BỘ CÔNG THƯƠNG

TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI

ĐỒ ÁN TỐT NGHIỆP

NGÀNH CÔNG NGHIỆP KỸ THUẬT ĐIỆN TỬ VIỄN THÔNG

THIẾT KẾ HỆ THỐNG ĐO VÀ GIÁM SÁT HIỆU SUẤT MÁY HÀN

Giảng viên hướng dẫn : TS. Hà Thị Kim Duyên

Sinh viên thực hiện : Lê Quang Thịnh

Mã sinh viên : 2018605265

Lớp : Điện tử 04 – K13

Hà Nội - 2022

LỜI CẢM ƠN

Trước tiên em xin bày tỏ lòng biết ơn chân thành và sâu sắc nhất của mình tới TS. Hà Thị Kim Duyên, người đã hướng dẫn tận tình và hiệu quả, thường xuyên động viên chúng em trong quá trình hoàn thiện đề tài. Người đã dành cho em sự ưu ái nhất trong thời gian học tập, nghiên cứu cũng như quá trình hoàn thành thực tập tốt nghiệp.

Em xin cảm ơn các Thầy giáo, Cô giáo trong khoa Điện Tử trường Đại học Công Nghiệp Hà Nội cùng tất cả thành viên lớp Điện tử 04 – K13 đã tạo điều kiện và đóng góp ý kiến để em hoàn thành tốt đồ án tốt nghiệp.

Mặc dù em đã cố gắng để hoàn thành thực tập nhưng do kiến thức cũng như khả năng còn hạn hẹp nên quá trình thực hiện đề tài còn có sai sót. Rất mong nhận được sự góp ý và chỉ bảo của quý thầy cô.

Em xin chân thành cảm ơn!

Hà Nội, Ngày 26 tháng 03 năm 2022

Sinh viên thực hiện

Lê Quang Thịnh

MỤC LỤC

DANH MỤC TỪ VIẾT TẮT

STT	Từ viết tắt	Tiếng anh	Tiếng việt
1	HMI	Human Machine Interface	Giao diện người máy
2	RAM	Random Access Memory	Bộ nhớ dữ liệu
4	DC	Direct Current	Dòng điện một chiều
5	LCD	Liquid Crystal Display	Màn hình tinh thể lỏng
6	IDE	Integrated Development Environment	Môi trường phát triển tích hợp

DANH MỤC HÌNH ẢNH VÀ BẢNG BIỂU

LỜI MỞ ĐẦU

Lịch sử nhân loại đã trải qua nhiều cuộc cách mạng khoa học kĩ thuật, tinh thần tìm tòi sáng tạo giúp con người ngày càng có nhiều phát minh, sáng kiến, tìm ra những công cụ, phương tiện mới, con đường mới để chinh phục tự nhiên. Ngày nay, chúng ta đang phấn đấu cho mục tiêu công nghiệp hóa - hiện đại hóa, rượt đuổi làn sóng các cuộc cách mạng khoa học, con người đã và đang tiếp cận với vi xử lý.

Trong những năm gần đây, công nghệ vi điện tử phát triển rất mạnh mẽ. Sự ra đời của các vi mạch cỡ lớn, cực lớn với giá thành giảm nhanh, khả năng lập trình ngày càng cao đã mang lại những thay đổi sâu sắc trong ngành kĩ thuật điện tử. Nền công nghiệp thế giới đã đạt được những thành tựu to lớn nhờ ứng dụng những tiến bộ của khoa học kĩ thuật và công nghệ, máy móc đã thay thế con người trong nhiều hoạt động lao động sản xuất.

Lập trình vi điều khiển là phần việc không thể thiếu khi chế tạo những máy móc tự động. Trong một cỗ máy tự động, nếu phần cơ khí tạo nên hình dạng và một cơ cấu hoạt động linh hoạt thì phần lập trình và mạch điện tử như một bộ não điều khiển những hoạt động đó.

Để thực hiện công giám sát hiệu suất của các máy hoạt động trong công nghiệp một cách khoa học và chính xác. Hệ thống đo và giám sát hiệu suất là một trong các giải pháp nhà sản xuất đã và đang áp dụng vào dây truyền sản xuất nhằm đáp ứng được nhu cầu tự động hóa, giảm sức lao động cho người lao động và tiết kiệm được chi phí sản xuất. Qua quá trình học tập và tìm hiểu một số môn học ở trên trường, em đã quyết định chọn đề tài: "**Thiết kế hệ thống đo và giám sát hiệu suất của máy hàn**" là đề tài đồ án tốt nghiệp.

CHƯƠNG 1: TỔNG QUAN HỆ THỐNG ĐO GIÁM SÁT HIỆU SUẤT

1.1 Tổng quan chung về hệ thống đo và giám sát hiệu suất

Trong quản lý sản xuất, việc giám sát và theo dõi hoạt động của máy móc là một vấn đề cần thiết của người quản lý. Việc theo dõi hoạt động của máy và đưa ra các kế hoạch làm việc giúp nâng cao được hiệu suất làm việc của nhà máy, đặc biệt là các nhà máy với dây chuyền sản xuất hàng loạt. Trước khi ứng dụng các hệ thống đo đạc hiệu suất của máy một cách tự động, người quản lý thường phải ghi chép hoạt động của máy vào cuối ngày. Điều này có phần ước chừng và không có độ chính xác và độ tin cậy cao, mặt khác nó còn làm mất thời gian của người quản lý.

Với sự phát triển của xã hội, khoa học kĩ thuật nói chung và sự phát triển của IOT nói riêng, việc giám sát và quản lý hiệu suất các máy ngày càng được phát triển và đưa vào các nhà máy sản suất để giảm thiểu chi phí và thời gian.

Vi điều khiển hiện nay được sử dụng rất nhiều trong các thiết bị dân dụng cũng như trong công nghiệp. Việc ứng dụng vi điều khiển vào công nghiệp để giảm giá thành và sử dụng linh hoạt đang được sử dụng rộng rãi. Đặc biệt, một số vi điều khiển có kết hợp với các chuẩn truyền thông không dây giúp vi điều khiển kết hợp với một số hệ thống giám sát qua Internet ngày càng thuận tiện. Đó cũng là lúc IOT nên được ứng dụng rộng rãi trong môi trường công nghiệp nhiều hơn.

Mô hình này tận dụng được việc nhỏ việc sử dụng vi điều khiển và cảm biến thu thập các tín hiệu từ hệ thống máy trong công nghiệp, sau đó gửi lên Server thông qua các chuẩn truyền thông không dây như Wifi, Bluetooth, Lora… Là một mô hình đang được ứng dụng rất nhiều trong công nghiệp với việc tận dụng được sự linh hoạt của vi điều khiển và sự phổ biến của mạng Internet hiện nay

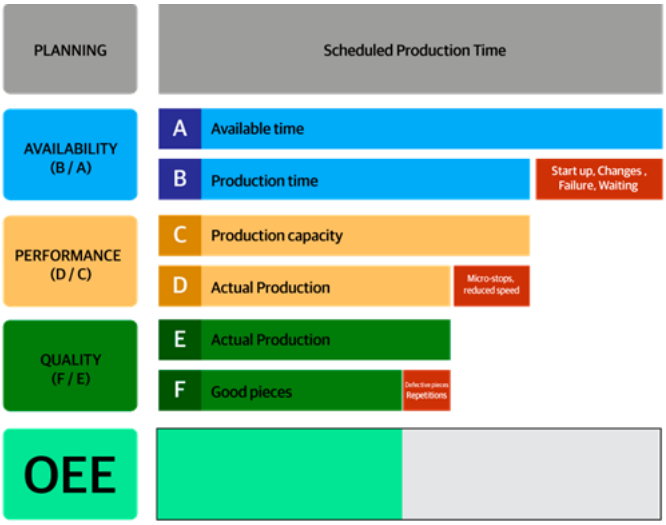
Trước những vấn đề được đặt ra ở trên: “Thiết kế hệ thống đo và giám sát hiệu suất máy hàn”. Đề tài này được thiết kế và lập trình dựa trên vi điều

khiển ESP32 – một vi điều khiển có tích hợp chuẩn truyền thông WiFi. Kết hợp với cảm biến đo dòng áp PZEM-017 nhằm thu thập các thông số về dòng điện, điện áp và công suất từ các loại máy hàn trong công nghiệp.

Kết hợp các thông số đó và việc tính toán thời gian hàn, dừng hàn, hệ thống đã giải quyết được vấn đề đo đạc trong máy hàn và giải quyết được các bài toán tính toán hiệu suất của máy hàn. Ngoài ra, hệ thống còn có thể giao tiếp với Server thông qua WiFi để việc theo dõi và hiển thị, thống kê dễ dàng.

1.2 Các hệ thống đo và giám sát hiệu suất trên thị trường

[1]Hiện nay trên thị trường cũng có rất nhiều hệ thống giám sát hiệu suất tích hợp trong các loại máy CNC, máy đúc nhựa, máy làm khuôn… Trên thị trường, các hệ thống đó được gọi là OEE (Overall Equipment Effectiveness)



Hình 1-1: Phần mền tính hiệu suất OEE và tạo kế hoạch

Hình 1-2: Phần mềm OEE giám sát hiệu suất máy trong sản xuất

Các công ty trong nước có giải pháp về hệ thống đo và giám sát OEE

- Công Ty Cổ Phần Quốc Tế Ecozen
- Công Ty Cổ Phần Advantech
- Công Ty TNHH Công Nghệ Và Tự Động Hóa Rostek

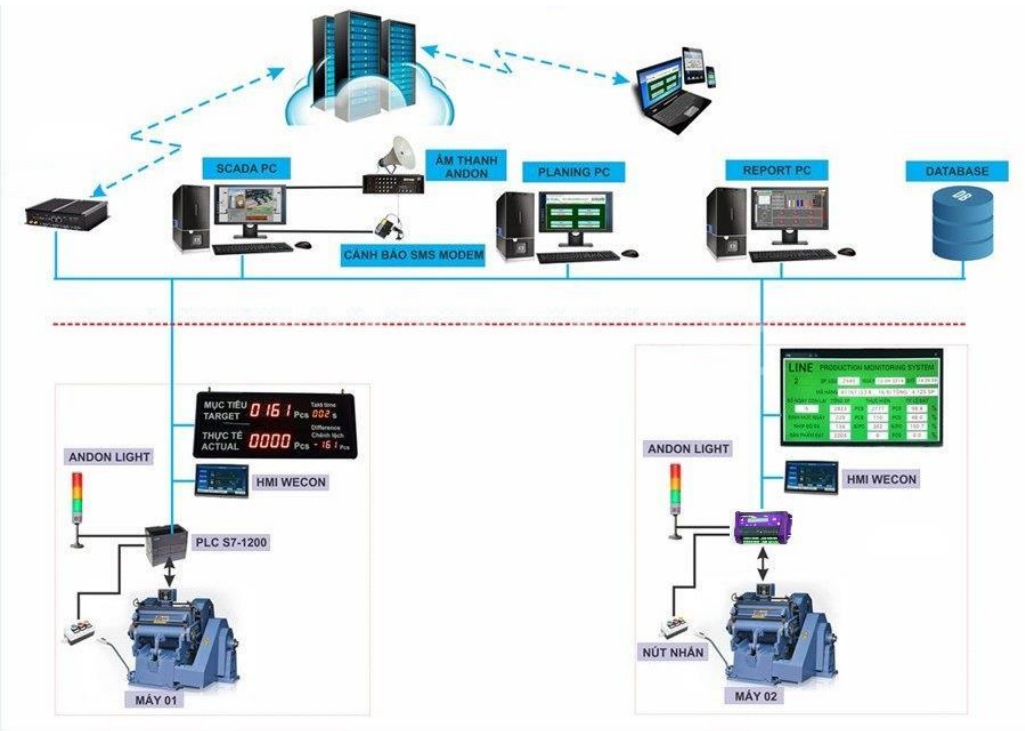
Các công ty nước ngoài có giải pháp về hệ thống đo và giám sát OEE

- ABB
- Siemens
- SL Controls

1.3 Một số mô hình phổ biến của hệ thống đo và giám sát

Với xu thế của cuộc cách mạng 4.0 hiện nay, IOT, Bigdata… đang được ứng dụng rất nhiều trong công nghiệp. Một số mô hình sử dụng IOT và Bigdata hiện nay:

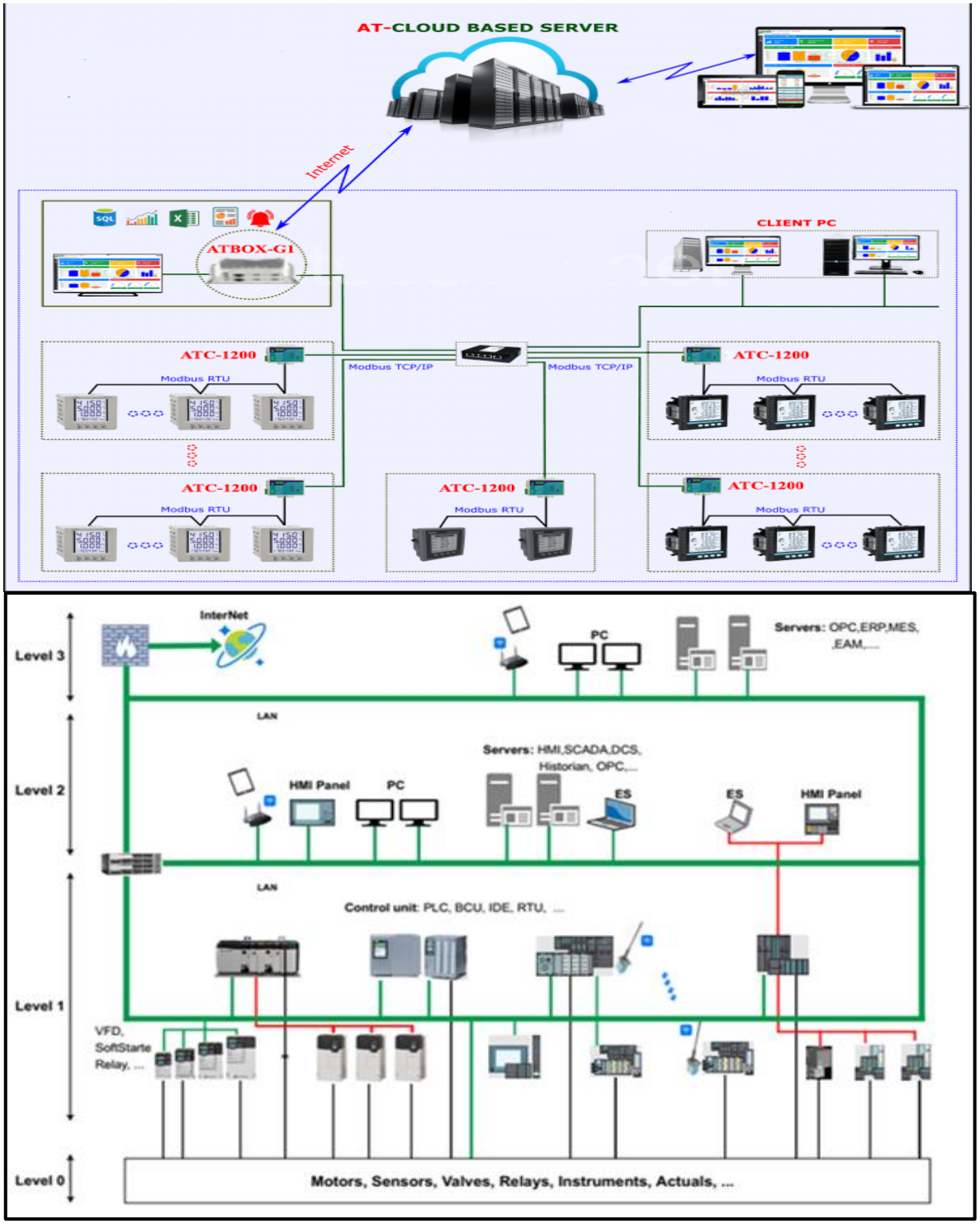
Hệ thống giám sát, quản trị trong nhà máy sản xuất: Bao gồm việc thu thập dữ liệu từ các máy trong sản xuất, hiển thị sản lượng lên nhà máy và đồng bộ giữa các máy với nhau dựa trên mạng internet



Hình 1-3: Hệ thống quản trị sản suất trong nhà máy

Hệ thống đo và giám sát điện năng trong nhà máy sản xuất:

Bao gồm việc thu thập dữ liệu từ các trạm điện, hiển thị lên máy chủ và đồng bộ giữa các máy với nhau dựa trên mạng internet.



Hình 1-4: Hệ thống đo và giám sát điện năng

Nhờ có việc đồng bộ các thiết bị với máy tính làm cho việc giúp cho người quản lý và giám sát có thể điều hành và đưa ra các kế hoạch phát triển cho nhà máy ngày càng phát triển hơn.

Do có những lợi ích như vậy nên nhiều nhà máy và doanh nghiệp đang dần dần đưa các thiết bị điều khiển và giám sát thông minh vào trong nhà máy, song song với việc tự động hóa nhà máy, trở thành nhà máy thông minh…

1.4 Một số lợi ích của hệ thống đo và giám sát

Tiết kiệm thời gian lưu dữ liệu

Việc nhân viên giám sát phải ghi chép bằng tay và mất nhiều thời gian cho việc thống kê các sự cố xảy ra; đã được thay thế bởi hệ thống giám sát sản xuất được thiết kế dựa trên công nghệ vi xử lý; kết nối trực tiếp với máy tính thông qua hệ thống server được thiết kế chuyên dụng sẽ báo cáo thông tin các sự cố trong suốt quá trình sản xuất. Thông tin về những sự cố sẽ được lưu trữ trên Server hoặc các máy tính cùng mạng.

Dễ dàng kiểm soát sự cố

Nhờ việc xử lý và truyền số liệu qua mạng LAN bằng Server và được kết nối trực tuyến; người quản lý có thể dễ dàng kiểm soát những sự cố xảy ra trong quá trình sản xuất trên màn hình máy tính; ngoài bảng hiển thị và cảnh báo Alarm ngay tại nơi sản xuất.

Hệ thống giám sát sản xuất còn có thể đưa ra những báo cáo chi tiết hoặc tổng thể về những sự cố đã xảy ra và thời gian dừng hoạt động trong ca hoặc trong ngày làm việc.

Ngoài ra, người quản lý còn có thể sử dụng các thông tin đã được lưu trữ trong Server nhằm phục vụ cho công việc xử lý số liệu thống kê; phân tích khả năng xảy ra sự cố ở mỗi công đoạn hoặc từng khu làm việc; để tìm ra chính xác nguyên nhân của sự cố và có thể khắc phục.

Vì vậy hệ thống giám sát sản xuất giúp người quản lý kiểm soát quá trình sản xuất được tốt hơn; nhanh hơn và độ tin cậy cao nhất.

Lợi ích của hệ thống giám sát sản xuất

Hiển thị giá trị sản lượng đặt ra mục tiêu và giá trị thực tế tính đến thời điểm hiện tại. Giúp người quản lý và công nhân nắm được năng suất mình đạt được; từ đó sẽ điều chỉnh tiến độ sản xuất cho phù hợp.

Hiển thị tại các vị trí quan trọng trên dây chuyền một cách tức thời và trực quan tất cả các thông tin được lưu thành cơ sở dữ liệu để báo cáo và in ấn.

Ứng dụng rộng rãi trong các ngành sản xuất như: thực phẩm, hóa chất, dệt may, điện tử… Giúp quản lý được sản lượng từ xa dùng mạng LAN hay Internet.

Ngoài ra còn có một số lợi ích khác nữa như:

- Quản lý chất lượng sản phẩm, báo cáo sự cố
- Thay thế các công việc thủ công, giúp tiết kiệm về thời gian và sức lao động.
- Độ chính xác cao, giảm sự thấ thoát nguyên vật liệu và thành phẩm.
- Hoạt động tự động giúp việc giám sát sản xuất trở nên nhanh chóng và tối ưu.
- Hệ thống giám sát sản xuất còn giúp khâu quản lý sản lượng có thể quản lý mọi hoạt động từ xa thông qua mạng Lan nội bộ.
- Tạo thêm mối liên kết chặt chẽ giữa khu sản xuất với các bộ phận văn phòng khác; từ đó đưa ra kế hoạch cải thiện sản xuất phù hợp để nâng cao năng suất lao động và sản xuất.

1.5 Kết luận chương 1

Dựa vào nhu cầu sử dụng hệ thống đo và giám sát hiệu suất tại các nhà máy, xí nghiệp trong thực tiễn ngày càng nhiều, yêu cầu công nghệ được cải tiến ngày càng hiện đại và hệ thống có nhiều ưu điểm nên em đã lựa chọn đề tài này để làm đồ án tốt nghiệp.

CHƯƠNG 2: CƠ SỞ LÝ THUYẾT

2.1 Phần cứng của hệ thống

2.1.1 Vi điều khiển ESP32

[2]ESP32 là một series các vi điều khiển trên một vi mạch giá rẻ, năng lượng thấp có hỗ trợ WiFi và Bluetooth chế độ kép. Dòng ESP32 sử dụng bộ vi xử lý Tensilica Xtensa LX6 ở cả hai biến thể lõi kép và lõi đơn, và bao gồm các công tắc antenna tích hợp, RF balun, bộ khuếch đại công suất, bộ khuếch đại thu nhiễu thấp, bộ lọc và module quản lý năng lượng.

ESP32 được chế tạo và phát triển bởi Espressif Systems, một công ty Trung Quốc có trụ sở tại Thượng Hải, và được sản xuất bởi TSMC bằng cách sử dụng công nghệ 40 nm. ESP32 là sản phẩm kế thừa từ vi điều khiển ESP8266.

Hình 2-1: Các loại ESP trên thị trường thông dụng

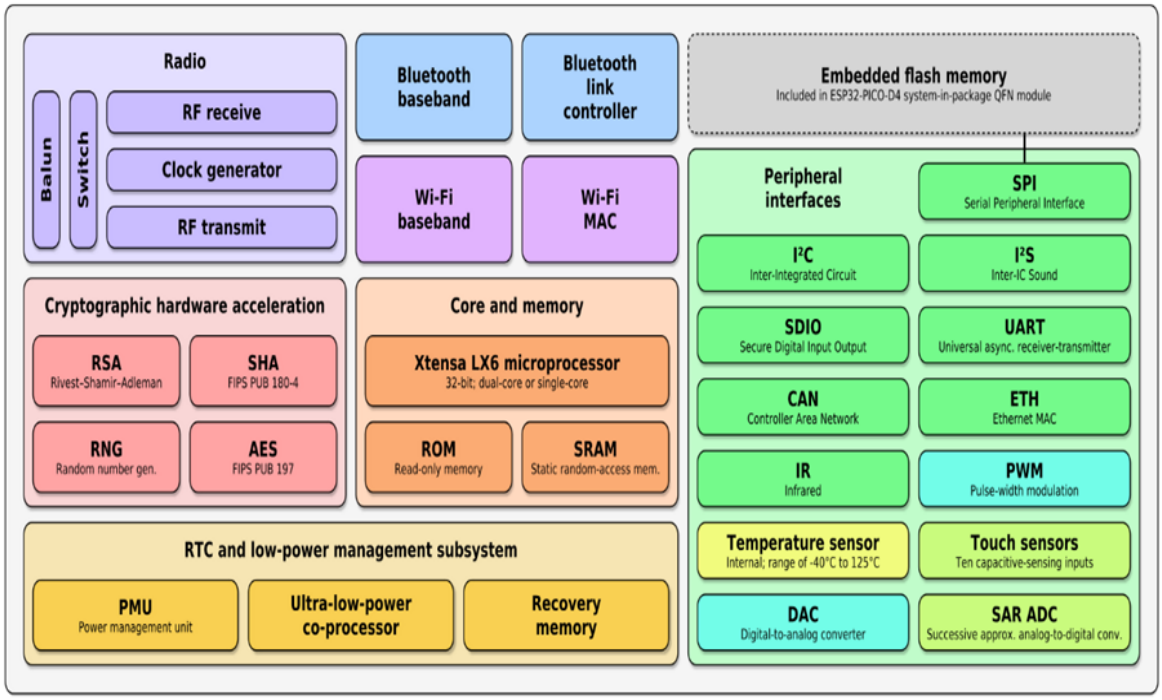
Các tính năng của ESP32 bao gồm:

- Bộ xử lý: CPU: Bộ vi xử lý Xtensa lõi kép (hoặc lõi đơn) 32-bit LX6, hoạt động ở tần số 240 MHz (160 MHz cho ESP32-S0WD và ESP32-4WDH) [4] và hoạt động ở tối đa 600 MIPS (200 MIPS với ESP32-S0WD/ESP32-U4WDH)

- Bộ đồng xử lý (co-processor) công suất cực thấp (Ultra low power, viết tắt: ULP)

- Hệ thống xung nhịp: CPU Clock, RTC Clock và Audio PLL Clock
- Bộ nhớ nội: 448 KB bộ nhớ ROM cho việc booting và các tính năng lõi, 520 KB bộ nhớ SRAM trên chip cho dữ liệu và tập lệnh
- Kết nối không dây:Wi-Fi: 802.11 b/g/n
- Bluetooth: v4.2 BR/EDR và BLE (chia sẻ sóng vô tuyến với Wi-Fi)
- 34 GPIO và các ngoại vi
- ADC SAR 12 bit, 18 kênh, DAC 2 × 8-bit, 10 cảm biến cảm ứng (touch sensor) (GPIO cảm ứng điện dung)
- 3 SPI (SPI, HSPI và VSPI) hoạt động ở cả 2 chế độ master/slave.[8] Module ESP32 hỗ trợ 4 ngoại vi SPI với SPI0 và SPI1 kết nối đến bộ nhớ flash của ESP32 còn SPI2 và SPI3 tương ứng với HSPI và VSPI.[9]
- 2 I²S
- 2 I²C, hoạt động được ở cả chế độ master và slave, với chế độ Standard mode (100 Kbit/s) và Fast mode (400 Kbit/s). Hỗ trợ 2 chế độ định địa chỉ là 7-bit và 10-bit.Các GPIO đều có thể được dùng để triển khai I²C.
- 3 UART (UART0, UART1, UART2) với tốc độ lên đến 5 Mbps[10]
- SD/SDIO/CE-ATA/MMC/eMMC host controller
- SDIO/SPI slave controller
- Ethernet MAC interface cho DMA và IEEE 1588 Precision Time Protocol (tạm dịch: Giao thức thời gian chính xác IEEE 1588)
- CAN bus 2.0
- Bộ điều khiển hồng ngoại từ xa (TX/RX, lên đến 8 kênh)
- PWM cho điều khiển động cơ
- LED PWM (lên đến 16 kênh)
- Cảm biến hiệu ứng Hall
- Bộ tiền khuếch đại analog công suất cực thấp (Ultra low power analog pre-amplifier)
- Hỗ trợ tất cả các tính năng bảo mật chuẩn IEEE 802.11, bao gồm WFA, WPA/WPA2 và WAPI.

- Secure boot (tạm dịch: khởi động an toàn)
- Mã hóa flash 1024-bit OTP, lên đến 768-bit cho khách hàng
- Tăng tốc mã hóa phần cứng: AES, SHA-2, RSA, elliptic curve cryptography (ECC, tạm dịch: mật mã đường cong ellip), trình tạo số ngẫu nhiên (random number generator, viết tắt: RNG)
- Bộ ổn áp nội với điện áp rơi thấp (internal low-dropout regulator)
- Miền nguồn riêng (individual power domain) cho RTC
- Dòng 5 µA cho chế độ deep sleep
- Trở lại hoạt động từ ngắt GPIO, timer, đo ADC, ngắt với cảm ứng điện dung

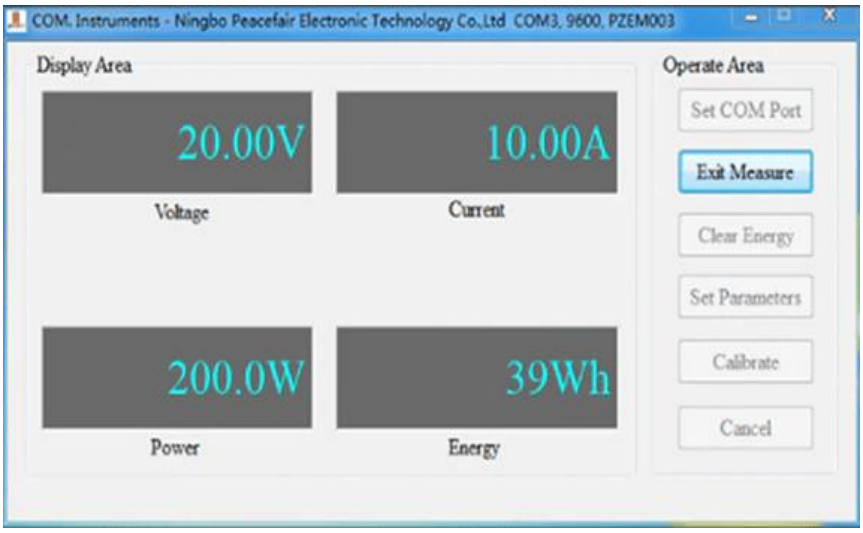


Hình 2-2: Sơ đồ khối các chức năng của ESP32

2.1.2 Cảm biến dòng điện điện áp PZEM-17

Hình 2-3: Cảm biến PZEM-17

[3]PZEM-17 là một module được sản suất ra để đo điện áp một chiều và dòng điện một chiều. Việc đo dòng điện một chiều được thực hiện qua một điện trở có giá trị nhỏ được gọi là điện trở Shunt (R_{shunt}). Nó là một module của hãng Peacefair. Peacefair là một thương hiệu rất nổi tiếng của Trung Quốc với chất lượng tốt và giá cả chuyên về các sản phẩm đo lường. Ngoài đo dòng điện và điện áp một chiều, module này còn có thể đo năng lượng và điện năng tiêu thụ.



Hình 2-4: Giao diện nhà sản suất cung cấp của PZEM-17

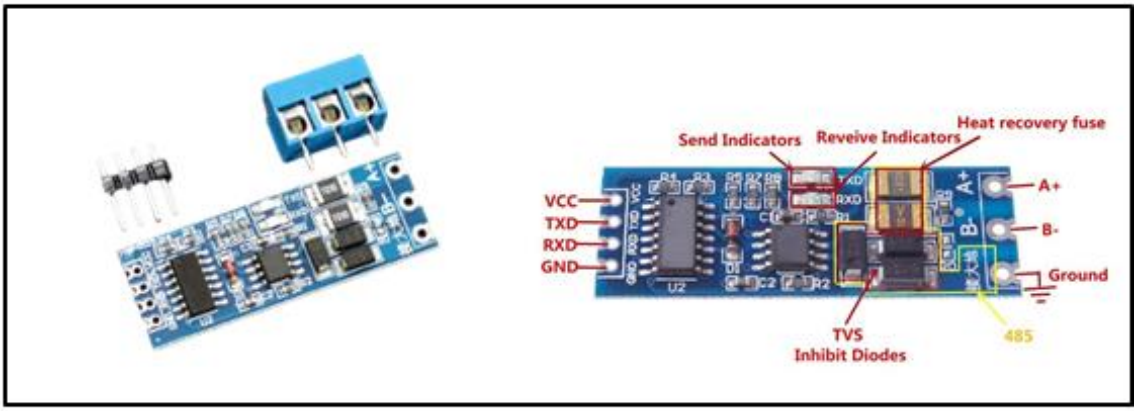
PZEM-17 không có màn hình hiển thị, vì vậy giá thành của nó rất rẻ. Nó được kết nối thông qua tầng RS485 và giao thức Modbus-RTU nhằm tương thích với tất cả các thiết bị công nghiệp.

Sử dụng phần mềm Modbus Poll hoặc phần mềm của hãng Peacefair cung cấp cùng với dây cable chuyển đổi từ RS485 sang USB, chúng ta có thể kiểm tra thử và hiệu chỉnh các giá trị của cảm biến.

2.1.3 Module chuyển đổi từ UART sang RS485

Sử dụng IC chuyển đổi MAX485 kết hợp với IC số 74HC14, module này có chức năng chuyển đổi dạng kết nối từ UART sang RS485.

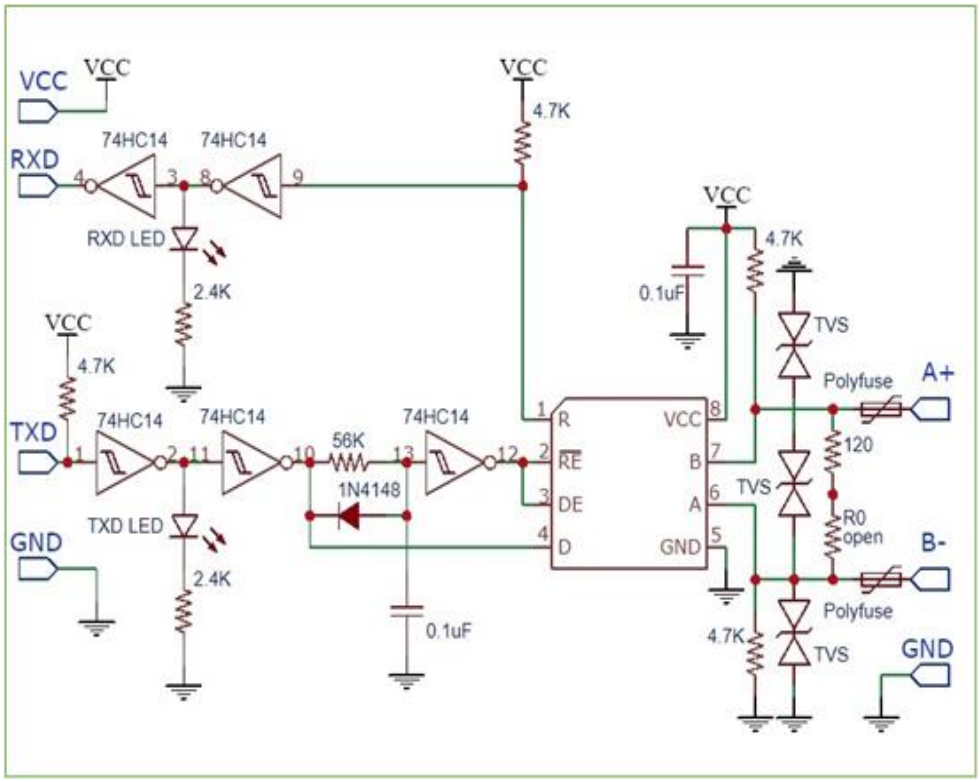
[2]IC Max485 là bộ chuyển đổi tín hiệu của chuẩn giao tiếp RS232 sang tín hiệu của chuẩn giao tiếp RS485 để có thể truyền tín hiệu đi trên đường dây RS485 và từ đó có thể truyền tín hiệu đi xa và nhanh được. Max485 truyền nhận dữ liệu năng lượng thấp, bus Max485 truyền dữ liệu vi sai bằng 2 dây A, B nên khoảng cách truyền lớn, khả năng chống nhiễu tốt.



Hình 2-5: Module chuyển đổi USART sang RS485

[4]Thông số kỹ thuật IC giao tiếp MAX485:

- Tốc độ bit Max= 2,5Mbps
- Có thể kết nối tối đa 32 thiết bị trên bus 485.
- Điện áp hoạt động: -7V ~ 12V (ổn định nhất ở 5V)
- Với A-B > 200mV sẽ tạo mức logic 1.
- Với B-A>200mV sẽ tạo mức logic 0.



Hình 2-6: Sơ đồ nguyên lý của module USART – RS485

2.1.4 Màn hình HMI Kinseal SUP

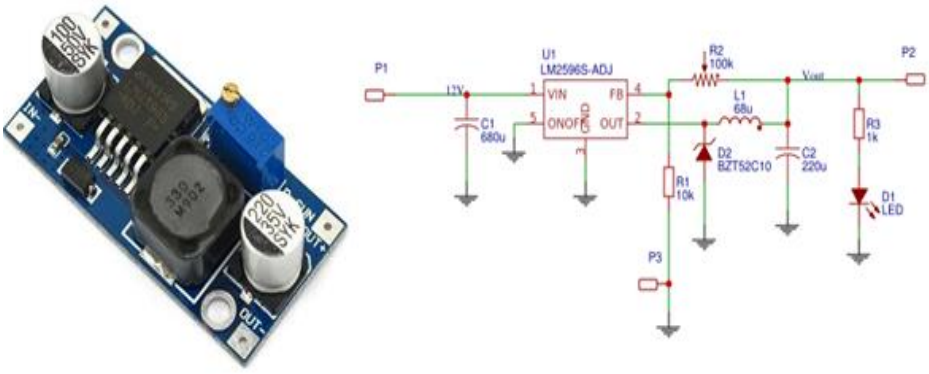
SUP043SDRG1 - Màn hình cảm ứng 4.3 inch - HMI Kinseal SUP

Hình 2-7: Màn hình HMI Kinseal SUP

- Model: SUP043SDRG1
- Cổng kết nối: COM1 RS232/RS485/RS422
- Cáp lập trình: Cáp USB Type A (2 đầu đực)
- Thương hiệu: Jinxi / KinSeal

2.1.5 Module nguồn hạ áp DC-DC 3A LM2596

[5]Mạch giảm áp DC-DC LM2596 có led hiển thị điện áp ngõ ra - vào, cho dòng điện ngõ ra lên đến 3A



Hình 2-8: Mạch giảm áp DC-DC LM2596

Thông số kỹ thuật:

- Điện áp đầu vào 2,5 V - 36 V
- Điện áp đầu ra 1.25 V - 35 V (có thể điều chỉnh)
- Dùng IC LM2596 chuyển đổi với tần số lên đến 150Khz
- Có nút nhấn thay đổi hiển thị điện áp ngõ ra - vào
- Dòng ngõ ra tối đa 3A, công suất 15W

- Kích thước: 66mm x 36mm x 14mm

2.2 Phần Firmware của hệ thống

2.2.1 Arduino framework

- Arduino là một nền tảng mã nguồn mở được sử dụng để xây dựng các ứng dụng điện tử tương tác với nhau hoặc với môi trường được thuận lợi hơn.
- Arduino giống như một máy tính nhỏ để người dùng có thể lập trình và thực hiện các dự án điện tử mà không cần phải có các công cụ chuyên biệt để phục vụ việc nạp code. [6]
- Arduino tương tác với thế giới thông qua các cảm biến điện tử, đèn, và động cơ.

Arduino gồm

- Phần cứng gồm một board mạch mã nguồn mở (thường gọi là vi điều khiển): có thể lập trình được.
- Các phần mềm hỗ trợ phát triển tích hợp IDE (Integrated Development Environment) dùng để soạn thảo, biên dịch code và nạp chương cho board.

Ứng dụng của Arduino trong đời sống

- Làm Robot. Arduino có khả năng đọc các thiết bị cảm biến, điều khiển động cơ,… nên nó thường được dùng để làm bộ xử lý trung tâm của rất nhiều loại robot.
- Game tương tác: Arduino có thể được sử dụng để tương tác với Joystick, màn hình,… khi chơi các game như Tetrix, phá gach, Mario…
- Máy bay không người lái.
- Điều khiển đèn tín hiệu giao thông, làm hiệu ứng đèn Led nhấp nháy trên các biển quảng cáo…
- Điều khiển các thiết bị cảm biến ánh sáng, âm thanh.
- Làm máy in 3D
- Làm đàn bằng ánh sáng
- Làm lò nướng bánh biết tweet để báo cho bạn khi bánh chín.

Arduino còn rất nhiều ứng dụng hữu ích khác tùy vào sự sáng tạo của

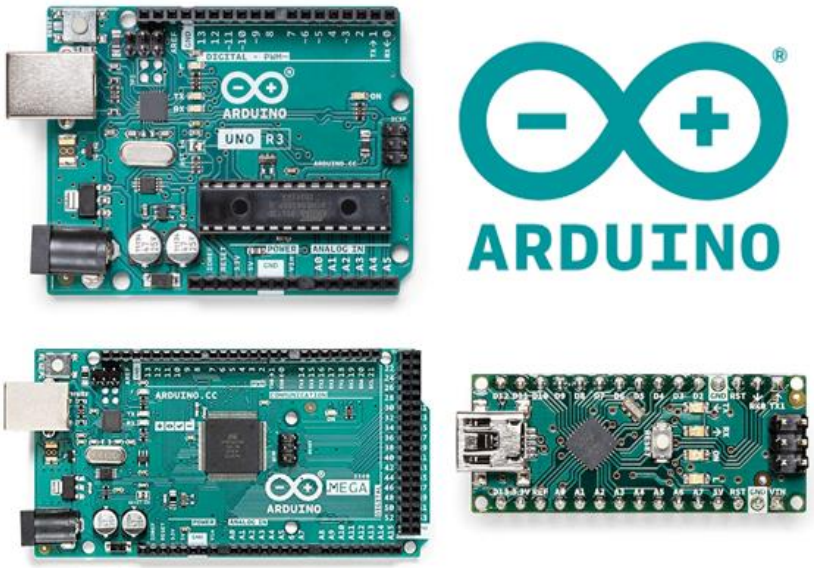
người dùng.

Mã nguồn mở

- Phần cứng và phần mềm của Arduino đều là nguồn mở - các sơ đồ đều được public trực tuyến nên bạn hoàn toàn có thể mua linh kiện về và tự làm lấy.

Khả năng kết nối

- Arduino có thể hoạt động độc lập.
- Arduino có thể kết nối với một máy tính. Máy tính của bạn được phép truy cập dữ liệu cảm biến từ thế giới bên ngoài và cung cấp thông tin phản hồi.
- Các Arduino có thể kết nối với nhau.
- Arduino có thể kết nối với thiết bị điện tử khác.
- Arduino có thể kết nối với các chip điều khiển.



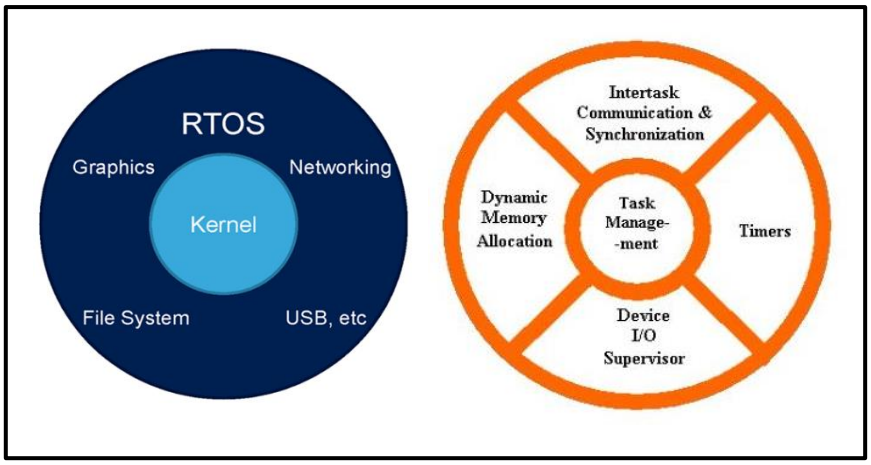
Hình 2-9: Phần cứng và phần mềm Arduino

2.2.2 FreeRTOS

FreeRTOS là một hệ điều hành nhúng thời gian thực (Real Time Operating System) mã nguồn mở được phát triển bởi Real Time Engineers Ltd, sáng lập và sở hữu bởi Richard Barry. [7]

Hình 2-10: RTOS

FreeRTOS được thiết kế phù hợp cho nhiều hệ nhúng nhỏ gọn vì nó chỉ triển khai rất ít các chức năng như: cơ chế quản lý bộ nhớ và tác vụ cơ bản, các hàm API quan trọng cho cơ chế đồng bộ. Nó không cung cấp sẵn các giao tiếp mạng, drivers, hay hệ thống quản lý tệp (file system) như những hệ điều hành nhúng cao cấp khác.



Hình 2-11: RTOS kernel

Tuy vậy, FreeRTOS có nhiều ưu điểm, hỗ trợ nhiều kiến trúc vi điều khiển khác nhau, kích thước nhỏ gọn (4.3 Kbytes sau khi biên dịch trên Arduino), được viết bằng ngôn ngữ C và có thể sử dụng, phát triển với nhiều trình biên dịch C khác nhau (GCC, OpenWatcom, Keil, IAR, Eclipse, …), cho phép không giới hạn các tác vụ chạy đồng thời, không hạn chế quyền ưu tiên thực thi, khả năng khai thác phần cứng. [4]

Ngoài ra, nó cũng cho phép triển khai các cơ chế đồng bộ giữa các tiến trình như: Queues, Counting Semaphore, Mutexes.

2.2.3 Tasmota framework

[4]Tasmota là một mã nguồn mở cho Espressif ESP8266, ESP32, ESP32-S hoặc ESP32-C3 các thiết bị dựa trên chipset được tạo ra và phát triển bởi Theo Arends.

Hình 2-12: Tasmota

Tasmota được viết bằng Arduino Framework và FreeRTOS

cho vi điều khiển dòng ESPxx. Các chức năng chính của Tasmota bao gồm:

- Tích hợp nhiều thư viện cảm biến
- Tích hợp hệ điều hành FreeRTOS
- Hỗ trợ update qua Websever hoặc Url
- Hỗ trợ các tính năng debug mạnh như console, scripts…

2.3 Phần mềm của hệ thống

2.3.1 Python flask Backend

Flask là một micro web framework được viết bằng Python. Nó được phân loại là một microframework vì nó không yêu cầu các công cụ hoặc thư viện cụ thể. Nó không có lớp trừu tượng cơ sở dữ liệu, xác thực biểu mẫu hoặc bất kỳ thành phần nào khác trong đó các thư viện của bên thứ ba có sẵn cung cấp các chức năng chung. Tuy nhiên, Flask hỗ trợ các phần mở rộng có thể thêm các tính năng ứng dụng như thể chúng được thực hiện trong chính bình. Tiện ích mở rộng cho quan hệ đối tượng, xác thực biểu mẫu, xử lý tải lên, các công nghệ xác thực mở khác nhau và một số công cụ liên quan đến các framework phổ biến. [3]

Hình 2-13: Python Flask

2.3.2 Giao thức MQTT

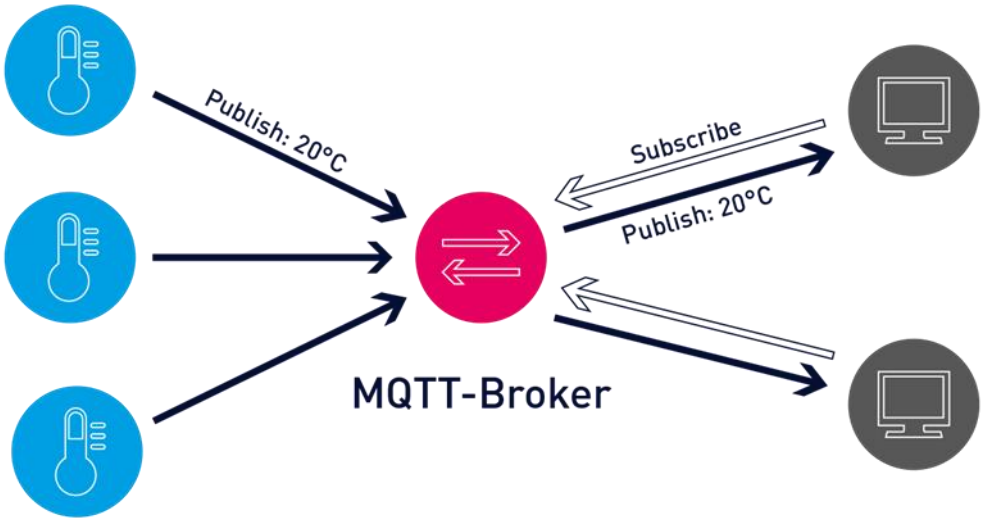
Giao thức MQTT (Message Queuing Telemetry Transport)

MQTT là một giao thức truyền tải dữ liệu, sử dụng mô hình mạng Publish – Subscribe nhằm mục đính truyền dữ liệu giữa các thiết bị. [2]

Hình 2-14: Giao thức MQTT

Giao thức thường chạy qua TCP / IP. Tuy nhiên, bất kỳ giao thức mạng nào cung cấp các kết nối theo thứ tự, không mất dữ liệu, hai chiều đều có thể hỗ trợ MQTT. Nó được thiết kế cho các kết nối với các vị trí ở xa hoặc băng

thông mạng bị hạn chế.

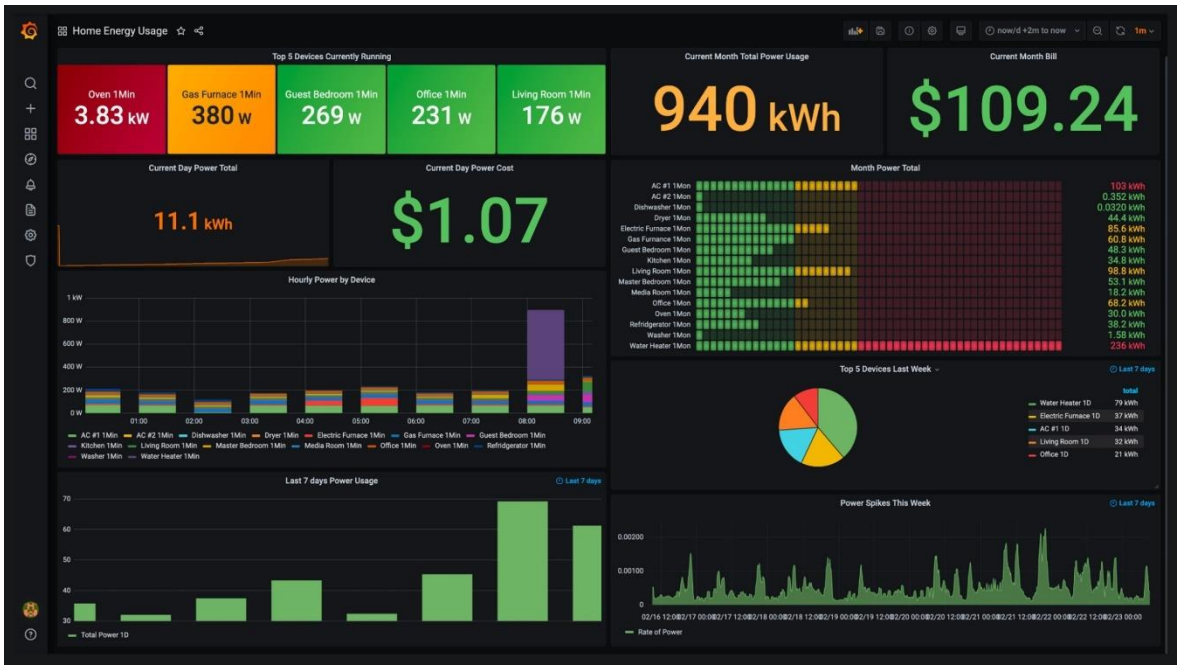


Hình 2-15: Cơ chế Pub/Sub của MQTT

2.3.3 Frontend với Grafana

Grafana được mệnh danh là bảng điều khiển bất cứ điều gì và bảng quan sát tất cả mọi thứ. Nó được dùng để truy vấn, trực quan hóa, cảnh báo và hiểu dữ liệu bất kể nó được lưu trữ ở đâu. Với Grafana, chúng ta có thể tạo, khám phá và chia sẻ tất cả dữ liệu của mình linh hoạt thông qua bảng điều khiển

Hình 2-16: Grafana và giao diện



Hình 2-17: Một giao diện về giám sát điện năng của grafana

Hình 2-18: Visualization của grafana

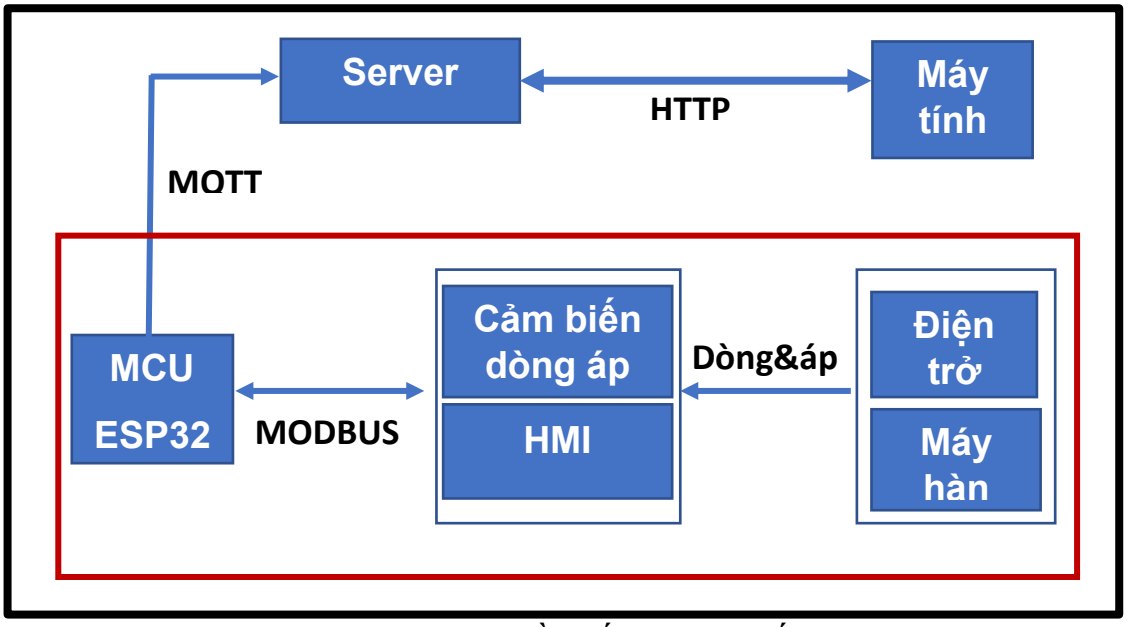
2.4 Kết luận chương 2

Chương 2 nói về việc sử dụng các thiết bị, các phần mềm và các công cụ đã dùng trong quá trình hoàn thành sản phẩm và các kiến trúc tổng quan phục vụ cho việc giải bài toán được đặt ra. Cùng với đó, chương 2 giới thiệu cũng như cách sử dụng các công cụ để lập trình và debug chương trình vi điều khiển ESP32. Ngoài ra chương 2 cũng giới thiệu về các công cụ lập trình bên phía sever như flask, giao diện hiển thị cho người dùng như HMI, Grafana

CHƯƠNG 3: THIẾT KẾ HỆ THỐNG ĐO VÀ GIÁM SÁT HIỆU SUẤT MÁY HÀN

3.1 Sơ đồ khối của hệ thống

Hệ thống gồm có các khối chính như sau:



Hình 3-1: Sơ đồ khối của hệ thống

- Khối MCU: Bao gồm mạch ESP32 và các linh kiện liên quan, khối này có nhiệm vụ đọc dữ liệu từ cảm biến dòng áp, đồng thời hiển thị lên màn hình HMI

- Khối cảm biến dòng áp: Bao gồm mạch cảm biến dòng áp (PZEM – 17) có nhiệm vụ xử lý tín hiệu dòng áp và biến đổi chúng thành khung truyền Modbus

- Khối HMI: Gồm có màn hình HMI có chức năng hiển thị hiệu suất làm việc của máy, tình trạng dòng điện, điện áp của máy

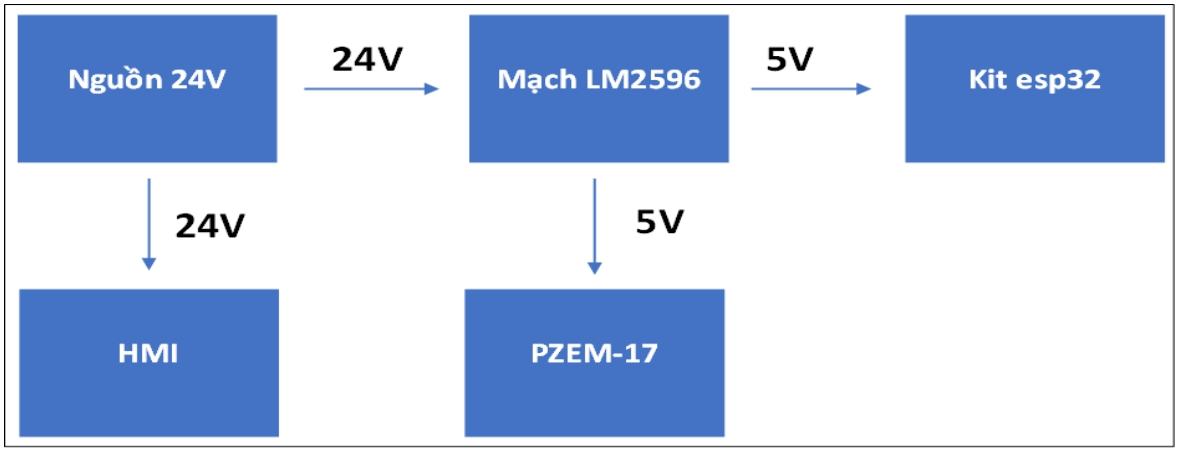
- Khối điện trở R_{shunt} và máy hàn: Bao gồm một điện trở R_{shunt} gián tiếp chuyển tín hiệu dòng về tín hiệu áp trên điện trở để đo đạc dòng.

- Server: có nhiệm vụ tiếp nhận thông tin từ ESP32, sau đó xử lý thông tin và thống kê, lưu trữ các thông tin đó vào cơ sở dữ liệu

- Máy tính: Được cài Grafana, giúp hiển thị và xem lại lịch sử hoạt động của máy hàn, lấy thông tin từ máy chủ và hiển thị ra cho người dùng

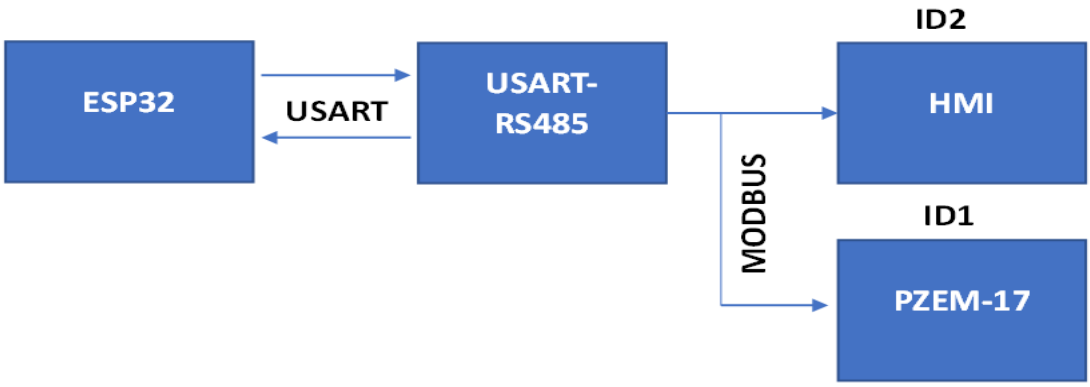
3.2 Kết nối phần cứng

Khối nguồn cấp



Hình 3-2: Sơ đồ nguồn cấp của hệ thống

Khối giao tiếp vi xử lý và thiết bị cấp trường

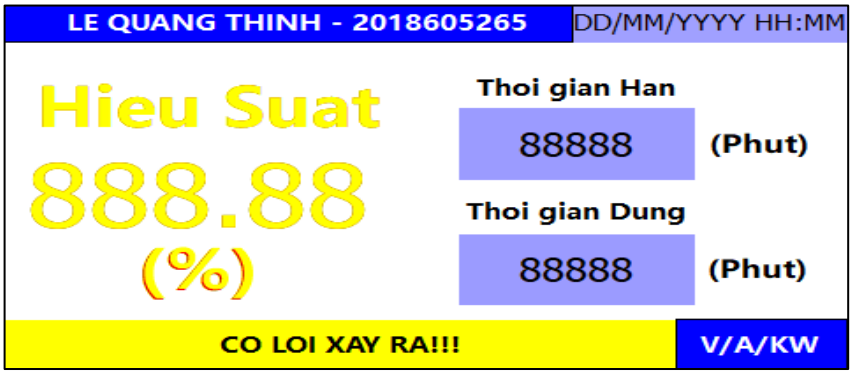


Hình 3-3: Sơ đồ ngoại vi của hệ thống

3.3 Thiết kế giao diện HMI

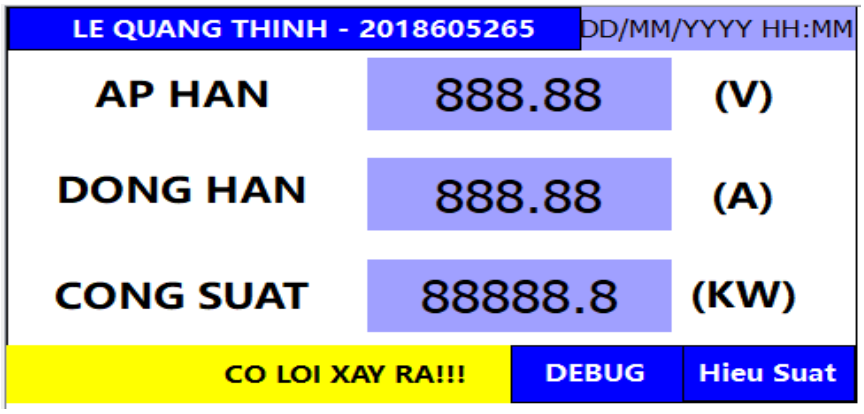
Sử dụng phần mềm do hãng cung cấp là Kinseal Studio 3.1 để thiết kế giao diện cho HMI.

Màn hình 01: Màn hình giám sát hiệu suất của máy hàn



Hình 3-4: Màn hình giám sát hiệu suất

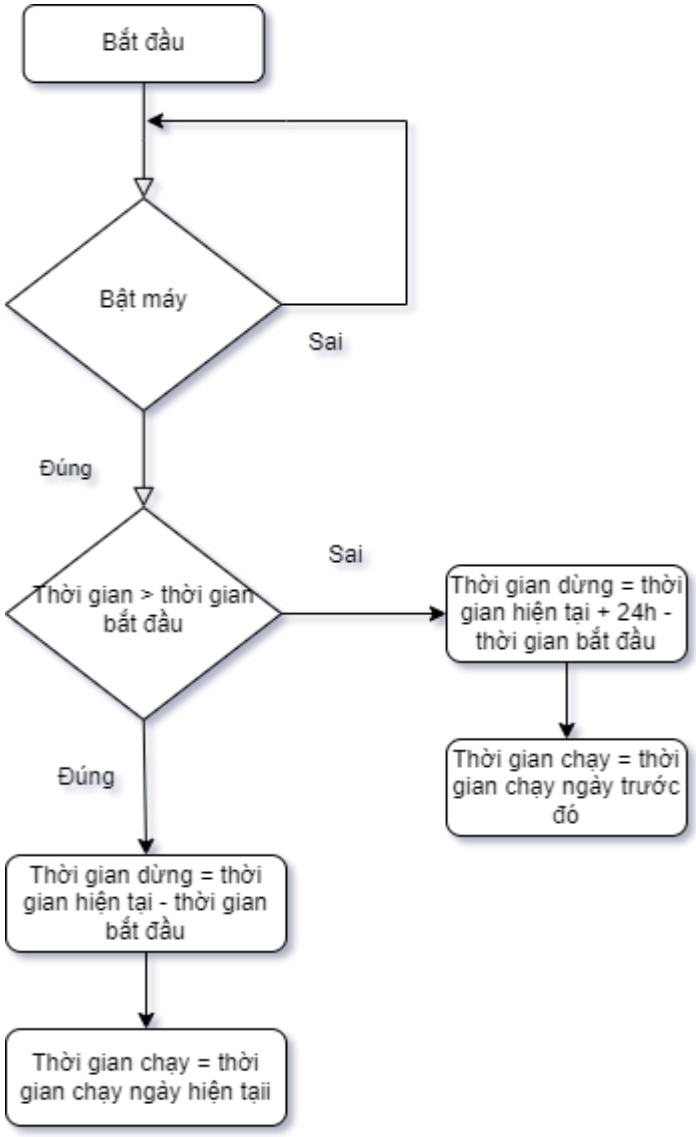
Màn hình 02: Màn hình giám sát dòng, áp và công suất của máy hàn



Hình 3-5: Màn hình giám sát dòng áp và công suất

3.4 Thiết kế firmware cho ESP32

Tính thời gian chạy và dừng của máy khi bật:

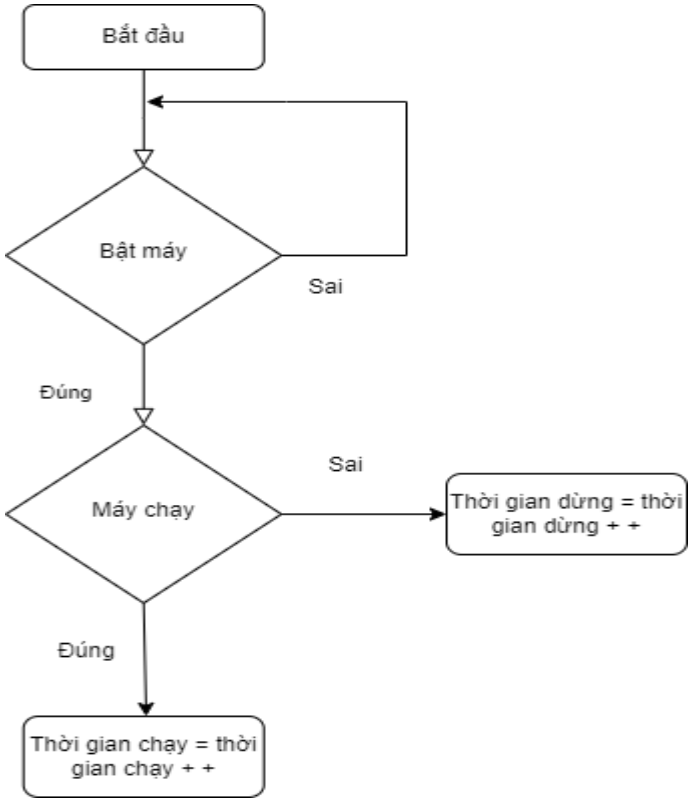


Hình 3-6: Flowchart thời gian chạy và thời gian dừng

Khi bắt đầu bật máy, nếu thời gian tại lúc bật máy lớn hơn thời gian bắt đầu đo hiệu suất, thì công thức tính thời gian dừng sẽ như sau:

- Thời gian dừng = thời gian hiện tại – thời gian bắt đầu
- Thời gian chạy = tời gian chạy của ngày hiện tại tính cả trường hợp bật tắt trong ngày
- Hiệu suất của máy = thời gian chạy/thời gian dừng

Tính thời gian chạy và dừng của máy khi đang hoạt động:



Hình 3-7: Flowchart sau khi bật máy

Khi bắt đầu bật máy, nếu thời gian tại lúc bật máy nhỏ hơn thời gian bắt đầu đo hiệu suất, thì công thức tính thời gian dừng sẽ như sau:

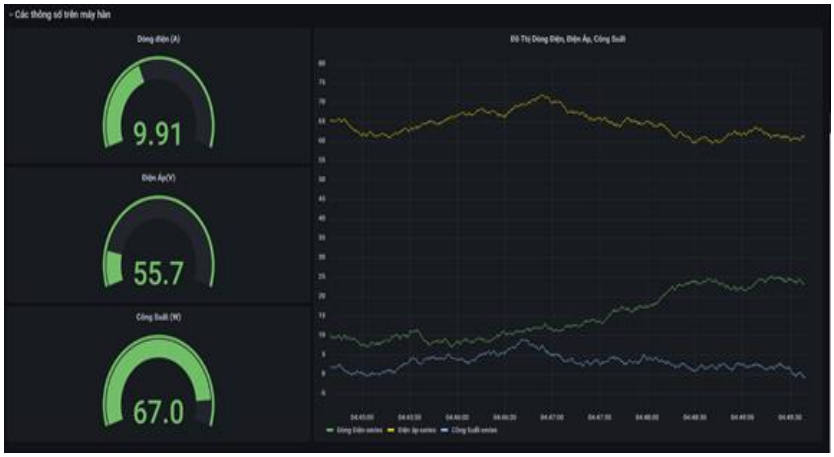
- Thời gian dừng = thời gian hiện tại + 24 giờ – thời gian bắt đầu
- Thời gian chạy = tời gian chạy của ngày trước đó tính cả trường hợp bật tắt ở thời điểm trước đó
- Hiệu suất của máy = thời gian chạy/thời gian dừng

Sau khi đã bật máy, việc hàn sẽ đưa dòng lớn lên một mức ngưỡng nào đó (threadhold), khi dòng hàn đạt tới một mức ngưỡng, thì thời gian chạy sẽ được cộng lại. Ngược lại, khi dòng hàn ở dưới ngưỡng, thì thời gian chạy sẽ

không được cộng nữa, thay vì đó sẽ cộng vào thời gian chạy

3.5 Thiết kế giao diện trên Web (Frontend)

Đồ thị hiển thị các thông số về dòng, áp và công suất trên máy hàn



Hình 3-8: Đồ thị hiển thị dòng, áp và công suất

Đồ thị hiển thị các thông số về hiệu suất trên máy hàn

Hình 3-9: Đồ thị hiển thị các thông số về hiệu suất

3.6 Một số hình ảnh thực tế của sản phẩm

Hình 3-10: Màn HMI dòng áp trên sản phẩm

Hình 3-11: Màn HMI hiệu suất trên sản phẩm

3.7 So sánh với đồng hồ đã hiệu chỉnh

Bảng 3-1: So sánh thông số hệ thống với thiết bị đã hiệu chỉnh

STT	Dòng điện	Dòng điện hiệu chỉnh	Điện áp	Điện áp hiệu chỉnh
1	1.55	1.57	10.4	10.2
2	1.63	1.65	11.4	11.7
3	1.72	1.71	12.4	12.2
4	1.81	1.82	13.4	13.9
5	2.5	2.6	14.4	14.3
6	4.0	4.1	16.4	16.8
7	3.0	3.2	16.4	16.1
8	3.5	3.7	22.6	22.8
9	2.1	2.2	23.9	24.0
10	3.0	2.9	24.8	25.1

3.8 Kết luận chương 3.

Về phần thiết kế mô hình:

- Mạch giao tiếp với máy hàn – là một thiết bị có dòng và áp cao. Việc thiết kế mạch thực tế phải tính tới các phần cách ly để không bị nhiễu tới các khối khác, đặc biệt là khối vi điều khiển

- Mạch chủ yếu lắp trong các máy hàn công nghiệp, vì vậy mạch phải được thiết kế để chống chịu trong môi trường công nghiệp khắc nhiệt như: Độ ẩm cao, có nhiều xung gây nhiễu …
- Mạch giao tiếp với nhau qua Wifi, vì vậy cần phải thiết kế anten và đặt đúng vị trí trên mạch và ngoài mô hình.
- Anten của mạch nên chọn loại PCB hoặc loại thiết kế bên ngoài nếu hộp kín, phải đảm bảo đúng tần số hoạt động của các giao tiếp, nếu không đúng sẽ gây ra việc truyền nhận bị mất dữ liệu hoặc truyền nhận được với khoảng cách ngắn

Về phần lập trình:

- Việc lập trình vi điều khiển yêu cầu người lập trình phải nắm rõ các kiến thức về ngôn ngữ lập trình C, kiến thức về các ngoại vi cơ bản của vi điều khiển, xử lý dữ liệu để đưa ra tín hiệu điều khiển…
- Việc lập trình với Sever yêu cầu người lập trình phải nắm rõ các kiến thức về Python, về công cụ mình đang sử dụng. Ngoài ra việc sử dụng thành thạo các công cụ debug, các công cụ kiểm soát và lập trình rất quan trọng để debug trong quá trình sảy ra lỗi
- Việc lập trình và đưa trang web lên websever yêu cầu người lập trình phải nắm vững các kiến thức về web cơ bản, xử lý sự kiện trên websever, thao tác với giao diện điều khiển …

Về phần thiết kế HMI:

- Có kỹ năng đọc các tài liệu của các hãng HMI bằng tiếng anh
- Thiết kế giao diện yêu cầu người thiết kế phải hiểu rõ cách hoạt động của máy móc và hệ thống
- Thiết kế yêu cầu có sự trực quan, dễ dàng thao tác sử dụng, có màn hình debug để xem các trạng thái của thiết bị, các phần nhỏ bên trong

KẾT LUẬN

Qua quá trình thực hiện làm đồ án, em đã trình bày các cơ sở lý thuyết liên quan và chạy thành công hệ thống "Thiết kế hệ thống đo và giám sát hiệu suất máy hàn". Tuy thời gian làm đồ án thực sự không quá dài nhưng được sự giúp đỡ tận tình của TS. Hà Thị Kim Duyên cùng với sự nỗ lực và cố gắng của bản thân, sự chỉ bảo của các Thầy Cô trong khoa Điện tử em đã hoàn thành đề tài theo yêu cầu và đúng thời gian quy định với những nội dung sau:

- Nghiên cứu và tìm hiểu về các loại cảm biến đo dòng áp trên thị trường, đặc biệt là cảm biến đo dòng áp PZEM-17
- Nghiên cứu và tìm hiểu về vi điều khiển ESP32, các ngoại vi và lập trình ESP32 dựa trên Arduino Framework
- Tìm hiểu về các chuẩn truyền thông có dây trong công nghiệp như Modbus, tìm hiểu về chuẩn truyền thông không dây như WiFi

Hướng phát triển đề tài:

- ✓ Tạo giao diện phong phú và đa dạng hơn
- ✓ Giảm độ trễ khi đo và giám sát
- ✓ Phát triển đa nền tảng (Android, IOS…)
- ✓ Điều khiển và giám sát thêm các thiết bị khác

Thông qua quá trình làm đồ án, em đã được vận dụng những kiến thức chuyên ngành trong 4 năm học. Qua đó đã giúp cho em rèn luyện được kỹ năng, cách tiếp cận với các vấn đề, các bài toán thực tế phức tạp tại các doanh nghiệp, nhà máy khi ra trường làm việc.

Việc xây dựng mô hình đã đáp ứng được yêu cầu đặt ra, tuy nhiên do trình độ và kinh nghiệm thực tiễn còn hạn chế nên không thể tránh khỏi sai sót và thiếu hoàn chỉnh. Rất mong được đón nhận sự đóng góp ý kiến từ thầy cô và các bạn.

Em xin chân thành cảm ơn!

TÀI LIỆU THAM KHẢO

[1] Grokhotkov, ESP8266 Arduino Core Documentation, Oxford University, 2010.

[2] L. V. Đại, Công Nghệ Internet Of Things, Nhà Xuất Bản Giáo Dục Việt Nam., 2010.

[3] N. V. N. Đỗ Xuân Thụ, Kỹ Thuật Điện Tử, Nhà Xuất Bản Giáo Dục Việt Nam, 2007

[4] "https://www.arduino.cc," [Online]. Available: https://www.arduino.cc/en/software.

[5] "https://commons.wikimedia.org/," [Online]. Available: https://commons.wikimedia.org/wiki/File:Visual_Studio_Code_1.35_icon.svg

[6] "https://cedalo.com," [Online]. Available: https://cedalo.com/guide-to-mqtt/

[7] "https://ifactory.com.vn," [Online]. Available: https://ifactory.com.vn/iot-trong-cong-nghiep-dang-thay-doi-nhu-the-nao/.

[8] "https://cambiendoapsuat.vn," [Online]. Available: https://cambiendoapsuat.vn/iot-la-gi-ung-dung-iot-trong-cong-nghiep/.

[9] "https://www.softwaretestinghelp.com," [Online]. Available: https://www.softwaretestinghelp.com/iot-devices/.

[10] "https://components101.com/," [Online]. Available: https://components101.com/switches/5v-relay-pinout-working-datasheet.

[11] "https://www.alldatasheet.com," [Online]. Available: https://www.alldatasheet.com/datasheet-pdf/pdf/30083/TOSHIBA/2SC1815.html.

[12] "https://www.alldatasheet.com," [Online]. Available: https://www.alldatasheet.com/view.jsp?Searchword=Pc817.

[13] "https://www.espressif.com," [Online]. Available: https://www.espressif.com/en/products/socs/esp8266.

[14] "https://www.digikey.bg," [Online]. Available:

https://www.digikey.bg/product-detail/en/mornsun-america-llc/B0505S-1WR3/2725-B0505S-1WR3-ND/13168098.

[15] "https://datasheet.lcsc.com," [Online]. Available: https://datasheet.lcsc.com/szlcsc/1912111437_HI-LINK-HLK-5M05_C209907.pdf.

PHỤ LỤC

Code phầm firmware hệ thống:

```
#ifdef ESP32
#ifdef USE_RM
#define XDRV_100          100
/*****************USER DEFINE START*********************/
#define REG_VALUE_TO_HMI_ADDR 0
//***data register for voltage, current and power
/*length of REG_VAP_ADDR 6
*register 20+21: voltage (v)
*register 22+23: current (A);
*register 24+25: power (W);
*/
#define REG_VAP_ADDR 20
//***data register for start time of user enter in HMI
/*length of REG_TIME_START_ADDR 2
*register 40: start time in hour
*register 41: start time in minute
*/
#define REG_TIME_START_ADDR 40
//***telegram read button status in screen 3
#define TELEGRAM_READ_BUTTON 3
#define TELEGRAM_WRITE_BUTTON 4
//*define threashold voltage and current*/
#define RM_VOLTAGE_THREASHOLD 2000
#define RM_CURRENT_THREASHOLD 6000
/*define Slave ID of HMI*/
#define HMI_SLAVE_ID 3
#define SENSOR_SLAVE_ID 1
```

```
/*define modbus parameter*/
#define MODBUS_BAURATE 9600
#define MODBUS_PIN_RX_A 16
#define MODBUS_PIN_TX_B 17
#define MODBUS_STOP_BIT 2

/*****************USER DEFINE END*********************/




/*****************USER VARIABLE START*********************/
uint16_t dataOverWrite = 0;

//variable for day process
uint16_t dayTemp;
uint16_t regRunningTimeDay = 3000+(dayTemp*3)-3;
bool dataButton = false;
uint32_t startTimeSec;
uint16_t machineStatus;
uint16_t pageCurrentHMI;
bool isGetTimeFromSoftWare = false; //for command
uint32_t upTimePrevious;
////time start
uint32_t runTimePrevious;
uint32_t downTimePrevious;

uint32_t runningTimeTemporary;
```

```
uint32_t downingTimeTemporary;

uint32_t lastRunTime;
uint32_t lastDownTime;
uint32_t partialRunTime;
uint32_t partialDownTime;
////time end
uint32_t totalTimeSec = 0;
/*****************USER VARIABLE END*******************/
/*****************USER ARRAY START*******************/
uint16_t vapx100[4];
uint16_t dataSensor[8];
uint16_t readDataHMI[3];
/*data save to HMI
dataSaveToHMI[0]: running time
dataSaveToHMI[1]: avaibility
dataSaveToHMI[2]: power
*/
uint16_t dataSaveToHMI[3];
/*get time from HMI
HMITimeNow[0] is HMI second
HMITimeNow[1] is HMI minute
HMITimeNow[2] is HMI hour
HMITimeNow[3] is HMI day
HMITimeNow[4] is HMI month
HMITimeNow[5] is HMI year
HMITimeNow[6] is HMI weak
*/
uint16_t HMITimeNow[7];
```

```
/*data to write availability, running and downing
RMValueHMI[1]: running time
RMValueHMI[2]: downing time
RMValueHMI[3]+RMValueHMI[4]: Avalilibity
*/
uint16_t RMValueHMI[9];
/*****************USER ARRAY END*********************/



/****************DEFINE STRUCT START*****************/

struct TimeNow
{
  uint8_t       second;
  uint8_t       minute;
  uint8_t       hour;
  uint8_t       day_of_month;
  uint8_t       month;
  uint16_t      day_of_year;
  uint16_t      year;
};
TimeNow timeNow;

struct OEEParameter
{
  uint32_t runningTime;
  uint32_t downingTime;
```

```
  float efficiencyPercent;
};
OEEParameter oeeParameter;

struct ModbusTelegram
{
  uint8_t u8id;          /*!< Slave address between 1 and 247. 0 means broadcast
*/
  uint8_t u8fct;         /*!< Function code: 1, 2, 3, 4, 5, 6, 15 or 16 */
  uint16_t u16RegAdd;    /*!< Address of the first register to access at slave/s
*/
  uint16_t u16CoilsNo;   /*!< Number of coils or registers to access */
  uint16_t *au16reg;     /*!< Pointer to memory image in master */
  bool *abreg;           /*!< Pointer to memory image in master */
};
ModbusTelegram telegram[MODBUS_MASTER_TELEGRAM];

/***************DEFINE STRUCT END***************/

/***************CREATE QUEUE AND SEMAPHORE
START****************/
xSemaphoreHandle xMBSyncMutex;
xQueueHandle xQueue;
/***************CREATE QUEUE AND SEMAPHORE
END****************/

/**********************USER FUNCTION
START***************************/
//transfer bcd to dec
```

```
uint8_t bcd2dec(uint8_t n);

//get next day and previous day
uint16_t getNextDay(uint16_t d, uint16_t m, uint16_t y);
uint16_t getPreviousDay(uint16_t d,uint16_t m, uint16_t y);

//get time now
void getTimeNow(TimeNow *time);

//calculate running time
void calculateTime(OEEParameter *oeeParameter);

//calculate OEE parameter
void calculateOEE(TimeNow time, uint16_t startTime, OEEParameter
*oeeParameter);

//check machine status
void checkMachineStatus();

//HMI database
void hmiSaveData(uint16_t day);
void hmiDeleteData(uint16_t day);
void hmiGetRunningTime(uint16_t day);

//RM behavior
void rmSaveData(TimeNow time, uint16_t startTime);
void rmGetRunningTime(TimeNow time, uint16_t startTime);

//reset RM
```

```
void resetRM();
void calculateTime(TimeNow *time, uint32_t timeSecond);

/**********************USER                                FUNCTION
END**************************/

telegram[TELEGRAM_READ_DATA_SENSOR].u8id                =
SENSOR_SLAVE_ID; // slave address
telegram[TELEGRAM_READ_DATA_SENSOR].u8fct                =
Modbus::FC_READ_INPUT_REGS; // function code (this one is registers
read)
telegram[TELEGRAM_READ_DATA_SENSOR].u16RegAdd = 0; // start
address in slave
telegram[TELEGRAM_READ_DATA_SENSOR].u16CoilsNo = 8; // number
of elements (coils or registers) to read
telegram[TELEGRAM_READ_DATA_SENSOR].au16reg =  dataSensor; //
}

bool ResCallback(Modbus::ResultCode event, uint16_t transactionId, void*
data)
{
  err = event;
}
Modbus::ResultCode MBTaskSync(ModbusTelegram telegramSync)
{
  xSemaphoreTake(xMBSyncMutex, portMAX_DELAY);
  if (MbMasterHMI->slave()){
    AddLog_P(LOG_LEVEL_DEBUG, PSTR("RM: Modbus transmitting"));
    xSemaphoreGive(xMBSyncMutex);
```

```
   return Modbus::EX_GENERAL_FAILURE;
  }
  switch( telegramSync.u8fct)
  {
    case Modbus::FC_READ_REGS:
      MbMasterHMI->readHreg(telegramSync.u8id, telegramSync.u16RegAdd,
telegramSync.au16reg, telegramSync.u16CoilsNo, ResCallback);
      break;
    case Modbus::FC_WRITE_REGS:
      MbMasterHMI->writeHreg(telegramSync.u8id,
telegramSync.u16RegAdd, telegramSync.au16reg, telegramSync.u16CoilsNo,
ResCallback);
      break;
    case Modbus::FC_READ_COILS:
      MbMasterHMI->readCoil(telegramSync.u8id,  telegramSync.u16RegAdd,
telegramSync.abreg, telegramSync.u16CoilsNo, ResCallback);
      break;
    case Modbus::FC_WRITE_COIL:
      MbMasterHMI->writeCoil(telegramSync.u8id,
telegramSync.u16RegAdd,  telegramSync.abreg,  telegramSync.u16CoilsNo,
ResCallback);
      break;
    case Modbus::FC_READ_INPUT_REGS:
      MbMasterHMI->readIreg(telegramSync.u8id,  telegramSync.u16RegAdd,
telegramSync.au16reg, telegramSync.u16CoilsNo, ResCallback);
      break;
    default:
      AddLog_P(LOG_LEVEL_DEBUG, PSTR("RM: Invalid Modbus Funtion
Code"));
```

```
    return Modbus::EX_ILLEGAL_FUNCTION;
   break;
  }
  while (MbMasterHMI->slave()) {
   MbMasterHMI->task();
   vTaskDelay(1/ portTICK_PERIOD_MS);
  }
  Modbus::ResultCode res = err;
  xSemaphoreGive(xMBSyncMutex);
  return res;
}

/**********MODBUS INIT END**********/


/**********************TASK
START*******************************/

/**************MODBUS SEND QUEUE TASK
START********************/
void StartModsbusSendQueueTask(){
   // if (MI32.mode.connected) return;
   // MI32.mode.runningScan = 1;
   xTaskCreatePinnedToCore(
   ModbusSendQueueTask,    /* Function to implement the task */
   "ModbusSendQueueTask",  /* Name of the task */
   4096,            /* Stack size in words */
   NULL,             /* Task input parameter */
   0,               /* Priority of the task */
```

```
    NULL,            /* Task handle. */
    0);              /* Core where the task should run */
}
void ModbusSendQueueTask(void *pvParameters)
{
  ModbusTelegram telegramQueue;
  for(;;){

    //receive from queue
    xQueueReceive(xQueue, &telegramQueue, portMAX_DELAY);
    // AddLog_P(LOG_LEVEL_NONE, PSTR("RM: queue available is %d"),
uxQueueSpacesAvailable(xQueue));
    Modbus::ResultCode MBResult;

    //send data from queue to modbus
    MBResult = MBTaskSync(telegramQueue);

    //check modbus send done success or failure
    if(MBResult != Modbus::EX_SUCCESS)
    {
      AddLog_P(LOG_LEVEL_NONE, PSTR("RM: Modbus master task is not
success"));
    }
    vTaskDelay(200/portTICK_PERIOD_MS);
  }
  vTaskDelete( NULL );
}
/**************MODBUS            SEND            QUEUE            TASK
END********************/
```

```
/**************TASK SEND QUEUE TASK START********************/
void StartTaskSendToQueue(){
   xTaskCreatePinnedToCore(
   taskSendToQueue,    /* Function to implement the task */
   "TaskSendToQueue",  /* Name of the task */
   4096,            /* Stack size in words */
   NULL,             /* Task input parameter */
   0,               /* Priority of the task */
   NULL,             /* Task handle. */
   0);              /* Core where the task should run */
}
void taskSendToQueue(void *pvParameters)
{
  for(;;)
  {
   // AddLog_P(LOG_LEVEL_NONE, PSTR("RM: voltage is %d"), dataSensor[0]);
   // AddLog_P(LOG_LEVEL_NONE, PSTR("RM: current is %d"), dataSensor[1]);
   // AddLog_P(LOG_LEVEL_NONE, PSTR("RM: power value low %d"), dataSensor[2]);
   // AddLog_P(LOG_LEVEL_NONE, PSTR("RM: power value high %d"), dataSensor[3]);
   // AddLog_P(LOG_LEVEL_NONE, PSTR("RM: Energy low is %d"), dataSensor[4]);
   // AddLog_P(LOG_LEVEL_NONE, PSTR("RM: Energy high is: %d"), dataSensor[5]);
```

```
   xQueueSendToBack(xQueue,
&telegram[TELEGRAM_READ_DATA_SENSOR], ( TickType_t ) 20);
     /**********************check machine status
start**********************/
   totalTimeSec = timeNow.hour*3600 + timeNow.minute*60 +
timeNow.second + (TasmotaGlobal.uptime - upTimePrevious);
   checkMachineStatus();
  /**********************check machine status end**********************/
   vTaskDelay(1000/portTICK_PERIOD_MS);
  }
}
/***************TASK SEND QUEUE TASK END********************/

/***************READ PAGE CURRENT START********************/
void startReadPageCurrentTask()
{
 xTaskCreatePinnedToCore(
 readPageCurrentTask,    /* Function to implement the task */
 "ReadPageCurrentTask",  /* Name of the task */
 4096,            /* Stack size in words */
 NULL,             /* Task input parameter */
 0,               /* Priority of the task */
 NULL,             /* Task handle. */
 0);
}
void readPageCurrentTask(void *pvParameters)
{
 bool isPageCurrentEqual07 = false;
 for(;;)
```

```
  {
    //                                                xQueueSendToBack(xQueue,
&telegram[TELEGRAM_READ_PAGE_CURRENT], ( TickType_t ) 20);
    // AddLog_P(LOG_LEVEL_NONE, PSTR("RM: PAGE CURRENT IS
%d"), pageCurrentHMI);
    vTaskDelay(1000/portTICK_PERIOD_MS);
  }
}
/**************READ PAGE CURRENT END*******************/


/**************CALCULATE AND DISPLAY EFFICIENCY
START*******************/
void startCalculateAndDisplay()
{
  xTaskCreatePinnedToCore(
  calculateAndDisplay,    /* Function to implement the task */
  "CalculateAndDisplay",  /* Name of the task */
  4096,           /* Stack size in words */
  NULL,           /* Task input parameter */
  0,              /* Priority of the task */
  NULL,           /* Task handle. */
  0);
}
void calculateAndDisplay(void *pvParameters)
{
  for(;;)
  {
    /*******************Time start*****************************/
```

```
   // AddLog_P(LOG_LEVEL_NONE, PSTR("RM: TOTAL TIME START
%ld"), totalTimeSec);

    /*******************Calculate start**********************/
   calculateTime(&oeeParameter);

   calculateOEE(timeNow, startTimeSec, &oeeParameter);
   uint32_t                    efficiencyPercentX10000                    =
(uint32_t)(oeeParameter.efficiencyPercent * 10000);
   //value to write to screen 2 (OEE value)
   RMValueHMI[0] = efficiencyPercentX10000;
   RMValueHMI[1] = efficiencyPercentX10000 >> 16;
   RMValueHMI[2] = oeeParameter.runningTime / 60;
   RMValueHMI[3] = oeeParameter.downingTime / 60;

   //value to write to screen 2 (Sensor value)
   RMValueHMI[4] = dataSensor[0];
   RMValueHMI[5] = dataSensor[1];
   RMValueHMI[6] = dataSensor[2];
   RMValueHMI[7] = dataSensor[3]>>16;


   //machine status
   RMValueHMI[8] = machineStatus;

   // send OEE value , Sensor value and machine status to HMI
   xQueueSendToBack(xQueue,
&telegram[TELEGRAM_WRITE_TO_HMI], ( TickType_t ) 20);
```

```
    /*******************Calculate end*********************/

   vTaskDelay(500/portTICK_PERIOD_MS);
  }
}
/**************CALCULATE AND DISPLAY EFFICIENCY
END*******************/

/*****************SEND DATA TO MQTT
START**********************/
void startSendDataMQTTTask()
{
  xTaskCreatePinnedToCore(
  sendDataMQTTTask,    /* Function to implement the task */
  "SendDataMQTTTask",  /* Name of the task */
  4096,            /* Stack size in words */
  NULL,             /* Task input parameter */
  0,               /* Priority of the task */
  NULL,             /* Task handle. */
  0);
}
void sendDataMQTTTask(void *pvParameters)
{
  for(;;)
  {
    float vol = dataSensor[0]/100.0;
    float amp = dataSensor[1]/100.0;
    float pow = (dataSensor[2] + dataSensor[3] * 16)/10.0;
```

```
    Response_P(PSTR("{\"" "RM_DATA" "\":{"
                "\"Target\":0"
                ",\"Actual\":0"
                ",\"NG\":0"
                ",\"RunTime\":%d"
                ",\"HeldTime\":%d"
                ",\"MCStatus\":%d"
                ",\"A\":%f"
                ",\"P\":0"
                ",\"Q\":0"
                ",\"Vol\":%f"
                ",\"Amp\":%f"
                ",\"Pow\":%f"
                "}}"),
                  oeeParameter.runningTime,
                  oeeParameter.downingTime,
                  machineStatus,
                  oeeParameter.efficiencyPercent,
                  vol,
                  amp,
                  pow
                  );
    MqttPublishPrefixTopic_P(STAT, PSTR("RM_DATA"));
    vTaskDelay(5000/portTICK_PERIOD_MS);
  }
}
/*****************SEND DATA TO MQTT
END**********************/
```

```
/****************TASK SAVE DATA START*********************/
void startSaveDataTask()
{
  xTaskCreatePinnedToCore(
  saveDataTask,    /* Function to implement the task */
  "SaveDataTask",  /* Name of the task */
  4096,            /* Stack size in words */
  NULL,            /* Task input parameter */
  0,               /* Priority of the task */
  NULL,            /* Task handle. */
  0);
}
void saveDataTask(void *pvParameters)
{
  for(;;)
  {
    vTaskDelay(20000/portTICK_PERIOD_MS);
    rmSaveData(timeNow, startTimeSec);

  }
}
/****************TASK SAVE DATA END*********************/

/****************RM DEFAULT TASK START*********************/
void startRMDefauTask()
{
  xTaskCreatePinnedToCore(
```

```
  RMDefauTask,    /* Function to implement the task */
  "RMDefauTask",  /* Name of the task */
  4096,           /* Stack size in words */
  NULL,           /* Task input parameter */
  0,              /* Priority of the task */
  NULL,           /* Task handle. */
  0);
}
void RMDefauTask(void *pvParameters)
{
  if(TasmotaGlobal.uptime < 5) {
    AddLog_P(LOG_LEVEL_DEBUG, PSTR("RM: Starting"));
    vTaskDelay(3000/ portTICK_PERIOD_MS);
  }
  for(;;)
  {
    xQueue = xQueueCreate(30, sizeof(ModbusTelegram));
    xMBSyncMutex = xSemaphoreCreateMutex();

    //start task send data to mqtt
    startSendDataMQTTTask();
    //modbus init
    ModbusInit();
    modbusDataInit();

    StartModsbusSendQueueTask();

    StartTaskSendToQueue();
```

```
    startReadPageCurrentTask();

    getTimeNow(&timeNow);

    rmGetRunningTime(timeNow, startTimeSec);

    uint16_t nextDay = getNextDay((uint16_t)(timeNow.day_of_month),
(uint16_t)(timeNow.month), (uint16_t)(timeNow.year+2000-1970));
    hmiDeleteData(nextDay);

    startCalculateAndDisplay();

    startSaveDataTask();

    vTaskDelete( NULL );
  }

}
/*****************RM DEFAULT TASK END **********************/

/***********************TASK
END*******************************/
void RMInit()
{
  startRMDefauTask();

  //init variable
  startTimeSec = Settings.startTimeSecFlash;
}
```

```
void getTimeNow(TimeNow *time)
{
  // AddLog_P(LOG_LEVEL_NONE, PSTR("RM: get time start"));
  //send queue to back to wait get time HMI
  vTaskDelay(500/portTICK_PERIOD_MS);
  xQueueSendToBack(xQueue,
&telegram[TELEGRAM_READ_TIME_HMI], ( TickType_t ) 20);
  time->second = bcd2dec(HMITimeNow[0]);
  time->minute = bcd2dec(HMITimeNow[1]);
  time->hour = bcd2dec(HMITimeNow[2]);
  time->day_of_month = bcd2dec(HMITimeNow[3]);
  time->month = bcd2dec(HMITimeNow[4]);
  time->year = bcd2dec(HMITimeNow[5])+2000-1970;
  while(time->day_of_month <= 0)
  {
    xQueueSendToBack(xQueue,
&telegram[TELEGRAM_READ_TIME_HMI], ( TickType_t ) 20);
    vTaskDelay(500/portTICK_PERIOD_MS);
    time->second = bcd2dec(HMITimeNow[0]);
    time->minute = bcd2dec(HMITimeNow[1]);
    time->hour = bcd2dec(HMITimeNow[2]);
    time->day_of_month = bcd2dec(HMITimeNow[3]);
    time->month = bcd2dec(HMITimeNow[4]);
    time->year = bcd2dec(HMITimeNow[5])+2000-1970;
}
```