

**TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI**

**ĐỒ ÁN TỐT NGHIỆP**

**Thiết kế hệ thống giám sát và thu thập dữ  
liệu cho bộ biến đổi công suất và pin năng  
lượng mặt trời**

**VÕ THANH TÙNG**

tung.vt189674@sis.hust.edu.vn

**Ngành Điều Khiển và Tự Động Hóa**  
**Chuyên ngành Tự Động Hóa Công Nghiệp**

**Giảng viên hướng dẫn:** TS. Hoàng Đức Chính

Chữ ký của GVHD

**Bộ môn:** Tự Động Hóa Công Nghiệp

**Viện:** Điện

**HÀ NỘI, 01/2020**



## **ĐỀ TÀI TỐT NGHIỆP**

Đề tài: “Thiết kế hệ thống giám sát và thu thập dữ liệu cho bộ biến đổi công suất và pin năng lượng mặt trời”

Mục tiêu và nhiệm vụ: Thiết kế và phát triển hệ thống theo dõi từ xa tấm pin mặt trời và các bộ biến đổi công suất kèm theo. Hệ thống bao gồm các thiết bị đo lường và xử lý tín hiệu được kết nối với bộ biến đổi công suất để thu thập các dữ liệu điện như dòng điện, điện áp vào/ra, công suất, v.v... để phân tích hiệu quả hoạt động thu và phát năng lượng của tấm pin mặt trời. Các thành phần khác của hệ thống cần được phát triển bao gồm các module truyền thông không dây (Zigbee), hệ thống xử lý và lưu trữ dữ liệu trung tâm cũng như giao diện người dùng.

Giáo viên hướng dẫn  
Ký và ghi rõ họ tên



## **Lời cảm ơn**

Để hoàn thành đồ án này ngoài sự cố gắng của cá nhân em và sự giúp đỡ của hai anh ThS. Nguyễn Đình Ngọc, ThS. Hoàng Thành Nam cùng sự hỗ trợ của tất cả thành viên PELAB thì em có được sự hướng dẫn tận tụy, chỉ bảo nhiệt tình của TS. Hoàng Đức Chính và TS. Vũ Hoàng Phương. Đề tài cấp Khoa học cấp Nhà nước mã số KC05.22/16-20 đã tạo điều kiện cho em thực hiện đề tài này. Mặc dù đã cố gắng hết sức để thực hiện đề tài, tuy nhiên do khả năng còn hạn chế nên báo cáo này vẫn còn thiếu sót. Em rất mong nhận được sự góp ý để báo cáo này trở nên hoàn thiện hơn.

Em xin chân thành cảm ơn.

## **Tóm tắt nội dung đồ án**

Đồ án này trình bày về thiết kế hệ thống giám sát và thu thập dữ liệu cho bộ biến đổi điện tử công suất và pin năng lượng mặt trời. Nội dung đồ án được chia thành các chương:

Chương 1: Tổng quan về hệ thống điện năng lượng mặt trời

Chương 2: Tổng quan về hệ thống PV Inverter

Chương 3: Thiết kế phần cứng

Chương 4: Thiết kế hệ thống điều khiển giám sát và thu thập dữ liệu cho hệ PV Inverter.

Chương 5: Ghép nối và kết quả thực nghiệm

Chương 6: Kết luận

Bao gồm trình bày lý thuyết về các bộ biến đổi điện tử công suất, pin năng lượng mặt trời và cách thức tiến hành thiết kế frontend, backend cho một hệ thống điều khiển giám sát và thu thập dữ liệu cho hệ thống điện tử công suất trong công nghiệp sử dụng các phần mềm hỗ trợ thiết kế điều khiển cũng như là giao diện. Tuy chưa hoàn thiện được hệ thống giám sát 100% vì hệ thống thực chưa hoàn thiện để thử nghiệm cũng như còn nhiều hạn chế về mặt kiến thức nhưng kết quả đồ án cũng đã bám sát yêu cầu được đã đặt ra. Hướng phát triển của đồ án em dự định sẽ hoàn thiện 100% hệ thống cũng như là cải tiến hệ thống thêm về thu thập và phân tích dữ liệu đưa ra các cảnh báo điều khiển vận hành hệ thống.



## MỤC LỤC

<b>CHƯƠNG 1. TỔNG QUAN VỀ HỆ THỐNG ĐIỆN NĂNG LƯỢNG MẶT TRỜI.....</b>	<b>1</b>
1.1    Tổng quan về năng lượng mặt trời.....	1
1.1.1    Tình hình và xu thế phát triển năng lượng mặt trời .....	1
1.1.2    Phương pháp thu năng lượng mặt trời bằng pin mặt trời.....	3
1.1.3    Pin mặt trời.....	3
1.1.4    Các hệ thống với pin mặt trời .....	5
1.2    Giới thiệu về hệ thống biến đổi điện tử công suất sử dụng pin năng lượng mặt trời .....	6
1.2.1    Giới thiệu .....	6
1.2.2    Hệ thống PV Inverter ba pha .....	6
<b>CHƯƠNG 2. TỔNG QUAN VỀ HỆ THỐNG PV INVERTER .....</b>	<b>8</b>
2.1    Tổng quan về hệ thống PV Inverter .....	8
2.2    Các thành phần của bộ điều khiển .....	9
2.2.1    Mạch điều khiển Slave .....	9
2.2.2    Mạch điều khiển trung tâm Master .....	11
2.3    Các mảng mạch đo lường.....	11
2.4    Truyền thông trong hệ thống.....	11
2.4.1    Truyền thông giữa các bộ điều khiển.....	11
2.4.2    Truyền thông giữa mạch điều khiển trung tâm với màn hình... ..	15
2.4.3    Truyền thông giữa mạch điều khiển trung tâm và Raspberry Pi ..	19
<b>CHƯƠNG 3. THIẾT KẾ PHẦN CỨNG.....</b>	<b>22</b>
3.1    Thiết kế và chế tạo các mảng mạch đo lường và mạch Master .....	22
3.1.1    Thiết kế và chế tạo mạch đo lường .....	22
3.1.2    Thiết kế và chế tạo mảng mạch điều khiển cho phần Master ... ..	25
<b>CHƯƠNG 4. THIẾT KẾ HỆ THỐNG ĐIỀU KHIỂN GIÁM SÁT VÀ THU THẬP DỮ LIỆU CHO HỆ PV INVERTER .....</b>	<b>29</b>
4.1    Internet of Things .....	29
4.1.1    Giới thiệu .....	29
4.1.2    Ứng dụng của IoT trong hệ thống điện tử công suất .....	29
4.2    Nhiệm vụ của hệ thống điều khiển giám sát và thu thập dữ liệu cho hệ PV Inverter .....	30



4.3	Thiết kế hệ thống điều khiển giám sát và thu thập dữ liệu cho hệ PV Inverter .....	30
4.3.1	Giao thức giữa mạch điều khiển Master và Raspberry .....	31
4.3.2	Giao thức MQTT .....	32
4.3.3	Thiết kế phần mềm.....	34
<b>CHƯƠNG 5. GHÉP NỐI VÀ KẾT QUẢ THỰC NGHIỆM .....</b>		<b>40</b>
5.1	Quy trình ghép nối .....	40
5.1.1	Ghép nối khối truyền thông Zigbee .....	40
5.1.2	Ghép nối mạch điều khiển và Raspberry Pi.....	41
5.2	Truyền tin giữa các mạch đo lường và mạch điều khiển .....	42
5.3	Truyền tin giữa các bộ điều khiển.....	43
5.4	Truyền tin giữa bộ điều khiển trung tâm và màn hình.....	43
5.5	Truyền tin giữa mạch điều khiển trung tâm và Raspberry Pi .....	44
5.6	Chạy thử nghiệm hệ thống IoT đối với hệ điều áp tích cực.....	45
<b>CHƯƠNG 6. KẾT LUẬN.....</b>		<b>47</b>
6.1	Kết luận .....	47
6.2	Hướng phát triển của đồ án trong tương lai .....	47
<b>TÀI LIỆU THAM KHẢO .....</b>		<b>48</b>
<b>PHỤ LỤC .....</b>		<b>49</b>



## **DANH MỤC HÌNH VẼ**

Hình 1. 1 Sự tăng trưởng tổng điện năng theo mặt trời qua các năm .....	2
Hình 1. 2 Cấu tạo pin mặt trời.....	4
Hình 1. 3 Đường đặc tính P-V của một thiết bị quang điện.....	4
Hình 1. 4 Đặc tính I-V của pin SQ160 khi cường độ thay đổi .....	5
Hình 1. 5 Đặc tính I-V của pin SQ khi nhiệt độ pin thay đổi .....	5
Hình 1. 6 Cấu trúc hệ thống PV Inverter .....	6
Hình 2. 1 Sơ đồ hệ thống PV Inverter.....	8
Hình 2. 2 Mạch điều khiển bộ biến đổi DC/DC.....	9
Hình 2. 3 Mạch điều khiển bộ biến đổi Battery.....	10
Hình 2. 4 Mạch điều khiển bộ biến đổi DC/AC.....	10
Hình 2. 5 Phạm vi định nghĩa CAN trong mô hình OSI [8] .....	12
Hình 2. 6 Lưu đồ thuật toán truyền thông CAN .....	15
Hình 2. 7 Lưu đồ thuật toán chương trình Modbus RTU .....	18
Hình 2. 8 Mô hình vật các lớp tầng của chuẩn Zigbee .....	20
Hình 2. 9 Sơ đồ cấu hình baudrate cho module Zigbee .....	21
Hình 2. 10 Sơ đồ cấu hình chế độ hoạt động cho module Zigbee .....	21
Hình 2. 11 Sơ đồ cấu hình chế độ truyền cho module Zigbee .....	21
Hình 3. 1 Sơ đồ nguyên lý phần đo điện áp .....	23
Hình 3. 2 Sơ đồ nguyên lý phần đo dòng điện .....	23
Hình 3. 3 Layout lớp trên của mạch đo tích hợp mạch điều khiển .....	24
Hình 3. 4 Layout lớp dưới của mạch đo tích hợp mạch điều khiển .....	24
Hình 3. 5 Sơ đồ cấu trúc tổng quát của mạch điều khiển trung tâm .....	25
Hình 3. 6 Sơ đồ kết nối khối MCU .....	26
Hình 3. 7 Sơ đồ khối truyền thông CAN với các mạch Slave .....	26
Hình 3. 8 Sơ đồ khối truyền thông Modbus.....	27
Hình 3. 9 Sơ đồ khối truyền thông Zigbee.....	27
Hình 3. 10 Layout lớp trên của mạch điều khiển trung tâm .....	28
Hình 3. 11 Layout lớp dưới của mạch điều khiển trung tâm .....	28
Hình 4. 1 Sơ đồ hệ thống IoT cho hệ PV Inverter .....	30
Hình 4. 2 Sơ đồ chi tiết hoạt động của hệ thống IoT .....	30
Hình 4. 3 Sơ đồ cấu trúc giao thức MQTT .....	33
Hình 4. 4 Lưu đồ thuật toán trên mạch điều khiển trung tâm .....	34
Hình 4. 5 Lưu đồ thuật toán trên Raspberry Pi .....	35
Hình 4. 6 Lưu đồ thuật toán trên Server.....	36
Hình 4. 7 Cơ sở dữ liệu được tạo trên phpmyadmin.....	37
Hình 4. 8 Giao diện Dashboard của hệ thống điều khiển .....	38



Hình 4. 9 Đồ thị hiển thị sự thay đổi giá trị của hệ thống.....	38
Hình 4. 10 Bảng lưu giá trị nhận được từ hệ thống .....	38
Hình 4. 11 Trang kết nối với Raspberry Pi qua địa chỉ IP .....	39
Hình 5. 1 Kết nối zigbee với mạch điều khiển trung tâm .....	40
Hình 5. 2 Mặt trên của mạch điều khiển trung tâm.....	40
Hình 5. 3 Kết nối Zigbee với Raspberry Pi.....	41
Hình 5. 4 Ghép nối Zigbee với mạch điều khiển trung tâm của hệ AVC.....	41
Hình 5. 5 Ghép nối Zigbee với Raspberry Pi cho hệ AVC.....	42
Hình 5. 6 Thiết lập cấu hình truyền giữa các mạch đo lường .....	42
Hình 5. 7 Thiết lập thông số kỹ thuật cho CAN của STM32.....	43
Hình 5. 8 Thiết lập thông số cho bộ điều khiển trung tâm.....	44
Hình 5. 9 Đồ thị hiển thị kết quả nhận từ mạch Master .....	45
Hình 5. 10 Bảng lưu dữ liệu nhận được từ Master gửi về .....	45
Hình 5. 11 Sơ đồ hệ điều áp tích cực và hệ thống IoT .....	46



## **DANH MỤC BẢNG BIỂU**

Bảng 1. 1 Top 10 quốc gia có tổng công suất sản lượng điện mặt trời lớn .....	2
Bảng 2. 1 Thông số của 2 chế độ truyền CAN .....	13
Bảng 2. 2 Khung bản tin dữ liệu CAN ở chế độ chuẩn .....	14
Bảng 2. 3 Bảng cấu trúc ký tự gửi đi .....	16
Bảng 2. 4 Bảng cấu trúc bản tin Modbus .....	17
Bảng 2. 5 Hàm tiêu biểu trong Modbus .....	17
Bảng 3. 1 Bảng thông số giá trị đo của mạch .....	23
Bảng 4. 1 Cấu trúc bản tin truyền từ STM32 lên Raspberry Pi .....	31
Bảng 4. 2 Bảng cấu trúc kí tự gửi đi từ STM32 lên Raspberry Pi.....	31
Bảng 4. 3 Bảng cấu trúc bản tin truyền về STM32 từ Raspberry Pi.....	32
Bảng 4. 4 Khung truyền bản tin của MQTT .....	34



# CHƯƠNG 1. TỔNG QUAN VỀ HỆ THỐNG ĐIỆN NĂNG LƯỢNG MẶT TRỜI

Trong chương này sẽ trình bày về tình hình và xu thế phát triển của việc sử dụng năng lượng mặt trời. Cũng như giới thiệu về các hệ thống đang sử dụng năng lượng mặt trời hiện nay.

## 1.1 Tổng quan về năng lượng mặt trời

### 1.1.1 Tình hình và xu thế phát triển năng lượng mặt trời

#### 1.1.1.1. Năng lượng mặt trời

Năng lượng mặt trời (NLMT) được phát ra từ mặt trời là nguồn năng lượng sạch, có đặc tính “tái tạo” và có trữ lượng khổng lồ. Nó còn là nguồn gốc của các nguồn năng lượng sạch và tái tạo khác như: năng lượng gió, năng lượng sinh khối, thuỷ năng và năng lượng đại dương.

Mặt trời là một “nhà máy” nhiệt hạch nhân không lò công suất  $3,865 \cdot 10^{17}$  GW. Tuy nhiên, Trái đất chỉ nhận được một phần rất nhỏ năng lượng đó. Cụ thể là mỗi giây Trái đất nhận được  $17,57 \cdot 10^{10}$  MJ, bằng năng lượng khi đốt cháy hết 6 triệu tấn than đá.

Tuy vậy, nguồn năng lượng mặt trời cũng tồn tại nhiều khó khăn trong việc khai thác và sử dụng chúng [12].

#### 1.1.1.2. Công nghệ năng lượng mặt trời

Công nghệ NLMT là công nghệ khai thác năng lượng mặt trời.

Hiện nay, có 2 công nghệ sản xuất điện mặt trời được phát triển rộng rãi đó là :

- Công nghệ quang điện SPV (Solar Photovoltaic, PV)
- Công nghệ hội tụ năng lượng mặt trời CSP (concentrated solar power)

Theo [12] trong công nghệ quang điện SPV, năng lượng ánh sáng mặt trời được chuyển thành dòng điện, nhờ hiệu ứng quang điện, qua các tế bào quang điện hay các Pin mặt trời bé nhỏ. Các pin nhỏ ghép lại thành tấm Pin Mặt trời lớn. Các tấm pin lớn này ghép lại với nhau thành mỏm đun hay dây. Ban đầu, các tấm Pin mặt trời được dùng cho vệ tinh nhân tạo hay phi thuyền không gian, nhưng giờ đây ở nhiều nước đã sử dụng rộng rãi trong công nghiệp và dân dụng.

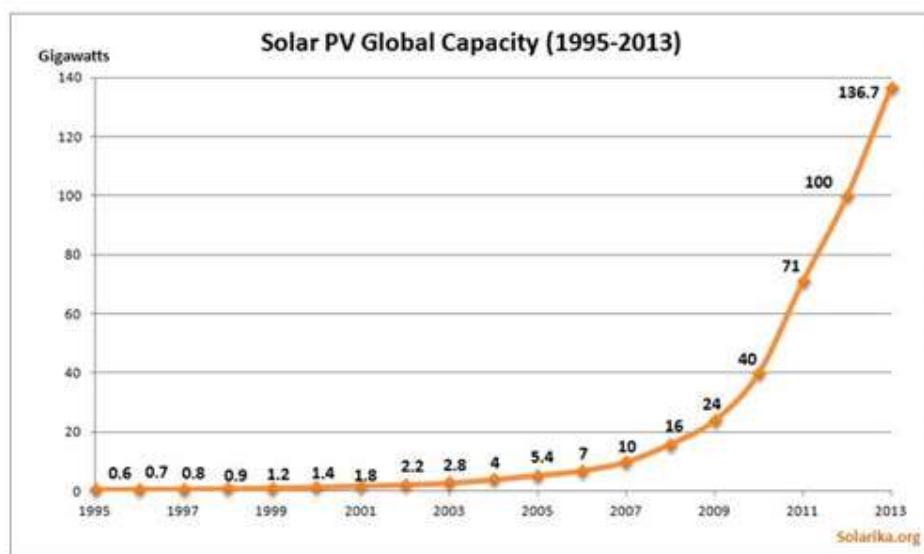
Trong công nghệ CSP hay còn được gọi là công nghệ nhiệt năng mặt trời STE (Solar thermal energy), sử dụng một hệ thống nhiều ống kính, gương phản chiếu và các hệ thống theo dõi nhằm tập trung ánh sáng mặt trời từ một khu vực rộng lớn vào một diện tích nhỏ. Ở đây, nước hoặc chất lỏng đặc biệt khác chứa trong các bể chứa hay ống dẫn được làm nóng lên đến từ nhiệt độ vài chục độ (sử dụng, như sưởi ấm bể bơi, cung cấp nước ấm cho các hộ gia đình, lưu trữ năng lượng phòng khi không có mặt trời chiếu sáng), đến vài trăm độ (tạo thành những dòng hơi nước mạnh làm quay tuôc-bin để sản xuất điện).

Xu thế của phát triển năng lượng mặt trời trên toàn cầu đang chuyển dần sang phát triển các công nghệ năng lượng mặt trời, trong đó công nghệ điện pin mặt trời có vai trò quan trọng nhất.

Trên thế giới, sản lượng điện khai thác từ năng lượng mặt trời tăng nhanh chóng. Dưới đây là bảng số liệu về tổng điện năng sản xuất theo công nghệ SPV của những nước đứng đầu tính đến năm 2013.

Bảng 1. 1 Top 10 quốc gia có tổng công suất sản lượng điện mặt trời lớn

Nước	Tổng điện năng lượng trời (GWp)	Tỷ lệ điện năng mặt trời trong tổng điện năng quốc gia
Đức	35.65	5.3%
Ý	18	9%
Trung Quốc	17.7	0.1%
Nhật	11.86	0.8%
Mỹ	11,42	0.3%
Tây Ba Nha	5.1	2.8%
Pháp	4.67%	0.9%
Úc	3.159%	1.2%
Bỉ	2.82	2.5%
Tiệp Khắc	2.0	3.1%



Hình 1. 1 Sự tăng trưởng tổng điện năng theo mặt trời qua các năm

### **1.1.2 Phương pháp thu năng lượng mặt trời bằng pin mặt trời**

Như đã trình bày ở trên thì năng lượng mặt trời là một trong những nguồn năng lượng tái tạo phổ biến nhất. Từ xa xưa, con người đã biết tận dụng năng lượng mặt trời để sưởi ấm, làm khô lương thực thực phẩm. Ngày nay, nguồn năng lượng đó được khai thác đa dạng và nổi bật nhất là hệ thống sản xuất điện từ năng lượng mặt trời.

Để sản xuất điện từ nguồn năng lượng mặt trời, người ta có thể khai thác trực tiếp thông qua các tấm pin quang điện biến đổi trực tiếp năng lượng ánh sáng mặt trời thành điện năng. Phương pháp thứ hai là chuyển năng lượng mặt trời thành nhiệt năng bằng cách tập trung ánh sáng bằng hệ thống gương để đun sôi nước làm quay tuabin phát ra điện. Và giải pháp chuyển đổi năng lượng mặt trời thành điện năng phù hợp với quy mô các hộ gia đình, hay các nhà máy công nghiệp có quy mô vừa phải, và dễ dàng triển khai mà ít gây ảnh hưởng đến cảnh quan môi trường xung quanh. Nhờ sự phát triển của công nghiệp vật liệu, các tấm pin được sản xuất ngày càng cho hiệu suất cao, giá thành giảm nhưng vẫn cho thấy nhiều ưu điểm hứa hẹn.

#### **1.1.3 Pin mặt trời**

##### *1.1.3.1. Cấu tạo*

Pin mặt trời có cấu tạo từ tinh thể silic là loại phổ biến hiện nay [12]. Ở dạng tinh khiết, nguyên tử silic trung hòa về điện hay còn gọi là số electron và số lỗ trống bằng nhau. Để làm pin mặt trời từ bán dẫn tinh khiết này phải tạo ra hai loại bán dẫn loại p và loại n. Sau đó, ghép chúng lại với nhau để có được lớp tiếp giáp p-n là nền tảng cho tế bào pin.

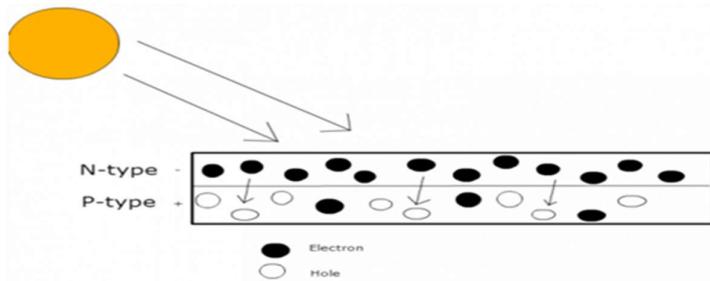
Xuất phát từ phiên bản dẫn tinh khiết người ta pha thêm một ít nguyên tử khác, gọi là pha tạp. Nguyên tử Si có 4 electron ngoài cùng dùng để liên kết với 4 nguyên tử Si đứng cạnh nguyên tử đó. Để tạo ra bán dẫn loại n cần pha thêm nguyên tử có 5 electron ngoài cùng (Photpho) dẫn đến cấu trúc sau pha tạp có thừa 1 electron tự do. Ngược lại, để tạo bán dẫn loại p ta pha tạp một nguyên tử có 3 electron ngoài cùng (Bo) tức là thiếu một electron mới tạo đủ 4 liên kết. Bán dẫn pha tạp thàu một lỗ trống mang điện dương. Bằng cách ghép nối công nghệ hai loại bán dẫn này tạo thành cấu trúc chứa lớp chuyển tiếp p-n. Ở chỗ tiếp xúc p-n này một ít electron ở bán dẫn loại n chạy sang bán dẫn loại p lắp vào lỗ trống thiếu electron. Kết quả là ở lớp tiếp xúc p-n có một vùng thiếu electron cũng thiếu cả lỗ trống, người ta gọi đó là vùng nghèo. Sự dịch chuyển điện tử để lắp vào lỗ trống tạo ra vùng nghèo này cũng tạo nên hiệu ứng gọi là hiệu ứng ở tiếp xúc p-n, đối với Si vào cỡ 0,6V đến 0,7V. Đây là hiệu điện thế tạo nên điện trường tại lớp tiếp giáp.

##### *1.1.3.2. Nguyên lý hoạt động của pin mặt trời*

Pin năng lượng mặt trời (hay pin quang điện, tế bào quang điện) là thiết bị bán dẫn chứa lượng lớn các diode p-n, dưới sự hiện diện của ánh sáng mặt trời có khả năng tạo ra dòng điện sử dụng được. Sự chuyển đổi này gọi là hiệu ứng quang điện.

Pin mặt trời gồm nhiều tế bào quang điện(cells) ghép lại với nhau bởi mỗi tế điện chỉ tạo ra được dòng điện và công suất rất nhỏ nên cần phải ghép nối lại. Trên thị trường, mỗi tấm pin mặt trời thường được ghép chủ yếu từ 36 cells hoặc 72 cells.

Vật liệu chủ yếu chế tạo pin Mặt trời (và cho các thiết bị bán dẫn) là silic dạng tinh thể.

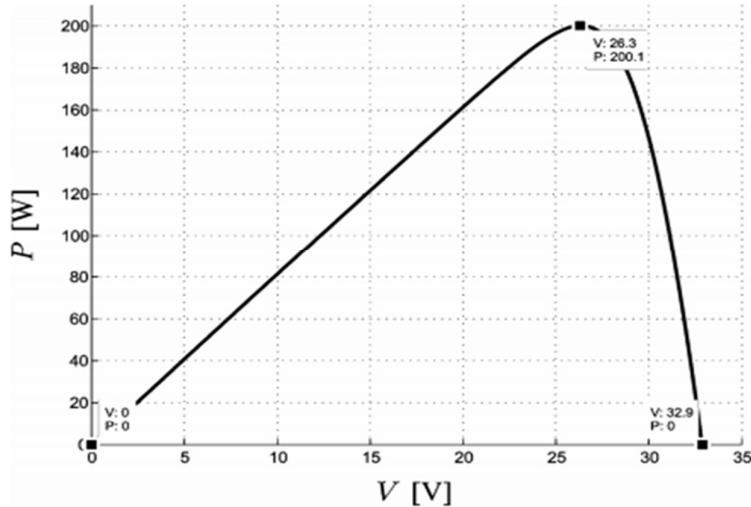


Hình 1. 2 Cấu tạo pin mặt trời

Khi chất bán dẫn silicon tiếp xúc với năng lượng (photon từ ánh sáng mặt trời), các electron tự do ở điện cực N sẽ di chuyển sang để lấp đầy các lỗ trống bên điện cực P. Sau đó, các electron từ điện cực N và điện cực P sẽ cùng nhau tạo ra điện trường. Các tế bào năng lượng mặt trời sẽ trở thành một diode, cho phép electron di chuyển từ điện cực P đến điện cực N, không cho phép di chuyển ngược lại. Sự di chuyển của các electron tự do từ điện cực N tới điện cực P tạo ra dòng điện 1 chiều.

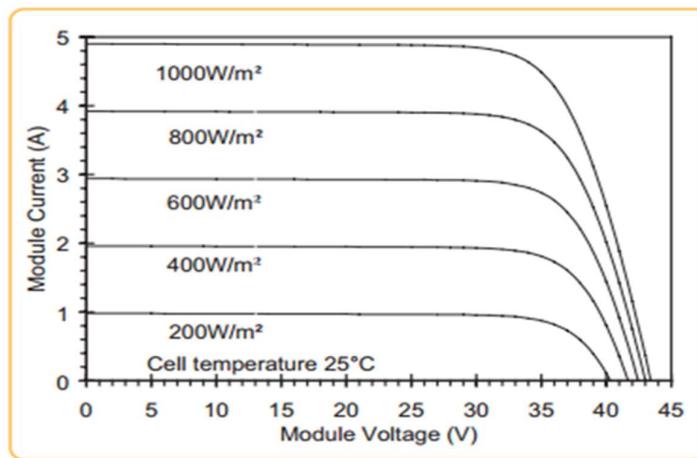
#### 1.1.3.5. Đặc tính của pin mặt trời

Mỗi tấm pin PV được đặc trưng bởi 2 đường cong phi tuyến: đặc tính I-V và P-V. Đường đặc tính P-V được tính toán dựa trên đường đặc tính I-V. Hai đường đặc tính này có dạng như sau:

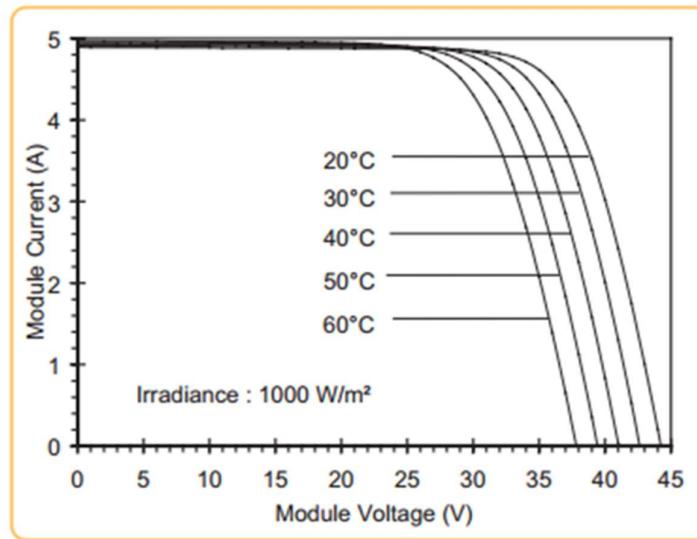


Hình 1. 3 Đường đặc tính P-V của một thiết bị quang điện

Với yêu cầu sử dụng các tấm pin SQ160, từ tài liệu [3] chúng ta sẽ có được đường đặc tính I-V của tấm pin này như sau:



Hình 1. 4 Đặc tính I-V của pin SQ160 khi cường độ thay đổi



Hình 1. 5 Đặc tính I-V của pin SQ khi nhiệt độ pin thay đổi

Từ đường đặc tính trên một điều cần chú ý là đường cong đặc tính còn rất phụ thuộc rất lớn vào nhiệt độ và cường độ chiếu sáng: Như 2 đồ thị trên ta có thể đưa ra nhận xét, khi cường độ ánh sáng tăng lên, đường đặc tính I-V có xu hướng di chuyển lên trên. Với trường hợp thay đổi nhiệt độ, khi tăng nhiệt độ, đường đặc tính có xu hướng di chuyển về phía bên phải và ngược lại.

#### 1.1.4 Các hệ thống với pin mặt trời

Hiện tại, đối với hệ thống sử dụng pin mặt trời, người ta đã xây dựng một số hệ thống tiêu biểu như sau:

- Hệ thống điện mặt trời độc lập
- Hệ thống điện mặt trời độc lập có bù lưới
- Hệ thống điện mặt trời độc lập kết hợp hòa lưới
- Hệ thống điện mặt trời trực tiếp
- Hệ thống điện mặt trời nối lưới
- Hệ thống điện mặt trời nối lưới có dự trữ
- Hệ thống điện mặt trời độc lập kết hợp
- Hệ thống điện mặt trời độc lập thông minh

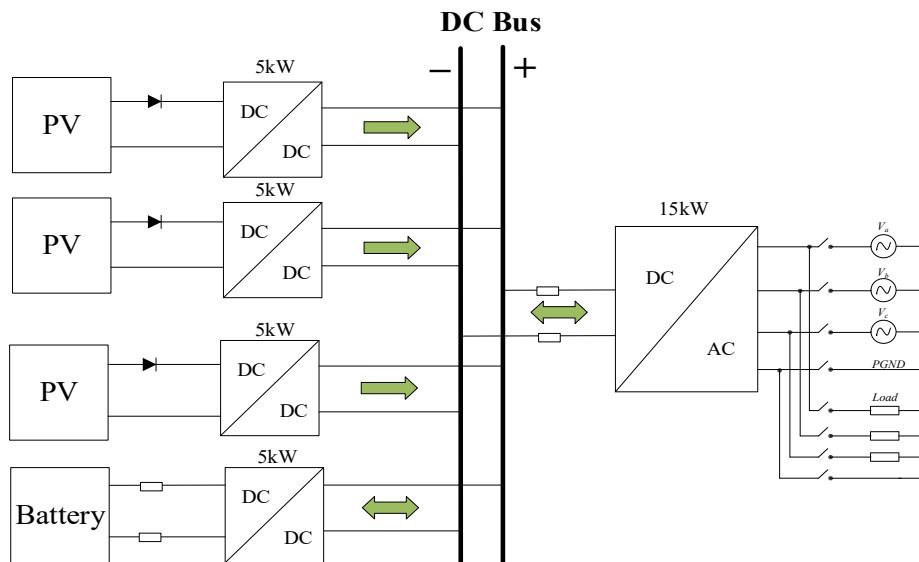
## 1.2 Giới thiệu về hệ thống biến đổi điện tử công suất sử dụng pin năng lượng mặt trời

### 1.2.1 Giới thiệu

Trong các hệ thống pin mặt trời kết nối lưới điện thì bộ biến đổi điện tử công suất giữ vai trò rất quan trọng trong các hệ thống điều khiển, bởi đặc tính của pin mặt trời là có công suất phát luôn biến đổi do phụ thuộc điều kiện thời tiết. Sự thay đổi công suất phát của chúng có thể gây ảnh hưởng tiêu cực đến chất lượng điện năng của lưới điện, như gây dao động điện áp, tăng độ méo sóng hài dòng điện,... Để đáp ứng yêu cầu ngày càng cao về chất lượng điện năng đã đặt ra yêu cầu thực tế là: cần thiết phải có những bộ biến đổi điện tử công suất đáp ứng linh hoạt, trao đổi công suất và đảm bảo được các chỉ tiêu chất lượng điện năng.

### 1.2.2 Hệ thống PV Inverter ba pha

Cấu trúc hệ thống PV Inverter được thể hiện như sau:



Hình 1. 6 Cấu trúc hệ thống PV Inverter

Nhìn vào sơ đồ cấu trúc ta có thể thấy hệ thống gồm các bộ biến đổi điện tử công suất chính như sau:

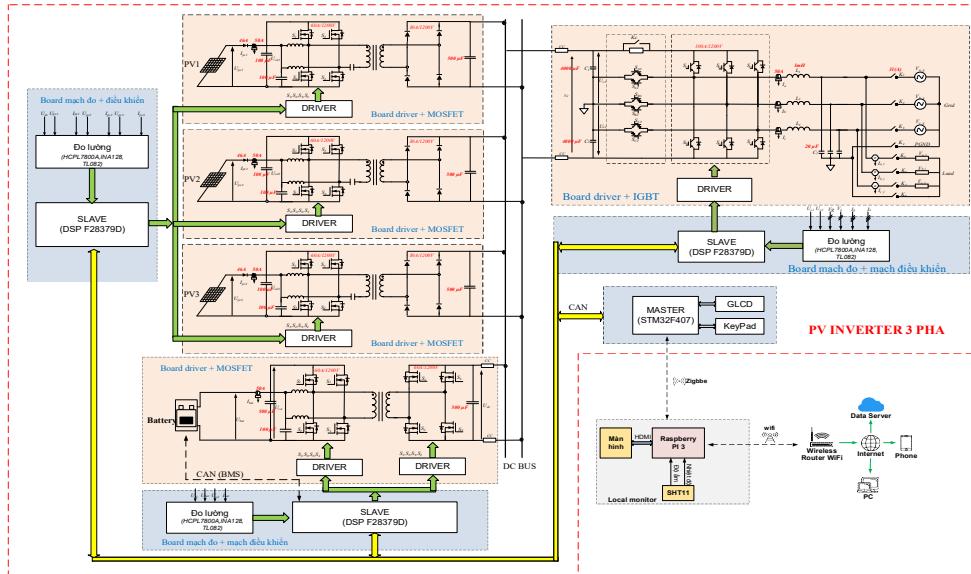
- Pin mặt trời (PV array): Có chức năng biến đổi ánh sáng thành dòng điện DC.
- Bộ biến đổi DC/DC: Được dùng để cấp nguồn điện DC cho bộ biến tần Inverter.
- Battery: Được dùng để cấp điện DC cho bộ biến tần Inverter.
- Biến tần (Inverter): Có chức năng biến đổi điện DC từ ắc-quy và từ bộ biến đổi DC/DC thành dòng điện AC cấp cho tải và lưới.

Ưu điểm của hệ thống sử dụng bộ lưu trữ điện Battery đó là giúp hệ thống có thể hoạt động 24/7. Còn ở hệ thống không dùng Battery thì điện áp DC của pin mặt trời sẽ được chuyển thành điện áp AC 3 pha cấp cho tải.

## CHƯƠNG 2. TỔNG QUAN VỀ HỆ THỐNG PV INVERTER

Trong chương 2 này sẽ trình bày tổng quan về hệ thống PV Inverter được sử dụng cho đề tài.

### 2.1 Tổng quan về hệ thống PV Inverter



Hình 2. 1 Sơ đồ hệ thống PV Inverter

Hệ thống thí nghiệm gồm 5 phần chính:

- Bộ DC/DC cho PV 3 pha.
- Bộ DC/DC cho Battery 3 pha.
- Bộ DC/AC 3 pha.
- Khối điều khiển Master.
- Hệ thống IOT cho PV Inverter.

Nhìn vào sơ đồ tổng quan hệ thống chúng ta có thể thấy rằng hệ thống trao đổi dữ liệu sẽ đây xây dựng qua các kết nối: mạch điều khiển trung tâm Master với các mảng mạch điều khiển điện tử công suất Slave (MĐK Master – MĐK Slave – Mạch đo lường), mạch điều khiển trung tâm Master với phần IOT (MĐK Master – Raspberry Pi 3 – Database).

Mạch điều khiển trung tâm có vai trò thu thập dữ liệu, ra quyết định, điều khiển các hoạt động của hệ thống, ra lệnh cho các mạch điều khiển điện tử công suất làm việc trao đổi dữ liệu với các thiết bị giám sát.

Mạch điều khiển mảng điện tử công suất có vai trò trực tiếp điều khiển các van bán dẫn, tính toán lượng đặt, tham số các mạch vòng điều chỉnh, có vai trò đặc biệt quan trọng đến tính toán động học của hệ thống.

Phần IOT – Raspberry có vai trò giống như một máy tính nhúng có connect với màn hình để thu thập dữ liệu, hiển thị giám sát, điều khiển từ xa cũng như là đưa dữ liệu lưu trữ vào database [3].

## 2.2 Các thành phần của bộ điều khiển

Phần điều khiển bao gồm các mạch chính là mạch đo, mạch điều khiển Slave và mạch điều khiển chính Master.

### 2.2.1 Mạch điều khiển Slave

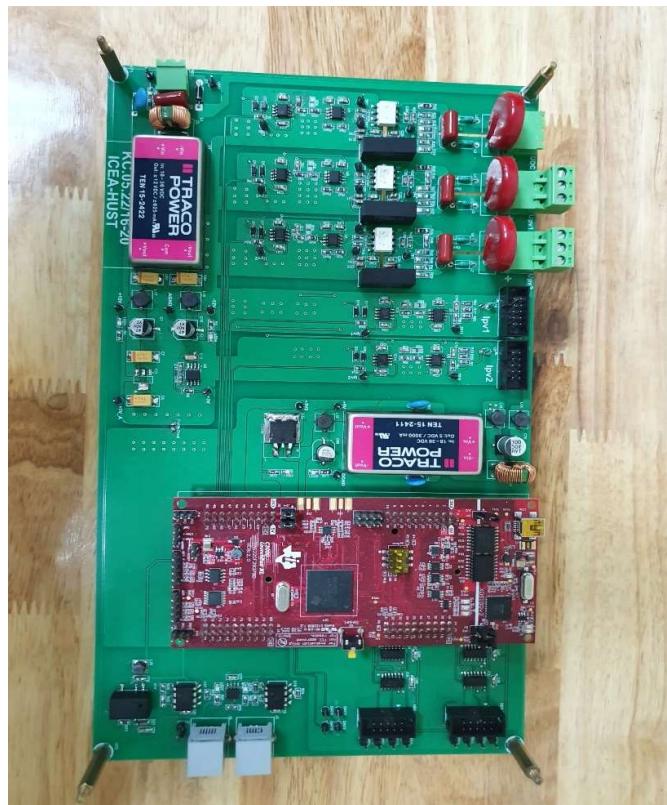
Mạch điều khiển Slave có nhiệm vụ thực hiện các mạch vòng điều chỉnh. Ngoài ra, mạch điều khiển Slave còn giao tiếp với mạch điều khiển Master qua truyền thông CAN.

Hệ thống có ba mạch điều khiển Slave:

- Mạch điều khiển bộ biến đổi DC/DC.
- Mạch điều khiển bộ biến đổi DC/AC.
- Mạch điều khiển bộ biến đổi Battery.

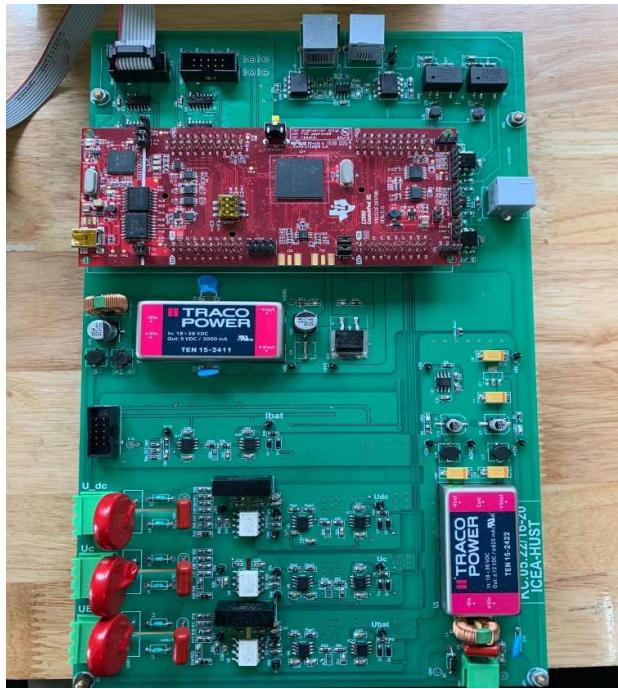
Cả ba mạch đều sử dụng vi điều khiển DSP 28379D của hãng Texas Intrusment. Đây là dòng vi điều khiển dấu phẩy động 32 bit, tần số xung clock có thể lên đến 200MHz, hỗ trợ tính toán dấu phẩy động. Ngoài ra, DSSP 28379D có bộ tính toán số học (Control Law Accelerator – CLA) chuyên dùng cho các bài toán có cấu trúc điều khiển và tính toán phức tạp.

#### 2.2.1.1. Mạch điều khiển bộ biến đổi DC/DC



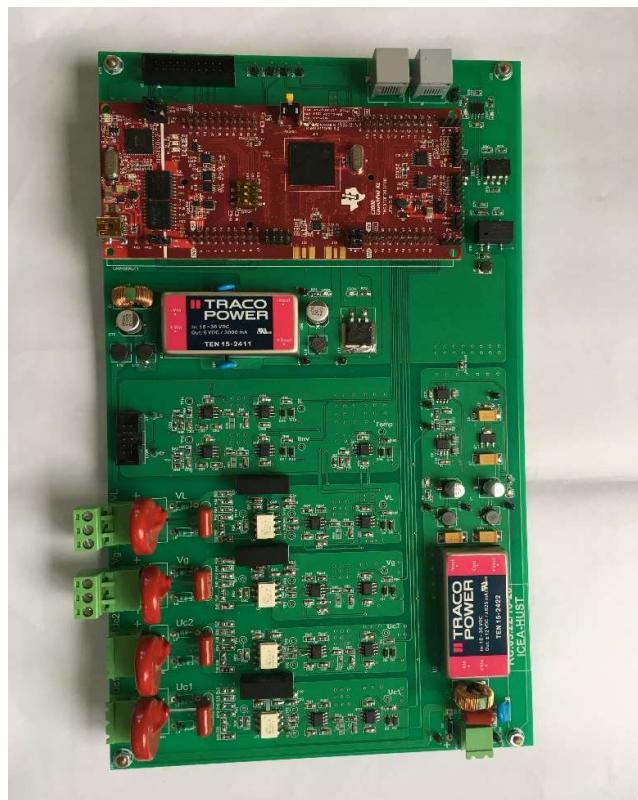
Hình 2. 2 Mạch điều khiển bộ biến đổi DC/DC

### 2.2.1.2. Mạch điều khiển bộ biến đổi Battery



Hình 2. 3 Mạch điều khiển bộ biến đổi Batterry

### 2.2.1.3. Mạch điều khiển bộ biến đổi DC/AC



Hình 2. 4 Mạch điều khiển bộ biến đổi DC/AC

### **2.2.2 Mạch điều khiển trung tâm Master**

Ý tưởng mạch điều khiển Master với nhiệm vụ điều khiển, giám sát và cài đặt các thông số xuống mạch Slave để giảm bớt công việc cho Slave. Đây có thể được coi là mạch điều khiển cấp trên. Với vai trò đưa ra các lượng đặt, tham số điều chỉnh, mạch Master được thiết kế sử dụng vi điều khiển STM32F407 của hãng STMicroelectronics. Mạch Master giao tiếp với các mạch điều khiển Slave qua truyền thông CAN. Mạch điều khiển Master cũng kết nối với Raspberry Pi qua truyền thông Zigbee. Và bản thân Raspberry cũng được kết nối với màn hình HMI qua HDMI. Chính vì vậy, tất cả việc giám sát, cài đặt, hiển thị dữ liệu đều được thực hiện trên màn hình hoặc trên máy tính thông qua ứng dụng website.

Mạch sử dụng vi điều khiển STM32F407 để giao tiếp với các ngoại vi khác. Vi điều khiển sẽ giao tiếp với các ngoại vi khác như bàn phím, eeprom, IC Read Time, GLCD.

### **2.3 Các mảng mạch đo lường**

Trong các hệ thống điều khiển điện tử công suất, để thực hiện thuật toán điều khiển, cần phải đo các tín hiệu phản hồi là dòng điện, điện áp thực của hệ thống. Vì thế mạch vòng này cần phải có đáp ứng động học nhanh nhất. Trong hệ thống PV Inverter khâu đo lường cho các bộ biến đổi DC/DC, bộ biến đổi DC/AC, bộ biến đổi Battery bao gồm đo điện áp xoay chiều, dòng điện xoay chiều, điện áp một chiều, điện áp PV, dòng điện PV. Với yêu cầu điều khiển đã được nêu trên thì một điều kiện tiên quyết đó là mạch đo lường phải hoàn toàn cách ly với mạch điều khiển.

Các tín hiệu dòng điện, điện áp thực tế qua mạch đo chuyển đổi thành tín hiệu 0-3V để vi điều khiển có thể đọc được qua module ADC. Mạch đo sử dụng HCPL 7800A để đo điện áp, LEM LA55-P/SPI để đo dòng điện.

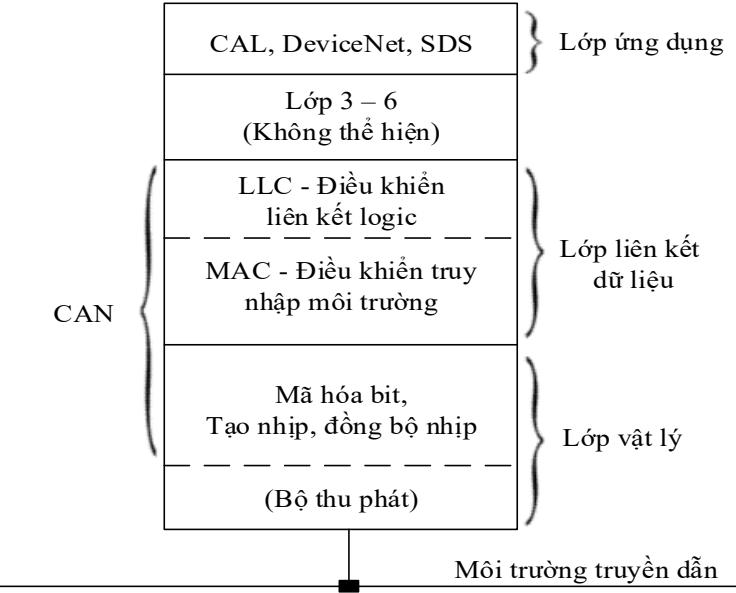
### **2.4 Truyền thông trong hệ thống**

#### **2.4.1 Truyền thông giữa các bộ điều khiển**

##### *2.4.1.1. Truyền thông CAN*

CAN (Controller Area Network) là giao thức giao tiếp nối tiếp hỗ trợ mạnh cho những hệ thống điều khiển thời gian thực phân bố với độ ổn định, bảo mật và đặc biệt chống nhiễu cực kỳ tốt.

CAN xuất phát là một phát triển chung của hai hãng Bosch và Intel phục vụ việc nối mạng trong các phương tiện giao thông cơ giới để thay thế cách nối điểm-điểm cổ điển, sau được chuẩn hóa quốc tế trong ISO 11898. Chiều dài dây dẫn tổng cộng trong cách nối điểm-điểm có thể lên tới vài km, tốc độ truyền dẫn tương đối cao ở khoảng cách ngắn.



Hình 2. 5 Phạm vị định nghĩa CAN trong mô hình OSI [8]

CAN thực chất là chuẩn giao thức từ phần trên lớp vật lý cho tới hết lớp liên kết dữ liệu, vì vậy không quy định cụ thể về chuẩn truyền dẫn cũng như môi trường truyền thông. Thực tế, cáp đôi dây xoắn kết hợp với chuẩn RS485 cũng như cáp quang được sử dụng rộng rãi. Do có sự ràng buộc giữa tốc độ truyền và chiều dài dây dẫn trong phương pháp truy nhập bus CSMA/CA, tốc độ truyền tối đa là 1Mbit/s ở khoảng cách 40m và 50kbit/s ở khoảng cách 1000m. Số trạm thông thường hạn chế ở con số 64 đối với cấu trúc đường thẳng sử dụng cáp đôi dây xoắn. Công nghệ cáp của mạng CAN có đường dây dẫn đơn giản, giảm tối thiểu hiện tượng sự đột biến tín hiệu. Sự truyền dữ liệu thực hiện nhờ cặp dây truyền tín hiệu vi sai, có nghĩa là chúng ta đo sự khác nhau giữa hai đường dây (CANH và CANL). Đường dây bus kết thúc bằng điện trở 120 ohm (thấp nhất là 108 ohm và tối đa là 132 ohm) ở mỗi đầu.

Đặc trưng của CAN là phương thức định địa chỉ và giao tiếp hướng đối tượng. Mỗi thông tin trao đổi trong mạng được coi như một đối tượng, được gán một mã số căn cước. Thông tin được gửi trên bus theo kiểu truyền thông báo với độ dài có thể khác nhau. Các thông báo không gửi tới một địa chỉ nhất định mà bất cứ trạm nào cũng có thể nhận theo nhu cầu. Nội dung thông báo được phân biệt qua mã căn cước (Identifier). Một trạm có thể yêu cầu một trạm khác gửi dữ liệu bằng cách gửi một khung yêu cầu (Remote Frame), trạm có khả năng cung cấp nội dung thông tin sẽ gửi trả lại một khung dữ liệu (Data Frame) có cùng mã căn cước với khung yêu cầu.

Cơ chế giao tiếp hướng đối tượng CAN mang lại tính linh hoạt và tính nhất quán dữ liệu của hệ thống. Một trạm CAN không cần biết thông tin cấu hình hệ thống nên việc bổ sung hay bỏ đi một trạm trong mạng không đòi hỏi bất cứ một sự thay đổi nào về phần cứng hay phần mềm ở các trạm khác. Tính nhất quán dữ liệu được đảm bảo qua các phương pháp gửi đồng loạt và xử lý lỗi. Mỗi trạm có thể nhận nhiều loại bản tin khác nhau, ngược lại một bản tin có thể được nhận

bởi nhiều trạm và công việc được thực hiện một cách đồng bộ trong hệ thống phân bố. ID của bản tin phụ thuộc vào mức độ ưu tiên của bản tin. Điều này cho phép phân tích thời gian phản hồi của từng bản tin. Điều này có ý nghĩa quan trọng trong việc thiết kế hệ thống nhúng thời gian thực.

Ngoài ra, chuẩn CAN còn định nghĩa nhiều cơ chế khác để kiểm tra lỗi, xử lý lỗi... cơ chế kiểm tra và xử lý lỗi chia làm 5 loại lỗi: Bit error, Stuff error, CRC error, Form error, ACK error.

Một số đặc điểm quan trọng khác trong truyền thông CAN:

Phương pháp mã hóa bit: None-return-to-zero

Kỹ thuật nhồi bit: một bit có giá trị ngược lại khi có 5 bit liên tiếp giống nhau trong khi truyền.

Trạng thái “trội”, “lặn” của bit: hai trạng thái “0” và “1” lần lượt ứng với trạng thái “trội” và “lặn”. Việc định nghĩa hai trạng thái này tạo khả năng giải quyết tranh chấp khi nhiều hơn một Master cùng muốn chiếm quyền sử dụng bus.

Tính chất vật lý trên đường truyền: đường truyền có tính vi sai, sự miễn trừ tác động điện từ được bảo đảm vì hai dây của bus đều bị tác động như nhau cùng một lúc bởi tín hiệu nhiễu. Ngoài ra, một số tính chất vật lý của đường truyền được phân biệt bởi hai chế độ truyền là CAN tốc độ thấp và CAN tốc độ cao.

Bảng 2.1 Thông số của 2 chế độ truyền CAN

Thông số	CAN Low Speed	CAN High Speed
Tốc độ	125 kb/s	125 kb/s tới 1 Mb/s
Số trạm trên bus	2 tới 20	2 tới 30
Trạng thái trội	CANH = 4V CANL = 1V	CANH = 3,25V CANL = 1,5V
Trạng thái lặn	CANH = 1,75V CANL = 3,25V	CANH = 2,5V CANL = 2,5V
Tính chất của cáp	30pF giữa cap và dây	R = 2 * 120 Ω
Mức điện áp cung cấp	5V	5V

**Giải quyết tranh chấp trên bus:** phương thức giao tiếp của bus CAN là sự phát tán thông tin (broadcast): mỗi điểm kết nối vào mạng thu nhận frame truyền từ trạm phát. Sau đó, mỗi trạm sẽ quyết định việc xử lý bản tin, có trả lời hay không, có phản hồi hay không. Giao thức CAN cho phép các trạm khác nhau đưa dữ liệu cùng lúc và một quá trình nhanh chóng, ổn định của cơ chế phân xử sẽ xác định xem trạm nào được phát đầu tiên. Để xử lý thời gian thực, dữ liệu phải được truyền nhanh. Điều này ảnh hưởng không chỉ đường truyền vật lý cho phép tới 1Mbit/s, mà còn đòi hỏi một sự cấp phát nhanh bus trong trường hợp xung đột, khi mà rất nhiều trạm muốn truyền đồng thời. Khi trao đổi dữ liệu trên bus, thứ tự sẽ được xác định dựa vào loại thông tin. Ví dụ, các giá trị hay biến đổi

nhanh, như trạng thái của một cảm biến, hay phản hồi của một động cơ, phải được truyền liên tục với độ trễ thấp nhất, hơn là các giá trị khác như nhiệt độ của động cơ, các giá trị thay đổi ít. Trong mạng CAN, phần ID của mỗi bản tin, là một từ gồm 11 bit (version 2.0A) xác định mức ưu tiên. Phần ưu tiên này nằm ở đầu mỗi bản tin. Mức ưu tiên được xác định bởi 7 bit cho version 2.0A, tối 127 mức và mức 128 là 0000000 theo NMT (Netword Management) Quy trình phân xử của bus dựa trên phân giải từng bit, theo những trạm đang tranh chấp, phát đồng thời trên Bus, trạm nào mức ưu tiên thấp hơn sẽ mất cạnh tranh với trạm có mức ưu tiên cao.

**Các khung bản tin trong truyền tin CAN:** CAN định nghĩa 4 kiểu khung bản tin.

Khung dữ liệu (Data Frame): mang dữ liệu từ một trạm truyền tới các trạm nhận.

Khung yêu cầu dữ liệu (Remote Frame): được gửi từ một trạm yêu cầu truyền khung dữ liệu với cùng mã căn cước.

Khung lỗi (Error Frame): được gửi từ bất cứ trạm nào phát hiện lỗi bus.

Khung quá tải (Overload Frame): được sử dụng nhằm tạo một khoảng cách thời gian bù sung giữa hai khung dữ liệu hoặc yêu cầu dữ liệu trong trường hợp một trạm bị quá tải

Sau đây, chỉ giới thiệu về hai loại bản tin: Data Frame và Remote Frame.

#### **Khung dữ liệu-Frame data (Standard frame):**

Bảng 2. 2 Khung bản tin dữ liệu CAN ở chế độ chuẩn

SOF	ID	RTR	IDE	R <sub>0</sub>	DLC	Data	CRC	ACK	EOF	IFS
1 bit	11 bit	1 bit	1 bit	1 bit	3 bit	0-8 byte	16 bit	2 bit	7 bit	-

Trong đó:

SOF: bit Start of frame luôn ở trạng thái trội (0)

ID: 11 bit mã hóa bản tin

RTR: 1 bit RTR – remote transmit request để phân biệt data frame và remote frame

Bảng “trội” : data frame, bảng “lặn” : remote frame.

IDE: 1 bit để phân biệt frame chuẩn và frame mở rộng

Bảng “trội” : frame chuẩn, bảng “lặn” : frame mở rộng

R0: 1 bit trội .

DLC: 3 bit định độ dài data trong frame .

Data: từ 0 -8 byte dựa vào DLC .

CRC: gồm 15 bit CRC và 1 bit CRC delimiter.

ACK: gồm 1 bit ACKnowledge và 1 bit delimiter

EOF: 7 bit recessive

IFS: inter-frame space, khoảng cách tối thiểu giữa hai khung truyền.

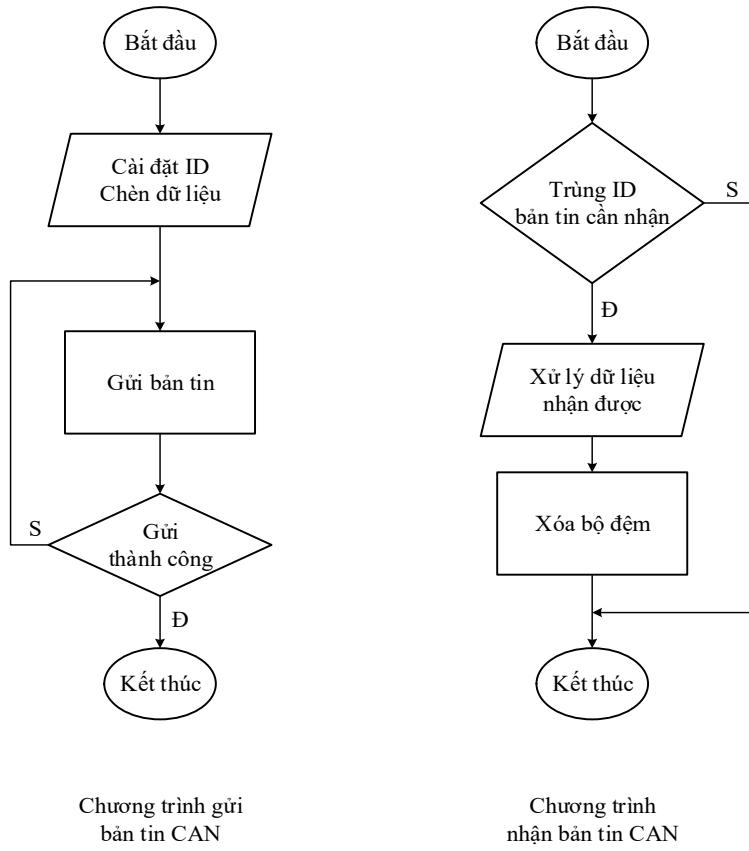
### **Khung yêu cầu dữ liệu-Frame remote:**

Dùng để yêu cầu truyền data frame tới một trạm khác. Remote Frame có cấu trúc giống với Data Frame, nhưng có DLC = 0, và không có Data field.

#### *2.4.1.2. Lưu đồ thuật toán truyền thông CAN*

Như đã trình bày ở trên, truyền thông CAN định hướng bản tin, do đó mỗi bản tin sẽ có một ID xác định. Mỗi khi gửi hay nhận cần xác định ID của bản tin đó để bên nhận quyết định có xử lý bản tin hay không. Tuy nhiên, đối với mạch điều khiển trung tâm, là trạm giao tiếp với tất cả các bộ điều khiển phía dưới, do đó nó sẽ chấp nhận tất cả các bản tin mà bộ điều khiển phía dưới truyền đến.

Thuật toán bao gồm 2 phần: phần gửi bản tin đi và xử lý bản tin nhận về.



Hình 2. 6 Lưu đồ thuật toán truyền thông CAN

#### **2.4.2 Truyền thông giữa mạch điều khiển trung tâm với màn hình**

Chuẩn truyền thông kết nối với PLC được sử dụng là Modbus. Do đó, sử dụng Modbus RTU để trao đổi dữ liệu giữa vi điều khiển với màn hình, với vi điều khiển đóng vai trò là master, còn màn hình đóng vai trò là slave.

#### *2.4.2.3. Truyền thông Modbus RTU*

**Modbus:** là một giao thức do hãng Modicon phát triển. Theo mô hình ISO/OSI thì Modbus thực chất là một chuẩn giao thức và dịch vụ thuộc lớp ứng dụng.

Modbus mô tả quá trình giao tiếp giữa một bộ điều khiển với các thiết bị khác thông qua cơ chế yêu cầu/đáp ứng.

Dựa trên cơ chế giao tiếp phụ thuộc vào hệ thống truyền thông cấp thấp. Cụ thể có thể phân chia Modbus ra hai loại là Modbus chuẩn, và Modbus trên các mạng khác.

**Modbus chuẩn:** sử dụng giao diện nối tiếp RS485. Các bộ điều khiển này có thể được nối mạng trực tiếp qua các modem. Các trạm giao tiếp qua cơ chế Master/Slave, và chỉ một thiết bị chủ có quyền gửi yêu cầu, thiết bị từ sẽ gửi lại thông báo đáp ứng.

**Modbus trên các mạng khác:** với một số mạng như Modbus Plus và MAP sử dụng Modbus là giao thức cho lớp ứng dụng, các thiết bị có thể giao tiếp theo cơ chế riêng của mạng đó.

**Chu trình yêu cầu đáp ứng:** Giao thức Modbus định nghĩa khuôn dạng bản tin

- Địa chỉ trạm nhận: 0-247, với 0 là địa chỉ gửi đồng loạt.
- Mã hàm: gọi chỉ thị hành động trạm từ cần thực hiện yêu cầu.
- Dữ liệu: chứa các thông tin bổ sung cho việc thực hiện hàm.
- Thông tin kiểm lỗi: giúp kiểm tra độ chính xác bản tin.

Chế độ truyền: ASCII, RTU, RTU Extend...

Trong ứng dụng giao tiếp giữa STM32F4 và màn hình sử dụng chế độ truyền Modbus RTU với STM32F4 là trạm chủ Master, và màn hình là trạm từ Slave.

#### Chế độ RTU:

Mỗi byte được gửi thành một ký tự 8 bit, mỗi thông báo phải truyền thành một dòng liên tục. Cấu trúc một ký tự gửi đi:

Bảng 2.3 Bảng cấu trúc ký tự gửi đi

Start	0	1	2	3	4	5	6	7	P	Stop
-------	---	---	---	---	---	---	---	---	---	------

Bao gồm:

- 1 bit Start.
- 8 bit dữ liệu.
- 1 bit Parity chẵn/lẽ (nếu có).
- 1 bit Stop hoặc 2 bit Stop nếu không sử dụng bit Parity.

**Cấu trúc bản tin:** Mỗi khung bản tin bao gồm nhiều ký tự, các ký tự được truyền liên tục thành dòng.

Trong chế độ RTU, một thông báo bắt đầu và kết thúc với một khoảng trống yên lặng tối thiểu là 3.5 thời gian ký tự, khoảng trống kết thúc của bản tin trước có thể là khởi đầu của bản tin sau.

Cấu trúc bản tin Modbus:

Bảng 2. 4 Bảng cấu trúc bản tin Modbus

<b>Khởi đầu</b>	<b>Địa chỉ</b>	<b>Mã hàm</b>	<b>Dữ liệu</b>	<b>Mã CRC</b>	<b>Kết thúc</b>
(---)	8 bit	8 bit	n x 8 bit	16 bit	(---)

Tiếp đến là trường địa chỉ: 1 byte; trường mã hàm: 1 byte; trường dữ liệu: n byte ( $n \geq 0$ ); trường kiểm tra lỗi: CRC, 2 byte.

### Trường địa chỉ:

Các giá trị địa chỉ hợp lệ: 0-247, với địa chỉ 0 dành riêng cho bản tin gửi đồng loạt. Khi thiết bị tò nhận thông báo và gửi lại đáp ứng, sẽ đưa địa chỉ của mình vào bản tin để thiết bị chủ biết trạm nào đã phản hồi.

### Trường mã hàm:

Giá trị hợp lệ: 1-255; trong đó các mã hàm yêu cầu: 1-127. Khi thiết bị tò trả lời, nó cũng dùng chính mã hàm đó trong thông báo đáp ứng bình thường.

Một số hàm tiêu biểu trong Modbus [13]:

Bảng 2. 5 Hàm tiêu biểu trong Modbus

<b>Mã hàm Modbus</b>	<b>Tên hàm</b>	<b>Ý nghĩa</b>	<b>Loại địa chỉ tương ứng</b>
01	Đọc trạng thái coil	Đọc dữ liệu bit (N bits)	0x
02	Đọc đầu vào rời rạc	Đọc dữ liệu bit	1x
03	Đọc nhiều thanh ghi	Đọc giá trị nhiều thanh ghi (số nguyên, số thực,...)	4x
04	Đọc thanh ghi đầu vào	Đọc giá trị một thanh ghi	3x
05	Ghi giá trị coil đơn	Ghi giá trị một bit	0x
06	Ghi giá trị một thanh ghi	Ghi giá trị một thanh ghi	4x
15	Ghi nhiều coil	Ghi giá trị nhiều bit	0x
16	Ghi nhiều thanh ghi	Ghi giá trị nhiều thanh ghi	4x

Với các vùng nhớ 0x, 1x, 3x, 4x là các vùng nhớ mặc định của giao thức truyền thông Modbus.

0x: vùng nhớ bit, cho phép đọc và ghi.

1x: vùng nhớ bit, là vùng nhớ chỉ đọc.

3x: vùng nhớ word (2 byte), là vùng nhớ chỉ đọc.

4x: vùng nhớ word, cho phép đọc và ghi.

### Trường dữ liệu:

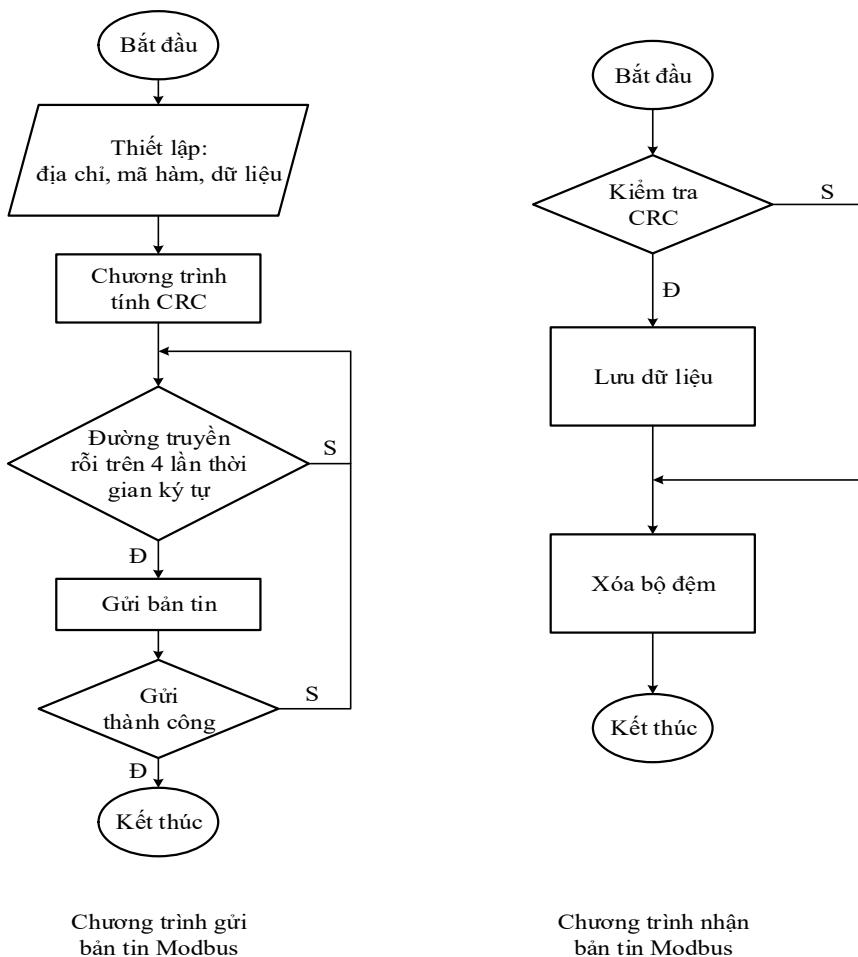
Chứa thông tin bổ sung cho mã hàm. Có thể chứa địa chỉ thanh ghi, giá trị thanh ghi, số lượng thanh ghi... tùy thuộc vào từng hàm.

### Trường kiểm tra lỗi (bảo toàn dữ liệu):

Modbus chuẩn chế độ RTU sử dụng lồng ghép hai chế độ bảo toàn dữ liệu. Khi mỗi byte được gửi đi có thể kiểm tra từng byte thông qua bit Parity, và cả bản tin được kiểm tra lỗi thông qua mã CRC.

Mã CRC được ứng dụng trong chế độ RTU dài 16 bit. Đa thức phát được sử dụng là:  $G = 1010\ 0000\ 0000\ 0001$ . Trong khung bản tin, byte thấp của mã CRC được gửi đi trước, byte cao được gửi đi sau.

#### 2.4.2.4. Lưu đồ thuật toán truyền thông Modbus



Hình 2. 7 Lưu đồ thuật toán chương trình Modbus RTU

Giao tiếp giữa mạch điều khiển trung tâm và màn hình (LCD) sử dụng chuẩn truyền thông Modbus RTU. Trong đó, mạch điều khiển trung tâm đóng vai trò là trạm chủ, còn màn hình đóng vai trò là trạm từ.

Thuật toán bao gồm hai phần: phần gửi bản tin đi và xử lý bản tin nhận về.

## 2.4.3 TruyỀn thÔng giỮA mẠCH ĐIỀU KHIỂN trUNG tÂM VÀ RASPBERRY PI

### 2.4.3.5. TruyỀn thÔng Zigbee

Zigbee là một giao thức truyỀn thÔng khÔng dÂy bÂc cao đƯợc phát triËn dựa trÊn chuÂn truyỀn thÔng khÔng dÂy IEEE 802.15.4, sử dụng tín hiệu radio tương đối ngắn, cấu trúc cơ bản của nó bao gồm tầng vật lý và địa chỉ MAC cho các mạng cá nhân. Zigbee thích hợp với những ứng dụng khÔng đòi hỏi tốc độ truyỀn dữ liệu quá cao nhưng cần độ bảo mật lớn và thời gian hoạt động dài. Các mạng sử dụng song radio tương tự Zigbee đƯỢC đƯỢc nghiên cứu từ những năm 1998-1999 khi giới khoa học bắt đầu nhận thấy WiFi và Bluetooth khÔng phù hợp cho những ứng dụng trong công nghiệp. Tuy nhiên chỉ đẾN năm 2004, bộ tiêu chuÂn Zigbee mới chính thức đƯỢc đƯỢc tạo dựng và thông qua bởi tổ chức Zigbee Alliance [14].

Tên gọi Zigbee lấy cảm hứng từ điendo nhảy theo đường zig-zag của ong mật. Điendo nhảy này đƯỢc loài ong sử dụng để trao đổi thông tin với nhau về vị trí của hoa và nguồn nước.

#### 2.4.3.6. KhẢ nĂng truyỀn tín hiệu cỦa Zigbee

ChuÂn mạng khÔng dÂy Zigbee có thể truyỀn đi xa từ 75 mét trở lên bắt đầu từ trạm phát, tín hiệu Zigbee còn có thể phát đi xa hơn đẾN các nút mạng khác trong hệ thống.

Toàn bộ dữ liệu đƯỢc truyỀn trong hệ thống mạng Zigbee đƯỢc truyỀn dưới dạng gói tin, quy định kích thước mỗi gói tối đa 128 bytes và chỉ cho phép tải xuống với tối đa 104 bytes.

ChuÂn Zigbee hỗ trợ cho cả 2 loại địa chỉ 32 bit và 64 bit. Để có thể xác định các thiết bị nào cùng có một địa chỉ IP duy nhất sẽ dựa trên loại địa chỉ 64 bit. Đối với địa chỉ ngắn đƯỢc sử dụng trong trường hợp khi mạng đƯỢc thiết lập, lúc này hệ thống cho phép hơn 65000 nút mạng đƯỢc liên kết với nhau.

#### Dải tần Zigbee:

Tín hiệu truyỀn trong giao thức Zigbee thực chất là tín hiệu radio. Zigbee đƯỢc hỗ trợ.

Dải 868,3 Mhz: Chỉ một kênh tín hiệu. Trong dải này tốc độ truyỀn là 20kb/s. Thường cho những khu vực Châu Âu và Nhật Bản.

Dải 902 Mhz – 928 Mhz: Có 10 kênh tín hiệu từ 1-10 với tốc độ truyỀn thường là 40kb/s. Thường cho những khu vực Bắc Mỹ.

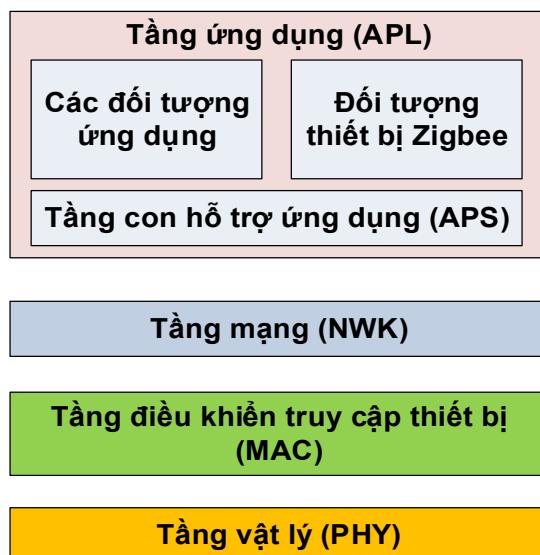
Dải 2,4 Ghz – 2,835 Ghz: Có 16 kênh tín hiệu từ 11-26 với tốc độ truyỀn 250kb/s. Thường cho những khu vực còn lại.

Trong những ứng dụng, người ta hay dùng giao thức Zigbee ở dải tần 2,4 Ghz – 2,835 Ghz. Đây là dải tần phổ biến và đƯỢc hỗ trợ bởi nhiều thiết bị. Hơn nữa với Zigbee, dải tần này có tới 16 kênh tín hiệu trong dải ( mỗi kênh cách nhau 5Mhz tần số ) với tốc độ truyỀn lớn nhất: 250kb/s.

#### 2.4.3.7. Cấu trúc của mạng Zigbee

Các nhóm nghiên cứu Zigbee và tổ chức IEEE đã làm việc cùng nhau để chỉ rõ toàn bộ các khía cạnh giao thức của công nghệ này. IEEE 802.15.4 tập trung nghiên cứu vào 2 tầng thấp nhất của giao thức đó là tầng vật lý và tầng liên kết dữ liệu \_MAC.

Ngoài 2 tầng này xác định bởi tiêu chuẩn 802.15.4 thì ở tiêu chuẩn Zigbee còn có thêm các tầng trên của hệ thống bao gồm: tầng mạng, tầng hỗ trợ ứng dụng, tầng đối tượng thiết bị và các đối tượng ứng dụng. Zigbee thiết lập cơ sở cho những tầng cao hơn trong giao thức (từ tầng mạng đến tầng ứng dụng) về bảo mật, dữ liệu, chuẩn phát triển để đảm bảo chắc chắn rằng các khách hàng dù mua sản phẩm từ các hãng khác nhau những vẫn theo một chuẩn riêng để làm việc cùng nhau được mà không tương tác lẫn nhau.



Hình 2. 8 Mô hình vật lý các lớp tầng của chuẩn Zigbee

#### 2.4.3.8. Kết nối Zigbee

Trong đề tài này, em chọn module Zigbee UART TTL CC2530 loại module đã được lập trình sẵn firmware để có thể dễ dàng sử dụng như một module truyền nhận dữ liệu không dây chuẩn Zigbee với giao tiếp UART rất dễ kết nối với vi điều khiển hoặc máy tính thông qua cáp chuyển USB-UART.

Module có thông số kỹ thuật như sau:

- Điện áp sử dụng: 3-5.5VDC
- Dòng tiêu thụ: <30mA
- Chuẩn truyền sóng Zigbee: 2,4GHz
- Tốc độ truyền sóng tối đa: 3300 bps
- Công suất truyền: 4,5dbm
- Khoảng cách truyền lên đèn: 1,2km
- Giao thức kết nối UART TTL
- Baurate tối đa: 115200

Cách sử dụng và cài đặt 2 module Zigbee truyền nhận cho nhau:

Bước 1: Không cấp nguồn, nhấn giữ phím key, sau đó cấp nguồn, 4 led trên module sẽ chớp sáng báo hiệu vào chế độ cấu hình, thả nút nhấn ra chúng ta sẽ vào chế độ cấu hình baurate cho kết nối UART đầu tiên. Nhấn phím key tuần tự để chọn baurate phù hợp theo bảng sau:

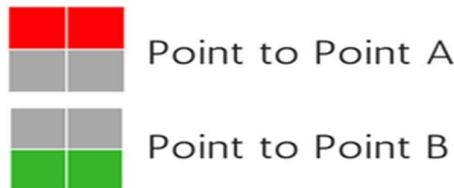
	2400		4800
	9600		14400
	19200		38400
	57600		115200

Hình 2. 9 Sơ đồ cấu hình baudrate cho module Zigbee

Bước 2: Sau khi chọn Baurate xong thì đè nút nhấn để chuyển sang chế độ chọn kênh. Lúc này đèn led sẽ nhấp nháy như ở bước 1 để báo hiệu chuyển sang bước 2. Nhấn nút lần lượt để chọn 1 kênh, tổng cộng sẽ có 16 kênh tương ứng với 16 trạng thái của 4 led.

Bước 3: Tiếp tục đè nút nhấn và đèn led sẽ tiếp tục nháy như 2 bước trên. Lúc này là bước chọn chế độ hoạt động. Có hai chế độ hoạt động chính là Point – to – Point và Broadcast.

Point – to – Point: thì chỉ có 2 module trong cùng 1 mạng và truyền nhận với nhau. Lần lượt 2 module sẽ cấu hình là:



Hình 2. 10 Sơ đồ cấu hình chế độ hoạt động cho module Zigbee

Broadcast: 1 mạng lưới gồm nhiều module Zigbee với nhau. Khi một module truyền thì tất cả các module còn lại sẽ nhận. Cấu hình tất cả các module giống như hình dưới.



Hình 2. 11 Sơ đồ cấu hình chế độ truyền cho module Zigbee

Bước 4: Tiếp tục đè nút nhấn, đèn sẽ nhấp nháy khi báo hiệu thành công.

## CHƯƠNG 3. THIẾT KẾ PHẦN CỨNG

Trong chương 3 này sẽ trình bày về hướng thiết kế phần cứng cho mạch điều khiển trung tâm Master phục vụ cho việc giao tiếp dữ liệu với các mảng mạch điều khiển điện tử công suất và hệ thống IoT cho việc điều khiển giám sát và thu thập dữ liệu.

### 3.1 Thiết kế và chế tạo các mảng mạch đo lường và mạch Master

Để thực tế hóa hệ thống điều khiển giám sát cho hệ PV Inverter thì ở đây em xin phân thiết kế mạch đo lường cho bộ biến đổi Slave của hệ thống PV Inverter để tiện cho việc trình bày đồ án.

Thiết kế và chế tạo các mảng mạch Master cần thiết kế các mảng mạch sau đây:

- Thiết kế chế tạo mạch đo lường cho các bộ biến đổi.
- Thiết kế chế tạo mảng mạch điều khiển cho phần Master.

Do phần thiết kế các mảng mạch đo lường cho các bộ biến đổi là giống nhau về nội dung thiết kế nên ở đây em xin trình bày phần mạch đo lường cho bộ biến đổi DC/AC.

#### 3.1.1 Thiết kế và chế tạo mạch đo lường

##### 3.1.1.1 Cơ sở thiết kế

Mạch đo lường cho bộ biến đổi DC/AC có nhiệm vụ đo các tín hiệu điện áp tải, điện áp lưới và dòng tải, dòng lưới rồi truyền về mạch điều khiển DC/AC sau đó giao tiếp với mạch điều khiển Master để xác định các đại lượng trên đã phù hợp chưa và đưa ra các thông số điều chỉnh.

Các tín hiệu dòng điện, điện áp thực tế qua mạch đo chuyển đổi thành tín hiệu 0-3V để vi điều khiển có thể đọc được qua module ADC. Mạch đo sử dụng IC HCPL7800A để điện áp, LEM55-P/SPI để đo dòng điện.

Mạch cần đo các tín hiệu sau:

- $I_{a\_L}$ ,  $I_{b\_L}$ ,  $I_{c\_L}$ : Dòng điện phía tải.
- $U_{a\_L}$ ,  $U_{b\_L}$ ,  $U_{c\_L}$ : Điện áp phía tải.
- $I_{a\_g}$ ,  $I_{b\_g}$ ,  $I_{c\_g}$ : Dòng điện phía lưới.
- $U_{a\_g}$ ,  $U_{b\_g}$ ,  $U_{c\_g}$ : Điện áp phía lưới.

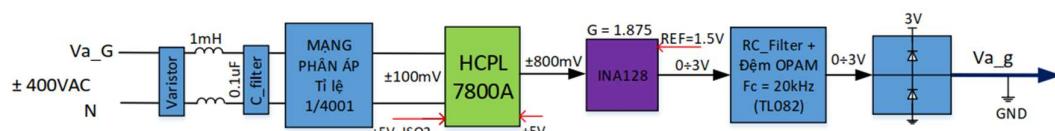
Bảng 3. 1 Bảng thông số giá trị đo của mạch

Tín hiệu	Đại lượng	Hệ số khuếch đại	Giá trị Offset	Dải đo
$U_{a_g}$	Điện áp lưới pha A	400/1.5	1.5 V	$\pm 400$ V
$U_{b_g}$	Điện áp lưới pha B	400/1.5	1.5 V	$\pm 400$ V
$U_{c_g}$	Điện áp lưới pha C	400/1.5	1.5 V	$\pm 400$ V
$U_{a_L}$	Điện áp tải pha A	400/1.5	1.5 V	$\pm 400$ V
$U_{b_L}$	Điện áp tải pha B	400/1.5	1.5 V	$\pm 400$ V
$U_{c_L}$	Điện áp tải pha C	400/1.5	1.5 V	$\pm 400$ V
$I_{a_g}$	Dòng điện lưới pha A	50/1.5	1.5 V	$\pm 50$ A
$I_{b_g}$	Dòng điện lưới pha B	50/1.5	1.5 V	$\pm 50$ A
$I_{c_g}$	Dòng điện lưới pha C	50/1.5	1.5 V	$\pm 50$ A
$I_{a_L}$	Dòng điện tải pha A	50/1.5	1.5 V	$\pm 50$ A
$I_{b_L}$	Dòng điện tải pha B	50/1.5	1.5 V	$\pm 50$ A
$I_{c_L}$	Dòng điện tải pha C	50/1.5	1.5 V	$\pm 50$ A
$U_{C1}$	Điện áp một chiều	500/3.0	0 V	0 – 500V
$U_{C2}$	Điện áp một chiều	500/3.0	0 V	0 – 500V

### 3.1.1.2. Thiết kế và chế tạo

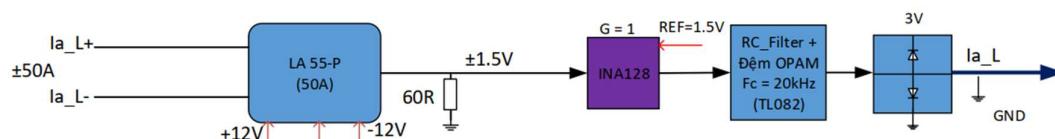
Thiết kế nguyên lý các phần đo điện áp và dòng điện như sau:

Mạch đo điện áp sử dụng IC cách ly HCPL 7800A với đầu vào là điện áp được phân áp qua cầu điện trở. Điện áp đầu ra được đưa qua INA128 để chuyển đổi điện áp cho phù hợp với vi điều khiển.



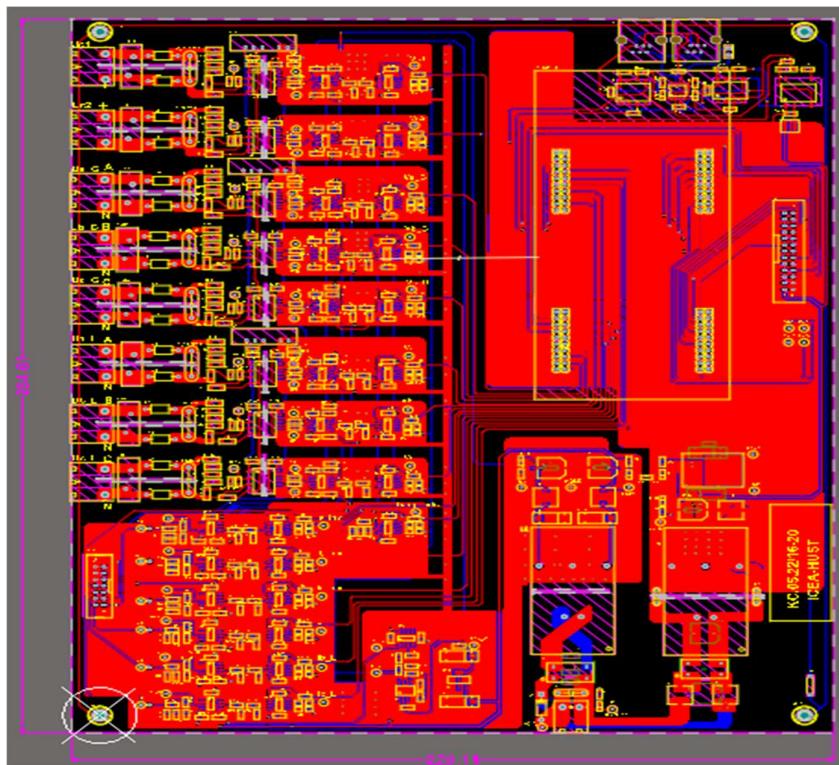
Hình 3. 1 Sơ đồ nguyên lý phần đo điện áp

Mạch đo dòng điện sử dụng LEM LA55-P/SPI với đầu vào là dòng điện, đầu ra là điện áp. Sau đó, điện áp đầu ra được đưa qua INA128 để chuyển đổi mức điện áp trong dải 0-3V để vi điều khiển có thể đọc được.

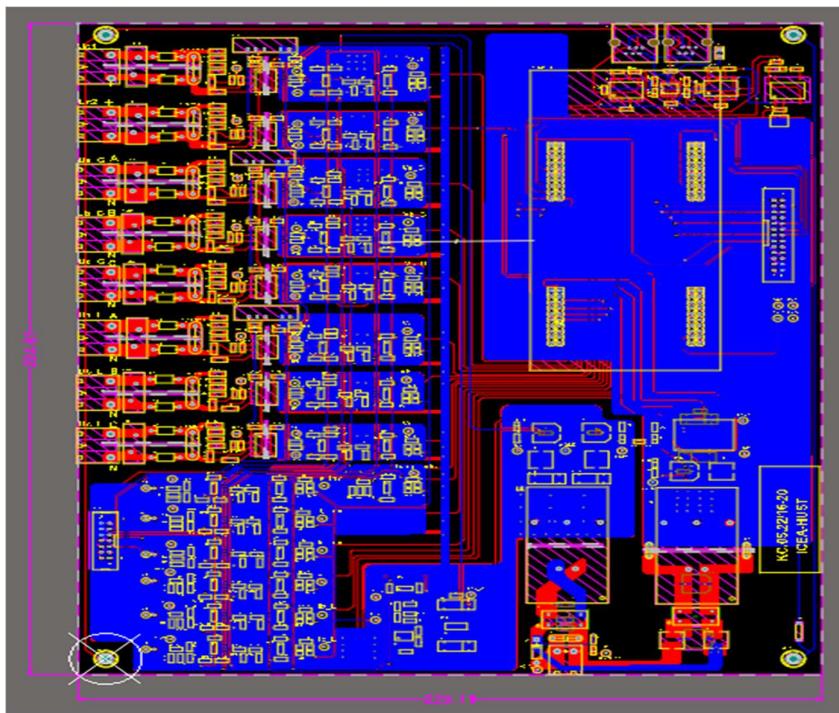


Hình 3. 2 Sơ đồ nguyên lý phần đo dòng điện

Từ những yêu cầu trên chúng ta đi vào thiết kế layout trên phần mềm Altium Designer. Và dưới đây đây là hình ảnh về layout mảng mạch đo lường.



Hình 3. 3 Layout lớp trên của mạch đo tích hợp mạch điều khiển



Hình 3. 4 Layout lớp dưới của mạch đo tích hợp mạch điều khiển

### 3.1.2 Thiết kế và chế tạo mảng mạch điều khiển cho phần Master

#### 3.1.2.3 Cơ sở thiết kế

Mạch điều khiển Master có chức năng điều khiển ba bộ điều khiển Slave và giao tiếp với Raspberry Pi. Dữ liệu đo đặc của hệ thống sẽ được truyền thông lên GLCD và lên Web thông qua Raspberry.

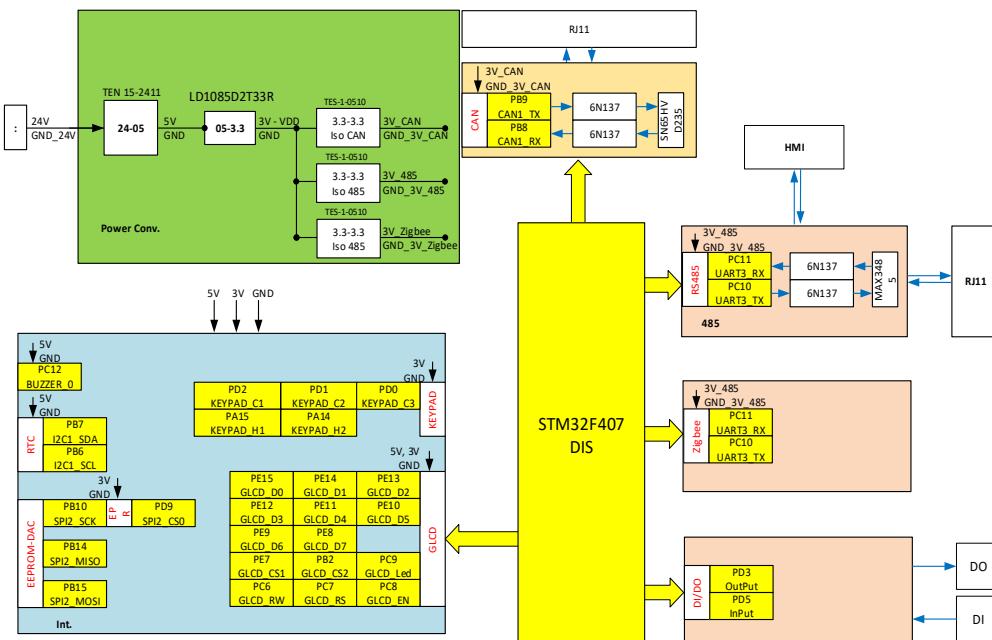
Mạch điều khiển trung tâm Master được kết nối với màn hình HMI, GLCD, mạch Slave và Raspberry Pi thông qua truyền thông Modbus, RS485, Ethernet, CAN và Zigbee.

Giao diện Setting các thông số hệ thống trên màn hình GLCD cũng như trên giao diện Web giúp người dùng vận hành giám sát, vận hành hệ thống thông qua mạch điều khiển trung tâm Master. Lớp giao diện có chức năng như sau:

- Hiển thị trạng thái hoạt động, các giá trị dòng điện cũng như điện áp của lưới và tải.
- Cài đặt các tham số hoạt động của hệ thống.
- Giao diện vận hành hệ thống.

Qua đó có thể thấy được tầm quan trọng của mạch điều khiển trung tâm Master là khâu điều khiển toàn bộ hệ thống thông qua giao diện vận hành trên màn hình GLCD và Web.

Dựa vào yêu cầu đó, cấu trúc tổng quát của mạch điều khiển trung tâm được đưa ra như sau:



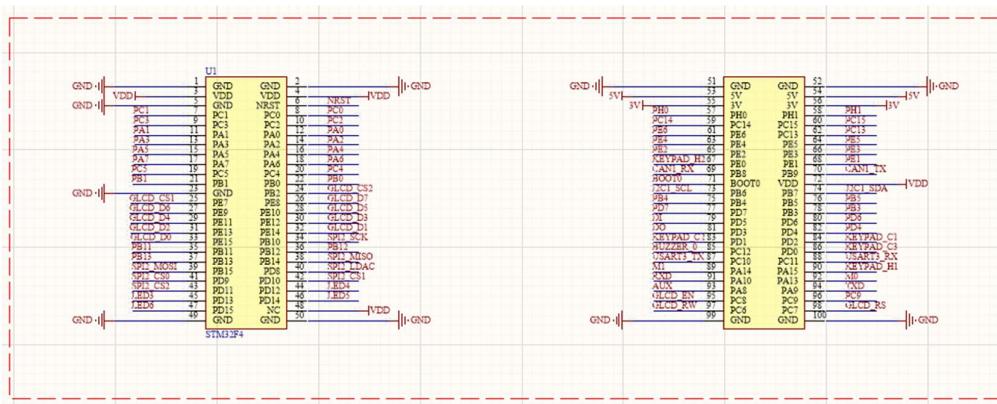
Hình 3. 5 Sơ đồ cấu trúc tổng quát của mạch điều khiển trung tâm

Mạch bao gồm các ngoại vi như sau:

- Vi điều khiển STM32F407
  - Giao tiếp truyền thông CAN
  - Giao tiếp truyền thông SPI với EEPROM
  - Giao tiếp modbus với màn hình HMI
  - Giao tiếp ADC để đọc tín hiệu từ mạch đo

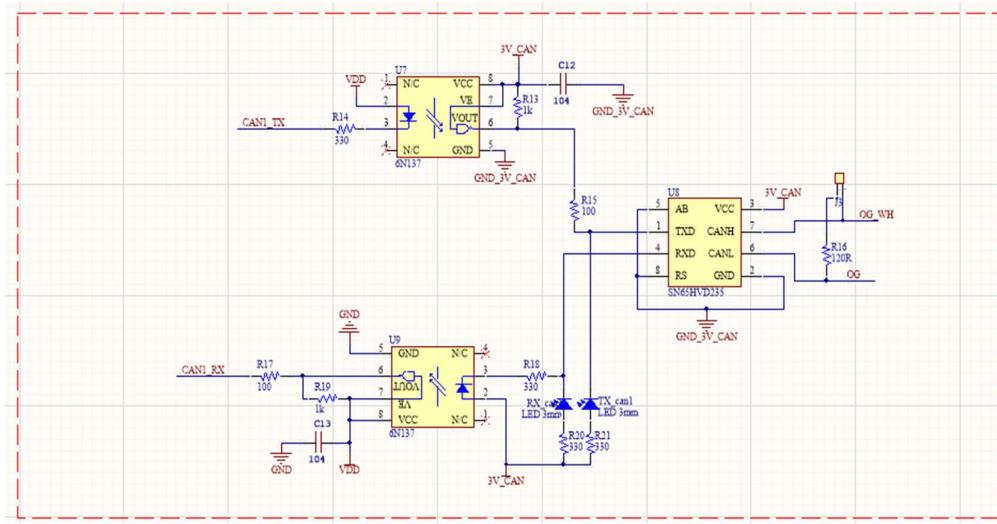
#### 3.1.2.4. Thiết kế và chế tạo

Khối vi điều khiển MCU sử dụng vi điều khiển STM32F407XX với tốc độ xử lý cao lên đến 168MHz, với hỗ trợ đầy đủ các giao thức truyền thông cũng như ngoại vi mở rộng.



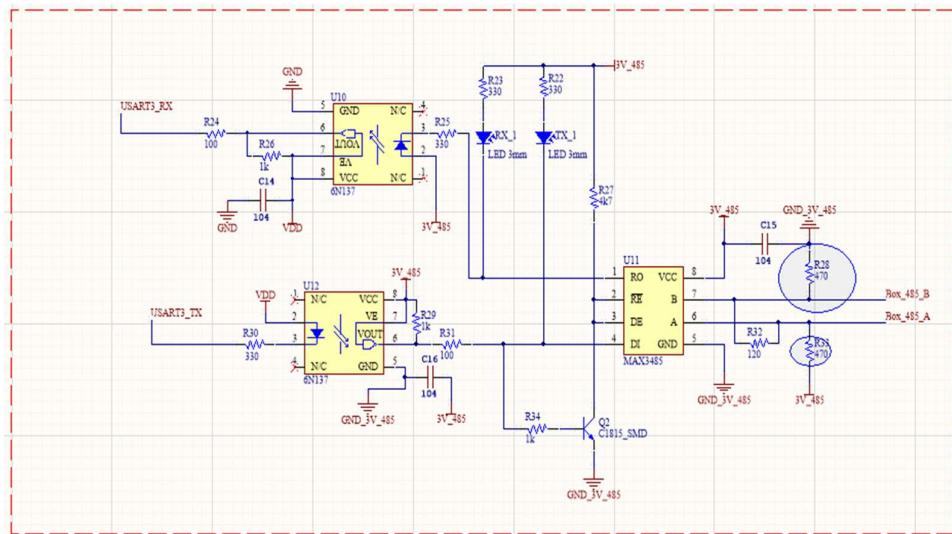
Hình 3. 6 Sơ đồ kết nối khối MCU

Khối truyền thông CAN với các mạch Slave.

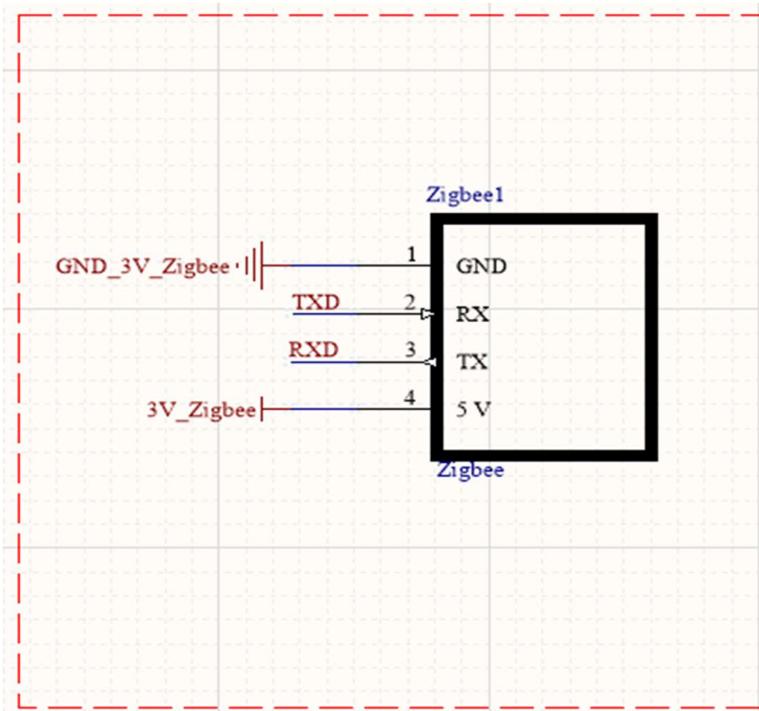


Hình 3. 7 Sơ đồ khái truyền thông CAN với các mạch Slave

Và các khối truyền thông Modbus và Zigbee.

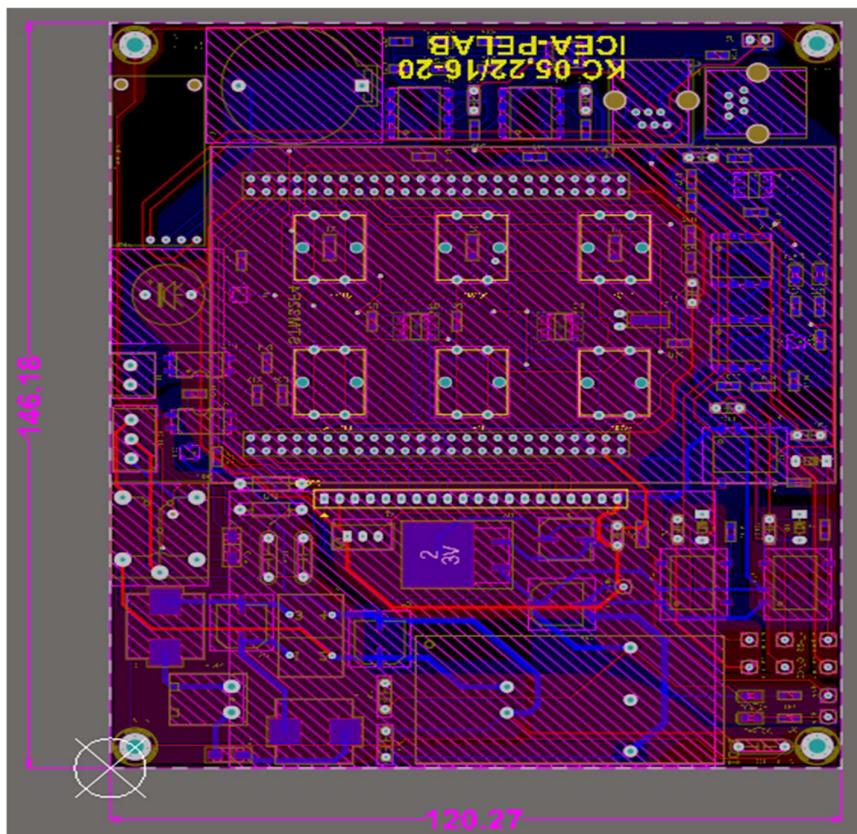


Hình 3. 8 Sơ đồ khối truyền thông Modbus

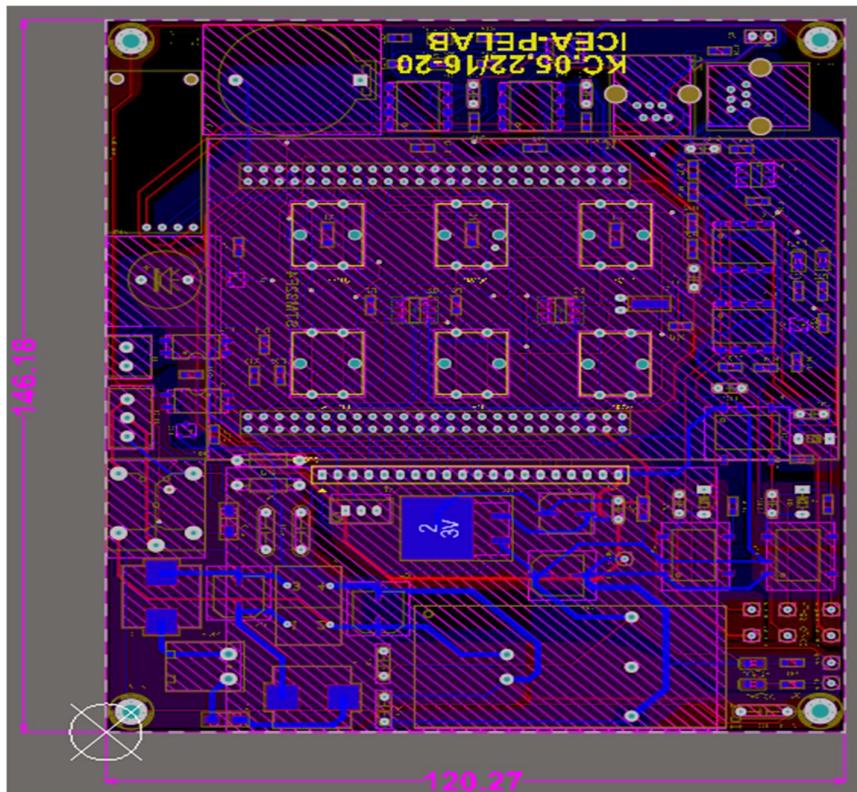


Hình 3. 9 Sơ đồ khối truyền thông Zigbee

Thiết kế layout trên phần mềm Altium Designer



Hình 3. 10 Layout lop trên của mạch điều khiển trung tâm



Hình 3. 11 Layout lop dưới của mạch điều khiển trung tâm

## **CHƯƠNG 4. THIẾT KẾ HỆ THỐNG ĐIỀU KHIỂN GIÁM SÁT VÀ THU THẬP DỮ LIỆU CHO HỆ PV INVERTER**

Trong chương 4 này sẽ trình bày về IOT, ứng dụng của IOT trong hệ thống điện tử công suất và nhiệm vụ, hướng thiết kế hệ thống giám sát và thu thập dữ liệu cho hệ PV Inverter.

### **4.1 Internet of Things**

#### **4.1.1 Giới thiệu**

Internet of things theo định nghĩa của Wiki là mạng lưới vạn vật kết nối internet hoặc là mạng lưới thiết bị kết nối internet, khi mà mỗi đồ vật, con người được cung cấp một định danh của riêng mình, và tất cả có khả năng truyền tải, trao đổi thông tin, dữ liệu qua một mạng duy nhất mà không cần đến sự tương tác trực tiếp giữa con người với con người, hay người với máy tính. IoT đã phát triển từ sự hội tụ công nghệ không dây, công nghệ vi cơ điện tử và internet. Hay nói đơn giản là một tập hợp các thiết bị có khả năng kết nối với nhau, với internet với thế giới bên ngoài để thực hiện một công việc nào đó.

Hay hiểu một cách đơn giản IoT là tất cả thiết bị có thể kết nối với nhau. Việc kết nối thì có thể thực hiện qua Wifi, mạng viễn thông băng rộng (3G, 4G), Bluetooth, Zigbee, hồng ngoại,...

#### **4.1.2 Ứng dụng của IoT trong hệ thống điện tử công suất**

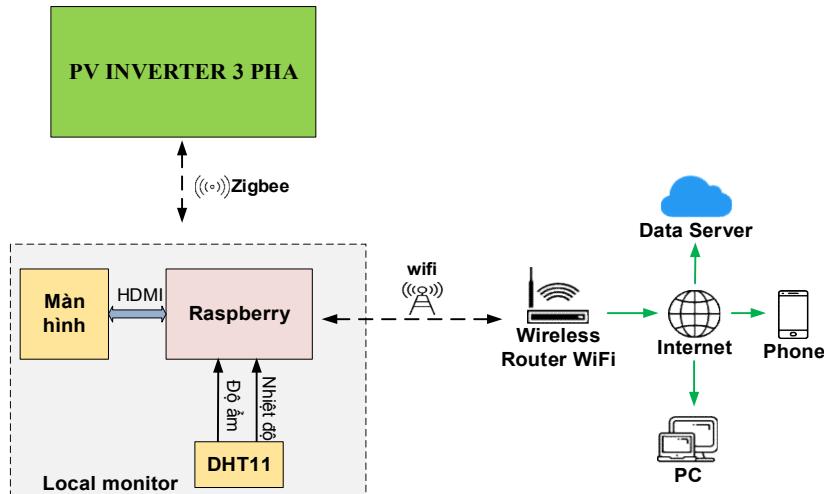
IoT được ứng dụng khá nhiều trong các lĩnh vực nổi bật hiện nay như:

- Nhà thông minh.
- Quản lý các thiết bị cá nhân.
- Quản lý môi trường.
- Quản lý giao thông.

Nhận thấy IoT không chỉ ứng dụng vào những lĩnh vực trên mà nó còn được ứng dụng rất nhiều trong công nghiệp tự động hóa. Các khu công nghiệp sẽ được quản lý thông qua internet. Người quản lý sẽ không cần phải đến nhà máy cũng biết được hệ thống đang hoạt động như thế nào để đưa ra các yêu cầu tới hệ thống. Ngoài ra, việc ứng dụng IoT vào trong công nghiệp sẽ giúp chất lượng nhà máy cao hơn, tăng năng suất lao động,...

Và cụ thể ở đây, em muốn trình bày về ứng dụng IoT trong hệ thống điện tử công suất. Việc áp dụng IoT vào sẽ giúp các kỹ sư không cần phải thao tác điều khiển hệ thống tại hiện trường mà có thể ngồi ở khu vực kỹ thuật giám sát và đưa ra các thông số điều khiển tới các hệ thống điện tử công suất.

## 4.2 Nhiệm vụ của hệ thống điều khiển giám sát và thu thập dữ liệu cho hệ PV Inverter



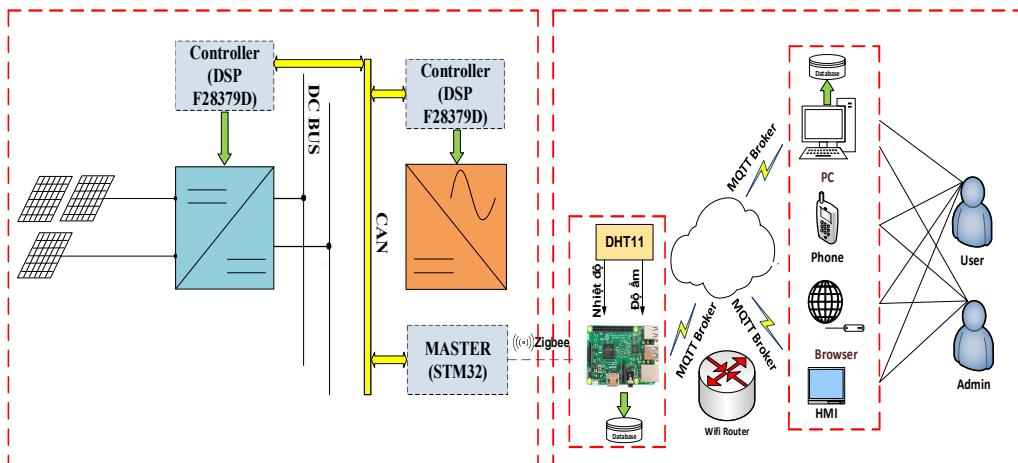
Hình 4. 1 Sơ đồ hệ thống IoT cho hệ PV Inverter

Nhu đã trình bày ở chương 2 thì hệ thống không chỉ sử dụng các tín hiệu từ các mạch đo lường gửi về mà còn sử dụng thông tin được lấy từ nhiều nơi khác trong hệ thống. Do vậy, cần xây dựng một hệ thống truyền thông để trao đổi dữ liệu giữa các thành phần trong hệ thống PV Inverter.

Nhiệm vụ của hệ thống điều khiển giám sát và thu thập dữ liệu đó là:

- Thu thập, quản lý dữ liệu phục vụ cho việc điều khiển.
- Theo dõi, bám sát các hoạt động của hệ thống.
- Điều khiển hệ thống theo các cấp độ.

## 4.3 Thiết kế hệ thống điều khiển giám sát và thu thập dữ liệu cho hệ PV Inverter



Hình 4. 2 Sơ đồ chi tiết hoạt động của hệ thống IoT

Về ý tưởng thiết kế hệ thống điều khiển giám sát này đó là dùng Raspberry Pi 3 là vật xúc tác chính giữa việc nhận thông tin dữ liệu từ mạch Master và truyền lên database và hiển thị lên web.

Từ sơ đồ thiết kế tổng quan cấu trúc hệ thống điều khiển giám sát và thu thập dữ liệu\_IOT cho hệ PV Inverter ta có thể liệt kê các bước thiết kế hệ thống như sau:

- Giao thức giữa STM và Raspberry Pi
- Tương tác giữa Raspberry Pi và Database
- Webserver

#### **4.3.1 Giao thức giữa mạch điều khiển Master và Raspberry**

Giao tiếp giữa vi xử lý trung tâm mạch điều khiển Master\_STM32 và Raspberry là thông qua truyền thông không dây Zigbee.

Quy định chung nếu 1 số có 2 byte thì byte trước sẽ là byte cao, byte sau là byte thấp.

##### *4.3.1.1. Khung truyền từ STM32 lên Raspberry Pi*

Như đã đề cập ở chương 2 phần truyền thông trong hệ thống thì việc truyền nhận dữ liệu giữa mạch điều khiển trung tâm Master với màn hình GLCD là thông qua chuẩn truyền thông Modbus RTU. Vì vậy, em cũng sử dụng luôn khung truyền của Modbus RTU cho việc giao tiếp giữa STM32 và Raspberry thông qua Zigbee.

Mỗi khung bản tin bao gồm nhiều ký tự, các ký tự được truyền liên tục thành dòng.

Cấu trúc bản tin:

*Bảng 4. 1 Cấu trúc bản tin truyền từ STM32 lên Raspberry Pi*

Khởi đầu	Địa chỉ	Mã hàm	Dữ liệu	Mã CRC	Kết thúc
(---)	8 bit	8 bit	n x 8 bit	16 bit	(---)

*Bảng cấu trúc kí tự gửi đi từ STM32 đến Raspberry Pi*

*Bảng 4. 2 Bảng cấu trúc kí tự gửi đi từ STM32 lên Raspberry Pi*

Start	0	1	2	3	4	5	6	7	P	Stop
-------	---	---	---	---	---	---	---	---	---	------

Bao gồm:

- 1 bit Start.
- 8 bit dữ liệu.
- 1 bit Parity chẵn/lẻ (nếu có).
- 1 bit Stop hoặc 2 bit Stop nếu không sử dụng bit Parity.

#### 4.3.1.2. Khung truyền từ Raspberry xuống STM32

Bảng 4. 3 Bảng cấu trúc bản tin truyền về STM32 từ Raspberry Pi

Khởi đầu	Địa chỉ	Mã hàm	Dữ liệu	Mã CRC	Kết thúc
(---)	8 bit	8 bit	n x 8 bit	16 bit	(---)

Về lý thuyết thì khung truyền về từ Raspberry xuống STM32 sẽ tương tự như khung truyền từ STM32 lên Raspberry nhưng phần data sẽ là dữ liệu điều khiển gửi về hệ thống.

#### 4.3.2 Giao thức MQTT

##### 4.3.2.3. Giới thiệu MQTT

**MQTT** (Message Queuing Telemetry Transport) là một giao thức gói dạng publish/subscribe sử dụng cho các thiết bị Internet of Things với băng thông thấp, độ tin cậy cao và khả năng được sử dụng trong mạng lưới không ổn định.

MQTT được tạo ra từ năm 1999 bởi hai kỹ sư Andy Stanford-Clark và Arlen Nipper (Eurotech) khi họ phải phát minh ra một giao thức mới để kết nối các đường ống dẫn dầu trên các mạng vệ tinh không đáng tin cậy.

Năm 2011, IBM và Eurotech đã tặng MQTT cho dự án Eclipse được đề xuất có tên là Paho. Trong năm 2013, nó đã được đệ trình lên OASIS để chuẩn hóa.

Bởi vì giao thức này sử dụng băng thông thấp trong môi trường có độ trễ cao nên nó là một giao thức lý tưởng cho các ứng dụng M2M (Machine to Machine).

Hai thành phần publisher và subscriber là đặc trưng tạo nên giao thức MQTT. Các MQTT Client không kết nối trực tiếp với nhau, mọi gói dữ liệu được gửi đi đều thông qua MQTT Broker. Để có thể triển khai các ứng dụng của MQTT Client, chúng ta cần MQTT Broker và Broker ở đây sẽ là Raspberry Pi. Raspberry Pi sẽ thu thập dữ liệu nhận được từ STM32 và giao tiếp với các Client.

Một số khái niệm đi kèm khi sử dụng giao thức MQTT như sau:

**Broker:** là server cài MQTT server thu thập dữ liệu và giao tiếp các Client.

**Topic:** Về mặt kỹ thuật thì topic là các hàng đợi chứa message. Về logic, topic cho phép clients trao đổi thông tin và dữ liệu.

**Publish:** là gói tin thiết bị gửi lên Broker. Khi thiết bị publish dữ liệu vào topic, các Client sẽ nhận được dữ liệu khi đăng ký topic. Gói Publish có thể là gói gửi dữ liệu vào topic hoặc gói cài đặt Retain message hay LWT Message do cờ Retain, LWT quy định.

**Subscribe:** là gói đăng ký nhận data hay thông tin từ topic đăng ký (Broker sẽ tạo topic mới nếu không tìm được topic đăng ký). Khi Client gửi gói tin Subscribe tới Broker để đăng ký một topic, ví dụ “PV Inverter”. Bất cứ khi nào thiết bị gửi gói tin Publish vào topic “PV Inverter” thì dữ liệu trong gói Publish sẽ chuyển về

Client đó. Giống như việc bạn Subscribe kênh Youtube. Khi có thông báo mới thì bạn sẽ nhận được thông tin đó chính là Publish.

**Message:** Là các đơn vị dữ liệu được trao đổi giữa các topic clients.

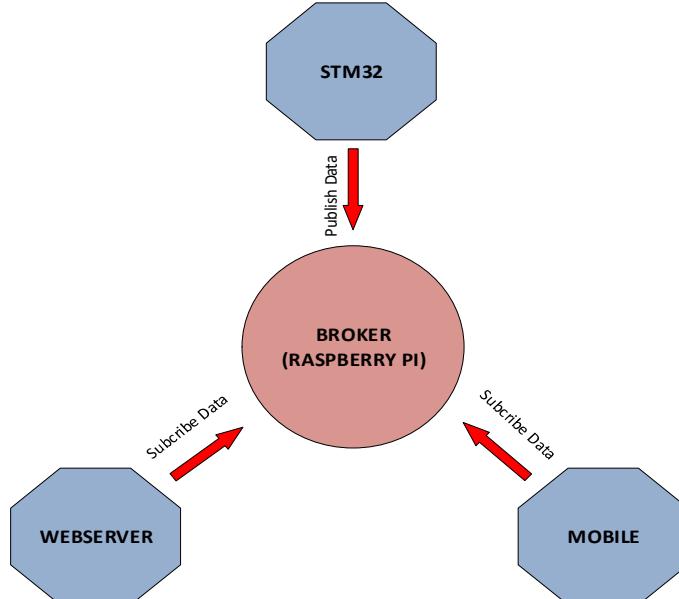
**QoS:** chất lượng đường truyền.

- QoS 0: thiết bị gửi Publish tới Server không cần quan tâm đến gói tin gửi.
- QoS 1: thiết bị gửi Publish tới Server. Sau đó, nếu Server nhận được gói tin và gửi lại thiết bị gói PUBACK để xác nhận đã nhận được gói Publish từ thiết bị.
- QoS 2: thiết bị gửi Publish tới Server. Server nhận gói Publish và gửi PUBREC lại thiết bị kèm theo ID đã nhận. Thiết bị nhận PUBREC gửi PUBREL kèm ID đó lại Server. Server gửi lại PUBCOMP lại thiết bị.

**Retain:** là cờ báo gói Publish, thiết bị gửi tới Broker cài đặt Retain Message cho topic. Khi Client subscribe vào topic đã cài đặt Retain Message sẽ nhận được ngay Retain Message. Ví dụ “Device online”.

**LWT:** là cờ báo gói Publish, thiết bị gửi tới Broker topic cài đặt LWT Message. Khi Client subscribe vào topic đã cài đặt LWT Message sẽ nhận được tin nhắn LWT Message khi Device offline.

Ta có sơ đồ cấu trúc minh họa cho giao thức này như sau:



Hình 4. 3 Sơ đồ cấu trúc giao thức MQTT

Nhìn vào sơ đồ cấu trúc ta có thể thấy nguyên lý hoạt động của hệ thống theo giao thức MQTT Broker như sau:

Raspberry Pi ở đây sẽ đóng vai trò là Broker. Và các Client sẽ là STM32, Webserver và Mobile. STM32 sẽ publish data lên Raspberry Pi\_Broker và các

Client còn lại là Webserver và Mobile sẽ subscribe đến Raspberry để nhận data và hiển thị lên interface.

#### 4.3.2.4. Khung truyền của MQTT

Như đã đề cập ở trên thì MQTT là một giao thức truyền message theo mô hình publish/subscribe.

Về định dạng khung truyền của MQTT thì tất cả các message luôn chưa phần cố định như bảng dưới đây.

Bảng 4. 4 Khung truyền bản tin của MQTT

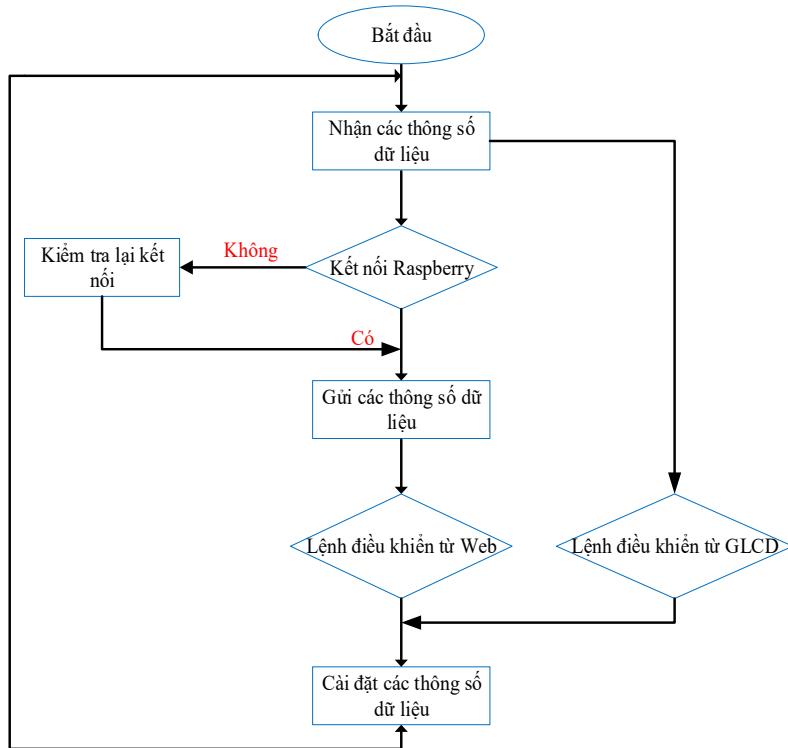
Bit	7	6	5	4	3	2	1	0
Byte 1	Loại Message		Cờ DUP		QoS level		RETAIN	
Byte 2	Độ dài còn lại							

Nhưng do giới hạn về khả năng và thời gian có hạn nên em chỉ tập trung vào loại message và độ dài byte của nó.

Message sẽ được định nghĩa đơn giản như sau đó là: Khi 1 client gửi 1 message publish đến server thì server sẽ hiểu rằng nên giữ message này lại kể cả sau khi chuyển nó đến các subscribers khác. Tức là khi STM32F407 gửi thông số dữ liệu cho Raspberry thì bản thân Raspberry sẽ lưu dữ liệu đó lại server và truyền đến các client khác là web hoặc mobile.

#### 4.3.3 Thiết kế phần mềm

##### 4.3.3.5. Lưu đồ thuật toán trên mạch điều khiển Master

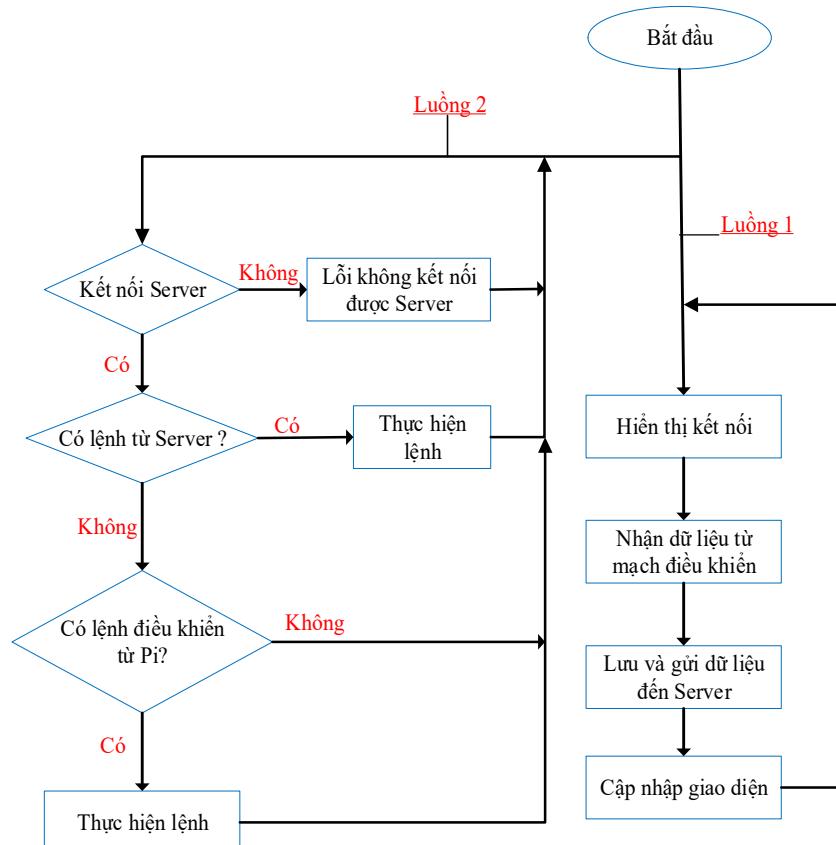


Hình 4. 4 Lưu đồ thuật toán trên mạch điều khiển trung tâm

Ngoài việc thực hiện giao tiếp với các mạch Slave để nhận giá trị đo lường ra thì mạch điều khiển trung tâm Master còn thực hiện chức năng đó là nhận các thông số dữ liệu và truyền lên Raspberry và GLCD như ở lưu đồ thuật toán ở trên.

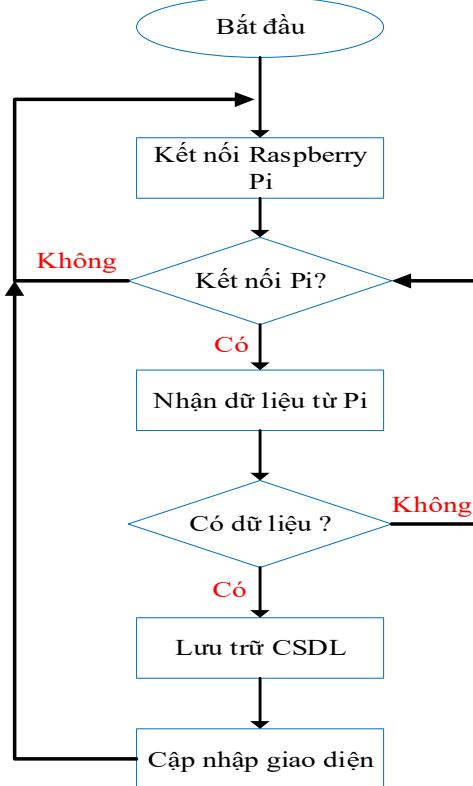
#### 4.3.3.6. Lưu đồ thuật toán trên Raspberry Pi

Raspberry Pi là một board mạch hay còn gọi là máy tính nhúng được dùng như vật xúc tác chính trong việc nhận và truyền dữ liệu với STM32 và với Webserver. Dưới đây là lưu đồ thuật toán điều khiển trên Raspberry Pi.



Hình 4. 5 Lưu đồ thuật toán trên Raspberry Pi

#### 4.3.3.7. Lưu đồ thuật toán trên Server



Hình 4. 6 Lưu đồ thuật toán trên Server

Sau khi đã thiết kế xong lưu đồ thuật toán thì chúng ta đi vào thiết kế giao diện điều khiển giám sát hệ thống.

#### 4.3.3.8. Web Server

**Web server** là một máy chủ web mà khi có bất kỳ một Web Client nào (chẳng hạn Web Browser) truy cập vào, thì nó sẽ căn cứ trên các thông tin yêu cầu truy cập để xử lý, và phản hồi lại nội dung. Đa phần các nội dung Web Server phục vụ là PHP, HMTL, Javascripts, CSS, JSON và bao gồm cả các dữ liệu Binary.

**PHP (Personal Home Page)** là ngôn ngữ lập trình kịch bản mã nguồn mở được dùng phổ biến để tạo ra các ứng dụng web chạy trên máy chủ, ví dụ như: chỉ định các đoạn văn bản, tiêu đề, bảng dữ liệu, hoặc nhúng hình ảnh hoặc video vào Web. Mỗi trang web chứa một loạt các liên kết đến các trang khác được gọi là hyperlinks, mỗi trang được tạo ra từ nhiều tag khác nhau. Mã lệnh PHP có thể được nhúng vào trong trang HMTL nhờ sử dụng cặp thẻ PHP.

**Javascript** là một ngôn ngữ được thiết kế chủ yếu để thêm tương tác vào các trang Web, và tạo ra các ứng dụng Web. Các chương trình Javascript có thể được nhúng trực tiếp vào HTML của Web. Và tùy thuộc vào mục đích cụ thể, script có thể chạy khi mở trang Web, nhấp chuột, gõ phím, gửi biểu mẫu, cập nhật dữ liệu, giao tiếp với cơ sở dữ liệu... Để nhúng chương trình viết bằng Javascript vào trang HTML, chỉ cần thêm tag <script> và thuộc tính type.

CSS là từ viết tắt của Cascading Style Sheets là một ngôn ngữ được thiết kế để xử lý giao diện Web, giúp các trang Web được đẹp hơn. CSS có thể kiểm soát được màu sắc văn bản, font chữ, kích cỡ chữ...

Như ta đã biết, website là một tập hợp các trang web bao gồm văn bản, hình ảnh,...thường chỉ nằm trong một tên miền (domain name) hoặc tên miền phụ (subdomain). Trang web được lưu trữ (web hosting) trên máy chủ web (web server) có thể truy cập thông qua Internet.

Và do chưa đăng ký tên miền cũng như có hosting nên việc tạo server em dùng **XAMPP** là chương trình tạo server web được tích hợp sẵn Apache, PHP, MySQL,...XAMPP là một mã nguồn mở web server đa nền giúp người dùng có thể dễ dàng tạo ra máy chủ web local để kiểm tra và triển khai trang web của mình. Vì hầu hết việc triển khai web server thực tế cũng đều sử dụng cùng thành phần như XAMPP nên rất dễ dàng để chuyển từ máy chủ local sang máy chủ online.

Số	id	tin_hieu	dai_luong	he_so_khuech	dai	gia_tri_off_set	gia_tri	don_vi	created_at	updated_at		
1	174	Chép	Xóa bđ	174	UA	Giá trị điện áp tắt pha A	400/1.5	1.5	123	V	2019-12-30 18:12:37	2019-12-30 18:12:37
2	175	Chép	Xóa bđ	175	UB	Giá trị điện áp tắt pha A	400/1.5	1.5	124	V	2019-12-30 18:12:37	2019-12-30 18:12:38
3	176	Chép	Xóa bđ	176	UC	Giá trị điện áp tắt pha A	400/1.5	1.5	125	V	2019-12-30 18:12:37	2019-12-30 18:12:38
4	177	Chép	Xóa bđ	177	IA	Giá trị điện áp tắt pha A	400/1.5	1.5	10	V	2019-12-30 18:12:37	2019-12-30 18:12:38
5	178	Chép	Xóa bđ	178	IB	Giá trị điện áp tắt pha A	400/1.5	1.5	11	V	2019-12-30 18:12:37	2019-12-30 18:12:38
6	179	Chép	Xóa bđ	179	IC	Giá trị điện áp tắt pha A	400/1.5	1.5	12	V	2019-12-30 18:12:37	2019-12-30 18:12:38
7	180	Chép	Xóa bđ	180	UA	Giá trị điện áp tắt pha A	400/1.5	1.5	123	V	2019-12-30 18:13:05	2019-12-30 18:13:09
8	181	Chép	Xóa bđ	181	UB	Giá trị điện áp tắt pha A	400/1.5	1.5	124	V	2019-12-30 18:13:05	2019-12-30 18:13:09
9	182	Chép	Xóa bđ	182	UC	Giá trị điện áp tắt pha A	400/1.5	1.5	125	V	2019-12-30 18:13:05	2019-12-30 18:13:09
10	183	Chép	Xóa bđ	183	IA	Giá trị điện áp tắt pha A	400/1.5	1.5	10	V	2019-12-30 18:13:05	2019-12-30 18:13:09
11	184	Chép	Xóa bđ	184	IB	Giá trị điện áp tắt pha A	400/1.5	1.5	11	V	2019-12-30 18:13:05	2019-12-30 18:13:09
12	185	Chép	Xóa bđ	185	IC	Giá trị điện áp tắt pha A	400/1.5	1.5	12	V	2019-12-30 18:13:05	2019-12-30 18:13:09
13	186	Chép	Xóa bđ	186	UR	Giá trị điện áp tắt pha A	400/1.5	1.5	123	V	2019-12-30 18:15:24	2019-12-30 18:15:24
14	187	Chép	Xóa bđ	187	UR	Giá trị điện áp tắt pha A	400/1.5	1.5	124	V	2019-12-30 18:15:24	2019-12-30 18:15:24

Hình 4. 7 Cơ sở dữ liệu được tạo trên phpmyadmin

Đây là cơ sở dữ liệu dùng MySQL phục vụ cho việc lưu trữ các thông số dữ liệu nhận được từ việc truyền nhận dữ liệu với hệ thống PV Inverter.

Các bước để thiết kế website như sau:

Bước 1: Lên ý tưởng về nội dung, hình ảnh và thông tin mình muốn đưa lên website. Đó là cơ sở để thiết kế được trang web đầy đủ nội dung cần biểu đạt.

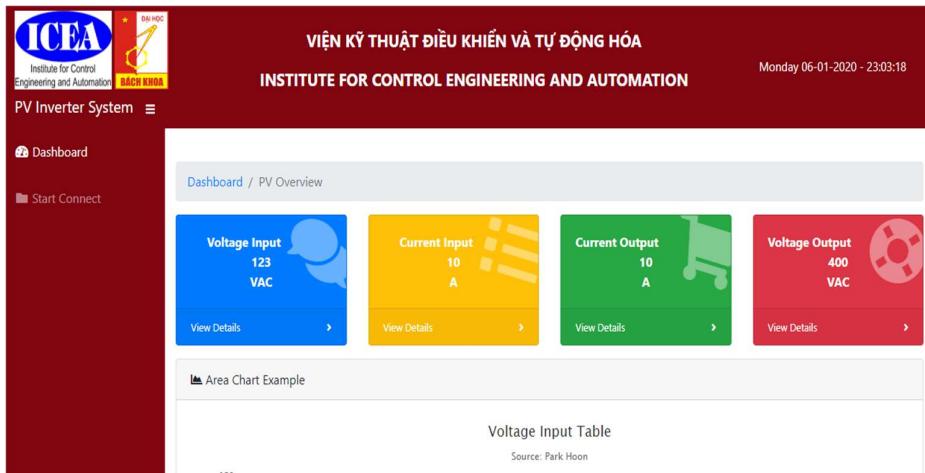
Bước 2: Đó chính là đăng ký domain và webhosting. Nhưng vì lý do đã được trình bày ở trên nên bước này chính là bước tạo server bằng XAMPP tại localhost.

Bước 3: Thống nhất ý tưởng thiết kế với team thiết kế hệ thống PV Inverter về nội dung, tính năng, yêu cầu kỹ thuật về website giám sát điều khiển.

Bước 4: Thiết kế website, bước này sẽ bao gồm thiết kế về cấu trúc, giao diện, tính năng.

#### Bước 5: Nghiêm thu.

Dưới đây là một số hình ảnh về giao diện sau khi đã thiết kế.



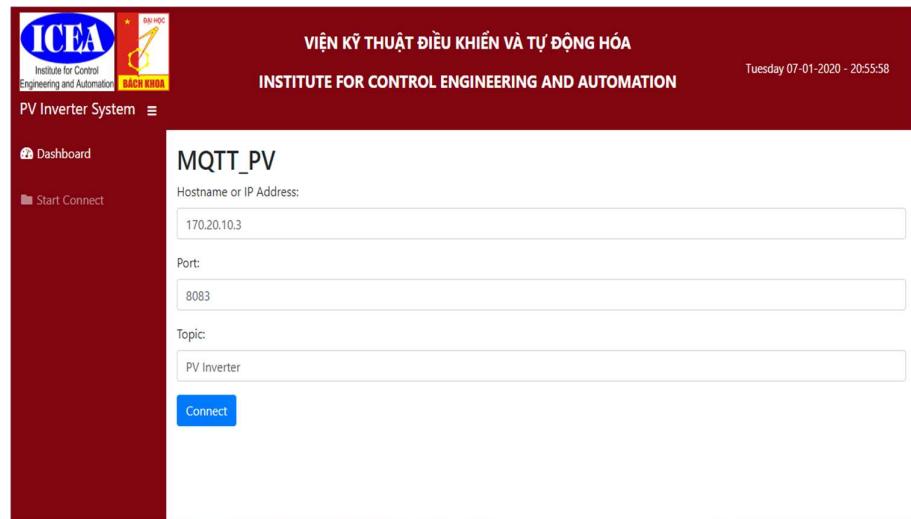
Hình 4. 8 Giao diện Dashboard của hệ thống điều khiển



Hình 4. 9 Đồ thị hiển thị sự thay đổi giá trị của hệ thống

Data Table Example						
Show	10	entries	Search:			
	Tín hiệu	Đại lượng	Hệ số khuếch đại	Giá trị Offset	Giá trị	Đơn vị
	UA	Giá trị điện áp tải pha A	400/1.5	1.5	123	V
	UB	Giá trị điện áp tải pha B	400/1.5	1.5	124	V
	UC	Giá trị điện áp tải pha C	400/1.5	1.5	125	V
	IA	Giá trị dòng điện tải pha A	400/1.5	1.5	10	V
	IB	Giá trị dòng điện tải pha B	400/1.5	1.5	11	V
	IC	Giá trị dòng điện tải pha C	400/1.5	1.5	12	V
	UA	Giá trị điện áp tải pha A	400/1.5	1.5	123	V
	UB	Giá trị điện áp tải pha B	400/1.5	1.5	124	V
	UC	Giá trị điện áp tải pha C	400/1.5	1.5	125	V

Hình 4. 10 Bảng lưu giá trị nhận được từ hệ thống



Hình 4. 11 Trang kết nối với Raspberry Pi qua địa chỉ IP

## CHƯƠNG 5. GHÉP NỐI VÀ KẾT QUẢ THỰC NGHIỆM

Ở chương này sẽ trình bày về quy trình ghép nối hệ thống và truyền tin giữa các thành phần với nhau. Do các thành phần bộ biến đổi điện tử công suất còn đang trong quá trình test chạy thử từng thành phần riêng biệt nên trong chương này để kiểm tra tính thực tế của hệ thống IoT thì em xin mượn hệ thống điều áp tích cực của viện Kỹ thuật điều khiển và Tự động hóa để chạy thực nghiệm.

### 5.1 Quy trình ghép nối

#### 5.1.1 Ghép nối khối truyền thông Zigbee

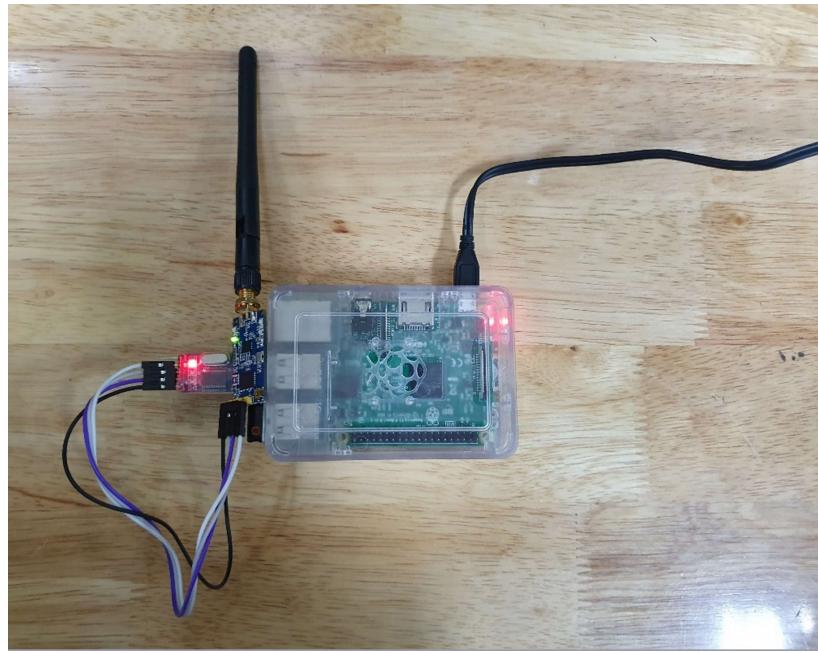
Kết nối module Zigbee với mạch điều khiển cũng như với Raspberry Pi. Thực hiện cấp nguồn cho từng thành phần.



Hình 5. 1 Kết nối zigbee với mạch điều khiển trung tâm



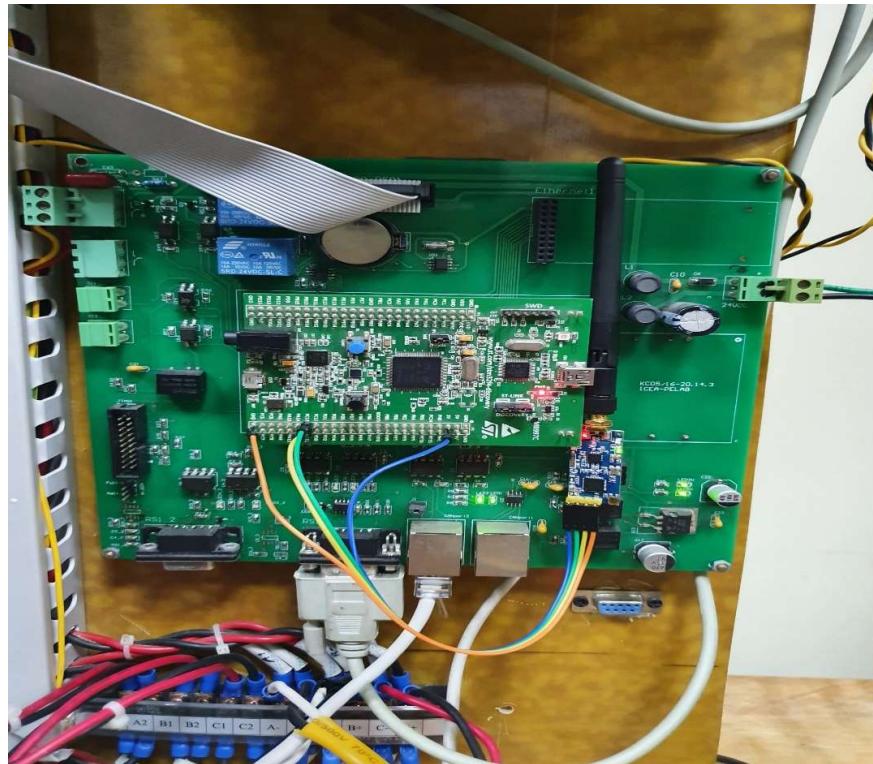
Hình 5. 2 Mặt trên của mạch điều khiển trung tâm



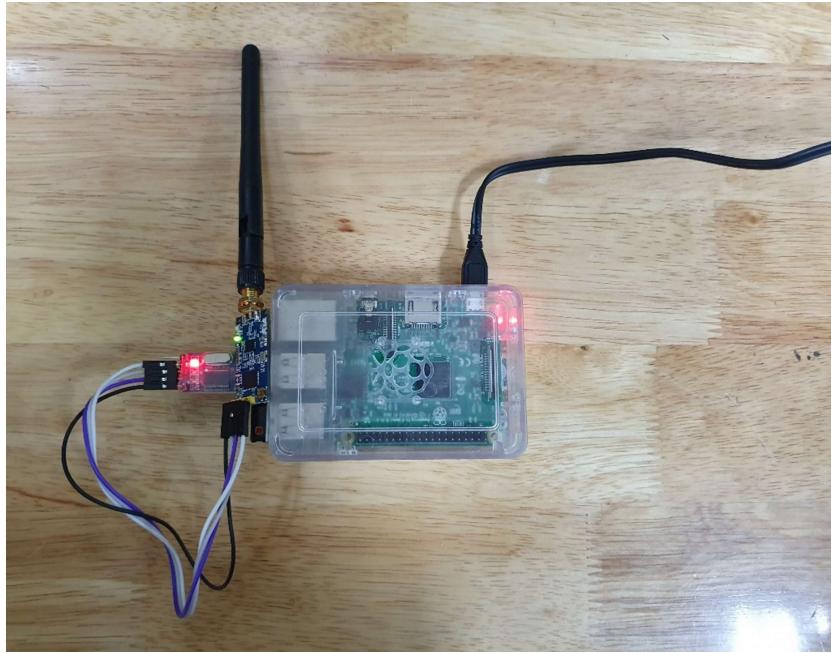
Hình 5. 3 Kết nối Zigbee với Raspberry Pi

### 5.1.2 Ghép nối mạch điều khiển và Raspberry Pi

Ghép nối STM32F407 vào mạch. Thiết lập trạng thái kết nối giữa 2 module Zigbee phần truyền (Master) và phần nhận (Raspberry Pi).



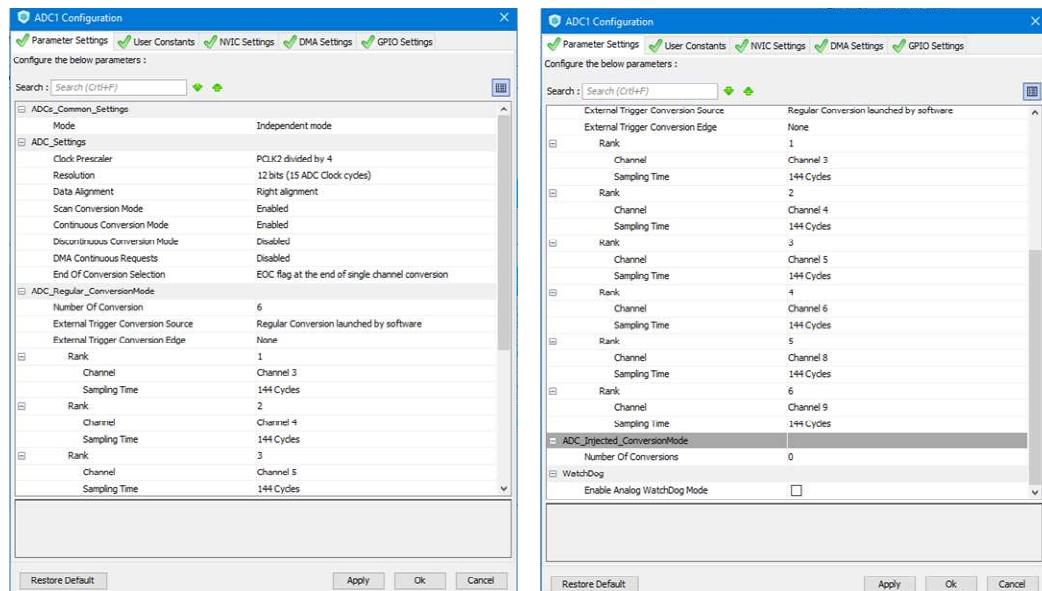
Hình 5. 4 Ghép nối Zigbee với mạch điều khiển trung tâm của hệ AVC



Hình 5. 5 Ghép nối Zigbee với Raspberry Pi cho hệ AVC

## 5.2 Truyền tin giữa các mạch đo lường và mạch điều khiển

Đây là thiết lập việc truyền tin giữa các mạch đo lường của hệ thống AVC



Hình 5. 6 Thiết lập cấu hình truyền giữa các mạch đo lường

### Kết quả và đánh giá

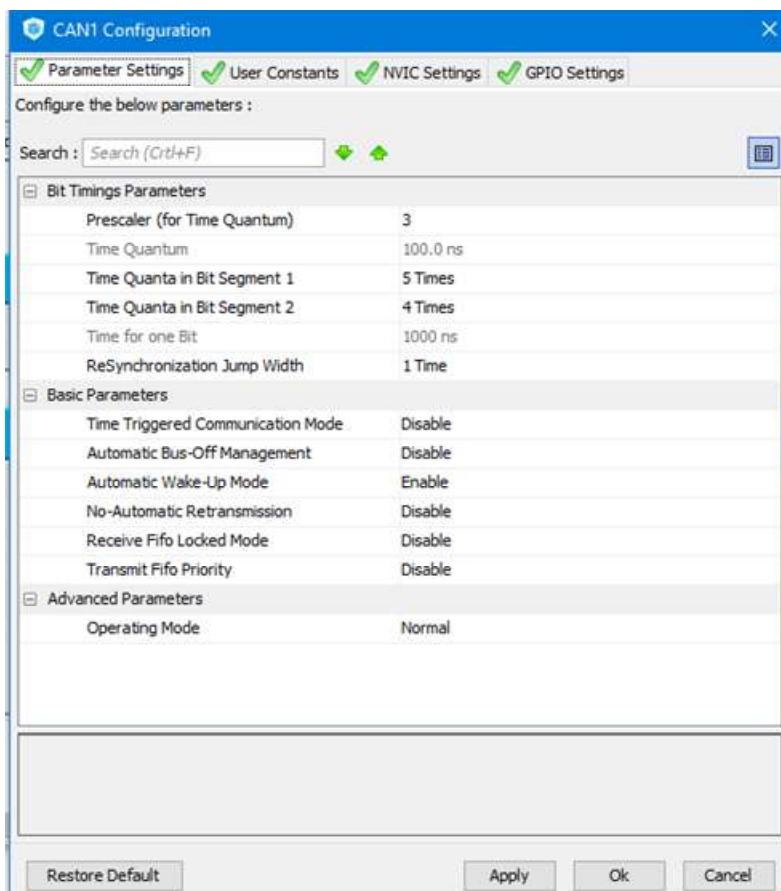
Với thiết lập như hình trên: tốc độ của IC là 60Mhz, tỉ số chia cho kênh ADC là 4, lấy mẫu sau 144 chu kỳ, thì tốc độ chuyển đổi của ADC là 104166 bit/s.

Độ chính xác: ADC có độ phân giải là 12 bit, với điện áp chuẩn là 3,3V thì cho phép sai số là 0,8mV, tức là 0,02%. Tuy nhiên trên thực tế sai số lớn hơn do điện áp chuẩn không ổn định, nhiều ảnh hưởng đến chuyển đổi của ADC, chuyển

đổi không tuyến tính. Do đó cần hiệu chỉnh độ chính xác. Khi đó, độ sai lệch là 0.08%, trong mức cho phép.

### 5.3 Truyền tin giữa các bộ điều khiển

Hình ảnh thiết lập thông số kỹ thuật cho CAN của STM32



Hình 5. 7 Thiết lập thông số kỹ thuật cho CAN của STM32

### Kết quả và đánh giá

Qua các thử nghiệm, truyền thông CAN giữa các bộ điều khiển cho kết quả:

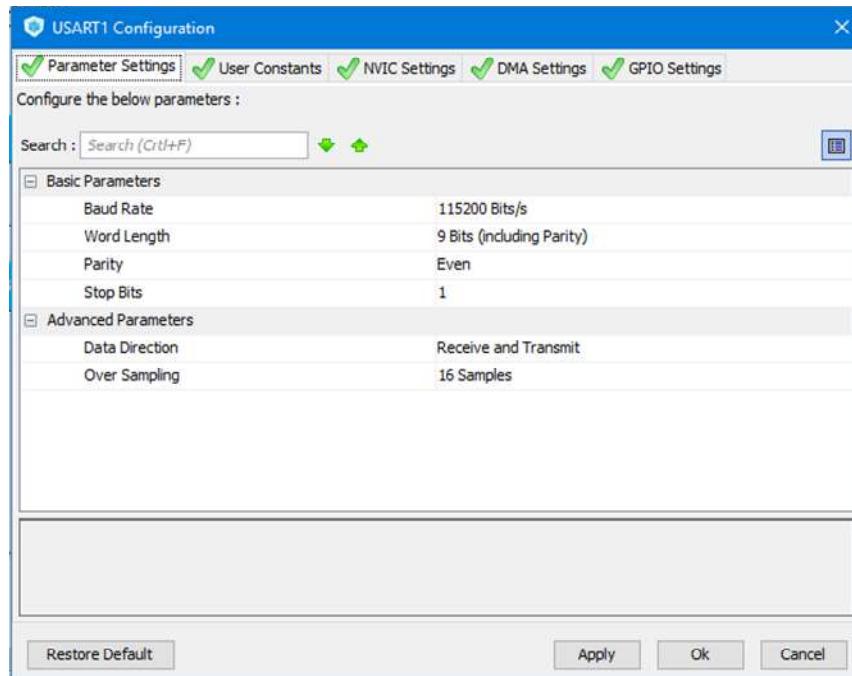
- Đường truyền ổn định.
- Tốc độ cao, tối đa 1Mbit/s.
- Số bản tin bị mất: 0%.
- Số bản tin bị lỗi: 0%.

Truyền thông CAN đã đảm bảo được yêu cầu đề ra về truyền tin giữa các bộ điều khiển.

### 5.4 Truyền tin giữa bộ điều khiển trung tâm và màn hình

Truyền tin giữa bộ điều khiển trung tâm và màn hình sử dụng giao thức Modbus RTU thông qua RS-485.

Thiết lập thông số cho bộ điều khiển trung tâm:



Hình 5. 8 Thiết lập thông số cho bộ điều khiển trung tâm

### Kết quả và đánh giá

Sử dụng truyền tin nối tiếp UART, với chuẩn kết nối RS-485, giao thức Modbus RTU đã cho thấy kết quả ổn định giữa bộ điều khiển trung tâm và màn hình.

Thông tin truyền chính xác, không xảy ra lỗi.

### 5.5 Truyền tin giữa mạch điều khiển trung tâm và Raspberry Pi

Truyền tin giữa bộ điều khiển trung tâm và Raspberry Pi sử dụng giao thức Zigbee thông qua UART.

Phần thiết lập thông số cho bộ điều khiển trung tâm đã được nêu ở trên.

Thiết lập thông số cho Raspberry Pi có thể nhận được dữ liệu từ mạch điều khiển sẽ nằm ở trong chương trình chạy trên Raspberry nằm ở mục A2 trong phần Phụ Lục.

### Kết quả và đánh giá

Việc truyền tin bằng giao thức Zigbee lần đầu tiên đã cho thấy được sự ổn định giữa mạch điều khiển trung tâm và Raspberry Pi.

Thông tin truyền chính xác và không xảy ra lỗi.



Hình 5. 9 Đồ thị hiển thị kết quả nhận từ mạch Master

Data Table Example

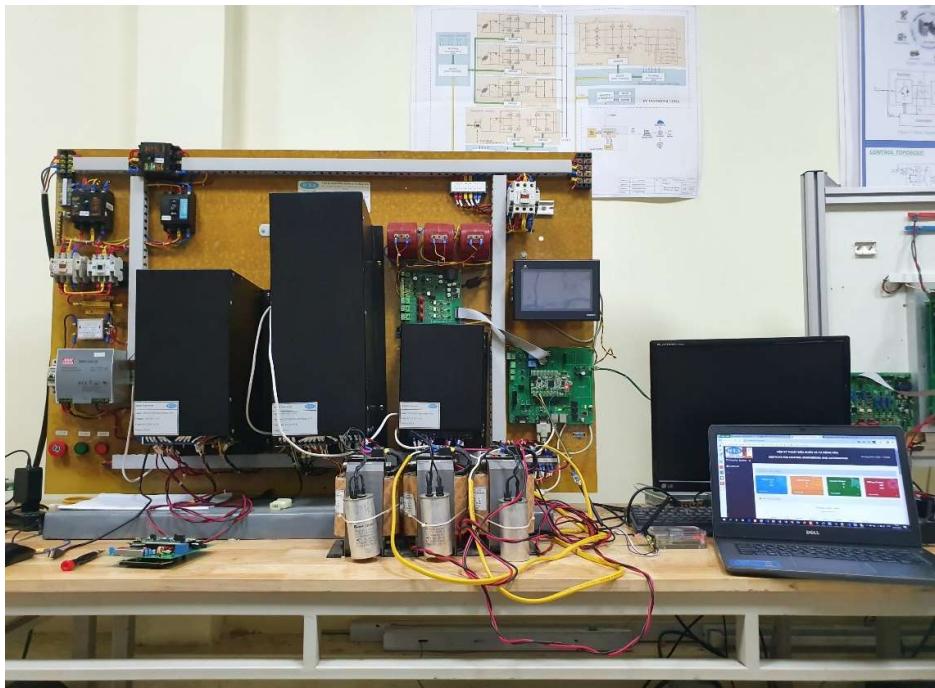
Show 10 entries Search:

Tín hiệu	Đại lượng	Hệ số khuếch đại	Giá trị Offset	Giá trị	Đơn vị
UA	Giá trị điện áp tái pha A	400/1.5	1.5	123	V
UB	Giá trị điện áp tái pha B	400/1.5	1.5	124	V
UC	Giá trị điện áp tái pha C	400/1.5	1.5	125	V
IA	Giá trị dòng điện tái pha A	400/1.5	1.5	10	A
IB	Giá trị dòng điện tái pha B	400/1.5	1.5	11	A
IC	Giá trị dòng điện tái pha C	400/1.5	1.5	12	A

Hình 5. 10 Bảng lưu trữ liệu nhận được từ Master gửi về

## 5.6 Chạy thử nghiệm hệ thống IoT đối với hệ điều áp tích cực

Kết nối các thành phần của hệ thống lại, cấp nguồn cho hệ thống ta tiến hành chạy thử nghiệm.



Hình 5. 11 Sơ đồ hệ điều áp tích cực và hệ thống IoT

Với kịch bản thực nghiệm là: Đo và hiển thị điện áp và dòng điện phía tải.

### Kết quả và đánh giá

Do thời gian có hạn và việc vận hành hệ thống AVC là tương đối phức tạp, cần nhiều người vận hành cùng lúc nên em chưa thể thực hiện chạy thực nghiệm để lấy kết quả đo và đưa vào trong quyển đồ án được. Vì vậy, em chỉ thử nghiệm được phần truyền thông giữa mạch Master và hệ thống IoT theo chương trình chuẩn của hệ thống AVC đã được thiết kế trước đó.

Phần truyền thông giữa Mạch Master và Raspberry bằng giao thức Zigbee đã cho kết quả truyền chính xác và không bị mất dữ liệu. Việc Raspberry đầy dữ liệu nhận được từ mạch Master lên web thông qua giao thức MQTT cũng đã cho kết quả chính xác. Dưới đây là một số hình ảnh biểu đạt kết quả truyền nhận giữa mạch Master và hệ thống IoT.

## CHƯƠNG 6. KẾT LUẬN

### 6.1 Kết luận

Sau một quá trình nghiên cứu, thực hiện đề tài “*Thiết kế hệ thống điều khiển giám sát và thu thập dữ liệu cho hệ PV Inverter*” em đã đạt được một số kết quả như sau:

- Nắm được hoạt động của các bộ biến đổi điện tử công suất trong hệ thống năng lượng mặt trời.
- Đưa ra được các yêu cầu cần đạt được khi thực hiện giám sát điều khiển hệ thống
- Xây dựng hệ thống điều khiển, giám sát cho và thu thập dữ liệu ở chế độ tại chỗ và chế độ từ xa.

### 6.2 Hướng phát triển của đồ án trong tương lai

Do hạn chế về mặt thời gian và thiết bị nên đồ án vẫn còn tồn tại một số vấn đề cần được cải tiến và phát triển hơn trong tương lai gần:

- Chính định các kết quả đo lường để đạt độ chính xác cao hơn.
- Mở rộng phạm vi và số lượng các thông số cần đo và thu thập.
- Tiếp tục phát triển phần mềm giám sát và điều khiển để có giao diện thân thiện hơn với người dùng, và mở rộng các chức năng cảnh báo, báo cáo định kỳ, đặt chế độ điều khiển, ...
- Thu thập và phân tích dữ liệu trong khoảng thời gian đủ dài có xem xét đến các điều kiện thời tiết và vận hành khác nhau để từ đó tìm ra xác định lỗi nếu có cũng như mối liên hệ giữa hiệu suất của hệ thống và điều kiện hoạt động.

Những vấn đề trên đây sẽ giúp cho việc cải tiến bản thân hệ thống inverter cũng như tối ưu chế độ vận hành để đạt hiệu quả cao hơn.

## TÀI LIỆU THAM KHẢO

- [1] Trần Bách, Lưới điện và hệ thống điện, Nhà xuất bản Khoa học Kỹ thuật, 2004.
- [2] Trần Trọng Minh, Vũ Hoàng Phương, Thiết kế điện tử công suất, Nhà xuất bản Đại Học Bách Khoa Hà Nội, 2014.
- [3] Yongfu Li, "On-line Monitoring System Based on Open Source Platform for Photovoltaic Array", 2017.
- [4] M.K. Deshmukh, "On-line Monitoring of Roof-Mounted Stand-Alone Solar Photovoltaic System on Residential Building", 2019.
- [5] MQTT, "[CloudMQTT - A globally distributed MQTT broker](#)",
- [6] Github, "<https://github.com/laravel/laravel>",
- [7] Võ Thanh Tùng, "Thiết kế hệ thống điều khiển giám sát cho hệ điều áp tích cực", 2018.
- [8] Tạ Đức Anh, Controller Area Network (CAN), Đại Học Bách Khoa Hồ Chí Minh, 2002
- [9] Chang-Sic Choi, "Implementation of IoT based PV Monitoring System with Message Queuing Telemetry Transfer protocol and Smart Utility Network", 2017
- [10] Siva Ramakrishna Madeti,"Monitoring System for photovoltaic plants: A review", 2017
- [11] Cristina Ventura,"Utility scale photovoltaic plant indices and models for on-line monitoring and fault detection purpose", 2016
- [12] "<http://diennangluongmattroi.biz/photovoltaic-la-gi-dien-mat-troi-la-gi.html>"
- [13] Modicon,"<http://www.modbus.org/>, Modbus Application Protocol V1.1b3",
- [14] Zigbee Alliance," <https://zigbeealliance.org/solution/zigbee/>",

## PHỤ LỤC

### A1. Chương trình hệ thống trên STM32F407

```
/* Includes -----*/
#include "main.h"
#include "stm32f4xx_hal.h"

/* USER CODE BEGIN Includes */
#include "math.h"
#include "modbus.h"
#include "can.h"
#include "adc.h"
#include "eeprom.h"
#include "data.h"
#include "rtc_ds1307.h"
/* USER CODE END Includes */

/* Private variables -----*/
ADC_HandleTypeDef hadc1;
DMA_HandleTypeDef hdma_adc1;

CAN_HandleTypeDef hcan1;

I2C_HandleTypeDef hi2c1;

SPI_HandleTypeDef hspi2;

TIM_HandleTypeDef htim2;
TIM_HandleTypeDef htim3;
TIM_HandleTypeDef htim4;
TIM_HandleTypeDef htim5;
TIM_HandleTypeDef htim12;

UART_HandleTypeDef huart3;
UART_HandleTypeDef huart6;

float Filter(Filter_TypeDef* filter);
```

```

/* USER CODE BEGIN PV */

/* Private variables -----*/
//1ms          htim3.Init.Period = 7;
//100ms         799
//200ms         1599
//250ms  htim3.Init.Period = 1999;
//500ms  htim3.Init.Period = 3999;
//1s            htim3.Init.Period = 7999;
//2s            htim3.Init.Period = 15999;
//4s            htim3.Init.Period = 31999;

CanRxMsgTypeDef RxMsg;           //khai bao dia chi con tro Rx Can
CanTxMsgTypeDef TxMsg;          //khai bao dia chi con tro Tx Can

// CAN_FilterTypeDef sFilterConfig_nam_test;
//Filter Config Can 1
//Data TypeDef
Master_TypeDef           Master;
SLAVE_TypeDef            SLAVE;
ADCs_TypeDef             ADCs;
RTC_DS1307_TypeDef      RTC_DS1307;
EEPROM_TypeDef           EEPROM;
Flag_TypeDef              Flag;

Filter_TypeDef Filter_Ua;
Filter_TypeDef Filter_Ia;

Filter_TypeDef Filter_Ub;
Filter_TypeDef Filter_Ib;

Filter_TypeDef Filter_Uc;
Filter_TypeDef Filter_Ic;

Filter_TypeDef Filter_Ia_adc;
Filter_TypeDef Filter_Ib_adc;
Filter_TypeDef Filter_Ic_adc;

struct u16ADC u16_uA;

```

```

        struct u16ADC u16_uB;
        struct u16ADC u16_uC;
        struct u16ADC u16_iA;
        struct u16ADC u16_iB;
        struct u16ADC u16_iC;

        uint8_t test_err = 3;
/* USER CODE END PV */

/* Private function prototypes -----*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DMA_Init(void);
static void MX_CAN1_Init(void);
static void MX_I2C1_Init(void);
static void MX_USART3_UART_Init(void);
static void MX_ADC1_Init(void);
static void MX_TIM2_Init(void);
static void MX_TIM3_Init(void);
static void MX_TIM4_Init(void);
static void MX_TIM5_Init(void);
static void MX_TIM12_Init(void);
static void MX_SPI2_Init(void);
static void MX_USART6_UART_Init(void);

/* USER CODE BEGIN PFP */
/* Private function prototypes -----*/
float Filter(Filter_TypeDef* filter)
{
    filter->gain_in = 0.05;
    filter->gain_out = 0.95;

    filter->out_k = filter->out_k_1*filter->gain_out + filter->in_k*filter->gain_in;

    filter->out_k_1 = filter->out_k;

```

```

        return filter->out_k;
    }
float Filter_adc(Filter_TypeDef* filter)
{
    filter->gain_in = 0.2;
    filter->gain_out = 0.8;

    filter->out_k = filter->out_k_1*filter->gain_out + filter->in_k*filter-
>gain_in;

    filter->out_k_1 = filter->out_k;

    return filter->out_k;
}
void Data_From_UART()
{
    // | Button
    Master.RUN_ALL = Master.UART_RX_0x03[4];
    Master.RESET_LOG = Master.UART_RX_0x03[6];
    Master.SET_TIME = Master.UART_RX_0x03[8];
    Master.MASTER_PAGES = Master.UART_RX_0x03[10];
    // Master.REMOTE = Master.UART_RX[12];
    // | Set Data
    RTC_DS1307.DateTime_W[0] = Master.UART_RX_0x03[14];
    //SET_Seconds
    RTC_DS1307.DateTime_W[1] = Master.UART_RX_0x03[16];
    //SET_Minutes
    RTC_DS1307.DateTime_W[2] = Master.UART_RX_0x03[18];
    //SET_Hours
    RTC_DS1307.DateTime_W[3] = 2;
    RTC_DS1307.DateTime_W[4] = Master.UART_RX_0x03[20];
    //SET_Date
    RTC_DS1307.DateTime_W[5] = Master.UART_RX_0x03[22];
    //SET_Month
}

```

```

RTC_DS1307.DateTime_W[6]      =      Master.UART_RX_0x03[24];
//SET_Year

Master.SET_VDC_REF
=
(Master.UART_RX_0x03[25]<<8|Master.UART_RX_0x03[26])*10;
Master.SET_LIMIT_VDC_MIN
=
(Master.UART_RX_0x03[27]<<8|Master.UART_RX_0x03[28])*10;
Master.SET_LIMIT_VDC_MAX
=
(Master.UART_RX_0x03[29]<<8|Master.UART_RX_0x03[30])*10;
Master.SET_LIMIT_VGRID_MIN
=
(Master.UART_RX_0x03[31]<<8|Master.UART_RX_0x03[32])*10;
Master.SET_LIMIT_VGRID_MAX
=
(Master.UART_RX_0x03[33]<<8|Master.UART_RX_0x03[34])*10;
Master.SET_LIMIT_I_GRID_MAX_SHUNT
=
(Master.UART_RX_0x03[35]<<8|Master.UART_RX_0x03[36])*10;
Master.SET_LIMIT_I_INV_MAX_SERIES
=
(Master.UART_RX_0x03[37]<<8|Master.UART_RX_0x03[38])*10;
Master.SET_VOLTAGE_SERIES
= (Master.UART_RX_0x03[39]<<8|Master.UART_RX_0x03[40])*10;

}

```

```

void Data_To_UART()
{
    unsigned n;
    n = *((unsigned *)(&Master.floa.SERIES_V_A_GRID));
    //Dis_Ua_grid
        Master.Data_Tx[0] = n >> 8;
        Master.Data_Tx[1] = n;
        Master.Data_Tx[2] = n >> 24;
        Master.Data_Tx[3] = n >> 16;
    n = *((unsigned *)(&Master.floa.SERIES_V_B_GRID));
    //Dis_Ub_grid
        Master.Data_Tx[4] = n >> 8;
        Master.Data_Tx[5] = n;
        Master.Data_Tx[6] = n >> 24;
        Master.Data_Tx[7] = n >> 16;
}

```

```

n = *((unsigned*)(void*)&Master.floa.SERIES_V_C_GRID);
//Dis_Uc_grid
    Master.Data_Tx[8] = n >> 8;
    Master.Data_Tx[9] = n;
    Master.Data_Tx[10] = n >> 24;
    Master.Data_Tx[11] = n >> 16;
n = *((unsigned*)(void*)&Master.floa.SHUNT_FREQUENCY);
//Dis_f_grid
    Master.Data_Tx[12] = n >> 8;
    Master.Data_Tx[13] = n;
    Master.Data_Tx[14] = n >> 24;
    Master.Data_Tx[15] = n >> 16;
n = *((unsigned*)(void*)&Master.floa.uA_RMS_Load);
//Dis_Ua_load
    Master.Data_Tx[16] = n >> 8;
    Master.Data_Tx[17] = n;
    Master.Data_Tx[18] = n >> 24;
    Master.Data_Tx[19] = n >> 16;
n = *((unsigned*)(void*)&Master.floa.uB_RMS_Load);
//Dis_Ub_load
    Master.Data_Tx[20] = n >> 8;
    Master.Data_Tx[21] = n;
    Master.Data_Tx[22] = n >> 24;
    Master.Data_Tx[23] = n >> 16;
n = *((unsigned*)(void*)&Master.floa.uC_RMS_Load);
//Dis_Uc_load
    Master.Data_Tx[24] = n >> 8;
    Master.Data_Tx[25] = n;
    Master.Data_Tx[26] = n >> 24;
    Master.Data_Tx[27] = n >> 16;
n = *((unsigned*)(void*)&Master.floa.iA_RMS_Load);
//Dis_Ia_load
    Master.Data_Tx[28] = n >> 8;
    Master.Data_Tx[29] = n;
    Master.Data_Tx[30] = n >> 24;
    Master.Data_Tx[31] = n >> 16;
n = *((unsigned*)(void*)&Master.floa.iB_RMS_Load);
//Dis_Ib_load

```

```

        Master.Data_Tx[32] = n >> 8;
        Master.Data_Tx[33] = n;
        Master.Data_Tx[34] = n >> 24;
        Master.Data_Tx[35] = n >> 16;
        n = *((unsigned *)(&Master.floa.iC_RMS_Load));
        //Dis_Ic_load
        Master.Data_Tx[36] = n >> 8;
        Master.Data_Tx[37] = n;
        Master.Data_Tx[38] = n >> 24;
        Master.Data_Tx[39] = n >> 16;
        n = *((unsigned *)(&Master.floa.SHUNT_VDC));
        //Dis_Vdc
        Master.Data_Tx[40] = n >> 8;
        Master.Data_Tx[41] = n;
        Master.Data_Tx[42] = n >> 24;
        Master.Data_Tx[43] = n >> 16;

        Master.Data_Tx[51] = Flag.AVC_loop;

                                //Add 56
        Master.Data_Tx[53] = Flag.Shunt_Status;           // 0 NO
ERR | 1 Run_err | 2 Check ok | 3 Check err | 4 Config... //Add 57
        Master.Data_Tx[55] = Flag.Series_Status;         // 0 NO ERR | 1
Run_err | 2 Check ok | 3 Check err | 4 Config... //Add 58
        Master.Data_Tx[57] = Flag.AVC_Status;           //
AVC_Status 0-5

                                //Add 59

//Master.Data_Tx[65] = test_err; // Add 63 test
Master.Data_Tx[65] = RTC_DS1307.DateTime_R[0]; // Add 63
Master.Data_Tx[67] = RTC_DS1307.DateTime_R[1];
Master.Data_Tx[69] = RTC_DS1307.DateTime_R[2];
Master.Data_Tx[71] = RTC_DS1307.DateTime_R[4];
Master.Data_Tx[73] = RTC_DS1307.DateTime_R[5];
Master.Data_Tx[75] = RTC_DS1307.DateTime_R[6]; // Add 68

for (int i = 0; i < 8; i++)

```

```

{
    Master.Data_Tx[77+i*7*2] = EEPROM.Secs[8*Master.MASTER_PAGES + i]; //Secs
    Master.Data_Tx[79+i*7*2] = EEPROM.Mins[8*Master.MASTER_PAGES + i]; //Mins
    Master.Data_Tx[81+i*7*2] = EEPROM.Hours[8*Master.MASTER_PAGES + i]; //Hours
    Master.Data_Tx[83+i*7*2] = EEPROM.Date[8*Master.MASTER_PAGES + i]; //Date
    Master.Data_Tx[85+i*7*2] = EEPROM.Month[8*Master.MASTER_PAGES + i]; //Month
    Master.Data_Tx[87+i*7*2] = EEPROM.Year[8*Master.MASTER_PAGES + i]; //Year
    Master.Data_Tx[89+i*7*2] = EEPROM.EventType[8*Master.MASTER_PAGES + i]; //Type
}

for (int i = 0; i < 9; i++) Master.Data_Tx[189 + i*2] =
Master.MASTER_PAGES*8 + i + 1; //Add 125 | (125-31+1)*2-1=189

Master.Data_Tx[205] = Master.MASTER_PAGES + 1;
//Add 133

Master.Data_Tx[207] = EEPROM.Last_Reset[0];
Master.Data_Tx[209] = EEPROM.Last_Reset[1];
Master.Data_Tx[211] = EEPROM.Last_Reset[2];
Master.Data_Tx[213] = EEPROM.Last_Reset[3];
Master.Data_Tx[215] = EEPROM.Last_Reset[4];
Master.Data_Tx[217] = EEPROM.Last_Reset[5];
}

void Main_Response_Slave_Data()
{
    if (hcan1.pRxMsg->StdId == SERIES_V_A_GRID_SEND)
        SLAVE.floa.SERIES_V_A_GRID = (hcan1.pRxMsg-
>Data[0]<<8|hcan1.pRxMsg->Data[1])/10.0;
    if (hcan1.pRxMsg->StdId == SERIES_V_B_GRID_SEND)
        SLAVE.floa.SERIES_V_B_GRID = (hcan1.pRxMsg-
>Data[0]<<8|hcan1.pRxMsg->Data[1])/10.0;
    if (hcan1.pRxMsg->StdId == SERIES_V_C_GRID_SEND)
        SLAVE.floa.SERIES_V_C_GRID = (hcan1.pRxMsg-
>Data[0]<<8|hcan1.pRxMsg->Data[1])/10.0;
}

```

```

if (hcan1.pRxMsg->StdId == SHUNT_VDC_SEND)
    SLAVE.floa.SHUNT_VDC = (hcan1.pRxMsg-
>Data[0]<<8|hcan1.pRxMsg->Data[1])/10.0;
    if (hcan1.pRxMsg->StdId == SHUNT_FREQUENCY_SEND)
        SLAVE.floa.SHUNT_FREQUENCY = (hcan1.pRxMsg-
>Data[0]<<8|hcan1.pRxMsg->Data[1])/10.0;
    }
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
    if (hadc->Instance == hadc1.Instance && Flag.StartSys > 3000)
    {
        switch (Flag.offset)
        {
            case 0:
                if(ADCs.count < 135)
                {
                    ADCs.floa.offset_uA += ADCs.DMA_data[0];
                    ADCs.count++;
                }
                else
                {
                    ADCs.count = 0;
                    ADCs.floa.offset_uA =
ADCs.floa.offset_uA/135.0;
                    Flag.offset = 1;
                }
                break;
            case 1:
                if(ADCs.count < 135)
                {
                    ADCs.floa.offset_uB += ADCs.DMA_data[1];
                    ADCs.count++;
                }
                else
                {
                    ADCs.count = 0;
                }
        }
    }
}

```

```

        ADCs.floa.offset_uB          =
ADCs.floa.offset_uB/135.0;
        Flag.offset = 2;
    }
    break;
case 2:
if(ADCs.count < 135)
{
    ADCs.floa.offset_uC += ADCs.DMA_data[2];
    ADCs.count++;
}
else
{
    ADCs.count = 0;
    ADCs.floa.offset_uC          =
ADCs.floa.offset_uC/135.0;

    Flag.offset = 3;
}
break;
case 3:
if(ADCs.count < 135)
{
    ADCs.floa.offset_iA += ADCs.DMA_data[3];
    ADCs.count++;
}
else
{
    ADCs.count = 0;
    ADCs.floa.offset_iA          =
ADCs.floa.offset_iA/135.0;
    Flag.offset = 4;
}
break;
case 4:
if(ADCs.count < 135)
{

```

```

        ADCs.floa.offset_iB += ADCs.DMA_data[4];
        ADCs.count++;
    }
    else
    {
        ADCs.count = 0;
        ADCs.floa.offset_iB =
ADCs.floa.offset_iB/135.0;
        Flag.offset = 5;
    }
    break;
case 5:
    if(ADCs.count < 135)
    {
        ADCs.floa.offset_iC += ADCs.DMA_data[5];
        ADCs.count++;
    }
    else
    {
        ADCs.count = 0;
        ADCs.floa.offset_iC =
ADCs.floa.offset_iC/135.0;
        Flag.offset = 6;
    }
    break;
///////////
case 6:
    ADCs.floa.uA_ADC_Load =
(3.3*(ADCs.DMA_data[0] - ADCs.floa.offset_uA)/4095.0)*400/1.5;

    if(ADCs.count < 135)
    {
        ADCs.floa.sum_uA = ADCs.floa.sum_uA +
ADCs.floa.uA_ADC_Load*ADCs.floa.uA_ADC_Load;

        ADCs.count++;
    }
    else

```

```

    {
        ADCs.count = 0;
        ADCs.floa.uA_RMS_Load      =
sqrt(ADCs.floa.sum_uA/135.0); ADCs.floa.sum_uA = 0;
        Filter_Ua.in_k = ADCs.floa.uA_RMS_Load ;
        ADCs.floa.uA_RMS_Load      =
Filter(&Filter_Ua);
        Flag.offset = 7;
    }
    break;
case 7:
    ADCs.floa.uB_ADC_Load      =
(3.3*(ADCs.DMA_data[1] - ADCs.floa.offset_uB)/4095.0)*400/1.5;
    if(ADCs.count < 135)
    {
        ADCs.floa.sum_uB = ADCs.floa.sum_uB +
ADCs.floa.uB_ADC_Load*ADCs.floa.uB_ADC_Load;
        ADCs.count++;
    }
    else
    {
        ADCs.count = 0;
        ADCs.floa.uB_RMS_Load      =
sqrt(ADCs.floa.sum_uB/135.0); ADCs.floa.sum_uB = 0;
        Filter_Ub.in_k = ADCs.floa.uB_RMS_Load ;
        ADCs.floa.uB_RMS_Load      =
Filter(&Filter_Ub);
        Flag.offset = 8;
    }
    break;
case 8:
    ADCs.floa.uC_ADC_Load      =
(3.3*(ADCs.DMA_data[2]- ADCs.floa.offset_uC)/4095.0)*400/1.5;

    if(ADCs.count < 135)
    {
        ADCs.floa.sum_uC = ADCs.floa.sum_uC +
ADCs.floa.uC_ADC_Load*ADCs.floa.uC_ADC_Load;

```

```

        ADCs.count++;
    }
    else
    {
        ADCs.count = 0;
        ADCs.floa.uC_RMS_Load      =
sqrt(ADCs.floa.sum_uC/135.0); ADCs.floa.sum_uC = 0;

        Filter_Uc.in_k = ADCs.floa.uC_RMS_Load ;
        ADCs.floa.uC_RMS_Load      =
Filter(&Filter_Uc);
        Flag.offset = 9;
    }
    break;
case 9:
    ADCs.floa.iA_ADC_Load      =
(3.3*(ADCs.DMA_data[3]-      ADCs.floa.offset_iA)/4095.0)*(20*5.0/0.7908);

    Filter_Ia_adc.in_k = ADCs.floa.iA_ADC_Load ;
    ADCs.floa.iA_ADC_Load      =
Filter_adc(&Filter_Ia_adc);
    if(ADCs.count < 135)
    {
        ADCs.floa.sum_iA = ADCs.floa.sum_iA +
ADCs.floa.iA_ADC_Load*ADCs.floa.iA_ADC_Load;

        ADCs.count++;
    }
    else
    {
        ADCs.count = 0;
        ADCs.floa.iA_RMS_Load      =
(sqrt(ADCs.floa.sum_iA/135.0)); ADCs.floa.sum_iA = 0;
        Filter_Ia.in_k = ADCs.floa.iA_RMS_Load ;
        ADCs.floa.iA_RMS_Load = Filter(&Filter_Ia)
- 0.0;      //////
        Flag.offset = 10;
    }
    break;

```

```

case 10:
    ADCs.floa.iB_ADC_Load = (3.3*(ADCs.DMA_data[4]- ADCs.floa.offset_iB)/4095.0)*(20*5.0/0.7908);

    Filter_Ib_adc.in_k = ADCs.floa.iB_ADC_Load;
    ADCs.floa.iB_ADC_Load = Filter_adc(&Filter_Ib_adc);
    if(ADCs.count < 135)
    {
        ADCs.floa.sum_iB = ADCs.floa.sum_iB + ADCs.floa.iB_ADC_Load*ADCs.floa.iB_ADC_Load;
        ADCs.count++;
    }
    else
    {
        ADCs.count = 0;
        ADCs.floa.iB_RMS_Load = (sqrt(ADCs.floa.sum_iB/135.0)); ADCs.floa.sum_iB = 0;
        Filter_Ib.in_k = ADCs.floa.iB_RMS_Load;
        ADCs.floa.iB_RMS_Load = Filter(&Filter_Ib)
- 0.0; //////
        Flag.offset = 11;
    }
    break;
case 11:
    ADCs.floa.iC_ADC_Load = (3.3*(ADCs.DMA_data[5]- ADCs.floa.offset_iC)/4095.0)*(20*5.0/0.7908);

    Filter_Ic_adc.in_k = ADCs.floa.iC_ADC_Load;
    ADCs.floa.iC_ADC_Load = Filter_adc(&Filter_Ic_adc);
    if(ADCs.count < 135)
    {
        ADCs.floa.sum_iC = ADCs.floa.sum_iC + ADCs.floa.iC_ADC_Load*ADCs.floa.iC_ADC_Load;
        ADCs.count++;
    }
    else

```

```

    {
        ADCs.count = 0;
        ADCs.floa.iC_RMS_Load      =
(sqrt(ADCs.floa.sum_iC/135.0)); ADCs.floa.sum_iC = 0;
        Filter_Ic.in_k = ADCs.floa.iC_RMS_Load ;
        ADCs.floa.iC_RMS_Load      =
Filter(&Filter_Ic);                                //-----///
        Flag.offset = 6;
    }
    break;
}
if (ADCs.floa.uA_RMS_Load < 10) { ADCs.floa.uA_RMS_Load
= 0; ADCs.floa.iA_RMS_Load = 0;}
if (ADCs.floa.uB_RMS_Load < 10) { ADCs.floa.uB_RMS_Load =
0; ADCs.floa.iB_RMS_Load = 0;}
if (ADCs.floa.uC_RMS_Load < 10) { ADCs.floa.uC_RMS_Load =
0; ADCs.floa.iC_RMS_Load = 0;}
}

}

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
/*****
*****htim2 - 1ms
*****STARTING SYSTEM FLAG
*****LED TOGGLE
*****Flag.Error
*****Master.RUN_ALL
*****/
if (htim->Instance == htim2.Instance)
{
/* STARTING SYSTEM FLAG -----*/
{
    if (Flag.StartSys <= 3000) Flag.StartSys++;
}

```

```

/* LED TOGGLE -----*/
{
    if (Flag.StartSys < 3000)
    {
        if (Flag.Led == 0)    HAL_GPIO_WritePin(GPIOD,
LD3_Pin|LD4_Pin|LD5_Pin|LD6_Pin, GPIO_PIN_SET);
        if (Flag.Led == 200)HAL_GPIO_WritePin(GPIOD,
LD3_Pin|LD4_Pin|LD5_Pin|LD6_Pin, GPIO_PIN_RESET);
        Flag.Led++;
        if (Flag.Led > 400) Flag.Led = 0;
    }
    else
    {
        if (Flag.Led == 0)    HAL_GPIO_WritePin(GPIOD,
LD3_Pin|LD4_Pin|LD5_Pin|LD6_Pin, GPIO_PIN_SET);
        if (Flag.Led == 40)   HAL_GPIO_WritePin(GPIOD,
LD3_Pin|LD4_Pin|LD5_Pin|LD6_Pin, GPIO_PIN_RESET);
        Flag.Led++;
        if (Flag.Led > 500) Flag.Led = 0;
    }
}

/* Flag.ERROR -----*/
{
    //| 0 Empty      | 1 Error      |
    if (Flag.Error == 1) Flag.UART = 0; //Write Stop
}

/* Flag.Event_Log -----*/
// 0 Empty | 1 Log Start
if (Flag.Event_Log == 1)
{
    Flag.EEPROM = 1;
    Flag.Event_Log = 0;
}

```

## A2. Chương trình xử lý dữ liệu nhận về trên Raspberry Pi

```
#!/usr/bin/python3
import time
import serial
ser = serial.Serial(
    port = '/dev/ttyUSB0',
    baudrate = 9600,#115200,
    parity = serial.PARITY_EVEN,
    stopbits = serial.STOPBITS_ONE,
    bytesize = serial.EIGHTBITS,
    timeout = 1
)

while True:
    data = ser.read(50)
    # ser.write(data)
    print("Raspberry's receiving : ", data)
    print(len(data))
    # print(type(data))
    if len(data) > 2:
        #numByte = int.from_bytes(data[6],"big")
        numByte = int(data[6])
        print(numByte)

        offset = 6
        length = offset+numByte
        i = offset+1
        while i<length:
            value_int = int(data[i])*256+int(data[i+1])
            i=i+2
            # value_int = int.from_bytes(value_str,"big")
            print(value_int)
ser.close()
```

### A3. Định nghĩa bản tin CAN của hệ thống

- Khung truyền Modbus

Truyền giá trị vào thanh ghi trên HMI								
Master -> HMI								
Slave Add.	Function Code	Starting Area/Variable : LW (16bit)		Number of LW		Byte Count (8 bit)	Data	CRC
		Add. High	Add. Low	Add. High	Add. Low			
0x01	0x10	...	...	...	...	...	...	*** ***
Địa chỉ HMI	Mã hàm	Địa chỉ bắt đầu đọc từ HMI	Số lượng thanh ghi 16bit cần đọc từ HMI	SL data (8bit)	Data	Mã hàm kết thúc		

### A4. Định nghĩa bản tin CAN của hệ thống AVC

- Mạch Master gửi

Tên	ID		Data		
	3 bit cao	8 bit thấp	2byte	2 byte	2 byte
SET_GRID_VOLTAGE_SHUNT	001	0x40	V_grid	-	-
SET_GRID_FREQUENCY_SHUNT	001	0x41	f_grid	-	-
SET_V_DC_REF_SHUNT	001	0x42	U_dc_ref	-	-
SET_T_START_UP_SHUNT	001	0x43	t_startup	-	-
SET_LIMIT_VGRID_MIN_SHUNT	001	0x44	V_grid_min	-	-
SET_LIMIT_VGRID_MAX_SHUNT	001	0x45	V_grid_max	-	-
SET_LIMIT_VDC_MIN_SHUNT	001	0x46	V_vdc_min	-	-
SET_LIMIT_VDC_MAX_SHUNT	001	0x47	V_vdc_max	-	-
SET_LIMIT_I_GRID_MAX_SHUNT	001	0x48	I_grid_max	-	-
SET_LIMIT_TEMP_MAX_SHUNT	001	0x49	Nhiệt độ van	-	-
SET_VOLTAGE_SERIES	001	0x60	Uđm	-	-
SET_FREQUENCY_SERIES	001	0x61	F	-	-
SET_POWER_SERIES	001	0x62	P_đm (x10)	-	-
SET_HS_POWER_SERIES	001	0x63	Hệ số (x10)	-	-
SET_CURRENT_SERIES	001	0x64	Idm (x10)	-	-
SET_LIMIT_V_INV_MAX_SERIES	001	0x65	V_bu	-	-
SET_LIMIT_I_INV_MAX_SERIES	001	0x66	I_inv (x10)	-	-
SET_LIMIT_V_GRID_MIN_SERIES	001	0x67	V_grid_min	-	-
SET_LIMIT_V_GRID_MAX_SERIES	001	0x68	V_grid_max	-	-
SET_LIMIT_TEMP_MAX_SERIES	001	0x69	Nhiệt độ van	-	-
REQUEST_V_DC_SHUNT	001	0x80	0x00FF	-	-
REQUEST_V_GRID_ALL_SHUNT	001	0x81	0x00FF	-	-

REQUEST_V_GRID_A_SHUNT	001	0x82	0x00FF			
REQUEST_V_GRID_B_SHUNT	001	0x83	0x00FF			
REQUEST_V_GRID_C_SHUNT	001	0x84	0x00FF			
REQUEST_V_GRID_ALL_SERIES	001	0x85	0x00FF	-	-	-
REQUEST_V_GRID_A_SERIES	001	0x86	0x00FF			
REQUEST_V_GRID_B_SERIES	001	0x87	0x00FF			
REQUEST_V_GRID_C_SERIES	001	0x88	0x00FF			
REQUEST_V_LOAD_ALL_SERIES	001	0x89	0x00FF	-	-	-
REQUEST_V_LOAD_A_SERIES	001	0x8A	0x00FF	-	-	-
REQUEST_V_LOAD_B_SERIES	001	0x8B	0x00FF	-	-	-
REQUEST_V_LOAD_C_SERIES	001	0x8C	0x00FF	-	-	-
REQUEST_FREQUENCY_SERIES	001	0x8D	0x00FF	-	-	-
REQUEST_FREQUENCY_SHUNT	001	0x8E	0x00FF			

- Mạch Slave gửi

Tên	ID		Data			
	3 bit cao	8 bit thấp	2 byte	2 byte	2 byte	2 byte
RESPONSE_V_GRID_SEND_SHUNT	010	0x08	Va_grid	Vb_grid	Vc_grid	Freq (x10)
RESPONSE_V_A_GRID_SEND_SHUNT	010	0x09	Va_grid	-	-	-
RESPONSE_V_B_GRID_SEND_SHUNT	010	0x0A	Vb_grid	-	-	-
RESPONSE_V_C_GRID_SEND_SHUNT	010	0x0B	Vc_grid	-	-	-
RESPONSE_FREQ_GRID_SEND_SHUNT	010	0x0C	freq (x10)			

Tên	ID		Data			
	3 bit cao	8 bit thấp	2 byte	2 byte	2 byte	2 byte
RESPONSE_V_LOAD_SEND_SERIES	011	0x05	Va_out	Vb_out	Vc_out	-
RESPONSE_V_A_LOAD_SEND_SERIES	011	0x06	Va_out	-	-	-
RESPONSE_V_B_LOAD_SEND_SERIES	011	0x07	Vb_out	-	-	-
RESPONSE_V_C_LOAD_SEND_SERIES	011	0x08	Vc_out	-	-	-
RESPONSE_V_GRID_SEND_SERIES	011	0x09	Va_grid	Vb_grid	Vc_grid	-
RESPONSE_V_A_GRID_SEND_SERIES	011	0x0A	Va_grid	-	-	-
RESPONSE_V_B_GRID_SEND_SERIES	011	0x0B	Vb_grid	-	-	-
RESPONSE_V_C_GRID_SEND_SERIES	011	0x0C	Vc_grid	-	-	-

