TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

**Trường Điện – Điện tử**

----- □ & □ -----



**Môn học: Hệ điều hành**

***Đề tài:* Tiếp kiệm năng lượng cho ung dụng hệ điều hành FreeRTOS và vi điều khiển ESP32 trong hệ thống quan trắc môi trường**

Giảng viên: TS. Hàn Huy Dũng

Nhóm sinh viên thực hiện:

|  |  |  |
| --- | --- | --- |
| STT | Họ và tên | MSSV |
| 1 | Lê Chí Tuyền | 20193191 |
| 2 | Lê Danh Huy | 20192910 |
| 3 | Trần Thanh Lâm | 20192959 |
| 4 | Đinh Công Bình | 20192711 |

**Hà Nội, tháng 8 năm 2023**

**LỜI NÓI ĐẦU**

Hiện nay, chất lượng không khí đang là vấn đề rất được quan tâm trên toàn thế giới. Việt Nam đang đứng trong tốp 10 các nước ô nhiễm không khí trên thế giới (do trung tâm nghiên cứu môi trường thuộc Đại học Yale và Columbia của Mỹ thực hiện báo cáo thường niên mang tên EPL). Tỷ lệ mắc các bệnh do ô nhiễm không khí ở nước ta ngày một tăng cao. Vì lý do đó, hệ thống AirSENSE ra đời, cung cấp những thiết bị, web – app quan trắc chất lượng không khí tại Hà Nội.

Dưới chỉ bảo rất nhiệt tình từ thầy Hàn Huy Dũng cùng các anh chị, các bạn trong SPARC Lab (Signal Processing and Radio Communication) – là phòng nghiên cứu của thầy Hàn Huy Dũng, em cùng nhóm AirSENSE đã thiết kế phiên bản thứ tư của thiết bị. Em xin chân thành gửi lời cảm ơn đến thầy Tạ Sơn Xuất, thầy Hàn Huy Dũng và các bạn trong lab đã giúp đỡ chúng em trong quá trình này

Trong quá trình làm đồ án, ngoài những thuận lợi thì em cũng gặp không ít khó khăn vì thiếu kỹ năng cũng như kinh nghiệm làm về các dự án thực tế, do vậy cũng không thể tránh khỏi được những thiếu sót. Em rất mong nhận được ý kiến đóng góp từ các thầy cô để có thể hoàn thiện bản thân hơn nữa.

**PHÂN CHIA CÔNG VIỆC**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Họ và tên** | **Thiết kế mạch điện** | **Viết nội dung MQTT** | **Tìm hiểu về quản lý Task** | **Lập trình đọc dữ liệu** | **Đo dữ liệu và viết báo cáo** | **Lắp ráp thiết bị thực nghiệm ngoài trời** | **Chỉnh sửa báo cáo** | **Khác** |
| Tuyền | X |  |  | X |  |  | X |  |
| Huy | X | X |  |  |  | X | X |  |
| Lâm |  | X | X | X | X |  | X |  |
| Bình |  |  |  |  | X | X | X |  |

MỤC LỤC

[MỤC LỤC 4](#_Toc143508566)

[DANH MỤC KÝ HIỆU VÀ CHỮ VIẾT TẮT 5](#_Toc143508567)

[DANH MỤC HÌNH VẼ 6](#_Toc143508568)

[DANH MỤC BẢNG BIỂU 7](#_Toc143508569)

[CHƯƠNG I. TỔNG QUAN 8](#_Toc143508570)

[**1. Giới thiệu** 8](#_Toc143508571)

[**1.1. Internet of Things (IOT)** 8](#_Toc143508572)

[**1.2. ESP32** 9](#_Toc143508573)

[**2. Cơ sở lý thuyết** 9](#_Toc143508574)

[**2.1. Message Queuing Telemetry Transport (MQTT)** 9](#_Toc143508575)

[**2.3. Khái quát về FreeRTOS** 11](#_Toc143508576)

[CHƯƠNG II. Hệ thống đo chất lượng không khí 13](#_Toc143508577)

[1. Hệ thống đo chất lượng không khí 13](#_Toc143508578)

[2. Tính toán Firmware 14](#_Toc143508579)

[1.1 Sơ đồ Các tệp tin trong hệ thống 17](#_Toc143508580)

[3. Mô hình hệ thống do đạc 18](#_Toc143508581)

[4. Thiết bị phần cứng 18](#_Toc143508582)

[3. Tiến hành đo đạc 20](#_Toc143508583)

[KẾT LUẬN 24](#_Toc143508584)

[TÀI LIỆU THAM KHẢO 25](#_Toc143508585)

# **DANH MỤC KÝ HIỆU VÀ CHỮ VIẾT TẮT**

|  |  |
| --- | --- |
| ASV4 | AirSENSE version 4 |
| IoT | Internet of Things |
| PM | particle mass |
|  |  |
|  |  |
|  |  |
|  |  |

# **DANH MỤC HÌNH VẼ**

[Hình 1 Kiến trúc MQTT Pub/Sub 9](file:///E:\Ca_nhan_cv\hoc_tap\HDH\BTL_HeDieuHanh_20222.docx#_Toc143507940)

[Hình 2 Sơ đồ của hệ thống. 13](#_Toc143507941)

[Hình 3 Danh sách nhiệm vụ và yêu cầu từng nhiệm vụ 14](#_Toc143507942)

[Hình 4 Danh sách các nhiệm vụ và các Queue chính 14](#_Toc143507943)

[Hình 5 Các Semaphore được sử dụng 15](#_Toc143507944)

[Hình 6 Các thành phần trong mã nguồn 16](#_Toc143507945)

[Hình 7 Hình mạch điện AirSENSE tiến hành thử nghiệm 18](#_Toc143507946)

[Hình 8 Mô hình kết nối của hệ thống. 19](#_Toc143507947)

[Hình 9 Đồ thị hiển thị mức tiêu thụ năng lượng khi dùng serial logging và không logging 20](#_Toc143507948)

[Hình 10 Dồ thị tiêu thụ năng lượng liên quan đến tần số hoạt động của esp32 21](#_Toc143507949)

[Hình 11 đồ thị tiêu thụ năng lượng của ESP32 trong mô hình phòng thí nghiệm. 21](#_Toc143507950)

[Hình 12 . So sánh mức độ tiêu hao năng lượng trước và sau khi áp dụng kịch bản tiết kiệm đối với 1 cụm 3 nút ESP32 trong 3 chu kỳ gửi dữ liệu 21](file:///E:\Ca_nhan_cv\hoc_tap\HDH\BTL_HeDieuHanh_20222.docx#_Toc143507951)

[Hình 13 đồ thị tiêu thụ năng lượng của ESP32 trong mô hình thực nghiệm. 22](file:///E:\Ca_nhan_cv\hoc_tap\HDH\BTL_HeDieuHanh_20222.docx#_Toc143507952)

# **DANH MỤC BẢNG BIỂU**

[Bảng 4‑1 Danh mục các nhiệm vụ trong source code 15](#_Toc143507743)

# **CHƯƠNG I. TỔNG QUAN**

**1. Giới thiệu**

**1.1. Internet of Things (IOT)**

- IOT là một công nghệ mới nổi trong đó truyền dữ liệu và truy cập thông tin giữa bất kỳ đối tượng hoặc các thiết bị trở nên dễ dàng hơn, do đó làm cho cuộc sống dễ dàng hơn về nhiều mặt. Công nghệ này đã phát triển theo cấp số nhân. Theo báo cáo của Cisco, số lượng thiết bị kết nối internet sẽ nhiều hơn gấp ba lần so với dân số toàn cầu. Sự gia tăng các thiết bị IoT như vậy sẽ tạo ra nhiều ứng dụng đa dạng như năng lượng thông minh, tòa nhà thông minh và theo dõi sức khỏe thiết bị. IoT không gì khác ngoài giao tiếp giữa các thiết bị trên toàn cầu. Điều này về cơ bản làm tăng khả năng kết nối và chia sẻ dữ liệu giữa các thiết bị và con người. Điều này dẫn đến nhiều vấn đề khiến khung IoT dễ bị tổn thương về khả năng mở rộng, bảo mật, quyền riêng tư, hiệu quả năng lượng , v.v. .

Mặc dù các công nghệ IoT luôn đi đầu trong nghiên cứu và phát triển , nhưng các công nghệ pin không phát triển nhanh để theo kịp các công nghệ mới nổi này. Ngoài ra, các yêu cầu bảo mật của các thiết bị IoT làm hạn chế mức tiêu thụ năng lượng, khiến việc tối ưu hóa trở nên cần thiết. Thật không may, hầu hết các lập trình viên và nhà phát triển đều thiếu kiến thức về sự phụ thuộc vào phần mềm/năng lượng, thất bại TRÊN thân thiện với năng lượng phát triển thuật toán. Các nhà nghiên cứu đang liên tục tìm cách cải thiện hiệu suất năng lượng của phần mềm chạy trên các hệ thống nhúng. Hiệu quả năng lượng có tác động lớn đến việc lựa chọn đúng MCU cho một ứng dụng cụ thể . Quản lý năng lượng là một chủ đề cho các hệ thống tĩnh và di động. Trong các hệ thống tĩnh, việc quản lý năng lượng tốt sẽ làm giảm hóa đơn tiền điện và nhiệt được tạo ra, và đối với các hệ thống di động sẽ mang lại tuổi thọ pin lâu hơn. Điện năng tiêu thụ của một MCU phụ thuộc vào điện áp và dòng điện hoạt động. Các kỹ thuật khác nhau có thể được thực hiện để giảm mức tiêu thụ điện năng, đặc biệt là trong các bộ vi điều khiển.

Các thiết bị IoT đã mang lại sự cải tiến để phát triển trong nhiều lĩnh vực, nhưng những những lợi ích có thể bị giới hạn bởi phần cứng và thiết kế điện tử. Có một danh sách nhỏ các ưu điểm và nhược điểm:

● Thuận lợi:

○ Cao tính di động.

○ Thấp Trị giá.

● Nhược điểm:

○ Giới hạn bộ nhớ

○ Giới hạn xử lý

○ Giới hạn năng lượng

Hầu hết các thiết bị IoT đều chạy bằng pin, vì vậy những thiết bị này bị hạn chế về pin. Hơn nữa, các thiết bị này trở nên vô dụng hoặc không thể hoạt động cho đến khi thực hiện thay thế pin, điều này có thể đại diện cho Một cao điều hành trị giá vì các iốt thiết bị. Để giảm chi phí, cần tối đa hóa việc sử dụng năng lượng và giảm việc thay thế pin càng nhiều càng tốt. khả thi.

**1.2. ESP32**

ESP32-WROOM-32 là một module đa dụng, được sử dụng rộng rãi trong thiết kế mạch PCB Wifi-Bluetooth, BLE được ứng dụng rất phổ biến cho nhiều ứng dụng về IoT. Phạm vi ứng dụng từ mạng sensor tiết kiệm năng lượng đến những ứng dụng với nhiều tác vụ phức tạp, như mã hóa âm thanh, âm nhạc trực tuyến đến giải mã MP3. Lõi của module là dòng chip nhúng ESP32-D0WDQ6, được thiết kế với khả năng mở rộng và tùy biến cao. Có đến 2 lõi CPU độc lập nên chip này có thể điều chỉnh tần số clock của CPU từ 80 đến 240 MHz. Người lập trình có thể tắt CPU để sử dụng bộ đồng xử lý công suất thấp, giúp theo dõi sự biến đổi hoặc vượt ngưỡng của các ngoại vi nhờ tích hợp bộ ngoại vi khá phong phú từ cảm biến điện dung, cảm biến Hall, SD card, Ethernet, SPI tốc độ cao, đến UART, I2S hay I2C.

Việc tích hợp cả Bluetooth, BLE và Wifi đảm bảo cho ESP32- WROOM-32 có khả năng đáp ứng nhiều loại ứng dụng khác nhau, dù module đó sử dụng ngoại vi, thiết bị nào (wifi cho phép kết nối rộng rãi về mặt vật lý ra Internet qua Wifi router, trong khi Bluetooth cho phép thuận tiện kết nối với smartphone...). Ở chế độ ngủ, chíp ESP32 tiêu thụ dòng dưới 5 A, phù hợp với những thiết kế mạch dùng pin hay thiết bị di động. Tốc độ truyền thông tin lên đến 150 Mbps, công suất tín hiệu khoảng 20 dBm trên anten cho phép phạm vi tín hiệu xa… cho thấy module phù hợp với việc thiết kế các hệ thống điện tử, tự động hóa, đòi hỏi phạm vi hoạt động rộng, tiết kiệm năng lượng, cũng như khả năng kết nối đa dạng.

Do đó, kịch bản chung về ghi dữ liệu được so sánh với hiệu suất và mức tiêu thụ năng lượng khi giao tiếp với các thẻ SD và tần số CPU khác nhau thông qua bus SPI và SD. Kết quả thử nghiệm của chúng tôi cho thấy các thành phần ngoại vi (chẳng hạn như bộ điều chỉnh điện áp) có tác động lớn đến mức tiêu thụ điện năng của bộ vi điều khiển ESP32, đặc biệt là ở chế độ ngủ. Để ghi dữ liệu, tốc độ xung nhịp cao hơn kết hợp với thẻ SD chất lượng cao và sử dụng bus SD ở chế độ 4 bit dẫn đến pin thấp nhất phóng điện

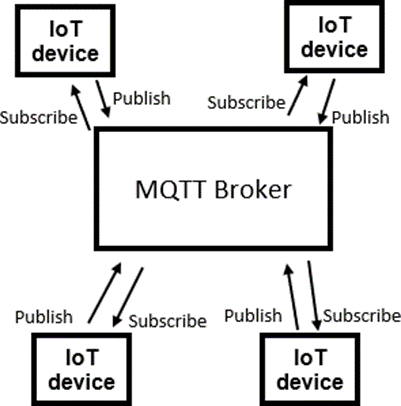
**2. Cơ sở lý thuyết**

**2.1. Message Queuing Telemetry Transport (MQTT)**

MQTT (Message Queuing Telemetry Transport) là một giao thức truyền thông cực nhẹ và đơn giản, thường được sử dụng trong các ứng dụng IoT (Internet of Things) để truyền tải dữ liệu giữa các thiết bị.

MQTT hỗ trợ ba mức chất lượng (Quality of Service - QoS) khác nhau để đảm bảo việc truyền tải tin nhắn giữa nhà cung cấp dịch vụ MQTT (broker) và các thiết bị MQTT. Các mức chất lượng bao gồm:

* QoS 0 (At most once): Đây là mức chất lượng thấp nhất. Tin nhắn được gửi đi mà không cần xác nhận hoặc đảm bảo rằng nó đã được nhận bởi máy chủ MQTT. Mức này đảm bảo rằng tin nhắn sẽ không bị gặp trở ngại trong quá trình truyền tải, nhưng không đảm bảo rằng nó đã đến được đích.
* QoS 1 (At least once): Với mức chất lượng này, tin nhắn được gửi đi và máy chủ MQTT sẽ gửi lại xác nhận (PUBACK) cho thiết bị sau khi nhận được tin nhắn. Nếu thiết bị không nhận được xác nhận, nó sẽ gửi lại tin nhắn. Điều này đảm bảo rằng tin nhắn sẽ được gửi đi ít nhất một lần, nhưng có thể có các bản sao tin nhắn trong quá trình truyền tải.
* QoS 2 (Exactly once): Đây là mức chất lượng cao nhất. Tin nhắn được gửi đi và máy chủ MQTT sẽ xác nhận (PUBREC) rằng nó đã nhận được tin nhắn. Sau khi nhận được xác nhận, thiết bị gửi lại xác nhận cuối cùng (PUBCOMP) cho máy chủ MQTT. Quá trình này đảm bảo rằng tin nhắn chỉ được gửi đi một lần duy nhất và đảm bảo tính toàn vẹn của tin nhắn trong quá trình truyền tải.

**2.2. Tiêu thụ năng lượng của vi điều khiển**

Hình 1 Kiến trúc MQTT Pub/Sub

Để hiểu cơ bản về mức tiêu thụ điện năng của vi điều khiển, công thức sau đây minh họa các yếu tố góp phần vào mức tiêu thụ điện năng của vi điều khiển [8]:

Pd = fCU2 (1)

P d — Phần động của công suất tiêu thụ

f — Tần số xung nhịp CPU

C — Tổng điện dung của các bóng bán dẫn hiệu ứng trường (FET) trong mạch

U — Điện áp hoạt động

Mối quan hệ này cho thấy mức tiêu thụ năng lượng có thể giảm bằng cách giảm tần số xung nhịp f của CPU , điện dung C hoặc bằng cách giảm điện áp hoạt động U [8].

Vì mức tiêu thụ điện năng tỷ lệ thuận với bình phương điện áp làm việc U, rõ ràng ban đầu nên giảm nó càng nhiều càng tốt. Về mặt lịch sử, các bộ vi xử lý đời đầu thường chạy trên điện áp nguồn 5 V. Kể từ đó, điện áp liên tục bị giảm vì lý do đó. Ngược lại, tần số xung nhịp tối đa đã tăng lên trong những năm qua để đạt được hiệu suất tính toán cao hơn. Tuy nhiên, điều này cũng đã dẫn đến tăng tiêu thụ điện năng. Ngày nay, chiến lược là tối đa hóa khả năng tần số xung nhịp của vi điều khiển trong khi chạy ở tần số xung nhịp thấp hơn đáng kể.

Các tổng cộng điện dung là các Tổng của các điện dung của các cá nhân tranzito hiệu ứng trường (FET). Quá hạn to lớn thu nhỏ, các FET ' cá nhân điện dung đã trở thành nhỏ hơn, Nhưng các con số của FET mỗi bộ vi xử lý \_ cốt lõi tiếp tục to lớn phát triển. tổng số điện dung của Một được cho hệ thống, Vì vậy, Có thể chỉ một be giảm qua chuyển mạch tắt các bộ phận riêng lẻ của bộ xử lý.

**2.3. Khái quát về FreeRTOS**

FreeRTOS là một hệ điều hành thời gian thực mã nguồn mở được phát triển bởi Richard Barry. Nó được thiết kế để chạy trên nhiều kiến trúc bộ vi xử lý khác nhau và là một trong những RTOS phổ biến nhất trong việc phát triển các ứng dụng nhúng. Dưới đây là một số cơ sở lý thuyết chính về FreeRTOS:

1. Task (Tác vụ): FreeRTOS quản lý công việc thông qua các tác vụ (task). Mỗi tác vụ là một luồng thực thi độc lập, có thể thực hiện một công việc cụ thể. Khi tạo một tác vụ, bạn xác định hàm thực thi và ưu tiên của nó. Các tác vụ thực thi song song và chia sẻ tài nguyên, nhưng FreeRTOS cung cấp các cơ chế đồng bộ hóa như semaphore và mutex để quản lý việc truy cập vào tài nguyên chung.
2. Scheduler (Lập lịch): FreeRTOS có một lập lịch nhiệm vụ có thể lựa chọn và chuyển đổi giữa các tác vụ dựa trên ưu tiên của chúng. Nhiệm vụ với ưu tiên cao hơn được lựa chọn thực hiện trước, đảm bảo tính thời gian thực trong hệ thống.
3. Context Switch (Chuyển đổi ngữ cảnh): Khi lập lịch chuyển đổi giữa các tác vụ, FreeRTOS thực hiện chuyển đổi ngữ cảnh (context switch) để lưu trạng thái của tác vụ hiện tại và chuyển sang tác vụ khác. Chuyển đổi ngữ cảnh đảm bảo rằng các tác vụ có thể chạy song song mà không ảnh hưởng lẫn nhau.
4. Semaphore và Mutex: FreeRTOS cung cấp các cơ chế đồng bộ hóa như semaphore và mutex để quản lý việc truy cập vào tài nguyên chung giữa các tác vụ. Semaphore được sử dụng để đếm số lần truy cập được thực hiện và mutex được sử dụng để bảo vệ tài nguyên chia sẻ khỏi xung đột.
5. Queue (Hàng đợi): Hàng đợi trong FreeRTOS được sử dụng để truyền dữ liệu giữa các tác vụ một cách an toàn. Dữ liệu được đưa vào hàng đợi bởi một tác vụ và được lấy ra bởi tác vụ khác, đảm bảo tính toàn vẹn dữ liệu và tránh xung đột.
6. Tick (Tín hiệu thời gian): FreeRTOS sử dụng tín hiệu thời gian (tick) để theo dõi thời gian chạy của các tác vụ. Một số phiên bản FreeRTOS hỗ trợ multiple tick rate, cho phép cấu hình độ phân giải thời gian phù hợp với yêu cầu ứng dụng.

Tóm lại, FreeRTOS là một hệ điều hành nhúng linh hoạt và mạnh mẽ, cung cấp các cơ chế quan trọng để quản lý và điều phối các tác vụ trong hệ thống nhúng thời gian thực. Nó đã được sử dụng rộng rãi trong các ứng dụng điện tử tiêu thụ, ô tô thông minh, thiết bị y tế, và nhiều lĩnh vực khác.

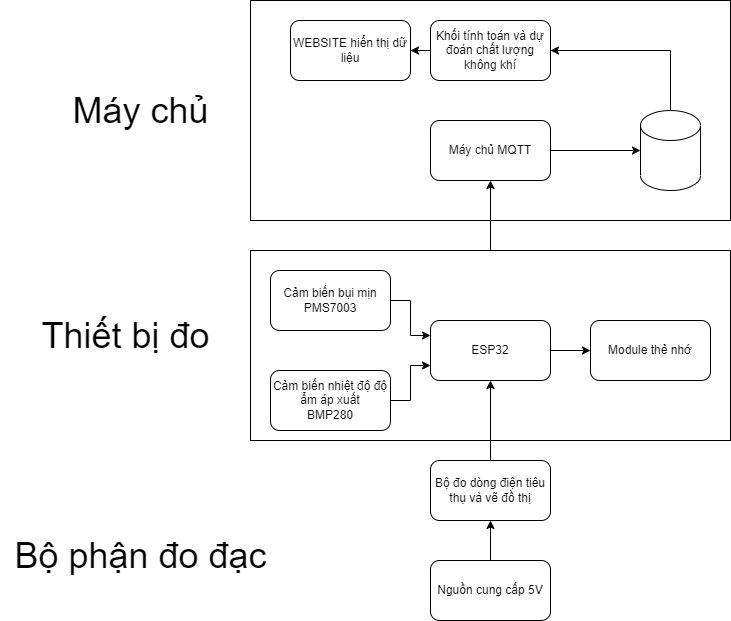
**CHƯƠNG II. Hệ thống đo chất lượng không khí**

1. **Hệ thống đo chất lượng không khí**

Trên thị trường hiện nay, có một số sản phẩm kết hợp công nghệ IoT để đo chất lượng không khí như AirVisual, Pam Air và GAIA Air. Tuy nhiên, trong số đó, hệ thống của AirVisual được xem là tiên tiến nhất. Hệ thống này có khả năng giám sát các yếu tố ảnh hưởng đến chất lượng không khí như nhiệt độ, độ ẩm, cường độ sáng, độ bụi mịn PM2.5 và cung cấp thông báo cho người dùng. Ngoài ra, nó cũng có khả năng dự đoán tình trạng chất lượng không khí và đưa ra gợi ý cho người dùng về cách cải thiện. Tuy nhiên, hệ thống này vẫn chưa có khả năng tự động phân tích dữ liệu để đưa ra chiến lược điều khiển nhằm tiết kiệm tài nguyên hệ thống.

Từ việc khảo sát, có thể nhận thấy rằng một hệ thống đo chất lượng không khí dựa trên công nghệ IoT và trí tuệ nhân tạo cần đáp ứng những yêu cầu chung sau: xác thực thiết bị để đảm bảo thông tin gửi về máy chủ là chính xác; hệ thống các mạch điều khiển thiết bị có khả năng kết nối wifi, gửi và nhận dữ liệu trên máy chủ; hệ thống cảm biến để đo các thông số môi trường và truyền dữ liệu về máy chủ để xử lý theo thời gian thực; hệ thống phân tích dữ liệu; hệ thống quản lý module; hệ thống lưu trữ các thao tác điều khiển; hệ thống tạo biểu đồ từ các thông số đo được theo thời gian thực.

Để xây dựng hệ thống này, cần thực hiện các nhiệm vụ sau: xây dựng chương trình máy chủ để nhận dữ liệu được gửi về từ các cảm biến; xây dựng chức năng phân tích dữ liệu nhận được và điều khiển ngược lại thiết bị thông qua các API của hệ thống. Nhóm tác giả đề xuất một giải pháp trong đó mô hình hoạt động của hệ thống được mô tả trong hình 1.

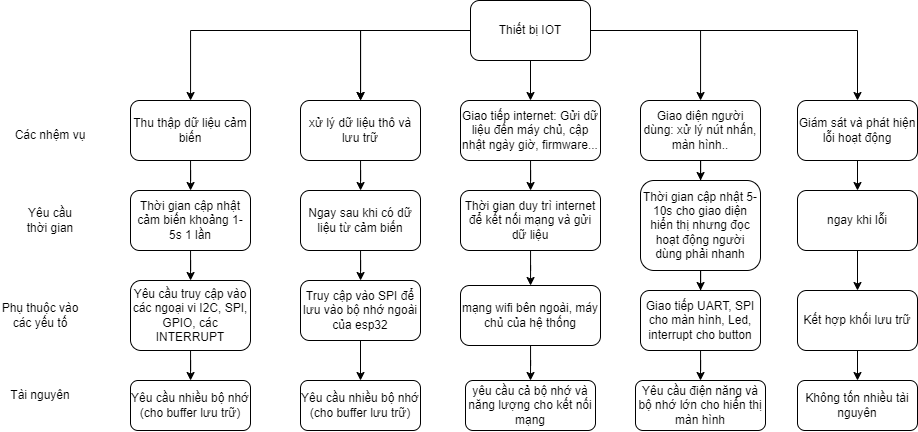


Hình 2 Sơ đồ của hệ thống.

Trong giải pháp này, để thu thập dữ liệu chất lượng không khí và truyền về máy chủ để xử lý, hệ thống sử dụng FreeRTOS và ESP32, và giao thức truyền thông MQTT (Message Queuing Telemetry Transport). Sử dụng hệ điều hành mã nguồn mở FreeRTOS cho phép tùy chỉnh và thích nghi hệ thống để đáp ứng các yêu cầu của ứng dụng, cũng như các yêu cầu phi chức năng như tính ổn định và tiết kiệm năng lượng.

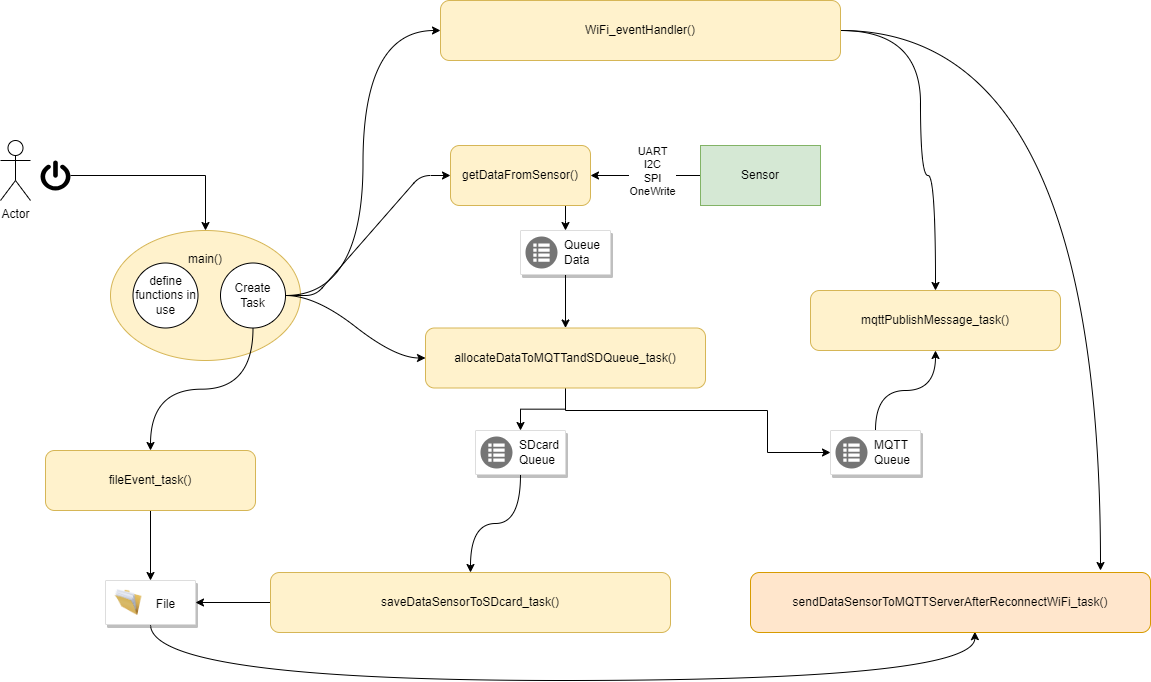
1. **Tính toán Firmware**

Một hệ thống IOT đo chất lượng không khí đơn giản cần thực hiện những nhiệm vụ sau đây:



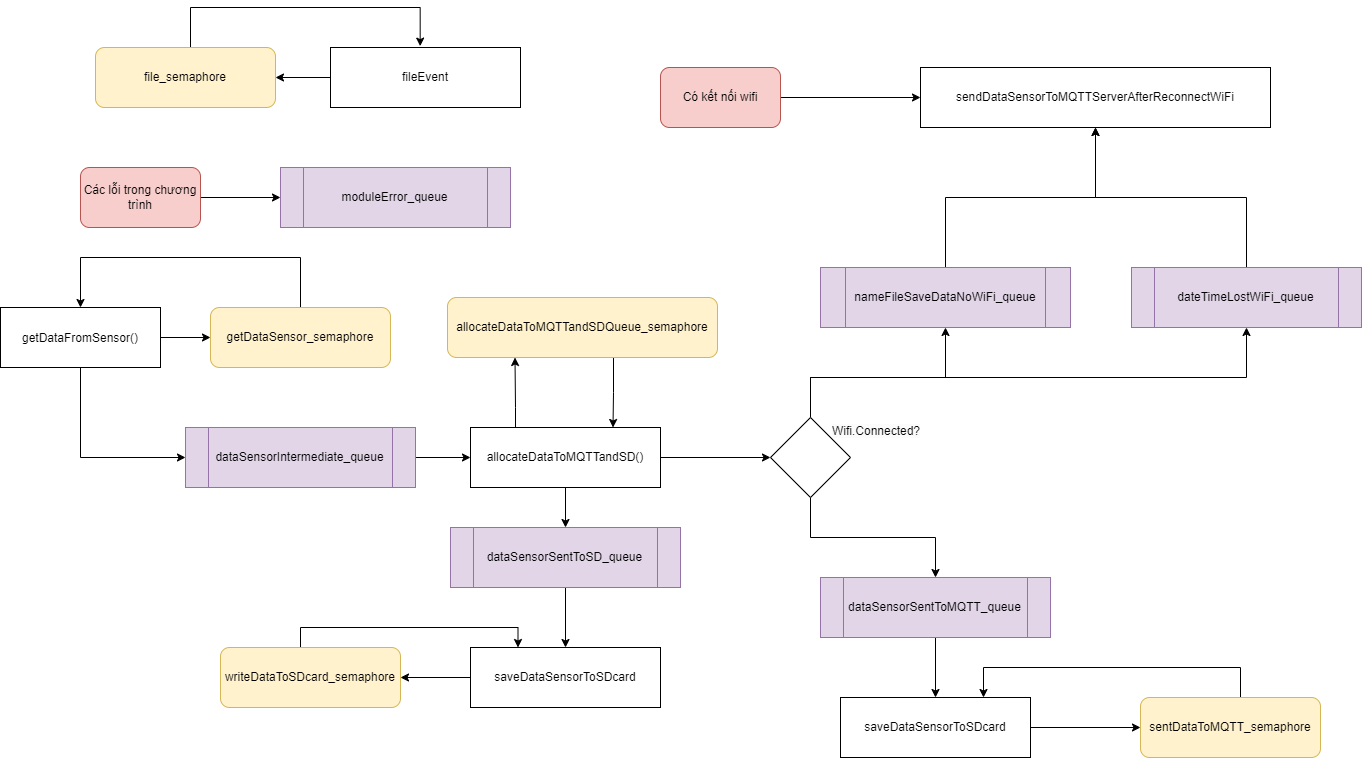
Hình 3 Danh sách nhiệm vụ và yêu cầu từng nhiệm vụ

Từ đó ta có mô hình các nhiệm vụ phải thực hiện

-

Hình 4 Danh sách các nhiệm vụ và các Queue chính

Để quản lý các task trong RTOS ta cần thêm các Semaphore để điều khiển và đồng bộ các luồng dữ liệu:



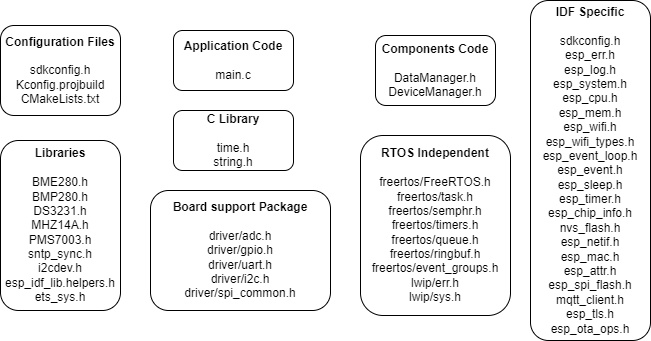
Hình 5 Các Semaphore được sử dụng

Bảng 4‑1 Danh mục các nhiệm vụ trong source code

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Tên nhiệm vụ** | **Thời gian thực hiện ms (C)** | **Thời gian trường hợp sấu nhất ms (T)** | **Thời gian lặp lại ms (Period)** | **Bộ nhớ stack (byte)** | **Độ ưu tiên** | **Hệ số sử dụng (U)** |
| getDataFromSensor | 10 | 100 | 5000 | (1024 \* 32) | 25 | 0.1 |
| fileEvent | 1 | 100 | 0 | (1024 \* 8) | 7 | 0.01 |
| mqttPublishMessage | 10 | 100 | 5000 | (1024 \* 16) | 10 | 0.1 |
| saveDataSensorToSDcard | 5 | 25 | 2500 | (1024 \* 16) | 10 | 0.2 |
| allocateDataToMQTTandSD | 1 | 10 | 100 | (1024 \* 16) | 15 | 0.1 |
| sendDataSensorToMQTTServerAfterReconnectWiFi | 10 | 100 | 100 | (1024 \* 16) | 15 | 0.1 |

Tổng cộng hệ số sử dụng U = 0.61, hệ điều hành hoàn toàn hoạt động bình thường với ESP32 số lõi là 2.

## Sơ đồ Các tệp tin trong hệ thống



Hình 6 Các thành phần trong mã nguồn

Trong đó:

* + Configuration Fiels: các file cài đặt của thiết bị, của hệ điều hành và vi điều khiển esp32 như: Các cảm biến sử dụng, độ ưu tiên tối đa, tối thiểu của RTOS, xung nhịp esp32…
  + IDF Specific: Các mã nguồn của nhà sản xuất esp32 cho chip esp32
  + RTOS independent: Mã nguồn của hệ điều hành RTOS
  + Board Support Package: Mã nguồn của phần cứng cho mạch Esp32 sử dụng trong đồ án
  + C Library: các thư viện C cơ bản
  + Libraries: Các thư viện tổng hợp các các ngoại vi sử dụng trong hệ thống như: Cảm biến BME, Module thời gian thực DS3231…
  + Components Code: 2 File DataManager và DeviceManager giúp quản lý dữ liệu và các thiết bị ngoại vi gắn vào esp32
  + Aplication Code: code thuật toán điều khiển của thiết bị.

1. **Mô hình hệ thống do đạc**

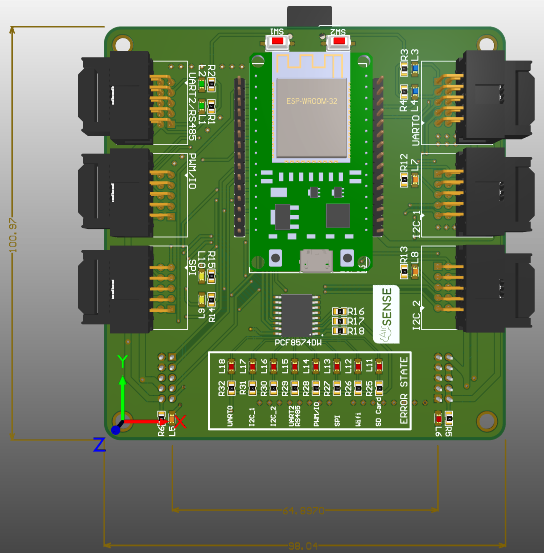
Đối với việc tiết kiệm năng lượng, đó là tiêu chí quan trọng đối với hệ thống đo chất lượng không khí ứng dụng công nghệ IoT. Hệ thống luôn cần năng lượng để hoạt động, nhưng duy trì mức năng lượng cao liên tục sẽ dẫn tới sự lãng phí không cần thiết và giảm tính thực tiễn và giá trị của hệ thống. ESP32 được sử dụng trong hệ thống này cho phép hai chế độ tiết kiệm năng lượng là chế độ "light sleep" và "deep sleep". Trong chế độ "light sleep", các thiết bị ngoại vi kỹ thuật số, hầu hết RAM và CPU được giảm tần số và điện áp cung cấp. Khi thoát khỏi chế độ ngủ, các thiết bị ngoại vi và CPU tiếp tục hoạt động và trạng thái của chúng được bảo toàn. Trong chế độ "deep sleep", CPU và hầu hết RAM cùng với các thiết bị ngoại vi kỹ thuật số có tốc độ trên mức APB\_CLK (mặc định là 80 MHz) sẽ được tắt nguồn. Một số bộ phận khác của chip như bộ điều khiển RTC, thiết bị ngoại vi RTC và bộ nhớ RTC vẫn được cấp nguồn.

Ngoài ra, module Wifi của ESP32 cũng hỗ trợ chế độ "Modem-sleep" để tiết kiệm năng lượng. Việc sử dụng một phiên bản FreeRTOS tùy chỉnh phù hợp để tận dụng tối đa các chức năng tiết kiệm năng lượng của ESP32.

Chi tiết mã nguồn : <https://github.com/Air-SENSE>

1. **Thiết bị phần cứng**

Về phần cứng nhóm sử dụng mạch AirSENSE V4.0 với thiết kế như sau :

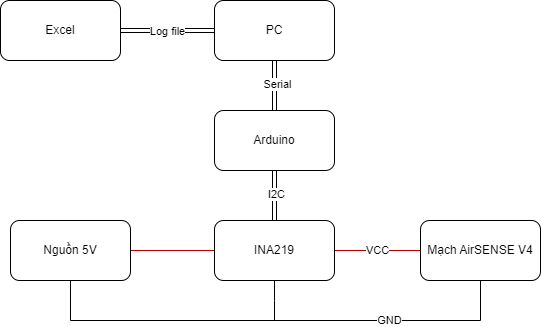


Hình 7 Hình mạch điện AirSENSE tiến hành thử nghiệm

Các cảm biến sử dụng như sau :

1. ESP32 DevKit Module:
   * Thông tin kỹ thuật:
     + MCU: ESP32-Wroom-32
     + Wi-Fi: 802.11 b/g/n
     + Bluetooth: Bluetooth v4.2 BR/EDR và BLE
     + Số chân GPIO: 36
     + RAM: 520 KB
     + Flash: 4 MB
     + Điện áp hoạt động: 2.2V - 3.6V
   * Dòng điện tiêu thụ (Ước tính):
     + Hoạt động trung bình (không kết nối Wi-Fi): Khoảng 80 mA
     + Hoạt động trung bình (kết nối Wi-Fi): Khoảng 100 mA
     + Chế độ Deep Sleep: Khoảng 10 μA
2. Module thẻ nhớ microSD:
   * Thông tin kỹ thuật:
     + Giao tiếp: SPI
     + Hỗ trợ thẻ nhớ microSD và microSDHC
     + Điện áp hoạt động: 3.3V
   * Dòng điện tiêu thụ (Ước tính):
     + Hoạt động trung bình (đọc/ghi dữ liệu): Khoảng 100 mA
     + Chế độ chờ (idle): Khoảng 0.1 mA
3. Module cảm biến BME280:
   * Thông tin kỹ thuật:
     + Cảm biến đa chức năng đo nhiệt độ, độ ẩm và áp suất không khí
     + Độ phân giải:
       1. Nhiệt độ: ±0.01°C
       2. Độ ẩm: ±3%
       3. Áp suất: ±1 hPa
     + Khoảng đo:
       1. Nhiệt độ: -40°C đến +85°C
       2. Độ ẩm: 0% đến 100%
       3. Áp suất: 300 hPa đến 1100 hPa
     + Điện áp hoạt động: 1.8V - 5.5V
     + Giao tiếp: I2C hoặc SPI
     + Dòng điện tiêu thụ: Khoảng 1.8 μA ở chế độ chờ
4. Cảm biến PMS7003:
   * Thông tin kỹ thuật:
     + Cảm biến dạng bụi laser
     + Đo kích thước bụi từ 0.3 μm đến 10 μm
     + Đo nồng độ bụi trong không khí (PM1.0, PM2.5, PM10)
     + Giao tiếp: UART (TTL)
     + Điện áp hoạt động: 5V DC
     + Dòng điện tiêu thụ: Khoảng 100 mA

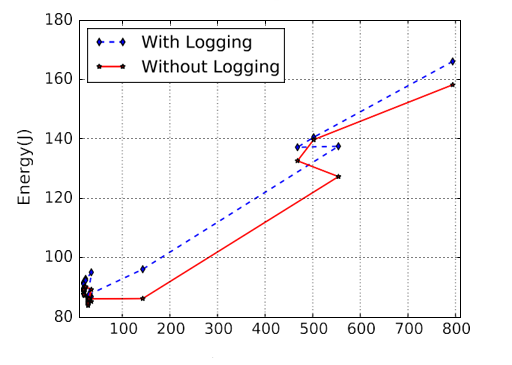
Để đo đạc được năng lượng tiêu thụ, nhóm sử dụng một cảm biến đo công suất INA219 gắn vào đầu nguồn của esp32 và đo năng lượng trong các trường hợp. Mô hình của hệ thống như sau :



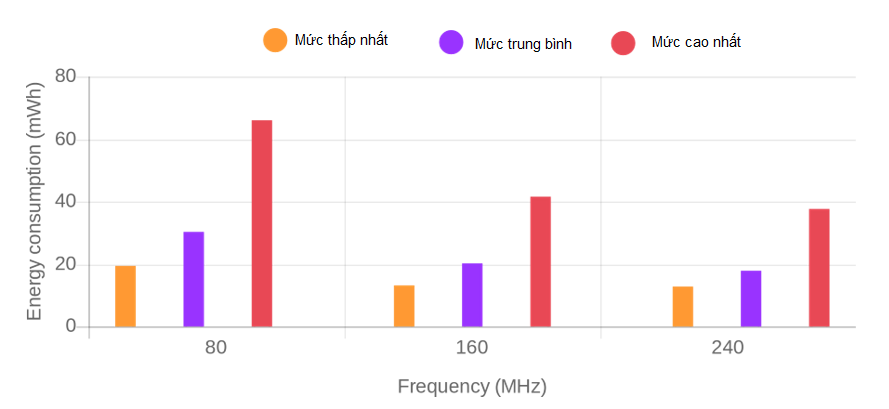
Hình 8 Mô hình kết nối của hệ thống.

**3. Tiến hành đo đạc**

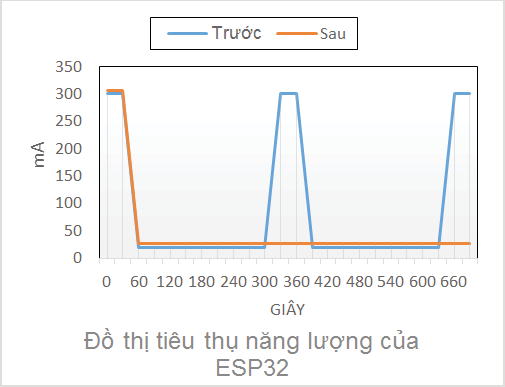
Khi thử nghiệm hệ thống trong phòng thí nghiệm, kết quả đánh giá khả năng tiết kiệm năng lượng khi sử dụng hệ điều hành tùy biến là hoàn toàn vượt trội. Khi một cụm được kết nối với càng nhiều nút thì chỉ số tiết kiệm năng lượng sẽ tăng lên đáng kể, cụ thể là tăng theo cấp số nhân. Một ví dụ đơn giản, trong một cụm nút cảm biến được kết nối bao gồm 3 nút, nếu không sử dụng hệ điều hành tùy biến, trong 3 chu kỳ gửi dữ liệu, mỗi nút đều phải thực hiện tác vụ gửi, sau 3 chu kỳ đã tiêu tốn đến 3 đơn vị năng lượng cho mỗi nút. Trong khi đó, nếu sử dụng hệ điều hành tùy biến, sau 3 chu kỳ gửi dữ liệu, mỗi nút trong cụm chỉ cần gửi một lần, nên hệ thống chỉ tốn một đơn vị năng lượng cho mỗi nút (hình 9 - 13).



Hình 9 Đồ thị hiển thị mức tiêu thụ năng lượng khi dùng serial logging và không logging



Hình 10 Dồ thị tiêu thụ năng lượng liên quan đến tần số hoạt động của esp32



Hình 11 đồ thị tiêu thụ năng lượng của ESP32 trong mô hình phòng thí nghiệm.

3000

2500

2000

1500

1000

500

0

Đối với 1 nút

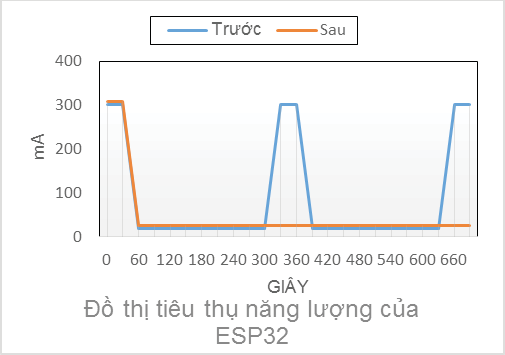
Đối với cả cụm

So sánh mức độ tiêu hao năng lượng

Trước Sau

**0**

Hình 12 . So sánh mức độ tiêu hao năng lượng trước và sau khi áp dụng kịch bản tiết kiệm đối với 1 cụm 3 nút ESP32 trong 3 chu kỳ gửi dữ liệu



Hình 13 đồ thị tiêu thụ năng lượng của ESP32 trong mô hình thực nghiệm.

Khi thử nghiệm trong môi trường thực để đánh giá khả năng tiết kiệm năng lượng của nút cảm biến cho thấy, kết quả không chênh lệch quá nhiều so với trong phòng thí nghiệm (hình 13 và hình 11).

**KẾT LUẬN**

Tích hợp hệ điều hành FreeRTOS đã tùy biến vào ESP32 để điều khiển hệ thống thủy canh thông minh mang lại hiệu quả tiết kiệm điện năng rõ rệt. Điều này sẽ giúp tối ưu hóa các hệ thống thủy canh thông minh, nâng cao giá trị gia tăng cho người sử dụng. Ngoài ra, còn rất nhiều vấn đề khác có thể được khai thác và tối ưu hiệu quả nhờ sử dụng và tùy biến FreeRTOS

Các nghiên cứu trong bài báo này được hỗ trợ bởi đề tài “Tiếp kiệm năng lượng cho ung dụng hệ điều hành FreeRTOS và vi điều khiển ESP32 trong hệ thống quan trắc môi trường ”.

Các tác giả xin trân trọng cảm ơn.

**TÀI LIỆU THAM KHẢO**

|  |  |  |  |
| --- | --- | --- | --- |
| [1] | | "Tổng hợp các bài hướng dẫn về Design Pattern - 23 mẫu cơ bản của GoF," 12 1 2019. [Online]. Available: https://viblo.asia/p/tong-hop-cac-bai-huong-dan-ve-design-pattern-23-mau-co-ban-cua-gof-3P0lPQPG5ox. | |
| [2] | | "Mẫu thiết kế phần mềm," [Online]. Available: https://vi.wikipedia.org/wiki/M%E1%BA%ABu\_thi%E1%BA%BFt\_k%E1%BA%BF\_ph%E1%BA%A7n\_m%E1%BB%81m. | |
| [3] | | "Circular Queue Data Structure," [Online]. Available: https://www.programiz.com/dsa/circular-queue. | |
| [4] | | "Tạo server OTA cập nhật firmware cho ESP8266," [Online]. Available: https://hocarm.org/tao-server-ota-cap-nhat-firmware-cho-esp8266/. | |
| [5] | | D. Landman, "OTA Updates," [Online]. Available: https://github.com/esp8266/Arduino/tree/master/doc/ota\_updates#http-server. | |
|  |  | |
|  |  | |
|  |  | |