TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

Trường Điện – Điện tử

BÁO CÁO

Hệ Điều Hành Nhúng

Xây dựng device Driver cho SSD1306 OLED trên Raspberry Pi 5

Giảng viên hướng dẫn: TS. Phạm Doãn Tĩnh

Nhóm thực hiện: 12

Danh sách sinh viên:

Họ và tên	MSSV	Email
Đặng Trung Kiên	20224423	Kien.dt224423@sis.hust.edu.vn
Vũ Lâm Huy	20224438	Huy.vl224438@sis.hust.edu.vn
Nguyễn Quang Phú	20224451	Phu.nq224451@sis.hust.edu.vn
Nguyễn Duy Tùng	20224463	Tung.nd224463@sis.hust.edu.vn
Nguyễn Minh Đức	20203895	Duc.nm203895@sis.hust.edu.vn

Hà Nội, 4/2025

Table of Contents

LỜI MỞ ĐẦU

Môn học Hệ Điều Hành Nhúng đã cung cấp những kiến thức nền tảng về cách thức một hệ điều hành quản lý tài nguyên phần cứng, phần mềm và tương tác với người dùng cũng như các ứng dụng. Một trong những khía cạnh quan trọng là cách hệ điều hành cho phép các chương trình ở không gian người dùng (user-space) giao tiếp và điều khiển các thiết bị phần cứng ngoại vi. Đề tài "Xây Dựng Device Driver User-Space Điều Khiển Màn Hình OLED SSD1306 Giao Tiếp I2C Trên Raspberry Pi 5" được lựa chọn với mong muốn áp dụng kiến thức lý thuyết vào một bài toán thực tế, cụ thể là xây dựng một thư viện phần mềm để điều khiển một thiết bị phần cứng phổ biến.

Raspberry Pi, đặc biệt là phiên bản Raspberry Pi 5 mạnh mẽ, ngày càng được sử dụng rộng rãi trong các dự án nhúng, IoT và giáo dục nhờ vào tính linh hoạt và cộng đồng hỗ trợ lớn. Màn hình OLED SSD1306 là một lựa chọn hiển thị thông dụng cho các dự án này do kích thước nhỏ gọn, độ tương phản cao và tiêu thụ năng lượng thấp. Việc điều khiển màn hình này thông qua giao thức I2C là một kỹ năng hữu ích.

Mục tiêu chính của đề tài này bao gồm:

- Nghiên cứu sâu về nguyên lý hoạt động của chip điều khiển màn hình SSD1306 và cách thức giao tiếp qua chuẩn I2C.
- Tìm hiểu cách hệ điều hành Linux (cụ thể là Raspberry Pi OS trên Raspberry Pi 5) cung cấp cơ chế cho các ứng dụng user-space truy cập và điều khiển các thiết bị I2C.
- Thiết kế và triển khai một thư viện driver bằng ngôn ngữ C, cung cấp một API đơn giản cho phép người dùng thực hiện các thao tác cơ bản với màn hình SSD1306 như: khởi tạo, xóa màn hình, bật/tắt từng điểm ảnh (pixel), và hiển thị nội dung từ một bộ đệm (framebuffer).
- Nghiên cứu và triển khai một kernel module driver cơ bản cho cùng thiết bị SSD1306, qua giao tiếp I2C.

Phạm vi của đề tài tập trung vào việc xây dựng một device driver, sử dụng các system call tiêu chuẩn của Linux để tương tác với thiết bị I2C, cụ thể là màn hình OLED SSD1306 có độ phân giải 128x64 pixels. Driver sẽ được viết bằng ngôn ngữ C và thử nghiệm trên Raspberry Pi 5. Các chức năng đồ họa phức tạp hơn như vẽ đường thẳng, đường chéo,cuộn hay hiển thị văn bản có thể được xem xét như hướng phát triển tiềm năng.

Báo cáo này sẽ trình bày chi tiết từ cơ sở lý thuyết về các công nghệ liên quan, quá trình thiết kế và triển khai driver, cho đến các kết quả kiểm thử thực nghiệm và đánh giá. Giới thiệu các kiến thức nền tảng về Raspberry Pi 5, giao thức I2C và màn hình SSD1306. Đi sâu vào quá trình thiết kế kiến trúc và triển khai mã nguồn cho driver. Trình bày các phương pháp kiểm thử, kết quả đạt được cùng với những hình ảnh minh họa trực quan. Cuối cùng, sẽ đưa ra kết luận, những bài học kinh nghiệm rút ra và các hướng phát triển có thể cho đề tài trong tương lai.

CHƯƠNG 1: Tổng quan

1.1.Tổng quan đề tài

Ngày nay, các hệ thống nhúng đang ngày càng đóng vai trò quan trọng trong nhiều lĩnh vực như tự động hóa, điện tử tiêu dùng, y tế, công nghiệp và đặc biệt là Internet of Things (IoT). Trong các hệ thống này, màn hình hiển thị nhỏ gọn và tiết kiệm năng lượng như **OLED SSD1306** được sử dụng rộng rãi để hiển thị thông tin người dùng, trạng thái thiết bị hoặc dữ liệu cảm biến.

SSD1306 là một loại màn hình OLED đơn sắc 0.96 inch phổ biến, sử dụng giao tiếp **I2C** hoặc **SPI** để truyền dữ liệu từ vi điều khiển hoặc hệ điều hành nhúng đến màn hình. Việc giao tiếp với SSD1306 yêu cầu nắm vững kiến thức về bus I2C, cấu trúc dữ liệu hiển thị, và đặc biệt là cách hệ điều hành tương tác với phần cứng thông qua các driver thiết bị.

Trong khuôn khổ đề tài này, nhóm thực hiện sử dụng **Raspberry Pi 5 (8GB RAM)** – một nền tảng máy tính nhúng mạnh mẽ, nhỏ gọn, hỗ trợ hệ điều hành Linux, để xây dựng một driver cho thiết bị SSD1306.

Thông qua đề tài này, không chỉ áp dụng được kiến thức đã học trong môn **Hệ điều hành Nhúng** mà còn có cơ hội tiếp cận thực tế với quá trình phát triển phần mềm hệ thống trên nền tảng Linux, đặc biệt là trong môi trường kiến trúc ARM – một kiến trúc phổ biến cho các thiết bị nhúng hiện đại.

1.2. Tổng quan về Raspberry Pi 5

1.2.1. Giới thiệu Raspberry Pi 5

Raspberry Pi 5 là thế hệ máy tính đơn bảng (SBC – Single Board Computer) mới nhất được phát triển bởi Raspberry Pi Foundation, ra mắt vào cuối năm 2023. Phiên bản **Raspberry Pi 5 với RAM 8GB** mang lại sự cải tiến mạnh mẽ về hiệu năng và khả năng xử lý so với các thế hệ trước, đặc biệt là Raspberry Pi 4.

Với vi xử lý **Broadcom BCM2712** – một SoC 4 nhân ARM Cortex-A76 chạy ở xung nhịp 2.4GHz, Raspberry Pi 5 cho hiệu suất CPU nhanh hơn khoảng 2–3 lần so với Pi 4, đồng thời tích hợp GPU VideoCore VII cải tiến giúp xử lý đồ họa mượt mà hơn, hỗ trợ đầu ra 4K kép ở 60Hz.

Bên cạnh đó, Pi 5 (8GB) cung cấp bộ nhớ LPDDR4X dung lượng cao, cho phép chạy đa tác vụ, xử lý dữ liệu lớn hoặc các ứng dụng như máy chủ nhẹ, thị giác máy (computer vision), học máy (machine learning) hay lập trình nhúng với tốc độ phản hồi tốt hơn.

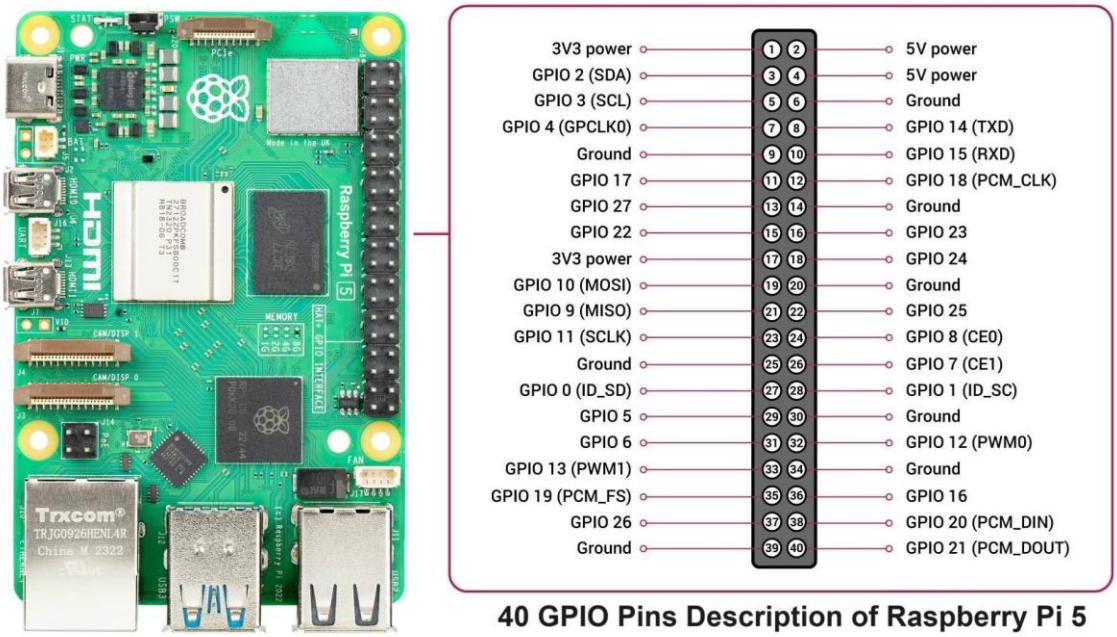
Về kết nối, Pi 5 hỗ trợ:

- 2 cổng **USB 3.0**, 2 cổng **USB 2.0**
- Cổng **Gigabit Ethernet**, hỗ trợ PoE+ qua HAT mở rộng
- 2 cổng **micro-HDMI 2.1**
- Giao tiếp **PCIe** qua cổng FFC mới
- Cổng **GPIO 40 chân** tương thích với các phụ kiện đời trước

Ngoài ra, hệ thống cấp nguồn qua cổng USB-C cũng được nâng cấp để cung cấp dòng lên đến 5V/5A, hỗ trợ các thiết bị ngoại vi mạnh hơn như ổ SSD, webcam, hoặc module cảm biến.

Với hiệu năng cao và sự linh hoạt, Raspberry Pi 5 (8GB) là lựa chọn lý tưởng cho các ứng dụng IoT, nhúng, robot, học tập, nghiên cứu và phát triển sản phẩm.

(Tham khảo: Trang chủ Raspberry Pi)



Hình 1.1: Sơ đồ chân GPIO của Raspberry Pi 5

-

1.2.2. Hệ điều hành Raspberry Pi OS

Raspberry Pi OS (trước đây gọi là Raspbian) là hệ điều hành chính thức được phát triển và duy trì bởi Raspberry Pi Foundation dành riêng cho các thiết bị Raspberry Pi. Đây là một bản phân phối Linux nhẹ, tối ưu hóa cho kiến trúc ARM, giúp khai thác tối đa hiệu suất phần cứng của dòng Raspberry Pi.

Raspberry Pi OS được xây dựng dựa trên nền tảng **Debian GNU/Linux**, kế thừa sự ổn định, bảo mật và kho phần mềm phong phú của Debian. Giao diện mặc định sử dụng môi trường **PIXEL Desktop** (Pi Improved Xwindows Environment, Lightweight) thân thiện, nhẹ nhàng, phù hợp với người mới bắt đầu làm quen với Linux cũng như người dùng chuyên sâu trong các ứng dụng nhúng.
Một số đặc điểm nổi bật của Raspberry Pi OS:
Tương thích tốt với phần cứng Raspberry Pi, từ Pi 1 đến Pi 5.
Hỗ trợ lập trình phong phú: đi kèm sẵn Python, GCC, Thonny IDE, Scratch, và nhiều công cụ phát triển khác.
Giao diện đồ họa dễ sử dụng, có thể hoạt động cả ở chế độ GUI và chế độ dòng lệnh (terminal).
Kho phần mềm APT lớn, dễ dàng cài đặt thêm các gói như OpenCV, Node-RED, Apache, MySQL, v.v.
Tích hợp công cụ cấu hình chuyên biệt như raspi-config, cho phép thiết lập Wi-Fi, SSH, I2C, SPI, camera, v.v. một cách đơn giản.

1.3. Giao thức I2C (Inter-Integrated Circuit)

1.3.1. Nguyên lý hoạt động

- I2C là một giao thức truyền thông nối tiếp đồng bộ, đa chủ (multi-master), đa tớ (multi-slave), được phát triển bởi Philips Semiconductors (nay là NXP Semiconductors).
- Sử dụng hai đường tín hiệu chính:
  - **SCL (Serial Clock Line):** Đường xung nhịp, do master tạo ra để đồng bộ hóa việc truyền dữ liệu.
  - **SDA (Serial Data Line):** Đường dữ liệu hai chiều, dùng để truyền dữ liệu giữa master và slave.
- Mỗi thiết bị slave trên bus I2C có một địa chỉ 7-bit (hoặc 10-bit) duy nhất.

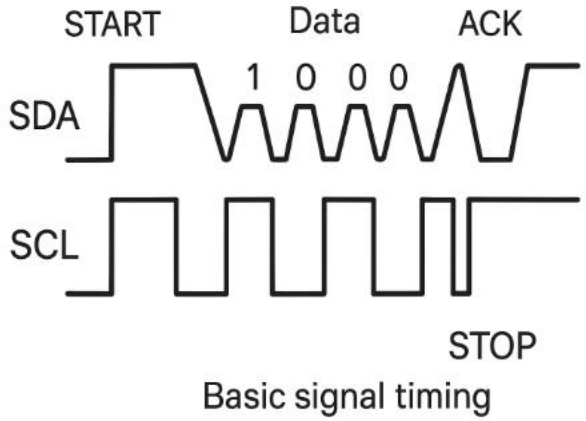
- Là một giao thức truyền thông nối tiếp, nên I2C sẽ truyền từng bit dữ liệu dọc theo đường dây SDA. Ngoài ra, tương tự như SPI, giao thức I2C có đầu ra dữ liệu bit được đồng bộ với việc lấy dữ liệu bằng tín hiệu đồng hồ được chia sẻ

giữa Master và Slave. Thiết bị Master có vai trò điều khiển tín hiệu đồng hồ này.

Số dây	2
Tốc độ tối đa	Chế độ tiêu chuẩn: 100 kbps
	Chế độ nhanh: 400 kbps
	Chế độ cực nhanh: 3,4 Mpps
	Ultra fast: 5 mbos
Đồng bộ hay không?	Đồng bộ
Serial hay Parallel?	Serial
Số lượng thiết bị Master tối đa	Không giới hạn
Số lượng thiết bị Slave tối đa	1008

Quy trình giao tiếp cơ bản:

1. **Start Condition:** Master kéo SDA xuống thấp trong khi SCL đang ở mức cao.
2. **Addressing:** Master gửi địa chỉ 7-bit của slave muốn giao tiếp, theo sau là bit Read/Write (R/W#). 0 cho Write (master gửi dữ liệu đến slave), 1 cho Read (master nhận dữ liệu từ slave).
3. **ACK/NACK (Acknowledge/Not Acknowledge):** Sau mỗi byte dữ liệu (địa chỉ hoặc data) được truyền, thiết bị nhận (slave khi master gửi địa chỉ/data, hoặc master khi slave gửi data) phải kéo SDA xuống thấp trong xung nhịp thứ 9 để báo hiệu ACK (đã nhận thành công). Nếu SDA vẫn ở mức cao, đó là NACK.
4. **Data Transfer:** Dữ liệu được truyền từng byte, MSB đi trước.
5. **Stop Condition:** Master kéo SDA lên cao trong khi SCL đang ở mức cao.



(Hình 1.2: Giản đồ tín hiệu cơ bản của giao thức I2C)

1.3.2. Giao tiếp I2C trên Raspberry Pi và Linux

- Raspberry Pi 5 hỗ trợ nhiều bus I2C. Bus I2C mặc định thường được sử dụng là i2c-1, kết nối với các chân GPIO2 (SDA) và GPIO3 (SCL).
- Trong Linux, mỗi bus I2C được biểu diễn như một file thiết bị trong thư mục /dev, ví dụ /dev/i2c-1.
- Để giao tiếp với một thiết bị I2C từ user-space bằng ngôn ngữ C, các system call và thao tác sau thường được sử dụng:
  - open(): Mở file thiết bị I2C (open("/dev/i2c-1", O_RDWR)).
  - ioctl(): Sử dụng với yêu cầu I2C_SLAVE để thiết lập địa chỉ của thiết bị slave mà chương trình muốn giao tiếp. Lệnh này yêu cầu header <linux/i2c-dev.h>.
  - write(): Gửi dữ liệu từ master đến slave.
  - read(): Đọc dữ liệu từ slave về master (không sử dụng trong driver SSD1306 này vì chỉ ghi).
  - close(): Đóng file thiết bị I2C.
- Cần có quyền truy cập (thường là root hoặc người dùng trong nhóm i2c) để thao tác với file /dev/i2c-X.

1.4. Màn hình OLED SSD1306

1.4.1. Công nghệ OLED và Chip SSD1306

- OLED (Organic Light Emitting Diode) là công nghệ hiển thị tự phát sáng, không cần đèn nền, cho độ tương phản rất cao (màu đen thực sự đen), góc nhìn rộng và thời gian đáp ứng nhanh.
- SSD1306 là một chip điều khiển (controller IC) đơn sắc cho màn hình OLED ma trận điểm, được sản xuất bởi Solomon Systech. Nó hỗ trợ nhiều giao diện giao tiếp như I2C, SPI và giao tiếp song song 8-bit. Đề tài này tập trung vào giao diện I2C.
- Các độ phân giải phổ biến của màn hình SSD1306 là 128x64 pixels và

128x32 pixels.

1.4.2. Giao tiếp I2C với SSD1306

- Khi giao tiếp qua I2C, SSD1306 hoạt động ở chế độ slave. Địa chỉ I2C mặc định thường là 0x3C hoặc 0x3D (tùy thuộc vào cấu hình chân SA0 của module).
- Sau khi master gửi địa chỉ slave và bit R/W# (luôn là 0 cho Write trong trường hợp này), byte tiếp theo được gửi là **Control Byte**.
- **Control Byte:**
  - Bit 7 (Co - Continuation bit): Nếu Co=0, chỉ có một control byte và các byte dữ liệu/lệnh theo sau. Nếu Co=1, byte tiếp theo cũng là một control byte. Trong trường hợp đơn giản, Co thường là 0.
  - Bit 6 (D/C# - Data/Command Select bit):
    - Nếu D/C#=0: Byte theo sau control byte là một lệnh (Command). Control byte sẽ là 0x00 (nếu Co=0).
    - Nếu D/C#=1: Byte theo sau control byte là dữ liệu (Data) để ghi vào GDDRAM. Control byte sẽ là 0x40 (nếu Co=0).
  - Các bit còn lại (5-0) thường là 0.

1.4.3. Bộ nhớ GDDRAM và Các Chế độ Địa chỉ

- SSD1306 có một bộ nhớ RAM nội gọi là Graphic Display Data RAM (GDDRAM) để lưu trữ dữ liệu hiển thị bitmap. Kích thước của GDDRAM phụ thuộc vào độ phân giải màn hình. Ví dụ, với màn hình 128x64, GDDRAM có kích thước 128 * (64/8) = 1024 bytes (vì mỗi byte điều khiển 8 pixel theo chiều dọc).
- GDDRAM được tổ chức thành các "trang" (pages). Mỗi trang cao 8 pixel. Với màn hình 128x64, có 8 trang (Page 0 đến Page 7). Với màn hình 128x32, có 4 trang.
- SSD1306 hỗ trợ ba chế độ địa chỉ bộ nhớ (Memory Addressing Modes), được thiết lập bằng lệnh 0x20:

1. **Page Addressing Mode (0x02 - mặc định khi reset):** Sau khi thiết lập trang (lệnh 0xB0 - 0xB7) và địa chỉ cột (lệnh 0x00-0x0F cho 4 bit thấp, 0x10-0x1F cho 4 bit cao), con trỏ cột sẽ tự động tăng sau mỗi lần ghi dữ liệu. Khi đến cuối cột, nó sẽ quay về đầu cột của trang đó. Cần thiết lập lại trang để ghi sang trang khác.
2. **Horizontal Addressing Mode (0x00):** Sau khi thiết lập địa chỉ cột bắt đầu/kết thúc (lệnh 0x21) và trang bắt đầu/kết thúc (lệnh 0x22), con trỏ cột sẽ tự động tăng. Khi đến cuối cột, nó sẽ reset về cột bắt đầu và con trỏ trang sẽ tăng lên. Chế độ này thuận tiện khi muốn ghi toàn bộ framebuffer một cách tuần tự.
3. **Vertical Addressing Mode (0x01):** Tương tự Horizontal mode, nhưng con trỏ trang sẽ tăng trước, sau đó đến con trỏ cột.

- Trong đề tài này, **Horizontal Addressing Mode (0x00)** được lựa chọn để đơn giản hóa việc ghi toàn bộ framebuffer từ bộ đệm của Raspberry Pi lên

GDDRAM của SSD1306.

1.4.4. Các Lệnh Điều Khiển Cơ bản của SSD1306

- (Liệt kê một số lệnh quan trọng đã sử dụng trong driver này:)
  - 0xAE: Tắt màn hình (Display OFF).
  - 0xAF: Bật màn hình (Display ON).
  - 0x81, [contrast_value]: Đặt độ tương phản.
  - 0xA4: Hiển thị nội dung từ RAM (Entire Display ON, Resume to RAM content).
  - 0xA5: Bật tất cả các pixel (Entire Display ON, Output ignores RAM content).
  - 0xA6: Chế độ hiển thị bình thường.
  - 0xA7: Chế độ hiển thị đảo ngược.
  - 0x20, [mode]: Đặt chế độ địa chỉ bộ nhớ.
  - 0x21, [start_col], [end_col]: Đặt địa chỉ cột (cho Horizontal/Vertical mode).
  - 0x22, [start_page], [end_page]: Đặt địa chỉ trang (cho Horizontal/Vertical mode).
  - 0x40 - 0x7F (Set Display Start Line): Đặt dòng bắt đầu hiển thị của RAM (cho scrolling).
  - 0x8D, [0x14/0x10]: Cài đặt Charge Pump (0x14 để bật, quan trọng cho OLED).
  - 0xA0/0xA1: Set Segment Re-map (ảnh hưởng hướng ngang).
  - 0xC0/0xC8: Set COM Output Scan Direction (ảnh hưởng hướng dọc).
  - 0xA8, [mux_ratio]: Đặt tỷ lệ MUX (Multiplex Ratio, thường là (chiều cao - 1)).
  - 0xD3, [offset]: Đặt Display Offset.
  - 0xD5, [ratio]: Đặt Display Clock Divide Ratio/Oscillator Frequency.
  - 0xDA, [config]: Đặt cấu hình chân COM.
    (Tham khảo: Datasheet SSD1306)

CHƯƠNG 2: THIẾT KẾ VÀ TRIỂN KHAI DEVICE DRIVER

2.1. Yêu cầu và Mục tiêu Thiết kế

Driver user-space cho màn hình SSD1306 cần đáp ứng các yêu cầu chức năng cơ bản sau:

- Khởi tạo và cấu hình màn hình SSD1306 để sẵn sàng hoạt động.
- Cung cấp khả năng xóa toàn bộ nội dung hiển thị trên màn hình.
- Cho phép bật/tắt từng điểm ảnh (pixel) tại một tọa độ (x,y) cụ thể.
- Quản lý một bộ đệm màn hình (framebuffer) trong bộ nhớ của Raspberry Pi.
- Cập nhật nội dung hiển thị trên màn hình OLED từ framebuffer.
- Giao tiếp với màn hình thông qua chuẩn I2C.

Mục tiêu thiết kế là tạo ra một thư viện C đơn giản, dễ sử dụng và dễ hiểu, thể hiện được các nguyên tắc tương tác với thiết bị phần cứng từ không gian người dùng trong hệ điều hành Linux.

2.2. Môi trường Phát triển

2.2.1. Phần cứng

- Máy tính đơn bo: Raspberry Pi 5 Model B (8GB RAM).
- Màn hình: Module OLED SSD1306, kích thước 0.96 inch, độ phân giải 128x64 pixels, giao tiếp I2C (địa chỉ mặc định 0x3C).
- Dây cắm (Jumper wires) để kết nối.
- Nguồn cấp cho Raspberry Pi.

2.2.2. Phần mềm

- Hệ điều hành: Raspberry Pi OS (64-bit) dựa trên Debian Bookworm
- Trình biên dịch: GCC (GNU Compiler Collection) đi kèm với Raspberry Pi OS.
- Ngôn ngữ lập trình: C.
- Các tệp header hệ thống cần thiết: <stdio.h>, <stdlib.h>, <stdint.h>, <fcntl.h>, <unistd.h>, <sys/ioctl.h>, <linux/i2c-dev.h>, <string.h>, <time.h>.
- Công cụ dòng lệnh: i2c-tools (để kiểm tra thiết bị I2C với i2cdetect).

2.3. Sơ đồ Kết nối Phần cứng

Việc kết nối màn hình OLED SSD1306 với Raspberry Pi 5 được thực hiện như sau:

- Chân VCC của SSD1306 nối với chân 3.3V trên Raspberry Pi (ví dụ: Pin 1).
- Chân GND của SSD1306 nối với chân GND trên Raspberry Pi (ví dụ: Pin 6).
- Chân SCL (Serial Clock) của SSD1306 nối với chân SCL1 (GPIO3, Pin 5) trên Raspberry Pi.
- Chân SDA (Serial Data) của SSD1306 nối với chân SDA1 (GPIO2, Pin 3) trên Raspberry Pi.

(Hình 2.1: Sơ đồ kết nối Raspberry Pi 5 với module SSD1306 I2C)

2.4. Thiết kế Cấu trúc Thư viện Driver

2.4.1. Tổ chức File và Các Thành phần Chính

- Toàn bộ mã nguồn driver được triển khai trong một file duy nhất là ssd1306_c_driver.c để đơn giản hóa việc biên dịch và quản lý cho mục đích bài tập.
- Các thành phần chính trong file bao gồm:
    1. **Khai báo hằng số và biến toàn cục:** Bao gồm địa chỉ I2C, đường dẫn bus I2C, kích thước màn hình, các mã lệnh của SSD1306, file descriptor cho bus I2C (i2c_fd), và mảng display_buffer đóng vai trò là framebuffer.
    2. **Các hàm giao tiếp I2C:** Nhóm hàm chịu trách nhiệm mở, đóng bus I2C, gửi lệnh và gửi dữ liệu đến SSD1306.
    3. **Hàm khởi tạo màn hình:** Chứa chuỗi lệnh cần thiết để cấu hình SSD1306.
    4. **Các hàm thao tác với framebuffer:** Xóa, đổ đầy, vẽ pixel.
    5. **Hàm hiển thị framebuffer:** Gửi nội dung framebuffer lên màn hình.
    6. **Hàm main():** Dùng để thử nghiệm và minh họa cách sử dụng các chức năng của driver.

2.4.2. Quản lý Framebuffer

- Một mảng uint8_t display_buffer[SSD1306_BUFFER_SIZE] được khai báo

toàn cục.

- SSD1306_BUFFER_SIZE được tính bằng (SSD1306_WIDTH * SSD1306_PAGES). Với màn hình 128x64 (8 trang), kích thước buffer là 128 * 8 = 1024 bytes.
- Mỗi bit trong display_buffer tương ứng với một pixel trên màn hình. Bit 0 của một byte trong buffer tương ứng với pixel ở hàng trên cùng của 8 pixel mà byte đó quản lý, và bit 7 tương ứng với pixel ở hàng dưới cùng.
- Khi ở Horizontal Addressing Mode, display_buffer được tổ chức sao cho display_buffer[x + (page * SSD1306_WIDTH)] là byte chứa pixel tại cột x của trang page.

2.5. Triển khai Chi tiết Các Hàm Chức năng

2.5.1. Khởi tạo và Đóng Giao tiếp I2C

- **int ssd1306_open_i2c():**
  - Sử dụng open(I2C_BUS_PATH, O_RDWR) để mở file thiết bị I2C. I2C_BUS_PATH được định nghĩa là "/dev/i2c-1".
  - Sau đó, ioctl(i2c_fd, I2C_SLAVE, SSD1306_I2C_ADDR) được gọi để thông báo cho kernel biết driver sẽ giao tiếp với thiết bị có địa chỉ SSD1306_I2C_ADDR (0x3C).
  - Lưu file descriptor trả về vào biến toàn cục i2c_fd. Xử lý lỗi nếu có.
- **void ssd1306_close_i2c():**
  - Đơn giản gọi close(i2c_fd) để giải phóng tài nguyên bus I2C.

2.5.2. Gửi Lệnh và Dữ liệu qua I2C

- **int ssd1306_send_single_command(uint8_t command):**
  - Tạo một buffer 2 byte. Byte đầu tiên là SSD1306_COMMAND_MODE (0x00). Byte thứ hai là command.
  - Sử dụng write(i2c_fd, buffer, 2) để gửi.
- **int ssd1306_send_command_1param(uint8_t command, uint8_t param1):** (Tương tự cho 2 params)
  - Tạo buffer 3 byte: [0x00, command, param1]. Gửi bằng write().
- **int ssd1306_send_data(const uint8_t *data, int len):**
  - Cấp phát động một buffer len + 1 byte. Byte đầu tiên là SSD1306_DATA_MODE (0x40).
  - Sao chép len byte từ con trỏ data vào phần còn lại của buffer.
  - Sử dụng write() để gửi toàn bộ buffer. Giải phóng bộ nhớ sau khi gửi.

2.5.3. Khởi tạo Màn hình SSD1306 (int ssd1306_init())

- Hàm này thực hiện một chuỗi các lệnh gửi đến SSD1306 để cấu hình nó. Các bước chính bao gồm:

1. SSD1306_DISPLAY_OFF: Tắt màn hình trong quá trình cấu hình.
2. SSD1306_SET_DISPLAY_CLOCK_DIV, 0x80: Đặt tần số oscillator.
3. SSD1306_SET_MULTIPLEX_RATIO, SSD1306_HEIGHT - 1: Đặt MUX ratio (63 cho màn 128x64).
4. SSD1306_SET_DISPLAY_OFFSET, 0x00: Không có offset hiển thị.
5. SSD1306_SET_DISPLAY_START_LINE_CMD | 0x00: Đặt dòng bắt đầu RAM là 0.
6. SSD1306_CHARGE_PUMP_SETTING, SSD1306_CHARGE_PUMP_ENABLE (0x14): Bật charge pump, rất quan trọng để OLED phát sáng.
7. SSD1306_SET_MEMORY_ADDR_MODE, 0x00: Chọn Horizontal Addressing Mode.
8. SSD1306_SET_SEGMENT_REMAP_REVERSE (0xA1) và SSD1306_SET_COM_OUTPUT_SCAN_DIR_REMAPPED (0xC8): Cấu hình hướng quét để hình ảnh hiển thị đúng chiều (tùy module).
9. SSD1306_SET_COM_PINS_HW_CONFIG, 0x12 (cho 128x64): Cấu hình chân COM.
10. SSD1306_SET_CONTRAST, 0xCF: Đặt độ tương phản.
11. SSD1306_SET_PRECHARGE_PERIOD, 0xF1.
12. SSD1306_SET_VCOMH_DESELECT_LEVEL, 0x40.
13. SSD1306_DISPLAY_ALL_ON_RESUME: Hiển thị nội dung từ GDDRAM.
14. SSD1306_NORMAL_DISPLAY: Chế độ hiển thị bình thường.
15. SSD1306_DISPLAY_ON: Bật màn hình.

- Hàm sử dụng các hàm ssd1306_send_single_command và ssd1306_send_command_1param để gửi các lệnh này.

2.5.4. Thao tác với Framebuffer (Xóa, Đổ đầy)

- **void ssd1306_clear_buffer():**
  - Sử dụng memset(display_buffer, 0x00, SSD1306_BUFFER_SIZE) để đặt tất cả các byte trong framebuffer thành 0, tương ứng với việc tắt tất cả các pixel.
- **void ssd1306_fill_buffer(uint8_t pattern):**
  - Sử dụng memset(display_buffer, pattern, SSD1306_BUFFER_SIZE) để đổ đầy framebuffer bằng một mẫu pattern cho trước. Ví dụ, 0xFF sẽ bật tất cả các pixel.

2.5.5. Vẽ Pixel (void ssd1306_draw_pixel(int x, int y, int color))

- Kiểm tra xem tọa độ (x,y) có nằm trong giới hạn màn hình không.
- Tính toán trang (page = y / 8).

- Tính toán vị trí bit trong byte của trang đó (bit_offset_in_page = y % 8).
- Tính toán chỉ số của byte trong display_buffer tương ứng với cột x và page đã tính: byte_index_in_buffer = x + (page * SSD1306_WIDTH).
- Nếu color là 1 (bật pixel): display_buffer[byte_index_in_buffer] |= (1 << bit_offset_in_page);
- Nếu color là 0 (tắt pixel): display_buffer[byte_index_in_buffer] &= ~(1 << bit_offset_in_page);

2.5.6. Hiển thị Framebuffer Lên Màn hình (void ssd1306_display_buffer())

- Hàm này chịu trách nhiệm gửi toàn bộ nội dung của display_buffer (1024 bytes) lên GDDRAM của SSD1306.
- Trước khi gửi dữ liệu, cần thiết lập vùng nhớ ghi trên SSD1306 cho Horizontal Addressing Mode:
  - ssd1306_send_command_2params(SSD1306_SET_COLUMN_ADDR, 0, SSD1306_WIDTH - 1); (Đặt dải cột từ 0 đến 127).
  - ssd1306_send_command_2params(SSD1306_SET_PAGE_ADDR, 0, SSD1306_PAGES - 1); (Đặt dải trang từ 0 đến 7).
- Sau đó, gọi ssd1306_send_data(display_buffer, SSD1306_BUFFER_SIZE) để gửi toàn bộ buffer.

CHƯƠNG 3: KIỂM THỬ VÀ KẾT QUẢ

3.1. Phương pháp Kiểm thử

Để đảm bảo driver hoạt động chính xác, các kịch bản kiểm thử sau đã được thiết kế và thực hiện thông qua hàm main() trong file ssd1306_c_driver.c:

1. **Kiểm tra kết nối I2C và khởi tạo driver:** Xác minh rằng bus I2C được mở thành công và địa chỉ slave được thiết lập đúng. Lệnh sudo i2cdetect -y 1 cũng được sử dụng để kiểm tra từ terminal.
2. **Kiểm tra khởi tạo màn hình:** Sau khi gọi ssd1306_init(), màn hình phải được bật và sẵn sàng nhận dữ liệu.
3. **Kiểm tra chức năng xóa và đổ đầy màn hình:**
    - Gọi ssd1306_clear_buffer() sau đó ssd1306_display_buffer(). Màn hình phải hiển thị màu đen hoàn toàn.
    - Gọi ssd1306_fill_buffer(0xFF) sau đó ssd1306_display_buffer(). Màn hình phải hiển thị màu trắng hoàn toàn.
4. **Kiểm tra chức năng vẽ pixel đơn lẻ:**
    - Xóa buffer, sau đó vẽ một pixel tại các vị trí đặc biệt như (0,0), (127,0), (0,63), (127,63), và (63,31) (trung tâm). Hiển thị buffer sau mỗi lần vẽ để quan sát.
5. **Kiểm tra chức năng vẽ nhiều pixel (hình ảnh đơn giản):**
    - Vẽ một đường kẻ ngang ở giữa màn hình.
    - Vẽ một đường kẻ dọc ở giữa màn hình.
    - Vẽ một đường chéo từ góc trên trái đến góc dưới phải.
    - Hiển thị buffer sau mỗi thao tác.

Các hàm delay_ms() được chèn vào giữa các bước kiểm thử để có đủ thời gian quan sát kết quả trên màn hình OLED.

3.2. Kết quả Thực nghiệm

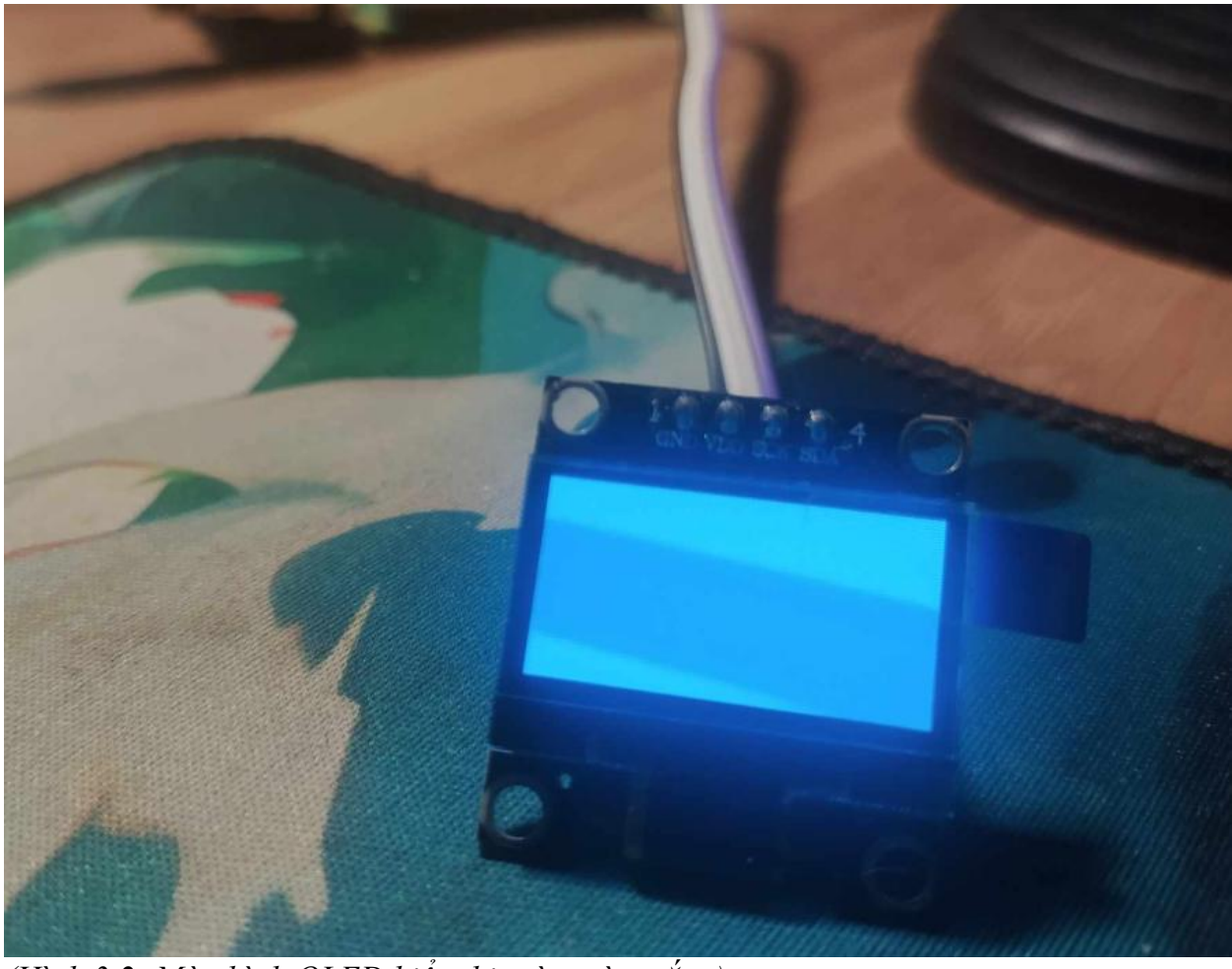
3.2.1. Kiểm tra Khởi tạo và Mở Bus I2C

- Lệnh sudo i2cdetect -y 1 trên terminal cho thấy thiết bị tại địa chỉ 3c, khớp với cấu hình SSD1306_I2C_ADDR.
  (Hình 3.1: Kết quả lệnh i2cdetect -y 1 trên terminal)
- Khi chạy chương trình, log output từ terminal cho thấy:
- Starting SSD1306 C driver test...
- I2C bus /dev/i2c-1 opened and slave address 0x3C set.

SSD1306 initialized.
Điều này xác nhận bus I2C đã được mở và màn hình được khởi tạo thành công.

3.2.2. Kiểm tra Hiển thị Màn hình Trắng Toàn bộ

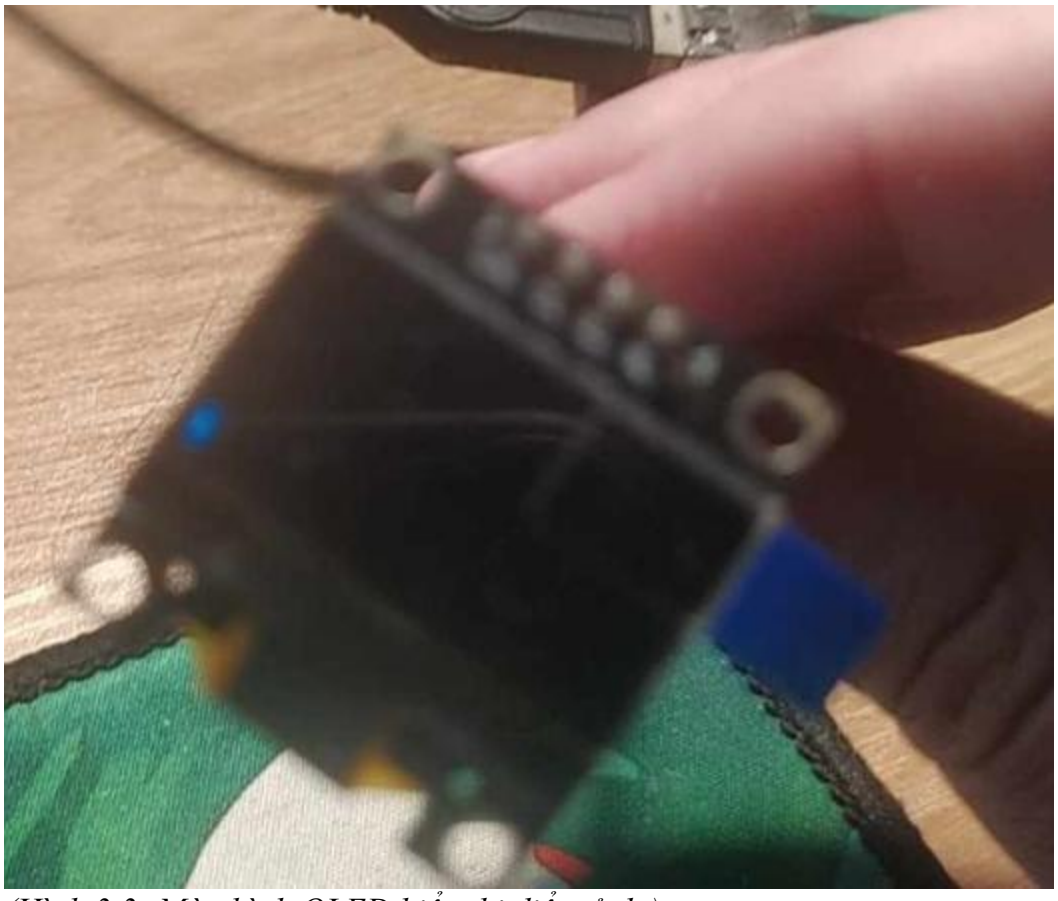
- Sau khi gọi ssd1306_fill_buffer(0xFF); và ssd1306_display_buffer();, màn hình OLED đã hiển thị sáng trắng toàn bộ.



- *(Hình 3.2: Màn hình OLED hiển thị toàn màu trắng)*

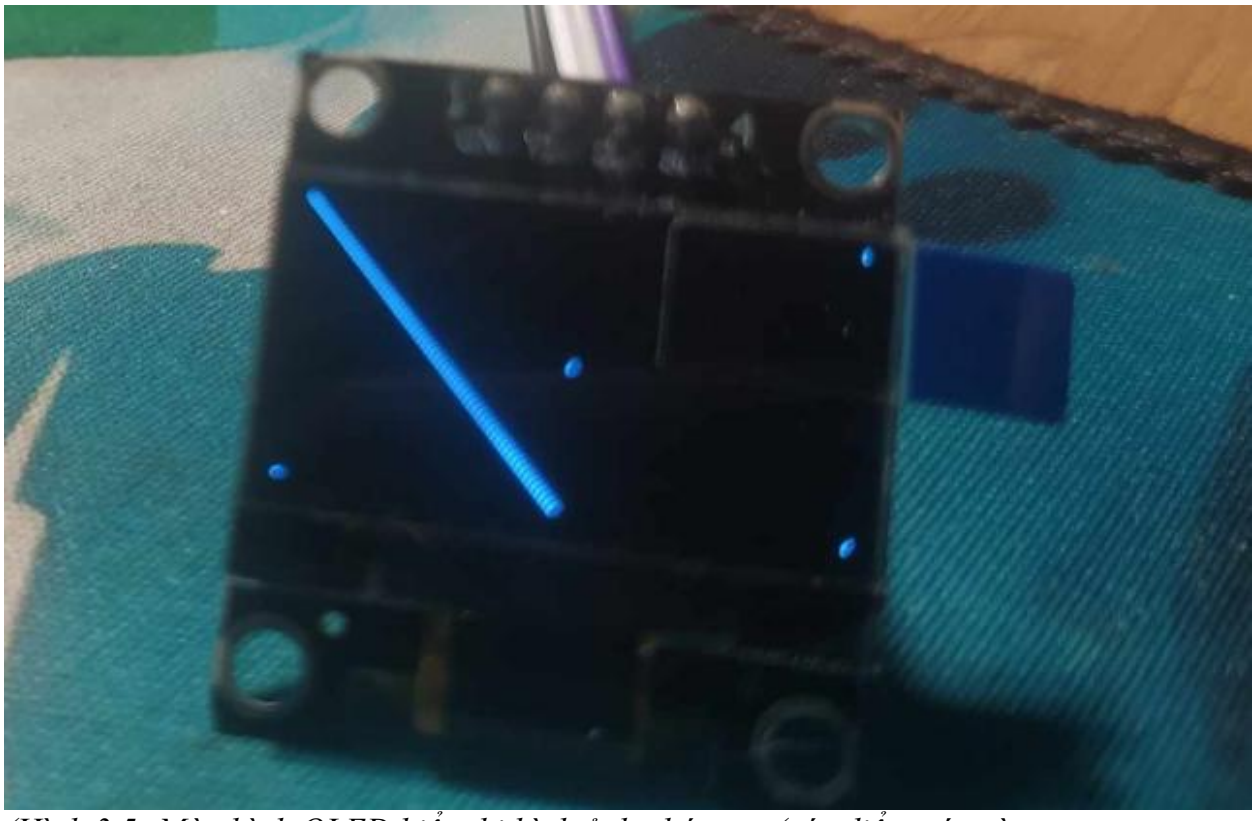
3.2.3. Kiểm tra Vẽ Các Điểm Ảnh Đơn lẻ

- Khi thực hiện vẽ pixel (0,0) và hiển thị, một điểm sáng xuất hiện đúng ở góc trên cùng bên trái màn hình.

- 
  (Hình 3.3: Màn hình OLED hiển thị điểm ảnh)
- Tương tự, khi vẽ pixel (10,10), một điểm sáng xuất hiện đúng tại vị trí dự kiến.
  (Hình 3.4: Màn hình OLED hiển thị điểm ảnh)

3.2.4. Kiểm tra Vẽ Đường Thẳng và Hình Ảnh Phức Tạp Hơn

- Khi thực hiện kịch bản vẽ các điểm ở góc và một đường chéo, các điểm ảnh và đường chéo đã hiển thị đúng như mong đợi.

- 
  (Hình 3.5: Màn hình OLED hiển thị hình ảnh phức tạp (các điểm góc và đường chéo))

3.3. Phân tích và Đánh giá Kết quả

Dựa trên các kết quả kiểm thử thực nghiệm, có thể đánh giá như sau:

- **Chức năng:** Device driver đã triển khai thành công các chức năng cốt lõi đã đề ra, bao gồm khởi tạo màn hình, giao tiếp I2C, quản lý framebuffer, xóa màn hình, đổ đầy màn hình, vẽ từng pixel và hiển thị nội dung lên màn hình SSD1306.
- **Tính đúng đắn:** Các thao tác hiển thị (màn hình trắng, các pixel và đường kẻ riêng lẻ) đều cho kết quả chính xác trên màn hình vật lý, cho thấy logic tính toán vị trí pixel trong framebuffer và quá trình gửi dữ liệu lên GDDRAM là đúng đắn.
- **Tính ổn định:** Trong quá trình kiểm thử, driver hoạt động ổn định, không gây ra lỗi hệ thống hoặc treo thiết bị.
- **Ưu điểm:**
  - Driver được viết bằng C, cho phép tương tác trực tiếp với các system call của Linux, mang lại hiểu biết sâu sắc về cách hệ điều hành làm việc với thiết bị.
  - Cấu trúc code tương đối đơn giản, dễ theo dõi và hiểu cho mục đích học tập.
  - Cung cấp nền tảng tốt để phát triển các chức năng đồ họa phức tạp hơn.
- **Nhược điểm và Hạn chế:**
  - Driver hiện tại chỉ hỗ trợ các thao tác vẽ cơ bản (pixel). Chưa có các hàm vẽ hình học (đường tròn, chữ nhật) hay hiển thị văn bản.

- Chưa có cơ chế tối ưu hóa việc cập nhật màn hình (hiện tại luôn gửi toàn bộ framebuffer mỗi lần gọi ssd1306_display_buffer()).
- Xử lý lỗi còn ở mức cơ bản (chủ yếu dựa vào perror).
- Driver được thiết kế cho một cấu hình cụ thể (SSD1306 128x64, I2C), việc mở rộng cho các loại màn hình hoặc giao diện khác sẽ cần nhiều thay đổi.

Nhìn chung, driver đã đáp ứng tốt các mục tiêu của bài tập môn học Hệ Điều Hành Nhúng, thể hiện khả năng điều khiển thiết bị phần cứng từ user-space.

CHƯƠNG 4: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

4.1. Kết luận

Đề tài "Xây Dựng Device Driver User-Space Điều Khiển Màn Hình OLED SSD1306 Giao Tiếp I2C Trên Raspberry Pi 5" đã được hoàn thành, đạt được các mục tiêu chính đã đề ra. Thông qua việc nghiên cứu lý thuyết về Raspberry Pi, giao thức I2C, chip SSD1306 và cách hệ điều hành Linux quản lý thiết bị, một thư viện driver cơ bản bằng ngôn ngữ C đã được thiết kế và triển khai thành công.

Driver cho phép người dùng thực hiện các thao tác cần thiết để điều khiển màn hình OLED SSD1306, bao gồm khởi tạo, xóa nội dung, bật/tắt từng điểm ảnh và hiển thị dữ liệu từ bộ đệm. Quá trình kiểm thử thực nghiệm đã xác nhận tính đúng đắn và ổn định của các chức năng đã triển khai. Sản phẩm của đề tài không chỉ là một chương trình hoạt động mà còn là minh chứng cho việc áp dụng kiến thức môn học Hệ Điều Hành vào giải quyết một bài toán thực tế liên quan đến tương tác phần cứng.

4.2. Bài học Kinh nghiệm

Quá trình thực hiện đề tài đã mang lại nhiều bài học kinh nghiệm quý báu:

- **Hiểu biết sâu sắc hơn về giao tiếp phần cứng:** Nắm vững hơn về cách hoạt động của giao thức I2C, cách đọc và hiểu datasheet của một IC (SSD1306), tầm quan trọng của chuỗi lệnh khởi tạo và các thanh ghi cấu hình.
- **Kỹ năng lập trình hệ thống với C trên Linux:** Thành thạo hơn trong việc sử dụng các system call (open, ioctl, write, close), làm việc với con trỏ, quản lý bộ nhớ và biên dịch chương trình C trong môi trường Linux.
- **Kỹ năng gỡ lỗi (Debugging):** Quá trình gỡ lỗi khi làm việc với phần cứng đòi hỏi sự kiên nhẫn và phương pháp tiếp cận có hệ thống, từ kiểm tra kết nối vật lý đến phân tích tín hiệu logic (nếu có công cụ) và gỡ lỗi từng dòng code. Việc "màn hình có sáng" nhưng không hiển thị đúng là một ví dụ điển hình cần phân tích từng bước.
- **Tầm quan trọng của tài liệu:** Datasheet là tài liệu không thể thiếu. Các diễn đàn và cộng đồng trực tuyến cũng là nguồn hỗ trợ hữu ích.
- **Sự khác biệt giữa user-space và kernel-space driver:** Mặc dù đề tài này là user-

space driver, nó cũng gợi mở về sự phức tạp và quyền hạn cao hơn của kernel-space driver.

Những khó khăn chính gặp phải bao gồm việc đảm bảo kết nối phần cứng chính xác và ổn định, tìm ra chuỗi lệnh khởi tạo phù hợp cho module SSD1306 cụ thể, và gỡ lỗi hiển thị khi màn hình không lên hoặc hiển thị sai lệch. Việc chia nhỏ vấn đề, kiểm thử từng phần và kiên trì đã giúp vượt qua các khó khăn này.

4.3. Hướng Phát triển của Đề tài

Driver hiện tại là một nền tảng tốt và có nhiều tiềm năng để phát triển thêm:

1. **Mở rộng thư viện đồ họa:**
    - Triển khai các hàm vẽ hình học cơ bản: drawLine(), drawRectangle(), drawCircle(), fillRectangle().
    - Thêm chức năng hiển thị văn bản: drawString(), hỗ trợ các font chữ bitmap với kích thước khác nhau.
2. **Tối ưu hóa hiệu suất:**
    - Triển khai cơ chế chỉ cập nhật những vùng màn hình có thay đổi (dirty rectangles) thay vì gửi toàn bộ framebuffer mỗi lần, giúp tăng tốc độ làm tươi màn hình.
3. **Cải thiện tính linh hoạt và khả năng cấu hình:**
    - Cho phép người dùng dễ dàng thay đổi địa chỉ I2C hoặc bus I2C thông qua tham số hoặc file cấu hình.
    - Hỗ trợ các kích thước màn hình SSD1306 khác (ví dụ: 128x32) bằng cách điều chỉnh các tham số cấu hình MUX ratio, COM pins.
4. **Đóng gói thành thư viện chuẩn:**
    - Tách mã nguồn thành các file .h và .c riêng biệt, tạo Makefile để dễ dàng biên dịch và liên kết vào các dự án khác.
5. **Nghiên cứu và triển khai dưới dạng Kernel Module:**
    - Để hiểu sâu hơn về kiến trúc driver trong Linux, có thể nghiên cứu cách viết một character device driver ở mức kernel cho SSD1306, tạo ra một thiết bị framebuffer (/dev/fbX) chuẩn của Linux. Đây là một hướng đi phức tạp nhưng mang lại hiệu suất và sự tích hợp hệ thống tốt hơn.

TÀI LIỆU THAM KHẢO

1. Raspberry Pi Foundation. *Raspberry Pi Documentation*. https://www.raspberrypi.com/documentation/
2. The Linux Kernel Archives. *I2C and SMBus Subsystem*. https://www.kernel.org/doc/html/latest/i2c/index.html