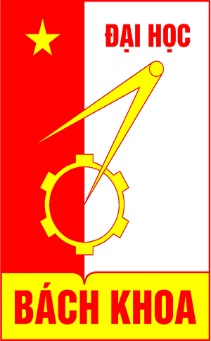
**ĐẠI HỌC BÁCH KHOA HÀ NỘI**

**TRƯỜNG ĐIỆN – ĐIỆN TỬ**



**BÁO CÁO BÀI TẬP LỚN**

**Đề tài :** *Kiểm soát thiết bị điện dựa trên nhiệt độ, ánh sáng*

***Môn học*** Thiết kế hệ thống nhúng­­

***Giảng viên hướng dẫn*** TS. Nguyễn Cảnh Quang

***Nhóm thực hiện*** Nhóm 7

Trương Nho Tuấn 20200566

Nguyễn Cảnh Anh

Nguyễn Đỗ Hồng Phương

*Hà Nội, 12/2023*

**MỤC LỤC**

[DANH MỤC BẢNG BIỂU 3](#_Toc155547479)

[CHƯƠNG 1. TỔNG QUAN 4](#_Toc155547480)

[1.1 Yêu cầu bài tập lớn 4](#_Toc155547481)

[1.2 Mục tiêu 4](#_Toc155547482)

[CHƯƠNG 2. Giới thiệu phần cứng 5](#_Toc155547483)

[2.1 Nguyên lý các module phần cứng 5](#_Toc155547484)

[2.1.1 Kit STM32F103C8T6 Blue Pill ARM Cotex-M3 5](#_Toc155547485)

[2.1.2 GPIO 6](#_Toc155547486)

[2.1.3 ADC+DMA 8](#_Toc155547487)

[2.1.4 UART 10](#_Toc155547488)

[2.1.5 PWM 12](#_Toc155547489)

[2.1.6 I2C 13](#_Toc155547490)

[2.2 Sơ đồ phần cứng 17](#_Toc155547491)

[CHƯƠNG 3. PHẦN MỀM 18](#_Toc155547492)

[3.1 Hệ điều hành thời gian thực 18](#_Toc155547493)

[3.1.1 Tổng quan 18](#_Toc155547494)

[3.1.2 Hệ điều hành FreeRTOS 20](#_Toc155547495)

[3.2 Super Loop 21](#_Toc155547496)

[CHƯƠNG 4. KẾT QUẢ VÀ ĐÁNH GIÁ 22](#_Toc155547497)

[4.1 Kết quả đạt được 22](#_Toc155547498)

[4.2 Đánh giá 23](#_Toc155547499)

[4.3 Phân công công việc 24](#_Toc155547500)

**DANH MỤC HÌNH ẢNH**

[Hình 2.1 Kit STM32F103C8T6 5](#_Toc155547501)

[Hình 2.2 Minh họa đọc dht22 8](#_Toc155547502)

[Hình 2.3 Minh họa đọc DHT22 8](#_Toc155547503)

[Hình 2.4 Bảng chi tiếp ngoại vi DMA trên stm32f1 9](#_Toc155547504)

[Hình 2.5 Giao tiếp UART thông qua 2 chân TX , RX 10](#_Toc155547505)

[Hình 2.6 Khung truyền UART 10](#_Toc155547506)

[Hình 2.7 Modul CH340 USB to TTL 12](#_Toc155547507)

[Hình 2.8 Minh họa giá trị điện áp tải và thời gian đóng/mở van 13](#_Toc155547508)

[Hình 2.9 Mô hình I2C Master - Slave 14](#_Toc155547509)

[Hình 2.10 Màn hình LCD và modul I2C 16](#_Toc155547510)

[Hình 2.11 Sơ đồ phần cứng – các ngoại vi 17](#_Toc155547511)

[Hình 3.1 Lưu đồ của chương trình FreeRTOS 21](#_Toc155547512)

[Hình 3.2 Lưu đồ thuật toán chương trình super loop 21](#_Toc155547513)

[Hình 4.1 Thử nghiệm mạch trên testboard 22](#_Toc155547514)

[Hình 4.2. Kết quả vi điều khiển gửi về máy tính thông qua UART 23](#_Toc155547515)

# DANH MỤC BẢNG BIỂU

[Bảng 2.1 : Thông số kỹ thuật của chíp STM32F103C8T6 5](#_Toc155547516)

[Bảng 3.2 So sánh super Loop và RTOS 18](#_Toc155547517)

# TỔNG QUAN

## Yêu cầu bài tập lớn

* Tên dự án : Kiểm soát thiết bị điện thông qua nhiệt độ, ánh sáng
* Yêu cầu
* Điều khiển một vài các thiết bị điện như đèn led thông qua ánh sáng của môi trường
* Điều khiển tốc độ quay của quạt bằng tay
* Giám sát dữ liệu về nhiệt độ, độ ẩm trong phòng
* Hiển thị thông tin trực quan với người dùng

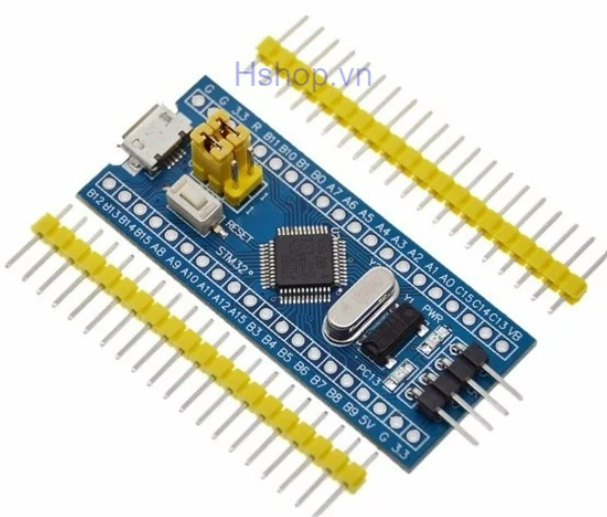
## Mục tiêu

* Lập trình lấy dữ liệu từ cảm biến nhiệt độ, điều khiển thiết bị và hiển thị thông qua mạch tự làm( sử dụng chíp stm32f103c8t6)
* Thiết kế chương trình dựa trên hệ điều hành thời gian thực

# Giới thiệu phần cứng

## Nguyên lý các module phần cứng

### Kit STM32F103C8T6 Blue Pill ARM Cotex-M3



Hình 2.1 Kit STM32F103C8T6

* STM32F103C8T6 là một bộ vi điều khiển 32 bit dựa trên hạt nhân Cortex-M3 , thuộc dòng STM 32 trong bộ vi điều khiển
* Vi điều khiển có dòng hiệu suất mật độ trung bình kết hợp lõi RISC hiệu suất cao với tần số hoạt động 72MHz
* Bo mạch dựa trên mạch MCU cơ bản nhất, mạch tinh thể 8M và 32768, mạch cấp nguồn USB.
* Bảng lõi được chia thành hai hàng dẫn đến tất cả các cổng I / O.
* Tất cả các thiết bị đều cung cấp hai ADC 12 bit, ba bộ định thời 16 bit đa năng cộng với một bộ hẹn giờ PWM, cũng như các giao diện giao tiếp tiêu chuẩn và nâng cao: lên đến hai I2C và SPI, ba USART, một USB và CAN..
* Với việc sử dụng giao diện Mirco USB, bạn có thể thực hiện giao tiếp USB và cấp nguồn, giao diện USB, tương thích với giao diện sạc điện thoại di động Andrews thông thường.
* Thông số kỹ thuật

Bảng . : Thông số kỹ thuật của chíp STM32F103C8T6

|  |  |
| --- | --- |
| Manufacturer | STMicroelectronics |
| Product Category | ARM Microcontrollers - MCU |
| Series | STM32F103C8 |
| Mounting Style | SMD/SMT |
| Package / Case | LQFP-48 |
| Core | ARM Cortex M3 |
| Program Memory Size | 64 kB |
| Data Bus Width | 32bit |
| ADC Resolution | 12bit |
| Maximum Clock Frequency | 72Mhz |
| Number of I/Os | 37 |
| Data RAM Size | 20Kb |
| Allowable operating temperature range | - 40 C -> + 85 C |
| Supply Voltage - Min | 2V |
| Supply Voltage - Max | 3.6V |
| Data RAM Type | SRAM |
| Interface Type | CAN, I2C, SPI, USART, USB |
| Moisture Sensitive | Yes |
| Processor Series | Processor Series |
| Product Type | ARM Microcontrollers - MCU |
| Program Memory Type | Flash |
| Subcategory | Microcontrollers - MCU |
| Tradename | STM32 |
| Unit Weight |  |

### GPIO

GPIO là một chân tín hiệu kỹ thuật số trên mạch tích hợp mà hành vi của nó (đầu vào hoặc đầu ra) được điều khiển bởi phần mềm ứng dụng. GPIO về cơ bản là một chân có thể được cấu hình làm đầu vào hoặc đầu ra. Nếu chúng ta cấu hình chân như một đầu ra, chúng ta có thể ghi 0 (LOW) hoặc 3,3 / 5 V (VDD) vào chân đó. Khi được cấu hình làm đầu vào, chúng ta có thể đọc tín hiệu trên chân đó. GPIO là giao diện tiêu chuẩn mà qua đó vi điều khiển có thể giao tiếp với thế giới bên ngoài. Nó có thể được sử dụng để đọc các giá trị từ cảm biến analog hoặc kỹ thuật số, điều khiển đèn LED, điều khiển đồng hồ cho giao tiếp I2C,...

* Ứng dụng giao tiếp với DHT22
* Nguồn sử dụng: 3~5VDC.
* Dòng sử dụng: 2.5mA max (khi truyền dữ liệu).
* Đo tốt ở độ ẩm 0100%RH với sai số 2-5%.
* Đo tốt ở nhiệt độ -40 to 80°C sai số ±0.5°C.
* Tần số lấy mẫu tối đa 0.5Hz (2 giây 1 lần)
* Kích thước 27mm x 59mm x 13.5mm (1.05" x 2.32" x 0.53")
* 4 chân, khoảng cách chân 0.1''.
* Nguyên tắc hoạt động
* Gửi yêu cầu đo tới DHT11, sau đó đợi DHT11 phản hồi.
* Khi DHT22 sẵn sàng giao tiếp, nó sẽ gửi lại 5 byte dữ liệu chứa giá trị nhiệt độ và độ ẩm do được.
* Giao tiếp:

***Bước 1:***

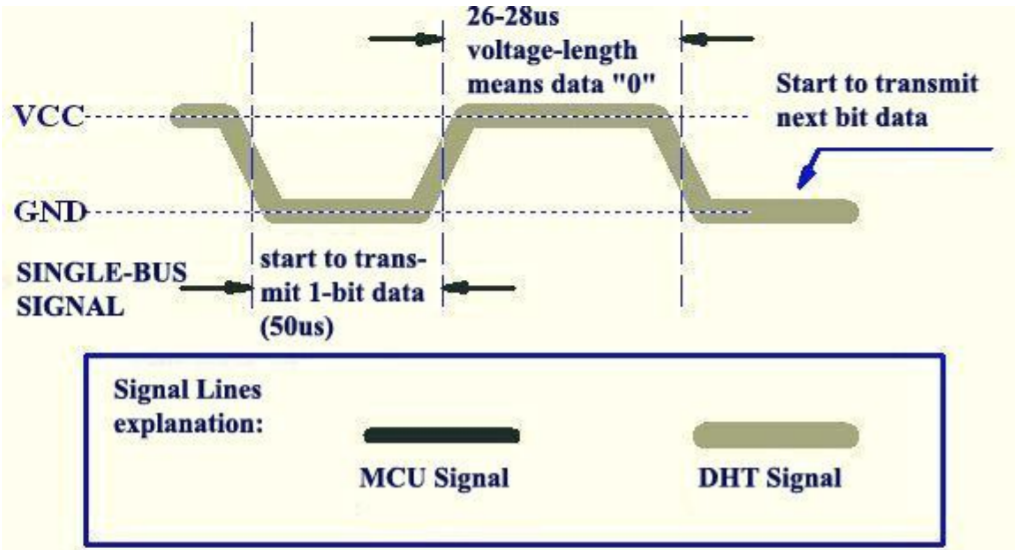
1. Đầu tiên vi điều khiển gửi tín hiệu Start: cấu hình chân DATA ở chế độ OUTPUT, kéo chân DATA xuống mức logic 0 trong khoảng thời gian tối thiểu 18 ms.
2. Vi điều khiển kéo chân DATA lên mức logic 1, sau đó cấu hình lại chế độ INPUT.
3. Sau khoảng thời gian 2040 s, DHT11 sẽ kéo chân DATA xuống mức logic 0. Trong trường hợp quá 40 s mà chân DATA vẫn chưa được kéo xuống mức logic 0 nghĩa là giao tiếp với DHT11 thất bại.
4. Chân DATA sẽ được giữ ở mức logic 0 thời gian 80 s sau đó nó được kéo lên mức logic 1 trong 80 s. Vi điểu khiển xác định việc giao tiếp thành công hay không với DHT11 qua chân DATA.

***Bước 2****:*

1. Giá trị nhiệt độ và độ ẩm đo được sẽ được DHT11 gửi về vi điều khiển dưới dạng 5 byte:
   1. Byte 1: giá trị phần nguyên của độ ẩm.
   2. Byte 2: giá trị phần thập phân của độ ẩm.
   3. Byte 3: giá trị phần nguyên của nhiệt độ.
   4. Byte 4: giá trị phần thập phân của nhiệt độ.
   5. Byte 5: byte checksum (nếu byte 5 bằng tổng giá trị 4 byte trên thì dữ liệu thu được là chính xác).
2. Sau khi xác nhận thành công yêu cầu giao tiếp với DHT11, nó sẽ gửi liên tiếp 40 bit về vi điều khiển:

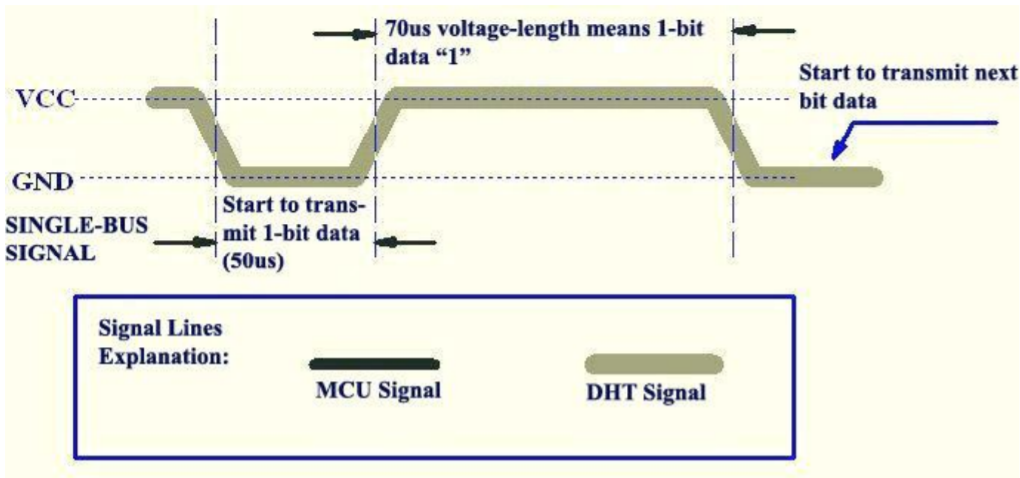
Với mỗi bit dữ liệu gửi về vi điều khiển, DHT11 sẽ kéo chân DATA xuống mức logic 0 trong 50 s, sau đó:

* Nó sẽ kéo chân DATA lên mức logic 1 và giữ trong 2628 s nếu dữ liệu truyền về vi điều khiển là bit 0.



Hình 2.2 Minh họa đọc dht22

* Nó sẽ kéo chân DATA lên mức logic 1 và giữ trong 70 µs nếu dữ liệu truyền về vi điều khiển là bit 1.



Hình 2.3 Minh họa đọc DHT22

### ADC+DMA

* 1. *ADC*

ADC là 1 mạch điện tử lấy điện áp tương tự làm đầu vào và chuyển đổi nó thành dữ liệu số (1 giá trị đại diện cho mức điện áp trong mã nhị phân)

* Một vài khái niệm cần chú ý
* Độ phân giải (resolution): dùng để chỉ số bit cần thiết để chứa hết các mức giá trị số (digital) sau quá trình chuyển đổi ở ngõ ra. Bộ chuyển đổi ADC của STM32F103C8T6 có độ phân giải mặc định là 12 bit, tức là có thể chuyển đổi ra 212= 4096 giá trị ở ngõ ra số.
* Thời gian lấy mẫu (sampling time): là khái niệm được dùng để chỉ thời gian giữa 2 lần số hóa của bộ chuyển đổi, thời gian lấy mẫu càng lâu độ chính xác càng cao. Nhìn vào đồ thị dưới ta sẽ thấy ADC cần 1 khoảng thời gian ổn định tSTAB trước khi bắt đầu chuyển đổi. Sau khi chuyển đổi xong cờ EOC sẽ được set, và kết quả được lưu vào thanh ghi. Trước khi bắt đầu quá trình chuyển đổi tiếp theo thì cờ EOC clear. Để hiểu quá trình số hóa trong STM32 diễn ra như thế nào ta theo dõi ví dụ sau. Giả sử ta cần đo điện áp tối thiểu là 0V và tối đa là 3.3V, trong STM32 sẽ chia 0 → 3.3V thành 4096 khoảng giá trị (từ 0 → 4095, do 212 = 4096), giá trị đo được từ chân IO tương ứng với 0V sẽ là 0, tương ứng với 1.65V là 2047 và tương ứng 3.3V sẽ là 4095.
* Các mode hoạt động của ADC:

+ Single conversion mode: Trong chế độ này, ADC sẽ chỉ thực hiện 1 chuyển đổi cho tới khi người dùng cho phép chuyển đổi tiếp.

+ Continuous Conversion Mode: ở chế độ này, ADC sẽ ngay lập tức thực hiển 1 chuyển đổi khác khi chuyển đổi trước vừa kết thúc.

+ Scan Mode: Chế độ này được sử dụng để quét 1 nhóm các kênh. 1 chuyển đổi duy nhất được thực hiện cho mỗi kênh. Sau khi 1 kênh chuyển đổi xong, kênh tiếp theo sẽ tự động chuyển đổi.

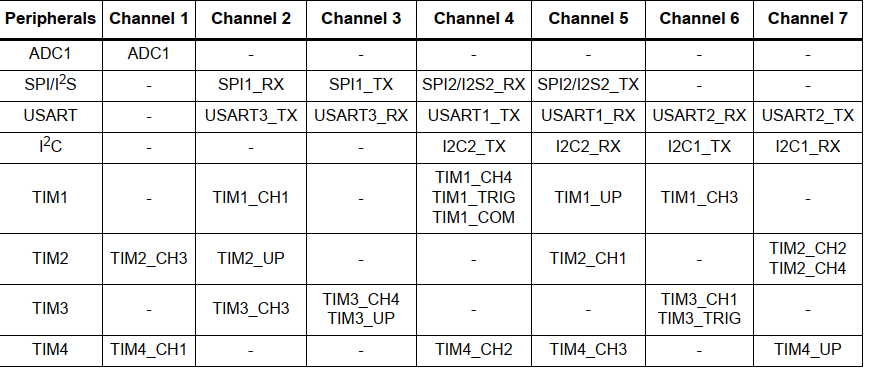
+ Discontinuous Mode: Chế độ này được sử dụng để chuyển đổi n lần (n <= 8). Giá trị của n được xác định tại bit DISCNUM[2:0] trong thanh ghi ADC\_CR1.

* 1. *DMA*

DMA – Direct memory access(DMA) được sử dụng với mục đích truyền data với tốc độ cao từ thiết bị ngoại vi đến bộ nhớ cũng như từ bộ nhớ đến bộ nhớ. Data được truyền với tốc độ cao hơn khi sử dụng DMA do không cần nhiều lệnh xử lý từ CPU. Điều đó làm cho tài nguyên CPU được rảnh rỗi cho các hoạt động khác.

Ở STM32F103C8 chỉ có 1 bộ DMA với 7 kênh hỗ trợ cho các chức năng như: ADC1, SPI, USART, I2C, TIM1 -> TIM4.

* DMA trên stm32f103c8t6



Hình 2.4 Bảng chi tiếp ngoại vi DMA trên stm32f1

* DMA có 2 chế độ hoạt động là normal và circular:

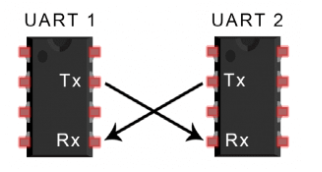
+ Normal mode: Với chế độ này, DMA truyền dữ liệu cho tới khi truyền đủ 1 lượng dữ liệu giới hạn đã khai báo DMA sẽ dừng hoạt động. Muốn nó tiếp tục hoạt động thì phải khởi động lại

+ Circular mode: Với chế độ này, Khi DMA truyền đủ 1 lượng dữ liệu giới hạn đã khai báo thì nó sẽ truyền tiếp về vị trí ban đầu (Cơ chế như Ring buffer)

* Đối với dự án lần này, nhóm chúng em sử dụng ADC đọc song kênh kết hợp cùng DMA để tối đa hiệu suất của CPU và tiết kiệm năng lương. Thiết bị cần đọc là quang điện trở ( điều khiển độ sáng đèn ) và biến trở (đóng vai trò là điểu khiển , quyết định tốc độ quạt ).

### UART

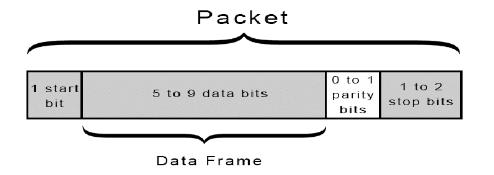
UART hay bộ thu-phát không đồng bộ đa năng là một trong những hình thức giao tiếp kỹ thuật số giữa thiết bị với thiết bị đơn giản và lâu đời nhất. Bạn có thể tìm thấy các thiết bị UART trong một phần của mạch tích hợp (IC) hoặc dưới dạng các thành phần riêng lẻ. Các UART giao tiếp giữa hai nút riêng biệt bằng cách sử dụng một cặp dẫn và một nối đất chung.



Hình 2.5 Giao tiếp UART thông qua 2 chân TX , RX

* UART truyền dữ liệu nối tiếp, theo một trong ba chế độ:
* Full duplex: Giao tiếp đồng thời đến và đi từ mỗi master và slave
* Half duplex: Dữ liệu đi theo một hướng tại một thời điểm
* Simplex: Chỉ giao tiếp một chiều

Dữ liệu truyền qua UART được tổ chức thành các gói. Mỗi gói chứa 1 bit bắt đầu, 5 đến 9 bit dữ liệu (tùy thuộc vào UART), một bit chẵn lẻ tùy chọn và 1 hoặc 2 bit dừng.



Hình 2.6 Khung truyền UART

* *Bit bắt đầu*: Đường truyền dữ liệu UART thường được giữ ở mức điện áp cao khi không truyền dữ liệu. Để bắt đầu truyền dữ liệu, UART truyền sẽ kéo đường truyền từ mức cao xuống mức thấp trong một chu kỳ clock. Khi UART nhận phát hiện sự chuyển đổi điện áp cao xuống thấp, nó bắt đầu đọc các bit trong khung dữ liệu ở tần số của tốc độ truyền.
* *Khung dữ liệu*: Khung dữ liệu chứa dữ liệu thực tế được chuyển. Nó có thể dài từ 5 bit đến 8 bit nếu sử dụng bit chẵn lẻ. Nếu không sử dụng bit chẵn lẻ, khung dữ liệu có thể dài 9 bit. Trong hầu hết các trường hợp, dữ liệu được gửi với bit ít quan trọng nhất trước tiên.
* *Bit chẵn lẻ* : Bit chẵn lẻ là một cách để UART nhận cho biết liệu có bất kỳ dữ liệu nào đã thay đổi trong quá trình truyền hay không. Bit có thể bị thay đổi bởi bức xạ điện từ, tốc độ truyền không khớp hoặc truyền dữ liệu khoảng cách xa. Sau khi UART nhận đọc khung dữ liệu, nó sẽ đếm số bit có giá trị là 1 và kiểm tra xem tổng số là số chẵn hay lẻ. Nếu bit chẵn lẻ là 0 (tính chẵn), thì tổng các bit 1 trong khung dữ liệu phải là một số chẵn. Nếu bit chẵn lẻ là 1 (tính lẻ), các bit 1 trong khung dữ liệu sẽ tổng thành một số lẻ. Khi bit chẵn lẻ khớp với dữ liệu, UART sẽ biết rằng quá trình truyền không có lỗi. Nhưng nếu bit chẵn lẻ là 0 và tổng là số lẻ; hoặc bit chẵn lẻ là 1 và tổng số là chẵn, UART sẽ biết rằng các bit trong khung dữ liệu đã thay đổi.
* *Bit dừng*: Để báo hiệu sự kết thúc của gói dữ liệu, UART gửi sẽ điều khiển đường truyền dữ liệu từ điện áp thấp đến điện áp cao trong ít nhất khoảng 2 bit.
* Có thể tóm tắt lại như sau. Quá trình truyền dữ liệu diễn ra dưới dạng các gói dữ liệu, bắt đầu bằng một bit bắt đầu, đường mức cao được kéo xuống đất. Sau bit bắt đầu, năm đến chín bit dữ liệu truyền trong khung dữ liệu của gói, theo sau là bit chẵn lẻ tùy chọn để xác minh việc truyền dữ liệu thích hợp. Cuối cùng, một hoặc nhiều bit dừng được truyền ở nơi đường đặt ở mức cao. Như vậy là kết thúc một gói.
* Đối với dự án lần này, chúng em sử dụng UART để truyền thông tin nhiệt độ, độ ẩm đến máy tính thông qua USB to TTL CH340 , baudrate là 9600



Hình 2.7 Modul CH340 USB to TTL

* Điện áp hoạt động 5V được cấp trực tiếp từ cổng USB
* Dùng chip CH340G để chuyển đổi dữ liệu qua UART
* Chuẩn USB sử dụng: 1.1 / 2.0 / 3.0 truyền thông
* Hỗ trợ trên máy tính với hệ thống XP, WIN 7, WIN 8.1, WIN 10, Apple
* Đầu ra ở dạng giao tiếp UART ở 2 chân RX và TX
* Trên module có 3 led báo gồm led báo nguồn, led TX và led RX
* Kích thước: 16×33 mm
* Module được gom chân: GND, TX, RX, VCC, 3.3V
* Hỗ trợ thêm điện áp 3,3V và 5V
* Giao tiếp hồng ngoại SIR IRDA cũng được hỗ trợ

### PWM

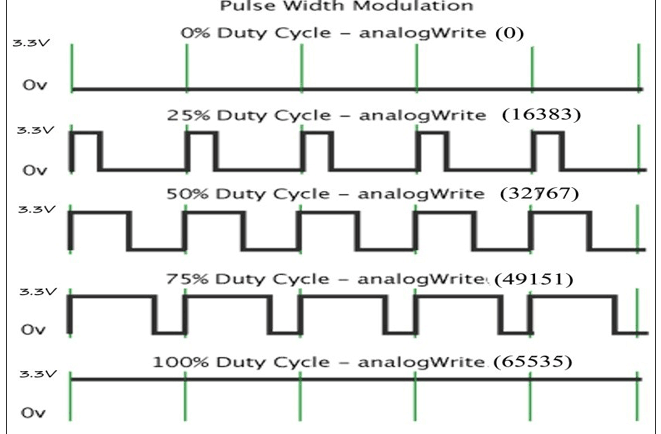
PWM viết tắt của Pulse Width Modulation, có nghĩa là phương pháp điều chỉnh điện áp tải, hay hiểu đơn giản hơn đây là phương pháp điều chỉnh, thay đổi điện áp tải ra bằng việc thay đổi độ rộng của chuỗi xung vuông, từ đó có sự thay đổi điện áp.

* Nguyên lý hoạt động
* PWM hoạt động theo nguyên tắc đóng ngắt có chu kỳ của nguồn của tải. Khi van G mở, toàn bộ điện áp được dùng cho tải. Khi van đóng, tải bị cắt nguồn điện áp.
* Vì vậy, trong suốt chu kỳ đóng mở van G này, tải sẽ có lúc nhận được toàn bộ nguồn điện áp, có lúc nhận được một phần và cũng có lúc hoàn toàn không nhận được gì.
* Công thức tính giá trị trung bình của điện áp ra tải

Ud = Umax.( t1/T) (V) hay Ud = Umax.D

* Trong STM32F103c8T6

STM32F103C8 có 15 chân PWM và 10 chân ADC. Có 7 timer và mỗi đầu ra PWM được cung cấp bởi một kênh được kết nối với 4 timer. Nó có độ phân giải PWM 16 bit (2^16 ), nghĩa là bộ đếm và biến có thể lớn tới 65535. Với cường độ xung 72 MHz, đầu ra PWM có thể có thời gian tối đa khoảng một mili giây.



Hình 2.8 Minh họa giá trị điện áp tải và thời gian đóng/mở van

Vì vậy, giá trị 65535 mang lại cho LED và Quạt DC cường độ tối đa (Chu kỳ hoạt động 100%)

Giá trị tương tự 32767 mang lại cho LED VÀ Quạt DC cường độ một nửa (Chu kỳ hoạt động 50%)

Và giá trị của 13107 mang lại (20%) độ sáng của LED VÀ (20%) TỐC ĐỘ quạt quay (Chu kỳ hoạt động 20%)

* Trong dự án của chúng em, PWM được dùng để điều khiển độ sáng của led và tốc độ của quạt : PSC = 7, ARR = 999 -> duty giao động từ 0 -> 1000

Đèn và quạt chạy tối đa khi duty bằng 1000 , dừng khi duty bằng 0.

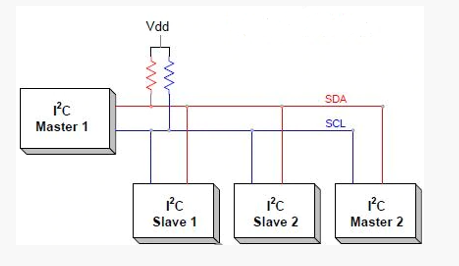
### I2C

I2C là 1 giao thức giao tiếp nối tiếp đồng bộ được phát triển bởi Philips Semiconductors, sử dụng để truyền nhận dữ liệu giữa các IC với nhau chỉ sử dụng hai đường truyền tín hiệu.

Các bit dữ liệu sẽ được truyền từng bit một theo các khoảng thời gian đều đặn được thiết lập bởi 1 tín hiệu đồng hồ.

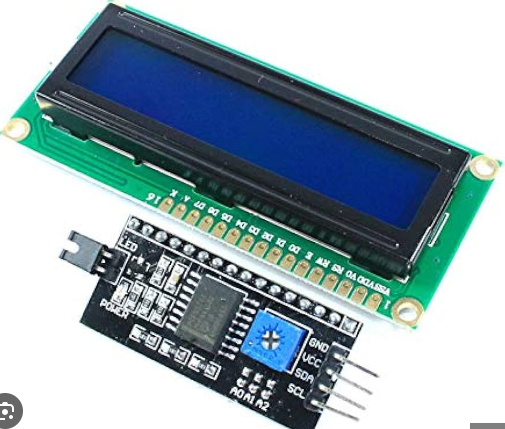
Bus I2C thường được sử dụng để giao tiếp ngoại vi cho rất nhiều loại IC khác nhau như các loại vi điều khiển, cảm biến, EEPROM, … .

* Nguyên lý hoạt động
* SCL - Serial Clock Line : Tạo xung nhịp đồng hồ do Master phát đi
* SDA - Serial Data Line : Đường truyền nhận dữ liệu.



Hình 2.9 Mô hình I2C Master - Slave

* Giao tiếp I2C bao gồm quá trình truyền nhận dữ liệu giữa các thiết bị chủ tớ, hay Master - Slave.
* Thiết bị Master là 1 vi điều khiển, nó có nhiệm vụ điều khiển đường tín hiệu SCL và gửi nhận dữ liệu hay lệnh thông qua đường SDA đến các thiết bị khác.
* Các thiết bị nhận các dữ liệu lệnh và tín hiệu từ thiết bị Master được gọi là các thiết bị Slave. Các thiết bị Slave thường là các IC, hoặc thậm chí là vi điều khiển.
* Master và Slave được kết nối với nhau như hình trên. Hai đường bus SCL và SDA đều hoạt động ở chế độ Open Drain, nghĩa là bất cứ thiết bị nào kết nối với mạng I2C này cũng chỉ có thể kéo 2 đường bus này xuống mức thấp (LOW), nhưng lại không thể kéo được lên mức cao. Vì để tránh trường hợp bus vừa bị 1 thiết bị kéo lên mức cao vừa bị 1 thiết bị khác kéo xuống mức thấp gây hiện tượng ngắn mạch. Do đó cần có 1 điện trờ ( từ 1 – 4,7 kΩ) để giữ mặc định ở mức cao
* Khung truyền I2C
* *Khối địa chỉ*: Thông thường quá trình truyền nhận sẽ diễn ra với rất nhiều thiết bị, IC với nhau. Do đó để phân biệt các thiết bị này, chúng sẽ được gắn 1 địa chỉ vật lý 7 bit cố định.
* *Bit Read/Write*: Bit này dùng để xác định quá trình là truyền hay nhận dữ liệu từ thiết bị Master. Nếu Master gửi dữ liệu đi thì ứng với bit này bằng ‘0’, và ngược lại, nhận dữ liệu khi bit này bằng ‘1’.
* *Bit ACK/NACK*: Viết tắt của Acknowledged / Not Acknowledged. Dùng để so sánh bit địa chỉ vật lý của thiết bị so với địa chỉ được gửi tới. Nếu trùng thì Slave sẽ được đặt bằng ‘0’ và ngược lại, nếu không thì mặc định bằng ‘1’.
* *Khối bit dữ liệu*: Gồm 8 bit và được thiết lập bởi thiết bị gửi truyền đến thiết bị nhân. Sau khi các bit này được gửi đi, lập tức 1 bit ACK/NACK được gửi ngay theo sau để xác nhận rằng thiết bị nhận đã nhận được dữ liệu thành công hay chưa. Nếu nhận thành công thì bit ACK/NACK được set bằng ‘0’ và ngược lại.
* Quá trình truyền/nhận:
* Bắt đầu: Thiết bị Master sẽ gửi đi 1 xung Start bằng cách kéo lần lượt các đường SDA, SCL từ mức 1 xuống 0.
* Tiếp theo đó, Master gửi đi 7 bit địa chỉ tới Slave muốn giao tiếp cùng với bit Read/Write.
* Slave sẽ so sánh địa chỉ vật lý với địa chỉ vừa được gửi tới. Nếu trùng khớp, Slave sẽ xác nhận bằng cách kéo đường SDA xuống 0 và set bit ACK/NACK bằng ‘0’. Nếu không trùng khớp thì SDA và bit ACK/NACK đều mặc định bằng ‘1’.
* Thiết bị Master sẽ gửi hoặc nhận khung bit dữ liệu. Nếu Master gửi đến Slave thì bit Read/Write ở mức 0. Ngược lại nếu nhận thì bit này ở mức 1.
* Nếu như khung dữ liệu đã được truyền đi thành công, bit ACK/NACK được set thành mức 0 để báo hiệu cho Master tiếp tục.
* Sau khi tất cả dữ liệu đã được gửi đến Slave thành công, Master sẽ phát 1 tín hiệu Stop để báo cho các Slave biết quá trình truyền đã kết thúc bằng các chuyển lần lượt SCL, SDA từ mức 0 lên mức 1.
* Các chế độ hoạt động của I2C
* Chế độ chuẩn (standard mode) với tốc độ 100 kBit/s.
* Chế độ tốc độ thấp (low speed mode) với tốc độ 10 kBit/s.
* Trong dự án này, chúng em sử dụng I2C để giao tiếp với màn hình LCD thông qua module i2c



Hình 2.10 Màn hình LCD và modul I2C

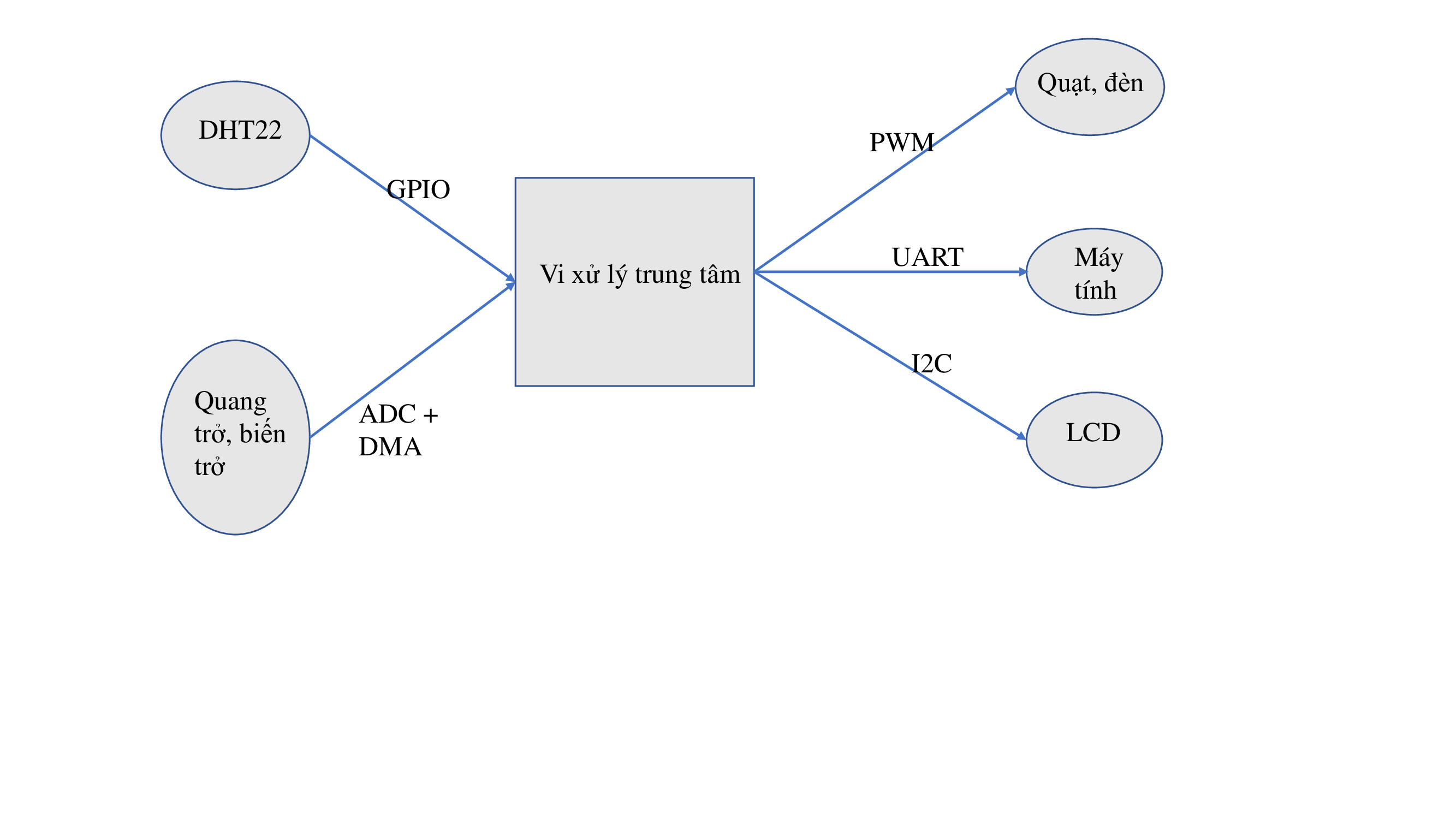
Module I2C thường có một địa chỉ I2C được thiết lập trước (ví dụ: 0x27). Đảm bảo rằng bạn biết địa chỉ này, vì nó sẽ được sử dụng trong mã lập trình để truyền dữ liệu đến màn hình LCD.

Nguyên lý việc hiển thị ký tự lên LCD :

* LCD thông thường yêu cầu một số chân để điều khiển, chẳng hạn như chân RS (Register Select), chân RW (Read/Write), chân E (Enable), và các chân dữ liệu (D0-D7).
* Khi sử dụng IC chuyển đổi I2C, số lượng chân này được mở rộng thông qua giao tiếp I2C. Module I2C được kết nối với vi điều khiển thông qua hai chân chính là SDA (Serial Data) và SCL (Serial Clock). Sử dụng giao thức I2C, vi điều khiển có thể gửi các lệnh và dữ liệu đến IC chuyển đổi.
* IC chuyển đổi nhận lệnh từ vi điều khiển thông qua I2C và thực hiện các hoạt động điều khiển tương ứng trên LCD. Chẳng hạn, khi vi điều khiển muốn gửi một ký tự đến LCD, nó sẽ gửi lệnh chọn chế độ ghi và sau đó gửi dữ liệu ký tự. IC chuyển đổi sẽ xử lý các tín hiệu điều khiển và chuyển tiếp chúng đến LCD.
* Khi vi điều khiển gửi lệnh và dữ liệu phù hợp, LCD sẽ thực hiện công việc hiển thị ký tự tương ứng trên màn hình. Điều này bao gồm việc đặt các pixel trên màn hình LCD theo đúng vị trí để tạo ra ký tự mong muốn.

Nếu SDA ở mức 0, nghĩa là đang có 1 thiết bị Master khác đang có quyền điều khiển và phải chờ đến khi truyền xong.

## Sơ đồ phần cứng



Hình 2.11 Sơ đồ phần cứng – các ngoại vi

* *Chu trình thực hiện* : dữ liệu được đọc từ quang trở, biến trở, DHT22 truyền đến vi điều khiển. Dữ liêu thô này được xử lý ở đây, sau đó nhiệt độ, độ ẩm được hiển thị trên LCD và truyền đến máy tính; còn dữ liệu đọc từ quang, biến trở sau khi được xử lý phù hợp trở thành đầu vào của khối PWM

# PHẦN MỀM

## Hệ điều hành thời gian thực

### Tổng quan

RTOS (Real-Time Operating System) một hệ điều hành thời gian thực, được dùng trong lĩnh vực thiết kế các hệ thống nhúng. RTOS được thiết kế để giải quyết các yêu cầu về phản hồi thời gian thực và hiệu suất cao cho các ứng dụng cần xử lý số lượng lớn dữ liệu và thực hiện nhiều tác vụ đồng thời.

RTOS có khả năng quản lý và phân chia thời gian CPU cho các tiến trình và luồng dữ liệu một cách linh hoạt, đồng thời cung cấp các tính năng liên quan đến bộ định thời, bộ lập lịch, bộ định thời thực, và các giải pháp kiểm soát lỗi.

* Điểm khác biệt giữa RTOS và super loop

Bảng . So sánh super Loop và RTOS

|  |  |  |
| --- | --- | --- |
| Đặc điểm | Super loop | RTOS |
| Đáp ứng thời gian | Không có cơ chế đáp ứng thời gian chính xác. Các nhiệm vụ thực hiện theo thứ tự tuần tự trong một vòng lặp lớn. | Được thiết kế để đảm bảo đáp ứng thời gian chính xác cho các nhiệm vụ thời gian thực. Các công việc được ưu tiên và lập lịch sao cho chúng hoàn thành trong khoảng thời gian cố định. |
| Quản lý nhiệm vụ | Các nhiệm vụ thường được thực hiện theo kiểu tuần tự trong một vòng lặp lớn. | Sử dụng bộ lập lịch để quản lý ưu tiên và thời gian của các nhiệm vụ khác nhau. Có thể có nhiều luồng (threads) chạy đồng thời. |
| Đối tượng sử dụng | Phù hợp cho các ứng dụng đơn giản, không yêu cầu đáp ứng thời gian cao và có yêu cầu tài nguyên thấp. | Thường được sử dụng trong các hệ thống nhúng yêu cầu đáp ứng thời gian chính xác và phức tạp |

* Các khái niệm cơ bản của hệ điều hành thời gian thực
* *Kernel ( Nhân )* : Kernel hay còn gọi là Nhân có nhiệm vụ quản lý và điều phối các Task. Mọi sự kiện (Even) như ngắt, Timer, data truyền tới… đều qua Kernel xử lý để quyết định xem nên làm gì tiếp theo. Thời gian xử lý của Kernel thường rất nhanh nên độ trễ rất thấp.
* *Scheduler ( Lập lịch )* : Đây là 1 thành phần của kernel quyết định task nào được thực thi. Có một số luật cho scheduling như:

+ Cooperative: giống với lập trình thông thường, mỗi task chỉ có thể thực thi khi task đang chạy dừng lại, nhược điểm của nó là task này có thể dùng hết tất cả tài nguyên của CPU

+ Round-robin: mỗi task được thực hiện trong thời gian định trước (time slice) và không có ưu tiên.

+ Priority base: Task được phân quyền cao nhất sẽ được thực hiện trước, nếu các task có cùng quyền như nhau thì sẽ giống với round-robin, các task có mức ưu tiên thấp hơn sẽ được thực hiện cho đến cuối time slice

+ Priority-based pre-emptive: Các task có mức ưu tiên cao nhất luôn nhường các task có mức ưu tiên thấp hơn thực thi trước.

* *Task* : Task là một đoạn chương trình thực thi một hoặc nhiều vấn đề gì đó, được Kernel quản lý. Kernel sẽ quản lý việc chuyển đổi giữa các task, nó sẽ lưu lại ngữ cảnh của task sắp bị hủy và khôi phục lại ngữ cảnh của task tiếp theo bằng cách:

+ Kiểm tra thời gian thực thi đã được định nghĩa trước (time slice được tạo

ra bởi ngắt systick)

+ Khi có các sự kiện unblocking một task có quyền cao hơn xảy ra (signal,

queue, semaphore,…)

+ Khi task gọi hàm Yield() để ép Kernel chuyển sang các task khác mà

không phải chờ cho hết time slice

+ Khi khởi động thì kernel sẽ tạo ra một task mặc định gọi là Idle Task.

Trong Rtos thì task có 4 trạng thái:

+ Ready: Task đã sẵn sàng để có thể thực thi nhưng chưa được thực thi do

có các task khác với độ ưu tiên ngang bằng hoặc hơn đang chạy (tương tự

như đối với ngắt).

+ Running: Task đang thực thi.

+ Blocked: Task đang chờ 1 sự kiện nào đó xảy ra, sự kiện này có thể là

khoảng thời gian hoặc 1 sự kiện nào đó từ task khác.

+ Suspended: Task ở trạng thái treo, về cơ bản thì trạng thái này cũng tương

tự như Blocked. Nhưng điểm khác nhau là “cách” chuyển từ trạng thái

hiện tại sang Ready State. Chỉ khi gọi hàm vTaskResume() thì task bị

treo mới được chuyển sang trạng thái Ready để có thể thực thi.

* *Queue :* Message queue là cơ chế cho phép các task có thể kết nối với nhau, nó là một FIFO ( First In First Out) buffer được định nghĩa bởi độ dài (số phần tử mà buffer có thể lưu trữ) và kích thước dữ liệu (kích thước của các thành phần trong buffer).

Một ứng dụng tiêu biểu là buffer cho Serial I/O, buffer cho lệnh được gửi tới task

Task có thể ghi vào hằng đợi (queue):

+ Task sẽ bị khóa (block) khi gửi dữ liệu tới một message queue đầy đủ

+ Task sẽ hết bị khóa (unblock) khi bộ nhớ trong message queue trống

+ Trường hợp nhiều task mà bị block thì task với mức ưu tiên cao nhất sẽ

được unblock trước

Task có thể đọc từ hằng đợi (queue) :

+ Task sẽ bị block nếu message queue trống

+ Task sẽ được unblock nếu có dữ liệu trong message queue.

+ Tương tự ghi thì task được unblock dựa trên mức độ ưu tiên

* *Semaphore* : một cơ chế đồng bộ hóa được sử dụng trong hệ thống đa nhiệm để quản lý quyền truy cập vào tài nguyên chia sẻ giữa các luồng (threads) hoặc các tiến trình. Trong ngữ cảnh của RTOS, semaphore thường được sử dụng để đảm bảo sự đồng bộ hóa giữa các nhiệm vụ (tasks) và đảm bảo an toàn khi chúng truy cập các biến hoặc tài nguyên chia sẻ.

Semaphore có hai loại cơ bản: Binary Semaphore và Counting Semaphore.

+ Binary Semaphore:

Chỉ có hai trạng thái: 0 hoặc 1 (hoặc có thể được biểu diễn là "đã có" và "chưa có").

Thường được sử dụng để kiểm soát quyền truy cập vào một tài nguyên duy nhất, chẳng hạn như một vùng nhớ hoặc thiết bị.

+ Counting Semaphore:

Có thể có giá trị lớn hơn 1, và được sử dụng để kiểm soát quyền truy cập vào một tài nguyên có sẵn nhiều lần (ví dụ: một pool bộ nhớ động).

Khi một luồng hoặc nhiệm vụ muốn truy cập tài nguyên được bảo vệ bởi semaphore, nó phải yêu cầu (hoặc "take") semaphore trước khi tiếp tục. Nếu semaphore có sẵn (ví dụ, giá trị không âm), nhiệm vụ sẽ giữ nó và tiếp tục thực hiện công việc của mình. Ngược lại, nếu semaphore không có sẵn (ví dụ, giá trị là 0), nhiệm vụ sẽ chờ cho đến khi semaphore trở thành có sẵn.

### Hệ điều hành FreeRTOS

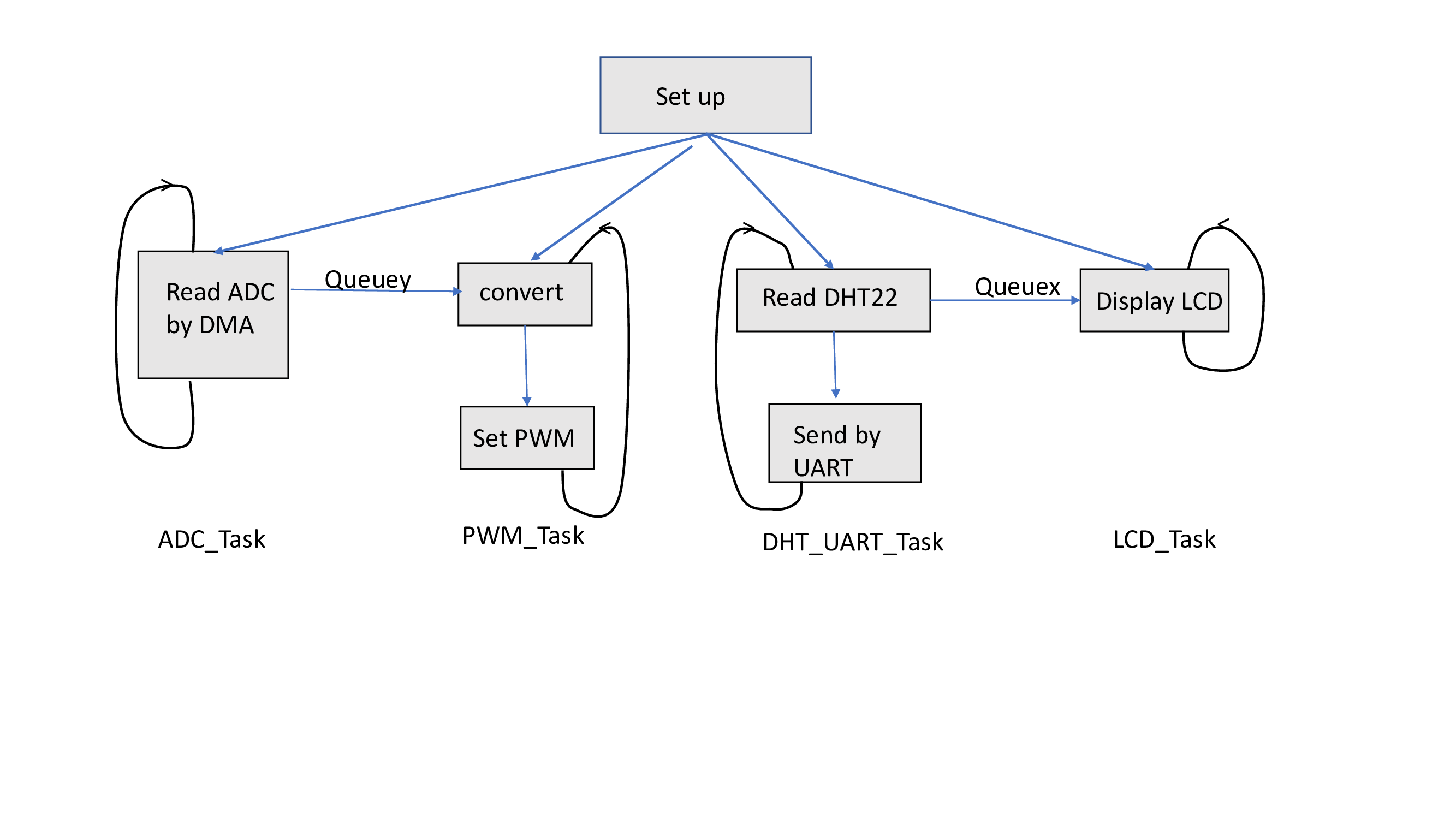
1. ***Khái niệm***

FreeRTOS là lõi của hệ điều hành thời gian thực miễn phí. Hệ điều hành này được Richard Barry công bố rộng rãi từ năm 2003, phát triển mạnh đến nay và được cộng động mạng mã nguồn mở ủng hộ. FreeRTOS có tính khả chuyển, mã nguồn mở, lõi có thể tải miễn phí và nó có thể dùng cho các ứng dụng thương mại. Nó có nhiều ưu điểm nỗi bật so với các hệ điều hành nhúngthời gian thực khác như có kích thước nhỏ gọn nên rất phù hợp với các hệ thống nhúng thời gian thực nhỏ, được viết bằng ngôn ngữ C nên có độ phù hợp cao với nền phần cứng khác nhau

1. ***Ứng dụng***

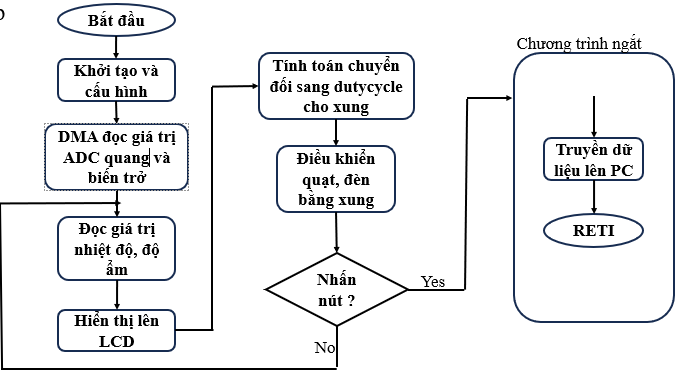
Chương trình chúng em đưa ra gồm các task sau

* *ADC\_Task* – Mức ưu tiên 4 : đọc giá trị từ 2 ADC ở chân A3 và A4,sau đó gửi giá trị này đến PWM Task qua Queuey
* *PWM\_Task* – Mức ưu tiên 3: Nhận giá trị của 2 ADC từ ADC\_Task thông qua Queuey, chuyển đổi thành giá trị duty hợp lý để điều khiển quạt, đèn ( qua chân A1 và A2)
* *DHT\_UART\_Task* – Mức ưu tiên 2 : Đọc giá trị từ cảm biến DHT22, truyền dữ liệu nhiệt/độ ẩm đến máy tính thông qua UART và StartDefaultTask thông qua Queuex
* *LCD\_Task –* mức ưu tiên 1 Nhận data từ DHT\_Task thông qua Queuex và hiển thị nó lên LCD



Hình 3.1 Lưu đồ của chương trình FreeRTOS

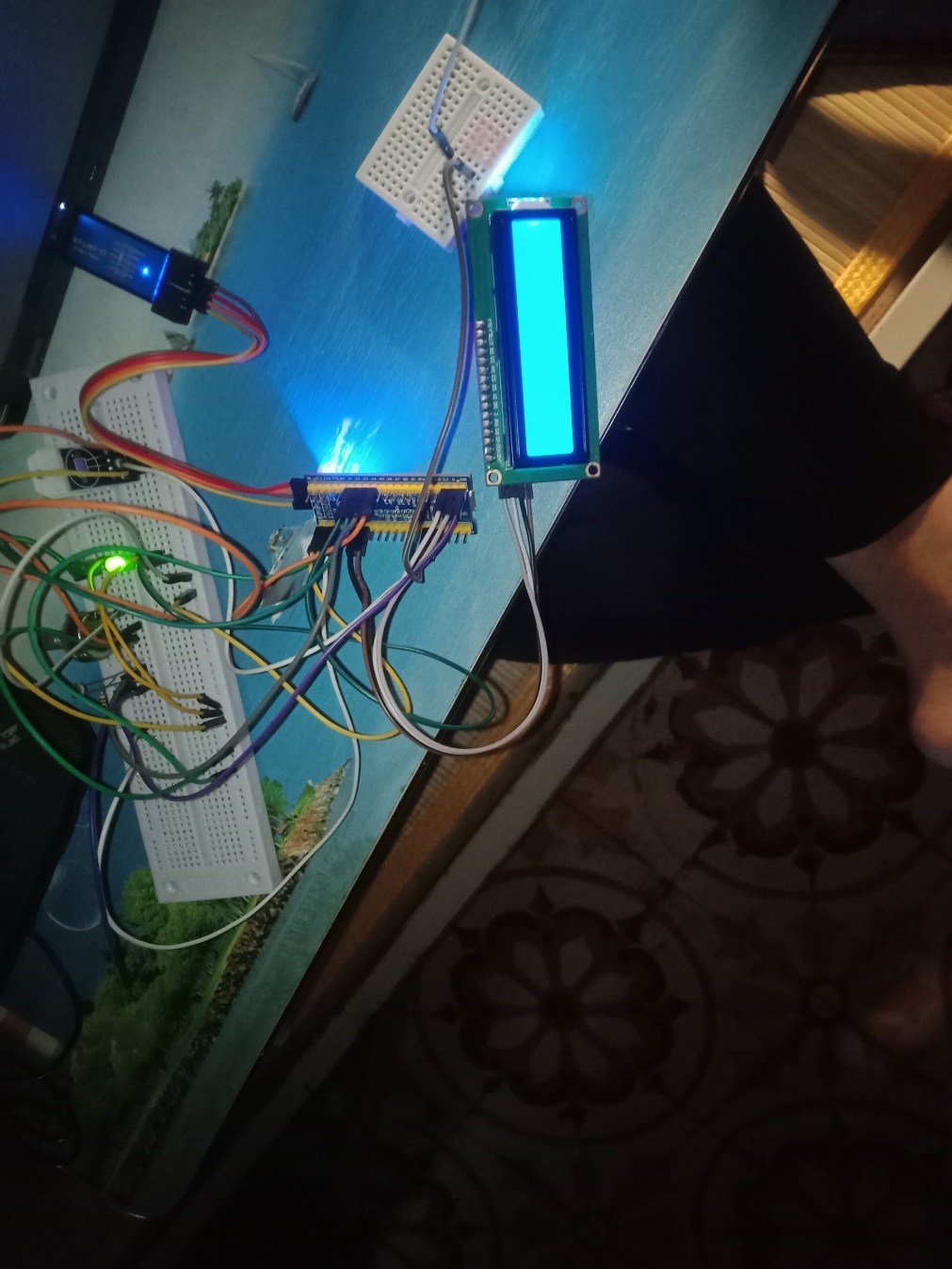
## Super Loop



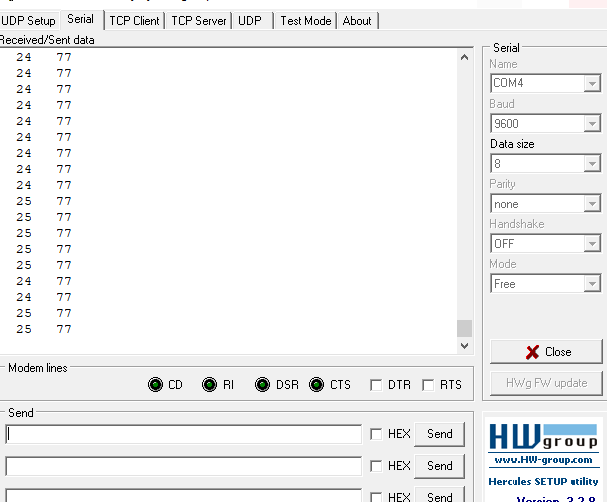
Hình 3.2 Lưu đồ thuật toán chương trình super loop

# KẾT QUẢ VÀ ĐÁNH GIÁ

## Kết quả đạt được



Hình 4.1 Thử nghiệm mạch trên testboard



Hình 4.2. Kết quả vi điều khiển gửi về máy tính thông qua UART

## Đánh giá

* Trong khuôn khổ bài tập lớn môn học Thiết kế hệ thống nhúng, chúng em đã làm được những việc sau:
* Lập trình cho STM32F103C8T6 sử dụng các ngoại vi cơ bản là GPIO, ADC, PWM, I2C và ngoại vi nâng cao DMA.
* Xây dựng phần mềm thành công trên FreeRTOS sử dụng các kiến thức cơ bản như Task, Schedule và biết cách giao tiếp giữa các Task với nhau thông qua Queue
* Tạo giao diện trực quan với người dùng thông qua LCD và máy tính
* Tuy nhiên, bên cạnh đó chúng em vẫn còn một vài thiếu sót ở độ thực tế của dự án. Chúng em mong rằng sau này sẽ có cơ hội phát triển dự án trở nên thực tế và dễ tiếp cận đối với người dùng hơn

## Phân công công việc

|  |  |  |
| --- | --- | --- |
| Thành viên | Phân công công việc | Đánh giá mức độ đóng góp |
| Nguyễn Cảnh Anh  20202291 | Lên ý tưởng, chủ trì phần cứng, viết chương trình cho I2C, DHT22, DMA, ADC. | 9.5 |
| Nguyễn Đỗ Hồng Phương  20202490 | Viết chương trình cho PWM và super Loop  + hoàn thành slide | 9 |
| Trương Nho Tuấn  20200566 | Viết chương trình cho UART và FreeRTOS + hoàn thành báo cáo | 9 |