**HỌC VIỆN KỸ THUẬT MẬT MÃ**

**KHOA ĐIỆN TỬ - VI MẠCH**

-----&-----



**BÁO CÁO MÔN HỌC**

**HỆ THỐNG NHÚNG**

**ỨNG DỤNG GIAO THỨC SPI TRONG GIAO TIẾP**

**STM32 VỚI MÀN HÌNH TFT**

Giảng viên hướng dẫn: ThS Nguyễn Thanh Ngọc

Sinh viên thực hiện: Hà Văn Đạt – DT060209

Dương Hải Đăng – DT060206

Nguyễn Mạnh Lân – DT060231

Nguyễn Quang Huy – DT060225

Hà Nội – 2025

LỜI NÓI ĐẦU

Trong thời đại công nghệ phát triển mạnh mẽ hiện nay, các hệ thống nhúng ngày càng giữ vai trò quan trọng trong nhiều lĩnh vực như điện tử tiêu dùng, tự động hóa, điều khiển công nghiệp và Internet of Things (IoT). Việc hiểu và ứng dụng các giao thức truyền thông trong hệ thống nhúng là nền tảng quan trọng giúp sinh viên có thể thiết kế và phát triển các thiết bị thông minh một cách hiệu quả.

Đề tài **“Ứng dụng giao thức SPI trong giao tiếp STM32 với màn hình TFT”** được thực hiện nhằm mục đích tìm hiểu và áp dụng giao thức truyền thông nối tiếp SPI – một trong những chuẩn giao tiếp phổ biến và hiệu quả trong các hệ thống nhúng. Thông qua đề tài, chúng em tiến hành nghiên cứu nguyên lý hoạt động của SPI, cách cấu hình và lập trình trên vi điều khiển STM32, đồng thời triển khai việc hiển thị dữ liệu lên màn hình TFT nhằm minh họa tính ứng dụng thực tế của giao thức này. Báo cáo này không chỉ giúp củng cố kiến thức lý thuyết đã học mà còn nâng cao kỹ năng thực hành, tư duy lập trình và khả năng thiết kế hệ thống của sinh viên. Hy vọng rằng kết quả của đề tài sẽ là nền tảng hữu ích cho những nghiên cứu và ứng dụng sâu hơn trong lĩnh vực hệ thống nhúng sau này.

Trước hết, chúng em xin gửi lời cảm ơn chân thành đến **cô Nguyễn Thanh Ngọc**, giảng viên bộ môn **Hệ thống nhúng**, người đã tận tình hướng dẫn, truyền đạt những kiến thức quý báu và luôn tạo điều kiện thuận lợi để chúng em hoàn thành tốt đề tài này. Chúng em cũng xin cảm ơn các thầy cô trong khoa, cùng bạn bè đã hỗ trợ, đóng góp ý kiến và giúp đỡ trong quá trình tìm hiểu, thực hiện và hoàn thiện báo cáo.

Mặc dù chúng em đã cố gắng hết sức, song do kiến thức và kinh nghiệm còn hạn chế, bài báo cáo không tránh khỏi thiếu sót. Chúng em rất mong nhận được sự thông cảm và góp ý của cô để đề tài được hoàn thiện hơn.

Chúng em chân thành cảm ơn!

MỤC LỤC

[CHƯƠNG 1 : CƠ SỞ LÝ THUYẾT 1](#_Toc211154276)

[1.1. Giới thiệu về Vi điều khiển STM32 1](#_Toc211154277)

[1.1.1. Tổng quan 1](#_Toc211154278)

[1.1.2. Dòng vi điều khiển được sử dụng 1](#_Toc211154279)

[1.1.3. Môi trường lập trình và thư viện HAL 1](#_Toc211154280)

[1.2. Giao thức truyền thông SPI (Serial Peripheral Interface) 2](#_Toc211154281)

[1.2.1. Khái niệm 2](#_Toc211154282)

[1.2.2. Nguyên lý hoạt động 5](#_Toc211154283)

[1.2.3. Ưu và nhược điểm 6](#_Toc211154284)

[1.3. Màn hình TFT LCD 6](#_Toc211154285)

[1.3.1. Giới thiệu 6](#_Toc211154286)

[1.3.2. Driver điều khiển 7](#_Toc211154287)

[1.3.3. Giao tiếp với Vi điều khiển 7](#_Toc211154288)

[1.3.4. Giao thức lệnh và dữ liệu 7](#_Toc211154289)

[CHƯƠNG 2: CÔNG NGHỆ VÀ CÔNG CỤ THIẾT KẾ 8](#_Toc211154290)

[2.1. Sơ đồ khối của mạch 8](#_Toc211154291)

[2.2. Phần cứng 9](#_Toc211154292)

[2.2.1. Vi điều khiển 9](#_Toc211154293)

[2.2.2. Màn hình TFT 9](#_Toc211154294)

[2.2.3. Các linh kiện khác 9](#_Toc211154295)

[2.2.4. Sơ đồ kết nối 10](#_Toc211154296)

[2.3. Phần mềm 11](#_Toc211154297)

[2.3.1. STM32CubeMX 11](#_Toc211154298)

[2.3.2. Keil C (Keil µVision ) 12](#_Toc211154299)

[2.3.3. Ngôn ngữ lập trình C 12](#_Toc211154300)

[2.3.4. Thư viện HAL (Hardware Abstraction Layer) 12](#_Toc211154301)

[2.3.5. Công cụ chuyển đổi hình ảnh 13](#_Toc211154302)

[CHƯƠNG 3 : THIẾT KẾ VÀ THI CÔNG 14](#_Toc211154303)

[3.1. Sơ đồ nguyên lý 14](#_Toc211154304)

[3.2. Thuật toán và mô tả chương trình 14](#_Toc211154305)

[3.2.1. Khởi tạo và giao tiếp màn hình TFT 15](#_Toc211154306)

[3.2.2. Các hàm điều khiển cơ bản 18](#_Toc211154307)

[3.2.3.Các hàm hiển thị ký tự và hình ảnh 20](#_Toc211154308)

[3.2.4. Thuật toán và trình tự hoạt động 24](#_Toc211154309)

[3.3. Thực hiện nạp và khởi chạy chương trình trên vi điều khiển 26](#_Toc211154310)

[3.3.1. Cấu hình trên STM32cubeMX 26](#_Toc211154311)

[3.3.2. Tạo File ảnh hiển thị 29](#_Toc211154312)

[3.3.3. Viết chương trình và nạp code 30](#_Toc211154313)

[3.4 Sản phẩm thực tế 32](#_Toc211154314)

[3.4.1. Sản phẩm khi chưa cấp nguồn 32](#_Toc211154315)

[3.4.2. Sản phẩm thực tế khi đã cấp nguồn 32](#_Toc211154316)

[KẾT LUẬN 33](#_Toc211154317)

[1. Ưu điểm 33](#_Toc211154318)

[2. Nhược điểm 33](#_Toc211154319)

[3. Phương hướng phát triển 33](#_Toc211154320)

[TÀI LIỆU THAM KHẢO 35](#_Toc211154321)

[PHỤ LỤC 36](#_Toc211154322)

DANH MỤC TỪ VIẾT TẮT

|  |  |
| --- | --- |
| ARM | Advanced RISC Machine |
| CS | Chip Select |
| HAL | Hardware Abstraction Layer |
| LCD | Liquid Crystal Display |
| MCU | Microcontroller Unit |
| MISO | Master In Slave Out |
| MOSI | Master Out Slave In |
| SPI | Serial Peripheral Interface |
| TFT | Thin Film Transistor |

DANH MỤC HÌNH ẢNH

[Hình 1.1: STM32F103C8T6 7](#_Toc211128873)

[Hình 1.2: Sơ đồ kết nối giao tiếp SPI 9](#_Toc211128874)

[Hình 1.3:Nguyên lý truyền nhận dữ liệu trong SPI 9](#_Toc211128875)

[Hình 1.4: Các chế độ trong SPI 10](#_Toc211128876)

[Hình 1.5:Cấu hình master và các slave độc lập 10](#_Toc211128877)

[Hình 1.6: Cấu hình Daisy Chain 11](#_Toc211128878)

[Hình 1.7 : LCD TFT ST7735 12](#_Toc211128879)

[Hình 2.1: Sơ đồ khối 14](#_Toc211128880)

[Hình 2.2:Sơ đồ nguyên lý của TTP223B 15](#_Toc211128881)

[Hình 2.3: Phần mềm STM32CubeMX 17](#_Toc211128882)

[Hình 2.4: KeilC 18](#_Toc211128883)

[Hình 3.1: Sơ đồ nguyên lý 20](#_Toc211128884)

[Hình 3.2: Thuật toán khởi tạo hiển thị màn hình TFT 21](#_Toc211128885)

[Hình 3.3: Sơ đồ giải thuật chi tiết chuỗi thiết lập nguồn 22](#_Toc211128886)

[Hình 3.4: Sơ đồ giải thuật gửi lệnh và dữ liệu đến màn hình TFT 23](#_Toc211128887)

[Hình 3.5: Các lệnh khởi tạo màn hình ST7735 24](#_Toc211128888)

[Hình 3.6: Sơ đồ thuật toán hàm drawPixel() 25](#_Toc211128889)

[Hình 3.7:Thuật toán mô tả hàm setPos() 26](#_Toc211128890)

[Hình 3.8: Sơ đồ thuật toán hmaf drawChar() 27](#_Toc211128891)

[Hình 3.9: Thuật toán mô tả hàm drawString() 28](#_Toc211128892)

[Hình 3.10: Sơ đồ lưu đồ thuật toán hàm drawImage() 29](#_Toc211128893)

[Hình 3.11: Sơ đồ giải thuật hàm fullDisplay(); 30](#_Toc211128894)

[Hình 3.11: Lưu đồ thuật toán của hệ thống 31](#_Toc211128895)

[Hình 3.12: Cấu hình cho SerialWire 32](#_Toc211128896)

[Hình 3.13: Chọn thạch anh ngoài 33](#_Toc211128897)

[Hình 3.14: Cấu hình chân GPIO 33](#_Toc211128898)

[Hình 3.15: Cài đặt tần số 34](#_Toc211128899)

[Hình 3.16: Cài đặt giao tiếp SPI 34](#_Toc211128900)

[Hình 3.17: Generatecode 35](#_Toc211128901)

[Hình 3.18: Tạo ảnh hiển thị 35](#_Toc211128902)

[Hình 3.19: Chèn thư viện ảnh 36](#_Toc211128903)

[Hình 3.20: Khai báo thư viện 36](#_Toc211128904)

[Hình 3.21: Viết chương trình con 36](#_Toc211128905)

[Hình 3.22: Chương trình chính của hệ thống 37](#_Toc211128906)

[Hình 3.23: Nạp chương trình 37](#_Toc211128907)

[Hình 3.24: Sản phẩm khi chưa cấp nguồn 38](#_Toc211128908)

[Hình 3.25:Màn hình TFT hiện chuỗi ký tự 38](#_Toc211128909)

[Hình 3.26:Màn hình TFT hiện hình ảnh 38](#_Toc211128910)

DANH MỤC BẢNG

[Bảng 1.1: Các đường tín hiệu 8](#_Toc211129029)

[Bảng 1.2: Các chế độ SPI 10](#_Toc211129030)

[Bảng 1.3:Ưu nhược điểm của SPI 12](#_Toc211129031)

[Bảng 2.1: Trạng thái hoạt động của TTP223B 16](#_Toc211129032)

[Bảng 2.2: Sơ đồ kết nối chân 16](#_Toc211129033)

CHƯƠNG 1 : CƠ SỞ LÝ THUYẾT

1.1. Giới thiệu về Vi điều khiển STM32

1.1.1. Tổng quan

STM32 là họ vi điều khiển 32-bit hàng đầu do công ty STMicroelectronics sản xuất. Dòng sản phẩm này được xây dựng dựa trên kiến trúc bộ xử lý ARM Cortex-M, nổi bật với sự đa dạng về hiệu năng (từ các dòng tốc độ thấp, siêu tiết kiệm điện đến các dòng hiệu năng cao) và một hệ sinh thái công cụ phát triển mạnh mẽ. STM32 được sử dụng rộng rãi trong các ứng dụng IoT, thiết bị y tế, tự động hóa công nghiệp và điện tử tiêu dùng nhờ sự cân bằng giữa hiệu suất và giá thành.

1.1.2. Dòng vi điều khiển được sử dụng

Dự án này sử dụng vi điều khiển STM32F103C8T6, thuộc dòng STM32F1 Series (Dòng Performance Line). Đây là một dòng chip phổ biến, thường được tích hợp trên các module phát triển giá rẻ như "Blue Pill".



Hình 1.1: STM32F103C8T6

1.1.3. Môi trường lập trình và thư viện HAL

Môi trường Phát triển Tích hợp (IDE): Sử dụng STM32CubeIDE, một công cụ chính thức và miễn phí từ STMicroelectronics. IDE này tích hợp các chức năng cấu hình dự án (STM32CubeMX), phát triển mã nguồn, biên dịch và gỡ lỗi.

Thư viện HAL (Hardware Abstraction Layer): HAL là một lớp trừu tượng hóa phần cứng, cung cấp các hàm API cấp cao, dễ sử dụng (ví dụ: HAL\_SPI\_Transmit(), HAL\_GPIO\_WritePin()) để thao tác với các ngoại vi.

Vai trò: Giúp lập trình viên không cần tương tác trực tiếp với các thanh ghi phần cứng phức tạp. Quan trọng hơn, HAL giúp mã nguồn dễ dàng chuyển đổi (porting) giữa các dòng vi điều khiển STM32 khác nhau (ví dụ: từ F1 sang F4 hoặc L4), tiết kiệm thời gian và công sức phát triển.

1.2. Giao thức truyền thông SPI (Serial Peripheral Interface)

1.2.1. Khái niệm

SPI (**Serial Peripheral Interface**) là một chuẩn truyền thông nối tiếp đồng bộ dùng để truyền dữ liệu ở chế độ **song công toàn phần (full duplex)**.

Giao thức SPI thường được sử dụng **On-Board** hoặc với các đường tín hiệu ngắn. Tốc độ truyền của SPI khá cao và phụ thuộc vào tốc độ xung nhịp được cung cấp cho bộ SPI.

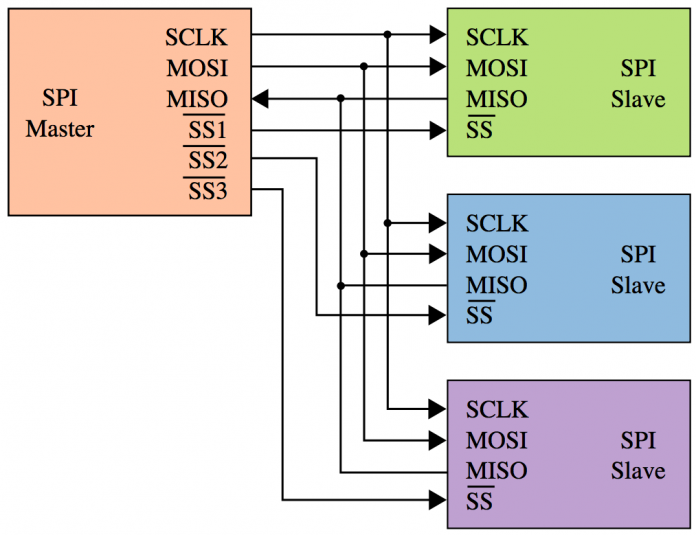
SPI là một giao thức truyền thông theo mô hình **Master – Slave**, trong đó Master giữ quyền điều khiển xung Clock và quyết định chọn Slave nào sẽ giao tiếp với mình.

Bus SPI bao gồm **4 đường tín hiệu chính**:

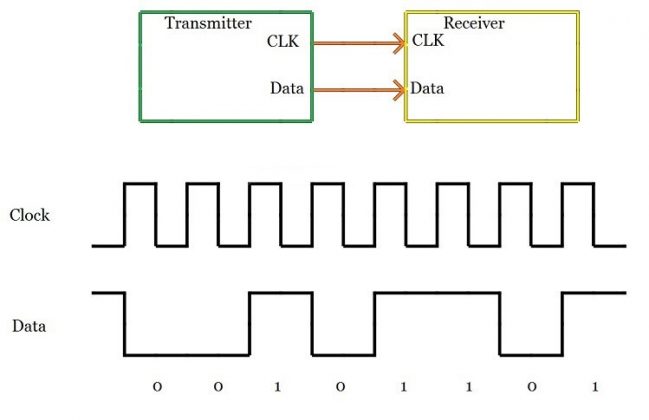
Bảng 1.1: Các đường tín hiệu

|  |  |  |
| --- | --- | --- |
| Tên tín hiệu | Chức năng | Vai trò |
| SCLK | Xung nhịp đồng bộ | Được Master tạo ra để đồng bộ hoá quá trình truyền/nhận dữ liệu |
| MOSI | Dữ liệu Master -> Slave | Đường truyền dữ liệu từ vi điều khiển đến màn hình TFT |
| MISO | Dữ liệu Slave -> Master | Đường truyền dữ liệu từ màn hình TFT về vi điều khiển |
| CS | Chọn Slave | Tín hiệu được Master kéo xuống LOW để chọn duy nhất một Slave giao tiếp trên bus |

Sơ lược về guyên lý hoạt động của giao thức SPI: Khi muốn truyền nhận dữ liệu tới các Slave, Master sẽ kéo đường CS kết nối từ Master tới Slave đó xuống mức 0, sau đó gửi xung Clock và với mỗi xung Clock sẽ truyền dữ liệu trên chân MOSI tại thời điểm Clock ở mức cao (hoặc thấp tùy theo cách lập trình). Đồng thời, Slave cũng có thể gửi dữ liệu ngược lại qua chân MISO về Master. Quá trình truyền nhận dữ liệu diễn ra liên tục nên SPI thường đạt tốc độ rất cao



Hình 1.2: Sơ đồ kết nối giao tiếp SPI

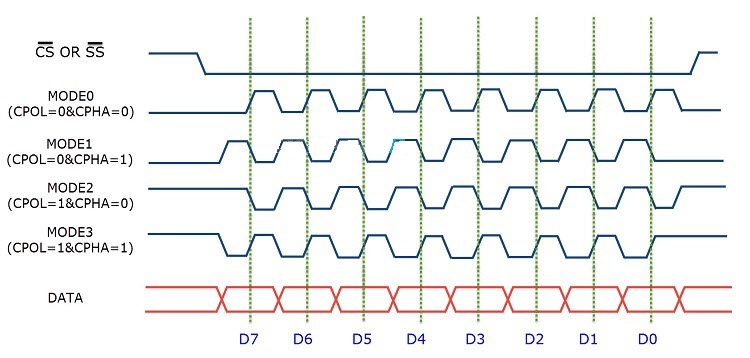


Hình 1.3:Nguyên lý truyền nhận dữ liệu trong SPI

Có 4 chế độ hoạt động của SPI dựa trên hai bit **CPOL** và **CPHA**:

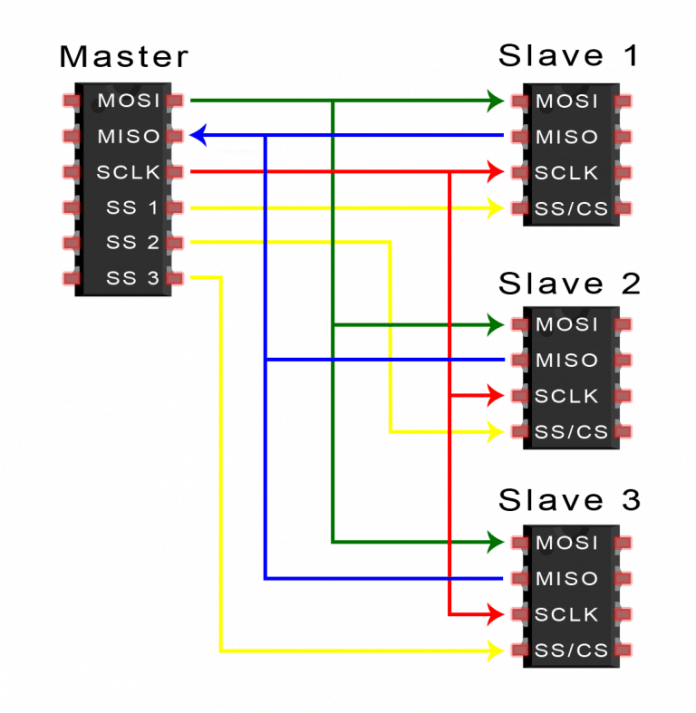
Bảng 1.2: Các chế độ SPI

|  |  |  |  |
| --- | --- | --- | --- |
| Chế độ | CPOL | CPHA | Cạnh lấy mẫu dữ liệu |
| Mode 0 | 0 | 0 | Truyền dữ liệu tại cạnh lên của xung Clock. |
| Mode 1 | 0 | 1 | Truyền dữ liệu tại cạnh xuống của xung Clock. |
| Mode 2 | 1 | 0 | Truyền dữ liệu tại cạnh lên của xung Clock. |
| Mode 3 | 1 | 1 | Truyền dữ liệu tại cạnh xuống của xung Clock |



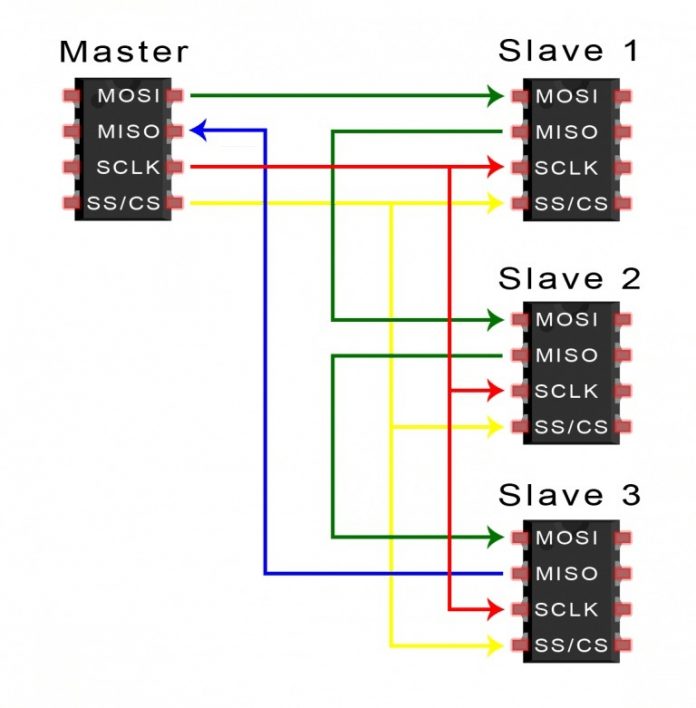
Hình 1.4: Các chế độ trong SPI

Có 2 loại cấu hình SPI, loại đầu tiên là cấu hình Master và các slave độc lập (Independent Slave Configuration): Trong cấu hình này, Master sẽ dành riêng các đường Slave Select cho tất cả các Slave, và mỗi Slave có thể được chọn riêng lẻ. Tất cả tín hiệu đồng hồ của các Slave được kết nối chung với SCK của Master



Hình 1.5:Cấu hình master và các slave độc lập

Loại thứ 2 là cấu hình Daisy Chain(Daisy Chain Configuration) : Trong cấu hình này, chỉ có một đường Slave Select được kết nối với tất cả các Slave. MOSI của Master được nối với MOSI của Slave 1, MISO của Slave 1 được nối với MOSI của Slave 2, và cứ tiếp tục như vậy. MISO của Slave cuối cùng được kết nối trở lại với MISO của Master



Hình 1.6: Cấu hình Daisy Chain

1.2.2. Nguyên lý hoạt động

Khởi tạo: Master (STM32) cấu hình tốc độ SPI, CPOL/CPHA và kéo chân CS của Slave xuống LOW để bắt đầu giao tiếp.

Truyền/Nhận đồng thời: Dữ liệu được truyền theo từng bit. Mỗi thiết bị (Master và Slave) đều có một Thanh ghi Dịch 8-bit hoặc 16-bit.

Khi xung SCLK xuất hiện, Master đồng thời đặt một bit dữ liệu lên MOSI và đọc một bit từ MISO.

Sau một chu kỳ 8 xung SCLK, 8 bit dữ liệu đã được truyền đi qua MOSI và 8 bit dữ liệu khác (hoặc giá trị rác) đã được nhận về qua MISO.

Chế độ Hoạt động (CPOL và CPHA): SPI có 4 chế độ (Mode 0-3) quy định cách Master và Slave tương tác với xung nhịp:

CPOL (Clock Polarity): Quy định trạng thái của SCLK khi IDLE (không truyền). CPOL=0: IDLE ở mức LOW. CPOL=1: IDLE ở mức HIGH.

CPHA (Clock Phase): Quy định thời điểm lấy mẫu dữ liệu. CPHA=0: Dữ liệu được lấy mẫu (sampled) ở cạnh xung nhịp thứ nhất (ví dụ: cạnh lên nếu CPOL=0). CPHA=1: Dữ liệu được lấy mẫu ở cạnh xung nhịp thứ hai.

1.2.3. Ưu và nhược điểm

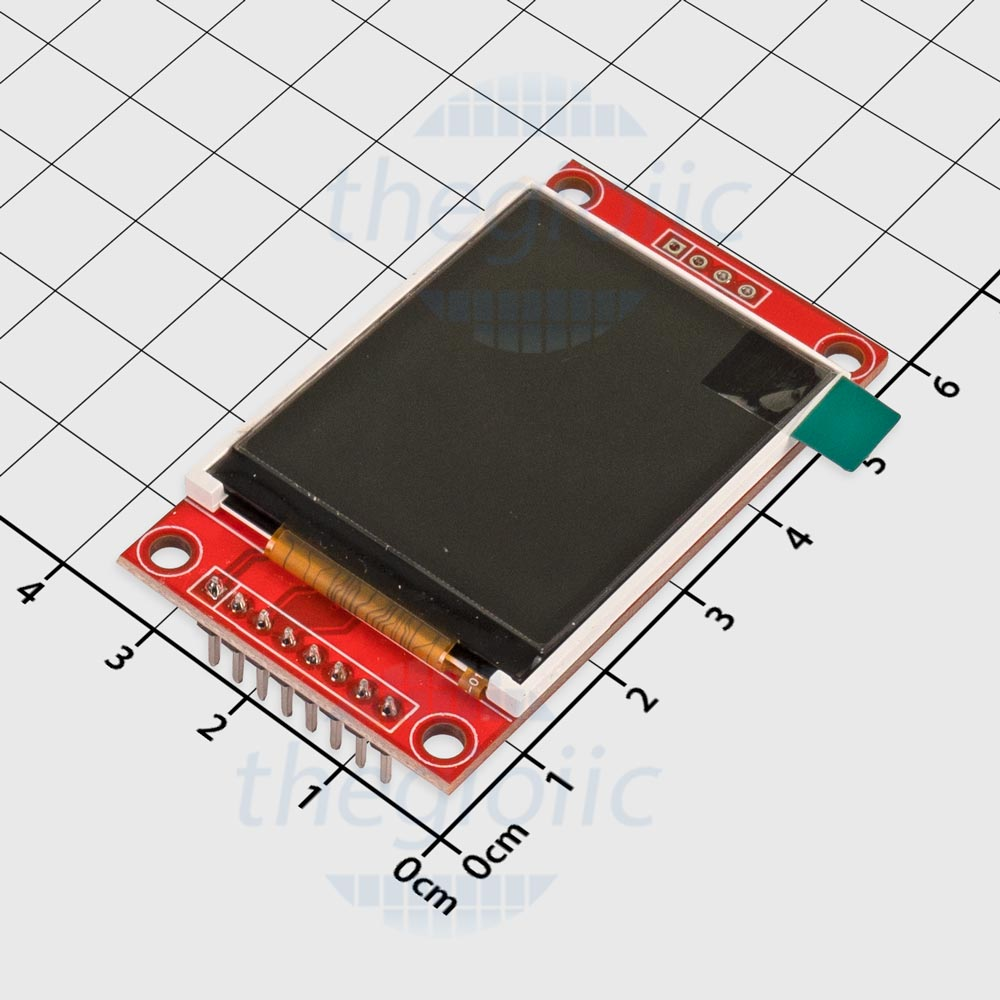
Bảng 1.3:Ưu nhược điểm của SPI

|  |  |
| --- | --- |
| Ưu điểm | Nhược điểm |
| Tốc độ cao : Dễ dàng đạt tốc độ vài chục MHz | Nhiều chân tín hiệu : Cần tối thiểu 4 dây (SCLK,MOSI,MISO,CS), nhiều hơn so với I2C 2 dây |
| Full-Duplex : Truyền và nhận dữ liệu diễn ra đồng thời | Không có cơ chế xác nhận : Slave không gửi tín hiệu ACK để xác nhận dữ liệu thành công, cần cơ chế kiểm tra lỗi ở tầng cao hơn |
| Giao thức đơn giản : Không yêu cầu cơ chế địa chỉ phức tạp | Chỉ giao tiếp tầm ngắn : Không phù hợp giao tiếp đường dài |

1.3. Màn hình TFT LCD

1.3.1. Giới thiệu

TFT LCD (Thin-Film-Transistor Liquid-Crystal Display) là một công nghệ màn hình tinh thể lỏng sử dụng ma trận các bóng bán dẫn màng mỏng (TFT) để điều khiển độc lập từng điểm ảnh (pixel). Điều này giúp cải thiện đáng kể tốc độ phản hồi, độ tương phản và góc nhìn so với các loại LCD ma trận thụ động cũ.



Hình 1.7 : LCD TFT ST7735

1.3.2. Driver điều khiển

Hầu hết các module TFT đều tích hợp một IC Driver (bộ điều khiển) như ST7735 (được suy luận từ mã nguồn), ILI9341, hoặc ILI9488.

Vai trò của Driver:

Nhận và thực thi Lệnh từ vi điều khiển (ví dụ: lệnh khởi tạo, lệnh đặt vùng vẽ).

Quản lý bộ nhớ GRAM (Graphic RAM) để lưu trữ trạng thái màu của từng pixel.

Chuyển đổi dữ liệu màu nhận được thành tín hiệu điều khiển điện áp chính xác để sắp xếp các tinh thể lỏng, qua đó tái tạo hình ảnh.

1.3.3. Giao tiếp với Vi điều khiển

Các phương thức giao tiếp phổ biến bao gồm:

Giao tiếp Song song (8-bit/16-bit): Tốc độ rất cao nhưng cần một số lượng lớn chân GPIO chuyên dụng (tối thiểu 12-20 chân).

Giao tiếp Nối tiếp SPI: Tiết kiệm chân (chỉ cần 4-5 chân) và cung cấp tốc độ đủ dùng cho màn hình nhỏ (1.44 inch, 1.8 inch, 2.4 inch). Dự án này sử dụng phương thức SPI.

1.3.4. Giao thức lệnh và dữ liệu

Để điều khiển màn hình, vi điều khiển phải phân biệt rõ hai loại thông tin khi truyền qua SPI:

Lệnh (Command): Là các byte có chức năng cấu hình hoặc ra lệnh (ví dụ: 0x2A để đặt địa chỉ cột, 0x29 để bật màn hình).

Dữ liệu (Data): Là các byte đi kèm lệnh, chứa tham số cho lệnh (ví dụ: tọa độ X, Y) hoặc là thông tin màu sắc (pixel color data).

Sự phân biệt này được thực hiện bằng chân D/C (Data/Command) hay A0:

D/C = LOW (0): Byte tiếp theo được truyền là Lệnh.

D/C = HIGH (1): Byte tiếp theo được truyền là Dữ liệu (Tham số lệnh hoặc dữ liệu màu).

CHƯƠNG 2: CÔNG NGHỆ VÀ CÔNG CỤ THIẾT KẾ

2.1. Sơ đồ khối của mạch

Hình dưới đây là sơ đồ khối của mạch:



Hình 2.1: Sơ đồ khối

Mạch bao gồm 4 khối đó là khối nguồn, khối đầu vào, khối xử lý và khối đầu ra. Khối nguồn sử dụng dây nạp ST link V2 để cấp nguồn hoặc các nguồn 5V được cấp qua cổng micro (hoặc type C) của vi điều khiển. Khối đầu vào là nút nhấn điều khiển (TTP223). Khối xử lý xử dụng STM32F103C8T6 và khối đầu ra là màn hình TFT.

Trong hệ thống “Ứng dụng giao thức SPI trong giao tiếp STM32 với màn hình TFT”, vi điều khiển STM32 đóng vai trò trung tâm xử lý. Tất cả dữ liệu cần hiển thị sẽ được STM32 truyền đến màn hình TFT thông qua giao thức SPI. Ngoài ra, hệ thống còn bao gồm các khối phụ trợ như khối nguồn, mạch reset, bộ nạp chương trình.

Nguyên lý hoạt động tổng quát:

Máy tính cá nhân được sử dụng để viết mã chương trình bằng KeilC, cấu hình bằng STM32CubeMX và nạp chương trình xuống vi điều khiển thông qua ST-Link.

Vi điều khiển STM32 sau khi được lập trình sẽ khởi tạo các ngoại vi cần thiết (SPI, GPIO, Timer,…).

Dữ liệu (chữ, số, hình ảnh) được xử lý trong STM32 và truyền qua giao tiếp SPI đến màn hình TFT.

Màn hình TFT nhận dữ liệu, giải mã và thực hiện hiển thị theo yêu cầu.

2.2. Phần cứng

2.2.1. Vi điều khiển

Kit/Module: Blue Pill Module sử dụng chip STM32F103C8T6.

Tác dụng: Đóng vai trò Master, thực hiện logic điều khiển, tạo xung Clock SPI và cung cấp các tín hiệu điều khiển (CS, DC, RST) cho màn hình.

2.2.2. Màn hình TFT

Model: TFT LCD 1.8 inch.

Độ phân giải: 128x160 pixels.

Driver: ST7735 (dựa trên các mã lệnh khởi tạo).

Tác dụng: Hiển thị hình ảnh và chữ theo dữ liệu 16-bit (RGB565) nhận được qua SPI.

2.2.3. Các linh kiện khác

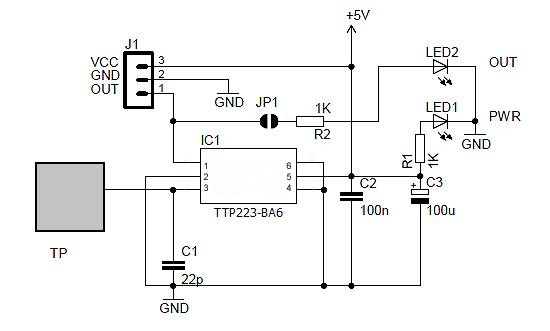
Mạch nạp: ST-Link/V2 (hoặc mạch nạp tương thích) để nạp chương trình và gỡ lỗi cho STM32.

Nguồn: Nguồn 3.3V DC (thường được cung cấp qua cổng USB của mạch nạp hoặc bộ chuyển đổi USB-TTL).

Dây cắm: Dây nối (Jumper Wires) để kết nối các chân giao tiếp.

Cảm ứng 1 chạm điện dung Capacitive Touch Button TTP223B

TTP223 (bao gồm phiên bản B) là IC cảm ứng điện dung 1 kênh, dùng để thay thế nút nhấn truyền thống.



Hình 2.2:Sơ đồ nguyên lý của TTP223B

Khi chưa chạm, điện dung tại TP ổn định, IC nhận biết trạng thái “không chạm”, ngõ ra Q ở mức thấp (0 V).

Khi chạm tay vào TP, điện dung tăng lên, IC nhận tín hiệu cảm ứng, ngõ ra Q lên mức cao (5 V). LED2 sáng báo có cảm ứng. Khi nhả tay, IC tự động đưa Q về thấp (0 V) và LED2 tắt.

Bảng 2.1: Trạng thái hoạt động của TTP223B

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| TOG | AHLB | Chế độ | Kiểu logic | Không chạm | Chạm | Q |
| 0 | 1 | Direct | H | 0 | 1 | Khi chạm thì Q lên 1, khi nhả thì về 0 |
| 0 | 0 | Direct | L | 1 | 0 | Khi chạm thì Q xuống 0, khi nhả thì lên 1 |
| 1 | 1 | Toggle | H | 0>1>0>... | Theo lần chạm | Mỗi lần chạm đổi trạng thái Q (1→0→1→0…) |
| 1 | 0 | Toggle | L | 1>0>1>... | Theo lần chạm | Mỗi lần chạm đổi trạng thái Q (ngược logic với Active-High) |

2.2.4. Sơ đồ kết nối

Màn hình TFT thường sử dụng giao thức SPI1 để kết nối :

Bảng 2.2: Sơ đồ kết nối chân

|  |  |  |  |
| --- | --- | --- | --- |
| Chức năng TFT | Chân | Tên ngoại vi | Vai trò |
| SCL | PA5 | SPI1\_SCK | Xung CLK |
| SDA | PA7 | SPI1\_MOSI | Dữ liệu-Màn hình |
| CS | PB0 | GPIO\_Output | Chip Select |
| A0 | PB1 | GPIO\_Output | Chọn lệnh/Dữ liệu |
| RST | PB10 | GPIO\_Output | Reset cứng |
| VCC | 3.3V | Nguồn | Cáp nguồn |
| GND | GND | Nguồn | Nối đất |

2.3. Phần mềm

2.3.1. STM32CubeMX

Dưới đây là hình ảnh về phần mềm STM32CubeMX:



Hình 2.3: Phần mềm STM32CubeMX

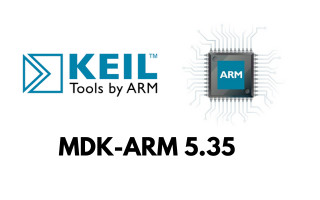
STM32CubeMX là công cụ cấu hình đồ họa và tạo mã nguồn khởi tạo của STMicroelectronics. Công cụ này được sử dụng để:

Cấu hình Pinout: Sử dụng giao diện đồ họa trực quan để chọn vi điều khiển STM32F103C8T6, kích hoạt ngoại vi SPI1 ở chế độ Master, và cấu hình các chân PB0, PB1, PB10 làm GPIO Output để điều khiển các tín hiệu CS (Chip Select), DC (Data/Command) và RST (Reset) của màn hình.

Cấu hình Clock: Thiết lập cây xung nhịp (clock tree) cho hệ thống, bao gồm tần số hoạt động của vi điều khiển (ví dụ: 72 MHz) và tần số cho bus ngoại vi APB2, từ đó xác định tần số SPI clock (ví dụ: 24 MHz).

Tạo Mã nguồn: Sau khi hoàn tất cấu hình, STM32CubeMX tự động sinh ra một bộ mã nguồn khởi tạo dựa trên thư viện HAL. Dự án này được tạo tương thích hoàn toàn để mở và phát triển tiếp bằng trình biên dịch Keil C.

2.3.2. Keil C (Keil µVision )



Hình 2.4: KeilC

Keil C là Môi trường phát triển tích hợp được sử dụng để viết mã, biên dịch, gỡ lỗi và nạp chương trình cho vi điều khiển. Sau khi STM32CubeMX tạo dự án, toàn bộ quá trình phát triển phần mềm ứng dụng sẽ diễn ra trên Keil C.

2.3.3. Ngôn ngữ lập trình C

Toàn bộ mã nguồn ứng dụng được viết bằng ngôn ngữ lập trình C. Ngôn ngữ này được chọn vì nó là tiêu chuẩn công nghiệp trong lĩnh vực lập trình nhúng, cung cấp hiệu suất cao, cho phép quản lý bộ nhớ trực tiếp và có khả năng tối ưu hóa tốt cho các hệ thống có tài nguyên hạn chế.

2.3.4. Thư viện HAL (Hardware Abstraction Layer)

Thư viện HAL do STM32CubeMX tích hợp sẵn vào mã nguồn khởi tạo, đóng vai trò là một lớp trừu tượng giúp đơn giản hóa việc tương tác với phần cứng. Các chức năng chính của thư viện HAL được sử dụng trong dự án bao gồm:

Khởi tạo ngoại vi: Sử dụng các cấu trúc (SPI\_HandleTypeDef hspi1) và hàm (HAL\_SPI\_Init) để thiết lập thông số và kích hoạt ngoại vi SPI1.

Điều khiển SPI: Hàm HAL\_SPI\_Transmit(&hspi1, buffer, size, timeout) là hàm cốt lõi, được dùng để gửi các lệnh và dữ liệu đến màn hình thông qua giao thức SPI.

Điều khiển GPIO: Hàm HAL\_GPIO\_WritePin(GPIOx, Pin, State) được dùng để điều khiển trạng thái logic (cao/thấp) của các chân CS, DC và RST.

2.3.5. Công cụ chuyển đổi hình ảnh

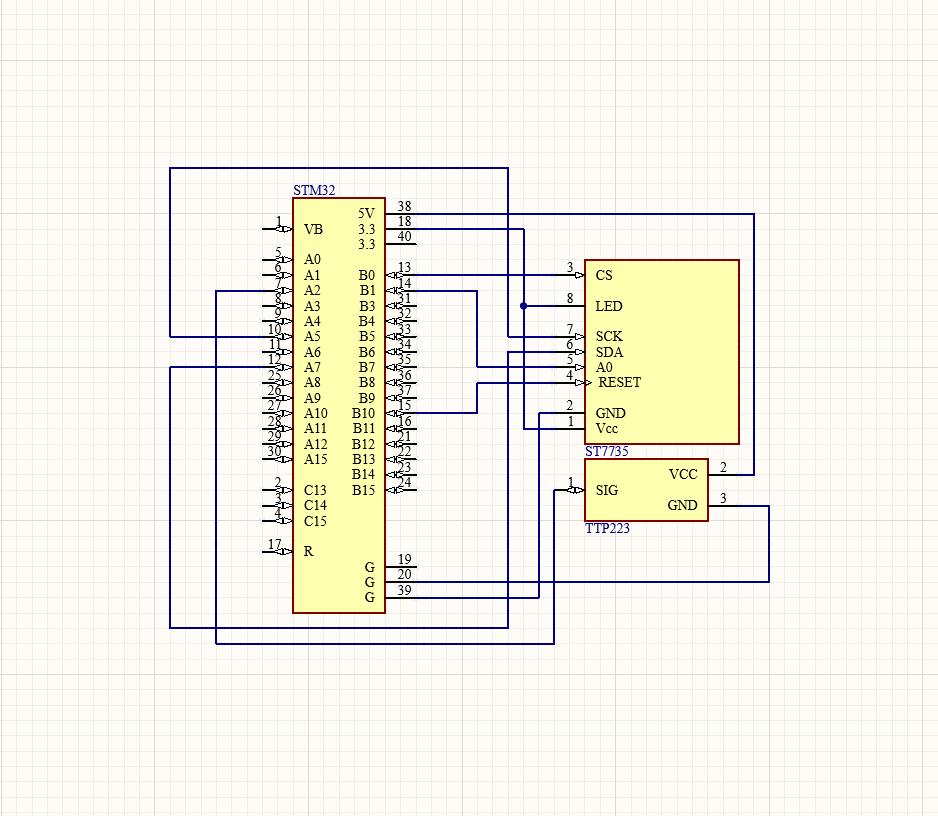
Để hiển thị ảnh bitmap (JPEG, BMP) lên màn hình TFT, ảnh cần được chuyển đổi thành mảng dữ liệu C. Các công cụ: Sử dụng các phần mềm như LCD Image Converter hoặc các script Python/Online Converter, Image2LCD.

Đầu ra: Công cụ chuyển đổi ảnh màu 16-bit RGB (phổ biến) thành mảng theo định dạng RGB565 (như mảng Image\_hvktmm trong code). Mảng này sau đó được nhúng vào file hvktmm.h của dự án để VĐK có thể truy cập và gửi qua SPI.

CHƯƠNG 3 : THIẾT KẾ VÀ THI CÔNG

3.1. Sơ đồ nguyên lý

Hình dưới đây là sơ đồ nguyên lý của mạch :



Hình 3.1: Sơ đồ nguyên lý

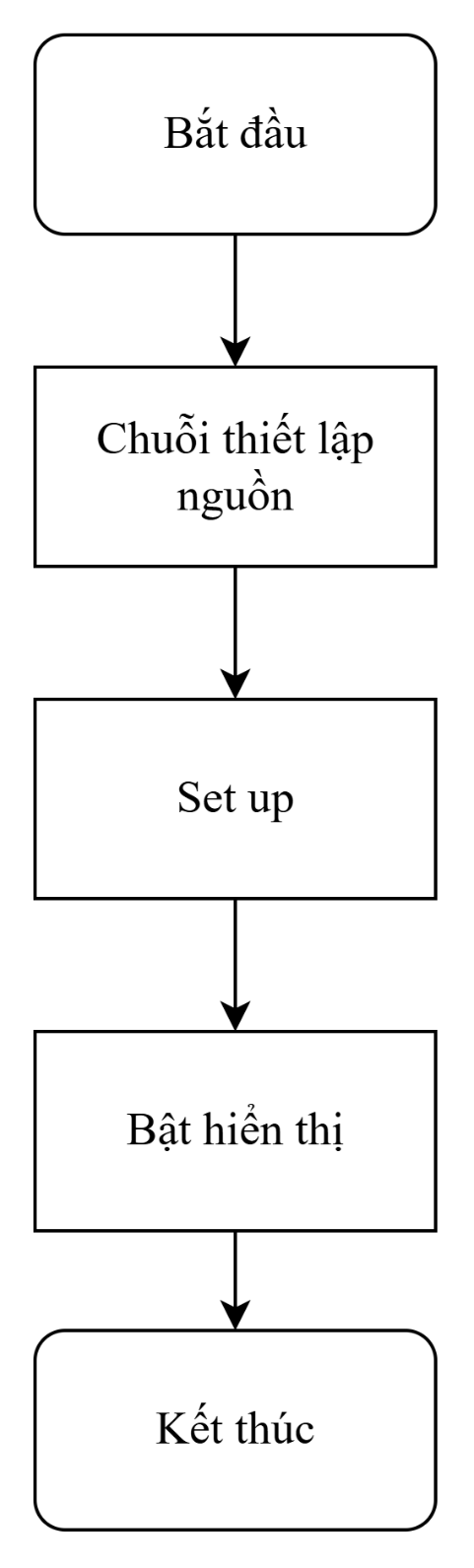
3.2. Thuật toán và mô tả chương trình

Trong chương trình, việc điều khiển và hiển thị dữ liệu lên màn hình TFT được thực hiện thông qua các hàm lập trình. Mỗi hàm đảm nhiệm một chức năng cụ thể như khởi tạo màn hình, gửi lệnh điều khiển, ghi dữ liệu hình ảnh hoặc hiển thị ký tự và chuỗi văn bản. Nội dung mục này trình bày các hàm và thuật toán chính được sử dụng trong chương trình.

3.2.1. Khởi tạo và giao tiếp màn hình TFT

Hàm khởi tạo TFT có nhiệm vụ cấu hình các chân giao tiếp, gửi các lệnh điều khiển ban đầu đến màn hình để đưa vào chế độ sẵn sàng hiển thị. Quá trình này bao gồm các bước gửi CMD (command) và Data để điều khiển các thanh ghi của IC TFT.

Sơ đồ dưới đây mô tả thuật toán khởi tạo TFT:



Hình 3.2: Thuật toán khởi tạo hiển thị màn hình TFT

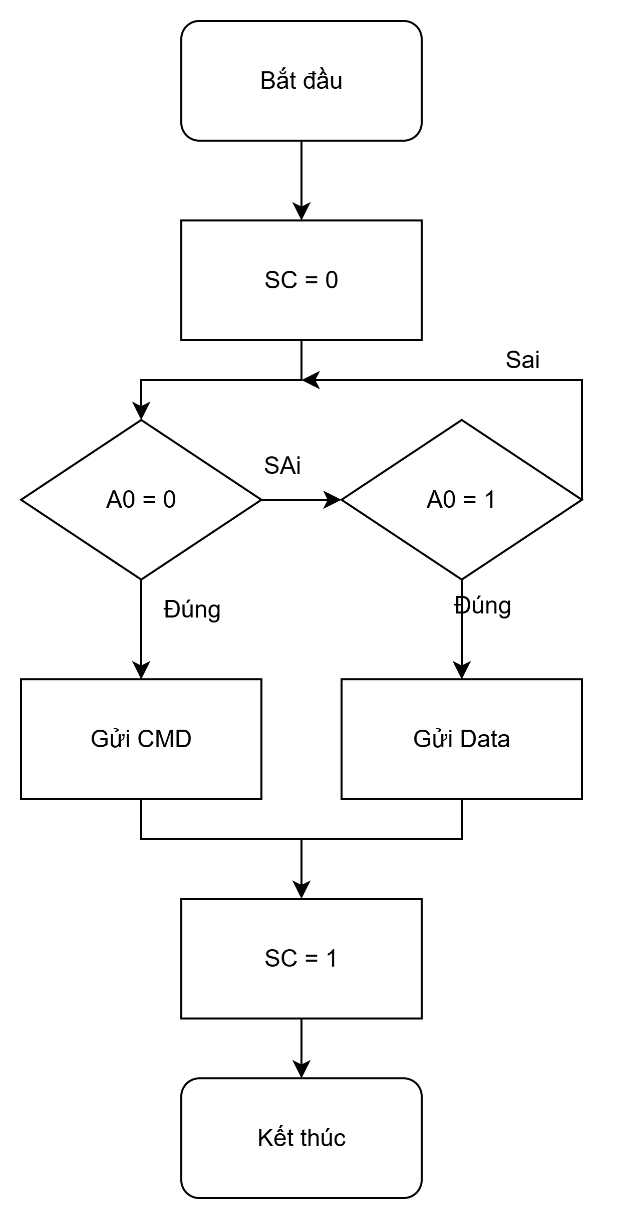
Phần chuỗi thiết lập nguồn mô tả bằng các bước sau :



Hình 3.3: Sơ đồ giải thuật chi tiết chuỗi thiết lập nguồn

Mô tả chi tiết quá trình khởi động phần mềm, điều khiển nguồn và hiệu chỉnh gamma.

MCU sẽ giao tiếp với controler của ST7735 bằng các CMD và Data :



Hình 3.4: Sơ đồ giải thuật gửi lệnh và dữ liệu đến màn hình TFT

Trình bày quá trình kiểm tra tín hiệu điều khiển A0 để xác định gửi CMD hay Data

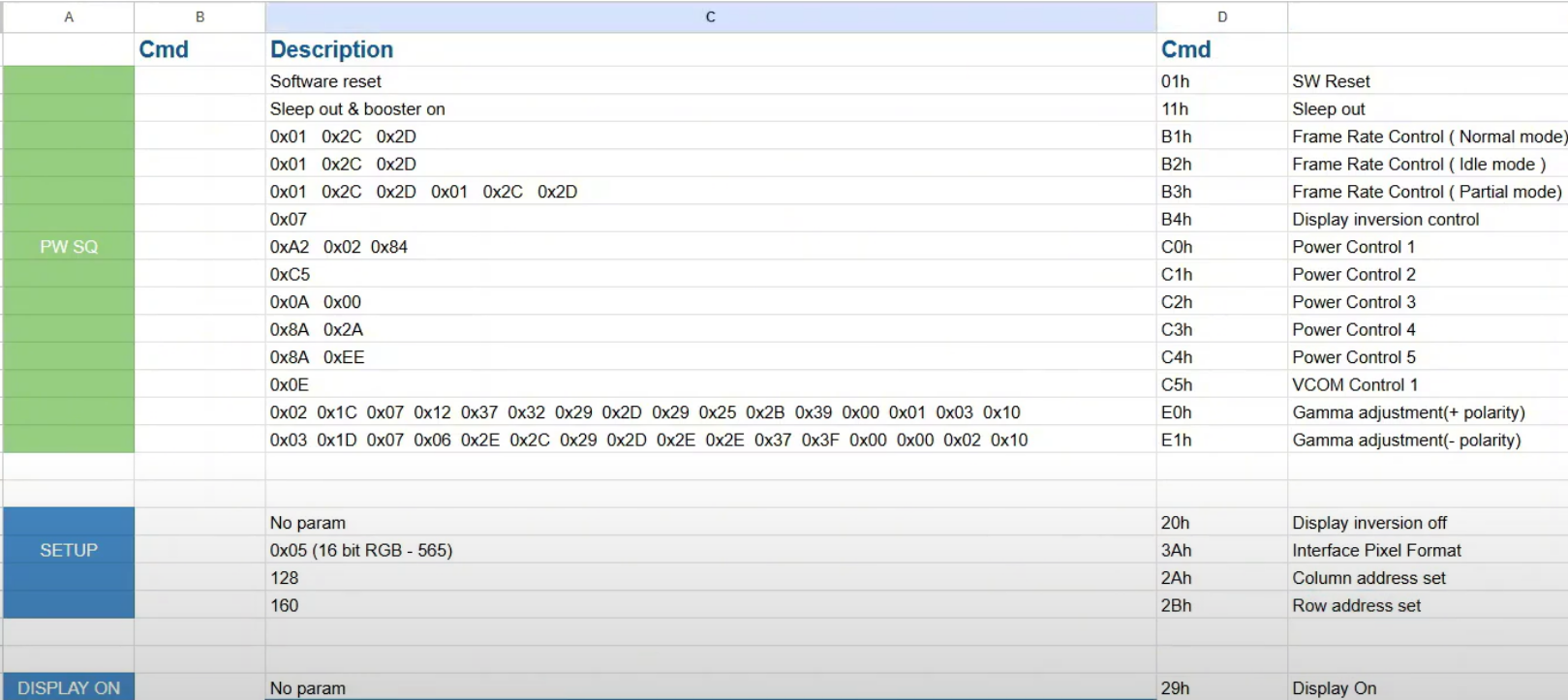
Ba sơ đồ trên thể hiện quá trình xử lý hiển thị của hệ thống.

Hình 1 biểu diễn thuật toán tổng quát từ khi bắt đầu khởi tạo đến khi hiển thị dữ liệu.

Hình 2 trình bày chi tiết các bước trong giai đoạn 'Chuỗi thiết lập nguồn', bao gồm khởi động phần mềm, điều khiển nguồn và hiệu chỉnh gamma.

Hình 3 minh họa thuật toán truyền lệnh và dữ liệu đến bộ điều khiển TFT, trong đó tín hiệu điều khiển A0 quyết định việc gửi lệnh (CMD) hoặc dữ liệu (Data).

Hình dưới đây mô tả chi tiết các lệnh và tham số dùng trong quá trình khởi tạo LCD ST7735, bao gồm các nhóm lệnh điều khiển nguồn, thiết lập hiển thị và định dạng giao diện dữ liệu:

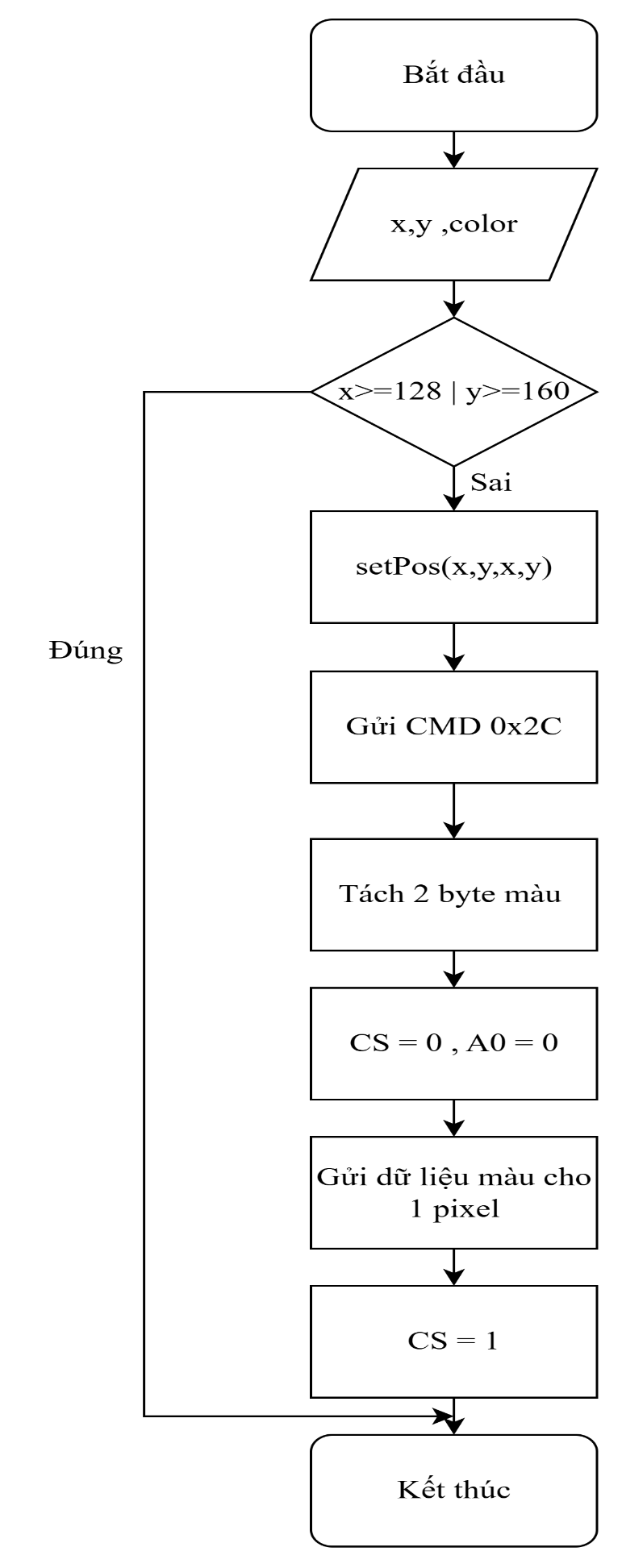


Hình 3.5: Các lệnh khởi tạo màn hình ST7735

3.2.2. Các hàm điều khiển cơ bản

**Hàm drawPixel(x, y, color):**

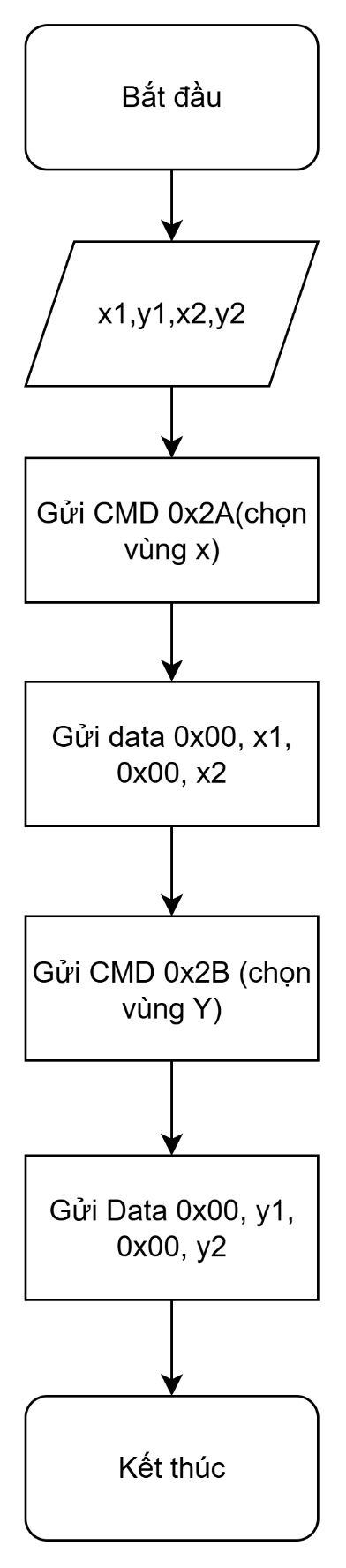
Dùng để vẽ điểm ảnh (pixel) tại toạ độ (x,y) với màu sắc được chỉ định. Hàm này thường được sử dụng như nền tảng cho các hàm vẽ cao hơn như vẽ đường, hình chữ nhật hoặc hiển thị ký tự.



Hình 3.6: Sơ đồ thuật toán hàm drawPixel()

**Hàm setPos(x1, y1, x2, y2):** Thiết lập vùng hiển thị trên màn hình bằng cách gửi các tọa độ bắt đầu (x1, y1) và kết thúc (x2, y2).

Hàm này giúp giới hạn vùng vẽ, tăng hiệu suất khi cần hiển thị một phần nhỏ trên màn hình. Sơ đồ dưới đây mô tả thuật toán của hàm :



Hình 3.7:Thuật toán mô tả hàm setPos()

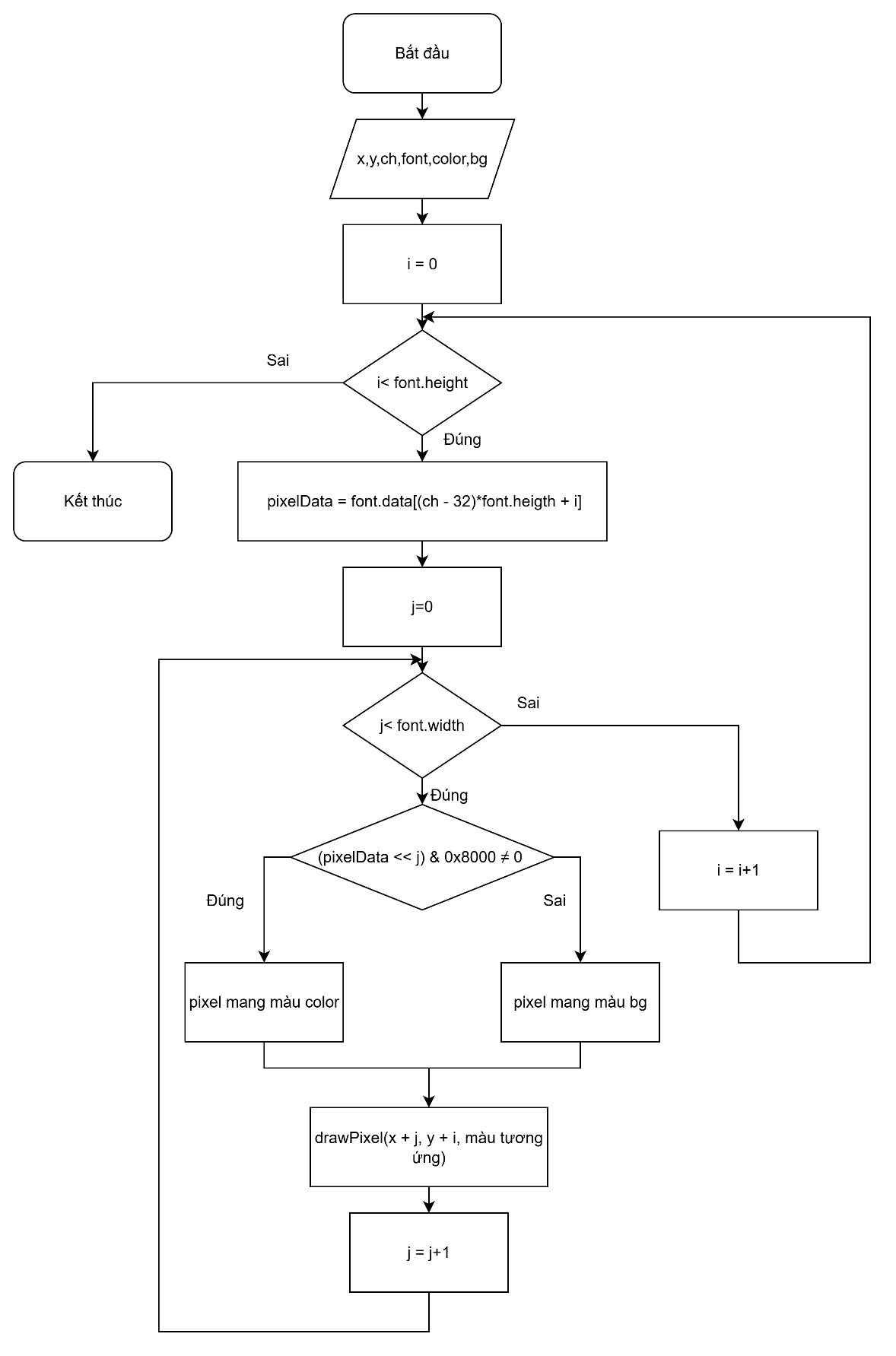
3.2.3.Các hàm hiển thị ký tự và hình ảnh

Để hiển thị ký tự lên màn hình, chương trình sử dụng một cấu trúc dữ liệu mô tả đặc tính của font chữ. Cấu trúc này lưu trữ các thông tin như chiều rộng, chiều cao và dữ liệu điểm ảnh của từng ký tự, giúp hệ thống xác định chính xác hình dạng và kích thước ký tự khi hiển thị.

Trong phần này, chương trình được xây dựng nhằm điều khiển việc hiển thị ký tự, chuỗi ký tự và hình ảnh trên màn hình TFT. Các hàm được thiết kế để thực hiện các nhiệm vụ như vẽ điểm ảnh (pixel), thiết lập vùng hiển thị, và quản lý dữ liệu của font chữ cũng như hình ảnh.

Mỗi hàm đều có chức năng và nguyên lý hoạt động riêng, được trình bày cụ thể ở các mục sau :

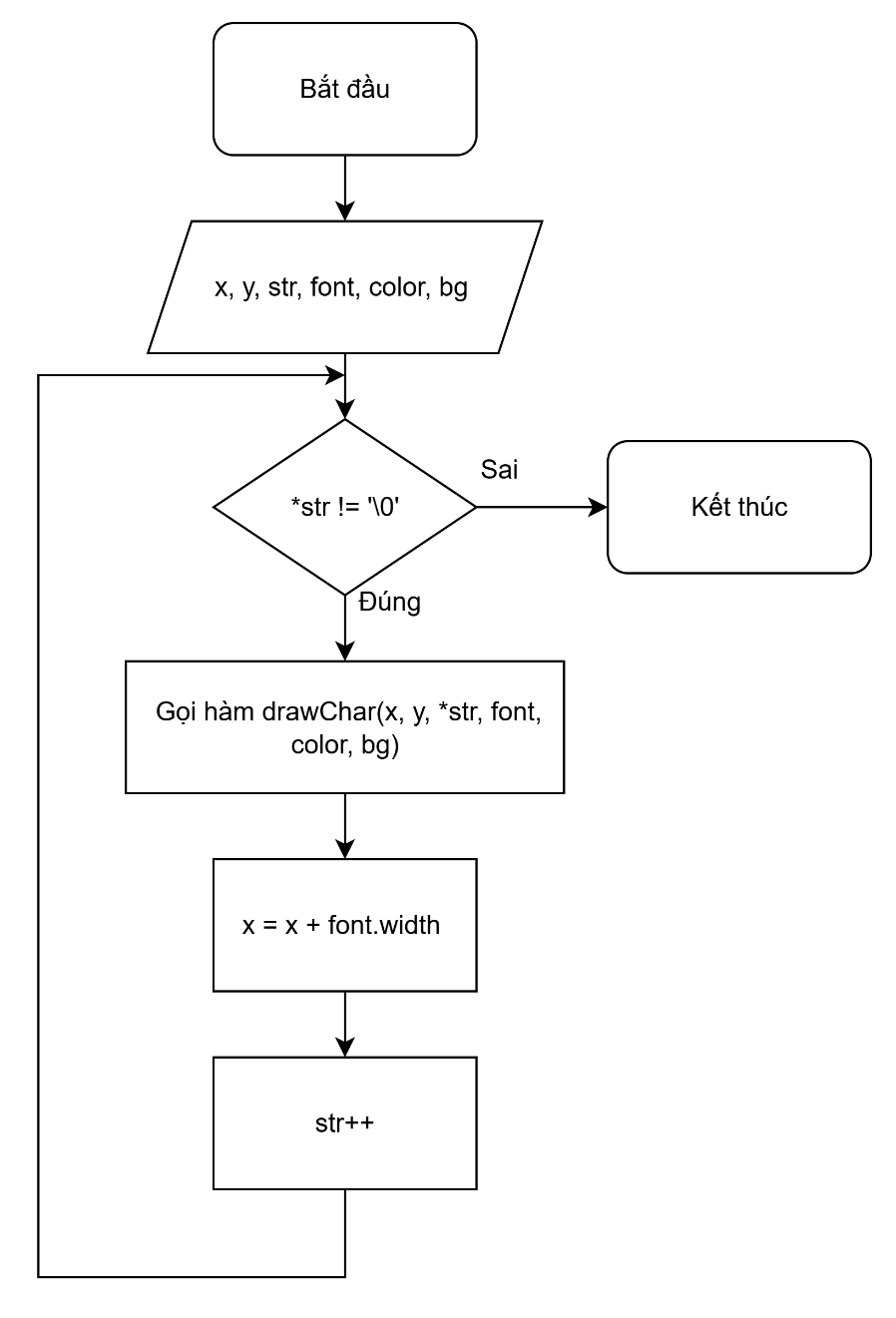
**Hàm drawChar():** Dùng để vẽ một ký tự tại vị trí (x, y) dựa trên dữ liệu mẫu trong font.



Hình 3.8: Sơ đồ thuật toán hmaf drawChar()

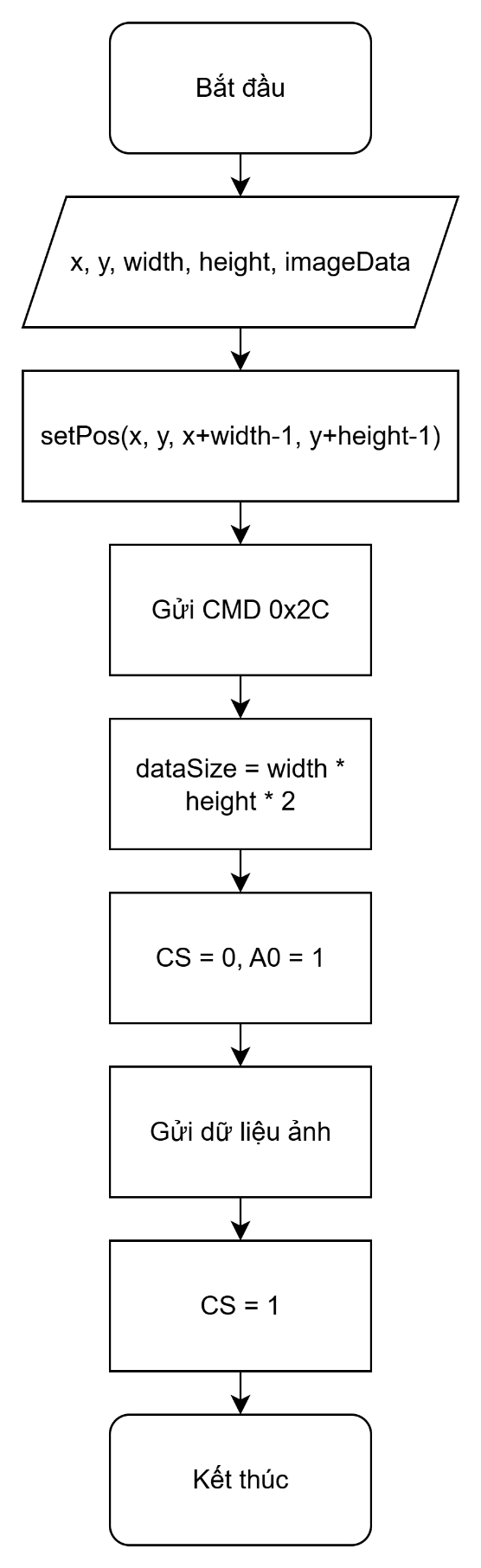
**Hàm drawString()**: Gọi lần lượt hàm drawChar() để hiển thị một chuỗi ký tự liên tiếp trên màn hình.

Sơ đồ dưới đây:



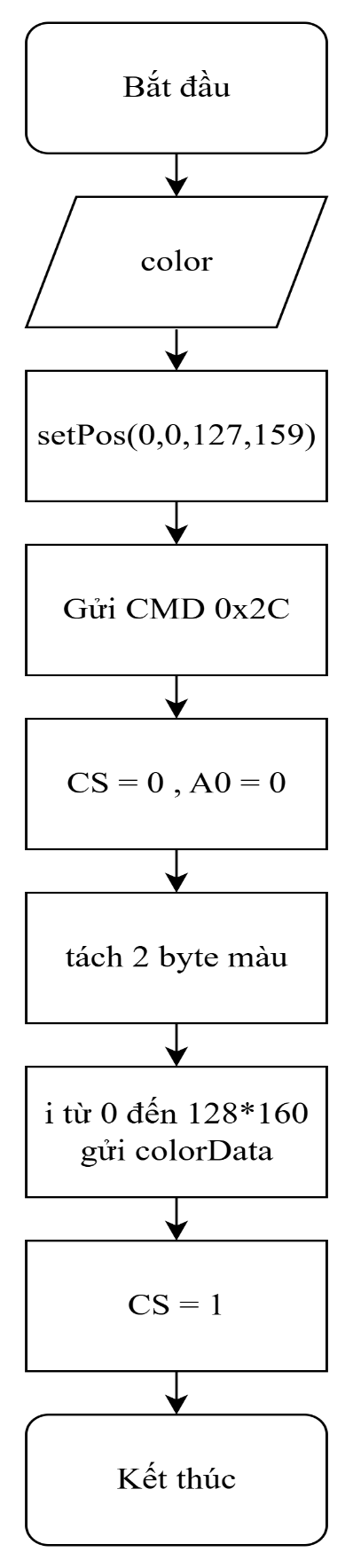
Hình 3.9: Thuật toán mô tả hàm drawString()

**Hàm drawImage**: Dùng để vẽ toàn bộ ảnh lên màn hình bằng cách gửi tuần tự tất cả dữ liệu điểm ảnh tới bộ nhớ GRAM của IC TFT.



Hình 3.10: Sơ đồ lưu đồ thuật toán hàm drawImage()

**Hàm fullDisplay()**: Dùng để tô toàn bộ màn hình bằng một màu duy nhất bằng cách gửi tuần tự dữ liệu màu của tất cả các điểm ảnh (pixel) tới bộ nhớ hiển thị (GRAM) của IC TFT.



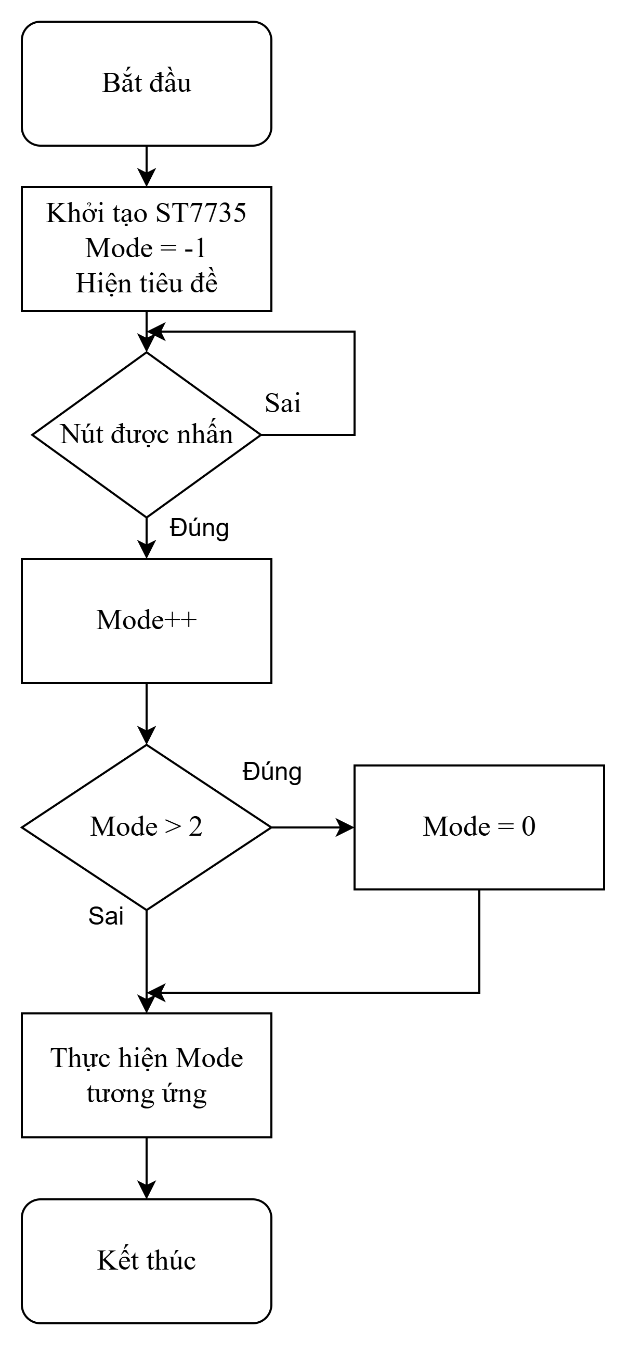
Hình 3.11: Sơ đồ giải thuật hàm fullDisplay();

3.2.4. Thuật toán và trình tự hoạt động

Để điều khiển màn hình TFT ST7735 hiển thị thông tin và hình ảnh theo yêu cầu, chương trình được xây dựng theo quy trình gồm nhiều bước xử lý liên tục. Trước hết, vi điều khiển thực hiện khởi tạo giao tiếp SPI và các chân điều khiển cần thiết để đảm bảo màn hình hoạt động ổn định. Sau đó, vùng hiển thị được thiết lập bằng hàm setPos() nhằm xác định khu vực cần hiển thị dữ liệu.

Trong quá trình thực thi, chương trình sẽ lần lượt hiển thị tên nhóm sinh viên và đề tài, hình ảnh minh họa thông qua các hàm hiển thị ký tự và drawImage(). Bên cạnh đó, hệ thống có kiểm tra trạng thái của nút nhấn để xác định người dùng có yêu cầu thay đổi chế độ hay không. Nếu nút nhấn được kích hoạt, chương trình sẽ bắt đầu chuyển đổi từ tên nhóm sinh viên qua hiển thị hình ảnh, sau đó thực hiện đổi màu nền toàn màn hình bằng hàm fullDisplay(), bằng việc tăng giá trị của biến color\_index để chuyển sang màu kế tiếp. Khi chỉ số này vượt quá lượng màu quy định, color\_index sẽ được đặt lại về 0 để lặp lại chu trình hiển thị.

Toàn bộ quá trình khởi tạo, hiển thị và cập nhật được lặp liên tục nhằm đảm bảo màn hình luôn hiển thị đúng nội dung. Trình tự hoạt động của chương trình được mô tả cụ thể trong **lưu đồ thuật toán** dưới đây :



Hình 3.11: Lưu đồ thuật toán của hệ thống

Trong lưu đồ trên, hệ thống bao gồm ba chế độ hoạt động (Mode 0, Mode 1 và Mode 2).

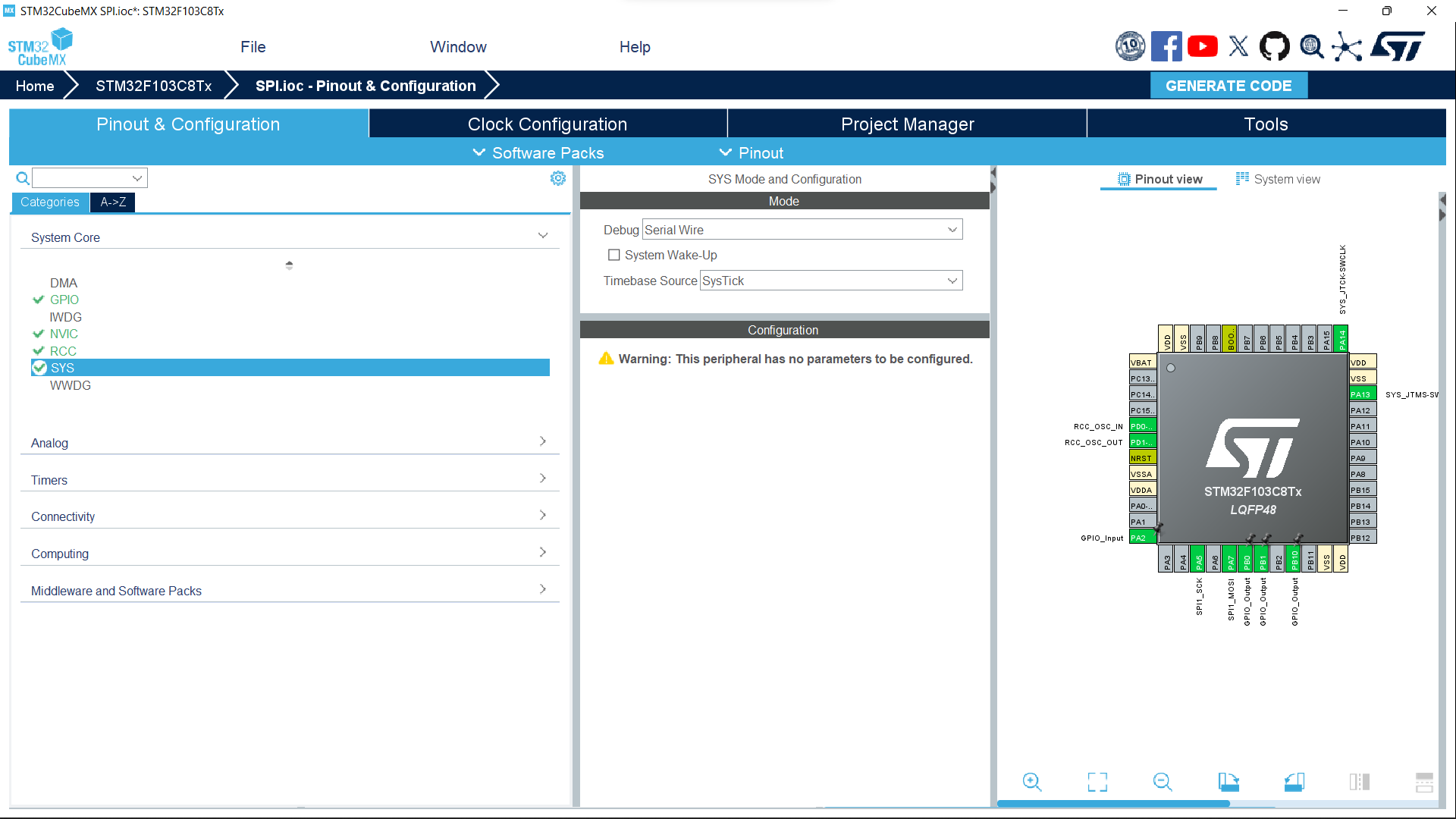
* **Mode 0:** Hiển thị tên nhóm sinh viên.
* **Mode 1:** Màn hình thay đổi màu nền theo chu kỳ.
* **Mode 2:** Hiển thị hình ảnh minh họa.

3.3. Thực hiện nạp và khởi chạy chương trình trên vi điều khiển

Sau khi hoàn thiện phần lập trình, bước tiếp theo là tiến hành cấu hình và nạp chương trình vào vi điều khiển. Đây là giai đoạn giúp mã nguồn được chuyển thành dạng thực thi, cho phép hệ thống hoạt động theo đúng thuật toán đã thiết kế. Quá trình này bao gồm các thao tác cấu hình clock, thiết lập chân giao tiếp, biên dịch và nạp code vào chip STM32

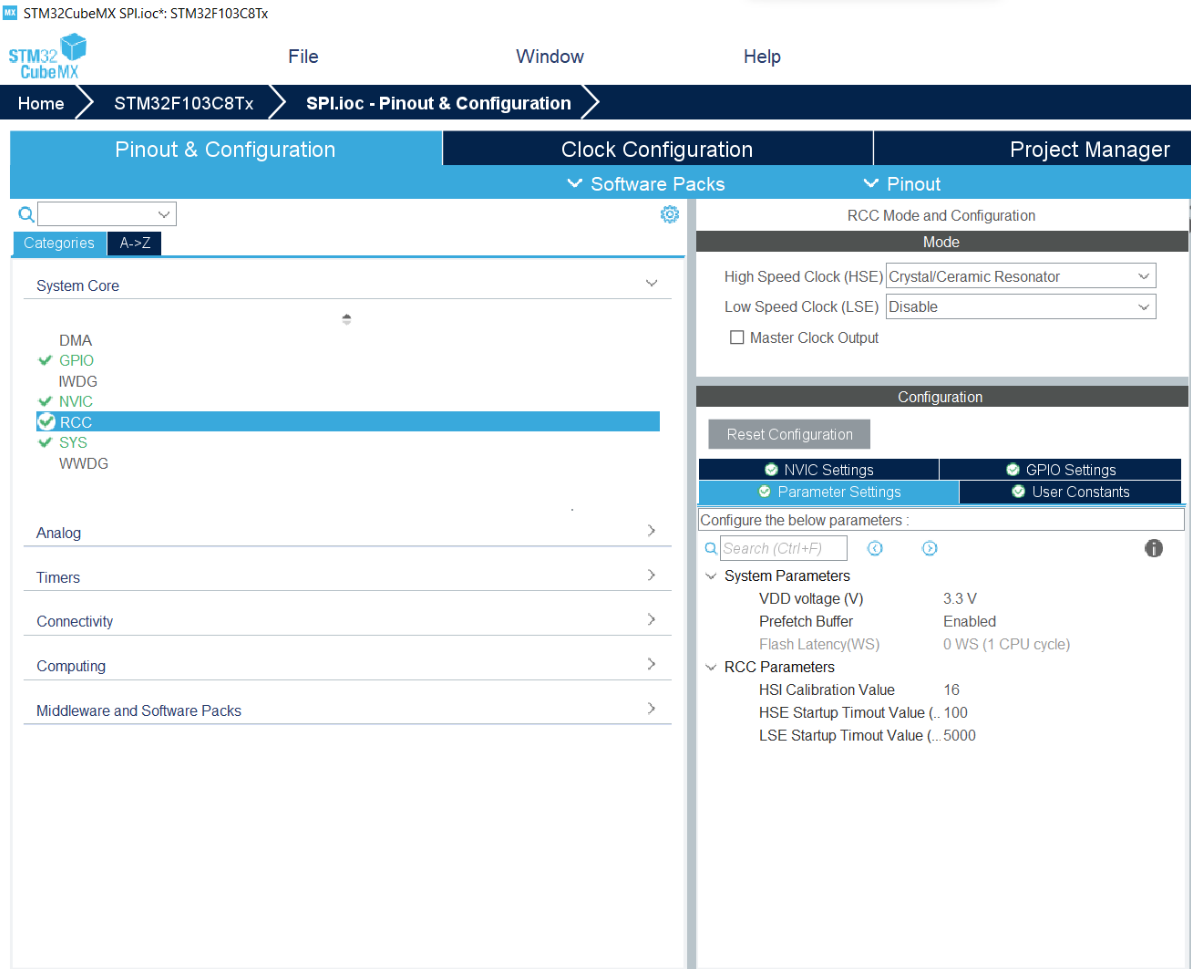
3.3.1. Cấu hình trên STM32cubeMX

Bước 1: Trong mục System Core chọn Serial Wire, giúp cho vi điều khiển có thể nạp chương trình qua ST-link và còn có thể gỡ lỗi khi cần thiết.



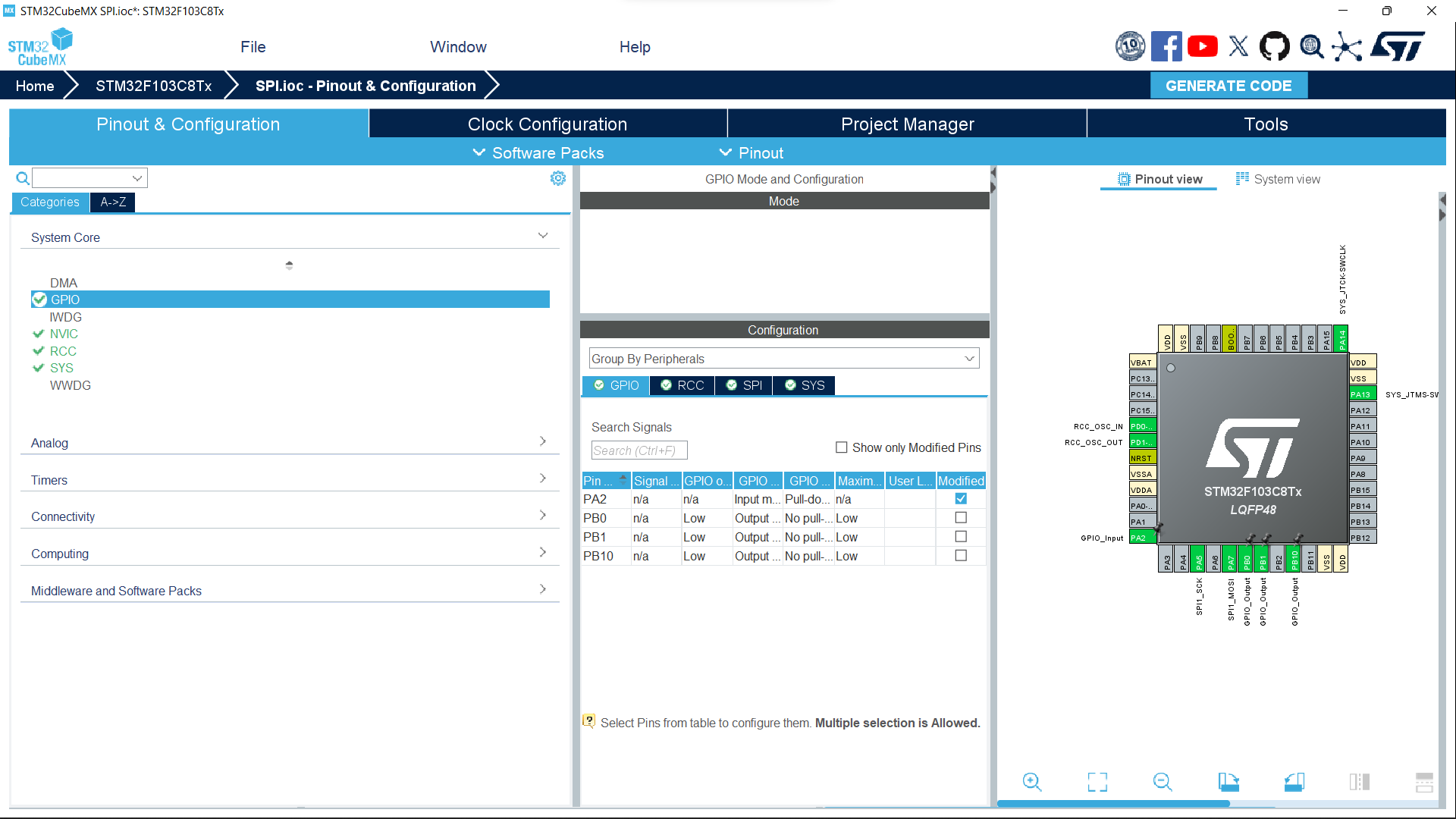
Hình 3.12: Cấu hình cho SerialWire

Bước 2: Trong Categories ta chọn Crystall/Ceramic Resonator



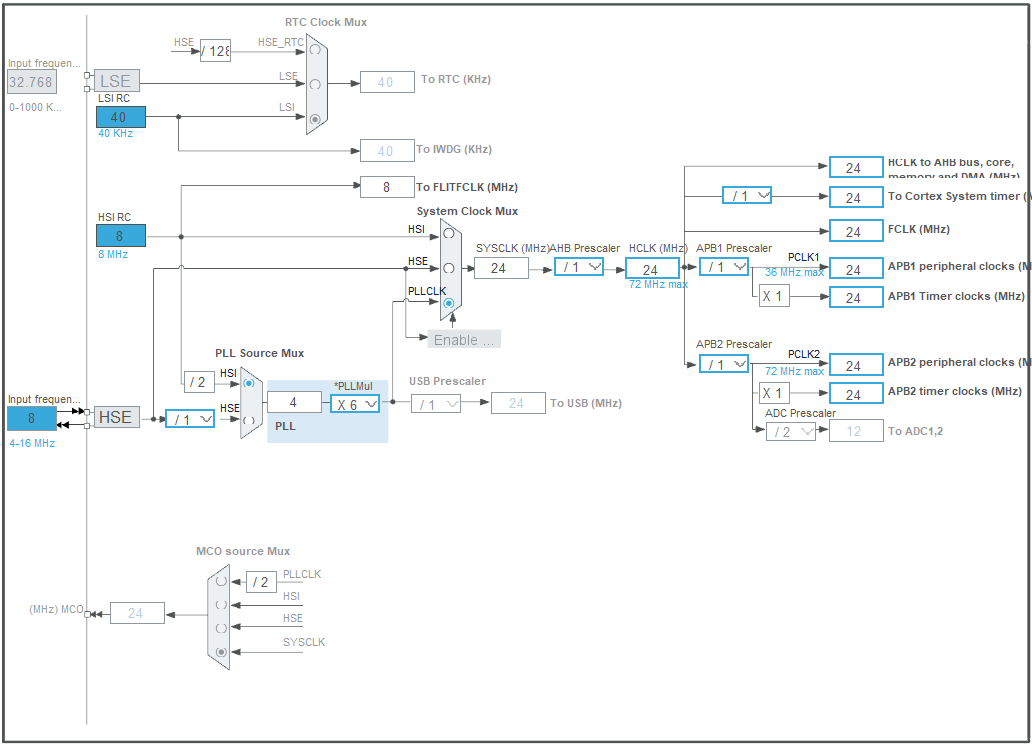
Hình 3.13: Chọn thạch anh ngoài

Bước 3: Cấu hình cho các chân PA2,PB0,PB1,PB10



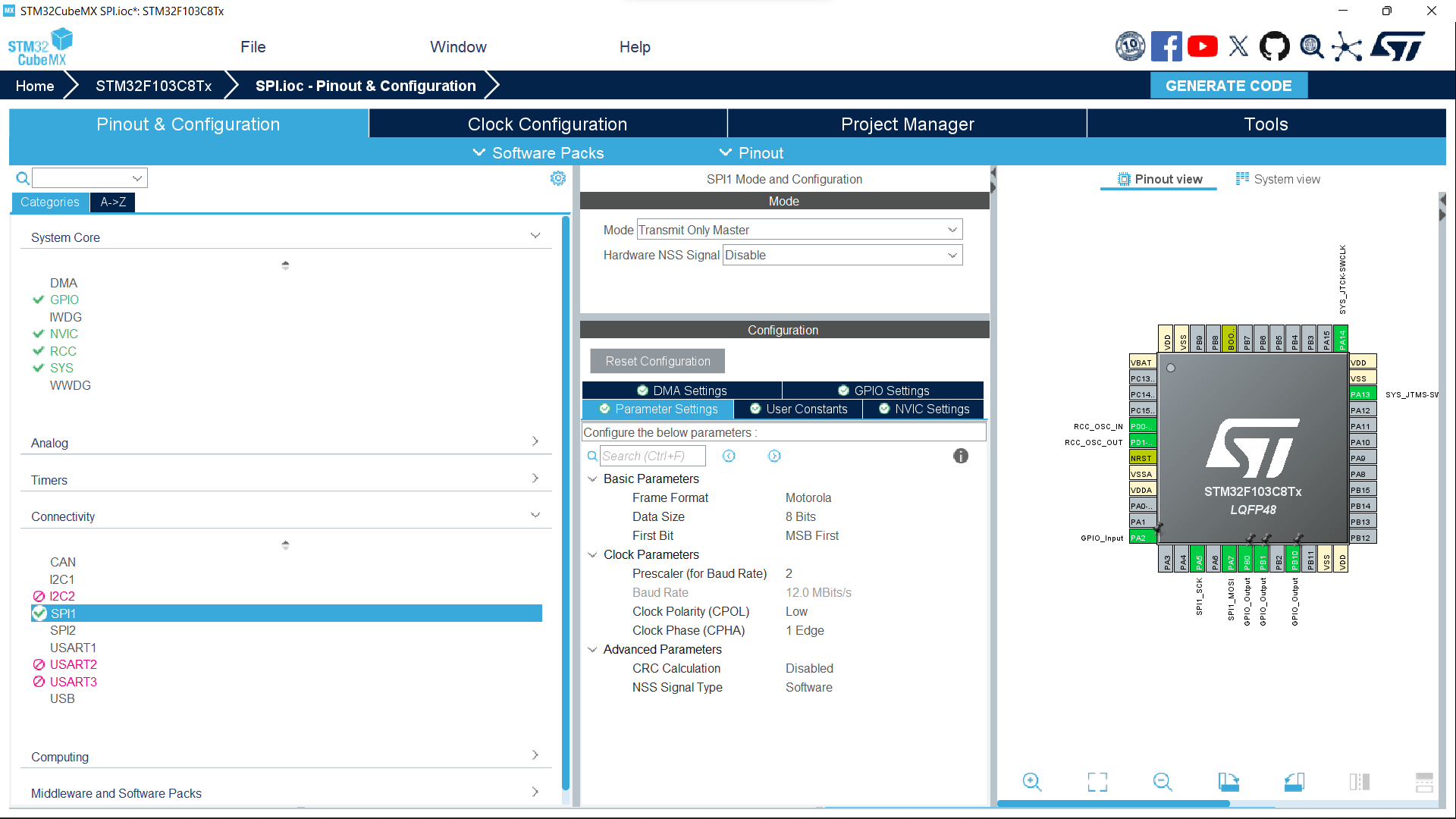
Hình 3.14: Cấu hình chân GPIO

Bước 4: Đặt tần số hệ thống là 24 MHz



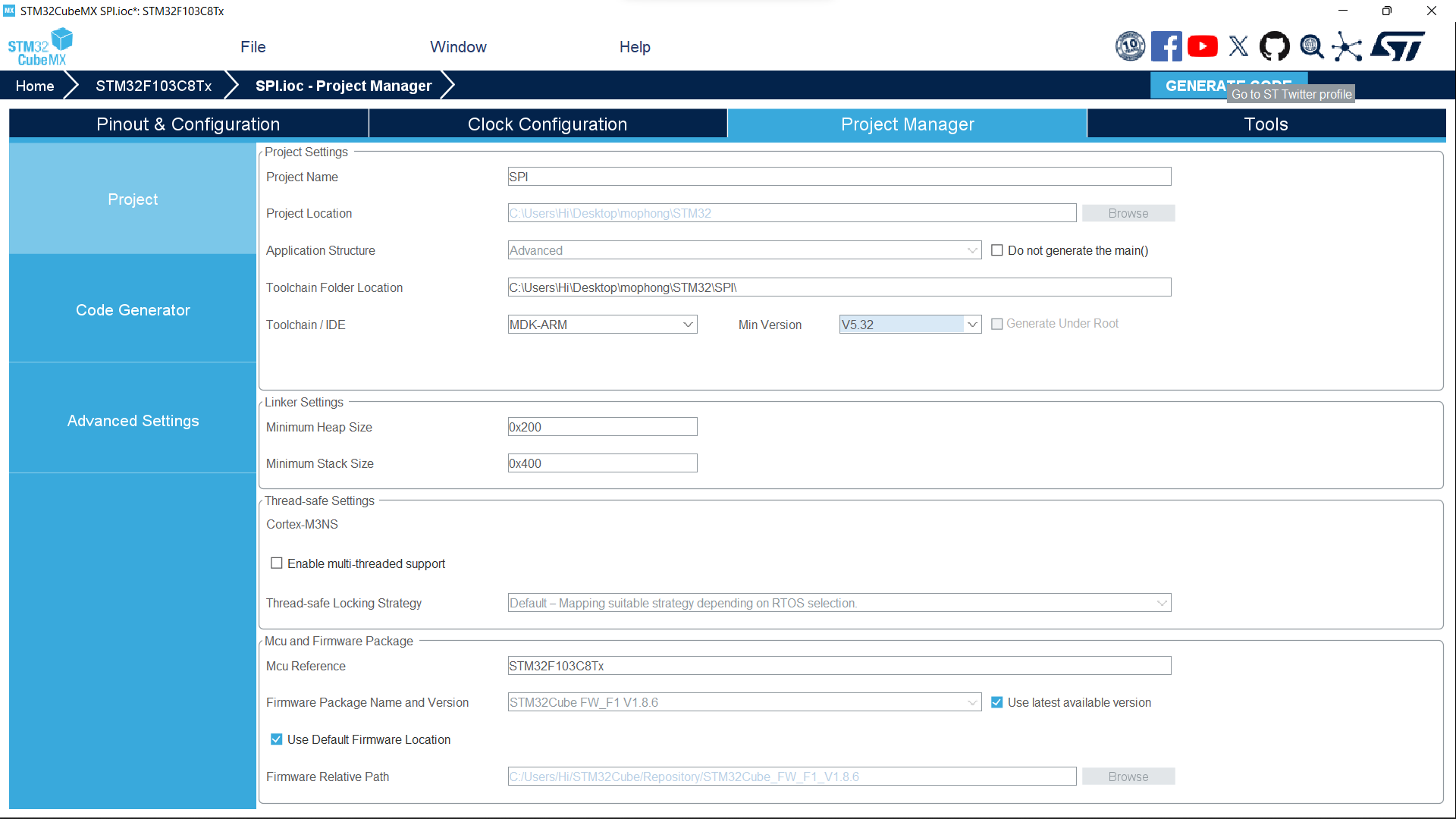
Hình 3.15: Cài đặt tần số

Bước 5: Chọn giao tiếp SPI để MCU giao tiếp được với ST7735



Hình 3.16: Cài đặt giao tiếp SPI

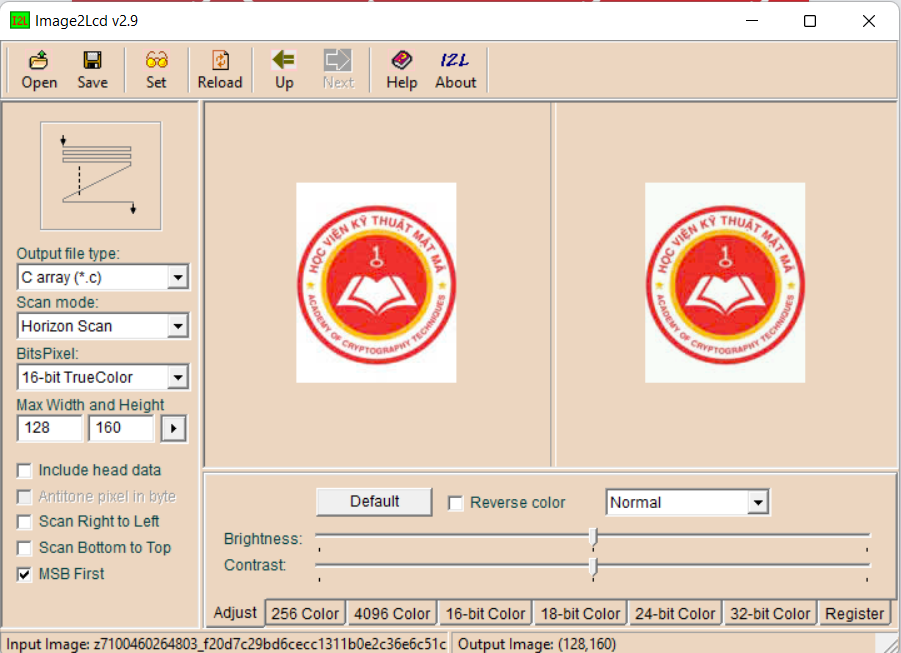
Bước 6: Đặt tên cho Project, chọn MDK-ARM rồi GENERATECODE



Hình 3.17: Generatecode

3.3.2. Tạo File ảnh hiển thị

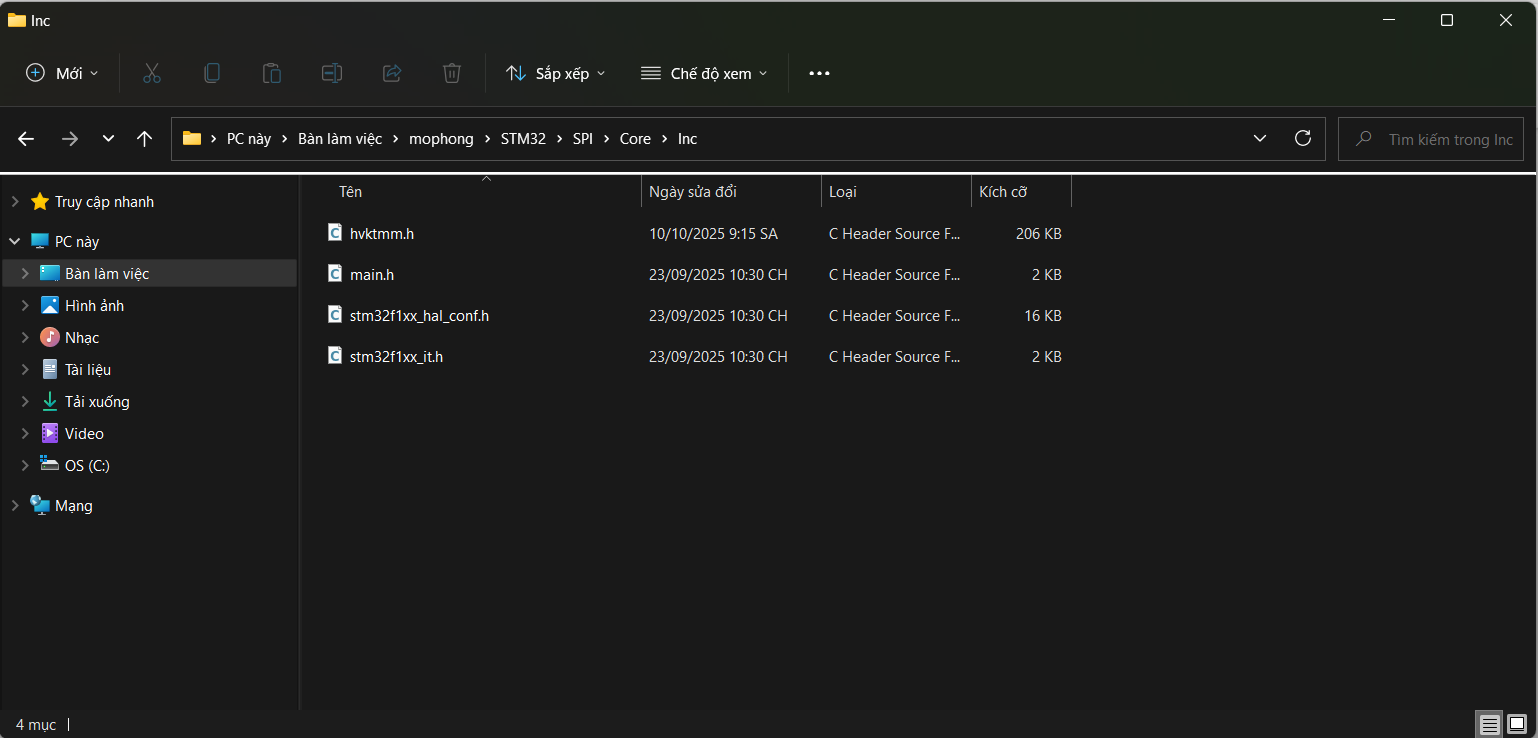
Bước 1: Tạo file ảnh đã được chuyển đổi bằng Image2LCD hoặc các phần mềm tương tự:



Hình 3.18: Tạo ảnh hiển thị

Chọn mục Open để chọn ảnh sau đó nhấn Save để lưu ảnh thành file đuôi .h.

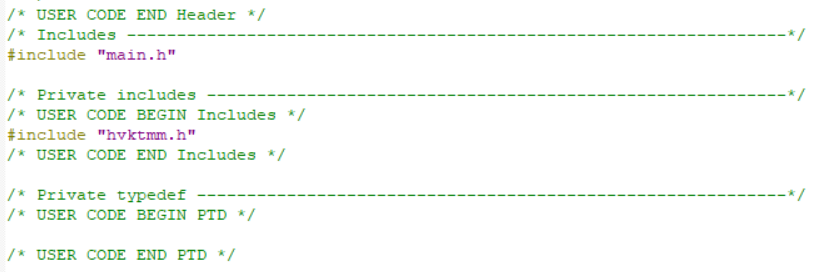
Bước 2: Lưu file vừa tạo thành vào mục Inc trong project được tạo thành từ STM32cubeMX.



Hình 3.19: Chèn thư viện ảnh

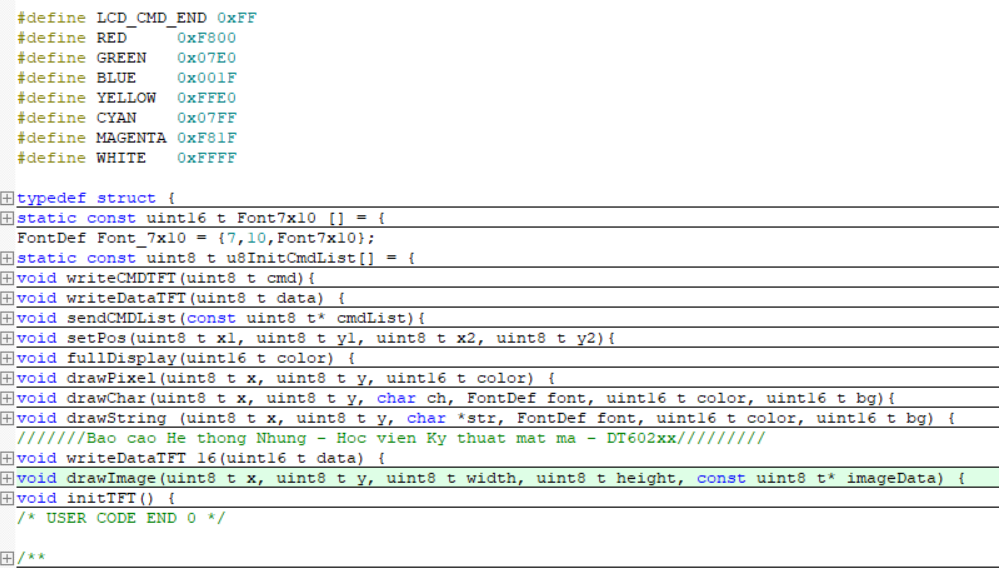
3.3.3. Viết chương trình và nạp code

Bước 1: Khai báo thư viện ảnh



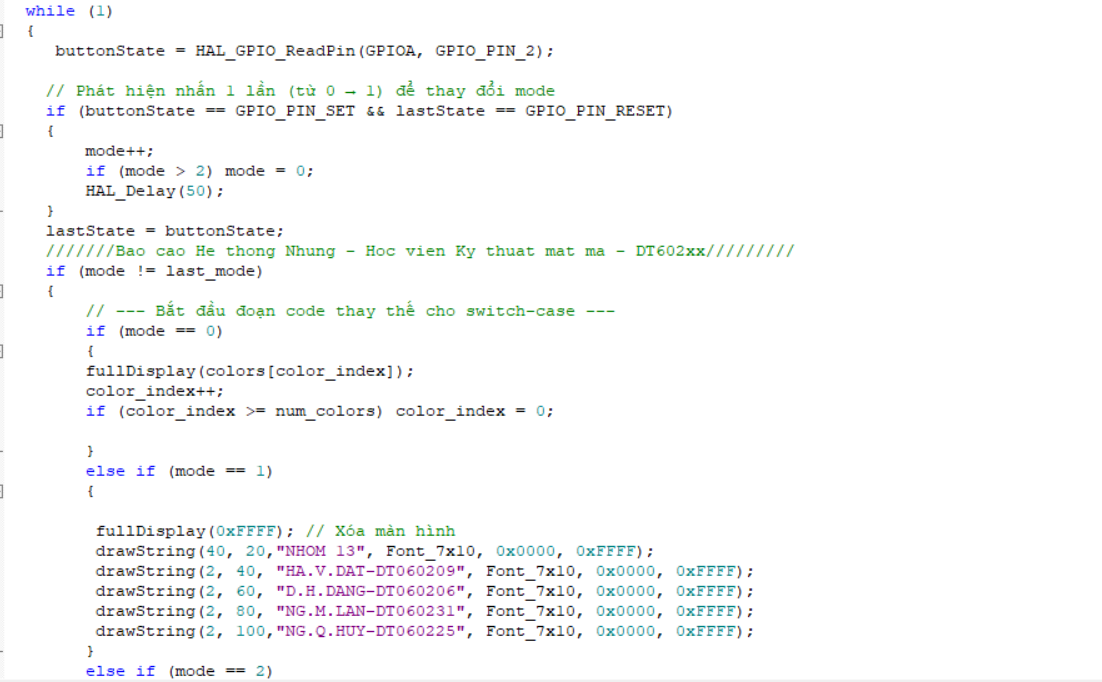
Hình 3.20: Khai báo thư viện

Bước 2: Viết các chương trình con (chương trình đầy đủ được lưu ở phần phụ lục)



Hình 3.21: Viết chương trình con

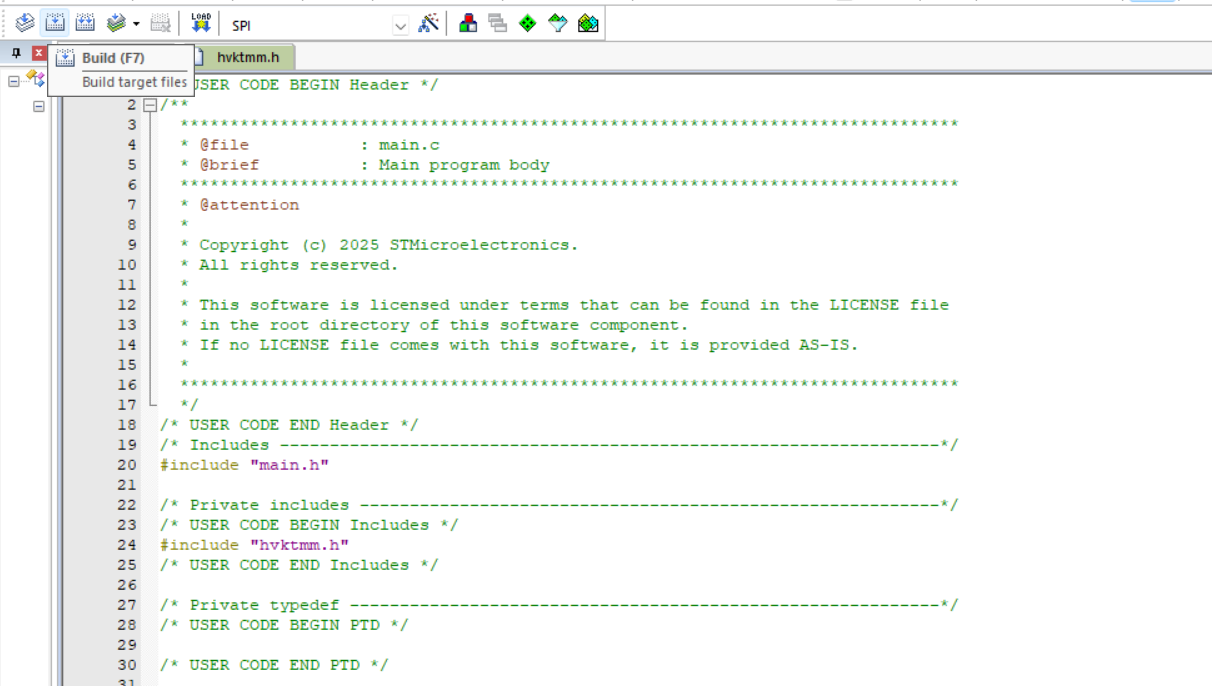
Bước 4: Viết chương trình chính trong hàm main()



Hình 3.22: Chương trình chính của hệ thống

Chương trình đầy đủ được cung cấp trong phần **PHỤ LỤC**

Bước 5: Nạp chương trình vào STM32F103C8T6, ta chọn Build rồi chọn Load để nạp code.



Hình 3.23: Nạp chương trình

3.4 Sản phẩm thực tế

3.4.1. Sản phẩm khi chưa cấp nguồn

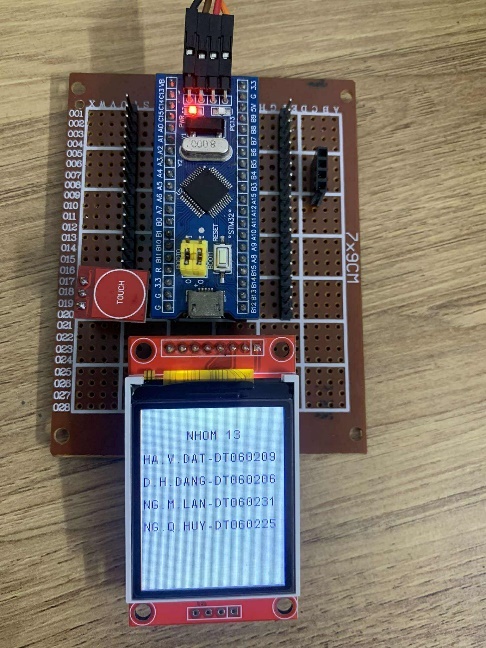
Sản phẩm thực tế khi chưa được cấp nguồn;



Hình 3.24: Sản phẩm khi chưa cấp nguồn

3.4.2. Sản phẩm thực tế khi đã cấp nguồn

Khi hiển thị chuỗi ký tự :



Hình 3.25:Màn hình TFT hiện chuỗi ký tự

Khi hiển thị hình ảnh:



Hình 3.26:Màn hình TFT hiện hình ảnh

KẾT LUẬN

1. Ưu điểm

Tốc độ cao và hiệu quả: Giao thức SPI cho phép truyền dữ liệu song công với tốc độ cao, nhanh hơn so với các giao thức như I2C, rất phù hợp cho các ứng dụng cần cập nhật màn hình nhanh như hiển thị hình ảnh.

Cấu hình đơn giản: So với các giao thức khác, SPI có cấu trúc master-slave và cơ chế chọn chip (CS/SS) đơn giản, giúp dễ dàng trong việc kết nối và lập trình.

Linh hoạt và phổ biến: SPI được hỗ trợ bởi hầu hết các dòng vi điều khiển và được sử dụng rộng rãi trong nhiều loại thiết bị ngoại vi như màn hình LCD, thẻ nhớ SD, cảm biến.

2. Nhược điểm

Sử dụng nhiều chân giao tiếp: SPI yêu cầu ít nhất 4 đường dây (SCLK, MOSI, MISO, CS), gây tốn tài nguyên chân của vi điều khiển hơn so với I2C hay UART chỉ dùng 2 dây.

Không có cơ chế xác nhận dữ liệu: Giao thức SPI không có cơ chế báo nhận (acknowledgement) từ slave, do đó master không biết chắc chắn liệu slave đã nhận được dữ liệu thành công hay chưa.

Chỉ hỗ trợ một master: Trong một mạng SPI, chỉ có thể có một thiết bị master duy nhất, hạn chế khả năng mở rộng hệ thống phức tạp với nhiều master.

Không có cơ chế kiểm tra lỗi: SPI thiếu cơ chế kiểm tra lỗi tích hợp như bit chẵn lẻ (parity bit) trong UART, đòi hỏi phải có các giải pháp phần mềm bổ sung nếu cần đảm bảo tính toàn vẹn dữ liệu.

3. Phương hướng phát triển

Nghiên cứu và áp dụng kỹ thuật DMA (Direct Memory Access) để truyền dữ liệu từ bộ nhớ đến ngoại vi SPI mà không cần sự can thiệp của CPU, giúp giải phóng tài nguyên vi điều khiển và tăng tốc độ làm tươi màn hình một cách đáng kể.

Khám phá các chế độ hoạt động khác nhau của SPI trên STM32 để tìm ra cấu hình tối ưu nhất cho việc truyền dữ liệu đồ họa.

Phát triển thư viện đồ họa nâng cao:

Xây dựng một thư viện đồ họa (GUI) hoàn chỉnh hơn, hỗ trợ vẽ các đối tượng phức tạp như biểu đồ, icon, và tạo các hiệu ứng chuyển động mượt mà.

Tích hợp khả năng xử lý font chữ đa dạng để hiển thị văn bản đẹp và chuyên nghiệp hơn.

Mở rộng ứng dụng trong các dự án IoT:

Kết hợp với các module không dây như Wi-Fi, Bluetooth để hiển thị dữ liệu từ cảm biến hoặc nhận lệnh điều khiển từ internet, điện thoại thông minh.

Xây dựng các giao diện người dùng tương tác cho các thiết bị IoT, biến màn hình TFT thành một trung tâm điều khiển và giám sát trực quan.

Tích hợp với các công nghệ hiển thị mới:

Nghiên cứu và thử nghiệm với các loại màn hình có độ phân giải cao hơn, hỗ trợ cảm ứng để phát triển các ứng dụng tương tác người-máy (HMI) phức tạp hơn.

Tìm hiểu các giao thức giao tiếp màn hình tiên tiến khác như MIPI-DSI được hỗ trợ trên các dòng STM32 cao cấp để đáp ứng yêu cầu về băng thông và hiệu suất cao hơn

TÀI LIỆU THAM KHẢO

|  |  |
| --- | --- |
| [1]. | Giáo trình Hệ thống nhúng, Nguyễn Thanh Ngọc, Học viện Kỹ thuật mật mã. |
| [2]. | TFT ST7735S Datasheet. |
| [3]. | Discovering STM32 Microcontrollers. |
| [4]. | Giáo trình Vi điều khiển STM32, Đại học Bách khoa T.P Hồ Chí Minh. |
| [5]. | Giáo trình Hệ thống nhúng với vi điều khiển STM32,Trường Đại học Công nghiệp Hà Nội. |

PHỤ LỤC

/\* USER CODE BEGIN Header \*/

/\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* @file : main.c

\* @brief : Main program body

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* @attention

\*

\* Copyright (c) 2025 STMicroelectronics.

\* All rights reserved.

\*

\* This software is licensed under terms that can be found in the LICENSE file

\* in the root directory of this software component.

\* If no LICENSE file comes with this software, it is provided AS-IS.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

/\* USER CODE END Header \*/

/\* Includes ------------------------------------------------------------------\*/

#include "main.h"

/\* Private includes ----------------------------------------------------------\*/

/\* USER CODE BEGIN Includes \*/

#include "hvktmm.h"

/\* USER CODE END Includes \*/

/\* Private typedef -----------------------------------------------------------\*/

/\* USER CODE BEGIN PTD \*/

/\* USER CODE END PTD \*/

/\* Private define ------------------------------------------------------------\*/

/\* USER CODE BEGIN PD \*/

/\* USER CODE END PD \*/

/\* Private macro -----------------------------------------------------------\*/

/\* USER CODE BEGIN PM \*/

/\* USER CODE END PM \*/

/\* Private variables -------------------------------------------------------\*/

SPI\_HandleTypeDef hspi1;

/\* USER CODE BEGIN PV \*/

/\* USER CODE END PV \*/

/\* Private function prototypes ---------------------------------------------\*/

void SystemClock\_Config(void);

static void MX\_GPIO\_Init(void);

static void MX\_SPI1\_Init(void);

/\* USER CODE BEGIN PFP \*/

/\* USER CODE END PFP \*/

/\* Private user code -------------------------------------------------------\*/

/\* USER CODE BEGIN 0 \*/

#define LCD\_CMD\_END 0xFF

#define RED 0xF800

#define GREEN 0x07E0

#define BLUE 0x001F

#define YELLOW 0xFFE0

#define CYAN 0x07FF

#define MAGENTA 0xF81F

#define WHITE 0xFFFF

typedef struct {

uint8\_t width;

uint8\_t heigth;

const uint16\_t \*data;

}FontDef;

static const uint16\_t Font7x10 [] = {

0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, // sp

0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x0000, 0x1000, 0x0000, 0x0000, // !

0x2800, 0x2800, 0x2800, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, // "

0x2400, 0x2400, 0x7C00, 0x2400, 0x4800, 0x7C00, 0x4800, 0x4800, 0x0000, 0x0000, // #

0x3800, 0x5400, 0x5000, 0x3800, 0x1400, 0x5400, 0x5400, 0x3800, 0x1000, 0x0000, // $

0x2000, 0x5400, 0x5800, 0x3000, 0x2800, 0x5400, 0x1400, 0x0800, 0x0000, 0x0000, // %

0x1000, 0x2800, 0x2800, 0x1000, 0x3400, 0x4800, 0x4800, 0x3400, 0x0000, 0x0000, // &

0x1000, 0x1000, 0x1000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, // '

0x0800, 0x1000, 0x2000, 0x2000, 0x2000, 0x2000, 0x2000, 0x2000, 0x1000, 0x0800, // (

0x2000, 0x1000, 0x0800, 0x0800, 0x0800, 0x0800, 0x0800, 0x0800, 0x1000, 0x2000, // )

0x1000, 0x3800, 0x1000, 0x2800, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, // \*

0x0000, 0x0000, 0x1000, 0x1000, 0x7C00, 0x1000, 0x1000, 0x0000, 0x0000, 0x0000, // +

0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x1000, 0x1000, 0x1000, // ,

0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x3800, 0x0000, 0x0000, 0x0000, 0x0000, // -

0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x1000, 0x0000, 0x0000, // .

0x0800, 0x0800, 0x1000, 0x1000, 0x1000, 0x1000, 0x2000, 0x2000, 0x0000, 0x0000, // /

0x3800, 0x4400, 0x4400, 0x5400, 0x4400, 0x4400, 0x4400, 0x3800, 0x0000, 0x0000, // 0

0x1000, 0x3000, 0x5000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x0000, 0x0000, // 1

0x3800, 0x4400, 0x4400, 0x0400, 0x0800, 0x1000, 0x2000, 0x7C00, 0x0000, 0x0000, // 2

0x3800, 0x4400, 0x0400, 0x1800, 0x0400, 0x0400, 0x4400, 0x3800, 0x0000, 0x0000, // 3

0x0800, 0x1800, 0x2800, 0x2800, 0x4800, 0x7C00, 0x0800, 0x0800, 0x0000, 0x0000, // 4

0x7C00, 0x4000, 0x4000, 0x7800, 0x0400, 0x0400, 0x4400, 0x3800, 0x0000, 0x0000, // 5

0x3800, 0x4400, 0x4000, 0x7800, 0x4400, 0x4400, 0x4400, 0x3800, 0x0000, 0x0000, // 6

0x7C00, 0x0400, 0x0800, 0x1000, 0x1000, 0x2000, 0x2000, 0x2000, 0x0000, 0x0000, // 7

0x3800, 0x4400, 0x4400, 0x3800, 0x4400, 0x4400, 0x4400, 0x3800, 0x0000, 0x0000, // 8

0x3800, 0x4400, 0x4400, 0x4400, 0x3C00, 0x0400, 0x4400, 0x3800, 0x0000, 0x0000, // 9

0x0000, 0x0000, 0x1000, 0x0000, 0x0000, 0x0000, 0x0000, 0x1000, 0x0000, 0x0000, // :

0x0000, 0x0000, 0x0000, 0x1000, 0x0000, 0x0000, 0x0000, 0x1000, 0x1000, 0x1000, // ;

0x0000, 0x0000, 0x0C00, 0x3000, 0x4000, 0x3000, 0x0C00, 0x0000, 0x0000, 0x0000, // <

0x0000, 0x0000, 0x0000, 0x7C00, 0x0000, 0x7C00, 0x0000, 0x0000, 0x0000, 0x0000, // =

0x0000, 0x0000, 0x6000, 0x1800, 0x0400, 0x1800, 0x6000, 0x0000, 0x0000, 0x0000, // >

0x3800, 0x4400, 0x0400, 0x0800, 0x1000, 0x1000, 0x0000, 0x1000, 0x0000, 0x0000, // ?

0x3800, 0x4400, 0x4C00, 0x5400, 0x5C00, 0x4000, 0x4000, 0x3800, 0x0000, 0x0000, // @

0x1000, 0x2800, 0x2800, 0x2800, 0x2800, 0x7C00, 0x4400, 0x4400, 0x0000, 0x0000, // A

0x7800, 0x4400, 0x4400, 0x7800, 0x4400, 0x4400, 0x4400, 0x7800, 0x0000, 0x0000, // B

0x3800, 0x4400, 0x4000, 0x4000, 0x4000, 0x4000, 0x4400, 0x3800, 0x0000, 0x0000, // C

0x7000, 0x4800, 0x4400, 0x4400, 0x4400, 0x4400, 0x4800, 0x7000, 0x0000, 0x0000, // D

0x7C00, 0x4000, 0x4000, 0x7C00, 0x4000, 0x4000, 0x4000, 0x7C00, 0x0000, 0x0000, // E

0x7C00, 0x4000, 0x4000, 0x7800, 0x4000, 0x4000, 0x4000, 0x4000, 0x0000, 0x0000, // F

0x3800, 0x4400, 0x4000, 0x4000, 0x5C00, 0x4400, 0x4400, 0x3800, 0x0000, 0x0000, // G

0x4400, 0x4400, 0x4400, 0x7C00, 0x4400, 0x4400, 0x4400, 0x4400, 0x0000, 0x0000, // H

0x3800, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x3800, 0x0000, 0x0000, // I

0x0400, 0x0400, 0x0400, 0x0400, 0x0400, 0x0400, 0x4400, 0x3800, 0x0000, 0x0000, // J

0x4400, 0x4800, 0x5000, 0x6000, 0x5000, 0x4800, 0x4800, 0x4400, 0x0000, 0x0000, // K

0x4000, 0x4000, 0x4000, 0x4000, 0x4000, 0x4000, 0x4000, 0x7C00, 0x0000, 0x0000, // L

0x4400, 0x6C00, 0x6C00, 0x5400, 0x4400, 0x4400, 0x4400, 0x4400, 0x0000, 0x0000, // M

0x4400, 0x6400, 0x6400, 0x5400, 0x5400, 0x4C00, 0x4C00, 0x4400, 0x0000, 0x0000, // N

0x3800, 0x4400, 0x4400, 0x4400, 0x4400, 0x4400, 0x4400, 0x3800, 0x0000, 0x0000, // O

0x7800, 0x4400, 0x4400, 0x4400, 0x7800, 0x4000, 0x4000, 0x4000, 0x0000, 0x0000, // P

0x3800, 0x4400, 0x4400, 0x4400, 0x4400, 0x4400, 0x5400, 0x3800, 0x0400, 0x0000, // Q

0x7800, 0x4400, 0x4400, 0x4400, 0x7800, 0x4800, 0x4800, 0x4400, 0x0000, 0x0000, // R

0x3800, 0x4400, 0x4000, 0x3000, 0x0800, 0x0400, 0x4400, 0x3800, 0x0000, 0x0000, // S

0x7C00, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x0000, 0x0000, // T

0x4400, 0x4400, 0x4400, 0x4400, 0x4400, 0x4400, 0x4400, 0x3800, 0x0000, 0x0000, // U

0x4400, 0x4400, 0x4400, 0x2800, 0x2800, 0x2800, 0x1000, 0x1000, 0x0000, 0x0000, // V

0x4400, 0x4400, 0x5400, 0x5400, 0x5400, 0x6C00, 0x2800, 0x2800, 0x0000, 0x0000, // W

0x4400, 0x2800, 0x2800, 0x1000, 0x1000, 0x2800, 0x2800, 0x4400, 0x0000, 0x0000, // X

0x4400, 0x4400, 0x2800, 0x2800, 0x1000, 0x1000, 0x1000, 0x1000, 0x0000, 0x0000, // Y

0x7C00, 0x0400, 0x0800, 0x1000, 0x1000, 0x2000, 0x4000, 0x7C00, 0x0000, 0x0000, // Z

0x1800, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1800, // [

0x2000, 0x2000, 0x1000, 0x1000, 0x1000, 0x1000, 0x0800, 0x0800, 0x0000, 0x0000, /\* \ \*/

0x3000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x3000, // ]

0x1000, 0x2800, 0x2800, 0x4400, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, // ^

0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0xFE00, // \_

0x2000, 0x1000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, // `

0x0000, 0x0000, 0x3800, 0x4400, 0x3C00, 0x4400, 0x4C00, 0x3400, 0x0000, 0x0000, // a

0x4000, 0x4000, 0x5800, 0x6400, 0x4400, 0x4400, 0x6400, 0x5800, 0x0000, 0x0000, // b

0x0000, 0x0000, 0x3800, 0x4400, 0x4000, 0x4000, 0x4400, 0x3800, 0x0000, 0x0000, // c

0x0400, 0x0400, 0x3400, 0x4C00, 0x4400, 0x4400, 0x4C00, 0x3400, 0x0000, 0x0000, // d

0x0000, 0x0000, 0x3800, 0x4400, 0x7C00, 0x4000, 0x4400, 0x3800, 0x0000, 0x0000, // e

0x0C00, 0x1000, 0x7C00, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x0000, 0x0000, // f

0x0000, 0x0000, 0x3400, 0x4C00, 0x4400, 0x4400, 0x4C00, 0x3400, 0x0400, 0x7800, // g

0x4000, 0x4000, 0x5800, 0x6400, 0x4400, 0x4400, 0x4400, 0x4400, 0x0000, 0x0000, // h

0x1000, 0x0000, 0x7000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x0000, 0x0000, // i

0x1000, 0x0000, 0x7000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0xE000, // j

0x4000, 0x4000, 0x4800, 0x5000, 0x6000, 0x5000, 0x4800, 0x4400, 0x0000, 0x0000, // k

0x7000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x0000, 0x0000, // l

0x0000, 0x0000, 0x7800, 0x5400, 0x5400, 0x5400, 0x5400, 0x5400, 0x0000, 0x0000, // m

0x0000, 0x0000, 0x5800, 0x6400, 0x4400, 0x4400, 0x4400, 0x4400, 0x0000, 0x0000, // n

0x0000, 0x0000, 0x3800, 0x4400, 0x4400, 0x4400, 0x4400, 0x3800, 0x0000, 0x0000, // o

0x0000, 0x0000, 0x5800, 0x6400, 0x4400, 0x4400, 0x6400, 0x5800, 0x4000, 0x4000, // p

0x0000, 0x0000, 0x3400, 0x4C00, 0x4400, 0x4400, 0x4C00, 0x3400, 0x0400, 0x0400, // q

0x0000, 0x0000, 0x5800, 0x6400, 0x4000, 0x4000, 0x4000, 0x4000, 0x0000, 0x0000, // r

0x0000, 0x0000, 0x3800, 0x4400, 0x3000, 0x0800, 0x4400, 0x3800, 0x0000, 0x0000, // s

0x2000, 0x2000, 0x7800, 0x2000, 0x2000, 0x2000, 0x2000, 0x1800, 0x0000, 0x0000, // t

0x0000, 0x0000, 0x4400, 0x4400, 0x4400, 0x4400, 0x4C00, 0x3400, 0x0000, 0x0000, // u

0x0000, 0x0000, 0x4400, 0x4400, 0x2800, 0x2800, 0x2800, 0x1000, 0x0000, 0x0000, // v

0x0000, 0x0000, 0x5400, 0x5400, 0x5400, 0x6C00, 0x2800, 0x2800, 0x0000, 0x0000, // w

0x0000, 0x0000, 0x4400, 0x2800, 0x1000, 0x1000, 0x2800, 0x4400, 0x0000, 0x0000, // x

0x0000, 0x0000, 0x4400, 0x4400, 0x2800, 0x2800, 0x1000, 0x1000, 0x1000, 0x6000, // y

0x0000, 0x0000, 0x7C00, 0x0800, 0x1000, 0x2000, 0x4000, 0x7C00, 0x0000, 0x0000, // z

0x1800, 0x1000, 0x1000, 0x1000, 0x2000, 0x2000, 0x1000, 0x1000, 0x1000, 0x1800, // {

0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, // |

0x3000, 0x1000, 0x1000, 0x1000, 0x0800, 0x0800, 0x1000, 0x1000, 0x1000, 0x3000, // }

0x0000, 0x0000, 0x0000, 0x7400, 0x4C00, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, // ~

};

FontDef Font\_7x10 = {7,10,Font7x10};

static const uint8\_t u8InitCmdList[] = {

// Command Length Data

0xB1, 0x03, 0x01, 0x2C, 0x2D, // Frame Rate Control (In normal mode/ Full colors)

0xB2, 0x03, 0x01, 0x2C, 0x2D, // Frame Rate Control (In Idle mode/ 8-colors)

0xB3, 0x06, 0x01, 0x2C, 0x2D, 0x01, 0x2C, 0x2D, // Frame Rate Control (In Partial mode/ full colors)

0xB4, 0x01, 0x07, // Display Inversion Control

0xC0, 0x03, 0xA2, 0x02, 0x84, // Power Control 1

0xC1, 0x01, 0xC5, // Power Control 2

0xC2, 0x02, 0x0A, 0x00, // Power Control 3 (in Normal mode/ Full colors)

0xC3, 0x02, 0x8A, 0x2A, // Power Control 4 (in Idle mode/ 8-colors)

0xC4, 0x02, 0x8A, 0xEE, // Power Control 5 (in Partial mode/ full colors)

0xC5, 0x01, 0x0E, // VCOM Control 1

0xE0, 0x10, 0x02, 0x1C, 0x07, 0x12, 0x37, 0x32, 0x29, 0x2D,0x29, 0x25, 0x2B, 0x39, 0x00, 0x01, 0x03, 0x10,

0xE1, 0x10, 0x03, 0x1D, 0x07, 0x06, 0x2E, 0x2C, 0x29, 0x2D,0x2E, 0x2E, 0x37, 0x3F, 0x00, 0x00, 0x02, 0x10,

LCD\_CMD\_END, LCD\_CMD\_END

};

void writeCMDTFT(uint8\_t cmd){

HAL\_GPIO\_WritePin(GPIOB,GPIO\_PIN\_0,0); // CS = 0

HAL\_GPIO\_WritePin(GPIOB,GPIO\_PIN\_1,0); // A0 = 0 -> CMD

HAL\_SPI\_Transmit(&hspi1, &cmd, 1, 1000);

HAL\_GPIO\_WritePin(GPIOB,GPIO\_PIN\_0,1);// CS = 1

}

void writeDataTFT(uint8\_t data) {

HAL\_GPIO\_WritePin(GPIOB,GPIO\_PIN\_0,0);

HAL\_GPIO\_WritePin(GPIOB,GPIO\_PIN\_1,1);// A0 = 1 -> Data

HAL\_SPI\_Transmit(&hspi1, &data, 1, 1000);

HAL\_GPIO\_WritePin(GPIOB,GPIO\_PIN\_0,1);

void sendCMDList(const uint8\_t\* cmdList){

uint8\_t index = 0;

uint8\_t cmd = 0;

uint8\_t num = 0;

while(1){

cmd = \*cmdList++;

num = \*cmdList++;

if(cmd == LCD\_CMD\_END){

break;

}else{

writeCMDTFT(cmd);

for(index = 0 ; index < num ; index++){

writeDataTFT(\*cmdList++);

}

}

}

}

void setPos(uint8\_t x1, uint8\_t y1, uint8\_t x2, uint8\_t y2){

// Column (X)

writeCMDTFT(0x2A);

writeDataTFT(0x00);

writeDataTFT(x1);

writeDataTFT(0x00);

writeDataTFT(x2);

// Row (Y)

writeCMDTFT(0x2B);

writeDataTFT(0x00);

writeDataTFT(y1);

writeDataTFT(0x00);

writeDataTFT(y2);

}

void fullDisplay(uint16\_t color) {

///////Bao cao He thong Nhung - Hoc vien Ky thuat mat ma - DT602xx/////////

setPos(0, 0, 127, 159);

///////Bao cao He thong Nhung - Hoc vien Ky thuat mat ma - DT602xx/////////

writeCMDTFT(0x2C);

///////Bao cao He thong Nhung - Hoc vien Ky thuat mat ma - DT602xx/////////

HAL\_GPIO\_WritePin(GPIOB, GPIO\_PIN\_0, 0);

HAL\_GPIO\_WritePin(GPIOB, GPIO\_PIN\_1, 1);

///////Bao cao He thong Nhung - Hoc vien Ky thuat mat ma - DT602xx/////////

uint8\_t colorData[2];

colorData[0] = color >> 8; // Byte cao trước

colorData[1] = color & 0xFF; // Byte thấp sau

///////Bao cao He thong Nhung - Hoc vien Ky thuat mat ma - DT602xx/////////

for (int i = 0; i < 128 \* 160; i++) {

///////Bao cao He thong Nhung - Hoc vien Ky thuat mat ma - DT602xx/////////

HAL\_SPI\_Transmit(&hspi1, colorData, 2, 100);

}

///////Bao cao He thong Nhung - Hoc vien Ky thuat mat ma - DT602xx/////////

HAL\_GPIO\_WritePin(GPIOB, GPIO\_PIN\_0, 1);

}

void drawPixel(uint8\_t x, uint8\_t y, uint16\_t color) {

///////Bao cao He thong Nhung - Hoc vien Ky thuat mat ma - DT602xx/////////

if (x >= 128 || y >= 160) {

return;

}

///////Bao cao He thong Nhung - Hoc vien Ky thuat mat ma - DT602xx/////////

setPos(x, y, x, y); // Sửa x+1, y+1 thành x, y cho đúng

///////Bao cao He thong Nhung - Hoc vien Ky thuat mat ma - DT602xx/////////

writeCMDTFT(0x2C);

///////Bao cao He thong Nhung - Hoc vien Ky thuat mat ma - DT602xx/////////

uint8\_t colorData[2];

colorData[0] = color >> 8; // Byte cao trước

colorData[1] = color & 0xFF; // Byte thấp sau

///////Bao cao He thong Nhung - Hoc vien Ky thuat mat ma - DT602xx/////////

HAL\_GPIO\_WritePin(GPIOB, GPIO\_PIN\_0, 0); // Kéo CS xuống THẤP

HAL\_GPIO\_WritePin(GPIOB, GPIO\_PIN\_1, 1); // A0 = 1 (chế độ Data)

// Gửi cả 2 byte màu trong khi CS vẫn đang ở mức thấp

HAL\_SPI\_Transmit(&hspi1, colorData, 2, 100);

///////Bao cao He thong Nhung - Hoc vien Ky thuat mat ma - DT602xx/////////

HAL\_GPIO\_WritePin(GPIOB, GPIO\_PIN\_0, 1); // Kéo CS lên CAO

}

void drawChar(uint8\_t x, uint8\_t y, char ch, FontDef font, uint16\_t color, uint16\_t bg){

uint16\_t i,j;

uint16\_t pixelData;

for( i = 0 ; i < font.heigth ; i++){

pixelData = font.data[(ch - 32)\*font.heigth + i ];

for(j = 0 ; j < font.width ; j++){

if((pixelData << j) & 0x8000){

drawPixel(x+j, y+i, color);

}else {

drawPixel(x+j, y+i, bg);

}

}

}

}

void drawString (uint8\_t x, uint8\_t y, char \*str, FontDef font, uint16\_t color, uint16\_t bg) {

while(\*str){

drawChar(x,y,\*str,font,color,bg);

x += font.width;

str++;

}

}

///////Bao cao He thong Nhung - Hoc vien Ky thuat mat ma - DT602xx/////////

void writeDataTFT\_16(uint16\_t data) {

uint8\_t buffer[2];

buffer[1] = data & 0xFF;

buffer[0] = data >> 8; // G?i byte cao tru?c

HAL\_GPIO\_WritePin(GPIOB, GPIO\_PIN\_0, 0); // CS = 0

HAL\_GPIO\_WritePin(GPIOB, GPIO\_PIN\_1, 1); // A0 = 1 -> Data

HAL\_SPI\_Transmit(&hspi1, buffer, 2, 100); // G?i 2 byte

HAL\_GPIO\_WritePin(GPIOB, GPIO\_PIN\_0, 1); // CS = 1

}

void drawImage(uint8\_t x, uint8\_t y, uint8\_t width, uint8\_t height, const uint8\_t\* imageData) {

///////Bao cao He thong Nhung - Hoc vien Ky thuat mat ma - DT602xx/////////

setPos(x, y, x + width - 1, y + height - 1);

///////Bao cao He thong Nhung - Hoc vien Ky thuat mat ma - DT602xx/////////

writeCMDTFT(0x2C);

///////Bao cao He thong Nhung - Hoc vien Ky thuat mat ma - DT602xx/////////

uint32\_t dataSize = (uint32\_t)width \* (uint32\_t)height \* 2; // Mỗi pixel có 2 byte

///////Bao cao He thong Nhung - Hoc vien Ky thuat mat ma - DT602xx/////////

HAL\_GPIO\_WritePin(GPIOB, GPIO\_PIN\_0, 0); // CS = 0 (Kích hoạt chip)

HAL\_GPIO\_WritePin(GPIOB, GPIO\_PIN\_1, 1); // A0 = 1 -> Chế độ gửi Dữ liệu

///////Bao cao He thong Nhung - Hoc vien Ky thuat mat ma - DT602xx/////////

HAL\_SPI\_Transmit(&hspi1, (uint8\_t\*)imageData, dataSize, HAL\_MAX\_DELAY);

///////Bao cao He thong Nhung - Hoc vien Ky thuat mat ma - DT602xx/////////

HAL\_GPIO\_WritePin(GPIOB, GPIO\_PIN\_0, 1); // CS = 1 (Ngừng kích hoạt chip)

}

void initTFT() {

// Hardware Reset

HAL\_GPIO\_WritePin(GPIOB, GPIO\_PIN\_10, 0); // RST low

HAL\_Delay(20);

HAL\_GPIO\_WritePin(GPIOB, GPIO\_PIN\_10, 1); // RST high

HAL\_Delay(200);

// Software Reset

writeCMDTFT(0x01);

HAL\_Delay(150)

// Sleep Out

writeCMDTFT(0x11);

HAL\_Delay(500);

sendCMDList(u8InitCmdList);

//Memory Data Access Control

writeCMDTFT(0x36);

writeDataTFT(0x00);

//Interface Pixel Format

writeCMDTFT(0x3A);

writeDataTFT(0x05);

//Display inversion off

writeCMDTFT(0x20);

// Set drawing window to full screen

setPos(0, 0, 127, 159);

// Display On

writeCMDTFT(0x29);

HAL\_Delay(100);

}

/\* USER CODE END 0 \*/

/\*\*

\* @brief The application entry point.

\* @retval int

\*/

int main(void)

{

/\* USER CODE BEGIN 1 \*/

/\* USER CODE END 1 \*/

/\* MCU Configuration-----------------------------------------------------\*/

/\* Reset of all peripherals, Initializes the Flash interface and the Systick. \*/

HAL\_Init();

/\* USER CODE BEGIN Init \*/

/\* USER CODE END Init \*/

/\* Configure the system clock \*/

SystemClock\_Config();

/\* USER CODE BEGIN SysInit \*/

/\* USER CODE END SysInit \*/

/\* Initialize all configured peripherals \*/

MX\_GPIO\_Init();

MX\_SPI1\_Init();

/\* USER CODE BEGIN 2 \*/

HAL\_GPIO\_WritePin(GPIOB,GPIO\_PIN\_0,1);

Font\_7x10.width = 7;

Font\_7x10.heigth = 10;

Font\_7x10.data = Font7x10;

initTFT();

uint16\_t colors[] = {RED, GREEN, BLUE, YELLOW, CYAN, MAGENTA, WHITE};

uint8\_t mode = -1;

uint8\_t buttonState = 0;

uint8\_t lastState = 0;

int num\_colors = 7;

int color\_index = 0;

int8\_t last\_mode = -1;

fullDisplay(0x0000); // Xóa màn hình

drawString(10, 20, "DE TAI: UNG DUNG", Font\_7x10, 0xFFFF, 0x0000);

drawString(10, 40, "GIAO THUC SPI", Font\_7x10, 0xFFFF, 0x0000);

drawString(10, 60, "TRONG GIAO TIEP", Font\_7x10, 0xFFFF, 0x0000);

drawString(10, 80, "VOI MAN HINH TFT ", Font\_7x10, 0xFFFF, 0x0000);

/\* USER CODE END 2 \*/

/\* Infinite loop \*/

/\* USER CODE BEGIN WHILE \*/

while (1)

{

buttonState = HAL\_GPIO\_ReadPin(GPIOA, GPIO\_PIN\_2);

// Phát hiện nhấn 1 lần (từ 0 → 1) để thay đổi mode

if (buttonState == GPIO\_PIN\_SET && lastState == GPIO\_PIN\_RESET)

{

mode++;

if (mode > 2) mode = 0;

HAL\_Delay(50);

}

lastState = buttonState;

///////Bao cao He thong Nhung - Hoc vien Ky thuat mat ma - DT602xx/////////

if (mode != last\_mode)

{

// --- Bắt đầu đoạn code thay thế cho switch-case ---

if (mode == 0)

{

fullDisplay(colors[color\_index]);

color\_index++;

if (color\_index >= num\_colors) color\_index = 0;

}

else if (mode == 1)

{

fullDisplay(0xFFFF); // Xóa màn hình

drawString(40, 20,"NHOM 13", Font\_7x10, 0x0000, 0xFFFF);

drawString(2, 40, "HA.V.DAT-DT060209", Font\_7x10, 0x0000, 0xFFFF);

drawString(2, 60, "D.H.DANG-DT060206", Font\_7x10, 0x0000, 0xFFFF);

drawString(2, 80, "NG.M.LAN-DT060231", Font\_7x10, 0x0000, 0xFFFF);

drawString(2, 100,"NG.Q.HUY-DT060225", Font\_7x10, 0x0000, 0xFFFF);

}

else if (mode == 2)

{

fullDisplay(0x0000); // Xóa màn hình

drawImage( 0,0,128,160, gImage\_hvktmm);

}

last\_mode = mode;

}

// Xử lý riêng cho chế độ hiệu ứng màu để nó có thể lặp lại liên tục

if (mode == 0)

{

fullDisplay(colors[color\_index]);

color\_index++;

if (color\_index >= num\_colors) color\_index = 0;

HAL\_Delay(200);

}

/\* USER CODE BEGIN 3 \*/

}

/\* USER CODE END 3 \*/

}

/\*\*

\* @brief System Clock Configuration

\* @retval None

\*/

void SystemClock\_Config(void)

{

RCC\_OscInitTypeDef RCC\_OscInitStruct = {0};

RCC\_ClkInitTypeDef RCC\_ClkInitStruct = {0};

/\*\* Initializes the RCC Oscillators according to the specified parameters

\* in the RCC\_OscInitTypeDef structure.

\*/

RCC\_OscInitStruct.OscillatorType = RCC\_OSCILLATORTYPE\_HSI;

RCC\_OscInitStruct.HSIState = RCC\_HSI\_ON;

RCC\_OscInitStruct.HSICalibrationValue = RCC\_HSICALIBRATION\_DEFAULT;

RCC\_OscInitStruct.PLL.PLLState = RCC\_PLL\_ON;

RCC\_OscInitStruct.PLL.PLLSource = RCC\_PLLSOURCE\_HSI\_DIV2;

RCC\_OscInitStruct.PLL.PLLMUL = RCC\_PLL\_MUL6;

if (HAL\_RCC\_OscConfig(&RCC\_OscInitStruct) != HAL\_OK)

{

Error\_Handler();

}

/\*\* Initializes the CPU, AHB and APB buses clocks

\*/

RCC\_ClkInitStruct.ClockType = RCC\_CLOCKTYPE\_HCLK|RCC\_CLOCKTYPE\_SYSCLK

|RCC\_CLOCKTYPE\_PCLK1|RCC\_CLOCKTYPE\_PCLK2;

RCC\_ClkInitStruct.SYSCLKSource = RCC\_SYSCLKSOURCE\_PLLCLK;

RCC\_ClkInitStruct.AHBCLKDivider = RCC\_SYSCLK\_DIV1;

RCC\_ClkInitStruct.APB1CLKDivider = RCC\_HCLK\_DIV1;

RCC\_ClkInitStruct.APB2CLKDivider = RCC\_HCLK\_DIV1;

if (HAL\_RCC\_ClockConfig(&RCC\_ClkInitStruct, FLASH\_LATENCY\_0) != HAL\_OK)

{

Error\_Handler();

}

}

/\*\*

\* @brief SPI1 Initialization Function

\* @param None

\* @retval None

\*/

static void MX\_SPI1\_Init(void)

{

/\* USER CODE BEGIN SPI1\_Init 0 \*/

/\* USER CODE END SPI1\_Init 0 \*/

/\* USER CODE BEGIN SPI1\_Init 1 \*/

/\* USER CODE END SPI1\_Init 1 \*/

/\* SPI1 parameter configuration\*/

hspi1.Instance = SPI1;

hspi1.Init.Mode = SPI\_MODE\_MASTER;

hspi1.Init.Direction = SPI\_DIRECTION\_2LINES;

hspi1.Init.DataSize = SPI\_DATASIZE\_8BIT;

hspi1.Init.CLKPolarity = SPI\_POLARITY\_LOW;

hspi1.Init.CLKPhase = SPI\_PHASE\_1EDGE;

hspi1.Init.NSS = SPI\_NSS\_SOFT;

hspi1.Init.BaudRatePrescaler = SPI\_BAUDRATEPRESCALER\_2;

hspi1.Init.FirstBit = SPI\_FIRSTBIT\_MSB;

hspi1.Init.TIMode = SPI\_TIMODE\_DISABLE;

hspi1.Init.CRCCalculation = SPI\_CRCCALCULATION\_DISABLE;

hspi1.Init.CRCPolynomial = 10;

if (HAL\_SPI\_Init(&hspi1) != HAL\_OK)

{

Error\_Handler();

}

/\* USER CODE BEGIN SPI1\_Init 2 \*/

/\* USER CODE END SPI1\_Init 2 \*/

}

/\*\*

\* @brief GPIO Initialization Function

\* @param None

\* @retval None

\*/

static void MX\_GPIO\_Init(void)

{

GPIO\_InitTypeDef GPIO\_InitStruct = {0};

/\* USER CODE BEGIN MX\_GPIO\_Init\_1 \*/

/\* USER CODE END MX\_GPIO\_Init\_1 \*/

/\* GPIO Ports Clock Enable \*/

\_\_HAL\_RCC\_GPIOD\_CLK\_ENABLE();

\_\_HAL\_RCC\_GPIOA\_CLK\_ENABLE();

\_\_HAL\_RCC\_GPIOB\_CLK\_ENABLE();

/\*Configure GPIO pin Output Level \*/

HAL\_GPIO\_WritePin(GPIOB, GPIO\_PIN\_0|GPIO\_PIN\_1|GPIO\_PIN\_10, GPIO\_PIN\_RESET);

/\*Configure GPIO pin : PA2 \*/

GPIO\_InitStruct.Pin = GPIO\_PIN\_2;

GPIO\_InitStruct.Mode = GPIO\_MODE\_INPUT;

GPIO\_InitStruct.Pull = GPIO\_PULLDOWN;

HAL\_GPIO\_Init(GPIOA, &GPIO\_InitStruct);

/\*Configure GPIO pins : PB0 PB1 PB10 \*/

GPIO\_InitStruct.Pin = GPIO\_PIN\_0|GPIO\_PIN\_1|GPIO\_PIN\_10;

GPIO\_InitStruct.Mode = GPIO\_MODE\_OUTPUT\_PP;

GPIO\_InitStruct.Pull = GPIO\_NOPULL;

GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_LOW;

HAL\_GPIO\_Init(GPIOB, &GPIO\_InitStruct);

/\* USER CODE BEGIN MX\_GPIO\_Init\_2 \*/

/\* USER CODE END MX\_GPIO\_Init\_2 \*/

}

/\* USER CODE BEGIN 4 \*/

/\* USER CODE END 4 \*/

/\*\*

\* @brief This function is executed in case of error occurrence.

\* @retval None

\*/

void Error\_Handler(void)

{

/\* USER CODE BEGIN Error\_Handler\_Debug \*/

/\* User can add his own implementation to report the HAL error return state \*/

\_\_disable\_irq();

while (1)

{

}

/\* USER CODE END Error\_Handler\_Debug \*/

}

#ifdef USE\_FULL\_ASSERT

/\*\*

\* @brief Reports the name of the source file and the source line number

\* where the assert\_param error has occurred.

\* @param file: pointer to the source file name

\* @param line: assert\_param error line source number

\* @retval None

\*/

void assert\_failed(uint8\_t \*file, uint32\_t line)

{

/\* USER CODE BEGIN 6 \*/

/\* User can add his own implementation to report the file name and line number,

ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) \*/

/\* USER CODE END 6 \*/

}

#endif /\* USE\_FULL\_ASSERT \*/