|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI  **VIỆN ĐIỆN TỬ - VIỄN THÔNG**  logo_128  **MÔN HỌC: THIẾT KẾ HỆ THỐNG NHÚNG VÀ GIAO TIẾP NHÚNG**  **Giảng Viên Hướng dẫn:** TS. Phạm Văn Tiến  **Đề Tài:**  **Máy soát vé lên, xuống tàu điện**  **Sinh viên thực hiện:**   |  |  | | --- | --- | | Nguyễn Thanh Lâm | 20172644 | | Nguyễn Nhật Bằng | 20172422 | | Phạm Hữu Chỉnh | 20172437 | | Bùi Trung Kiên | 20172638 | | Nguyễn Quang Huy | 20182583 |   Hà Nội, 12/2021 |

# Lời mở đầu

Đặt vấn đề Trong nhiều năm trở lại đây, sự phát triển vượt bậc của khoa học kỹ thuật đặc biệt là trong các lĩnh vực truyền thông công nghiệp và công nghệ phần mềm đã mang lại rất nhiều giải pháp, khả năng mới trong nhiều lĩnh vực. Theo thời gian, các hệ thống tự động cũng lần lượt ra đời và mang lại rất nhiều tiện ích trong các lĩnh vực khác nhau như hàng không, sản xuất công nghiệp, vận tải… Với khả năng ứng dụng lớn trong nhiều lĩnh vực như vậy, việc xây dựng một hệ thống soát vé lên xuống tàu điện là hết sức cần thiết.

Mục đích đề tài Từ vấn đề nêu trên, dựa vào việc tìm hiểu cở sở hạ tầng mạng tại Việt Nam cũng như trên thế giới, hiểu được nguyên lý soát vé lên xuống tàu điện, chúng em đã lên ý tưởng cho việc xây dựng một mô hình hệ thống tự động soát vé. Một hệ thống tự động soát vé lên xuống tàu điện với các chức năng đóng/mở barie tự động, trừ tiền, thu/trả vé, nối chuyến,… chắc chắn sẽ rất hữu ích và phù hợp trong môi trường nghiên cứu và học tập của sinh viên. Hơn nữa, hiện nay mỗi cá nhân đều sử dụng máy tính có kết nối internet làm phương tiện làm việc, học tập, nghiên cứu do đó việc thiết lập được một hệ thống như trên là vô cùng đơn giản

# MỤC LỤC

[Lời mở đầu 2](#_Toc92139351)

[MỤC LỤC 3](#_Toc92139352)

[DANH MỤC HÌNH VẼ 5](#_Toc92139353)

[DANH MỤC Bảng Biểu 6](#_Toc92139354)

[Bảng phân công công việc 7](#_Toc92139355)

[Chương I. Chỉ Tiêu Kỹ Thuật 8](#_Toc92139356)

[Chương II. Mô hình hóa hệ thống 9](#_Toc92139357)

[1. Sơ đồ Use Case 9](#_Toc92139358)

[2. Sơ đồ tuần tự 9](#_Toc92139359)

[2.1. Lên Tàu 9](#_Toc92139360)

[2.2. Xuống Tàu 10](#_Toc92139361)

[3. Scope Model 10](#_Toc92139362)

[Chương III. Đồng thiết kế cứng mềm 12](#_Toc92139363)

[1. Sơ đồ khối hệ thống 12](#_Toc92139364)

[2. Lựa chọn giải pháp phần cứng 12](#_Toc92139365)

[3. Tính toán thiết kế khối nguồn 15](#_Toc92139366)

[4. Lựa chọn giải pháp phần mềm 16](#_Toc92139367)

[4.1. Hệ Điều Hành: 16](#_Toc92139368)

[4.2. Web Client: 17](#_Toc92139369)

[Chương IV. Thiết kế giao tiếp ngoại vi 19](#_Toc92139370)

[1. ESP8266 12f với module đọc mã vạch 19](#_Toc92139371)

[2. STM32 với ESP8266 12f 20](#_Toc92139372)

[3. STM32 với các thiết bị ngoại vi khác 22](#_Toc92139373)

[Chương V. Tổng hợp, cấu hình, biên dịch hệ điều hành 30](#_Toc92139374)

[1. Tổng quan về FreeRtos 30](#_Toc92139375)

[2. Áp dụng FreeRtos vào dự án 31](#_Toc92139376)

[Chương VI. Phát triển phần mềm 36](#_Toc92139377)

[1. Web-client mqtt 36](#_Toc92139378)

[2. Esp-client mqtt 39](#_Toc92139379)

[Chương VII. Triển khai, mô phỏng hệ thống 41](#_Toc92139380)

[Chương VIII. Phân tích, đánh giá kết quả 44](#_Toc92139381)

[Báo Cáo Cá Nhân 46](#_Toc92139382)

[I. Thực hiện nhiệm vụ 46](#_Toc92139383)

[1. Lựa chọn giải pháp phần mềm 46](#_Toc92139384)

[1.1. Hệ Điều Hành: 46](#_Toc92139385)

[1.2. Web Client Broker: 47](#_Toc92139386)

[2. Triển khai FreeRtos 48](#_Toc92139387)

[3. Xây dựng web client broker 53](#_Toc92139388)

[II. Kết quả đạt được 56](#_Toc92139389)

# DANH MỤC HÌNH VẼ

[Hình 1: Sơ đồ Use Case hệ thống 8](#_Toc91444901)

[Hình 2: Sơ đồ trình tự xử lý khi khách lên tàu 9](#_Toc91444902)

[Hình 3: Sơ đồ trình tự xử lý khi khách xuống tàu 9](#_Toc91444903)

[Hình 4: Sơ đồ khối hệ thống soát vé 11](#_Toc91444904)

[Hình 5: Lựa chọn giải pháp phần cứng cho từng khối 11](#_Toc91444905)

[Hình 6: Adapter 5V 1A 14](#_Toc91444906)

[Hình 7: Khối nguồn nuôi Esp8266 12f 15](#_Toc91444907)

[Hình 8:Sơ đồ kết nối chân của GM65 và esp8266 12f 18](#_Toc91444908)

[Hình 9: Xử lý truyền dữ liệu và đọc dữ liệu để giao tiếp với GM65 18](#_Toc91444909)

[Hình 10: Kết quả 19](#_Toc91444910)

[Hình 11: Giao tiếp STM32 với ESP8266 12f 20](#_Toc91444911)

[Hình 12: Gửi mã tới Stm32f103 20](#_Toc91444912)

[Hình 13: Mã nhận được từ esp 20](#_Toc91444913)

[Hình 14:Cấu hình cho stm32f103c6 trên STM32CUBEIDE 21](#_Toc91444914)

[Hình 15: Kết quả mô phỏng 22](#_Toc91444915)

[Hình 16:LCD 1602 xanh lá 23](#_Toc91444916)

[Hình 17: LCD 1602 Xanh dương 5v 24](#_Toc91444917)

[Hình 18: Cấu hình stm32 với LCD 25](#_Toc91444918)

[Hình 19: Kết quả mô phỏng 25](#_Toc91444919)

[Hình 20: Cấu hình stm32 điều khiển motor 27](#_Toc91444920)

[Hình 21: Kết quả mô phỏng 28](#_Toc91444921)

[Hình 22: So sánh giữa đa tác vụ và siêu vòng lặp 29](file:///C:\Users\Admin\Desktop\Embedded.docx#_Toc91444922)

[Hình 23: Các mức độ ưu tiên trong phiên bản cmsis-v1 30](#_Toc91444923)

[Hình 24: Lịch trình của hệ thống 30](#_Toc91444924)

[Hình 25: CMSIS 31](#_Toc91444925)

[Hình 26: Thiết lập FreeRtos trong CubeMX 32](#_Toc91444926)

[Hình 27: Thiết lập các tác vụ 32](#_Toc91444927)

[Hình 28: Khởi tạo ID cho các Task 33](#_Toc91444928)

[Hình 29: Khởi tạo các Task 33](#_Toc91444929)

[Hình 30: Xây dựng các hàm thực thi nhiệm vụ 33](#_Toc91444930)

[Hình 31: Định nghĩa và tạo luồng cho các tác vụ 34](#_Toc91444931)

[Hình 32: Kết quả 34](#_Toc91444932)

[Hình 33: Sơ đồ mô tả các tác vụ của web-client 35](#_Toc91444933)

[Hình 34: Giao diện web-client 36](#_Toc91444934)

[Hình 35: Tính toán chi phí hành trình 37](#_Toc91444935)

[Hình 36: Thông tin trao đổi giữa máy soát vé và web-client 37](#_Toc91444936)

[Hình 37: Gửi tín hiệu cần xử lý lên web-client 38](#_Toc91444937)

[Hình 38: Kết quả trả về 39](#_Toc91444938)

# DANH MỤC Bảng Biểu

[Bảng 1: Thông số kỹ thuật GM65 13](#_Toc91444939)

[Bảng 2: Thông số kỹ thuật STM32f103 14](#_Toc91444940)

[Bảng 3: Thông số kỹ thuật Esp8266 12f 15](#_Toc91444941)

[Bảng 4: Thông số khối nguồn nuôi Esp8266 12f 16](#_Toc91444942)

# Bảng phân công công việc

|  |  |
| --- | --- |
| Tên thành viên | Nhiệm vụ |
| Nguyễn Thanh Lâm | - Đồng thiết kế cứng mềm (Lựa chọn giải pháp phần cứng)  - Thiết kế giao tiếp ngoại vi ESP8266 12f với module đọc mã vạch |
| Nguyễn Nhật Bằng | - Thiết kế giao tiếp ngoại vi STM32 với ESP8266 12f  - Phát triển phần mềm Esp-client mqtt |
| Phạm Hữu Chỉnh | - Thiết kế giao tiếp ngoại vi STM32 với các thiết bị ngoại vi khác |
| Bùi Trung Kiên | - Đồng thiết kế cứng mềm (lựa chọn giải pháp phần mềm)  - Biên dịch hệ điều hành  - Phát triển phần mềm Web-client mqtt |
| Nguyễn Quang Huy | - Mô hình hóa hệ thống  - Thiết kế giao tiếp ngoại vi ESP8266 12f với module đọc mã vạch |

# Chương I. Chỉ Tiêu Kỹ Thuật

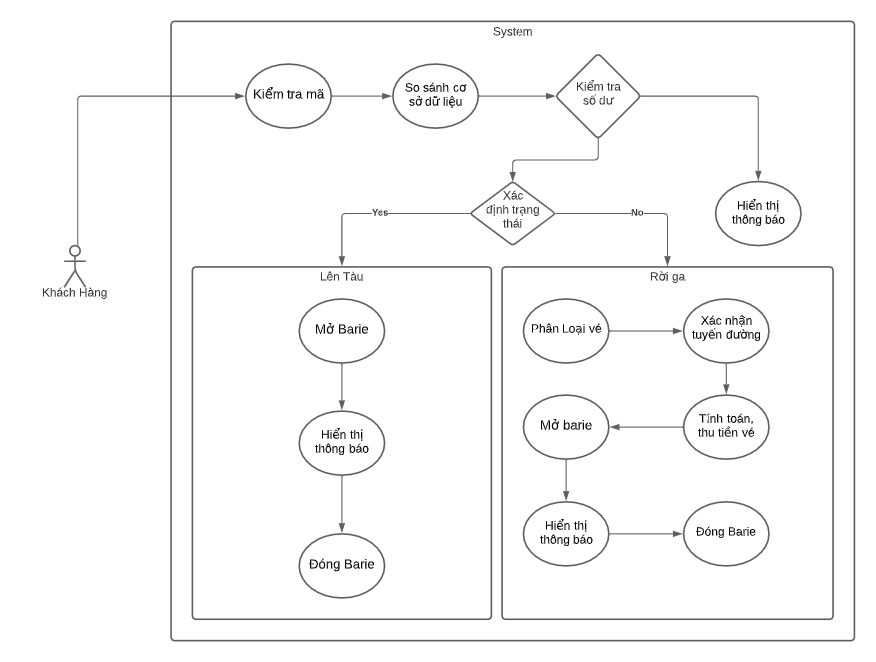
Với các yêu cầu Đóng/mở barie tự động, trừ tiền, thu/trả vé, nối chuyến, phân loại vé người lớn, trẻ nhỏ... Ta có thể xác định được các yếu tố sau đây về hệ thống khi áp dụng tại Việt Nam:

* Khi có khách hàng sử dụng hệ thống:
* Khi khách hàng quét vé lên tàu, Hệ thống sẽ quét mã -> Kiểm tra số dư khả dụng -> mở barie -> xác nhận điểm đi -> Một màn LCD hiển thị thông báo -> đóng barie.
* Khi khách hàng quét vé rời ga, Hệ thống sẽ quét mã -> Phân loại vé người lớn/ trẻ em -> kiểm tra điểm đi -> Xác nhận điểm đến -> Tính toán giá vé -> Thu tiền vé -> Một màn LCD hiển thị thông báo -> đóng barie.
* Phía server sẽ tiến hành nhận và xử lý mã các client gửi về và gửi trả tín hiệu phản hồi cho các client.
* Đầu vào:
* Mã vé (Một định dạng mã, kí hiệu dùng để nhận diện).
* Một ngắt được tạo ra ở tất cả các cổng khi có mã được quét.
* Một thông báo/ lệnh được gửi tới từng cổng.
* Đầu ra:
* Barie mở.
* Thông báo được hiển thị trên màn hình LCD.
* Barie đóng.
* Chức năng hệ thống:
* Hệ thống sử dụng module quét mã khi khách hàng quét thẻ, vé.
* Hệ thống mở barie khi đã kiểm tra số dư khả dụng hoặc xác nhận thanh toán từ khách hàng, màn hình LCD hiển thị thống báo, và barie đóng khi khách đã đi qua barie.
* Sử dụng CSDL chung cho toàn bộ hệ thống.
* Chỉ số thiết kế:
* Sự thất thoát năng lượng: Đảm bảo duy trì mức thấp.
* Thời hạn xử lý: Bộ đếm thời gian cần phải được đặt để bất cứ khi nào khách hàng sử dụng hệ thống, hệ thống cần phải đưa ra phản hồi cho khách hàng trong vòng vài giây để đóng/ mở barie, thu và hủy vé tuyến và giao tiếp qua màn hình LCD với khách hàng.
* Chi phí kỹ thuật: Chi phí cho việc kiểm tra - chỉnh sửa - gỡ lỗi phần cứng và phần mềm
* Sự an toàn: Các đảm bảo an toàn cho hệ thống cần phải được đảm bảo, tránh gây ra các sự cố cho người dùng. Có tác động xử lý kịp thời khi sự cố xảy ra.

# Chương II. Mô hình hóa hệ thống

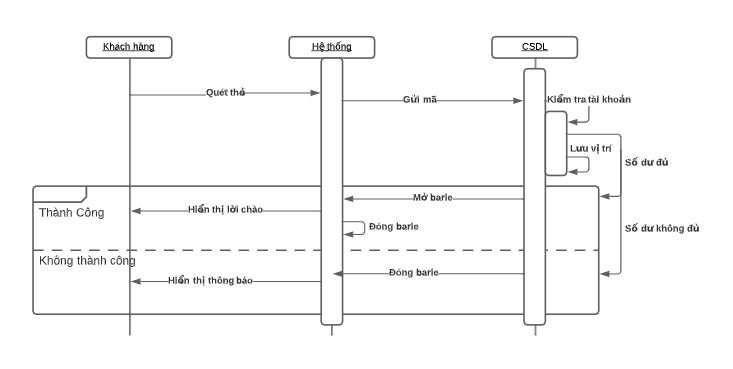
Từ các chỉ tiêu kỹ thuật đã được nêu ra ở Chương I của báo cáo này, sử dụng công cụ thiết kế hệ thống UML để mô hình hóa 1 cách khái quát lại các nhiệm vụ, chức năng và trình tự tiến hành mà hệ thống cần thực hiện.

1. Sơ đồ Use Case



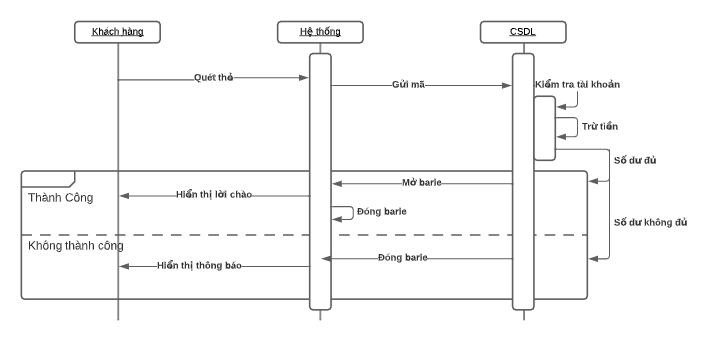
Hình 1: Sơ đồ Use Case hệ thống

1. Sơ đồ tuần tự
   1. Lên Tàu



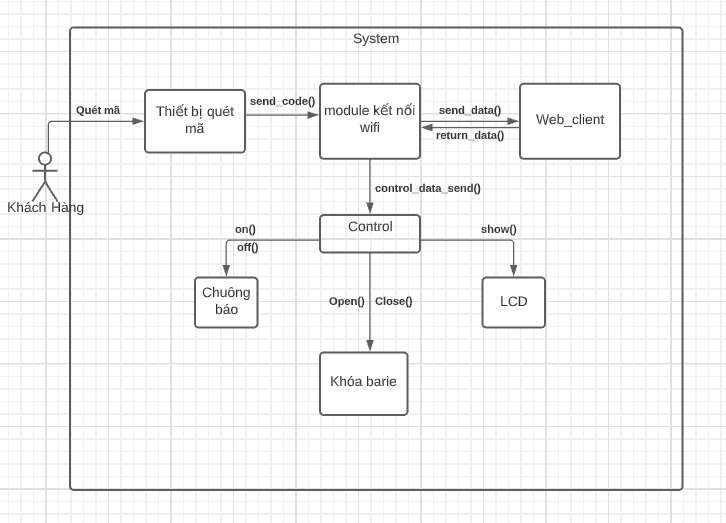
Hình 2: Sơ đồ trình tự xử lý khi khách lên tàu

* 1. Xuống Tàu



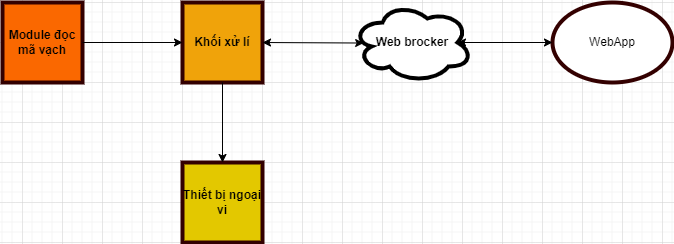
Hình 3: Sơ đồ trình tự xử lý khi khách xuống tàu

1. Scope Model



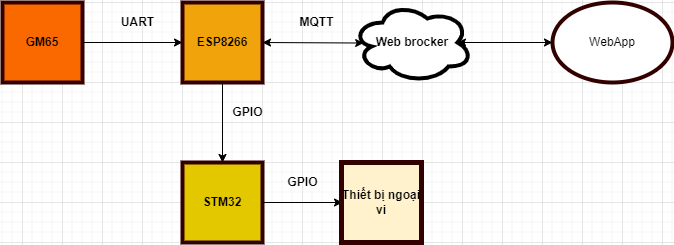
# Chương III. Đồng thiết kế cứng mềm

1. Sơ đồ khối hệ thống



Hình 4: Sơ đồ khối hệ thống soát vé

1. Lựa chọn giải pháp phần cứng



Hình 5: Lựa chọn giải pháp phần cứng cho từng khối

Chi tiết các linh kiện:

**Module đọc mã vạch GM65**: là module đọc mã vạch 2 chiều hiệu suất cao. Nó có thể đọc mã vạch 1D, 2D dễ dàng với tốc độ cao. Module cũng quét nhanh đối với các mã vạch tuyến tính, ngay cả các mã vạch trên nhãn dán, màn hình.

Thông số kĩ thuật:

|  |
| --- |
| Điện áp sử dụng: 5VDC |
| Điện áp giap tiếp: TTL 3.3~5VDC. |
| Current: 120mA during scanning / 30mA during standby |
| Interface: USB, UART (HID or VSP) |
| Optical system: CMOS |
| Capture light source: 617nm LED |
| Lighting source: 6500K LED |
| Reading angle:   * rotating 360 °   + Deflection ± 60 °   + Tilt ± 65° |
| Scanning angle: 34 ° (horizontal) 26 ° (vertical) |
| Minimum contrast: 30% |
| Resolution: ≥ 0. 1mm (4mil) |
| Ambient light: Ambient light: 0~86,000 lux |

Bảng 1: Thông số kỹ thuật GM65

**STM32f103**: là dòng chip 32bit của hãng STMicrochip sử dụng công nghệ lõi ARM Cortex mạnh mẽ, hiệu năng tốt nhưng vẫn giữ được giá thành rẻ. Phù hợp với đa số các công ty hiện nay

Các lý do nên chọn STM32f103 đó là:

* Tốc độ xử lý cao, ngoại vi hỗ trợ rất nhiều, dòng chip phân khúc thâp là STM32F0x cũng có thể hoạt động lên tới 48Mhz, 64kB Flash, 16kB RAM, 8 bộ Timer 16 bit, 1 bộ Timer 32 bit, 10 bộ ADC 12 bit, 8 bộ USART, 2 bộ SPI, 2 bộ I2C.
* Giá thành rẻ nhưng hiệu quả đem lại lớn.
* Học lập trình STM32 rất dễ dàng do cộng đồng hỗ trợ nhiều.
* Công cụ lập trình đều Free và đầy đủ tài liệu hỗ trợ

Thông số kĩ thuật:

|  |
| --- |
| Vi điều khiển chính: STM32F103RCT6 ARM Cortex-M3 |
| Nguồn sử dụng: 5VDC từ cổng Mini USB hoặc chân GPIO. |
| Tích hợp Led, Button. |
| Tích hợp cổng USB. |
| Tích hợp bộ nhớ Flash. |
| Tích hợp cổng nạp chuẩn Jtag. |
| Tích hợp khe cắm mạch RF NRF24L01+, UART, LCD,... |

Bảng 2: Thông số kỹ thuật STM32f103

**ESP8266 12f**: Mạch thu phát Wifi SoC ESP8266 ESP-12F Ai-Thinker có kích thước nhỏ gọn, ra chân đầy đủ của IC ESP8266, mạch được thiết kế và gia công chất lượng tốt với vỏ bọc kim loại chống nhiễu và anten Wifi PCB tích hợp cho khoảng các truyền xa và ổn định.

Thông số kĩ thuật:

|  |
| --- |
| Điện áp sử dụng: 3.0V~3.6V(Optimal 3.3V) |
| Working current: ≈70mA(170mA MAX), standby＜200uA |
| Deep sleep holding current 10 uA, shutdown current of less than 5 uA |
| 30 pins (10 GPIO, every GPIO can be PWM, I2C, 1-wire) |
| SRAM size: 36 KB |
| ROM size: 32Mbit (SPI External Flash) |
| Antena on PCB |
| Transmission data rate: 110-460800bps |
| 10 bit precision ADC pinout on board (0~1V) |
| WiFi @ 2.4 GHz, supports WPA / WPA2 security mode |
| Wi-Fi Connectivity (802.11 b/ g/ n) |
| Support UART/GPIO data communication interface |
| Support STA/AP/STA+AP 3 working modes |
| Built-in TCP/IP protocol stack, maximum 5 clients |
| Working temperature: -40℃～＋125℃ |
| Dimensions: lenght: 24mm, width: 16mm, height: 3mm |

Bảng 3: Thông số kỹ thuật Esp8266 12f

1. Tính toán thiết kế khối nguồn

Nguồn là 1 vấn đề quan trọng trong bất kỳ hệ thống nào. Trong hệ thống cần cung cấp nguồn cho module GM65, khối xử lý và thiết bị ngoại vi với các yêu cầu cụ thể là:

* Module GM65 và STM32: +5V
* ESP8266 12f: 3÷3.6V

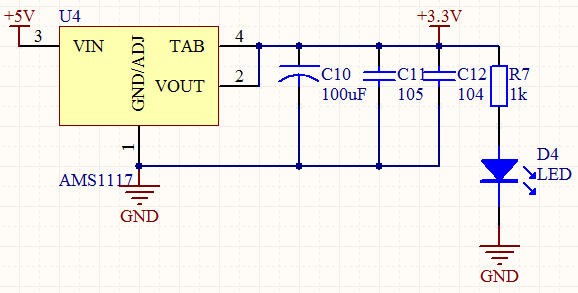
Sử dụng adapter đầu ra 5V 1000mA để cung cấp nguồn cho hệ thống.



Hình 6: Adapter 5V 1A

Tuy nhiên hệ thông sử dụng điện áp vào khác nhau tại mỗi module, do đó cần phải sử dụng mạch nguồn 3.3V để tạo ra điện áp phù hợp cho ESP8266 12f hoạt động.

Ở đây, AMS là dòng IC nguồn ổn áp tuyến tính hiệu suất cao của *Advanced Monolithic Systems, Inc*. Với các ưu điểm như nhiễu thấp, dòng rò nhỏ, tích hợp mạch chống quá tải... rất thích hợp cho nhiều ứng dụng như các mạch Op-amp, sạc pin hay MCU. Với những ưu điểm như vậy, ở đây AMS1117 được sử dụng cho mạch nguồn +3.3V.



Hình 7: Khối nguồn nuôi Esp8266 12f

|  |  |
| --- | --- |
| **Thông số** | **Giá trị** |
| IC | AMS1117 |
| Điện áp đầu vào | +5V |
| Điện áp đầu ra | +3.3V |
| Dòng | 1A |
| Dropout Voltage | 1V |

Bảng 4: Thông số khối nguồn nuôi Esp8266 12f

1. Lựa chọn giải pháp phần mềm
   1. Hệ Điều Hành:

Dựa trên nền tảng STM, có năm phương pháp truyền để lựa chọn nhằm đáp ứng các yêu cầu điều khiển thời gian thực của hệ điều hành. Chúng là μClinux, μC / OS-II, eCos, FreeRTOS và Hệ điều hành Dujiangyan (djyos).

* μClinux: Đặc điểm lớn nhất của μClinux là nó được thiết kế cho các bộ vi xử lý không có MMU. Điều này áp dụng cho stm32f103 không có chức năng MMU, nhưng hệ thống này yêu cầu ít nhất 512KB dung lượng RAM, 1MB dung lượng ROM / FLASH và stmf103 có 256K. FLASH yêu cầu bộ nhớ ngoài, điều này làm tăng chi phí thiết kế phần cứng. μClinux có cấu trúc phức tạp, khó cấy ghép và nhân lớn. Hiệu suất thời gian thực của nó cũng rất kém. Nếu trọng tâm phát triển của các sản phẩm nhúng là hệ thống tệp và các ứng dụng mạng, μClinux là một lựa chọn tốt
* μC / OS-II: Nó là một nhân hệ điều hành nhúng với cấu trúc đơn giản, đầy đủ các chức năng và hiệu suất thời gian thực mạnh mẽ. Rất thích hợp cho các CPU không có chức năng MMU. Nó yêu cầu ít không gian nhân và không gian lưu trữ dữ liệu, hiệu suất thời gian thực tốt, khả năng mở rộng tốt và mã nguồn mở. Có rất nhiều tài liệu và ví dụ trên Internet, vì vậy chuyển sang sử dụng CPU stm32f103 sẽ thích hợp hơn.
* ECos: eCos được đặc trưng bởi cấu hình linh hoạt, hỗ trợ di chuyển CPU, không yêu cầu MMU, là mã nguồn mở, có tính di động tốt và phù hợp hơn để di chuyển sang CPU trên nền tảng stm32. Nhưng ứng dụng của eCOS không rộng rãi, không rộng rãi như μC / OS-II và dữ liệu cũng không rộng rãi như μC/OS-II. eCos phù hợp với một số hệ thống nhúng công nghiệp hoặc thương mại nhạy cảm về chi phí, chẳng hạn như một số ứng dụng trong lĩnh vực điện tử tiêu dùng..
* FreeRTOS: Là một hệ điều hành nhẹ, FreeRTOS cung cấp các chức năng như quản lý tác vụ, quản lý thời gian, semaphores, hàng đợi tin nhắn, quản lý bộ nhớ và chức năng ghi nhật ký để đáp ứng nhu cầu của các hệ thống nhỏ hơn. Hạt nhân FreeRTOS hỗ trợ thuật toán lập lịch ưu tiên. Mỗi nhiệm vụ có thể được ưu tiên nhất định theo mức độ quan trọng. CPU luôn chạy tác vụ với mức ưu tiên cao nhất ở trạng thái sẵn sàng. Nhân FreeRT0S hỗ trợ thuật toán lập lịch quay cùng một lúc. Hệ thống cho phép các tác vụ khác nhau sử dụng cùng một mức độ ưu tiên. Khi không có tác vụ ưu tiên cao hơn nào sẵn sàng, các tác vụ ưu tiên giống nhau sẽ chia sẻ thời gian sử dụng CPU.
* hệ điều hành Dujiangyan (djyos): Hệ điều hành Dujiangyan, được gọi là djyos, được đặt theo tên của một dự án bảo tồn nước vĩ đại Dujiangyan. Không giống như các hệ điều hành truyền thống, djyos không phải là một luồng mà là một lõi lập lịch dựa trên sự kiện. Thuật toán lập lịch trình này giải phóng lập trình viên khỏi lối suy nghĩ về việc mô phỏng quá trình thực thi của máy tính mà viết ứng dụng theo cách của thế giới nhận thức của con người. Như VC được giới thiệu trong lập trình nhúng. Thuật toán lập lịch của Djyos cho phép các lập trình viên thoát khỏi những ràng buộc của các luồng và quy trình. DJyos không có API apis và một lập trình viên không biết kiến thức về luồng có thể viết ứng dụng dưới djyos một cách trôi chảy.

Qua các giới thiệu khái quát trên, có thể đưa ra lựa chọn ứng dụng hệ điều hành FreeRtos vào dự án do hệ điều hành FreeRTOS là hệ điều hành hoàn toàn miễn phí với mã nguồn mở, khả năng di động, khả năng mở rộng và chiến lược lập lịch linh hoạt, có thể dễ dàng chuyển sang các bộ vi điều khiển khác nhau.

* 1. Web Client:

MQTT là một giao thức truyền thông điệp nhỏ gọn dựa trên mô hình publish/subcsribe. Nó sử dụng băng thông thấp, độ tin cậy cao và có thể hoạt động trong điều kiện truyền không ổn định. MQTT được thiết kế cho các mạng SCADA và các tình huống sử dụng băng thông thấp và gần đây nó đã được chứng minh là rất mạnh mẽ trong việc hỗ trợ các dự án IoT. Trong dự án này, giao thức MQTT sẽ được sử dụng để trao đổi dữ liệu giữa hệ thống bán vé và khách hàng mạng tập trung dữ liệu.

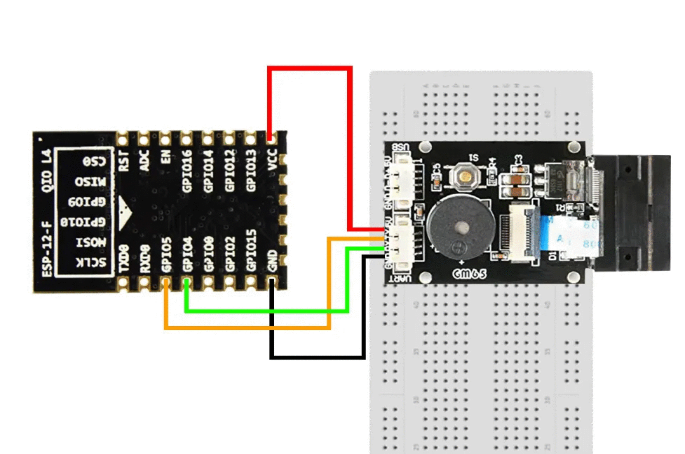
Ưu điểm của MQTT:

* Chuyển thông tin hiệu quả hơn
* Tăng khả năng mở rộng
* Giảm đáng kể tiêu thụ băng thông mạng
* Giảm tốc độ cập nhật xuống giây
* Rất phù hợp cho điều khiển và do thám
* Tối đa hóa băng thông có sẵn
* Chi phí cực nhẹ
* Rất an toàn với bảo mật dựa trên sự cho phép
* Được sử dụng bởi ngành công nghiệp dầu khí, Amazon, Facebook và các doanh nghiệp lớn khác
* Tiết kiệm thời gian phát triển
* Giao thức publish/subscribe thu thập nhiều dữ liệu hơn với ít băng thông hơn so với giao thức cũ.

# Chương IV. Thiết kế giao tiếp ngoại vi

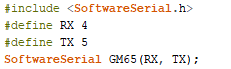
1. ESP8266 12f với module đọc mã vạch

Sơ đồ kết nối chân của GM65 và esp8266 12f

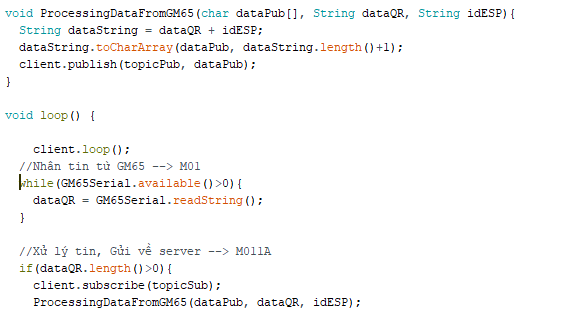


Hình 8:Sơ đồ kết nối chân của GM65 và esp8266 12f

Việc kết nối GM65 với ESP8266 12f ta sử dụng giao tiếp UART. Giao tiếp UART chỉ cần 2 dây để truyền dữ liệu. Do đó, ta kết nối chân Tx và Rx của GM65 với GPIO5 và GPIO4 trên ESP8266. Ta triển khai 2 chân GPIO5 và GPIO4 trong phần mềm như sau:



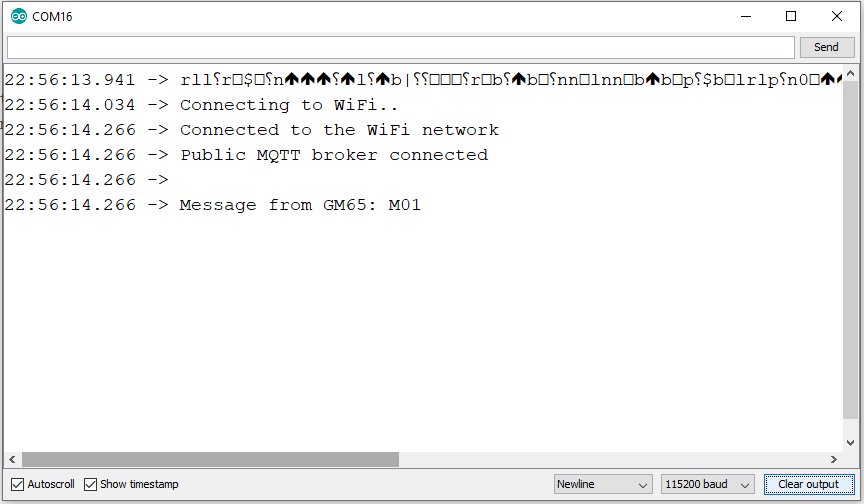
Tại đây, ta sẽ truyền dữ liệu và đọc dữ liệu để giao tiếp với GM65.



Hình 9: Xử lý truyền dữ liệu và đọc dữ liệu để giao tiếp với GM65

Sau khi nhận được tin “M01” từ GM65, ESP8266 xử lý tin rồi gửi về server mã có nội dung “M011A”.

Ta sẽ nhận được kết quả như dưới hình:



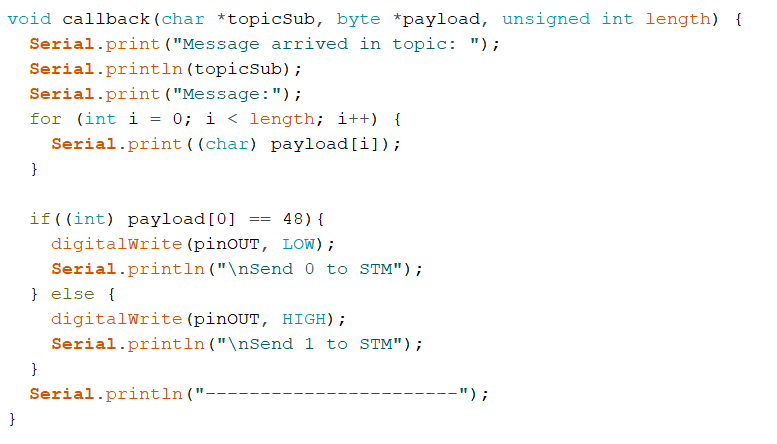
Hình 10: Kết quả

1. STM32 với ESP8266 12f

Sau khi ESP8266 nhận được thông tin đã qua sử lý từ server, ESP8266 tiếp tục đẩy thông tin về STM32 để tiến hành mở cửa (hoặc đóng cửa do khách hàng không đủ tiền)

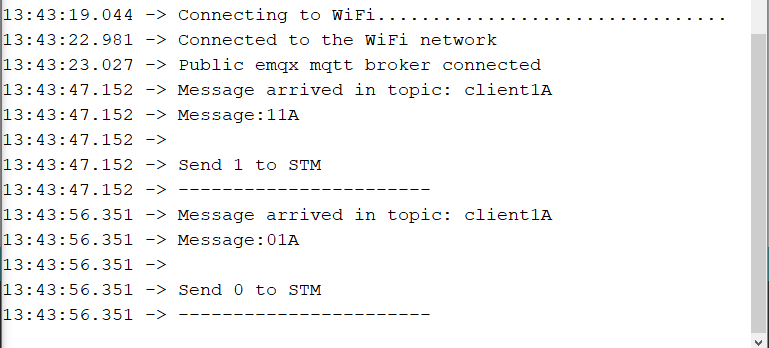
Việc kết nối STM32 với ESP8266 12f ta sử dụng GPIO. Giao tiếp này chỉ cần một dây để truyền dữ liệu là 0 và 1. Do nhu cầu truyền dữ liệu đơn giản là bit 0 và 1 để quyết định việc đóng hay mở cửa nên giao tiếp GPIO là lựa chọn tối ưu. Ta sẽ để đầu ra của ESP là cổng GPIO 1 và đầu vào của STM32 là cổng A4.

Sau khi nhận thông tin từ server ví dụ như 11A bit, với đầu tiên là ‘1’ có nghĩa là cho phép mở cửa để khách qua (ngược lại nếu bit đầu tiên là 0 thì đóng cửa không để khách qua), 2 bit tiếp theo là ‘1A’ là mã của máy soát vé; ta tiến hành tách chuỗi để lấy ra bit đầu tiên, gửi về STM32 với dòng hàm callback như sau:

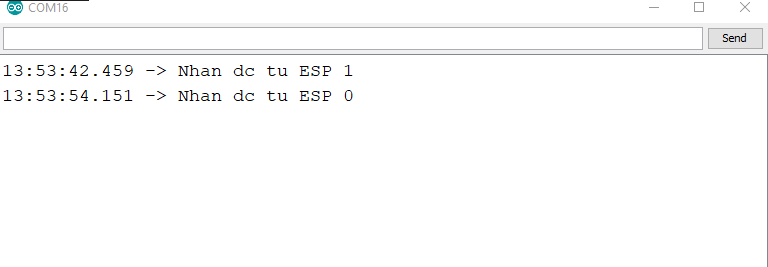


Hình 11: Giao tiếp STM32 với ESP8266 12f

Và ta được kết quả như hình dưới:



Hình 12: Gửi mã tới Stm32f103



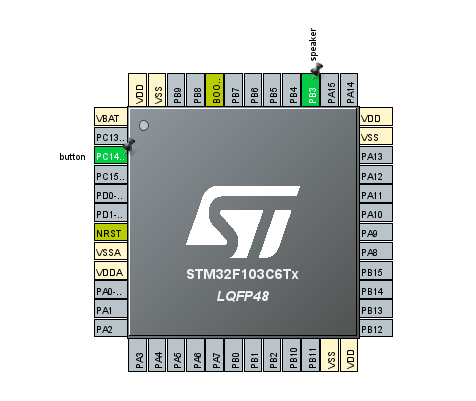
Hình 13: Mã nhận được từ esp

1. STM32 với các thiết bị ngoại vi khác
   1. STM32 kết nối chuông báo

a. Cấu hình cho stm32f103c6 (Firmware version V1.8.3) trên STM32CUBEIDE

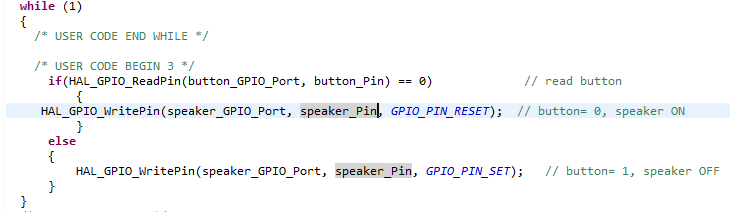
+) Input: PC14 (GPIO Pull-up)

+) Output: PB3 (GPIO output level: High, Output push pull, Pull-up, Speed: High)

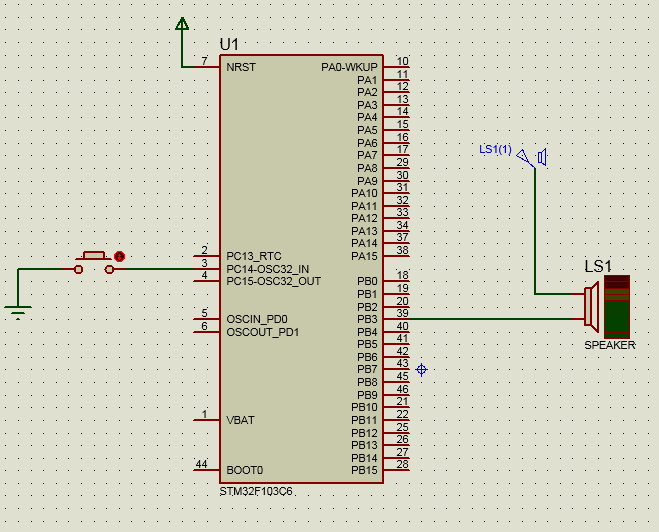


Hình 14:Cấu hình cho stm32f103c6 trên STM32CUBEIDE

b. Generate code và lập trình.

c. Chạy mô phỏng trên Proteus.

+) Speaker khởi chạy Audio.



Hình 15: Kết quả mô phỏng

* 1. Kết nối song song với LCD 16x02

Ngày nay, thiết bị hiển thị [LCD 1602](http://www.suachualaptop24h.com/Linh-kien-laptop.html) (Liquid Crystal Display) được sử dụng trong rất nhiều các ứng dụng của VĐK. LCD 1602 có rất nhiều ưu điểm so với các dạng hiển thị khác như: khả năng hiển thị kí tự đa dạng (chữ, số, kí tự đồ họa); dễ dàng đưa vào mạch ứng dụng theo nhiều giao thức giao tiếp khác nhau, tiêu tốn rất ít tài nguyên hệ thống, giá thành rẻ,…

Thông số kĩ thuật của sản phẩm LCD 1602:

- Điện áp MAX : 7V

- Điện áp MIN : - 0,3V

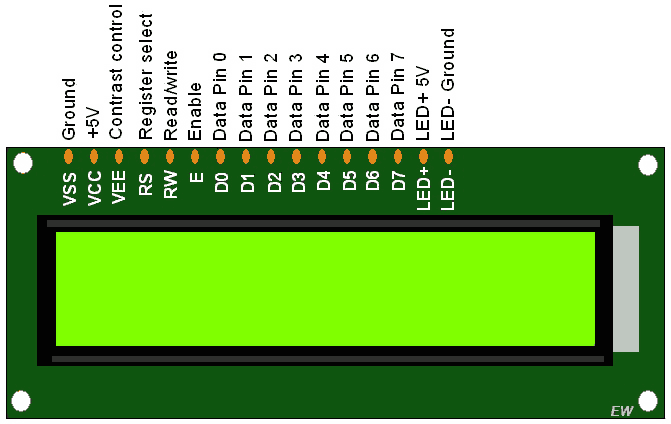
- Hoạt động ổn định : 2.7-5.5V

- Điện áp ra mức cao : > 2.4

- Điện áp ra mức thấp : <0.4V

- Dòng điện cấp nguồn : 350uA - 600uA

- Nhiệt độ hoạt động : - 30 - 75 độ C



Hình 16:LCD 1602 xanh lá

Chức năng của từng chân LCD 1602:

- Chân số 1 - VSS : chân nối đất cho LCD được nối với GND của mạch điều khiển

- Chân số 2 - VDD : chân cấp nguồn cho LCD, được nối với VCC=5V của mạch điều khiển

- Chân số 3 - VE : điều chỉnh độ tương phản của LCD

- Chân số 4 - RS : chân chọn thanh ghi, được nối với logic "0" hoặc logic "1":

+ Logic “0”: Bus DB0 - DB7 sẽ nối với thanh ghi lệnh IR của LCD (ở chế độ “ghi” - write) hoặc nối với bộ đếm địa chỉ của LCD (ở chế độ “đọc” - read)

+ Logic “1”: Bus DB0 - DB7 sẽ nối với thanh ghi dữ liệu DR bên trong LCD

- Chân số 5 - R/W : chân chọn chế độ đọc/ghi (Read/Write), được nối với logic “0” để ghi hoặc nối với logic “1” đọc

- Chân số 6 - E : chân cho phép (Enable). Sau khi các tín hiệu được đặt lên bus DB0-DB7, các lệnh chỉ được chấp nhận khi có 1 xung cho phép của chân này như sau:

+ Ở chế độ ghi: Dữ liệu ở bus sẽ được LCD chuyển vào thanh ghi bên trong khi phát hiện một xung (high-to-low transition) của tín hiệu chân E

+ Ở chế độ đọc: Dữ liệu sẽ được LCD xuất ra DB0-DB7 khi phát hiện cạnh lên (low-to-high transition) ở chân E và được LCD giữ ở bus đến khi nào chân E xuống mức thấp



Hình 17: LCD 1602 Xanh dương 5v

- Chân số 7 đến 14 - D0 đến D7: 8 đường của bus dữ liệu dùng để trao đổi thông tin với MPU. Có 2 chế độ sử dụng 8 đường bus này là: Chế độ 8 bit (dữ liệu được truyền trên cả 8 đường, với bit MSB là bit DB7) và Chế độ 4 bit (dữ liệu được truyền trên 4 đường từ DB4 tới DB7, bit MSB là DB7)

- Chân số 15 - A : nguồn dương cho đèn nền

- Chân số 16 - K : nguồn âm cho đèn nền

Trên đây là một số thông tin về thông số kĩ thuật của LCD 1602 mà bạn có thể tìm hiểu. Hãy tham khảo ngay để có thể sử dụng thiết bị linh kiện này một cách đúng đắn nhất nhé.

* 1. Cấu hình trên STM32CUBEIDE
  + Sử dụng 5 chân GPIO\_Output giao tiếp với LCD:

+) Nhóm chân dữ liệu cao của LCD:

+) PB10-D7

+) PB11-D6

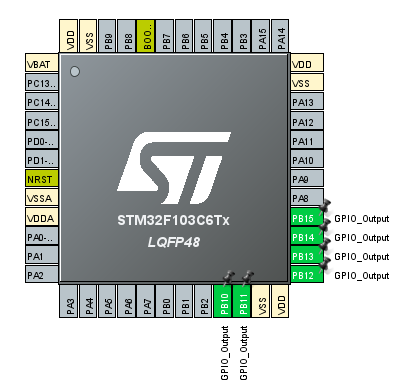
+) PB12-D5

+) PB13-D4

+) Nhóm chân điều khiển của LCD (R/W nối đất):

+) PB14-E

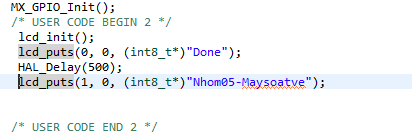
+) PB15-RS



Hình 18: Cấu hình stm32 với LCD

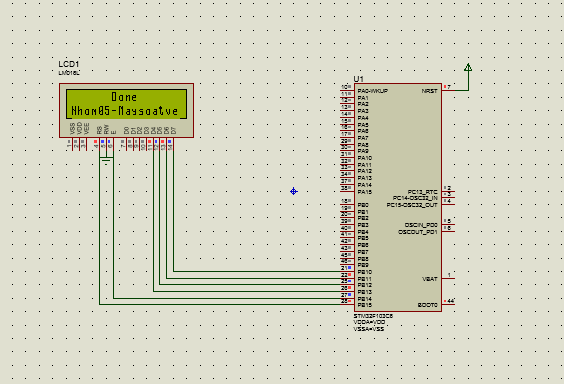
b. Generate code và lập trình.

* + Đây là phần code hiển thị trên LCD (Phần code define các chân sẽ có ở Phụ lục)



C. Chạy mô phỏng trên Proteus.

* + Sử dụng LCD LM016L



Hình 19: Kết quả mô phỏng

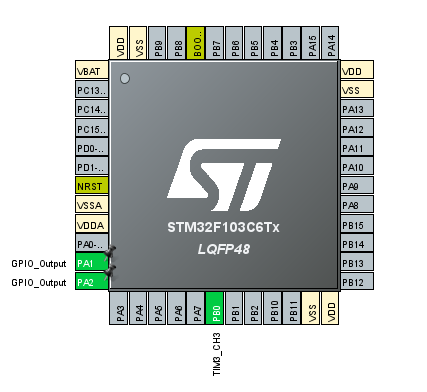
* 1. STM32 kết nối khóa barie thông qua Driver Motor Shield L293D (Barie được coi là DC Motor EL2030)
* Driver Motor Shield L293D là một module mở rộng chuyên dụng cho các ứng dụng điều khiển động cơ, robot. Driver Motor Shield L293D có nhiều chức năng, dễ vận hành và có thư viện hỗ trợ. Đây là một driver phổ biến cho điều khiển động cơ DC, sử dụng IC điều khiển động cơ DC L293D. Các chân tương thích với Arduino, do đó dễ dàng sử dụng để lập trình cho các dự án Arduino.

**Thông số Driver Motor:**

* Điện áp đầu vào: 4.5V đến 36V.
* Tương thích với các board Arduino Uno R3, Arduino Leonardo R3 và Arduino Mega 2560.
* Có thể điều khiển động cơ DC (4 động cơ), động cơ servo (2 động cơ) và động cơ bước (2 động cơ). 2 cổng điều khiển servo motor được đánh dấu: Servo\_1 và Servo\_2 trên linh kiện. Các cổng điều khiển động cơ DC được đánh dấu lần lượt là M1, M2, M, M4, chân giữa là chân GND.
* 2 cổng điều khiển động cơ servo có điện áp vào 5V với timer có độ phân giải cao, phù hợp cho các ứng dụng điều khiển bằng Arduino có độ chính xác cao. Đặc biệt không có jitter.
* Có 2 IC Driver L293D, do đó sẽ có 4 cầu H để điều khiển được 4 động cơ DC. Mỗi cầu H có dòng ra tối đa 0.6A (dòng chịu đựng cực đại là 1.2A) ở mỗi kênh điều khiển.
* Các cổng M dùng điều khiển động cơ DC được điều khiển bằng tín hiệu PWM.
* Driver còn hỗ trợ điều khiển 2 động cơ bước, với 2 cổng dùng cho 2 động cơ servo có thể được dùng cho động cơ bước. Với Shield L293D, động cơ bước có thể vận hành ở tất cả các chế độ: full step, half step và micro-step. Động cơ bước dùng cho driver có thể là loại đơn cực (unipolar) hoặc lưỡng cực (bipolar).
* Có sẵn nút RESET để khởi động.
  1. Cấu hình trên STM32CUBEIDE

+) PA1 và PA2 là 2 chân GPIO\_Output giao tiếp với L293D

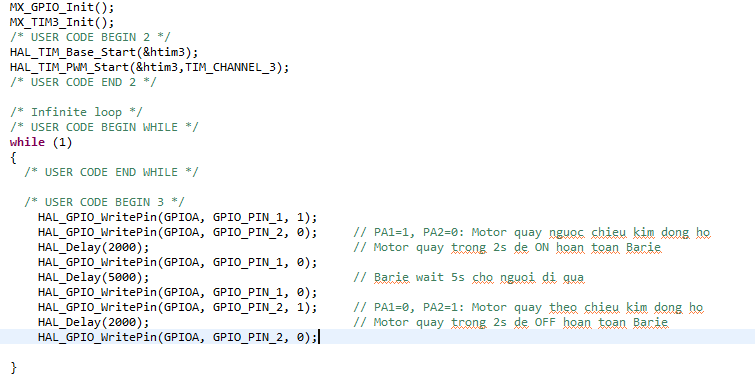
+) PB0 được cấu hình là TIMER Tim3\_CH3 (Internal Clock; Channel3- PWM Generation CH3: Prescale: 19, Counter Period: 1000, Pulse: 500)



Hình 20: Cấu hình stm32 điều khiển motor

b. Generate code và lập trình.

- Phần code điều khiển Motor có chú thích trong hình.



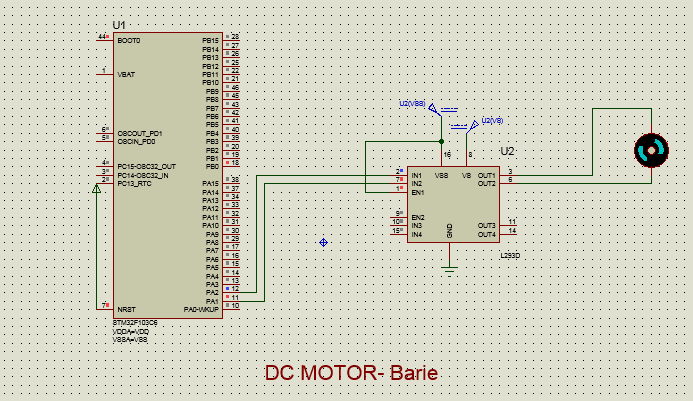
C. Chạy mô phỏng trên Proteus.

* + Driver Motor L293D:

+) Chân OUT1, OUT2 của L293D kết nối với Motor EL2030.

+) VSS được cấp nguồn DC 5 V, và được nối chung với EN1.

+) VS được cấp nguồn DC 12V.

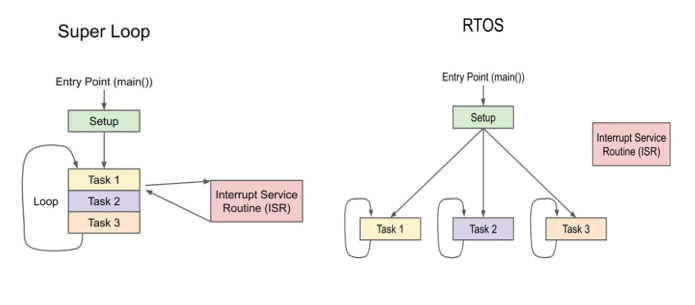


Hình 21: Kết quả mô phỏng

# Chương V. Tổng hợp, cấu hình, biên dịch hệ điều hành

1. Tổng quan về FreeRtos

* Khái niệm cơ bản về Rtos: RTOS (Real-Time operating system) hay được gọi là hệ điều hành thời gian thực mà cho phép ứng dụng có thể chạy đa tác vụ và có thể đáp ứng được “deadline” theo thời gian thực. Lưu ý rằng việc đáp ứng được “deadline” không nhất nhiết có nghĩa là phải nhanh mà ở đây là mang tính “đúng thời điểm” và chính xác.
* Ứng dụng của RTOS:
  + Chạy các dự án lớn đòi hỏi xử lý nhiều công việc nhưng vẫn phải đáp ứng được về mặt thời gian.
  + Các ứng dụng về viễn thông và IOT, các thiết bị liên quan đến y tế…
* Ưu điểm của RTOS:
  + Giúp chương trình của bạn dễ quản lý và phát triển vì nó giúp phân “chia” 1 vấn đề phức tạp thành các phần nhỏ hơn “để trị”.
  + Tăng tính linh động và dễ bảo trì
  + Dễ dàng hơn trong việc chia sẻ tài nguyên của CPU.
  + Tránh được siêu vòng lặp.
  + Thuận lợi trong làm việc nhóm.

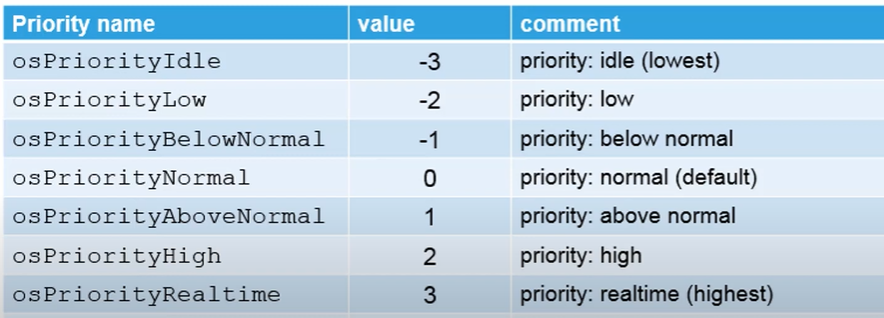


Hình 22: So sánh giữa đa tác vụ và siêu vòng lặp

* Các khái niệm trong hệ điều hành thời gian thực Rtos
  + Kernel: có nhiệm vụ quản lý và điều phối các Task.
  + Task State: có các thông số running, ready, waiting, inactive.
  + Scheduler: là 1 thành phần của kernel quyết định task nào được thực thi.
  + Inter-task & Resource Sharing: Các task cần phải kết nối và trao đổi dữ liệu với nhau để có thể chia sẻ tài nguyên.
  + Signal Event: Signal event được dùng để đồng bộ các task.
  + Message queue: Message queue là cơ chế cho phép các task có thể kết nối với nhau.
  + Semaphore: Được sử dụng để đồng bộ task với các sự kiện khác trong hệ thống.
  + Mutex: Sử dụng cho việc loại trừ (mutial exclution), hoạt động như là một token để bảo vệ tài nguyên được chia sẻ.

1. Áp dụng FreeRtos vào dự án
2. Xác định mức độ ưu tiên giữa các nhiệm vụ

Đặt task có nhiệm vụ nhận mã từ esp ở mức cao nhất và các task có nhiệm vụ thực hiện đóng/ mở barie, chuông báo, màn hình hiển thị sẽ ở mức ưu tiên thấp hơn.



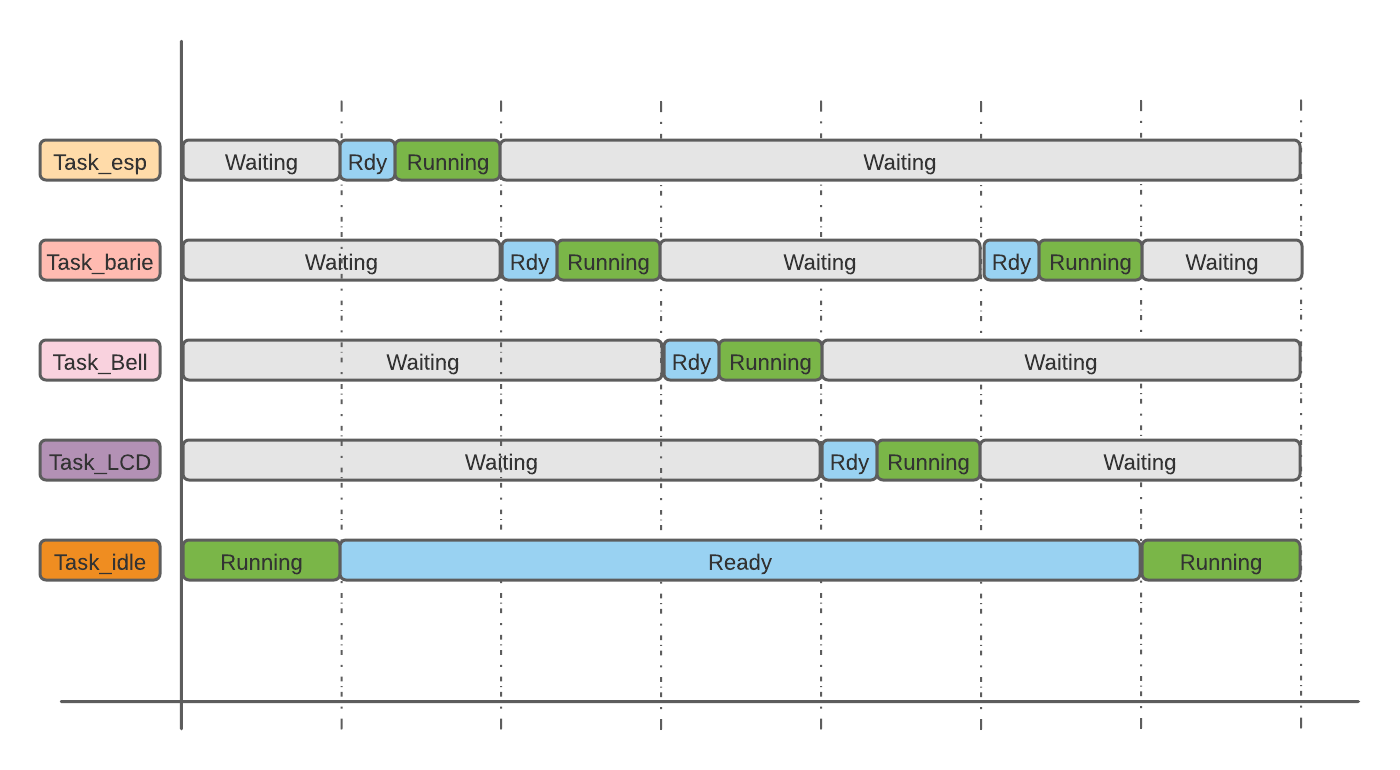
Hình 23: Các mức độ ưu tiên trong phiên bản cmsis-v1

* Task giao tiếp giữa stm32 với esp8266v1 sẽ nhận mức ưu tiên cao nhất do tính quan trọng của nhiệm vụ (mức hightest).
* Các task mở/đóng barie, giao tiếp với LCD, chuông báo ở mức ưu tiên thấp hơn (mức high).
* Lựa chọn thuật toán cho lập lịch

Có 3 kỹ thuật được áp dụng là:

* **Round Robin Scheduling**: Các task sẽ được chia 1 khe thời gian cố định, nếu chưa thực thi xong sẽ tạm dừng và đợi đến lượt.
* **Co-operative scheduling**: Các task được thực thi tới khi kết thúc, các task tiếp theo mới được thực thi.
* **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.

Có thể thấy, với các nhiệm vụ mà hệ thống nhúng cần thực hiện, kỹ thuật priority base phù hợp hơn cả do các nhiệm vụ có mức độ ưu tiên khác nhau.

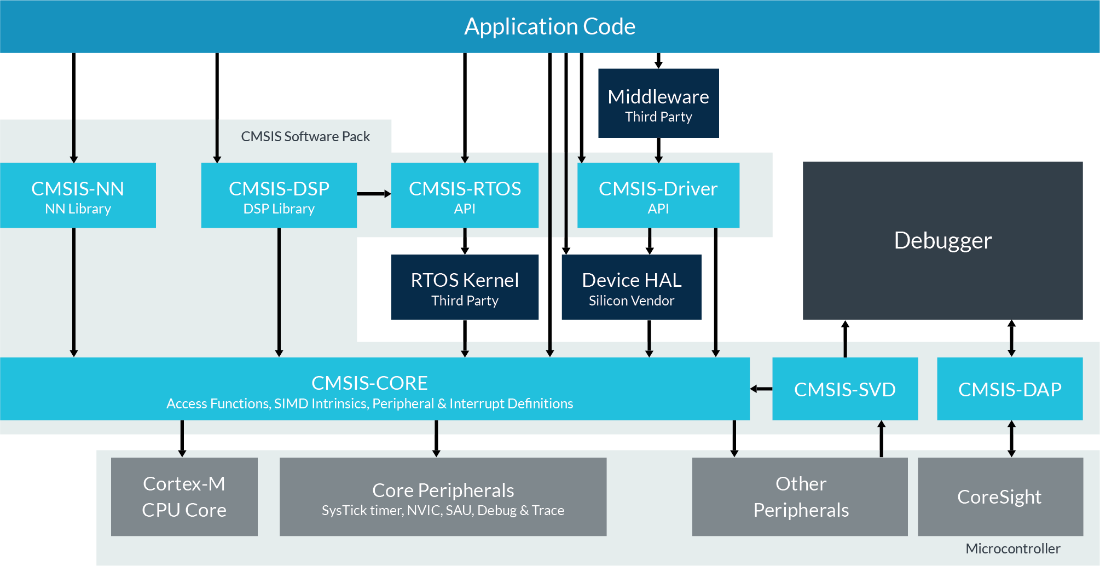


Hình 24: Lịch trình của hệ thống

* CMSIS-RTOS

FreeRTOS là một khung phần mềm cơ bản cho phép chuyển đổi tác vụ, lập lịch,... nhưng chúng ta sẽ không thực hiện các cuộc gọi trực tiếp đến FreeRTOS.

ARM đã tạo thư viện CMSIS-RTOS, cho phép chúng ta thực hiện các cuộc gọi đến RTOS bên dưới, do đó cải thiện tính di động của mã giữa các bộ xử lý ARM khác nhau. Hình ảnh này mô tả cách các thư viện CMSIS của ARM tương tác với phần mềm của bên thứ ba:

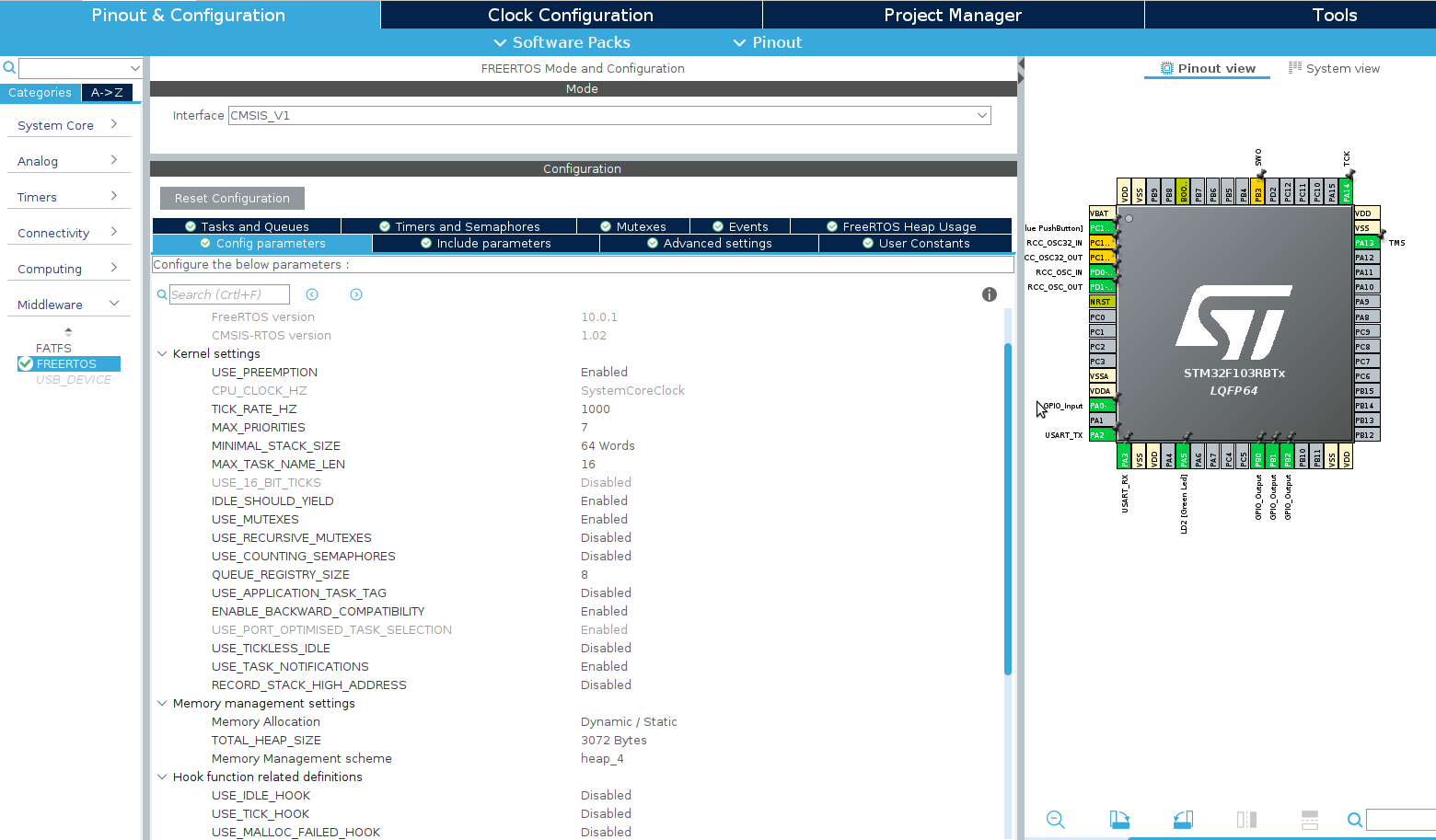


Hình 25: CMSIS

FreeRTOS là “RTOS Kernel” được mô tả trong sơ đồ. Chúng ta sẽ thực hiện các cuộc gọi tới CMSIS-RTOS (có 2 phiên bản là 1 và 2 ) để kiểm soát FreeRTOS.

* Sử dụng các công cụ CubeMX và CubeIDE để thiết lập FreeRtos
* Thiết lập CubeMX:

Sau khi chọn bộ điều khiển, CubeMX sẽ mở trang mặc định như hình dưới, chọn FreeRtos ở thanh menu và lựa chọn phiên bản CMSIS.

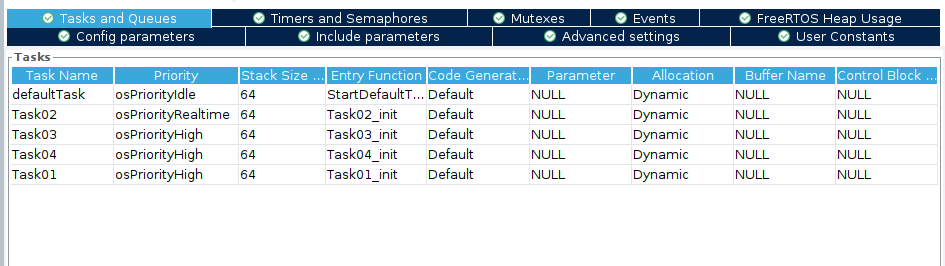


Hình 26: Thiết lập FreeRtos trong CubeMX

Lựa chọn phiên bản CMSIS\_V1 do nó được hỗ trợ bởi hầu hết các thiết bị STM32.

Trong phần kernel settings, cài đặt các tham số cho hệ điều hành (nếu cần thiết).

Tiếp theo chuyển tới tab "Task and Queues", thêm mới và thiết lập các thuộc tính cho các task.

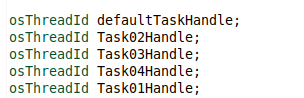


Hình 27: Thiết lập các tác vụ

* Tạo các task:

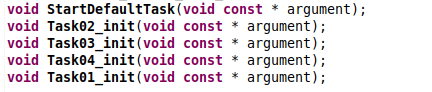
Trong giao diện CubeIDE, tiến hành lập trình, khai báo và cấu hình cho các Task.

Xác định một ThreadID cho mỗi nhiệm vụ. Biến này sẽ lưu trữ ID duy nhất của nhiệm vụ sau khi được tạo. Sau đó, tất cả các hoạt động sẽ yêu cầu ID này.



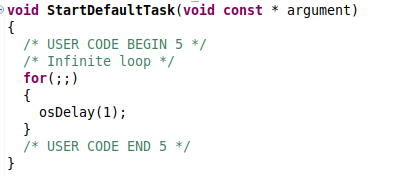
Hình 28: Khởi tạo ID cho các Task

Khởi tạo các hàm nhiệm vụ:



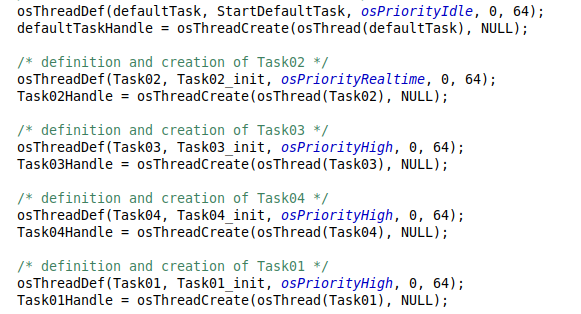
Hình 29: Khởi tạo các Task

Xây dựng các hàm cho nhiệm vụ. Các chức năng chính của nhiệm vụ sẽ được viết trong các hàm này.



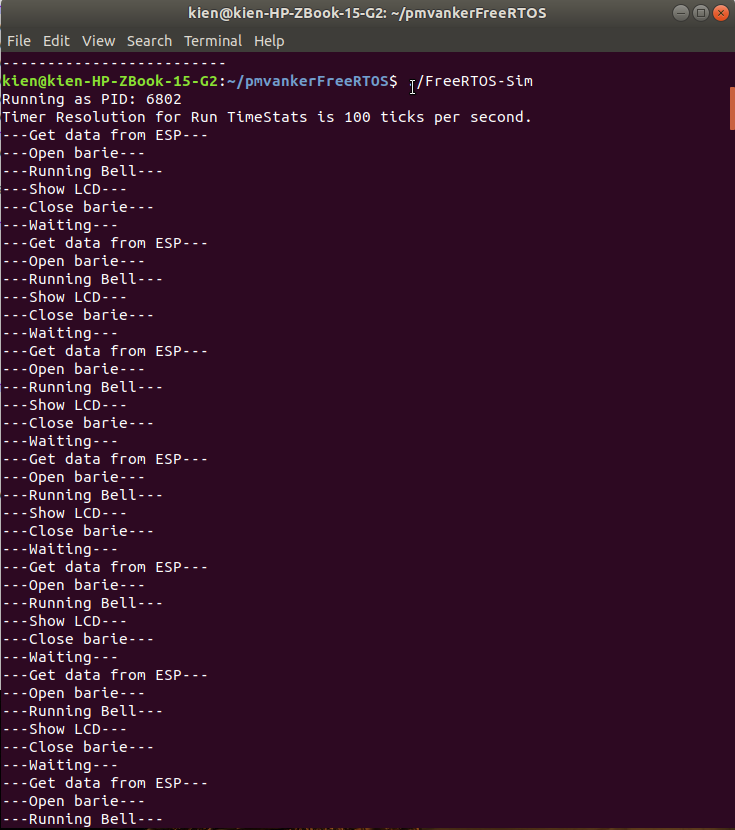
Hình 30: Xây dựng các hàm thực thi nhiệm vụ

Bên trong hàm "main.c", cần định nghĩa các luồng và tạo luồng thực thi cho các nhiệm vụ với các mức độ ưu tiên đã xác định trong phần "Lựa chọn thuật toán lập lịch".



Hình 31: Định nghĩa và tạo luồng cho các tác vụ

* osThreadDef có các tham số là tên tác vụ, hàm thực thi tác vụ, mức độ ưu tiên, đối tượng, kích thước ngăn xếp.
* Sau khi nhiệm vụ đã được xác đinh, khởi tạo bằng hàm osThreadCreate và gán ID tương ứng.
* Kết quả demo:



Hình 32: Kết quả

# Chương VI. Phát triển phần mềm

Việc phát triển phần mềm đảm bảo quá trình làm việc xuyên suốt của hệ thống. Các giá trị trao đổi giữa các module phải được thống nhất để thuận tiện cho việc lập trình cũng như làm việc nhóm.

Ta có thể quy định các tham số xuyên suốt dự án này như sau:

* Mã định danh thẻ (ID): Mxx - với xx là các số từ 00 tới 99 (giả sử có 100 thẻ)

VD: M01,M02,..M99.

Mã này sẽ được thu từ module quét mã vạch.

* Vị trí các máy quét (ADD): XY - với X là các số từ 1-9 và Y là các chữ cái in hoa

VD: thiết bị quét mã client1A sẽ có vị trí 1A, client9C sẽ có vị trí 9C.

* Tín hiệu gửi từ ESP8266 tới web-client : ID+ADD.

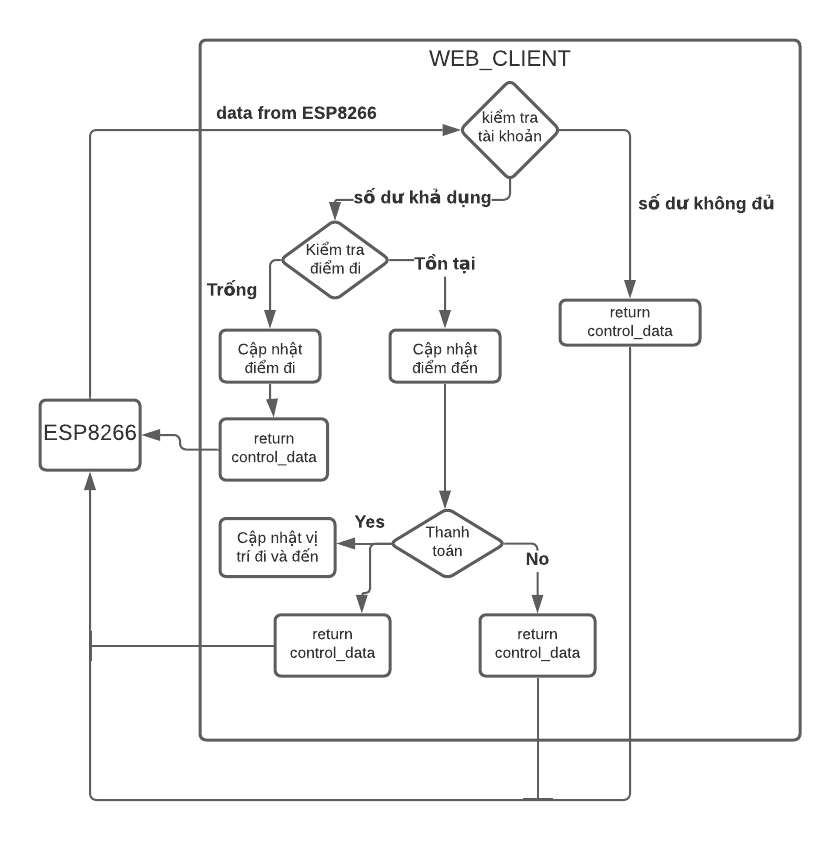
VD: M011A sẽ cho web-client biết phải xử lý thẻ có ID M01 và gửi trả tín hiệu phản hồi cho thiết bị quét mã vạch có vị trí 1A

* Tín hiệu phản hồi từ web-client: CS(control signal) sẽ có giá trị là 0 và 1. Tín hiệu có vai trò giúp vi điều khiển điều khiển chính xác hoạt động của các thiết bị ngoại vi.

1. Web-client mqtt

Với yêu cầu 1 vị trí để lưu trữ thông tin của các thẻ cũng như trạng thái của các thẻ vào từng thời điểm, dựa vào các thông tin đó để thực hiện các bước xử lý và tính toán đơn giản như tính chi phí đi lại.

* Yêu cầu chức năng đối với web:

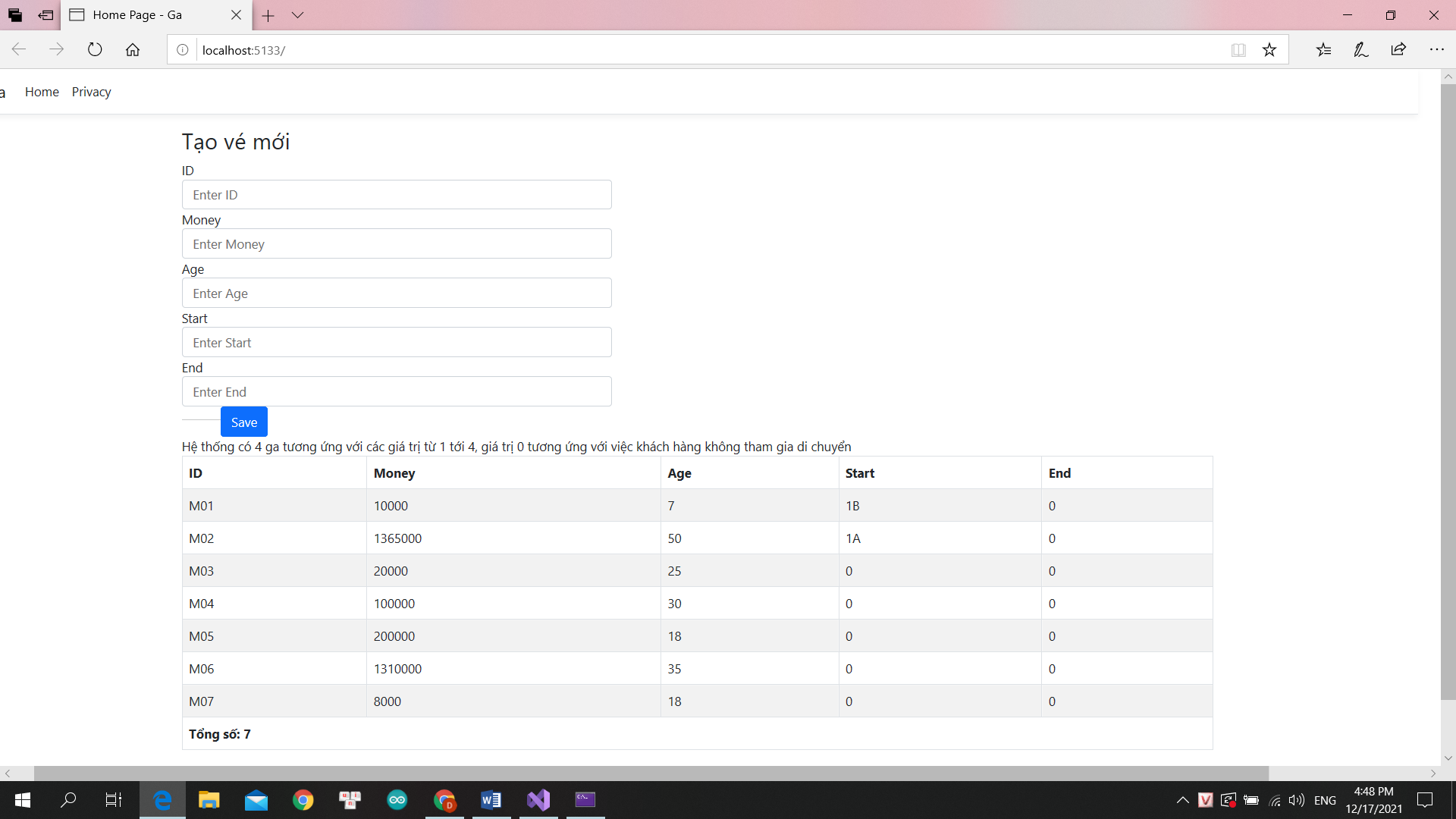


Hình 33: Sơ đồ mô tả các tác vụ của web-client

Với các yêu cầu như đã nêu các phần trước, có thể đưa ra một sơ đồ tổng quát mô tả lại quá trình hoạt động của đơn vị web-client.

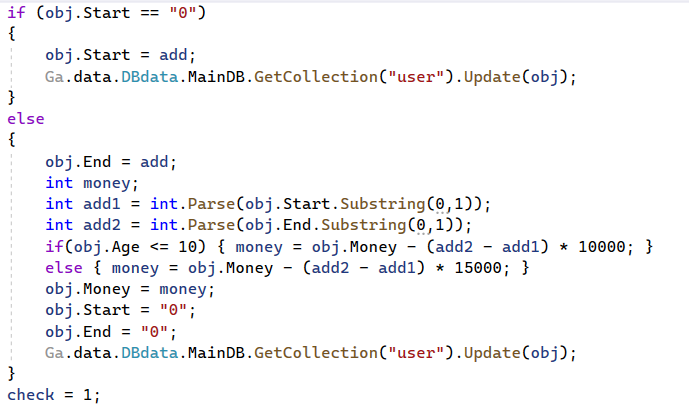
Về cơ bản, web-client cần phải đáp ứng được các nhiệm vụ cơ bản như: đọc mã được gửi từ các máy soát vé (ID+ADD), Kiểm tra số dư tài khoản thông qua ID, cập nhật vi trí xuất phát và kết thúc của hành khách, Thanh toán tiền theo lộ trình khách đã di chuyển, phản hồi tín hiệu điều khiển về các máy soát vé 1 cách chính xác.

* + Kết quả thu được:



Hình 34: Giao diện web-client

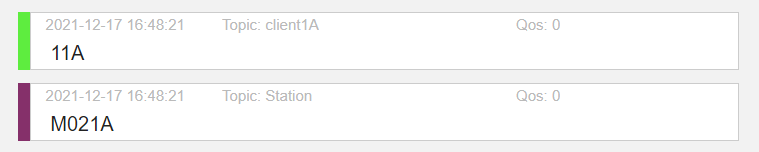
Việc xây dựng web-client sử dụng mô hình web api mvc với ngôn ngữ C#, web sẽ lưu và hiển thị các thông tin quan trọng về từ ID như số dư, tuổi chủ sở hữu, diểm bắt đầu và điểm kết thúc trong lộ trình. Các giá trị về Lộ trình của hành khách sẽ được để mặc định là 0, có nghĩa khách hàng không tham gia di chuyển nếu giá trị cột Start = 0 và khách hàng chưa hoàn thành thanh toán hoặc đang trong quá trình di chuyển nếu cột End = 0. Khi khách hàng thanh toán xong, các giá trị sẽ trở về 0 như hình dưới.

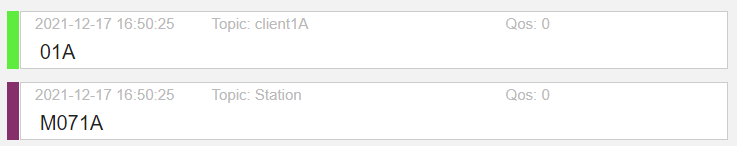


Hình 35: Tính toán chi phí hành trình

Với mô hình thanh toán đang áp dụng hiện nay tại Việt Nam là sử dung vé tuyến sẽ gây thua lỗ cho các công ty dịch vụ vận tải khách. Chính vì vậy, cần phải xác định chính xác quảng đường khách hàng đã di chuyển để tính toán chi phí cần chi trả.

Trong dự án này, tạm thời giả thuyết giá vé di chuyển giữa các ga cạnh nhau là bằng nhau. Do đó, giá vé bằng được tính bằng số chặng \* giá từng chặng. Giả sử vé cho trẻ em (<10 tuổi ) là 10000đ và người lớn (> 10 tuổi) là 15000đ.





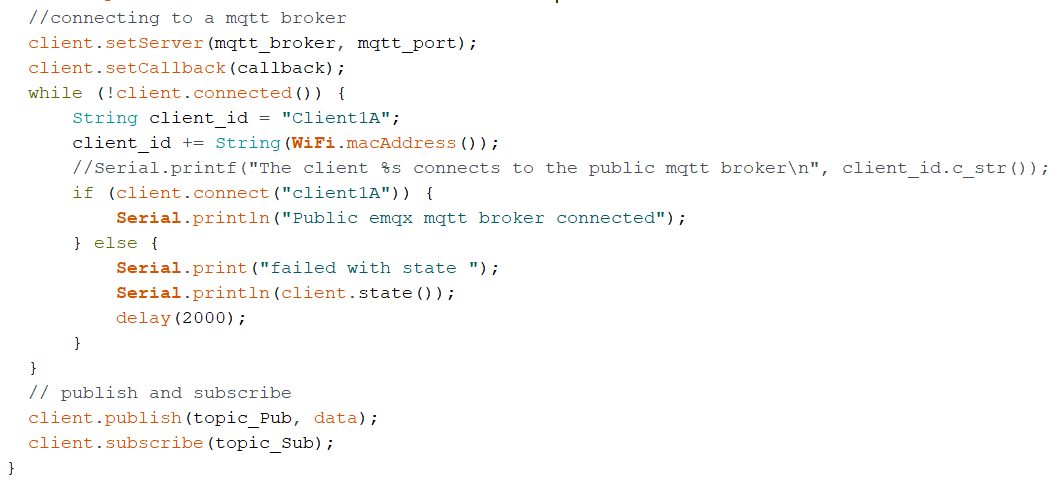
Hình 36: Thông tin trao đổi giữa máy soát vé và web-client

Trong hình trên, có thể thấy kết quả truyền tin giữa thiết bị soát vé (client1A) với web-client (Station). Khi thẻ có ID là M02 được quét, sẽ gửi mã M021A tới web-client. Mã này cho biết cần phải kiểm tra thẻ có ID là M02 và gửi trả tín hiệu về client1A, nếu thẻ hợp lệ trả CS là 1 về client1A. Tương tự mới thẻ có ID là M07 được quét tại client 1A, tuy nhiên, có thể thấy trong hình số x, số dư trong thẻ chỉ còn 8000đ, không đủ cho việc di chuyển giữa 1 chặng ( 10000đ với vé trẻ em). Do đó, tín hiệu trả về cho client1A là 0.

1. Esp-client mqtt

Với yêu cầu gửi tín hiệu cho web-client xử lý với giao thức MQTT đã được nêu ở trên, ta cần kết nối trạm soát vé với vai là một client với các chức năng:

Gửi tín hiệu cần xử lý lên web-client và nhận phản hồi từ nó.



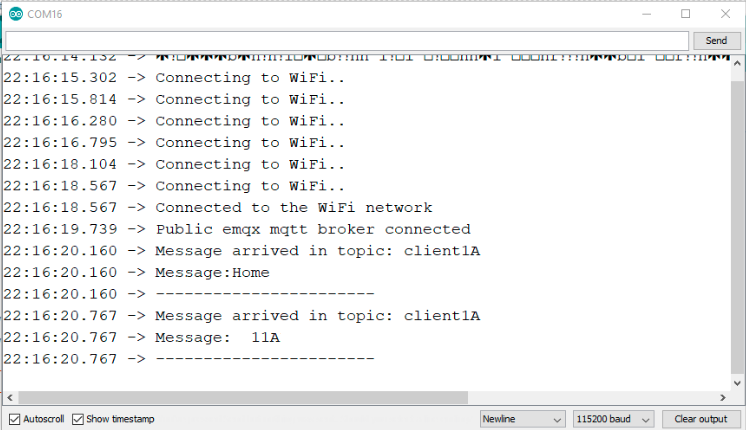
Hình 37: Gửi tín hiệu cần xử lý lên web-client

Ta kết nối với MQTT broker bằng câu lệnh client.setServer(mqtt\_broker, mqtt\_port) với mqtt\_broker là tên broker và mqtt\_port là cổng MQTT

Dùng vòng lặp while để kiểm tra kết nối giữa client với broker, sau khi kết nối thành công sẽ hiện thị thông báo đã kết nối thành công với câu lệnh Serial.println("Public emqx mqtt broker connected"), Nếu không thành công sẽ hiển thị trạng thái lỗi

Khi kết nối thành công, client sẽ gửi tín hiệu cần xử lý và nhận phản hồi theo 2 chủ đề đã được khai báo client.publish(topic\_Pub, data); và client.subscribe(topic\_Sub);

Ta sẽ nhận được kết quả như hình dưới:



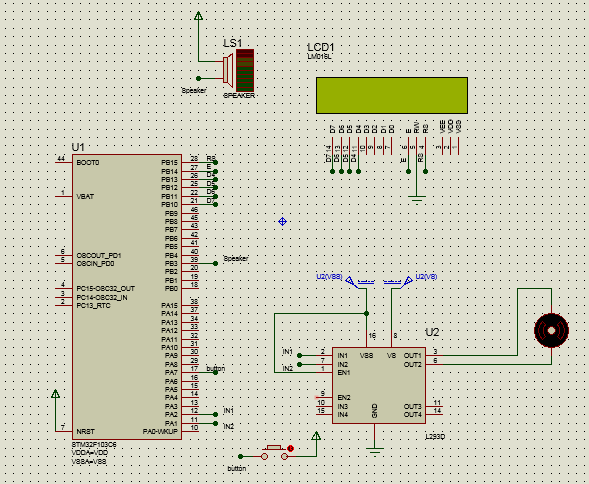
Hình 38: Kết quả trả về

Hình trên cho ta thấy client nhận được thông báo là Message: 11A và trùng khớp với tin nhắn trả về từ web-client đã nêu trong phần trên với thẻ có ID: M02.

Nhận thấy thời gian lúc truyền tín hiệu đi (22:16:20.16) và thời gian nhận được tín hiệu phản hồi (22:16:20.76) là vào khoảng 0.6s trong các điều kiện liên quan không mấy hoàn hảo.

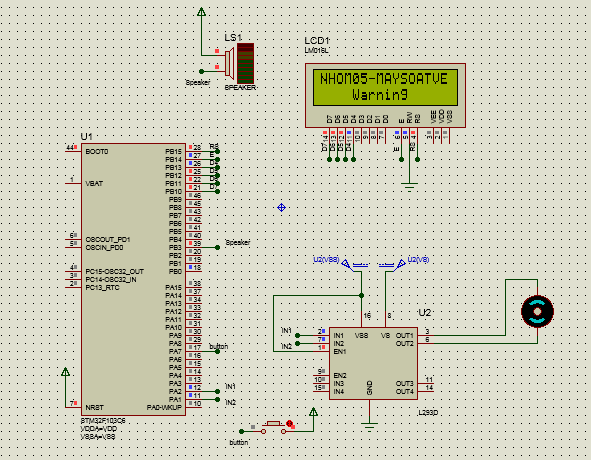
# Chương VII. Triển khai, mô phỏng hệ thống

Qua các bước đã thực hiện ở các phần trên, chúng em sử dụng công cụ mô phỏng mạch Proteus để tiến hành mô phỏng lại hoạt động của hệ thống. Ta có sơ đồ mô phỏng như hình dưới:

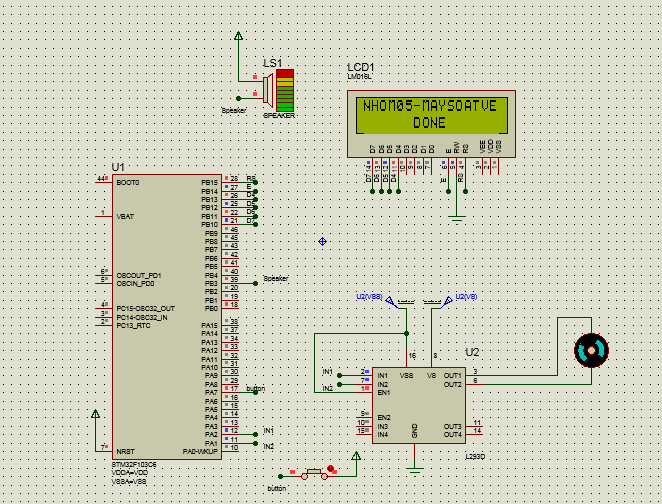


Hình 39: Sơ đồ mạch mô phỏng

Các kết quả mô phỏng thu được:



Hình 40: Mạch mô phỏng kia chua có tín hiệu vào



Hình 41: Khi có tín hiệu vào

# Chương VIII. Phân tích, đánh giá kết quả

Nghiên cứu và thiết kế một hệ thống vẫn luôn là một xu hướng phát triển mạnh mẽ trong nhiều năm trở lại đây. Việc tối ưu hóa hệ thống cũng cần được chú ý phát triển. Sau một thời gian nghiên cứu, thiết kế và thử nghiệm phần cứng cho hệ thống, nhóm chúng em đã đạt được cơ bản các mục tiêu ban đầu đặt ra:

• Nắm được các phương thức truyền nhận dữ liệu thông qua nhiều giao thức.

• Cách xây dựng một mô hình Client-Server cỡ nhỏ, từ đó phát triển lên thành một mô hình lớn.

• Có khả năng thiết kế phần cứng cho một hệ thống thu thập dữ liệu dựa trên yêu cầu có sẵn, từ đó có thể phát triển, nâng cấp dựa trên nền tảng hệ thống cũ khi có yêu cầu mới.

Trong thời gian thực hiện đề tài nhóm chúng em thấy rằng đề tài này là một vấn đề rất rộng, đòi hỏi cần có thêm thời gian để thực hiện nghiên cứu chuyên sâu hơn nữa. Dịch Covid-19 cũng là một trở ngại trong quá trình tìm hiểu và thực hiện đề tài nên chưa thể triển khai hệ thống vào thực tế. Đây là cơ hội cho những bạn sinh viên nghiên cứu về sau, cũng như là thách thức cho chúng em trong việc tối ưu hệ thống. Chúng em mong rằng từ phần cứng đã thiết kế được từ đề tài này sẽ kết hợp, hỗ trợ để phát triển thêm nhiều ứng dụng hơn nữa như trên điện thoại hoặc webserver.

**Tài Liệu Tham Khảo**

[1] Rtos: https://deviot.vn/blog/tong-quan-ve-rtos.32158142

[2] FreeRtos: https://www.pcbelectronics.com/contents/technologies/five-embedded-operating-systems-for-stm32

[3] Datasheet STM32f103: https://www.st.com/resource/en/datasheet/stm32f103c8.pdf

[4] Datasheet GM65: https://www.sunrom.com/p/gm65-interface-board-1dqr2d-bar-code-scanner-qr-code-reader

[5] Datasheet ESP8266 12f: https://pdf1.alldatasheet.com/datasheetpdf/view/1132995/ESPRESSIF/ESP8266.html?fbclid=IwAR0CMs5295lyHzCyOJI4x5Fl 6AEOAb5HEnX5CKvxo0KkcUC9WY8HrgMcTXg

[6] MQTT Explorer: <http://mqtt-explorer.com/>

[7] GM65: <https://pdf1.alldatasheet.com/datasheetpdf/view/1132995/ESPRESSIF/ESP8266.html?fbclid=IwAR0CMs5295lyHzCyOJI4x5Fl6AEOAb5HEnX5CKvxo0KkcUC9WY8HrgMcTXg>

# Báo Cáo Cá Nhân

Trong quá trình thực hiện bài tập dài, nhiệm vụ được giao là tìm hiểu và thực hiện các phần:

* Lựa chọn giải pháp phần mềm
* Triển khai FreeRtos trong STM32CubeMx và STM32CubeIDE
* Xây dựng web client broker
  1. Thực hiện nhiệm vụ

1. Lựa chọn giải pháp phần mềm
   1. Hệ Điều Hành:

Dựa trên nền tảng STM, có năm phương pháp truyền để lựa chọn nhằm đáp ứng các yêu cầu điều khiển thời gian thực của hệ điều hành. Chúng là μClinux, μC / OS-II, eCos, FreeRTOS và Hệ điều hành Dujiangyan (djyos).

* μClinux: Đặc điểm lớn nhất của μClinux là nó được thiết kế cho các bộ vi xử lý không có MMU. Điều này áp dụng cho stm32f103 không có chức năng MMU, nhưng hệ thống này yêu cầu ít nhất 512KB dung lượng RAM, 1MB dung lượng ROM / FLASH và stmf103 có 256K. FLASH yêu cầu bộ nhớ ngoài, điều này làm tăng chi phí thiết kế phần cứng. μClinux có cấu trúc phức tạp, khó cấy ghép và nhân lớn. Hiệu suất thời gian thực của nó cũng rất kém. Nếu trọng tâm phát triển của các sản phẩm nhúng là hệ thống tệp và các ứng dụng mạng, μClinux là một lựa chọn tốt
* μC / OS-II: Nó là một nhân hệ điều hành nhúng với cấu trúc đơn giản, đầy đủ các chức năng và hiệu suất thời gian thực mạnh mẽ. Rất thích hợp cho các CPU không có chức năng MMU. Nó yêu cầu ít không gian nhân và không gian lưu trữ dữ liệu, hiệu suất thời gian thực tốt, khả năng mở rộng tốt và mã nguồn mở. Có rất nhiều tài liệu và ví dụ trên Internet, vì vậy chuyển sang sử dụng CPU stm32f103 sẽ thích hợp hơn.
* ECos: eCos được đặc trưng bởi cấu hình linh hoạt, hỗ trợ di chuyển CPU, không yêu cầu MMU, là mã nguồn mở, có tính di động tốt và phù hợp hơn để di chuyển sang CPU trên nền tảng stm32. Nhưng ứng dụng của eCOS không rộng rãi, không rộng rãi như μC / OS-II và dữ liệu cũng không rộng rãi như μC/OS-II. eCos phù hợp với một số hệ thống nhúng công nghiệp hoặc thương mại nhạy cảm về chi phí, chẳng hạn như một số ứng dụng trong lĩnh vực điện tử tiêu dùng…
* FreeRTOS: Là một hệ điều hành nhẹ, FreeRTOS cung cấp các chức năng như quản lý tác vụ, quản lý thời gian, semaphores, hàng đợi tin nhắn, quản lý bộ nhớ và chức năng ghi nhật ký để đáp ứng nhu cầu của các hệ thống nhỏ hơn. Hạt nhân FreeRTOS hỗ trợ thuật toán lập lịch ưu tiên. Mỗi nhiệm vụ có thể được ưu tiên nhất định theo mức độ quan trọng. CPU luôn chạy tác vụ với mức ưu tiên cao nhất ở trạng thái sẵn sàng. Nhân FreeRT0S hỗ trợ thuật toán lập lịch quay cùng một lúc. Hệ thống cho phép các tác vụ khác nhau sử dụng cùng một mức độ ưu tiên. Khi không có tác vụ ưu tiên cao hơn nào sẵn sàng, các tác vụ ưu tiên giống nhau sẽ chia sẻ thời gian sử dụng CPU.
* hệ điều hành Dujiangyan (djyos): Hệ điều hành Dujiangyan, được gọi là djyos, được đặt theo tên của một dự án bảo tồn nước vĩ đại Dujiangyan. Không giống như các hệ điều hành truyền thống, djyos không phải là một luồng mà là một lõi lập lịch dựa trên sự kiện. Thuật toán lập lịch trình này giải phóng lập trình viên khỏi lối suy nghĩ về việc mô phỏng quá trình thực thi của máy tính mà viết ứng dụng theo cách của thế giới nhận thức của con người. Như VC được giới thiệu trong lập trình nhúng. Thuật toán lập lịch của Djyos cho phép các lập trình viên thoát khỏi những ràng buộc của các luồng và quy trình. DJyos không có API apis và một lập trình viên không biết kiến thức về luồng có thể viết ứng dụng dưới djyos một cách trôi chảy.

Qua các giới thiệu khái quát trên, có thể đưa ra lựa chọn ứng dụng hệ điều hành FreeRtos vào dự án do hệ điều hành FreeRTOS là hệ điều hành hoàn toàn miễn phí với mã nguồn mở, khả năng di động, khả năng mở rộng và chiến lược lập lịch linh hoạt, có thể dễ dàng chuyển sang các bộ vi điều khiển khác nhau.

* 1. Web Client Broker:

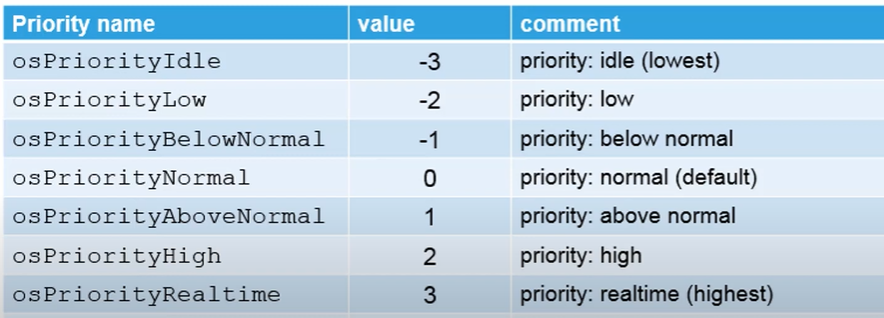
MQTT là một giao thức truyền thông điệp nhỏ gọn dựa trên mô hình publish/subcsribe. Nó sử dụng băng thông thấp, độ tin cậy cao và có thể hoạt động trong điều kiện truyền không ổn định. MQTT được thiết kế cho các mạng SCADA và các tình huống sử dụng băng thông thấp và gần đây nó đã được chứng minh là rất mạnh mẽ trong việc hỗ trợ các dự án IoT. Trong dự án này, giao thức MQTT sẽ được sử dụng để trao đổi dữ liệu giữa hệ thống bán vé và khách hàng mạng tập trung dữ liệu.

Ưu điểm của MQTT:

* Chuyển thông tin hiệu quả hơn
* Tăng khả năng mở rộng
* Giảm đáng kể tiêu thụ băng thông mạng
* Giảm tốc độ cập nhật xuống giây
* Rất phù hợp cho điều khiển và do thám
* Tối đa hóa băng thông có sẵn
* Chi phí cực nhẹ
* Rất an toàn với bảo mật dựa trên sự cho phép
* Được sử dụng bởi ngành công nghiệp dầu khí, Amazon, Facebook và các doanh nghiệp lớn khác
* Tiết kiệm thời gian phát triển
* Giao thức publish/subscribe thu thập nhiều dữ liệu hơn với ít băng thông hơn so với giao thức cũ.

1. Triển khai FreeRtos
   1. Xác định mức độ ưu tiên giữa các nhiệm vụ

Đặt task có nhiệm vụ nhận mã từ esp ở mức cao nhất và các task có nhiệm vụ thực hiện đóng/ mở barie, chuông báo, màn hình hiển thị sẽ ở mức ưu tiên thấp hơn.



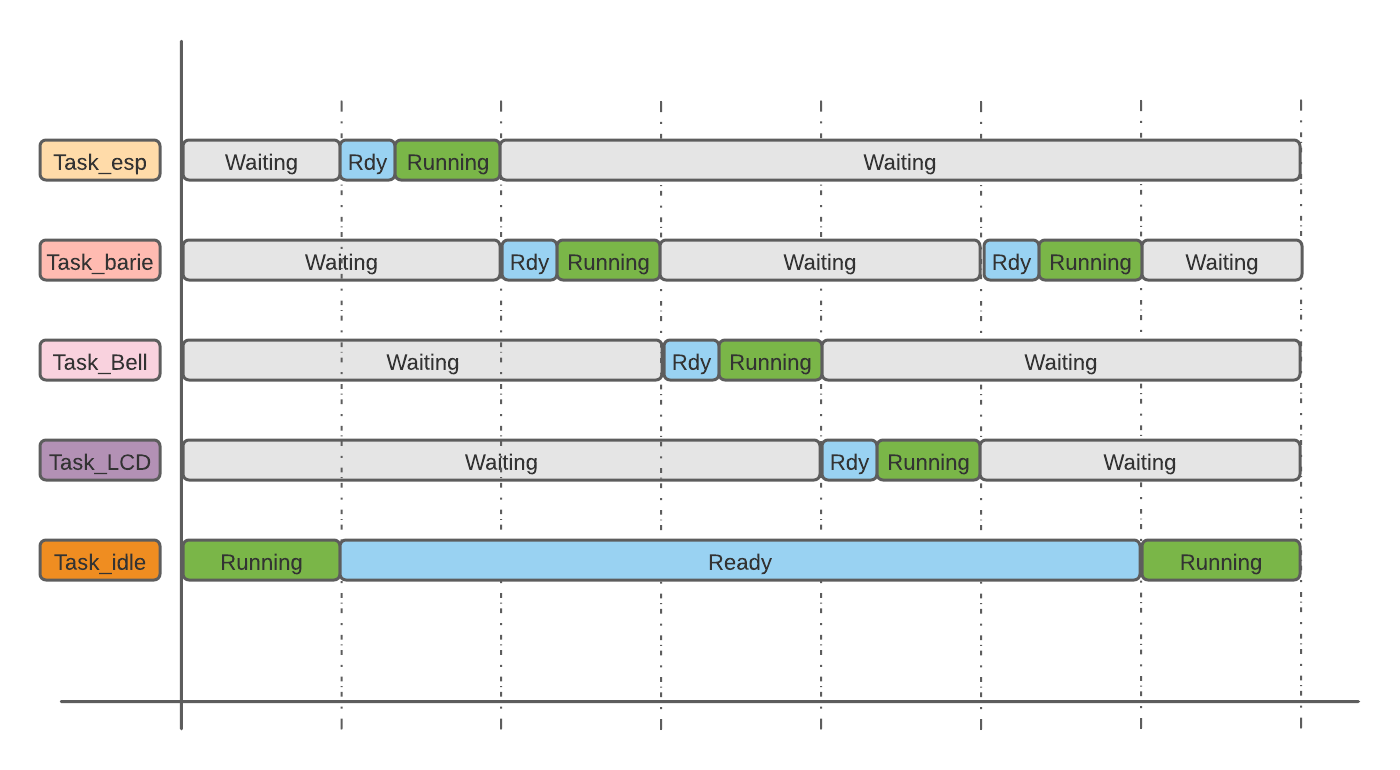
Hình 1: Các mức độ ưu tiên trong phiên bản cmsis-v1

* Task giao tiếp giữa stm32 với esp8266v1 sẽ nhận mức ưu tiên cao nhất do tính quan trọng của nhiệm vụ (mức hightest).
* Các task mở/đóng barie, giao tiếp với LCD, chuông báo ở mức ưu tiên thấp hơn (mức high).
  1. Lựa chọn thuật toán cho lập lịch

Có 3 kỹ thuật được áp dụng là:

* **Round Robin Scheduling**: Các task sẽ được chia 1 khe thời gian cố định, nếu chưa thực thi xong sẽ tạm dừng và đợi đến lượt.
* **Co-operative scheduling**: Các task được thực thi tới khi kết thúc, các task tiếp theo mới được thực thi.
* **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.

Có thể thấy, với các nhiệm vụ mà hệ thống nhúng cần thực hiện, kỹ thuật priority base phù hợp hơn cả do các nhiệm vụ có mức độ ưu tiên khác nhau.

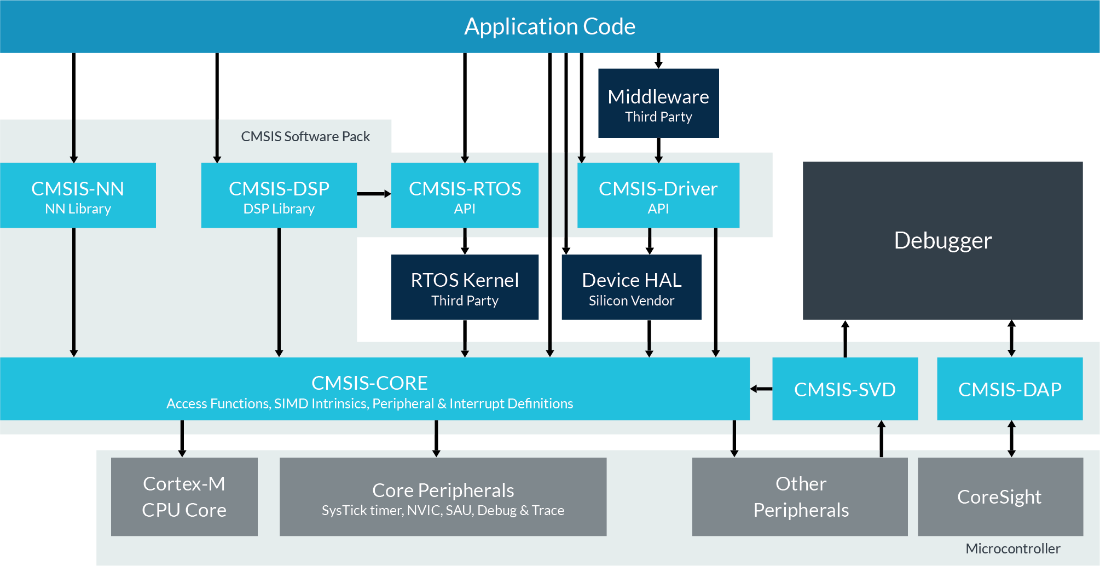


Hình 2: Lịch trình của hệ thống

* 1. CMSIS-RTOS

FreeRTOS là một khung phần mềm cơ bản cho phép chuyển đổi tác vụ, lập lịch,... nhưng chúng ta sẽ không thực hiện các cuộc gọi trực tiếp đến FreeRTOS.

ARM đã tạo thư viện CMSIS-RTOS, cho phép chúng ta thực hiện các cuộc gọi đến RTOS bên dưới, do đó cải thiện tính di động của mã giữa các bộ xử lý ARM khác nhau. Hình ảnh này mô tả cách các thư viện CMSIS của ARM tương tác với phần mềm của bên thứ ba:

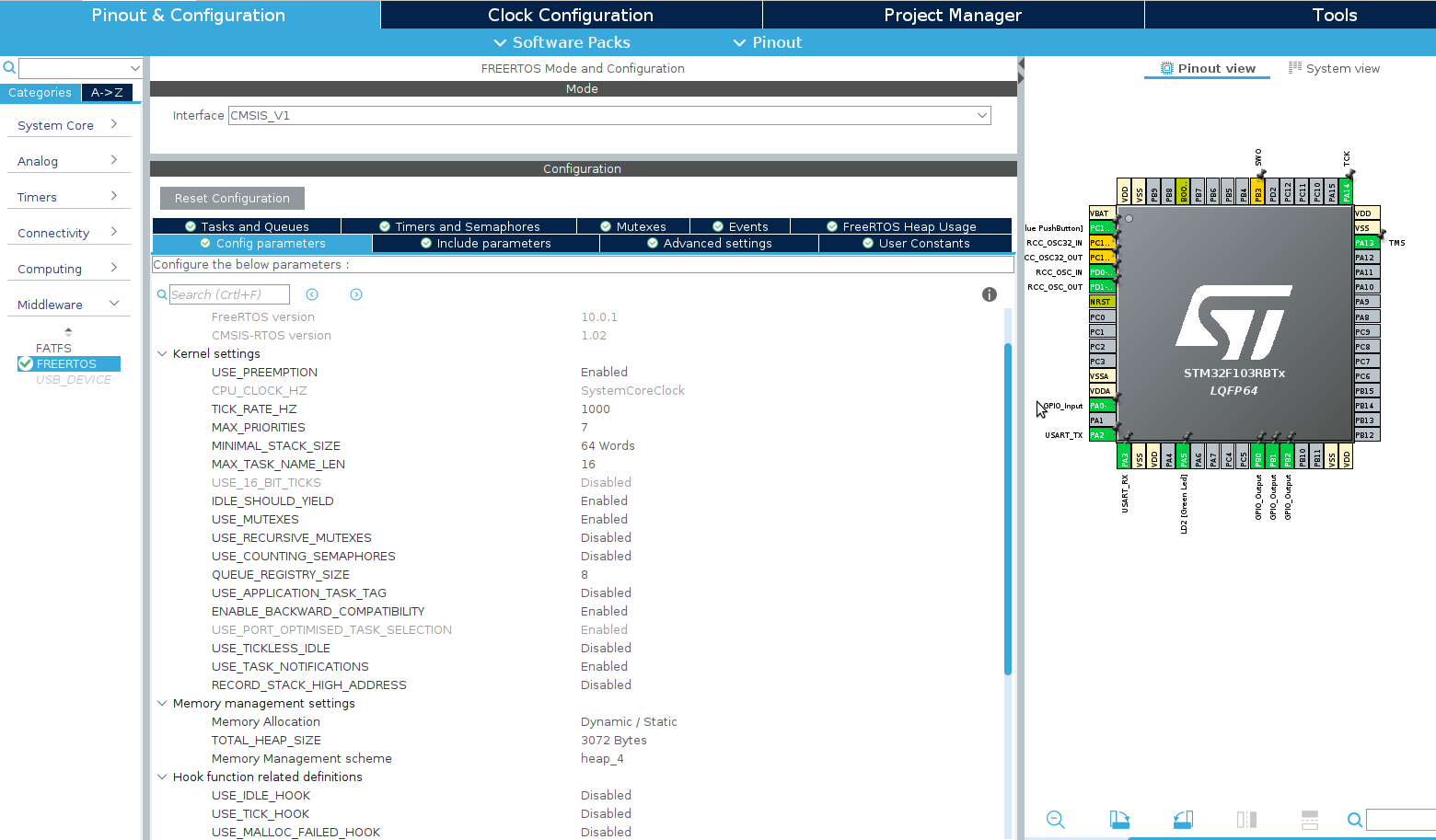


Hình 3: CMSIS

FreeRTOS là “RTOS Kernel” được mô tả trong sơ đồ. Chúng ta sẽ thực hiện các cuộc gọi tới CMSIS-RTOS (có 2 phiên bản là 1 và 2) để kiểm soát FreeRTOS.

* 1. Sử dụng các công cụ CubeMX và CubeIDE để thiết lập FreeRtos
* Thiết lập CubeMX:

Sau khi chọn bộ điều khiển, CubeMX sẽ mở trang mặc định như hình dưới, chọn FreeRtos ở thanh menu và lựa chọn phiên bản CMSIS.



Hình 4: Thiết lập FreeRtos trong CubeMX

Lựa chọn phiên bản CMSIS\_V1 do nó được hỗ trợ bởi hầu hết các thiết bị STM32.

Thẻ Config parameters cho phép ta có thể cấu hình cài đặt ban đầu những thứ liên quan tới kernel và memory.

Thẻ Include parameters cho phép bật/tắt một số define về Task, Queue, Semaphore, …

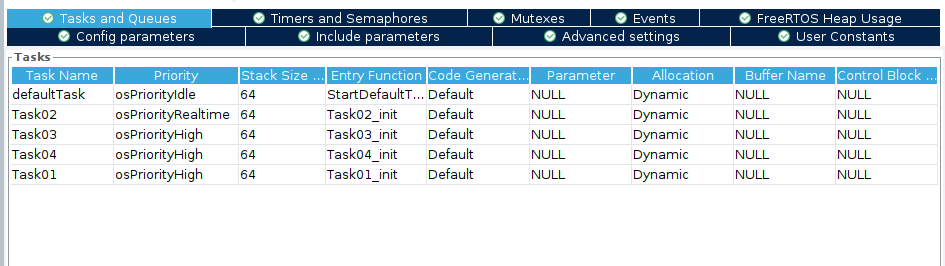
Thẻ Task and Queue giúp ta có thể tạo/ thêm các task và queue.

Thẻ Timers and Semaphores cho phép thêm Timers, Mutexes, Semaphore, có Mutex và Binary Semaphore mặc định là có thể add thêm mới, với Recursive Mutexes cũng như Binary Semaphore để hiện được nút add thì cần phải enable các define ở thẻ Include parameters.

Thẻ User Constant để có thể thêm các constant của user có thể tự định nghĩa vào.

Trong phần kernel settings, cài đặt các tham số cho hệ điều hành (nếu cần thiết).

Tiếp theo chuyển tới tab "Task and Queues", thêm mới và thiết lập các thuộc tính cho các task.

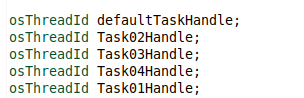


Hình 5: Thiết lập các tác vụ

* Tạo các task:

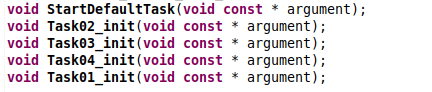
Trong giao diện CubeIDE, tiến hành lập trình, khai báo và cấu hình cho các Task.

Xác định một ThreadID cho mỗi nhiệm vụ. Biến này sẽ lưu trữ ID duy nhất của nhiệm vụ sau khi được tạo. Sau đó, tất cả các hoạt động sẽ yêu cầu ID này.



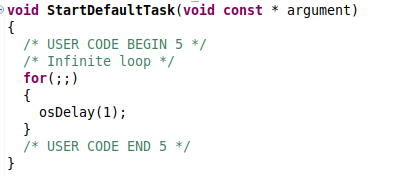
Hình 6: Khởi tạo ID cho các Task

Khởi tạo các hàm nhiệm vụ:



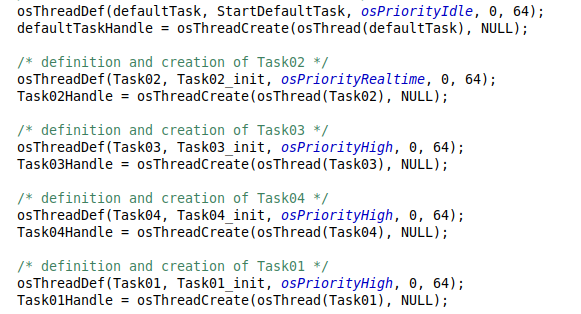
Hình 7: Khởi tạo các Task

Xây dựng các hàm cho nhiệm vụ. Các chức năng chính của nhiệm vụ sẽ được viết trong các hàm này.



Hình 8: Xây dựng các hàm thực thi nhiệm vụ

Bên trong hàm "main.c", cần định nghĩa các luồng và tạo luồng thực thi cho các nhiệm vụ với các mức độ ưu tiên đã xác định trong phần "Lựa chọn thuật toán lập lịch".



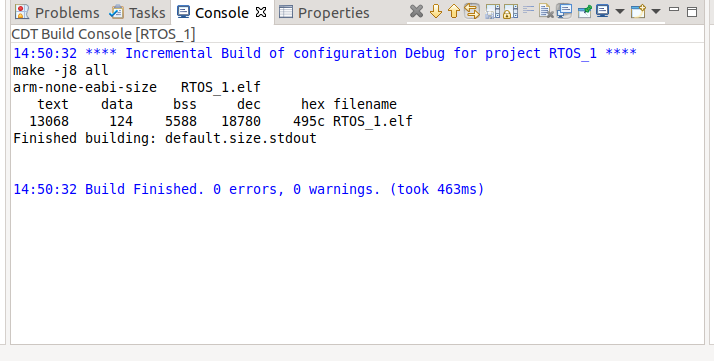
Hình 9: Định nghĩa và tạo luồng cho các tác vụ

* osThreadDef có các tham số là tên tác vụ, hàm thực thi tác vụ, mức độ ưu tiên, đối tượng, kích thước ngăn xếp.
* Sau khi nhiệm vụ đã được xác đinh, khởi tạo bằng hàm osThreadCreate và gán ID tương ứng.

Sau khi cài đặt cũng như khai báo hoàn tất, sẽ có câu lệnh osKernelStart() được gọi trong hàm main(). Lúc này, Các task sẽ thực thi như những gì đã cài đặt trong phần lập lịch.

Ngoài ra, thay vì dùng HAL\_Delay(), chúng ta nên đổi thành osDelay(). Hàm HAL\_delay() trong các tác vụ có mức độ ưu tiên cao có thể sẽ làm hỏng bộ xử lý do nó sẽ ngăn việc chuyển đổi xuống các tác vụ thấp hơn. Khi sử dụng osDelay(), bộ lập lịch sẽ hiểu rằng có thể chuyển sang 1 tác vụ khác khi tác vụ cao hơn đang chờ đợi.

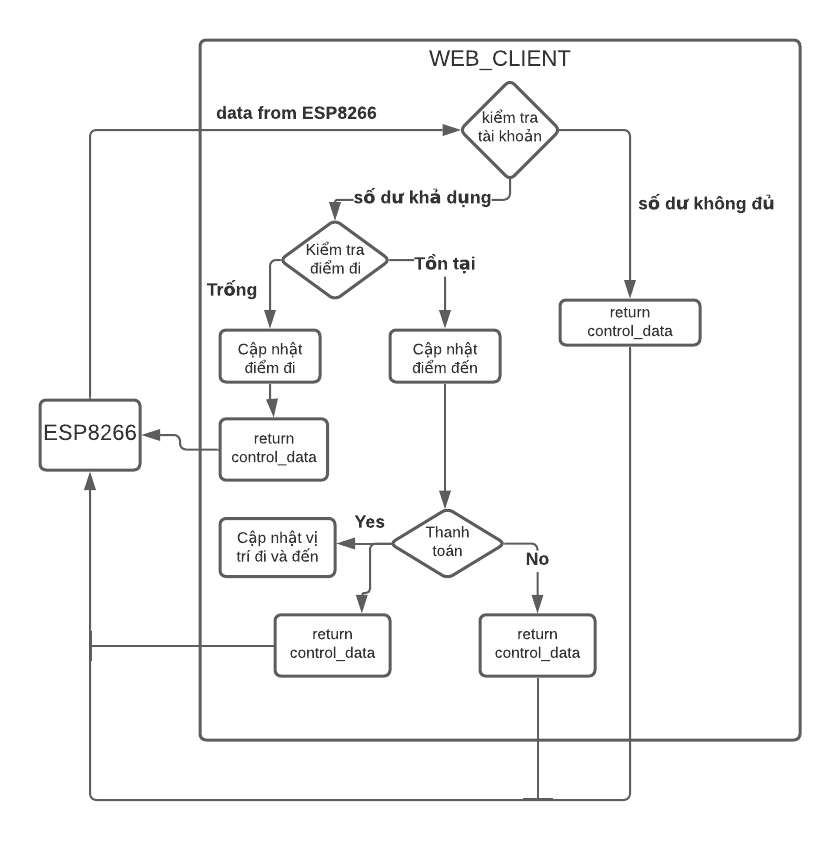
Màn hình hiển thị kết quả khi Build thành công:



1. Xây dựng web client broker

Với yêu cầu 1 vị trí để lưu trữ thông tin của các thẻ cũng như trạng thái của các thẻ vào từng thời điểm, dựa vào các thông tin đó để thực hiện các bước xử lý và tính toán đơn giản như tính chi phí đi lại.

* Yêu cầu chức năng đối với web:

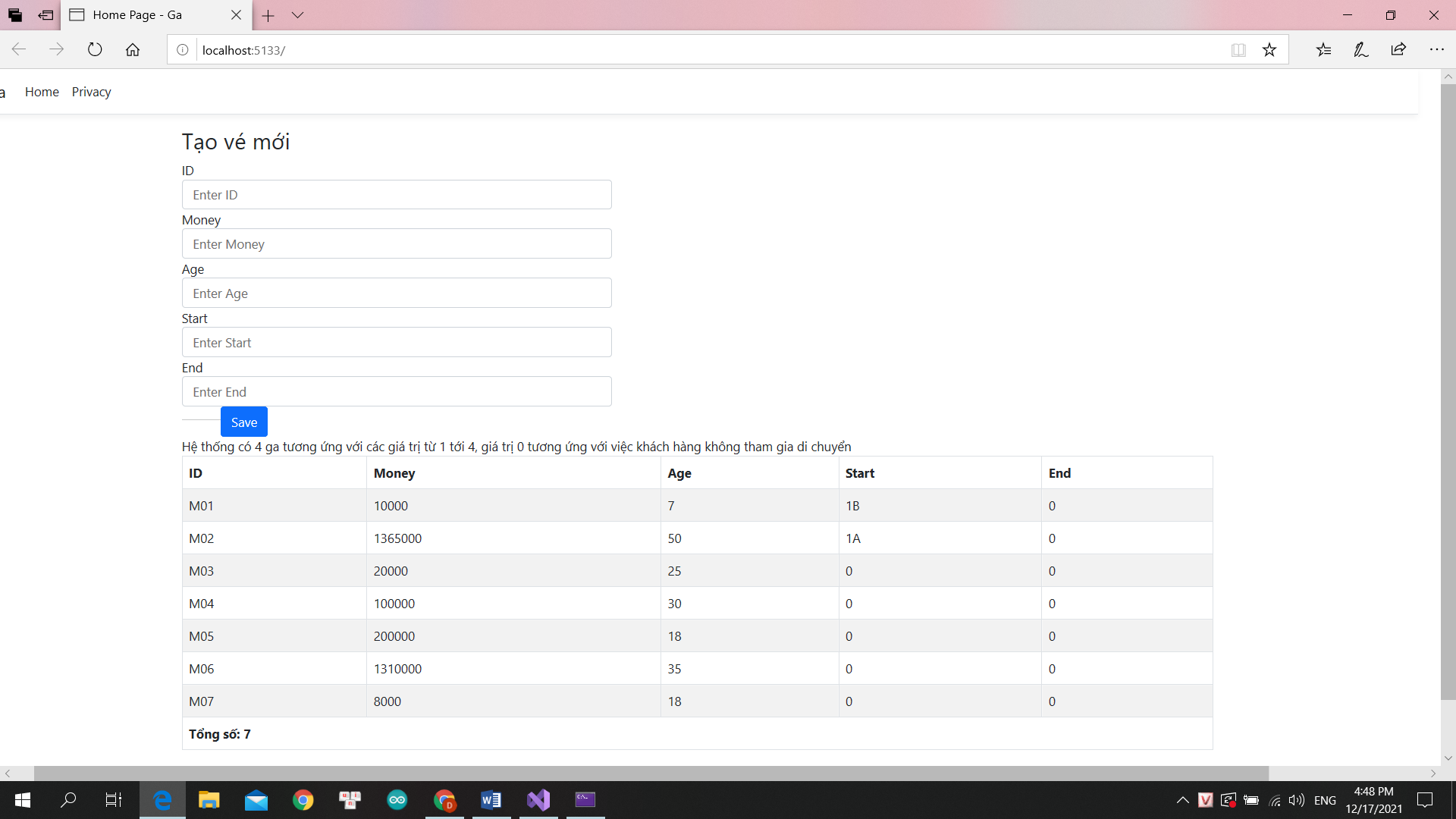


Hình 10: Sơ đồ mô tả các tác vụ của web-client

Với các yêu cầu như đã nêu các phần trước, có thể đưa ra một sơ đồ tổng quát mô tả lại quá trình hoạt động của đơn vị web-client.

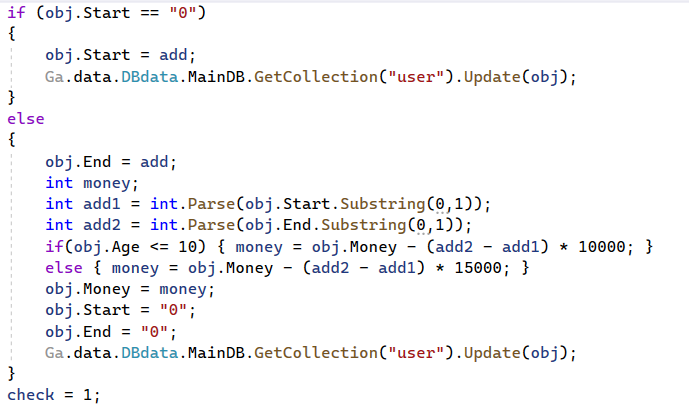
Về cơ bản, web-client cần phải đáp ứng được các nhiệm vụ cơ bản như: đọc mã được gửi từ các máy soát vé (ID+ADD), Kiểm tra số dư tài khoản thông qua ID, cập nhật vi trí xuất phát và kết thúc của hành khách, Thanh toán tiền theo lộ trình khách đã di chuyển, phản hồi tín hiệu điều khiển về các máy soát vé 1 cách chính xác.

* + Kết quả thu được:



Hình 11: Giao diện web-client

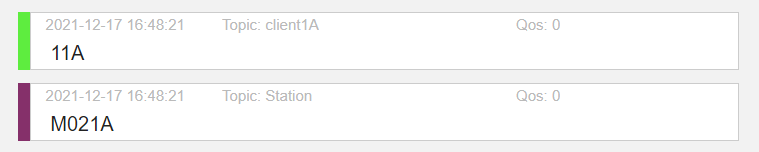
Việc xây dựng web-client sử dụng mô hình web api mvc với ngôn ngữ C#, web sẽ lưu và hiển thị các thông tin quan trọng về từ ID như số dư, tuổi chủ sở hữu, diểm bắt đầu và điểm kết thúc trong lộ trình. Các giá trị về Lộ trình của hành khách sẽ được để mặc định là 0, có nghĩa khách hàng không tham gia di chuyển nếu giá trị cột Start = 0 và khách hàng chưa hoàn thành thanh toán hoặc đang trong quá trình di chuyển nếu cột End = 0. Khi khách hàng thanh toán xong, các giá trị sẽ trở về 0 như hình dưới.

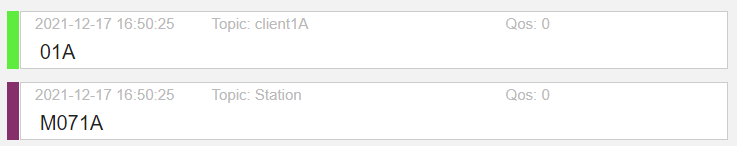


Hình 12: Tính toán chi phí hành trình

Với mô hình thanh toán đang áp dụng hiện nay tại Việt Nam là sử dung vé tuyến sẽ gây thua lỗ cho các công ty dịch vụ vận tải khách. Chính vì vậy, cần phải xác định chính xác quảng đường khách hàng đã di chuyển để tính toán chi phí cần chi trả.

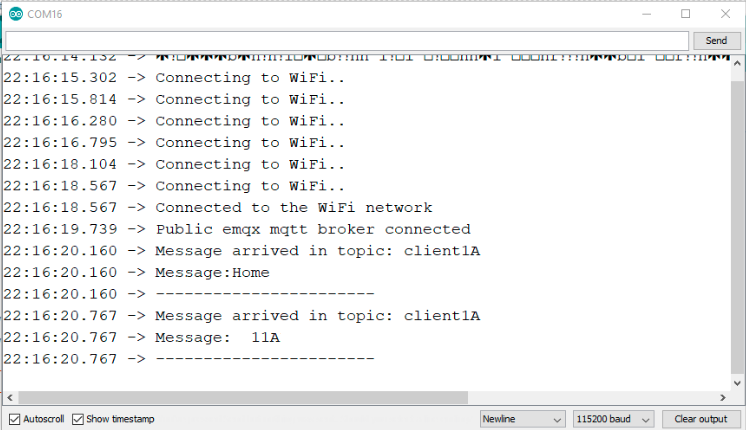
Trong dự án này, tạm thời giả thuyết giá vé di chuyển giữa các ga cạnh nhau là bằng nhau. Do đó, giá vé bằng được tính bằng số chặng \* giá từng chặng. Giả sử vé cho trẻ em (<10 tuổi ) là 10000đ và người lớn (> 10 tuổi) là 15000đ.





Hình 13: Thông tin trao đổi giữa máy soát vé và web-client

Trong hình trên, có thể thấy kết quả truyền tin giữa thiết bị soát vé (client1A) với web-client (Station). Khi thẻ có ID là M02 được quét, sẽ gửi mã M021A tới web-client. Mã này cho biết cần phải kiểm tra thẻ có ID là M02 và gửi trả tín hiệu về client1A, nếu thẻ hợp lệ trả CS là 1 về client1A. Tương tự mới thẻ có ID là M07 được quét tại client 1A, tuy nhiên, có thể thấy trong hình số x, số dư trong thẻ chỉ còn 8000đ, không đủ cho việc di chuyển giữa 1 chặng (10000đ với vé trẻ em). Do đó, tín hiệu trả về cho client1A là 0.



Hình 14: Kết quả thu được khi trao đổi thông tin giữa module esp8266 12f của máy soát vé và web-client

Hình trên cho thấy kết quả thử nghiệm thực tế về việc trao đổi dữ liệu giữa esp 8266 12f với web client.

* 1. Kết quả đạt được

Qua quá trình tìm hiểu cũng như học tập và triển khai thử nghiệm, em đã có các kết quả sau:

1. Lựa chọn hệ điều hành FreeRtos và giao thức MQTT broker cho việc trao đổi dữ liệu.
2. Triển khai thử nghiệm hệ điều hành FreeRtos trên Stm32CubeMx và Stm32CubeIDE.
3. Xây dựng thành công web client có nhiệm vụ lưu trữ các thông tin cơ bản nhất của các thẻ.
4. Triển khai thực tế thành công việc trao đổi dữ liệu giữa module esp8266 12f và web client.
5. Mã nguồn thực hiện: <https://github.com/Huster2017/Embedded_20211_group5>