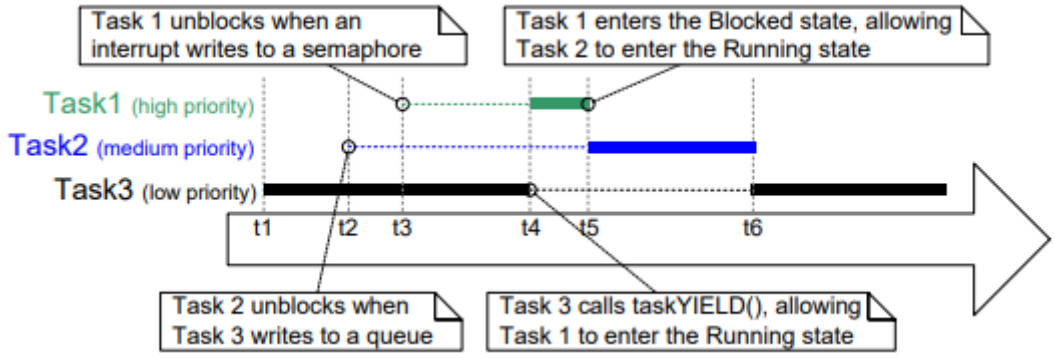
3.3. Phân tích và lập trình cho yêu cầu BTL theo 2 mô hình đa nhiệm

3.3.1. Co-operative Scheduling

a) Giới thiệu mô hình lập lịch Co-operative Scheduling

Định nghĩa:

- Trong chế độ này, các tác vụ (tasks) tự nguyện nhường quyền điều khiển CPU cho các tác vụ khác bằng cách gọi hàm taskYIELD() hoặc khi kết thúc (thông qua vTaskDelete()).
- Không có cơ chế ngắt (preemption): Bộ lập lịch không tự động thu hồi CPU từ tác vụ đang chạy, mà phụ thuộc vào sự "hợp tác" của các tác vụ. (Lập lịch không preemptive, tức là CPU chỉ chuyển task khi task hiện tại chủ động nhường).
- Phù hợp với hệ thống đơn giản, ít yêu cầu khắt khe về thời gian thực.



Hình 3.5. *Mẫu thực hiện thể hiện hành vi của trình lập lịch hợp tác*

Ưu điểm:

- Đơn giản, dễ triển khai.
- Tiết kiệm tài nguyên CPU do không có overhead từ cơ chế ngắt.
- Tránh được vấn đề race condition do không có sự tranh chấp tài nguyên giữa các tác vụ.

Nhược điểm:

- Tác vụ chạy quá lâu có thể làm nghẽn hệ thống (vì không bị ngắt).
- Không phù hợp cho ứng dụng yêu cầu phản hồi nhanh (real-time).

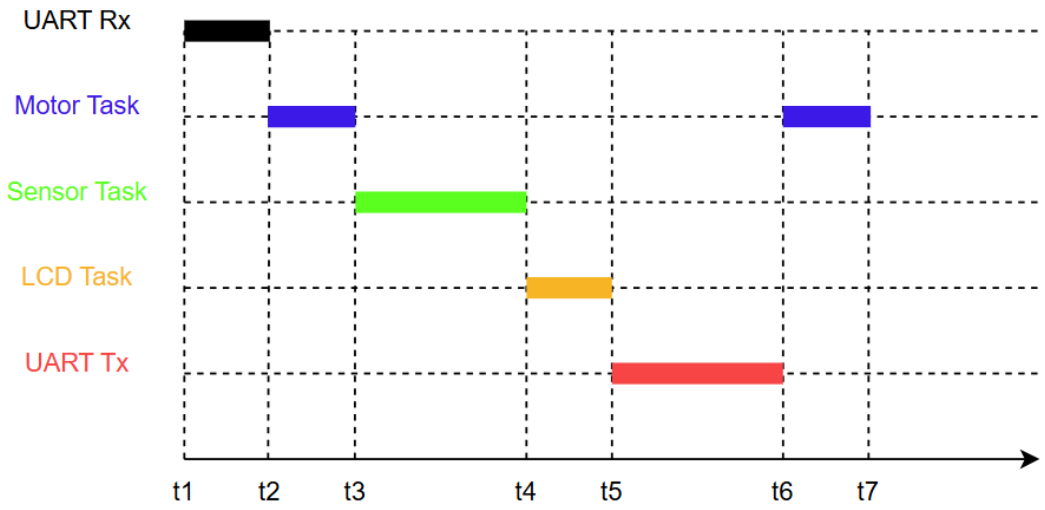
Cách hoạt động

- Khi một tác vụ được chọn để chạy, nó sẽ giữ CPU cho đến khi:
  - Tự động kết thúc (gọi vTaskDelete()).
  - Chủ động nhường CPU (gọi taskYIELD()).
- Bộ lập lịch chỉ chọn tác vụ mới khi tác vụ hiện tại ngừng hoặc nhường quyền.

Co-operative Scheduling trong FreeRTOS là một mô hình đơn giản, phù hợp cho ứng dụng nhúng có yêu cầu thấp.

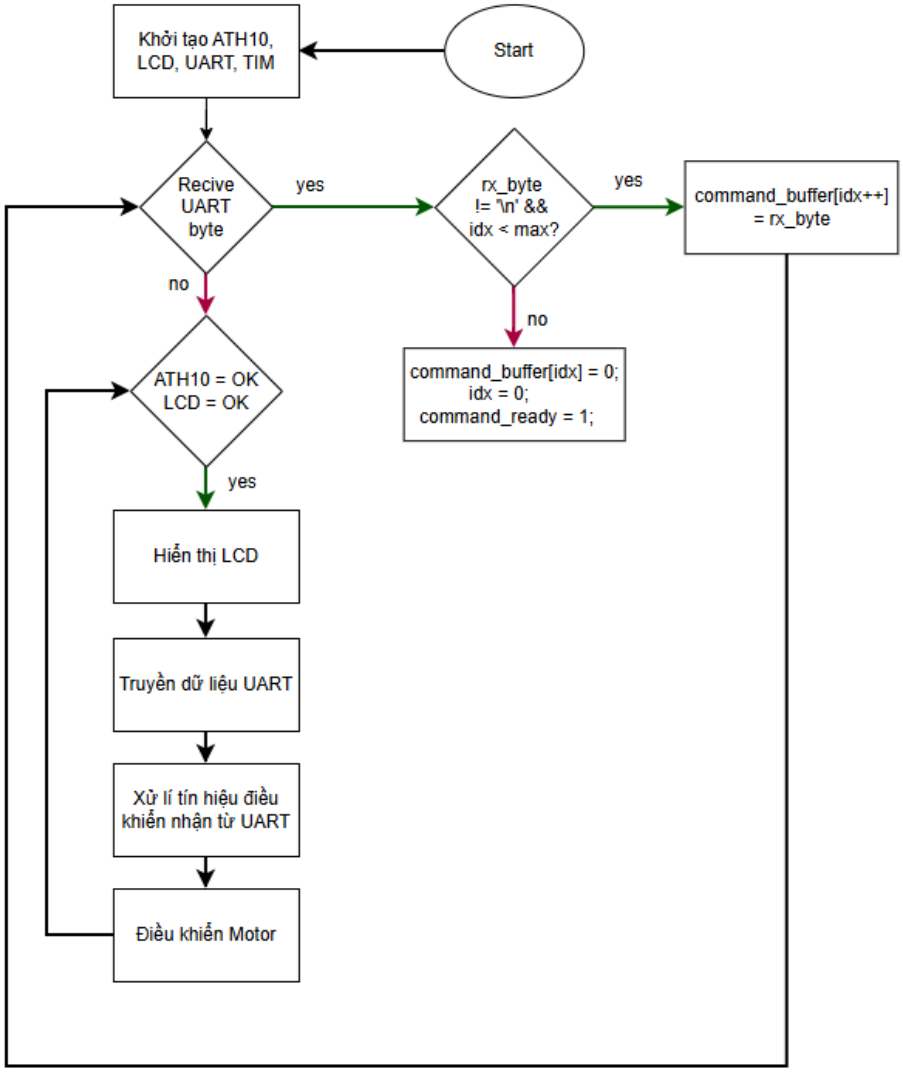
b) Xây dựng mô hình Co-operative Scheduling

Task	Ghi chú
Task 1: UART Rx	Xử lý lệnh điều khiển từ người dùng
Task 2: Motor	Điều chỉnh chiều và tốc độ động cơ
Task 3: Sensor	Đọc cảm biến mỗi 2s
Task 4: LCD	Cập nhật LCD liên tục mỗi 0.5s
Task 5: UART Tx	In dữ liệu nhiệt độ, độ ẩm mỗi 5s

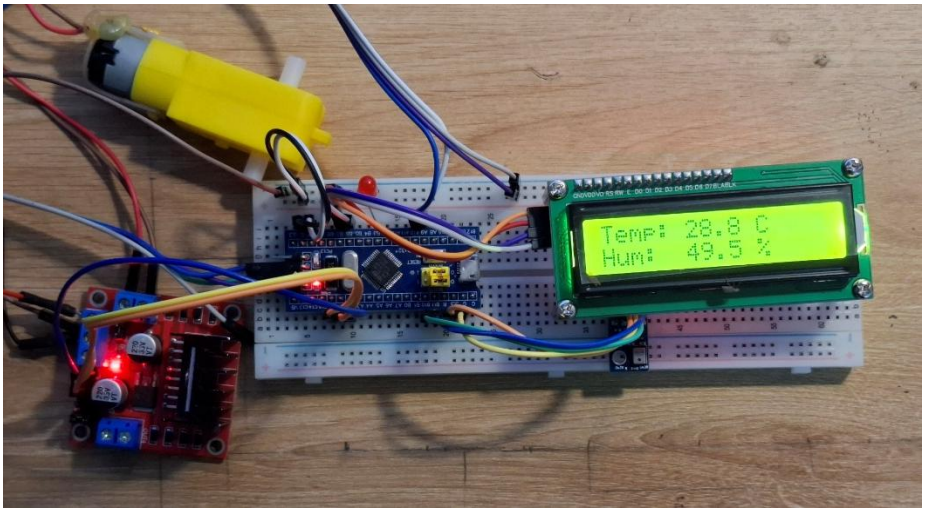


Hình 3.6. *Sơ đồ luồng thời gian Co-operative Scheduling*

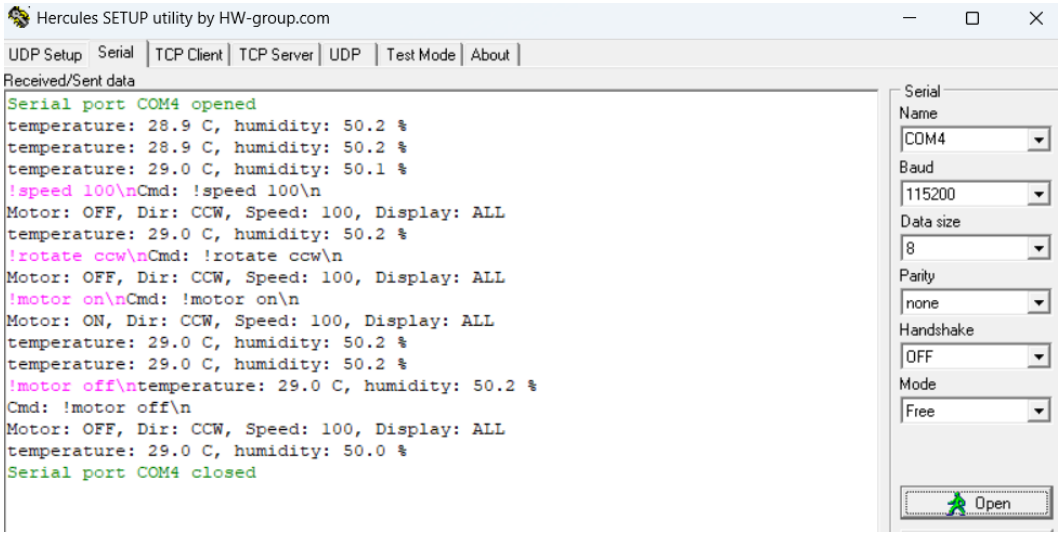
c) Lưu đồ thuật toán giao tiếp giữa phần mềm và phần cứng



d) Kết quả thực nghiệm



Hình 3.7. *Phần cứng bao gồm kitSTM32f103 C8T6, ATH10, L298N, MotorDC 12V*



Hình 3.8. *Dữ liệu hiển thị qua UART và gửi lệnh điều khiển qua phần mềm Hercules*

e) Đánh giá

- !speed 100, !rotate ccw, !motor on, !motor off đều được nhận, hiển thị và cập nhật đúng trạng thái.
- Phản hồi Cmd: và Motor: đều chính xác và phản ánh đúng nội dung lệnh.
- Hệ thống vẫn hiển thị dữ liệu cảm biến định kỳ:

Minh chứng rõ ràng cho việc các task phối hợp hoạt động thông qua Yield. Các task không chặn nhau, cho thấy scheduler hoạt động đúng trong mô hình co-operative.

Hạn chế còn tồn tại:

Trong mô hình cooperative, nếu một task quên osDelay() hoặc taskYIELD() thì sẽ ngăn không cho các task khác thực thi, ví dụ:

- Nếu StartLCDTask vẽ nhiều lần LCD mà không có delay → UART sẽ bị nghẽn.

Nếu có các tác vụ thời gian thực (như phát hiện ngưỡng cảnh báo), co-operative sẽ không đảm bảo phản ứng tức thời như preemptive.

3.3.2. Fixed Prioritized Pre-emptive Scheduling with Time Slicing

a) Giới thiệu mô hình

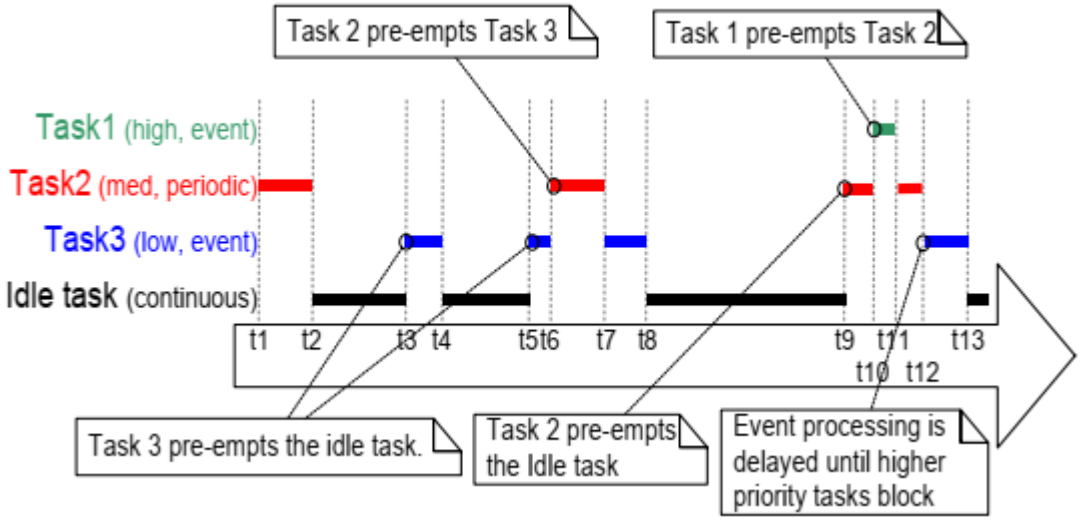
Định nghĩa:

- ***Ưu tiên cố định (Fixed Priority)***: Các thuật toán lập lịch được mô tả là "ưu tiên cố định" sẽ không thay đổi mức ưu tiên đã gán cho các tác vụ đang được lập lịch. Tuy nhiên, điều này không ngăn cản các tác vụ tự thay đổi mức ưu tiên của chính mình hoặc của các tác vụ khác.
- ***Chiếm quyền (Pre-emptive)***: Các thuật toán lập lịch chiếm quyền sẽ ngay lập tức "chiếm quyền" tác vụ đang ở trạng thái Running nếu có một tác vụ có mức ưu tiên cao hơn chuyển sang trạng thái Ready. Bị chiếm quyền nghĩa là tác vụ đang chạy sẽ bị buộc (một cách không tự nguyện – không phải do

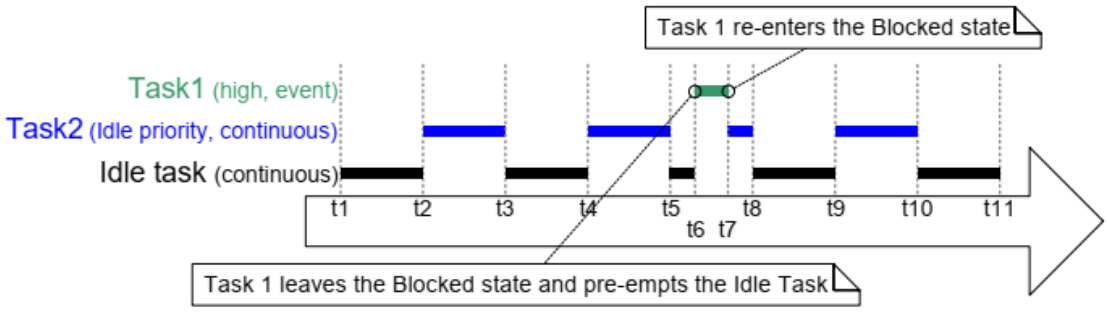
tự yield hay block) phải rời khỏi trạng thái Running và chuyển về trạng thái Ready, để một tác vụ khác (ưu tiên cao hơn) được chạy.

- ***Chia thời gian (Time Slicing)***: Chia thời gian được sử dụng để chia sẻ thời gian xử lý giữa các tác vụ có cùng mức ưu tiên, ngay cả khi các tác vụ đó không tự yield hoặc không rơi vào trạng thái Blocked. Thuật toán lập lịch sử dụng chia thời gian sẽ chọn một tác vụ mới để chuyển vào trạng thái Running sau mỗi chu kỳ chia thời gian, nếu có các tác vụ khác trong trạng thái Ready có cùng mức ưu tiên với tác vụ đang chạy. Một chu kỳ chia thời gian chính là khoảng thời gian giữa hai ngắt tick của RTOS.

Cơ chế hoạt động:



Hình 3.9. *Mẫu thực thi mô tả việc ưu tiên và chiếm quyền giữa các tác vụ*



Hình 3.10. *Mẫu thực thi mô tả việc ưu tiên tác vụ và phân chia thời gian*

- Mỗi task có một giá trị ưu tiên không thay đổi trong suốt thời gian sống của nó (trừ khi được lập trình thay đổi). Task có ưu tiên cao hơn luôn được chọn chạy trước task có ưu tiên thấp hơn.
- Khi một task có ưu tiên cao hơn chuyển sang trạng thái Ready (ví dụ: vừa hết delay hoặc vừa nhận dữ liệu), nó sẽ lập tức chiếm CPU, ngắt tác vụ đang chạy (nếu ưu tiên thấp hơn).

- Nếu có nhiều task cùng mức ưu tiên ở trạng thái Ready, hệ thống sẽ luân phiên chọn từng task chạy theo chu kỳ chia thời gian (thường là 1 tick – ví dụ 1ms). Giúp công bằng xử lý giữa các task nền không ưu tiên cao.

Ưu điểm:

- Phản hồi nhanh cho tác vụ quan trọng: Ưu tiên cao giúp các tác vụ quan trọng được thực thi ngay lập tức, giảm độ trễ.
- Chống độc chiếm CPU: Time slicing ngăn chặn việc bất kỳ tác vụ nào (kể cả ưu tiên thấp) độc chiếm CPU, duy trì tính công bằng và tương tác.
- Phù hợp với hệ thống thời gian thực (RTOS): Đây là phương pháp phổ biến và hiệu quả cho các hệ thống đòi hỏi độ tin cậy và dự đoán cao về thời gian.
- Khả năng dự đoán: Có thể dự đoán được thời gian thực thi của các tác vụ, quan trọng cho các hệ thống thời gian thực cứng.

Nhược điểm:

- Nguy cơ "đói" (Starvation): Các tác vụ ưu tiên thấp có thể không bao giờ được thực thi nếu luôn có tác vụ ưu tiên cao hơn sẵn sàng.
- Chi phí chuyển đổi ngữ cảnh (Overhead): Việc chuyển đổi giữa các tác vụ thường xuyên tiêu tốn tài nguyên CPU và bộ nhớ.
- Khó xác định ưu tiên tối ưu: Việc gán mức ưu tiên cố định một cách chính xác có thể phức tạp và dễ gây ra hiệu suất không mong muốn nếu gán sai.

Mô hình "Fixed Prioritized Pre-emptive Scheduling with Time Slicing" là lựa chọn tối ưu cho hầu hết các ứng dụng FreeRTOS, mang lại sự cân bằng giữa khả năng phản hồi tức thì cho các tác vụ quan trọng và sự công bằng trong việc chia sẻ tài nguyên cho các tác vụ có cùng mức độ ưu tiên.

b) Xây dựng mô hình

Để xây dựng mô hình này trên FreeRTOS, ta cần phải thiết kế các tác vụ (Tasks) với các mức độ ưu tiên khác nhau, đồng thời sử dụng các cơ chế đồng bộ hóa như Mutex và Queue để đảm bảo an toàn dữ liệu và giao tiếp giữa các tác vụ.

Các nguyên tắc chung khi xác định mức độ ưu tiên:

- Quan trọng nhất và có độ trễ thấp nhất: ***Ưu tiên cao nhất***
- Có tính phản hồi nhanh: ***Ưu tiên cao***
- Tác vụ định kỳ chặt chẽ: ***Ưu tiên tương đối cao***
- Tác vụ không quá nhạy cảm về thời gian: ***Ưu tiên thấp hơn***
- Nên tránh nhiều tác vụ nặng có cùng mức ưu tiên nếu không thực sự cần Time-Slicing để tránh overhead chuyển ngữ cảnh không cần thiết.

Các thành phần chính:

- ***Tác vụ (Task)***: Ngoài 5 tác vụ chính đã phân tích từ trước, TaskISR (tác vụ phân tích lệnh) được bổ sung nhằm mục đích hỗ trợ Task1 (tác vụ xử lý/thực thi lệnh), giúp Task1 trở nên đơn giản và dễ kiểm soát hơn. TaskISR có mức ưu tiên cao nhất sau Task1 để đảm bảo lệnh được thực thi nhanh nhất.

Bảng 3.8. *Xây dựng tác vụ*

Task	Mô tả

TaskISR: UART_ParseCommandTask	Ưu tiên cao nhất (osPriorityHigh1). Nhiệm vụ chính là đọc dữ liệu từ UART khi có ngắt nhận dữ liệu (RxCpltCallback), phân tích lệnh nhận được và gửi lệnh đã phân tích vào một hàng đợi (CommandQueue). Task này được kích hoạt bởi một cờ sự kiện (Event Flag) từ ngắt UART.
Task1: UART_CommandHandlerTask	Ưu tiên cao nhất (osPriorityRealtime). Nhận các lệnh từ CommandQueue và xử lý chúng, bao gồm việc cập nhật trạng thái hoạt động của động cơ , hướng quay, tốc độ, chế độ hiển thị LCD, và cập nhật chu kỳ các Task. Task này sẽ sử dụng Mutex để bảo vệ các biến toàn cục chia sẻ.
Task2: MotorControlTask	Ưu tiên cao (osPriorityHigh). Dựa trên các biến trạng thái điều khiển động cơ (Mode, Direction, SpeedValue) được cập nhật bởi Task1, task này sẽ điều khiển động cơ thông qua PWM và các chân GPIO. Task này cũng sử dụng Mutex để truy cập các biến điều khiển. Chu kỳ hoạt động của task này có thể được điều chỉnh thông qua lệnh "!ptask2".
Task3: SensorReadTask	Ưu tiên trên mức trung bình (osPriorityAboveNormal). Đọc dữ liệu nhiệt độ và độ ẩm từ cảm biến AHT10 định kỳ. Dữ liệu cảm biến sau khi đọc sẽ được lưu vào biến toàn cục SensorData và được bảo vệ bằng Mutex. Chu kỳ hoạt động của task này có thể được điều chỉnh thông qua lệnh "!ptask3".
Task4: LCD_DisplayTask	Ưu tiên trung bình (osPriorityNormal). Hiển thị dữ liệu nhiệt độ và độ ẩm lên màn hình LCD dựa trên chế độ hiển thị hiện tại (ALL, TEMP, HUMID). Task này đọc dữ liệu cảm biến và chế độ hiển thị từ các biến toàn cục được bảo vệ bởi Mutex. Chu kỳ hoạt động của task này có thể được điều chỉnh thông qua lệnh "!ptask4".
Task5: UART_ReportSendTask	Ưu tiên trung bình (osPriorityNormal). Gửi định kỳ dữ liệu cảm biến hiện tại qua UART để báo cáo trạng thái hệ thống. Task này cũng sử dụng Mutex để đảm bảo quyền truy cập UART độc quyền khi gửi dữ liệu. Chu kỳ hoạt động của task này có thể được điều chỉnh thông qua lệnh "!ptask5".

- ***Hàng đợi (Queue)***:

Bảng 3.9. *Xây dựng hàng đợi*

Queue	Mô tả
CommandQueue	Dùng để gửi các lệnh đã phân tích từ TaskISR đến Task1.

- ***Khóa (Mutex)***:

Bảng 3.10. *Xây dựng khóa bảo vệ*

Mutex	Mô tả
SensorDataMutex	Bảo vệ biến *SensorData* (nhiệt độ, độ ẩm) để tránh xung đột khi Task3 ghi và Task4, Task5 đọc.
DisplayModeMutex	Bảo vệ biến *currentModeLCD* để tránh xung đột khi Task1 ghi và Task4 đọc.
UART_AccessMutex	Bảo vệ tài nguyên UART để đảm bảo chỉ một task được gửi dữ liệu qua UART tại một thời điểm, tránh việc trộn lẫn dữ liệu giữa các task TaskISR (khi in thông báo lệnh) và Task5.
SpeedValueMutex, DirectionMutex, MotorModeMutex	Bảo vệ các biến trạng thái điều khiển động cơ (*SpeedValue*, *Direction*, *Mode*) để đảm bảo tính nhất quán khi Task1 cập nhật và Task2 đọc.

- ***Cờ sự kiện (Event Flag)***:

Bảng 3.11. *Xây dựng cờ sự kiện*

Event Flag	Mô tả
ISR_RxUARTHandle (FlagISR)	Được sử dụng bởi ngắt UART (HAL_UART_RxCpltCallback) để thông báo cho TaskISR rằng một lệnh hoàn chỉnh đã được nhận.

c) Lập trình mô hình

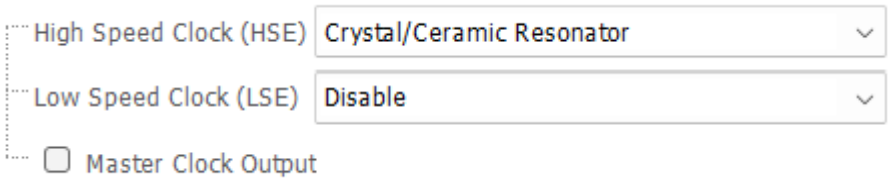
Khởi tạo hệ thống: (Clock là 72MHz)

System Core:

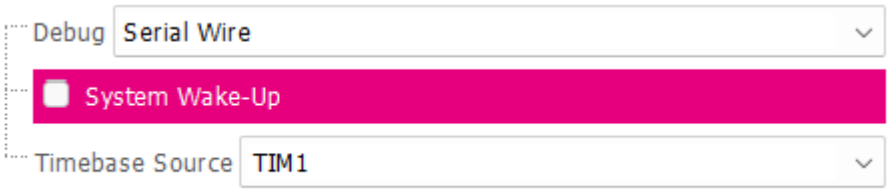
- GPIO: Cấu hình 2 chân PA0, PA1 thành Output

Pin N...	Signal...	GPIO ...	GPIO ...	GPIO ...	Maxi...	User L...	Modified
PA0-W...	n/a	Low	Output ...	No pull-...	Low	IN1	☑
PA1	n/a	Low	Output ...	No pull-...	Low	IN2	☑

- RCC: Chọn HSE là Crystal

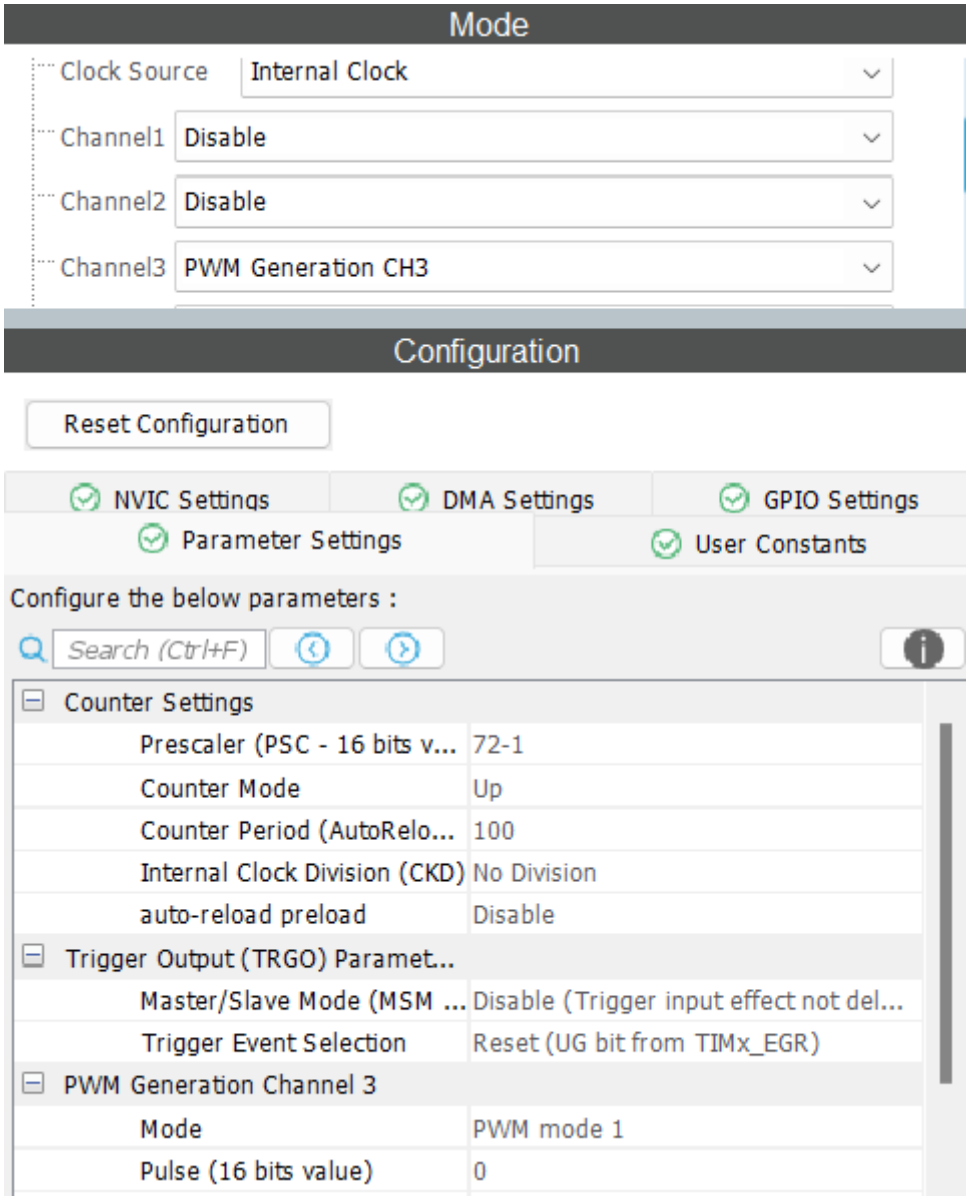


- SYS: Chọn Debug là Serial Wire và Timebase source khác SysTick



Timers:

- TIM2: Chọn Clock Source là Internal Clock và sử dụng Channel3 tạo xung PWM. Cấu hình Prescale là 72-1, Counter Mode là Up, và Counter Period là 100



Connectivity:

- I2C1: Chọn I2C. Mode là Standard cho LCD
- I2C2: Chọn I2C. Mode là Fast Mode cho AHT10
- USART1: Chọn Mode là Asynchronous. Cấu hình thông số và bật global interrupt

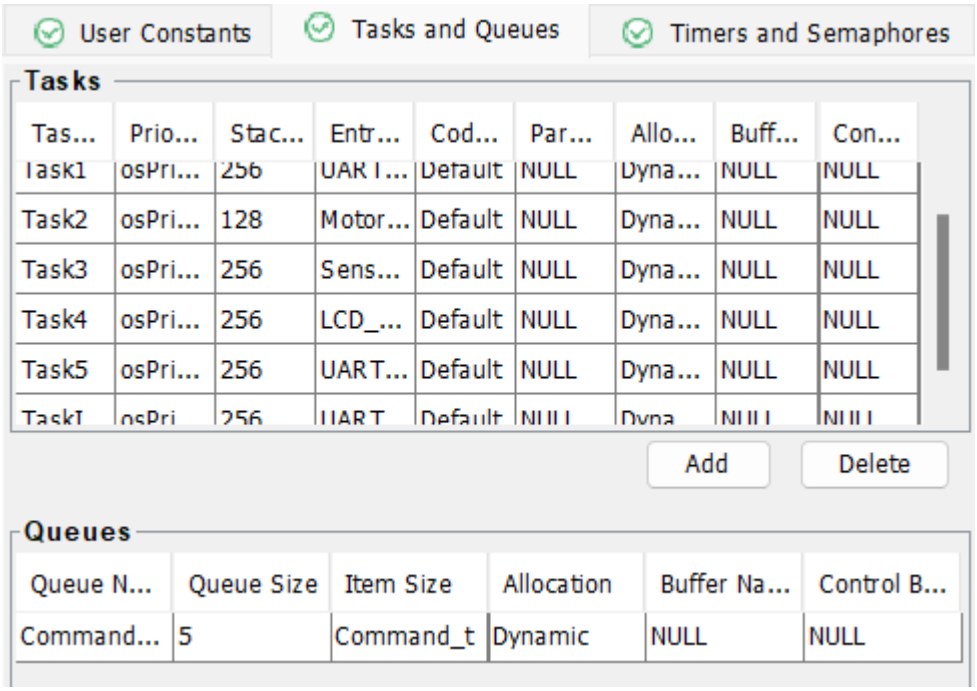
Basic Parameters	
Baud Rate	115200 Bits/s
Word Length	8 Bits (including Parity)
Parity	None
Stop Bits	1
Advanced Parameters	
Data Direction	Receive and Transmit
Over Sampling	16 Samples

FreeRTOS: Chọn Interface là CMSIS_V2

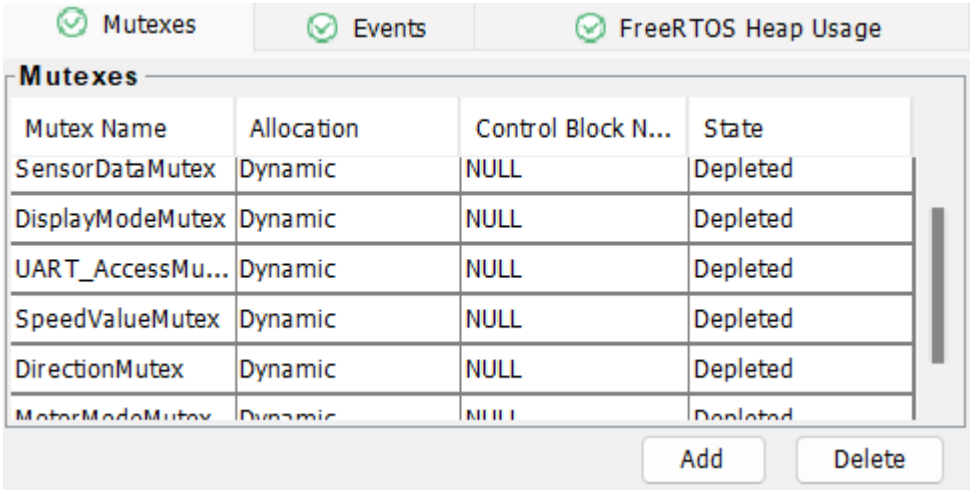
- Config parameters: Cấu hình cho mô hình Preemptive với Time Slicing là 1ms (1000Hz)

Kernel settings	
USE_PREEMPTION	Enabled
CPU_CLOCK_HZ	SystemCoreClock
TICK_RATE_HZ	1000

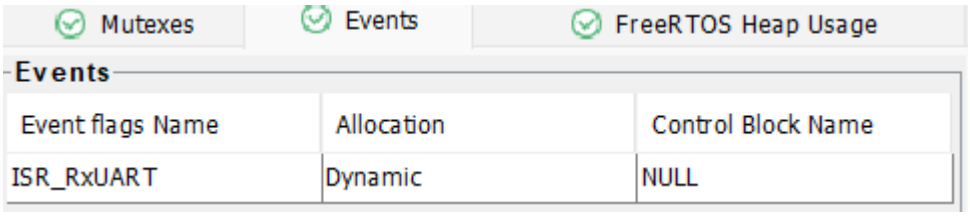
- Tasks and Queues: Tạo 6 tác vụ và 1 hàng đợi như đã xây dựng



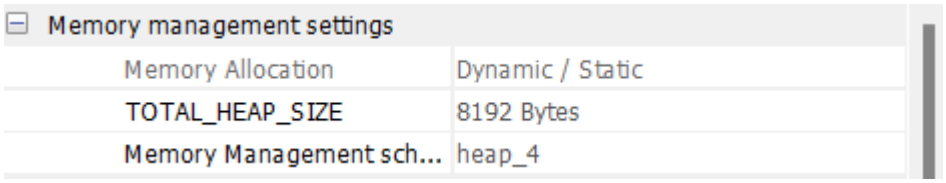
- Mutexes: Tạo 6 khóa bảo vệ như đã xây dựng



- Events: Tạo 1 cờ sự kiện như đã phân tích

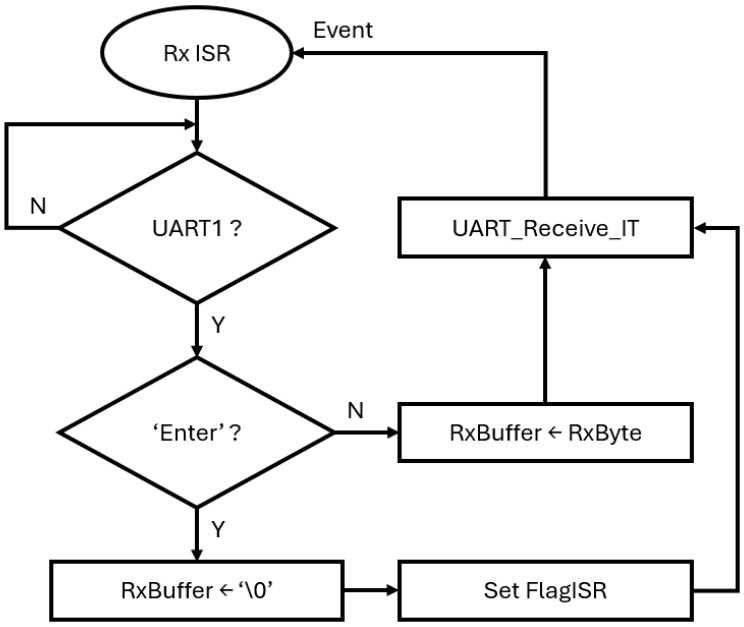


- FreeRTOS Heap Usage: Sau khi lập trình các tác vụ thì đã xác định được Stack Size tương ứng từng tác vụ, từ đó phần mềm đã tính toán được tổng bộ nhớ Heap sử dụng là 7025 Bytes. Ta cấp cho OS dung lượng RAM là 8MB



Lập trình các hàm và tác vụ:

- Hàm ***HAL_UART_RxCpltCallback***: Hàm được kích hoạt khi có ngắt UART Rx. Nó sẽ kiểm tra và lưu trữ byte nhận được dựa vào phím ‘Enter’ (<CR>). Nếu nhận được ‘Enter’ thì kết thúc câu lệnh và bật cờ sự kiện FlagISR.



- Hàm ***MOTOR_SetSpeed***:

```
void MOTOR_SetSpeed(uint8_t speed)
{
    if(speed > 100) {
        speed = 100;
    }

    __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_3, speed);
}
```

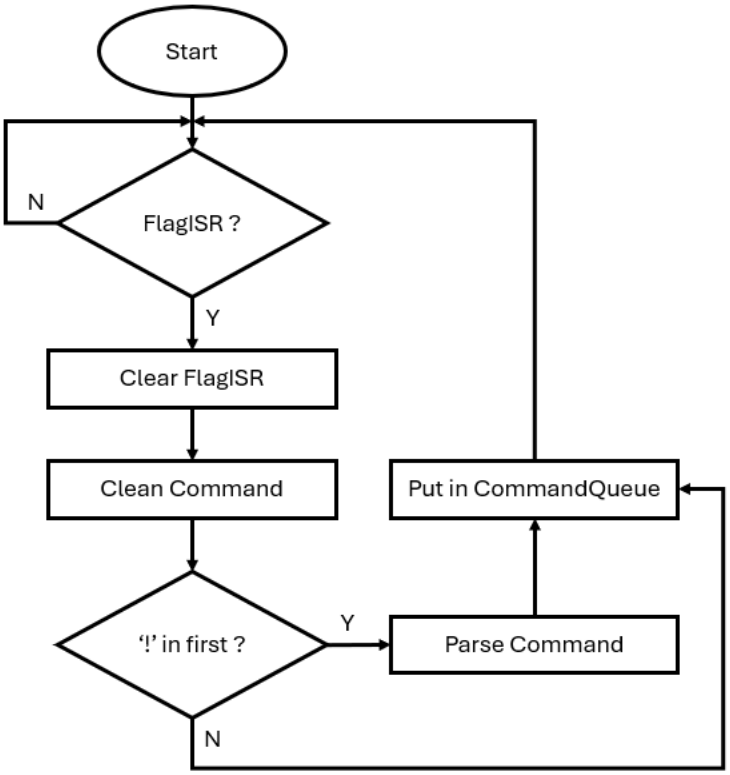
- Hàm ***MOTOR_SetDirection***:

```
void MOTOR_SetDirection(CommandParameter_t rotate)
{
    if(rotate == CW) {
        HAL_GPIO_WritePin(IN1_GPIO_Port, IN1_Pin, GPIO_PIN_SET);
        HAL_GPIO_WritePin(IN2_GPIO_Port, IN2_Pin, GPIO_PIN_RESET);
    }
    else if(rotate == CCW) {
        HAL_GPIO_WritePin(IN1_GPIO_Port, IN1_Pin, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(IN2_GPIO_Port, IN2_Pin, GPIO_PIN_SET);
    }
}
```

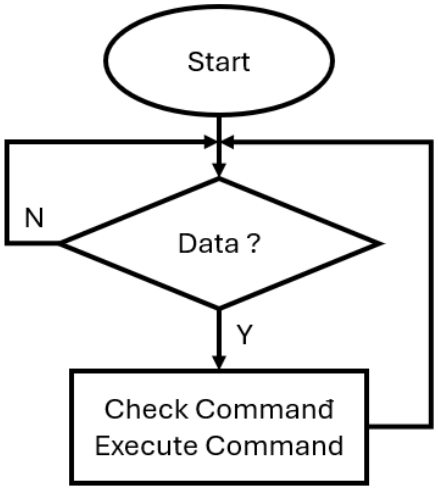
- Hàm ***MOTOR_Stop***:

```
void MOTOR_Stop(void)
{
    MOTOR_SetSpeed(0);
    HAL_GPIO_WritePin(IN1_GPIO_Port, IN1_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(IN2_GPIO_Port, IN2_Pin, GPIO_PIN_RESET);
}
```

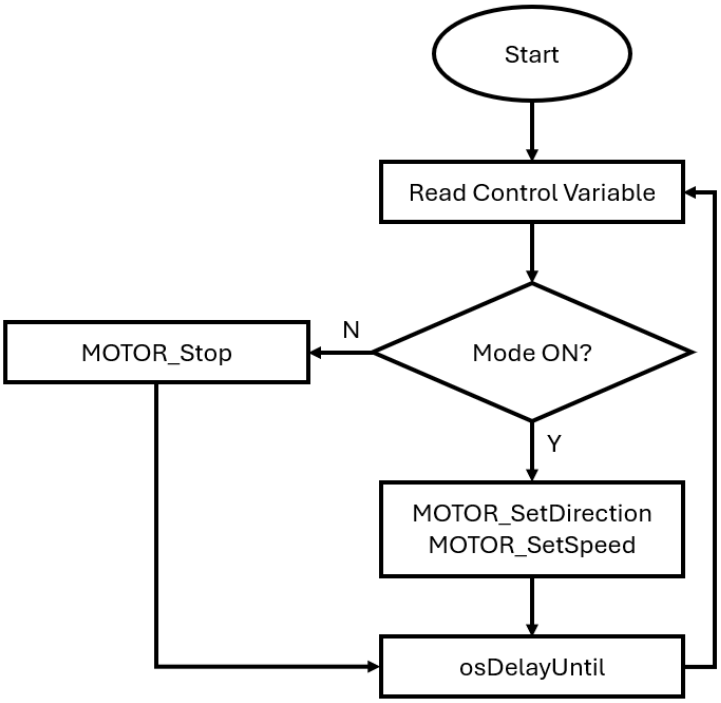
- TaskISR: Tác vụ chạy khi cờ sự kiện FlagISR được bật. Sau khi kích hoạt thì tác vụ sẽ xóa cờ sự kiện này, tiếp theo là phần đọc và phân tích lệnh dựa vào các hàm như strtok_r(), strcmp(), sscanf(), rồi gửi lệnh đã phân tích vào một hàng đợi để task xử lý lệnh tiếp nhận.



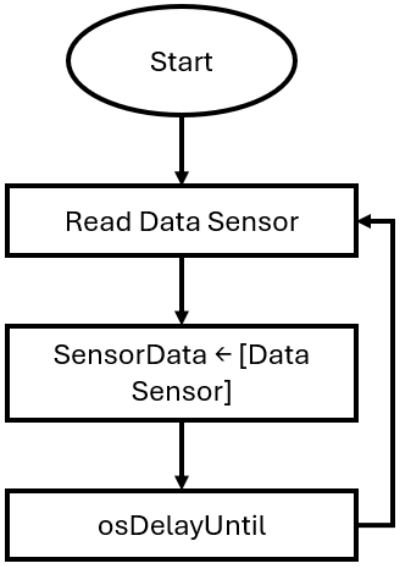
- Task1: Tác vụ chạy khi trong hàng đợi CommandQueue có dữ liệu. Sau khi được kích hoạt, tác vụ sẽ tiến hành xử lý và thực thi câu lệnh dựa trên cấu trúc switch-case.



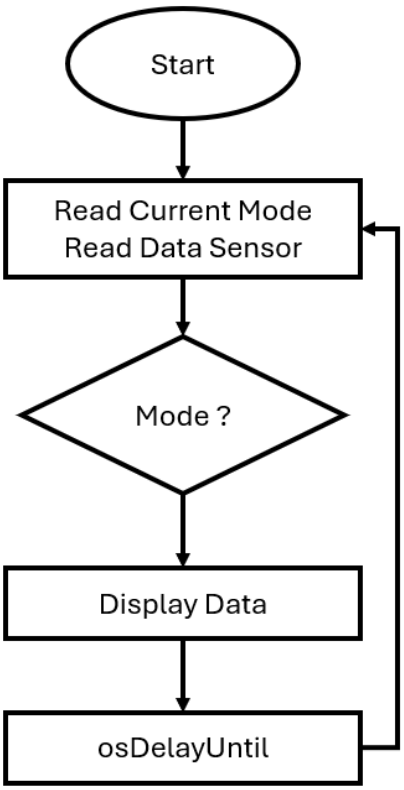
- Task2: Tác vụ chạy định kỳ (chu kỳ có thể điều chỉnh được) dựa vào hàm osDelayUntil(). Tác vụ sẽ tiến hành đọc biến điều khiển (có thể cập nhật) và thực hiện diều khiển động cơ bằng các hàm MOTOR đã viết ở trên.



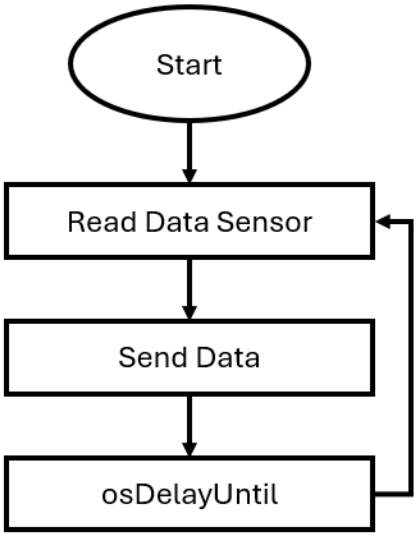
- Task3: Tác vụ chạy định kỳ (chu kỳ có thể điều chỉnh được) dựa vào hàm osDelayUntil(). Tác vụ sẽ gọi hàm đọc cảm biến trong thư viện, lấy thông số, và gán vào biến chung SensorData.



- Task4: Tác vụ chạy định kỳ (chu kỳ có thể điều chỉnh được) dựa vào hàm osDelayUntil(). Tác vụ sẽ đọc biến điều khiển chế độ hiển thị và biến chung SensorData, dựa trên chế độ hiển thị sẽ tiến hành sắp xếp dữ liệu và gửi vào LCD qua các hàm trong thư viện



- Task5: Tác vụ chạy định kỳ (chu kỳ có thể điều chỉnh được) dựa vào hàm osDelayUntil(). Tác vụ sẽ đọc biến chung SensorData và gửi thông số lên máy tính qua UART



- Kernel sẽ chịu trách nhiệm chính trong việc điều phối, quản lý và cấp phát tài nguyên CPU cho các tác vụ theo quy tắc lập lịch đã định (ưu tiên cố định, chiếm quyền, chia thời gian), đồng thời cung cấp các công cụ để các tác vụ giao tiếp và đồng bộ hóa với nhau một cách an toàn

d) Thử nghiệm và đánh giá

Mục tiêu thử nghiệm:

- Xác minh hoạt động đa nhiệm của các task với các mức ưu tiên khác nhau.
- Kiểm tra tính đúng đắn của việc điều khiển động cơ dựa trên lệnh UART.

- Kiểm tra việc đọc và hiển thị dữ liệu cảm biến.
- Đánh giá hiệu quả của các cơ chế đồng bộ hóa (Mutex, Queue, Event Flags) trong việc tránh xung đột và đảm bảo giao tiếp giữa các task.
- Kiểm tra khả năng thay đổi chu kỳ hoạt động của các task qua lệnh UART.

Phương pháp thử nghiệm:

- Nạp chương trình: Biên dịch và nạp mã nguồn vào vi điều khiển STM32.
- Giám sát UART: Sử dụng một chương trình terminal Hercules để gửi lệnh và nhận báo cáo từ UART.
- Quan sát động cơ và LCD: Trực tiếp quan sát hoạt động của động cơ và thông tin hiển thị trên màn hình LCD.
- Sử dụng Debugger: Trong môi trường phát triển (ví dụ: STM32CubeIDE), sử dụng debugger để theo dõi trạng thái của các task, queue, mutex và các biến toàn cục.

Các trường hợp thử nghiệm:

Điều khiển động cơ:

- Gửi lệnh !motor on và !speed 75 để bật động cơ với tốc độ 75%. Quan sát động cơ quay và tốc độ có thay đổi.
- Gửi lệnh !rotate ccw để thay đổi hướng quay. Quan sát động cơ đổi chiều.
- Gửi lệnh !motor off để dừng động cơ. Quan sát động cơ dừng.
- Gửi lệnh tốc độ vượt quá giới hạn (ví dụ: !speed 120). Kiểm tra xem tốc độ có bị giới hạn ở 100% không.

Hiển thị LCD:

- Quan sát LCD mặc định hiển thị cả nhiệt độ và độ ẩm.
- Gửi lệnh !display temp. Kiểm tra LCD chỉ hiển thị nhiệt độ.
- Gửi lệnh !display humid. Kiểm tra LCD chỉ hiển thị độ ẩm.
- Gửi lệnh !display all. Kiểm tra LCD hiển thị lại cả hai.

Báo cáo UART:

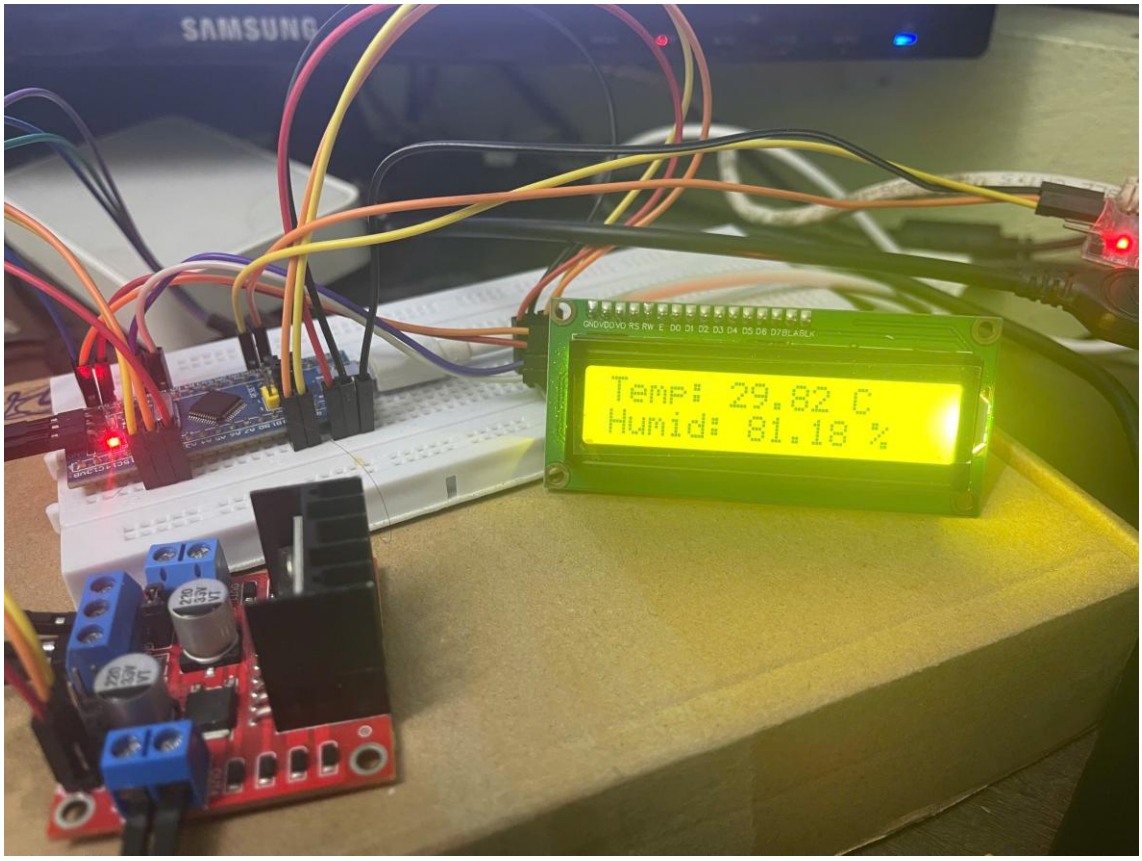
- Quan sát terminal hiển thị định kỳ dữ liệu cảm biến nhiệt độ và độ ẩm.
- Gửi liên tục các lệnh điều khiển qua UART để kiểm tra xem báo cáo cảm biến có bị gián đoạn hoặc lỗi định dạng không.

Điều chỉnh chu kỳ task:

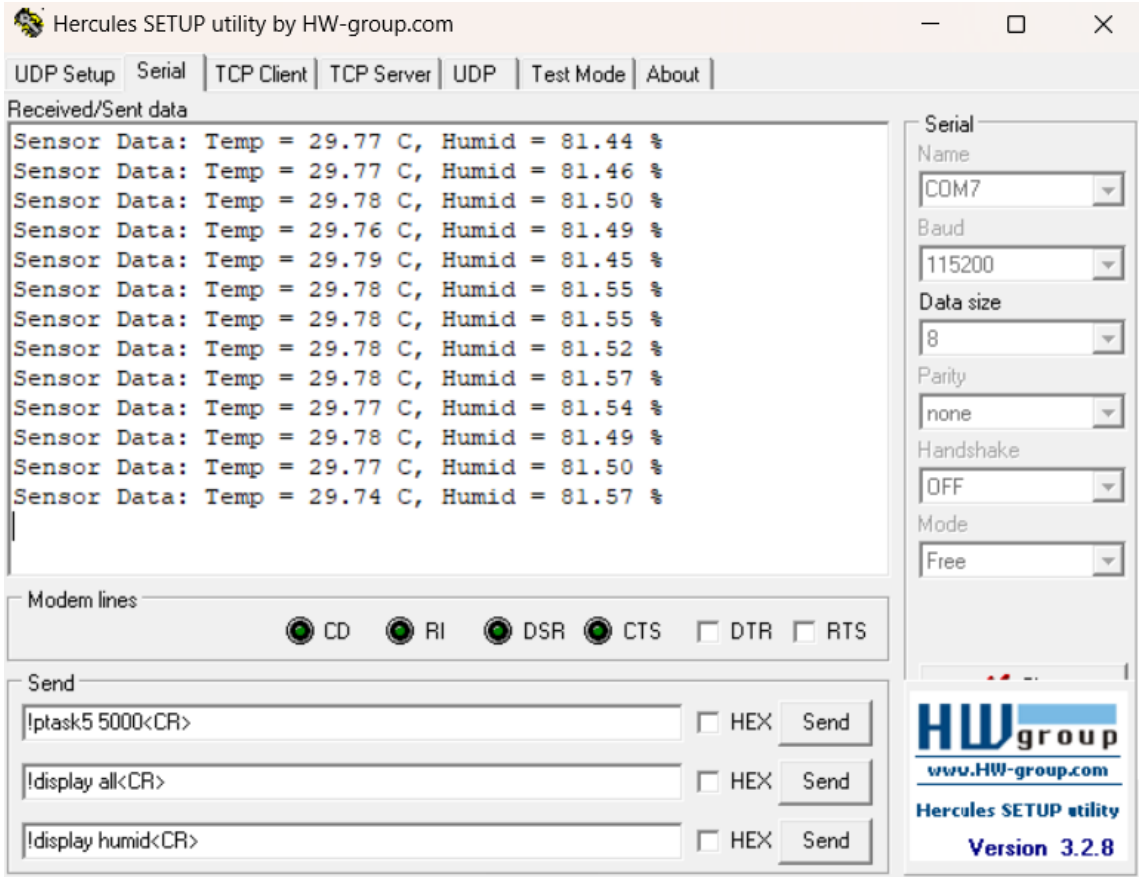
- Gửi !ptask2 500 để thay đổi chu kỳ của MotorControlTask xuống 500ms. Quan sát sự thay đổi trong phản ứng của động cơ.
- Gửi !ptask3 1000 để thay đổi chu kỳ đọc cảm biến xuống 1000ms. Kiểm tra tốc độ cập nhật dữ liệu trên LCD và báo cáo UART.
- Gửi !ptask4 1000 để thay đổi chu kỳ cập nhật LCD lên 1000ms. Quan sát tốc độ làm mới màn hình LCD.
- Gửi !ptask5 5000 để thay đổi chu kỳ báo cáo UART lên 5000ms. Quan sát tần suất gửi báo cáo qua terminal.

Kết quả thử nghiệm:

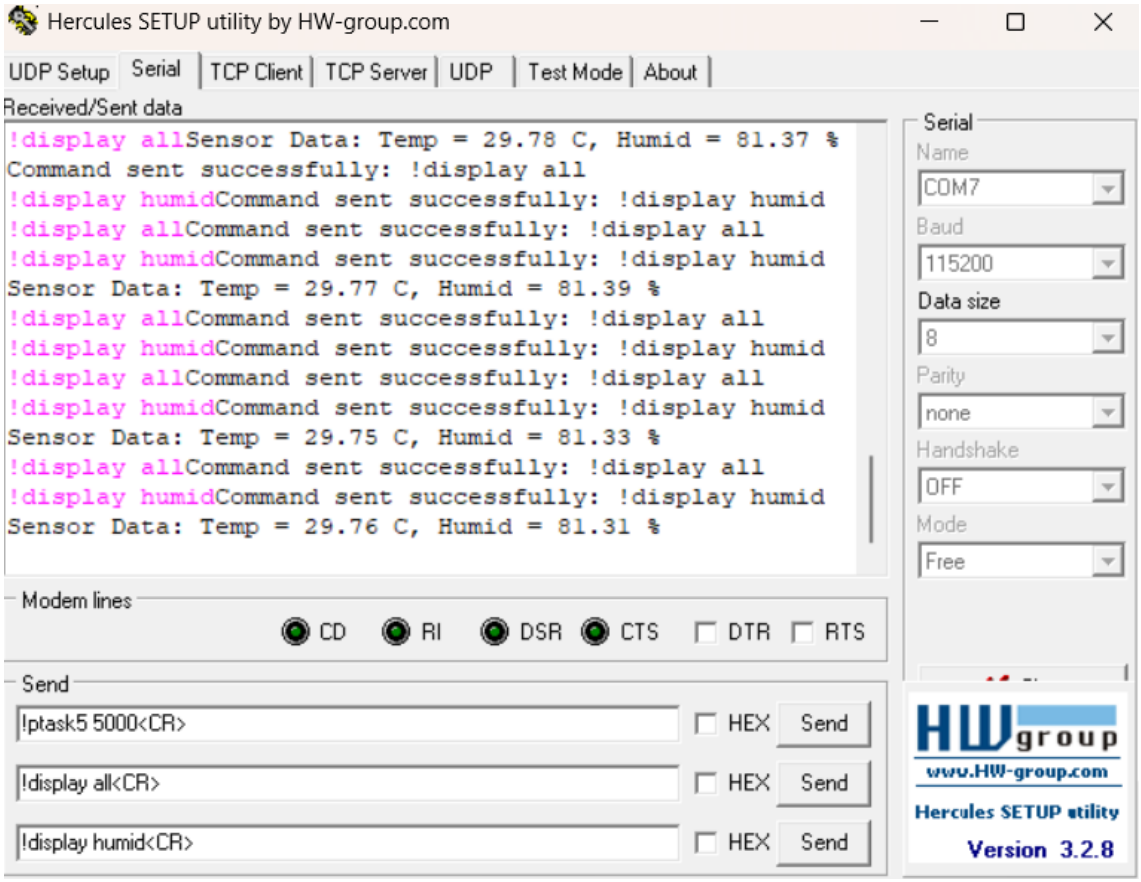
(Các chức năng khác sẽ được Demo thực tế vì chụp qua hình ảnh không phản ánh được quá trình thực thi.)



Hình 3.11. *Hiển thị LCD*



Hình 3.12. *Gửi dữ liệu lên máy tính*



Hình 3.13. *Gửi liên tục lệnh*

Đánh giá:

Các thử nghiệm đã được thực hiện một cách toàn diện, bao gồm kiểm tra hoạt động của động cơ, hiển thị LCD, báo cáo UART và điều chỉnh chu kỳ hoạt động của các task. Dựa trên các trường hợp thử nghiệm đã nêu và các kết quả đều đáp ứng mong đợi, có thể đưa ra đánh giá sau:

→ Xác minh hoạt động đa nhiệm của các task với các mức ưu tiên khác nhau:

- Các thử nghiệm điều khiển động cơ, hiển thị LCD, đọc cảm biến và báo cáo UART hoạt động đồng thời và độc lập, chứng tỏ khả năng hoạt động đa nhiệm của hệ thống.
- Việc thay đổi chu kỳ hoạt động của các task khác nhau thông qua lệnh UART (ví dụ: !ptask2 500, !ptask3 1000, v.v.) mà không gây ra xung đột hoặc lỗi hệ thống, xác nhận rằng các mức ưu tiên và cơ chế lập lịch của FreeRTOS đang hoạt động chính xác, cho phép các task với các ưu tiên khác nhau được thực thi một cách hiệu quả.

→ Kiểm tra tính đúng đắn của việc điều khiển động cơ dựa trên lệnh UART:

- Hệ thống đã điều khiển động cơ một cách chính xác theo các lệnh UART. Các lệnh bật/tắt động cơ, điều chỉnh tốc độ, và thay đổi hướng quay đều được thực thi đúng.
- Chức năng kiểm soát giới hạn tốc độ hoạt động như mong đợi, đảm bảo an toàn và tính ổn định của hệ thống.

→ Kiểm tra việc đọc và hiển thị dữ liệu cảm biến:

- Dữ liệu cảm biến nhiệt độ và độ ẩm đã được đọc thành công và hiển thị chính xác trên màn hình LCD ở chế độ mặc định.
- Các lệnh tùy chỉnh hiển thị hoạt động đúng, cho phép người dùng lựa chọn thông tin cần hiển thị, thể hiện tính linh hoạt trong giao diện người dùng.

→ Đánh giá hiệu quả của các cơ chế đồng bộ hóa (Mutex, Queue, Event Flags) trong việc tránh xung đột và đảm bảo giao tiếp giữa các task:

- Việc gửi liên tục các lệnh điều khiển qua UART mà không làm gián đoạn hoặc lỗi định dạng báo cáo cảm biến chứng tỏ các cơ chế đồng bộ hóa đang hoạt động hiệu quả. Điều này ngụ ý rằng các tài nguyên chia sẻ (ví dụ: UART, dữ liệu cảm biến) được bảo vệ đúng cách, ngăn chặn các tình huống race condition và đảm bảo tính toàn vẹn của dữ liệu.
- Sự ổn định của hệ thống khi nhiều task cùng truy cập hoặc gửi/nhận dữ liệu cho thấy việc sử dụng Mutex, Queue và Event Flags đã thành công trong việc điều phối luồng dữ liệu và tránh xung đột giữa các task có độ ưu tiên khác nhau.

→ Kiểm tra khả năng thay đổi chu kỳ hoạt động của các task qua lệnh UART:

- Khả năng thay đổi chu kỳ của các task thông qua lệnh UART đã được xác minh thành công. Sự thay đổi trong phản ứng của động cơ, tốc độ cập nhật LCD và tần suất báo cáo UART là bằng chứng rõ ràng.
- Tính năng này cho phép cấu hình hệ thống linh hoạt trong thời gian chạy, một yêu cầu quan trọng đối với các hệ thống nhúng cần thích ứng với các yêu cầu hiệu suất hoặc năng lượng khác nhau.

Kết luận: Dựa trên các kết quả thử nghiệm được báo cáo, hệ thống vi điều khiển STM32 đã đáp ứng đầy đủ tất cả các mục tiêu thử nghiệm đề ra. Hoạt động đa nhiệm, điều khiển động cơ chính xác, hiển thị cảm biến tin cậy, và khả năng cấu hình động chu kỳ task đều đã được xác nhận. Có thể kết luận mô hình ***Fixed Prioritized Pre-emptive Scheduling with Time Slicing*** đã được xây dựng và triển khai thành công, cho phép hệ thống điều khiển động cơ và giám sát cảm biến hoạt động một cách ổn định, phản hồi nhanh chóng với lệnh từ người dùng và cung cấp thông tin cập nhật định kỳ.

Thống kê công việc và tự đánh giá mức độ đóng góp của các thành viên

Nội dung công việc	1. Tìm hiểu các phần cứng, nguyên lý làm việc 2. Tìm hiểu các giao tiếp, vẽ sơ đồ kết nối phần cứng 3. Phân tích yêu cầu xử lý đảm bảo tính thời gian thực cho các đối tượng 4. Phân tích, xây dựng, lập trình 2 mô hình đơn nhiệm 5. Phân tích, xây dựng, lập trình 2 mô hình đa nhiệm
Mức độ đóng góp	**Bùi Quốc Doanh**: Hoàn thành phân tích, xây dựng lập trình 2 mô hình đơn nhiệm Time-Trigger Cyclic Executive Scheduler, Non-Preemptive Event-Triggered Scheduling và một mô hình đa nhiệm Co-operative Scheduling. **Nguyễn Trung Dũng**: Phân tích và lập lịch mô hình đa nhiệm Fixed Prioritized Pre-emptive Scheduling with Time Slicing. **Lê Thái Phát**: Hoàn thành tìm hiểu nguyên lí làm việc các module phần cứng, nguyên lí làm việc các ngoại vi của STM32, thiết kế phần cứng, phân tích yêu cầu xử lý đảm bảo tính thời gian thực cho các đối tượng.
Tự đánh giá điểm	**Bùi Quốc Doanh**: 9 điểm **Nguyễn Trung Dũng**: 8.5 điểm **Lê Thái Phát**: 8 điểm

Link github các dự án đã thực hiện:

- **Code đơn nhiệm và đa nhiệm**
- **Dự án robot dò line sử dụng PID**
- **Dự án mạng cảm biến đo nhiệt độ, độ ẩm không khí**
- Project Pre-emptive Scheduling Multitasking

TÀI LIỆU THAM KHẢO

[1]. Slide bài giảng môn thiết kế hệ thống nhúng

[2]. Mastering_the_FreeRTOS_Real_Time_Kernel-A_Hands-On_Tutorial_Guide

[3]. R. Barry, *FreeRTOS Reference Manual – API Functions and Configuration Options*, Real Time Engineers Ltd., 2022. [Online]. Available: https://www.freertos.org

[4]. STMicroelectronics, *STM32F1 Series Reference Manual*, RM0008, Rev. 21, May 2020. [Online]. Available: https://www.st.com/resource/en/reference_manual/rm0008-stm32f101xx-stm32f102xx-stm32f103xx-stm32f105xx-and-stm32f107xx-advanced-armbased-32bit-mcus-stmicroelectronics.pdf

[5]. Hitachi, *HD44780U (LCD Controller/Driver) Datasheet*, 2001. [Online]. Available: https://www.sparkfun.com/datasheets/LCD/HD44780.pdf

[6]. T. Noergaard, Embedded Systems Architecture: A Comprehensive Guide for Engineers and Programmers, 2nd ed., Newnes, 2012.

[7]. N. Storey, Real-Time Embedded Systems: Design Principles and Engineering Practices, Pearson, 2020.

[8]. ARM Ltd., *Cortex-M3 Technical Reference Manual*, 2021. [Online]. Available: https://developer.arm.com/documentation/ddi0337/latest/

[9]. STMicroelectronics, *STM32CubeF1 Firmware Package*, [Online]. Available: https://www.st.com/en/embedded-software/stm32cubef1.html