|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI  **VIỆN ĐIỆN TỬ - VIỄN THÔNG**  **-----□□&□□-----**  logo_128  **BÀI TẬP LỚN MÔN HỌC**  **HỆ ĐIỀU HÀNH**  ***Đề tài: Xây dựng device driver cho màn hình Nokia 5110 để giao tiếp với GPIO trên Raspberry Pi 3, và xây dựng một ứng dụng tương tác với device drivers.***   |  |  |  | | --- | --- | --- | | **Giảng viên hướng dẫn** | TS. Phạm Doãn Tĩnh | | | **Nhóm sinh viên thực hiện:** | | | | **Họ và tên** | **MSSV** | **Lớp** | | Bùi Huy Hoàng | 20172568 | ĐTVT-07.K62 | | Vũ Đức Thái | 20172804 | ĐTVT-08.K62 | | Nguyễn Văn Hoàng | 20172579 | ĐTVT-07.K62 | | Nguyễn Đức Long | 20172668 | ĐTVT-08.K62 | | Đặng Thành Long | 20172667 | ĐTVT-07.K62 |   Hà Nội, 11-2020 |

**LỜI NÓI ĐẦU**

Như tên gọi của nó, Embedded Linux gồm có hai phần Embedded và Linux. Điều kiện cần và đủ để tìm hiểu về chủ đề này là phải có kiến thức nền tảng về embedded và biết cách sử dụng Linux cơ bản. Ngày nay, các thiết bị nhúng Linux hiện diện ở khắp mọi nơi từ chiếc smartwatch nhỏ bé đến Tivi hay chiếc điện thoại di động phức tạp, đến siêu máy tính, xe ô tô và cả tàu vũ trụ,… Những điều này làm cho embedded Linux trở thành một miền đất đầy hứa hẹn cho những ai đam mê và chinh phục nó.

Trong quá trình thực hiện đề tài liên quan đến Embedded Linux, chúng em có gặp một chút khó khăn nhỏ trong vài ngày đầu như đọc tài liệu tiếng anh, tìm hiểu các khái niệm cơ bản xung quanh linux, các kiến thức về vi điều khiển,... Nhưng nhờ có những kiến thức bài giảng trên lớp, cùng sự giúp đỡ của thầy, chúng em đã nhanh chóng thích ứng được với định hướng đề tài của mình. Đối với chúng em đó là những thuận lợi khi bắt đầu tiếp cận với một vấn đề mới như Embedded Linux. Báo cáo gồm có các nội dung như sau:

* ***Chương 1: Kiến thức cơ bản về hệ điều hành***
* ***Chương 2: Giới thiệu chung về raspberry pi***
* ***Chương 3: Nạp hệ điều hành cho raspberry pi***
* ***Chương 4: Xây dựng device drivers cho raspberry pi***
* ***Chương 5: Xây dựng ứng dụng để tương tác với device drivers***

“Học tập là một quá trình suốt đời chứ không phải ngày một ngày hai, học Embedded Linux cũng vậy, đó là một hành trình đầy vất vả và gian nan.”

Chúng em chân thành cảm ơn thầy Phạm Doãn Tĩnh đã tận tình chỉ bảo góp  
ý, gợi ý giải pháp để nhóm có thể hoàn thành đề tài bài tập lớn!

**MỤC LỤC**

[DANH MỤC HÌNH VẼ i](#_Toc59786622)

[DANH MỤC BẢNG BIỂU ii](#_Toc59786623)

[CHƯƠNG 1. Kiến thức cơ bản về hệ điều hành 1](#_Toc59786624)

[1.1 Interrupts (Ngắt) 1](#_Toc59786625)

[1.2 Cách thành phần tạo nên hệ điều hành 1](#_Toc59786626)

[1.3 OPERATING SYSTEM CONTROL 2](#_Toc59786627)

[1.4 MODES (CÁC CHẾ ĐỘ) 3](#_Toc59786628)

[1.5 CÁC CHƯƠNG TRÌNH CHO PHÉP VÀ HỖ TRỢ HỆ ĐIỀU HÀNH 4](#_Toc59786629)

[1.5.1 Booting nói chung 4](#_Toc59786630)

[1.5.2 Quy trình boot 4](#_Toc59786631)

[CHƯƠNG 2. Giới thiệu chung về Raspberry Pi 6](#_Toc59786632)

[2.1 Giới thiệu chung về SoC/Raspberry Pi 6](#_Toc59786633)

[2.2 Cấu hình phần cứng 6](#_Toc59786634)

[2.3 Thiết bị đi kèm với Raspberry Pi 8](#_Toc59786635)

[CHƯƠNG 3. Nạp hệ điều hành cho raspberry Pi 10](#_Toc59786636)

[3.1 Giới thiệu các hệ điều hành mà Raspberry Pi hỗ trợ 10](#_Toc59786637)

[3.2 Booting với Raspberry Pi 10](#_Toc59786638)

[3.3 Chuẩn bị thiết bị phần cứng và phần mềm hỗ trợ cài đặt 11](#_Toc59786639)

[3.3.1 Thiết bị phần cứng 11](#_Toc59786640)

[3.3.2 Phần mềm cần thiết 12](#_Toc59786641)

[CHƯƠNG 4. XÂY DỰNG DEVICE DRIVERS CHO RASPBERRY PI 13](#_Toc59786642)

[4.1 Xây dựng Device Driver cho màn hình LCD Nokia 5110 13](#_Toc59786643)

[4.1.1 Giới thiệu chung về Nokia 5110 13](#_Toc59786644)

[4.1.2 Cấu hình và giao tiếp với Nokia 5110 14](#_Toc59786645)

[4.1.3 Xây dựng device driver nhúng vào Linux 20](#_Toc59786646)

[4.2 Xây dựng Device Driver cho button 22](#_Toc59786647)

[4.2.1 Phân tích chung về cấu hình cho cụm nút bấm 22](#_Toc59786648)

[4.2.2 Xây dựng device driver nhúng vào Linux 22](#_Toc59786649)

[CHƯƠNG 5. XÂY DỰNG ỨNG DỤNG ĐỂ TƯƠNG TÁC VỚI DEVICE DRIVERS 25](#_Toc59786650)

[5.1 Giới thiệu về ứng dụng xây dựng 25](#_Toc59786651)

[5.2 Lập trình đa luồng trong ứng dụng 25](#_Toc59786652)

[5.2.1 Thư viện pthread 25](#_Toc59786653)

[5.2.2 Tạo thread trong code 26](#_Toc59786654)

[5.2.3 Cơ chế mutex 26](#_Toc59786655)

[5.3 Đọc ghi data xuống buffer của device drivers 26](#_Toc59786656)

[PHỤ LỤC 28](#_Toc59786657)

[KẾT LUẬN 29](#_Toc59786658)

[TÀI LIỆU THAM KHẢO 1](#_Toc59786659)

# DANH MỤC HÌNH VẼ

[Hình 1.1 Kernel quản lý tài nguyên máy tính 2](#_Toc59786660)

[Hình 1.2 Các tiến trình trong hệ điều hành của RP ngay khi boot 3](#_Toc59786661)

[Hình 2.1 Cấu hình phần cứng của Raspberry Pi 7](#_Toc59786662)

[Hình 2.2 GPIO của Raspberry Pi 7](#_Toc59786663)

[Hình 2.3 Các phụ kiện đi kèm của Raspberry Pi 9](#_Toc59786664)

[Hình 3.1 Raspberry Pi booting 11](#_Toc59786665)

[Hình 4.1 Hình ảnh thực tế của màn hình Nokia 5110 13](#_Toc59786666)

[Hình 4.2 Cấu trúc bên trong vi điều khiển PCD8544 16](#_Toc59786667)

[Hình 4.3 Cấu trúc DDRAM của Nokia5110 17](#_Toc59786668)

[Hình 4.4 LCD driver waveforms 18](#_Toc59786669)

[Hình 4.5 Định dạng của DDRAM 19](#_Toc59786670)

[Hình 4.6 Ghi data tuần tự theo chiều dọc (V = 1) 19](#_Toc59786671)

[Hình 4.7 Ghi data tuần tự theo chiều ngang (V = 0) 19](#_Toc59786672)

[Hình 4.8 Các hệ số điện áp tương ứng với nhiệt độ của LCD 20](#_Toc59786673)

[Hình 4.9 Những cờ ngắt có sẵn trong kernel 24](#_Toc59786674)

[Hình 5.1 Ví dụ đơn giản về cơ chết Mutex 26](#_Toc59786675)

# DANH MỤC BẢNG BIỂU

[Bảng 4.1 Cấu hình các chân ngoại vi của LCD Nokia 5110 14](#_Toc59786676)

[Bảng 4.2 Các chân giao tiếp của PCD8544 16](#_Toc59786677)

[Bảng 4.3 Một số hàm khai báo gpio cơ bản 23](#_Toc59786678)

# Kiến thức cơ bản về hệ điều hành

## Interrupts (Ngắt)

Một máy tính chỉ có thể thực hiện một lệnh trong 1 thời điểm, hết lệnh này chuyển sang lệnh khác. Nó sẽ tiếp tục chạy tập lệnh (của chương trình) tới khi kết thúc hoặc nhận được tín hiệu ngắt. Ngắt ra hiệu cho CPU của máy tính và các thiết bị phần cứng các dừng những thao tác đang thực hiện và chạy một (hoặc 2, 3) tập lệnh khác, sau đó quay trở lại thao tác chưa hoàn thành ban đầu. Điều này cho phép phân chia công việc và là khái niệm cơ bản để tạo nên đa nhiệm (multitasking).

Các ngắt sẽ được hoàn thành dựa trên tốc độ máy tính, vì thế mà người dừng thường sẽ không phát hiện ra bất cứ sự chậm trễ nào trong các ứng dụng khi máy tính đang chạy các chương trình khác, background processes. Những background processes bao gồm những thao tác như cập nhật ngày giờ, kiểm tra cập nhật phần mềm, nhận thông tin input tử bàn phím, vv…. Nó cũng cho phép ứng dụng cho thể yêu cầu và nhận về dữ liệu. Hệ điều hành bao gồm một chương trình lập lịch, một chương trình quản lý lý ngắt, chúng chạy để theo dõi và ưu tiên các tiến trình xử lý theo một chu trình hợp lý. Chương trình lập lịch cho phép OS quyết định chương trình nào sẽ được phân chia trong thời gian chạy kế tiếp. OS còn có thể làm quá trình lập lịch hiệu quả hơn bằng cách tìm kiếm các khoảng thời gian không hoạt động để nhét thêm các interrupts vào để tăng tốc độ đa nhiệm. Trong OS có 3 loại ngắt: ngắt phần cứng, ngắt phần mềm, traps.

## Cách thành phần tạo nên hệ điều hành

Ta có thể chia chúng ra thành 4 phần chính:

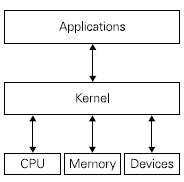
1. Kernel: Đây là trái tim của mọi hệ điều hành. Nó tạo ra cầu nối giữa ứng dụng và các tiến trình, cho phép và điều khiển CPU cũng như các phần cứng khác khi thực hiện xử lý dữ liệu và quản lý cũng như phân phối tài nguyên bộ nhớ.

2. Networking: Dưới sự kiểm soát của kernel thường là các hệ thống con và user space component cung cấp các giao thức mạng khác nhau như Ethernet.

3. Security (Bảo mật): Giữ cho máy tính an toàn .

4. User interface: Giúp OS giao tiếp với người sử dụng.

**KERNEL**



Hình 1.1 Kernel quản lý tài nguyên máy tính

Kernel còn thực hiện chức năng đa nhiệm. Đa nhiệm xảy ra khi mà OS sử dụng các ngắt để có thể chia thời gian của CPU thành các phần nhỏ cho mỗi tiến trình đang chạy, điều này cho phép hàng trăm tiến trình, chương trình và yêu cầu từ nhiều người dùng có thể chạy trong cùng một lúc. Được hỗ trợ bởi kernel drivers, đây là một chương trình nhỏ nằm ở giữa chương trình và kernel. Kernel driver hoạt động như một chất để gắn các thành phần và giúp chúng giao tiếp với nhau và đảm bảo rằng tiến trình có thể phản hồi tới OS và OS có thể điều khiển nó.

Các CPU nhiều lõi hiện nay thay vì thực hiện chia thời gian của CPU, tiến trình có thể được chia thành các tiến trình nhỏ hơn và thực hiện song song trên các lõi khác nhau. Thực hiện điều này đòi hỏi dữ liệu phải thỏa mãn việc xử lý đồng thời.

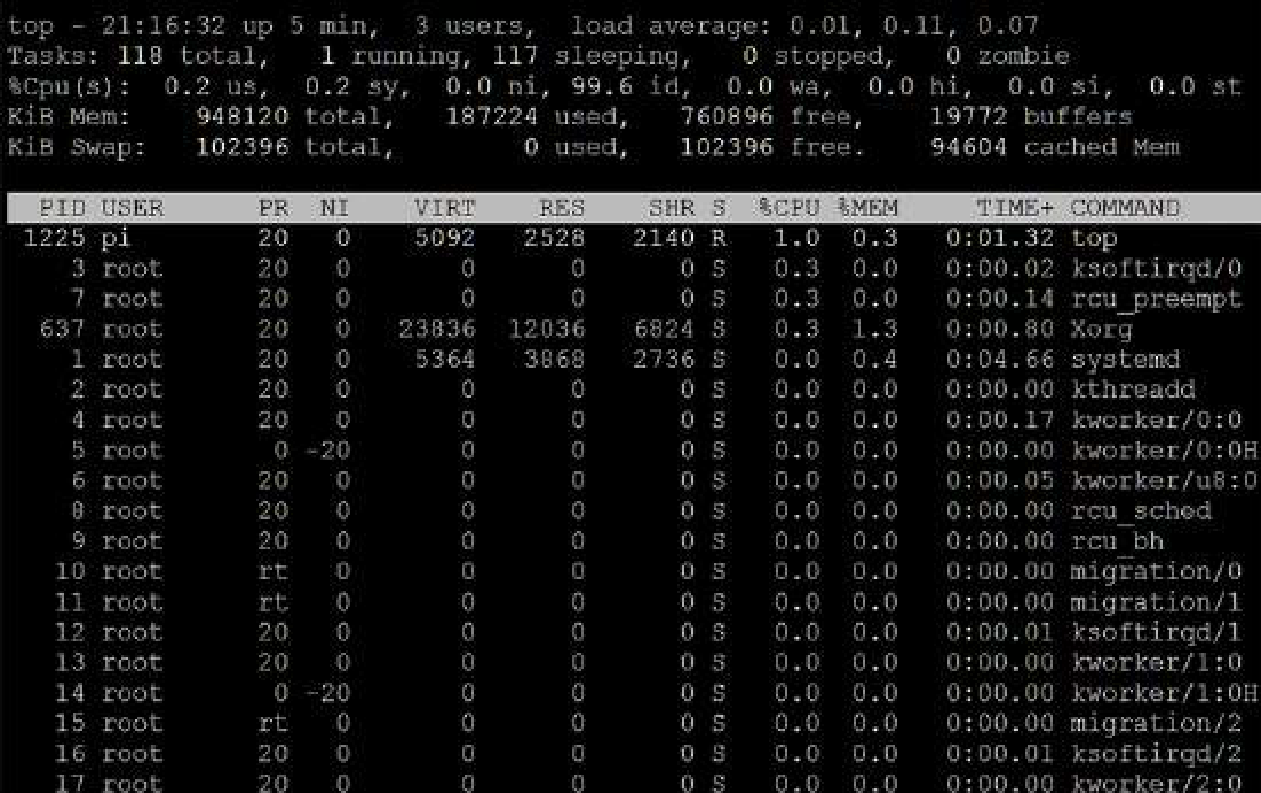
Xử lý đồng thời và xử lý song song sử dụng một vài phương pháp như mạng Petri, tiến trình calculi, mô hình Truy cập máy song song ngẫu nhiên…Kết quả của các phương pháp này là các phần nhỏ của một tiến trình được chia ra sau đó được thực hiện liên tiếp trên các lõi và cuối cùng kết quả được hợp nhất.

OS thực hiện việc xử lý song song trong khi quản lý file hệ thống và quản lý bộ nhớ và các tác vụ khác. Kernel của Raspberry Pi cung cấp xử lý song song cũng như các hệ thống máy tính lớn khác.

## OPERATING SYSTEM CONTROL

Hệ điều hành bao gồm rất nhiều chương trình con mà mỗi chương trình chiếm một phần của chu trình CPU cho bản thân nó. Các chương trình con này, khi chạy, bao gồm các tiến trình cần thiết để hệ điều hành có thể thực hiện được chức năng của nó là quản lý tài nguyên máy tính.

Phía dưới là các tiến trình trong hệ điều hành của Raspberry Pi ngay sau khi boot lên, có tổng cộng 117 tiến trình đang được thực hiện.



Hình 1.2 Các tiến trình trong hệ điều hành của RP ngay khi boot

Có rất nhiều những hoạt động phía sau điều khiển phần rất sâu của OS. Một số chúng chạy vĩnh viễn ngay sau khi được boot lên. Một tiến trình được sử dụng trong phần lớn các hệ điều hành có nhân Linux và tồn tại cả trên Raspberry Pi là cron. Cái tên “Cron” bắt nguồn từ một từ tiếng anh là “Chronological” nghĩa là theo thời gian. Khi mà ta backup file thì cron là tiến trình giúp ta có thể thực hiện được tác vụ này.

## MODES (CÁC CHẾ ĐỘ)

Hệ điều hành kiểm soát người dùng nào có quyền truy cập vào file nào và có thể chạy được chương trình nào. Các chế độ thực hiện bảo mật ở tầng sâu bên dưới. CPU hiện nay cho chúng ta vài lựa chọn hoạt động. Hai trong số đó là supervisor mode và protected mode.

Hệ điều hành sử dụng chế độ tối đa công suất supervisor mode rất tiết kiệm. Mặc dù vậy, tại thời điểm mà chế độ supervisor được chạy mà không có bất cứ sự quản lý nào đến từ OS là trong lúc boot. Bởi vì OS chưa hoàn toàn được khởi động nên nó không có quyền kiểm soát. Thực ra, những chương trình bắt đầu lúc mà máy tính khởi động ví dụ như bootloader, bắt buộc phải có quyền không bị giới hạn truy cập tới phần cứng. Khả năng CPU chạy trong chế độ protected chỉ có thể được thiết lập khi đang ở trong chế độ supervisor.

Sau khi mà OS được khởi động nó đưa CPU vào chế độ protected. Chế độ protected sẽ hạn chế tập lệnh mà CPU có thể thực hiện, ngăn mọi chương trình có thể có tác động xấu tới phần cứng. Trong hầu hết thời gian thì OS bắt các chương trình và kể cả các tiến trình của chính nó cũng phải trong chế độ protected.

## CÁC CHƯƠNG TRÌNH CHO PHÉP VÀ HỖ TRỢ HỆ ĐIỀU HÀNH

### Booting nói chung

Ở các máy tính hiện đại, thứ giúp khởi động OS bao gồm có bootloader, đây là một chương trình nhỏ có sẵn trên ROM. Loader sẽ chạy tự động khi máy tính được bật nguồn, thực hiện các thiết lập truy cập và cung cấp các thông tin cần thiết để các chương trình của hệ điều hành có thể được đưa vào trong bộ nhớ đang hoạt động và được thực thi.

ROM ngoài chứa Bootloader thì còn chứa vài thông tin về máy tính như Hệ thống vào ra cơ bản Basic Input/Output System (BIOS). BIOS thực hiện khởi động phần cứng bất cứ khi nào mà máy tính boot. Các máy tính thế hệ mới thì BIOS có thể được thay bằng giao tiếp UEFI (Unified Extensible Firmware interface). Cả BIOS lần UEFI đều là firmware - hay còn có thể hiểu là chương trình có kích thước nhỏ được cụ thể cho phần cứng và nằm vĩnh viễn trên bộ nhớ ROM, hoặc các thiết bị nhớ khác.

### Quy trình boot

Khi chip của BIOS hay UEFI nhận được nguồn, nó bắt đầu chạy chẩn đoán (để đảm bảo phần cứng vẫn ổn), các phần tử của máy tính bắt đầu khởi động, và chương trình bootstrap bắt đầu chạy. Loader sẽ load hệ điều hành lên bộ nhớ hoạt động từ bộ nhớ lưu trữ và bắt đầu chạy nó. OS tạo ra các cấu trúc dữ liệu trên bộ nhớ hoạt động, đặt ra các thanh ghi trong CPU và bắt đầu khởi động các chương trình ở tầng người dùng. Từ đó trở đi, hệ điều hành bắt đầu tiếp nhận các ngắt và thực thi các lệnh.

Bước thứ 2 Bootloader: Chương trình bootstrap có những hạn chế, một trong số đó đến từ dung lượng nhớ nhỏ của ROM. Điều này sẽ gây khó khi ta yêu cầu một chu trình boot phức tạp hơn, vì thế mà loader sử dụng 2 bước là giải pháp. Đây là một khái niệm như sau:

* Chương trình bootstrap sẽ load một loader cao cấp hơn từ đĩa lên bộ nhớ. Loader mới này sẽ có những chức năng thêm vào dẫn tới ta sẽ có nhiều sự lựa chọn hơn. Một trong những chức năng đó là ta có thể chọn hệ điều hành trong nhiều hệ điều hành sẵn có trong đĩa để load.
* Một loader được sử dụng phổ biến đó là Grand Unified Bootloader (GNU GRUB), từ GNU Project và Free Software Foundation. GRUB hỗ trợ tiến trình boot trong nhiều hệ điều hành Linux. GRUB bao gồm shell capacity, cho phép các thao tác ở tầng thấp thực hiện trước khi hệ điều hành được load; đôi khi nó khá hữu dụng khi ta muốn khôi phục một hệ thống không có khả năng load hệ điều hành.
* Bootloader thứ 2 còn giúp boot networking

Network Booting: Do chương trình Bootloader thứ 2 lớn hơn và cũng phức tạp hơn, sẽ có khả năng boot từ network. Điều này sẽ dẫn đến máy tính không cần tới ổ cứng và trở nên hữu dụng với những máy tính nhúng hay với các ứng dụng khác.

# Giới thiệu chung về Raspberry Pi

## Giới thiệu chung về SoC/Raspberry Pi

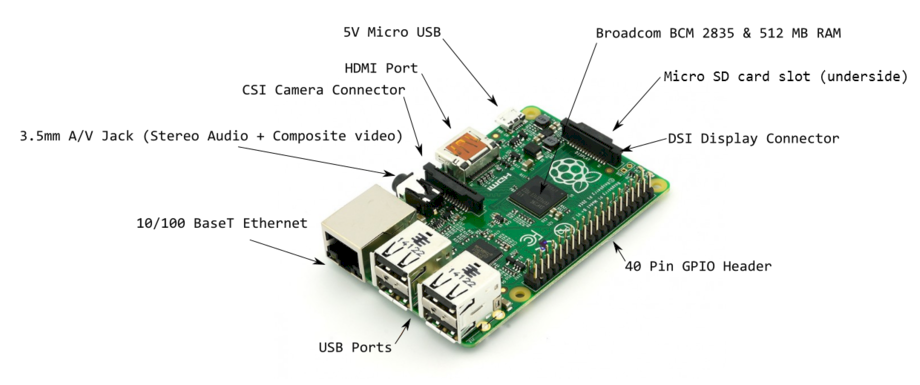
Một SoC hay hệ thống trên chip là một mạch tích hợp (IC) có chứa tất cả những phần tử quan trọng của một máy tính hay bất kì một hệ thống điện tử nào đó trên duy nhất một con chip. Các thành phần bao gồm bộ xử lý trung tâm (CPU), khối xử lý đồ họa (GPU) và hàng loạt các mạch tương tự và số lẫn nhau trên cùng một con chip. SoC giúp máy tính thu vào một diện tích rất nhỏ trở nên khả thi, điển hình đó chính là Raspberry Pi.

Do vậy, Raspberry Pi là chiếc máy tính kích thước nhỏ được tích hợp nhiều phần cứng mạnh mẽ đủ khả năng chạy hệ điều hành và cài đặt được nhiều ứng dụng trên nó. Với giá chỉ vài chục USD, Raspberry hiện đang là mini computer nổi bật nhất hiện nay. Ban đầu, tổ chức Raspberry Pi Foundation phát triển dự án Raspberry với mục tiêu chính là giảng dạy máy tính cho trẻ em và tạo ra một công cụ giá rẻ (chỉ vài chục USD) để sinh viên nghiên cứu học tập. Tuy nhiên, sau khi xuất hiện,  Raspberry Pi được cộng đồng đánh giá cao về tính ứng dụng với phần cứng được hỗ trợ tốt,  Pi đã nhanh chóng phát triển một cách rộng rãi. Pi phù hợp cho những ứng dụng cần khả năng xử lý mạnh mẽ, đa nhiệm hoặc giải trí và đặc biệt cần chi phí thấp. Hiện nay đã có hàng ngàn ứng dụng đa dạng được cài đặt trên Raspberry Pi.

Tổ chức Raspberry Pi Foundation được thành lập năm 2009, và cùng với sự giúp đỡ của 3 hãng cung cấp linh kiện điện tử lớn là EGOMAN, QSIDA, SONY phát triển lên Raspberry Pi. Trên thế giới, chúng ta có thể tìm mua tại những nhà phân phối của hãng như EGOMAN, ELEMENT14 hoặc RS Component.

## Cấu hình phần cứng

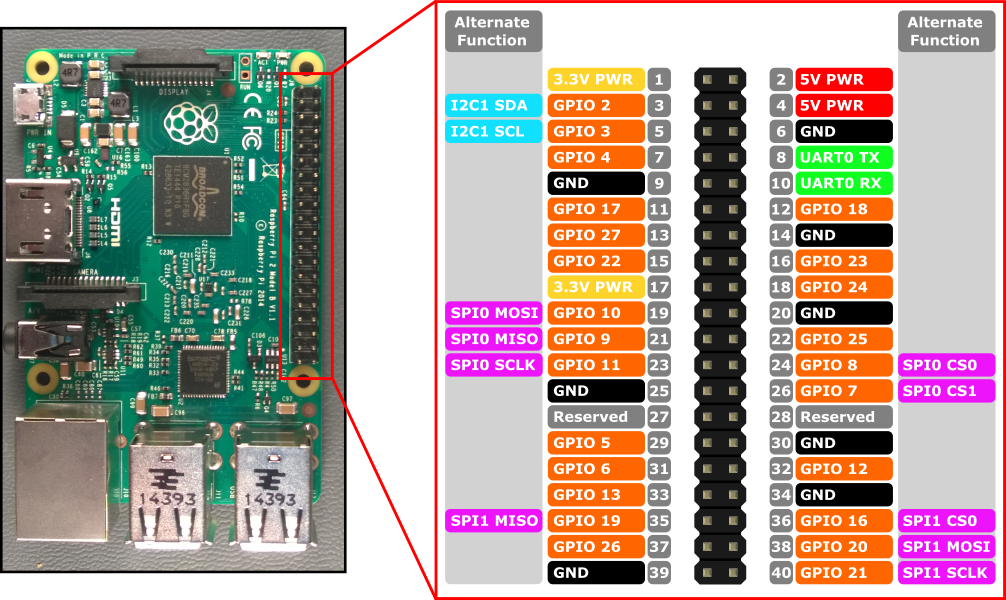
Phiên bản Raspberry Pi đầu tiên được phát hành tháng 2 năm 2012, và tới nay đã có nhiều phiên bản khác nhau, với sự nâng cấp của phần cứng, cũng như hướng tới những mục tiêu khác nhau. Phiên bản theo thứ tự ra mắt là : Pi A → Pi A+ → Pi 1 B → Pi 1B+ → Pi 2B → Pi Zero → Pi 3B → Pi 4B



Hình 2.1 Cấu hình phần cứng của Raspberry Pi

Các chip của RP được phát triển và sản suất bởi Broadcom Limited. Đặc biệt các model cũ được đi kèm với Broadcom BCM2835 và RP 2 có Broadcom BCM2836, và Model 3 mới sử dụng BCM2837. Điểm khác biệt lớn nhất giữa BCN2835 và BCM2836 đó chính là sự thay thế của CPU đơn lõi với vi xử lý 4 lõi, còn về phần kiến trúc chúng tương đương nhau.

*Những phần tử được cung cấp bởi Broadcom SoCs:*



Hình 2.2 GPIO của Raspberry Pi

* CPU: Thực hiện xử lý dữ liệu dưới sự điều khiển của hệ điều hành
* GPU: Cung cấp giao diện hệ điều hành
* Bộ nhớ: Bộ nhớ cố định được sử dụng cho các thao tác CPU và GPU, storage cho các phần mềm bootstrap và những phần mềm nhỏ để load hệ điều hành và kích hoạt nó
* Timers: Cho phép phần mềm phụ thuộc thời gian phục vụ lập lịch, đồng bộ hóa
* Interrupt controller: Giúp hệ điều hành điều khiển tài nguyên hệ thống, biết được khi nào CPU sẵn sàng cho lệnh mới, vv
* General purpose input output (GPIO): Cung cấp layout giúp cho việc điều khiển các kết nối, đầu vào, đầu ra và các chế độ khác của chân GPIO cho phép RP có thể điều khiển các mạch điện tử, thiết bị điện tử, vv. Nói ngắn gọn nó giúp RP trở thành một hệ thống điều khiển nhúng
* Trong 40 chân GPIO bao gồm: 26 chân GPIO. Khi thiết lập là input, GPIO có thể được sử dụng như chân interrupt, GPIO 14 & 15 được thiết lập sẵn là chân input.
* USB: Cung cấp giao thức Universal Serial Bus cho đầu vào và đầu ra, cho phép kết nối tới RP thông qua USB
* PCM/IS
* Direct memory access (DMA) controller
* I^2C master
* I^2C/SPI (Serial Peripheral Interface) slave
* SPI Interface
* Pulse width modulation (PWM)
* Universal asynchronous receiver/transmitter (UART0, UART1)

## Thiết bị đi kèm với Raspberry Pi

Có 2 phụ kiện bắt buộc phải có để có thẻ chạy Raspberry là : nguồn cung cấp và thẻ nhớ để cài hệ điều hành. Ngoài ra, để sử dụng tốt hơn những ứng dụng của Raspberry các bạn có thẻ lựa chọn thêm những sản phẩm hỗ trợ cho Raspberry. Như dây HDMI để kết nối với màn hình máy tính, cáp nối mạng hay camera.

1. Nguồn cho Raspberry 5V-2A
2. Dây HDMI cabos 1.5m
3. Vỏ hộp Pi2/B+
4. Raspberry Pi Camera Board
5. Kết nối wifi cho Pi : Wireless USB  EP-N8508GS
6. Màn hình cảm ứng 7 inch
7. Thẻ nhớ



Hình 2.3 Các phụ kiện đi kèm của Raspberry Pi

# Nạp hệ điều hành cho raspberry Pi

## Giới thiệu các hệ điều hành mà Raspberry Pi hỗ trợ

- Các hệ điều hành hỗ trợ cho Raspberry Pi bao gồm : Raspbian, Arch Linux ARM, OSMC, OpenELEC, Snappy Ubuntu Core, Ubuntu MATE, Debian Jessie, Windows and Android.

* **Raspbian** : Đây là một hệ điều hành thuận tiện cho việc cài đặt và sử dụng với sự hỗ trợ lớn từ cộng đồng mã nguồn mở trên thế giới. Raspbian là một hệ điều hành đơn giản và thân thiện. Ta có thể download và sử dụng Raspbian ở 2 dạng : NOOBS hoặc file Raspbian.
* Với NOOBS (New Out of Box Software), khi thực hiện cài đặt bạn sẽ được lựa chọn giữa các hệ điều hành khác nhau cho Raspberry Pi, khi đó bạn cũng có thể lựa chọn Raspbian.
* Với file Raspbian, khi đó sẽ không có quá trình lựa chọn giữa các hệ điều hành, mà Raspberry Pi sẽ tự thực hiện nhiệm vụ của mình để chuyển tới việc khởi động của Linux kernel.

(file NOOBS được cộng đồng khuyến khích sử dụng, để tránh một số lỗi ngoài mong muốn có thể xảy ra trong quá trình cài đặt).

* **Debian Jessie** : Debian OS thể hiện cho sự phát triển của UNIX trên Raspberry Pi qua 4 version.
* V1. Version đầu tiên của Debian cho Pi có tên Debian Squeeze.
* V2. Sau Squeeze, đó là Debian Wheezy.
* V3. Raspbian (Debian Wheezy cải tiến) là phiên bản 3 của Debian cho Pi
* V4. Phiên bản hiện tại của Debian là Debian Jessie. (Debian 8.0)
* **Ubuntu MATE** : Ubuntu MATE được hỗ trợ cho Pi 2,3.
* **Window 10** : Hỗ trợ cho các dòng Pi 2,3 . Window 10 là phiên bản free, hỗ trợ cho cả Raspberry Pi và Intel Galileo. Đây là điều lý tưởng cho các nhà phát triển ứng dụng có sử dụng các bộ công cụ phát triển của Microsoft. Đây là một platform linh hoạt hỗ trợ cho số lượng lớn các thiết bị bao gồm : tablet. Smartphone, PC, Xbox, Internet of Things.

## Booting với Raspberry Pi

Kiến trúc của máy tính single board như RP chắc chắn sẽ ảnh hưởng tới thiết kế của nó. Nhưng quá trình boot thì vẫn theo chu trình chung mà chúng ta đã nêu ra tại ***1.5***



Hình 3.1 Raspberry Pi booting

CPU trên các board của RP đều dựa trên ARM. Khi được cấp nguồn, chu trình boot sẽ thực hiện như sau:

* RP sẽ có GPU được bật khi board được cấp nguồn, các lõi của vi xử lý ARM vẫn tắt.
* GPU thực hiện bước đầu tiên của bootloader từ ROM tới SoC.Bước đầu tiên này thực hiện việc đọc thẻ SD, load bootcode.bin, đây chính là bootloader thứ 2, cho mọi hệ điều hành đang nằm trên thẻ vào cache L2
* Tiếp theo, bootcode.bin bật SDRAM, load chương trình ở bước thứ 3 loader.bin và khởi động nó.
* Loader.bin đọc start.elf, firmware của GPU
* Start.elf đọc config.txt, cmdline.txt và kernel.img và bắt đầu khời động OS.

Bất cứ OS nào được khởi động bởi tiến trình này sẽ chạy trên Raspberry Pi khi được cấp nguồn

## Chuẩn bị thiết bị phần cứng và phần mềm hỗ trợ cài đặt

### Thiết bị phần cứng

* 1 board mạch Raspberry Pi: trong báo cáo này, nhóm dùng Raspberry Pi 3 Model B.
* [Thẻ nhớ Micro SD 16GB Class 10](http://mlab.vn/raspberry-pi-b-raspberry-pi2/phu-kien-cho-raspberry-pi/1050625-the-nho-micro-sd-16g-class-10-sandisk-ultra-48mb-s-full-hd-video.html). Pi có thể làm việc với thẻ nhớ lên tới 32Gb. Dung lượng thẻ nhớ tối thiểu 8GB. Tốc độ của thẻ nhớ nên từ Class 10 trở lên để hiệu năng sử dụng bộ nhớ của Pi là tốt nhất.
* 1 Cable HDMI (có thể sử dụng [1 Cable HDMI - VGA](http://mlab.vn/raspberry-pi-b-raspberry-pi2/nguon-va-cac-loai-day-ket-noi/1758276-cable-hdmi-to-vga-m-pard.html))
* [1 màn hình có cổng HDMI](http://mlab.vn/raspberry-pi-b-raspberry-pi2/man-hinh-lcd-cho-raspberry-pi) (Có thể sử dụng màn hình desktop với đầu vào là cổng VGA)
* 1 [bộ nguồn cấp 5V/2A micro-USB](http://mlab.vn/raspberry-pi-b-raspberry-pi2/nguon-va-cac-loai-day-ket-noi) cho Raspberry Pi.
* Bàn phím và chuột : sử dụng trong quá trình setup ban đầu cho Raspberry Pi.

### Phần mềm cần thiết

- Phần mềm format thẻ nhớ :

* Trong Window :Sử dụng phần mềm **SD card Formatter 4.0**
* Trong Ubuntu : Sử dụng phần mềm **GParted**

- Phần mềm ghi file .img của hệ điều hành lên thẻ nhớ :**Win32DiskImager**

(File hệ điều hành với version phù hợp với Model của Pi)

# XÂY DỰNG DEVICE DRIVERS CHO RASPBERRY PI

## Xây dựng Device Driver cho màn hình LCD Nokia 5110

### Giới thiệu chung về Nokia 5110

Nokia 5110 là một màn hình LCD đồ họa cơ bản cho nhiều ứng dụng. Ban đầu nó được dự định như là một màn hình điện thoại di động. Điều này được gắn trên một cách dễ dàng để hàn PCB.

Nokia 5110  Graphic LCD sử dụng bộ điều khiển PCD8544, trước đó là được sử dụng trong Nokia 3310 màn hình LCD. Các PCD8544 có một sức mạnh CMOS LCD điều khiển/tiêu thụ điện năng thấp, được thiết kế để điểu khiển một màn hình hiển thị đồ họa gồm 48 hàng và 84 cột. Tất cả các chức năng cần thiết cho màn hình đều được cung cấp trong một chip duy nhất, bao gồm thế hệ nguồn LCD và điện áp phân cực trên chip, dẫn đến tối thiểu các thành phần bên ngoài và tiêu thụ điện năng thấp. PCD8544 interfaces để vi điều khiển thông qua chuẩn SPI.

Vi điều khiển PCD8544 được sản suất theo công nghệ n-well CMOS. [1]



Hình 4.1 Hình ảnh thực tế của màn hình Nokia 5110

### Cấu hình và giao tiếp với Nokia 5110

#### Các chân ngoại vi của Nokia 5110

Màn hình LCD Nokia 5110 có tất cả 8 chân để giao tiếp ngoại vi như bảng sau [1]:

|  |  |  |
| --- | --- | --- |
| Pin No: | Pin Name: | Description |
| 1 | Reset | This pin resets the module. It an active low pin (resets when 0V is provided) |
| 2 | Chip Enable (CE) | This pin is made low (0V) to select this particular display when more than one SPI peripherals are used. |
| 3 | Data/Command (DC) | This pin is used to switch between Data mode (high) and Command mode (low) |
| 4 | Serial Input (DIN) | This is the input pin (MOSI) through which serial instructions are sent |
| 5 | Clock (CLK) | All SPI modules require a common clock, this clock source is supplied to this pin |
| 6 | Power (Vcc) | This pin is used to power the display the supply voltage is from 2.7V to 3.3V |
| 7 | Back Light (BL) | This pin powers the backlight of the display (3.3V maximum). |
| 8 | Ground (Gnd) | Connects to the ground of the circuit. |

Bảng 4.1 Cấu hình các chân ngoại vi của LCD Nokia 5110

#### Thông số kỹ thuật của Nokia 5110

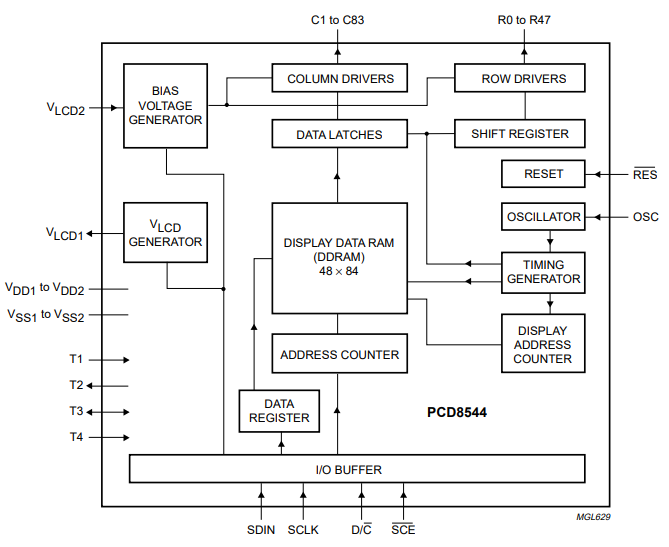
Thuộc tính kỹ thuật [1]:

* Kích thước màn hình 84x48 pixel
* Đơn chíp IC điều khiển PCD 8544
* Có card màn hình 504 byte Ram. (84x48 bits)
* 8pin  (2 pin nguồn, 5 pin giao tiếp SPI và màn hình)
* Điện áp nguồn: 3,5v
* Điện áp mức logic: 0-3,5v
* Serial interface có tốc độ lớn nhất là 4.0 Mbits/s
* Mux rate: 48
* Điện áp bù nhiệt: VLCD
* Giới hạn nhiệt độ trên màn hình -25 đến +70
* On-chip:
* Bộ tạo điện áp cung cấp cho LCD (cũng có thể cung cấp nguồn từ bên ngoài)
* Bộ tạo điện áp phân cực trung gian cho LCD
* Bộ tạo giao động độc lập (cũng có thể sử dụng xung clock từ master)

Vi điều khiển PCD8544 có các chân giao tiếp như Bảng 4-2, Các giao tiếp được thể hiện như Hình 4-2 [1]:

|  |  |
| --- | --- |
| SYMBOL | DESCRIPTION |
| R0 to R47 | LCD row driver outputs |
| Co to C83 | LCD colums driver outputs |
| VSS1, VSS2 | Ground |
| VDD1, VDD2 | Supply voltage |
| VLCD1, VLCD2 | LCD supply voltage |
| T1 | Test 1 input |
| T2 | Test 2 output |
| T3 | Test 3 input/output |
| T4 | Test 4 input |
| SDIN | Serial data input |
| SCLK | Serial clock input |
| D/C | Data/command |
| SCE | Chip enable |
| OSC | Oscillator |
| RES | External reset input |
| Dummy 1,2,3,4 | Not connected |

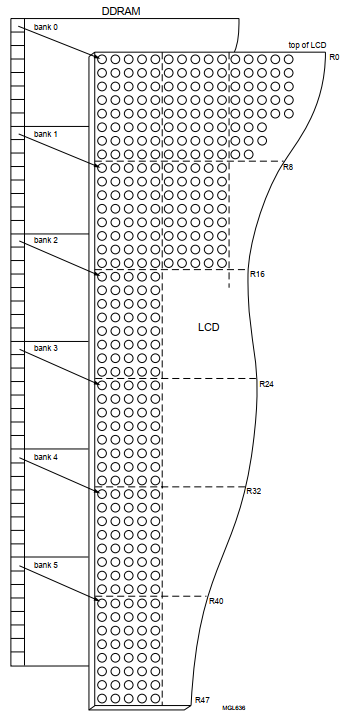
Bảng 4.2 Các chân giao tiếp của PCD8544



Hình 4.2 Cấu trúc bên trong vi điều khiển PCD8544

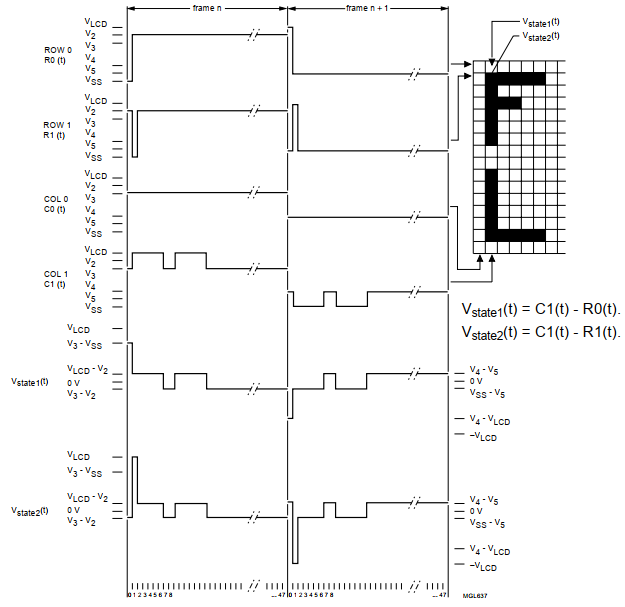
#### Mô tả chức năng của PCD8544

* Oscillator
* Bộ tạo giao động tích hợp trên chip cung cấp tín hiệu clock cho toàn bộ màn hình. Các thành phần bên ngoài không cần thiết (ví dụ một bộ tạo giao động từ bên ngoài) và đầu OSC phải được nối với VDD. Tín hiệu clock từ bên ngoài (ví dụ từ master), nếu được sử dụng sẽ nối với đầu vào ở đây.
* Address Counter (AC)
* Bộ đếm địa chỉ gán các địa chỉ vào RAM trong việc ghi dữ liệu. X-address X6 đến X0 và Y-address Y2 đến Y0 được thiết lập riêng biệt. Sau đó, việc ghi dữ liệu sẽ được thực hiện một cách tự động, AC sẽ tăng tự động lên 1 theo như V flag được đặt cho PCD8544.
* Display Data RAM (DDRAM)
* DDRAM là một static RAM với 48 x 84 bits dùng để lưu trữ data của màn hình hiển thị. RAM được chia thành 6 banks, mỗi banks chứa 84 bytes (6 x 8 x 84 bits). Khi truy cập vào RAM, dữ liệu sẽ được chuyển đến RAM thông qua Serial Interface.
* Truy cập một cách trực tiếp ứng với X-address và số lượng cột đầu ra.



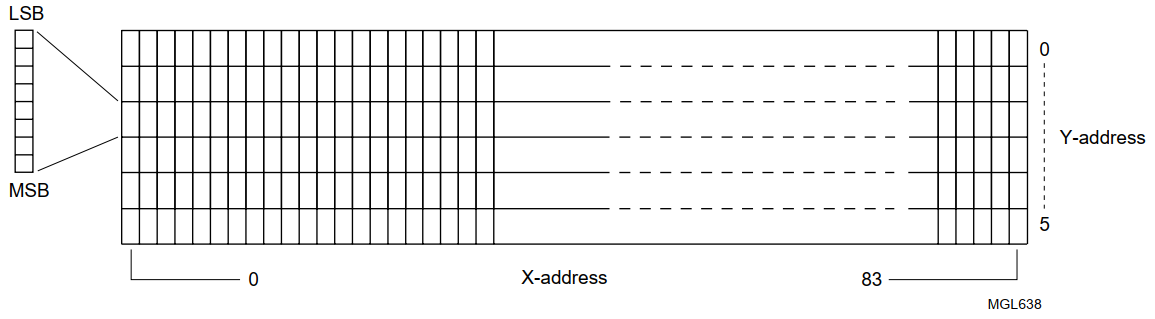
Hình 4.3 Cấu trúc DDRAM của Nokia5110

* Timing generator
* Bộ tạo giao động bên trong giúp cung cấp những xung tín hiệu khác nhau để điều khiển các linh kiện khác nhau bên trong mạch. Sự vận hành của nội chip sẽ không ảnh hưởng tới data buses.
* Display address counter
* Màn hình được tạo ra bởi việc dịch liên tục các hàng của RAM data đi vào ma trận LCD thông qua các outputs cột. Trạng thái của màn hình (tất cả các led on/off và normal/inverse video) được set bởi bits E và D trong lệnh điều khiển màn hình.
* LCD row and column drivers
* PCD8544 chứa các trình điều khiển 48 hàng và 84 cột. Nó kết nối các điện áp phân cực LCD một cách tuần tự để hiển thị dữ liệu tương ứng lên màn hình. Hình 4-3 minh họa driver waveforms của LCD.

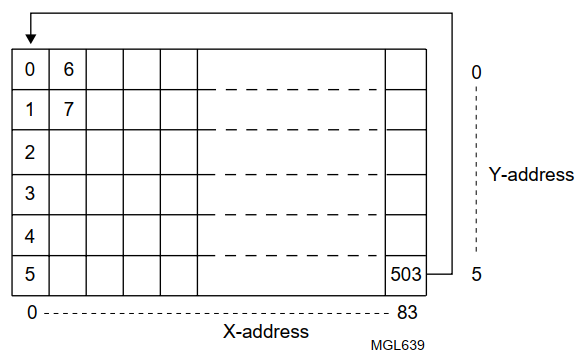


Hình 4.4 LCD driver waveforms

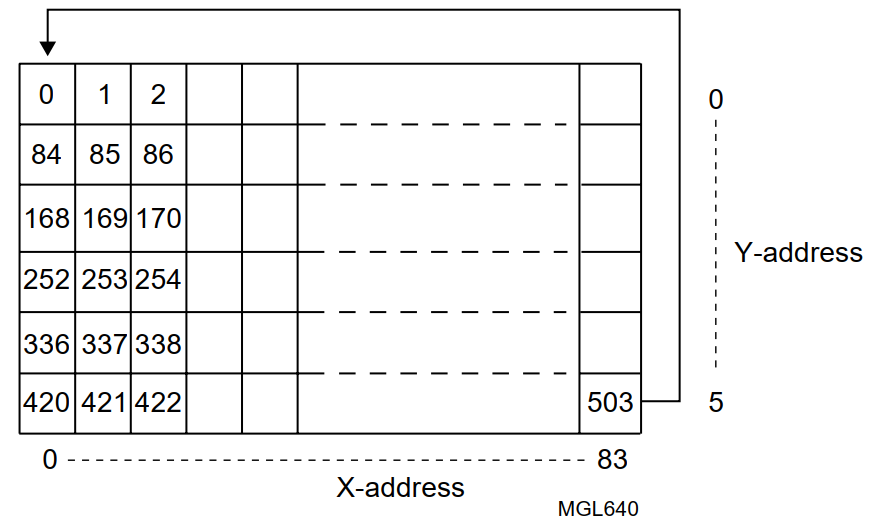
* Addressing
* Data được đưa vào ma trận 48 x 84 bits của RAM của PCD8544 được chỉ ra như Hình 4-5. địa chỉ X từ 0 đến 83, địa chỉ Y từ 0 đến 5. Những địa chỉ ngoài khoảng này không được cho phép.
* Có hai chế độ để đưa data vào RAM: vertial addressing mode (V flag = 1) và horizontal addressing mode (V flag = 0). Hình 4-6, Hình 4-7 thể hiện định dạng RAM và thể hiện các trường hợp của V flag tương ứng.



Hình 4.5 Định dạng của DDRAM

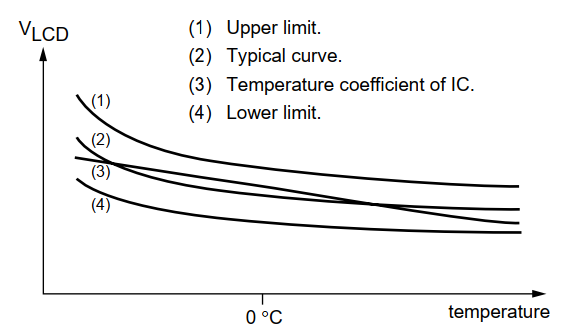


Hình 4.6 Ghi data tuần tự theo chiều dọc (V = 1)



Hình 4.7 Ghi data tuần tự theo chiều ngang (V = 0)

* Temperature Compensation (Sự cân bằng về nhiệt độ)
* Do sự phụ thuộc vào nhiệt độ liên quan đến độ kết dính của tinh thể lỏng cho LCD, điện áp điểu khiển LCD VLCD phải được tăng và duy trì ở nhiệt độ thấp hơn để tối ưu độ tương phản.
* Trong PCD8544, các hệ số tương ứng của VLCD có thể được thiết đặt (có 4 hệ số) bởi hai bits TC1 và TC0. Hình 4-8 thể hiện các hệ số điện áp tương ứng với nhiệt độ.



Hình 4.8 Các hệ số điện áp tương ứng với nhiệt độ của LCD

### Xây dựng device driver nhúng vào Linux

#### Driver Specification

* Struct file\_operations gpio\_lcd5110\_fops: là struct chứa các con trỏ hàm dùng cơ chế systemcall để đọc ghi xuống buffer của thanh ghi trong thư mục /dev (sẽ chứa các hàm liên quan đến đọc, ghi, mở, đóng file).
* Struct my\_device\_data: chứa các thông tin về device, bao gồm bội số <major, minor> trong struct cdev.
* static struct attribute \*gpio\_lcd5110\_attrs[]: là struct chứa các con trỏ hàm dùng để tương tác với sysfs.
* static struct attribute\_group attr\_group: là struct mô tả đối tượng nhóm các thuộc tính của device trong sysfs (sysfs is a [pseudo file system](https://en.wikipedia.org/wiki/Pseudo_file_system) provided by the [Linux kernel](https://en.wikipedia.org/wiki/Linux_kernel) that exports information about various kernel subsystems, hardware devices, and associated [device drivers](https://en.wikipedia.org/wiki/Device_driver) from the kernel's device model to [user space](https://en.wikipedia.org/wiki/User_space) through [virtual files](https://en.wikipedia.org/wiki/Virtual_file) [2]).

#### LCD Specification

* Static int \_init gpio\_lcd5110\_init (void): Hàm này sẽ là hàm được gọi ra đầu tiên khi chúng ta nạp kernel module vào hệ thống (sử dụng lệnh insmod). Nhiệm vụ của hàm này thực hiện:
  + Result=register\_chrdev(gpio\_lcd5110\_major,"gpio\_lcd5110",&gpio\_lcd5110\_fops): Thực hiện đăng ký một character device driver với kernel thông qua một số gpio\_lcd5110\_major (khi được nạp vào hệ thống kernel sẽ nắm giữ số major để định danh thiết bị).
  + gpio\_lcd5110\_kobj=kobject\_create\_and\_add("gpio\_lcd5110",kernel\_kobj->parent): Thực hiện tạo động một struct kobject và đăng ký nó với sysfs, chứa các thuộc tính về đối tượng (Kobject chứa một con trỏ parent cho phép Kobject được sắp xếp trong hệ thống sysfs).
  + result = sysfs\_create\_group(gpio\_lcd5110\_kobj, &attr\_group): Hàm sẽ thực hiện tạo ra một nhóm thư mục kobject liên quan đến device đang nạp, nó sẽ báo lỗi nếu thư mục trong sysfs đã tồn tại.
  + gpio\_request(pins[g], &pins\_name[g]): Hàm sẽ thực hiện khai báo các chân gpio được sử dụng cho device màn hình.
  + gpio\_direction\_output(pins[g], 0): Hàm sẽ thực hiện cấu hình các chân gpio là outputs.
* Static void \_\_exit gpio\_lcd5110\_exit(void): Hàm này sẽ là hàm được gọi ra ngay khi chúng ta gỡ kernel module khỏi hệ thống (sử dụng lệnh rmmod). Nhiệm vụ của hàm này thực hiện:
  + kobject\_put(gpio\_lcd5110\_kobj): Thực hiện xóa kobject trong sysfs đã đăng ký ở trên.
  + unregister\_chrdev(gpio\_lcd5110\_major, "gpio\_lcd5110"): Thực hiện giải phóng character device đã đăng ký khỏi hệ thống, và giải phóng số major để có thể tái sử dụng cho các device khác.
  + gpio\_free(pins[g]): Giải phóng các chân GPIO đã yêu cầu sử dụng ở trên.
  + kfree(memory\_buffer): Giải phóng buffer của device màn hình (chứa data để đẩy vào RAM màn hình).
* Void initLcdScreen(): Hàm này sẽ khởi tạo các tham số ban đầu cho màn hình (LCD Extended Commands, Set LCD Constrast, Set Temp coefficent, LCD bias mode…).
* Void sendByteToLcd (bool cd, unsigned char data): Hàm sẽ thực hiện chức năng truyền một byte (char) cho vi điều khiển lcd thông qua giao thức SPI (Serial Peripheral Interfaces).
* Void clearLcdScreen(): Hàm này sẽ thực hiện xóa màn hình khi được gọi.
* Void writeCharToLcd (char data): Hàm này sẽ thực hiện ghi một ký tự ASCII vào RAM của màn hình, thông qua giao thức SPI (sử dụng hàm sendByteToLcd()).
* Void writeStringToLcd (char\* data): Hàm này thực hiện ghi một chuỗi các ký tự ASCII xuống màn hình.
* Mã ASCII của từng ký tự được định nghĩa ở thư viện “ascii.h” được xây dựng dựa trên chế độ đánh địa chỉ theo chiều ngang (V = 0)

#### Makefile

* Makefile là một dạng file script chứa các thông tin về cấu trúc project (file, sự phụ thuộc, …), các câu lệnh để biên dịch file, các chỉ mục hướng dẫn để build file hoặc dọn file đã biên dịch,…
* Make file ở dạng đơn giản dùng để biên dịch một file sẽ chứa
  + Biến obj-m chỉ ra rằng: object file sẽ được biên dịch theo kiểu kernel module.
  + Cờ -Wall cho phép trình biên dịch hiển thị tất cả các bản tin cảnh báo trong quá trình biên dịch.
  + Thẻ all chứa câu lệnh để biên dịch các module trong thư mục hiện tại.
  + Thẻ clean chứa lệnh xóa tất cả các object file có trong thư mục hiện tại.

obj-m += gpio\_lcd5110.o

all:

make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:

make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean

## Xây dựng Device Driver cho buttons

### Phân tích chung về cấu hình cho cụm nút bấm

* Cụm phím bấm gồm 4 phím bấm, nối với các gpio 12, 16, 20, 21 của raspberry pi tương ứng, được mắc thông qua một điện trở 2kOhm nối lên Vcc.
* Các nút bấm thông qua hành vi của người user sẽ gửi tín hiệu ngắt cho chương trình để xử lý hành vi tương ứng.

### Xây dựng device driver nhúng vào Linux

#### Driver Specification

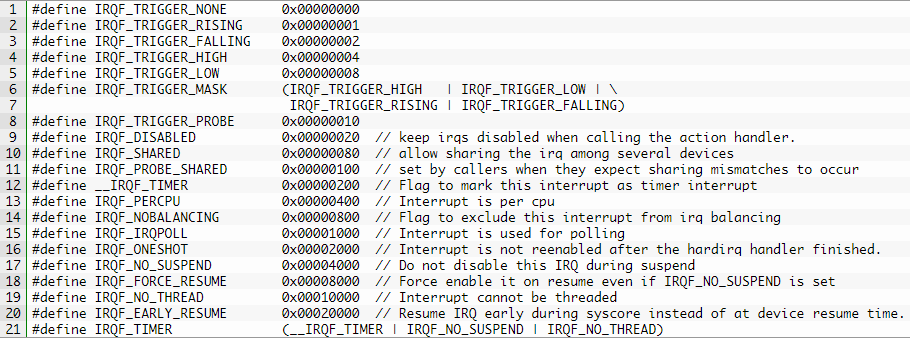
* Nói chung việc đăng ký device driver cụm nút bấm cũng tương tự với màn hình như trên. Chỉ khác một vài chỗ đặc biệt liên quan đến xử lý ngắt.
* Trong hàm static int \_\_init button\_init(void) khi ta khai báo các chân GPIO được khai báo với kernel thông qua thư viện <linux/gpio.h>. Bảng 4.3 đưa ra một số hàm thường dùng để giao tiếp với gpio [3].

|  |  |
| --- | --- |
| static inline bool gpio\_is\_valid(int number) | check validity of GPIO number (max on BBB is 127) |
| static inline int gpio\_request(unsigned gpio, const char \*label) | allocate the GPIO number, the label is for sysfs |
| Static inline int gpio\_export(unsigned gpio, bool direction\_may\_change) | make available via sysfs and decide if it can change from input to output and vice versa |
| Static inline int gpio\_direction\_input(unsigned gpio) | an input line (as usual, return of 0 is success) |
| static inline int gpio\_get\_value(unsigned gpio) | get the value of the GPIO line |
| static inline int gpio\_direction\_output(unsigned gpio, int value) | value is the state |
| static inline int gpio\_set\_debounce(unsigned gpio, unsigned debounce) | set debounce time in ms (platform dependent) |
| Static inline int gpio\_sysfs\_set\_active\_low(unsigned gpio, int value) | set active low (invert operation states) |
| static inline void gpio\_unexport(unsigned gpio) | remove from sysfs |
| static inline void gpio\_free(unsigned gpio) | deallocate the GPIO line |
| static inline int gpio\_to\_irq(unsigned gpio) | associate with an IRQ |

Bảng 4.3 Một số hàm khai báo gpio cơ bản

#### Xử lý ngắt cho các chân gpio

* irqNumber[i] = gpio\_to\_irq(gpioButton[i]): Hàm này sẽ thực hiện tự động lấy một mã ngắt tương ứng với chân gpioButton[i] mà chúng ta đã đăng ký ở trên (mã ngắt irqNumber và gpioNumber khác nhau, nhưng được thực hiện liên kết với nhau theo quan hệ 1-1).
* result=request\_irq(irqNumber[i],(irq\_handler\_t)button\_irq\_handler, IRQF\_TRIGGER\_RISING, "ebb\_gpio\_handler", NULL): Hàm này sẽ thực hiện đăng ký tự động một mã ngắt tương ứng với gpioNumber mà ta đã liên kết ở trên vào kernel (kernel sẽ lưu một bảng ngắt để nắm giữ tín hiệu ngắt này). Một con trỏ hàm trỏ vào hàm thực thi sau khi kernel nhận tín hiệu ngắt này đó là hàm button\_irq\_handler(). Tiếp theo là một thông số liên quan đến cờ ngắt [3] (ảnh hưởng tới cách mắc buttons được thể hiện trong Hình 4.9).



Hình 4.9 Những cờ ngắt có sẵn trong kernel

#### Makefile

* Makefile dùng để build kernel module cho buttons cũng tương tự như Makefile của màn hình, được thể hiện như sau:

obj-m += button\_ctl.o

all:

make -C /lib/modules/$(shell uname -r)/build M=$(DIR)/driver modules

#make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:

make -C /lib/modules/$(shell uname -r)/build M=$(DIR)/driver clean

#make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean

# XÂY DỰNG ỨNG DỤNG ĐỂ TƯƠNG TÁC VỚI DEVICE DRIVERS

## Giới thiệu về ứng dụng xây dựng

* Ứng dụng được xây dựng sẽ mang các chức năng đơn giản, với mục đích để chứng minh sự hoạt động của các device drivers đã nạp vào kernel.
* Ứng dụng sẽ có chức năng thực hiện tính toán để thao tác với các device deivers cụ thể như sau:
  + Nhận tín hiệu ngắt từ cụm nút bấm và mỗi cụm nút bấm được gắn với một hình ảnh (được tạo sẵn trong file).
  + Màn hình sẽ vẽ hình ảnh đó lên màn hình.
  + Một vài chức năng mở rộng khác: có thể đẩy chữ ra màn hình, thông qua bàn phím máy tính kết nối với Raspberry Pi.

## Lập trình đa luồng trong ứng dụng

### Thư viện pthread

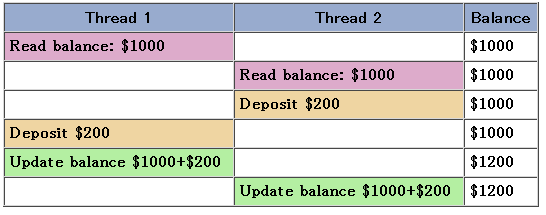
* Pthread được giới thiệu là thư viện implement chuẩn POSIX (IEEE 1003.1c) đặc tả việc tạo nhiều luồng tính toán song song và đồng bộ hóa.
* Hai cách chính để thực thi thư viện:
  + Thực thi toàn bộ quyền trên user space.
  + Thư viện ở cấp kernel được hỗ trợ bởi hệ điều hành (được sử dụng chung cho các hệ thống UNIX (Solaris, Linux, Mac OS X…))
* Phân loại API của pthread gồm có 4 loại:
  + Thread management: Là các hàm sử dụng để tạo, hủy, detached, join thread cũng như set/get các thuộc tính của thread.
  + Mutexes: Là các hàm sử dụng để tạo, hủy, unlocking, loking mutex (“mutual exclusion” : vùng tranh chấp), cũng như set/get các thuộc tính của mutex.
  + Condition variables: Là các hàm để tạo, hủy, đợi hoặc phát tín hiệu dựa trên giá trị của một biến cụ thể.
  + Synchronization: Là các hàm dùng để quản lý việc read/write lock và barriers.
* Những hàm thuộc cùng 1 loại API ở trên sẽ có tên tương tự nhau. Ví dụ: pthread\_create\_xXX, pthread\_join; pthread\_mutex\_XXX, pthread\_cond\_XXX etc

### Tạo thread trong code

* pthread\_t tid1 : Một giá trị được sử dụng như một định danh cho thread mới được tạo ra, nó sẽ được hàm này trả về, chúng ta không cần tạo trước giá trị, ta chỉ cần truyển con trỏ của một biến kiểu pthread\_t là đủ. Giá trị của biến truyển vào sẽ được thay đổi bên trong hàm tạo.
* pthread\_create(&tid1, NULL, thread\_ControlButtons, NULL): Hàm này sẽ thực hiện việc khởi tạo một thread (và bắt đầu chạy thread từ đây), tham số con trỏ hàm sẽ trỏ vào hàm thread\_ControlButtons (thực thi luồng nhận tín hiệu ngắt từ nút bấm).

### Cơ chế mutex

Mutual Exclusion, được sử dụng khi có nhiều thread cùng ghi vào một vùng địa chỉ. Sau đây chúng ta sẽ xem xét một ví dụ kinh điển để minh họa cơ chế này.



Hình 5.1 Ví dụ đơn giản về cơ chết Mutex

Từ ví dụ trên ta thấy, khi sử dụng hai thread để tính toán tiền gửi, trong tài khoản ban đầu là $1000, nạp 2 lần $200 mà số tiền cuối trong tài khoản khi hết xử lý là $1200 (mất $200). Tất nhiên trong thực tế chắc chắn không có chuyện này rồi. Để giải quyết tình trạng trên, ta sẽ sử mutex để lock đoạn xử lý liên quan đến Balance. Hay ở trên từ bước Read balance đển bước Update Balance ta phải để chúng trong 1 khối mà không có xử lý khác chen vào. Cơ chế Mutex sẽ giải quyết vấn đề này bằng cách nắm giữ một tài nguyên của một biến, mà không cho các thread khác xử lý đồng thời, cho đến khi thread nắm giữ biến xử lý xong. Điều này là một cách cần thiết và an toàn trong lập trình đa luồng.

## Đọc ghi data xuống buffer của device drivers

Để đọc ghi data (của ảnh) ta chỉ cần dùng các hàm sau:

* fd = open("/dev/gpio\_5110" , O\_RDWR): Hàm này sẽ có chức năng mở tệp (thông qua cơ chế SystemCall mà ta đã định nghĩa ở dưới driver), hàm thành công sẽ trả về một số nguyên (biểu thị index mở file tương ứng, được kernel cấp phát và nắm giữ, trả về -1 tức là việc mở file xảy ra lỗi).
* write(fd, buffer , sizeof(buffer)): Hàm này sẽ thực hiện việc ghi data ở tầng user xuống buffer của file.
* close(fd): Hàm này thực hiện đóng file vừa mở (giải phóng index mở file được kernel cấp phát).

void LCD\_DrawImage()

{

clear\_buffer(buffer);

int fd;

fd = open("/dev/gpio\_5110" , O\_RDWR);

if (fd == -1)

{

fprintf(stderr, "Error opening file\n");

// exit(-1);

return;

}

write(fd, buffer , sizeof(buffer));

close(fd);

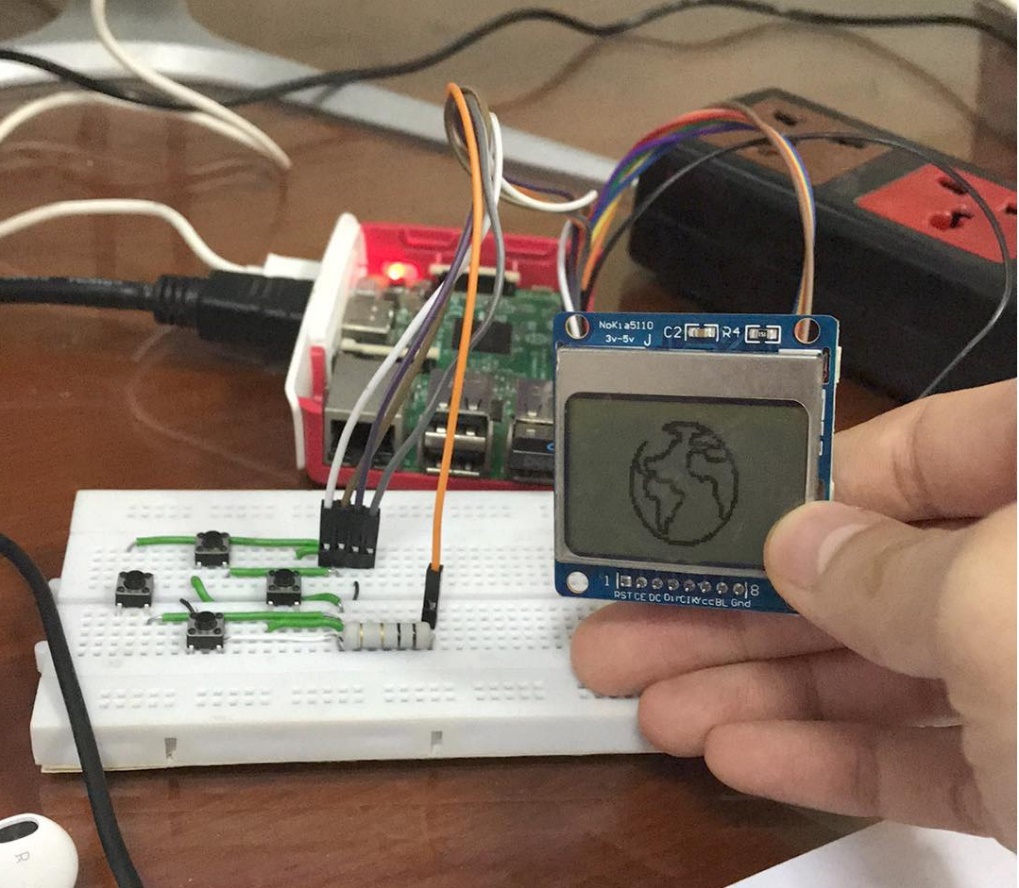
clear\_buffer(buffer);

return;

}

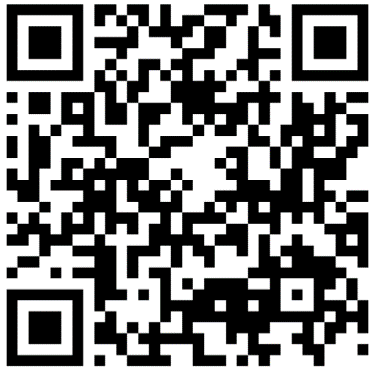
PHỤ LỤC

**Ảnh thực tế của project:**



**Toàn bộ code của project được lưu trong:**

<https://github.com/Thai-VuDuc169/OS_EmbLinuxProject>



KẾT LUẬN

Thông qua quá trình làm bài tập lớn môn hệ điều hành, chúng em đã có cơ hội được tìm hiểu sâu về hoạt động cơ bản của các loại ngắt và cách các chương trình có thể chạy đa luồng. Đây cũng chính là tiền đề giúp chúng em kết hợp những gì đã học trên lớp để có thể xây dựng được một device driver giúp cho Kit Raspberry Pi giao tiếp với nút bấm cũng như màn hình Nokia 5510. Kết quả chúng em đã có thể sử dụng các nút bấm cơ bản để giao tiếp và thay đổi dữ liệu hiển thị trên màn hình LCD thông qua các cổng GPIO của RP. Bên cạnh những kết quả bước đầu đạt được, nhóm cũng nhận thấy rằng kiến thức bản thân còn thiếu và những điều áp dụng thực hành còn đơn giản, chưa có tính thực tiễn cao. Trong tương lai nếu có thời gian, chúng em có thể phát triển lên để có thể chạy được những chương trình cũng như tác vụ phức tạp hơn so với hiện tại.

Chúng em xin gửi lời cảm ơn tới thầy Phạm Doãn Tĩnh đã giảng dạy và hỗ trợ nhiều trong quá trình khi chúng em thực hiện bài tập lớn môn Hệ điều hành.

# TÀI LIỆU THAM KHẢO

|  |  |
| --- | --- |
| [1] | "https://www.sparkfun.com," 12 4 1999. [Online]. Available: https://www.sparkfun.com/datasheets/LCD/Monochrome/Nokia5110.pdf. |
| [2] | wikipedia, "https://en.wikipedia.org," 13 11 2020. [Online]. Available: https://en.wikipedia.org/wiki/Sysfs. |
| [3] | "http://derekmolloy.ie," 26 4 2015. [Online]. Available: http://derekmolloy.ie/kernel-gpio-programming-buttons-and-leds/?fbclid=IwAR2shn2Fbh-lXM34gSG-VTK3GMFwvN\_gv5A4sl8TYHTM7yF-cd80PIlIn9g. |
| [4] | "https://mlab.vn," 15 10 2019. [Online]. Available: https://mlab.vn/index.php?\_route\_=bai-viet-ki-thuat/hoc-raspberry-pi. |