**ĐẠI HỌC QUỐC GIA TP.HỒ CHÍ MINH**

TRƯỜNG ĐẠI HỌC BÁCH KHOA

KHOA ĐIỆN – ĐIỆN TỬ

**BỘ MÔN ĐIỆN TỬ**

---------------o0o---------------

****

**LUẬN VĂN TỐT NGHIỆP ĐẠI HỌC**

**THIẾT KẾ VÀ THỰC HIỆN MODBUS IOT GATEWAY**

**GVHD: ThS. Bùi Quốc Bảo**

**SVTH: Trần Ngọc Trâm**

**MSSV: 1713587**

**TP. HỒ CHÍ MINH, THÁNG NĂM 2021**

ĐẠI HỌC QUỐC GIA TP.HỒ CHÍ MINH CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM

TRƯỜNG ĐẠI HỌC BÁCH KHOA Độc lập – Tự do – Hạnh phúc.

-----✩----- -----✩-----

Số: \_\_\_\_\_\_ /BKĐT

Khoa: **Điện – Điện tử**

Bộ Môn: **Điện Tử**

N**HIỆM VỤ LUẬN VĂN TỐT NGHIỆP**

1. HỌ VÀ TÊN :Trần Ngọc Trâm MSSV: 1713587

1. NGÀNH: **ĐIỆN TỬ - VIỄN THÔNG** LỚP: DD17DV7
2. Đề tài:

THIẾT KẾ VÀ THỰC HIỆN MODBUS IOT GATEWAY

1. Nhiệm vụ:

- Tìm hiểu các giao thức truyền tải thông dụng trong các giải pháp giám sát điện mặt trời.  
- Lập trình kết nối IoT Gateway với các thiết bị đầu cuối.  
- Lập trình kết nối IoT Gateway vào IoT Server.

1. Ngày giao nhiệm vụ luận văn: ...............................
2. Ngày hoàn thành nhiệm vụ: ...................................
3. Họ và tên người hướng dẫn: Phần hướng dẫn

Thầy Bùi Quốc Bảo .....................................

................................................................. .....................................

Nội dung và yêu cầu LVTN đã được thông qua Bộ Môn.

*Tp.HCM, ngày…... tháng….. năm 20*

**CHỦ NHIỆM BỘ MÔN NGƯỜI HƯỚNG DẪN CHÍNH**

**PHẦN DÀNH CHO KHOA, BỘ MÔN:**

Người duyệt (chấm sơ bộ):.......................

Đơn vị:......................................................

Ngày bảo vệ : ...........................................

Điểm tổng kết: .........................................

Nơi lưu trữ luận văn: ...............................

***LỜI CẢM ƠN***

Không có sự thành công nào mà không gắn liền với những sự hỗ trợ, giúp đỡ dù ít hay nhiều, dù trực tiếp hay gián tiếp của người khác. Trong suốt thời gian từ khi bắt đầu học tập tại trường đến nay, em đã nhận được rất nhiều sự quan tâm, giúp đỡ của quý Thầy Cô, gia đình và bạn bè. Với lòng biết ơn sâu sắc nhất, chúng em xin gửi đến quý Thầy Cô ở Khoa Điện – Điện Tử trường Đại học Bách Khoa TP. Hồ Chí Minh đã cùng với kiến thức và tâm huyết của mình để truyền đạt vốn kiến thức quý báu cho em trong suốt thời gian học tập tại trường.

Và đặc biệt, trong thời gian làm luận văn này, nếu không có sự hướng dẫn của thầy hướng dẫn - ThS. Bùi Quốc Bảo, thì đề tài cũng em rất khó để hoàn thành được. Một lần nữa, em xin chân thành cảm ơn thầy.

Bên cạnh đó em xin chân cảm ơn các thành viên trong phòng thí nghiệm 209B3 đã tận tình giúp đỡ và động viên để em có thêm tự tin và động lực để hoàn thành nhiệm vụ được giao.

Mặc dù đã cố gắng để thực hiện đề tài này một cách hoàn chỉnh nhất, song do sự thiếu sót trong tư duy và hạn chế về kiến thức, kinh nghiệm nên không thể tránh khỏi những thiếu sót nhất định mà bản thân chưa thấy được. Em mong nhận được sự góp ý của quý Thầy Cô và các bạn để luận văn được hoàn chỉnh hơn.

Em xin chân thành cảm ơn!

*Tp. Hồ Chí Minh, ngày tháng năm 2021.*

**Sinh viên**

Trần Ngọc Trâm

**TÓM TẮT LUẬN VĂN**

Luận văn này trình bày về việc thiết kế Modbus IoT Gateway để thu thập dữ liệu và điều khiển thiết bị thông qua giao thức: Modbus serial. Luận văn gồm hai phần chính:  
Firmware của Gateway:- Lập trình các chức năng và kết nối các giao thức trên nền tảng FreeRTOS cho các thiết bị chạy vi điều khiển.  
- Tìm hiểu, lập trình ứng dụng các giao thức (TCP, MQTT ...) và các chuẩn đóng gói dữ liệu (JSON …) để kết nối đến IoT Server.  
Software:- Lập trình giao diện Qt trên máy tính để cài đặt và kiểm tra trạng thái của Gateway

**MỤC LỤC**

[1. GIỚI THIỆU 1](#_Toc310380267)

[1.1 Tổng quan 1](#_Toc310380268)

[1.2 Tình hình nghiên cứu trong và ngoài nước 1](#_Toc310380269)

[1.3 Nhiệm vụ luận văn 1](#_Toc310380270)

[2. LÝ THUYẾT 1](#_Toc310380271)

[3. THIẾT KẾ VÀ THỰC HIỆN PHẦN CỨNG 1](#_Toc310380272)

[4. THIẾT KẾ VÀ THỰC HIỆN PHẦN MỀM (NẾU CÓ) 2](#_Toc310380273)

[5. KẾT QUẢ THỰC HIỆN 2](#_Toc310380274)

[6. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 4](#_Toc310380275)

[6.1 Kết luận 4](#_Toc310380276)

[6.2 Hướng phát triển 4](#_Toc310380277)

[7. TÀI LIỆU THAM KHẢO 4](#_Toc310380278)

[8. PHỤ LỤC 4](#_Toc310380279)

DANH SÁCH HÌNH MINH HỌA

[Hình 5‑1 Kết quả thi công 3](#_Toc310380287)

[Hình 5‑2 Kết quả mô phỏng 3](#_Toc310380288)

**DANH SÁCH BẢNG SỐ LIỆU**

[Bảng 1 Thông số hệ thống 3](#_Toc310380293)

# GIỚI THIỆU

## Tổng quan

IoT là viết tắt của cụm từ tiếng Anh Internet of Things, dùng để chỉ các thiết bị vật lý được kết nối internet có khả năng thu thập dữ liệu, chia sẻ thông tin với độ bao phủ toàn cầu, nghĩa là bất cứ thiết bị vật lý nào có khả năng kết nối internet, thu thập, lưu giữ và chia sẻ thông tin thì đều là IoT.

A picture containing map

Description automatically generatedIoT đã và đang là động lực phát triển của mọi lĩnh vực công nghệ. Trong năm 2018: theo Stantista, thị trường nhà thông minh toàn cầu đạt gần 60 tỷ USD; theo Gartner, chi tiêu cho an ninh IoT đạt 547 triệu USD; theo IDC, chi tiêu trên toàn thế giới về IoT đạt 772,5 tỷ USD. McKinsey ước tính có 127 thiết bị mới kết nối với Internet mỗi ngày.  
IoT có tác động kinh tế hàng năm từ 3.9 nghìn tỷ USD đến 11.1 nghìn tỷ USD vào năm 2025. Trong công nghiệp, Accenture ước tính Internet Công nghiệp có thể mang đến 14.2 nghìn tỷ USD cho nền kinh tế toàn cầu vào năm 2030. Trong vận tải, Business Insider ước tính rằng các nhà sản xuất toàn cầu sẽ đầu tư 70 tỷ USD vào năm 2020. Từ đó, ta có thể thấy rõ tầm ảnh hưởng lớn của IoT, cùng với những lợi ích và sự phát triển không ngừng của nó.

Hình 1: Mô hình tổng quan IoT.

## Tình hình nghiên cứu trong và ngoài nước

IoT hiện đang là động lực phát triển của hầu hết các lĩnh vực công nghệ. Các giải pháp IoT cho phép các doanh nghiệp cải thiến các hệ thống hiện có giúp nâng cao kết nối với khách hàng và đối tác. Một số ứng dụng IoT dẫn đầu xu hướng công nghệ hiện nay có thể kể đến như:

Application

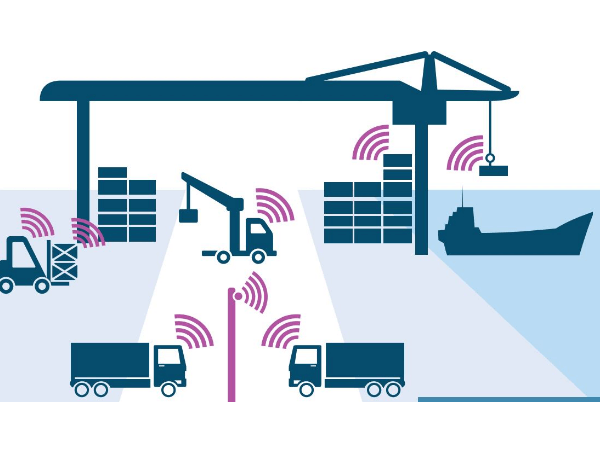
Description automatically generated with medium confidence- Công nghiệp thông minh (Smart industry): Một số máy móc công nghiệp có thể hư hỏng hoặc quá hạn sử dụng, cần sự can thiệp từ các kỹ sư. Việc tích hợp các cảm biến và các kết nối không dây cho phép các máy móc có thể tự động phản hồi thông tin kịp thời về cho người quản lý.

Hình: Minh họa công nghiệp thông minh

- Nông nghiệp thông minh (Smart agriculture): Với việc tích hợp các hệ thống cảm biến trên các cánh đồng, nông dân có thể nhận được các dữ liệu cần thiết để dự đoán về mùa màng, hoặc các hệ thống tự quản lý việc tưới tiêu, cung cấp ánh sáng.

A picture containing text, grass, building

Description automatically generatedHình: Minh họa nông nghiệp thông minh

- Vận chuyển thông minh (Logistics and transportation IoT): một trong những ứng dụng  
phổ biến là gán nhãn cho các xe container bằng cách sử dụng RFID. Điều này cho phép  
các công ty theo dõi quá trình vận chuyển hàng hóa, chọn ra đường đi tốt nhất.

Hình. Minh họa vận chuyển thông minh

Các công ty dẫn đầu công nghiệp IoT trên thế giới hiện nay có thể kể đến như Google,  
Microsoft, Amazon, IBM, Oracles... Trong đó, Google, Amazon và Microsoft chủ yếu cung cấp các nền tảng Cloud hỗ trợ lưu trữ dữ liệu (Google Cloud, Amazon Web Services, Microsoft Azure, IBM và Oracle thiên về các thiết bị Gateway và phần cứng hoặc ứng dụng công nghệ AI, Block Chain...



Hình. Ba nền tảng cung cấp các giải pháp IoT Cloud Platform lớn nhất hiện nay (Từ trái sang: Google Cloud Platform, Amazon Web Services, Microsoft Azure)

Trong việc phát triển, tích hợp các giải pháp IoT hiện nay, việc xây dựng, thiết kế các IoT Gateway đang được phát triển với nhiều triển vọng sẽ cung cấp các giải pháp xử lí dữ liệu đa dạng, mạnh mẽ cho điện toán biên, giúp giảm sức nặng đối với các nền tảng cloud. Các dịch vụ IoT Gateway lớn hiện nay có thể kể đến như AWS IoT, Microsoft Azure IoT… Một số nền tảng IoT mã nguồn mở như EdgeX Foundry, ThingsBoard IoT Gateway thì tập trung vào giải pháp phần mềm hỗ trợ cho việc kết nối IoT Gateway với các thiết bị phần cứng. Các IoT Gateway của các nền tảng trên hỗ trợ nhiều giao thức kết nối khác nhau, đơn giản hóa cho việc kết nối giữa Gateway và các end-device.

**Graphical user interface

Description automatically generated**

Hình: Nền tảng mở EdgeX Foundry cung cấp giải pháp điện toán biên mạnh mẽ, tích hợp nhiều thiết bị

A picture containing indoor, items, electronic, arranged

Description automatically generatedTại Việt Nam, nhu cầu về những giải pháp IoT đang rất lớn, để hỗ trợ cho các hệ thống  
IoT công-nông nghiệp như Nông nghiệp thông minh, Vận tải thông minh và Thành phố thông  
minh; tuy nhiên, những thương hiệu IoT lớn ở Việt Nam như Lumi, Javis Home… họ tập trung vào thiết kế và lắp đặt các thiết bị Nhà ở thông minh với những gói dịch vụ và sản  
phẩm đầu cuối nhất định, gây bất tiện cho người dùng trong việc nâng cấp và thiếu đa ngtrong việc chọn lựa. Một số công ty khác lại tập trung phát triển IoT Gateway chỉ hỗ trợ một vài giao thức nhất định, phù hợp với các dữ liệu mà Gateway thu thập trong các lĩnh vực khác nhau như là công nghiệp, nông nghiệp, nhà thông minh, năng lượng, y tế…

Hình: Giải pháp nhà thông minh Javis Home

Qua do cho thấy, để đạt được tối ưu giá trị từ IoT, việc làm chủ một nền tảng IoT (IoT  
platform) là cần thiết. Nền tảng IoT tập hợp những ứng dụng thực hiện việc liên kết giữa các thiết bị IoT và trung tâm dữ liệu từ đó tạo ra một hệ thống cung cấp đầy đủ thông tin được thu thập từ các thiết bị IoT (cảm biến, thiết điện tử tiêu dùng, thiết bị di động…) khác nhau để tạo và quản lý ứng dụng; lưu trữ và phân tích dữ liệu cũng như điều khiển từ xa các thiết bị IoT.  
Như vậy, việc tự xây dựng một nền tảng IoT hoàn chỉnh, hỗ trợ từ phần cứng đến phần mềm, cho phép quản lý, điều khiển, và giám sát là mục tiêu cần thiết trong việc phát triển công nghệ IoT hiện nay.

### 1.2.1 Một số hệ thống hiện nay

### 1.2.2 Đánh giá chung các hệ thống

### 1.2.3 Kết luận hướng nghiên cứu của luận văn

## Nhiệm vụ luận văn

Nhiệm vụ chính của luận văn là xây dựng một nền tảng IoT, trong đó tập trung xây dựng IoT Gateway thu thập dữ liệu theo các giao thức đặt ra và các phần mềm để kết nối các thiết bị đó với nền tảng ứng dụng trên server (nền tảng ứng dụng trên server được thực hiện bởi một sinh viên khác), cụ thể:

- Lập trình giao tiếp giữa Gateway với các thiết bị đầu cuối.  
- Lập trình kết nối các Gateway vào nền tảng IoT trên Server.

Từ các nhiệm vụ chính đó, cần phải thực hiện những nội dung:

- Nội dung 1: Tìm hiểu lý thuyết về các chuẩn giao tiếp, cách hoạt động, cách thức quản lý, các thông số, ưu nhược điểm khi triển khai trên các hệ thống cụ thể.  
- Nội dung 2: Đề xuất, lựa chọn các giải pháp trong từng ứng dụng cụ thể.  
- Nội dung 3: Phân tích, lựa chọn các thiết bị, phần cứng, cảm biến phù hợp cho hệ thống.  
- Nội dung 4: Tìm hiểu, tham khảo tài liệu liên quan và các thiết kế có sẵn. Từ đó đưa  
ra sơ đồ chi tiết cho hệ thống.  
- Nội dung 5: Tìm hiểu về hệ điều hành FreeRTOS và cách triển khai hệ thống đa tác  
vụ. Từ đó phát triển chương trình nhúng cho các thiết bị trên nền tảng này.  
- Nội dung 6: Tìm hiểu cách giao tiếp, điều khiển và trao đổi dữ liệu giữa các module  
truyền nhận (Modbus, Ethernet, Sdio …) từ đó viết trình điều khiển cho từng module trên vi điều khiển.  
- Nội dung 7: Thống nhất và chuẩn hoá các loại dữ liệu thu được từ các Gateway để giao tiếp với nền tảng IoT trên Server.  
- Nội dung 8: Kiểm thử, đo đạc các thông số của hệ thống. Sửa lỗi phần cứng và phần  
mềm.

### 1.3.1 Phạm vi luận văn

Xây dựng một nền tảng giám sát …. Trong đó tập trung xây dựng IoT Gateway thu thập dữ liệu theo các giao thức đặt ra và các phần mềm để kết nối các thiết bị đó với nền tảng ứng dụng trên server và thiết kế ứng dụng web để trực quan hóa dữ liệu thu được.

### 1.3.2 Nhiệm vụ luận văn

Về Modbus IoT Gateway:

- Lập trình giao tiếp giữa Gateway với các thiết bị đầu cuối.  
- Lập trình kết nối các Gateway vào nền tảng IoT trên Server.

Về ứng dụng Web:

- …

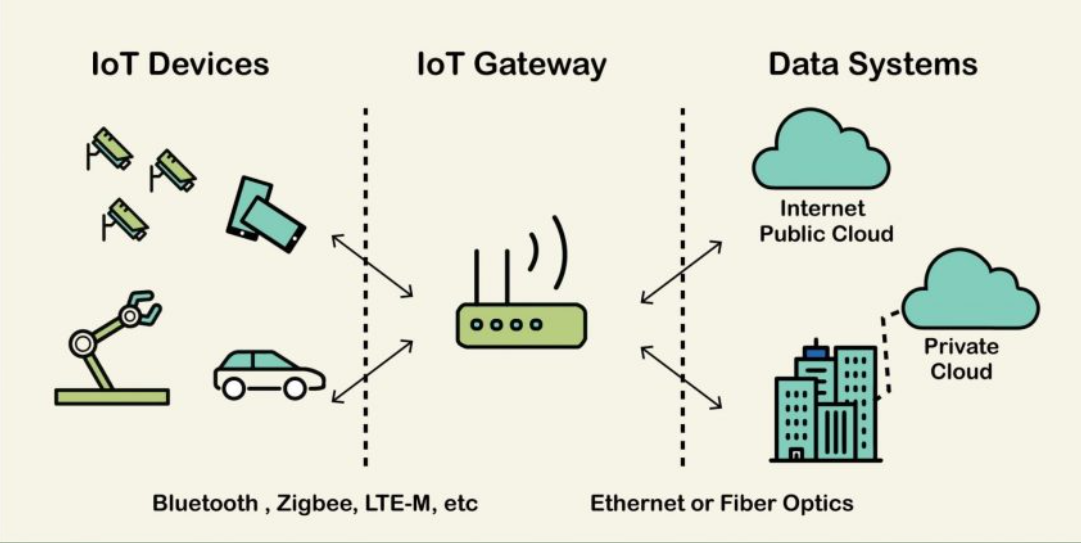
# LÝ THUYẾT

## 2.1 Tìm hiểu về các thiết bị IoT Gateway

### 2.1.1 Thiết bị IoT Gateway

- IoT Gateway là cầu nối giữa các thiết bị IoT (cảm biến, thiết bị, hệ thống) và mạng internet. Dữ liệu được truyền lên mạng internet sẽ đi qua gateway này. IoT Gateway có cùng nguyên tắc kết nối thông tin với các công nghệ khác nhau. Nó tạo ra một cầu nối giữa các cảm biến hay thiết bị truyền động IoT với Internet. Cổng kết nối IoT tổng hợp tất cả dữ liệu, dịch các giao thức của cảm biến và xử lý dữ liệu trước khi gửi.

- Trong truyền thông, mục đích chính của một gateway là cầu nối giữa các công nghệ truyền thông khác nhau. Các công nghệ này có thể khác nhau về cách kết nối, giao diện hoặc giao thức.



Hình: Mô hình cơ bản của một IoT Gateway

Các thiết bị IoT Gateway hiện nay đều có khả năng tổng hợp dữ liệu. Ví dụ: các cảm biến trong giao thông. Dữ liệu từ cảm biến gửi các dữ liệu thô đến IoT Gateway. Tại đây, các dữ liệu sẽ tuân theo các bước xử lý sau:

- Lọc dữ liệu thô

- Chuyển ngôn ngữ của các giao thức để mã hóa

- Gửi dữ liệu lên hệ thống xử lý trung tâm qua internet.

#### 2.1.1.1 IoT Gateway là gì?

#### 2.1.1.2 Các chức năng của IoT Gateway

**-** Data forwarding – chuyển tiếp dữ liệu: Gateway có trách nhiệm chuyển tiếp dữ liệu từ các lớp công nghệ thấp đến các lớp cao hơn trong kiến trúc nhà máy.

**-** Gateway management – quản lý cổng kết nối: Gateway là một yếu tố tích hợp quan trọng trong toàn bộ kiến trúc nhà máy. Một gateway không có kiểm soát và không quản lý được sẽ gây khó khăn trong việc triển khai hay mở rộng sau này.

**-** Device management – quản lý thiết bị: cho phép cấu hình runtime của các thiết bị, cài đặt trạng thái, chế độ chức năng, xác nhận lỗi và nhiều thứ khác.

**-** Data analysis – phân tích dữ liệu: IoT gateway giám sát và phân tích dữ liệu thời gian thực bằng cách sử dụng các thống kê và trí tuệ nhân tạo. Modun này hợp tác với device management để duy trì chất lượng dịch vụ, độ tin cậy và hoạt động lỗi. Các lỗi được phát hiện và các thiết bị lỗi được thay thế bằng các thiết bị dự phòng có sẵn. Phân tích dữ liệu với quản lý thiết bị cho phép hành vi tự cấu hình, thích ứng và mạnh mẽ liên quan đến lớp công nghệ được kết nối với cổng IoT. Mục tiêu là đạt được khả năng quản lý tự động các thiết bị mà không cần sự can thiệp của con người nhờ giao tiếp giữa máy với máy.

**-** Diagnostics – Chẩn đoán:  IoT Gateway phát hiện lỗi và các sai phạm trong toàn bộ lớp công nghệ và trong chính cổng IoT.

#### 2.1.1.3 Lợi ích sử dụng IoT Gateway

- IoT Gateway xử lý dữ kiệu tại biên, được gọi là điện toán biên (Edge Computing). Cổng IoT xử lý trước và lọc dữ liệu ra ở điểm cuối.

- Hỗ trợ bảo mật cho IoT: Với việc các thiết bị kết nối với IoT Gateway sẽ khiến cho việc tiếp xúc các thiết bị được hạn chế tối đa. Giảm thiểu khả năng có thể bị tin tặc tấn công qua các thiết bị, trong trường hợp này IoT Gateway là bức tường lửa để bảo vệ thông tin người dung.

- Xử lý các thông tin hỗn hợp từ các thiết bị IoT về một giao thức chuẩn nhất: Các cảm biến thu hập dữ liệu từ môi trường bên ngoài. Dữ liệu được thu về có nhiều nguồn khác nhau (chuyển động, ánh sáng, âm thanh….). Các nội dung của dữ liệu này sẽ được mã hóa về dạng dữ liệu chung của hệ thống để có thể giao tiếp với nhau một cách có hiệu quả.

Diagram

Description automatically generated

Hình: Dữ liệu được xử lí tại biên

#### 2.1.1.4 Một số sản phẩm IoT Gateway trên thị trường



Hình: Từ trái sang (Zigbee Gateway, Lorawan IoT Gateway, Modbus IoT Gateway)

Wiser Zigbee Gateway được Schneider Electric phát triển trong ứng dụng nhà thông minh. Gateway là bộ xử lí trung tâm cho hệ sinh thái nhà thông minh bao gồm các cảm biến, các ứng dụng được Schneider phát triển đi kèm.

LoRaWan IoT Gateway (Model: UG85) được Ursalink phát triển dùng để thu thập dữ liệu từ các thiết bị, cảm biến hỗ trợ giao thức Lorawan, dùng trong các ứng dụng đo đạc, tòa nhà thông minh, theo dõi, tự động hóa…

Modbus IoT Gateway (Model: S375E) được King Pigeon phát triển để thu thập dữ liệu từ các thiết bị công nghiệp (PLC, Inverter…), các cảm biến thu thập dữ liệu solar…

### 2.1.2 Các kết nối trong các thiết bị IoT Gateway thực tế

Các ứng dụng IoT Gateway đều có các ràng buộc rõ ràng về phạm vi kết nối (không dây, có dây), điện năng tiêu thụ, tốc độ truyền tải dữ liệu, độ tin cậy và bảo mật. Giao thức truyền thông tin có rất nhiều với nhiều chủng loại thiết bị khác nhau dành cho các

nhu cầu sử dụng khác nhau.

Logo, company name

Description automatically generated

Hình: Các giao thức phổ biến trong hệ thống IoT Gateway

Những giao thức phổ biến trong các hệ thống IoT Gateway hiện nay:  
- **LoRa**: LoRa là công nghệ giao tiếp ở khoảng cách xa, có thể lên đến hàng cây số mà  
không cần đến bộ khuếch đại. Ngoài ra, công suất tiêu thụ của các thiết bị sử dụng  
công nghệ LoRa là thấp, khoảng 0.1W. Công nghệ LoRa dễ bắt gặp trong các ứng dụng Sensor Network. Tuy nhiên, tốc độ truyền thấp khoảng 0.3 – 50Kbps.  
- **Wifi**: Wifi là giao thức thường được sử dụng để truyền hình ảnh, âm thanh nhờ băng  
thông lớn. Tuy nhiên, Wifi có phạm vi truyền sóng nhỏ và công suất tiêu thụ lớn  
dây), điện năng tiêu thụ, tốc độ truyền tải dữ liệu, độ tin cậy và bảo mật. Giao thức truyền thông tin có rất nhiều với nhiều chủng loại thiết bị khác nhau dành cho các nhu cầu khác nhau.

- **RS485**: Có tốc độ truyền tương đối cao và ổn định, tuy nhiên, RS485 là giao thức có  
dây nên chỉ phù hợp để kết nối các trạm trong mạng cục bộ.  
- **Ethernet**: là công nghệ truyền thông dùng để kết nối các mạng LAN cục bộ, có tốc độ cao hơn RS485, ít bị gián đoạn, độ bảo mật cao, tuy nhiên, giao thức này đòi hỏi các thiết bị quản lý trên nền tảng mạng và việc triển khai giao thức này lên các thiết bị  
nhúng khá phức tạp về phần cứng và phần mềm so với RS485.  
- **Zigbee**: Zigbee là một giao thức mở, sử dụng mạng lưới (mesh network) để mở rộng  
hệ thống. Zigbee phù hợp với hệ thống cần nhiều nút mạng. Tốc độ truyền khoảng 40-  
250Kbps. Cũng như các giao thức trên, công nghệ này cũng tiêu hao ít năng lượng.  
- **Z-Wave**: Tương tự như Zigbee, Z-Wave cũng sử dụng mạng lưới để mở rộng phạm vi kết nối. Ngoài ra, Z-Wave có hầu hết các ưu điểm của Zigbee và cải thiện được khoảng cách truyền sóng giữa các nút (lên đến 100m). Đây là công nghệ đang được chú ý và được các nhà sản xuát tập trung nhiều hơn.

Table

Description automatically generated

Hình: So sánh một số kết nối không dây

Từ đó có thể thấy, mỗi giao thức đều có điểm mạnh và yếu riêng trong từng điều kiện  
môi trường. Việc thu thập dữ liệu cảm biến, tổng hợp và truyền tải dữ liệu với một giao thức đơn thuần không thể đáp ứng được sự đa dạng về kiến trúc và địa hình ở Việt Nam. Thiết bị Wifi tốc độ cao nhưng khoảng cách hạn chế và tiêu hao năng lượng lớn, thiết bị Zigbee ít tiêu hao năng lượng hơn nhưng tốc độ chậm và để truyền được xa thì phải triển khai mạng mesh nhiều thiết bị. Những thiết bị có khả năng truyền xa như LoRa thì bị giới hạn về tốc độ, băng thông. Các thiết bị có khoảng cách rộng như GSM thì phải phụ thuộc vào phạm vi phủ sóng và độ ổn định của dịch vụ nhà mạng cung cấp.

Dựa trên đặc trưng của các giao thức trên, để đảm bảo một hệ thống cân bằng với các  
thông số khoảng cách, năng lượng, tốc độ và các đặc trưng về một IoT Gateway, trong đề tài sẽ triển khai 2 loại kết nối chính:  
- RS485: thích hợp dùng trong các ứng dụng có độ trễ thấp, trong mạng nội bộ. Trong  
đề tài sẽ thực hiện triển khai giao thức Modbus Serial dựa trên nền tảng vật lý của  
RS485 để thuận tiện hơn trong việc quản lý, điều khiển và thu thập dữ liệu từ các thiết  
bị. Ngoài ra, có thể mở rộng mạng lưới các thiết bị nhờ vào khả năng tương thích với  
các thiết bị bên ngoài theo tiêu chuẩn Modbus.  
- Ethernet: thích hợp trong các ứng dụng cần đảm bảo đường truyền ổn đinh, cung cấp mức độ bảo mật và kiểm soát mạng tốt hơn mạng LAN không dây. Trong đề tài sẽ thực hiện triển khai giao thức MQTT dựa trên nên tảng vật lý của Ethernet để thuận tiện hơn trong việc đảm bảo đường truyền kết nối mạng ổn định.

## 2.2 Giao thức Modbus

### 2.2.1 Giới thiệu

Modbus do Modicon (hiện nay thuộc Schneider Electric) [4] phát triển năm 1979, là một phương tiện truyền thông với nhiều thiết bị thông qua một cặp dây xoắn đơn. Ban đầu, nó hoạt động trên RS232, nhưng sau đó nó sử dụng cho cả RS485 để đạt tốc độ cao hơn, khoảng cách dài hơn, và mạng đa điểm (multi-drop). Modbus đã nhanh chóng trở thành tiêu chuẩn thông dụng trong ngành tự động hóa, và Modicon đã cho ra mắt công chúng như một protocol miễn phí.

### 2.2.2 Đặc điểm

Modbus là một hệ thống “Master - Slave”, “Master” được kết nối với một hay nhiều  
“Slave”. “Master” thường là một PLC, PC, DCS, hay RTU. “Slave” Modbus thường là các thiết bị hiện trường, tất cả được kết nối với mạng trong cấu hình multi-drop.   
Các thiết bị trên mạng Modbus không thể tạo ra kết nối, chúng chỉ có thể phản ứng. Nói cách khác, chúng “lên tiếng” chỉ khi được “nói tới”. Giao thức Modbus hỗ trợ tối đa 247 thiết bị Slave được kết nối nối tiếp vào Master (không qua các thiết bị chuyển tiếp). Với cổng kết nối RS232, Modbus có khoảng cách truyền tối đa là 200m; với cổng kết nối RS485 khoảng cách này là 1200m.

Nguyên tắc Master – Slave có những đặc điểm sau:

- Mỗi lần chỉ có một Master được kết nối mạng

- Chỉ có Master mới có thể bắt đầu giao tiếp và gửi yêu cầu cho các Slave

- Modbus có thể giải quyết từng Slave, bằng cách sử dụng địa chỉ cụ thể của nó; hoặc tất cả các Slave đồng thời sử dụng địa chỉ 0

- Slave chỉ có thể được gửi câu trả lời cho Master.

- Các Slave không thể bắt đầu giao tiếp với Master hoặc với các Slave khác.

Diagram

Description automatically generated

Hình: Mô hình các thiết bị trong mạng modbus

Hiện nay, có 03 chuẩn Modbus đang được sử dụng phổ biến trong công nghiệp - tự  
động hóa là: Modbus RTU, Modbus ASCII, Modbus TCP. Tất cả thông điệp được gửi dưới cùng một format. Sự khác nhau duy nhất giữa 3 loại Modbus là cách thức thông điệp được mã hóa. Cụ thể:

Modbus ASCII: Mọi thông điệp được mã hóa bằng hexadeci-mal, sử dụng đặc tính  
ASCII 4 bit. Đối với mỗi một byte thông tin, cần có 2 byte truyền thông, gấp đôi so với Modbus RTU hay Modbus TCP. Tuy nhiên, Modbus ASC II chậm nhất trong số 3 loại protocol, nhưng lại thích hợp khi modem điện thoại hay kết nối sử dụng song radio do ASCII sử dụng các tính năng phân định thông điệp. Do tính năng phân định này, mọi rắc rối trong phương tiện truyền dẫn sẽ không làm thiết bị nhận dịch sai thông tin. Điều này quan trọng khi đề cập đến các modem chậm, điện thoại di động, kết nối ồn hay các phương tiện truyền thông khó tính khác.

Modbus RTU: Modbus RTU là một giao thức nối tiếp đơn giản có thể truyền qua công nghệ UART truyền thống. Dữ liệu được được mã hóa theo hệ nhị phân, truyền theo byte 8 bit, mỗi lần 1 bit với tốc độ baud dao động từ 1200bit/s – 115200 bit/s. Các thiết bị Modbus RTU chỉ hỗ trợ tốc độ lên tới 38400 bit/s.

Modbus TCP: Modbus TCP đơn giản là Modbus qua Ethernet. Thay vì sử dụng thiết  
bị này cho việc kết nối với các thiết bị tớ, do đó các địa chỉ IP được sử dụng. Với Modbus TCP, dữ liệu Modbus được tóm lược đơn giản trong một gói TCP/IP. Do đó, bất cứ mạng Ethernet hỗ trợ TCP/IP sẽ ngay lập tức hỗ trợ Modbus TCP.

Trong mô hình mạng OSI, Modbus được triển khai ở tầng ứng dụng dựa trên nền vật lý RS232 hoặc RS485 đối với Modbus RTU và Ethernet đối với Modbus TCP

Diagram

Description automatically generated

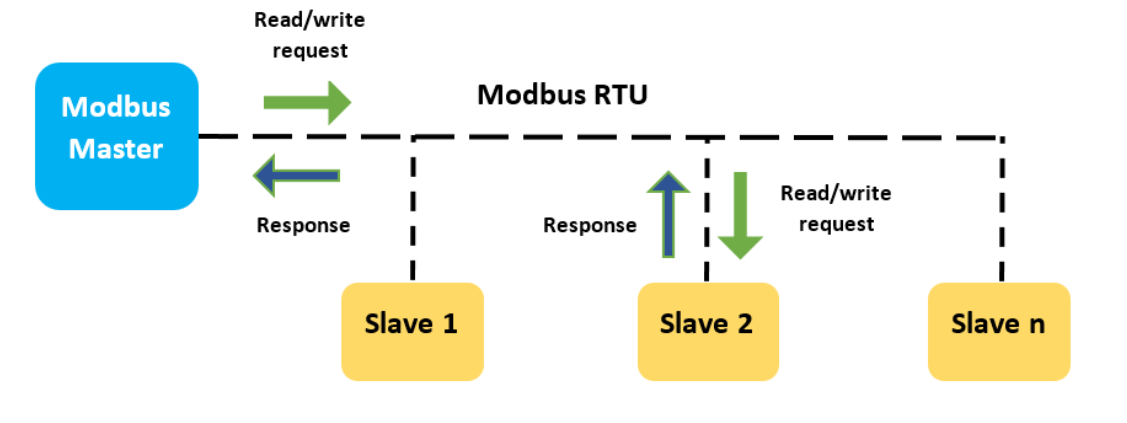
Hình: Tương quan giữa Modbus và mô hình OSI

### 2.2.3 Hoạt động

#### .2.2.3.1 Hoạt động của Modbus RTU

Modbus RTU hoạt động dựa trên nguyên tắc Master – Slave tức là một bên nhận (Master) và một bên truyền tín hiệu (Slave) thông qua địa chỉ thanh ghi. Phương thức truyền của [Modbus RTU](https://huphaco.vn/modbus-rtu-la-gi/) bằng đường truyền vật lý RS232 hoặc RS485.

Khi một chủ Modbus muốn có thông tin từ thiết bị, chủ sẽ gửi một thông điệp về dữ liệu cần, tóm tắt dò lỗi tới địa chỉ thiết bị. Mọi thiết bị khác trên mạng sẽ nhận thông điệp này nhưng chỉ có thiết bị nào được chỉ định mới có phản ứng.



Hình: Mô hình Master Slave trong Modbus RTU

Khung truyền của giao thức Modbus RTU xây dựng trên giao thức truyền thông nối tiếp UART và nội dung của các tin nhắn Modbus được xây dựng bằng nhiều gói tin ghé lại.

Một bản tin Modbus RTU bao gồm: 1 byte địa chỉ - 1 byte mã hàm - n byte dữ liệu - 2 byte CRC như hình ở dưới:

Diagram

Description automatically generated

Hình: Cấu trúc gói tin Modbus RTU

Chức năng và vai trò cụ thể như sau:

**-** Byte địa chỉ: xác định thiết bị mang địa chỉ được nhận dữ liệu (đối với Slave) hoặc dữ liệu nhận được từ địa chỉ nào (đối với Master). Địa chỉ này được quy định từ 0 - 254

**-** Byte mã hàm: được quy định từ Master, xác định yêu cầu dữ liệu từ thiết bị Slave. Ví dụ mã 01: đọc dữ liệu lưu trữ dạng Bit, 03: đọc dữ liệu tức thời dạng Byte, 05: ghi dữ liệu 1 bit vào Slave, 15: ghi dữ liệu nhiều bit vào Slave ...

**-** Byte dữ liệu: xác định dữ liệu trao đổi giữa Master và Slave.

+ Đọc dữ liệu:

Master:  2 byte địa chỉ dữ liệu - 2 byte độ dài dữ liệu

Slave: 2 byte địa chỉ dữ liệu - 2 byte độ dài dữ liệu - n byte dữ liệu đọc được

+ Ghi dữ liệu:

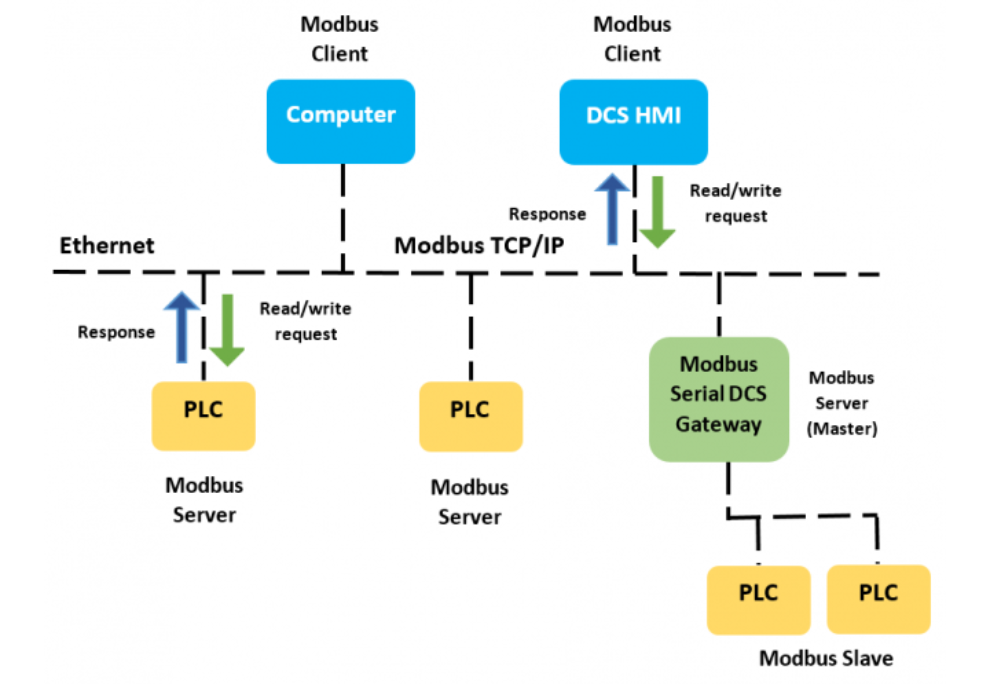
Master: 2 byte địa chỉ dữ liệu  - 2 byte độ dài dữ liệu - n byte dữ liệu cần ghi

Slave: 2 byte địa chỉ dữ liệu - 2 byte độ dài dữ liệu

- Byte CRC: 2 byte kiểm tra lỗi của hàm truyền. cách tính giá trị của Byte CRC 16 Bit

#### 2.2.3.2 Hoạt động của Modbus TCP

Modbus TCP hoạt động dưa trên mô hình client – server, trong mô hình này quan hệ giữa chủ nô bị đảo ngược. Gateway sẽ đóng vai trò là client, sẽ request đến các thiết bị Modbus (đóng vai trò là server) và chờ phản hồi từ server



Hình: Mô hình Client – Server trong Modbus TCP

Modbus TCP/IP sử dụng truyền thông TCP/IP để đóng gói khối PDU của Modbus RTU vào gói tin TCP/IP, cho phép định tuyến việc truyền gói tin, hỗ trợ nhiều thiết bị hơn, đồng thời tăng khoảng cách truyền tin.

Modbus TCP/IP sử dụng khối PDU của khung tin nhắn Modbus RTU



Hình: Khối PDU trong Modbus RTU

Sau đó chèn thêm phần MBAP (Modbus Application Protocol), được khung tin nhắn Modbus TCP/IP MBAP Header, 7 bytes được thêm vào đầu của khung tin nhắn.

Diagram

Description automatically generated

Hình: Cấu trúc gói tin của Modbus TCP

Chức năng và vai trò cụ thể như sau:

- Gói MBAP Header: 2 bytes Transaction ID, 2 bytes Protocol ID, 2 bytes Length, 1 byte Unit ID

+ Transaction ID: mã định danh của một phiên làm việc.

+ Protocol ID: mã định dạng giao thức.

+ Length: độ dài tin nhắn còn lại bắt đầu từ unit ID.

+ Unit ID: mã định danh của thiết bị modbus cần đọc dữ liệu

- Gói PDU: bao gồm Function Code và Data tương tự như Modbus RTU

## 2.3 Giao thức MQTT

### 2.3.1 Giới thiệu

MQTT viết tắt của MQ Telemetry Transport (trước đó MQTT được định nghĩa là Message Queuing Telemetry Transport), là giao thức ở tầng ứng dụng chạy trên nền TCP thuộc tập giao thức TCP/IP.

MQTT được phát minh vào năm 1999 bởi Andy Stanford-Clark (IBM) và Arlen  
Nipper (Eurotech). Ban đầu, họ thiết kế MQTT để dùng trong hệ thống điều khiển giám sát và thu thập dữ liệu (SCADA) cho hệ thống dẫn dầu thông qua vệ tinh.

### 2.3.2 Đặc điểm

Giao thức MQTT có kiến trúc theo mô hình Publish/Subscribe giúp truyền tải dữ liệu giữa các thiết bị, ứng dụng với nhau. Dữ liệu trong giao thức này là một chuỗi nhị phân (binary) chứ không phải chuỗi văn bản (text string), được định dạng theo gói tin command hoặc gói tin command acknowledgement.

Các thành phần lõi trong kiến trúc của MQTT:

– MQTT Broker: Được cung cấp dưới dạng mã nguồn mở hoặc các phiên bản thương mại, có thể đi kèm với các dịch vụ điện toán đám mây. Công việc của Broker là lọc các tin nhắn dựa trên topic, sau đó phân phối các tin nhắn đến các thiết bị/ứng dụng đã đăng ký topic đó. Các bạn có thể tham khảo một số MQTT Broker như: HiveMQ , Mosquitto, MQTTRoute, Jmqtt, …

– MQTT Client: Là các thiết bị/ứng dụng Client kết nối đến Broker để thực hiện truyền nhận dữ liệu. Hiện nay có rất nhiều mã nguồn mở MQTT Client được viết dưới nhiều ngôn ngữ khác nhau như HiveMQ MQTT Client được phát triển dựa trên ngôn ngữ Java, Eclipse Paho dựa trên C/C++, Python, …

– Topic: Mỗi MQTT Client thực hiện truyền/nhận dữ liệu với nhau thông qua các Topic được quản lý bởi Broker. Một Client đăng ký nhận dữ liệu được gọi là một Subcriber còn một Client gửi dữ liệu đi được gọi là một Publisher. Để nhận dữ liệu từ Publisher, đầu tiên Subcriber phải subscribe (đăng ký theo dõi) đến một Topic, sau đó bất cứ Client nào publish dữ liệu đến đúng Topic, thì Broker sẽ lọc và chuyển tiếp gói tin đến đúng Subscriber đó. Một Client có thể subscribe hoặc publish đến nhiều Topic khác nhau.

Một số ưu điểm của MQTT:

– MQTT là một “light weight protocol” giúp Broker và Client trao đổi gói tin có kích thước nhỏ, không chứa nhiều các dữ liệu phụ thêm vào nên có thể truyền nhận một cách mượt mà trong điều kiện bị giới hạn về băng thông đường truyền.

– Là một giao thức kết nối M2M/Internet of Things, MQTT được thiết kế để truyền tải tin nhắn trong các ứng dụng yêu cầu công suất thấp, kích thước bộ nhớ vi điều khiển nhỏ; băng thông thấp; đảm bảo việc truyền nhận các gói tin diễn ra thành công, …

– Với giao thức MQTT, các gói tin sẽ được truyền đi theo hai chiều: từ thiết bị đến Broker và từ Broker đến thiết bị. Một Client có khả năng gửi gói tin đến nhiều Client khác trong một lần gửi và nó cũng có thể nhận nhiều gói tin khác nhau từ các Client khác nhau.

– MQTT có thể mở rộng quy mô để kết nối với hàng triệu thiết bị IoT.

– MQTT có 3 mức Quality of Service (QoS) được xác định đảm bảo độ tin cậy trong việc gửi gói tin. Các bạn có thể đọc thêm về QoS ở tài liệu tham khảo cuối bài.

– MQTT giúp dễ dàng mã hóa tin nhắn bằng TLS và xác thực phía client bằng các phương thức xác thực như OAuth.

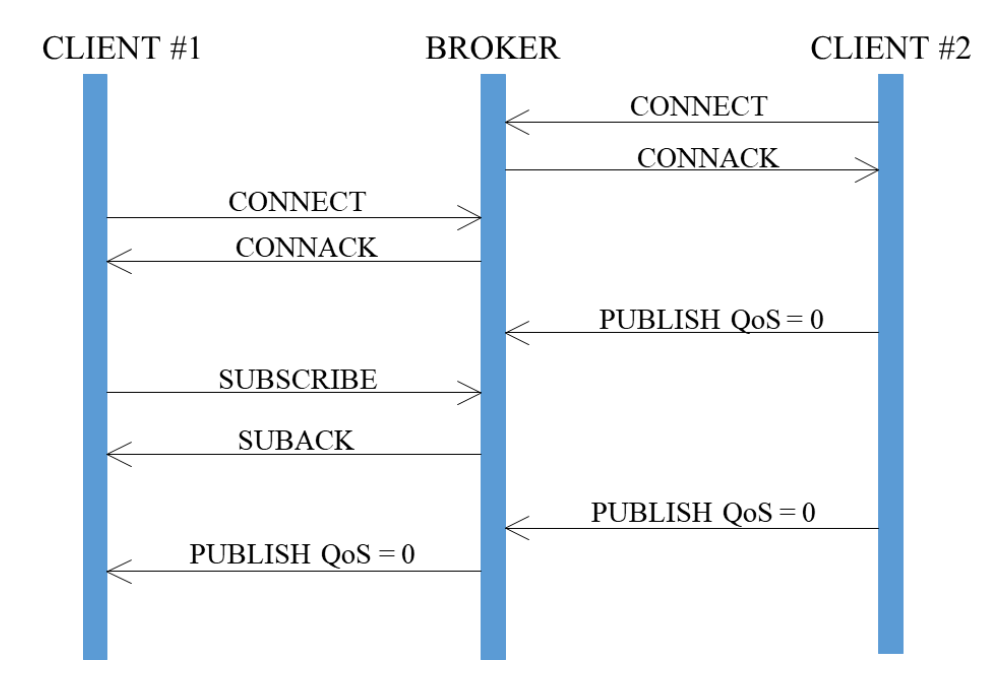
– Hiện nay, các nền tảng điện toán đám mây lớn như Amazon WebService, Google IoT Core, Microsoft Azure … đều hỗ trợ giao thức MQTT giúp các nhà phát triển dễ dàng thử nghiệm và sử dụng.

### 2.3.3 Hoạt động

#### 2.3.3.1 Mô hình Publisher - Subscriber

Để giao thức này hoạt động được, cần có một thành phần trung tâm gọi là **Broker** và các thiết bị/ứng dụng khách còn được gọi là các **Client**, các Client sẽ kết nối đến Broker. Broker và Client sẽ “nói chuyện” với nhau thông qua các gói tin MQTT được xây dựng dựa theo chuẩn OASIS. Tiêu chuẩn này còn định nghĩa các mức quality of service để đảm bảo độ tin cậy khi truyền nhận gói tin, các kịch bản always connected (luôn giữ kết nối) hoặc sometimes connected (thỉnh thoảng kết nối); khả năng mở rộng để hỗ trợ kết nối số lượng lớn thiết bị…

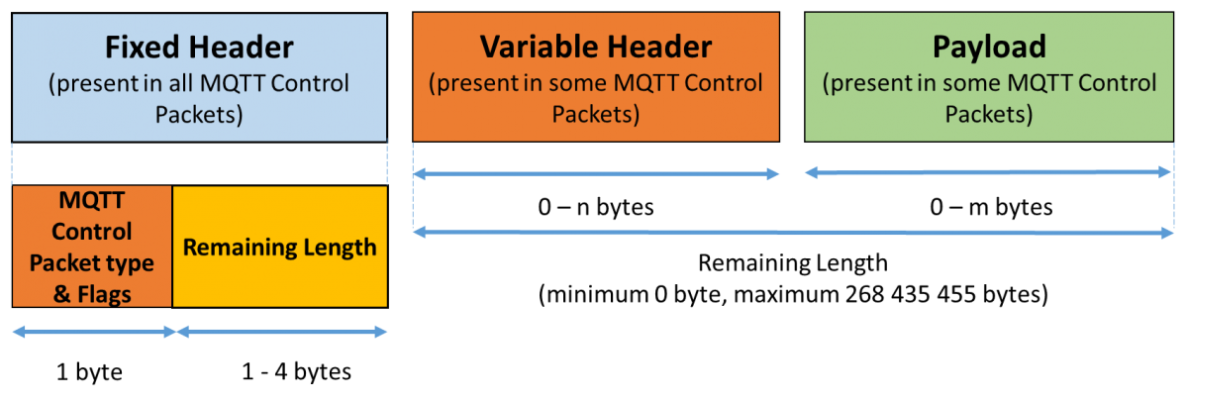
Một Client có thể đóng vai trò là một Publisher khi thực hiện publish một gói tin hoặc là một Subscriber khi thực hiện subscribe một Topic. Các Client muốn thực hiện publish hoặc subscribe thì phải thực hiện các bước sau:  
- Thiết lập kết nối TCP đến máy chủ có chứa Broker  
- Thiết lập kết nối đến Broker bằng cách gửi gói tin CONNECT   
- Đợi gói tin phản hồi CONNACK từ Broker để kiểm tra kết nối được thiết lập thành công hay thất bại  
- Nếu kết nối thành công, Client có thể gửi gói tin PUBLISH hoặc SUBSCRIBE đến một Topic tại Broker.  
- Khi quá trình truyền nhận kết thúc, Client có thể ngắt kết nối đến Broker bằng cách gửi gói tin DISCONNECT. Nếu Client muốn giữ kết nối đến Broker khi không có bất kỳ hoạt động nào diễn ra trong một khoảng thời gian nhất định thì Client thực hiện gửi gói tin PINGREQ để thông báo Broker rằng kết nối này vẫn được duy trì.



Hình: Quá trình hoạt động giữa hai Client (Publisher, Subscriber) và Broker

#### 2.3.3.2 Cấu trúc gói tin trong MQTT

Gói tin MQTT Control bao gồm các gói tin CONNECT, CONNACK, PUBLISH, SUBSCRIBE, SUBACK



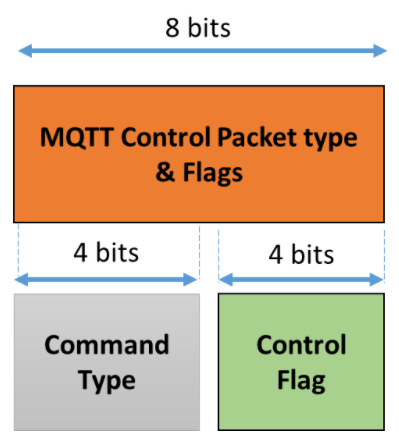
Hình: Cấu trúc gói tin MQTT Control

Trong một gói tin MQTT Control sẽ có tối đa 3 thành phần gồm Fixed Header, Variable Header và Payload. Các gói tin MQTT Control ngoài chứa mã định danh còn chứa một số thông tin khác như tên Topic, Client ID, tên User,…  
Trường Fixed Header bắt buộc phải có trong các gói tin MQTT Control, đây là trường phân biệt các gói tin CONNECT, PUBLISH, SUBSCRIBE, PINGREQ, DISCONNECT…  
Hai trường Variable Header và Payload là hai trường không bắt buộc, tùy vào gói tin MQTT Control mà có thể có hoặc không 2 trường này. Ví dụ:  
– Gói tin CONNACT: Chỉ có trường Fixed Header.  
– Gói tin PUBACK: Gồm các trường Fixed Header và Variable Header.  
– Gói tin CONNECT: Gồm các trường Fixed Header, Variable Header và Payload.

##### **Fixed Header**

Trường này có tối thiểu 2 byte. Fixed Header chứa MQTT Control Packet type và Flags và Remaining Length. Trường Remaining Length sẽ cho biết tổng số byte của Variable Header và Payload (tối thiểu là 0 Byte và tối đa là 256MByte). Ở đây chúng ta sẽ phân tích trường MQTT Control Packet type và Flags, đây là trường được dùng để phân biệt các gói tin MQTT Control với nhau.

– Trong 8 bits của trường MQTT Control Packet type và Flags này, 4 bits có trọng số cao (MSB) thể hiện Command Type, 4 bits còn lại là các bit Control Flag chứa cờ cụ thể cho từng loại Command Type.



Hình. Fix Header trong gói tin MQTT Control

##### **Variable Header**

Đây là trường không bắt buộc có trong các gói tin MQTT Control. Một vài gói tin như CONNECT, PUBLISH, SUBSCRIBE … sử dụng trường Variable Header này để cung cấp thêm thông tin và chúng khác nhau tùy thuộc vào các gói tin.

##### **Payload**

Cuối một số gói tin MQTT Control có thể chứa một trường Payload vì trường Payload là tùy chọn và không bắt buộc phải xuất hiện trong tất cả các gói tin MQTT như trường Fixed Header. Trường này thường chứa nội dung dữ liệu được gửi. Ví dụ:  
– Ở gói tin CONNECT, Payload sẽ là ClientID, username và password (nếu có).  
– Ở gói tin PUBLISH, Payload sẽ là nội dung dữ liệu cần gửi đi.

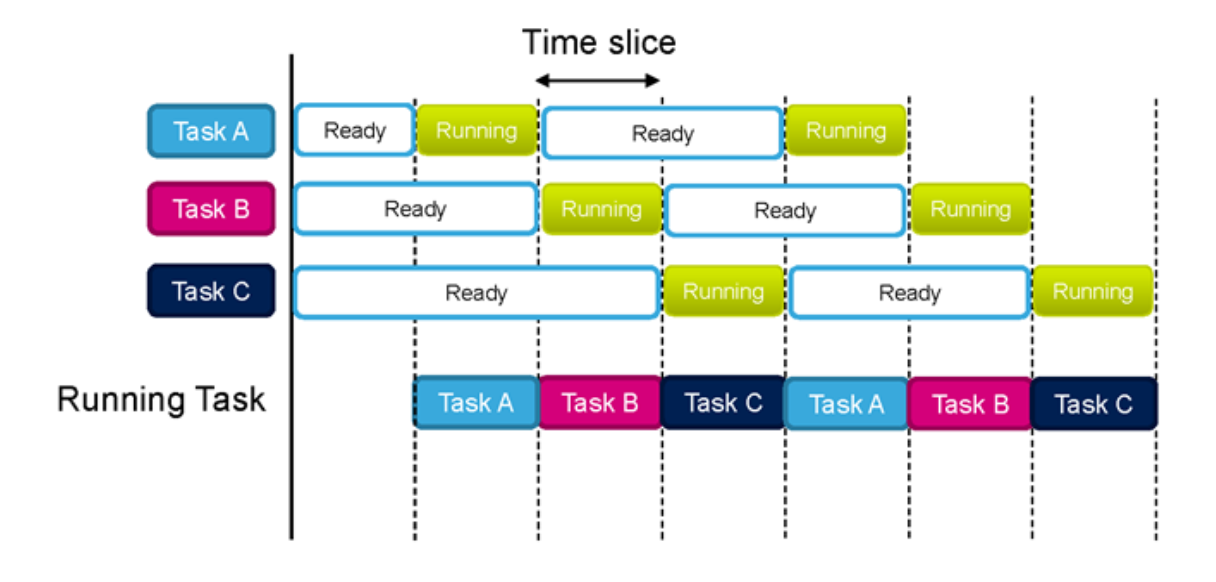
## 2.4 Hệ điều hành thời gian thực

### 2.4.1 Giới thiệu

FreeRTOS là một hệ điều hành nhúng thời gian thực (Real Time Operating System) mã nguồn mở được phát triển bởi Real Time Engineers Ltd, sáng lập và sở hữu bởi Richard Barry [6].

### 2.4.2 Đặc điểm

Hệ điều hành thời gian thực được thiết kế ra cho các nhiệm vụ đặc biệt. Các ứng dụng  
cần được thực thi với thời gian thật chính xác, các lỗi phát sinh cần được cô lập và xử lý nhanh chóng. Mọi sự chậm trễ, lỗi phát sinh không lường trước có thể khiến hệ thống bị đổ vỡ. Vì vậy, RTOS sử dụng trong những ứng dụng yêu cầu thời gian đáp ứng nhanh, chính xác về thời gian. Vi điều khiển có tài nguyên rất giới hạn. Do đó, hệ điều hành này chỉ tập trung vào một số ít các tính năng. Chúng cố gắng tối ưu tối đa số luồng, bộ lập lịch và các tác vụ (task) trên một hệ thống cỡ nhỏ.



Hình: Ví dụ về multi-tasking trong các hệ thống thời gian thực

FreeRTOS được thiết kế phù hợp cho nhiều hệ nhúng nhỏ gọn vì nó chỉ triển khai rất ít các chức năng như: cơ chế quản lý bộ nhớ và tác vụ cơ bản, các hàm API quan trọng cho cơ chế đồng bộ. Nó không cung cấp sẵn các giao tiếp mạng, drivers, hay hệ thống quản lý tệp (file system) như những hệ điều hành nhúng cao cấp khác. Tuy vậy, FreeRTOS có nhiều ưu điểm, hỗ trợ nhiều kiến trúc vi điều khiển khác nhau, kích thước nhỏ gọn (4.3 Kbytes sau khi biên dịch trên Arduino), được viết bằng ngôn ngữ C và có thể sử dụng, phát triển với nhiều trình biên dịch C khác nhau (GCC, OpenWatcom, Keil, IAR, Eclipse, …), cho phép không giới hạn các tác vụ chạy đồng thời, không hạn chế quyền ưu tiên thực thi, khả năng khai thác phần cứng. Ngoài ra, nó cũng cho phép triển khai các cơ chế điều độ giữa các tiến trình như: queues, counting semaphore, mutexes.

Diagram

Description automatically generated

Hình: Các thành phần cơ bản của một RTOS

Các lợi ích của việc dùng RTOS:  
- Chia sẻ tài nguyên một cách đơn giản: cung cấp cơ chế để phân chia các yêu cầu về bộ nhớ và ngoại vi của MCU.  
- Dễ debug và phát triển: Mọi người trong nhóm có thể làm việc một cách độc lập, các  
lập trình viên thì có thể tránh được các tương tác với ngắt, timer, với phần cứng (cái  
này mình không khuyến khích lắm vì hiểu được phần cứng vẫn sẽ tốt hơn nhiều)  
- Tăng tính linh động và dễ dàng bảo trì: thông qua API của RTOS, việc theo dõi hoạt  
động của hệ thống dễ dàng hơn.  
Dựa trên mục tiêu đề tài, sử dụng FreeRTOS giúp đơn giản việc xây dựng hệ thống chạy đa tác vụ để đảm bảo các tính ra. Bên cạnh đó, khả năng phát triển, kiểm tra gỡ lỗi nền vi điều khiển cũng được tiện lợi hơn.

### 2.4.3 Hoạt động

Một RTOS hoạt động có các chức năng cơ bản sau đây:

- Bộ lập lịch (Scheduler)

- Các dịch vụ thời gian thực (Realtime Services)

- Đồng bộ và xử lý thông điệp

#### 2.4.3.1 Bộ lập lịch

Mỗi task có thể có các trạng thái hoạt động chính.

Diagram

Description automatically generated

Hình: Các trạng thái hoạt động của task

**Ready:**Task đã sẵn sàng để có thể thực thi nhưng chưa được thực thi do có các task khác với độ ưu tiên ngang bằng hoặc hơn đang chạy.

**Running:** khi task thực sự đang chạy

**Blocked (Waiting):**Task đang đợi một event tạm hoặc event từ bên ngoài

**Suspended:**Task không khả dụng để lên lịch (scheduling)

Để lập lịch cho Task, có 3 kỹ thuật được áp dụng:

**- Round Robin Scheduling:** Mỗi task được chia cho một khe thời gian cố định, nếu trong khoảng thời gian được chia đó mà task chưa thực hiện xong thì sẽ bị tạm dừng, chờ đến lượt tiếp theo để thực hiện tiếp công việc sau khi hệ thống xử lý hết một lượt các task.

**- Co-operative scheduling:** Mỗi task được thực thi đến khi kết thúc quá trình thì task tiếp theo mới được thực thi.

**- Preemptive Scheduling:** Phương pháp này ưu tiên phân bổ thời gian cho các task có mức ưu tiên cao hơn. Mỗi task được gán 1 mức ưu tiên duy nhất. Có thể có 256 mưc ưu tiên trong hệ thống, và có thể có nhiều task có cùng mức ưu tiên.

+ **Preemptive:** Các task có mức ưu tiên cao nhất luôn được kiểm soát bởi CPU, khi phát sinh ISR thì hệ thống sẽ tạm dừng task đang thực thi, hoàn thành ISR sau đó hệ thống thực thi task có mức ưu tiên cao nhất tại thời điểm đó. Sau đó hệ thống mới tiến hành nối lại các task đang bị gián đoạn. Ở chế độ preemptive, hệ thống có thể đáp ứng các công việc khẩn cấp một cách nhanh chóng. Đa số các hệ thống thực tế đang chạy ở chế độ này.

Diagram

Description automatically generated

Hình: **Preemptive scheduling**

+ **Non-preemptive:** Ở chế độ non-preemptive thì các task được chạy cho đến khi nó hoàn tất. Khi phát sinh ISR thì hệ thống sẽ tạm dừng task đang thực thi và hoàn thành ISR, sau khi hoàn thành ISR thì hệ thống sẽ quay lại thực thi nốt phần việc còn lại của task bị gián đoạn. Task có mức ưu tiên cao hơn sẽ giành quyền kiểm soát CPU sau khi task bị gián đoạn thực thi xong.

Diagram

Description automatically generated

Hình: Non-**Preemptive scheduling**

Kernel tiến hành quản lý task ở nhiều giai đoạn. Chúng bao gồm: tạo task, huỷ task, thay đổi mức ưu tiên của task, thay đổi trạng thái của task.

#### 2.4.3.2 Các dịch vụ thời gian thực

Kernel thực hiện chức năng quản lý và lập lịch các quá trình sử dụng CPU và điều phối tài nguyên. Mỗi task chỉ được thưc thi bởi CPU trong một khoảng thời gian, trong các khoảng thời gian còn lại thì task được quản lý bởi các dịch vụ quản lý của hệ điều hành.

Các dịch vụ của RTOS bao gồm:

- Xử lý ngắt (Interrupt handling services).

- Dịch vụ quản lý thời gian (Time services).

- Dịch vụ quản lý thiết bị (Device management services).

- Dịch vụ quản lý bộ nhớ (Memory management services).

- Dịch vụ quản lý các kết nối Vào - Ra (IO services).

#### 2.4.3.3 Đồng bộ và xử lý thông điệp

Các thông điệp sử dụng để trao đổi thông tin giữa các hệ thống khác nhau hoặc giữa các task. Các dịch vụ quản lý thông điệp bao gồm: Semaphores, Event flags, Mailboxes, Pipes, Message queues

Semaphores: Dùng để đồng bộ hóa quyền truy cập vào các tài nguyên dùng chung.

Event Flags: Dùng để đồng bộ hóa các hoạt động cần có sự phối hợp của nhiều task.

Mailboxes, Pipes, Message queues: Dùng để quản lý các thông điệp gửi đi - đến giữa các task.

# THIẾT KẾ VÀ THỰC HIỆN PHẦN CỨNG

## 3.1 Yêu cầu

- Có các cổng RS485 để giao tiếp với các thiết bị chuẩn Modbus có sẵn trên thị trường, cổng RS232, UART để dễ dàng cài đặt, hiệu chỉnh.  
- Có cổng Ethernet để triển khai mô hình Modbus TCP hoặc các dịch vụ trên lớp mạng  
để giao tiếp với IoT Platform trên server.

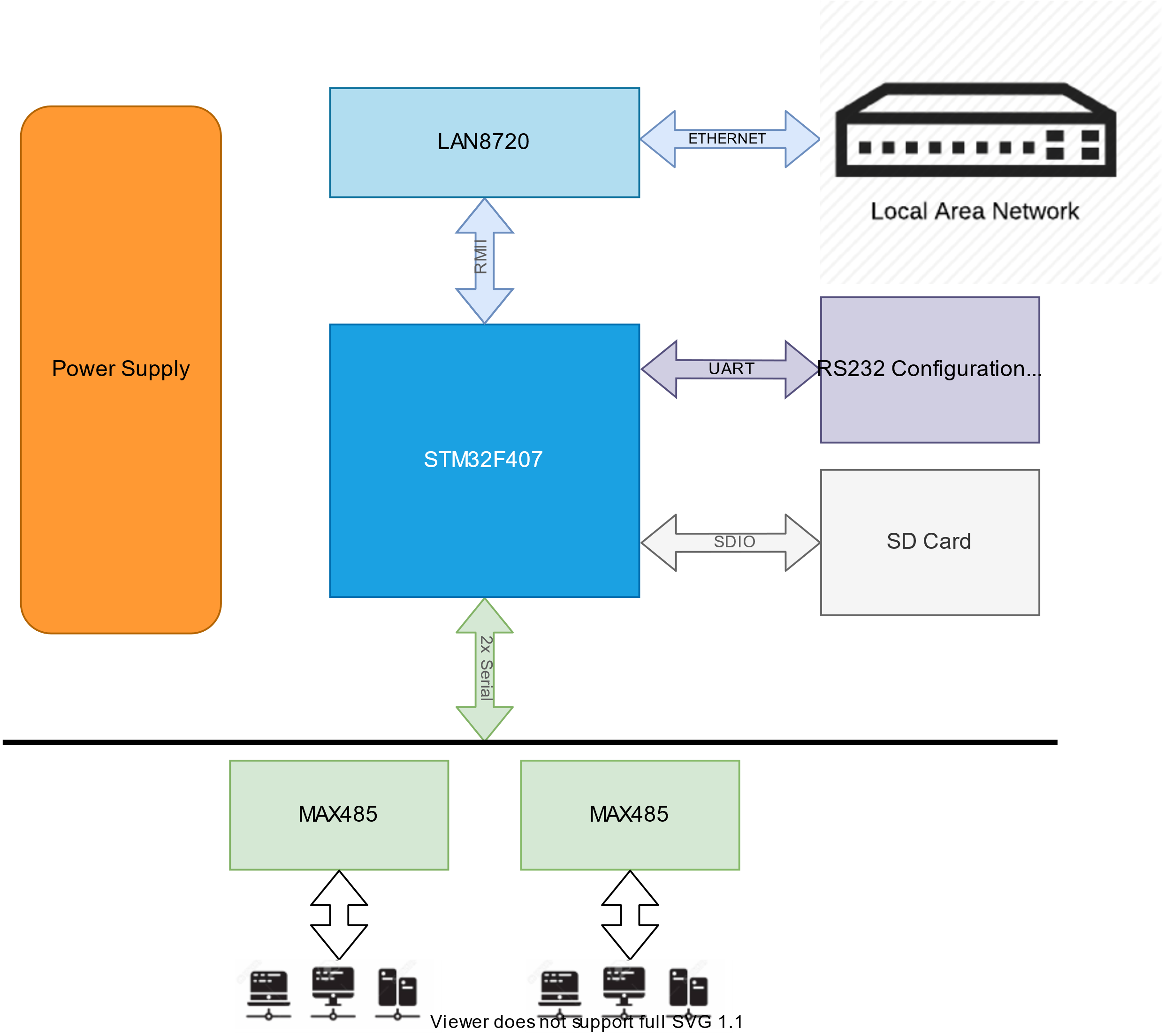
- Có khe cắm SD Card để lưu các thông tin cấu hình cho Gateway và các sự kiện.  
- Phần cứng vi điều khiển đủ mạnh để đảm bảo chạy các ứng dụng một cách ổn định.  
- Đèn báo trạng thái để thông báo cho người dùng.  
- Hỗ trợ tối đa 2 cổng RS485 để quản lý được nhiều thiết bị

## 3.2 Phân tích

Bảng: Phân tích các phương án cho Gateway Modbus Serial

|  |  |  |  |
| --- | --- | --- | --- |
| Phương án | Mô tả hoạt động | Ưu điểm | Nhược điểm |
| Sử dụng STM32F407 làm vi điều khiển chính cho Gateway | Triển khai Modbus Protocol ở chế độ master. Kết nối mạng nội bộ LAN để có thể gửi nhận dữ liệu tổng hợp được đến IoT Platform trên server Các thông báo chức năng qua đèn Led | Cấu hình và ngoại vi cơ bản đáp ứng được yêu cầu. Được dùng phố biến nên có sẵn các thư viện cơ bản, cộng đồng hỗ trợ động đảo. Dễ tìm thấy và mua ở thị trường trong nước, giá thành vừa phải khi so cùng cấu hình so với một số nhà sản xuất khác |  |
| Sử dụng IC LAN8720A | IC giao tiếp Ethernet, hỗ trợ tốc độ cao 10/100 Mbps | Rẻ, nhỏ gọn, giao tiếp RMII dễ sử dụng. Đơn giản hoá trong việc xử lý các gói ở lớp PHY |  |
| Sử dụng module đổi mức tín hiệu MAX485 | Module chuyển tiếp UART-TTL sang RS485 | Chuẩn giao tiếp thích hợp cho tốc độ cao và truyền xa. Chuẩn giao tiếp hỗ trợ kết nối đa điểm | Một số IC không chính hãng hoạt động không ổn định |
| Sử dụng module thẻ SD | Lưu các thông tin cấu hình Gateway và các sự kiện | Hỗ trợ chuẩn SDIO/SPI giúp tăng tốc dộ truy cập dữ liệu | Một số module hỗ trợ SDIO ngoài thị trường họạt động không ổn định |

## 3.3 Sơ đồ khối tổng quát

****

Hình: Sơ đồ khối tổng quát của Gateway Modbus Serial

Khối vi điều khiển STM32F407ZG đóng vai trò làm trung tâm xử lý dữ liệu, bao gồm  
các nhiệm vụ chính:

- Giao tiếp với MAX485 để tương tác với các thiết bị trọng mạng thông qua Modbus  
protocol.  
- Giao tiếp với RS232 để cung cấp môi trường cài đặt với máy tính.  
- Giao tiếp với LAN8720 để kết nối đến mạng nội bộ thông qua cổng giao tiếp  
Ethernet.

- Giao tiếp với SD card để lấy thông tin cài đặt cho Gateway và lưu các sự kiện

Các khối khác bao gồm: Khối nguồn để đảm bảo các mức điện áp ổn định cho các linh  
kiện hoạt động chính xác. MAX485 để chuyển tiếp qua lại tín hiệu UART và RS485. Khối  
LAN8720 để giao tiếp và xử lý các gói tin ở lớp PHY qua cổng Ethernet. Khối SD Card để giao tiếp và xử lý các thông tin cấu hình Gateway và các sự kiện

## 3.4 Sơ đồ mạch chi tiết

Do thời gian gấp rút và khả năng thiết kế mạch còn hạn chế nên em sử dụng development board OLIMEX-STM32-E407 để đảm bảo thời gian thực hiện hệ thống đúng hạn.

OLIMEX-STM32-E407 là một phần cứng mã nguồn mở, dựa trên ST Microelectronics STM32F407ZG ARM Cortex-M4 CPU

**A close-up of a circuit board

Description automatically generated with medium confidence**

Hình: Olimex STM32-E407 Cortex-M4 Dev Board

# THIẾT KẾ VÀ THỰC HIỆN PHẦN MỀM

## 4.1 Modbus IoT Gateway

### 4.1.1 Yêu cầu

- Gateway phải chạy được Modbus Stack ở chế độ Master để thu thập, thay đổi, quản lý dữ liệu cho các thiết bị đầu cuối. Đồng thời để nâng cao tối đa số lượng thiết bị trong mạng, 2 cổng RS485 trên board đều phải được khai thác, có thể xem như Gateway quản lý 2 mạng Modbus riêng biệt.  
- Gateway được kết nối đến IoT Platform thông qua MQTT, trong đó mỗi Gateway sẽ  
có địa chỉ ID định danh riêng trong mạng, đồng thời đảm bảo 2 luồng dữ liệu chính là  
downlink (gói tin yêu cầu từ IoT Server) và uplink (gói tin phản hồi yêu cầu và gói tin  
đồng bộ).  
- Gateway phải có chế độ tuỳ chỉnh, bao gồm: các thông số liên quan đến địa chỉ IP của Gateway trong mạng (Static IP, Netmask, Default Gateway), Gateway ID, MQTT  
Server IP và các thông số để điều chỉnh cho mạng Modbus: Baudrate, Stopbit, Parity.  
- Các cơ chế thông báo lỗi, đèn Led thông báo trang thái đường truyền và lỗi trong quá trình chạy thực nghiệm.

### 4.1.2 Lưu đồ giải thuật

Diagram

Description automatically generated with medium confidence

Hình: Quá trình khởi động của gateway

Quá trình khởi động của Gateway: vi điều khiển STM32 sẽ cài đặt các thông số hoạt  
động cơ bản và khởi tạo các ngoại vi, sau đó sẽ tìm đến vùng nhớ lưu trữ thông số cài đặt (trên vùng nhớ Flash hoặc SD Card). Nếu các thông số đã được thiết đặt, Gateway sẽ tiến hành load các cài đặt và bắt đầu boot vào kernel của RTOS, ngược lại sẽ báo trạng thái chưa cài đặt trên LED và tiến hành load các thông số mặc định

**Diagram

Description automatically generated**

Hình: Các khối nhiệm vụ của gateway

Sau khi boot thành công, Gateway sẽ tiến vào Main Controller Task – đây là task chính của Gateway, bao gồm các cơ chế tạo, quản lý các task sau này. Cũng trên Main Controller Task, các cơ chế monitoring tài nguyên hệ thống được triển khai và gửi lên người dùng thông qua Serial RS232.  
Để đảm bảo việc khởi động các protocol và các application một cách chính xác (MQTT chỉ chạy khi LWIP đã kết nối mạng, Modbus Uplink và Modbus Downlink sẽ chạy khi Modbus Stack sẵn sàng, ...), các Task phải được khởi tạo một cách có tuần tự. Các task mới chỉ được khởi động khi được cho phép bởi Main Controller Task cũng như phải reponse trạng thái khởi động để Main Controller Task quyết định các Task tiếp theo được khởi tạo.  
Modbus Stack: Trong đề tài, Modbus Stack được triển khai dựa trên một thư Modbus có sẵn là FreeModbus, đây là một gói thư viện miễn phí, hỗ trợ các chế độ RTU/ASCII và TCP đối với các thiết bị nhúng kết nối được mạng, hỗ trợ nhiều nền tảng vi điều khiển nên việc porting không quá phức tạp, bên cạnh đó, FreeModbus còn được thiết kế để chạy trên các hệ điều hành thời gian thực thông dụng, việc triển khai FreeModbus không tốn nhiều tài nguyên (1 Timer và 1 Serial port), 300 bytes ram tuỳ theo các function được triển khai. Tuy nhiên, trong yêu cầu đặt ra, cần tuỳ biến thư viện để có thể chạy tổng cộng 2 master để có thể đạt được khoảng 500 thiết bị.

Command Line: Giao diện command line thông qua Serial để cài đặt các thông số cho  
Gateway

**Diagram

Description automatically generated**

Hình: Luồng dữ liệu của Gateway Modbus Serial

MQTT Handler: Trong quá trình khởi động, MQTT Handler sẽ khởi tạo dịch vụ  
MQTT dựa trên lớp TCP đã được cung cấp bởi thư viện LWIP khởi tạo trước đó. Sau đó, MQTT Handler sẽ tiến hành Subscribe vào Topic định danh trước (thống nhất với IoT Server và được lưu lại thông qua bộ nhớ SD), dịch vụ MQTT sẽ tạo ra callback mỗi khi có gói tin được gửi đến Topic đã đăng ký, trong đề tài gói tin bao gồm 2 gói: Uplink packet từ Modbus Uplink để chuyển dữ liệu đọc được từ gateway đến server và Downlink Request từ phía server để yêu cầu gateway trả về các thông tin từ các thiết bị mà nó quản lý. Các gói tin được chuẩn hóa theo kiểu Json, gói từ uplink và downlink sẽ được gửi qua cơ chế hàng đợi queue trong FreeRTOS

Modbus Uplink: Sau khi Modbus Stack đã sẵn sàng, Modbus Uplink sẽ được khởi tạo. Modbus Uplink phải chờ cờ cho phép từ người dùng để được phép truy cập vào tài nguyên modbus để có thể lần lượt đọc dữ liệu từ các thiết bị. Sau khi dữ liệu đọc thành công, Modbus Uplink sẽ gửi thông tin trả về từ thiết bị để chuyển dữ liệu qua luồng Uplink MQTT cho MQTT handler xử lí.

Modbus Downlink: Sau khi Modbus Stack đã sẵn sàng, Modbus Downlink sẽ  
được khởi tạo. Modbus Downlink sẽ kiểm tra hàng đợi Queue từ khối MQTT handler và gọi các hàm cung cấp từ Modbus Stack để truy cập các thông tin liên quan đến giao thức modbus, bao gồm:  
- Read Input Register (0x04)  
- Read Holding Registers (0x03)  
- Write Single Register (0x06)  
- Write Multiple Registers (0x10)  
- Read/Write Multiple Registers (0x17)  
- Read Coils (0x01)  
- Write Single Coil (0x05)  
- Write Multiple Coils (0x0F)  
- Read Discrete Inputs (0x02)  
- Report Slave ID (0x11)

Các thông tin trả về sẽ được Modbus Stack trả về và gửi ngược lên đường Uplink Respose, đồng thời sẽ báo lỗi sang Main Controller Task nếu có bất kỳ gói tin nào lỗi.

Lưu đồ giải thuật chi tiết của MQTT Handler:

**Diagram

Description automatically generated**

Hình: Lưu đồ giải thuật của khối MQTT Handler.

Giải thích:  
- Khối MQTT Handler thiết lập kết nối đến Broker và đăng ký các Topic cần thiết. Từ  
đó tạo ra các luồng callback khi thoả điều kiện: nhận được gói tin MQTT và lỗi trong  
kết nối MQTT.  
- Khi nhận được, gói tin sẽ được giải mã theo chuẩn JSON và kiểm tra tính hợp lệ. Nếu thoả điều kiện, các yêu cầu của gói tin sẽ được lấy ra và gửi xuống tầng Modbus Downlink thông qua cơ chế hàng đợi của FreeRTOS.  
- MQTT Handler sẽ chờ các phản hồi từ Modbus Downlink và gửi lên MQTT Broker.

Lưu đồ giải thuật chi tiết của Modbus Uplink:

**Diagram

Description automatically generated**

Hình: Lưu đồ giải thuật của khối Modbus Uplink

Giải thích:  
- Modbus Uplink sẽ đợi, cấp quyền sử dụng tài nguyên freemodbus từ người dùng, nếu nhận được yêu cầu đọc dữ liệu, nó sẽ gọi các hàm tương ứng để Modbus Stack xử lý.  
- Modbus Stack phản hồi gói tin, dữ liệu sẽ được gửi lên MQTT Handler.

Lưu đồ giải thuật chi tiết của Modbus Downlink:

**Diagram

Description automatically generated**

Hình: Lưu đồ giải thuật của khối Modbus Downlink

Giải thích:  
- Modbus Downlink sẽ đợi yêu cầu từ MQTT Handler qua cơ chế hàng đợi, nếu nhận được yêu cầu, nó sẽ gọi các hàm tương ứng để Modbus Stack xử lý.  
- Khi Modbus Stack phản hồi gói tin, dữ liệu sẽ được gửi lên MQTT Handler.

#### 4.1.2.1 Quá trình khởi động Modbus IoT Gateway

#### 4.1.2.2 Quá trình xử lí các gói tin của MQTT Handler

#### 4.1.2.3 Quá trình xử li các gói tin của Modbus Application

## 4.2 Phần mềm trên máy tính

### 4.2.1 Yêu cầu

- Cung cấp giao diện kết nối với Gateway thông qua cổng serial

- Cung cấp giao diện hiển thị thông tin và thay đổi các giá trị trên Gateway

### 4.2.2 Lưu đồ giải thuật

Phần mềm đặt ra 2 luồng dữ liệu chính:  
- Các lệnh Get, Set để lấy các thông tin từ Gateway hiển thị lên giao diên và gửi các  
thông tin từ giao diện xuống Gateway.  
- Các Reponse phản hồi từ Gateway tương ứng.

**Graphical user interface, text

Description automatically generated**

Hình: Luồng dữ liệu của phần mềm trên máy tính.

# KẾT QUẢ THỰC HIỆN

## 5.1 Gateway Modbus

### 5.1.1 Phần cứng

Dựa trên phần cứng có sẵn từ trước, các khối chức năng cần thiết hoạt động bình thường, bao gồm:

- Khối Ethernet có thể kết nối vào mạng giúp board có thể tuyển nhận dữ liệu qua ethernet

- Các cổng RS485 cho giao thức Modbus hoạt động ở chế độ haft-duplex.

### 5.1.2 Phần mềm

Các kết quả đạt được:  
- Tích hợp được hệ điều hành thơi gian thực FreeRTOS và thư viện LwIP lên Gateway  
giúp cho việc triển khai multitasking và sử dụng các dịch vụ liên quan đến mạng dễ  
dàng hơn.  
- Tích hợp và tuỳ chỉnh FreeModbus ở chế độ 2 Master thành công lên nền tảng  
FreeRTOS.  
- FreeModbus ở chế độ RTU, triển khai các Function chính: Read/Write  
Single/Multiple Register, Read/Write Coil Register, Read Input Register, Read Discrete Register. Các cơ chế báo lỗi: lỗi gói tin, lỗi quá thời gian chờ, lỗi sai  
mã chức năng, …  
- Tích hợp thư viện MQTT, kết nối thành công với Broker MQTT ở các QoS khác  
nhau.  
- Triển khai cầu chuyển đổi giữa giao thức MQTT và Modbus, bao gồm 2 luồng dữ liệu chính là Downlink Request từ phía Server và Uplink Reponse từ phía Gateway.Các gói tin được chuẩn hoá theo kiểu JSON để dễ dàng tương tác với IoT Platform Server.  
- Giao diện RS232 Serial Configuration để có thể thay đổi các giá trị khi cài đặt. Các  
giá trị sẽ được lưu vào vùng nhớ Flash của vi điều khiển hoặc SD card trên board.

## 5.2 App trên máy tính

Phần mềm hoạt động trên máy tính chạy đúng theo các yêu cầu đặt ra.

**(Hình của cái Desktop app)**

## 5.3 Các bài thử nghiệm đánh giá

### 5.3.1 Kết quả đạt được về mặt chức năng

#### 5.3.1.1 Thử nghiệm với mạng đơn thiết bị

##### Thực hiện trên thiết bị ảo

Trong phần này, gateway sẽ được kết nối phần mềm Modbus Slave giả lập thiết bị modbus thông qua cổng serial.  
**Kết quả**: Gateway có thể giao tiếp với phần mềm giả lập Modbus Slave thông qua các function: Read/Write Single/Multiple Holding Registers, Read/Write Coil Register, Read Input Register, Read Discrete Register

(**Chèn hình vô**)

##### Thực hiện trện thiết bị thực

Trong phần này, Gateway sẽ được lần lượt kết nối với biến tần Delta VDF EL [trích dẫn] có hỗ trợ Modbus, cảm biến ánh sáng VELT-W-LUX-RS[trích dẫn] và cảm biến nhiệt độ, độ ẩm SHT20 [trích dẫn]  
**Kết quả**: Gateway có thể giao tiếp với biến tần, các cảm biến thông qua các function: Read/Write Single/Multiple Holding Registers.

(**Chèn hình vô**)

#### 5.3.1.2 Thử nghiệm với mạng nhiều thiết bị

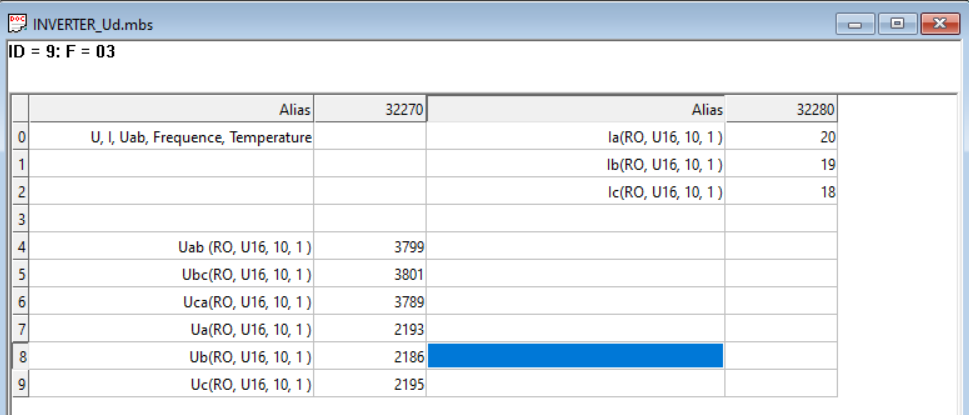
##### Thực hiện trên thiết bị ảo

Thực hiện đọc ghi lần lượt và liên tục tất cả thiết bị trọng mang Modbus. Các thông số  
bài kiểm tra:  
- Modbus Slave: RTU Mode, Serial 9600, 8, N1.  
- Function: Read Single/Multi Register. Write Single Resigter

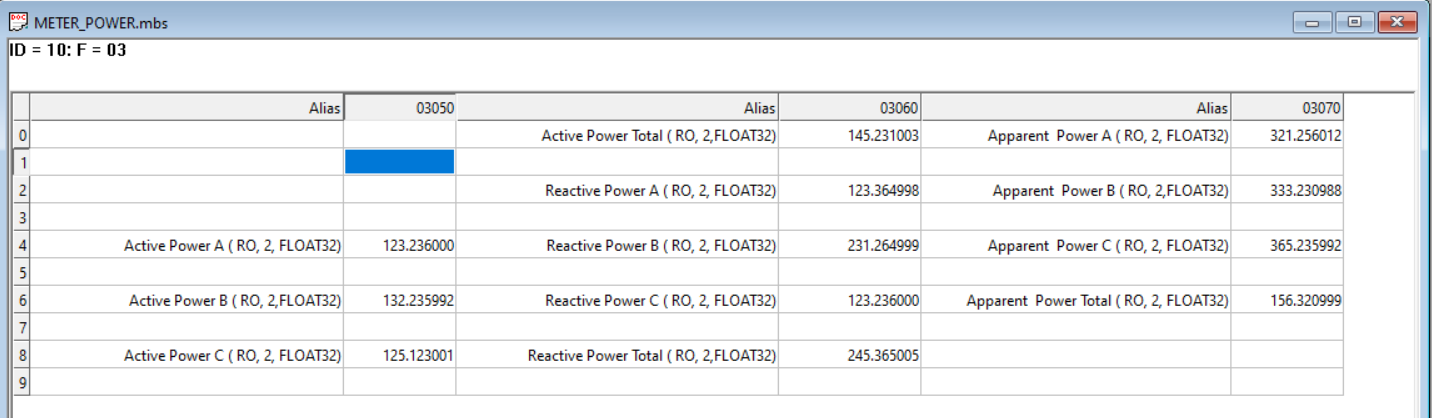
Trong phần này, gateway sẽ được kết nối phần mềm Modbus Slave giả lập nhiều thiết bị modbus thông qua cổng serial. Các thanh ghi, kiểu dữ liệu được lấy từ tài liệu kĩ thuật được cấp từ nhà sản xuất, bao gồm giả lập các thiết bị: Inverter (Model: SUN2000 V200R002 được cung cấp bởi Huawei), Energy Meter (Model: PM5300 được cung cấp bởi Schneider Electric), Weather Station (Model: VSN800-14 được cung cấp bởi Sunspec).



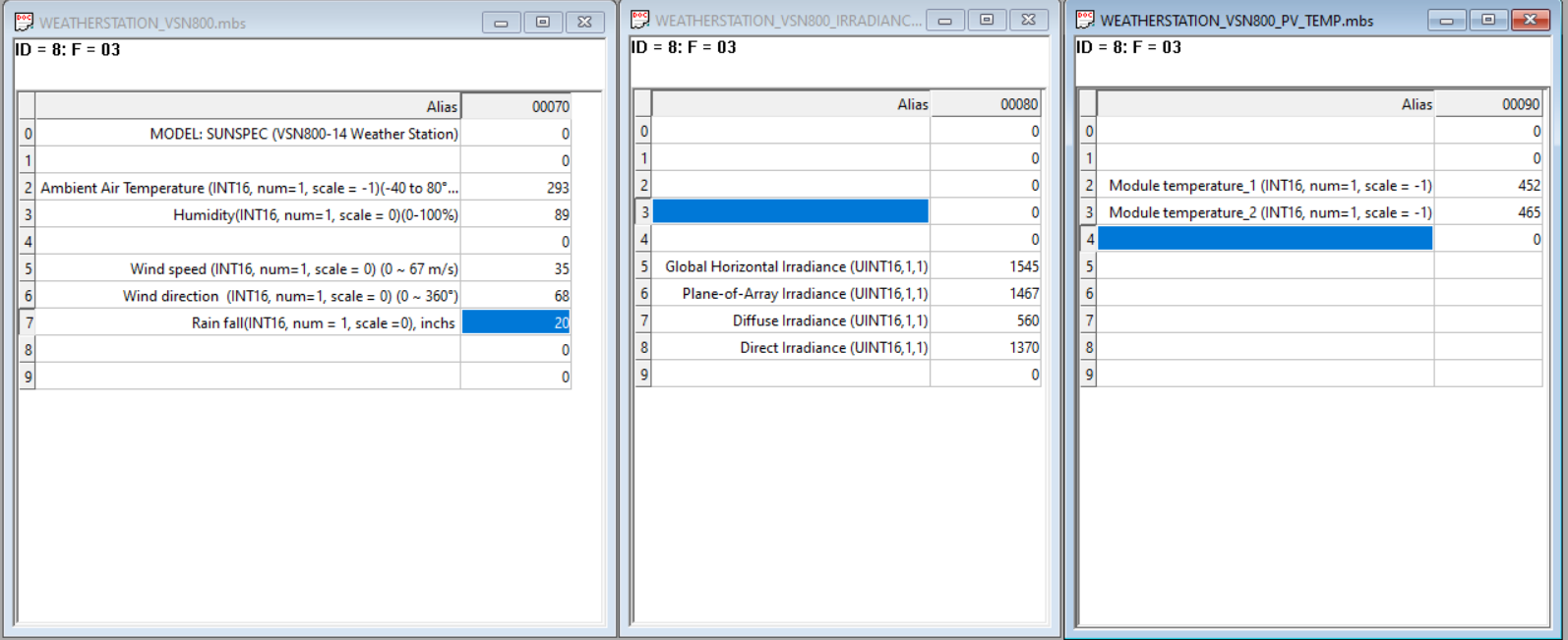
Hình: Các thiết bị thực được giả lập dữ liệu (Inverter SUN2000 V200R002, Energy Meter PM5350, Weather Station VSN800-14)



Hình: Inverter simulator



Hình: Energy Meter simulator



Hình: Weather station simulator

**Kết quả**: nhìn chung các thiết bị hoạt động bình thường và không xung đột lẫn nhau,  
tuy nhiên vẫn có các gói bị lỗi

Bảng: Bảng thống kê kết quả phản hồi của Modbus Slave trong mạng đa thiết bị

|  |  |  |  |
| --- | --- | --- | --- |
| Tổng số channel | Tổng số gói đọc | Số gói lỗi | Loại lỗi |
| 66 | 10032 | 23 | Timeout |

**Nhận xét**: qua các quá trình kiểm tra và tìm hiểu, các gói tin quá thời gian không phản  
hồi do bị lỗi thông qua đường truyền và bị loại khỏi hàng đợi bởi FreeModbus; các đường bus kết nối tiếp xúc chưa tốt gây ra lỗi ở vài gói tin.

##### Thực hiện trện thiết bị thực

Thực hiện đọc ghi lần lượt và liên tục tất cả thiết bị trọng mang Modbus. Các thông số  
bài kiểm tra:  
- Modbus Slave: RTU Mode, Serial 9600, 8, N1.  
- Function: Read Single/Multi Register. Write Single Resigter

Trong phần này, gateway sẽ được kết các thiết bị thực bao gồm: biến tần Delta-VFD-EL, Cảm biến ánh sánh LUX-RS-485, Cảm biến nhiệt độ độ ẩm SHT20. Các thiết bị này có thể giao tiếp qua chuẩn Modbus thông qua các function code: Read/Write Single/Multiple Holding Registers

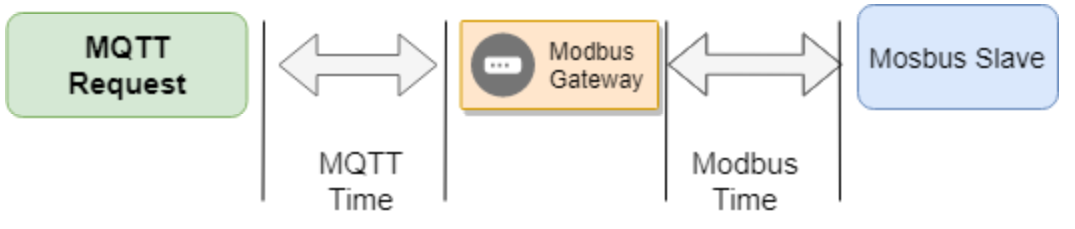
**Kết quả đạt được**: nhìn chung các thiết bị hoạt động bình thường và không xung đột lẫn nhau.

**Nhận xét**:

### 5.3.2 Kết quả đạt được về mặt tốc độ

Mô hình thời gian trễ của luồng dữ liệu:

- MQTT Time: Thời gian của giao thức MQTT, bao gồm thời gian gói tin Downlink  
Request được gửi đến Gateway thông qua trung gian Broker và thời gian gói tin  
Uplink Reponse phản hồi đến nơi yêu cầu. Trong bài kiểm tra, gói tin kiểm tra sẽ  
không được gửi xuống thiết bị Slave mà sẽ phản hồi ngay lập tức về Server khi đi  
đến Gateway.  
- Modbus Time: Thời gian của giao thức Modbus từ khi bắt đầu nhận Request được  
yêu cầu đến khi trả về gói tin phản hồi từ Slave. Trong bài kiểm tra, thời gian này  
được tính bằng thời gian tổng gói tin đi trong mạng trừ MQTT Time.



Hình: Mô hình thời gian gói tin đi trong mạng Modbus.

Các thông số trong bài kiểm tra:  
- Mạng LAN: 100Mbps, Asus Router RT-AC1300UHP.  
- MQTT Broker: Mosquitto on cloud (https://mosquito.org)  
- MQTT Client: Gateway Modbus with LAN8720A at 100Mbps  
- Modbus: RTU Mode, Serial: 9600, 8, N, 1. Timeout: 3000ms

Kết quả:  
- Thời gian MQTT Time thu được:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No | QoS | Tổng số gói | Số gói lỗi | Độ trễ (ms) |
| 1 | 0 | 100 | 0 |  |
| 2 | 1 | 100 | 0 |  |
| 3 | 2 | 100 | 0 |  |
| 4 | 0 | 1000 | 0 |  |
| 5 | 1 | 1000 | 0 |  |
| 6 | 2 | 1000 | 0 |  |

**Đánh giá**: thời gian latency cao, việc thiết đặt QoS lên 1 và 2 làm tăng đáng kể độ trễ,  
nguyên nhân tạm thời có thể xem là do việc triển khai MQTT dưới vi điều khiển chưa thật sự  
là một giải pháp tốt. Bên cạnh đó, MQTT Broker của Mosquitto trên cloud cũng gây ra  
độ trễ nhất định.

- Thời gian tổng của gói tin đi trong mạng:

Bảng 6: Bảng thống kê tổng thời gian gói tin đi trong mạng Modbus và MQTT

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No | QoS | Tổng số gói | Số gói lỗi | Độ trễ (ms) |
| 1 | 0 | 100 | 0 |  |
| 2 | 0 | 100 | 2 |  |
| 3 | 0 | 100 | 3 |  |
| 4 | 0 | 1000 |  |  |
| 5 | 0 | 1000 |  |  |
| 6 | 0 | 1000 |  |  |

# KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

## Kết luận

### 6.1.1 Ưu điểm

### 6.1.2 Nhược điểm

## Hướng phát triển

Sinh viên trình bày hướng phát triển và khả năng ứng dụng của đề tài

# TÀI LIỆU THAM KHẢO

Trong mục này, sinh viên liệt kê những tài liệu đã tham khảo khi thực hiện đề tài luận văn. Những nội dung trình bày ở mục trên có tham khảo tài liệu thì sinh viên cần ghi chú bằng chỉ số (ví dụ [1], [2]). Chỉ số này cần tương ứng danh mục tài liệu tham khảo. Sinh viên xem thêm hướng dẫn cách viết trích dẫn kiểu IEEE.

Ví dụ:

1. Tống Văn On, “Thiết kế mạch số với VHDL & Verilog”, Nhà xuất bản Lao động Xã Hội, 2007.
2. Altera Corp., “SDRAM Controller for Altera’s DE2/ DE1 boards”, [www.altera.com](http://www.altera.com)

# PHỤ LỤC

Trong phần này, sinh viên có thể trình bày:

* Những kết quả nghiên cứu bổ sung mà trong phần Kết quả luận văn chưa trình bày hết.
* Phần mã nguồn chương trình, sinh viên cũng có thể trình bày trong mục này. Để ngắn gọn, sinh viên chỉ đưa những mã nguồn chính vào phần Phụ lục.
* Sơ đồ toàn mạch chi tiết

## Mô hình demo hệ thống