

ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC BÁCH KHOA

KHOA ĐIỆN – ĐIỆN TỬ

BỘ MÔN THIẾT BỊ ĐIỆN

-----oo-----



LUẬN VĂN TỐT NGHIỆP

THIẾT KẾ MÔ HÌNH I-HOME

GVHD : TS. TRỊNH HOÀNG HƠN

SVTH : TRẦN XUÂN TUẤN ANH

MSSV : 41200117

TP. HỒ CHÍ MINH, THÁNG 12 NĂM 2016

LỜI CẢM ƠN

Trước hết xin cảm ơn gia đình là nguồn động viên rất lớn và là chỗ dựa tinh thần vững chắc trong suốt quá trình học tập và rèn luyện tại Đại học Bách Khoa TP,HCM

Em xin chân thành cảm ơn tất cả thầy trong khoa Điện – Điện Tử đã truyền đạt cho em nhiều kiến thức quan trọng và quý giá trong thời gian học tập vừa qua. Các thầy cô đã nhiệt tình giúp đỡ em trong quá trình học tập.

Em xin gửi lời cảm ơn chân thành đến thầy Trịnh Hoàng Hợi, thầy đã tận tình giúp đỡ em, luôn tạo mọi điều kiện thuận lợi để em hoàn thành luận văn này. Thầy đã cung cấp cho em nhiều kiến thức mới mẻ và quý báu góp phần quan trọng trong việc hoàn thành luận văn.

Cuối cùng xin cảm ơn các bạn bè trong trường đã động viên và giúp đỡ em trong suốt khóa học này.

TÓM TẮT LUẬN VĂN

Nhà thông minh không còn là một định nghĩa xa lạ trong thời đại hiện nay. Việc phát triển nhanh chóng của các công nghệ công nghệ thông minh như: Điện thoại thông minh, Đồng hồ thông minh, các vật dụng thông minh đã báo hiệu về một cuộc cách mạng số trong tương lai không xa. Năm bắt được điều này, em đã theo đuổi đề tài về nhà thông minh trong luận văn. Luận văn về thiết kế và thi công mô hình nhà thông minh với một số tính năng đã hiện hữu hiện nay trên các sản phẩm nhà thông minh của các công ty trên thị trường.

Chương 1: Khái quát định nghĩa về nhà thông minh, các chức năng nhà thông minh.

Chương 2: Cơ sở lý thuyết về chuẩn truyền RS485 và vi điều khiển.

Chương 3: Thiết kế thi công và thực hiện phần cứng cho mô hình nhà thông minh.

Chương 4: Giải thuật sử dụng trong mô hình, ứng dụng các phần mềm để thiết kế giao diện điều khiển cho mô hình.

Chương 5: Kết quả thực hiện mô hình và hình ảnh thực tế của mô hình.

Chương 6: Kết luận và hướng phát triển của đề tài.

MỤC LỤC

TRANG BÀI	i
LỜI CẢM ƠN	ii
TÓM TẮT LUẬN VĂN	iii
MỤC LỤC	iv
DANH MỤC HÌNH VẼ	vii
DANH MỤC BẢNG	x
DANH MỤC TỪ VIẾT TẮT	xi
Chương 1. GIỚI THIỆU	1
1.1. TỔNG QUAN VỀ I-HOME:	1
1.1.1. Giới thiệu	1
1.1.2. Các tính năng	2
1.2. TÌNH HÌNH NGHIÊN CỨU TRONG VÀ NGOÀI NƯỚC	6
1.2.1. Thế giới	6
1.2.2. Trong nước	12
1.3. NHIỆM VỤ LUẬN VĂN	13
Chương 2. CƠ SỞ LÝ THUYẾT	14
2.1. CHUẨN GIAO TIẾP RS485:	14
2.1.1. Giới thiệu	14
2.1.2. Đặc tính kỹ thuật	14
2.2. VI ĐIỀU KHIỂN:	20
2.2.1. Lý thuyết vi điều khiển	20
Chương 3. THIẾT KẾ VÀ THỰC HIỆN PHẦN CỨNG	22

3.1. SƠ ĐỒ KẾT NỐI:	22
3.2. SƠ ĐỒ NGUYÊN LÝ:	22
3.3. VI ĐIỀU KHIỂN:	28
3.3.1. Vi điều khiển MSP430	28
3.3.2. ARM Cortex M4	29
3.3.3. Dòng AVR	31
3.4. MODULE WIFI:	33
3.5. MODULE BLUETOOTH HC-05:	35
3.6. IC RS485 – SN75176B:	37
3.7. MODULE THỜI GIAN THỰC	38
3.8. CẢM BIẾN CUỜNG ĐỘ ÁNH SÁNG	41
3.9. CẢM BIẾN NHIỆT ĐỘ	42
3.10. IC NGUỒN TUYẾN TÍNH	43
3.10.1. 7805	43
3.10.2. LM1117	44
3.11. RELAY	44
3.12. CÔNG TẮC BÁN DÂN	45
3.13. OPTO PC817	46
Chương 4. GIẢI THUẬT VÀ XÂY DỰNG PHẦN MỀM	48
4.1. THIẾT KẾ I-HOME	48
4.1.1. Tính năng kỹ thuật	48
4.1.2. Sơ đồ khói	49
4.1.3. Giao thức truyền thông	52
4.2. THIẾT KẾ GIAO DIỆN ĐIỀU KHIỂN	58

4.2.1. Xây dựng ứng dụng cho smartphone dùng giao diện Blue	58
4.2.2. Xây dựng ứng dụng cho PC dùng Visual Studio	63
Chương 5. KẾT QUẢ THỰC HIỆN	66
 5.1. TỔNG THỂ MÔ HÌNH:.....	66
 5.2. KHỐI MASTER:.....	67
 5.3. KHỐI SLAVE:.....	69
5.3.1. Slave đa chức năng (#1).....	69
5.3.2. Slave Input Button (#2)	71
5.3.3. Slave Output Relay (#3)	73
 5.4. KHỐI CẢM BIÊN VÀ HIỂN THỊ:.....	75
 5.5. KHỐI DIMMER:.....	76
 5.6. KHỐI MSP-ESP8266:	78
Chương 6. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN	80
 6.1. Kết luận	80
6.1.1. Kết quả đạt được	80
6.1.2. Hạn chế	80
 6.2. Hướng phát triển	81
TÀI LIỆU THAM KHẢO	82

DANH MỤC HÌNH VẼ

Hình 1.1 Sơ đồ tổng thể một hệ thống nhà thông minh	1
Hình 1.2 Sơ đồ hệ thống giám sát bằng camera	2
Hình 1.3 Hệ thống đèn ngoài trời bật tắt dựa theo ánh sáng tự nhiên	3
Hình 1.4 Thiết bị cảm biến chuyển động	4
Hình 1.5 Giao diện điều khiển nhà thông minh của BKAV	5
Hình 1.6 Các hãng công nghệ tên tuổi tham gia thị trường Smart-home	7
Hình 1.7 Nest có thiết kế và tính năng ưu việt so với hệ thống điều hòa cũ	8
Hình 1.8 Echo có giao diện tương tác bằng giọng nói	9
Hình 1.9 Samsung mua SmartThings để đặt chân vào thị trường nhà thông minh	10
Hình 1.10 Apple dùng iOS làm bàn đạp để vào thị trường nhà thông minh	11
Hình 1.11 Hai thành tố chính cho hệ sinh thái nhà thông minh: smartphone và smart home hub	12
Hình 2.1 Các thông số đặc tính cơ bản của chuẩn RS485	15
Hình 2.2 Dạng tín hiệu trên 2 đường truyền RS485	15
Hình 2.3 Tương quan giữa tốc độ truyền và chiều dài đường dây	16
Hình 2.4 Phân cực fail-safe trên đường truyền đa trạm chuẩn RS485	17
Hình 2.5 Vị trí đặt điện trở đầu cuối trên đường truyền RS485	18
Hình 2.6 Dạng sóng ngõ ra trên dây A tương ứng với 2 giá trị điện trở đầu cuối	18
Hình 2.7 Cấu trúc sơ đồ 4 dây cho chế độ full-duplex	19
Hình 2.8 Cách đấu dây thực tế của mạng RS485 sử dụng cặp dây xoắn	19
Hình 2.9 Cáp xoắn đôi 24AWG có bọc chống nhiễu (trái)	19
Hình 3.1 Sơ đồ kết nối phần cứng	22
Hình 3.2 Sơ đồ mạch nguồn	22
Hình 3.3 Sơ đồ vi điều khiển	23
Hình 3.4 Sơ đồ giao tiếp RS485	23
Hình 3.5 Sơ đồ mạch nguồn	23
Hình 3.6 Sơ đồ vi điều khiển output	24
Hình 3.7 Sơ đồ giao tiếp RS485	24
Hình 3.8 Sơ đồ mạch relay	24

Hình 3.9 Sơ đồ mạch nguồn.....	25
Hình 3.10 Sơ đồ vi điều khiển TIVA	25
Hình 3.11 Sơ đồ giao tiếp RS485.....	25
Hình 3.12 Sơ đồ mạch nguồn chính.....	26
Hình 3.13 Sơ đồ vi điều khiển.....	26
Hình 3.14 Sơ đồ kết nối module Wifi ESP8266	26
Hình 3.15 Sơ đồ mạch Dimmer LED	27
Hình 3.16 Sơ đồ mạch kết nối với Adruino.....	27
Hình 3.17 Sơ đồ chân Chip MSP430	29
Hình 3.18 Các dòng vi xử lý, DSP của hãng Texas Instrument	30
Hình 3.19 Sơ đồ các khói chức năng của chip TM4C123GHPM.....	31
Hình 3.20 Sơ đồ chân module Wifi ESP8266	34
Hình 3.21 Module Bluetooth HC05	35
Hình 3.22 Bảng trạng thái ngõ vào ra các bộ thu phát của SN75176B	37
Hình 3.23 Sơ đồ cấu trúc logic SN75176B	37
Hình 3.24 Bố trí các chân của SN75176B	38
Hình 3.25 Module thời gian thực DS1307.....	38
Hình 3.26 Hai gói cấu tạo chip DS1307.....	39
Hình 3.27 Schematic cảm biến cường độ ánh sáng BH1750	42
Hình 3.28 Cảm biến nhiệt độ DTH11	43
Hình 3.29 Hình ảnh relay	44
Hình 3.30 Sơ đồ chân và cấu tạo TIP122	46
Hình 3.31 Sơ đồ chân Opto PC817	47
Hình 4.1 Phòng khách thông minh tiện dụng	48
Hình 4.2 Sơ đồ khói chức năng nhà thông minh.....	49
Hình 4.3 Lưu đồ trạng thái Master.....	57
Hình 4.4 Lưu đồ trạng thái Slave	57
Hình 4.5 Lập trình trò chơi "Bắt chuột chui" với App Inventor.	59
Hình 4.6 Giao diện Kết nối với thiết bị Bluetooth khác.....	61
Hình 4.7 Giao diện nhận diện và xử lý tín hiệu âm thanh từ người sử dụng	61
Hình 4.8 Giao diện điều khiển thông qua các Button.....	61
Hình 4.9 Khối code kết nối với thiết bị Bluetooth khác	62

Hình 4.10 Khối code nhận diện và xử lý tín hiệu âm thanh từ người sử dụng	62
Hình 4.11 Khối code điều khiển thông qua các Button.	63
Hình 4.12 Giao diện kết nối đến địa chỉ IP	64
Hình 4.13 Giao diện send text đến Server	65
Hình 4.14 Giao diện điều khiển thông qua các Button.	65
Hình 5.1 Tổng thể mô hình	66
Hình 5.2 Mạch master hoàn chỉnh.....	67
Hình 5.3 Layout Master	68
Hình 5.4 Mạch Slave đa chức năng hoàn chỉnh	69
Hình 5.5 Layout Slave đa chức năng	70
Hình 5.6 Mạch Slave Input Button hoàn chỉnh.....	71
Hình 5.7 Layout Slave Input Button.....	72
Hình 5.8 Slave Output Relay hoàn chỉnh.....	73
Hình 5.9 Layout Output Relay	74
Hình 5.10 Mạch cảm biến và hiển thị hoàn chỉnh	75
Hình 5.11 Layout cảm biến và hiển thị	75
Hình 5.12 Mạch dimmer hoàn chỉnh	76
Hình 5.13 Layout dimmer	77
Hình 5.14 Mạch MSP-ESP8266 hoàn chỉnh	78
Hình 5.15 Layout MSP-ESP8266	79

DANH MỤC BẢNG

Bảng 1 Dòng tiêu thụ ở các chế độ hoạt động của vi điều khiển Cortex M4	30
Bảng 2 Thông số vi điều khiển ATmega328	33
Bảng 3 Bảng mô tả thanh ghi của module RTC	40

DANH MỤC TỪ VIẾT TẮT

ARM (Advanced RISC Machine) : Cấu trúc vi xử lý tân tiến được phát triển bởi RISC

CPU (Central Processing Unit) : Bộ xử lý trung tâm.

ADC (Analog to digital converter): là bộ chuyển đổi tín hiệu tương tự sang tín hiệu số

DAC (Digital to analog converter): Có chức năng ngược lại với ADC. DAC thường được sử dụng để giám sát các thiết bị tương tự.

TI (Texas Instruments): Hãng sản xuất linh kiện điện tử của Mỹ

I2C (Inter-Integrated Circuit) : một chuẩn giao tiếp được phát minh bởi Philips' semiconductor division (giờ là NXP) nhằm đơn giản hóa việc trao đổi dữ liệu giữa các Ics.

CSMA/CD (Carrier Sense Multiple Access with Collision Detect) : đa truy cập nhận biết sóng mang phát hiện xung đột.

VS (Visual Studio) Phần mềm môi trường sử dụng để xây dựng ứng dụng.

RISC (Reduced Instructions Set Computer) - Máy tính với tập lệnh đơn giản hóa

Chương 1. GIỚI THIỆU

1.1. TỔNG QUAN VỀ I-HOME:

1.1.1. Giới thiệu

Thuật ngữ “Intelligent Home” đã xuất hiện cách đây khá lâu, thế nhưng đến gần đây, nhà thông minh mới thật sự xuất hiện ở Việt Nam do giá thành còn tương đối khó chấp nhận bởi số đông người tiêu dùng có thu nhập hạn chế. Tuy nhiên với sự phát triển của công nghệ, nhà thông minh ngày càng dễ tiếp cận được và không còn là từ xa xỉ khi nhắc đến. Thời đại công nghệ đã mang cho chúng ta rất nhiều tiện ích mà thậm chí cách đây vài thập niên còn là “viễn tưởng”! Với khởi đầu chỉ là những chiếc remote điều khiển các bóng đèn, mở tivi... thì ngày nay, chúng ta thậm chí không cần cử động vẫn có thể làm được điều đó, và thậm chí còn hơn thế nữa. Các kiểu mẫu nhà thông minh hiện đại được trang bị đầy đủ những công nghệ tiên tiến như những hệ thống giải trí kỹ thuật số đa phương tiện, hệ thống ánh sáng tự động nhận biết sự có mặt của con người, những chiếc rèm cửa tự động thu lại khi trời sáng, trả lời chuông cửa mặt dù vẫn đang ở nơi làm việc qua hệ thống giám sát bằng camera, và đặc biệt là hệ thống an ninh tối tân vẫn bảo vệ chặt chẽ ngôi nhà khỏi những sự cố khi chúng ta vắng mặt.



Hình 1.1 Sơ đồ tổng thể một hệ thống nhà thông minh

Chương 1: Giới thiệu

1.1.2. Các tính năng

1.1.2.1. An ninh

Như đã đề cập, với mạng lưới internet băng thông rộng phủ khắp hiện nay, cùng với hệ thống camera hiện đại giúp chúng ta có thể quan sát mọi hoạt động trong và ngoài nhà mình từ bất kỳ nơi nào chỉ với vài cú click chuột. Hơn thế nữa, hệ thống an ninh còn có thể cảnh báo cho chúng ta biết khi có kẻ gian qua điện thoại di động hay email và phát tín hiệu để xua đuổi, báo động cho các hộ xung quanh, và thậm chí khi các thiết bị đó không hoạt động thì các khoá điện tử và cảm biến gắn trong ngôi nhà vẫn là rào cản vững chắc với những kẻ có ý đồ xấu.

Hệ thống an ninh thường bao gồm 3 bộ phận, hoạt động độc lập: một bộ điều khiển trung tâm (kiểm soát trung tâm); các thiết bị cảm biến ngoại vi khác (các sensor) như cảm biến báo nhiệt, báo khói, báo trộm, cảm biến chuyển động; các thiết bị báo hiệu như loa, còi hú, đèn chớp, điện thoại...



Hình 1.2 Sơ đồ hệ thống giám sát bằng camera

Hình 1.2 minh họa cho một hệ thống giám sát hiện đại, có khả năng ghi lại mọi hoạt động xảy ra đồng thời có thể quan sát từ xa qua máy tính xách tay hoặc máy tính để bàn có truy cập internet. Các thiết bị quan sát còn giúp người chủ có thể tiếp khách qua camera ngoài cửa khi không có mặt tại nhà.

Cùng với kỹ thuật xử lý ảnh hiện đại và các camera độ phân giải cao, hệ thống an ninh còn có thể nhận dạng chủ nhà và gia đình qua gương mặt, giúp giảm thiểu

các rắc rối và sự cố ngoài ý muốn như thất lạc chìa khoá với các khoá truyền thống, quên mật mã đối với các khoá điện tử ...

Báo cháy báo khói : Một bộ điều khiển trung tâm có thể quản lý nhiều thiết bị cảm biến khói và nhiệt sẽ kích hoạt tín hiệu báo cháy bằng còi kêu tại chỗ và xử lý theo một số tính năng được thiết lập sẵn như bật sáng toàn bộ đèn trong, ngoài nhà, gửi cảnh báo đến người chủ qua điện thoại, email... sau khoảng thời gian định trước, nếu không có phản ứng của người sử dụng, hệ thống sẽ tự động ngắt điện toàn nhà tránh việc cháy lan truyền, phun nước dập lửa.

1.1.2.2. Chiếu sáng

Các mẫu nhà thông minh hiện nay đều đơn giản tối đa việc điều khiển bằng tay hệ thống chiếu sáng, thay vào đó là các thiết bị tự động bật tắt đèn khi nhận thấy sự hiện diện của con người, các tính năng này được mở rộng hơn với các mô hình được lịch trình sẵn ứng với thói quen của chủ nhà theo các mốc thời gian trong ngày.

- Tính năng theo lịch trình : các đèn ngoài sân và mặt trước và xung quanh nhà sẽ tự động bật sáng khi trời暗 tối (khoảng thời gian 6h chiều), tắt khi cả nhà đi ngủ (10h), đèn trong nhà sẽ bật khi chủ nhà đi làm về (6h30)...



Hình 1.3 Hệ thống đèn ngoài trời bật tắt dựa theo ánh sáng tự nhiên

- Tính năng cảm biến chuyển động : khi có người bước vào sân, các nhóm đèn ở đây sẽ tự động bật sáng để chiếu sáng lối đi. Khi khách vào nhà từ tiền sảnh, các đèn lầu trệt sẽ tự động tăng dần độ sáng đến 30%, tạo ánh sáng dịu mắt và đảm

bảo đủ sáng cho việc đi lại trong nhà. Sự tăng dần của ánh sáng tạo nên một hiệu ứng đặc biệt như chào đón khách vào nhà. Đèn tự động tắt sau một phút ngay khi không có người trong các khu vực này. Chức năng này được thực hiện bởi các cảm biến chuyển động ngoài trời và trong nhà có tích hợp chức năng đo độ sáng được bố trí khoa học.



Hình 1.4 *Thiết bị cảm biến chuyển động*

- Tính năng điều khiển theo hoạt cảnh : Với tính năng này, người chủ có thể thiết lập các chế độ định sẵn như một kịch bản ứng với mỗi hoàn cảnh cụ thể, giúp tiết kiệm thời gian chỉnh định, mang lại hiệu quả sử dụng cao nhất và tiết kiệm nhất cho chủ nhà. Các kịch bản này ta có thể thay đổi tùy theo sở thích sinh hoạt của từng gia đình. Ví dụ, khi bấm phím “tiếp khách”, đèn chùm sẽ tăng dần độ sáng, nhóm đèn hành lang tăng dần độ sáng đến 60%, các nhóm đèn phụ trợ cần thiết sẽ bật sáng. Tất cả các hoạt động này được thực hiện cùng một lúc với chỉ một phím bấm. Tương tự như vậy, các phím bấm “dạ tiệc”, “ăn tối” hay “bình thường” sẽ điều chỉnh các thiết bị điện về các chế độ hoạt động tối ưu cho từng hoàn cảnh tương ứng,

1.1.2.3. Giải trí

Các thành viên trong gia đình cùng một lúc có thể lựa chọn các phương thức giải trí khác nhau tùy theo sở thích của mình từ các thiết bị ở những không gian khác nhau mà không ảnh hưởng đến nhu cầu của những người khác. Hoặc cả gia đình có thể cùng chia sẻ các hình ảnh, video hay từ những phòng khác nhau nhờ vào hệ thống cáp tín hiệu được nối tới TV từng phòng. Các hệ thống giải trí đa phương tiện hiện nay hỗ trợ kết nối radio, đầu DVD, Media Server, MP3, iPod...

1.1.2.4. Điều khiển

Tính năng điều khiển thông minh chính được thực hiện qua các panel điều khiển với công nghệ tiên tiến, được đặt ở các vị trí phù hợp với nội thất và thuận tiện cho việc sử dụng. Từ các panel này, ta có thể thiết lập và tùy chỉnh toàn bộ các hệ thống chiếu sáng, rèm cửa, an ninh, điều hoà nhiệt độ trong nhà.



Hình 1.5 Giao diện điều khiển nhà thông minh của BKAV

Ngoài ra còn có các bộ công cụ điều khiển nâng cao như phần mềm điều khiển được cài đặt trên điện thoại di động, laptop, trang web điều khiển, remote... Các tiện ích này giúp người dùng linh hoạt và chủ động hơn trong việc giám sát và quản lý từ xa các hoạt động trong ngôi nhà của mình.

Hệ thống cáp truyền tín hiệu có thể sử dụng độc lập hoặc dùng chính đường dây điện dân dụng có sẵn. Mỗi thiết kế đều có ưu và nhược điểm riêng nhưng nhìn chung đều có độ tin cậy cao. Các chuẩn truyền ngày càng được cải tiến nhằm gia tăng tốc độ và sự ổn định, nhất là khi các cáp tín hiệu thường được lắp đặt âm tường cùng với dây điện nhà.

1.1.2.5. Tiết kiệm năng lượng

Trong hoàn cảnh giá cả và tác động của việc sản xuất điện đối với môi trường hiện nay, vấn đề sử dụng năng lượng một cách hợp lý và tiết kiệm trở thành bài toán hàng đầu trong mọi lĩnh vực. Do đó, nếu một ngôi nhà chưa đạt được yêu cầu này chưa thể thực sự trở thành ngôi nhà thông minh. Nhiều giải pháp đã được các nhà

thiết kế giới thiệu và cung phần nào giải quyết được bài toán này như điều phôi hệ thống chiếu sáng, sử dụng các thiết bị điện hiệu năng cao (đèn compact hoặc LED thay cho đèn sợi đốt...), hạn chế tối đa việc sử dụng các thiết bị khi không cần thiết (cảm biến nhận biết con người để tắt mờ đèn, máy điều hoà, các thiết bị giải trí...), các bộ công cụ quản lý năng lượng tiêu thụ...

1.2. TÌNH HÌNH NGHIÊN CỨU TRONG VÀ NGOÀI NƯỚC

1.2.1. Thế giới

Nhà thông minh là sẽ là xu hướng thịnh hành trong tương lai khi IoT ngày một phát triển. Các ông lớn đang chạy đua để chiếm lĩnh thị trường đầy tiềm năng này. Với xu thế công nghệ Internet of Things (IoT) đang bùng nổ thì các thiết bị trong nhà ngày càng trở nên thông minh (smart devices) do được trang bị khả năng tính toán (computing power), khả năng cảm biến môi trường (sensor), và khả năng kết nối mạng (networking, internet connected). Các thiết bị khi được áp dụng công nghệ IoT thường được gọi là thiết bị thông minh (smart devices).

Thị trường thiết bị thông minh trong gia đình (Smart Home) là thị trường đầy hứa hẹn; theo ước tính của Business Insider, thì đến năm 2020 sẽ có 13 tỷ thiết bị được gọi là thiết bị Smart Home. Các thiết bị này rất đa dạng: camera an ninh, thiết bị báo cháy, khóa cửa, nhiệt kế, tủ lạnh, đèn chiếu sáng. Trong một hộ gia đình (ước tính ở thị trường Bắc Mỹ) có khoảng 100 đồ vật có thể áp dụng được công nghệ IoT để trở thành thiết bị thông minh.

Tuy tiềm năng như vậy, nhưng đây là thị trường chưa định hình sân chơi từ công nghệ, đến sản phẩm, và thói quen của người tiêu dùng. Ví dụ, hiện nay chưa có một chuẩn chung cho việc kết nối các thiết bị thông minh trong gia đình (smart home networking); Wi-Fi, Bluetooth, Z-Wave và Zigbee là các chuẩn kết nối thông dụng mà các thiết bị thông minh hiện tại đang sử dụng; mỗi công nghệ có ưu và nhược điểm riêng.

Tất cả các hãng công nghệ lớn như Google, Amazon, Apple, và Samsung đều đang tìm cách chiếm lĩnh thị trường Smart Home này. Đối với những hãng công nghệ lớn thì việc chiếm lĩnh và làm chủ hệ sinh thái của một xu hướng mới là vô

Chương 1: Giới thiệu

cùng quan trọng; việc này có ý nghĩa sống còn với họ. Ví dụ nhãn tiền gần đây là khi xu hướng công nghệ chuyển dịch từ PC/desktop sang hệ sinh thái smartphone (di động), Nokia tuy đã từng là hãng di động hàng đầu thế giới nhưng do không nắm bắt và theo kịp xu hướng smartphone mà giờ đây đã bị xóa sổ. Blackberry là cha đẻ của khái niệm smartphone nhưng do không vào được hệ sinh thái mới mà giờ cũng bị loại khỏi cuộc chơi.



Hình 1.6 Các hãng công nghệ tên tuổi tham gia thị trường Smart-home
Các sản phẩm Smart Home

Về cơ bản thì các ông lớn đều muốn kiểm soát và chi phối hệ sinh thái Smart Home, tuy vậy mỗi hãng có những động thái và chiến lược khác nhau, chúng ta sẽ xem xét chiến lược (cho tới thời điểm này) của bốn hãng lớn: Google, Amazon, Samsung, và Apple.

Google

Google đầu tư rất nhiều vào các công nghệ liên quan đến lĩnh vực IoT nói chung và smart hoome nói riêng: Google beacon platform, Brillo & Weave, OnHub Router, Google Cloud IoT. Tuy vậy để đi vào hộ tiêu dùng thì một chiến lược chính của Google là mua lại công ty Nest trong năm 2014. Nest đến giờ có ba sản phẩm chính khá thành công trên thị trường đó là: hệ thống điều hòa thông minh, hệ thống báo cháy, và camera giám sát gia đình. Doanh thu của Nest năm 2015 là \$340 triệu USD, tuy công ty mẹ Google tỏ ra thất vọng với kết quả này, nhưng đây vẫn là con

Chương 1: Giới thiệu

số khá lớn chứng tỏ Google (Nest) bán được một số lượng lớn thiết bị và đã đặt được chân vào thị trường này.



Hình 1.7 *Nest có thiết kế và tinh năng ưu việt so với hệ thống điều hòa cũ Amazon*

Amazon về cách tiếp cận thị trường có cách làm tương tự như Google (Nest), tức là bán một sản phẩm thông minh trong nhà rất hấp dẫn (killer product) để người dùng mua, sau đó biến thiết bị này thành một bộ điều khiển (hub) cho các thiết bị thông minh khác. Sản phẩm chiếm lĩnh thị trường Smart Home của Amazon là chiếc loa thông minh điều khiển bằng giọng nói (hands-free speaker), Amazon Echo. Điểm thú vị của chiếc loa này là không có phím điều khiển, mà người dùng giao tiếp với loa qua giọng nói; Amazon Echo hiểu được ngôn ngữ tự nhiên với độ chính xác rất cao. Ngoài việc chơi nhạc, Amazon Echo có thể trả lời người dùng các câu hỏi về thời tiết, đọc chuyện, mua hàng (tích hợp với hệ thống thương mại của Amazon).



Hình 1.8 Echo có giao diện tương tác bằng giọng nói

Cũng giống như chiến lược của Google, một khi người dùng đặt Amazon Echo trong nhà thì chính thiết bị này sẽ hoạt động như một bộ điều khiển trung tâm (hub) để tương tác với các thiết bị thông minh khác trong gia đình. Amazon mở ra bộ API để lập trình viên (developers) và các nhà phát triển thiết bị khác (OEM) tích hợp vào Echo; Amazon gọi một tính năng tích hợp mới này là “skill”, cho tới nay đã có tới 950 skills được tích hợp vào Amazon Echo. Sản phẩm Amazon Echo này thành công đến mức Google vừa phải nhanh chóng công bố một sản phẩm tương tự gọi là Google Home để cạnh tranh.

Samsung

Samsung cũng rất quan tâm đến thị trường Smart Home, một phần là thị trường thiết bị smartphone đang chững lại, Samsung tuy là nhà sản xuất thiết bị lớn nhưng không làm chủ được hệ sinh thái di động (mobile ecosystem), về phần mềm Samsung phụ thuộc vào Android của Google, về phần cứng các hãng Trung Quốc càng ngày càng cạnh tranh khốc liệt. Samsung tiến vào thị trường Smart Home bằng cách mua lại công ty SmartThings với giá \$200 triệu USD (Google mua Nest với giá \$3.2 tỷ USD).



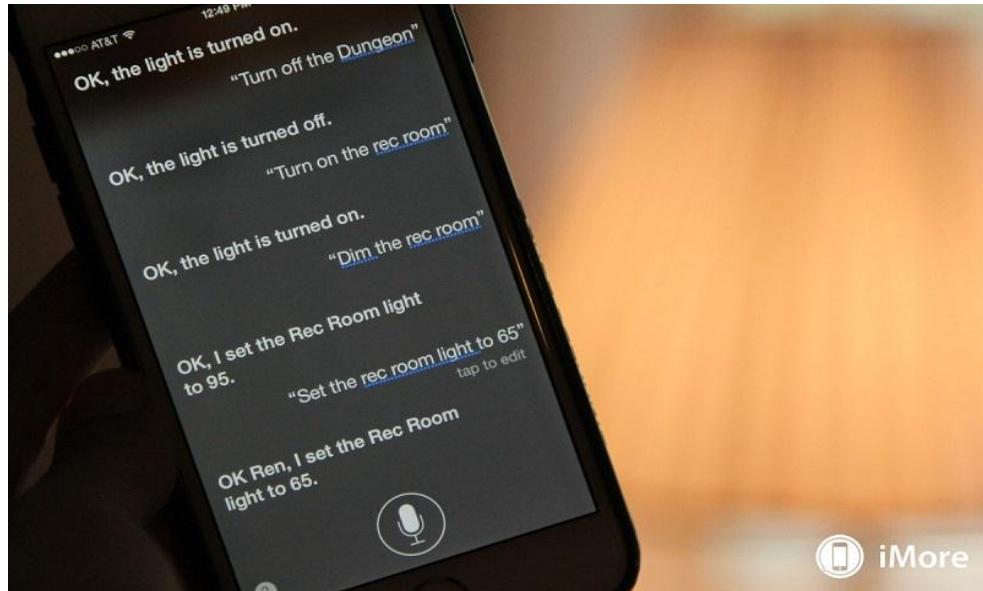
Hình 1.9 Samsung mua SmartThings để đặt chân vào thị trường nhà thông minh

SmartThings bán một bộ hub điều khiển trung tâm (smart-home controller); thế mạnh của thiết bị này là có thể giao tiếp với rất nhiều các thiết bị thông minh có trên thị trường qua các chuẩn giao tiếp (không dây) khác nhau. SmartThings cũng có một chương trình “Works with SmartThings” để chứng nhận các thiết bị tương thích với hệ thống này. Tuy vậy, cách tiếp cận bằng cách bán một bộ điều khiển trung tâm không được thành công lắm, việc thiếu một “killer product” khiến rất khó thuyết phục người tiêu dùng về lợi ích của hệ thống (hoặc người dùng không thấy nhu cầu đó là thiết yếu). Tuy vậy trong cuộc chơi này, Samsung có một thế mạnh là họ đã đang sản xuất rất nhiều thiết bị gia dụng (home appliances) có tiếng, Samsung có thể thâm nhập thị trường Smart Home qua các sản phẩm này, ví dụ gần đây họ vừa ra mắt chiếc tủ lạnh thông minh, tuy đang còn mang tính thử nghiệm cao, nhưng đây là một chiến lược rất hứa hẹn.

Apple

Apple bước vào thị trường Smart Home có một lợi thế là có nền tảng di động iPhone, với phiên bản iOS 9, Apple đưa ra chuẩn HomeKit để kết nối các thiết bị thông minh trong nhà với hệ điều hành iOS và điện thoại iPhone. Người dùng có thể dùng hệ thống nhận dạng giọng nói Siri trên iPhone để điều khiển các thiết bị trong nhà; một thành tố quan trọng trong bài toán chiếm lĩnh thị trường Smart Home của

Apple đó là Apple TV; thiết bị này sẽ hoạt động như một bộ điều khiển trung tâm để kết nối với các thiết bị khác trong gia đình. Apple đã bán được hơn 25 triệu sản phẩm Apple TV cho tới nay.



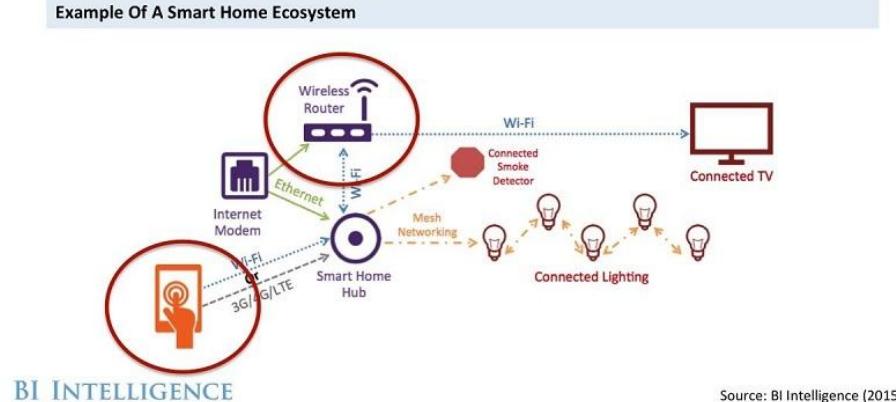
Hình 1.10 Apple dùng iOS làm bàn đạp để vào thị trường nhà thông minh

Có thể nói sau iPhone và iPad, Apple vẫn chưa đưa ra được một thiết bị nào có tính cách mạng; do vậy thị trường Smart Home có thể sẽ là một thị trường Apple mà Apple nhắm tới để đưa ra sản phẩm có tính đột phá.

Kiểm soát hệ sinh thái

Một đặc điểm chung trong chiến lược của các hãng là không chỉ nhắm tới việc tạo doanh thu trước mắt mà lâu dài phải kiểm soát được hệ sinh thái (ecosystem) của thị trường nhà thông minh (smart home). Với nền tảng công nghệ hiện nay thì 2 thành tố quan trọng trong việc xây dựng hệ sinh thái cho nhà thông minh đó là: smartphone và smart home hub như được minh họa ở hình dưới. Google và Apple có lợi thế nhất định khi làm chủ hai nền tảng chính cho smartphone là Android và iOS. Họ có thể tích hợp thắng những công nghệ để giúp họ triển khai dễ dàng các sản phẩm và dịch vụ trong thị trường mới này.

Both of these are crucial factors for the smart home.



Hình 1.11 Hai thành tố chính cho hệ sinh thái nhà thông minh: smartphone và smart home hub

1.2.2. Trong nước

Nhà thông minh là khái niệm còn khá mới mẻ tại Việt Nam. Tuy nhiên, thị trường nhà thông minh ở Việt Nam lại đang cạnh tranh khốc liệt với sự góp mặt của hàng loạt các thương hiệu như: BKAV, Lumi, Hager, Acis, Arkos, Gamma(Siemens), Legrand, Crestron...

Sau đây chúng ta hãy điểm danh qua 1 số thương hiệu nhà thông minh đình đám tại Việt Nam. Thương hiệu nhà thông minh Việt Nam. Cạnh tranh trực tiếp với các thương hiệu đến từ nước ngoài là những nhà sản xuất thiết bị điện thông minh và giải pháp nhà thông minh Việt Nam như BKAV, Lumi, Acis... trong đó BKAV và Lumi là 2 tên tuổi lớn.

Nhà thông minh BKAV là sản phẩm của tập đoàn công nghệ BKAV, thuộc đại học bách khoa Hà Nội. Ngay từ khi ra mắt, Smarthome BKAV được tập trung vào phân khúc cao cấp trên thị trường, cạnh tranh trực tiếp với các sản phẩm smarthome đến từ nước ngoài với chi phí tương đương. Điều này cũng là thách thức không nhỏ với BKAV trong việc đưa nhà thông minh - smarthome đến gần hơn với người dùng Việt Nam.

Khác với BKAV, Nhà thông minh Lumi tập trung mạnh vào phân khúc nhà thông minh trung và cao cấp trên thị trường Việt Nam. Ra mắt công tắc cảm ứng và

Chương 1: Giới thiệu

giải pháp nhà thông minh ra thị trường đầu năm 2012, sau 4 năm xây dựng và phát triển, Lumi đã vươn lên trở thành nhà cung cấp sản phẩm công tắc điện cảm ứng, thiết bị điện thông minh và giải pháp nhà thông minh có thị phần lớn nhất trong phân khúc trung và cao cấp tại Việt Nam. Ra đời dưới sự tìm tòi, nghiên cứu và sáng tạo không ngừng bởi 3 cựu thành viên Robocon Bách Khoa 2008, Lumi tự hào là sản phẩm công nghệ cao made in Việt Nam. Chiếc công tắc cảm ứng điều khiển từ xa của Lumi đã dần xuất hiện trong những công trình, những dự án bất động sản lớn trên cả nước như: Times City, Royal City,... Ngoài 2 ông lớn Lumi và BKAV, Acis cũng là 1 thương hiệu nhà thông minh Việt Nam được biết đến. Ra đời với sự nghiên cứu tìm tòi của các cựu sinh viên Bách Khoa TP HCM, Acis tự hào là sản phẩm nhà thông minh thương hiệu Việt. Có trụ sở tại TP HCM, Acis tập trung chủ yếu tại thị trường miền Nam.

Ngoài những cái tên kể trên, thị trường nhà thông minh tại Việt Nam còn có sự góp mặt của các sản phẩm thiết bị điện thông minh và giải pháp nhà thông minh giá rẻ đến từ Trung Quốc. Tuy nhiên, chất lượng những sản phẩm này chưa được đánh giá cao trên thị trường. Những cái tên phải kể đến như Kawa, Broadlink, Bluetech, ...

1.3. NHIỆM VỤ LUẬN VĂN

Để bắt kịp xu hướng phát triển trong nước cũng như thế giới, trong giới hạn phạm vi đề tài luận văn, em xin được thiết kế và thi công mô hình nhà thông minh I-home dựa trên các tính năng tham khảo từ các mô hình nhà thông minh trên thị trường. Bên cạnh đó là việc tìm hiểu các lý thuyết để có thể điều khiển và vận hành nhà thông minh. Xây dựng cơ chế quản lý cho nhà thông minh từ đó thực tế hóa mô hình.

Chương 2. CƠ SỞ LÝ THUYẾT

2.1. CHUẨN GIAO TIẾP RS485:

2.1.1. Giới thiệu

RS485 hay TIA/EIA-485 (ANSI Telecommunications Industry Associations/Electronic Industries Allience) là tiêu chuẩn được định nghĩa bởi những đặc tính điện của phần phát và phần nhận được sử dụng trong các hệ thống truyền nhận kỹ thuật số cân bằng đa điểm. Chuẩn RS485 được sử dụng hiệu quả trong các hệ thống mạng tín hiệu đòi hỏi khoảng cách, nhiều nhiễu điện từ với tốc độ truyền tương đối cao.

Trong công nghiệp ngày nay, chuẩn truyền thông RS232 không thể đáp ứng được nhu cầu truyền thông nữa vì sử dụng đường truyền không cân bằng (các tín hiệu đều lấy điểm chuẩn là đường mass chung, bị ảnh hưởng của nhiều tác động) do đó tốc độ truyền và khoảng cách truyền bị giới hạn (khoảng cách truyền lý thuyết tối đa 100m). Vì vậy để đáp ứng nhu cầu truyền thông công nghiệp, người ta sử dụng chuẩn truyền RS485 khi cần tăng khoảng cách và tốc độ (khoảng cách truyền thông tối đa 1200m và vận tốc truyền lên đến 10Mbits/s). Nguyên nhân mà RS485 có thể tăng tốc độ và khoảng cách truyền thông là do RS485 sử dụng phương pháp truyền 2 dây vi sai cân bằng (vì 2 dây có đặc tính giống nhau, tín hiệu truyền đi là hiệu số điện áp giữa 2 dây do đó loại trừ được nhiễu chung). Một khác do chuẩn truyền thông RS232 không cho phép có hơn 2 thiết bị truyền nhận tin trên đường dây trong khi đó với chuẩn RS485 ta có thể sử dụng lên tới 32 thiết bị thu phát. Các dây tín hiệu thường được ký hiệu là A (+) và B (-).

2.1.2. Đặc tính kỹ thuật

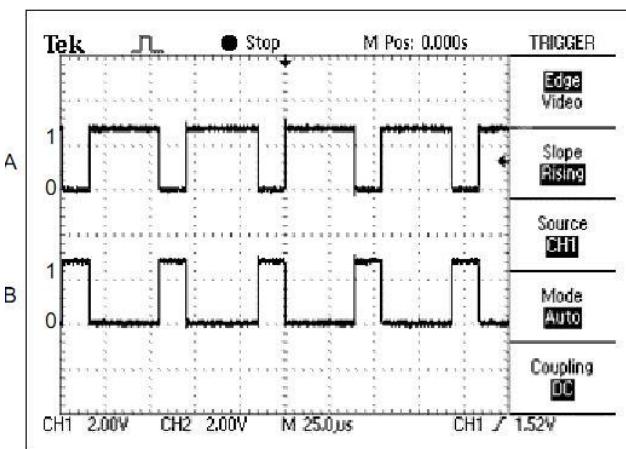
2.1.2.1. Thông số cơ bản

Chương 2: Cơ sở lý thuyết

RS485	
Differential	yes
Max number of drivers	32
Max number of receivers	32
Modes of operation	half duplex
Network topology	multipoint
Max distance (acc. standard)	1200 m
Max speed at 12 m	35 Mbs
Max speed at 1200 m	100 kbs
Max slew rate	n/a
Receiver input resistance	$\geq 12 \text{ k}\Omega$
Driver load impedance	54Ω
Receiver input sensitivity	$\pm 200 \text{ mV}$
Receiver input range	-7..12 V
Max driver output voltage	-7..12 V
Min driver output voltage (with load)	$\pm 1.5 \text{ V}$

Hình 2.1 Các thông số đặc tính cơ bản của chuẩn RS485

Như đã nêu trên, RS485 là chuẩn truyền sử dụng đường truyền vi sai, mức tín hiệu ở các ngõ ra được xác định dựa trên sai biệt điện áp giữa 2 dây tín hiệu, nếu $V_{AB} > 200 \text{ mV}$ sẽ cho ra mức logic 1, $V_{AB} < 200 \text{ mV}$ sẽ cho ra mức logic 0, khi độ chênh điện áp ở dây A và B nằm giữa mức này được xem là vùng bất định. Điện thế của mỗi dây tín hiệu so với mass bên phía thu phải nằm trong giới hạn -7V đến 12V. Nhờ đặc tính này cùng với việc sử dụng cáp tín hiệu loại dây xoắn giúp loại bỏ được nhiễu chung nên chuẩn RS485 có khả năng kháng nhiễu mạnh.

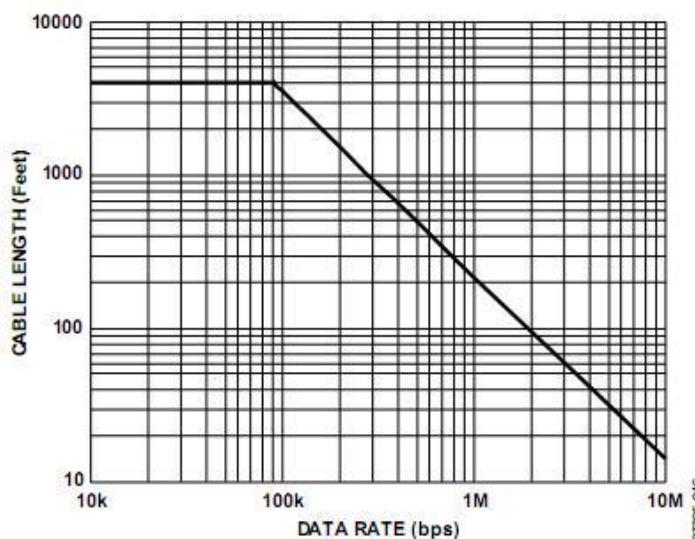


Hình 2.2 Dạng tín hiệu trên 2 đường truyền RS485.

Hình trên cho thấy sự đối nhau ở 2 đường truyền, khi đường này mức 1 thì đường kia mức 0 và ngược lại, điều này nhằm đảm bảo cho sự chênh lệch điện áp giữa 2 dây để xác định chính xác mức logic ở ngõ ra.

Chương 2: Cơ sở lý thuyết

Một khả năng nổi bật nữa của chuẩn RS485 là khả năng mở rộng đến 32 trạm, tùy theo cấu hình hệ thống mà các trạm có thể là bộ phát hoặc bộ thu, đây là một ưu thế lớn đối với các hệ thống cần sử dụng tính ngang quyền (các trạm đều có thể chủ động truyền tín hiệu trên đường dây rỗng) mà đa số các chuẩn truyền nối tiếp khác không làm được.

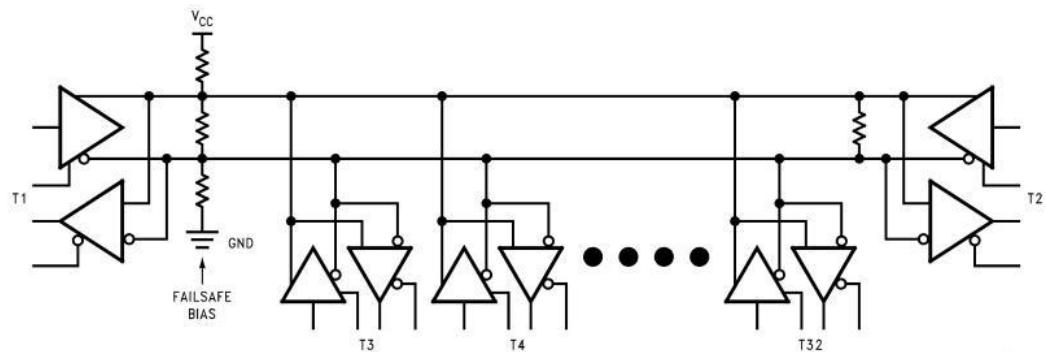


Hình 2.3 Tương quan giữa tốc độ truyền và chiều dài đường dây

Điểm mạnh khác của RS485 là tốc độ truyền khá cao, hiện nay có thể đạt đến hơn 10Mbit/s và khoảng cách truyền có thể lên tới 1200m (4000feet). Tuy nhiên 2 giới hạn này không thể đạt được cùng lúc. Theo đồ thị trên, ta thấy được mối tương quan giữa 2 đại lượng này, tốc độ 10Mbit/s chỉ có thể dùng với cự ly không quá 3m (10feet). Ngược lại, với khoảng cách 1200m tốc độ tối đa có thể lên tới là khoảng 100kbit/s.

Về cơ bản RS485 chỉ có thể truyền bán song công do sử dụng cùng lúc cả 2 đường tín hiệu.

2.1.2.2. Phân cực đường truyền

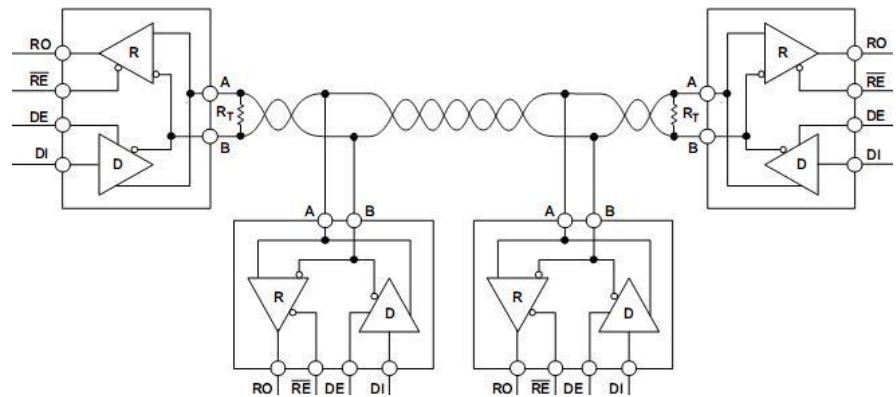


Hình 2.4 Phân cực fail-safe trên đường truyền đa trạm chuẩn RS485

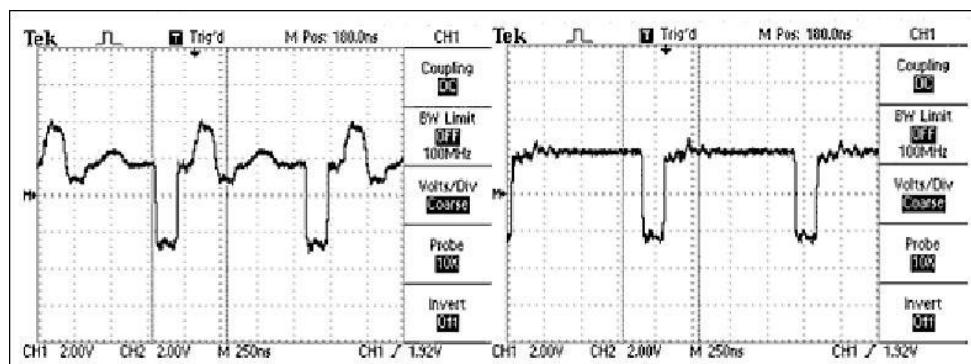
Trong mạng đa trạm ngang quyền, khi không có trạm nào phát, đường truyền phải được nằm trong một trạng thái idle xác định và các ngõ vào mỗi trạm đều ở trạng thái tổng trở cao. Đối với RS485, khi ở trạng thái idle, ngõ ra phải được đặt ở mức cao để chờ Startbit (mức thấp) báo hiệu có dữ liệu được phát. Điều này được đảm bảo bằng việc phân cực fail-safe trên đường truyền. Mục đích việc phân cực này nhằm giữ cho điện áp trên dây A luôn lớn hơn dây B ít nhất 200mV khi không có trạm nào phát, do đó giữ được mức tín hiệu ở ngõ ra ở mức cao.

2.1.2.3. Điện trở đầu cuối

Điện trở đầu cuối là điện trở nối giữa 2 dây tín hiệu được đặt tại đầu ngoài cùng của đường truyền phía thu, có tác dụng phối hợp với trở kháng đặc tính Z_0 của cáp tín hiệu (do nhà sản xuất cung cấp) nhằm loại bỏ sóng phản xạ trên đường truyền dài. Do chuẩn RS485 sử dụng chung cặp dây tín hiệu cho việc thu và phát nên cần đặt điện trở đầu cuối ở cả 2 phía của đường truyền. Giá trị của điện trở đầu cuối lý tưởng bằng giá trị trở kháng đặc tính cáp tín hiệu, thông thường vào khoảng 100Ω - 120Ω .



Hình 2.5 Vị trí đặt điện trở đầu cuối trên đường truyền RS485.



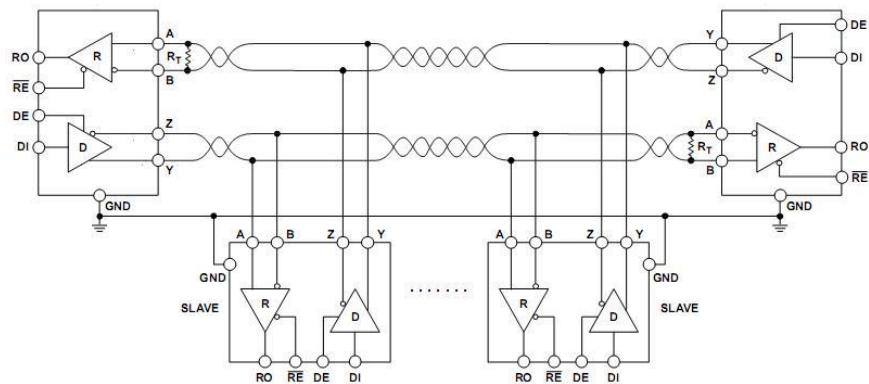
RT=54Ω

RT=120Ω

Hình 2.6 Dạng sóng ngoã ra trên dây A tương ứng với 2 giá trị điện trở đầu cuối

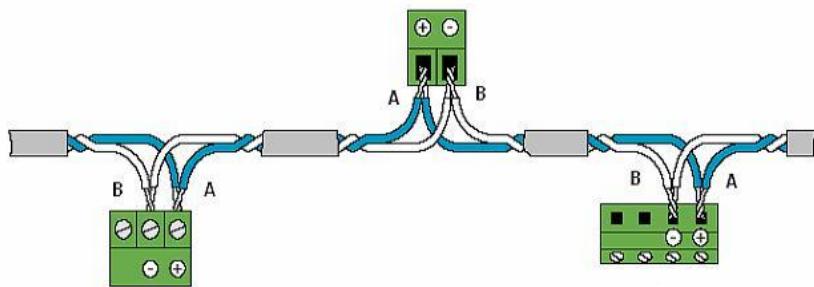
2.1.2.4. Cấu trúc kết nối

Cấu trúc mạng của RS485 cơ bản chỉ có 2 đường tín hiệu dùng chung cho việc truyền nhận, do đó chỉ có thể hoạt động ở chế độ bán song công (half-duplex, hình 2.7). Tuy nhiên, để cải tiến về chế độ truyền người ta cũng có thể lắp đặt mạng theo sơ đồ 4 dây. Khi đó, mỗi cặp dây sẽ chỉ làm nhiệm vụ truyền hoặc phát nén có thể hoạt động ở chế độ song công (full-duplex).



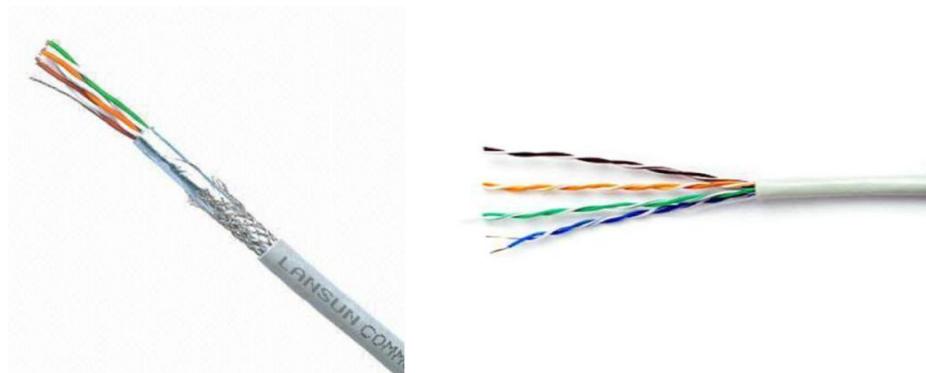
Hình 2.7 Cấu trúc sơ đồ 4 dây cho chế độ full-duplex

2.1.2.5. Dây dẫn tín hiệu



Hình 2.8 Cách đấu dây thực tế của mạng RS485 sử dụng cặp dây xoắn

Việc lựa chọn dây dẫn cho đường truyền cũng là một vấn đề khá quan trọng do yêu cầu về phối hợp trở kháng và sử dụng trong môi trường nhiều nhiễu điện từ. Theo khuyến cáo, nên chọn loại dây xoắn 24AWG có trở kháng khoảng 120Ω . Nếu có thể nên chọn loại có bọc kim cho khả năng chống nhiễu tốt hơn tuy nhiên giá thành cao hơn đáng kể.



Hình 2.9 Cáp xoắn đôi 24AWG có bọc chống nhiễu (trái)

2.2. VI ĐIỀU KHIỂN:

2.2.1. Lý thuyết vi điều khiển

2.2.1.1 Giới thiệu

Vi điều khiển (μ C hay UC) là một siêu máy tính với kích thước rất nhỏ. Vi điều khiển là một hệ thống độc lập với thiết bị ngoại vi, bộ nhớ và bộ vi xử lý sử dụng như một hệ thống nhúng. Ngày nay hầu hết vi điều khiển được lập trình để nhúng vào các sản phẩm tiêu dùng hoặc thiết bị máy móc, điện thoại, thiết bị ngoại vi, xe ô tô và chế tạo thiết bị cho các hệ thống máy tính. Có rất nhiều loại vi điều khiển trên thị trường như: 4bit, 8bit, 64bit và 128bit. Vi điều khiển sử dụng trong các thiết bị được người dùng kiểm soát bằng các tập lệnh.

2.2.1.2. Cấu trúc cơ bản

CPU- là bộ não trung tâm của vi điều khiển. CPU là thiết bị quản lý tất cả các hoạt động của hệ thống và thực hiện tất cả các thao tác trên dữ liệu như: nạp, giải mã và thực thi lệnh. CPU kết nối tất cả các thành phần của vi điều khiển thành một hệ thống duy nhất.

Memory(bộ nhớ): trong vi điều khiển, bộ nhớ hoạt động giống như bộ vi xử lý. Bộ nhớ lưu trữ tất cả các chương trình và dữ liệu. Bộ nhớ của vi điều khiển là bộ nhớ ROM(EPROM, EEPROM) hoặc bộ nhớ RAM với dung lượng nhất định. Ngày nay còn có bộ nhớ flash lưu trữ mã nguồn chương trình.

Cổng Input/output (vào/ ra)- cổng I/O sử dụng để giao tiếp hoặc điều khiển các thiết bị khác nhau như máy in, LCD, LED, ...

Serial Ports - Những cổng này cung cấp giao tiếp nối tiếp giữa vi điều khiển và thiết bị ngoại vi khác nhau.

Timers - Vi điều khiển được xây dựng với một hoặc nhiều Timer hoặc bộ định thời. Các Timer và bộ định thời kiểm soát tất cả bộ đếm và thời gian hoạt động bên trong vi điều khiển. Timer được sử dụng đếm xung bên ngoài. Các hoạt động chính được thực hiện bởi timers “tạo xung, đo tần số, điều chế, tạo dao động,...”

ADC - ADC là bộ chuyển đổi tín hiệu tương tự sang tín hiệu số.

DAC (digital to analog converter) : Có chức năng ngược lại với ADC. DAC thường được sử dụng để giám sát các thiết bị tương tự.

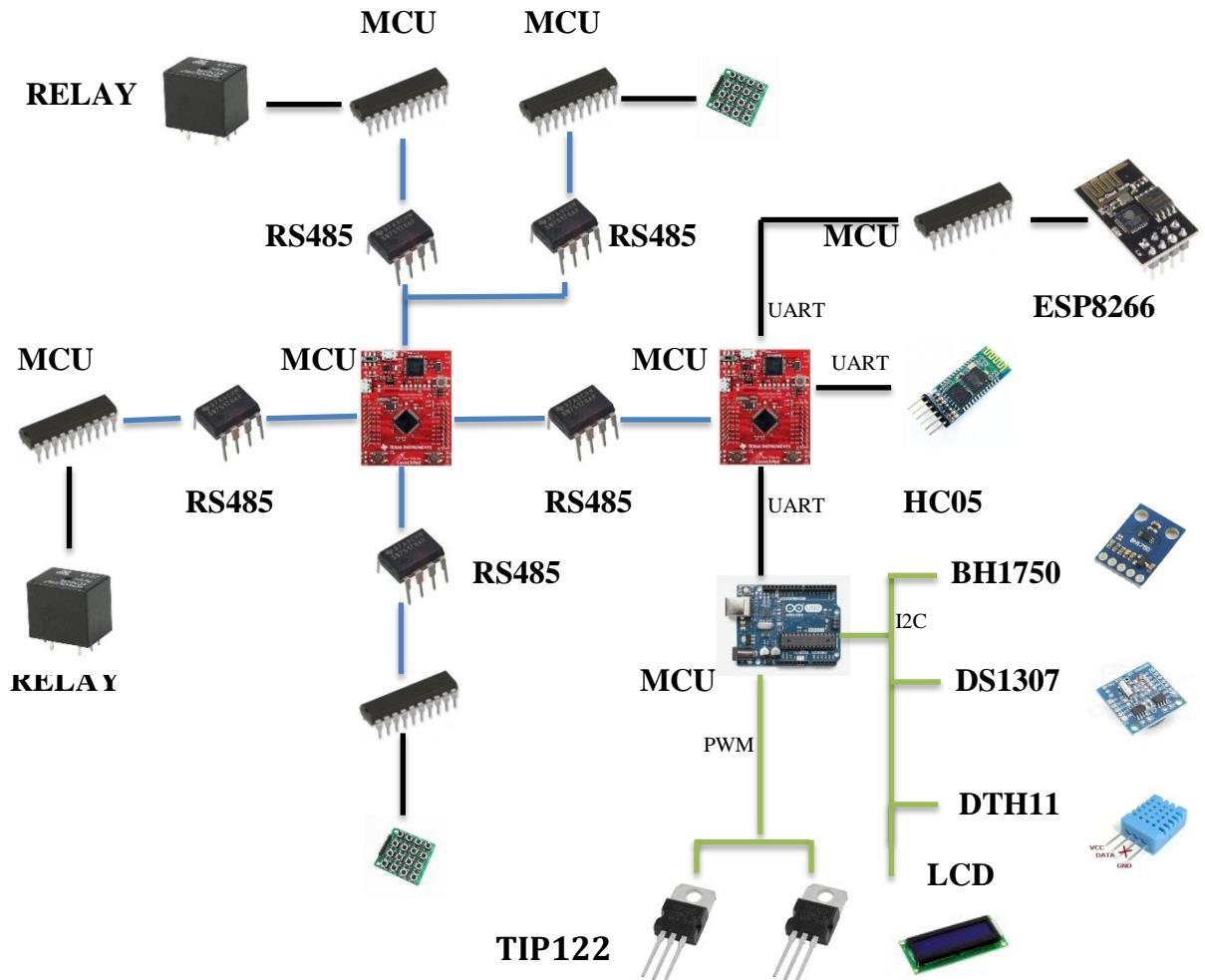
Interpret Control (điều khiển ngắn) - là một số sự kiện khẩn cấp bên trong hoặc bên ngoài bộ vi điều khiển xảy ra, buộc vi điều khiển tạm dừng thực hiện chương trình hiện tại, phục vụ ngay lập tức nhiệm vụ mà ngắn yêu cầu – nhiệm vụ này gọi là trình phục vụ ngắn (ISR).

2.2.1.3. Một số dòng vi điều khiển

- Freescale 68HC11 (8-bit) Intel 8051
- STMicroelectronics STM8S (8-bit), ST10 (16-bit) và STM32 (32-bit)
- Atmel AVR (8-bit), AVR32 (32-bit), và AT91SAM (32-bit)
- Freescale ColdFire (32-bit) và S08 (8-bit)
- Hitachi H8 (8-bit), Hitachi SuperH (32-bit)
- MIPS (32-bit PIC32)
- PIC (8-bit PIC16, PIC18, 16-bit dsPIC33 / PIC24)
- PowerPC ISE
- PSoC (Programmable System-on-Chip)
- Texas Instruments Microcontrollers MSP430 (16-bit), C2000 (32-bit), và Stellaris (32-bit)
- Toshiba TLCS-870 (8-bit/16-bit)
- Zilog eZ8 (16-bit), eZ80 (8-bit)
- Philips Semiconductors LPC2000, LPC900, LPC700

Chương 3. THIẾT KẾ VÀ THỰC HIỆN PHẦN CỨNG

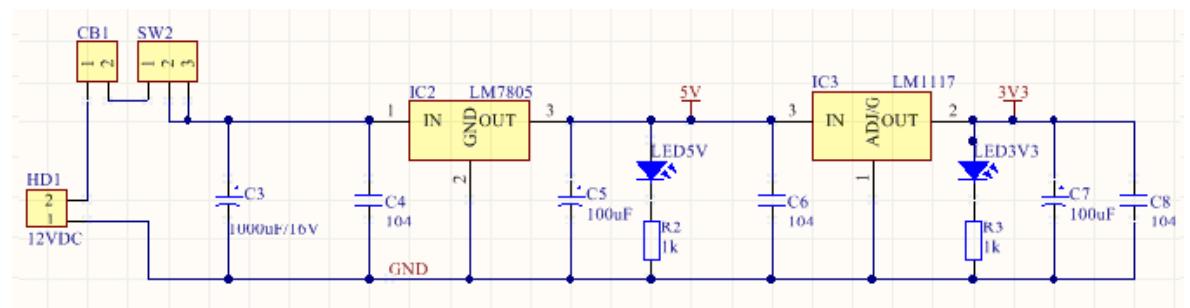
3.1. SƠ ĐỒ KẾT NỐI:



Hình 3.1 Sơ đồ kết nối phần cứng

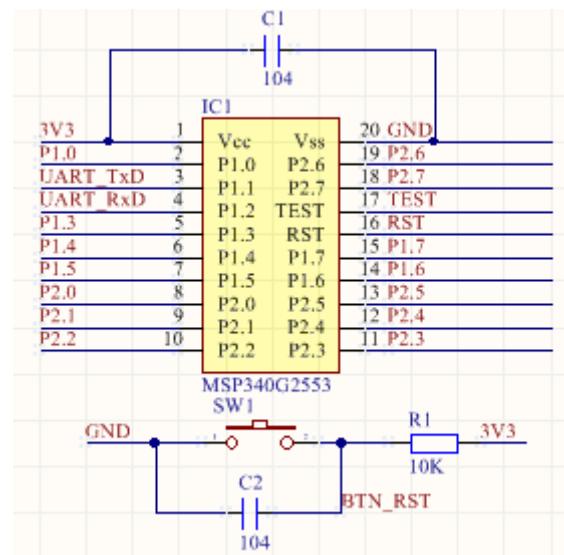
3.2. SƠ ĐỒ NGUYÊN LÝ:

Mạch Slave Input

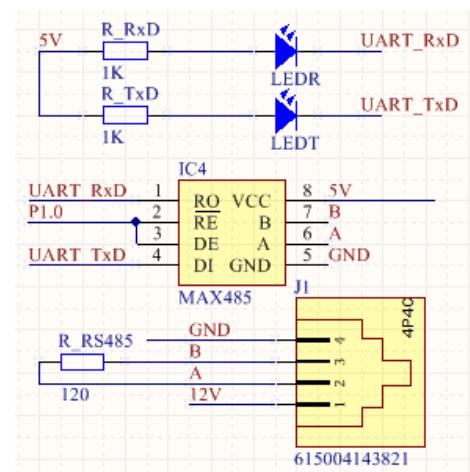


Hình 3.2 Sơ đồ mạch nguồn

Chương 3. Thiết kế và thực hiện phần cứng

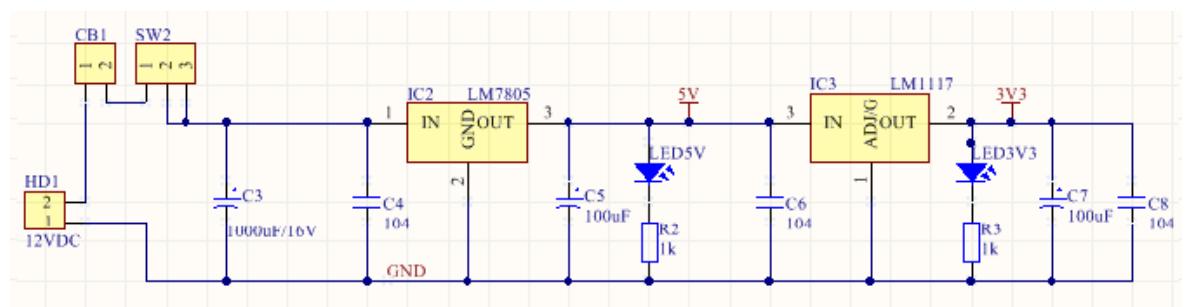


Hình 3.3 Sơ đồ vi điều khiển



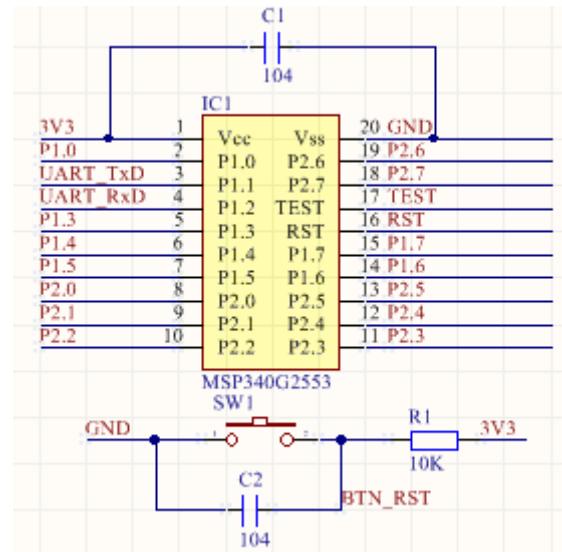
Hình 3.4 Sơ đồ giao tiếp RS485

Mạch Slave Output

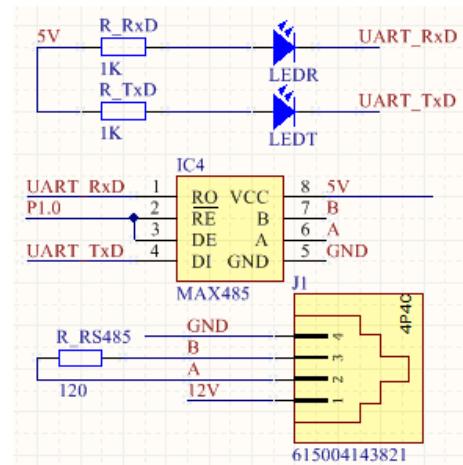


Hình 3.5 Sơ đồ mạch nguồn

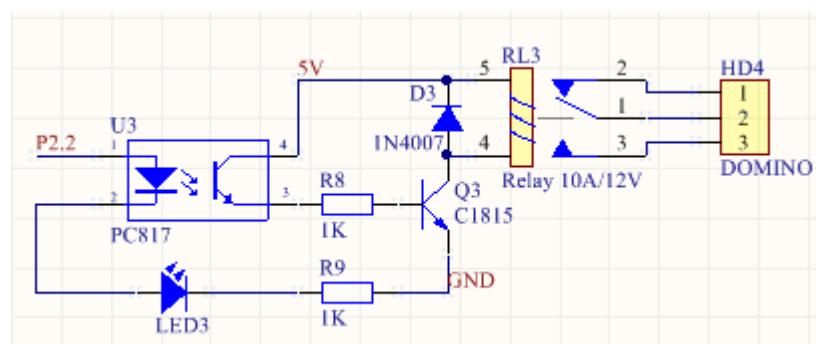
Chương 3. Thiết kế và thực hiện phần cứng



Hình 3.6 Sơ đồ vi điều khiển output



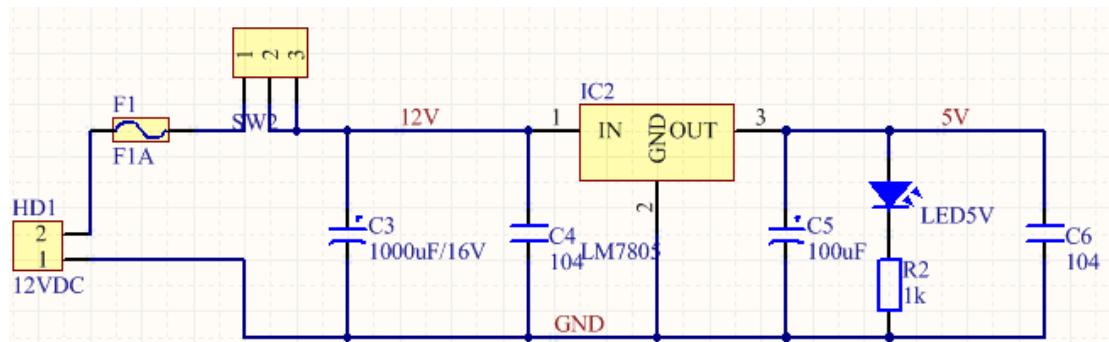
Hình 3.7 Sơ đồ giao tiếp RS485



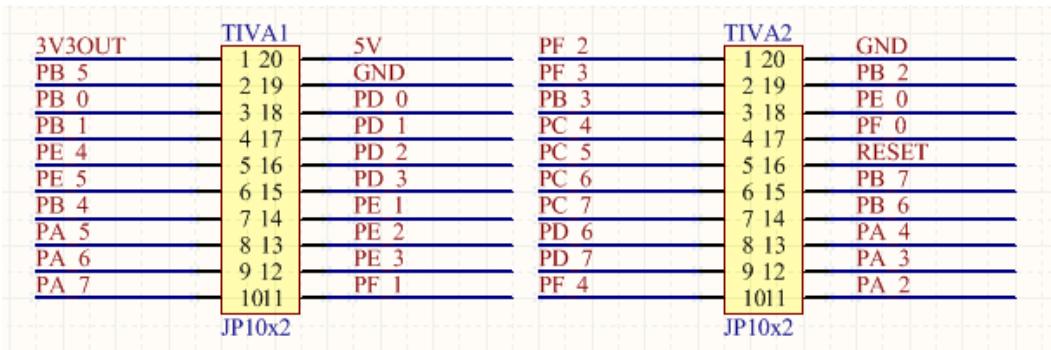
Hình 3.8 Sơ đồ mạch relay

Chương 3. Thiết kế và thực hiện phần cứng

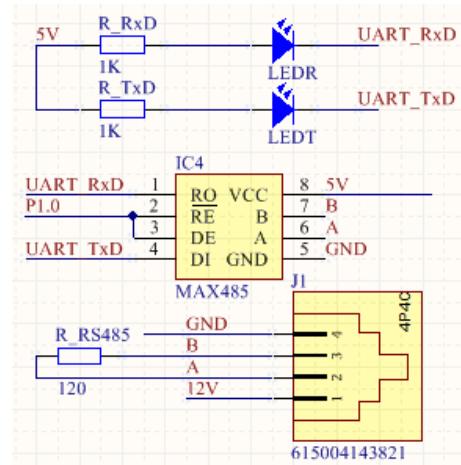
Mạch Slave đa chức năng



Hình 3.9 Sơ đồ mạch nguồn



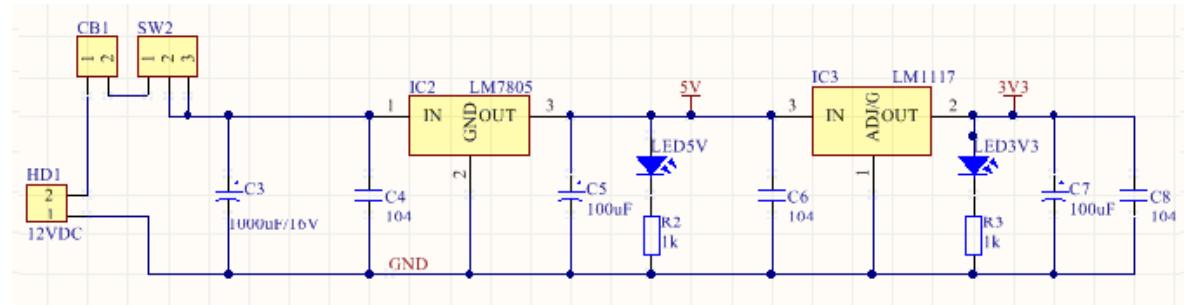
Hình 3.10 Sơ đồ vi điều khiển TIVA



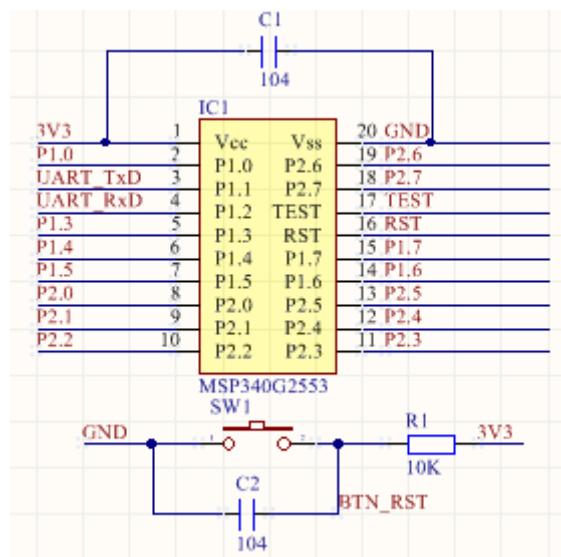
Hình 3.11 Sơ đồ giao tiếp RS485

Chương 3. Thiết kế và thực hiện phần cứng

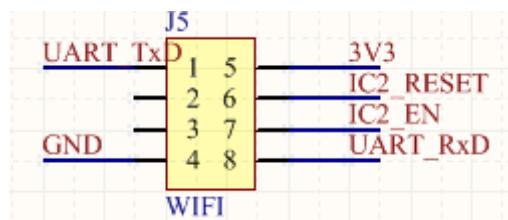
Mạch MSP-ESP8266



Hình 3.12 Sơ đồ mạch nguồn chính



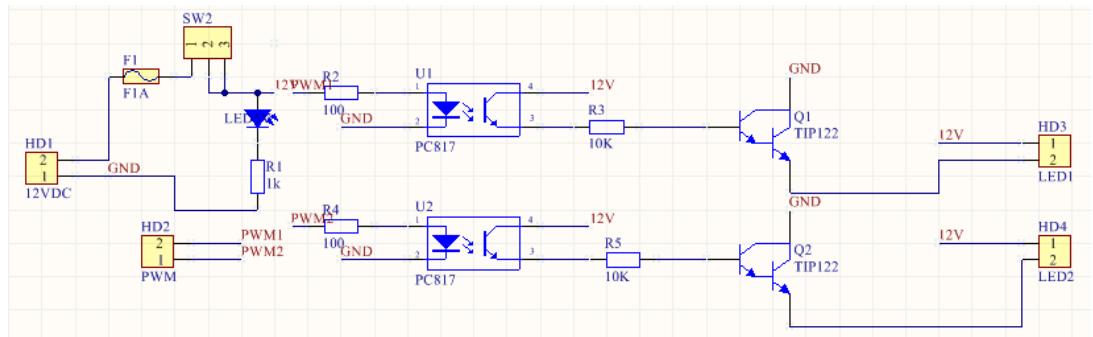
Hình 3.13 Sơ đồ vi điều khiển



Hình 3.14 Sơ đồ kết nối module Wifi ESP8266

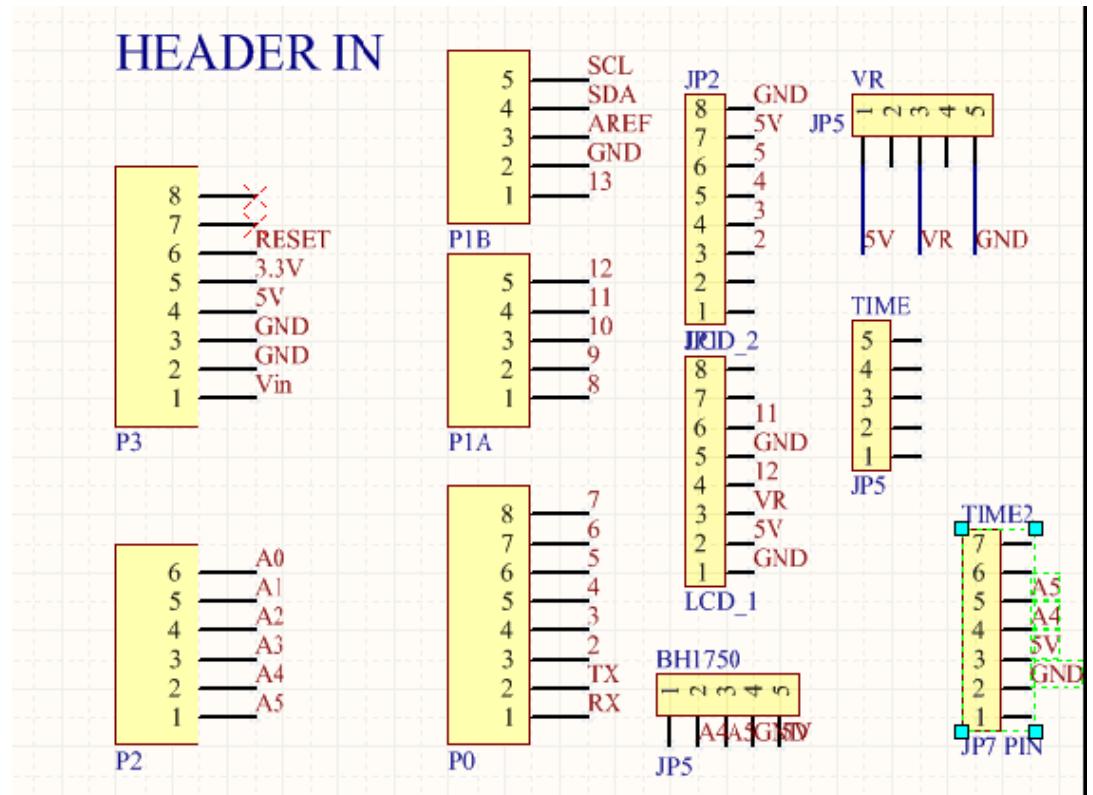
Chương 3. Thiết kế và thực hiện phần cứng

Mạch Dimmer



Hình 3.15 Sơ đồ mạch Dimmer LED

Mạch Adruino



Hình 3.16 Sơ đồ mạch kết nối với Adruino

3.3. VI ĐIỀU KHIỂN:

3.3.1. Vi điều khiển MSP430

Dòng vi điều khiển MSP430 được sản xuất bởi TI (Texas Instruments) . Dòng vi điều khiển MSP430 có một số phiên bản như: MSP430x1xx, MSP430x2xx, MSP430x3xx, MSP430x4xx, MSP430x5xx. Ở trong luận văn này sử dụng MSP430g2553.

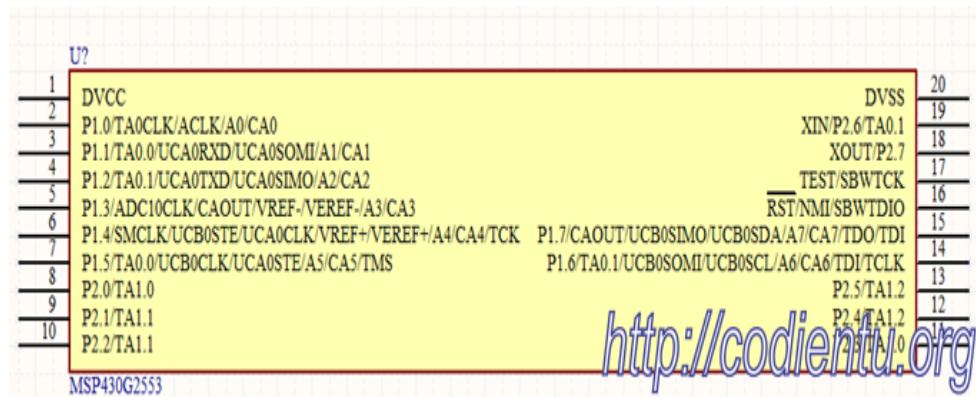
Một số đặc điểm của dòng vi điều khiển MSP430 như sau:

- Cấu trúc sử dụng nguồn thấp giúp kéo dài tuổi thọ của Pin
- Duy trì $0.1\mu A$ dòng nuôi RAM.
- Chỉ $0.8\mu A$ real-time clock.
- Bộ tương tự hiệu suất cao cho các phép đo chính xác
- Bộ giám sát điện áp nguồn.
- Thanh ghi lớn nên loại trừ được trường hợp tắt nghẽn tập tin khi đang làm việc.
- Thiết kế nhỏ gọn làm giảm lượng tiêu thụ điện và giảm giá thành.
- Tối ưu hóa cho những chương trình ngôn ngữ bậc cao như C, C++
- Có 7 chế độ định địa chỉ.
- Khả năng ngắt theo vec tor lớn.
- Trong lập trình cho bộ nhớ Flash cho phép thay đổi Code một cách linh hoạt, phạm vi rộng, bộ nhớ Flash còn có thể lưu lại như nhật ký của dữ liệu.

Sơ đồ chân :

Chip MSP430 có kích thước nhỏ gọn , chỉ với 20 chân đối với kiểu chân DIP. Bao gồm 2 port I/O (hay GPIO general purpose input/ output : cổng nhập xuất chung). Ta thấy rằng mỗi port đều có 8 chân.

Chương 3. Thiết kế và thực hiện phần cứng



Hình 3.17 Sơ đồ chân Chip MSP430

Port 1 : Có 8 chân từ P1.0 đến P1.7 tương ứng với các chân từ 2-7 và 14,15.

Port 2 : Cũng gồm 8 chân P2.0 đến P2.7 ứng với chân 8-13,18,19.

Ngoài chức năng I/O thì trên mỗi pin của các port đều là những chân đa chức năng.

3.3.2. ARM Cortex M4

ARM® Cortex®-M4 là dòng vi xử lý được hãng TI phát triển giành cho hệ thống nhúng sử dụng các phương pháp xử lý tín hiệu số yêu cầu hiệu năng cao và dễ dàng trong việc liên kết. Sự kết hợp giữa tính đầy đủ và đa dạng các chức năng với tiêu thụ công suất thấp, dễ sử dụng và giá thành ngày càng hạ khiết các vi điều khiển dòng Cortex M phù hợp trong nhiều ứng dụng thực tế như điều khiển động cơ, xe thông minh tiết kiệm năng lượng, âm thanh số hay các hệ thống tự động hóa trong công nghiệp.

ARM® Cortex®-M4 kế thừa các tài nguyên từ các dòng M0, M1, M3 trước và bổ sung thêm những tính năng mới (nổi bật là khả năng tính toán dấu chấm động Floating Point Unit tăng tốc độ tính toán rất nhiều lần), đồng thời sở hữu khả năng tiêu thụ dòng thấp kỷ lục và giá cả hợp lý khiến nó trở thành một trong những dòng vi xử lý thông dụng nhất.

TI Embedded Processing Portfolio

Embedded Processing Portfolio						
Microcontroller (MCU) Portfolio at a Glance		ARM®-Based Processor Portfolio at a Glance			Digital Signal Processor (DSP) Portfolio at a Glance	
MCU		Software, Tools, Kits & Boards			DSP & ARM® MPU	
16-bit ultra-low power MCUs MSP430™	32-bit real-time MCUs C2000™	32-bit ARM® MCUs Tiva™ C Series ARM Cortex™-M4F	32-bit ARM® safety MCUs Hercules™ ARM Cortex-R4F	32-bit ARM® processors Sitara™ ARM Cortex-A8 ARM9™	Singlecore DSPs C5000™ C6000™	Multicore processors C6000™ DSP and ARM Cortex-A15
Overview Device Table SW & Kits Up to 25 MHz Flash 1 KB to 256 KB Analog I/O, ADC, LCD, USB, FRAM Measurement, sensing, general purpose \$0.25 to \$9.00	Overview Device Table SW & Kits 40 MHz to 300 MHz Flash, RAM 16 KB to 512 KB PWM, ADC, CAN, SPI, I²C Motor control, digital power, lighting, ren. energy \$1.85 to \$20.00	Overview Device Table SW & Kits Up to 80 MHz Flash 32 KB to 256 KB USB, CAN, ADC, PWM, SPI Home, building, and industrial \$2.15 to \$5.25	Overview Device Table SW & Kits Fixed floating up to 220 MHz Flash 256 KB to 3 MB USB, ENET, FlexRay™, Timer/PWM, ADC, CAN, LIN, SPI, I²C, EMIF Safety, transportation, industrial & medical \$5.00 to \$30.00	Overview Device Table SW & Kits Up to 1.35 GHz Up to 32 KB I/D cache 256 KB L2, LPDDR, DDR2/3 support GEMAC, PCIe+PHY, SATA+PHY, CAN, USB+PHY, PR-ICSS Consumer, industrial, connected home, POS smart grid, medical \$5.00 - \$25.00	Overview Device Table SW & Kits Up to 800 MHz SDRAM, DDR2 uPP, I²C, I²S, UHP, MCASP/McBSP, LCD/C, integrated connectivity options, USB 2.0, EMAC Patient monitoring, biometric security, smart e-meter, industrial drives \$2.00 to \$25.00	Overview Device Table SW & Kits Up to 10 GHz multicore, fixed/floating + accelerators Up to 4 MB SL2, 32 KB L1, 1 MB L2 RapidIO®, PCIe, McBSP, 10/100 MAC, uPP, UART, Hyperlink, DDR2/3 Telecom, medical, mission critical, base stations \$30 to \$225.00

Hình 3.18 Các dòng vi xử lý, DSP của hãng Texas Instrument

Bảng 1 Dòng tiêu thụ ở các chế độ hoạt động của vi điều khiển Cortex M4

Chế độ hoạt động	Dòng Cortex M4
Bình thường (Run)	~ 32mA
Ngủ (Sleep)	~ 8mA
Ngủ sâu (Deep Sleep)	220µA
Ngủ đông (Hibernate)	1.5µA (2.5 nếu bật RTC)

Vì điều khiển TM4C123G

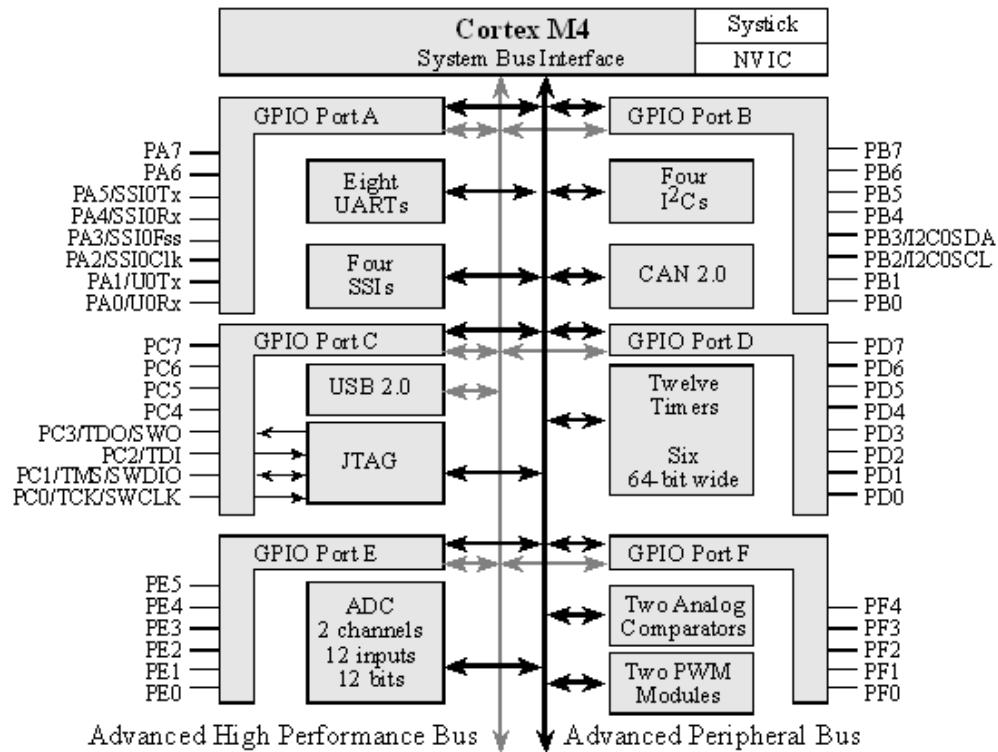
Tần số xung nhịp 80 MHz xử lý lõi 32 bit với hệ thống Timer (SysTick), tích hợp điều khiển ngắt bằng vector (NVIC), điều khiển ngắt Wake-up (WIC), khởi bão vệ bộ nhớ (MPU) và xử lý dấu chấm động (FPU) đảm bảo tốc độ xử lý.

4 khói I2C với chuẩn 100Mbps và 400 Mbps để giao tiếp với IMU.

8 khói UART có thể cấu hình ngắt, kết hợp với bluetooth để giao tiếp không dây với máy tính, phục vụ mục đích đánh giá chất lượng điều khiển.

Chương 3. Thiết kế và thực hiện phần cứng

2 module PWM với tổng cộng 16 ngõ ra khả dụng độc lập đáp ứng nhu cầu về số lượng ngõ. Chất lượng xung PWM và giảm đáng kể tiếng ồn (có thể gây ra từ các động cơ BLDC sử dụng xung PWM có tần số nằm trong ngưỡng nghe của tai người)



Hình 3.19 Sơ đồ các khối chức năng của chip TM4C123GHPM

Ngoài ra còn có những tính năng đáng chú ý khác:

- Bộ nhớ on-chip có 256KB flash, 32KB SRAM; ROM nội bộ có thể sử dụng phần mềm hỗ trợ Tivaware™ và 2KB EEPROM.
- Hỗ trợ 4 khối SSI, SPI.
- Hỗ trợ 12 ngõ vào tương tự với hai bộ chuyển đổi ADC 12 bit với tốc độ lấy mẫu lên đến 10^6 mẫu/giây.
- Hỗ trợ 2 module QEI hỗ trợ đọc xung encoder x4 và đảo giá trị bằng firmware.
- Hỗ trợ tối đa 43 GPIO mức cao +3.3V, điện trở kéo lên/xuống và dòng ra chân, đều có khả năng lập trình ngắn cạnh và ngắn mức.
- Ngoài clock chính còn nhiều nguồn cấp xung cho hệ thống như thạch anh 32.768 kHz, 30kHz; vòng khóa pha (PLL) 200MHz...

3.3.3. Dòng AVR

Giới thiệu

AVR là họ vi điều khiển 8 bit theo công nghệ mới, với những tính năng rất mạnh được tích hợp trong chip của hãng Atmel theo công nghệ RISC, nó mạnh ngang hàng với các họ vi điều khiển 8 bit khác như PIC, Pisoc. Do ra đời muộn hơn nên họ vi điều khiển AVR có nhiều tính năng mới đáp ứng tối đa nhu cầu của người sử dụng, so với họ 8051 89xx sẽ có độ ổn định, khả năng tích hợp, sự mềm dẻo trong việc lập trình và rất tiện lợi.

Tính năng mới của họ AVR:

- Giao diện SPI đồng bộ.
- Các đường dẫn vào/ra (I/O) lập trình được.
- Giao tiếp I2C.
- Bộ biến đổi ADC 10 bit.
- Các kênh băm xung PWM.
- Các chế độ tiết kiệm năng lượng như sleep, stand by..vv.
- Một bộ định thời Watchdog.
- 3 bộ Timer/Counter 8 bit.
- 1 bộ Timer/Counter 16 bit.
- 1 bộ so sánh analog.
- Bộ nhớ EEPROM.
- Giao tiếp USART...

Vi điều khiển ATmega328:

ATmega328 là một chíp vi điều khiển được sản xuất bởi hãng Atmel thuộc họ MegaAVR. ATmega328 là một bộ vi điều khiển 8 bit dựa trên kiến trúc RISC bộ nhớ chương trình 32KB ISP flash có thể ghi xóa hàng nghìn lần, 1KB EEPROM, một bộ nhớ RAM vô cùng lớn trong thế giới vi xử lý 8 bit (2KB SRAM) Với các thông số cụ thể như sau:

Chương 3. Thiết kế và thực hiện phần cứng

Bảng 2 Thông số vi điều khiển ATmega328

Ví điều khiển	ATmega328 họ 8bit
Điện áp hoạt động	5V DC (chỉ được cấp qua cổng USB)
Tần số hoạt động	16 MHz
Dòng tiêu thụ	khoảng 30mA
Điện áp vào khuyên dùng	7-12V DC
Điện áp vào giới hạn	6-20V DC
Số chân Digital I/O	14 (6 chân hardware PWM)
Số chân Analog	6 (độ phân giải 10bit)
Dòng tối đa trên mỗi chân I/O	30 mA
Dòng ra tối đa (5V)	500 mA
Dòng ra tối đa (3.3V)	50 mA
Bộ nhớ flash	32 KB
SRAM	2 KB
EEPROM	1 KB

3.4. MODULE WIFI:

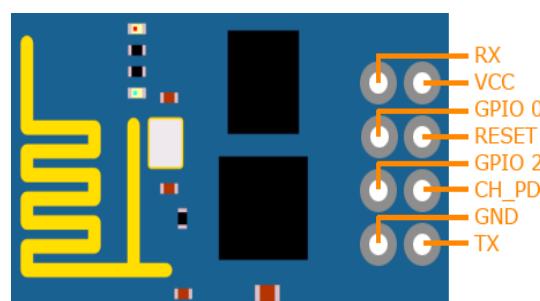
ESP8266 là một chip tích hợp cao - System on Chip (SoC), có khả năng xử lý và lưu trữ tốt, cung cấp khả năng vượt trội để trang bị thêm tính năng wifi cho các hệ thống khác hoặc đóng vai trò như một giải pháp độc lập. Mỗi module ESP8266 đi kèm được lập trình sẵn với một lệnh AT firmware bộ. Các module ESP8266 có chi phí không cao và một cộng đồng sử dụng khá nhiều nên có nhiều ứng dụng phát triển.

Thông số kỹ thuật

- Wifi 802.11 b/g/n
- Wifi 2.4 GHz, hỗ trợ WPA/WPA2
- Chuẩn điện áp hoạt động 3.3V
- Chuẩn giao tiếp nối tiếp UART với tốc độ Baud lên đến 115200
- Có 3 chế độ hoạt động: Client, Access Point, Both Client and Access Point

Chương 3. Thiết kế và thực hiện phần cứng

- Hỗ trợ các chuẩn bảo mật như: OPEN, WEP, WPA_PSK, WPA2_PSK, WPA_WPA2_PSK
- Hỗ trợ cả 2 giao tiếp TCP và UDP
- Tích hợp công suất thấp 32-bit CPU có thể được sử dụng như là bộ vi xử lý ứng dụng
- SDIO 1.1 / 2.0, SPI, UART
- Làm việc như các máy chủ có thể kết nối với 5 máy con



Hình 3.20 Sơ đồ chân module WiFi ESP8266

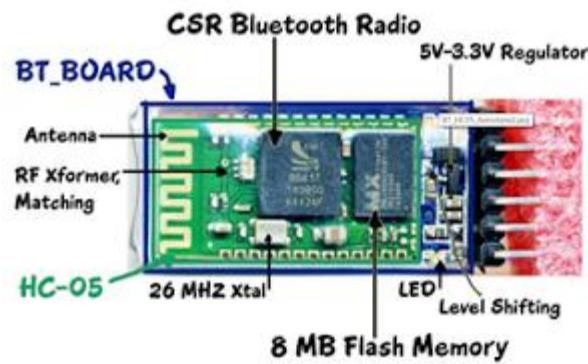
- UTXD(RX): dùng để nhận tín hiệu trong giao tiếp UART với vi điều khiển
- VCC: đầu vào 3.3V
- GPIO 0: kéo xuống thấp cho chế độ upload bootloader
- RST: chân reset cứng của module, kéo xuống mass để reset
- GPIO 2: thường được dùng như một cổng TX trong giao tiếp UART để debug lỗi
- CH_PD: kích hoạt chip, sử dụng cho Flash Boot và updating lại module, nối với mức cao
- GND: nối với mass
- UTXD (TX: dùng để truyền tín hiệu trong giao tiếp UART với vi điều khiển.

Tập lệnh AT cho ESP8266: Tham khảo phụ lục

3.5. MODULE BLUETOOTH HC-05:

Bluetooth là chuẩn truyền thông không dây để trao đổi dữ liệu ở khoảng cách ngắn. Chuẩn truyền thông này sử dụng sóng radio ngắn(UHF radio) trong dải tần số ISM (2.4 tới 2.485 GHz). Khoảng cách truyền của module này vào khoảng 10m.

Module này được thiết kế dựa trên chip BC417. Chip BC417 này khá phức tạp và sử dụng bộ nhớ flash ngoài 8Mbit. Nhưng việc sử dụng module này hoàn toàn đơn giản bởi nhà sản xuất đã tích hợp mọi thứ cho bạn trên module HC-05.



Hình 3.21 Module Bluetooth HC05

Sơ đồ chân HC-05 gồm có:

- KEY: Chân này để chọn chế độ hoạt động AT Mode hoặc Data Mode. VCC chân này có thể cấp nguồn từ 3.6V đến 6V bên trong module đã có một ic nguồn chuyển về điện áp 3.3V và cấp cho IC BC417.
- GND nối với chân nguồn GND
- TxD, RxD đây là hai chân UART để giao tiếp module hoạt động ở mức logic 3.3V
- STATE Chân này không kết nối.

Cơ chế hoạt động:

Chương 3. Thiết kế và thực hiện phần cứng

HC-05 có hai chế độ hoạt động là Command Mode và Data Mode. Ở chế độ Command Mode ta có thể giao tiếp với module thông qua cổng serial trên module bằng tập lệnh AT quen thuộc. Ở chế độ Data Mode module có thể truyền nhận dữ liệu tới module bluetooth khác. Chân KEY dùng để chuyển đổi qua lại giữa hai chế độ này. Có hai cách để bạn có thể chuyển module hoạt động trong chế độ Data Mode.

Nếu đưa chân này lên mức logic cao trước khi cấp nguồn module sẽ đưa vào chế độ Command Mode với baudrate mặc định 38400. Chế độ này khá hữu ích khi bạn không biết baudrate trong module được thiết lập ở tốc độ bao nhiêu. Khi chuyển sang chế độ này đèn led trên module sẽ nháy chậm (khoảng 2s) và ngược lại khi chân KEY nối với mức logic thấp trước khi cấp nguồn module sẽ hoạt động chế độ Data Mode.

Nếu module đang hoạt động ở chế Data Mode để có thể đưa module vào hoạt động ở chế độ Command Mode bạn đưa chân KEY lên mức cao. Lúc này module sẽ vào chế độ Command Mode nhưng với tốc độ Baud Rate được bạn thiết lập lần cuối cùng. Vì thế bạn phải biết baudrate hiện tại của thiết bị để có thể tương tác được với nó. Chú ý nếu module của bạn chưa thiết lập lại lần nào thì mặc định của nó như sau:

- Baudrate 9600, data 8 bits, stop bits 1, parity : none, handshake: none
- Passkey: 1234
- Device Name: HC-05

Ở chế độ Data Mode HC-05 có thể hoạt động như một master hoặc slave tùy vào việc bạn cấu hình (riêng HC-06 bạn chỉ có thể cấu hình ở chế độ SLAVE)

Ở chế độ SLAVE: bạn cần thiết lập kết nối từ smartphone, laptop, usb, bluetooth để dò tìm module sau đó pair với mã PIN là 1234. Sau khi pair thành công, bạn đã có 1 cổng serial từ xa hoạt động ở baud rate 9600.

Tập lệnh AT cho Module HC05: Tham khảo phụ lục

Ở chế độ MASTER: module sẽ tự động dò tìm thiết bị bluetooth khác (1 module bluetooth HC-06, usb bluetooth, bluetooth của laptop...) và tiến hành pair chủ động mà không cần thiết lập gì từ máy tính hoặc smartphone.

Chương 3. Thiết kế và thực hiện phần cứng

3.6. IC RS485 – SN75176B:

SN75176B là vi mạch tích hợp được thiết kế cho việc truyền nhận dữ liệu vi sai trong các mạng nhiều trạm, đáp ứng tiêu chuẩn ANSI TIA/EIA-485.

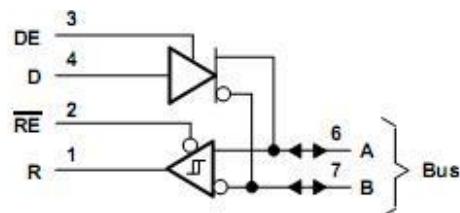
SN75176B kết hợp một mạch lái vi sai 3 trạng thái và mạch thu vi sai hoạt động ở điện áp 5V. Bộ thu và phát có các ngõ điều khiển tích cực thấp và cao, thường được nối với nhau để dễ dàng điều khiển hướng truyền nhận. Tổng trở ngõ vào là $12k\Omega$, độ nhạy ngõ vào là 200mV.

DRIVER			
INPUT D	ENABLE DE	OUTPUTS	
		A	B
H	H	H	L
L	H	L	H
X	L	Z	Z

RECEIVER		
DIFFERENTIAL INPUTS A-B	ENABLE RE	OUTPUT R
$V_{ID} \geq 0.2 V$	L	H
$-0.2 V < V_{ID} < 0.2 V$	L	?
$V_{ID} \leq -0.2 V$	L	L
X	H	Z
Open	L	?

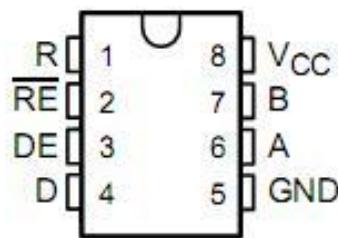
H = high level, L = low level, ? = indeterminate,
X = irrelevant, Z = high impedance (off)

Hình 3.22 Bảng trạng thái ngõ vào ra các bộ thu phát của SN75176B



Hình 3.23 Sơ đồ cấu trúc logic SN75176B

Chương 3. Thiết kế và thực hiện phần cứng



Hình 3.24 *Bố trí các chân của SN75176B*

Mô tả các chân:

- Chân 1 (R): ngõ ra của bộ thu theo mức logic TTL, khi $V_A - V_B > 200mV$ thì $R=1$, ngược lại khi $V_B - V_A > 200mV$ thì $R=0$.
- Chân 2 (): ngõ vào điều khiển bộ thu, tích cực thấp.
- Chân 3 (DE): ngõ vào điều khiển bộ phát, tích cực cao.
- Chân 4 (D): ngõ vào bộ phát theo mức logic TTL, khi $D=1$ thì $V_A=1$ và $V_B=0$, khi $D=0$ thì $V_A=0$ và $V_B=1$.
- Chân 5 (GND): nối mass nguồn.
- Chân 6 (A): chân tín hiệu A
- Chân 7 (B): chân tín hiệu B
- Chân 8 (Vcc): nguồn cung cấp cho IC, tối đa là 7VDC.

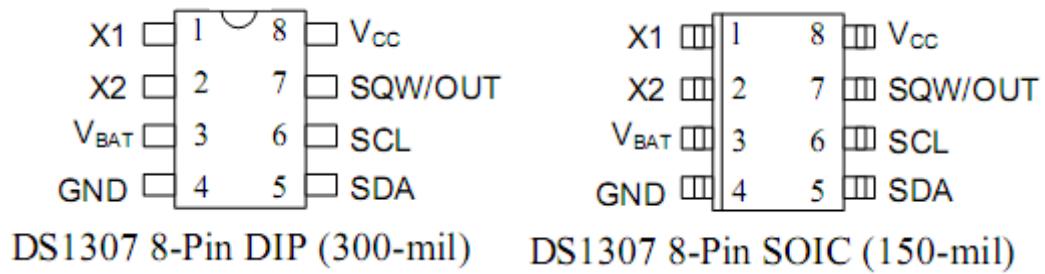
3.7. MODULE THỜI GIAN THỰC



Hình 3.25 *Module thời gian thực DS1307*

Chương 3. Thiết kế và thực hiện phần cứng

DS1307 là chip đồng hồ thời gian thực (RTC : Real-time clock), khái niệm thời gian thực ở đây được dùng với ý nghĩa thời gian tuyệt đối mà con người đang sử dụng, tình bằng giây, phút, giờ. Chip này có 7 thanh ghi 8-bit chứa thời gian là: giây, phút, giờ, thứ (trong tuần), ngày, tháng, năm. Ngoài ra DS1307 còn có 1 thanh ghi điều khiển ngõ ra phụ và 56 thanh ghi trống có thể dùng như RAM. DS1307 được đọc và ghi thông qua giao diện nối tiếp nên cấu tạo bên ngoài rất đơn giản. DS1307 xuất hiện ở 2 gói SOIC và DIP có 8 chân như trong hình 1.



Hình 3.26 Hai gói cấu tạo chip DS1307.

Các chân của DS1307 được mô tả như sau:

- X1 và X2: là 2 ngõ kết nối với 1 thạch anh 32.768KHz làm nguồn tạo dao động cho chip.
- V_{BAT}: cực dương của một nguồn pin 3V nuôi chip.
- GND: chân mass chung cho cả pin 3V và Vcc.
- Vcc: nguồn cho giao diện I2C, thường là 5V và dùng chung với vi điều khiển.
- Vcc không được cấp nguồn nhưng V_{BAT} được cấp thì DS1307 vẫn đang hoạt động (nhưng không ghi và đọc được).
- SQW/OUT: một ngõ phụ
- SCL và SDA là 2 đường giao xung nhịp và dữ liệu của giao diện I2C mà chúng ta đã tìm hiểu trong bài TWI của AVR.

Chương 3. Thiết kế và thực hiện phần cứng

Bảng 3 Bảng mô tả thanh ghi của module RTC

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	FUNCTION	RANGE			
00h	CH	10 Seconds				Seconds				Seconds 00-59			
01h	0	10 Minutes				Minutes				Minutes 00-59			
02h	0	12	10 Hour	10 Hour	Hours				Hours	1-12 +AM/PM 00-23			
		24	PM/ AM										
03h	0	0	0	0	0	DAY			Day	01-07			
04h	0	0	10 Date		Date				Date	01-31			
05h	0	0	0	10 Month	Month				Month	01-12			
06h	10 Year				Year				Year	00-99			
07h	OUT	0	0	SQWE	0	0	RS1	RS0	Control	—			
08h-3Fh									RAM 56 x 8	00h-FFh			

Tổ chức các thanh ghi thời gian:

Điều đầu tiên cần chú ý là giá trị thời gian lưu trong các thanh ghi theo dạng BCD. BCD là viết tắt của cụm từ Binary-Coded Decimal, tạm dịch là các số thập phân theo mã nhị phân. Tất cả các phần mềm lập trình hay thanh ghi của chip điều khiển đều sử dụng mã nhị phân thông thường, không phải mã BCD, do đó chúng ta cần viết các chương trình con để quy đổi từ số thập nhị phân (hoặc thập phân thường) sang BCD, phần này sẽ được trình bày trong lúc lập trình giao tiếp với DS1307. Thoạt nhìn, mọi người đều cho rằng số BCD chỉ làm vẩn đền thêm rắc rối, Tuy nhiên số BCD rất có ưu điểm trong việc hiển thị nhất là khi hiển thị từng chữ số như hiển thị bằng LED 7 đoạn chẳng hạn.

Thanh ghi giây (SECONDS): thanh ghi này là thanh ghi đầu tiên trong bộ nhớ của DS1307, địa chỉ của nó là 0x00. Bốn bit thấp của thanh ghi này chứa mã BCD 4-bit của chữ số hàng đơn vị của giá trị giây. Do giá trị cao nhất của chữ số hàng chục là 5 (không có giây 60) nên chỉ cần 3 bit (các bit SECONDS 6:4) là có thể mã hóa được ($5 = 101_2$, 3 bit). Bit cao nhất, bit 7, trong thanh ghi này là 1 điều khiển có tên CH (Clock halt – treo đồng hồ), nếu bit này được set bằng 1 bộ dao động trong chip bị vô hiệu hóa, đồng hồ không hoạt động. Vì vậy, nhất thiết phải reset bit này xuống 0 ngay từ đầu.

Chương 3. Thiết kế và thực hiện phần cứng

Thanh ghi phút (MINUTES): có địa chỉ 0x01, chứa giá trị phút của đồng hồ. Tương tự thanh ghi SECONDS, chỉ có 7 bit của thanh ghi này được dùng lưu mã BCD của phút, bit 7 luôn luôn bằng 0.

Thanh ghi giờ (HOURS): có thể nói đây là thanh ghi phức tạp nhất trong DS1307. Thanh ghi này có địa chỉ 0x02. Trước hết 4-bits thấp của thanh ghi này được dùng cho chữ số hàng đơn vị của giờ. Do DS1307 hỗ trợ 2 loại hệ thống hiển thị giờ (gọi là mode) là 12h (1h đến 12h) và 24h (1h đến 24h) giờ, bit6 (màu xanh lá cây trong hình 4) xác lập hệ thống giờ. Nếu bit6=0 thì hệ thống 24h được chọn, khi đó 2 bit cao 5 và 4 dùng mã hóa chữ số hàng chục của giá trị giờ. Do giá trị lớn nhất của chữ số hàng chục trong trường hợp này là 2 (=10, nhị phân) nên 2 bit 5 và 4 là đủ để mã hóa. Nếu bit6=1 thì hệ thống 12h được chọn, với trường hợp này chỉ có bit 4 dùng mã hóa chữ số hàng chục của giờ, bit 5 (màu màu cam trong hình 4) chỉ buổi trong ngày, AM hoặc PM. Bit5 =0 là AM và bit5=1 là PM. Bit 7 luôn bằng 0. (thiết kế này hơi dở, nếu đổi 2 bit mode và A-P sang 2 bit 7 và 6 thì sẽ đơn giản hơn).

Thanh ghi thứ (DAY – ngày trong tuần): nằm ở địa chỉ 0x03. Thanh ghi DAY chỉ mang giá trị từ 1 đến 7 tương ứng từ Chủ nhật đến thứ 7 trong 1 tuần. Vì thế, chỉ có 3 bit thấp trong thanh ghi này có nghĩa.

Các thanh ghi còn lại có cấu trúc tương tự, DATE chứa ngày trong tháng (1 đến 31), MONTH chứa tháng (1 đến 12) và YEAR chứa năm (00 đến 99). Chú ý, DS1307 chỉ dùng cho 100 năm, nên giá trị năm chỉ có 2 chữ số, phần đầu của năm do người dùng tự thêm vào (ví dụ 20xx).

3.8. CẢM BIẾN CƯỜNG ĐỘ ÁNH SÁNG

Giới thiệu:

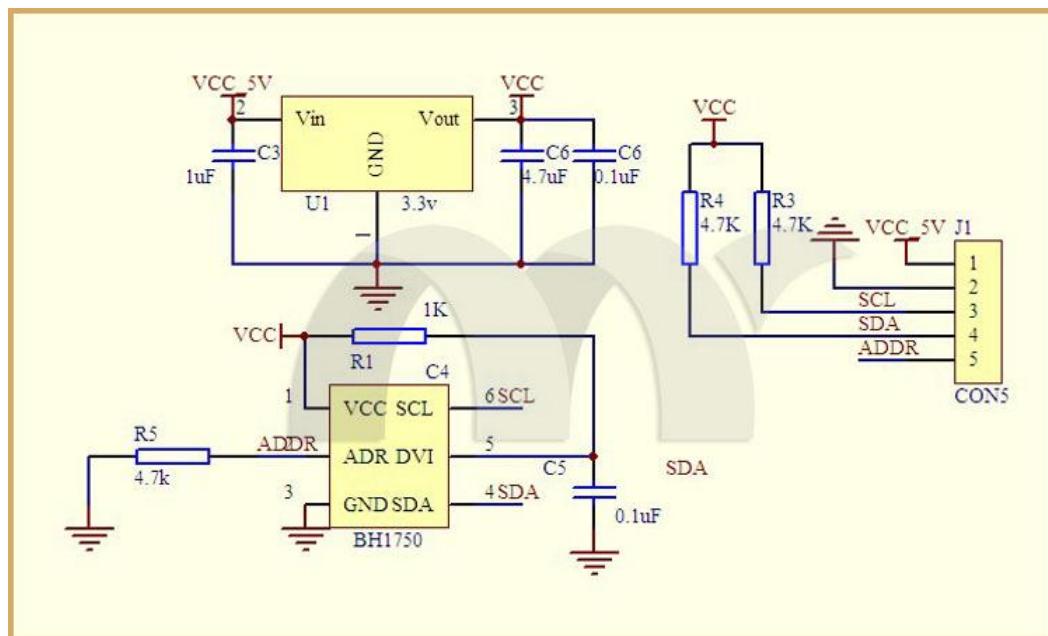
Cảm Biến Cường Độ Ánh Sáng BH1750 là cảm biến ánh sáng với bộ chuyển đổi AD 16 bit tích hợp trong chip và có thể xuất ra trực tiếp dữ liệu theo dạng digital. cảm biến không cần bộ tính toán cường độ ánh sáng khác.BH1750 sử dụng đơn giản và chính xác hơn nhiều lần so với dùng cảm biến quang trở để đo cường độ ánh sáng với dữ liệu thay đổi trên điện áp dẫn đến việc sai số cao.Với cảm biến BH1750 cho dữ liệu đo ra trực tiếp với dạng đơn vị là LUX không cần phải tính toán chuyển đổi thông qua chuẩn truyền I2C.

Cường độ được tính như sau:

Chương 3. Thiết kế và thực hiện phần cứng

- Ban đêm: 0.001 - 0.02 lx.
- Trời sáng trăng: 0.02 - 0.3 lx
- Trời mây trong nhà: 5 - 50 lx.
- Trời mây ngoài trời: 50 - 500 lx.
- Trời nắng trong nhà: 100- 1000 lx.

Schematic:



Hình 3.27 Schematic cảm biến cường độ ánh sáng BH1750

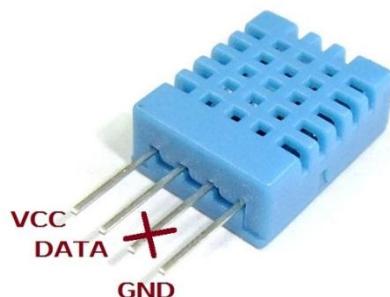
Thông số kĩ thuật:

- Chuẩn kết nối I2C
- Độ phân giải cao(1 - 65535 lx)
- Tiêu hao nguồn ít.
- Khả năng chống nhiễu sáng ở tần số 50 Hz/60 Hz
- Sự biến đổi ánh sáng nhỏ (+/- 20%)
- Độ ảnh hưởng bởi ánh sáng hồng ngoại rất nhỏ
- Nguồn cung cấp : 3.3V-5V
- Kích thước board : 0.85*0.63*0.13"(21*16*3.3mm)

3.9. CẢM BIẾN NHIỆT ĐỘ

Chương 3. Thiết kế và thực hiện phần cứng

Cảm biến nhiệt độ và độ ẩm DHT11 là cảm biến cơ bản và giá rẻ so với các cảm biến khác, rất thích hợp cho những ứng dụng thu thập dữ liệu cơ bản. Cảm biến DHT11 có 2 phần, 1 cảm biến độ ẩm điện dung và một điện trở nhiệt. Dữ liệu ngõ ra của cảm biến DHT là dạng số, có thể dùng bất cứ vi điều khiển nào để lấy dữ liệu ra. Dữ liệu độ ẩm mà cảm biến đo được mức từ 20% ~ 90%. Nhiệt độ đo từ 0 ~ 50 Độ C, thời gian trả dữ liệu < 50ms.



Hình 3.28 *Cảm biến nhiệt độ DTH11*

Sơ đồ chân:

- Pin 1 : Vcc
- Pin 2 : Data
- Pin 3 : NC (not conected). Chân này không sử dụng
- Pin 4 : GND

Thông số kỹ thuật:

- Đo độ ẩm: 20%-95%
- Nhiệt độ: 0-50°C
- Sai số độ ẩm $\pm 5\%$
- Sai số nhiệt độ: $\pm 2^\circ\text{C}$

3.10. IC NGUỒN TUYẾN TÍNH

3.10.1. 7805

IC 7805 là IC hạ áp 12V- 5V được sử dụng nhiều trong các mạch yêu cầu dòng nhỏ. Giá rẻ và dễ sử dụng.

Thông số kỹ thuật:

Chương 3. Thiết kế và thực hiện phần cứng

- Điện áp vào lớn nhất: 20V
- Điện áp vào nhỏ nhất: 7V
- Kiểu đóng vỏ: TO-220
- Nhiệt độ hoạt động lớn nhất: 85°C
- Nhiệt độ hoạt động nhỏ nhất: -20°C
- Dòng đầu ra: 1.5A
- Điện áp ổn định: 5V

3.10.2. LM1117

IC AMS1117 là IC có khả năng hạ áp từ đầu vào 5V trở thành nguồn 3.3V cung cấp dòng lên đến 1A. Dòng IC này được tối ưu hóa cho các thiết bị có điện áp thấp, các mạch cần nguồn có dòng ổn định và kích thước nhỏ.

Các tính năng chính

- Điện áp bỏ học thấp
- Load quy định: 0,2% điển hình
- Tối ưu hóa cho điện áp thấp
- On-chip nhiệt hạn chế
- Tiêu chuẩn SOT-223 gói
- Ba-terminal điều tiết 1.2V bỏ học thấp cố định

3.11. RELAY

Rơ le (relay) là một công tắc chuyển đổi hoạt động bằng điện. Nói là một công tắc vì rơ le có 2 trạng thái ON và OFF. Rơ le ở trạng thái ON hay OFF phụ thuộc vào có dòng điện chạy qua rơ le hay không.



Hình 3.29 *Hình ảnh relay*

Chương 3. Thiết kế và thực hiện phần cứng

Nguyên tắc hoạt động:

Khi có dòng điện chạy qua rơ le, dòng điện này sẽ chạy qua cuộn dây bên trong và tạo ra một từ trường hút. Từ trường hút này tác động lên một đòn bẩy bên trong làm đóng hoặc mở các tiếp điểm điện và như thế sẽ làm thay đổi trạng thái của rơ le. Số tiếp điểm điện bị thay đổi có thể là 1 hoặc nhiều, tùy vào thiết kế.

Rơ le có 2 mạch độc lập nhau hoạt động. Một mạch là để điều khiển cuộn dây của rơ le: Cho dòng chạy qua cuộn dây hay không, hay có nghĩa là điều khiển rơ le ở trạng thái ON hay OFF. Một mạch điều khiển dòng điện ta cần kiểm soát có qua được rơ le hay không dựa vào trạng thái ON hay OFF của rơ le.

Dòng chạy qua cuộn dây để điều khiển rơ le ON hay OFF thường vào khoảng 30mA với điện áp 12V, 5V hoặc có thể lên tới 100mA. Hầu hết các con chip đều không thể cung cấp dòng này, lúc này ta cần có một BJT để khuếch đại dòng nhỏ ở ngõ ra IC thành dòng lớn hơn phục vụ cho rơ le.

Trên rơ le có 3 kí hiệu là: NO, NC và COM.

- COM (common): là chân chung, nó luôn được kết nối với 1 trong 2 chân còn lại. Còn việc nó kết nối chung với chân nào thì phụ thuộc vào trạng thái hoạt động của rơ le.
- NC (Normally Closed): Nghĩa là bình thường nó đóng. Nghĩa là khi rơ le ở trạng thái OFF, chân COM sẽ nối với chân này.
- NO (Normally Open): Khi rơ le ở trạng thái ON (có dòng chạy qua cuộn dây) thì chân COM sẽ được nối với chân này.

3.12. CÔNG TẮC BÁN DẪN

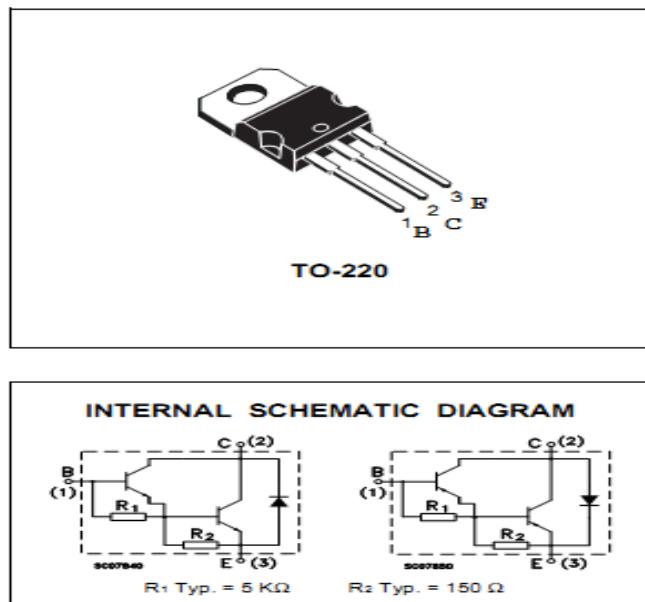
Transistor TIP122 là darlington transistor thuộc loại transistor NPN. Darlington transistor TIP122 là một sự sắp xếp đặc biệt của hai transistor lưỡng cực NPN kết nối với nhau.

Thông số kỹ thuật:

- Model: NPN – TO220
- Điện áp cực đại: $VCBO = 100V$
 $VCEO = 100V$
 $VEBO = 5V$

Chương 3. Thiết kế và thực hiện phần cứng

- Dòng điện cực đại: IC = 5A
IB = 120mA
- Công suất tối đa: PC = 65 W
- Nhiệt độ làm việc: -65oC ~ 150oC



Hình 3.30 Sơ đồ chân và cấu tạo TIP122

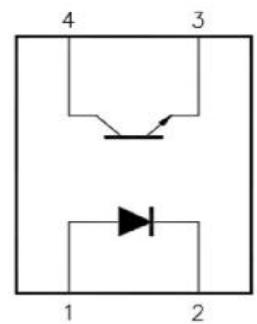
Transistor TIP122 có độ nhạy cảm lớn hơn, lợi dòng lớn nhiều hơn và rất hữu ích trong các ứng dụng khuếch đại dòng điện hoặc chuyển mạch. TIP122 có U_c cực đại = 100V dòng I_c cực đại = 5A, công suất cực đại 65W. Hệ số khuếch đại h_{FE} của darlington transistor TIP122 thấp nhất là 1000.

3.13. OPTO PC817

PC 817 cũng là dạng opto nó hoạt động tương tự như các opto khác.

Khi cung cấp 5V vào chân số 1, LED phía trong Opto nối giữa chân số 1 và 2 sáng, xảy ra hiệu ứng quang điện dẫn đến 3-4 thông, mức logic sẽ bị chuyển từ 1 sang 0 mà không cần tác động trực tiếp từ IC

Chương 3. Thiết kế và thực hiện phần cứng



1. Anode 3. Emitter
2. Cathode 4. Collector

Hình 3.31 Sơ đồ chân Opto PC817

Chương 4. GIẢI THUẬT VÀ XÂY DỰNG PHẦN MỀM

4.1. THIẾT KẾ I-HOME

4.1.1. Tính năng kỹ thuật

Trong việc thiết kế mô hình I-home. Việc đáp ứng được các tính năng kỹ thuật được đặt ra trước khi thực hiện là hết sức quan trọng. Trong phạm vi đề tài luận, mô hình được thi công có thể đáp ứng được để ứng dụng trong 1 phòng khách và 1 phòng ngủ cỡ trung ở Việt Nam. Cụ thể các tính năng như sau:

Khả năng đóng- ngắt tiện dụng:

Phòng khách:

Sẽ cung cấp 8 mạch relay để có thể đóng – ngắt cắt thiết bị điện AC. Với các chế độ điều khiển tiện dụng như công tắc 2 trạng thái, điều khiển trực tiếp tại nhà thông qua chuẩn kết nối Bluetooth sử dụng Smart-phone, bên cạnh đó là chế độ điều khiển qua Internet khi chủ nhân không có ở nhà. Điều này sẽ vô cùng tiện lợi trong trường hợp chúng ta muốn máy lạnh bật sẵn để nhiệt độ phòng đủ lạnh trước khi về nhà. Bên cạnh đó việc kết hợp thiết bị đóng- ngắt với các hoạt cảnh được định sẵn : Như chế độ tiếp khách, chế độ không cần nhiều ánh sáng,...



Hình 4.1 Phòng khách thông minh tiện dụng

Phòng ngủ:

Phòng ngủ trong mô hình thi công được cung cấp 4 relay 5V dùng để đóng ngắt các thiết bị điện AC. Với các tính năng tiện dụng như đã đề cập ở phòng khách.

Các tính năng hữu ích khác của ngôi nhà:

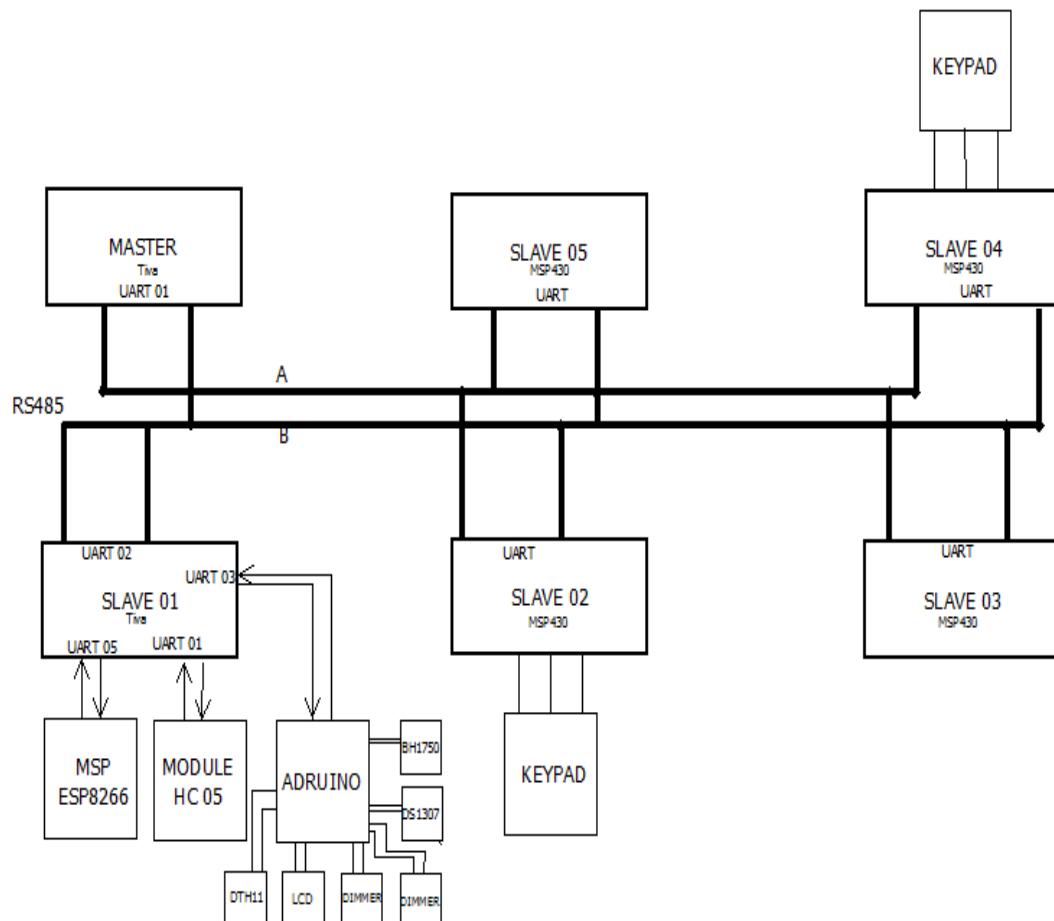
Tính năng hẹn giờ bật tắt thiết bị. Ngôi nhà có thể được hẹn giờ để bật tắt thiết bị theo nhu cầu của chủ nhân.

Trang bị cảm biến nhiệt độ và độ ẩm giúp chúng ta biết được trạng thái hiện tại của ngôi nhà.

Trang bị cảm biến cường độ ánh sáng giúp ta có thể biết được mức độ ánh sáng cần cung cấp phù hợp với từng nhu cầu tại thời điểm sử dụng. Ngoài ra cảm biến cường độ ánh sáng sẽ có tác dụng đưa tín hiệu để đóng- ngắt các thiết bị đèn khi không cần thiết, bên cạnh ngôi nhà có trang bị 2 mạch để có thể điều khiển độ sáng bóng đèn LED-12V để từ đó có thể thực hiện tính năng tiết kiệm điện.

Các yếu tố trên góp phần nâng cao chất lượng cuộc sống của chủ nhân ngôi nhà.

4.1.2. Sơ đồ khái



Hình 4.2 Sơ đồ khái niệm nhà thông minh

Mô tả sơ đồ khối: Sơ đồ khối gồm 6 khối chức năng chính và 4 khối chức năng phụ.

Master:

Gồm 1 Kit Tiva Launchpad của nhà sản xuất TI kết nối với mạch chức năng gồm khối nguồn và khối giao tiếp RS485.

Master có nhiệm vụ quản lý các Slave con, nhận dữ liệu từ các Slave sau đó gửi thông tin truyền thông ứng với các nhiệm vụ được giao. Ngoài ra Master được kết với máy tính thông qua cổng USB để hiển thị thông tin truyền lên máy tính.

Slave Input Button(#02,#04)

Slave Input Button(#01,#04) có chức năng và thiết kế tương tự nhau. Đều sử dụng vi điều khiển MSP430G2553g của nhà sản xuất TI được cung cấp nguồn 3.3V từ khối nguồn và giao tiếp với đường truyền chung thông qua chuẩn giao tiếp thông qua khối giao tiếp RS485

Slave Input Button(#01,#04) sẽ có nhiệm vụ chính là nhận thông tin từ Keypad được kết nối thông qua 6 Bus từ đó phân tích và xử lý thông tin từ đó gửi frame tín hiệu lên đường truyền đến Master để thực hiện các chức năng được quy định sẵn. Ngoài ra Slave 02 phải gửi thông tin đăng kí đến Master để chuyển sang trạng thái hoạt động.

Slave Output Button(#03,#05)

Slave Output Button(#03,#05) sử dụng vi điều khiển MSP430G2553 của TI được cung cấp nguồn 3.3V từ khối Nguồn và giao tiếp với đường truyền chung thông qua chuẩn giao tiếp thông qua khối giao tiếp RS485, bên cạnh đó Slave 03 có 8 khối Relay 5V sử dụng để đóng ngắt, trong khi đó ở Slave 05 có 8 khối Ngoài ra Slave 02 phải gửi đăng kí đến Master để chuyển sang trạng thái hoạt động.

Slave Output Button(#03,#05) có nhiệm vụ nhận dữ liệu từ đường truyền sau đó phân tích và xử lý thông tin từ đó cấp tín hiệu đến mạch Relay để đóng ngắt các thiết bị 220V. Ngoài ra Slave 03 phải gửi thông tin đăng kí đến Master để chuyển sang trạng thái chờ hoạt động.

Slave đa chức năng (#01)

Slave đa chức năng (#01) gồm 1 Kit Tiva Launchpad của nhà sản xuất TI kết nối với mạch chức năng gồm khối nguồn và khối giao tiếp RS485, bên cạnh đó Slave đa chức năng (#01) được liên kết với module Wifi HC05, mạch MSP-ESP8266 để kết nối đến Wifi và mạch Adruino hiển thị, cảm biến, dimmer đèn LED.

Chương 4 : Giải thuật và xây dựng phần mềm

Chức năng chủ yếu của Slave đa chức năng (#01) là nhận dữ liệu từ các mạch kết nối, sau đó phân tích, xử lý và gửi fram truyền lên đường truyền. Slave đa chức năng (#01) được kết nối với các mạch có chức năng như sau:

- MSP-ESP8266:

- Mạch chức năng MSP-ESP8266 sử dụng chip sử dụng vi điều khiển MSP430G2553 của TI được cung cấp nguồn 3.3V từ khối Nguồn, giao tiếp với module wifi ESP8266 thông qua giao tiếp Uart.
- Module wifi ESP8266 có nhiệm vụ kết nối đến mạng Wifi sử dụng trong nhà để từ đó kết nối đến mạng Internet thông qua IP. Thông tin nhận được từ Internet(cụ thể là qua App được cài đặt trên PC và App trên điện thoại Adruino) sẽ được chuyển đến vi điều khiển MSP430G2553 sau đó sẽ truyền thông tin qua cổng UART đến Slave 01.
 - Mạch đa chức năng - Thời gian thực, cảm biến ánh sáng, cảm biến nhiệt độ và hiển thị LCD:
- Khối đa chức năng gồm Kit lập trình Adruino Uno R3 sử dụng dòng vi điều khiển ATmega328, module thời gian thực DS1307, module cảm biến ánh sáng BH1750, cảm biến nhiệt độ DTH131 và màn hình hiển thị LCD16x2. Nguồn 12V sẽ cấp trực tiếp đến Kit lập trình Adruino Uno R3 từ đó sẽ ra nguồn 5V cung cấp cho các thiết bị. Ngoài ra mạch còn kết nối với 2 mạch Khóa điện tử sử dụng MOSFET TIP122 thông qua 2 dây tín hiệu PWM.
- Điện áp 12V cấp nguồn cho Kit lập trình Adruino Uno R3. Kit sẽ cung cấp nguồn 5V để cho các thiết bị: Module thời gian thực DS1307, cảm biến ánh sáng, cảm biến nhiệt độ và màn hình LCD hoạt động. Module thời gian thực DS1307, cảm biến ánh sáng và cảm biến nhiệt độ sẽ kết nối với kit Adruino Uno R3 thông qua chuẩn giao tiếp I2C để liên tục đọc các thông tin từ môi trường từ đó Kit Adruino sẽ xử lý và xuất đến màn hình LCD thông tin về nhiệt độ, độ ẩm, cường độ ánh sáng, thời gian thực hiện tại. Từ thông tin thời gian thực ta có thể lập trình để hẹn giờ sau đó gửi thông tin đến Slave 01 để đóng mở thiết bị theo thời gian đã đặt từ trước. Bên cạnh đó, với mức cường độ ánh sáng đọc từ cảm biến cường độ ánh sáng BH1750 , Kit Adruino sẽ cấp xung PWM để điều chỉnh điện áp đầu ra của mạch khóa điều chỉnh BJT từ đó thay đổi độ sáng bóng đèn LED DC 12V.

Keypad: Là một ma trận phím 3x3 hoặc 4x4 có nhiệm vụ dùng để nhập dữ liệu đầu vào (Giống như một công tắc điện)

4.1.3. Giao thức truyền thông

Lý thuyết về mô hình truyền:

Trong mô hình I-home được thiết kế trong luận văn. Việc truyền dẫn dữ liệu giữa các thiết bị bằng cách thông qua các cổng UART của vi điều khiển, từ đó tín hiệu được đưa đến IC RS485 và đưa lên đường truyền. Như vậy, chuẩn giao tiếp RS485 sử dụng duy nhất 1 cáp mạng sẽ kết nối đến tất cả thiết bị trong mô hình I-home. Trong đó cáp mạng sẽ gồm 4 dây bao gồm:

- Dây 12V- có chức năng đưa nguồn điện đến các mạch điều khiển
- Dây A- Dây truyền tín hiệu
- Dây B- Dây truyền tín hiệu
- Dây GND- Kết nối đất chung giữa các mạch điều khiển.

Tuy nhiên việc truyền dẫn sử dụng chuẩn giao tiếp RS485 chỉ mang tính truyền dẫn vật lý vì vậy đòi hỏi một Giao Thức Truyền dẫn để có thể quản lý dữ liệu một cách khoa học từ đó phát triển mô hình thêm nhiều thiết bị hơn. Ngoài ra, để IC RS485 hoạt động ta cần điều khiển trực tiếp chân cho phép hoạt động của IC RS485 bằng vi xử lý. Chính vì vậy việc thiết kế và xây dựng một giao thức truyền là vô cùng cần thiết trong việc thiết kế mô hình I-home.

Trong quá trình tìm hiểu và nghiên cứu để thiết kế một giao thức truyền phù hợp với mô hình luận văn. Em có tìm hiểu và đưa ra hai mô hình truyền dẫn dựa trên các mô hình trên lý thuyết như sau:

Thiết kế giao thức theo hình thức hỏi vòng tuần tự:

- Master:

Nhiệm vụ quan trọng nhất trong giao thức truyền. Master có nhiệm vụ hỏi vòng tuần tự các Slave được kết nối trong hệ thống. Sau khi gửi một thông tin hỏi để yêu cầu làm việc đến một Slave, Master sẽ Delay một thời gian nhỏ để chờ thông tin từ Slave đang được hỏi. Nếu như Slave đang được hỏi không có yêu cầu làm việc gì thì Master sẽ bỏ qua và tiến hành hỏi một Slave khác. Nếu một Slave Input đang được Master hỏi và có tín hiệu từ bên ngoài. Slave này sẽ được quyền ngắt vòng hỏi của Master bằng cách gửi một thông tin đến cho Master để yêu cầu quyền làm việc,

sau đó thông tin nhận được sẽ được ưu tiên truyền dữ liệu trực tiếp đến Slave Output để thực hiện công việc được yêu cầu. Kết thúc quá trình kết nối giữa Master- Slave Input- Slave Output, Master sẽ tiếp tục hỏi tuần tự các Slave sau đó lặp lại chu kỳ hỏi lặp. Thời gian hỏi lặp của Master phải rất nhỏ(tính bằng ms) để có thể hỏi tất cả các Slave đang hiện có của hệ thống. Ngoài ra, thời gian hỏi và thời gian nhận tín hiệu từ Slave phải trùng nhau để tín hiệu có thể được gửi đi.

- Slave Input:

Có nhiệm vụ chờ được Master hỏi và gửi thông tin lên trên đường truyền. Slave Input luôn ở trong trạng thái nhận dữ liệu(tức chân Unable của RS485 luôn ở trạng thái LOW). Khi đồng thời được hỏi và có tín hiệu từ bên ngoài Slave Input sẽ bật trạng thái Unable của RS485 lên HIGH và gửi thông tin theo thông tin được yêu cầu từ bên ngoài.

- Slave Output:

Chân Unable của RS485 của Slave Ouput luôn ở trạng thái LOW (tức trạng thái chờ làm việc) Khi được Master hỏi hoặc có dữ liệu gửi từ một Slave Input. Slave Ouput có nhiệm vụ phân tích và xử lý thông tin được truyền đến. Nếu thông tin truyền đúng với cài đặt thì sẽ làm việc theo chức năng định sẵn.

- Phân tích:

Như vậy quá trình hỏi lặp sẽ được diễn ra liên tục. Việc này sẽ hạn chế hiện tượng trùng khi có 2 tín hiệu được gửi cùng một lúc, khi Slave nào được hỏi Slave đó sẽ trả lời. Tuy nhiên sẽ rất khó quản lý dữ liệu vì những nguyên nhân sau đây:

- Về phần cứng: dựa theo mô hình thiết kế 2 chân RI và RO của IC RS485 được kết nối với 2 đèn LED để phát hiện tín hiệu. Nếu hỏi vòng liên tục thì đèn LED của các mạch sẽ nhấp nháy liên tục gây khó chịu cho người sử dụng.
- Về vấn đề thời gian: Sau khi Master gửi thì sẽ phải một khoảng thời gian delay để chờ nhận thông tin truyền từ Slave và chuyển trạng thái RS485 về trạng thái nhận (LOW). Bên cạnh đó là một khoảng thời gian Delay để có thể đặt trạng thái của RS485 lên trạng thái (HIGH) trạng thái gửi. Như vậy khi càng có nhiều Slave thì thời gian Delay sẽ càng lớn lên gây thông tin truyền sẽ trở nên chậm.

- Về vấn đề quản lý: Việc truyền thông tin liên tục như vậy sẽ dẫn đến trên đường truyền sẽ bận liên tục, điều này sẽ dẫn đến phí phạm tài nguyên chưa kể là trường hợp có 2 thông tin truyền cùng lúc sẽ dẫn đến sai lệch thông tin truyền, ngoài ra còn các trường hợp nhiều, tín hiệu truyền liên tục nên các Slave sẽ phải nhận và xử lý thông tin một cách liên tục dẫn đến tình trạng xử lý sai. Mất frame truyền.

Vì nhiều bất cập trong việc sử dụng hỏi vòng lặp nên trong luận văn đã chuyển sang xây dựng một giao thức truyền thông khác dựa trên lý thuyết về giao thức truyền thông CSMA/CD.

Thiết kế giao thức theo mô hình làm việc tuần tự:

Dựa trên lý thuyết về giao thức truyền thông CSMA/CD, nghĩa là đa truy cập nhận biết sóng mang phát hiện xung đột.

Tư tưởng của nó là: khi một máy trạm cần truyền dữ liệu trước hết phải “nghe” xem đường truyền bận hay rỗi. Nếu đường truyền rỗi thì truyền dữ liệu theo khuôn dạng chuẩn. Ngược lại, nếu đường truyền đang bận(đã có máy trạm khác truyền dữ liệu rồi) thì máy trạm đợi một khoảng thời gian ngẫu nhiên rồi bắt đầu nghe lại hoặc tiếp tục nghe cho đến khi đường truyền rỗi thì truyền đi với xác suất $p(0 < p < 1)$.

Để có thể phát hiện xung đột, người ta bổ sung thêm quy tắc “ nghe trong khi nói” tức là trong khi một trạm đang truyền nó vẫn “nghe” đường truyền. Nếu phát hiện xung đột thì nó dừng ngay việc truyền và phát đi sóng mang báo hiệu xung đột cho máy trạm khác.Việc phát hiện xung đột đã khiến cho việc truyền dẫn thông tin trên cáp được đảm bảo hơn.

Việc xây dựng giao thức truyền thông được dựa trên lý thuyết về giao tiếp Master- Slave kết hợp với lý thuyết về lắng nghe đường truyền rỗi cụ thể như sau

- Master:

Nhiệm vụ quan trọng nhất trong giao thức truyền thông. Khi hệ thống được thiết lập Master sẽ được khởi động đầu tiên. Master sẽ ở trạng thái chờ thông tin truyền, khi có một Slave hòa vào mạng giao tiếp thì Master sẽ nhận một Frame đăng ký đến từ Slave sau đó Master sẽ phải hồi lại để cho phép Slave được hoạt động, chỉ khi Slave nhận được Frame cho phép chuyển sang trạng thái hoạt động từ Master thì Slave mới có quyền được hoạt động. Sau khi gửi dữ liệu để các Slave được hoạt động Master sẽ chuyển về trạng thái chờ. Ngoài việc ghi nhận, quản lý các Slave

đăng kí trạng thái hoạt động Master còn hoạt động giống như một trung gian để truyền thông tin giữa các Slave. Khi các frame truyền từ Slave Input yêu cầu các công việc, chỉ duy nhất Master có thể xử lý thông tin được truyền đi đó và từ đó gửi thông tin truyền đến các Slave Out. Chứ không có kết nối trực tiếp giữa Slave Input và Slave Output trên đường truyền. Ngoài nhiệm vụ, Master sẽ có nhiệm vụ khi nhận trạng thái hoạt động của Slave và lưu vào các biến quản lý, bên cạnh đó là các trạng thái hoạt động của Slave Input và Slave Output đều được lưu lại và xử lý để quản lý đường truyền một cách hiệu quả.

- Slave Input

Khi được khởi động và hòa vào mạng truyền thông của mô hình. Slave Input sẽ có nhiệm vụ gửi Frame truyền đăng kí trạng thái hoạt động đến Master, sau đó nếu được Master cho phép hoạt động thì Slave Input sẽ chuyển về trạng thái hoạt động. Khi ở trạng thái hoạt động, Slave Input sẽ ở trạng thái chờ(Chân Unable của RS485 ở trạng thái LOW). Khi có yêu cầu làm việc từ bên ngoài Slave Input sẽ đọc và xử lý thông tin từ đó gửi Frame truyền đến Master sau đó trở về trạng thái chờ hoạt động. Tuy nhiên trong quá trình gửi dữ liệu Slave Input sẽ lắng nghe xem đường truyền bận hay rỗi. Nếu đường truyền rỗi thì sẽ truyền dữ liệu như thông thường. Ngược lại, nếu đường truyền đang bận(đã có Slave khác truyền dữ liệu lên trên đường truyền) thì Slave này sẽ đợi một khoảng thời gian random rồi bắt đầu truyền dữ liệu lên trên đường truyền sau đó trở về trạng thái truyền và tiếp tục lắng nghe đường truyền.

- Slave Output

Khi được khởi động và hòa vào mạng truyền thông của mô hình. Slave Onput sẽ có nhiệm vụ gửi Frame truyền đăng kí trạng thái hoạt động đến Master, sau đó nếu được Master cho phép hoạt động thì Slave Output sẽ chuyển về trạng thái hoạt động. Khi ở trạng thái hoạt động, Slave Output sẽ ở trạng thái chờ(Chân Unable của RS485 ở trạng thái LOW). Khi nhận được Frame truyền yêu cầu làm việc đến từ Master , Slave Output sẽ chuyển sang trạng thái hoạt động và xử lý yêu cầu đến từ Master sau đó trở về trạng thái chờ hoạt động.

- Phân tích

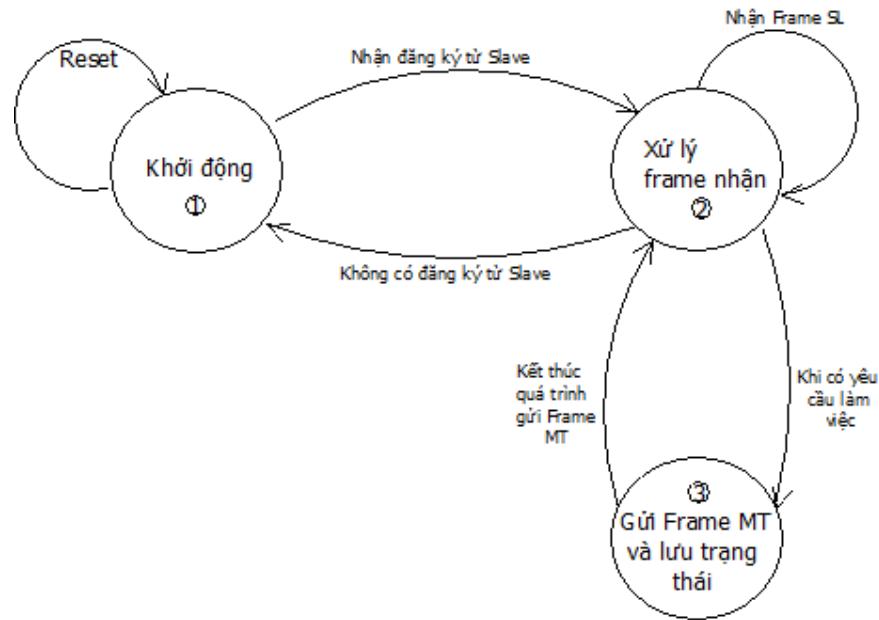
Như vậy khi bắt đầu khởi động Master sẽ cập nhật tất cả trạng thái của các Slave. Sau đó cả Master và Slave sẽ đi vào trạng thái “chờ hoạt động” . Phương thức

Chương 4 : Giải thuật và xây dựng phần mềm

truyền thông sẽ giải quyết các vấn đề vẫn còn tồn tại ở cơ chế hỏi lặp vòng cù thế như sau:

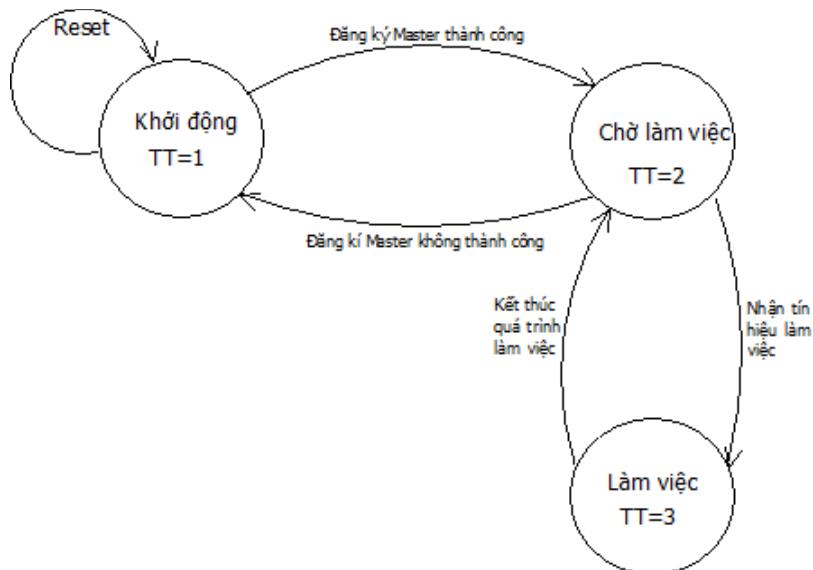
- Về phần cứng: khi ở trạng thái chờ, chỉ có vi điều khiển hoạt động, các đèn LED hiển thị sẽ không sáng vì không có frame hoạt động liên tục trên đường truyền.
- Về vấn đề thời gian: Phương thức truyền thông không truyền frame lên đường truyền liên tục nên thời gian Delay liên tục dường như không có, thay vào đó khi một frame được gửi lên đường truyền thì khoảng Delay sẽ rất nhỏ (đủ để bật trạng thái IC RS485 lên HIGH) sau đó trở về trạng thái chờ. Điều này khiến ta có thể mở rộng thêm nhiều Slave mà không cần quan tâm nhiều về vấn đề Delay.
- Về quản lý: Phương thức truyền thông không truyền frame lên đường truyền liên tục nên đường truyền sẽ trống. Điều này sẽ tránh các trường hợp đường truyền chồng chéo dẫn đến sai frame truyền. Bên cạnh đó với cơ chế lắng nghe đường truyền sẽ giúp giải quyết vấn đề chồng chéo đường truyền khi 2 Slave cùng truyền một lúc. Tuy nhiên tỉ lệ 2 Slave cùng frame rất hiếm xảy vì thời gian gửi và xử lý được tính bằng ms nên tỉ lệ rất rất nhỏ. Bên cạnh đó, trạng thái của các Slave sẽ được lưu lại và hiển thị lên máy tính(Hoặc có thể là một thiết bị màn hình hiển thị để người sử dụng có thể cập nhật được tình hình mô hình). Bên cạnh đó là các biến lưu giá trị để quản trị trạng thái của các thiết bị. Trong phương thức truyền như trên thì việc luân chuyển giữa các trạng thái là rất quan trọng. Vì ở mỗi trạng thái các thiết bị sẽ làm những công việc khác nhau. Sau đây là sơ đồ trạng thái đối với các thiết bị sử dụng trong mô hình.

Master:



Hình 4.3 Lưu đồ trạng thái Master

Slave:



Hình 4.4 Lưu đồ trạng thái Slave

Thiết kế và qui định Frame truyền:

Một Frame truyền gồm 12 byte được cấu trúc theo dạng:

0xM₁X₂X₃X₄0xFF và 0xS₁L₁...L_n0xFF

Chương 4 : Giải thuật và xây dựng phần mềm

Trong đó 2 byte đầu 0x là 2 byte đầu của Frame truyền chỉ có tác dụng để nhận biết, tuy nhiên khi phát triển mô hình với nhiều Master thì sẽ sử dụng 2 byte này, 2 byte tiếp theo để xác định Frame được truyền từ đối tượng là:

MT: Master

SL: Slave

Khi đó 4 byte mở đầu sẽ có dạng 0xMT tức là được gửi từ Master và 4 byte có dạng 0xSL thì sẽ được gửi từ Slave.

2 byte tiếp theo X₁X₂ dùng biểu diễn mã số của Slave.

2 byte tiếp theo X₃X₄ dùng biểu diễn trạng thái của Slave.

3 byte cuối 0xFE để kết thúc Frame. Có thể dùng để mở rộng số lượng Slave và trạng thái,

Với quy định như vậy Master sẽ có một số dạng Frame truyền như sau:

0xMT03XX0xFE : gửi Frame truyền đến Slave 03 yêu cầu chuyển sang trạng thái XX.

0xMT01XX0xFE : gửi Frame truyền đến Slave 01 yêu cầu chuyển sang trạng thái XX.

XX ở đây sẽ là trạng thái làm việc của Slave. Các trạng thái khác sẽ phụ thuộc vào dạng Slave là Input hay Output thể hiện rõ trong đoạn code chạy chương trình.

Với quy định như vậy Slave sẽ có một số dạng Frame truyền như sau:

0xSL02XX0xFE : gửi Frame đến Master với 02 là Slave 02, XX là trạng thái của Slave. Tương tự với các Slave còn lại.

Frame truyền và trạng thái sẽ được thể hiện rõ trong phần code chạy chương trình.

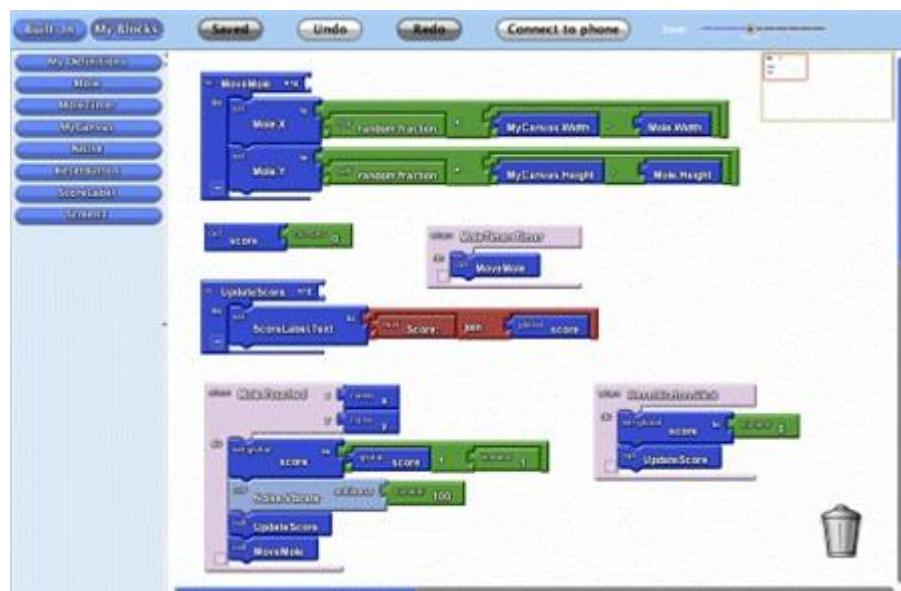
4.2. THIẾT KẾ GIAO DIỆN ĐIỀU KHIỂN

4.2.1. Xây dựng ứng dụng cho smartphone dùng giao diện Blue

4.2.1.1. App Inventor

Ngày 12/7/2010, Google chính thức giới thiệu công cụ lập trình trực quan App Inventor dùng để phát triển phần mềm ứng dụng trên hệ điều hành Android. App Inventor là công cụ lập trình dành cho mọi người, kể cả trẻ em. Được công bố dưới dạng phần mềm tự do (free software), App Inventor trở thành hiện tượng chưa từng

có trong lĩnh vực lập trình cho thiết bị di động. Các thiết bị di động (chủ yếu là điện thoại thông minh) có năng lực xử lý thông tin ngày càng mạnh, đang trở thành một chủng loại "máy tính cá nhân". Khác với máy tính cá nhân thông thường, các thiết bị di động có bộ đo gia tốc (accelerometer), con quay hồi chuyển (gyroscope), bộ định vị GPS và có thể có thêm các bộ cảm ứng khác trong tương lai, từ đó mở ra những lĩnh vực ứng dụng mới mẻ và rộng lớn. Với công cụ App Inventor, Google tạo điều kiện để mọi người có thể tự xây dựng phần mềm ứng dụng cho thiết bị di động dùng hệ điều hành Android. Trang web của dự án App Inventor nêu rõ: "Với App Inventor, bạn có thể xây dựng phần mềm ứng dụng bất kỳ theo ý tưởng của mình. App Inventor rất dễ dùng và cũng rất mạnh mẽ. Phần mềm ứng dụng của bạn có thể lưu trữ dữ liệu do người dùng tạo ra trong một cơ sở dữ liệu. App Inventor thực chất là một ứng dụng web, chạy bởi trình duyệt trên máy tính cá nhân. Tuy nhiên, người dùng vẫn phải cài đặt một phần mềm Java mang tên App Inventor Extras, có nhiệm vụ điều khiển điện thoại Android (kết nối với máy tính thông qua cổng USB). Nhờ vậy, người dùng có thể nhanh chóng chuyển ứng dụng từ máy tính cá nhân qua điện thoại Android để chạy thử.



Hình 4.5 Lập trình trò chơi "Bắt chuột chui" với App Inventor.

Ưu điểm:

- Không cần biết nhiều về code
- Chỉ có động tác kéo thả đơn giản
- Trực quan, dễ hiểu

- Hỗ trợ đủ các tập lệnh cảm biến (sensor), cơ sở dữ liệu (database), kết nối (bluetooth) – dành cho những bạn theo IoT, nghĩa là bạn có thể điều khiển như các thiết bị điện trong nhà chỉ với điện thoại Android
- Hỗ trợ các kết nối mạng xã hội, google maps....

Nhược điểm:

- Nặng, chạy chậm nếu như có quá nhiều code do chương trình phát sinh thêm nhiều code thừa không cần thiết trong quá trình build sang file cài APK
- Không tối ưu hóa code được
- Mỗi screen (Activity) hoạt động độc lập với nhau, chỉ có thể truyền 1 biến duy nhất sang screen kế tiếp. Nghĩa là không có biến toàn cục cho toàn bộ screen
- Tổng kết
- Nếu như bạn là người mới bắt đầu tìm hiểu Android, thì nên dùng MIT Inventor train trong khoảng từ 1 đến 2 tuần.
- MIT App Inventor không thể tạo ứng dụng phức tạp, chỉ có thể là một chương trình nhỏ, game nhỏ.
- Ứng dụng App Inventor xây dựng phần mềm Blue_Control:

4.2.1.2. Xây dựng ứng dụng điều khiển I-HOME Blue_Control

Mục đích:

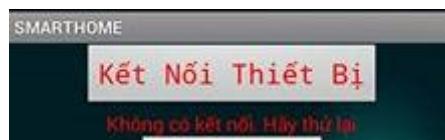
App Blue_Control được viết trên mã nguồn mở của App Inventor với chức năng dùng để kết nối Smart-phone với kết nối Bluetooth của mô hình, từ đó sẽ lập trình để gửi thông tin đến vi điều khiển, vi điều khiển sẽ đọc thông tin và xử lý tùy theo tác vụ người sử dụng mong muốn

Giao diện : App Blue_Control gồm 4 phần:

Phần 1: Kết nối với thiết bị Bluetooth khác

Có tác dụng kết nối với thiết bị Blue xung quanh (Cụ thể ở đây là module Bluetooth HC05). Khi không có kết nối thì sẽ hiển thị dòng chữ màu đỏ. Để kết nối ta ấn vào phần Kết Nối Thiết Bị. Nếu chưa có kết nối thì trạng thái sẽ ở dòng chữ màu đỏ, nếu có kết nối sẽ chuyển sang trạng thái màu xanh

Chương 4 : Giải thuật và xây dựng phần mềm



Hình 4.6 *Giao diện Kết nối với thiết bị Bluetooth khác*

Phần 2: Nhận diện và xử lý tín hiệu âm thanh từ người sử dụng



Hình 4.7 *Giao diện nhận diện và xử lý tín hiệu âm thanh từ người sử dụng*

Có tác dụng xử lý âm thanh từ người sử dụng sau đó truyền thông tin dưới dạng text đến vi điều khiển, vi điều khiển sẽ xử lý thông tin và thực hiện theo yêu cầu cài đặt sẵn của người sử dụng. Phần xử lý âm thanh được kết nối với Mic thu của Smart- phone. Khi ta ấn vào biểu tượng Micro, Smart-phone sẽ yêu cầu ta nói vào để truyền thông tin, thông tin này sẽ được nhận dạng và xử lý bằng smart phone từ đó chuyển sang dạng text và được gửi qua Bluetooth. Phần Thông Tin Truyền sẽ hiển thị thông tin nhận được từ người sử dụng

Phần 3: Điều khiển thông qua các Button.



Hình 4.8 *Giao diện điều khiển thông qua các Button.*

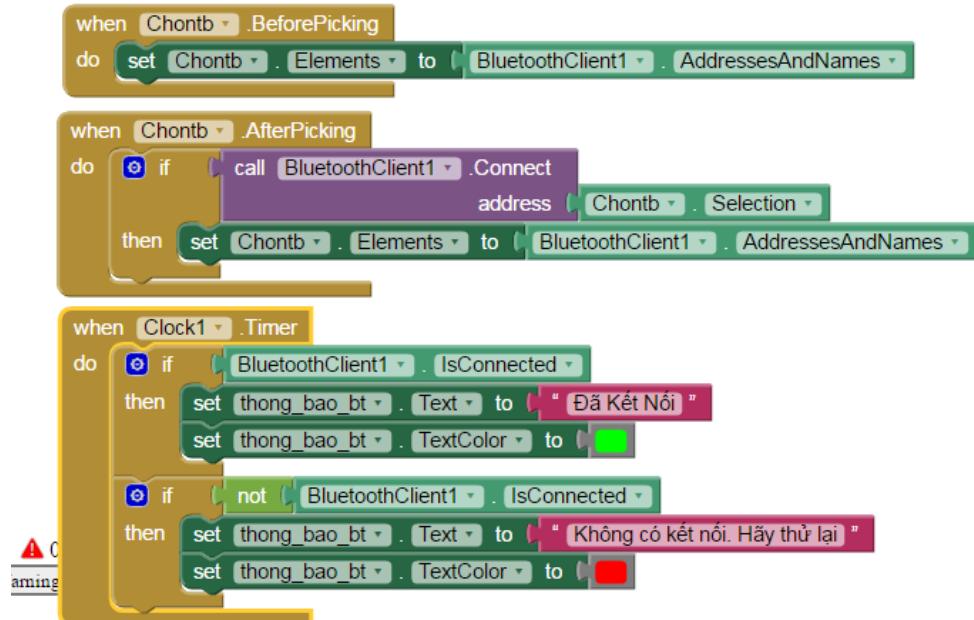
Các Button hiển thị tương ứng với thiết bị cần điều khiển trong mô hình và được chia làm 2 phần. Phần LivingRoom dùng để điều khiển các thiết bị phòng khách. Phần BedRoom dùng để điều khiển các thiết bị phòng ngủ. Các Button sử dụng ở 2 trạng thái. Khi Trạng thái màu trắng nghĩa là thiết bị đang tắt và ở trạng thái

Chương 4 : Giải thuật và xây dựng phần mềm

xanh là thiết bị đang mở. Mỗi trạng thái sẽ tương ứng với một lệnh text khác nhau được gửi qua Bluetooth.

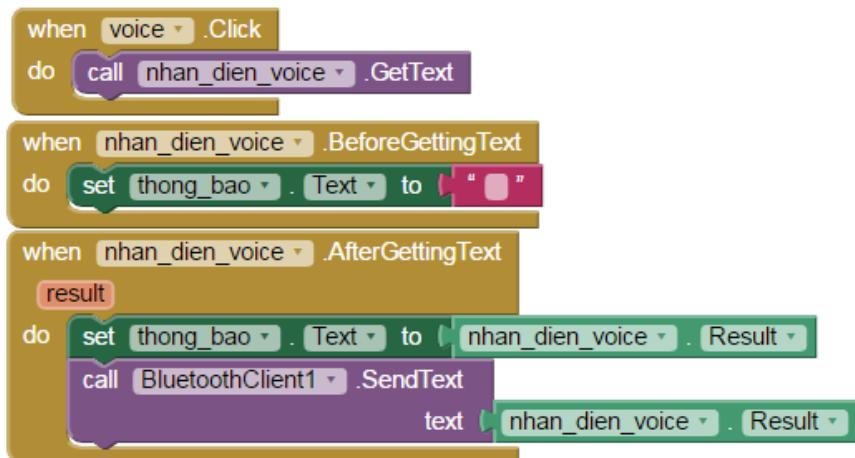
Các khối code của chương trình:

Phần 1: Kết nối với thiết bị Bluetooth khác



Hình 4.9 Khối code kết nối với thiết bị Bluetooth khác

Phần 2: Nhận diện và xử lý tín hiệu âm thanh từ người sử dụng



Hình 4.10 Khối code nhận diện và xử lý tín hiệu âm thanh từ người sử dụng

Phần 3: Điều khiển thông qua các Button.

Khối lệnh cho Button 1



Hình 4.11 Khối code điều khiển thông qua các Button.

Các khối lệnh điều khiển cho các Button khác đều tương tự Button 1. Tuy nhiên đổi với mỗi Button sẽ SendText khác nhau tương ứng với từng thiết bị.

4.2.2. Xây dựng ứng dụng cho PC dùng Visual Studio

4.2.2.1. Giới thiệu phần mềm Visual Studio 2013

Microsoft Visual Studio là một môi trường phát triển tích hợp (IDE) từ Microsoft. Nó được sử dụng để phát triển chương trình máy tính cho Microsoft Windows, cũng như các trang web, các ứng dụng web và các dịch vụ web. Visual Studio sử dụng nền tảng phát triển phần mềm của Microsoft như Windows API, Windows Forms, Windows Presentation Foundation, Windows Store và Microsoft Silverlight. Nó có thể sản xuất cả hai ngôn ngữ máy và mã số quản lý.

Visual Studio bao gồm một trình soạn thảo mã hỗ trợ IntelliSense cũng như cài đặt mã nguồn. Trình gõ lỗi tích hợp hoạt động cả về trình gõ lỗi mức độ mã nguồn và gõ lỗi mức độ máy. Công cụ tích hợp khác bao gồm một mẫu thiết kế các hình thức xây dựng giao diện ứng dụng, thiết kế web, thiết kế lớp và thiết kế giản đồ cơ sở

dữ liệu. Nó chấp nhận các plug-in nâng cao các chức năng ở hầu hết các cấp bao gồm thêm hỗ trợ cho các hệ thống quản lý phiên bản (như Subversion) và bổ sung thêm bộ công cụ mới như biên tập và thiết kế trực quan cho các miền ngôn ngữ cụ thể hoặc bộ công cụ dành cho các khía cạnh khác trong quy trình phát triển phần mềm.

Visual Studio hỗ trợ nhiều ngôn ngữ lập trình khác nhau và cho phép trình biên tập mã và gỡ lỗi để hỗ trợ (mức độ khác nhau) hầu như mọi ngôn ngữ lập trình. Các ngôn ngữ tích hợp gồm có C, C++ và C++/CLI (thông qua Visual C++), VB.NET (thông qua Visual Basic.NET), C# (thông qua Visual C#) và F# (như của Visual Studio 2010). Hỗ trợ cho các ngôn ngữ khác như J++/J#, Python và Ruby thông qua dịch vụ cài đặt riêng rẽ. Nó cũng hỗ trợ XML/XSLT, HTML/XHTML, JavaScript và CSS.

Microsoft cung cấp phiên bản "Express" (đối với phiên bản Visual Studio 2013 trở về trước) và "Community" (đối với bản Visual Studio 2015) là phiên bản miễn phí của Visual Studio.

4.2.2.2. Xây dựng ứng dụng điều khiển I-HOME với VS 2013

Mục đích:

Phần mềm điều khiển được viết dựa trên phần mềm Visual studio 2013 của Microsoft. Phần mềm điều khiển sẽ kết nối với module ESP8266 thông qua địa chỉ IP và port. Từ đó sẽ lập trình để gửi các thông tin đến vi điều khiển, vi điều khiển sẽ đọc thông tin và xử lý tùy theo tác vụ người sử dụng mong muốn

Thiết kế giao diện:

Giao diện của phần mềm điều khiển trên PC được chia làm 3 phần:

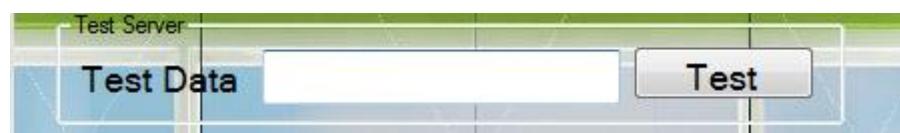
Phần 1: Kết nối đến địa chỉ IP



Hình 4.12 Giao diện kết nối đến địa chỉ IP

Chức năng của phần này là kết nối với địa chỉ IP của thiết bị. Khi muốn kết nối vào một thiết bị, ta nhập địa chỉ IP và Port vào phần này. Địa chỉ IP ở đây có thể là IP nội hoặc IP ngoại của thiết bị. Ví dụ như kết nối với module ESP8266 sử dụng mạng wifi ở nhà ta được cấp địa chỉ IP 192.168.1.134/8000. Thì phần IP ta sẽ nhập: 192.168.1.134 và phần Port ta sẽ nhập 8000. Sau đó click vào biểu tượng kết nối. Nếu kết nối thành công hay thất bại ta đều nhận được thông báo. Để ngắt kết nối ta ấn vào biểu tượng Close.

Phần 2: Send text đến Sever



Hình 4.13 Giao diện send text đến Server

Chức năng của phần này là send nội dung text được nhập vào từ bàn phím của máy tính đến thiết bị đang kết nối.

Phần 3: Điều khiển thông qua các Button.



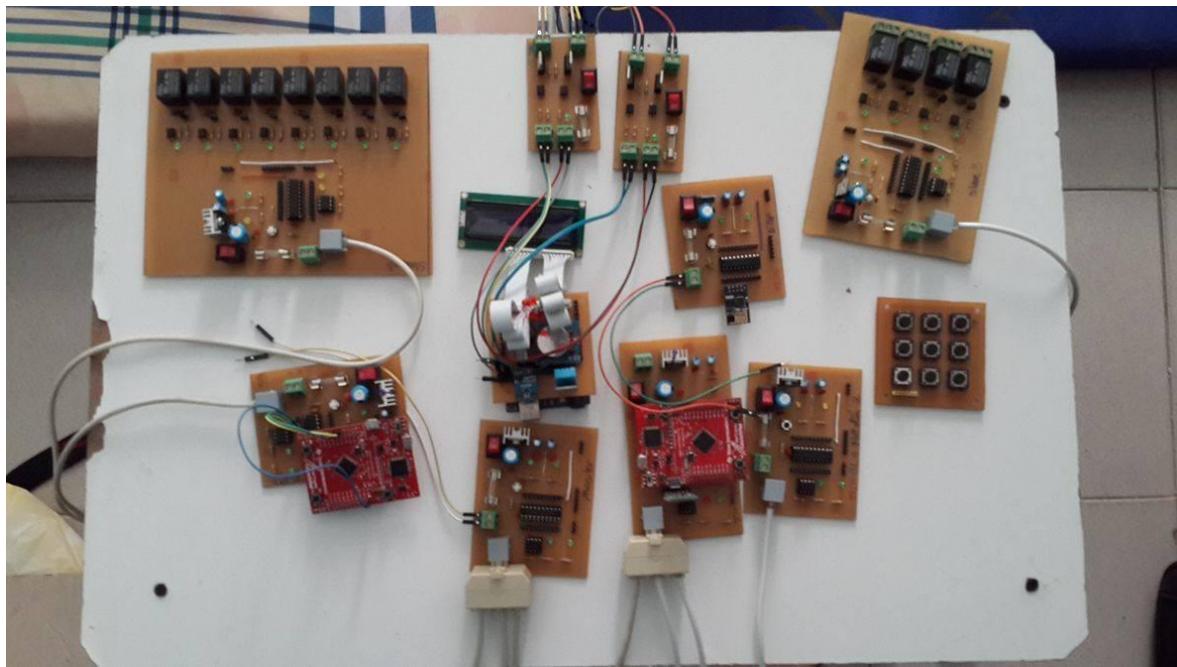
Hình 4.14 Giao diện điều khiển thông qua các Button.

Các Button hiển thị tương ứng với thiết bị cần điều khiển trong mô hình và được chia làm 2 phần. Phần LivingRoom dùng để điều khiển các thiết bị phòng khách. Phần BedRoom dùng để điều khiển các thiết bị phòng ngủ. Các Button sử dụng ở 2 trạng thái. Khi Trạng thái màu xám nghĩa là thiết bị đang tắt và ở trạng thái cam là thiết bị đang mở. Mỗi trạng thái sẽ tương ứng với một lệnh text khác nhau và gửi đến thiết bị đang kết nối.

Code của chương trình: Tham khảo phụ lục

Chương 5. KẾT QUẢ THỰC HIỆN

5.1. TỔNG THỂ MÔ HÌNH:

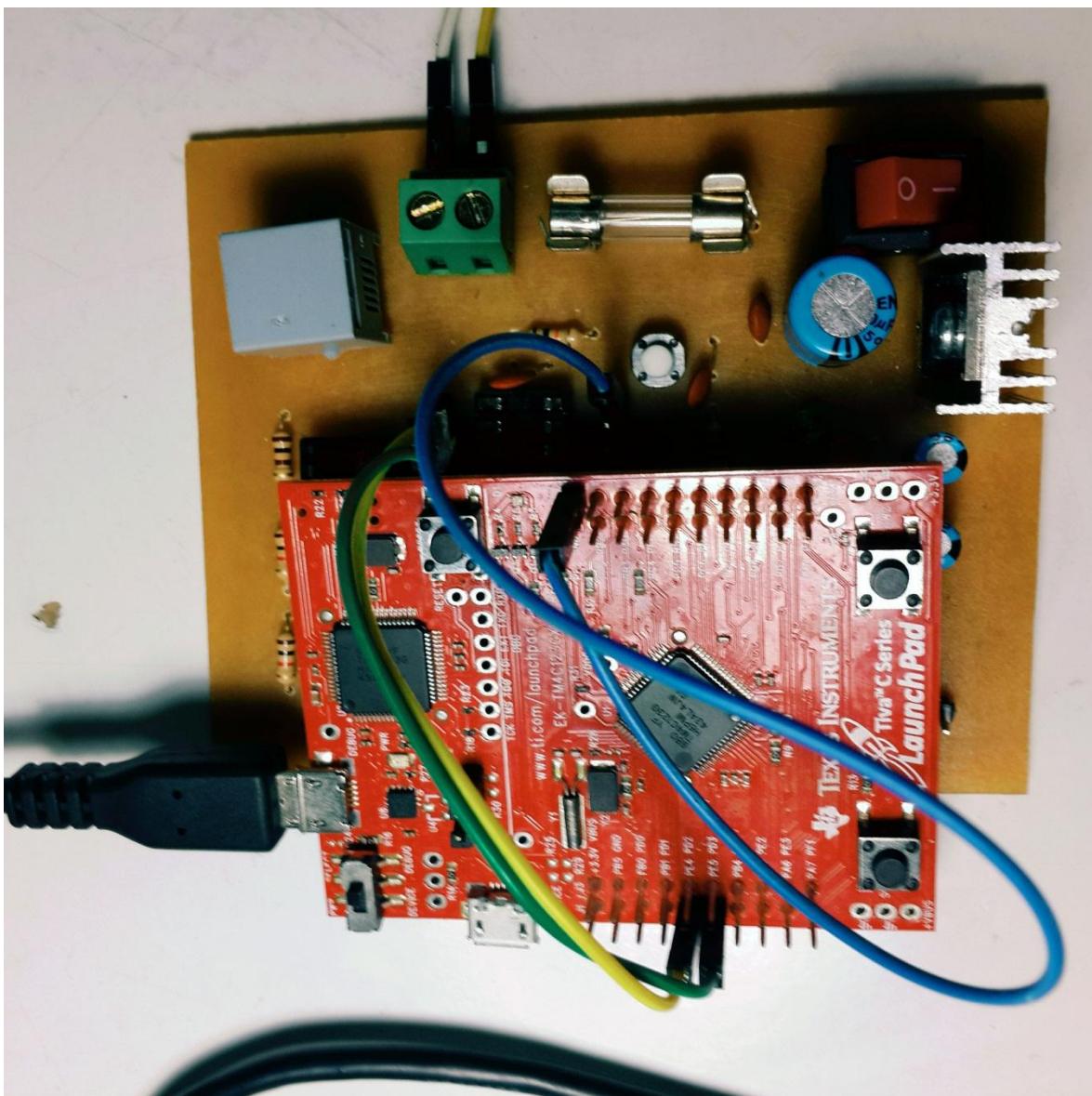


Hình 5.1 *Tổng thể mô hình*

Chương 5. Kết quả thực hiện

5.2. KHỐI MASTER:

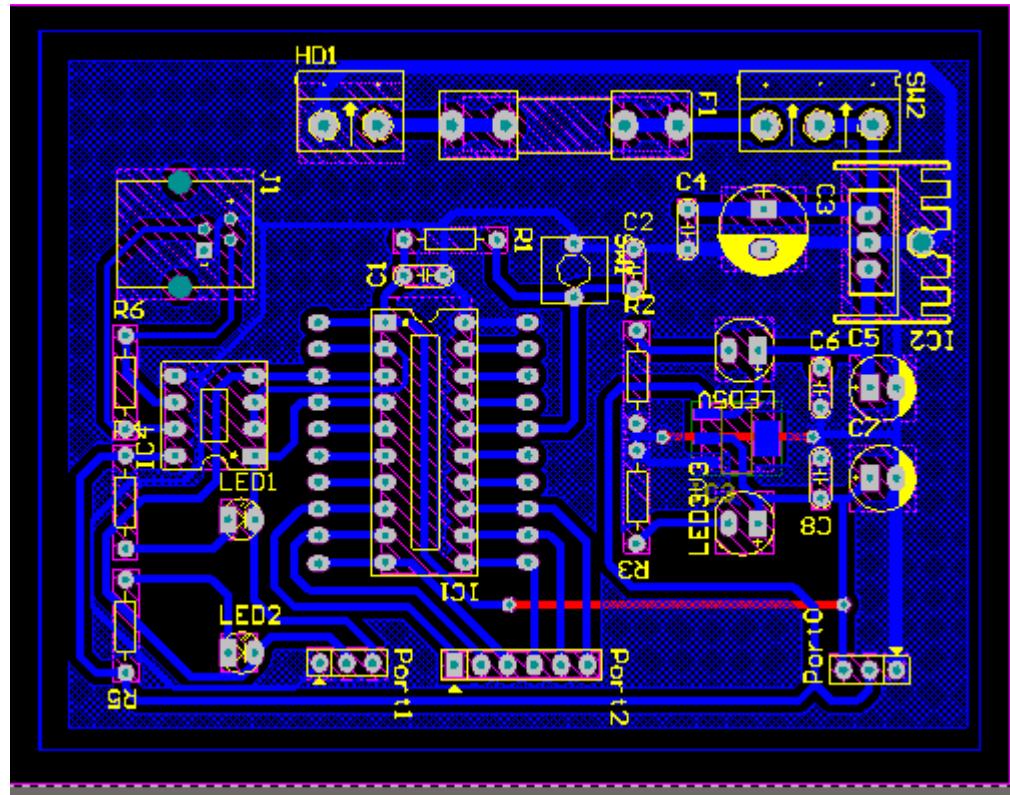
Mạch hoàn chỉnh:



Hình 5.2 Mạch master hoàn chỉnh

Chương 5. Kết quả thực hiện

Layout:



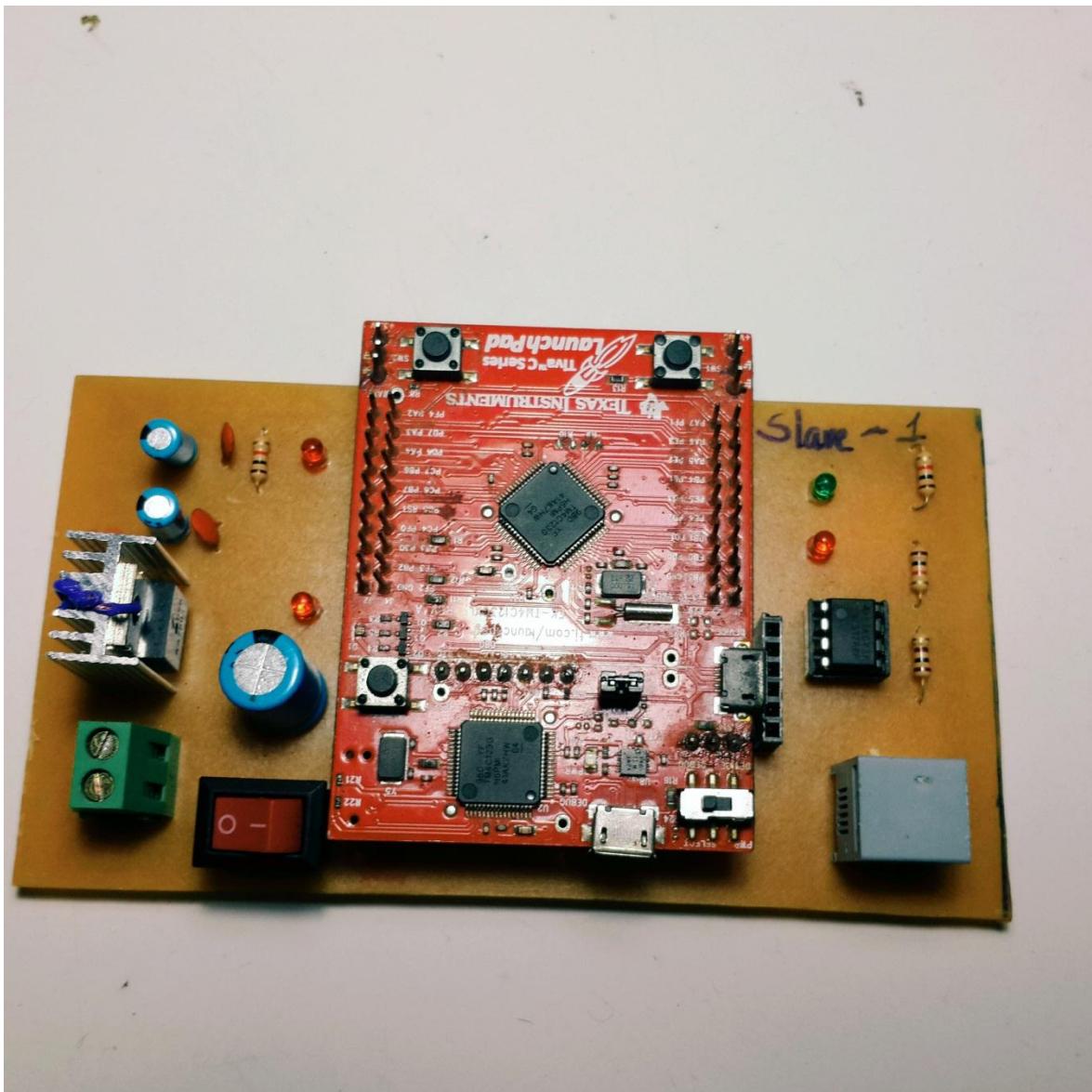
Hình 5.3 Layout Master

Chương 5. Kết quả thực hiện

5.3. KHỐI SLAVE:

5.3.1. Slave đa chức năng (#1)

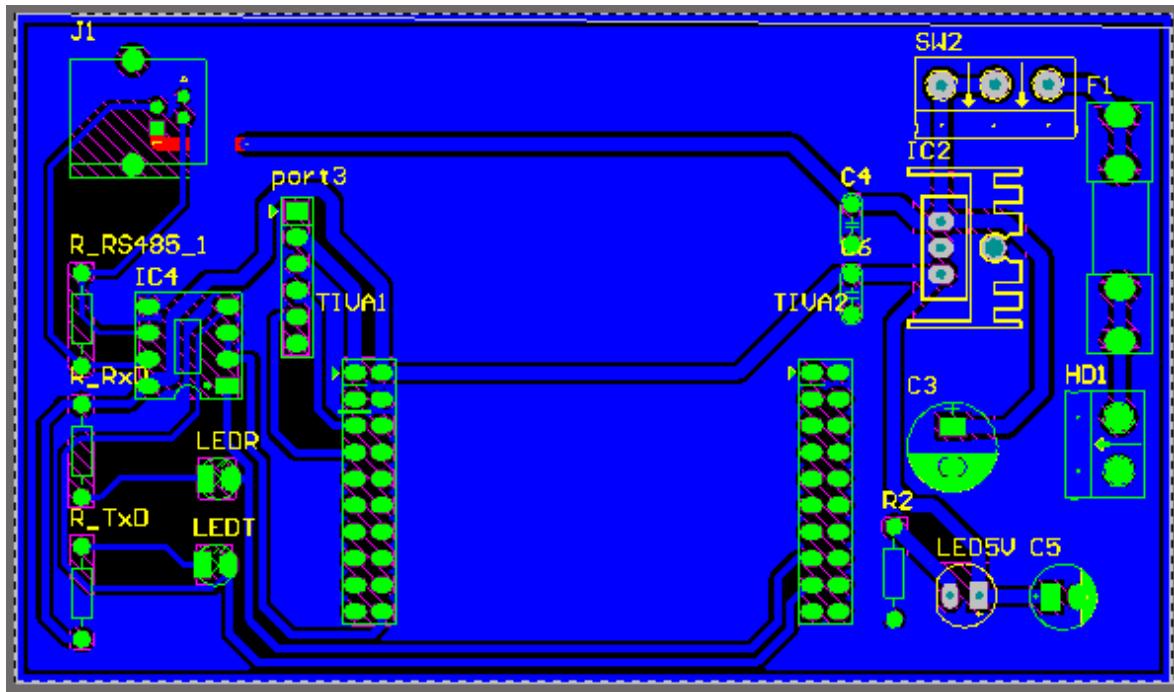
Mạch hoàn chỉnh:



Hình 5.4 Mạch Slave đa chức năng hoàn chỉnh

Chương 5. Kết quả thực hiện

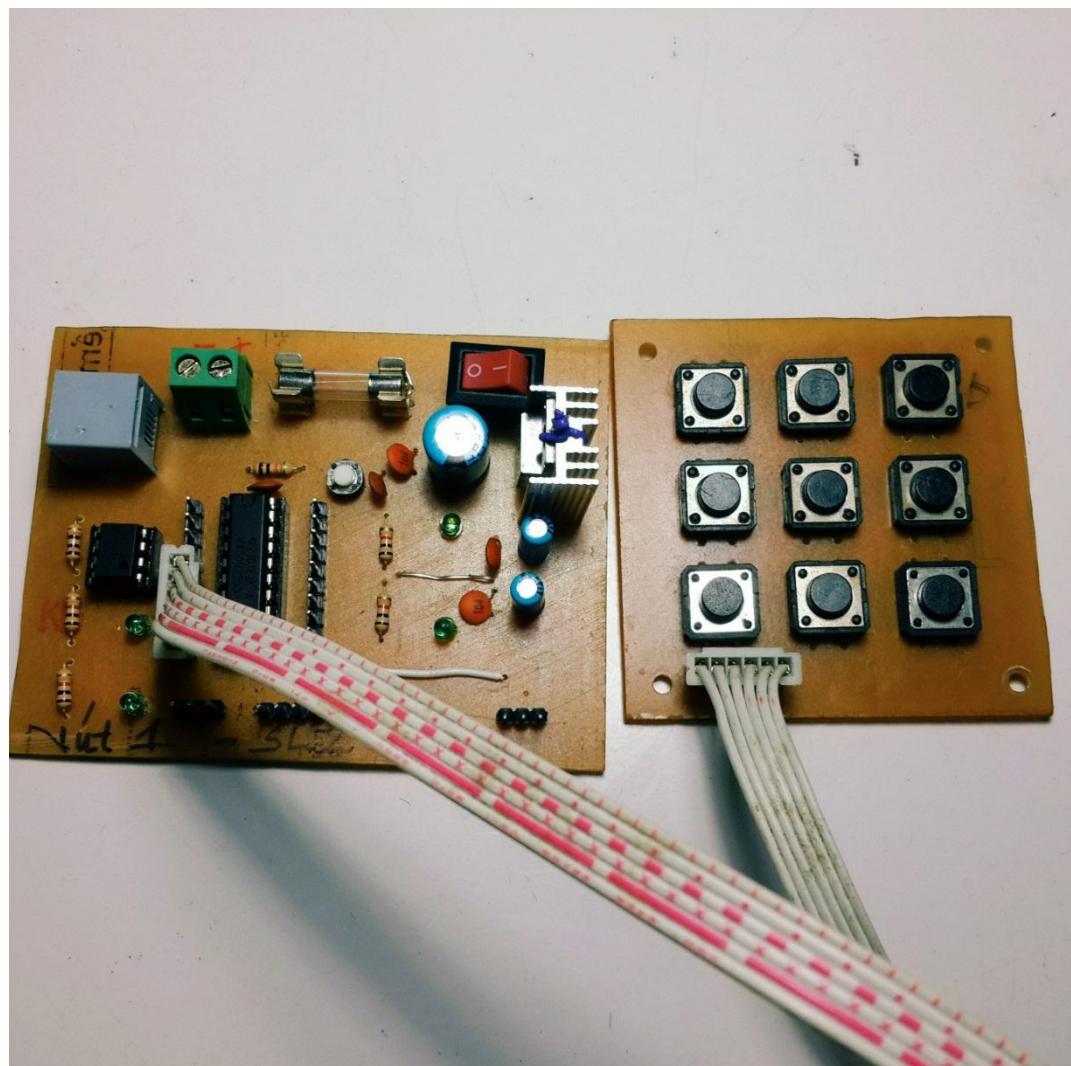
Layout:



Hình 5.5 Layout Slave đa chức năng

5.3.2. Slave Input Button (#2)

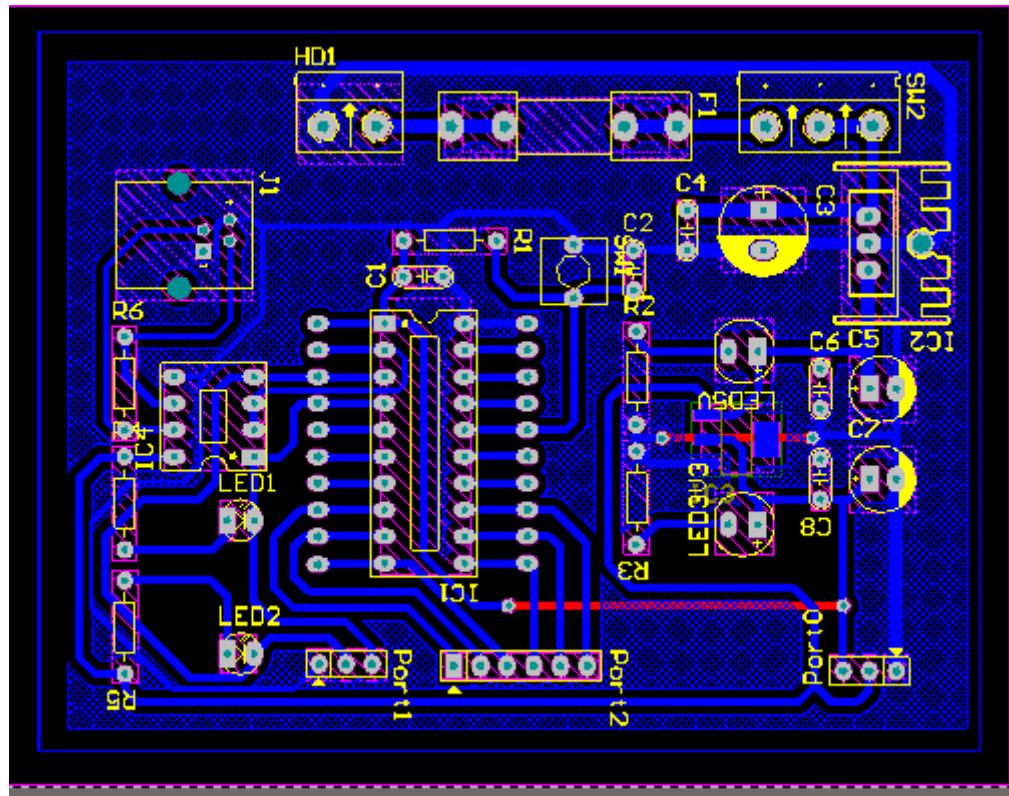
Mạch hoàn chỉnh:



Hình 5.6 Mạch Slave Input Button hoàn chỉnh

Chương 5. Kết quả thực hiện

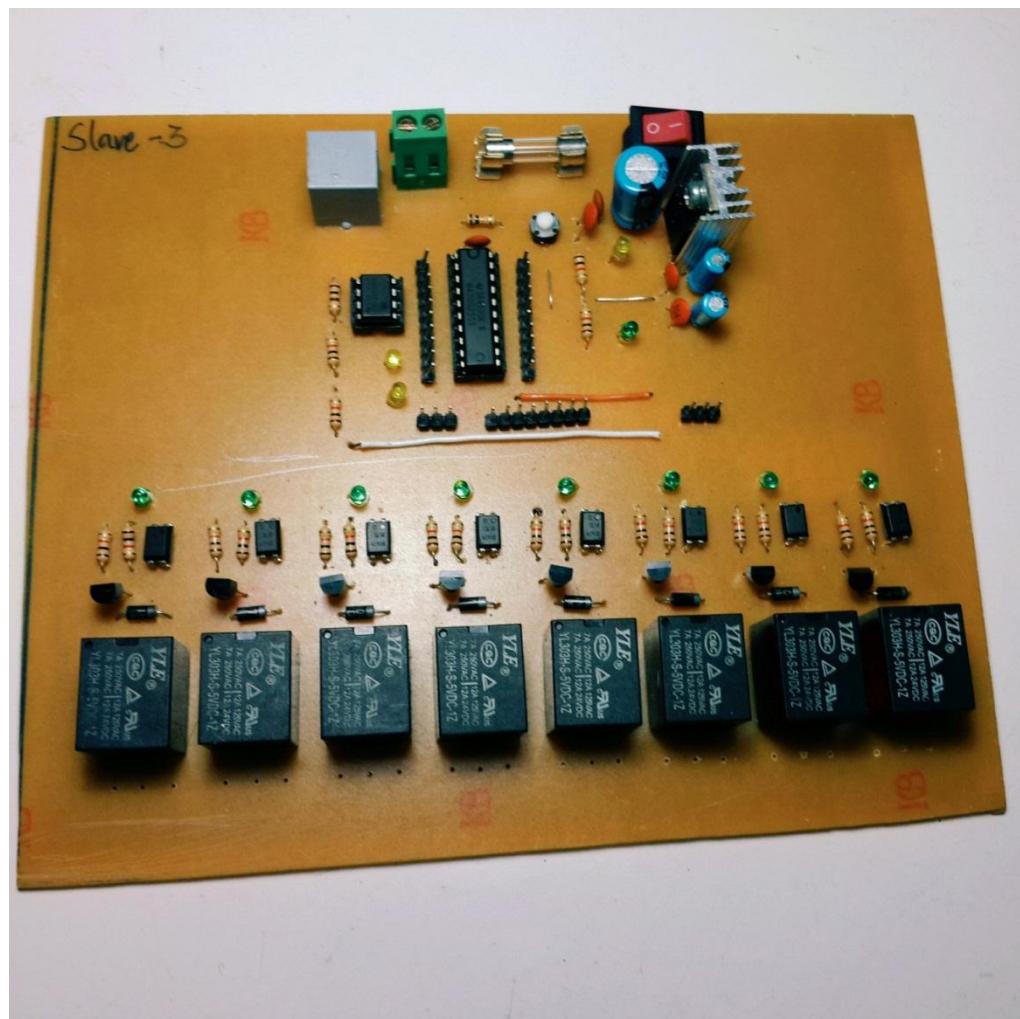
Layout:



Hình 5.7 Layout Slave Input Button

5.3.3. Slave Output Relay (#3)

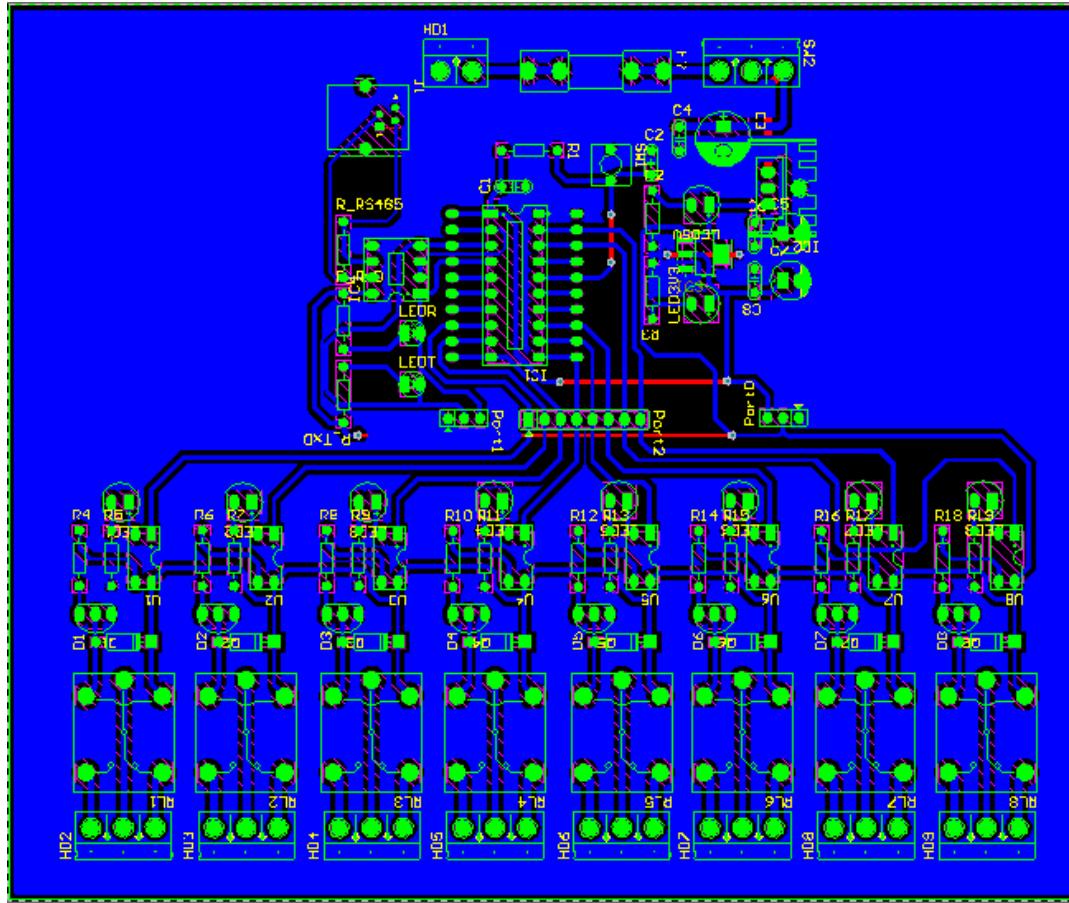
Mạch hoàn chỉnh:



Hình 5.8 Slave Output Relay hoàn chỉnh

Chương 5. Kết quả thực hiện

Layout:

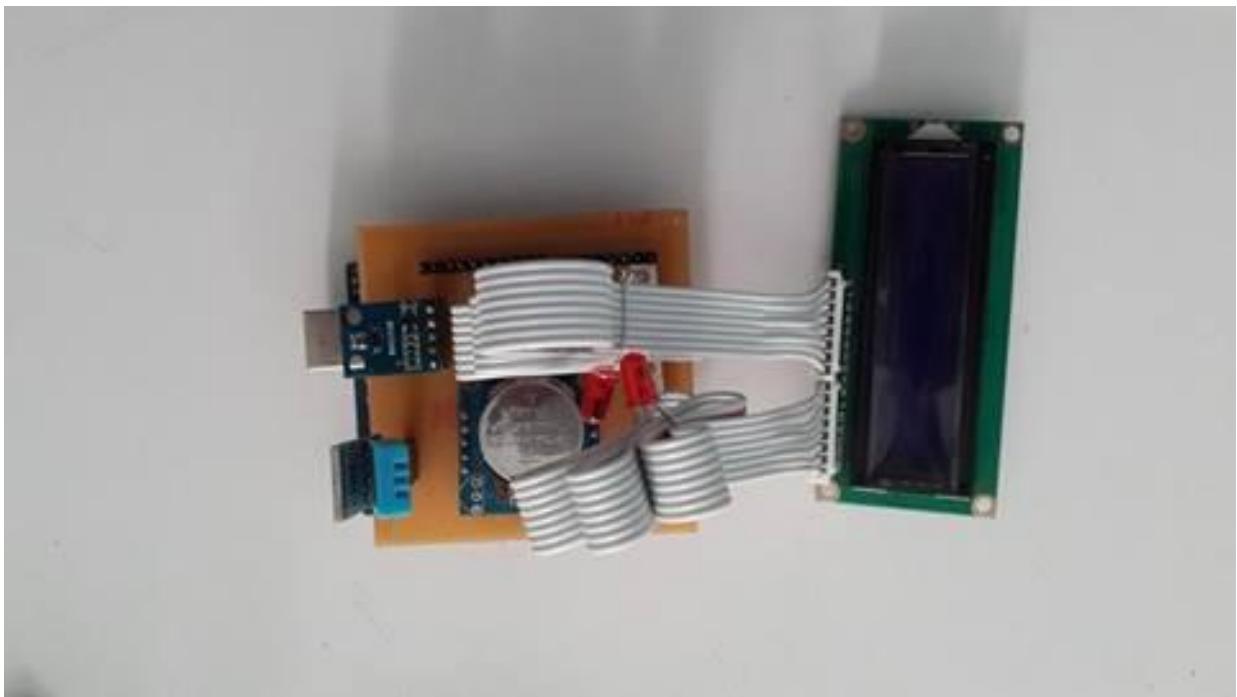


Hình 5.9 Layout Output Relay

Chương 5. Kết quả thực hiện

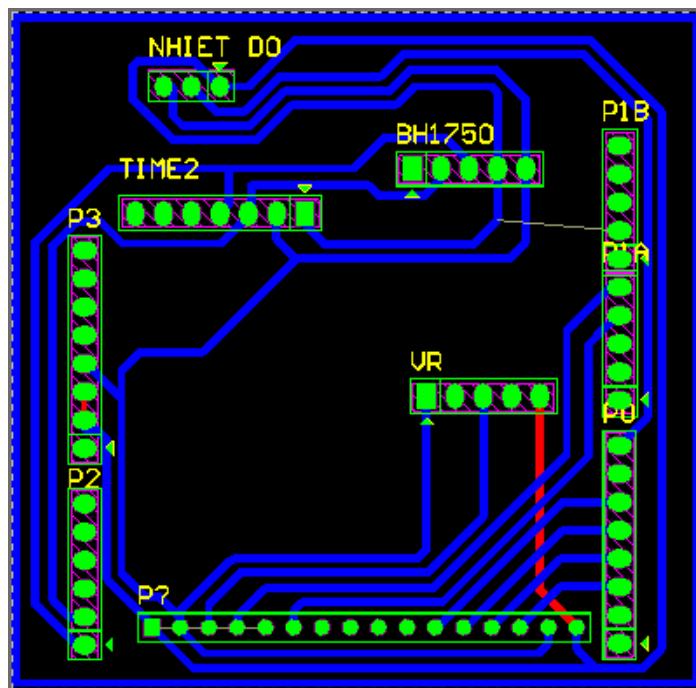
5.4. KHỐI CẢM BIẾN VÀ HIỂN THỊ:

Mạch hoàn chỉnh:



Hình 5.10 Mạch cảm biến và hiển thị hoàn chỉnh

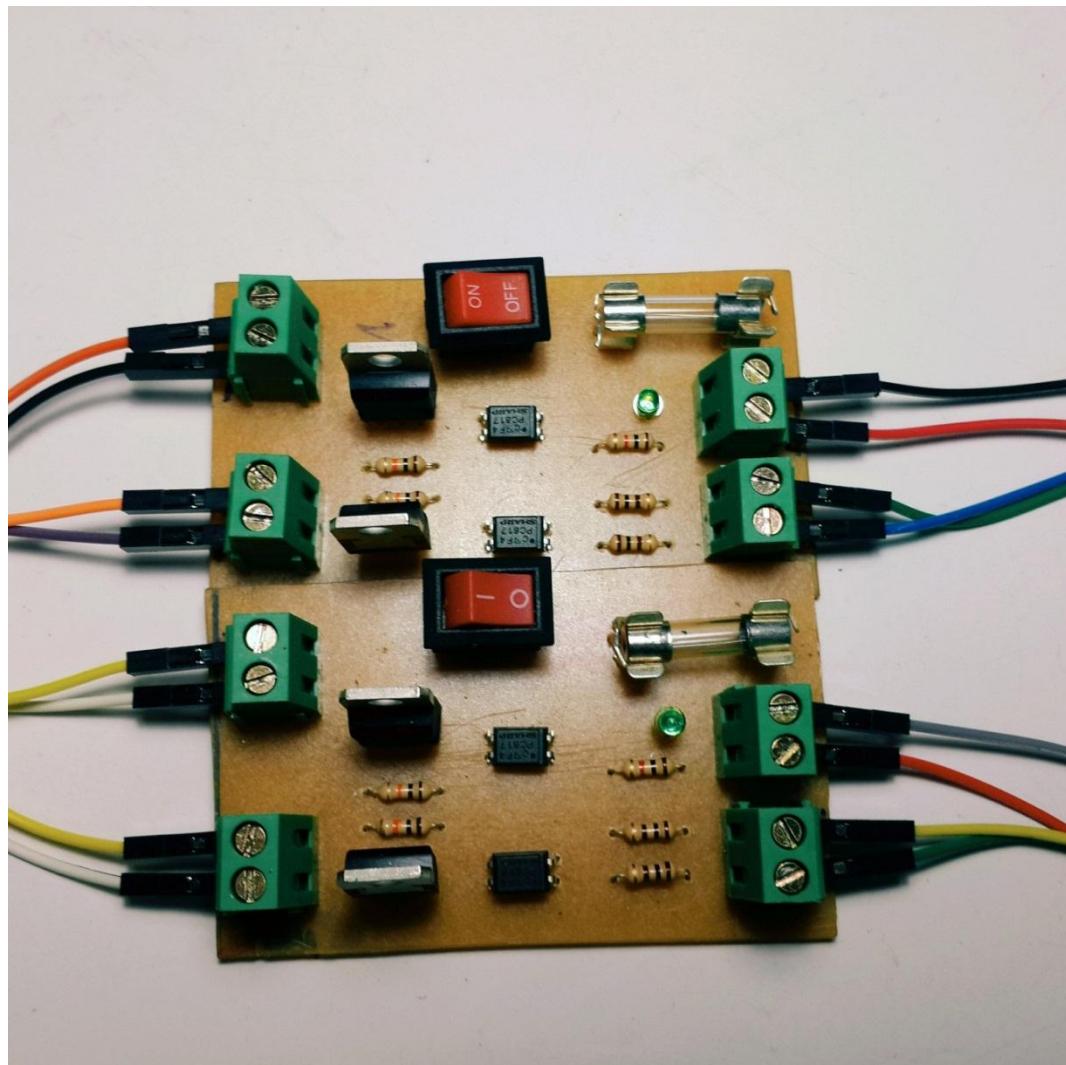
Layout:



Hình 5.11 Layout cảm biến và hiển thị

5.5. KHỐI DIMMER:

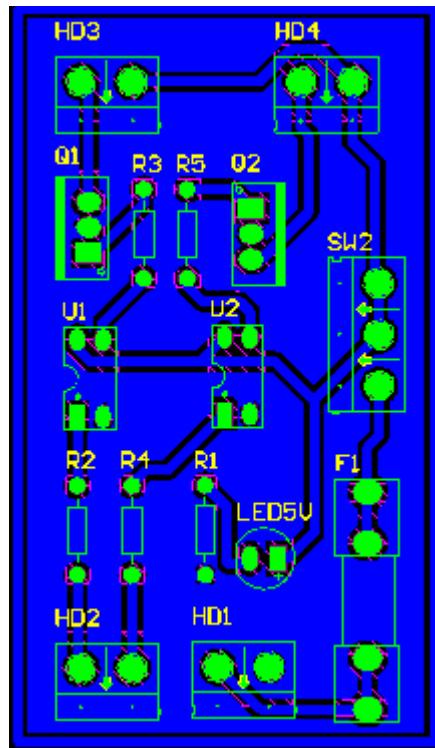
Mạch hoàn chỉnh:



Hình 5.12 Mạch dimmer hoàn chỉnh

Chương 5. Kết quả thực hiện

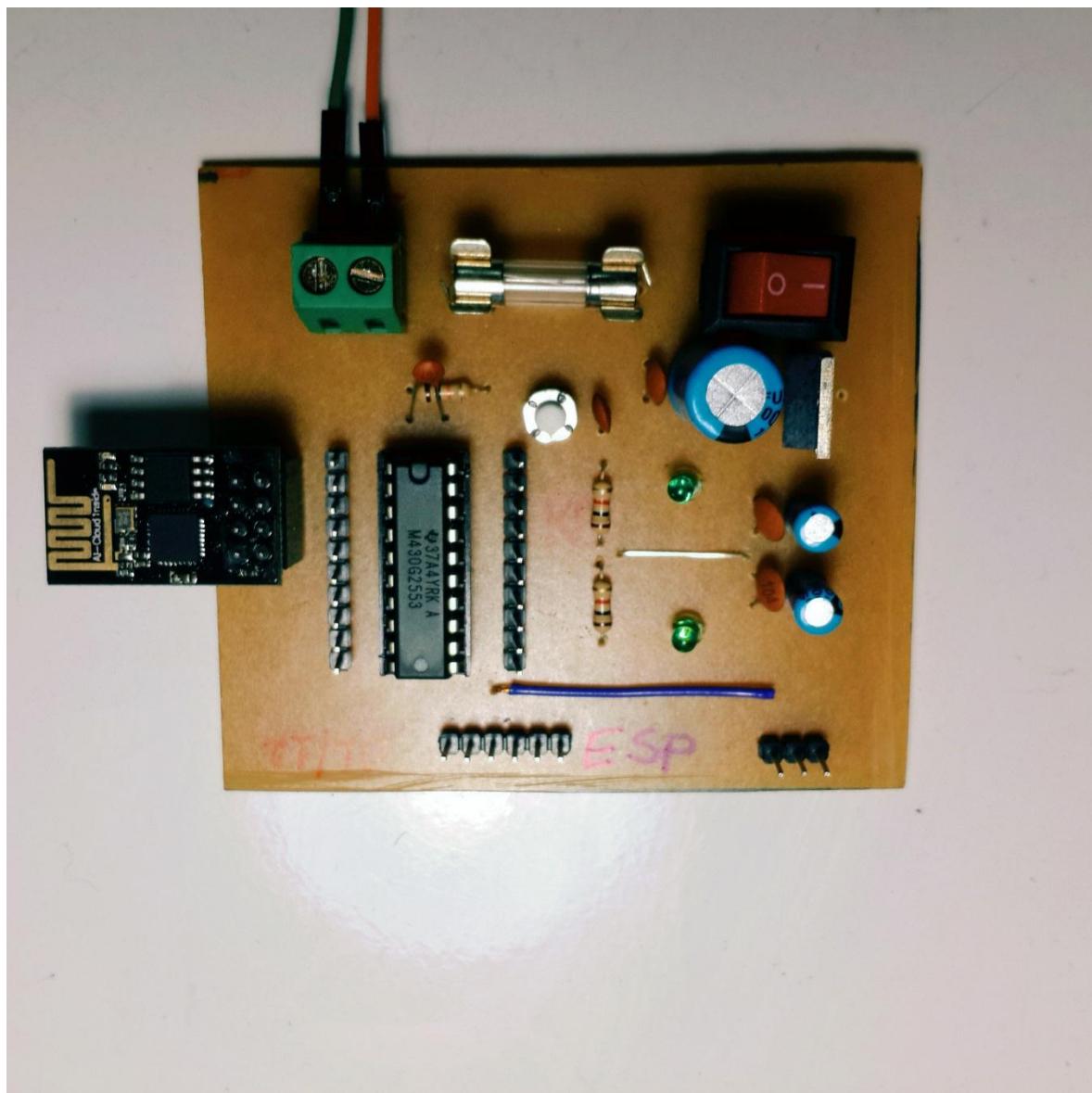
Layout:



Hình 5.13 Layout dimmer

5.6. KHỐI MSP-ESP8266:

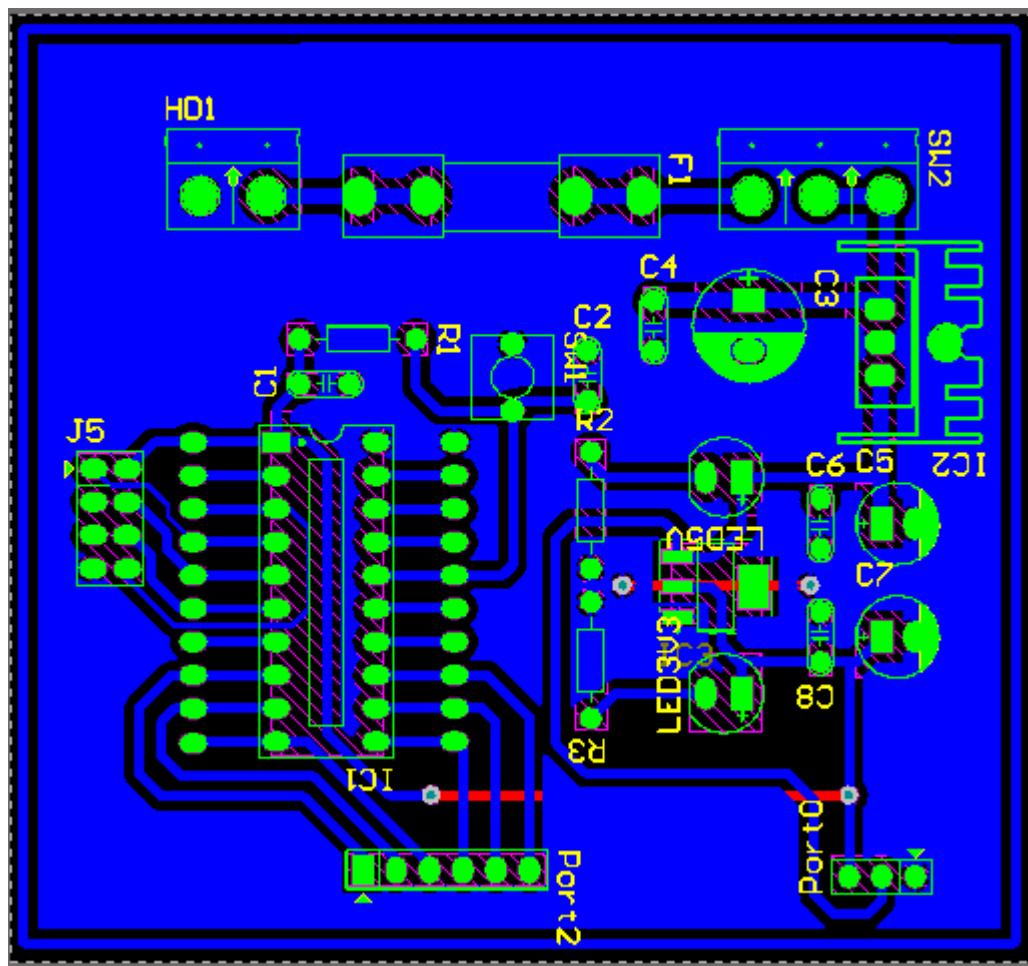
Mạch hoàn chỉnh:



Hình 5.14 Mạch MSP-ESP8266 hoàn chỉnh

Chương 5. Kết quả thực hiện

Layout:



Hình 5.15 Layout MSP-ESP8266

Chương 6. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

6.1. Kết luận

6.1.1. Kết quả đạt được

- Mô hình đã chạy thành công như mong muốn ban đầu thiết kế
- Thiết kế và chạy thành công giao thức truyền thông.
- Thiết kế được sản phẩm bằng Altium và thi công mạnh
- Thiết kế và thi công được mạch điều khiển cho sản phẩm.
- Quản lý được các thiết bị thông qua Master.

6.1.2. Hạn chế

Tuy nhiên, bên cạnh những kết quả đạt được và không ít những hạn chế làm ảnh hưởng đến chất lượng của luận văn.

❖ Về phần cứng:

- Mạch còn lớn và thiết kế chưa được tối ưu.
- Các kết nối sử dụng bus trong mạch điều khiển với thiết bị ngoại vi chưa đảm bảo được tính thẩm mỹ mô hình
- Mô hình chưa mang lại tính tối ưu và khả dụng khi lắp đặt cho một ngôi nhà.
- Chưa tính toán các yếu tố về thẩm mỹ, yếu tố về kinh tế để có thể đưa sản phẩm ra thị trường

❖ Về thuật toán:

- Chưa có bước mô phỏng bằng phần mềm.
- Thuật toán chưa có chế độ phải hồi và update cơ sở dữ liệu lên internet để giám sát ngôi nhà từ xa
- Chưa tối ưu được cơ sở quản lý dữ liệu các thiết bị

❖ Về hạn chế của người thực hiện:

- Thiếu kinh nghiệm trong việc thiết kế thi công mạch điện có kích thước nhỏ.
- Hạn chế về khả năng sắp xếp và phân bố thời gian làm ảnh hưởng đến việc phát triển thuật toán. Chưa thể áp dụng các giải thuật phức tạp hơn nhưng hứa hẹn cho chất lượng điều khiển tốt hơn.
- Thiếu kinh nghiệm trong việc xây dựng thư viện để tối ưu code, thiếu kiến thức trong các ứng dụng IOT.

6.2. Hướng phát triển

Luận văn này chủ yếu tập trung thi công hệ thống điều khiển làm nền tảng cơ bản cho việc thiết kế các tính năng khác như an ninh, ánh sáng, giải trí... Các hướng có thể phát triển để đưa ra thị trường như: Tối ưu hóa kích thước mạch, xây dựng cơ sở dữ liệu để quản lý đường truyền tốt hơn, xây dựng chuẩn đường truyền không dây, xây dựng theo module công nghiệp với tính năng thẩm mĩ và hiệu xuất cao...

TÀI LIỆU THAM KHẢO

- [1] Trương Trọng Tiên, “*Intelligent Home*”, Luận văn tốt nghiệp ngành Kỹ Thuật Điện, ĐHBK TP.HCM, 2010.
- [2] Hồ Trung Mỹ, “Vi xử lý”, Nhà xuất bản ĐHQG TP.HCM, ĐHBK TP.HCM.
- [3] Trần Văn Sư, “Truyền số liệu và mạng”, Nhà xuất bản ĐHQG TP.HCM, ĐHBK TP.HCM.
- [4] Hein Marais. *RS-485/RS-422 Circuit Implementation Guide* .
- [5] Tự học lập trình Adruino: Adruino.vn
- [6] Học AVR.VN

PHỤ LỤC

Tập lệnh AT cho Module Wifi ESP8266:

Các lệnh AT chung:

AT	Kiểm tra lệnh, luôn trả về "OK"	
AT+RTS	Khởi động lại module	
AT+GMR	Truy vấn phiên bản Firmware	

Các lệnh AT đối với Module Wifi cấu hình là Station / client

AT+CWJAP = <ssid>, <password>	Kết nối với 1 mạng wifi	ssid "SSID" pass "password"	AT+CWJAP = "MLAB", "1235678"
AT+CWJAP?	Truy vấn mạng wifi đang kết nối		AT+CWJAP?
AT+CWLAP	Truy vấn các mạng wifi có thể kết nối		AT+CWLAP
AT+CWQAP	Đóng kết nối wifi với một Access Point		AT+ CWQAP
AT+CIFSR	Xem địa chỉ IP của module		AT+CIFSR

Tập lệnh AT cho Module HC05:

AT: Lệnh test, nó sẽ trả về OK nếu module đã hoạt động ở Command Mode

AT+VERSION? : trả về firmware hiện tại của module

AT+UART=9600,0,0 (thiết lập baudrate 9600,1 bit stop, no parity)

Các lệnh ở chế độ Master:

AT+RMAAD : ngắt kết nối với các thiết bị đã ghép

AT+ROLE=1 : đặt là module ở master

AT+RESET: reset lại thiết bị

AT+CMODE=0: Cho phép kết nối với bất kì địa chỉ nào

AT+INQM=0,5,5: Dừng tìm kiếm thiết bị khi đã tìm được 5 thiết bị hoặc sau 5s

AT+PSWD=1234 Set Pin cho thiết bị

AT+INQ: Bắt đầu tìm kiếm thiết bị để ghép nối

Sau lệnh này một loạt các thiết bị tìm thấy được hiện thị. Định ra kết quả sau lệnh này như sau INQ:address,type,signal

Phần địa chỉ (address) sẽ có định dạng như sau: 0123:4:567890. Để sử dụng địa chỉ này trong các lệnh tiếp theo ta phải thay dấu ":" thành ","

0123:4:567890 -> 0123,4,5678

AT+PAIR=<address>,<timeout> : Đặt timeout(s) khi kết nối với 1 địa chỉ slave

AT+LINK=<address> Kết nối với slave

Các lệnh ở chế độ Slave:

AT+ORGL: Reset lại cài đặt mặc định

AT+RMAAD: Xóa mọi thiết bị đã ghép nối

AT+ROLE=0: Đặt là chế độ SLAVE

AT+ADDR: Hiển thị địa chỉ của SLAVE

Code ứng dụng điều khiển I-HOME với Visual Studio 2013:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO;
using System.Net;
using System.Net.Sockets;

namespace WEB_SOCKET_CLIENT
{
    public partial class mainWindow : Form
    {
        ****
        ****
        * PUBLICs
```

```

*****
*****
/* Frame */
string strFrame = string.Empty;           // String for sending data of members

/* Web Socket */
bool isConnectedToServer = false;          // Indicate whether connected to Server
static TcpClient clientSocket = null;       // TCP Client object connect to internal
socket
static Stream streamSocket = null;          // Stream of TCP
static byte[] buffer = new byte[512];        // Buffer for reading

/* Button */
Color btnColorTurnOff = Color.LightGray;
Color btnColorTurnOn = Color.DarkOrange;
uint btnCmdTurnOn = 1;
uint btnCmdTurnOff = 0;

/*****
*****
* INITIALIZATION

*****
****/
public mainWindow()
{
    /* Initize Components */
    InitializeComponent();
}

/*****
*****
* SEND TO SERVER

*****
****/
private void ClientSend(string text)
{
    if(isConnectedToServer == true)
    {
        try
        {

```

```
        byte[] data = ASCIIEncoding.ASCII.GetBytes(text);
        streamSocket.Write(data, 0, data.Length);
    }
    catch (Exception ex)
    {
        MessageBox.Show("Error: " + ex.Data, "Error", MessageBoxButtons.OK,
                        MessageBoxIcon.Error);
    }
}
else
{
    MessageBox.Show("Hasn't connected to Server yet!", "Transaction",
MessageBoxButtons.OK,
                        MessageBoxIcon.Error);
}
}
```

```
=====
=====
* BUTTON "TEST"
=====
```

```
=====
=====
private void buttonTest_Click(object sender, EventArgs e)
{
    if(isConnectedToServer == true)
    {
        ClientSend(textBoxTest.Text);
        MessageBox.Show("Send to Server successfully!", "Transaction",
MessageBoxButtons.OK,
                        MessageBoxIcon.Information);
    }
    else
    {
        MessageBox.Show("Hasn't connected to Server yet!", "Transaction",
MessageBoxButtons.OK,
                        MessageBoxIcon.Error);
    }
}
```

```
=====
=====
* BUTTON "CONNECT"
```

```
*****
*****/
private void buttonConnect_Click(object sender, EventArgs e)
{
    uint varAttemp = 0; // Indicate how many times Client try to connect with Server
    try
    {
        /* Declare */
        clientSocket = new TcpClient();
        IPAddress address = IPAddress.Parse(tbIP.Text);
        clientSocket.Connect(address, Convert.ToInt16(tbPort.Text));

        /* Try to connect with Server */
        while (!clientSocket.Connected)
        {
            varAttemp++;
            if (varAttemp == 10)
            {
                isConnectedToServer = false;
                MessageBox.Show("Can't connect to Server", "Connection",
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Error);
                break;
            }
            clientSocket.Connect(address, Convert.ToInt16(tbPort.Text));
        }

        /* If can't try, break */
        if (varAttemp < 10)
        {
            MessageBox.Show("Connected to Server", "Connection",
                MessageBoxButtons.OK,
                MessageBoxIcon.Information);
            streamSocket = clientSocket.GetStream();
            isConnectedToServer = true;

            String str = "SERVER GUI";
            byte[] data = ASCIIEncoding.ASCII.GetBytes(str);
            streamSocket.Write(data, 0, data.Length);
        }
    }
    /* Exception */
    catch (Exception ex)
    {
        MessageBox.Show("Error: " + ex.Data, "Error", MessageBoxButtons.OK,
            MessageBoxIcon.Error);
    }
}
```

```
    }  
}
```

```
*****  
*****
```

```
* BUTTON "CLOSE"
```

```
*****  
*****
```

```
*****
```

```
private void buttonClose_Click(object sender, EventArgs e)  
{  
    /* Turn off all buttons */  
    buttonTurnOff(btn1);  
    buttonTurnOff(btn2);  
    buttonTurnOff(btn3);  
    buttonTurnOff(btn4);  
    buttonTurnOff(btn5);  
    buttonTurnOff(btn6);  
    buttonTurnOff(btn7);  
    buttonTurnOff(btn8);  
    buttonTurnOff(btn9);  
    buttonTurnOff(btn10);  
    buttonTurnOff(btn11);  
    buttonTurnOff(btn12);  
  
    /* Close connection */  
    clientSocket.Close();  
    MessageBox.Show("Disconnected Server", "Connection",  
    MessageBoxButtons.OK,  
        MessageBoxIcon.Information);  
    isConnectedToServer = false;  
}
```

```
*****  
*****
```

```
* "ENTER" KEY FOR "TEST" BUTTON
```

```
*****  
*****
```

```
*****
```

```
private void keyTest(object sender, KeyEventArgs e)  
{  
    if (e.KeyCode == Keys.Enter)  
    {
```

```
        buttonTest.PerformClick();
    }
else if (e.KeyCode == Keys.NumPad1 && e.Modifiers == Keys.Control)
{
    btn1.PerformClick();
}
else if (e.KeyCode == Keys.NumPad2 && e.Modifiers == Keys.Control)
{
    btn2.PerformClick();
}
else if (e.KeyCode == Keys.NumPad3 && e.Modifiers == Keys.Control)
{
    btn3.PerformClick();
}
else if (e.KeyCode == Keys.NumPad4 && e.Modifiers == Keys.Control)
{
    btn4.PerformClick();
}
else if (e.KeyCode == Keys.NumPad5 && e.Modifiers == Keys.Control)
{
    btn5.PerformClick();
}
else if (e.KeyCode == Keys.NumPad6 && e.Modifiers == Keys.Control)
{
    btn6.PerformClick();
}
else if (e.KeyCode == Keys.NumPad7 && e.Modifiers == Keys.Control)
{
    btn7.PerformClick();
}
else if (e.KeyCode == Keys.NumPad8 && e.Modifiers == Keys.Control)
{
    btn8.PerformClick();
}
else if (e.KeyCode == Keys.Q && e.Modifiers == Keys.Control)
{
    btn9.PerformClick();
}
else if (e.KeyCode == Keys.W && e.Modifiers == Keys.Control)
{
    btn10.PerformClick();
}
else if (e.KeyCode == Keys.E && e.Modifiers == Keys.Control)
{
    btn11.PerformClick();
}
else if (e.KeyCode == Keys.R && e.Modifiers == Keys.Control)
```

```
{  
    btn12.PerformClick();  
}  
else if (e.KeyCode == Keys.D1 && e.Modifiers == Keys.Control)  
{  
    btn1.PerformClick();  
}  
else if (e.KeyCode == Keys.D2 && e.Modifiers == Keys.Control)  
{  
    btn2.PerformClick();  
}  
else if (e.KeyCode == Keys.D3 && e.Modifiers == Keys.Control)  
{  
    btn3.PerformClick();  
}  
else if (e.KeyCode == Keys.D4 && e.Modifiers == Keys.Control)  
{  
    btn4.PerformClick();  
}  
else if (e.KeyCode == Keys.D5 && e.Modifiers == Keys.Control)  
{  
    btn5.PerformClick();  
}  
else if (e.KeyCode == Keys.D6 && e.Modifiers == Keys.Control)  
{  
    btn6.PerformClick();  
}  
else if (e.KeyCode == Keys.D7 && e.Modifiers == Keys.Control)  
{  
    btn7.PerformClick();  
}  
else if (e.KeyCode == Keys.D8 && e.Modifiers == Keys.Control)  
{  
    btn8.PerformClick();  
}  
else if (e.KeyCode == Keys.F1 && e.Modifiers == Keys.Control)  
{  
    btn9.PerformClick();  
}  
else if (e.KeyCode == Keys.F2 && e.Modifiers == Keys.Control)  
{  
    btn10.PerformClick();  
}  
else if (e.KeyCode == Keys.F3 && e.Modifiers == Keys.Control)  
{  
    btn11.PerformClick();  
}
```

```

        else if (e.KeyCode == Keys.F4 && e.Modifiers == Keys.Control)
    {
        btn12.PerformClick();
    }

}

/*************************************
*****
* "ENTER" KEY FOR "CONNECT" BUTTON
*****
************************************/


private void keyConnection(object sender, KeyEventArgs e)
{
    if (e.KeyCode == Keys.Enter)
    {
        btConnect.PerformClick();
    }
    else if (e.KeyCode == Keys.NumPad1 && e.Modifiers == Keys.Control)
    {
        btn1.PerformClick();
    }
    else if (e.KeyCode == Keys.NumPad2 && e.Modifiers == Keys.Control)
    {
        btn2.PerformClick();
    }
    else if (e.KeyCode == Keys.NumPad3 && e.Modifiers == Keys.Control)
    {
        btn3.PerformClick();
    }
    else if (e.KeyCode == Keys.NumPad4 && e.Modifiers == Keys.Control)
    {
        btn4.PerformClick();
    }
    else if (e.KeyCode == Keys.NumPad5 && e.Modifiers == Keys.Control)
    {
        btn5.PerformClick();
    }
    else if (e.KeyCode == Keys.NumPad6 && e.Modifiers == Keys.Control)
    {
        btn6.PerformClick();
    }
    else if (e.KeyCode == Keys.NumPad7 && e.Modifiers == Keys.Control)
    {

```

```
        btn7.PerformClick();
    }
else if (e.KeyCode == Keys.NumPad8 && e.Modifiers == Keys.Control)
{
    btn8.PerformClick();
}
else if (e.KeyCode == Keys.Q && e.Modifiers == Keys.Control)
{
    btn9.PerformClick();
}
else if (e.KeyCode == Keys.W && e.Modifiers == Keys.Control)
{
    btn10.PerformClick();
}
else if (e.KeyCode == Keys.E && e.Modifiers == Keys.Control)
{
    btn11.PerformClick();
}
else if (e.KeyCode == Keys.R && e.Modifiers == Keys.Control)
{
    btn12.PerformClick();
}
else if (e.KeyCode == Keys.D1 && e.Modifiers == Keys.Control)
{
    btn1.PerformClick();
}
else if (e.KeyCode == Keys.D2 && e.Modifiers == Keys.Control)
{
    btn2.PerformClick();
}
else if (e.KeyCode == Keys.D3 && e.Modifiers == Keys.Control)
{
    btn3.PerformClick();
}
else if (e.KeyCode == Keys.D4 && e.Modifiers == Keys.Control)
{
    btn4.PerformClick();
}
else if (e.KeyCode == Keys.D5 && e.Modifiers == Keys.Control)
{
    btn5.PerformClick();
}
else if (e.KeyCode == Keys.D6 && e.Modifiers == Keys.Control)
{
    btn6.PerformClick();
}
else if (e.KeyCode == Keys.D7 && e.Modifiers == Keys.Control)
```

```

    {
        btn7.PerformClick();
    }
    else if (e.KeyCode == Keys.D8 && e.Modifiers == Keys.Control)
    {
        btn8.PerformClick();
    }
    else if (e.KeyCode == Keys.F1 && e.Modifiers == Keys.Control)
    {
        btn9.PerformClick();
    }
    else if (e.KeyCode == Keys.F2 && e.Modifiers == Keys.Control)
    {
        btn10.PerformClick();
    }
    else if (e.KeyCode == Keys.F3 && e.Modifiers == Keys.Control)
    {
        btn11.PerformClick();
    }
    else if (e.KeyCode == Keys.F4 && e.Modifiers == Keys.Control)
    {
        btn12.PerformClick();
    }
}

```

```

/******************
*****
* MAKE FRAME FOR SEND TO SERVER
*****
*****************/
private void makeFrame (uint varBtnNumber, uint btnCmd)
{
    strFrame = "";
    strFrame += "/";
    strFrame += "<";
    strFrame += varBtnNumber.ToString();
    strFrame += btnCmd.ToString();
    strFrame += ">";
    strFrame += "\r";
    strFrame += "\n";
    strFrame += "\0";
}

```

```

*****
*****  

* DETECT WHETHER BUTTON IS TURNED ON  

*****  

*****  

*****  

private bool isButtonTurnOn (Button btn)
{
    if(btn.BackColor == btnColorTurnOn)
    {
        return true;
    }
    else
    {
        return false;
    }
}

*****
*****  

* TURN ON BUTTON  

*****  

*****  

private void buttonTurnOn (Button btn)
{
    btn.BackColor = btnColorTurnOn;
}

*****
*****  

* TURN OFF BUTTON  

*****  

*****  

private void buttonTurnOff(Button btn)
{
    btn.BackColor = btnColorTurnOff;
}

```

```

*****
*****
*



*****



private void btn1_Click(object sender, EventArgs e)
{
    if(isConnectedToServer == true)
    {
        if (!isButtonTurnOn(btn1))
        {
            buttonTurnOn(btn1);
            makeFrame(1, btnCmdTurnOn);
            ClientSend(strFrame);
            strFrame = "";
        }
        else
        {
            buttonTurnOff(btn1);
            makeFrame(1, btnCmdTurnOff);
            ClientSend(strFrame);
            strFrame = "";
        }
    }
    else
    {
        MessageBox.Show("Hasn't connected to Server yet!", "Transaction",
MessageBoxButtons.OK,
MessageBoxIcon.Error);
    }
}

*****



*****



private void btn2_Click(object sender, EventArgs e)
{
    if(isConnectedToServer == true)
    {
        if (!isButtonTurnOn(btn2))

```

```

        {
            buttonTurnOn(btn2);
            makeFrame(2, btnCmdTurnOn);
            ClientSend(strFrame);
        }
        else
        {
            buttonTurnOff(btn2);
            makeFrame(2, btnCmdTurnOff);
            ClientSend(strFrame);
        }
    }
    else
    {
        MessageBox.Show("Hasn't connected to Server yet!", "Transaction",
        MessageBoxButtons.OK,
        MessageBoxIcon.Error);
    }
}

/*****
*****
*
*****
****/
private void btn3_Click(object sender, EventArgs e)
{
    if(isConnectedToServer == true)
    {
        if (!isButtonTurnOn(btn3))
        {
            buttonTurnOn(btn3);
            makeFrame(3, btnCmdTurnOn);
            ClientSend(strFrame);
        }
        else
        {
            buttonTurnOff(btn3);
            makeFrame(3, btnCmdTurnOff);
            ClientSend(strFrame);
        }
    }
    else
    {

```

```

        MessageBox.Show("Hasn't connected to Server yet!", "Transaction",
MessageBoxButtons.OK,
        MessageBoxIcon.Error);
    }
}

/******************
*****
*
*****/
private void btn4_Click(object sender, EventArgs e)
{
    if(isConnectedToServer == true)
    {
        if (!isButtonTurnOn(btn4))
        {
            buttonTurnOn(btn4);
            makeFrame(4, btnCmdTurnOn);
            ClientSend(strFrame);
        }
        else
        {
            buttonTurnOff(btn4);
            makeFrame(4, btnCmdTurnOff);
            ClientSend(strFrame);
        }
    }
    else
    {
        MessageBox.Show("Hasn't connected to Server yet!", "Transaction",
MessageBoxButtons.OK,
        MessageBoxIcon.Error);
    }
}

/******************
*****
*
*****/

```

```

private void btn5_Click(object sender, EventArgs e)
{
    if(isConnectedToServer == true)
    {
        if (!isButtonTurnOn(btn5))
        {
            buttonTurnOn(btn5);
            makeFrame(5, btnCmdTurnOn);
            ClientSend(strFrame);
        }
        else
        {
            buttonTurnOff(btn5);
            makeFrame(5, btnCmdTurnOff);
            ClientSend(strFrame);
        }
    }
    else
    {
        MessageBox.Show("Hasn't connected to Server yet!", "Transaction",
        MessageBoxButtons.OK,
        MessageBoxIcon.Error);
    }
}

/******************
*****
*
*****/
private void btn6_Click(object sender, EventArgs e)
{
    if(isConnectedToServer == true)
    {
        if (!isButtonTurnOn(btn6))
        {
            buttonTurnOn(btn6);
            makeFrame(6, btnCmdTurnOn);
            ClientSend(strFrame);
        }
        else
        {
            buttonTurnOff(btn6);
            makeFrame(6, btnCmdTurnOff);
        }
    }
}

```

```

        ClientSend(strFrame);
    }
}
else
{
    MessageBox.Show("Hasn't connected to Server yet!", "Transaction",
MessageBoxButtons.OK,
    MessageBoxIcon.Error);
}
}

/******************
*****
*/
private void btn7_Click(object sender, EventArgs e)
{
    if(isConnectedToServer == true)
    {
        if (!isButtonTurnOn(btn7))
        {
            buttonTurnOn(btn7);
            makeFrame(7, btnCmdTurnOn);
            ClientSend(strFrame);
        }
        else
        {
            buttonTurnOff(btn7);
            makeFrame(7, btnCmdTurnOff);
            ClientSend(strFrame);
        }
    }
    else
    {
        MessageBox.Show("Hasn't connected to Server yet!", "Transaction",
MessageBoxButtons.OK,
    MessageBoxIcon.Error);
}
}

```

```

/******************
*****
*
*****/
private void btn8_Click(object sender, EventArgs e)
{
    if(isConnectedToServer == true)
    {
        if (!isButtonTurnOn(btn8))
        {
            buttonTurnOn(btn8);
            makeFrame(8, btnCmdTurnOn);
            ClientSend(strFrame);
        }
        else
        {
            buttonTurnOff(btn8);
            makeFrame(8, btnCmdTurnOff);
            ClientSend(strFrame);
        }
    }
    else
    {
        MessageBox.Show("Hasn't connected to Server yet!", "Transaction",
        MessageBoxButtons.OK,
        MessageBoxIcon.Error);
    }
}

private void Control_Enter(object sender, EventArgs e)
{
}

private void btn9_Click(object sender, EventArgs e)
{
    if (isConnectedToServer == true)
    {
        if (!isButtonTurnOn(btn9))
        {
            buttonTurnOn(btn9);
            makeFrame(9, btnCmdTurnOn);
            ClientSend(strFrame);
            strFrame = "";
        }
    }
}

```

```

        }
    else
    {
        buttonTurnOff(btn9);
        makeFrame(9, btnCmdTurnOff);
        ClientSend(strFrame);
        strFrame = "";
    }
}
else
{
    MessageBox.Show("Hasn't connected to Server yet!", "Transaction",
    MessageBoxButtons.OK,
    MessageBoxIcon.Error);
}
}

private void btn10_Click(object sender, EventArgs e)
{
    if (isConnectedToServer == true)
    {
        if (!isButtonTurnOn(btn10))
        {
            buttonTurnOn(btn10);
            makeFrame(10, btnCmdTurnOn);
            ClientSend(strFrame);
            strFrame = "";
        }
        else
        {
            buttonTurnOff(btn10);
            makeFrame(10, btnCmdTurnOff);
            ClientSend(strFrame);
            strFrame = "";
        }
    }
    else
    {
        MessageBox.Show("Hasn't connected to Server yet!", "Transaction",
        MessageBoxButtons.OK,
        MessageBoxIcon.Error);
    }
}

private void btn11_Click(object sender, EventArgs e)
{
    if (isConnectedToServer == true)

```

```

    {
        if (!isButtonTurnOn(btn11))
        {
            buttonTurnOn(btn11);
            makeFrame(11, btnCmdTurnOn);
            ClientSend(strFrame);
            strFrame = "";
        }
        else
        {
            buttonTurnOff(btn11);
            makeFrame(11, btnCmdTurnOff);
            ClientSend(strFrame);
            strFrame = "";
        }
    }
    else
    {
        MessageBox.Show("Hasn't connected to Server yet!", "Transaction",
        MessageBoxButtons.OK,
        MessageBoxIcon.Error);
    }
}

private void btn12_Click(object sender, EventArgs e)
{
    if (isConnectedToServer == true)
    {
        if (!isButtonTurnOn(btn12))
        {
            buttonTurnOn(btn12);
            makeFrame(12, btnCmdTurnOn);
            ClientSend(strFrame);
            strFrame = "";
        }
        else
        {
            buttonTurnOff(btn12);
            makeFrame(12, btnCmdTurnOff);
            ClientSend(strFrame);
            strFrame = "";
        }
    }
    else
    {
        MessageBox.Show("Hasn't connected to Server yet!", "Transaction",
        MessageBoxButtons.OK,

```

```

        MessageBoxIcon.Error);
    }

private void groupBox1_Enter(object sender, EventArgs e)
{
}

private void mainWindow_Load(object sender, EventArgs e)
{
    this.label1.BackColor = Color.Transparent;
    this.label2.BackColor = Color.Transparent;
    this.groupBox4.BackColor = Color.Transparent;
    this.groupBox5.BackColor = Color.Transparent;
}

private void groupBox5_Enter(object sender, EventArgs e)
{
}

private void groupBox4_Enter(object sender, EventArgs e)
{
}

private void pictureBox1_Click(object sender, EventArgs e)
{
}

/*
*****
*****
*/
}

}

```

Code gửi tập lệnh AT đến ESP8266:

```

#define USE_CHIP00

/*****
*****
* INCLUDEs
*****
****/

#include "HEADER00.h"

#ifndef USE_CHIP00
/*****
*****
* GLOBALs
*****
****/

```



```

/*****
*****
* MAIN
*****
****/

```



```

void main (void)
{
    //WDT-//
    wdtSetup(WDT_OFF, WDT_TIME_8MS);
    //=====

    //CLK-//
    clkSetup(CLK_DCO_1MHZ);
    //=====

    //GPIO-//
    gpioSetup(PORT1, PIN_LED_RED, PIN_LED_RED);
    //=====

    //TIMER-//
    timerASetup(TIMER_A_0, MODE_UP, ID_3, INT_OVF_ON, 0, 65000);
    wifi00StopTimerCount();
    //=====

    //UART-//

```

```

uartSetup(CLKS_SMCLK_1MHZ, BAUD_9600, INT_RXD_ON);
//=====
=====

//ESP8266-
esp8266Setup(ESP8266_PORT_1, PIN_WIFI_RST, PIN_WIFI_EN);
wifi00StartTimerCount(65000);
while(!wifi00CheckATCmdComplete("ready")) wifi00StartTimerCount(65000);

RESETUP_ESP:
esp8266CheckModule(); // Send AT command
wifi00StartTimerCount(65000); // Start Timer for counting timeout
while(!wifi00CheckATCmdComplete("OK")) wifi00StartTimerCount(65000); // Loop for
detecting successful progress

esp8266ConnectWifi("0988851122.", "875269Abc");
wifi00StartTimerCount(65000);
while(!wifi00CheckATCmdComplete("OK")) wifi00StartTimerCount(65000);

esp8266MultiConnection();
wifi00StartTimerCount(65000);
while(!wifi00CheckATCmdComplete("OK")) wifi00StartTimerCount(65000);

esp8266MakeServer(8000);
wifi00StartTimerCount(65000);
while(!wifi00CheckATCmdComplete("OK")) wifi00StartTimerCount(65000);

esp8266GetIP();
wifi00StartTimerCount(65000);
while(!wifi00CheckATCmdComplete("OK")) wifi00StartTimerCount(65000);

wifi00ClearBuffer(arrUART00, 256);
sRegisterStatus00.bIsEnoughWifiData = false;
//-----
/* LOOP */
while(1)
{
    /* Send TXD when receiving enough data from WiFi */
    if(sRegisterStatus00.bIsEnoughWifiData)
    {
        fw00DetectButton(arrUART00);
        cmn00Send();

        sRegisterStatus00.bIsEnoughWifiData = false;
        varCountUART00 = 0;
    }
}

```

```

/* If an incident Reset occurs with ESP8266 */
if(arrUART00[varCountUART00 - 1] == 'y' &&
    arrUART00[varCountUART00 - 2] == 'd' &&
    arrUART00[varCountUART00 - 3] == 'a' &&
    arrUART00[varCountUART00 - 4] == 'e' &&
    arrUART00[varCountUART00 - 5] == 'r') goto RESETUP_ESP;
}

*******/

*      INTERRUPT SERVICE ROUTINES

*****  

*****  

/* UART RXD */  

#pragma vector = USCIAB0RX_VECTOR  

_interrupt void rxdIsr (void)  

{  

    /* Read RXD */  

    varUART00 = rxdChar();  

    gpioHigh(PORT1, LED_RED);  

    /* Process data */  

    if(varUART00 == '/') varCountUART00 = 0; // If  

meeting the first '/', reset counter index  

    arrUART00[varCountUART00++] = varUART00; // Store  

in arrUART00  

    if(arrUART00[varCountUART00 - 1] == '\0' &&
        arrUART00[varCountUART00 - 2] == '\n' &&
        arrUART00[varCountUART00 - 3] == 'r' &&
        arrUART00[varCountUART00 - 4] == '>')  

    { // If  

enough needed data, start processing it  

        sRegisterStatus00.bIsEnoughWifiData = true;  

    }  

    if(varUART00 == '\0') // If meet '\0', set  

IsNullCharacter and stop timer
    {
        sRegisterStatus00.bIsNullCharacter = true;
        wifi00StopTimerCount();
    }
}

```

```

/* Escape ISR*/
gpioLow(PORT1, LED_RED);
}

//-----
/* TIMERA0 */
#pragma vector = TIMER0_A1_VECTOR
__interrupt void TIMER0_A1_ISR (void)
{
    if(TA0IV) sRegisterStatus00.bIsNullCharacter = true;      // If overflow, escape
while loop
}

/*
***** END OF MAIN00.c
***** */

#endif /* USE_CHIP00 */

#include "ESP8266_1_0_1.h"

/*
***** PUBLICs
***** */

/*
***** PRIVATEs
***** */

static volatile     uint8_t*      ESP8266_PORT;
static             uint16_t      ESP8266_PIN_RST;
static             uint16_t      ESP8266_PIN_EN;

/*
***** FUNCTIONS
***** */

```

```
*****
 ****
/*
 * Function: esp8266Setup
 * Purpose   :
 * Input     :
 * Output    :      None
 */
void
esp8266Setup (volatile uint8_t* ESP8266_PORT_, const uint16_t ESP8266_PIN_RST_,
                           const uint16_t ESP8266_PIN_EN_)

{
    /* Assign pins */
    ESP8266_PORT    = ESP8266_PORT_;
    ESP8266_PIN_RST = ESP8266_PIN_RST_;
    ESP8266_PIN_EN  = ESP8266_PIN_EN_;

    /* Setup GPIO */
    if(ESP8266_PORT == ESP8266_PORT_1)
        gpioSetup(PORT1, ESP8266_PIN_RST | ESP8266_PIN_EN,
ESP8266_PIN_RST | ESP8266_PIN_EN);
    else if(ESP8266_PORT == ESP8266_PORT_2)
        gpioSetup(PORT2, ESP8266_PIN_RST | ESP8266_PIN_EN,
ESP8266_PIN_RST | ESP8266_PIN_EN);

    /* Enable module */
    *ESP8266_PORT |= ESP8266_PIN_EN;

    /* Reset by Hardware */
    esp8266ResetHardware();
}

//-----
/*
 * Function:
 * Purpose   :
 * Input     :
 * Output    :
 */
bool
esp8266CheckModule(void)
{
    txdStr(ESP8266_AT_CHECK_MODULE);
    return false;
}
//-----
```

```

/*
 * Function:
 * Purpose    :
 * Input      :
 * Output     :
 */
void
esp8266ResetHardware(void)
{
    *ESP8266_PORT    &= ~ESP8266_PIN_RST;
    __delay_cycles(1000);
    *ESP8266_PORT    |=    ESP8266_PIN_RST;
    __delay_cycles(1000);
}
//-----
/*
 * Function:
 * Purpose    :
 * Input      :
 * Output     :
 */
bool
esp8266ResetSoftware(void)
{
    txdStr(ESP8266_AT_RESET);
    return false;
}
//-----
/*
 * Function:
 * Purpose    :
 * Input      :
 * Output     :
 */
bool
esp8266CheckFirmware(void)
{
    txdStr(ESP8266_AT_CHECK_FIRMWARE);
    return false;
}
//-----
/*
 * Function:
 * Purpose    :
 * Input      :
 * Output     :
 */

```

```

bool
esp8266ConnectWifi(char strSSID[], char strPassword[])
{
    char strBuff[50];
    strcpy(strBuff, ESP8266_AT_CONNECT_WIFI);
    strcat(strBuff, "\"");
    strcat(strBuff, strSSID);
    strcat(strBuff, "\",\"");
    strcat(strBuff, strPassword);
    strcat(strBuff, "\\r\\n");

    txdStr((uint8_t*)strBuff);
    return false;
}
//-----
/*
* Function:
* Purpose   :
* Input     :
* Output    :
*/
bool
esp8266DisconnectWifi(void)
{
    txdStr(ESP8266_AT_CONNECT_WIFI);
    return false;
}
//-----
/*
* Function:
* Purpose   :
* Input     :
* Output    :
*/
bool
esp8266MakeServer(uint16_t varPort)
{
    char strPort[6];
    sprintf(strPort, "%d", varPort);

    char strBuff[25];
    strcpy(strBuff, ESP8266_AT_MAKE_SERVER);
    strcat(strBuff, strPort);
    strcat(strBuff, "\\r\\n");

    txdStr((uint8_t*)strBuff);
    return false;
}

```

```
    }
//-----
/*
 * Function:
 * Purpose   :
 * Input     :
 * Output    :
 */
bool
esp8266SingleConnection(void)
{
    txdStr(ESP8266_AT_SET_TCP_SINGLE);
    return false;
}
//-----
/*
 * Function:
 * Purpose   :
 * Input     :
 * Output    :
 */
bool
esp8266MultiConnection(void)
{
    txdStr(ESP8266_AT_SET_TCP_MULTI);
    return false;
}
//-----
/*
 * Function:
 * Purpose   :
 * Input     :
 * Output    :
 */
bool
esp8266GetIP(void)
{
    txdStr(ESP8266_AT_GET_IP);
    return false;
}
//-----
/*
 * Function:
 * Purpose   :
 * Input     :
 * Output    :
 */

```

```

bool
esp8266ConnectTCP(char strIPAddr[], uint16_t varPort)
{
    char strPort[6];
    sprintf(strPort, "%d", varPort);

    char strBuff[50];
    strcpy(strBuff, ESP8266_AT_CONNECT_TCP);
    strcat(strBuff, "\"TCP\\\"\"");
    strcat(strBuff, strIPAddr);
    strcat(strBuff, "\\");
    strcat(strBuff, strPort);
    strcat(strBuff, "\\r\\n");

    txdStr((uint8_t*)strBuff);
    return false;
}

//-----
/*
 * Function:
 * Purpose   :
 * Input     :
 * Output    :
 */
bool
esp8266DisconnectTCP(void)
{
    txdStr(ESP8266_AT_DISCONNECT_TCP);
    return false;
}

//-----
/*
 * Function:
 * Purpose   :
 * Input     :
 * Output    :
 */
bool
esp8266ChecStatus(void)
{
    txdStr(ESP8266_AT_CHECK_STATUS);
    return false;
}

```