**TRƯỜNG ĐẠI HỌC BÌNH DƯƠNG**

**KHOA CÔNG NGHỆ THÔNG TIN, ROBOT VÀ TRÍ TUỆ NHÂN TẠO**

**---oOo---**



**TIỂU LUẬN CUỐI KỲ**

**LẬP TRÌNH HỆ THỐNG**

**Tên đề tài:**

**HỆ THỐNG GIÁM SÁT MÔI TRƯỜNG THÔNG MINH SỬ DỤNG FREERTOS**

Người hướng dẫn: **Lê Duy Hùng**

Sinh viên thực hiện: **Chen Woei Haur – 22050056**

**Nguyễn Hoàng An – 22050040**

**Bình Dương, năm 2025**

**LỜI CẢM ƠN**

Trước hết, chúng em xin gửi lời cảm ơn chân thành và sâu sắc đến thầy Lê Duy Hùng, giảng viên môn Lập trình hệ thống, người đã tận tình truyền đạt kiến thức, định hướng và hướng dẫn chúng em trong suốt quá trình học tập cũng như thực hiện tiểu luận này.

Trong quá trình làm tiểu luận, nhờ sự chỉ dẫn tận tâm của thầy, chúng em đã có cơ hội tiếp cận, rèn luyện và củng cố thêm nhiều kiến thức bổ ích về lĩnh vực lập trình hệ thống, đồng thời phát triển kỹ năng nghiên cứu, phân tích và ứng dụng thực tiễn.

Chúng em cũng xin chân thành cảm ơn các thầy cô trong khoa đã trang bị cho chúng em những nền tảng kiến thức quý báu, cùng bạn bè đã luôn đồng hành, trao đổi và hỗ trợ chúng em trong quá trình thực hiện đề tài.

Mặc dù đã cố gắng hoàn thành tiểu luận một cách tốt nhất, nhưng do kiến thức và kinh nghiệm còn hạn chế nên khó tránh khỏi những thiếu sót. Kính mong thầy thông cảm và góp ý để chúng em có thể rút kinh nghiệm, hoàn thiện hơn trong học tập và nghiên cứu sau này.

Xin trân trọng cảm ơn!

**DANH MỤC HÌNH ẢNH**

[Hình 1.1. Sơ đồ khối mô tả luồng dữ liệu 1](#_Toc207092461)

[Hình 3.1. Sơ đồ use case 12](#_Toc207092462)

[Hình 3.2. Sơ đồ lớp 14](#_Toc207092463)

[Hình 4.1. Kết quả triển khai trên Thingsboard 16](#_Toc207092464)

[Hình 4.2. Gửi dữ liệu lên Thingsboard 16](#_Toc207092465)

**DANH MỤC BẢNG BIỂU**

[Bảng 1.1. Bảng các kết quả cần đạt và tiêu chí đánh giá: 4](#_Toc207092578)

[Bảng 3.1. Danh sách actor 10](#_Toc207092579)

[Bảng 3.2. Danh sách use case 10](#_Toc207092580)

Mục lục

[Chương 1. GIỚI THIỆU TỔNG QUAN 1](#_Toc207092652)

[1.1. Lý do thực hiện đề tài 1](#_Toc207092653)

[1.1.1. Hiện trạng và tính cần thiết: 1](#_Toc207092654)

[1.1.2. Quy trình nghiệp vụ hiện trạng: 1](#_Toc207092655)

[1.2. Các hệ thống tương tự 2](#_Toc207092656)

[1.2.1. Các nghiên cứu, đề tài đã công bố: 2](#_Toc207092657)

[1.2.2. Những vấn đề còn tồn tại: 2](#_Toc207092658)

[1.2.3. Vấn đề đề tài quan tâm và tính mới: 2](#_Toc207092659)

[1.3. Phát biểu bài toán 3](#_Toc207092660)

[1.3.1. Mục tiêu 3](#_Toc207092661)

[1.3.2. Phạm vi 3](#_Toc207092662)

[1.3.3. Sản phẩm 3](#_Toc207092663)

[1.3.4. Ràng buộc 4](#_Toc207092664)

[1.3.5. Giả định và phụ thuộc 4](#_Toc207092665)

[1.4. Kết quả cần đạt và tính ứng dụng 4](#_Toc207092666)

[Chương 2. Cơ sở lý thuyết 6](#_Toc207092667)

[2.1. Cơ sở lý thuyết 6](#_Toc207092668)

[2.1.1. Kiến trúc hệ thống IoT 6](#_Toc207092669)

[2.1.2. Lý thuyết về Hệ điều hành thời gian thực (RTOS) và FreeRTOS 6](#_Toc207092670)

[2.1.3. Mô hình hoạt động của hệ thống 7](#_Toc207092671)

[2.2. Công nghệ sử dụng 7](#_Toc207092672)

[2.2.1. Phần cứng 7](#_Toc207092673)

[2.2.2. Phần mềm & Nền tảng 8](#_Toc207092674)

[2.3. Cách tiếp cận, giải quyết vấn đề 8](#_Toc207092675)

[2.3.1. Mô hình tiếp cận 8](#_Toc207092676)

[2.3.2. Phương pháp phát triển hệ thống 8](#_Toc207092677)

[2.3.3. Quy trình thực hiện 9](#_Toc207092678)

[Chương 3. Phân tích – Thiết kế 10](#_Toc207092679)

[3.1. Các yêu cầu chức năng [Use case view] 10](#_Toc207092680)

[3.1.1. Ngữ cảnh sử dụng 10](#_Toc207092681)

[3.1.2. Các use case 10](#_Toc207092682)

[3.2. Các yêu cầu phi chức năng 12](#_Toc207092683)

[3.2.1. Hiệu năng (Performance) & Tốc độ: 12](#_Toc207092684)

[3.2.2. Tính khả dụng (Availability) & Khả năng chịu tải: 12](#_Toc207092685)

[3.2.3. Tính bảo mật (Security): 13](#_Toc207092686)

[3.2.4. Tính dễ sử dụng (Usability): 13](#_Toc207092687)

[3.2.5. Tính ổn định (Reliability) & Khả năng chịu lỗi (Fault Tolerance): 13](#_Toc207092688)

[3.2.6. Khả năng mở rộng (Scalability): 13](#_Toc207092689)

[3.3. Mô hình hệ thống [Logical view] 14](#_Toc207092690)

[3.3.1. Sơ đồ lớp (Class Diagram) 14](#_Toc207092691)

[Chương 4. Kết quả và thực nghiệm 15](#_Toc207092692)

[4.1. Các kịch bản thử nghiệm 15](#_Toc207092693)

[4.2. Thực Nghiệm 16](#_Toc207092694)

[Chương 5. Kết luận - Hướng phát triển 16](#_Toc207092695)

[5.1. Kết quả đối chiếu với mục tiêu 16](#_Toc207092696)

[5.2. Các hạn chế của đồ án 17](#_Toc207092697)

[5.2.1. Hạn chế về phần cứng và độ chính xác: 17](#_Toc207092698)

[5.2.2. Hạn chế về tính năng: 17](#_Toc207092699)

[5.2.3. Hạn chế về quy mô thử nghiệm: 17](#_Toc207092700)

[5.3. Hướng phát triển 17](#_Toc207092701)

[5.3.1. Nâng cấp phần cứng: 17](#_Toc207092702)

[5.3.2. Phát triển tính năng phần mềm: 18](#_Toc207092703)

[5.3.3. Mở rộng hệ thống: 18](#_Toc207092704)

# GIỚI THIỆU TỔNG QUAN

## Lý do thực hiện đề tài

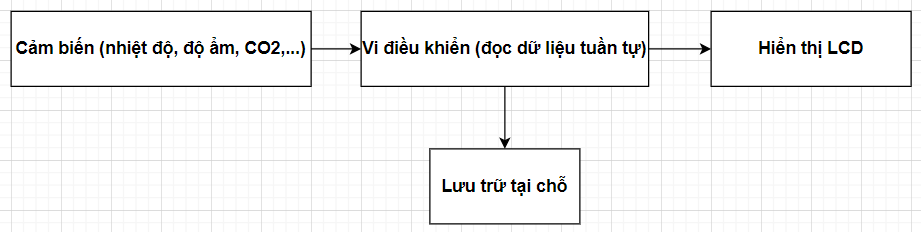
### Hiện trạng và tính cần thiết:

Trong bối cảnh cách mạng công nghiệp 4.0 và đô thị hóa mạnh mẽ, vấn đề ô nhiễm môi trường (không khí, tiếng ồn, nhiệt độ...) đang trở thành mối quan tâm hàng đầu trên toàn cầu, đặc biệt là tại các thành phố lớn như Hà Nội và TP. Hồ Chí Minh. Việc giám sát liên tục và chính xác các thông số môi trường là nhu cầu cấp thiết không chỉ của các cơ quan quản lý nhà nước mà còn của từng hộ gia đình, văn phòng, nhà xưởng sản xuất.

Các hệ thống giám sát môi trường truyền thống thường có quy mô lớn, chi phí cao, hoạt động độc lập và chưa có khả năng cảnh báo tức thời. Người dùng cuối khó có thể tiếp cận được dữ liệu một cách trực quan và chủ động. Hơn nữa, nhiều hệ thống nhỏ lẻ hiện nay sử dụng các vi điều khiển đơn nhiệm, dẫn đến hạn chế trong việc xử lý đa tác vụ đồng thời (đọc cảm biến, giao tiếp mạng, hiển thị, lưu trữ), gây ra độ trễ và thiếu ổn định.

### Quy trình nghiệp vụ hiện trạng:

Một quy trình giám sát cơ bản thường được mô tả qua sơ đồ sau:



Hình 1.1. **Sơ đồ khối mô tả luồng dữ liệu**

**Ưu điểm:** Đơn giản, dễ triển khai.  
**Nhược điểm:**

+ Xử lý tuần tự, gây trễ nếu có nhiều cảm biến hoặc tác vụ phức tạp.

+ Không có khả năng kết nối Internet để giám sát từ xa.

+ Không thể đồng thời vừa hiển thị vừa lưu trữ dữ liệu một cách mượt mà.

+ Thiếu cơ chế cảnh báo chủ động (email, SMS) khi các ngưỡng môi trường bị vượt.

Do đó, một hệ thống giám sát thông minh, có khả năng xử lý đa nhiệm ổn định, kết nối Internet để quan sát từ xa theo thời gian thực và đưa ra cảnh báo kịp thời là một nhu cầu tất yếu. Đề tài này ra đời để giải quyết những hạn chế trên, mang lại một giải pháp hiệu quả, chi phí hợp lý và dễ tiếp cận cho người dùng.

## Các hệ thống tương tự

### Các nghiên cứu, đề tài đã công bố:

Hệ thống dùng Arduino (ATMega) và ESP8266: Đây là hướng tiếp cận phổ biến. Các đề tài này thường sử dụng thư viện loop() truyền thống của Arduino để đọc cảm biến và gửi dữ liệu lên server (ví dụ: Blynk, Thingspeak). Ưu điểm là nhanh chóng và dễ thực hiện.

Hệ thống dùng Raspberry Pi: Các hệ thống này sử dụng các ngôn ngữ cấp cao như Python, Node.js trên nền tảng Linux để thu thập và xử lý dữ liệu. Ưu điểm là sức mạnh xử lý lớn, dễ dàng tích hợp nhiều thư viện.

### Những vấn đề còn tồn tại:

Với hệ thống dùng Arduino (loop()): Gặp khó khăn khi mở rộng hệ thống với nhiều cảm biến và tác vụ phức tạp. Việc xử lý tuần tự có thể gây ra nghẽn cổ chai, dẫn đến mất dữ liệu hoặc trễ trong các tác vụ quan trọng như giao tiếp mạng. Thiếu cơ chế ưu tiên tác vụ.

Với hệ thống dùng Raspberry Pi: Mặc dù mạnh mẽ nhưng chi phí phần cứng cao hơn, tiêu thụ năng lượng lớn hơn, và không thực sự phù hợp cho các ứng dụng nhúng thời gian thực chặt chẽ (real-time) do Linux không phải là một HĐH thời gian thực.

### Vấn đề đề tài quan tâm và tính mới:

Đề tài này tập trung vào việc khắc phục những nhược điểm trên bằng cách:

**+ Sử dụng vi điều khiển ESP32:** Kết hợp sức mạnh xử lý dual-core, hỗ trợ WiFi và Bluetooth, giá thành rẻ.

**+ Sử dụng FreeRTOS - một Hệ điều hành thời gian thực (RTOS):** Đây là điểm mấu chốt tạo nên tính mới của đề tài. FreeRTOS cho phép:

* **Xử lý đa nhiệm mạnh mẽ:** Mỗi tác vụ (đọc cảm biến, giao tiếp MQTT, hiển thị) chạy trong một Task độc lập, được lập lịch tối ưu bởi kernel.
* **Tính ổn định và phản hồi thời gian thực:** Các tác vụ quan trọng (ví dụ: cảnh báo) có thể được cấp độ ưu tiên cao, đảm bảo luôn được xử lý kịp thời.
* **Dễ dàng bảo trì và mở rộng:** Code được module hóa theo từng tác vụ, giúp việc thêm/bớt tính năng trở nên dễ dàng hơn so với phương pháp loop() truyền thống.

**+ Kết hợp với giao thức MQTT:** Một giao thức publish/subscribe nhẹ, lý tưởng cho IoT, giúp truyền dữ liệu ổn định và tiết kiệm năng lượng.

Như vậy, tính mới của đề tài nằm ở việc áp dụng có chủ đích FreeRTOS trên nền tảng ESP32 để xây dựng một hệ thống giám sát có tính ổn định cao, khả năng đáp ứng thời gian thực và dễ dàng mở rộng, vượt trội so với các giải pháp dựa trên loop() truyền thống.

## Phát biểu bài toán

### Mục tiêu

Xây dựng một hệ thống phần cứng và phần mềm nhúng, sử dụng FreeRTOS trên vi điều khiển ESP32 để giám sát liên tục các thông số môi trường bao gồm nhiệt độ, độ ẩm, nồng độ khí CO2 (hoặc CO, VOC tùy chọn) và hiển thị chúng lên màn hình, đồng thời gửi dữ liệu lên một server để theo dõi từ xa qua giao diện web. Hệ thống có khả năng cảnh báo khi các thông số vượt ngưỡng cho phép.

### Phạm vi

**Phần cứng:** ESP32, cảm biến nhiệt độ - độ ẩm (DHT11), cảm biến khí (MQ-135), màn hình OLED/I2C LCD, cảm biến ánh sáng (BH1750), cảm biến áp suất không khí (BMP180).

**Phần mềm nhúng:** Lập trình trên Arduino IDE/PlatformIO, sử dụng FreeRTOS để quản lý đa tác vụ.

**Kết nối:** Giao thức MQTT thông qua kết nối WiFi.

**Phía server:** Sử dụng một MQTT Broker mã nguồn mở (ví dụ: Mosquitto) và một giao diện web đơn giản (có thể dùng Node-RED hoặc một framework web cơ bản) để hiển thị dữ liệu dạng biểu đồ.

### Sản phẩm

Một prototype phần cứng hoàn chỉnh của hệ thống giám sát.

Source code phần firmware cho ESP32, được viết bằng Arduino và tận dụng các API của FreeRTOS.

Giao diện web cho phép xem dữ liệu môi trường theo thời gian thực và lịch sử.

Tài liệu hướng dẫn cài đặt, vận hành và báo cáo đồ án đầy đủ.

### Ràng buộc

**Nghiệp vụ:** Dữ liệu từ cảm biến cần được cập nhật ít nhất mỗi 5 giây một lần trên giao diện web. Cảnh báo phải được gửi đi trong vòng dưới 2 giây kể từ khi phát hiện sự cố.

**Công nghệ:** Hệ thống phải được lập trình sử dụng FreeRTOS. Giao thức truyền thông chính là MQTT.

### Giả định và phụ thuộc

Hệ thống hoạt động trong phạm vi phủ sóng WiFi ổn định.

Các cảm biến được hiệu chuẩn chính xác trước khi sử dụng.

Server MQTT Broker luôn trong trạng thái hoạt động và có thể truy cập được.

## Kết quả cần đạt và tính ứng dụng

Bảng 1.1. Bảng các kết quả cần đạt và tiêu chí đánh giá:

|  |  |  |  |
| --- | --- | --- | --- |
| STT | Kết Quả Cần Đạt | Tiêu Chí Đánh Giá | Tính Ứng Dụng |
| 1 | Thiết kế, lắp ráp thành công board mạch phần cứng. | Board mạch hoạt động ổn định, kết nối chính xác giữa ESP32 và các cảm biến, màn hình. | Ứng dụng trong thực tế để lấy dữ liệu môi trường tại các điểm cần đo. |
| 2 | Xây dựng thành công firmware dựa trên FreeRTOS. | Firmware ổn định, không treo máy. Các Task (đọc cảm biến, MQTT, hiển thị) hoạt động độc lập và đồng thời. | Tạo nền tảng phần mềm ổn định, có thể dễ dàng tích hợp thêm cảm biến mới hoặc tính năng mới trong tương lai. |
| 3 | Thu thập chính xác dữ liệu từ các cảm biến. | Giá trị đọc được từ cảm biến sai lệch < 5% so với thiết bị đo chuẩn. Dữ liệu được cập nhật liên tục. | Cung cấp dữ liệu đáng tin cậy cho người dùng để phân tích và ra quyết định. |
| 4 | Truyền dữ liệu lên server và hiển thị trên giao diện web. | Dữ liệu được hiển thị trên web theo thời gian thực (< 5s độ trễ) với biểu đồ trực quan. | Cho phép giám sát từ xa mọi lúc, mọi nơi, đặc biệt hữu ích cho giám sát tại các khu vực khó tiếp cận. |
| 5 | Thực hiện được cơ chế cảnh báo (qua còi/đèn tại chỗ và notification trên web). | Hệ thống kích hoạt cảnh báo trong vòng < 2s khi phát hiện giá trị vượt ngưỡng. | Ứng dụng trong cảnh báo cháy nổ (khói, khí gas), cảnh báo nhiệt độ cao trong nhà máy, cảnh báo chất lượng không khí kém. |

**Tính ứng dụng của đề tài:**

Đề tài mang tính ứng dụng thực tiễn cao, có thể được triển khai trong nhiều lĩnh vực:

**+ Gia đình & Văn phòng:** Giám sát và cải thiện chất lượng không khí, điều khiển tự động các thiết bị làm mát, lọc không khí.

**+ Nông nghiệp:** Giám sát nhiệt độ, độ ẩm trong các nhà kính, trang trại, từ đó tự động hóa việc tưới tiêu, thông gió.

**+ Công nghiệp:** Giám sát môi trường làm việc trong các nhà xưởng, kho bãi (nhiệt độ, khí độc), đảm bảo an toàn lao động.

**+ Y tế:** Theo dõi điều kiện môi trường trong các phòng bệnh, phòng lưu trữ thuốc.

Việc sử dụng FreeRTOS đảm bảo hệ thống hoạt động ổn định, phản hồi nhanh và là cơ sở vững chắc để phát triển thành các sản phẩm thương mại hóa trong tương lai.

# Cơ sở lý thuyết

## Cơ sở lý thuyết

### Kiến trúc hệ thống IoT

Hệ thống IoT (Internet of Things) cho giám sát môi trường thường tuân theo một kiến trúc phân tầng cơ bản [1]:

**+ Tầng cảm biến (Perception Layer):** Bao gồm các cảm biến (DHT11, MQ135, BH1750, BMP180) và vi điều khiển (ESP32). Nhiệm vụ là thu thập dữ liệu vật lý từ môi trường.

**+ Tầng mạng (Network Layer):** Bao gồm module WiFi trên ESP32. Nhiệm vụ là truyền dữ liệu đã thu thập được lên internet.

**+ Tầng ứng dụng (Application Layer):** Là nền tảng đám mây ThingsBoard. Nhiệm vụ là lưu trữ, xử lý, hiển thị dữ liệu và cung cấp giao diện người dùng để giám sát, cảnh báo.

### Lý thuyết về Hệ điều hành thời gian thực (RTOS) và FreeRTOS

**RTOS (Real-Time Operating System):** Là một hệ điều hành được thiết kế để xử lý các ứng dụng đòi hỏi phản hồi trong một khoảng thời gian xác định và có thể dự đoán được. Khác với hệ điều hành đa nhiệm thông thường (như trên PC), RTOS ưu tiên tính kịp thời và độ tin cậy hơn là raw throughput [2].

**FreeRTOS:** Là một RTOS mã nguồn mở, phổ biến hàng đầu cho các vi điều khiển nhúng. Nó cung cấp một kernel rất nhỏ gọn và mạnh mẽ, bao gồm các cơ chế: lập lịch tác vụ (scheduling), đồng bộ hóa (semaphore, mutex), truyền thông giữa các tác vụ (queue), và quản lý bộ nhớ [3].

**+ Task (Tác vụ):** Là một đơn vị thực thi độc lập, tương tự như một thread. Mỗi task có một mức độ ưu tiên (priority) riêng. Kernel của FreeRTOS sẽ dùng thuật toán lập lịch ưu tiên (Priority-based Preemptive Scheduling) để quyết định task nào được thực thi tại một thời điểm. Task có độ ưu tiên cao hơn sẽ "chiếm quyền" (preempt) task có độ ưu tiên thấp hơn ngay lập tức khi nó sẵn sàng.

**+ Mutex (Mutual Exclusion Object):** Là một cơ chế đồng bộ hóa được sử dụng để quản lý quyền truy cập vào một tài nguyên được chia sẻ (shared resource) - chẳng hạn như một biến toàn cục, một cổng giao tiếp (I2C), hoặc một file - giữa nhiều task. Chỉ một task có thể "sở hữu" mutex tại một thời điểm. Điều này ngăn ngừa hiện tượng race condition (điều kiện cạnh tranh), nơi nhiều task cùng ghi đè lên dữ liệu của nhau, dẫn đến kết quả không mong muốn hoặc hỏng dữ liệu [3].

### Mô hình hoạt động của hệ thống

Dựa trên mô tả, hệ thống hoạt động theo mô hình đa nhiệm được đồng bộ hóa bằng FreeRTOS.

Các tác vụ chính:

+ **task\_read\_sensors:** Một task có nhiệm vụ đọc dữ liệu từ tất cả các cảm biến (DHT11, MQ135, BH1750, BMP180).

**+ task\_send\_data:** Một task có nhiệm vụ kết nối WiFi (nếu chưa kết nối) và gửi dữ liệu đã đọc được lên ThingsBoard thông qua giao thức MQTT hoặc HTTP.

Cơ chế đồng bộ - Mutex: Cảm biến DHT22 sử dụng giao thức one-wire, trong khi các cảm biến khác (BH1750, BMP180) thường sử dụng giao thức I2C. Cổng I2C trên vi điều khiển là một tài nguyên dùng chung. Nếu task\_read\_sensors và một task khác (nếu có) cố gắng truy cập cùng một bus I2C để giao tiếp với các cảm biến khác nhau cùng một lúc, dữ liệu trên bus sẽ bị xung đột. Do đó, một mutex cho bus I2C là bắt buộc. Trước khi truy cập vào bus I2C, task phải "lấy" mutex. Nếu mutex đã được task khác nắm giữ, task hiện tại sẽ phải chờ cho đến khi mutex được giải phóng. Điều này đảm bảo tính toàn vẹn của dữ liệu truyền trên bus.

## Công nghệ sử dụng

### Phần cứng

Vi điều khiển ESP32: Là trái tim của hệ thống. Ưu điểm: có tích hợp WiFi và Bluetooth, 2 lõi xử lý, tốc độ xung nhịp 240MHz, RAM đủ lớn để chạy FreeRTOS, và hệ sinh thái phong phú [4].

Cảm biến nhiệt độ & độ ẩm DHT11: Cảm biến số, đo nhiệt độ (0°C đến 50°C) và độ ẩm (20-90% RH) với độ chính xác tương đối cao. Giao tiếp one-wire [5].

Cảm biến khí MQ135: Cảm biến analog, nhạy với một loạt khí độc hại như NH3, NOx, CO2, Benzen. Dữ liệu analog cần được đọc qua ADC của ESP32 và hiệu chuẩn [6].

Cảm biến ánh sáng BH1750: Cảm biến số đo cường độ ánh sáng (0-65535 lux). Giao tiếp I2C, độ chính xác cao, không cần ADC ngoài [7].

Cảm biến áp suất BMP180: Cảm biến số đo áp suất khí quyển và nhiệt độ (có thể dùng để xác định độ cao). Giao tiếp I2C [8].

### Phần mềm & Nền tảng

**FreeRTOS:** Được sử dụng như kernel để quản lý đa nhiệm và tài nguyên.

**Arduino Core for ESP32:** Một framework phần mềm cung cấp các thư viện và API để lập trình cho ESP32 một cách dễ dàng, tương thích hoàn toàn với FreeRTOS.

**PlatformIO:** Một IDE chuyên nghiệp và ecosystem cho phát triển ứng dụng nhúng, hỗ trợ quản lý thư viện và build project tốt hơn so với Arduino IDE truyền thống.

**ThingsBoard:** Nền tảng IoT mã nguồn mở (Open-Source IoT Platform) dùng để thu thập, trực quan hóa dữ liệu (dashboard), quản lý thiết bị và tạo rule engine để cảnh báo. Hỗ trợ giao thức MQTT và HTTP một cách mạnh mẽ [9].

**Giao thức MQTT (Message Queuing Telemetry Transport):** Là giao thức publish-subscribe dựa trên TCP/IP, được thiết kế cho các thiết bị có băng thông thấp và độ tin cậy cao. Là giao thức tiêu chuẩn de facto trong IoT [10].

## Cách tiếp cận, giải quyết vấn đề

### Mô hình tiếp cận

Đề tài tiếp cận theo mô hình ứng dụng nhúng thời gian thực (Real-Time Embedded Application). Vấn đề giám sát môi trường được giải quyết bằng cách phân tách các chức năng độc lập thành các task riêng biệt trong FreeRTOS, thay vì gộp chung vào một hàm loop() duy nhất. Các task này giao tiếp và đồng bộ với nhau thông qua các cơ chế của RTOS như mutex và queue.

### Phương pháp phát triển hệ thống

Phương pháp phát triển được áp dụng là Mô hình thác nước (Waterfall Model) với các giai đoạn tuần tự, phù hợp với quy mô của một đồ án tốt nghiệp:

+ **Phân tích yêu cầu:** Xác định các thông số môi trường cần giám sát, các chức năng chính (đọc cảm biến, gửi dữ liệu, hiển thị), và các ràng buộc về hiệu năng.

+ **Thiết kế hệ thống:** Thiết kế sơ đồ khối phần cứng, sơ đồ kết nối, và đặc biệt là thiết kế phần mềm với sơ đồ các task, mức độ ưu tiên và cơ chế đồng bộ giữa chúng.

+ **Triển khai (Coding):** Lập trình firmware cho ESP32 sử dụng FreeRTOS, viết code cho từng task, tích hợp thư viện cảm biến và MQTT. Cấu hình dashboard trên ThingsBoard.

+ **Kiểm thử (Testing):**Kiểm thử từng module (đọc từng cảm biến, kết nối WiFi, gửi MQTT), sau đó kiểm thử tích hợp toàn hệ thống, kiểm tra tính ổn định và khả năng đáp ứng thời gian thực.

+ **Vận hành & Bảo trì:** Triển khai hệ thống thực tế, theo dõi hoạt động và bảo trì code nếu cần.

### Quy trình thực hiện

+ **Nghiên cứu, lựa chọn linh kiện** phù hợp với yêu cầu và ngân sách.

+ **Lắp ráp mạch** trên breadboard/protoboard, kiểm tra kết nối.

+ **Cài đặt môi trường phát triển:** PlatformIO, thư viện cho ESP32, FreeRTOS, và các cảm biến.

+ **Viết code cho từng task:**

* Tạo task task\_read\_sensors: Sử dụng mutex để đồng bộ truy cập I2C, lần lượt đọc các cảm biến và lưu giá trị vào một struct dữ liệu chung.
* Tạo task task\_send\_data: Kết nối WiFi, kết nối MQTT đến ThingsBoard, và định kỳ publish struct dữ liệu chung lên các topic tương ứng.
* Cấu hình độ ưu tiên task (ví dụ: task đọc cảm biến có thể có độ ưu tiên cao hơn task gửi dữ liệu).

+ **Cấu hình ThingsBoard:** Tạo device, tạo dashboard với các widget (chart, gauge, label) để hiển thị dữ liệu.

+ **Kiểm thử tổng thể, tối ưu hóa code** và hoàn thiện báo cáo.

# Phân tích – Thiết kế

## Các yêu cầu chức năng [Use case view]

### Ngữ cảnh sử dụng

Bảng 3.1. Danh sách actor

|  |  |
| --- | --- |
| Tên Actor | Mô tả |
| **Người dùng (User)** | Người trực tiếp theo dõi, giám sát các thông số môi trường thông qua giao diện web. Có thể là chủ gia đình, kỹ sư, người vận hành. |
| **Hệ thống ThingsBoard** | Nền tảng đám mây đóng vai trò là một actor hệ thống. Nó nhận dữ liệu từ thiết bị, cung cấp giao diện web và gửi các lệnh cảnh báo/điều khiển ngược lại. |
| **Thiết bị Ngoại vi** | Các thiết bị được hệ thống điều khiển dựa trên ngưỡng dữ liệu (ví dụ: quạt, đèn). |

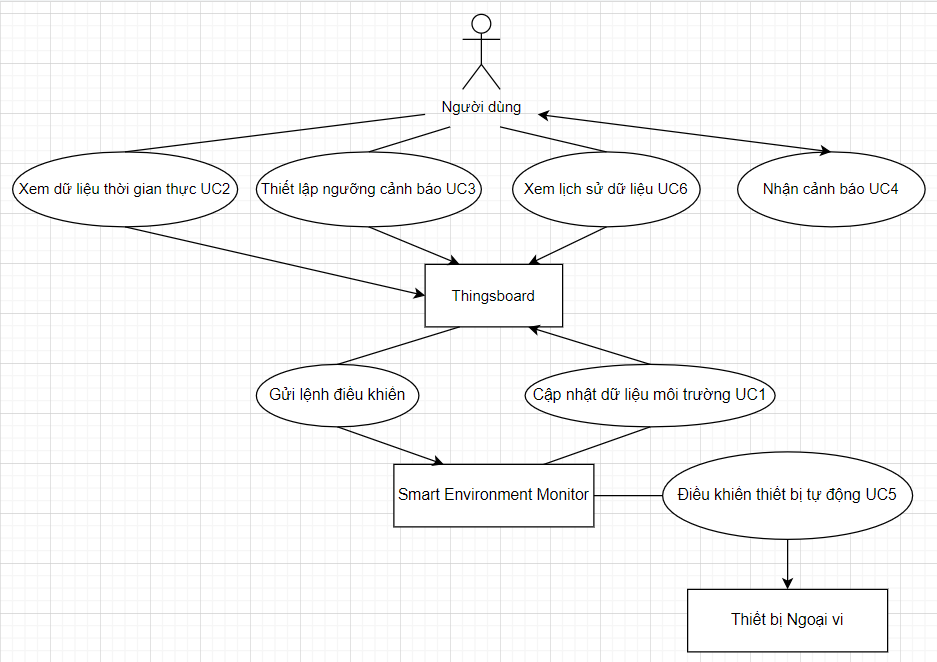
### Các use case

#### Danh sách use case

Bảng 3.2. Danh sách use case

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Tên Use Case | Actor Chính | Mô tả Ngắn gọn |
| UC1 | **Cập nhật dữ liệu môi trường** | Hệ thống ThingsBoard | Hệ thống thu thập dữ liệu từ các cảm biến và gửi lên ThingsBoard định kỳ. |
| UC2 | **Giám sát dữ liệu thời gian thực** | Người dùng | Người dùng xem các thông số môi trường (dạng số, biểu đồ) được cập nhật liên tục trên dashboard. |
| UC3 | **Thiết lập ngưỡng cảnh báo** | Người dùng | Người dùng cấu hình các ngưỡng giá trị (ví dụ: nhiệt độ > 35°C) để kích hoạt cảnh báo hoặc điều khiển. |
| UC4 | **Nhận cảnh báo** | Người dùng | Hệ thống gửi thông báo (trên dashboard, email, SMS) khi dữ liệu vượt ngưỡng cho phép. |
| UC5 | **Điều khiển thiết bị tự động** | Hệ thống (ThingsBoard) | Hệ thống tự động bật/tắt các thiết bị (quạt, đèn) dựa trên ngưỡng dữ liệu đã được cài đặt. |
| UC6 | **Xem lịch sử dữ liệu** | Người dùng | Người dùng xem lại diễn biến của các thông số môi trường trong quá khứ (theo giờ, ngày, tháng). |

#### Sơ đồ use case chính



Hình 3.1. Sơ đồ use case

## Các yêu cầu phi chức năng

### Hiệu năng (Performance) & Tốc độ:

Tốc độ lấy mẫu: Hệ thống phải có khả năng đọc và xử lý dữ liệu từ tất cả các cảm biến ít nhất mỗi 3 giây một lần.

Độ trễ dữ liệu: Độ trễ từ khi cảm biến đọc giá trị đến khi dữ liệu xuất hiện trên dashboard ThingsBoard không được vượt quá 5 giây.

Thời gian phản hồi cảnh báo: Hệ thống phải phát hiện và kích hoạt cơ chế cảnh báo (bật còi, gửi lệnh điều khiển) trong vòng dưới 2 giây kể từ khi giá trị vượt ngưỡng.

### Tính khả dụng (Availability) & Khả năng chịu tải:

Thời gian hoạt động: Hệ thống phần cứng phải hoạt động ổn định 24/7.

Khả năng chịu lỗi kết nối: Hệ thống phải có cơ chế lưu trữ dữ liệu tạm thời (trong RAM) hoặc trên thẻ SD trong trường hợp mất kết nối WiFi hoặc mất kết nối đến ThingsBoard. Dữ liệu sẽ được đồng bộ hóa lại khi kết nối được phục hồi.

Tải kết nối: Hệ thống phải hoạt động ổn định khi kết nối cùng lúc nhiều thiết bị (nếu mở rộng) và khi có ít nhất 5 người dùng cùng truy cập dashboard một lúc.

### Tính bảo mật (Security):

Xác thực: Thiết bị ESP32 phải sử dụng Access Token do ThingsBoard cung cấp để xác thực trước khi được phép publish dữ liệu.

Bảo mật giao thức: Giao thức MQTT phải được sử dụng trên nền TLS/SSL (MQTTS) để mã hóa dữ liệu truyền đi, ngăn chặn nghe lén.

Tính toàn vẹn dữ liệu: Dữ liệu truyền đi phải được đảm bảo không bị thay đổi hoặc giả mạo.

### Tính dễ sử dụng (Usability):

Giao diện người dùng: Dashboard trên ThingsBoard phải trực quan, dễ hiểu, hiển thị rõ ràng các thông số và biểu đồ.

Cài đặt và vận hành: Quy trình cấu hình ban đầu cho thiết bị (kết nối WiFi) phải đơn giản (ví dụ: sử dụng chế độ SmartConfig hoặc Web Server).

Bảo trì: Hệ thống phải cung cấp cơ chế log lỗi cơ bản (qua Serial hoặc gửi cảnh báo) để hỗ trợ chẩn đoán sự cố từ xa.

### Tính ổn định (Reliability) & Khả năng chịu lỗi (Fault Tolerance):

Watchdog Timer: Hệ thống phải sử dụng Hardware/Software Watchdog Timer để tự động reset trong trường hợp firmware bị treo do lỗi phần mềm hoặc nhiễu.

Xử lý lỗi cảm biến: Hệ thống phải có khả năng phát hiện cảm biến hỏng hoặc mất kết nối và gửi cảnh báo về tình trạng này cho người dùng, thay vì ngừng hoạt động hoàn toàn.

### Khả năng mở rộng (Scalability):

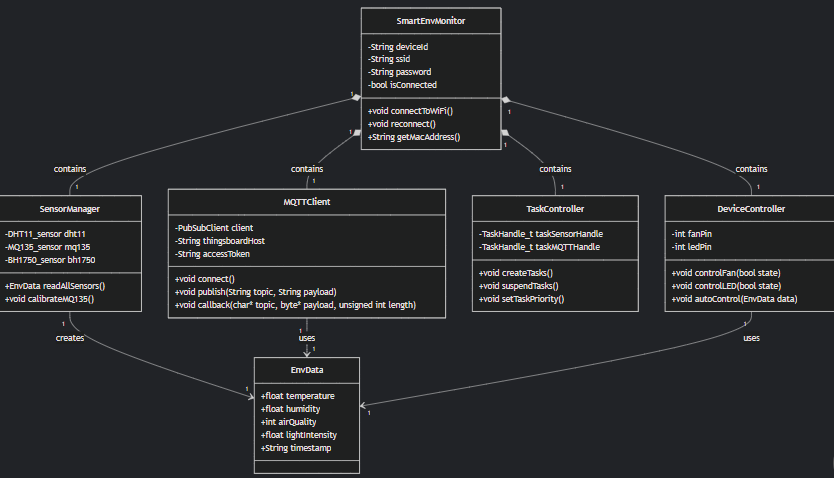
Kiến trúc phần mềm: Kiến trúc dựa trên FreeRTOS và module hóa cho phép dễ dàng thêm hoặc bớt cảm biến mới trong tương lai mà không cần thay đổi cấu trúc code lớn.

Tài nguyên phần cứng: Vi điều khiển ESP32 được chọn phải có đủ tài nguyên (RAM, GPIO, khả năng xử lý) để đáp ứng các yêu cầu mở rộng sau này.

## Mô hình hệ thống [Logical view]

### Sơ đồ lớp (Class Diagram)

Sơ đồ lớp mô tả cấu trúc tĩnh của hệ thống, các thành phần (lớp), thuộc tính, phương thức và mối quan hệ giữa chúng.



Hình 3.2. Sơ đồ lớp

**Giải thích các lớp chính:**

**SmartEnvMonitor:** Lớp chính, đóng vai trò điều phối toàn bộ hệ thống. Chứa các đối tượng của các lớp quản lý chức năng khác.

**SensorManager:** Lớp quản lý việc đọc dữ liệu từ tất cả các cảm biến. Sử dụng Mutex để đồng bộ truy cập bus I2C.

**EnvData:** Lớp dữ liệu (Data Transfer Object - DTO) dùng để đóng gói các giá trị đọc từ cảm biến, giúp dễ dàng truyền dữ liệu giữa các task.

**MQTTClient:** Lớp đóng gói toàn bộ logic kết nối và giao tiếp với MQTT Broker (ThingsBoard).

**TaskController:** Lớp quản lý việc tạo, hủy, và thiết lập độ ưu tiên cho các FreeRTOS Task.

**DeviceController:** Lớp điều khiển các thiết bị ngoại vi (quạt, đèn) thông qua các chân GPIO và relay.

# Kết quả và thực nghiệm

## Các kịch bản thử nghiệm

Để đánh giá hiệu năng và tính ưu việt của hệ thống sử dụng FreeRTOS, đề tài tiến hành thử nghiệm so sánh giữa 02 mô hình:

**Mô hình 1 (Đối chứng):** Hệ thống được lập trình theo phương pháp truyền thống, sử dụng hàm loop() tuần tự của Arduino.

**Mô hình 2 (Đề xuất):** Hệ thống được thiết kế với kiến trúc đa nhiệm dựa trên FreeRTOS, sử dụng các Task độc lập và cơ chế đồng bộ Mutex.

**Các kịch bản thử nghiệm được thiết kế như sau:**

**Kịch bản 1:** Thử nghiệm tốc độ phản hồi và độ trễ

**+ Mục tiêu:** Đo lường thời gian từ khi cảm biến đọc giá trị đến khi dữ liệu xuất hiện trên Dashboard ThingsBoard.

**+ Phương pháp:** Ghi lại timestamp khi đọc cảm biến và timestamp khi dữ liệu được publish thành công. So sánh độ trễ trung bình giữa hai mô hình.

**Kịch bản 2:** Thử nghiệm khả năng xử lý đa tác vụ dưới tải

**+ Mục tiêu:** Đánh giá khả năng duy trì hoạt động ổn định khi phải xử lý đồng thời nhiều tác vụ (đọc cảm biến, gửi MQTT, điều khiển thiết bị, hiển thị).

**+ Phương pháp:** Bổ sung thêm tác vụ taskBlinkLED (nhấp nháy đèn LED với chu kỳ 100ms) để mô phỏng một tác vụ có độ ưu tiên cao. Quan sát xem hệ thống nào bị ảnh hưởng (đèn nhấp nháy không đều, mất dữ liệu cảm biến, nghẽn MQTT).

**Kịch bản 3:** Thử nghiệm tính ổn định khi mất kết nối mạng

**+ Mục tiêu:** Đánh giá phản ứng của hệ thống khi kết nối WiFi hoặc MQTT Broker bị ngắt đột ngột.

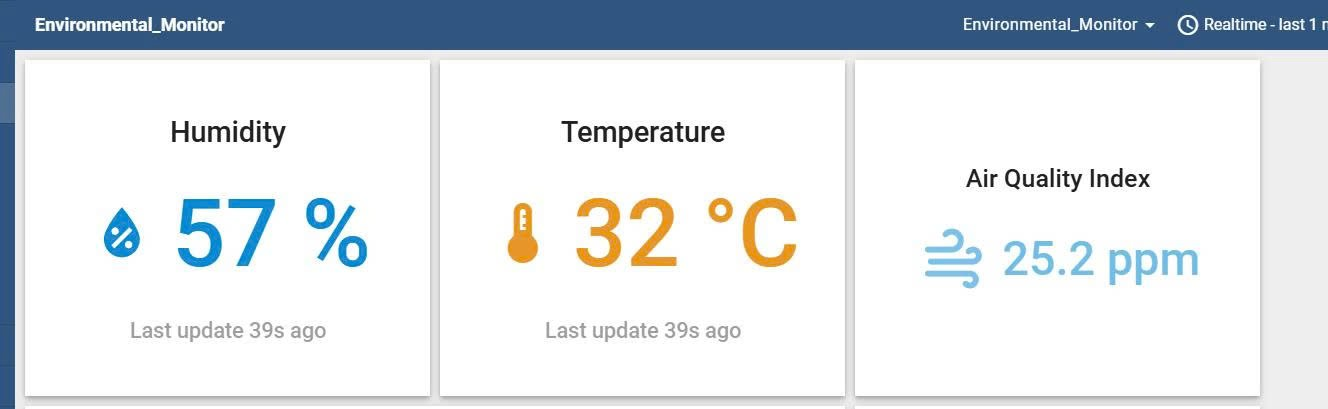
**+ Phương pháp:** Rút dây mạng WiFi Router trong khoảng 30 giây. Quan sát khả năng tự động kết nối lại và không bị treo cứng của hệ thống.

**Kịch bản 4:** Thử nghiệm khả năng chịu tải của cảm biến

**+ Mục tiêu:** Đánh giá độ chính xác của dữ liệu khi nhiều cảm biến cùng chia sẻ bus I2C.

**+ Phương pháp:** So sánh giá trị đọc được từ cảm biến BH1750 (I2C) khi hệ thống hoạt động bình thường và khi đang gửi dữ liệu MQTT với tần suất cao. Mô hình dùng FreeRTOS phải sử dụng Mutex để đồng bộ truy cập I2C.

## Thực Nghiệm



Hình 4.1. Kết quả triển khai trên Thingsboard



Hình 4.2. Gửi dữ liệu lên Thingsboard

# Kết luận - Hướng phát triển

## Kết quả đối chiếu với mục tiêu

Hệ thống đã triển khai thành công các sensor được lập trình bằng FreeRTOS kết hợp mutex.

## Các hạn chế của đồ án

Mặc dù đạt được các mục tiêu cốt lõi, đồ án vẫn còn tồn tại một số hạn chế cần được nhìn nhận:

### Hạn chế về phần cứng và độ chính xác:

Cảm biến MQ-135: Đây là cảm biến bán dẫn, có độ nhạy tương đối với nhiều loại khí (NH3, NOx, CO, Alcohol) nhưng không thể xác định chính xác nồng độ của một loại khí cụ thể (ví dụ: CO2) mà chỉ cung cấp giá trị tổng quát về "chất lượng không khí" (Air Quality). Việc hiệu chuẩn chính xác đòi hỏi môi trường và thiết bị chuyên dụng.

Thời gian phản hồi của cảm biến: Cảm biến MQ-135 cần thời gian gia nhiệt (pre-heating) để cho kết quả ổn định, dẫn đến dữ liệu những phút đầu tiên có thể không chính xác.

### Hạn chế về tính năng:

Giao diện người dùng: Giao diện dashboard trên ThingsBoard mặc dù đầy đủ chức năng nhưng chưa được tùy biến sâu để thân thiện và đẹp mắt hơn với người dùng cuối.

Lưu trữ dữ liệu dự phòng: Dữ liệu chỉ được lưu trữ trên ThingsBoard. Hệ thống chưa có cơ chế lưu trữ dự phòng tại chỗ (ví dụ: trên thẻ SD) để đề phòng trường hợp mất kết nối mạng dài hạn.

### Hạn chế về quy mô thử nghiệm:

Hệ thống mới chỉ được thử nghiệm trong quy mô phòng thí nghiệm (indoor), với các nguồn khí thử nghiệm hạn chế (khói thuốc, khí gas bật lửa). Chưa được đánh giá hiệu năng trong môi trường thực tế rộng lớn và đa dạng hơn (outdoor, nhà xưởng).

## Hướng phát triển

Để khắc phục những hạn chế và nâng cấp hệ thống trong tương lai, một số hướng phát triển được đề xuất:

### Nâng cấp phần cứng:

Thay thế cảm biến chuyên dụng: Sử dụng các cảm biến chuyên dụng cho từng loại khí để có độ chính xác cao hơn. Ví dụ: sử dụng cảm biến SCD40 ho SGP30 để đo nồng độ CO2 và TVOC một cách chính xác.

Bổ sung viên pin và module sạc: Tích hợp pin Li-Po và module sạc để hệ thống có thể hoạt động độc lập, không phụ thuộc vào nguồn USB, giúp triển khai linh hoạt ở nhiều vị trí.

Bổ sung màn hiển thị tại chỗ: Sử dụng màn hình OLED để hiển thị các thông số môi trường ngay trên thiết bị, giúp người dùng quan sát trực tiếp mà không cần xem qua app.

### Phát triển tính năng phần mềm:

Triển khai cảnh báo đa nền tảng: Tích hợp ThingsBoard với các dịch vụ gửi thông báo phổ biến như Telegram, Facebook Messenger hoặc Zalo để gửi cảnh báo tức thì.

Xây dựng ứng dụng di động riêng: Phát triển một ứng dụng di động (Android/iOS) kết nối trực tiếp đến ThingsBoard thông qua REST API để giám sát và điều khiển, thay thế cho dashboard trên web, mang lại trải nghiệm người dùng tốt hơn.

Phát triển thuật toán dự báo: Ứng dụng các mô hình Machine Learning đơn giản (như Linear Regression, ARIMA) trên server để phân tích dữ liệu lịch sử và dự báo xu hướng biến đổi của môi trường trong thời gian tới.

### Mở rộng hệ thống:

Xây dựng mạng lưới cảm biến (Sensor Network): Kết nối nhiều thiết bị giám sát lại với nhau thông qua giao thức không dây tiết kiệm năng lượng như LoRaWAN hoặc Zigbee, tạo thành một mạng lưới bao phủ một khu vực rộng lớn (ví dụ: một tòa nhà, một nông trại).

Tích hợp với các nền tảng Smart Home: Kết nối hệ thống với các nền tảng nhà thông minh phổ biến như Google Home, Amazon Alexa thông qua IFTTT để điều khiển bằng giọng nói và tạo các kịch bản phức tạp hơn.

**TÀI LIỆU THAM KHẢO**

[1] Espressif Systems. (2022). *ESP32 Datasheet Version 4.6* [PDF]. <https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf>

[2] Espressif Systems. (2023). *ESP-IDF Programming Guide*. <https://docs.espressif.com/projects/esp-idf/en/stable/esp32/>

[3] Huang, H., & Li, Q. (2018). *IoT and Smart City: Technologies, Applications and Challenges*. Springer International Publishing. <https://doi.org/10.1007/978-3-319-95037-2>

[4] OASIS. (2019). *MQTT Version 5.0. OASIS Standard*. <https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html>

[5] ROHM Semiconductor. (2015). *BH1750FVI Datasheet: Ambient Light Sensor IC* [PDF]. <https://www.mouser.com/datasheet/2/348/bh1750fvi-e-186247.pdf>

[6] ThingsBoard. (2023). *ThingsBoard Documentation*. <https://thingsboard.io/docs/>

[7] Tran, V. L., & Nguyen, T. H. (2020). A Low-Cost IoT Platform for Environmental Monitoring Based on Microservices Architecture. *2020 5th International Conference on Green Technology and Sustainable Development (GTSD)*, 287-292. <https://doi.org/10.1109/GTSD50082.2020.9303072>