**TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI**

**ĐỒ ÁN TỐT NGHIỆP**

**Thiết kế thiết bị đóng cắt 4 kênh sử dụng WiFi**

**Nguyễn Tuấn Anh**

anh.nt173616@sis.hust.edu.vn

**Ngành Kĩ thuật điện**

**Chuyên ngành Thiết bị điện – Điện tử**

|  |  |
| --- | --- |
| **Giảng viên hướng dẫn:** | TS. Nguyễn Văn Ánh  Chữ ký của GVHD |
| **Bộ môn:** | Thiết bị điện – Điện tử |
| **Viện:** | Điện |

**HÀ NỘI, 01/2021**

-------------------------------------------------------------------------------------------

# Phiếu giao nhiệm vụ đồ án tốt nghiệp

# Lời cảm ơn

Lời đầu tiên, con xin gửi lời cảm ơn đến ông bà, bố mẹ đã dạy dỗ, chỉ bảo và nuôi nấng con thành người. Bố mẹ không chỉ là đấng sinh thành mà còn là những người thầy, người bạn yêu thương con một cách vô điều kiện. Suốt quãng thời gian là học sinh – sinh viên, bố mẹ đã không quản ngại vất vả, kiếm những đồng tiền bằng mồ hôi nước mắt cho con ăn học, không để con phải thiệt thòi so với bạn bè. Con cảm ơn bố mẹ rất nhiều.

Để đạt được thành quả như ngày hôm nay, em xin gửi lời cảm ơn đến thầy Nguyễn Văn Ánh, Bộ môn Thiết bị điện – Điện tử. Thời gian học và làm đồ án với thầy trên trường đã giúp em học hỏi được rất nhiều điều bổ ích cũng như rèn luyện được các kĩ năng giao tiếp và chia sẻ kiến thức tới mọi người.

Em cũng xin gửi lời cảm ơn đến toàn thể các thầy các thầy, các cô đã truyền đạt cho chúng em kiến thức và những kinh nghiệm quý báu bằng cả tâm huyết, giúp chúng em có hành trang tự tin bước vào đời.

Em xin cảm ơn tất cả những người bạn đã đồng hành cùng em trong suốt năm tháng Bách Khoa, luôn là chỗ dựa tinh thần vững chắc trong suốt quá trình học tập. Thời gian học ở Bách Khoa sẽ mãi là những kí ức tuyệt đẹp trong trí nhớ của em.

Trong quá trình làm đồ án tốt nghiệp, em đã cố gắng hết mình nhưng do thời gian và trình độ bản thân còn hạn chế nên đồ án tốt nghiệp của em còn nhiều thiếu sót. Em rất mong nhận được sự chia sẻ, và những ý kiến đóng góp từ thầy cô và các bạn để có thể hoàn thiện đề tài khi đưa và áp dụng thực tế.

Người thực hiện đề tài

Nguyễn Tuấn Anh

# Tóm tắt nội dung đồ án

Trong những năm gần đây, số lượng các thiết bị điện thông minh với khả năng tự động hóa và kết nối với các thiết bị khác đang xuất hiện ngày một nhiều hơn mang đến nhiều lợi ích cho cuộc sống hàng ngày và lĩnh vực sản xuất nông nghiệp, công nghiệp cũng như dịch vụ. Với mong muốn thiết kế một hệ thống các thiết bị có khả năng kết nối với nhau từ xa qua mạng internet, cung cấp khả năng điều khiển, giám sát các thiết bị không phụ thuộc vào khoảng cách, tôi quyết định chọn đề tài “Thiết kế thiết bị đóng cắt 4 kênh qua WiFi”. Đồ án này trình bày các nội dung đã thực hiện để xây dựng thiết bị đóng cắt từ xa và cung cấp giao diện điều khiển và theo dõi từng kênh đóng cắt của thiết bị.

Dưới đây là đề mục các chương được trình bày trong đồ án. Đồ án được chia làm 5 chương chính:

Chương 1: Giới thiệu đề tài

Chương 2: Thiết kế phần cứng

Chương 3: Thiết kế chương trình cho vi điều khiển

Chương 4: Phát triển chương trình ứng dụng điều khiển

Chương 5: Mô hình và kết quả đạt được

Trong mỗi chương sẽ có các đề mục nhỏ nhằm phân tích cụ thể và nêu ra các bước cần triển khai cho từng nhiệm vụ.

Contents

[Phiếu giao nhiệm vụ đồ án tốt nghiệp 1](#_Toc61880887)

[Lời cảm ơn 1](#_Toc61880888)

[Tóm tắt nội dung đồ án 2](#_Toc61880889)

[Chương 1. Giới thiệu đề tài 4](#_Toc61880890)

[1.1. Đặt vấn đề 4](#_Toc61880891)

[1.2. Giới thiệu chung về thiết bị điện thông minh 4](#_Toc61880892)

[1.3. Một số thiết bị điện thông minh trong thực tế 5](#_Toc61880893)

[1.4. Mô tả tổng quan hệ thống 6](#_Toc61880894)

[1.5. Mục tiêu và nhiệm vụ của đề tài 6](#_Toc61880895)

[1.6. Giải pháp 7](#_Toc61880896)

[1.6.1. Giải pháp phần cứng 7](#_Toc61880897)

[1.6.2. Giải pháp phần mềm 7](#_Toc61880898)

[1.6.3. Công cụ lập trình và môi trường phát triển 7](#_Toc61880899)

[Chương 2. Thiết kế phần cứng 8](#_Toc61880900)

[2.1. Giới thiệu các linh kiện sử dụng 8](#_Toc61880901)

[2.1.1. Vi điều khiển ESP8266 8](#_Toc61880902)

[2.1.2. Triac 9](#_Toc61880903)

[2.1.3. Triac driver 10](#_Toc61880904)

[2.1.4. Module chuyền đổi nguồn AC/DC 11](#_Toc61880905)

[2.1.5. Các linh kiện thụ động khác 11](#_Toc61880906)

[2.2. Sơ đồ tổng quan phần cứng 13](#_Toc61880907)

[2.3. Thiết kế mạch 14](#_Toc61880908)

[2.3.1. Thiết kế khối đóng cắt 14](#_Toc61880909)

[2.3.2. Thiết kế khối đọc tín hiệu nút bấm 14](#_Toc61880910)

[2.3.3. Thiết kế khối điều khiển 15](#_Toc61880911)

[2.3.4. Sơ đồ nguyên lý phần cứng 15](#_Toc61880912)

[2.3.5. Mạch in 15](#_Toc61880913)

[Chương 3. Phát triển firmware 15](#_Toc61880914)

[3.1. Giới thiệu một số giao thức truyền thông sử dụng trong thiết bị điện thông minh 15](#_Toc61880915)

[3.2. Giới thiệu giao thức truyền thông MQTT 16](#_Toc61880916)

[3.2.1. Mô hình của giao thức MQTT 16](#_Toc61880917)

[3.2.2. Tại sao lại lựa chọn giao thức MQTT? 17](#_Toc61880918)

[3.2.3. Thông tin server và quy định bản tin MQTT 17](#_Toc61880919)

[3.3. Lưu đồ thuật toán và chương trình cho Vi điều khiển 18](#_Toc61880920)

[Chương 4. Phát triển ứng dụng điều khiển 29](#_Toc61880921)

[4.1. Giới thiệu ngôn ngữ Python và thư viện Kivy 29](#_Toc61880922)

[4.1.1. Ngôn ngữ Python 29](#_Toc61880923)

[4.1.2. Thư viện Kivy cho Python 29](#_Toc61880924)

[4.2. Lưu đồ thuật toán và chương trình cho ứng dụng điều khiển 30](#_Toc61880925)

[4.2.1. Chương trình logic chính 30](#_Toc61880926)

[4.2.2. Chương trình giao diện 36](#_Toc61880927)

[Chương 5. Mô hình và kết quả đạt được 41](#_Toc61880928)

[Kết luận 41](#_Toc61880929)

[Tài liệu tham khảo 42](#_Toc61880930)

# Danh mục hình vẽ

# Danh mục bảng biểu

# Danh mục các từ thuật ngữ và viết tắt

# Chương 1. Giới thiệu đề tài

## Đặt vấn đề

Khoa học và công nghệ đang phát triển với tốc độ rất nhanh, đặc biệt trong lĩnh vực thiết bị điện và điện tử. Các thiết bị điện và điện tử hiện nay ngoài khả năng thực hiện các chức năng cơ bản còn có được tích hợp thêm khả năng tính toán và kết nối truyền thông nhờ sự phát triển mạnh của lĩnh vực điện tử nhúng. Với khả năng tính toán, các thiết bị có thể có các chế độ tự động, tự giám sát, … Với khả năng kết nối truyền thông và đặc biệt là các phương thức truyền thông không dây, các thiết bị điện có thể giao tiếp với nhau, từ đó cung cấp khả năng tự động hóa ở mức cao hơn cũng như khả năng điều khiển và giám sát quá trình hoạt động của các thiết bị từ xa. Tuy nhiên hiện nay các thiết bị điện còn sử dụng nhiều giao thức truyền thông không dây khác nhau như hồng ngoại, Bluetooth, Zwave, Zigbee, WiFi, … Sự thiếu đồng nhất về giao thức truyền thông này khiển cho việc kết nối giữa các thiết bị sử dụng giao thức khác nhau trở nên rất khó khăn. Trong khi đó, nhờ sự phát triển của công nghệ, cơ sở hạ tầng internet nói chung và WiFi nói riêng đã được mở rộng và nâng cấp với tốc độ truyền dữ liệu cao hơn, số lượng thiết bị cho phép kết nối vào mạng tăng được tăng từ 232 thiết bị với IPv4 lên 2128 thiết bị nhờ IPv6. Với sự phát triển của hạ tầng internet, số lượng các thiết bị IoT (Internet vạn vật) đã bùng nổ. Các hệ thống trong nông nghiệp, công nghiệp, dịch vụ và gia đình sử dụng internet và WiFi ngày càng phổ biến, mang đến cơ hội năng cao hiệu năng, độ tin cậy, lợi ích kinh tế cũng như sự tiện nghi cho con người.

Nhận thấy sự phát triển của các thiết bị điện thông minh ở Việt Nam còn nhiều hạn chế, các thiết bị thông minh hiện nay còn sử dụng nhiều chuẩn giao tiếp không dây khác nhau, chưa được thống nhất, tôi quyết định thiết kế Thiết bị đóng cắt 4 kênh sử dụng WiFi

## Giới thiệu chung về thiết bị điện thông minh

Vị trí của thiết bị điện thông minh ngày cảng trở nên quan trọng trong đời sống thường ngày cũng như trong các lĩnh vực sản xuất nông nghiệp, công nghiệp và dịch vụ. Các thiết bị này đem lại sự tiện lợi khi sử dụng, góp phần giúp con người có được sự tiện thoải mái, tiện nghi, tiết kiệm điện và công sức trong việc điều khiển các thiết bị điện trong nhà.

Các thiết bị điện thông minh trong nhà là các thiết bị điện mà bạn có thể điều khiển chúng thông qua những thao tác mạng internet hoặc kết nối không dây của các thiết bị bạn đang sử dụng như Laptop, Smartphone. Thậm chí các loại thiết bị điện thông minh có khả năng cảm ứng nhiệt, ánh sáng giúp cho các thao tác đóng cắt được tự động, không cần đến sự can thiệp của người dùng.

Thiết bị điện thông minh là hệ thống cho phép kết nối các thiết bị sử dụng điện trong nhà với nhau. Thiết bị mang đến nhiều cách để điều khiển, có khả năng tự động xử lý và thông báo cho người sử dụng. Ngoài ra, có khả năng tương tác được với các thông số môi trường, giúp người sử dụng có thể giám sát và điều khiển các thiết bị từ xa, đem lại sự an toàn, tiện nghi, linh hoạt, tiết kiệm.

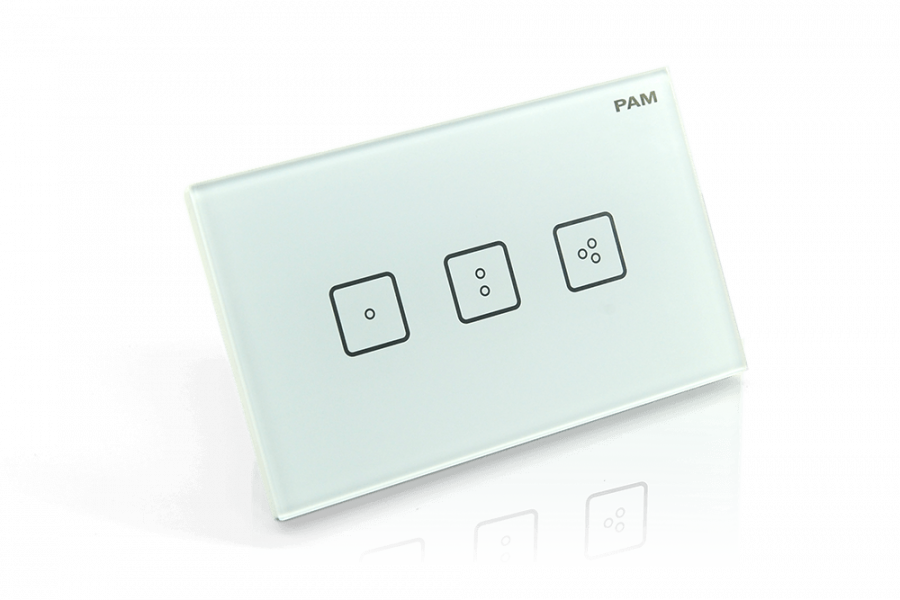
Xu thế thiết bị điện thông minh là phát triển công nghệ vi mạch ngày càng nhỏ, công nghệ nano, tốc độ xử lý ngày càng nhanh, với chip đa nhân, khả năng kết hợp trao đổi dữ liệu với các thiết bị với nhau, các chuẩn giao tiếp dùng dây dẫn và đặc biệt là giao tiếp không dây như hồng ngoại, RF, WiFi, zigbee, …

## Một số thiết bị điện thông minh trong thực tế

Các thiết bị điện thông minh phát triển với tốc độ rất nhanh trong thời gian gần đây, mang lại nhiều tiện ích và cuộc sông tiện nghi cho người sử dụng. Một số thiết bị thông minh tiêu biểu có thể kể đến như:

* Công tắc thông minh cho phép điều khiển tại chỗ thông qua mặt cảm ứng và điều khiển từ xa thông qua ứng dụng trên thiết bị di động

Hình 1. Công tắc 3 kênh điều khiển qua WiFi



* Ổ cắm WiFi cho phép đóng cắt cung cấp điện ở ổ cắm thông qua ứng dụng điều khiển trên điện thoại thông minh và laptop kết nối Internet

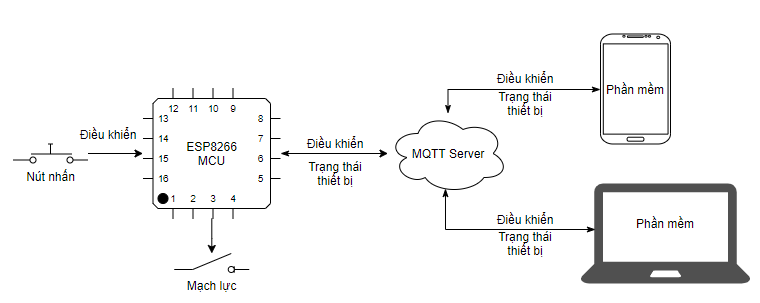
Hình 2. Ổ cắm điều khiển qua WiFi



## Mô tả tổng quan hệ thống

Hệ thống điều khiển đóng cắt 4 kênh qua WiFi gồm 3 thành phần chính: Thiết bị đóng cắt và điều khiển tại hiện trường, hệ thống server truyền dẫn dữ liệu, phần mềm điều khiển và giám sát từ xa.

Hình 3. Mô hình tổng quan của hệ thống



Phần thiết bị đóng cắt và điều khiển tại trường có chức năng đóng cắt trực tiếp các thiết bị điện thông qua Triac và các nút nhấn. Việc trao đổi dữ liệu giữa thiết bị đóng cắt, phần mềm điều khiển và server được thực hiện thông qua giao thức IoT thông dụng hiện nay là MQTT. Phía server MQTT có nhiệm vụ nhận và truyển các bản tin giữa thiết bị điều khiển và thiết bị trường.

## Mục tiêu và nhiệm vụ của đề tài

Đồ án tập trung vào phát triển thiết bị đóng cắt 4 kênh sử dụng WiFi, bao gồm:

* Phát triển phần cứng thiết bị:
  + Thiết kế mạch lực
  + Thiết kế mạch nút nhấn giao tiếp người – máy
  + Thiết kế khối điều khiển
* Phát triển Firmware cho thiết bị:
  + Phát triển chức năng điều khiển mạch lực
  + Phát triển chức năng giao tiếp người – máy
  + Phát triển trao đổi dữ liệu thiết bị và Server qua giao thức MQTT
* Phát triển Software cho thiết bị trên điện thoại và máy tính:
  + Phát triển ứng dụng cho phép hiển thị trạng thái và điều khiển thiết bị
  + Phát triển ứng dụng trao đổi dữ liệu thiết bị và Server qua giao thức MQTT

## Giải pháp

### Giải pháp phần cứng

Thiết bị được xây dựng xung quanh kit NodeMCU - vi điều khiển ESP8266 ở mạch điều khiển và phần tử bán dẫn Triac ở mạch lực.

### Giải pháp phần mềm

Hệ thống được xây dựng bao gồm phần điều khiển đóng cắt điện áp xoay chiều tại chỗ thông qua nút nhấn và điều khiển qua ứng dụng trên máy tính và điện thoại thông minh. Tôi quyết định sử dụng giao thức MQTT để truyền thông, phát triển firmware dựa trên thư viện ESP8266 cho Arduino và phát triển ứng dụng dựa trên thư viện Kivy cho ngôn ngữ Python.

Giao thức MQTT là giao thức phổ biến trong thời đại IoT ngày nay. Đây là một giao thức gọn nhẹ, được thiết kế để kết nối các thiết bị mà có mạng băng thông thấp rất phù hợp với hệ thống điều khiển đóng cắt này.

Kivy là một thư viện mã nguồn mở được viết bằng Python để phát triển ứng dụng, đặc biệt là giao diện người dùng. Ứng dụng viết bằng Kivy có thể chạy trên cả Windows, Linux, OS X, iOS và Android.

### Công cụ lập trình và môi trường phát triển

1. Công cụ để lập trình ESP8266

Trong đồ án sử dụng Arduino IDE làm nền tảng lập trình bởi vì các lý do sau:

* Miễn phí và cộng đồng phát triển mạnh
* Hỗ trợ nhiều thư viện

1. Công cụ để lập trình ứng dụng điều khiển

Trong đồ án sử dụng Python IDE làm nền tảng cho lập trình ứng dụng vì các lý do sau:

* Python có cú pháp đơn giản
* Python miễn phí và cộng đồng phát triển mạnh
* Hệ thống thư viện hỗ trợ lớn
* Hỗ trợ trên nhiều hệ điều hành

1. Công cụ thiết kế phần cứng

Phần cứng của đồ án được thiết kế bằng phần mềm Altium Designer 17. Đây là phần mềm chuyên dụng được dùng trong lĩnh vực thiết kế mạch điện tử.

# Chương 2. Thiết kế phần cứng

Chương này sẽ mô tả chi tiết các thành phần phần cứng của thiết bị và kèm với file nguyên lý. Khối điều khiển của thiết bị được xây dựng dựa trên module ESP8266 nodeMCU, bao gồm các chức năng: đọc tín hiệu nút bấm, điều khiển đóng cắt triac thông qua các triac driver.

## Giới thiệu các linh kiện sử dụng

### Vi điều khiển ESP8266

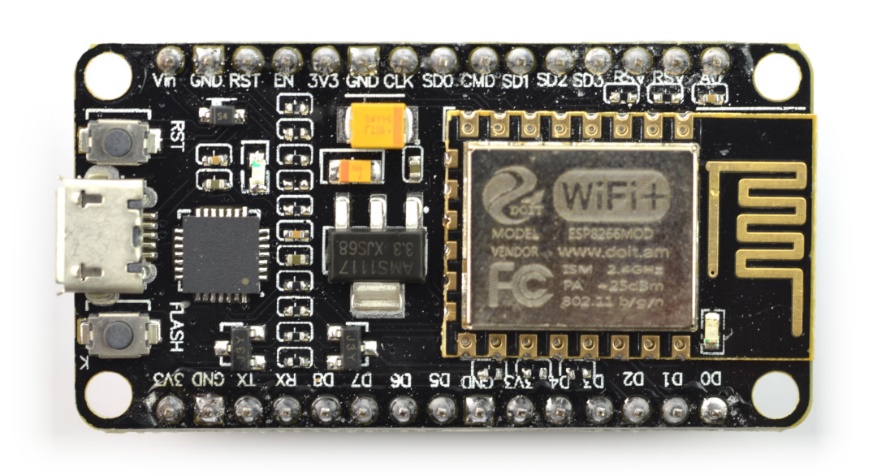
ESP8266 là dòng chip tích hợp Wi-Fi 2.4Ghz có thể lập trình được, rẻ tiền được sản xuất bởi một công ty bán dẫn Trung Quốc: Espressif Systems.

Được phát hành đầu tiên vào tháng 8 năm 2014, đóng gói đưa ra thị trường dạng Mô dun ESP-01, được sản xuất bởi bên thứ 3: AI-Thinker. Có khả năng kết nối Internet qua mạng Wi-Fi một cách nhanh chóng và sử dụng rất ít linh kiện đi kèm. Với giá cả có thể nói là rất rẻ so với tính năng và khả năng ESP8266 có thể làm được.

ESP8266 có một cộng đồng các nhà phát triển trên thế giới rất lớn, cung cấp nhiều Module lập trình mã mở giúp nhiều người có thể tiếp cận và xây dựng ứng dụng rất nhanh.

Hiện nay tất cả các dòng chip ESP8266 trên thị trường đều mang nhãn ESP8266EX, là phiên bản nâng cấp của ESP8266. Trong đồ án này sử dụng kit phát triển cho chip ESP8266 là NodeMCU

Hình 4. Vi mạch phát triển nodeMCU

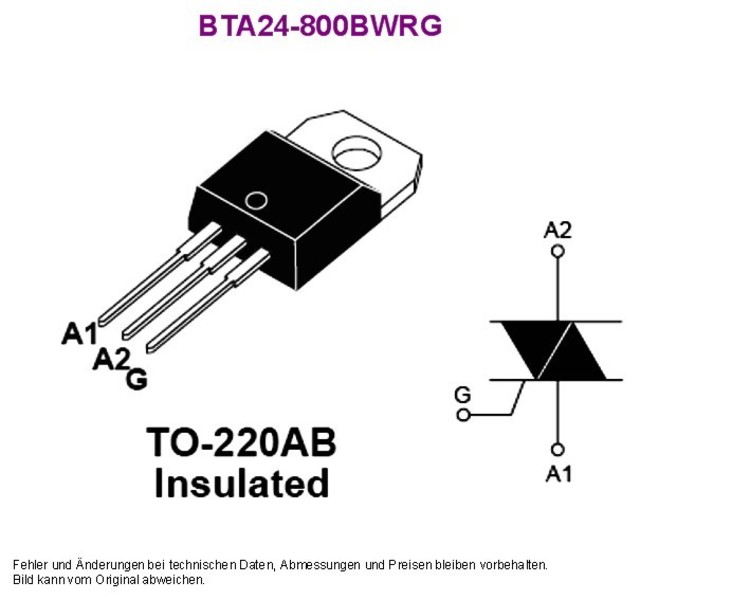


Thông số phần cứng:

* 32-bit RISC CPU: Tensilica Xtensa LX106 running at 80 MHz
* Hổ trợ Flash ngoài từ 512KiB đến 4MiB
* 64KBytes RAM thực thi lệnh
* 96KBytes RAM dữ liệu
* 64KBytes boot ROM
* Chuẩn wifi EEE 802.11 b/g/n, Wi-Fi 2.4 GHz Tích hợp TR switch, balun, LNA, khuếch đại công suất và matching network Hổ trợ WEP, WPA/WPA2, Open network
* Tích hợp giao thức TCP/IP
* Hổ trợ nhiều loại ăng-ten
* 16 chân GPIO
* Hổ trợ SDIO 2.0, UART, SPI, I²C, PWM, I²S với DMA
* 1 ADC 10-bit
* Dải nhiệt độ hoạt động rộng: -40oC ~ 125oC

### Triac

Hình 5. Hình ảnh thực và mô hình Triac

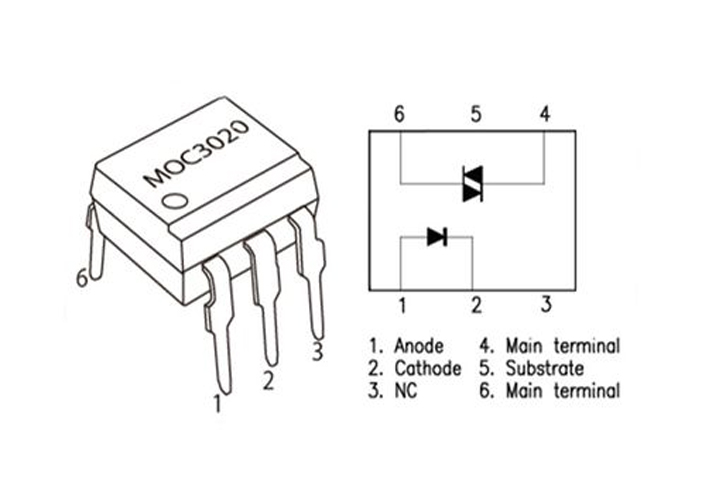


Triac là phần tử bán dẫn có cấu trúc bán dẫn gồm năm lớp, tạo nên cấu trúc p-n-p-n như ở Tiristo theo cả hai chiều giữa cực A1 và A2. Về nguyên tắc, triac hoàn toàn có thể coi tương đương với 2 tiristo đấu song song ngược. Triac có thể điều khiển mở dẫn dòng bằng cả xung dòng dương (dòng đi vào cực điều khiển) hoặc dung dòng âm (dòng đi ra khỏi cực điều khiển). Tuy nhiên xung dòng điều khiển xung dòng điều khiển âm có độ nhạy kém hơn, nghĩa là dòng chỉ có thể chạy qua triac khi điện áp giữa A1 và A2 phải lớn hơn một giá trị nhất định, lớn hơn khi dùng dòng điều khiển dương. Vì vậy trong thực tế để đảm bảo tính đối xứng của dòng điện qua triac thì sử dụng xung điều khiển âm là tốt hơn cả. Triac đặc biệt hữu ích trong các ứng dụng cần điều chỉnh điện áp xoay chiều hoặc các công tắc tơ tĩnh ở dải công suất vừa và nhỏ.

### Triac driver

Triac driver là IC cách ly quang được thiết kế đặc biệt để cung cấp dòng điều khiển cho triac đồng thời cách ly mạch điều khiển với mạch lực. Triac driver thường gồm đầu vào là 1 diode phát quang (LED) và đầu ra gắn với 1 chip điều khiển Triac nhạy sáng. Khi có dòng điện bên phía điều khiển LED sẽ phát sáng và chip điều khiển Triac sẽ cho phép dòng điện chạy qua driver. Dòng điện này sẽ chảy vào cực gate của Triac lực và từ đó cho phép đóng cắt tải.

Hình 6. IC cách ly quang điều khiển triac



### Module chuyền đổi nguồn AC/DC

Module nguồn AC-DC Hi-Link HLK-PM01 5VDC 3W có thiết kế nhỏ gọn với vỏ bọc nhựa an toàn, chuyên nghiệp, được sử dụng để chuyển nguồn xoay chiều AC sang 5VDC công suất tối đa 3W cấp cho thiết bị, module được sản xuất bởi hãng Hi-Link chuyên về các module nguồn được sử dụng trong công nghiệp với độ bền, chống nhiễu tốt và độ an toàn cao.

Hình 7. Khối chuyển nguồn AC/DC 5V 3W



Thông số kĩ thuật:

* Điện áp ngõ vào: 100~240 VAC/50~60 Hz
* Điện áp ngõ ra: 5 VDC
* Công suất trung bình: 3W
* Răng cưa và nhiễu nhỏ
* Bảo vệ quá tải và ngắn mạch
* Hiệu suất 72%

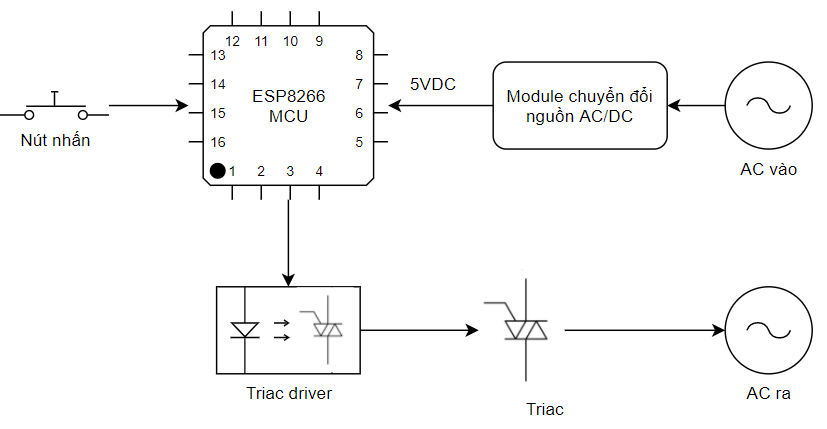
### Các linh kiện thụ động khác

* Điện trở: Điện trở là một linh kiện điện tử thụ động gồm 2 tiếp điểm kết nối, thường được dùng để hạn chế cường độ dòng điện chảy trong mạch, điều chỉnh mức độ tín hiệu, dùng để chia điện áp, kích hoạt các linh kiện điện tử chủ động như transistor, tiếp điểm cuối trong đường truyền điện và có trong rất nhiều ứng dụng khác. Điện trở công suất có thể tiêu tán một lượng lớn điện năng chuyển sang nhiệt năng có trong các bộ điều khiển động cơ, trong các hệ thống phân phối điện. Các điện trở thường có trở kháng cố định, ít bị thay đổi bởi nhiệt độ và điện áp hoạt động. Biến trở là loại điện trở có thể thay đổi được trở kháng như các núm vặn điều chỉnh âm lượng. Các loại cảm biến có điện trở biến thiên như: cảm biến nhiệt độ, ánh sáng, độ ẩm, lực tác động và các phản ứng hóa học. Điện trở là loại linh kiện phổ biến trong mạng lưới điện, các mạch điện tử, Điện trở thực tế có thể được cấu tạo từ nhiều thành phần riêng rẽ và có nhiều hình dạng khác nhau, ngoài ra điện trở còn có thể tích hợp trong các vi mạch IC. Điện trở được phân loại dựa trên khả năng chống chịu, trở kháng, ... tất cả đều được các nhà sản xuất ký hiệu trên nó.
* Tụ điện: Tụ điện là một loại linh kiện điện tử thụ động tạo bởi hai bề mặt dẫn điện được ngăn cách bởi điện môi. Khi có chênh lệch điện thế tại hai bề mặt, tại các bề mặt sẽ xuất hiện điện tích cùng điện lượng nhưng trái dấu. Sự tích tụ của điện tích trên hai bề mặt tạo ra khả năng tích trữ năng lượng điện trường của tụ điện. Khi chênh lệch điện thế trên hai bề mặt là điện thế xoay chiều, sự tích luỹ điện tích bị chậm pha so với điện áp, tạo nên trở kháng của tụ điện trong mạch điện xoay chiều. Về mặt lưu trữ năng lượng, tụ điện có phần giống với ắc qui. Mặc dù cách hoạt động của chúng thì hoàn toàn khác nhau, nhưng chúng đều cùng lưu trữ năng lượng điện. Ắc qui có 2 cực, bên trong xảy ra phản ứng hóa học để tạo ra electron ở cực này và chuyển electron sang cực còn lại. Tụ điện thì đơn giản hơn, nó không thể tạo ra electron - nó chỉ lưu trữ chúng. Tụ điện có khả năng nạp và xả rất nhanh. Đây là một ưu thế của nó so với ắc qui.
* Đèn LED: LED (viết tắt của Light Emitting Diode, có nghĩa là diode phát sáng hoặc diode phát quang) là các diode có khả năng phát ra ánh sáng hay tia hồng ngoại, tử ngoại. Cũng giống như diode, LED được cấu tạo từ một khối bán dẫn loại p ghép với một khối bán dẫn loại n.
* Nút bấm: Nút ấn là một loại công tắc đơn giản điều khiển hoạt động của máy hoặc một số loại quá trình. Hầu hết, các nút nhấn là nhựa hoặc kim loại. Hình dạng của nút ấn có thể phù hợp với ngón tay hoặc bàn tay để sử dụng dễ dàng. Tất cả phụ thuộc vào thiết kế cá nhân. Nút ấn có 2 loại chính là nút nhấn thường mở hoặc nút nhấn thường đóng. Nút nhấn có ba phần: Bộ truyền động, các tiếp điểm cố định và các rãnh. Bộ truyền động sẽ đi qua toàn bộ công tắc và vào một xy lanh mỏng ở phía dưới. Bên trong là một tiếp điểm động và lò xo. Khi nhấn nút, nó chạm vào các tiếp điểm tĩnh làm thay đổi trạng thái của tiếp điểm. Trong một số trường hợp, người dùng cần giữ nút hoặc nhấn liên tục để thiết bị hoạt động. Với các nút nhấn khác, chốt sẽ giữ nút bật cho đến khi người dùng nhấn nút lần nữa. Công tắc nút nhấn sử dụng nhiều trong các ứng dụng khác nhau như máy tính, điện thoại nút nhấn và nhiều thiết bị gia dụng. Bạn có thể nhìn thấy chúng trong nhà, văn phòng và trong các ứng dụng công nghiệp ngày nay. Chúng có thể bật, tắt máy hoặc làm cho thiết bị thực hiện các hoạt động cụ thể, như trường hợp với máy tính. Trong một số trường hợp, các nút nhấn có thể kết nối thông qua liên kết cơ học, điều khiển một nút nhấn khác hoạt động. Đa số, các nút sẽ có màu sắc cụ thể để biểu thị mục đích của chúng. Ví dụ như nút nhất màu xanh thường được sử dụng để bật thiết bị hay nút nhấn màu đỏ để tắt thiết bị. Điều này tránh gây nên một sô nhầm lẫn. Nút dừng khẩn cấp thường là các nút ấn lớn, thường có màu đỏ và có đầu lớn hơn để sử dụng dễ dàng hơn.

## Sơ đồ tổng quan phần cứng

Phần cứng của thiết bị được cấp nguồn bởi Module chuyển đổi nguồn từ 220VAC sang 5VDC. Vi điều khiển ESP8266 hoạt động với nguồn cấp 3.3VDC nhưng trên kit phát triển nodeMCU đã có sẵn IC hạ áp AMS1117 từ 5V xuống 3.3VDC nên chỉ cần một module nguồn duy nhất cung cấp 5VDC cho toàn bộ hệ thống.

Hình 8. Mô hình phần cứng thiết bị

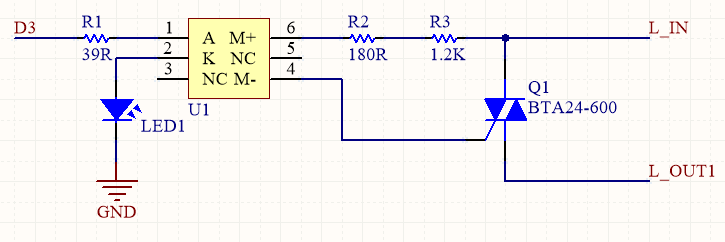


## Thiết kế mạch

### Thiết kế khối đóng cắt

Khối đóng cắt 1 kênh gồm 1 triac driver MOC3020 (U1) và 1 triac BTA24

Hình 9. Nguyên lý phần đóng cắt triac



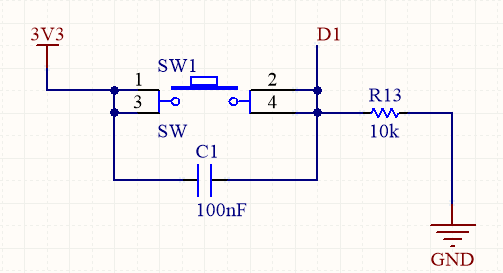
### Thiết kế khối đọc tín hiệu nút bấm

Khối đọc nút nhấn gồm:

* Nút nhấn nhả
* Tụ điện 100nF chống dội nút nhấn
* Điện trở 10K Ohm kéo xuống đất

Nút nhấn được thiết kế hoạt động ở chế độ tích cực mức cao. Khi không nhấn vi điều khiển sẽ đọc chân nút nhấn có mức logic thấp. Khi nút nhấn được nhấn vi điều khiển sẽ đọc chân nút nhấn có mức logic cao và xử lý tương ứng.

Hình 10. Nguyên lý phần nút nhấn



### Thiết kế khối điều khiển

### Sơ đồ nguyên lý phần cứng

### Mạch in

# Chương 3. Phát triển firmware

## Giới thiệu một số giao thức truyền thông sử dụng trong thiết bị điện thông minh

Trong IoT việc giao tiếp giữa các thiết bị với nhau là một trong các yếu tố cơ bản, từ đó dẫn đến yêu cầu phải có một giao thức chung để các thiết bị với phần cứng khác nhau có thể giao tiếp với nhau. Các giao thức truyền dữ liệu phổ biến trong lĩnh vực thiết bị điện thông minh hiện nay là CoAP, MQTT, AMQP, DSS, …

* Giao thức CoAP: Giao thức được thiết kế cho các thiết bị nhúng, về cơ bản giao thức CoAP tương tự như HTTP nhưng các gói tin nhỏ hơn, chủ yếu là giao thức 1-1 để truyền trạng thái thông tin giữa client và server
* Giao thức AMQP: Giao thức AMQP được thiết kế với các gói tin được định hướng, xếp hàng và định tuyến có độ tin cậy và bảo mật cao
* Giao thức DDS: Giao thức truyền tin được thiết kế cho giao tiếp máy-máy, hiệu năng cao, tương thích tốt giữa các phần cứng và phần mềm khác nhau, thời gian thực, có khả năng mở rộng tốt
* Giao thức MQTT: MQTT là giao thức truyền thông theo mô hình Publish/Subscribe.

## Giới thiệu giao thức truyền thông MQTT

MQTT là giao thức truyền thông theo mô hình Publish/Subscribe, sử dụng băng thông thấp, có độ tin cậy cao, có khả năng hoạt động trong môi trường đường truyền không ổn định.

### Mô hình của giao thức MQTT

Hình 11. Mô hình giao thức MQTT

Diagram

Description automatically generated

Mô hình gồm 2 phần chính là là Broker (với vai trò máy chủ) và Client (máy khách)

* Broker đóng vai trò máy chủ đảm nhận vai trò nhận và chuyển các bản tin. Broker có thể cài đặt trên các máy tính nhúng, hoặc sử dụng Broker Cloud
* Client có thể là các thiết bị chấp hành, các cảm biến, cloud nhận dữ liệu, các thiết bị điện thông minh, …
* Client của MQTT gửi và nhận bản tin dựa trên mô hình Topic. Để nhận bản tin của 1 Topic, Client sẽ Subscribe (đăng kí) Topic đó và khi có bản tin được Publish (xuất bản) đến Topic đó Broker sẽ tự động gửi bản tin đến Client.
* Broker nhận tin nhắn từ các Client gửi đến các Topic, chuyển tiếp bản tin hoặc lưu lại. Ngoài ra Broker cũng đảm nhận thêm một số tính năng khác như bảo mật, …
* QoS: Mức độ tin cậy khi gửi bản tin. Có 3 mức QoS: 0,1 và 2

+ QoS 0: Broker/Client gửi bản tin đúng 1 lần

+ QoS 1: Broker/Client gửi bản tin cho đến khi có xác nhận từ bên nhận

+ QoS 2: Broker/Client gửi bản tin và đảm bảo có xác nhận từ cả 2 phía

* Retain: Bản tin có flag Retain sẽ được Broker lưu lại tại Topic và chuyển đến các Client mới Subscribe Topic đó.
* Birth/Death/LWT:

+ Birth là bản tin được gửi tới các Client khi có thiết bị mới kết nối

+ Death là bản tin được gửi khi có thiết bị mất kết nối

+ LWT là bản tin được cài đặt bới Client, sẽ được gửi tới Topic chỉ định khi Client đó mất kết nối

* Mức độ bảo mật: 1 lớp xác thực bằng ID và mật khẩu khi các Client kết nối với Broker. Mức độ bảo mật có thể tăng thêm bằng các giải pháp bảo mật ở tầng mạng.

### Tại sao lại lựa chọn giao thức MQTT?

* Ưu điểm:

+ Yêu cầu băng thông thấp

+ Thích hợp với các thiết bị nhúng

+ Chi phí triển khai thấp

+ Dễ nâng cấp hệ thống

+ Các client dễ kết nối lại khi bị mất kết nối

+ Có các tính năng giúp duy trì kết nối và phát hiện khi các node bị mất kết nối

* Nhược điểm:

+ Sử dụng Broker nên khi Broker gặp sụ cố sẽ cả hệ thống sẽ mất kết nối. Để đảm bảo kết nối thông suốt cần có Broker backup

+ Số lượng Client bị giới hạn bởi cấu hình của Broker

+ Cần sử dụng các biện pháp bên ngoài để tăng tính bảo mật cho hệ thống

Có thể thấy đây là một giao thức gọn nhẹ, được thiết kế để kết nối các thiết bị mà có mạng băng thông thấp rất phù hợp với các thiết bị điện thông minh.

### Thông tin server và quy định bản tin MQTT

* Thông tin server MQTT

|  |  |
| --- | --- |
| Broker | broker.hivemq.com |
| Port | 1883 |

* Quy định bản tin MQTT: Bản tin MQTT được quy định để tạo sự thống nhất giữa các thiết bị, giúp công việc xử lý truyền thông dễ dàng hơn.

|  |  |  |
| --- | --- | --- |
| Nội dung | Topic | Payload |
| Trạng thái thiết bị đóng cắt | status/datnta | /1/x/2/x/3/x/4/x  - x là trạng thái của từng kênh. x = 1: kênh đang dẫn. x = 0: kênh mở ra  VD: /1/0/2/1/3/1/4/0: kênh 1 và 4 đang mở, kênh 2 và 3 đang dẫn. |
| Lệnh điều khiển thiết bị đóng cắt | cmd/datnta | /kênh đóng cắt/điều khiển/thời gian  - Kênh đóng cắt: 1, 2, 3, 4  - Điều khiển: 0 hoặc 1 tương ứng là mở hoặc dẫn  - Thời gian: thời gian dẫn nếu lệnh điều khiển là 1, thời gian trễ mở kênh nếu lệnh điều khiển là 0. Đơn vị tính bằng giây.  VD:  /1/1/60: điều khiển kênh 1 dẫn trong 60 giây rồi mở ra.  /1/1/00: điều khiển kênh 1 dẫn không giới hạn thời gian. |

## Lưu đồ thuật toán và chương trình cho Vi điều khiển

* Lưu đồ thuật toán hoạt động của vi điều khiển: Khi thiết bị được cấp nguồn, hàm Setup sẽ được gọi để cầu hình các chân GPIO vào cho các nút nhấn, các chân GPIO ra để điều khiển triac và đèn hiển thị trạng thái thiết bị, kiểm tra bộ nhớ để tìm cấu hình WiFi. Sau khi kết thúc quá trình khởi tạo ban đầu, vi điều khiển vào vòng lặp vô tận với chu trình lần lượt như sau:
  + Kiểm tra nút nhấn giữ
  + Theo dõi thời gian cho chế độ đóng cắt có hẹn giờ
  + Gửi dữ liệu về trạng thái đóng cắt
  + Kiểm tra kết nối WiFi
  + Kiểm tra kết nối với server MQTT

Ngoài ra chương trình cho vi điều khiển cũng sử dụng ngắt để đọc các nút nhấn nhận tín hiệu từ người sử dụng.

Hình 12. Lưu đồ thuật toán cho vi điều khiển

Diagram

Description automatically generated

* Hàm Setup: Hàm làm nhiệm vụ cấu hình các chân GPIO vào và ra cho thiết bị. Đọc bộ nhớ để xác định kết nối vơi WiFi đã lưu hay đưa thiết bị vào chế độ cấu hình WiFi để người sử dụng nhập WiFi mới. Cài đặt thông tin server MQTT và kết nối đến server MQTT.

Hình 13. Lưu đồ nguyên lý hàm cấu hình ban đầu

Diagram

Description automatically generated

Chương trình setup():

|  |
| --- |
| // put your setup code here, to run once:  pinMode(out1, OUTPUT);  digitalWrite(out1, LOW);  pinMode(out2, OUTPUT);  digitalWrite(out2, LOW);  pinMode(out3, OUTPUT);  digitalWrite(out3, LOW);  pinMode(out4, OUTPUT);  digitalWrite(out4, LOW);  Serial.begin(115200);  pinMode(ledConfig, OUTPUT);  digitalWrite(ledConfig, HIGH);  pinMode(bt1, INPUT);  attachInterrupt(digitalPinToInterrupt(bt1), isrPressed, RISING);  pinMode(bt2, INPUT);  attachInterrupt(digitalPinToInterrupt(bt2), isrPressed, RISING);  pinMode(bt3, INPUT);  attachInterrupt(digitalPinToInterrupt(bt3), isrPressed, RISING);  pinMode(bt4, INPUT);  attachInterrupt(digitalPinToInterrupt(bt4), isrPressed, RISING);  EEPROM.begin(512);  char check = char(EEPROM.read(0));  #ifdef DEBUG  Serial.print("check = ");  Serial.print(check);  #endif  EEPROM.write(0, 0);  EEPROM.commit();  EEPROM.end();  if(check == 'y')  {  digitalWrite(ledConfig, LOW);  WiFi.disconnect(true);  WiFiManager wifiManager;  wifiManager.setConfigPortalTimeout(180);  wifiManager.setCaptivePortalEnable(false);  wifiManager.setBreakAfterConfig(true);  wifiManager.setConnectRetries(5);  wifiManager.startConfigPortal("DATN-NTA-20173616");  ESP.restart();  }  else  {  setupWifi(10);  }  digitalWrite(ledConfig, HIGH);  client.setServer(mqttServer, 1883);  client.setCallback(callback);  reconnect(2);  } |

* Hàm xử lý ngắt nút nhấn

Hình 14. Lưu đồ thuật toán hàm ngắt nút nhấn

Diagram

Description automatically generated

Chương trình xử lý ngắt nút nhấn:

|  |
| --- |
| ICACHE\_RAM\_ATTR void isrPressed(void){  noInterrupts();  delayMicroseconds(1000);  if(digitalRead(bt1) == 1)  {  outState[0] = !outState[0];  switchTime[0] = 0;  buttonFlag = 1;  }  if(digitalRead(bt2) == 1)  {  outState[1] = !outState[1];  switchTime[0] = 0;  buttonFlag = 2;  }  if(digitalRead(bt3) == 1)  {  outState[2] = !outState[2];  switchTime[2] = 0;  buttonFlag = 3;  }  if(digitalRead(bt4) == 1)  {  outState[3] = !outState[3];  switchTime[3] = 0;  buttonFlag = 4;  }  changeOutput();  longPressTime = millis();  checkLongPress = true;  interrupts();  } |

* Hàm kiểm tra nút nhấn giữ: Hàm làm nhiệm vụ xác định trạng thái của nút nhấn bằng cách kiểm tra xem nút nhấn đã được người dùng nhả tay mỗi chu kì lặp của chương trình trong 10 giây. Nếu nút nhấn được giữ 10 giây sẽ đặt cờ để chương trình xử lý.

Hình 15. Lưu đồ thuật toán hàm kiểm tra nút nhấn giữ

Diagram

Description automatically generated

Chương trình kiểm tra nút nhấn giữ:

|  |
| --- |
| if(checkLongPress == true)  {  #ifdef DEBUG  Serial.print("checkLongPress");  #endif  switch(buttonFlag)  {  case 1:  if(digitalRead(bt1) == 0) checkLongPress = false;  break;  case 2:  if(digitalRead(bt2) == 0) checkLongPress = false;  break;  case 3:  if(digitalRead(bt3) == 0) checkLongPress = false;  break;  case 4:  if(digitalRead(bt4) == 0) checkLongPress = false;  break;  default:  break;  }  if(millis() - longPressTime > \_10sec)  {  longPressed = true;  }  }  if(longPressed == true)  {  longPress();  } |

* Hàm xử lý nhấn giữ: Khi người sử dụng thực hiện thao tác nhấn giữ. Chương trình sẽ mở bộ nhớ không bị mất đi khi mất điện, ghi kí tự ‘y’ vào để thông báo là lần khởi động sau thiết bị sẽ ở chế độ cấu hình WiFi và khởi động lại vi điều khiển.

Hình 16. Lưu đồ thuật toán hàm xử lý nút nhấn giữ

Diagram

Description automatically generated

Chương trình xứ lý nhấn giữ:

|  |
| --- |
| void longPress(void)  {  longPressed = false;  checkLongPress = false;  EEPROM.begin(512);  EEPROM.write(0, 'y');  EEPROM.commit();  EEPROM.end();  ESP.restart();  } |

* Hàm hẹn giờ đóng cắt: Khi người sử dụng chọn đóng cắt theo thời gian hàm này sẽ liên tục theo dõi thời gian hoạt động của từng kênh và ra lệnh đóng cắt khi thời điểm hẹn giờ đến.

Hình 17. Lưu đồ thuật toán hàm hẹn giờ đóng cắt

Diagram

Description automatically generated

Chương trình đóng cắt theo hẹn giờ:

|  |
| --- |
| void switchTimming(void)  {  for(int i = 0; i < 4; i++)  {  if(switchTime[i] != 0)  {  if(millis() - switchTimeRef[i] > switchTime[i]\*1000)  {  switchTime[i] = 0;  outState[i] = 0;  changeOutput();  }  }  }  } |

* Hàm gửi dữ liệu trạng thái đóng cắt: Hàm đọc trạng thái của các kênh đóng cắt, xây dựng bản tin trạng thái và đưa bản tin trạng thái tới server MQTT.

Hình 18. Lưu đồ nguyên lý hàm gửi dữ liệu trạng thái đóng cắt

Diagram

Description automatically generated

Chương trình gửi dữ liệu trạng thái đóng cắt:

|  |
| --- |
| void updateState(void)  {  String mqttPubString;  mqttPubString = "/1/" + String(outState[0]);  mqttPubString += "/2/" + String(outState[1]);  mqttPubString += "/3/" + String(outState[2]);  mqttPubString += "/4/" + String(outState[3]);  client.publish("status/datnta",(char\*)mqttPubString.c\_str());  } |

* Hàm xử lý truyền thông MQTT: Hàm làm nhiệm vụ xử lý các bản tin điều khiển gửi đến thiết bị. Đầu tiên hàm sẽ kiểm tra kết nối tới server, sau đó kiểm tra bộ nhớ đệm xem có bản tin không. Nếu có bản tin thì kiểm tra định dạng bản tin điều khiển. Thực hiện các chức năng đóng cắt và lưu thời gian đóng cắt theo nội dung bản tin điều khiển. Gửi bản tin phản hồi về trạng thái đầu ra hiện tại.

Hình 19. Lưu đồ thuật toán hàm xử lý truyền thông MQTT

Diagram

Description automatically generated

Chương trình xử lý truyền thông MQTT:

|  |
| --- |
| boolean PubSubClient::loop() {  if (connected()) {  unsigned long t = millis();  if ((t - lastInActivity > this->keepAlive\*1000UL) || (t - lastOutActivity > this->keepAlive\*1000UL)) {  if (pingOutstanding) {  this->\_state = MQTT\_CONNECTION\_TIMEOUT;  \_client->stop();  return false;  } else {  this->buffer[0] = MQTTPINGREQ;  this->buffer[1] = 0;  \_client->write(this->buffer,2);  lastOutActivity = t;  lastInActivity = t;  pingOutstanding = true;  }  }  if (\_client->available()) {  uint8\_t llen;  uint16\_t len = readPacket(&llen);  uint16\_t msgId = 0;  uint8\_t \*payload;  if (len > 0) {  lastInActivity = t;  uint8\_t type = this->buffer[0]&0xF0;  if (type == MQTTPUBLISH) {  if (callback) {  uint16\_t tl = (this->buffer[llen+1]<<8)+this->buffer[llen+2]; /\* topic length in bytes \*/  memmove(this->buffer+llen+2,this->buffer+llen+3,tl); /\* move topic inside buffer 1 byte to front \*/  this->buffer[llen+2+tl] = 0; /\* end the topic as a 'C' string with \x00 \*/  char \*topic = (char\*) this->buffer+llen+2;  // msgId only present for QOS>0  if ((this->buffer[0]&0x06) == MQTTQOS1) {  msgId = (this->buffer[llen+3+tl]<<8)+this->buffer[llen+3+tl+1];  payload = this->buffer+llen+3+tl+2;  callback(topic,payload,len-llen-3-tl-2);  this->buffer[0] = MQTTPUBACK;  this->buffer[1] = 2;  this->buffer[2] = (msgId >> 8);  this->buffer[3] = (msgId & 0xFF);  \_client->write(this->buffer,4);  lastOutActivity = t;  } else {  payload = this->buffer+llen+3+tl;  callback(topic,payload,len-llen-3-tl);  }  }  } else if (type == MQTTPINGREQ) {  this->buffer[0] = MQTTPINGRESP;  this->buffer[1] = 0;  \_client->write(this->buffer,2);  } else if (type == MQTTPINGRESP) {  pingOutstanding = false;  }  } else if (!connected()) {  // readPacket has closed the connection  return false;  }  }  return true;  }  return false;  }  void callback(char\* topic, byte\* payload, unsigned int length)  {  char timeBuffer[2];  int timeTemp;  timeBuffer[0] = (char)payload[5];  timeBuffer[1] = (char)payload[6];  timeTemp = atoi(timeBuffer);  if((char)payload[0] == '/')  {  if((char)payload[1] == '1')  {  switchTime[0] = timeTemp;  switchTimeRef[0] = millis();  if((char)payload[3] == '1')  {  outState[0] = 1;  }  else if(switchTime[0] == 0)  {  outState[0] = 0;  }  }  else if ((char)payload[1] == '2')  {  switchTime[1] = timeTemp;  switchTimeRef[1] = millis();  if((char)payload[3] == '1')  {  outState[1] = 1;  }  else if(switchTime[1] == 0)  {  outState[1] = 0;  }  }  else if ((char)payload[1] == '3')  {  switchTime[2] = timeTemp;  switchTimeRef[2] = millis();  if((char)payload[3] == '1')  {  outState[2] = 1;  }  else if(switchTime[2] == 0)  {  outState[2] = 0;  }  }  else if ((char)payload[1] == '4')  {  switchTime[3] = timeTemp;  switchTimeRef[3] = millis();  if((char)payload[3] == '1')  {  outState[3] = 1;  }  else if(switchTime[3] == 0)  {  outState[3] = 0;  }  }  changeOutput();  updateState();  }  } |

* Hàm cấu hình WiFi

# Chương 4. Phát triển ứng dụng điều khiển

## Giới thiệu ngôn ngữ Python và thư viện Kivy

### Ngôn ngữ Python

Python là một ngôn ngữ lập trình thông dịch, hướng đối tượng bậc cao ngữ nghĩa động. Python hỗ trợ các thư viện dưới dạng các mô-dun và gói. Python được thiết kế với ưu điểm mạnh là dễ đọc, dễ học và dễ nhớ. Python là ngôn ngữ có hình thức rất sáng sủa, cấu trúc rõ ràng. Cấu trúc của Python cho phép người sử dụng viết mã lệnh với số lần gõ phím tối thiểu. Một số đặc điểm chính của Python:

* Ngữ pháp đơn giản, dễ đọc
* Vừa hướng thủ tục, vừa hướng đối tượng
* Hỗ trợ mô-đun và hỗ trợ gói
* Xử lý lỗi bằng ngoại lệ
* Kiểu dữ liệu động ở mức cao
* Có các bộ thư viện chuẩn và các mô-đun ngoài, đáp ứng tất cả các nhu cầu lập trình
* Có khả năng tương tác với các mô-đun khác viết trên C/C++
* Có thể nhúng vào ứng dụng như một giao tiếp kịch bản

Nhờ những đặc điểm như dễ đọc, dễ học và có cộng đồng hỗ trợ lớn, ngôn ngữ Python ngày càng trở nên phổ biến trong lĩnh vực lập trình. Những chương trình viết bằng ngôn ngữ Python cũng cho phép chạy trên nhiều nền tảng hệ điều hành khác nhau nên được ưu tiên lựa chọn để thiết kế ứng dụng điều khiển trong đồ án này.

### Thư viện Kivy cho Python

Kivy là một mô-đun thư viện cho Python có mã nguồn mở hỗ trợ phát triển ứng dụng có giao diện. Kivy hỗ trợ thiết kế chương trình đa nền tảng. Một chương trình sử dụng mô-đun Kivy có thể chạy trên Linux, Windows, OS X, Android và iOS. Kivy được viết bằng Python, do đó nó rất nhanh và có độ chính xác cao. Ngoài Pyhton thì Kivy cũng tự xây dựng ngôn ngữ của riêng mình là ngôn ngữ Kivy, có thể tìm thấy ở các tệp có đuôi .kv. Với ngôn ngữ Kivy thì chúng ta có thể tạo giao diện người dùng với số ít mã lệnh, tách riêng phần giao diện với phần logic tính toán chình của chương trình.

Đồ án này thiết kế ứng dụng với mô-đun Kivy do tính đa nền tảng của mô-đun này. Chương trình ứng dụng điều khiển có thể được sử dụng trên máy tính xách tay hay điện thoại thông minh với kết nối internet.

## Lưu đồ thuật toán và chương trình cho ứng dụng điều khiển

### Chương trình logic chính

Chương trình logic chính của ứng dụng điều khiển được thiết kế bằng ngôn ngữ Python với 4 đối tượng gồm lớp logic chính của ứng dụng “DATNApp”, lớp giao diện dạng lưới “MyGrid” và 2 lớp cửa sổ thông báo “PopupTurnOn” và “PopupTurnOff” được xây dựng trên các lớp cha:

* Lớp cha ứng dụng: App
* Lớp cha làm khung giao diện: Widget
* Lớp cha giao diện lưới: GridLayout
* Lớp cha giao diện thả nổi: Floatlayout
* Lớp logic chính của ứng dụng: Lớp logic chính của ứng dụng “DATNApp” được xây dựng trên lớp cha “App” với các hàm riêng để xử lý truyền thông MQTT: hàm “onConnect” được gọi khi thiết bị kết nối thành công với server; hàm “onDisconnect” được gọi khi thiết bị mất kết nối với server; hàm “onMessage” được gọi khi có bản tin mới gửi đến thiết bị; hàm “sendCmd” được gọi khi thiết bị gửi bản tin điều khiển; hàm “mqttReconnect” được gọi để thiết bị kết nối lại với server MQTT.
* Hàm “onConnect” đặt cờ thông báo thiết bị đã kết nối với server MQTT và đăng kí (subscribe) các chủ để (topic) trạng thái của thiết bị trường.

Hình 20. Lưu đồ thuật toán hàm "onConnect"

Diagram

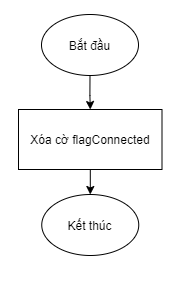
Description automatically generated

Chương trình của hàm “onConnect”

|  |
| --- |
| def onConnect(self, client, userdata, flags, rc):  global flagConnected  flagConnected = True  for topic in topicSubscribe:  client.subscribe(topic) |

* Hàm “onDisconnect” xóa cờ đã kết nối của thiết bị để tiến hành kết nối lại với server

Hình 21. Lưu đồ nguyên lí hàm "onDisconnect"



Chương trình của hàm “onDisconnect”

|  |
| --- |
| def onDisconnect(self, client, userdata, rc):  global flagConnected  flagConnected = False |

* Hàm “onMessage” định dạng lại bản tin MQTT cho phù hợp và truyền vào hàm “update” của lớp giao diện lưới để thay đổi thông tin hiển thị.

Diagram

Description automatically generated

Chương trình của hàm “onMessage”

|  |
| --- |
| def onMessage(self, client, userdata, message):  data = message.payload.decode("utf-8")  if data[0] == '/':  self.root.update(data) |

* Hàm “sendCmd” xây dựng bản tin điều khiển và tiến hành gửi bản tin tới server MQTT

Diagram

Description automatically generated

Chương trình của hàm “sendCmd”

|  |
| --- |
| def sendCmd(self, switchNumber, switchType, switchTime):  strTemp = "/" + str(switchNumber) + "/"  if buttonState[int(switchNumber) - 1] == 0:  strTemp += '1'  else:  strTemp += '0'  strTemp += '/' + str(switchTime)  self.client.publish(topicPublish, strTemp) |

* Hàm “mqttReconnect” được gọi mỗi 30 giây khi ứng dụng đang hoạt động để tiến hành kết nối lại với server nếu mất kết nối.

Chương trình của hàm “mqttReconnect”

|  |
| --- |
| def mqttReconnect(self):  global flagConnected  if flagConnected == False:  try:  self.client.connect(mqttServer, mqttPort, 60)  self.client.loop\_start()  except:  pass |

* Lớp giao diện dạng lưới: Lớp giao diện dạng lưới “MyGrid” của ứng dụng được xây dựng trên lớp cha “Widget” với các hàm riêng để cập nhật và xử lý thao tác từ người dùng: hàm “update” được gọi để thay đổi trạng thái các nút nhấn trên màn hình hiển thị; hàm “showPopup” được gọi để hiển thị cửa sổ thông báo khi người dùng nhấn nút; hàm “buttonConfigWifi” được gọi khi người dúng bấm vào nút Cấu hình WiFi.
* Hàm “update” nhận dữ liệu từ bản tin MQTT và thay đổi giao diện các nút nhấn tương ứng

Diagram

Description automatically generated

Chương trình của hàm “update”

|  |
| --- |
| def update(self, data):  buttonUpdateState = [data[3], data[7], data[11], data[15]]  if buttonUpdateState[0] == '1':  self.button1.text = "Cong tac 1 Bat"  buttonState[0] = 1  elif buttonUpdateState[0] == '0':  self.button1.text = "Cong tac 1 Tat"  buttonState[0] = 0  if buttonUpdateState[1] == '1':  self.button2.text = "Cong tac 2 Bat"  buttonState[1] = 1  elif buttonUpdateState[1] == '0':  self.button2.text = "Cong tac 2 Tat"  buttonState[1] = 0  if buttonUpdateState[2] == '1':  self.button3.text = "Cong tac 3 Bat"  buttonState[2] = 1  elif buttonUpdateState[2] == '0':  self.button3.text = "Cong tac 3 Tat"  buttonState[2] = 0  if buttonUpdateState[3] == '1':  self.button4.text = "Cong tac 4 Bat"  buttonState[3] = 1  elif buttonUpdateState[3] == '0':  self.button4.text = "Cong tac 4 Tat"  buttonState[3] = 0 |

* Hàm “showPopup” hiển thị một cửa sổ thông báo tương ứng với nút người dùng vừa nhấn.

Diagram

Description automatically generated

Chương trình của hàm “showPopup”

|  |
| --- |
| def showPopup(self, switchNumber):  global \_switchNumber, popupWindow  \_switchNumber = int(switchNumber)  if buttonState[\_switchNumber - 1] == 0:  show = PopupTurnOn()  popupWindow = Popup(title = "Bat cong tac", content = show,  size\_hint = (None, None), size = (500,500))  elif buttonState[\_switchNumber - 1] == 1:  show = PopupTurnOff()  popupWindow = Popup(title = "Tat cong tac", content = show,  size\_hint = (None, None), size = (500,500)) |

* Hàm “buttonConfigWifi” mở một trình duyệt cho phép người dùng cập nhật mạng WiFi cho thiết bị.

Diagram

Description automatically generated

Chương trình của hàm “buttonConfigWifi”

|  |
| --- |
| def buttonConfigWifi(self):  webbrowser.open('http://192.168.4.1/wifi?#p') |

* Lớp giao diện cửa sổ thông báo: Lớp giao diện cửa sổ thông báo được xây dựng trên lớp cha “FloatLayout” với các hàm riêng để xử lý thao tác từ người dùng: hàm “switch” được gọi để xử lý thao tác nhấn nút; hàm “close” được gọi khi người dùng chọn nút đóng cửa sổ.
* Hàm “switch” xử lý thao tác của người dùng và chuyển thành các biến tương ứng để truyền cho hàm gửi bản tin điều khiển.

Diagram

Description automatically generated

Chương trình của hàm “switch”

|  |
| --- |
| def switch(self, switchType, switchTime):  global \_switchNumber  DATNApp().sendCmd(\_switchNumber, switchType, switchTime)  popupWindow.dismiss() |

* Hàm “close”

Diagram

Description automatically generated

Chương trình của hàm “close”

|  |
| --- |
| def close(self):  popupWindow.dismiss() |

### Chương trình giao diện

Chương trình giao diện thiết kế bằng mô-đun Kivy được chia làm 2 phần gồm có phần giao diện hiển thị dạng lưới ở trạng thái bình thường và giao diện cửa sổ thông báo khi người dùng sử dụng các nút nhấn:

* Giao diện dạng lưới: Giao diện của ứng dụng được xây dựng trên mô hình dạng lưới. Cửa sổ làm việc được chia thành các ô nhỏ, mỗi ô được xây dựng với những chức năng khác nhau. Ô thông tin ở trên cùng được xây dựng bằng thuộc tính “Label”, chuyên dùng để viết văn bản và không cho phép người dùng thay đổi. Các ô nút nhấn được xây dựng bằng thuộc tính “Button”. Thuộc tính này cho phép người dùng tương tác với ứng dụng điều khiển với các thao tác như nhấn, nhấn giữ. Khi người dùng tương tác với nút nhấn, các hàm điều khiển sẽ được kích hoạt tương ứng. Ở trong đồ án này, khi người dùng nhả tay hoặc chuột điều khiển ra khỏi nút nhấn, hàm điều khiển trong mục “on\_release” của nút nhấn sẽ được gọi.

Mô hình giao diện dạng lưới có dạng như sau:

|  |  |
| --- | --- |
| Thông tin | |
| Nút nhấn 1 | Nút nhấn 2 |
| Nút nhấn 3 | Nút nhấn 4 |
| Nút cấu hình WiFi | |

Chương trình xây dựng giao diện dạng lưới:

|  |
| --- |
| <MyGrid>:  button1: button1  button2: button2  button3: button3  button4: button4  configWifi: configWifi  infoLabel1: infoLabel1  infoLabel2: infoLabel2  infoLabel3: infoLabel3  GridLayout:  cols: 1  size: root.width, root.height  Label:  id: infoLabel1  size\_hint: 1, 0.05  text: "Sinh vien thuc hien: NGUYEN TUAN ANH\_MSSV: 20173616"  Label:  id: infoLabel2  size\_hint: 1, 0.05  text: "Giang vien huong dan: TS. NGUYEN VAN ANH"  Label:  id: infoLabel3  size\_hint: 1, 0.05  text: "De tai: Thiet ke thiet bi dong cat 4 kenh qua WiFi"  GridLayout:  cols: 2  Button:  id: button1  text: "Cong tac 1"  font\_size: 50  on\_release: root.showPopup(1)  Button:  id: button2  text: "Cong tac 2"  font\_size: 50  on\_release: root.showPopup(2)  Button:  id: button3  text: "Cong tac 3"  font\_size: 50  on\_release: root.showPopup(3)  Button:  id: button4  text: "Cong tac 4"  font\_size: 50  on\_release: root.showPopup(4)  Button:  size\_hint: 1, 0.2  id: configWifi  text: "Cau hinh WiFi"  font\_size: 50  on\_release: root.buttonConfigWifi() |

* Giao diện cửa sổ thông báo: Giao diện cửa sổ thông báo của ứng dụng được xây dựng trên thuộc tính “Popup” của thư viện Kivy. Giao diện gồm trường thông tin và các trường nút nhấn tương tự như giao diện chính. Khi người dùng nhả tay trên màn hình cảm ứng hoặc nhả chuột trên máy tính, hàm tương ứng trường “on\_release” của chương trình sẽ được gọi.

Mô hình giao diện cửa sổ:

|  |
| --- |
| Thông tin |
| Nút nhấn 1 |
| Nút nhấn 2 |
| Nút nhấn 3 |

Chương trình giao diện cho cửa sổ thông báo:

|  |
| --- |
| <PopupTurnOn>:  Button:  text: "Bat"  size\_hint: 1, 0.25  pos\_hint: {"x":0, "y":0.75}  on\_release: root.switch(1, 00)  Button:  text: "Bat 30 giay"  size\_hint: 1, 0.25  pos\_hint: {"x":0, "y":0.5}  on\_release: root.switch(1, 30)  Button:  text: "Bat 1 phut"  size\_hint: 1, 0.25  pos\_hint: {"x":0, "y":0.25}  on\_release: root.switch(1, 60)  Button:  text: "Bo qua"  size\_hint: 1, 0.25  pos\_hint: {"x":0, "y":0}  on\_release: root.close()  <PopupTurnOff>:  Button:  text: "Tat"  size\_hint: 1, 0.25  pos\_hint: {"x":0, "y":0.75}  on\_release: root.switch(0, 00)  Button:  text: "Tat sau 30 giay"  size\_hint: 1, 0.25  pos\_hint: {"x":0, "y":0.5}  on\_release: root.switch(0, 30)  Button:  text: "Tat sau 1 phut"  size\_hint: 1, 0.25  pos\_hint: {"x":0, "y":0.25}  on\_release: root.switch(0, 60)  Button:  text: "Bo qua"  size\_hint: 1, 0.25  pos\_hint: {"x":0, "y":0}  on\_release: root.close() |

# Chương 5. Mô hình và kết quả đạt được

## 5.1. Kết quả thiết kế phần cứng thiết bị

## 5.2. Kết quả thiết kế ứng dụng điều khiển

Hình ảnh thực tế ứng dụng điều khiển trên điện thoại Android

Chart, treemap chart

Description automatically generated

Graphical user interface

Description automatically generated with medium confidence

Hình ảnh thực tế ứng dụng điều khiển trên máy tính xách tay

Graphical user interface, application

Description automatically generated

Graphical user interface

Description automatically generated

## 5.3. Kết luận

Trong đồ án này đã thiết kế một thiết bị có khả năng đóng cắt 4 kênh ra tại hiện trường và từ xa thông qua kết nối internet với công suất đóng cắt mỗi kênh là x, có khả năng tạo trễ cắt điện ở 4 kênh ra. Xây dựng được ứng dung điều khiển và hiển thị trạng thái của 4 kênh đóng cắt.

Bên cạnh đó còn một vài hạn chế về loại tải đầu ra, giao diện điều khiển thiết bị chưa thực sự thân thiện với người dùng.

## 5.4. Định hướng phát triển

Khi quyết định thực hiện đề tài này, tôi mong muốn sẽ triển khai được một hệ thống thông minh với số lượng các thiết bị lớn hơn và đa dạng thiết bị hơn không chỉ thực hiện công việc đóng cắt mà còn tích hợp các các thiết bị cảm biến để giám sát môi trường xung quanh, giúp cho các thiết bị có thể tương tác với nhau từ đó nâng cao mức độ tự động hóa cho hệ thống.

# Tài liệu tham khảo