**TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI**

**VIỆN ĐIỆN**

**BỘ MÔN KỸ THUẬT ĐO VÀ TIN HỌC CÔNG NGHIỆP**

.......................\*\*\*........................



**BÁO CÁO BÀI TẬP LỚN**

Môn học**: Thiết Kế Hệ Thống Nhúng**

**Đề tài:**

**THIẾT KẾ THIẾT BỊ GIÁM SÁT THÔNG SỐ MÔI TRƯỜNG KẾT HỢP TƯỚI TIÊU CHO VƯỜN THÔNG MINH**

**CÓ SỬ DỤNG NGUỒN CẤP TỪ PIN MẶT TRỜI**

|  |  |
| --- | --- |
| **Sinh viên thực hiện:**  Trần Quang Minh  Trịnh Văn Thái  Phạm Xuân Chính  Phí Gia Phong | **MSSV**  20181659  20181749  20181357  20181691 |

**Giảng viên hướng dẫn: Thầy Lê Công Cường**

**Hà Nội - 2022**

# MỞ ĐẦU

Thiết bị phục vụ yêu cầu cuối kỳ của môn Thiết kế Hệ thống Nhúng, kỳ 20211. Mục tiêu của thiết bị là giám sát môi trường và tự động tưới tiêu bán tự chủ năng lượng giữa 2 chế độ sử dụng pin sạc năng lượng mặt trời và nguồn ngoài. Trong báo cáo này sẽ trình bày chi tiết quá trình phát triển từ ý tưởng tới sản phẩm cuối cùng. Các bước có thể kể tới như nguyên lý, PCB, lập trình ứng dụng FreeRTOS, kiểm tra và sửa lỗi.

Báo cáo của nhóm gồm 5 chương:

Chương 1: Xác định yêu cầu thiết bị

Chương 2: Giới thiệu phần cứng

Chương 3: Thiết kế phần mềm

Chương 4: Kết quả và đánh giá

Chương 5: Link share thư mục lập trình

Trước hết nhóm em xin gửi lời cảm ơn tới thầy Lê Công Cường đã hướng dẫn chỉ dạy để chúng em hoàn thành bản báo cáo môn học này. Vì hiểu biết còn hạn hẹp và thời gian hạn chế nhóm em còn nhiều thiếu sót, nhóm mong nhận được những sự góp ý, sửa đổi từ phía thầy để báo cáo của nhóm em được hoàn thiện hơn.

Bọn em xin chân thành cảm ơn thầy!

# MỤC LỤC

[MỞ ĐẦU ii](#_Toc96626101)

[MỤC LỤC iii](#_Toc96626102)

[DANH MỤC CÁC KÝ HIỆU, CÁC CHỮ VIẾT TẮT iv](#_Toc96626103)

[DANH MỤC CÁC HÌNH VẼ, ĐỒ THỊ v](#_Toc96626104)

[CHƯƠNG 1. XÁC ĐỊNH YÊU CẦU CỦA BÀI TẬP LỚN 1](#_Toc96626105)

[1.1. Phân tích yêu cầu của bài tập lớn 1](#_Toc96626106)

[1.2. Yêu cầu về lập trình 2](#_Toc96626107)

[CHƯƠNG 2. GIỚI THIỆU VỀ PHẦN CỨNG 3](#_Toc96626108)

[2.1. Nguyên lí các module phần cứng 3](#_Toc96626109)

[2.2. Ngoại vi của STM32 cần sử dụng 13](#_Toc96626110)

[2.3. Sơ đồ nghép nối giữa STM32 và module phần cứng 22](#_Toc96626111)

[2.4. Thiết kế mạch PCB 22](#_Toc96626112)

[CHƯƠNG 3. THIẾT KẾ PHẦN MỀM 26](#_Toc96626113)

[3.1. Yêu cầu xử lý để đảm bảo thời gian thực 26](#_Toc96626114)

[3.2. Thiết kế bộ lập lịch và xây dựng chương trình 26](#_Toc96626115)

[3.3. Phân tích các kỹ thuật lập trình đa nhiệm đã sử dụng 36](#_Toc96626116)

[CHƯƠNG 4. KẾT QUẢ VÀ ĐÁNH GIÁ 41](#_Toc96626117)

[4.1. Kết quả đạt được 41](#_Toc96626118)

[4.2. Phân tích và đánh giá 45](#_Toc96626119)

[4.3. Đánh giá thành viên và công việc được giao 45](#_Toc96626120)

[CHƯƠNG 5. LINK SHARE THƯ MỤC LẬP TRÌNH 48](#_Toc96626121)

[KẾT LUẬN 49](#_Toc96626122)

# DANH MỤC CÁC KÝ HIỆU, CÁC CHỮ VIẾT TẮT

|  |  |  |
| --- | --- | --- |
| **Từ viết tắt** | **Nghĩa tiếng anh** | **Nghĩa tiếng việt** |
| MCU | Main Control Unit | Khối xử lý trung tâm |
| UART | Universal Asynchronous  Receiver-Transmitter |  |
| SPI | Serial Peripheral Interface |  |
| I2C | Inter-Integrated Circuit |  |
| PCB | Printed Circuit Board | Bảng mạch in |
| GPIO | General Purpose In/Out |  |
| NVIC | Nested Vector Interrupt Control |  |
| RCC | Reset Clock Control |  |
| LCD | Liquid Crystal Display | Màn hình tinh thể lỏng |
| RTOS | Real-Time Operating System |  |
|  |  |  |
|  |  |  |
|  |  |  |

# DANH MỤC CÁC HÌNH VẼ, ĐỒ THỊ

[Hình 2‑1 Vi điều khiển STM32F103C8Y6 3](#_Toc96626045)

[Hình 2‑2 Kit phát triển STM32F103C8T6 BluePill 4](#_Toc96626046)

[Hình 2‑3 Module SIM800L 4](#_Toc96626047)

[Hình 2‑4 Sơ đồ nguyên lý của module SIM800L trên phần mềm Altium 5](#_Toc96626048)

[Hình 2‑5 Hình ảnh màn hình TFT 2.8” có cảm ứng 6](#_Toc96626049)

[Hình 2‑6 Sơ đồ nguyên lý của các Jump để kết nối STM32 và màn hình 7](#_Toc96626050)

[Hình 2‑7 Hình ảnh cảm biến AHT10 và kích thước 7](#_Toc96626051)

[Hình 2‑8 Hình ảnh cảm biển đo độ ẩm đất bằng điện dung 8](#_Toc96626052)

[Hình 2‑9 Sơ đồ nguyên lý các Jump đầu ra của các cảm biến 9](#_Toc96626053)

[Hình 2‑10 Hình ảnh động cơ bơm nước 9](#_Toc96626054)

[Hình 2‑11 Sơ đồ nguyên lý của khối đóng mở động cơ 10](#_Toc96626055)

[Hình 2‑12 Sơ đồ nguyên lý khối năng lượng mặt trời 11](#_Toc96626056)

[Hình 2‑13 Sơ đồ nguyên lý khối Buck 5V (có kiểm soát) 12](#_Toc96626057)

[Hình 2‑14 Sơ đồ nguyên lý khối Buck 3.8V (có kiểm soát) 12](#_Toc96626058)

[Hình 2‑15 Sơ đồ nguyên lý khối nguồn tuyến tính 3.3V 12](#_Toc96626059)

[Hình 2‑16 Sơ đồ nguyên lý nguồn pin Lipo (có kiểm soát) 12](#_Toc96626060)

[Hình 2‑17 Sơ đồ đấu dây của giao thức UART 13](#_Toc96626061)

[Hình 2‑18 Định dạng gói tin của giao thức UART 13](#_Toc96626062)

[Hình 2‑19 Sơ đồ đấu dây của giao thức I2C 15](#_Toc96626063)

[Hình 2‑20 Định dạng gói tin giao thức I2C 16](#_Toc96626064)

[Hình 2‑21 Sơ đồ đấu dây giao thức SPI 17](#_Toc96626065)

[Hình 2‑22 Mô tả truyền nhận dữ liệu theo xung đồng hồ 17](#_Toc96626066)

[Hình 2‑23 Mô tả biến đổi ADC 18](#_Toc96626067)

[Hình 2‑24 Cấu trúc hệ thống đồng hồ trong của STM32 19](#_Toc96626068)

[Hình 2‑25 Cấu trúc bên trong các chân GPIO của STM32 20](#_Toc96626069)

[Hình 2‑26 Cấu trúc tín hiệu của các chân ngắt ngoài 21](#_Toc96626070)

[Hình 2‑27 Sơ đồ khối giao tiếp giữa STM32 và các module phần cứng và các ngoại vi tương ứng 22](#_Toc96626071)

[Hình 2‑28 Hình ảnh mạch 3D mặt trên 23](#_Toc96626072)

[Hình 2‑29 Hình ảnh mạch 3D mặt dưới 23](#_Toc96626073)

[Hình 2‑30 Hình ảnh tổng quát mạch thực tế 24](#_Toc96626074)

[Hình 2‑31 Hình ảnh mạch chính (góc chéo) 24](#_Toc96626075)

[Hình 2‑32 Hình ảnh mạch chính (trực diện) 25](#_Toc96626076)

[Hình 3‑1 Lưu đồ thuật toán của thiết bị 28](#_Toc96626077)

[Hình 3‑2 Cấu hình thạch anh ngoại trong STM32CubeMX 29](#_Toc96626078)

[Hình 3‑3 Cấu hình cây đồng hồ xung nhịp trong STM32CubeMX 29](#_Toc96626079)

[Hình 3‑4 Cấu hình chế độ debug trong STM32CubeMX 29](#_Toc96626080)

[Hình 3‑5 Cấu hình các tác vụ trên STM32CubeMX 30](#_Toc96626081)

[Hình 3‑6 Cấu hình bộ định thời và khóa Semaphore trên STM32CubeMX 30](#_Toc96626082)

[Hình 3‑7 Cấu hình ngoại vi I2C trên STM32CubeMX 31](#_Toc96626083)

[Hình 3‑8 Cấu hình ngoại vi SPI trên STM32CubeMX 31](#_Toc96626084)

[Hình 3‑9 Cấu hình ngoại vi USART trên STM32CubeMX 32](#_Toc96626085)

[Hình 3‑10 Cấu hình bộ ADC trên STM32CubeMX 33](#_Toc96626086)

[Hình 3‑11 Sơ đồ các chân được sử dụng và chức năng của chúng 34](#_Toc96626087)

[Hình 3‑12 Cấu hình các chân GPIOs trong STM32CubeMX 34](#_Toc96626088)

[Hình 3‑13 Cấu hình các ngắt của các ngoại vi trong STM32CubeMX 35](#_Toc96626089)

[Hình 3‑14 Code được sinh thành thư mục được mở bằng VSCode 35](#_Toc96626090)

[Hình 3‑15 Mô hình chạy các tác vụ 37](#_Toc96626091)

[Hình 3‑16 Mô tả các trạng thái của tác vụ và cách chuyển đổi giữa các trạng thái 38](#_Toc96626092)

[Hình 3‑17 Mô tả hoạt động của khóa Semaphore 39](#_Toc96626093)

[Hình 3‑18 Mô tả hoạt động của khóa Semaphore 02 39](#_Toc96626094)

[Hình 3‑19 Mô tả hoạt động của khóa Semaphore 01 40](#_Toc96626095)

[Hình 3‑20 Mô tả hoạt động của Software Timer 40](#_Toc96626096)

[Hình 4‑1 Màn hình thiết bị sử dụng nguồn ngoài 41](#_Toc96626097)

[Hình 4‑2 Màn hình thiết bị ở chế độ 42](#_Toc96626098)

[Hình 4‑3 Các thông số về thời gian các tác vụ 42](#_Toc96626099)

[Hình 4‑4 Các đồ thị trên giao diện Website 44](#_Toc96626100)

## XÁC ĐỊNH YÊU CẦU CỦA BÀI TẬP LỚN

### Phân tích yêu cầu của bài tập lớn

Thiết bị phải đảm bảo các yêu cầu sau đây.

#### Chức năng thu thập dữ liệu môi trường

Thiết bị thu thập dữ liệu môi trường qua các cảm biến và ngoại vi, thực hiện lưu trữ hợp lý và nhanh chóng trong thời gian quy định. Dữ liệu được thu thập từ 2 node cảm biến gồm nhiệt độ, độ ẩm. Độ ẩm đất được đo riêng tại vị trí cần tưới tiêu. Các thông số liên quan tới năng lượng được lưu trữ để truyền lên server gồm điện áp pin lipo và điện áp đầu ra mạch MPPT pin mặt trời.

#### Chức năng tự động chuyển chế độ sử dụng năng lượng

Thiết bị có thể đổi chế độ bằng tay qua nút nhấn trên màn hình khi cả 2 nguồn cấp đều có thể được sử dụng. Nếu thiết bị đang sử dụng pin mà điện áp tụt xuống dưới mức cho phép và nguồn ngoài khả dụng thì sẽ tự động chuyển sang sử dụng nguồn ngoài.

#### Chức năng kết nối với Server

Thiết bị phải kết nối với server qua mạng 3G và gửi bản tin MQTT chứa toàn bộ các thông số cần giám sát trong thời gian quy định. Có chức năng tự động kết nối và báo cáo trạng thái kết nối hiển thị qua màn hình.

#### Chức năng tưới tiêu

Thiết bị có 2 chế độ tưới tiêu là tự động dựa vào độ ẩm của đất hoặc có người điều khiển. Lựa chọn chế độ tưới trên màn hình hiển thị qua dạng nút ấn. Khi sử dụng nguồn pin Lipo thì chức năng tưới tiêu chỉ không còn khả dụng và thao tác nút ấn sẽ không còn hiệu quả.

#### Chức năng hiển thị trên màn hình cảm ứng

Các thông số của cần giám sát cần được hiển thị trên màn hình cùng các trạng thái kết nối server và trạng thái nguồn của thiết bị. Thiết bị xử lý cảm ứng màn hình và chuyển các chế độ qua dạng nút ấn. Màn hình có chức năng tự động tắt và vào chế độ chờ khi không sử dụng để tiết kiệm năng lượng.

### Yêu cầu về lập trình

Lập trình sử dụng FreeRTOS, có chia các tác vụ và sử dụng các chức năng hợp lý. Phần lập trình phải rõ ràng và mạch lạc, bảo đảm tính ổn định của thiết bị.

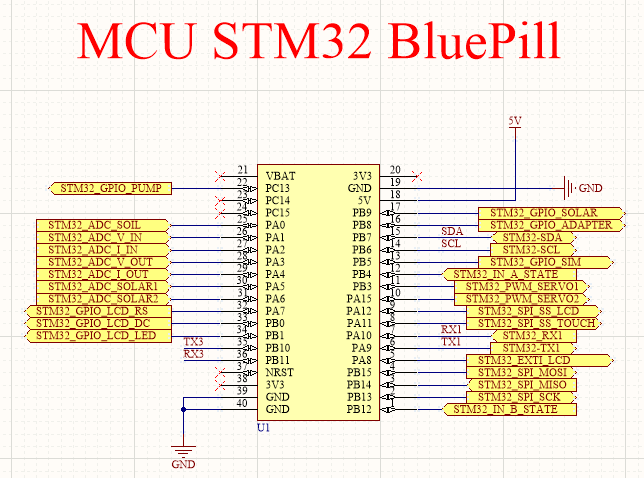
## GIỚI THIỆU VỀ PHẦN CỨNG

### Nguyên lí các module phần cứng

#### Khối vi điều khiển MCU

STM32 là một trong những dòng chip phổ biến của ST với nhiều họ thông dụng như F0,F1,F2,F3,F4….. Stm32f103 thuộc họ F1 với lõi là ARM COTEX M3. STM32F103 là vi điều khiển 32 bit, tốc độ tối đa là 72Mhz. Giá thành cũng khá rẻ 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 bits Cortex M3 * Điện áp hoạt động: 2 3,3V * Tần số lớn nhất: 72MHz * Bộ nhớ 64 128 Kb Flash, 20 Kb SRAM * ADC 2x12 bit, tần số lấy mẫu 16MHz * Timer: 7 bộ, 16 bits * Kết nối: 2xI2C, 3xUSART, 2xSPI, CAN, USB 2.0 full speed * Kiểu chân: LQFP48 * Watchdog: WWDG * Chế độ tiết kiệm năng lượng: Sleep, Stop, Standby * Hỗ trợ nạp xóa lên tới 100000 lần | Hình ‑ Vi điều khiển STM32F103C8Y6 |

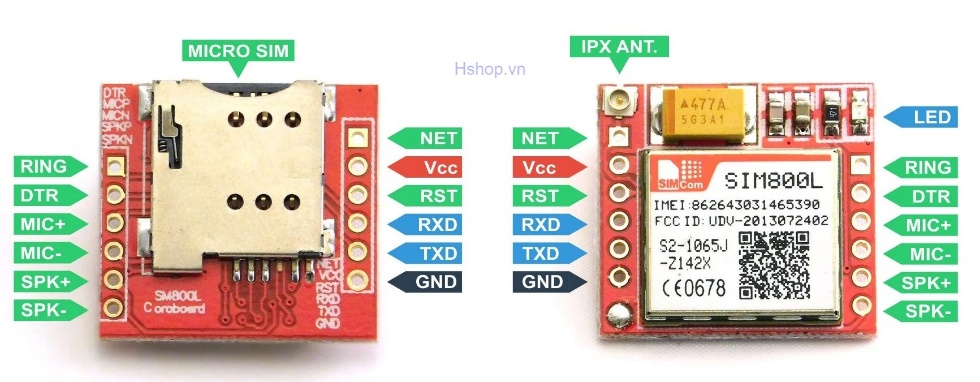


Hình ‑ Kit phát triển STM32F103C8T6 BluePill

Kit phát triển STM32F103C8T6 thuộc loại kit phát triển được thiết kế với đơn giản, kit ra đầy đủ chân của vi điều khiển, có cổng giao tiếp USB và cổng nạp SWD, sử dụng dòng vi điều khiển 32 Bit của dòng ST.

Trong project này bọn em sử dụng Kit phát triển này để làm nhằm tiết kiệm thời gian thiết kế mạch PCB.

#### Khối Module SIM800L



Hình ‑ Module SIM800L

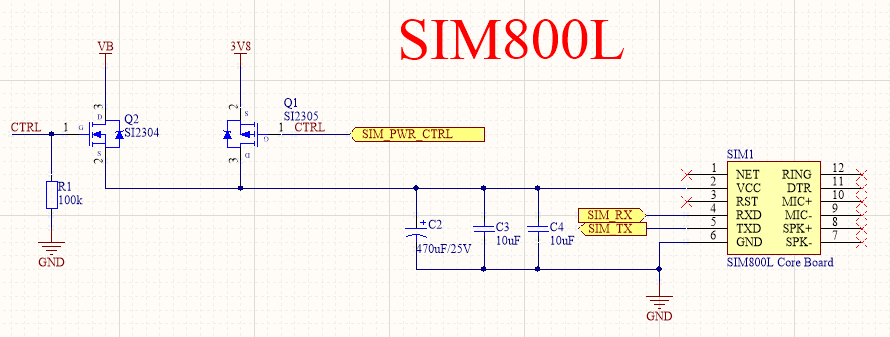
Là module có khả năng nhắn tin SMS, nghe, gọi, GPRS, … như một điện thoại nhưng có kích thước nhỏ nhất trong các loại module SIM (25 mm x 22 mm). Điều khiển module sử dụng bộ tập lệnh AT.

**Thông số kĩ thuật:**

* Nguồn cấp: 3.7 – 4.2VDC,
* Khe cắm SIM: MICROSIM
* Dòng khi ở chế độ chờ: 10mA
* Dòng khi hoạt động: 100mA đến 1A.
* Hỗ trợ 4 băng tần phổ biến.
* Kích thước: 25mm x 22mm

**Sơ đồ chân:**

* VCC: Nguồn vào 3.7 – 4.2V.
* TXD: Chân truyền Uart TX.
* RXD: Chân nhận Uart RX.
* DTR : Chân UART DTR, thường không xài.
* SPKP, SPKN: ngõ ra âm thanh, nối với loa để phát âm thanh.
* MICP, MICN: ngõ vao âm thanh, phải gắn thêm Micro để thu âm thanh.
* Reset: Chân khởi động lại Sim800L (thường không xài).
* RING : báo có cuộc gọi đến
* GND: Chân Mass, cấp 0V.

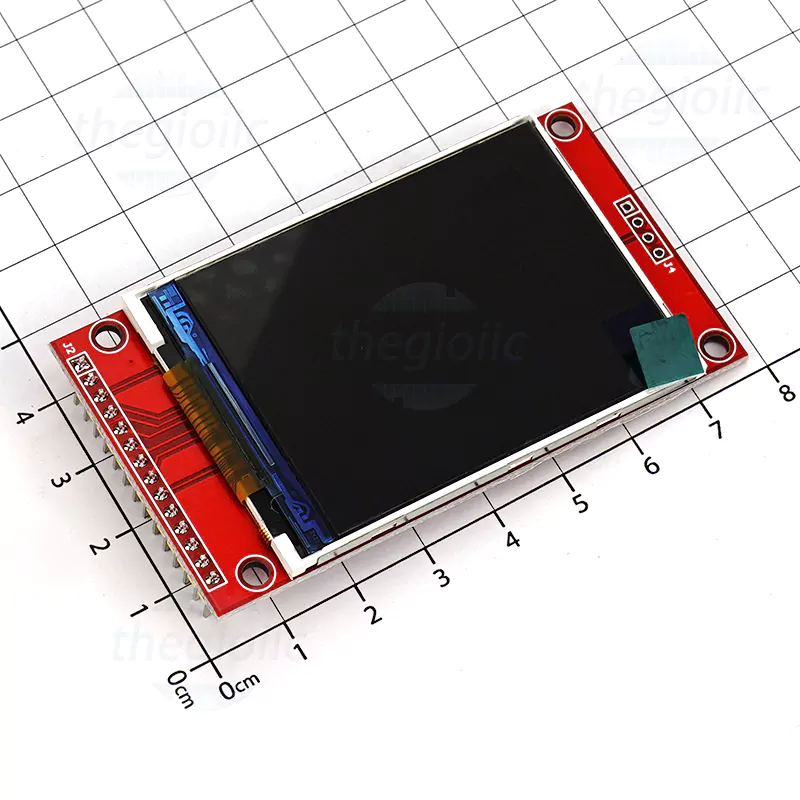


Hình ‑ Sơ đồ nguyên lý của module SIM800L trên phần mềm Altium

**Chức năng trong project:**

* Module Sim800L giao tiếp với STM32f03 thông qua giao tiếp UART
* Module SIM800L được sử dụng để gửi các thông số về môi trường như: nhiệt độ, độ ẩm không khí, độ ẩm đất, thông số về điện áp, dòng điện của pin mặt trời lên web thông qua giao thức MQTT.

#### Khối màn hình TFT



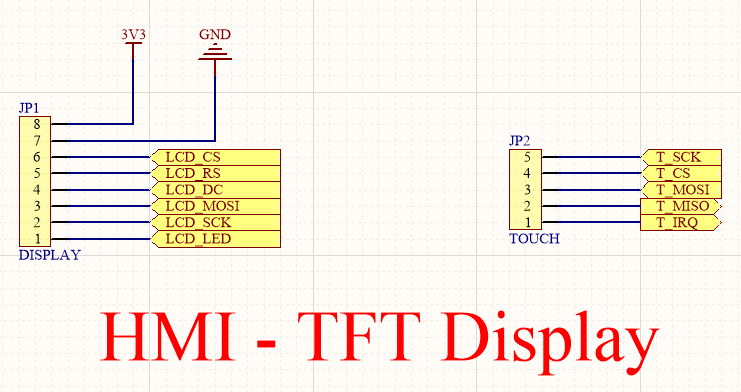
Hình ‑ Hình ảnh màn hình TFT 2.8” có cảm ứng

**Thông số kĩ thuật:**

|  |  |
| --- | --- |
| Kích thước đường chéo | 2.4inch |
| Giao tiếp | SPI |
| Độ phân giải | 240 x 320pixels |
| Cảm ứng | Không |
| Kích thước vùng nhìn | 36.72 x 48.96mm |
| Loại cảm ứng | Không có |
| Điện áp cấp | 5VDC |
| Loại đèn nền Backlight | LED |
| Chế độ hiển thị | Transmissive |
| Kích thước | 44.7 x 74.75 x 15.9mm |
| Công nghệ hiển thị | TFT |
| Số màu | 65,536 |
| Nhiệt độ hiển thị | -10 - +60°C |

**Chức năng trong project:**

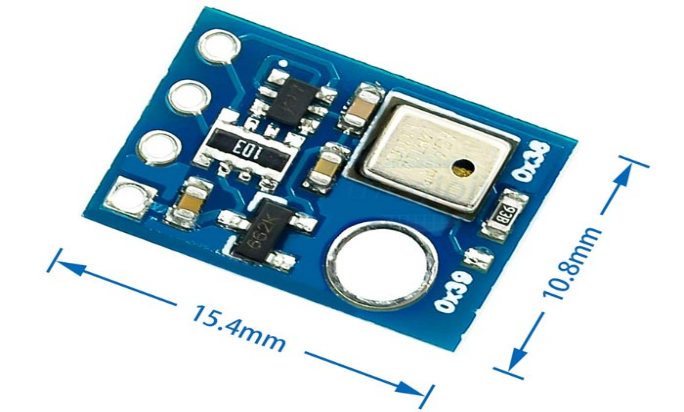
* Màn hình LCD TFT giao tiếp với STM32F103C8T6 thông qua giao thức SPI và các chân GPIO
* Màn được sử dụng để hiển thị các thông số về nhiệt độ, độ ẩm không khí, độ ẩm đất, điện áp và dòng điện pin mặt trời.
* Ngoài ra, màn hình còn được sử dụng để bật tắt máy bơm nước bằng nút cảm ứng được thiết kế trên màn.



Hình ‑ Sơ đồ nguyên lý của các Jump để kết nối STM32 và màn hình

#### Khối các cảm biến

2.1.4.1. Cảm biến AHT10 đo nhiệt độ và độ ẩm không khí



Hình ‑ Hình ảnh cảm biến AHT10 và kích thước

**Thông số kĩ thuật:**

* Điện áp cung cấp: 1.8V-6.0V (Điện áp hoạt động của cảm biến 1.8V-3.6V)
* Giao tiếp: I2C
* Dải đo nhiệt độ: -40°C đến 85°C
* Độ phân giải nhiệt độ: 0.01 °C
* Sai số đo nhiệt độ: ±0.3 °C (thông thường)
* Dải đo độ ẩm: 0% đến 100%
* Độ phân giải độ ẩm: 0.024%
* Sai số đo độ ẩm: ±2%
* Port kết nối: 4x2.54mm

**Chức năng trong project:**

* Đo nhiệt độ và độ ẩm không khí
* Giao tiếp với STM32F103C8T6 thông qua giao tiếp I2C

2.1.4.2. Cảm biến đo độ ẩm đất:

Ảnh có chứa văn bản

Mô tả được tạo tự động

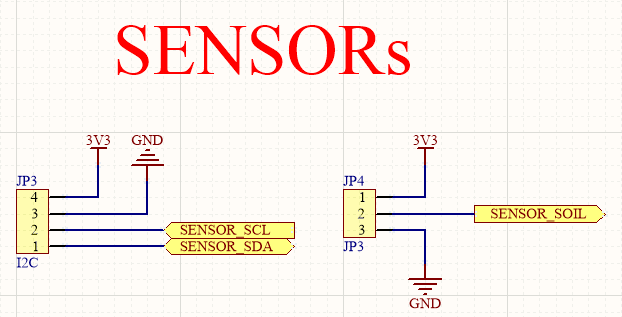
Hình ‑ Hình ảnh cảm biển đo độ ẩm đất bằng điện dung

**Thông số kỹ thuật:**

* Điện áp hoạt động: 3.3~5VDC
* Tín hiệu đầu ra: Analog, theo điện áp cấp nguồn tương ứng
* Kích thước: 3 x 1.6cm

**Chức năng trong project:**

* Đo độ ẩm đất
* Giao tiếp với STM32F103C8T6 thông qua ADC



Hình ‑ Sơ đồ nguyên lý các Jump đầu ra của các cảm biến

#### Khối điều khiển động cơ



Hình ‑ Hình ảnh động cơ bơm nước

**Thông số kỹ thuật:**

* Điện áp hoạt động: 5VDC
* Kích mở qua Relay và transistor
* Kích thước: 10x5x5cm

**Chức năng trong project:**

* Tưới tiêu
* Giao tiếp với STM32F103C8T6 thông qua GPIO OUTPUT



Hình ‑ Sơ đồ nguyên lý của khối đóng mở động cơ

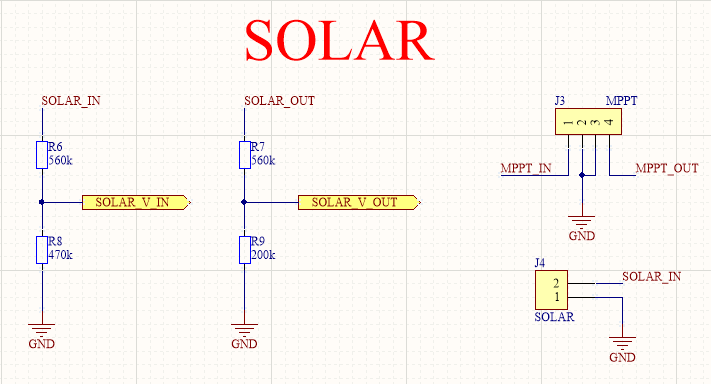
#### Khối giám sát pin và pin mặt trời

**Thông số kỹ thuật:**

* Cầu chia áp để đưa điện áp về chuẩn ADC 0-3.3V
* Module Solar MPPT sạc pin Lipo, đầu vào 5-6V
* Chân cắm cho pin và tấm năng lượng mặt trời

**Chức năng trong project:**

* Nạp cho pin Lipo bằng tấm năng lượng mặt trời
* Các điện áp của pin Lipo và tấm pin mặt trời sẽ được đưa và bộ chuyển đổi tương tự - số ADC.



Hình ‑ Sơ đồ nguyên lý khối năng lượng mặt trời

#### Khối nguồn

**Thông số kỹ thuật:**

* Sử dụng IC TPS5430 Buck Converter để hạ từ điện áp 1 chiều 9-12VDC từ nguồn ngoài xuống 5V, 3.3V và 3.8V
* Các MOSFETs và transistors có mức logic 3.3V

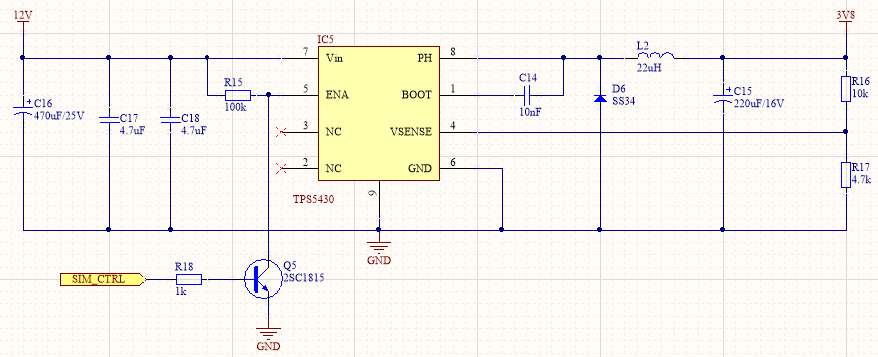
**Chức năng trong project:**

* Cung cấp nguồn 5V và 3.3V cho vi điều khiển, màn hình và các cảm biến
* Cung cấp nguồn 3.8V cho module SIM800L
* Phục vụ chuyển đổi chế độ nguồn tùy theo trạng thái hoạt động

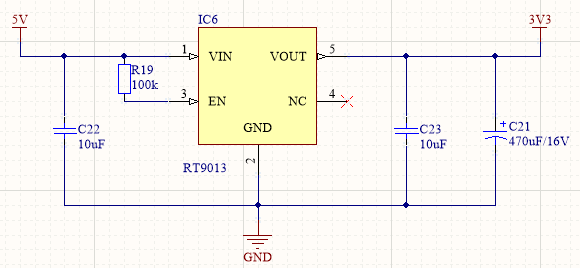
Ảnh có chứa văn bản, cửa shoji

Mô tả được tạo tự động

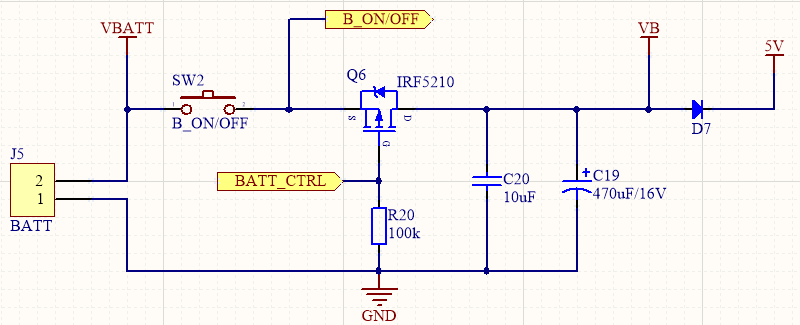
Hình ‑ Sơ đồ nguyên lý khối Buck 5V (có kiểm soát)



Hình ‑ Sơ đồ nguyên lý khối Buck 3.8V (có kiểm soát)



Hình ‑ Sơ đồ nguyên lý khối nguồn tuyến tính 3.3V



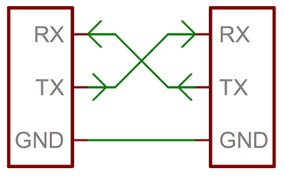
Hình ‑ Sơ đồ nguyên lý nguồn pin Lipo (có kiểm soát)

### Ngoại vi của STM32 cần sử dụng

#### Truyền thông nối tiếp UART

UART tiếng anh là Universal Asynchronous Reciver/Transmister là 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 receiver.



Hình ‑ Sơ đồ đấu dây của giao thức UART

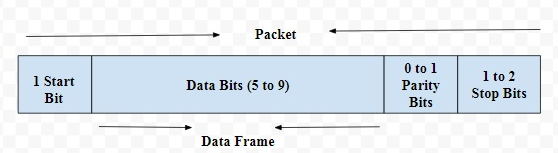
*Cách hoạt động của UART trong STM32:*

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

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 và phải được cài đặt giống nhau ở gửi và nhận.

Định dạng gói tin như sau:



Hình ‑ Định dạng gói tin của giao thức 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.

* **Paratity 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.

#### Chuẩn giao tiếp

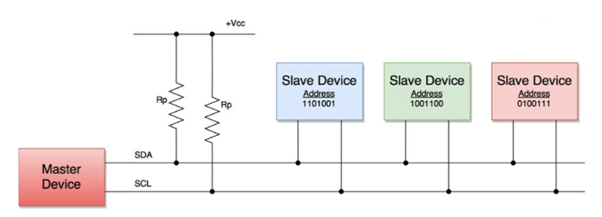
I2C viết tắt của Inter- Integrated Circuit là một phương thức giao tiếp được phát triển bởi hãng Philips Semiconductors. Dùng để truyền tín hiệu giữa vi xử lý và các IC trên các bus nối tiếp.

***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)

***Kết nối vật lý giao thức của 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 ‑ Sơ đồ đấu dây 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

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.

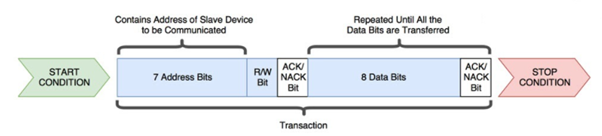
***Cách truyền dữ liệu của I2C:***

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:

1. Master thực hiện điều kiện bắt đầu I2C (Start Condition)
2. 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
3. Nhận phải hồi từ Bus, nếu có một bit ACK (Kéo SDA xuống thấp) Master sẽ gửi dữ liệu
4. 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
5. Truyền điều khiện kết thúc (Stop Condition)

Mỗi lần giao tiếp có cấu trúc như sau:



Hình ‑ Định dạng gói tin giao thức I2C

#### Chuẩn giao tiếp SPI

SPI Serial Peripheral Interface là một chuẩn truyền [thông nối tiếp đồng bộ](https://vi.wikipedia.org/wiki/Truy%E1%BB%81n_th%C3%B4ng_n%E1%BB%91i_ti%E1%BA%BFp) dùng để chuyền dữ liệu ở chế độ song công toàn phần (full duplex).

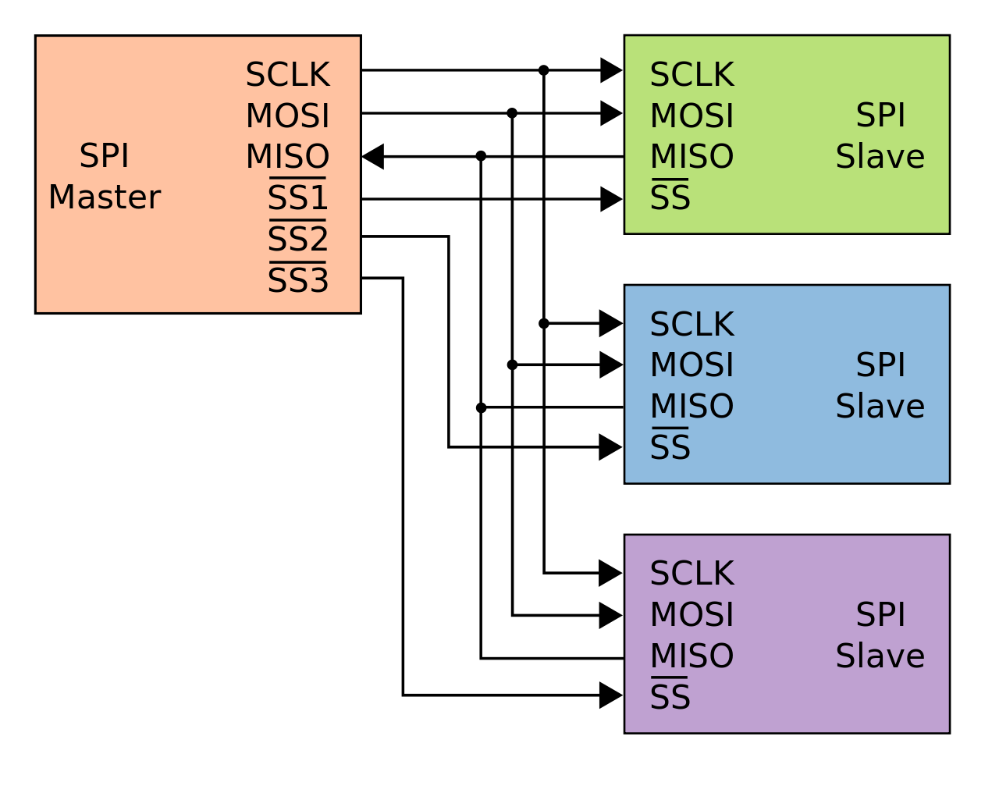
Giao thức SPI là dao thức dạng Master –Slave trong đó Master giữ quyền điều khiển xung Clock và chọn Slave nào giao tiếp với mình.

Bus SPI bao gồm 4 đường tín hiệu:

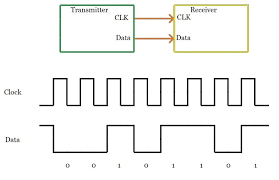
* MOSI (Master Out Slave In): Đường truyền tín hiệu từ Master đến Slave
* MISO (Master In Slave Out): Đường truyền tín hiệu từ Slave đến Master
* CLK (Serial Clock): Đường phát xung đồng hồ được điều khiển bởi Master
* CS (Chip Select): Đường chọn chip giao tiếp với Master, được kéo xuống 0 khi được chọn

Nguyên lí hoạt động như sau:

* Khi muốn truyền nhận dữ liệu tới các Slave, đầu tiên Master kéo đường CS kết nối từ Master tới Slave đó xuống 0.
* Gửi xung Clock, tương ứng với mỗi Clock sẽ gửi DATA trên chân MOSI tại thời điểm Clock ở mức cao (hoặc thấp tùy người lập trình)
* Slave cũng có thể gửi ngược lại DATA tại chân MISO tới Master
* Sự truyền nhận dữ liệu là liên tục nên SPI thường có tốc độ rất cao



Hình ‑ Sơ đồ đấu dây giao thức SPI



Hình ‑ Mô tả truyền nhận dữ liệu theo xung đồng hồ

#### Chuyển đổi tương tự - số ADC

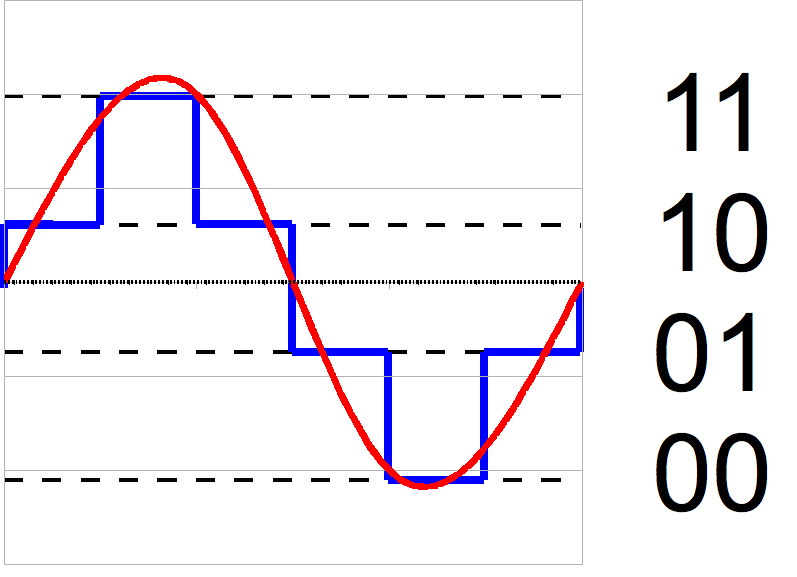
ADC Analog to Digital Convert là bộ chuyển đôi tương tự sang số. Đại lượng tương tự là Điện áp Vin được so sánh với điện áp mẫu Vref (giá trị lớn nhất), sau đó được chuyển đổi thành số lưu vào thanh ghi DATA của bộ chuyển đổi đó.

Có 2 tham số quan trọng của bộ ADC cần lưu ý:

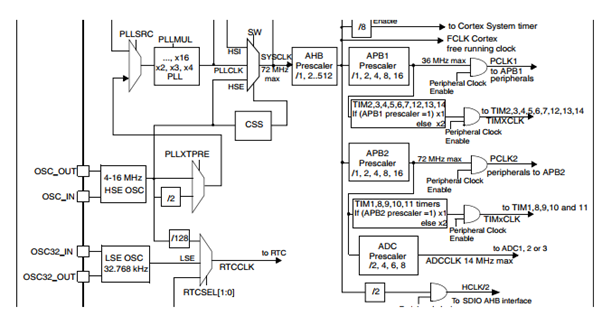
* Tốc độ lấy mẫu (sampling) được tính theo số chu kì chuyển đổi
* Độ phân giải: Tính theo Bit bộ ADC có độ phân giải 10 Bit sẽ có 2^10 = 1024 giá trị

***Các chức năng chính của ADC trong STM32***

* Độ phân giải 12Bit
* Sinh ra ngắt tại các sự kiện End of convert, End of Injected, Analog Watchdog
* Chế độ Single hoặc Continuous
* Chế độ Scan tự động quét từ Kênh 0 đến Kênh n (mỗi bộ có 10 kênh tối đa)
* Có cơ chế cân chỉnh tay
* Data Alignment (Căn chỉnh Data) căn trái hoặc căn phải
* Cài đặt thời gian chuyển đổi đến từng Kênh
* Có thể kích hoạt bằng xung bên ngoài
* Chế độ Dual mode sử dụng cùng lúc 2 hoặc nhiều bộ ADC
* Hỗ trợ DMA
* Tần số chuyển đổi ADC được lấy từ bộ ABP2 thông qua ADC prescaler và phải nhỏ hơn 14MHz



Hình ‑ Mô tả biến đổi ADC



Hình ‑ Cấu trúc hệ thống đồng hồ trong của STM32

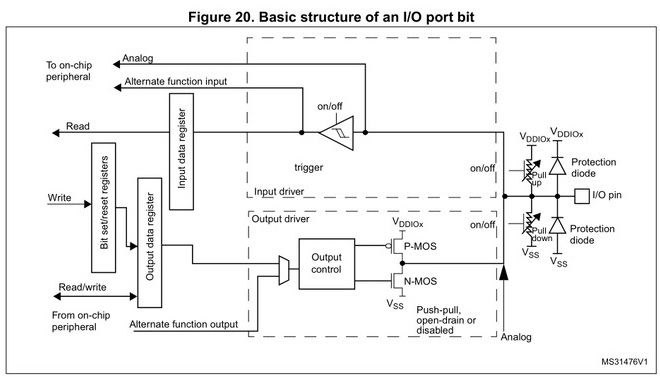
#### GPIO và ngắt ngoài

###### GPIO:

[GPIO](https://wiki.st.com/stm32mpu/wiki/GPIO_internal_peripheral) (General Purpose Input/Output) chính là các chân đầu ra hoặc đầu vào dùng chung.

***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



Hình ‑ Cấu trúc bên trong các chân GPIO của STM32

***Chức năng của STM32 GPIO bao gồm:***

* **Input:**
* 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
* **Output:**
* 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)
* **Ngắt Ngoài:**

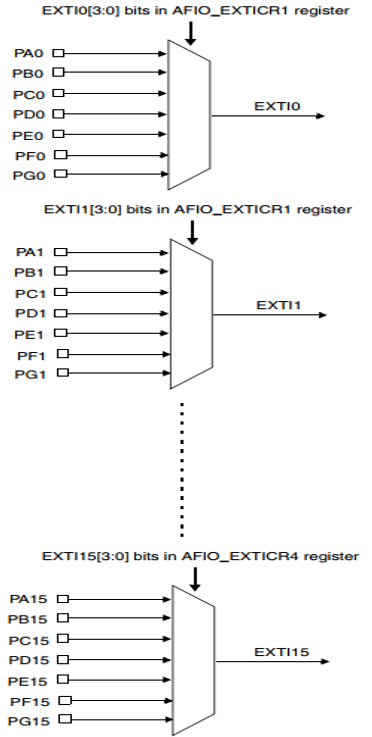
[**EXTI**](https://wiki.st.com/stm32mpu/wiki/EXTI_internal_peripheral) **(External Interupts)** tạm dịch là ngắt ngoài hay ngắt sự kiện bên ngoài. Ngắt EXTI được kích hoạt khi có sự kiện từ bên ngoài tác động vào chân EXTI đó, tùy theo sự kiện đó có phù hợp với điều kiện ngắt không thì ngắt ngoài mới sảy ra.

Ngắt ngoài STM32 được mô tả như sau:

Các tham số (Main Features):

* Kích hoạt độc lập trên mỗi dòng ngắt (Line Interrupts)
* Truy cập đến từng Bit trong mỗi dòng ngắt
* Tạo ra tối đa 20 sự kiện/ngắt
* Tín hiệu phải có độ rộng xung thấp hơn chu kì xung nhịp của APB2 (vì APB2 cấp xung cho GPIO)

Các Line ngắt ngoài được tổ chức như sau:



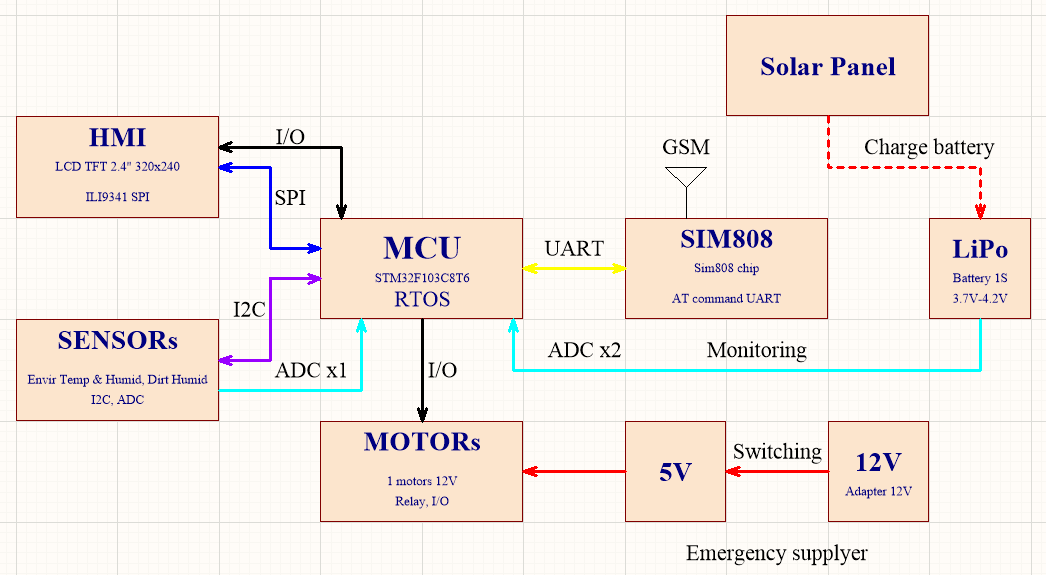
Hình ‑ Cấu trúc tín hiệu của các chân ngắt ngoài

Các chân PA0,PB0,…,PG0 sẽ chung là line EXTI0, Tiếp tục như vậy đến EXTI15 chúng ta có 16 ngắt.

4 ngắt ngoài đặc biệt đó là:

* EXTI line 16: dùng cho PVD output
* EXTI line 17: dùng cho sự kiện báo thức của bộ RTC
* EXTI line 18: dùng cho sự kiện thức dậy của USB
* EXTI line 19: dùng cho sự kiện thức dậy của ngoại vi Ethernet

### Sơ đồ nghép nối giữa STM32 và module phần cứng



Hình ‑ Sơ đồ khối giao tiếp giữa STM32 và các module phần cứng và các ngoại vi tương ứng

Hình 2.27 trình bày tổng qua về kết nối của thiết bị, trong đó có đầy đủ các module phần cứng được và các ngoại vi được sử dụng.

### Thiết kế mạch PCB

Mạch được thiết kế trên phần mềm thiết kế mạch Altium, mạch được thiết kế từ các nguyên lý đã nêu ở trên. Sau khi sắp xếp hợp lý mạch được đặt in 2 lớp và hàn thủ công bằng thiếc hàn.

#### Dạng 3D

Ảnh có chứa văn bản, thiết bị điện tử, mạch

Mô tả được tạo tự động

Hình ‑ Hình ảnh mạch 3D mặt trên

Ảnh có chứa văn bản, thiết bị điện tử, mạch

Mô tả được tạo tự động

Hình ‑ Hình ảnh mạch 3D mặt dưới

#### Mạch thực tế

Ảnh có chứa mặt đất, ngoài trời

Mô tả được tạo tự động

Hình ‑ Hình ảnh tổng quát mạch thực tế

Ảnh có chứa thiết bị điện tử, đầu nối

Mô tả được tạo tự động

Hình ‑ Hình ảnh mạch chính (góc chéo)

Ảnh có chứa thiết bị điện tử, mạch, đầu nối, bộ điều hợp

Mô tả được tạo tự động

Hình ‑ Hình ảnh mạch chính (trực diện)

## THIẾT KẾ PHẦN MỀM

### Yêu cầu xử lý để đảm bảo thời gian thực

Để đảm bảo đáp ứng với cảm ứng của màn hình, cần phải có 1 tác vụ (Task) ở mức ưu tiên cao xử lý. Tác vụ này chỉ cần thực thi khi có sự kiện màn hình cảm ứng được nhấn.

Để đảm bảo các tác vụ thực hiện đúng thời gian quy định cần phải có các bộ định thời (Timer) giữ thời gian.

Các nhiệm vụ quan trọng như hiển thị, kết nối và đẩy dữ liệu lên Server cần phải được thực thi riêng thành từng tác vụ với các mức ưu tiên hợp lý. Để giao tiếp trạng thái hoạt động của các tác vụ với nhau, cần phải sử dụng khóa Semaphore thích hợp.

### Thiết kế bộ lập lịch và xây dựng chương trình

#### Phân tích yêu cầu liên quan tới FreeRTOS

Với các yêu cầu như trên, chúng em chọn thiết kế chương trình theo mức độ ưu tiên của tác vụ cần xử lý, cụ thể như sau:

* Với tác vụ xử lý nút ấn, là tác vụ cần được ưu tiên nhất, được đặt với mức độ ưu tiên “AboveNormal” có số thứ tự ưu tiên là 32. Tác vụ này cần xử lý nhanh chóng và ngay lập tức nên kích thước vùng stack không cần nhiều, chúng em cho kích thước là 100 từ, mỗi từ tương ứng 32 bits, vậy kích thước tổng cộng là 400 bytes. Chức năng chính của tác vụ này bao gồm:
* Lấy tọa độ vị trí của lần ấn nút khi có ngắt ngoài, từ đó xác định được nút ấn nào được nhấn.
* Thay đổi trạng thái của thiết bị theo trạng thái hiện tại và nút ấn được nhấn.
* Thay đổi giao diện theo trạng thái mới nếu nút được ấn.
* Với tác vụ xử lý hiển thị màn hình LCD, nhiệm vụ chính của tác vụ là hiển thị các dữ liệu đã được thu thập, vì vậy tác vụ này cần được thực thi ngay sau khi các thông số được đọc xong, được đặt với mức độ ưu tiên “Normal4” với số thứ tự ưu tiên là 28. Việc hiển thị tốn khá nhiều RAM cho tác vụ phải lấy dữ liệu ký tự cần hiển thị nên kích thước của tác vụ yêu cần khá lớn, 220 từ tương ứng với 880 bytes.
* Tác vụ kiểm soát module SIM800L và xuất bản các bản tin MQTT chứa dữ liệu đọc được từ các cảm biến được đặt ở mức ưu tiên thấp nhất so với các tác vụ khác, “Normal” với thứ tự ưu tiên là 24. Tác vụ này có nhiệm vụ chạy ngầm và đẩy dữ liệu lên Server sau khoảng thời gian đặt trước, có thể kiểm soát tình trạng kết nối và tự động kết nối lại với Server MQTT.

Để lập trình cho các cảm biến và các hoạt động tuân thủ đúng theo các khoảng thời gian xác định, chúng em sử dụng các bộ định thời như sau:

* Bộ định thời 01 sử dụng để xác định chu kỳ kiểm tra trạng thái và chuyển đổi tự động chế độ làm việc của thiết bị hoặc tưới tiêu nếu đang ở chế độ tươi tiêu tự động. Bộ định thời này hoạt động liên tục theo chu kỳ nên ta chọn chế độ bộ định thời theo chu kỳ (Periodic).
* Bộ định thời 02 sử dụng để đặt thời gian tự động tắt của màn hình để bảo toàn năng lượng. Khi ở trong chế độ tắt thì tác vụ hiển thị màn hình LCD sẽ bị ngưng lại và sẽ được tiếp tục khi có lần chạm cảm ứng tiếp theo. Bộ định thời này chỉ cần đếm 1 lần và có thể được Reset nên ta chọn chế độ bộ định thời 1 lần (Once).
* Bộ định thời 03 sử dụng để đặt chu kỳ đọc các cảm biến. Bộ định thời này hoạt động liên tục theo chu kỳ nên ta chọn chế độ bộ định thời theo chu kỳ (Periodic).

Để các tác vụ có thể đồng bộ hoạt động thì ta cần thiết kế các khóa Semaphore hợp lý. Chúng em sử dụng 2 khóa Semaphore như sau:

* Khóa Semaphore 01 sử dụng để đưa tác vụ xử lý nút ấn từ trạng thái bị chặn và đang chờ sự kiện nút ấn về thành tác vụ chạy chính. Khóa Semaphore này chỉ có nhiều nhất là 1 chìa và được đưa từ trong hàm xử lý ngắt ngoài.
* Khóa Semaphore 02 sử dụng trong tác vụ xử lý MQTT để ngăn tác vụ này không đẩy dữ liệu lên Server khi cảm biến chưa đọc xong 1 lần đọc dữ liệu từ các cảm biến. Khóa này có nhiều nhất là 1 chìa và được đưa từ cuối của chương trình đọc cảm biến.

Dựa và các thiết lập trên, lưu đồ thuật toán của thiết bị được xây dựng như sau:



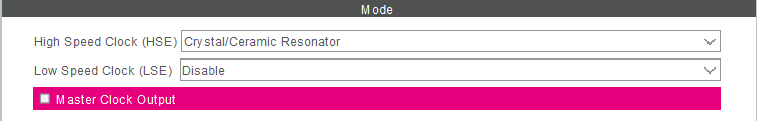
Hình ‑ Lưu đồ thuật toán của thiết bị

#### Cấu hình thiết bị trên phần mềm STM32CubeMX

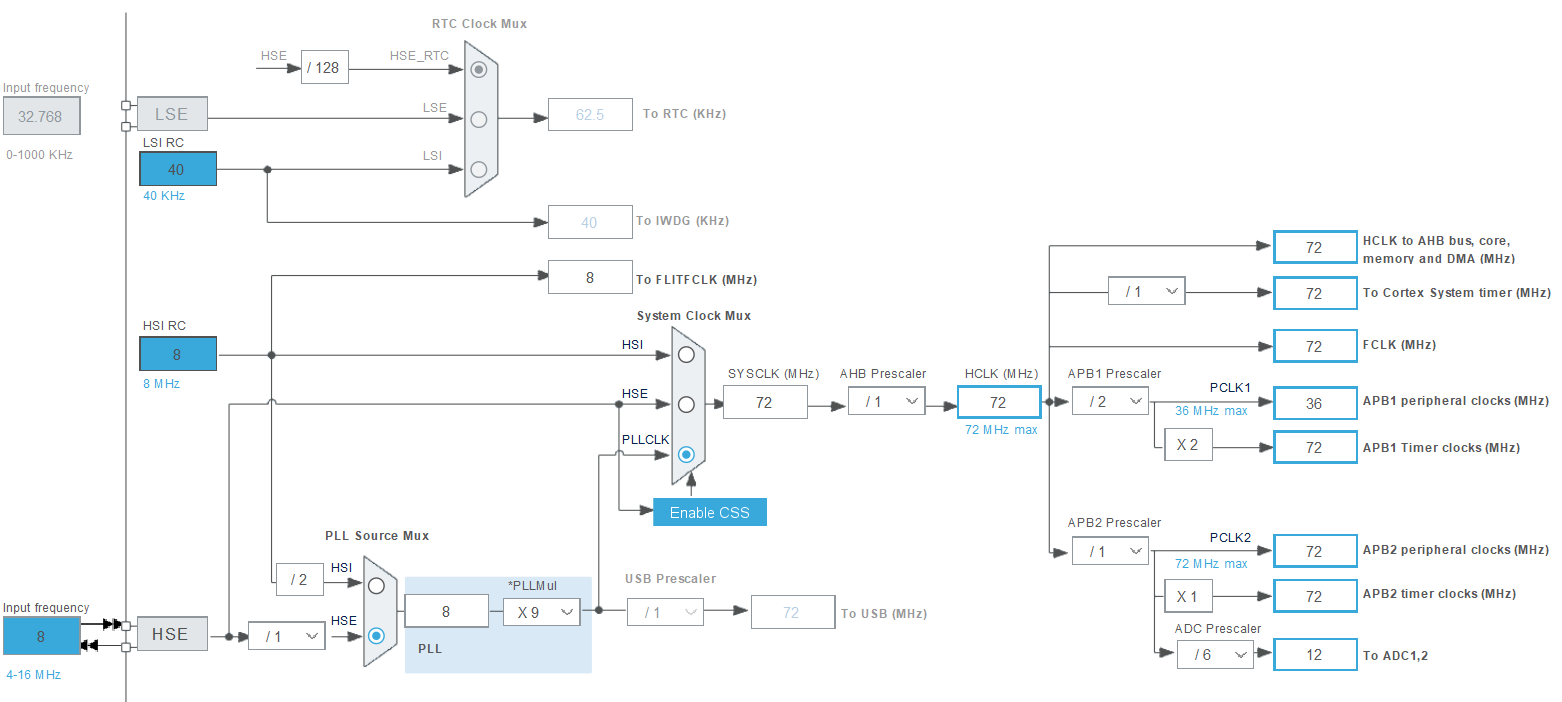
1. ***Clock và Debug***

Đầu tiên, ta tiến hành cấu hình đồng hồ xung nhịp cho chip bằng cách vào mục System Core phần RCC, ta chọn chế độ sử dụng thạch anh ngoài High Speed Clock. Trong mục Clock Configuration, ta đặt tốc độ HCLK ở mức tối đa (72MHz) như hình 3.2.

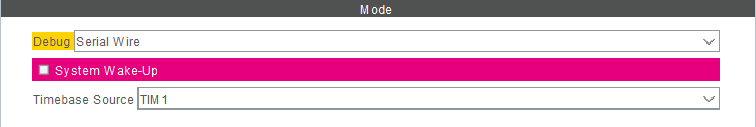
Để cấu hình chế độ debug cho chip ta vào phần SYS và chọn chế độ debug Serial Wire sử dụng giao thức SW với ST-Link như hình 3.3.



Hình ‑ Cấu hình thạch anh ngoại trong STM32CubeMX



Hình ‑ Cấu hình cây đồng hồ xung nhịp trong STM32CubeMX



Hình ‑ Cấu hình chế độ debug trong STM32CubeMX

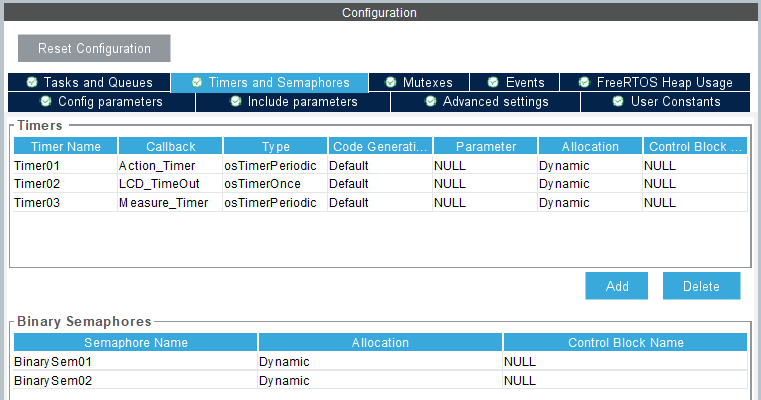
1. ***FreeRTOS***

Chúng em tiến hành cấu hình chip STM32F103C8T6 trên phần mềm STM32CubeMX. Trong phần mềm có chức năng cấu hình cho chip sử dụng FreeRTOS và tự động tạo code, thuận tiện cho việc cấu hình và lập trình. Hình 3.5 trình bày việc khai báo các tác vụ mà chúng em đã phân tích ở trên, hình 3.6 khai báo các bộ định thời các khóa Semaphore.

Ảnh có chứa bàn

Mô tả được tạo tự động

Hình ‑ Cấu hình các tác vụ trên STM32CubeMX



Hình ‑ Cấu hình bộ định thời và khóa Semaphore trên STM32CubeMX

1. ***I2C***

Để cấu hình cho ngoại vi I2C, ta vào mục Connectivity và vào phần I2C1, ngoại vi I2C được cấu hình 3.7. Chúng em sử dụng cấu hình mặc định của ngoại vi, mặc định tốc độ của xung đồng hồ I2C được đặt ở 100kHz, địa chỉ slave dạng 7-bit.

Ảnh có chứa văn bản

Mô tả được tạo tự động

Hình ‑ Cấu hình ngoại vi I2C trên STM32CubeMX

1. ***SPI***

Ảnh có chứa văn bản

Mô tả được tạo tự động

Hình ‑ Cấu hình ngoại vi SPI trên STM32CubeMX

Để cấu hình cho ngoại vi SPI, ta vào mục Connectivity và vào phần SPI2, chọn chế độ Full-Duplex Master. Chúng em sử dụng các cấu hình mặc định của ngoại vi như hình 3.8, phần prescale để set baudrate cho ngoại vi để giá trị 2 để tốc độ của ngoại vi là nhanh nhất có thể (tốc độ truyền dữ liệu tỉ lệ với tốc độ quét của màn hình).

1. ***UART***

Để cấu hình cho ngoại vi UART, ta vào mục Connectivity và vào phần USART1, chọn chế độ Asynchronous. Chúng em sử dụng các cấu hình mặc định của ngoại vi như hình 3.9, baudrate được chỉnh lên 115200 Bits/s để tăng tốc độ truyền nhận giữa vi điều khiển và module SIM800L.

Ảnh có chứa văn bản

Mô tả được tạo tự động

Hình ‑ Cấu hình ngoại vi USART trên STM32CubeMX

1. ***ADC***

Để cấu hình cho bộ chuyển đổi tương tự - số ADC, ta vào mục Analog và vào phần ADC1 để chọn các kênh muốn sử dụng, trong đây chúng em sử dụng kênh 0, 1 và 3. Các cấu hình của bộ ADC được giữ mặc định, ngoại trừ cho thời gian lấy mẫu được đưa lên mức cao nhất là 239.5 chu kỳ xung của bộ ADC (đồng hồ của ADC được set ở tần số 12MHz) như hình 3.10.

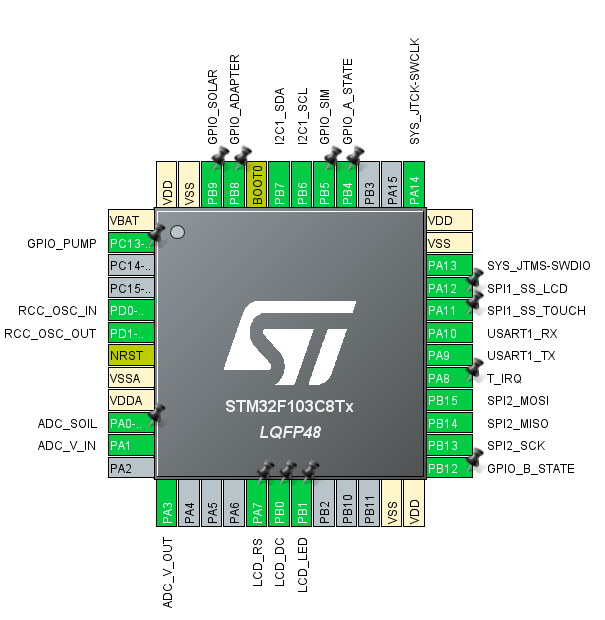
Ảnh có chứa văn bản

Mô tả được tạo tự động

Hình ‑ Cấu hình bộ ADC trên STM32CubeMX

1. ***NVIC và GPIOs***

Bước cuối cùng là cấu hình các chân GPIOs thực hiện các chứ năng điều kiển, hỗ trợ giao tiếp LCD và các cấu hình ngắt. Sơ đồ chân được sử dụng để biểu diễn ở hình 3.11, trong đó có chân PA8 được cấu hình ở chế độ GPIO\_EXTI8 ngắt ngoài. Cấu hình các chân GPIO đầu ra Push\_Pull được cấu hình ở hình 3.12. Các ngắt được cấu hình như trong hình 3.13. Chúng em sử dụng ngắt của các ngoại vi USART3, I2C1 và ngắt của TIM1 là xung clock cho FreeRTOS.



Hình ‑ Sơ đồ các chân được sử dụng và chức năng của chúng

Ảnh có chứa bàn

Mô tả được tạo tự động

Hình ‑ Cấu hình các chân GPIOs trong STM32CubeMX

Ảnh có chứa bàn

Mô tả được tạo tự động

Hình ‑ Cấu hình các ngắt của các ngoại vi trong STM32CubeMX

1. ***Lập trình Firmware***

Ảnh có chứa văn bản, màn hình, ảnh chụp màn hình, đen

Mô tả được tạo tự động

Hình ‑ Code được sinh thành thư mục được mở bằng VSCode

Thư mục lập trình được STM32CubeMX sinh ra từ các cấu hình chúng em đã đặt trước, các ngoại vi được trình bày thành các cặp file .h và .c tương ứng. Chúng em sử dụng Makefile để biên dịch code thành file hex và file bin để nạp cho chip.

File code chính là file freertos.c, các thư viện LCD, MQTT được tham khảo từ các nguồn trên mạng và được chỉnh lại theo yêu cầu của thiết bị. Đường dẫn đến các thư viện này sẽ được đính kèm ở mục tài liệu tham khảo.

File code bản hoàn chỉnh tốn tổng cộng là 60896 bytes bộ nhớ.

### Phân tích các kỹ thuật lập trình đa nhiệm đã sử dụng

1. ***Giới thiệu hệ điều hành thời gian thực***

Hệ điều hành thời gian thực là một hệ điều hành được thiết kế đặc biệt để chạy các ứng dụng với thời gian rất chính xác và mức độ tin cậy cao. Để được coi là thời gian thực, hệ điều hành phải có thời gian tối đa đã biết cho mỗi hoạt động quan trọng mà nó thực hiện. Chương trình hệ điều hành thời gian thực giúp tổ chức chương trình với nhiều tác vụ chạy đa nhiệm gần như song song với nhau. Phân bổ chương trình thành các tác vụ, các tác vụ với các mức ưu tiên khác nhau chạy đạn xen dưới sự quản lý của bộ lập lịch, việc thêm bớt các tác vụ vào mà không gây xáo trộn chương trình, cũng như không phải xây dựng lại toàn bộ chương trình. Hệ điều hành thời gian thực thực thi các tác vụ là các vòng lặp vô hạn, đo đó nó sử dụng ngắt làm cơ chế tạm dừng tác vụ đang chạy để bộ lập lịch lựa chọn tác vụ có mức ưu tiên cao nhất ở trạng thái sẵn sàng để chạy tiếp theo sau. Ngắt này có thể sinh ra bởi ngắt của các ngoại vi được người dùng xây dựng cho chương trình, ngoài ra thì hệ điều hành trang bị bộ đếm ngắt theo chu kì giúp bộ lập lịch lên hoạt động các tác vụ. Một số hệ điều hành thời gian thực phổ biến hiện nay: VxWorks Real-Time Operating System, QNX Neutrino, Free RTOS, Green Hills Software.

VxWorks Real-Time Operating System: VxWorks là phần mềm độc quyền và cần phải có giấy phép để sử dụng phần mềm đó. VxWorks được sử dụng rộng rãi trong lĩnh vực an toàn, bảo mật, IoT… và đây là một trong những Hệ điều hành thời gian thực nổi tiếng nhất được sử dụng cho các ứng dụng nhúng.

QNX Neutrino Real-Time Operating System: Chứa được sử dụng trong các ứng dụng tự động hóa công nghiệp, y tế và cả trong lĩnh vực robot.

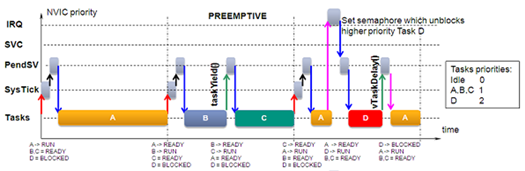
Free RTOS là một trong những hệ điều hành thời gian thực nổi tiếng với các lập trình viên. Free RTOS là một mã nguồn mở, miễn phí giúp cho lập trình viên dễ tiếp cận và sử dụng. FreeRTOS thích hợp cho những vi điều khiển có không gian chương trình bị giới hạn. Lõi FreeRTOS chiếm từ 10-15kb bộ nhớ chương trình

Phần mềm Green Hills, chủ yếu được sử dụng để bảo mật và các ứng dụng liên quan đến an toàn.

Do vậy, đối với hệ thống này, em sẽ sử dụng hệ điều hành FreeRTOS để phát triển phần mềm

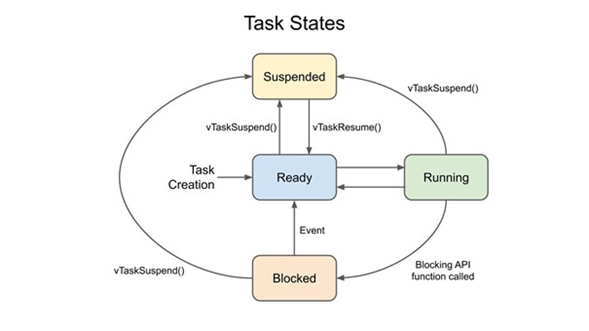
1. ***Hệ điều hành thời gian thực FreeRTOS***

Như hình trên khi xảy ra ngắt systick, tác vụ đang chạy sẽ được lưu vào ngăn xếp sau khi vào ngắt và ngay lập tức ra khỏi ngắt này, bộ lập lịch sẽ xem xét tác vụ nào có mức ưu tiên cao nhất ở trạng thái sẵn sàng để cho phép tác vụ đó được hoạt động ngay sau đó. Ở đây các tác vụ A, B, C đang ở trạng thái sẵn sàng và có mức ưu tiên ngang nhau thì bộ lập lịch sẽ lần lượt chọn đan xen các tác vụ này. Ở lần ngắt systick đầu tiên tác vụ A chạy và lần tiếp theo là tác vụ B. Tác vụ B trong quá trình hoạt động nhận API chặn tác vụ chính nó, ngay lúc này bộ lập lịch sẽ lựa chọn tác vụ tiếp theo được hoạt động lúc này là tác vụ C do cơ chế round-robin scheduling đan xen các tác vụ cùng mức ưu tiên. Các tác vụ A, B, C đan xen thực hiện cho đến khi tác vụ có mức ưu tiên cao hơn ở trạng thái sẵn sàng. Ở đây khi một ngắt diễn ra cho phép bỏ chặn tác vụ D, sau khi thực hiện xong ngắt bộ lập lịch xét các vụ mức ưu tiên cao nhất đang ở trạng thái sẵn sàng được thực hiện là tác vụ D. Khi tác vụ mưc ưu tiên cao nhất D bị chặn thì lại lần lượt các tác vụ khác có mức ưu tiên cao nhất đang sẵn sàng có thể được thực hiện.



Hình ‑ Mô hình chạy các tác vụ

1. ***Các trạng thái của tác vụ***



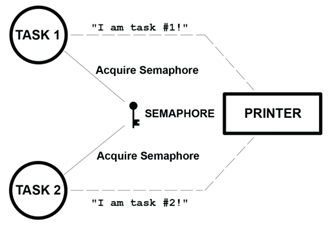
Hình ‑ Mô tả các trạng thái của tác vụ và cách chuyển đổi giữa các trạng thái

Trong RTOS thì task có 4 trạng thái:

* Ready: Task sẵn sang để có thể thực thi nhưng chưa được thực thi do các task khác với độ ưu tiên ngang bằng hoặc hơn đang chạu
* 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
* Suppended: Task ở trạng thái treo, về cơ bản thì trạng thái này cũng tương tự như Blocked. Nhưng ddiemr khác nhau là cách chuyển trạng thái hiện tại sang trạng thái Ready. Chỉ gọi hàm vTaskResume() thì task bị treo mới chuyển sang trạng thái ready để thực thi.

Trong dự án này em sử em sử dụng vTaskSupend này để điều khiển tác vụ hiển thị. Tác vụ hiển thị chỉ hoạt động trở lại khi cảm biến đọc xong dữ liệu, update giá trị hiển thị hoặc đến chu kì sáng của màn hình.

1. ***Quản lí truy cập tài nguyên và đồng bộ sự kiện***



Hình ‑ Mô tả hoạt động của khóa Semaphore

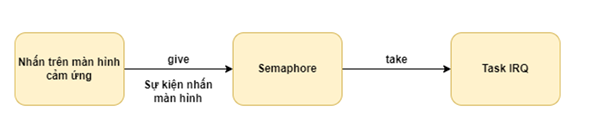
Để quản lý tài nguyên hệ điều hành thười gian thực sử dụng các khóa semaphores. Có thể sử dụng song song với nhiều lượng truy cập tài nguyên cùng lúc với số lượng khóa định sẵn, cho phép số lượng tác vụ nhiều nhất thao tác tài nguyên theo số khóa mở ra. Ở trên hình ví dụ một khóa semaphore giao tiếp in ra màn hình, nếu không có khóa semaphore này các tác vụ hoạt động đan xem thì dữ liệu in ra sẽ thao tác đạ xem giữa hai tác vụ và dòng chữ hiển thị sẽ là trộn lẫn của cả hai tác vụ như vậy dữ liệu xuất ra đã bị sai thông tin. Vì thế cơ chế này tạo ra để khóa chỉ cho phép một tác vụ truy xất tới tài nguyên này và nó sẽ bị khóa không cho tác vụ khác truy xuất vào cho tới khi tác vụ kia thực hiện xong trả khóa lại cho tác vụ khác có nhu cầu vào truy xuất với tài nguyên này.



Hình ‑ Mô tả hoạt động của khóa Semaphore 02

Ở đây, em sử dụng semarphore để cấp phép truy cập vào data. Dữ liệu của Tác vụ cảm biến đo nhiệt độ, độ ẩm. Khi thực hiện xong việc lấy dữ liệu nó sẽ cấp quyền truy cập dữ liệu này cho tác vụ MQTT. Việc sử dụng semaphore làm cho dữ liệu được toàn vẹn, tránh xung đột khi truy cập cùng tài nguyên.

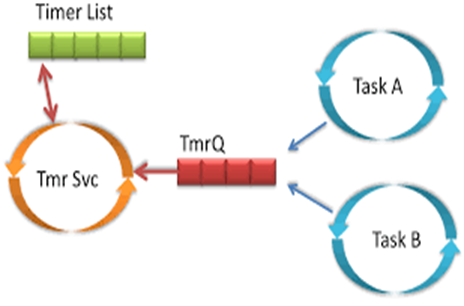
Ngoài ra, em còn sử dụng semarphore để đồng độ khi có sự kiện điều khiển từ màn hình cảm biến.



Hình ‑ Mô tả hoạt động của khóa Semaphore 01

Khi có sự kiện màn hình được nhấn, Task IRQ sẽ được phép hoạt động và thực hiện quá trình điều khiển động cơ và hiển thị màn hình TFT.

1. ***Định chu kì cho các task***



Hình ‑ Mô tả hoạt động của Software Timer

Trong đề tài này, em sử dụng timer mềm để định chu kì cho các task. Ưu điểm của việc sử dụng timer mềm sẽ không bị giới hạn bởi phần cứng, do đó sử dụng nhiều timer một lúc. Mỗi tác vụ sẽ được được cấp một chu kì xác định. Timer Service sẽ kiểm tra timer nào đến deadline, nó cho tác vụ đó hoạt động. Các tác vụ dùng timer mềm để định thời gian là: tác vụ measure, tác vụ action, tác vụ turn LCD on/off.

## KẾT QUẢ VÀ ĐÁNH GIÁ

### Kết quả đạt được

Ảnh có chứa văn bản, tường, màn hình, bảng điểm

Mô tả được tạo tự động

Hình ‑ Màn hình thiết bị sử dụng nguồn ngoài

Sau quá trình lập trình và sửa lỗi, thiết bị đã hoạt động đúng với dự tính, hình 4.1 là hình ảnh của màn hình cảm ứng LCD TFT hiển thị toàn bộ các thông số của các cảm biến và ADCs. Hàng trên cùng có hiển thị trạng thái module SIM và trạng thái kết nối với Server, dấu mũi tên ký hiệu thiết bị đang trong quá trình đẩy dữ liệu lên. Ở phía dưới là các nút điều khiển bơm tưới tiêu và nút chuyển chế độ sử dụng năng lượng, hiện tại thiết bị đang trong trạng thái sử dụng pin và tự động bơm mỗi 15s khi điện áp của cảm biến độ ẩm đất nằm trên 2.5V. Hình 4.2 là hình ảnh thiết bị trong trạng thái sử dụng năng lượng pin

Quá trình khởi tạo màn hình ban đầu mất tầm 1000ms và các số liệu được in rõ ràng sau khoảng thời gian quy định. Tốc độ đáp ứng với cảm ứng là rất nhanh, tầm 100ms và màn hình thay đổi theo trạng thái trong vòng 100ms tiếp theo. Thời gian đọc cảm biến là 1400ms, thời gian để thực hiện 1 lần cập nhập màn hình theo số liệu mới là 85ms, thời gian gửi bản tin ping để giữ kết nối là 4200ms và thời gian để đẩy toàn bộ dữ liệu của các cảm biến lên Server là 29400ms. Các số liệu về thời gian cùng số Tick được lấy ở chế độ debug và được biểu diễn ở hình 4.3.

Ảnh có chứa văn bản, bảng điểm, màn hình

Mô tả được tạo tự động

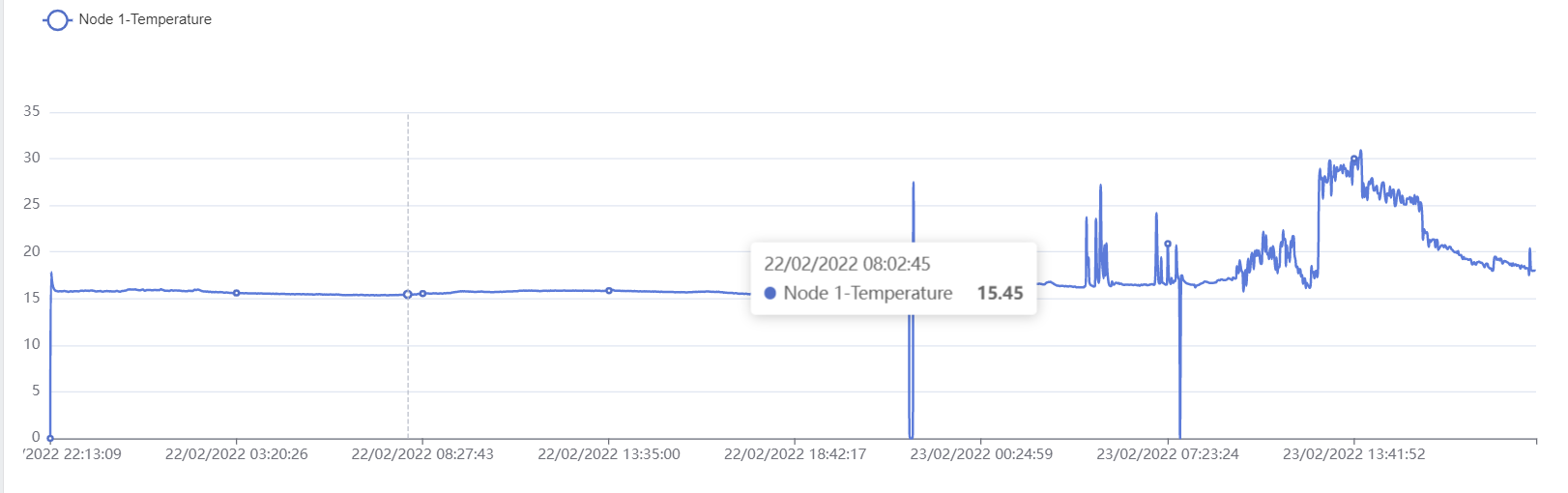
Hình ‑ Màn hình thiết bị ở chế độ

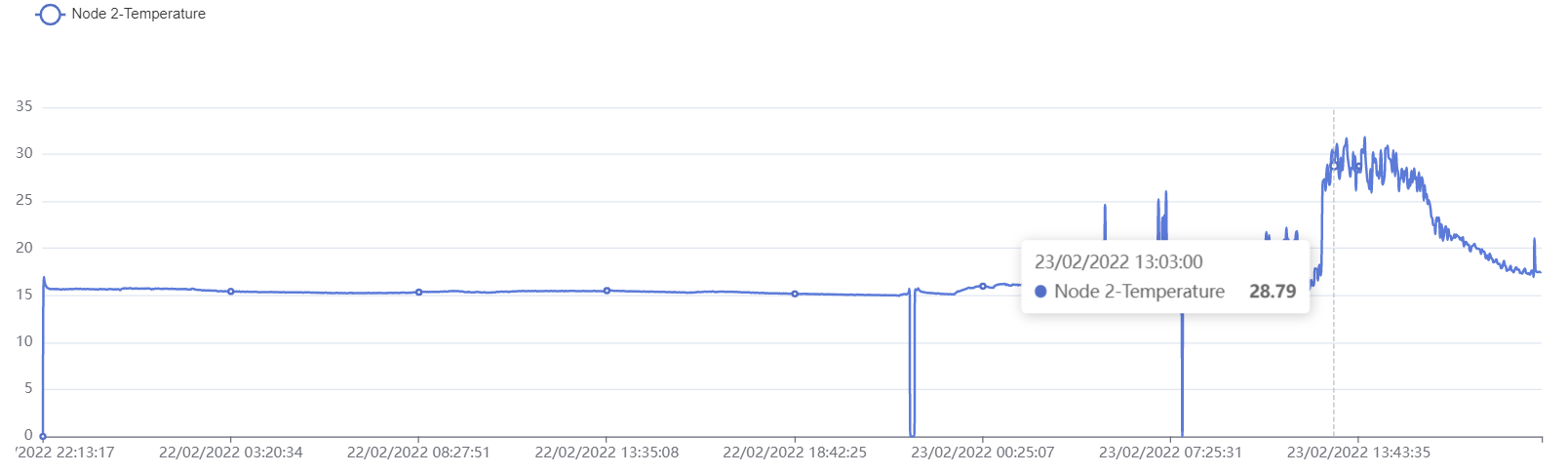
Ảnh có chứa văn bản, màn hình, bảng điểm

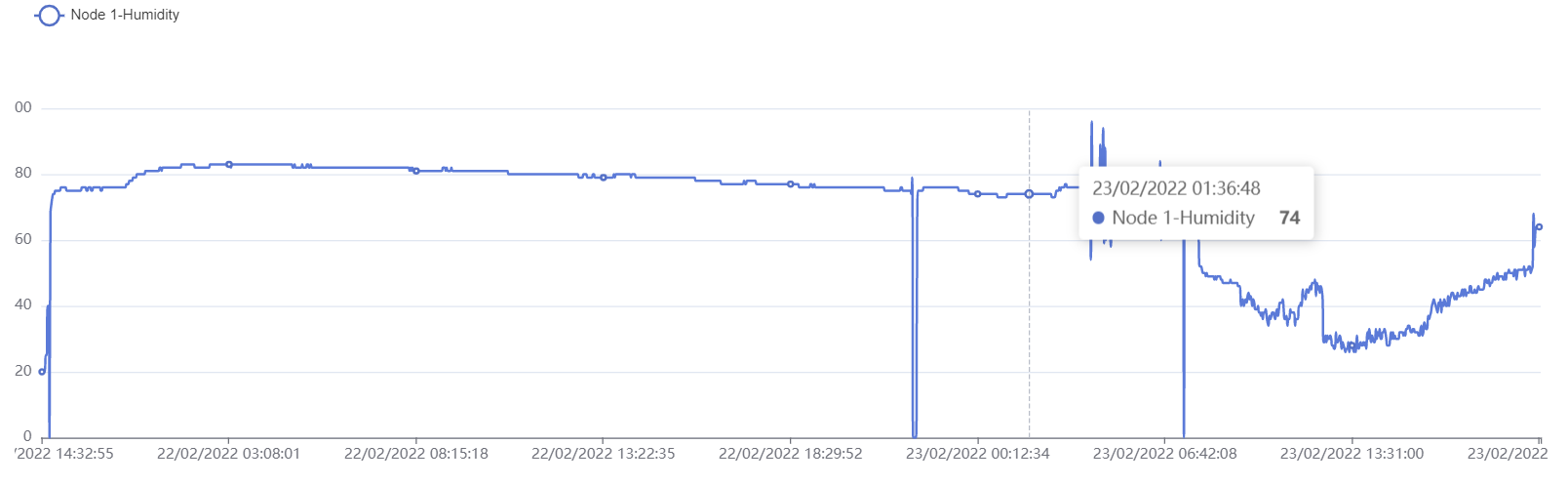
Mô tả được tạo tự độngẢnh có chứa văn bản, màn hình, bảng điểm

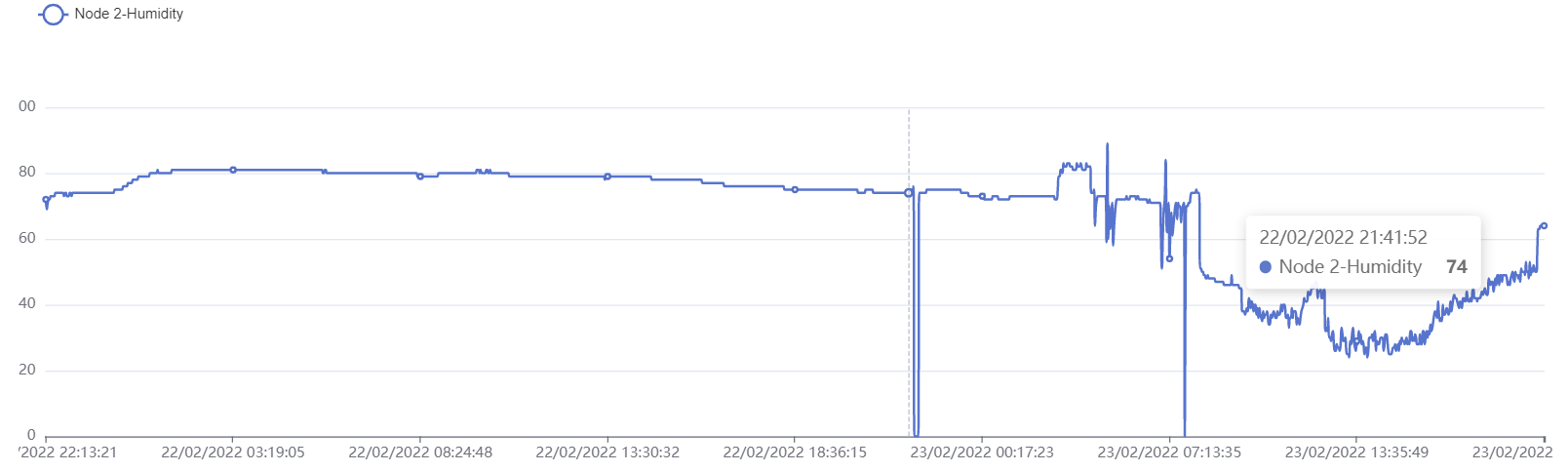
Mô tả được tạo tự động

Hình ‑ Các thông số về thời gian các tác vụ



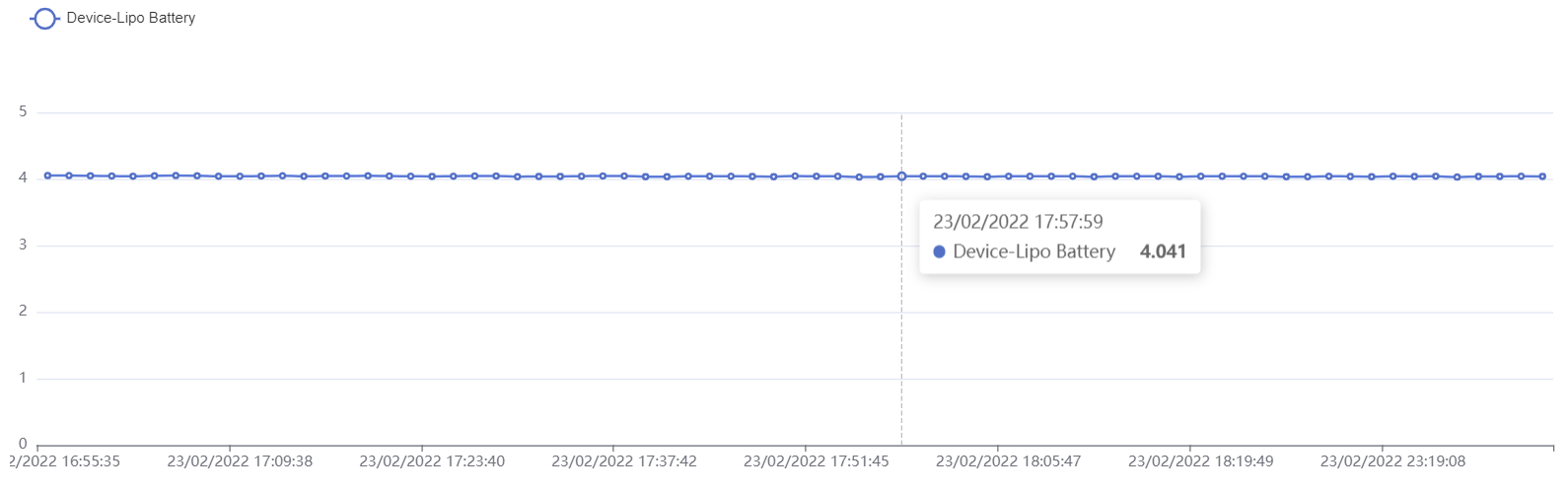






Ảnh có chứa bàn

Mô tả được tạo tự động





Hình ‑ Các đồ thị trên giao diện [Website](http://139.59.122.84:3000/)

Hình 4.4 trình bày đồ thị các số liệu thiết bị đã thu thập được và đưa lên Website. Dữ liệu đọc từ các cảm biến là khá chính xác dựa vào nhiệt độ trong phòng tại Hà Nội và ngoài trời tại sân ĐH Bách Khoa Hà Nội trưa chiều ngày 23/02/2021 dưới thời tiết nắng to. Các điểm dữ liệu bất thường là quá trình lập trình và kiểm tra đáp ứng của cảm biến.

### Phân tích và đánh giá

Tác vụ MQTT là tác vụ có mức ưu tiên thấp nhất nên thường phải nhường quyền cho các tác vụ cao hơn, do đó thời gian thực thi của mỗi lần truyền dữ liệu lên là khá lớn (gần 30s đẩy toàn bộ dữ liệu và hơn 4s ping). Dữ liệu được đưa lên MQTT Server ở đúng các topic quy định trước theo chu kỳ quy định là 30s một bản tin xem kẽ ping và dữ liệu và quá trình truyền không ảnh hưởng tới hoạt động của các tác vụ còn lại.

Tác vụ LCD được thực thi rất nhanh để đảm bảo tính thời gian thực của cảm ứng và hiển thị số liệu (tầm 85ms) do màn hình chỉ cần thay đổi các thông số ở các trường nhất định thay vì phải quét lại toàn màn hình. Tác vụ đi vào trạng thái ngừng khi hết thời gian quy định là 60s mà không có sự kiện chạm, trong trạng thái này màn hình tắt và thiết bị không còn cần phải cập nhập màn hình.

Tác vụ xử lý ngắt ngoài do sự kiện cảm ứng được thực thi nhanh chóng theo đúng dự định, thời gian đáp ứng tầm 103ms, trong đó 100ms đầu để xử lý chống rung nút và 3ms sau để xử lý vị trí cảm ứng và thực hiện thay đổi trên màn hình và các trạng thái. Tác vụ thực hiện không làm ảnh hưởng đến hoạt động của các tác vụ khác.

Thời gian đọc của các cảm biến là hợp lý, trong đó thời gian chờ đọc cảm biến nhiệt độ độ ẩm chiếm gần như toàn bộ thời gian (700ms chờ mỗi cảm biến, tổng 1400ms), thời gian tính toán, chuyển đổi ADCs và các công việc khác tốn dưới 1ms. Tốc độ này là có thể chấp nhận được. Các cảm biến được đọc và màn hình được cập nhập đúng theo chu kỳ quy định là 3s.

Thời gian của các bộ định thời là chuẩn xác và các chương trình của bộ định thời được thực thi mà không làm ảnh hưởng tới các hoạt động của các tác vụ khác.

### Đánh giá thành viên và công việc được giao

###### Trần Quang Minh

Nhận nhiệm vụ:

* Lên ý tưởng đề tài, tìm hiểu và xây dựng chương trình
* Thiết kế mạch nguyên lí và PCB
* Code giao tiếp giữa module phần cứng với STM32 như: khối cảm biến, động cơ bơm nước, ADC, module SIM 800L, màn hình TFT cảm ứng.
* Thiết kế giao diện cho màn hình TFT
* Xây dựng lưu đồ thuật toán cho chương trình
* Lập trình đa nhiệm FreeRTOS, chia công việc thành nhiều task, áp dụng các kĩ thuật như: phân task, timer, suspend, semaphore,...
* Thử nghiệm và đánh giá kết quả.

Nhận xét:

* Hoàn thành nhiệm vụ
* Liên kết tốt với các thành viên khác và tích cực tham gia đóng góp, sửa đổi và phát triển.

Đánh giá đóng góp: 10/10

###### Trịnh Văn Thái

Nhận nhiệm vụ:

* + Tìm hiểu, xây dựng các chương trình,
  + Thiết kế mạch nguyên lí và PCB
  + Code giao tiếp giữa module phần cứng với STM32 như: khối cảm biến, động cơ bơm nước, ADC, module SIM 800L
  + Xây dựng lưu đồ thuật toán cho chương trình
  + Lập trình đa nhiệm FreeRTOS, chia công việc thành nhiều task, áp dụng các kĩ thuật như: phân task, timer, suspend, semaphore,...
  + Thử nghiệm và đánh giá kết quả.

Nhận xét:

* Hoàn thành nhiệm vụ
* Liên kết tốt với các thành viên khác và tích cực tham gia đóng góp, sửa đổi và phát triển.

Đánh giá đóng góp: 10/10

###### Phạm Xuân Chính

Nhận nhiệm vụ:

* Tìm hiểu, xây dựng các chương trình
* Thiết kế mạch nguyên lí giao tiếp giữa STM32 và module SIM800L
* Code giao tiếp giữa module phần cứng với STM32 như: khối cảm biến, động cơ bơm nước, ADC, module SIM 800L
* Xây dựng lưu đồ thuật toán cho chương trình
* Lập trình đa nhiệm FreeRTOS, chia công việc thành nhiều task, áp dụng các kĩ thuật như: phân task, timer, suspend, semaphore,...
* Thử nghiệm và đánh giá kết quả

Nhận xét:

* Hoàn thành nhiệm vụ
* Liên kết tốt với các thành viên khác và tích cực tham gia đóng góp, sửa đổi và phát triển

Đánh giá đóng góp: 10/10

###### Phí Gia Phong

Nhận nhiệm vụ:

* Thiết kế hệ thống server lưu trữ thông tin đo đạc từ các thiết bị gửi lên
* Thiết kế giao diện website giúp hiển thị thông số đo đạc, đồ thị cập nhật thời gian thực
* Thuê máy chủ ảo hosting (VPS), setup môi trường production deploy website

Nhận xét:

* Hoàn thành nhiệm vụ
* Liên kết tốt với các thành viên khác và tích cực tham gia đóng góp, sửa đổi và phát triển

Đánh giá đóng góp: 10/10

## LINK SHARE THƯ MỤC LẬP TRÌNH

Link GitLab

<https://gitlab.com/zseefvhu12345/design-embedded-system-2021-1>

Link GitHub Web

<https://github.com/tpw284/embedded-system>

Link Website

[http://139.59.122.84:3000/](http://139.59.122.84:3000/?fbclid=IwAR1BDy4B9mL2TapFzZrZ5uo6Qq0xo3b3Ro6J64xjAZ04HEu23FTxNP6gK1k)

Link Onedrive

<https://husteduvn-my.sharepoint.com/:f:/g/personal/minh_tq181659_sis_hust_edu_vn/Erqbq5DXQTROjAvZInnZAU0B1zd6ImPW6LRNI4XTytJKIQ?e=x7hUni>

# KẾT LUẬN

Sau thời gian hơn 2 tháng thực hiện, các kết quả đo đạc đã đáp ứng được yêu cầu đặt ra ban đầu. Các thành viên hiểu được các xây dựng một hệ thống nhúng, lập trình đa nhiệm FreeRTOS và triển khai thực tế. Tuy nhiên do hạn chế về thời gian và dịch bệnh phức tạp, chúng em chưa hoàn thiện được phần vỏ của sản phẩm. Sau khi kết thúc học phần thiết kế hệ thống nhúng, chúng em sẽ cải thiện thêm về phần cứng giúp thu nhỏ kích thước, nâng cao hiệu suất thu hoạch năng lượng mặt trời. Đồng thời, chúng em cũng cải thiện, thêm giao thức truyền thông mới như NbIoT,… để giảm công suất tiêu thu của sản phẩm

**DANH MỤC TÀI LIỆU THAM KHẢO**

|  |  |
| --- | --- |
| [1] | Các tài liệu đính kèm trong file REFERENCE |
| [2] | Thư viện cho SIM800L MQTT  <https://github.com/leech001/SIM800MQTT> |
| [3] | Thư viện cho LCD TFT ILI9341  <https://github.com/afiskon/stm32-ili9341> |
| [4] | Các datasheet của các cảm biến |
| [5] | Tài liệu của CMSIS-RTOS2  <https://www.keil.com/pack/doc/cmsis/RTOS2/html/index.html> |
| [6] | Các tài liệu, blog, video,.. trên mạng liên quan đến FreeRTOS và các cảm biến |
| [7] |  |
| [8] |  |
| [9] |  |