ĐẠI HỌC QUỐC GIA TP.HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA ĐIỆN – ĐIỆN TỬ
----------o0o----------

Bài tập lớn

LẬP TRÌNH HỆ THỐNG NHÚNG

LẬP TRÌNH MẠCH ĐIỀU KHIỂN ĐÈN GIAO THÔNG SỬ DỤNG FREERTOS

Nhóm 13

Danh sách sinh viên thực hiện:

Nguyễn Hoàng Đạt	2111011
Nguyễn Huỳnh Minh Quang	2010548
Văn Công Bảo	2012675

TP. HỒ CHÍ MINH, THÁNG 11 NĂM 2024

MỤC LỤC

1. GIỚI THIỆU

1.1 Tổng quan

Hệ thống điều khiển đèn giao thông đóng vai trò quan trọng trong việc duy trì trật tự và đảm bảo an toàn giao thông tại các đô thị hiện đại. Với sự phát triển không ngừng của công nghệ nhúng, các giải pháp lập trình trên nền tảng vi điều khiển đã trở thành xu hướng chủ đạo trong thiết kế hệ thống tự động hóa. Trong bối cảnh đó, FreeRTOS, một hệ điều hành thời gian thực nhẹ và hiệu quả, cung cấp một công cụ mạnh mẽ để quản lý đa nhiệm, đảm bảo tính chính xác và linh hoạt trong việc điều khiển các tác vụ phức tạp như quản lý đèn giao thông.

Mục tiêu chính của nghiên cứu này là xây dựng một hệ thống điều khiển đèn giao thông sử dụng FreeRTOS nhằm đáp ứng yêu cầu vận hành ổn định, tối ưu hóa thời gian đáp ứng và dễ dàng tích hợp với các hệ thống giao thông thông minh trong tương lai. Từ đó, đề tài đặt ra nhiệm vụ nghiên cứu, thiết kế và triển khai một giải pháp hiệu quả cho bài toán này.

1.2 Nhiệm vụ đề tài

Để đạt được mục tiêu đã đề ra, đề tài cần thực hiện các nhiệm vụ chính sau:

Nội dung 1: Tìm hiểu nguyên lý hoạt động và lý thuyết liên quan đến hệ thống đèn giao thông, bao gồm các yêu cầu kỹ thuật, tiêu chuẩn thiết kế, và các thuật toán điều khiển thường được áp dụng. Quá trình này cung cấp nền tảng lý thuyết để phân tích và xây dựng hệ thống.

Nội dung 2: Nghiên cứu các thành phần phần cứng cần thiết như vi xử lý ESP32, các module ngoại vi (đèn LED, LED 7 đoạn) và bo mạch phát triển. Đồng thời, phân tích cách tích hợp các thành phần này trong hệ thống.

Nội dung 3: Thiết kế và triển khai bộ điều khiển đèn giao thông dựa trên FreeRTOS. Cụ thể, đề tài sẽ áp dụng giải thuật quản lý đa nhiệm để phân phối tài nguyên CPU một

cách hợp lý, đồng thời sử dụng các tính năng của FreeRTOS để xử lý các tác vụ ưu tiên cao như phát hiện trạng thái giao thông và điều khiển luồng đèn tương ứng.

Nội dung 4: Thực hiện kiểm thử và đánh giá hệ thống nhằm đảm bảo tính chính xác, độ tin cậy, và khả năng đáp ứng theo thời gian thực. Điều này bao gồm xây dựng các kịch bản thử nghiệm thực tế và so sánh kết quả đạt được với các tiêu chí thiết kế ban đầu.

Những nhiệm vụ này không chỉ đảm bảo tính khả thi của hệ thống mà còn cung cấp cơ sở để mở rộng và tích hợp vào các hệ thống giao thông thông minh trong tương lai.

2. LÝ THUYẾT VỀ LẬP TRÌNH HỆ THỐNG NHÚNG VỚI FREERTOS

2.1. Tổng quan về hệ thống nhúng

Hệ thống nhúng là các hệ thống điện tử được thiết kế để thực hiện một hoặc một vài chức năng cụ thể trong một thiết bị lớn hơn. Các hệ thống này thường hoạt động dựa trên các vi điều khiển hoặc vi xử lý, với khả năng tích hợp cao, tiêu thụ năng lượng thấp, và vận hành trong các môi trường hạn chế về tài nguyên.

Lập trình hệ thống nhúng đòi hỏi sự tối ưu hóa về phần cứng và phần mềm để đảm bảo hệ thống vận hành chính xác, đáng tin cậy và đạt hiệu suất cao. Các yêu cầu đặc thù của hệ thống nhúng, như thời gian thực, tính đồng thời và quản lý tài nguyên hạn chế, khiến việc phát triển phần mềm trở nên phức tạp.

2.2. Giới thiệu về freeRTOS

FreeRTOS (Free Real-Time Operating System) là một hệ điều hành thời gian thực mã nguồn mở, được thiết kế đặc biệt cho các hệ thống nhúng. Với khả năng hỗ trợ đa nhiệm (multitasking), FreeRTOS cho phép các tác vụ (task) chạy đồng thời trên một vi điều khiển đơn lẻ. FreeRTOS được phát triển nhằm cung cấp một giải pháp nhẹ, hiệu quả và dễ sử dụng cho các ứng dụng nhúng cần đáp ứng thời gian thực.

Các đặc điểm nổi bật của FreeRTOS bao gồm:

- *Đa nhiệm (Multitasking):* Hỗ trợ chạy nhiều tác vụ đồng thời bằng cách phân chia tài nguyên CPU một cách linh hoạt.
- *Quản lý tài nguyên:* Cung cấp các cơ chế quản lý tài nguyên như semaphore, mutex và queue để đảm bảo tính đồng bộ giữa các tác vụ.
- *Đáp ứng thời gian thực:* FreeRTOS sử dụng bộ lập lịch ưu tiên để đảm bảo các tác vụ quan trọng được xử lý trong thời gian yêu cầu.
- *Nhẹ và linh hoạt:* Phù hợp với các hệ thống nhúng hạn chế về tài nguyên, hỗ trợ nhiều kiến trúc vi xử lý.

2.3. Kiến trúc của freeRTOS

FreeRTOS bao gồm các thành phần chính sau:

Kernel: Lõi của FreeRTOS, chịu trách nhiệm lập lịch và quản lý tác vụ. Kernel sử dụng thuật toán ưu tiên dựa trên thời gian thực để quyết định tác vụ nào sẽ được thực thi.

Task: Tác vụ là đơn vị cơ bản của chương trình trong FreeRTOS, mỗi tác vụ hoạt động như một luồng riêng biệt và có thể được gán mức ưu tiên.

Queue: Cấu trúc dữ liệu dùng để giao tiếp giữa các tác vụ, cho phép truyền dữ liệu an toàn trong môi trường đa nhiệm.

Semaphore và Mutex: Các cơ chế đồng bộ hóa giúp giải quyết xung đột truy cập tài nguyên chung. Semaphore thường được dùng cho việc đồng bộ tín hiệu, trong khi Mutex thích hợp để bảo vệ tài nguyên dùng chung.

Timer: Công cụ giúp thực hiện các tác vụ định kỳ hoặc trì hoãn theo yêu cầu thời gian.

2.4. Lợi ích và ứng dụng của FreeRTOS trong lập trình hệ thống nhúng

Việc sử dụng FreeRTOS trong hệ thống nhúng mang lại nhiều lợi ích so với việc lập trình tuần tự thông thường. Nó cho phép đơn giản hóa việc quản lý các tác vụ đồng thời, đặc biệt trong các ứng dụng phức tạp, Đảm bảo các tác vụ quan trọng được xử lý đúng hạn, phù hợp với các ứng dụng yêu cầu độ chính xác cao. Đồng thời, với kiến trúc module, FreeRTOS cho phép dễ dàng tái sử dụng mã nguồn và mở rộng chức năng của

hệ thống. Cuối cùng vì là một hệ điều hành mã nguồn mở, FreeRTOS có tài liệu phong phú và được hỗ trợ bởi một cộng đồng lập trình viên lớn.

Với các ưu điểm về hiệu năng, tính linh hoạt và khả năng đáp ứng, FreeRTOS là một lựa chọn tối ưu cho việc phát triển các ứng dụng nhúng phức tạp, đặc biệt trong các hệ thống yêu cầu cao về thời gian thực như trong hệ thống IoT, tự động hóa công nghiệp, thiết bị y tế, hệ thống giao thông,…

3. THIẾT KẾ VÀ THỰC HIỆN PHẦN CỨNG

3.1. Các linh kiện và module

3.1.1. ESP8266

Hình 3-1 Module ESP8266

ESP8266 là vi điều khiển tích hợp Wi-Fi của Espressif Systems, nổi bật với chi phí thấp, hiệu suất cao, và khả năng kết nối không dây, có hỗ trợ freeRTOS đặc biệt phù hợp cho các ứng dụng IoT. ESP8266 hỗ trợ giao thức TCP/IP, bảo mật WPA/WPA2, lập trình qua AT Command, nâng cấp firmware OTA, và tích hợp nhiều nền tảng phát triển như Arduino IDE và NodeMCU. ESP8266 được sử dụng rộng rãi trong nhà thông minh, giám sát từ xa, và điều khiển không dây nhờ tính linh hoạt, hiệu năng tốt và dễ triển khai.

Thông số kỹ thuật chính

- Vi điều khiển: Tensilica L106 32-bit, xung nhịp 80–160 MHz.
- Bộ nhớ: RAM 96 kB (dành cho chương trình), flash hỗ trợ đến 16 MB.
- Điện áp hoạt động: 2.5–3.6V, tiêu chuẩn 3.3V.

- Kết nối Wi-Fi: Chuẩn IEEE 802.11 b/g/n, chế độ STA, AP hoặc STA+AP.
- GPIO: 9–17 chân, hỗ trợ linh hoạt cho cảm biến và thiết bị ngoại vi.
- Tiêu thụ năng lượng: Từ 10 µA (Deep Sleep) đến 170 mA (truyền dữ liệu).

3.1.2. Module đèn LED đỏ vàng xanh

Hình 3-2 Module đèn LED giao thông

Module đèn LED giao thông giúp cho phần cứng tổng thể trở nên gọn gàng hơn với việc các LED và điện trở hạn dòng được hàn lên PCB. Điều khiển bằng cách nối đất module, điều khiển các đèn bật tắt bằng cách cấp nguồn 3.3 – 5V từ chân GPIO từ vi xử lý vào các chân R – Red, Y – Yellow, G – Green tương ứng.

3.1.3. Module LED 7 đoạn TM1637

Hình 3-3 Module LED 7 đoạn TM1637

Module LED 7 đoạn TM1637 là một giải pháp hiển thị phổ biến trong các ứng dụng nhúng, được thiết kế để đơn giản hóa việc điều khiển LED 7 đoạn bằng cách tích hợp chip TM1637. Module này thường được sử dụng trong các thiết bị điện tử hiển thị số

như đồng hồ, bộ đếm, hoặc các giao diện người dùng cơ bản nhờ khả năng hoạt động ổn định, dễ lập trình và chi phí thấp.

Module TM1637 bao gồm:

Màn hình LED 7 đoạn: Thường có 4 chữ số, mỗi chữ số hiển thị từ 0 đến 9 hoặc các ký tự đơn giản như A, b, C, d.

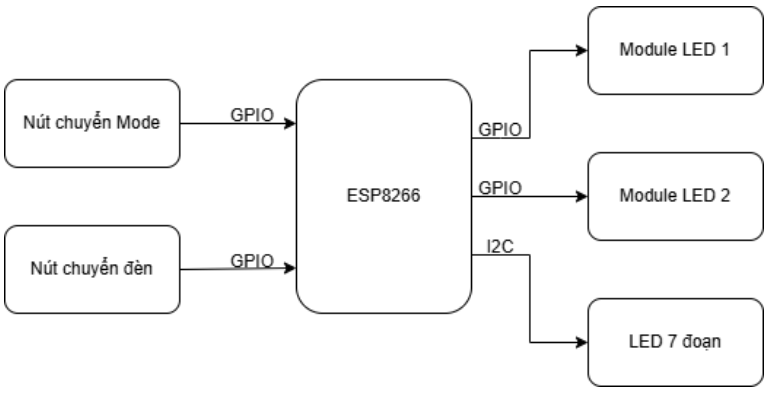
Chip điều khiển TM1637: Quản lý việc quét LED và điều chỉnh độ sáng, giúp giảm tải xử lý cho vi điều khiển.

Giao tiếp: Module sử dụng giao tiếp 2 dây (CLK và DIO), giúp tối ưu hóa số lượng chân cần sử dụng trên vi điều khiển.

Thông số kỹ thuật:

- *Điện áp hoạt động:* 3.3V đến 5V.
- *Dòng điện tiêu thụ:* ~80 mA (phụ thuộc vào độ sáng và số lượng LED hoạt động).
- *Độ sáng:* Có thể điều chỉnh với 8 mức độ thông qua phần mềm.
- *Kích thước:* Thường khoảng 42 mm x 24 mm.

3.2. Sơ đồ kết nối mạch phần cứng



Hình 3-4 Sơ đồ kết nối phần cứng

Module ESP8266 là vi xử lý trung tâm của hệ thống. Các nút bấm chuyển Mode và chuyển đèn dùng để điều khiển các tác vụ của hệ thống. Vi xử lý điều khiển module đèn

giao thông bằng cách dùng các chân GPIO bật tắt các màu LED tương ứng, và điều khiển module LED 7 đoạn thông qua giao tiếp I2C.

4. THIẾT KẾ VÀ THỰC HIỆN PHẦN MỀM

4.1. Yêu cầu thuật toán của hệ thống

4.1.1. Chế độ tự động

Chế độ này được thiết kế để hoạt động theo trình tự các trạng thái cố định, dựa trên các tín hiệu đầu vào và thời gian định sẵn.

Đầu vào:

- Timer: Thời gian được định sẵn cho mỗi trạng thái.
- Nút chuyển chế độ: Cho phép người dùng thay đổi giữa chế độ Tự động và Thủ công.

Đầu ra:

- Đèn tín hiệu giao thông: Điều khiển các đèn xanh, vàng, đỏ theo trạng thái hiện tại.
- Màn hình LED 7 đoạn: Hiển thị đếm ngược thời gian của trạng thái hiện tại.
- Timer: Cập nhật thời gian còn lại để kích hoạt trạng thái kế tiếp.

Các trạng thái hoạt động:

- **Trạng thái 1:** Đèn 1 Xanh, Đèn 2 Đỏ.
    - Đèn 1 sáng xanh trong 30 giây, đèn 2 sáng đỏ trong 35 giây hiển thị ra LED 7 đoạn.
- **Trạng thái 2:** Đèn 1 Vàng, Đèn 2 Đỏ.
    - Thời gian chuyển tiếp từ xanh sang đỏ trong 5 giây hiển thị ra LED 7 đoạn.

- **Trạng thái 3:** Đèn 1 Đỏ, Đèn 2 Xanh.
    - Đèn 2 sáng xanh trong 30 giây, đèn 1 sáng đỏ trong 35 giây hiển thị ra LED 7 đoạn.
- **Trạng thái 4:** Đèn 1 Đỏ, Đèn 2 Vàng.
    - Thời gian chuyển tiếp từ xanh sang đỏ trong 5 giây hiển thị ra LED 7 đoạn.

4.1.2. Chế độ thủ công

Chế độ này cho phép người dùng kiểm soát trạng thái của hệ thống bằng cách sử dụng nút nhấn.

Đầu vào:

- Nút chuyển màu: Chuyển đổi trạng thái đèn tín hiệu theo yêu cầu.
- Nút chuyển chế độ: Chuyển đổi giữa chế độ Tự động và Thủ công.

Đầu ra:

- Đèn tín hiệu giao thông: Hiển thị trạng thái hiện tại theo điều khiển của người dùng.

Các trạng thái hoạt động:

- **Trạng thái 1:** Đèn 1 Xanh, Đèn 2 Đỏ.
- **Trạng thái 2:** Đèn 1 Vàng, Đèn 2 Đỏ.
    - Khi nhấn nút chuyển màu, giữ đèn 1 Vàng trong 3 giây trước khi chuyển sang trạng thái Đèn 1 Đỏ, Đèn 2 Xanh.
- **Trạng thái 3:** Đèn 1 Đỏ, Đèn 2 Xanh.
    - Khi nhấn nút chuyển màu, đèn 2 chuyển Vàng trong 3 giây, sau đó chuyển sang Đỏ, đồng thời Đèn 1 chuyển sang Xanh.
- **Trạng thái 4:** Đèn 1 Đỏ, Đèn 2 Vàng.

4.1.3. Chuyển đổi giữa các chế độ

Từ Tự động sang Thủ công:

- Nếu đèn đang Xanh hoặc Đỏ: Tắt LED 7 đoạn và giữ nguyên trạng thái hiện tại.
- Nếu đèn đang Vàng: Chuyển sang trạng thái Đỏ, sau đó giữ nguyên trạng thái như trên.

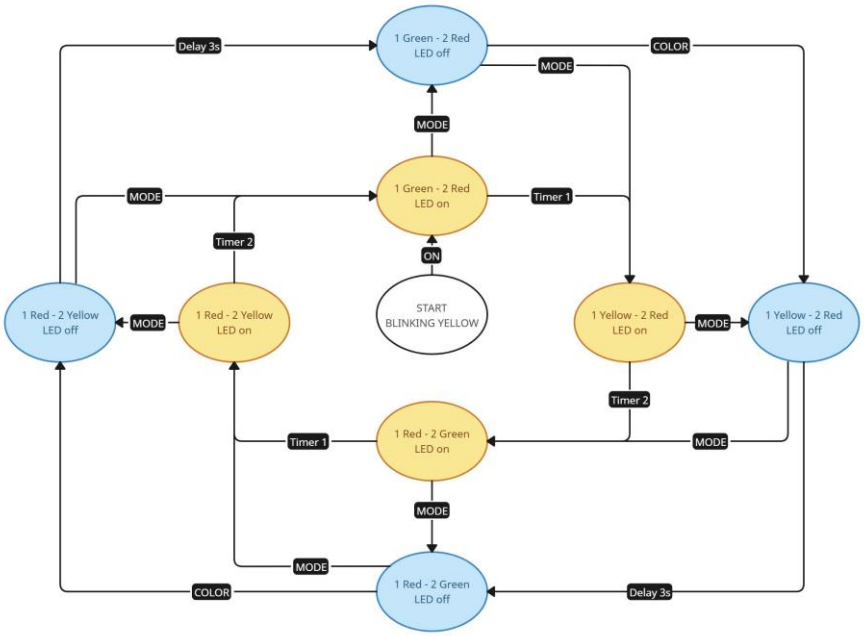
Từ Thủ công sang Tự động:

- Khi nhận tín hiệu chuyển chế độ, hệ thống chuyển sang trạng thái kế tiếp dựa trên trình tự của chế độ Tự động.
- Trong quá trình này, nếu đèn đang Xanh, bật cờ greenFlag; nếu đèn chuyển sang Vàng, bật cờ yellowFlag.

4.2. Máy trạng thái và lưu đồ giải thuật của hệ thống

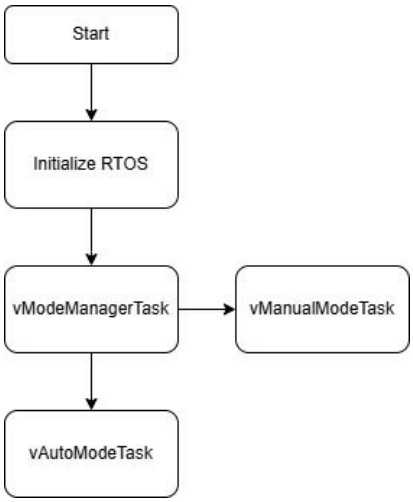
4.2.1. Máy trạng thái của hệ thống

Dựa vào các yêu cầu về thuật toán của hệ thống ở trên, thiết kế máy trạng thái sau:



Hình 4-1 Máy trạng thái của hệ thống đèn giao thông

4.2.2. Lưu đồ giải thuật của hệ thống



Hình 4-2 Lưu đồ giải thuật chương trình chính

Lưu đồ giải thuật chương trình chính mô tả quy trình hoạt động của hệ thống điều khiển đèn giao thông trong môi trường FreeRTOS. Các bước trong lưu đồ được thiết kế để quản lý và chuyển đổi giữa hai chế độ chính: Tự động và Thủ công với các bước trong lưu đồ như sau:

Start: Điểm xuất phát cho hệ thống. Quá trình bắt đầu khi hệ thống được cấp nguồn.

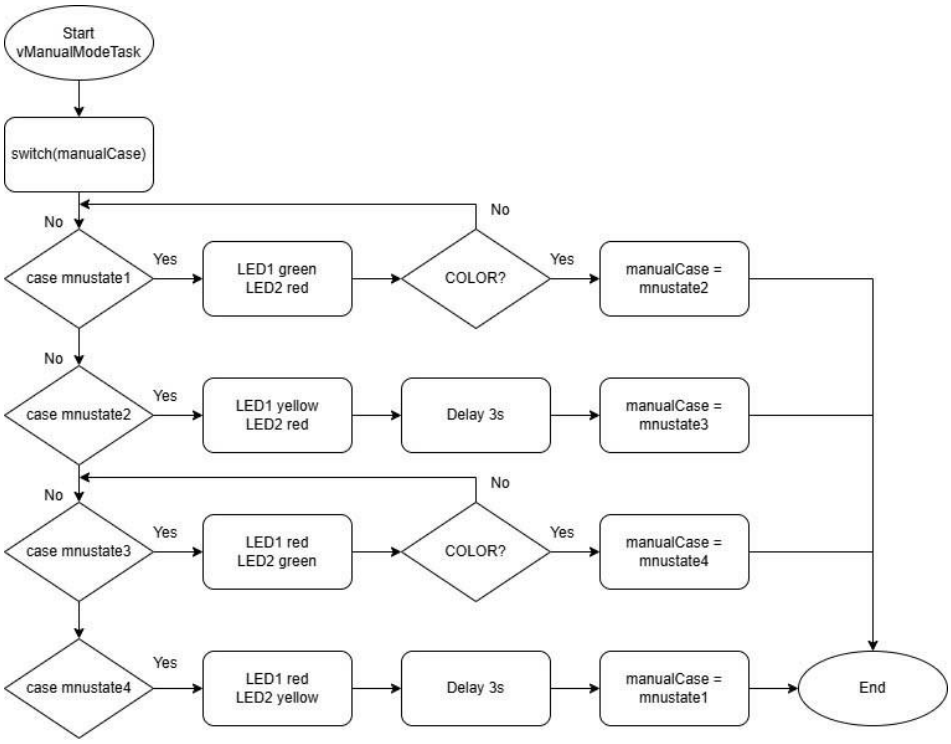
Initialize RTOS: Bước này khởi tạo Real-Time Operating System (RTOS), cụ thể là FreeRTOS, để điều khiển các tác vụ và tài nguyên trong hệ thống. RTOS sẽ đảm nhận việc phân chia thời gian và ưu tiên các tác vụ khác nhau, giúp đảm bảo hệ thống hoạt động mượt mà và hiệu quả.

vModeManagerTask: Sau khi RTOS được khởi tạo, hệ thống sẽ chuyển sang tác vụ **vModeManagerTask**. Tác vụ này có vai trò quản lý chế độ hoạt động của hệ thống, bao gồm việc chuyển đổi giữa chế độ Tự động và Thủ công. Các quyết định chuyển đổi giữa các chế độ sẽ được thực hiện trong tác vụ này.

vManualModeTask:
Nếu hệ thống đang ở chế độ **Thủ công**, tác vụ **vManualModeTask** sẽ được kích hoạt. Tác vụ này chịu trách nhiệm xử lý các yêu cầu và hành động từ người dùng.

vAutoModeTask:

Nếu hệ thống đang ở chế độ **Tự động**, tác vụ **vAutoModeTask** sẽ được kích hoạt. Tác vụ này sẽ thực hiện tự động các quy trình điều khiển đèn giao thông theo các trạng thái đã lập trình sẵn, bao gồm việc thay đổi đèn tín hiệu theo thời gian và các điều kiện đã định trước.



Hình 4-3 Lưu đồ giải thuật chế độ thủ công

Lưu đồ giải thuật trên mô tả quy trình hoạt động của tác vụ **vManualModeTask**, trong chế độ thủ công của hệ thống điều khiển đèn tín hiệu. Mục đích của tác vụ này là điều khiển các đèn LED (LED1 và LED2) theo các trạng thái khác nhau, với sự thay đổi màu sắc và thời gian hiển thị, dựa trên tác động của người dùng qua việc bấm nút.

Quá trình thực hiện trong **vManualModeTask** bao gồm các bước sau:

Bắt đầu (Start): Quá trình bắt đầu khi tác vụ **vManualModeTask** được kích hoạt trong hệ thống.

switch (manualCase): Tác vụ này kiểm tra giá trị của biến **manualCase**, đại diện cho trạng thái hiện tại của hệ thống. Dựa vào giá trị này, hệ thống sẽ chuyển sang một trong các trường hợp khác nhau để điều khiển màu sắc của đèn LED.

Trường hợp mnuState1:

Nếu **manualCase** có giá trị là **mnuState1**, LED1 sẽ được bật với màu xanh và LED2 với màu đỏ.

Sau đó, hệ thống kiểm tra xem người dùng có yêu cầu thay đổi màu sắc (biến **COLOR?**) hay không. Nếu không, hệ thống vẫn giữ nguyên trạng thái và tiếp tục.

Nếu người dùng yêu cầu thay đổi màu sắc (**COLOR? = Yes**), hệ thống sẽ chuyển sang trạng thái **mnuState2**.

Trường hợp mnuState2:

Khi hệ thống chuyển sang **mnuState2**, LED1 sẽ chuyển sang màu vàng và LED2 vẫn ở màu đỏ.

Hệ thống tiếp tục chờ 3 giây trước khi chuyển sang trạng thái **mnuState3**.

Nếu người dùng yêu cầu thay đổi màu sắc trước khi hết 3 giây, hệ thống sẽ ngay lập tức chuyển sang trạng thái **mnuState4**.

Trường hợp mnuState3:

Trong **mnuState3**, LED1 sẽ chuyển sang màu đỏ và LED2 sẽ chuyển sang màu xanh.

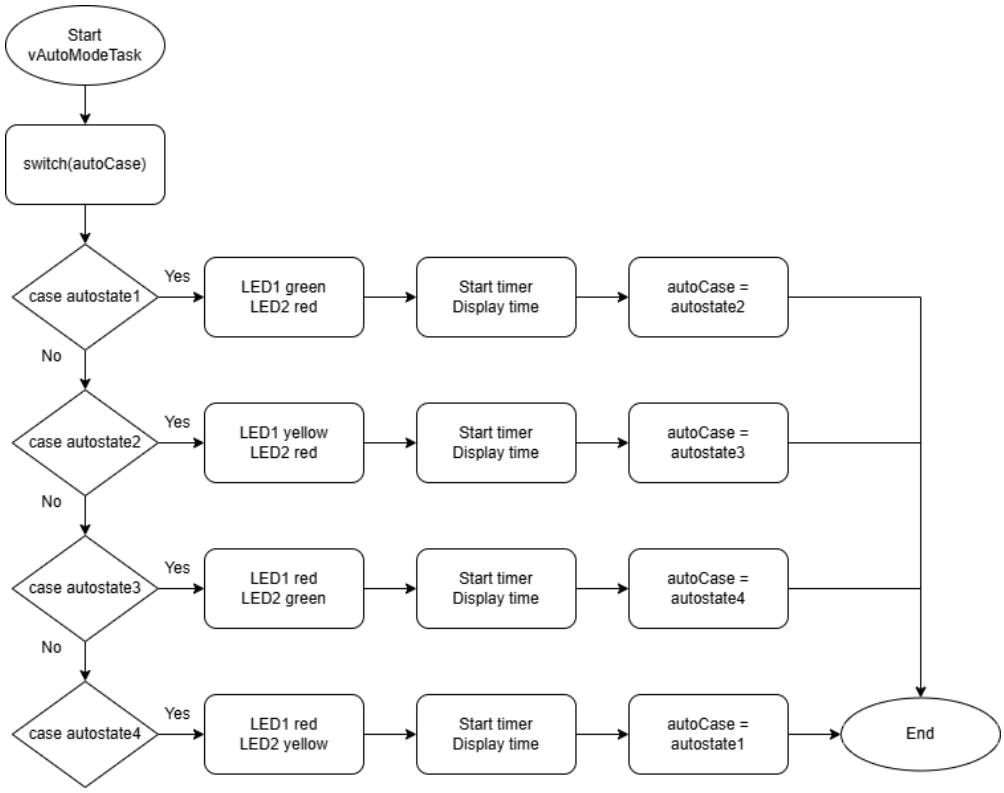
Hệ thống lại kiểm tra yêu cầu thay đổi màu sắc từ người dùng. Nếu người dùng yêu cầu thay đổi (**COLOR? = Yes**), hệ thống sẽ chuyển sang **mnuState4**. Nếu không có thay đổi, hệ thống sẽ tiếp tục.

Trường hợp mnuState4:

Tại trạng thái **mnuState4**, LED1 sẽ tiếp tục ở màu đỏ và LED2 sẽ chuyển sang màu vàng.

Sau 3 giây, hệ thống sẽ quay lại trạng thái **mnuState1** để bắt đầu lại chu trình.

Kết thúc (End): Quá trình điều khiển sẽ tiếp tục quay lại từ đầu sau khi hoàn thành một vòng lặp của các trạng thái. Tác vụ kết thúc khi hệ thống chuyển sang trạng thái ban đầu.



Hình 4-4 Lưu đồ giải thuật chế độ tự động

Lưu đồ giải thuật trên mô tả quy trình hoạt động của tác vụ **vAutoModeTask**, trong chế độ tự động của hệ thống điều khiển đèn tín hiệu. Mục đích của tác vụ này là điều khiển các đèn LED (LED1 và LED2) theo các trạng thái khác nhau, với sự thay đổi màu sắc và thời gian hiển thị, dựa trên các trạng thái được cài đặt trước.

Quá trình thực hiện trong **vAutoModeTask** bao gồm các bước sau:

Bắt đầu (Start): Quá trình bắt đầu khi tác vụ **vAutoModeTask** được kích hoạt trong hệ thống.

switch (autoCase): Tác vụ này kiểm tra giá trị của biến **autoCase**, đại diện cho trạng thái hiện tại của hệ thống. Dựa vào giá trị này, hệ thống sẽ chuyển sang một trong các trường hợp khác nhau để điều khiển màu sắc của đèn LED.

Trường hợp autoState1:

Nếu **autoCase** có giá trị là **autoState1**, LED1 sẽ được bật với màu xanh và LED2 với màu đỏ.

Sau đó, hệ thống kích hoạt timer và hiển thị thời gian ra LED 7 đoạn.

Khi hết timer hệ thống sẽ chuyển sang trạng thái **autoState2**.

Trường hợp autoState2:

Nếu **autoCase** có giá trị là **autoState2**, LED1 sẽ được bật với màu vàng và LED2 với màu đỏ.

Sau đó, hệ thống kích hoạt timer và hiển thị thời gian ra LED 7 đoạn.

Khi hết timer hệ thống sẽ chuyển sang trạng thái **autoState3**.

Trường hợp autoState3:

Nếu **autoCase** có giá trị là **autoState3**, LED1 sẽ được bật với màu đỏ và LED2 với màu xanh.

Sau đó, hệ thống kích hoạt timer và hiển thị thời gian ra LED 7 đoạn.

Khi hết timer hệ thống sẽ chuyển sang trạng thái **autoState4**.

Trường hợp autoState4:

Nếu **autoCase** có giá trị là **autoState4**, LED1 sẽ được bật với màu đỏ và LED2 với màu vàng.

Sau đó, hệ thống kích hoạt timer và hiển thị thời gian ra LED 7 đoạn.

Khi hết timer hệ thống sẽ chuyển sang trạng thái **autoState1**.

Kết thúc (End): Quá trình điều khiển sẽ tiếp tục quay lại từ đầu sau khi hoàn thành một vòng lặp của các trạng thái. Tác vụ kết thúc khi hệ thống chuyển sang trạng thái ban đầu.

5. KẾT QUẢ THỰC HIỆN

5.1. Chương trình chính của hệ thống đèn giao thông sử dụng freeRTOS

```
#include <Arduino_FreeRTOS.h>

#include <Arduino.h>

#include <task.h>

// Định nghĩa chân GPIO cho đèn LED
#define LED1_RED_PIN    16
#define LED1_YELLOW_PIN 17
#define LED1_GREEN_PIN  5
#define LED2_RED_PIN    4
#define LED2_YELLOW_PIN 0
#define LED2_GREEN_PIN  2

// Chân GPIO cho nút nhấn
#define BUTTON_PIN 14  // Chân GPIO cho nút nhấn COLOR
#define MODE_PIN 13     // Chân GPIO cho nút nhấn MODE

// Định nghĩa chế độ
#define AUTO    0
#define MANUAL  1

// Biến toàn cục để lưu trữ chế độ hiện tại
int currentMode = AUTO; // Khởi tạo ở chế độ AUTO

// Khai báo nguyên mẫu hàm vTimerCallback
void vTimerCallback(TimerHandle_t xTimer);

// Hàm callback được gọi khi timer hết hạn
void vTimerCallback(TimerHandle_t xTimer) {
  // Lấy ID của timer
  int timerID = (int)pvTimerGetTimerID(xTimer);

  // Biến static để theo dõi trạng thái đèn
  static int autoCase = 1;

  // Xử lý dựa trên ID của timer
  switch (timerID) {
    case 0: // Timer cho pha 1 (đèn 1 xanh, đèn 2 đỏ) - 30 giây
      autoCase = 2; // Chuyển sang pha 2
      xTimerStart(xTimerYellow1, 0); // Bắt đầu timer cho pha 2
      break;
    case 1: // Timer cho pha 2 (đèn 1 vàng, đèn 2 đỏ) - 5 giây
      autoCase = 3; // Chuyển sang pha 3
      xTimerStart(xTimerGreen2, 0); // Bắt đầu timer cho pha 3
```

```
      break;
    case 2: // Timer cho pha 3 (đèn 1 đỏ, đèn 2 xanh) - 30 giây
      autoCase = 4; // Chuyển sang pha 4
      xTimerStart(xTimerYellow2, 0); // Bắt đầu timer cho pha 4
      break;
    case 3: // Timer cho pha 4 (đèn 1 đỏ, đèn 2 vàng) - 5 giây
      autoCase = 1; // Chuyển sang pha 1
      xTimerStart(xTimerGreen1, 0); // Bắt đầu timer cho pha 1
      break;
  }

  // Cập nhật trạng thái đèn dựa trên autoCase
  switch (autoCase) {
    case 1:
      digitalWrite(LED1_GREEN_PIN, HIGH);
      digitalWrite(LED1_YELLOW_PIN, LOW);
      digitalWrite(LED1_RED_PIN, LOW);
      digitalWrite(LED2_GREEN_PIN, LOW);
      digitalWrite(LED2_YELLOW_PIN, LOW);
      digitalWrite(LED2_RED_PIN, HIGH);
      break;
    case 2:
      digitalWrite(LED1_GREEN_PIN, LOW);
      digitalWrite(LED1_YELLOW_PIN, HIGH);
      digitalWrite(LED1_RED_PIN, LOW);
      digitalWrite(LED2_GREEN_PIN, LOW);
      digitalWrite(LED2_YELLOW_PIN, LOW);
      digitalWrite(LED2_RED_PIN, HIGH);
      break;
    case 3:
      digitalWrite(LED1_GREEN_PIN, LOW);
      digitalWrite(LED1_YELLOW_PIN, LOW);
      digitalWrite(LED1_RED_PIN, HIGH);
      digitalWrite(LED2_GREEN_PIN, HIGH);
      digitalWrite(LED2_YELLOW_PIN, LOW);
      digitalWrite(LED2_RED_PIN, LOW);
      break;
    case 4:
      digitalWrite(LED1_GREEN_PIN, LOW);
      digitalWrite(LED1_YELLOW_PIN, LOW);
      digitalWrite(LED1_RED_PIN, HIGH);
      digitalWrite(LED2_GREEN_PIN, LOW);
      digitalWrite(LED2_YELLOW_PIN, HIGH);
      digitalWrite(LED2_RED_PIN, LOW);
      break;
  }
}

// Task quản lý chế độ (vModeManagerTask)
void vModeManagerTask(void *pvParameters) {
```

```
  // Tạo task vManualModeTask và lưu TaskHandle_t
  TaskHandle_t xManualTaskHandle;
  xTaskCreate(vManualModeTask, "ManualModeTask", 1024, NULL, 1,
&xManualTaskHandle);

  // Tạo task vAutoModeTask và lưu TaskHandle_t
  TaskHandle_t xAutoTaskHandle;
  xTaskCreate(vAutoModeTask, "AutoModeTask", 1024, NULL, 1, &xAutoTaskHandle);
  vTaskSuspend(xAutoTaskHandle); // Suspend vAutoModeTask ban đầu

  for (;;) {
    // Đọc giá trị MODE từ nút nhấn
    int MODE = digitalRead(MODE_PIN);

    if (MODE) { // Nếu MODE thay đổi
      if (currentMode == AUTO) {
        currentMode = MANUAL;
        // Tắt tất cả đèn LED
        digitalWrite(LED1_RED_PIN, LOW);
        digitalWrite(LED1_YELLOW_PIN, LOW);
        digitalWrite(LED1_GREEN_PIN, LOW);
        digitalWrite(LED2_RED_PIN, LOW);
        digitalWrite(LED2_YELLOW_PIN, LOW);
        digitalWrite(LED2_GREEN_PIN, LOW);

        // Resume manual task sử dụng TaskHandle_t
        vTaskResume(xManualTaskHandle);

        // Suspend auto task sử dụng TaskHandle_t
        vTaskSuspend(xAutoTaskHandle);
      } else {
        currentMode = AUTO;

        // Suspend manual task sử dụng TaskHandle_t
        vTaskSuspend(xManualTaskHandle);

        // Resume auto task sử dụng TaskHandle_t
        vTaskResume(xAutoTaskHandle);
      }
    }

        vTaskDelay(pdMS_TO_TICKS(100)); // Kiểm tra MODE mỗi 100ms
  }
}

// Task cho chế độ thủ công (vManualModeTask)
void vManualModeTask(void *pvParameters) {
  int manualCase = 1; // Khởi tạo trạng thái ban đầu
  int COLOR; // Biến để lưu giá trị nút nhấn
```

```
  for (;;) {
    switch (manualCase) {
      case 1:
        digitalWrite(LED1_GREEN_PIN, HIGH);
        digitalWrite(LED2_RED_PIN, HIGH);
        // Đọc giá trị COLOR từ nút nhấn
        COLOR = !digitalRead(BUTTON_PIN); // Đảo ngược giá trị đọc được
        if (COLOR) {
          manualCase = 2;
        }
        break;
      case 2:
        digitalWrite(LED1_YELLOW_PIN, HIGH);
        digitalWrite(LED2_RED_PIN, HIGH);
        vTaskDelay(pdMS_TO_TICKS(3000)); // Delay 3s
        manualCase = 3;
        break;
      case 3:
        digitalWrite(LED1_RED_PIN, HIGH);
        digitalWrite(LED2_GREEN_PIN, HIGH);
        // Đọc giá trị COLOR từ nút nhấn
        COLOR = !digitalRead(BUTTON_PIN); // Đảo ngược giá trị đọc được
        if (COLOR) {
          manualCase = 4;
        }
        break;
      case 4: // Chuyển đèn 2 sang vàng, rồi đỏ, sau đó đèn 1 sang xanh
        digitalWrite(LED2_YELLOW_PIN, HIGH);
        vTaskDelay(pdMS_TO_TICKS(3000)); // Delay 3s
        digitalWrite(LED2_YELLOW_PIN, LOW);
        digitalWrite(LED2_RED_PIN, HIGH);
        digitalWrite(LED1_RED_PIN, LOW);
        digitalWrite(LED1_GREEN_PIN, HIGH);
        manualCase = 1;
        break;
    }
    vTaskDelay(pdMS_TO_TICKS(100)); // Delay 100ms để nhường CPU
  }
}

// Tạo các timer
TimerHandle_t xTimerGreen1 = xTimerCreate("TimerGreen1",
pdMS_TO_TICKS(30000), pdFALSE, (void *)0, vTimerCallback);
TimerHandle_t xTimerYellow1 = xTimerCreate("TimerYellow1",
pdMS_TO_TICKS(5000), pdFALSE, (void *)1, vTimerCallback);
TimerHandle_t xTimerGreen2 = xTimerCreate("TimerGreen2",
pdMS_TO_TICKS(30000), pdFALSE, (void *)2, vTimerCallback);
TimerHandle_t xTimerYellow2 = xTimerCreate("TimerYellow2",
pdMS_TO_TICKS(5000), pdFALSE, (void *)3, vTimerCallback);
```

```
// Task cho chế độ tự động (vAutoModeTask)
void vAutoModeTask(void *pvParameters) {
  // Bắt đầu timer cho pha 1
  xTimerStart(xTimerGreen1, 0);

  for (;;) {

    vTaskDelay(pdMS_TO_TICKS(1000)); // Delay 1 giây để nhường CPU
  }
}

void setup() {
  Serial.begin(115200);

  // Khởi tạo các chân GPIO cho đèn LED
  pinMode(LED1_RED_PIN, OUTPUT);
  pinMode(LED1_YELLOW_PIN, OUTPUT);
  pinMode(LED1_GREEN_PIN, OUTPUT);
  pinMode(LED2_RED_PIN, OUTPUT);
  pinMode(LED2_YELLOW_PIN, OUTPUT);
  pinMode(LED2_GREEN_PIN, OUTPUT);

  // Khởi tạo chân nút nhấn với điện trở kéo lên
  pinMode(BUTTON_PIN, INPUT_PULLUP);  // Cho nút nhấn COLOR
  pinMode(MODE_PIN, INPUT_PULLUP);     // Cho nút nhấn MODE

  // Tạo task vModeManagerTask
  xTaskCreate(vModeManagerTask, "ModeManagerTask", 1024, NULL, 1, NULL);
}

void loop() {

}
```

5.2. Kết quả chạy thực tế của hệ thống

6. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

6.1 Kết luận

Đề tài *"Lập trình mạch điều khiển đèn giao thông sử dụng FreeRTOS"* đã hoàn thành các mục tiêu đặt ra, bao gồm nghiên cứu lý thuyết, thiết kế phần cứng, triển khai phần mềm, và kiểm thử hệ thống.

Về mặt ưu điểm, hệ thống đạt được mục tiêu thiết kế ban đầu, vận hành ổn định và dễ dàng mở rộng. Việc sử dụng FreeRTOS giúp tối ưu hóa tài nguyên phần cứng, đảm bảo

tính linh hoạt và khả năng đáp ứng theo thời gian thực. Hệ thống được thiết kế với chi phí thấp, dễ dàng triển khai với quy mô lớn hơn trong các mô hình giao thông.

Ngoài ra vẫn còn tồn tại những hạn chế như phạm vi ứng dụng còn hạn chế, chưa tích hợp các cảm biến giao thông thực tế để phát hiện tình trạng giao thông. Giao diện điều khiển chưa thân thiện và chưa có khả năng giám sát từ xa thông qua kết nối mạng. Hệ thống chưa được thử nghiệm trong điều kiện thực tế tại các nút giao thông phức tạp.

Từ đó nhóm rút ra kinh nghiệm cần hiểu rõ nền tảng lý thuyết và tiêu chuẩn kỹ thuật là bước quan trọng để đảm bảo tính chính xác trong thiết kế. Ngoài ra việc lựa chọn phần cứng phù hợp và tối ưu phần mềm giúp tăng hiệu quả triển khai, đặc biệt khi sử dụng các hệ điều hành thời gian thực như FreeRTOS. Cuối cùng quá trình kiểm thử thực tế là cần thiết để đánh giá toàn diện và khắc phục các hạn chế của hệ thống.

6.2 Hướng phát triển

Dựa trên những kết quả đạt được và các hạn chế còn tồn tại, đề tài có thể được phát triển theo nhiều hướng khác nhau ví dụ như mở rộng khả năng phát hiện giao thông bằng việc tích hợp các cảm biến thực tế (như cảm biến hồng ngoại, camera AI) để phát hiện mật độ phương tiện, từ đó điều chỉnh luồng đèn giao thông tự động theo thời gian thực, kết nối IoT và giám sát từ xa với việc ứng dụng giao thức MQTT hoặc HTTP để giám sát và điều khiển hệ thống từ xa thông qua một ứng dụng di động hoặc nền tảng đám mây, tối ưu thuật toán điều khiển bằng cách sử dụng các thuật toán học máy hoặc tối ưu hóa để nâng cao hiệu quả điều khiển giao thông tại các nút giao phức tạp,…

Những hướng phát triển trên không chỉ mở rộng phạm vi ứng dụng của hệ thống mà còn góp phần nâng cao hiệu quả quản lý giao thông, hướng tới một hệ thống giao thông thông minh và bền vững.

7. TÀI LIỆU THAM KHẢO

[1] freeRTOS.org, freeRTOS documents, FreeRTOS documentation - FreeRTOS™

[2] Qing Li, Caroline Yao, "Real-Time Concepts for Embedded Systems", CMP Books, 2003.

[3] Richard Barry, "Mastering the FreeRTOS Real Time Kernel", FreeRTOS Documentation - FreeRTOS™