

**ĐẠI HỌC BÁCH KHOA HÀ NỘI**  
**TRƯỜNG ĐIỆN- ĐIỆN TỬ**

\*\*\*\*\* □ □ \*\*\*\*\*



**BÁO CÁO ĐỒ ÁN**

**THIẾT KẾ PHẦN MỀM NHÚNG**

**Đề tài**

**THIẾT KẾ FIRMWARE CHO MẠCH CHẤT LƯỢNG KHÔNG KHÍ**

**Giảng viên hướng dẫn: TS. Hàn Huy Dũng**

**NGUYỄN TIẾN DŨNG**

**20213690**

Hà Nội, 7-2024

## LỜI NÓI ĐẦU

Trong bối cảnh hiện nay, vấn đề chất lượng không khí ngày càng trở nên cấp thiết, ảnh hưởng trực tiếp đến sức khỏe con người và môi trường sống xung quanh. Việc đo lường và theo dõi chất lượng không khí là chìa khóa giúp phát hiện sớm và đối phó hiệu quả với các nguy cơ ô nhiễm. Xu hướng ứng dụng công nghệ thông minh trong giám sát môi trường đang mở rộng, và trong số đó, thiết bị ESP32 là một công cụ vô cùng mạnh mẽ và tiện lợi.

Báo cáo của em tập trung vào việc sử dụng ESP32 - một nền tảng phát triển phần cứng và phần mềm hiệu quả - để thiết kế một hệ thống đo lường chất lượng không khí. ESP32 không chỉ cung cấp khả năng kết nối không dây linh hoạt mà còn hỗ trợ mạnh mẽ trong xử lý tín hiệu và giao tiếp dữ liệu, làm cho việc giám sát không khí trở nên thuận tiện và chính xác hơn.


Qua đề tài này, em mong muốn hiểu sâu hơn về nguyên lý hoạt động của ESP32 và khám phá các ứng dụng của nó trong việc đo và theo dõi chất lượng không khí. Đồng thời, đây cũng là cơ hội để em áp dụng và phát triển các kỹ năng liên quan đến vi xử lý, từ đó nâng cao khả năng tư duy và giải quyết vấn đề trong lĩnh vực công nghệ môi trường.

Em nhận thức được rằng trong quá trình thực hiện báo cáo này, chắc chắn sẽ gặp phải không ít khó khăn và thử thách. Vì vậy, em rất mong nhận được sự góp ý và chỉ dẫn từ thầy Hàn Huy Dũng để có thể hoàn thiện hơn nữa.

Cuối cùng, em xin bày tỏ lòng biết ơn sâu sắc đến thầy Hàn Huy Dũng, người đã không chỉ hướng dẫn mà còn đồng hành cùng em trong suốt quá trình thực hiện báo cáo này. Em xin chân thành cảm ơn thầy!

## Contents

<b>LỜI NÓI ĐẦU .....</b>	<b>2</b>
<b>DANH MỤC KÝ HIỆU VÀ CHỮ VIẾT TẮT .....</b>	<b>8</b>
<b>DANH MỤC HÌNH VẼ .....</b>	<b>9</b>
<b>CHƯƠNG 1: MỞ ĐẦU .....</b>	<b>10</b>
<b>1.1 GIỚI THIỆU .....</b>	<b>10</b>
1.2.1 Tình trạng ô nhiễm không khí .....	11
1.2.2 Ảnh hưởng đến sức khỏe con người .....	11
1.2.3 Ảnh hưởng đến môi trường .....	11
1.2.4 Cần có biện pháp giảm ô nhiễm không khí .....	11
<b>1.3 MỤC TIÊU VÀ MỤC ĐÍCH THỰC HIỆN .....</b>	<b>12</b>
1.3.1 Mục tiêu .....	12
1.3.2 Mục đích .....	12
<b>1.4 ĐỐI TƯỢNG TIÊU DÙNG.....</b>	<b>12</b>
1.4.1 Người quan tâm đến sức khỏe và chất lượng môi trường của gia đình và cộng đồng.....	13
1.4.2 Nhà nghiên cứu, tổ chức môi trường và cơ quan chính phủ .....	13
<b>1.5 SẢN PHẨM TRÊN THỊ TRƯỜNG.....</b>	<b>13</b>
1.5.1 Một số sản phẩm đo chất lượng không khí trên thị trường .....	13
1.5.2 Phân loại sản phẩm đo chất lượng không khí.....	17
1.5.3 Nhược điểm của một số sản phẩm trên thị trường.....	17
1.5.4 Lợi ích của việc sử dụng ESP32 trong đo chất lượng không khí .....	18
<b>CHƯƠNG 2: YÊU CẦU KỸ THUẬT .....</b>	<b>19</b>
<b>2.1 YÊU CẦU CHỨC NĂNG.....</b>	<b>19</b>
2.1.1 Đo lường các chỉ số chất lượng không khí.....	19

2.1.2	Hiển thị kết quả đo .....	19
2.1.3	Cập nhật thông tin thời gian thực .....	19
2.1.4	Cảnh báo và thông báo .....	19
<b>2.2</b>	<b>YÊU CẦU PHI CHỨC NĂNG.....</b>	<b>19</b>
2.2.1	Độ chính xác và độ tin cậy cao.....	19
2.2.2	Kết nối và tích hợp dữ liệu.....	19
2.2.3	Độ bền và khả năng chống thời tiết .....	20
2.2.4	Tiện ích và khả năng tương thích .....	20
2.2.5	Bảo mật và quyền riêng tư .....	20
<b>CHƯƠNG 3:</b>	<b>THIẾT KẾ HỆ THỐNG .....</b>	<b>20</b>
<b>3.1</b>	<b>THIẾT KẾ TỔNG THỂ HỆ THỐNG.....</b>	<b>20</b>
3.1.1	Chỉ tiêu kỹ thuật .....	20
3.1.2	Sơ đồ hệ thống.....	20
3.1.3	Danh sách các linh kiện sử dụng .....	21
<b>3.2</b>	<b>THIẾT KẾ PHẦN CỨNG.....</b>	<b>25</b>
3.2.1	Phân tích công suất.....	25
3.2.2	Sơ đồ nguyên lý mạch .....	26
3.2.3	Lưu đồ thuật toán .....	26
3.2.4	Sơ đồ cắm chân linh kiện.....	27
<b>3.3</b>	<b>THIẾT KẾ PHẦN MỀM.....</b>	<b>28</b>
3.3.1	Sơ đồ khối phần mềm .....	28
3.3.2	Biểu đồ tuần tự .....	28
3.3.4	Framework .....	30
	 <b>ESP-IDF</b>	 .....
		30
<b>CHƯƠNG 4 :</b>	<b>TRIỂN KHAI VÀ KIỂM THỬ.....</b>	<b>31</b>

<b>4.1 Kiểm thử .....</b>	<b>31</b>
4.1.1 Kiểm tra nguồn và các cảm biến .....	31
4.1.2 Kiểm tra thông mạch .....	31
4.1.3 Kiểm thử từng linh kiện .....	31
<b>CHƯƠNG 5 FIRMWARE</b>	
5.1 webpage	
5.1.1 app.css.....	36
5.1.2 app.js.....	37
5.1.3 html.....	38
5.2 DHT22.....	39
5.3 http_server.....	41
5.4 wifi_app.....	44
5.5 task_common.....	47
5.6 main .....	48
5.7 Kết quả .....	50
 <b>KẾT LUẬN.....</b>	 <b>51</b>
 <b>TÀI LIỆU THAM KHẢO.....</b>	 <b>53</b>
 <b>PHỤ LỤC.....</b>	 <b>54</b>
 <b>A. Mã nguồn hệ thống .....</b>	 <b>54</b>

### KẾ HOẠCH HÀNG TUẦN

<b>TUẦN 1</b>	Thu thập thông tin ban đầu và nghiên cứu tổng quan về chất lượng không khí và tầm quan trọng của việc giám sát.
<b>TUẦN 2</b>	Tìm hiểu về các công nghệ hiện có để đo lường chất lượng không khí.

<b>TUẦN 3</b>	Đánh giá các công nghệ và quyết định sử dụng ESP32 cho dự án; thu thập tài liệu kỹ thuật và các nghiên cứu liên quan đến ESP32.
<b>TUẦN 4</b>	Lập kế hoạch chi tiết cho dự án, xác định các mục tiêu cụ thể, và bố trí thời gian cho từng phần.
<b>TUẦN 5</b>	Bắt đầu học cơ bản về lập trình và cấu hình ESP32.
<b>TUẦN 6</b>	Thiết kế sơ đồ mạch điện và lựa chọn các cảm biến phù hợp cho hệ thống.
<b>TUẦN 7</b>	Bắt đầu viết báo cáo đầu tiên về quá trình nghiên cứu và phát triển.
<b>TUẦN 8</b>	Tìm hiểu sâu về ESP32 IDF (IoT Development Framework) và các module nâng cao.
<b>TUẦN 9</b>	Tiếp tục phát triển và hoàn thiện báo cáo, tích hợp các cảm biến với ESP32
<b>TUẦN 10</b>	Bắt đầu phát triển firmware, viết code để thu thập và xử lý dữ liệu từ các cảm biến.
<b>TUẦN 11</b>	Kiểm thử và gỡ lỗi hệ thống, đảm bảo dữ liệu thu thập được chính xác và ổn định.
<b>TUẦN 12</b>	Kiểm thử và gỡ lỗi hệ thống, đảm bảo dữ liệu thu thập được chính xác và ổn định.
<b>TUẦN 13</b>	Viết và cập nhật báo cáo với kết quả từ việc kiểm thử.
<b>TUẦN 14</b>	Học cơ bản về HTML và CSS.
<b>TUẦN 15</b>	Thiết kế và xây dựng giao diện người dùng cho trang web hiển thị dữ liệu chất lượng không khí.

<b>TUẦN 16</b>	Tổng hợp tất cả các phần, chỉnh sửa và hoàn thiện báo cáo cuối cùng.

## DANH MỤC KÝ HIỆU VÀ CHỮ VIẾT TẮT

Tên tiếng Anh	Viết tắt
Espressif System Platform 32	ESP32
Integrated Development Environment	IDE
Liquid crystal Display	LCD



## DANH MỤC HÌNH VẼ

Hình 1 Ô nhiễm không khí ở Hà Nội .....	11
Hình 2 Thiết bị đo Temtop M200 .....	14
Hình 3 Thiết bị đo Atmotube Pro.....	15
Hình 4 Thiết bị đo AirLink .....	15
Hình 5 Thiết bị đo Sensirion SPS30.....	16
Hình 6 Sơ đồ hệ thống.....	21
Hình 7 Cảm biến DHT11 .....	22
Hình 9 Vi xử lý ESP32 .....	23
Hình 10 Màn hình LCD1602 .....	24
Hình 11 Module I2C.....	25
Hình 12 Sơ đồ nguyên lý mạch.....	27
Hình 13 Lưu đồ thuật toán .....	28
Hình 14 Sơ đồ cắm chân.....	28
Hình 15 Sơ đồ khối phần mềm .....	29
Hình 16 Biểu đồ tuần tự.....	30
Hình 17 Framework esp 32 idf .....	<b>Error! Bookmark not defined.</b>
Hình 19 Code firmware .....	<b>Error! Bookmark not defined.1</b>
Hình 20 Code firmware .....	<b>Error! Bookmark not defined.2</b>
Hình 21 Code firmware .....	<b>Error! Bookmark not defined.3</b>
Hình 22 Code firmware .....	<b>Error! Bookmark not defined.4</b>
Hình 23 Kết quả web.....	<b>Error! Bookmark not defined.5</b>

## CHƯƠNG 1: MỞ ĐẦU

### 1.1 GIỚI THIỆU

Chất lượng không khí là một yếu tố quan trọng trong cuộc sống hàng ngày của tôi. Không khí sạch và trong lành đóng vai trò thiết yếu trong việc bảo vệ sức khỏe của bản thân và môi trường sống xung quanh. Tuy nhiên, ô nhiễm không khí là một thách thức đang dần trở nên nghiêm trọng trên toàn cầu.[1]

Ô nhiễm không khí gây ra nhiều vấn đề sức khỏe nghiêm trọng như các bệnh về đường hô hấp, tim mạch và ung thư phổi. Ngoài ra, nó còn ảnh hưởng đến chất lượng cuộc sống, gây ra khói bụi, mùi hôi,... Do đó, việc đo và giám sát chất lượng không khí là vô cùng cần thiết để nắm bắt tình hình ô nhiễm nhằm áp dụng các biện pháp phòng ngừa và kiểm soát.

Đo và giám sát chất lượng không khí cung cấp thông tin cần thiết để đánh giá mức độ ô nhiễm, xác định nguồn gốc và loại bỏ các chất gây ô nhiễm. Điều này giúp các cơ quan chính phủ, tổ chức môi trường và cộng đồng hiểu rõ hơn về tình trạng môi trường xung quanh và áp dụng các biện pháp phù hợp để bảo vệ sức khỏe cộng đồng cũng như duy trì một môi trường sống lành mạnh.[2]

Trong bối cảnh này, báo cáo của tôi tập trung vào việc phát triển một hệ thống đo chất lượng không khí sử dụng ESP32. Tôi nhận thấy rằng việc sử dụng công nghệ và nền tảng phát triển phần cứng thông minh như ESP32 có thể cung cấp một giải pháp hiệu quả và tiện lợi cho việc đo và giám sát chất lượng không khí. Sản phẩm được tạo ra sẽ giúp người dùng đo lường các chỉ số quan trọng về chất lượng không khí, thu thập dữ liệu và cung cấp thông tin cần thiết để đưa ra quyết định và hành động thích hợp.

### 1.2 Đặt vấn đề

Hiện nay, ô nhiễm không khí là một vấn đề nghiêm trọng đang ảnh hưởng đến sức khỏe và môi trường sống của con người trên toàn cầu. Sau đây là phân tích

về tình hình ô nhiễm không khí hiện nay và những vấn đề liên quan:



*Hình 1 Ô nhiễm không khí ở Hà Nội*

### **1.2.1 Tình trạng ô nhiễm không khí**

Ô nhiễm không khí đang ngày càng trở nên nghiêm trọng ở nhiều thành phố và khu vực trên thế giới. Các chất gây ô nhiễm như hạt bụi mịn (PM2.5 và PM10), khí thải từ phương tiện giao thông, năng lượng sản xuất và tiêu thụ, công nghiệp, nông nghiệp và đốt cháy rừng đều góp phần vào tình trạng ô nhiễm này.

### **1.2.2 Ảnh hưởng đến sức khỏe con người**

Ô nhiễm không khí gây ra nhiều vấn đề sức khỏe, bao gồm viêm đường hô hấp, các bệnh về tim mạch, ung thư phổi, suy giảm chức năng phổi, và tăng nguy cơ về các bệnh mãn tính như hen suyễn và bệnh phổi tắc nghẽn mạn tính (COPD). Trẻ em, người già và những người có hệ miễn dịch yếu đều là nhóm người có nguy cơ cao hơn khi tiếp xúc với không khí ô nhiễm.

### **1.2.3 Ảnh hưởng đến môi trường**

Ô nhiễm không khí gây tổn hại đáng kể đến môi trường tự nhiên. Nó có thể gây axit hóa đất, ô nhiễm các nguồn nước và sông suối thông qua hiện tượng mưa axit. Ngoài ra, ô nhiễm không khí cũng góp phần vào hiện tượng biến đổi khí hậu toàn cầu, ảnh hưởng đến hệ sinh thái và đa dạng sinh học.

### **1.2.4 Cần có biện pháp giảm ô nhiễm không khí**

Trong việc đối phó với ô nhiễm không khí, việc đo và giám sát chất lượng không khí đóng vai trò vô cùng quan trọng. Tuy nhiên, hiện nay vẫn còn tồn tại nhiều vấn đề liên quan đến đo và giám sát chất lượng không khí như độ chính xác, độ tin cậy, giá thành sản phẩm hay khả năng kết nối và truyền tải dữ liệu,...

### **1.3 MỤC TIÊU VÀ MỤC ĐÍCH THỰC HIỆN**

#### **1.3.1 Mục tiêu**

Mục tiêu chính của báo cáo là phát triển một hệ thống đo chất lượng không khí sử dụng ESP32, một module phát triển được tích hợp với vi xử lý và kết nối không dây. ESP32 là một module mạnh mẽ và linh hoạt, cho phép thiết kế một công cụ đo chất lượng không khí đơn giản, nhỏ gọn và tiện lợi.

#### **1.3.2 Mục đích**

- Thiết kế một công cụ đơn giản, nhỏ gọn và tiện lợi cho việc đo và theo dõi chất lượng không khí: Em nhằm tạo ra một sản phẩm đo chất lượng không khí có kích thước nhỏ gọn, dễ sử dụng và mang tính di động cao. Điều này giúp người dùng có khả năng đo chất lượng không khí ở nhiều vị trí khác nhau và trong thời gian thực.
- Thu thập dữ liệu từ các cảm biến và hiển thị thông tin chất lượng không khí qua giao diện người dùng: Hệ thống sẽ được trang bị các cảm biến như cảm biến khí CO<sub>2</sub>, cảm biến bụi mịn, cảm biến nhiệt độ và độ ẩm. Chúng tôi nhằm thu thập dữ liệu từ các cảm biến này và hiển thị thông tin chất lượng không khí một cách trực quan thông qua giao diện người dùng, giúp người dùng có cái nhìn tổng quan về chất lượng không khí xung quanh.
- Cung cấp giải pháp đáng tin cậy và tiết kiệm chi phí cho việc đo chất lượng không khí: Với việc sử dụng ESP32, em mong muốn tạo ra một giải pháp đo chất lượng không khí đáng tin cậy và tiết kiệm chi phí so với các sản phẩm trên thị trường. Điều này giúp người dùng dễ dàng tiếp cận và sử dụng công cụ đo chất lượng không khí một cách hiệu quả.

Thông qua việc đạt được các mục tiêu và mục đích nêu trên, em hy vọng rằng hệ thống đo chất lượng không khí sử dụng ESP32 của em sẽ đóng góp vào việc cải thiện nhận thức về chất lượng không khí, hỗ trợ người dùng trong việc đo và giám sát môi trường sống của mình, và tạo ra một giải pháp đơn giản và tiện lợi trong lĩnh vực này.

### **1.4 ĐỐI TƯỢNG TIÊU DÙNG**

### **1.4.1 Người quan tâm đến sức khỏe và chất lượng môi trường của gia đình và cộng đồng**

Hệ thống đo chất lượng không khí sử dụng ESP32 sẽ phục vụ những người quan tâm đến sức khỏe và chất lượng môi trường sống của gia đình và cộng đồng. Đây là những người muốn đảm bảo rằng không khí mà họ hít thở hàng ngày là trong lành và không gây hại cho sức khỏe. Họ có quan tâm đặc biệt đến việc đo và giám sát chất lượng không khí trong nhà, văn phòng, trường học hoặc các khu vực công cộng để đưa ra các biện pháp cần thiết để bảo vệ sức khỏe của mình và gia đình.

### **1.4.2 Nhà nghiên cứu, tổ chức môi trường và cơ quan chính phủ**

Đối tượng tiêu dùng tiếp theo là những nhà nghiên cứu, tổ chức môi trường và cơ quan chính phủ có nhu cầu thu thập dữ liệu chất lượng không khí để nghiên cứu, theo dõi và đánh giá tình trạng ô nhiễm không khí trong một khu vực cụ thể. Hệ thống đo chất lượng không khí sử dụng ESP32 sẽ cung cấp cho họ dữ liệu chính xác và đáng tin cậy để phân tích và đưa ra các biện pháp khắc phục ô nhiễm không khí, đồng thời giúp theo dõi hiệu quả các chính sách và biện pháp quản lý môi trường.

Đối tượng tiêu dùng này có kiến thức chuyên môn về chất lượng không khí và có nhu cầu sử dụng công nghệ để thu thập dữ liệu và phân tích chất lượng không khí một cách chi tiết và khoa học. Hệ thống đo chất lượng không khí sử dụng ESP32 sẽ đáp ứng được nhu cầu của họ với tính linh hoạt và khả năng tương tác cao.

Với việc đáp ứng được nhu cầu của cả người dân thông thường và các chuyên gia, hệ thống đo chất lượng không khí sử dụng ESP32 hứa hẹn mang lại lợi ích rất lớn cho đối tượng tiêu dùng và cả xã hội nói chung.

## **1.5 SẢN PHẨM TRÊN THỊ TRƯỜNG**

### **1.5.1 Một số sản phẩm đo chất lượng không khí trên thị trường**

Hiện nay có nhiều sản phẩm đo chất lượng không khí trên thị trường, phù hợp với các nhu cầu và yêu cầu khác nhau của người dùng. Dưới đây là một số sản phẩm phổ biến [3]:

- **Temtop M2000**



*Hình 2 Thiết bị đo Temtop M200*

Temtop M2000 đo nhiều loại chất gây ô nhiễm, có tích hợp tính năng hiệu chuẩn và đi kèm với cảnh báo âm thanh.

**Thông số kỹ thuật:**

Đo lường: CO<sub>2</sub>, HCHO (formaldehyde), PM<sub>2.5</sub>/PM<sub>10</sub>, Nhiệt độ, Độ ẩm

Đánh giá của AQ-SPEC: Tính nhất quán cao

Loại: Cầm Tay

**Ưu điểm:**

Có thể kiểm tra nhiều thông số trên một thiết bị

Tính nhất quán cao trên nhiều thiết bị

Có báo động âm thanh khi các thông số vượt mức

Đã được thử và kiểm tra bởi một số hệ thống trường học

**Nhược điểm:**

Không hoạt động ở nhiệt độ dưới mức đóng băng

• **Atmotube Pro**



*Hình 3 Thiết bị đo Atmotube Pro*

Atmotube Pro được gắn trực tiếp vào ba lô hoặc thắt lưng và gửi thông báo trực tiếp tới điện thoại của người dùng khi có sự cố.

### **Thông số kỹ thuật**

Đo: PM1/PM2.5/PM10, VOC, Nhiệt độ, Độ ẩm, Áp suất khí quyển

Đánh giá của AQ-SPEC: Độ chính xác cao.

Loại: Mang theo bên người

### **Ưu điểm**

Độ chính xác ấn tượng.

Có thể cảnh báo qua ứng dụng.

Hoạt động tốt trên nhiều nhiệt độ và độ ẩm khác nhau.

Kích thước nhỏ gọn.

### **Nhược điểm**

Tuổi thọ pin ngắn.

- **AirLink**



*Hình 4 Thiết bị đo AirLink*

AirLink có thể sử dụng cả trong nhà và ngoài trời, cung cấp các biểu đồ đẹp mắt và hoạt động tốt với các hệ thống khác của Davis Instruments.

### **Thông số kỹ thuật**

Các biện pháp: AQI, Nhiệt độ, Độ ẩm, Điểm sương, Chỉ số nhiệt

Báo cáo AQ-SPEC: Độ chính xác cao đối với PM1.0/PM2.5, nhưng không phải đối với PM10, nhất quán

Loại: Trong nhà hoặc ngoài trời

### **Ưu điểm**

Tích hợp với Weatherlink.com.

Đồ thị và biểu đồ hiển thị dễ đọc.

Thiết lập dễ dàng cả trong nhà và ngoài trời.

Tích hợp vào trạm thời tiết Davis của người dùng.

### **Nhược điểm**

Chỉ số PM10 đo được có độ chính xác thấp.

- **Sensirion SPS30**



*Hình 5 Thiết bị đo Sensirion SPS30*

Sensirion SPS30 là một máy đo đơn giản dành riêng cho kiểm tra vật chất dạng hạt.

### **Thông số kỹ thuật**

Biện pháp: PM1.0/PM2.5/PM4.0/PM10



Báo cáo AQ-SPEC: Độ chính xác cao

Loại: Di động

### **Ưu điểm**

Kích thước nhỏ, gọn

Tuổi thọ dài (8 năm)

Ổn định lâu dài và chống ô nhiễm

có thể sửa đổi

### **Nhược điểm**

Giao diện người dùng nhàm chán làm cho việc hiểu dữ liệu trở nên khó khăn.

## **1.5.2 Phân loại sản phẩm đo chất lượng không khí**

Qua khảo sát, chúng em nhận thấy trên thị trường hiện nay, có sự đa dạng về sản phẩm đo chất lượng không khí, và được phân thành hai loại chính: cảm biến di động và thiết bị cố định.

- **Cảm biến di động:** Đây là những sản phẩm nhỏ gọn và dễ dàng di chuyển, cho phép người dùng đo chất lượng không khí tại nhiều vị trí khác nhau. Cảm biến di động thường được tích hợp vào các thiết bị thông minh như điện thoại di động hoặc đồng hồ thông minh, và thông qua các ứng dụng đo chất lượng không khí, người dùng có thể thu thập và phân tích dữ liệu. Tuy nhiên, cảm biến di động thường có giới hạn về độ chính xác và khả năng đo, và không thể cung cấp dữ liệu liên tục và chi tiết trong thời gian dài.
- **Thiết bị cố định:** Đây là những sản phẩm được cài đặt tại một vị trí cố định để theo dõi chất lượng không khí trong khu vực đó. Thiết bị cố định thường có kích thước lớn hơn và được trang bị các cảm biến và công nghệ tiên tiến hơn để đo và phân tích chất lượng không khí. Chúng thường được sử dụng trong các trạm quan trắc môi trường và các điểm theo dõi chất lượng không khí trong thành phố. Tuy nhiên, thiết bị cố định có nhược điểm là cần phải cài đặt và duy trì tại một vị trí cố định, và không thể mang đi di chuyển như cảm biến di động.

## **1.5.3 Nhược điểm của một số sản phẩm trên thị trường**

Mặc dù có sự đa dạng về sản phẩm đo chất lượng không khí trên thị trường, nhưng một số sản phẩm cũng gặp phải nhược điểm sau:

- **Độ chính xác không cao:** Một số sản phẩm đo chất lượng không khí có độ chính xác không đảm bảo, dẫn đến kết quả đo không chính xác và không tin cậy. Điều này có thể gây ra sự nhầm lẫn và sai lệch trong đánh giá chất lượng không khí.
- **Hạn chế trong việc đo các chất độc hại:** Một số sản phẩm chỉ tập trung vào việc

đo một số chỉ tiêu cơ bản của chất lượng không khí như PM2.5, PM10, CO2, trong khi không thể đo được các chất gây ô nhiễm khác như CO, SO2, NO2, O3, VOCs và các chất độc hại khác. Điều này giới hạn khả năng phân tích và đánh giá toàn diện về chất lượng không khí.

- Hạn chế về thời gian sử dụng: Một số sản phẩm đo chất lượng không khí có tuổi thọ pin ngắn, thường chỉ từ khoảng 6 đến 24 giờ

#### **1.5.4 Lợi ích của việc sử dụng ESP32 trong đo chất lượng không khí**

Báo cáo sẽ sử dụng ESP32, một mạch tích hợp Wi-Fi và Bluetooth, để xây dựng hệ thống đo chất lượng không khí. Việc sử dụng ESP32 mang lại nhiều lợi ích như sau:

- Kết nối không dây: ESP32 cho phép kết nối không dây với mạng Wi-Fi và các thiết bị khác như điện thoại di động và máy tính bảng. Điều này giúp thu thập và chia sẻ dữ liệu chất lượng không khí một cách thuận tiện và nhanh chóng.
- Độ ổn định và độ tin cậy cao: ESP32 được thiết kế để đảm bảo độ ổn định và độ tin cậy cao, giúp đảm bảo rằng hệ thống đo chất lượng không khí hoạt động một cách liên tục và chính xác trong thời gian dài.
- Linh hoạt và mở rộng: ESP32 cho phép mở rộng và tùy chỉnh các chức năng và cảm biến theo nhu cầu. Người dùng có thể tùy chỉnh hệ thống đo chất lượng không khí theo nhu cầu cụ thể của họ và thêm các tính năng bổ sung nếu cần thiết.
- Tiết kiệm chi phí: ESP32 là một giải pháp chi phí hiệu quả so với một số sản phẩm đo chất lượng không khí khác trên thị trường. Việc sử dụng ESP32 trong đồ án giúp giảm thiểu chi phí đầu tư và vận hành, làm cho sản phẩm đo chất lượng không khí trở nên phổ biến và dễ tiếp cận hơn.

## **CHƯƠNG 2: YÊU CẦU KỸ THUẬT**

### **2.1 YÊU CẦU CHỨC NĂNG**

#### **2.1.1 Đo lường các chỉ số chất lượng không khí**

Hệ thống có khả năng đo lường các chỉ số quan trọng về chất lượng không khí như PM2.5, CO, nhiệt độ, độ ẩm

Hệ thống có khả năng điều khiển các cảm biến tự động lấy mẫu theo chu kỳ các chỉ số về chất lượng không khí, với dải đo mong muốn như sau:

- Nhiệt độ: 0-50°C, sai số 4%
- Độ ẩm: 20-90 %RH  $\pm$  5%RH
- Nồng độ khí gas: 10-600 ppm

#### **2.1.2 Hiển thị kết quả đo**

Hệ thống có giao diện người dùng trực quan để hiển thị kết quả đo chất lượng không khí. Các thông số đo được hiển thị một cách rõ ràng và dễ hiểu cho người dùng.

Màn hình hiển thị các thông số đo và trạng thái của thiết bị: nhiệt độ, độ ẩm, nồng độ khí gas.

#### **2.1.3 Cập nhật thông tin thời gian thực**

Hệ thống sẽ cung cấp thông tin chất lượng không khí trong thời gian thực, cho phép người dùng theo dõi tình trạng không khí ngay lập tức và đưa ra các quyết định phù hợp.

#### **2.1.4 Cảnh báo và thông báo**

Hệ thống có khả năng cảnh báo và thông báo cho người dùng khi chất lượng không khí vượt quá mức cho phép hoặc đạt đến ngưỡng nguy hiểm.

### **2.2 YÊU CẦU PHI CHỨC NĂNG**

#### **2.2.1 Độ chính xác và độ tin cậy cao**

Độ chính xác: Cung cấp kết quả đo chính xác và đáng tin cậy với sai số từ cảm biến mong muốn khoảng  $\pm 1.5\%$  đến  $\pm 5\%$ .

Thời gian đáp ứng cảm biến thấp (< 3 giây)

#### **2.2.2 Kết nối và tích hợp dữ liệu**

Hệ thống có khả năng kết nối và tích hợp dữ liệu từ các cảm biến và thiết bị đo khác nhau. Điều này cho phép thu thập thông tin từ nhiều nguồn và tích hợp dữ liệu để cung cấp cái nhìn toàn diện về chất lượng không khí.

### **2.2.3 Độ bền và khả năng chống thời tiết**

Hệ thống có khả năng chịu được các điều kiện thời tiết khắc nghiệt và đảm bảo hoạt động liên tục trong thời gian dài mà không bị ảnh hưởng.

### **2.2.4 Tiện ích và khả năng tương thích**

Thiết kế nhỏ gọn với kích thước của thiết bị mong muốn là 17x6.5x5 (cm) Sử dụng adapter 5V,1A để cấp nguồn

Màn hình có kích thước 80 x 36 x 12.5 (mm), giúp người dùng dễ dàng xem thông số chất lượng không khí một cách rõ ràng và thuận tiện.

Hệ thống có khả năng tương thích với các nền tảng và thiết bị khác nhau để người dùng có thể truy cập thông tin chất lượng không khí một cách thuận tiện và dễ dàng.

### **2.2.5 Bảo mật và quyền riêng tư**

Hệ thống bảo đảm an toàn và bảo mật dữ liệu chất lượng không khí và đảm bảo quyền riêng tư cho người dùng.

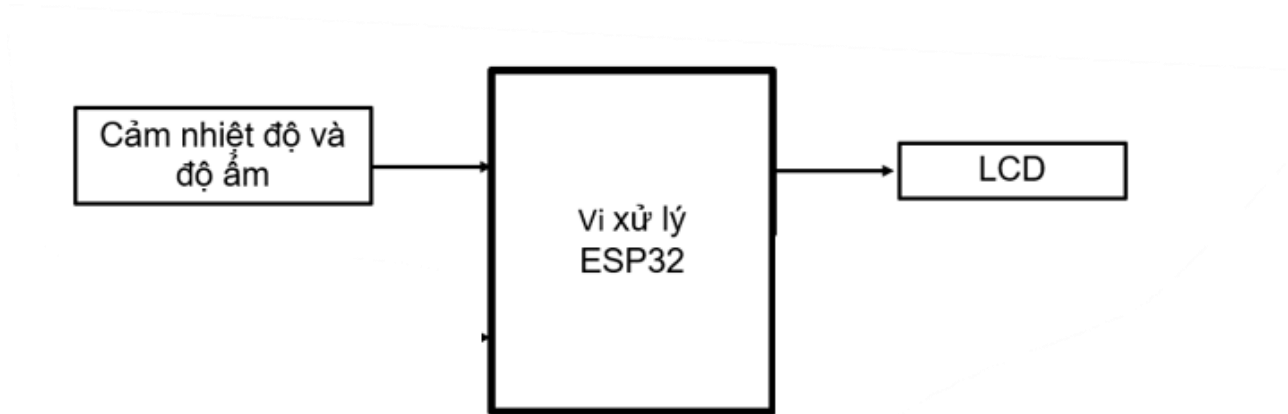
## **CHƯƠNG 3: THIẾT KẾ HỆ THỐNG**

### **3.1 THIẾT KẾ TỔNG THỂ HỆ THỐNG**

#### **3.1.1 Chỉ tiêu kỹ thuật**

- Đo nhiệt độ môi trường phạm vi từ -40 đến 90 độ C với sai số 2°C
- Đo độ ẩm không khí 20 đến 90% với sai số 5%
- Hiển thị lên màn hình giá trị của các đối tượng cần đo
- Cảnh báo nếu có chỉ số vượt ngưỡng cho phép
- Dữ liệu được kết nối và kiểm soát từ xa

#### **3.1.2 Sơ đồ hệ thống**



*Hình 6 Sơ đồ hệ thống*

Ý tưởng của em là sẽ sử dụng các cảm biến nhiệt độ, độ ẩm, khí gas để thu thập data và gửi về ESP32, sau khi xử lý data được lưu vào SD card và hiển thị ra màn hình LCD và webserver

- Thông tin đo được từ các cảm biến được truyền về ESP32 thông qua độ lớn điện áp.
- Thông qua độ lớn điện áp, ta sẽ tính được các thông tin cần đo như nhiệt độ, độ ẩm, nồng độ khí gas.
- Thông tin đó sẽ được hiển thị lên trên màn hình LCD.

### 3.1.3 Danh sách các linh kiện sử dụng

STT	Tên linh kiện	Số lượng
1	ESP32 DOIT DEVKIT V1	1
2	DHT11	1
3	LCD1602	1

*Bảng 1 Danh sách linh kiện*

#### **a. Khối nguồn**

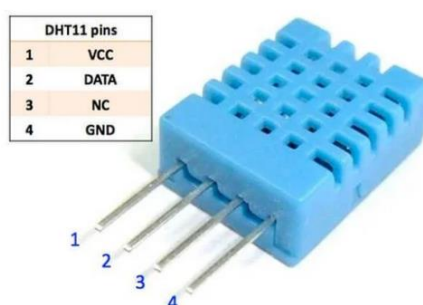
- Nguồn nuôi 5V cung cấp cho mạch thông qua dây cáp kết nối tới máy tính
- USB kết nối mạch tới máy tính

#### **b. Khối cảm biến**

- **DHT11: Cảm biến nhiệt độ, độ ẩm [4]**

Tiêu chí	DHT11	BME280	SHT85
Đơn giá	20,000 VNĐ	150,000 VNĐ	800,000 VNĐ
Kích cỡ	28 x 12mm	13 x 10.5mm	10.5 x 5.08mm
Độ chính xác	$\pm 4\%$	$\pm 3\%$	$\pm 1.8\%$
Thông số đo	Nhiệt độ, độ ẩm	Nhiệt độ, độ ẩm, áp suất	Nhiệt độ, độ ẩm

*Bảng 2 So sánh các loại cảm biến nhiệt độ, độ ẩm*



*Hình 7 Cảm biến DHT11*

Từ những yêu cầu kỹ thuật ở phần trước kết hợp với bảng so sánh ở trên, nhóm chúng em đã quyết định lựa chọn cảm biến **DHT11** bởi cảm biến đo được 2 thông số là nhiệt độ, độ ẩm theo mong muốn. Bên cạnh đó là kích thước nhỏ gọn và độ chính xác nằm trong khoảng mong muốn ( $\pm 5\%$ ) nhưng lại có giá thành thấp hơn nhiều các cảm biến còn lại rất nhiều và dễ mua

DHT11 là một cảm biến nhiệt độ và độ ẩm tương đối. Nó cung cấp đầu ra kỹ thuật số và có khả năng đo nhiệt độ trong khoảng từ  $0^{\circ}\text{C}$  đến  $50^{\circ}\text{C}$  phù hợp với đo nhiệt độ ở Việt Nam và độ ẩm từ 20% đến 90%.

### *c. Khối xử lý*

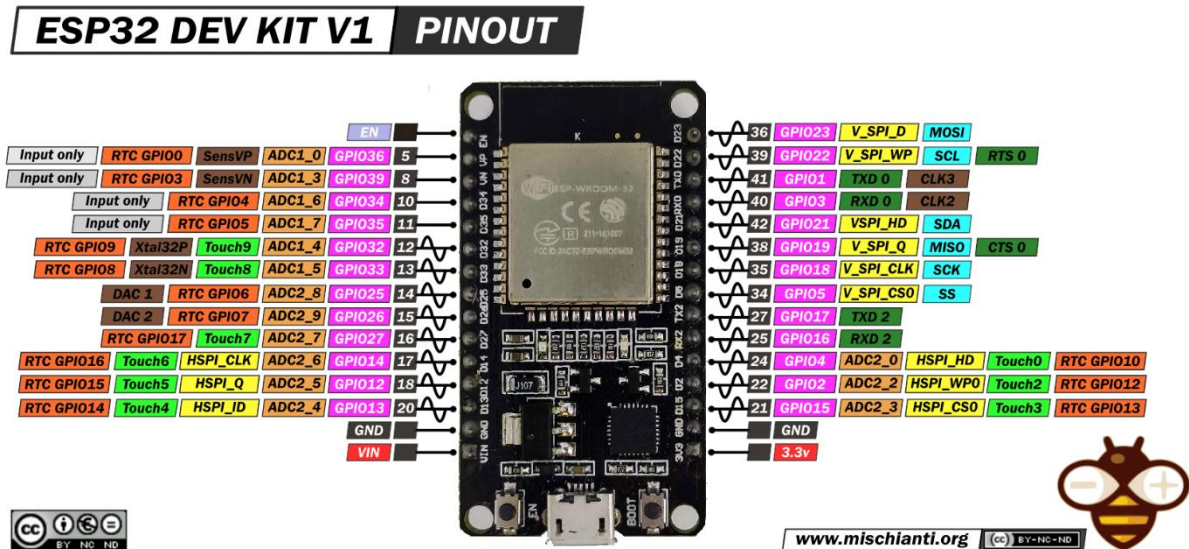
- **ESP32 DOIT DEVKIT V1 [6]**

**Một vài thông số cơ bản của ESP32:**

Thông số kỹ thuật	ESP32
MCU	Xtensa@ Dual-Core 32-bit LX6 600 DMIPS
802.11b/g/n Wifi	Yes, HT40
Bluetooth	Bluetooth 4.2 and below
Typical Frequency	160Mhz

SRAM	512 kbytes
Flash	SPI Flash, up to 16 MBytes
GPIO	36
Hardware/ Software PWM	1/16 Channels
SPI/I2C/I2S/UART	4/2/2/2
Working Temperature	-40°C to 125°C

Bảng 3 Một vài thông số cơ bản của esp32

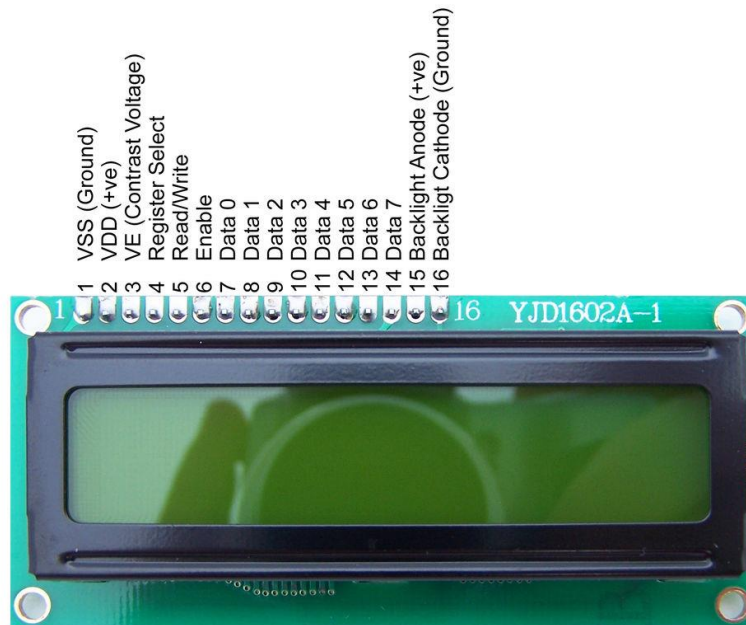


Hình 8 Vi xử lý ESP32

ESP32 Doit devkit v1 là một bo mạch phát triển dựa trên vi điều khiển ESP32. ESP32 là một vi điều khiển WiFi và Bluetooth tích hợp, cung cấp khả năng kết nối không dây và xử lý cao. Bo mạch ESP32 Doit devkit v1 có các giao tiếp ngoại vi phong phú và hỗ trợ việc phát triển ứng dụng IoT và các dự án liên quan.

#### d. Khởi hiển thị

- Màn hình LCD 1602 Xanh Lá [7]



*Hình 9 Màn hình LCD1602*

I2C LCD là một màn hình LCD có giao tiếp I2C (Inter-Integrated Circuit). Giao tiếp I2C cho phép truyền dữ liệu giữa mạch điều khiển chính và màn hình LCD thông qua chỉ cần sử dụng một số chân kết nối. Màn hình LCD I2C rất phổ biến trong các dự án điều khiển và hiển thị dữ liệu từ vi điều khiển mà không cần sử dụng quá nhiều chân I/O.

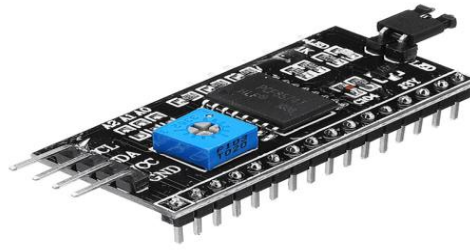
❖ Thông số kỹ thuật

- Điện áp hoạt động là 5V.
- Kích thước: 80 x 36 x 12.5mm
- Chữ trắng, nền xanh dương
- Khoảng cách giữa hai chân kết nối là 0.1 inch tiện dụng khi kết nối với Breadboard.
- Tên các chân được ghi ở mặt sau của màn hình LCD hỗ trợ việc kết nối, đi dây điện.
- Có đèn led nền, có thể dùng biến trở hoặc PWM điều chỉnh độ sáng để sử dụng ít điện năng hơn.
- Có thể được điều khiển với 6 dây tín hiệu

• **Module điều khiển màn hình giao tiếp I2C**

Để thuận tiện cho việc thiết kế phần cứng chúng ta kết hợp lcd với module điều khiển màn hình giao tiếp I2C: I2C Adapter dùng cho LCD 16x2 và LCD16x4 giúp cho việc giao tiếp với LCD trở nên đơn giản hơn, với chuẩn i2C chỉ cần dùng 2 dây thay vì 4 dây hay 8 dây như cách thông thường. Module sử dụng chip PCF8574.





Hình 10 Module I2C

#### ❖ Thông số kĩ thuật

- 16x2 LCD chỉ cần dùng 2 dây qua chuẩn I2C.
- Sử dụng 8 LCD hiển thị cùng lúc với bus I2C .
- Module đi kèm header 16 chân đơn giản chỉ cần hàn vào module LCD.
- 1 jumper điều chỉnh ON/OFF đèn màn hình.
- Module tích hợp biến trở điều chỉnh độ tương phản.
- Nguồn sử dụng cho module là 5V.

### 3.2 THIẾT KẾ PHẦN CỨNG

#### 3.2.1 Phân tích công suất

##### 3.2.1.1 Mạch sensor DHT11

	Conditions	Minimum	Typical	Maximum
Power Supply	DC	3V	5V	5.5V
Current Supply	Measuring	0.5mA		2.5mA
	Average	0.2mA		1mA
	Standby	100uA		150uA
Sampling period	Second	1		

Bảng 4 DHT11 datasheet

Mạch của nhóm sử dụng điện áp 5 V để cung cấp cho cảm biến. Vì thế, công suất tối đa mà cảm biến tiêu thụ được tính như sau:

$$P_{DHT11max} = 5 \times 2.5 \times 10^{-3} = 12.5 \times 10^{-3} (W)$$

##### 3.2.1.3 Vi xử lý ESP32

Power mode	Description	Power Consumption
Active (RF working)	Wi-Fi Tx packet	Please refer to Table 4-4 for details.
	Wi-Fi/BT Tx packet	
	Wi-Fi/BT Rx and listening	
Modem-sleep	The CPU is powered up.	240 MHz * Dual-core chip(s) 30 mA ~ 68 mA
		Single-core chip(s) N/A
		160 MHz * Dual-core chip(s) 27 mA ~ 44 mA
		Single-core chip(s) 27 mA ~ 34 mA
		Normal speed: 80 MHz Dual-core chip(s) 20 mA ~ 31 mA
Light-sleep		Single-core chip(s) 20 mA ~ 25 mA
		0.8 mA
Deep-sleep	The ULP coprocessor is powered up.	150 μA
	ULP sensor-monitored pattern	100 μA @1% duty
	RTC timer + RTC memory	10 μA
Hibernation	RTC timer only	5 μA
Power off	CHIP_PU is set to low level, the chip is powered down.	1 μA

#### 4.3 DC Characteristics (3.3 V, 25 °C)

Parameter	Description	Min	Typ	Max	Unit
C <sub>IN</sub>	Pin capacitance	—	2	—	pF
V <sub>IH</sub>	High-level input voltage	0.75×VDD <sup>1</sup>	—	VDD <sup>1</sup> +0.3	V
V <sub>IL</sub>	Low-level input voltage	0.3	—	0.25×VDD <sup>1</sup>	V
I <sub>IN</sub>	High-level input current	—	—	50	nA
I <sub>IL</sub>	Low-level input current	—	—	50	nA
V <sub>OIH</sub>	High-level output voltage	0.8×VDD <sup>1</sup>	—	—	V
V <sub>OL</sub>	Low-level output voltage	—	—	0.1×VDD <sup>1</sup>	V
I <sub>OAH</sub>	High-level source current (VDD <sup>1</sup> = 3.3 V, V <sub>OIH</sub> >= 2.64 V, output drive strength set to the maximum)	VDD3P3_CPU power domain <sup>1, 2</sup>	—	40	mA
			—	40	mA
			—	20	mA
I <sub>OL</sub>	Low-level sink current (VDD <sup>1</sup> = 3.3 V, V <sub>OL</sub> = 0.485 V, output drive strength set to the maximum)	VDD3P3_RTC power domain <sup>1, 2</sup>	—	28	mA
			—	45	mA
R <sub>PU</sub>	Resistance of internal pull-up resistor	—	45	—	kΩ
R <sub>PD</sub>	Resistance of internal pull-down resistor	—	45	—	kΩ
V <sub>IL,CHIP</sub>	Low-level input voltage of CHIP_PU to shut down the chip	—	—	0.6	V

Table 4-4. Current Consumption Depending on RF Modes

Work Mode	Min	Typ	Max	Unit
Transmit 802.11b, DSSS 1 Mbps, POUT = +19.5 dBm	—	240	—	mA
Transmit 802.11g, OFDM 54 Mbps, POUT = +16 dBm	—	190	—	mA
Transmit 802.11n, OFDM MCS7, POUT = +14 dBm	—	180	—	mA
Receive 802.11b/g/n	—	95 ~ 100	—	mA
Transmit BT/BLE, POUT = 0 dBm	—	130	—	mA
Receive BT/BLE	—	95 ~ 100	—	mA

Bảng 5 ESP32 datasheet

Theo datasheet, ta thấy ESP32 tại trạng thái hoạt động (activate) sẽ tiêu thụ lớn nhất. Nó bao gồm các thông số:

Utiêu thụ = 3.3V, I<sub>max</sub> = 240 mA

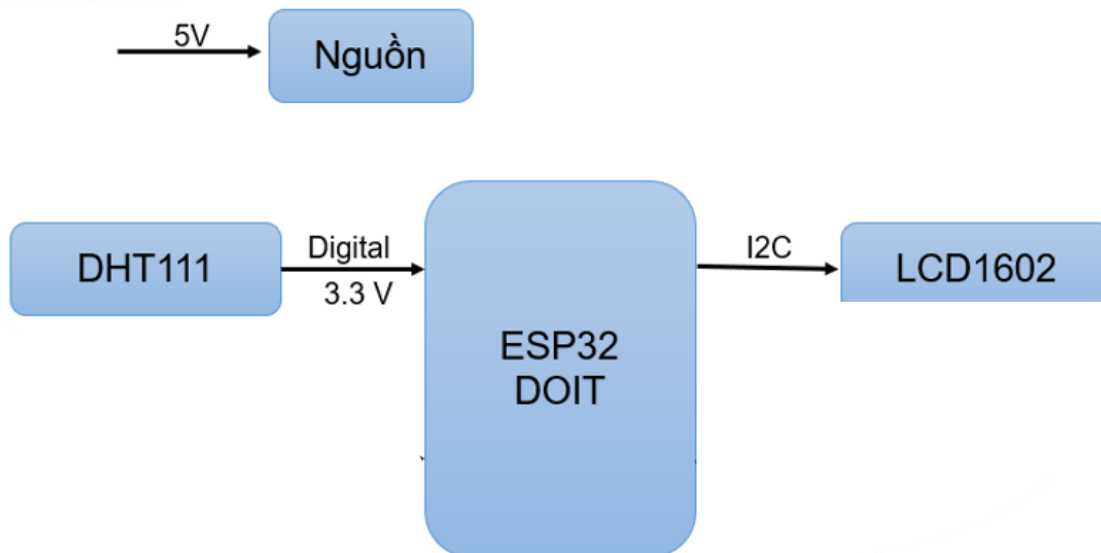
Ta có công thức:  $P = UI$ . Nên ta có công suất tiêu thụ là:

$$P_{\max} = 3.3 \times 240 \times 10^{-3} = 792 \text{ (mW)}$$

**Công suất tiêu thụ tổng cho các linh kiện:**

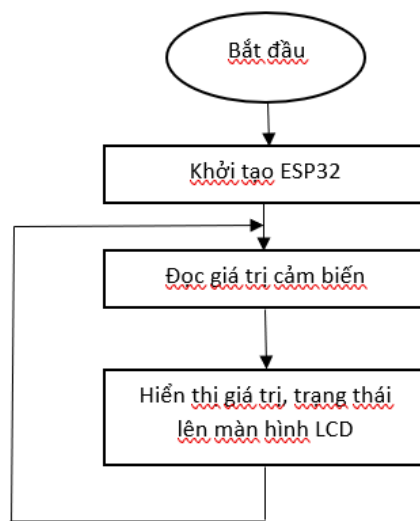
$$P = P_{ESP32} + P_{MQ135} + P_{DHT11} = 792 \times 10^{-3} + 800 \times 10^{-3} + 12.5 \times 10^{-3} = 1.604 \text{ (W)}$$

### 3.2.2 Sơ đồ nguyên lý mạch



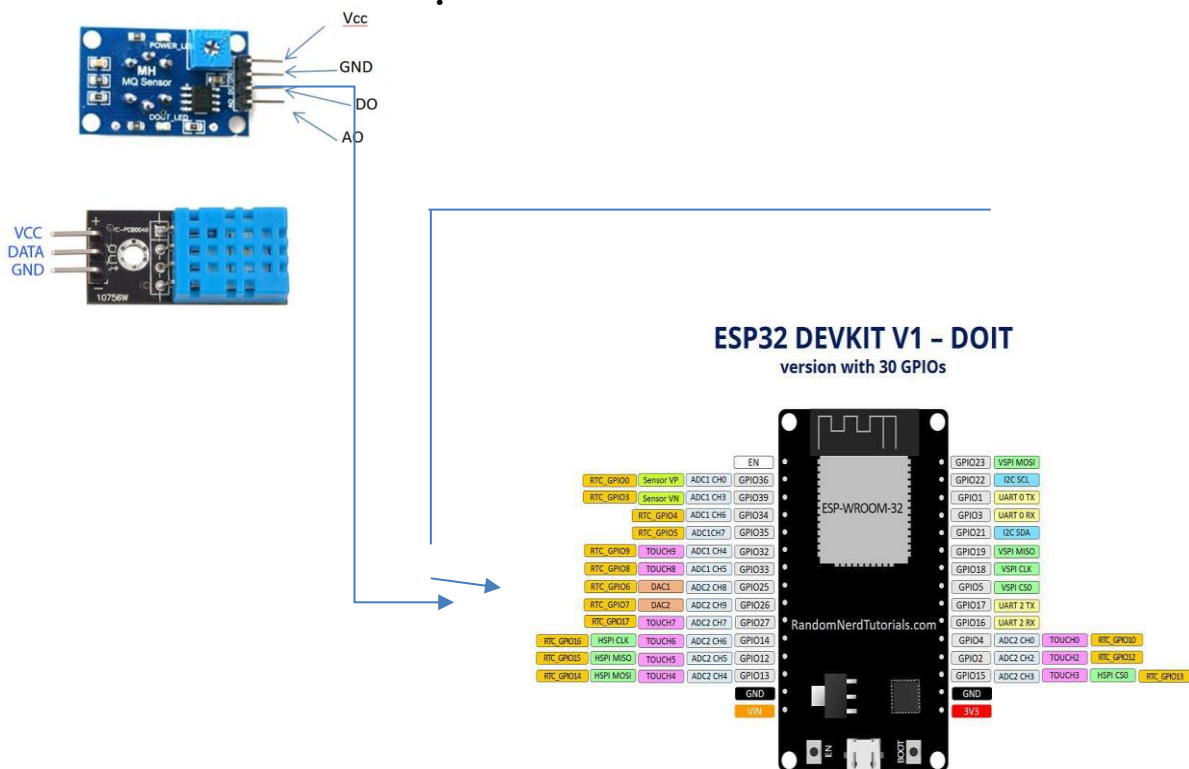
Hình 11 Sơ đồ nguyên lý mạch

### 3.2.3 Lưu đồ thuật toán



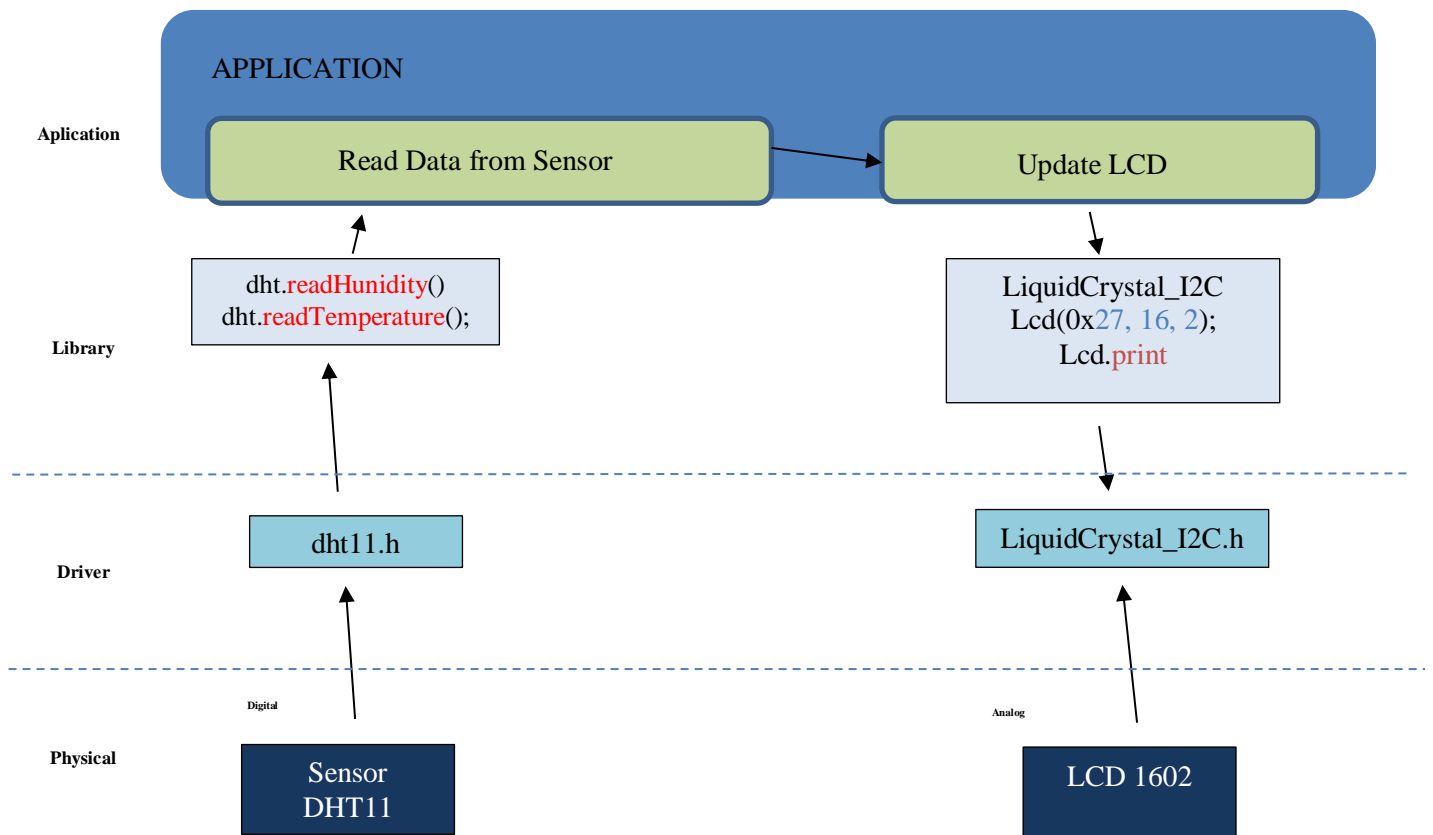
Hình 12 Lưu đồ thuật toán

### 3.2.4 Sơ đồ cắm chân linh kiện



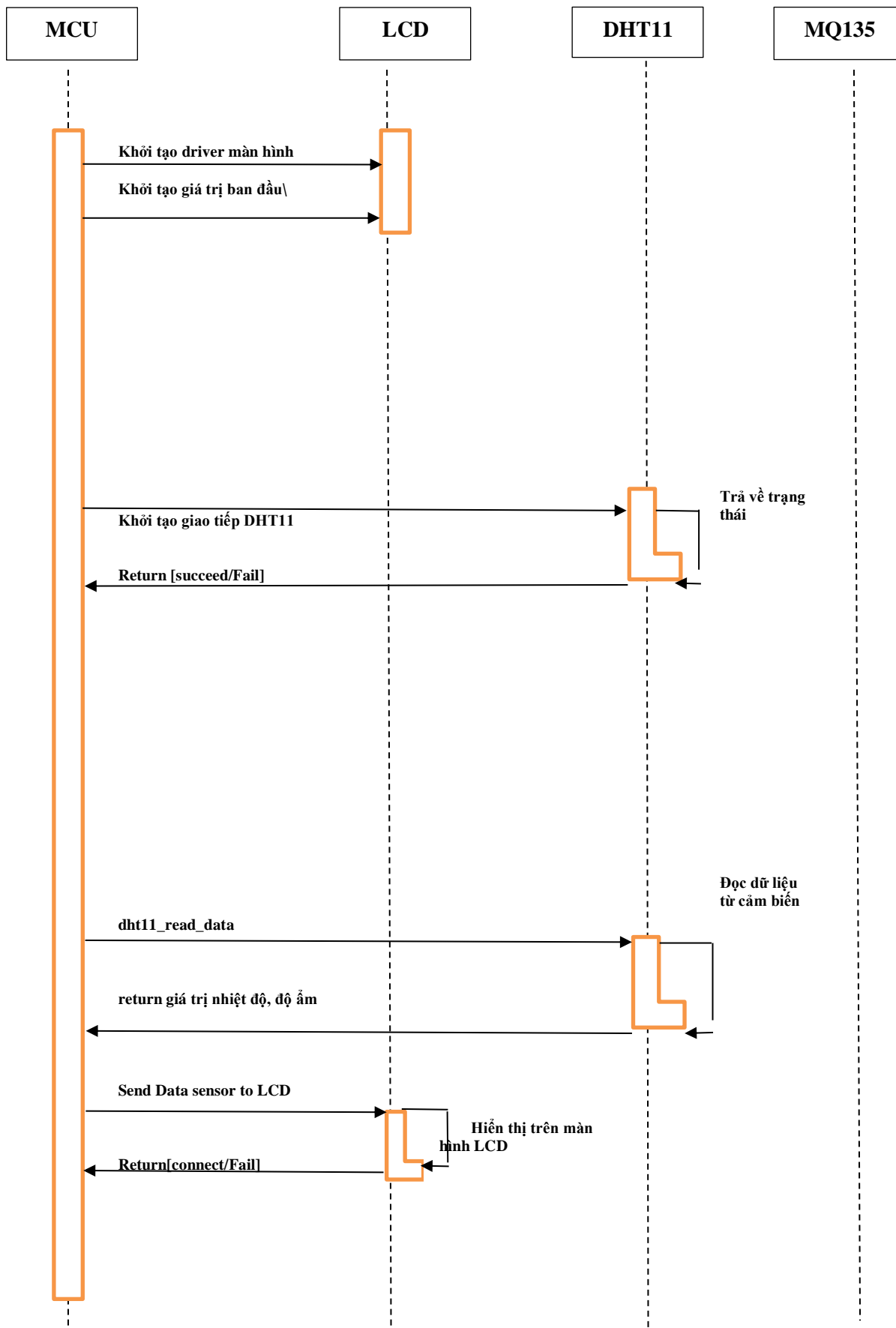
### 3.3 THIẾT KẾ PHẦN MỀM

#### 3.3.1 Sơ đồ khối phần mềm



Hình 15 Sơ đồ khối phần mềm

#### 3.3.2 Biểu đồ tuần tự



*Hình 16 Biểu đồ tuần tự*

Khi vào task `http_get_task`, task kiểm tra xem queue. Khi queue phản hồi là không rỗng, task sẽ tạo ra socket và đợi socket kết nối với sever. Sau khi kết nối xong, task sẽ gửi dữ liệu lên remote sever và remote sever phản hồi lại socket dữ liệu. Từ socket sẽ gửi lại dữ liệu lại cho một biến trong task. Sau đó là đóng socket.

### 3.3.4 Framework



*Hình 17 Framework ESP IDF*

ESP-IDF là viết tắt của "Espressif IoT Development Framework" – một bộ phát triển chính thức từ Espressif Systems dành cho các vi điều khiển dòng ESP32, ESP32-S và ESP32-C. Đây là một framework cực kỳ mạnh mẽ và linh hoạt, cung cấp các driver cần thiết, libraries, và các công cụ để phát triển các ứng dụng IoT trên chip ESP32.

ESP-IDF là một framework dựa trên FreeRTOS, một hệ điều hành thời gian thực cho phép bạn chạy nhiều tác vụ đồng thời, một tính năng quan trọng cho các ứng dụng IoT phức tạp. Nó hỗ trợ các giao tiếp không dây như WiFi và Bluetooth, cùng với nhiều giao thức bảo mật và mã hóa để đảm bảo an toàn cho các thiết bị kết nối.

## CHƯƠNG 4 : TRIỂN KHAI VÀ KIỂM THỬ

### 4.1 Kiểm thử

#### 4.1.1 Kiểm tra nguồn và các cảm biến

Cảm biến	Bài kiểm tra	Kết quả thực nghiệm
DHT11	Cấp nguồn vào cảm biến	Cảm biến hoạt động bình thường

*Bảng 6 Kiểm tra nguồn và các cảm biến*

#### 4.1.2 Kiểm tra thông mạch

Cần kiểm tra thông mạch các chân cắm của cảm biến để tránh cắm nhầm chân tín hiệu hay ngược chân nguồn gây cháy, hỏng linh kiện và tránh việc không đọc được tín hiệu từ cảm biến gửi về do đứt ngầm các dây nối.

Bài kiểm tra	Thực hiện	Kết quả	Đánh giá
Kiểm tra kết nối của các chân cắm, chân tín hiệu nối từ ESP32	Kiểm tra bằng chế độ đo thông mạch của đồng hồ vạn năng, đặt 1 que đo vào chân tín hiệu trên ESP32, que còn lại đặt vào điểm chân cắm cần kiểm tra trên mạch	Tất cả chân chân cắm cần sử dụng đều thông	Đạt yêu cầu

*Bảng 7 Kiểm tra thông mạch*

#### 4.1.3 Kiểm thử từng linh kiện

*a. DHT11*

<b>Chức năng</b>	<b>Bài kiểm tra</b>	<b>Mong muốn</b>	<b>Kết quả</b>	<b>Đánh giá</b>
Khởi tạo	Khởi tạo cảm biến		Khởi tạo cảm biến DHT11 thành công	Đạt yêu cầu
Đo nhiệt độ, độ ẩm	Thực hiện đo trong nhà <b>đóng kín cửa vào buổi sáng</b> trong 1h	Cảm biến gửi data một cách ổn định, không xuất hiện hoặc xuất hiện rất ít các giá trị nhiễu	Xuất hiện nhiễu ở giá trị đầu tiên khi mới khởi động thiết bị, sau đó thiết bị chạy ổn định	Đạt yêu cầu, chỉ xuất hiện 1 giá trị nhiễu
	Thực hiện đo trong nhà <b>mở cửa thoáng khí vào buổi sáng</b> trong 1h	Cảm biến gửi data một cách ổn định, không xuất hiện hoặc xuất hiện rất ít các giá trị nhiễu	Xuất hiện nhiễu ở giá trị đầu tiên khi mới khởi động thiết bị, sau đó thiết bị chạy ổn định	Đạt yêu cầu, chỉ xuất hiện 1 giá trị nhiễu

*Bảng 8 Kiểm thử linh kiện DHT11*

**Nhận xét:**

- Nhiệt độ, độ ẩm tương đối ổn định khi để trong môi trường đo.
- Giá trị nhiệt độ, độ ẩm đo tăng lên khi mở cửa thoáng khí.
- Giá trị nhiễu xuất hiện ở ngay lúc đầu có thể do lỗi khởi tạo hoặc có thể nhiễu từ các thiết bị cảm biến khác ảnh hưởng đến khả năng kết nối của thiết bị.

*b. LCD1602*

<b>Chức năng</b>	<b>Bài kiểm tra</b>	<b>Mong muốn</b>	<b>Kết quả</b>	<b>Đánh giá</b>
Khởi tạo	Khởi tạo riêng màn hình		Khởi tạo thành công	Đạt yêu cầu
Hiển thị thông số	Hiển thị các thông số đo được	Dữ liệu đưa lên màn hình ổn định	Dữ liệu được cập nhật lên màn hình	Đạt yêu cầu

*Bảng 9 Kiểm thử linh kiện LCD1602*

**Nhận xét:** Màn hình hiển thị thông số đo được đầy đủ, trực quan. Tuy nhiên, khi mới khởi động máy, thông số của các cảm biến vẫn xuất hiện giá trị nhiễu, gây độ trễ đối với sự hiển thị trên màn hình.



### 4.1.3 Kiểm thử toàn bộ mạch

Đối tượng kiểm tra	Bài kiểm tra	Kết quả thực nghiệm
DHT11	Đo nhiệt độ	Đo nhiệt độ môi trường phạm vi từ 0 đến 50 độ C với sai số 2 độ C
	Đo độ ẩm	Đo độ ẩm không khí 20 đến 90% với sai số 5%
Màn hình LCD	Hiển thị lên màn hình các thông số các cảm biến đo được	Màn hình hiển thị ổn định

*Bảng 10 Kiểm thử toàn bộ mạch*

## 4.2 KẾT QUẢ MẠCH THỰC TẾ

Chạy kiểm thử trong 2h, 10 phút lấy dữ liệu trên màn hình 1 lần

Lần đo	Nhiệt độ (°C)	Độ ẩm (%)
1	19.7	85%
2	20.1	85%
3	20.0	86%
4	19.8	85%
5	19.5	87%
6	19.6	85%
7	19.9	86%
8	20.2	86%
9	20.1	85%
10	19.9	85%
11	19.8	86%
12	19.6	85%
Trung bình	19.9	85.5%

*Bảng 11 Kết quả chạy mạch ở phòng không điều hòa*

Lần đo	Nhiệt độ (°C)	Độ ẩm (%)
1	12.7	85%

2	13.1	85%
3	13.0	86%
4	12.8	85%
5	12.5	87%
6	12.6	85%
7	12.9	86%
8	13.2	86%
9	13.1	85%
10	12.9	85%
11	12.8	86%
12	12.6	85%
Trung bình	12.9	85.5%

*Bảng 12 Kết quả chạy mạch ở phòng điều hòa*

Nhiệt độ:

- Cảm biến DHT11 có dải đo từ 0 – 50°C. Giá trị nhiệt độ trung bình đo được không vượt quá dải đo dù trong điều kiện mở cửa thoáng khí hay đóng cửa kín. Nhóm đánh giá dải đo đạt yêu cầu, không vượt quá thông số của cảm biến và thông tin có độ tin cậy cao.
- Giá trị đo gần chính xác với thực tế, sai số < 5%
- **Kết luận:** Đo nhiệt độ hoạt động tương đối đạt yêu cầu

Độ ẩm:

- Cảm biến DHT11 có dải đo từ 20 – 90%RH. Giá trị độ ẩm trung bình mà thiết bị đo được không vượt quá dải đo của cảm biến. Nhóm đánh giá dải đo đạt yêu cầu, không vượt quá thông số của cảm biến và thông tin có độ tin cậy cao và ổn định.
- **Kết luận:** Đo độ ẩm hoạt động tương đối đạt yêu cầu

## CHƯƠNG 5 : THIẾT KẾ FIRMWARE [12]

### 5.1 WEBPAGE

#### 5.1.1 APP.CSS

Các hàm chính , khái niệm có trong file app.css

##### 1. **Body:**

- background-color: #f1f1f1; - Thiết lập màu nền cho toàn bộ trang là màu xám nhạt.
- font-family: HelveticaNeueRegular, ... , Arial; - Định dạng font chữ mặc định cho toàn trang, ưu tiên font Helvetica.
- color: #0272B7; - Thiết lập màu chữ mặc định là màu xanh dương.

##### 2. **Headings (h1, h2, h3, h4):**

- Tất cả đều có màu chữ là #1d3557, một màu xanh đậm/navy.
- h1 có các thuộc tính về vị trí (text-align: center;), không gian (margin, padding), và kích thước font (font-size: 1.7em;).
- Các heading khác (h2, h3, h4) chỉ có sự thay đổi về kích thước font.

##### 3. **Classes (.gr, .rd):**

- .gr định nghĩa màu chữ là xanh lá.
- .rd định nghĩa màu chữ là đỏ.

##### 4. **ID selectors (e.g., #latest\_firmware, #temperature\_reading):**

- Các thành phần này được định dạng để hiển thị dạng inline, và có màu chữ là #1d3557.
- #disconnect\_wifi có thuộc tính display: none;, tức là nó sẽ không hiển thị trên trang.

##### 5. **Buttons:**

- Các nút được thiết kế với màu nền, bóng đổ, và các hiệu ứng chuyển đổi màu khi di chuột (hover) và khi nhấn (active).
- Thiết kế chi tiết với border-radius cho nút tròn và text-shadow cho bóng chữ.

Tổng quan, đoạn mã CSS này thiết kế một giao diện người dùng đơn giản nhưng hiệu quả với các điều chỉnh màu sắc, font, và tương tác nút bấm. Các thành phần được sắp xếp hợp lý và dễ dàng đọc, phù hợp với một ứng dụng web cơ bản, nhất là các ứng dụng liên quan đến IoT hoặc các hệ thống theo dõi trực tuyến mà có thể cần hiển thị nhiều thông tin động trên một màn hình.

### 5.1.2 APP.JS

Đoạn mã JavaScript liên quan đến việc quản lý kết nối WiFi, đọc giá trị cảm biến và cập nhật firmware qua OTA (Over-The-Air) trên một thiết bị. Dưới đây là các hàm chính và giải thích về chức năng của chúng trong đoạn mã:

#### Biến toàn cục

- seconds, otaTimerVar, wifiConnectInterval: Là các biến được sử dụng để quản lý thời gian trong các hàm đếm ngược và kiểm tra trạng thái.

#### Khởi tạo

- Khi tài liệu được tải (`$(document).ready()`), các hàm như `getUpdateStatus()`, `startDHTSensorInterval()`, và `getConnectInfo()` được gọi để lấy thông tin ban đầu và thiết lập các sự kiện cho nút.

#### Các hàm chính

- `getFileInfo()`: Lấy thông tin về tệp được chọn (tên và kích thước) và hiển thị trên trang web.
- `updateFirmware()`: Xử lý việc tải lên và cập nhật firmware. Nó sử dụng `XMLHttpRequest` để gửi tệp đã chọn tới máy chủ.
- `updateProgress(oEvent)`: Theo dõi tiến trình của việc tải lên tệp và gọi `getUpdateStatus()` nếu tiến trình có thể tính toán được.
- `getUpdateStatus()`: Gửi yêu cầu tới máy chủ để lấy trạng thái cập nhật OTA và hiển thị thông tin về firmware mới nhất.
- `otaRebootTimer()`: Hàm đếm ngược thời gian cho việc khởi động lại sau khi cập nhật firmware.
- `getDHTSensorValues()`: Đọc và hiển thị giá trị nhiệt độ và độ ẩm từ cảm biến DHT.
- `startDHTSensorInterval()`: Thiết lập khoảng thời gian để cập nhật liên tục giá trị cảm biến.
- `stopWifiConnectStatusInterval()`: Dừng việc kiểm tra trạng thái kết nối WiFi.
- `getWifiConnectStatus()`: Gửi yêu cầu tới máy chủ để lấy trạng thái kết nối WiFi.
- `startWifiConnectStatusInterval()`: Bắt đầu kiểm tra trạng thái kết nối WiFi định kỳ.
- `connectWifi()`: Gửi thông tin SSID và mật khẩu để kết nối WiFi.
- `checkCredentials()`: Kiểm tra thông tin đăng nhập cho WiFi.

- `showPassword()`: Hiển thị hoặc ẩn mật khẩu.
- `getConnectionInfo()`: Lấy thông tin kết nối WiFi hiện tại và hiển thị trên trang web.
- `disconnectWifi()`: Ngắt kết nối WiFi và tải lại trang.

## Tổng kết

Mã này có vẻ là một phần của giao diện điều khiển cho thiết bị IoT, cho phép người dùng quản lý kết nối WiFi, đọc các giá trị từ cảm biến và cập nhật firmware từ xa. Các hàm được tổ chức rõ ràng và phục vụ các chức năng quan trọng liên quan đến quản lý thiết bị và tương tác người dùng.

### 5.1.3 HTML

Dưới đây là phân tích của đoạn mã HTML, miêu tả một trang web được sử dụng để quản lý một thiết bị ESP32. Trang này bao gồm các tính năng như cập nhật firmware, đọc giá trị từ cảm biến DHT22, và quản lý kết nối WiFi.

#### Thành phần <head>

- **Meta Tags:** Các thẻ meta được thiết lập để đảm bảo trang tương thích với nhiều thiết bị và cấu hình trình duyệt khác nhau.
- **CSS và JavaScript:** Trang này liên kết đến một tập tin CSS (`app.css`) và một tập tin JavaScript (`app.js`), với việc tập tin JavaScript được tải bất đồng bộ (`async`).

#### Thành phần <body>

##### 1. Header

- **H1:** Tiêu đề "ESP32 Application Development" đặt trong một thẻ <header> để nhấn mạnh nó là phần đầu trang.

##### 2. OTA (Over-The-Air) Firmware Update Section

- **H2:** Tiêu đề phần cập nhật firmware là "ESP32 Firmware Update".
- **File Input:** Một trường nhập để chọn tệp firmware, mặc định được ẩn, và hiển thị thông tin tệp khi có tệp được chọn.
- **Buttons:** Có hai nút, một để mở hộp thoại chọn tệp và một để bắt đầu quá trình cập nhật firmware.

- **Thông tin và trạng thái:** Hiển thị thông tin tệp đã chọn và trạng thái cập nhật.

### 3. DHT22 Sensor Readings Section

- **H2:** Tiêu đề phần đọc giá trị cảm biến là "DHT22 Sensor Readings".
- **Temperature và Humidity Displays:** Hiển thị giá trị nhiệt độ và độ ẩm được đọc từ cảm biến DHT22.

### 4. WiFi Connection Management

- **H2:** Tiêu đề phần là "ESP32 WiFi Connect".
- **Text Inputs:** Hai trường nhập cho SSID và mật khẩu.
- **Checkbox:** Một checkbox để hiển thị hoặc ẩn mật khẩu.
- **Button và Status Display:** Nút để kết nối WiFi và một khu vực hiển thị trạng thái kết nối hoặc lỗi.

### 5. Connected WiFi Information

- **Display Sections:** Hiển thị các thông tin về kết nối WiFi hiện tại như SSID, địa chỉ IP, mặt nạ mạng, và gateway.
- **Disconnect Button:** Một nút để ngắt kết nối WiFi.

### Tổng kết

Mã HTML này cung cấp một giao diện đơn giản nhưng hiệu quả cho việc quản lý và theo dõi một thiết bị ESP32 thông qua trình duyệt web. Nó sử dụng jQuery cho logic phía client và có các tính năng như cập nhật firmware, đọc giá trị cảm biến, và quản lý kết nối WiFi, phù hợp cho một khóa học phát triển ứng dụng IoT.

## 5.2 DHT22.C

Đoạn mã là một driver cho cảm biến nhiệt độ và độ ẩm DHT22, được viết cho vi điều khiển ESP32 sử dụng framework ESP-IDF của Espressif. Dưới đây là phân tích chi tiết các phần chính của đoạn mã:

### Định nghĩa và Khai báo

- **Mã nguồn** dựa trên các đóng góp từ Adafruit Industries và những người khác.

- **Các biến toàn cục** như humidity, temperature, và DHTgpio được sử dụng để lưu trữ các giá trị đo được và cổng GPIO kết nối với cảm biến.

### Chức năng Cài đặt và Đọc Cảm biến

- **setDHTgpio(int gpio):** Cài đặt GPIO được sử dụng cho cảm biến.
- **getHumidity() và getTemperature():** Trả về các giá trị nhiệt độ và độ ẩm hiện tại.
- **errorHandler(int response):** Xử lý lỗi dựa trên mã phản hồi từ cảm biến, báo lỗi như timeout hoặc lỗi checksum.

### Hàm Đọc Dữ liệu từ Cảm biến DHT22

- **readDHT():** Chức năng chính để khởi động và đọc dữ liệu từ cảm biến. Mã này gửi tín hiệu khởi đầu, đợi phản hồi từ cảm biến, và sau đó đọc 40 bit dữ liệu trả về từ cảm biến.
- Mã kiểm tra các mức tín hiệu trên GPIO và thời gian đợi để xác định giá trị bit là '0' hay '1'.
- Tính toán giá trị độ ẩm và nhiệt độ từ dữ liệu thô, và kiểm tra checksum để xác minh tính hợp lệ của dữ liệu đọc được.

### Chức năng của Task và Khởi động Task

- **DHT22\_task(void \*pvParameter):** Một task RTOS để đọc dữ liệu từ cảm biến DHT22 liên tục với khoảng thời gian lặp lại nhất định.
- **DHT22\_task\_start():** Khởi tạo và bắt đầu task cho DHT22.

### Nhận xét

- Driver sử dụng thư viện freertos để lập trình đa nhiệm, điều rất quan trọng trong môi trường nhúng để đọc dữ liệu từ các cảm biến mà không làm ảnh hưởng tới hoạt động chính của chương trình.
- Mã có chứa một số ghi chú từ tác giả về cách cải tiến mã và những điểm không hài lòng trong logic hiện tại, điều này giúp cho việc duy trì và nâng cấp mã trong tương lai.
- Mã này đề cao việc xử lý lỗi và báo cáo tình trạng, điều cần thiết để đảm bảo độ tin cậy trong ứng dụng thực tế.

Tổng kết, đoạn mã này cung cấp một giải pháp đầy đủ và có khả năng mở rộng cho việc tích hợp cảm biến DHT22 vào một dự án ESP32, sử dụng các tiêu chuẩn của lập trình nhúng hiện đại.

```

137 ;-----*/
138
139 #define MAXdhtData 5    // to complete 40 = 5*8 Bits
140
141 int readDHT()
142 {
143     int uSec = 0;
144
145     uint8_t dhtData[MAXdhtData];
146     uint8_t byteInx = 0;
147     uint8_t bitInx = 7;
148
149     for (int k = 0; k<MAXdhtData; k++)
150         dhtData[k] = 0;
151
152     // == Send start signal to DHT sensor =====
153
154     gpio_set_direction( DHTgpio, GPIO_MODE_OUTPUT );
155
156     // pull down for 3 ms for a smooth and nice wake up
157     gpio_set_level( DHTgpio, 0 );
158     esp_rom_delay_us( 3000 );
159
160     // pull up for 25 us for a gentile asking for data
161     gpio_set_level( DHTgpio, 1 );
162     esp_rom_delay_us( 25 );
163
164     gpio_set_direction( DHTgpio, GPIO_MODE_INPUT );    // change to input mode
165
166     // == DHT will keep the line low for 80 us and then high for 80us ====
167
168     uSec = getSignalLevel( 85, 0 );
169     // ESP_LOGI( TAG, "Response = %d", uSec );
170     if( uSec<0 ) return DHT_TIMEOUT_ERROR;
171
172     // -- 80us up -----
173
174     uSec = getSignalLevel( 85, 1 );
175     // ESP_LOGI( TAG, "Response = %d", uSec );
176     if( uSec<0 ) return DHT_TIMEOUT_ERROR;
177
178     // == No errors, read the 40 data bits =====

```

### 5.3 HTTP\_sever

Đoạn mã là một tập header cho một máy chủ HTTP trong một ứng dụng ESP32, với các chức năng liên quan đến quản lý cập nhật firmware OTA (Over-The-Air) và kết nối WiFi. Dưới đây là mô tả chi tiết về các thành phần chính của tập này:

#### Định nghĩa và Tùy Chọn

- **OTA\_UPDATE\_PENDING, OTA\_UPDATE\_SUCCESSFUL, OTA\_UPDATE\_FAILED:** Định nghĩa các trạng thái cho quá trình cập nhật firmware qua OTA, giúp theo dõi tình trạng cập nhật.

#### Enumerations

- **http\_server\_wifi\_connect\_status\_e:** Liệt kê các trạng thái khác nhau của kết nối



WiFi, bao gồm đang kết nối, kết nối thất bại, và kết nối thành công.

- **http\_server\_message\_e**: Định nghĩa các loại thông điệp có thể gửi trong hệ thống, bao gồm khởi tạo kết nối WiFi, kết nối WiFi thành công/thất bại, và các trạng thái cập nhật OTA.

### Cấu Trúc Dữ Liệu

- **http\_server\_queue\_message\_t**: Cấu trúc dữ liệu này dùng để đưa thông điệp vào hàng đợi. Nó chứa một ID thông điệp từ enum `http_server_message_e`.

### Chức Năng

- **http\_server\_monitor\_send\_message()**: Hàm này gửi thông điệp vào hàng đợi của máy chủ HTTP. Trả về `pdTRUE` nếu thông điệp được gửi thành công và `pdFALSE` nếu không thành công. Hàm này có thể được mở rộng để bao gồm thêm các tham số dựa trên yêu cầu của hệ thống.
- **http\_server\_start()** và **http\_server\_stop()**: Các hàm này khởi động và dừng máy chủ HTTP. Chúng có thể liên quan đến cấu hình máy chủ, bắt đầu lắng nghe các kết nối đến, và dọn dẹp tài nguyên khi máy chủ dừng lại.
- **http\_server\_fw\_update\_reset\_callback()**: Hàm callback này được gọi để khởi động lại ESP32 sau khi cập nhật firmware thành công. Hàm này nhận một tham số là `arg`, có thể được sử dụng để truyền dữ liệu cụ thể liên quan đến cập nhật.

### Tổng Kết

Tập header này cung cấp một khung sườn cơ bản cho quản lý các hoạt động liên quan đến mạng và OTA trong một ứng dụng ESP32. Nó cấu hình các trạng thái, thông điệp và cung cấp các API để điều khiển máy chủ HTTP, gửi thông điệp trong hệ thống, và xử lý các hành động sau cập nhật firmware. Tập này cũng hỗ trợ tích hợp các chức năng với hệ điều hành thời gian thực (RTOS), một yếu tố quan trọng trong phát triển nhúng để đảm bảo phản hồi nhanh và đáng tin cậy từ thiết bị.

```

16  // Connection status for WiFi
17  */
18  typedef enum http_server_wifi_connect_status
19  {
20      NONE = 0,
21      HTTP_WIFI_STATUS_CONNECTING,
22      HTTP_WIFI_STATUS_CONNECT_FAILED,
23      HTTP_WIFI_STATUS_CONNECT_SUCCESS,
24  } http_server_wifi_connect_status_e;
25
26  /**
27   * Messages for the HTTP monitor
28   */
29  typedef enum http_server_message
30  {
31      HTTP_MSG_WIFI_CONNECT_INIT = 0,
32      HTTP_MSG_WIFI_CONNECT_SUCCESS,
33      HTTP_MSG_WIFI_CONNECT_FAIL,
34      HTTP_MSG_OTA_UPDATE_SUCCESSFUL,
35      HTTP_MSG_OTA_UPDATE_FAILED,
36  } http_server_message_e;
37
38  /**
39   * Structure for the message queue
40   */
41  typedef struct http_server_queue_message
42  {
43      http_server_message_e msgID;
44  } http_server_queue_message_t;
45
46  /**
47   * Sends a message to the queue
48   * @param msgID message ID from the http_server_message_e enum.
49   * @return pdTRUE if an item was successfully sent to the queue, otherwise
50   * @note Expand the parameter list based on your requirements e.g. how you'
51   */
52  BaseType_t http_server_monitor_send_message(http_server_message_e msgID);
53
54  /**
55   * Starts the HTTP server.
56   */
57  void http_server_start(void);
58

```

## 5.4 Wifi\_app

Đoạn mã là tệp header cho một ứng dụng WiFi được thiết kế cho ESP32, sử dụng trong môi trường lập trình FreeRTOS và ESP-IDF. Tệp này định nghĩa các cấu hình, loại tin nhắn, và các hàm quản lý chức năng WiFi của ứng dụng. Dưới đây là phân tích chi tiết các thành phần chính của tệp header này:

### Định Nghĩa và Cấu Hình

- **Định nghĩa WiFi AP (Access Point):** Cung cấp các thông tin cấu hình cho điểm truy cập WiFi như SSID, mật khẩu, kênh, độ ẩn SSID, số lượng kết nối tối đa, và khoảng thời gian giữa các tín hiệu beacon.
- **Cấu hình IP cho AP:** Địa chỉ IP mặc định, gateway, và subnet mask cho điểm truy cập.
- **Cài đặt băng thông và chế độ tiết kiệm năng lượng:** Thiết lập băng thông và không sử dụng chế độ tiết kiệm năng lượng cho STA (Station Mode).

### Enum và Cấu Trúc

- **wifi\_app\_message\_e:** Enumeration định nghĩa các ID tin nhắn mà có thể được gửi trong nhiệm vụ WiFi, như khởi động máy chủ HTTP, kết nối từ máy chủ HTTP, nhận được IP, yêu cầu ngắt kết nối từ người dùng, và các sự kiện khác liên quan đến quản lý kết nối WiFi.
- **wifi\_app\_queue\_message\_t:** Cấu trúc dữ liệu cho hàng đợi tin nhắn, chứa ID tin nhắn.

### Các Hàm Chính

- **wifi\_app\_send\_message:** Hàm gửi tin nhắn vào hàng đợi, sử dụng để giao tiếp giữa các nhiệm vụ hoặc sự kiện trong hệ thống.
- **wifi\_app\_start:** Khởi động nhiệm vụ RTOS cho ứng dụng WiFi.
- **wifi\_app\_get\_wifi\_config:** Trả về cấu hình WiFi hiện tại được sử dụng bởi module.
- **wifi\_app\_set\_callback và wifi\_app\_call\_callback:** Đăng ký và gọi một hàm callback khi có sự kiện kết nối WiFi thành công.
- **wifi\_app\_get\_rssi:** Lấy giá trị RSSI của kết nối WiFi, có thể được sử dụng để đánh giá chất lượng liên kết không dây.

### Ý Nghĩa Trong Ứng Dụng

Tệp header này cung cấp các API cần thiết để quản lý kết nối WiFi trong một ứng dụng nhúng dựa trên ESP32, cho phép phát triển các ứng dụng có khả năng kết nối mạng phức tạp như điều khiển thiết bị IoT qua mạng, hoặc bất kỳ ứng dụng nào yêu cầu truyền thông không dây. Các cấu hình và hàm được định nghĩa sẵn trong tệp này giúp cho việc triển khai và bảo trì mã nguồn trở nên dễ dàng và có tổ chức hơn.

```

7
8 #ifndef MAIN_WIFI_APP_H_
9 #define MAIN_WIFI_APP_H_
10
11 #include "esp_netif.h"
12 #include "esp_wifi_types.h"
13 #include "freertos/FreeRTOS.h"
14
15 // Callback typedef
16 typedef void (*wifi_connected_event_callback_t)(void);
17
18 // WiFi application settings
19 #define WIFI_AP_SSID "ESP32_AP" // AP
20 #define WIFI_AP_PASSWORD "password" // AP
21 #define WIFI_AP_CHANNEL 1 // AP
22 #define WIFI_AP_SSID_HIDDEN 0 // AP
23 #define WIFI_AP_MAX_CONNECTIONS 5 // AP
24 #define WIFI_AP_BEACON_INTERVAL 100 // AP
25 #define WIFI_AP_IP "192.168.0.1" // AP
26 #define WIFI_AP_GATEWAY "192.168.0.1" // AP
27 #define WIFI_AP_NETMASK "255.255.255.0" // AP
28 #define WIFI_AP_BANDWIDTH WIFI_BW_HT20 // AP
29 #define WIFI_STA_POWER_SAVE WIFI_PS_NONE // Pow
30 #define MAX_SSID_LENGTH 32 // IEE
31 #define MAX_PASSWORD_LENGTH 64 // IEE
32 #define MAX_CONNECTION_RETRIES 5 // Ret
33
34 // netif object for the Station and Access Point
35 extern esp_netif_t* esp_netif_sta;
36 extern esp_netif_t* esp_netif_ap;
37
38 /**
39  * Message IDs for the WiFi application task
40  * @note Expand this based on your application requirements.
41  */
42 typedef enum wifi_app_message
43 {
44     WIFI_APP_MSG_START_HTTP_SERVER = 0,
45     WIFI_APP_MSG_CONNECTING_FROM_HTTP_SERVER,
46     WIFI_APP_MSG_STA_CONNECTED_GOT_IP,
47     WIFI_APP_MSG_USER_REQUESTED_STA_DISCONNECT,
48     WIFI_APP_MSG_LOAD_SAVED_CREDENTIALS,
49     WIFI_APP_MSG_STA_DISCONNECTED,
50 } wifi_app_message_e;
51

```

## 5.5 task\_common

Tập header `tasks_common.h` chứa các định nghĩa cho cấu hình các nhiệm vụ (tasks) trong hệ thống RTOS, được sử dụng trong một ứng dụng ESP32. Tập này thiết lập kích thước ngăn xếp, độ ưu tiên, và ID lõi cho mỗi nhiệm vụ. Dưới đây là chi tiết các định nghĩa và vai trò của chúng:

### Định Nghĩa Các Nhiệm Vụ

- **WIFI\_APP\_TASK:** Định nghĩa cấu hình cho nhiệm vụ ứng dụng WiFi, bao gồm:
  - `WIFI_APP_TASK_STACK_SIZE`: Kích thước ngăn xếp là 4096 bytes.
  - `WIFI_APP_TASK_PRIORITY`: Độ ưu tiên của nhiệm vụ là 5.
  - `WIFI_APP_TASK_CORE_ID`: Nhiệm vụ này được gán chạy trên lõi 0 của ESP32.
- **HTTP\_SERVER\_TASK:** Định nghĩa cấu hình cho nhiệm vụ máy chủ HTTP, bao gồm:
  - `HTTP_SERVER_TASK_STACK_SIZE`: Kích thước ngăn xếp lớn hơn là 8192 bytes, phản ánh yêu cầu tài nguyên cao hơn cho xử lý máy chủ web.
  - `HTTP_SERVER_TASK_PRIORITY`: Độ ưu tiên của nhiệm vụ là 4.
  - `HTTP_SERVER_TASK_CORE_ID`: Nhiệm vụ này cũng được gán chạy trên lõi 0.
- **HTTP\_SERVER\_MONITOR\_TASK:** Định nghĩa cấu hình cho nhiệm vụ giám sát máy chủ HTTP, bao gồm:
  - `HTTP_SERVER_MONITOR_STACK_SIZE`: Kích thước ngăn xếp là 4096 bytes.
  - `HTTP_SERVER_MONITOR_PRIORITY`: Độ ưu tiên của nhiệm vụ là 3, thấp hơn các nhiệm vụ trên.
  - `HTTP_SERVER_MONITOR_CORE_ID`: Nhiệm vụ này cũng chạy trên lõi 0.
- **DHT22\_TASK:** Định nghĩa cấu hình cho nhiệm vụ đọc cảm biến DHT22, bao gồm:
  - `DHT22_TASK_STACK_SIZE`: Kích thước ngăn xếp là 4096 bytes.
  - `DHT22_TASK_PRIORITY`: Độ ưu tiên của nhiệm vụ là 5, tương đương với nhiệm vụ ứng dụng WiFi.
  - `DHT22_TASK_CORE_ID`: Nhiệm vụ này được gán chạy trên lõi 1, cho phép

nó hoạt động đồng thời với các nhiệm vụ khác mà không bị ảnh hưởng bởi việc chia sẻ lõi.

## Ý Nghĩa và Ứng Dụng

Việc phân bổ tài nguyên như kích thước ngăn xếp, độ ưu tiên, và lõi CPU cho mỗi nhiệm vụ giúp tối ưu hóa hiệu suất và đáp ứng trong môi trường thời gian thực của FreeRTOS trên ESP32. Các cấu hình này đặc biệt quan trọng trong việc đảm bảo rằng các nhiệm vụ quan trọng như xử lý WiFi và phục vụ trang web không bị cản trở bởi các tác vụ ít ưu tiên hơn hoặc không quan trọng. Việc chạy các nhiệm vụ trên lõi riêng biệt giúp giảm thiểu rủi ro về sự cạnh tranh tài nguyên và tối ưu hoá đáp ứng của hệ thống.

```
7
8 #ifndef MAIN_TASKS_COMMON_H_
9 #define MAIN_TASKS_COMMON_H_
10
11 // WiFi application task
12 #define WIFI_APP_TASK_STACK_SIZE      4096
13 #define WIFI_APP_TASK_PRIORITY        5
14 #define WIFI_APP_TASK_CORE_ID        0
15
16 // HTTP Server task
17 #define HTTP_SERVER_TASK_STACK_SIZE   8192
18 #define HTTP_SERVER_TASK_PRIORITY     4
19 #define HTTP_SERVER_TASK_CORE_ID     0
20
21 // HTTP Server Monitor task
22 #define HTTP_SERVER_MONITOR_STACK_SIZE 4096
23 #define HTTP_SERVER_MONITOR_PRIORITY  3
24 #define HTTP_SERVER_MONITOR_CORE_ID  0
25
26 #define DHT22_TASK_STACK_SIZE         4096
27 #define DHT22_TASK_PRIORITY           5
28 #define DHT22_TASK_CORE_ID           1
29 #endif /* MAIN_TASKS_COMMON_H_ */
30
```

## 5.6 Main

Đoạn mã là hàm `app_main()`, điểm khởi đầu của một ứng dụng trên ESP32 khi sử dụng framework ESP-IDF. Hàm này khởi tạo bộ nhớ không biến mất (NVS), khởi động mô-đun WiFi, và bắt đầu nhiệm vụ cho cảm biến DHT22. Dưới đây là giải thích chi tiết từng bước

thực hiện trong hàm:

### Khởi tạo NVS (Non-Volatile Storage)

- **NVS:** Là một hệ thống lưu trữ key-value trên bộ nhớ flash của ESP32, cho phép lưu trữ cài đặt cấu hình và dữ liệu giữa các lần khởi động lại.
- **nvs\_flash\_init():** Khởi tạo bộ nhớ NVS. Nếu gặp lỗi do không đủ trang trống hoặc tìm thấy phiên bản NVS mới, hệ thống sẽ xóa bộ nhớ NVS hiện tại và thử khởi tạo lại. Điều này đảm bảo rằng bộ nhớ NVS sẽ sẵn sàng để lưu trữ dữ liệu cần thiết cho ứng dụng.

### Khởi động Mô-đun WiFi

- **wifi\_app\_start():** Hàm này được gọi để khởi tạo và bắt đầu quản lý kết nối WiFi của thiết bị. Điều này bao gồm việc thiết lập cấu hình, đăng ký xử lý sự kiện, và kết nối với điểm truy cập hoặc thiết lập điểm truy cập.

### Khởi động Nhiệm vụ cho Cảm biến DHT22

- **DHT22\_task\_start():** Bắt đầu nhiệm vụ liên tục để đọc dữ liệu từ cảm biến DHT22, thường là nhiệt độ và độ ẩm. Nhiệm vụ này sẽ lấy dữ liệu từ cảm biến và có thể sử dụng các thông tin đó cho các mục đích như giám sát môi trường hoặc điều khiển tự động.

### Quá Trình Khởi Động Ứng Dụng

Hàm `app_main()` thể hiện cách ứng dụng ESP32 khởi động và thiết lập các thành phần cốt lõi của nó. Quá trình khởi tạo NVS trước tiên đảm bảo rằng cấu hình và dữ liệu cần thiết có sẵn cho toàn bộ quá trình vận hành của thiết bị. Sau đó, hệ thống WiFi được khởi tạo để quản lý kết nối mạng, và cuối cùng là bắt đầu nhiệm vụ cho cảm biến DHT22 để thu thập dữ liệu môi trường liên tục.

Quá trình này thể hiện một cấu trúc mạch lạc cho việc phát triển ứng dụng nhúng trên nền tảng ESP32, tận dụng các API của ESP-IDF để quản lý các tài nguyên phần cứng và nhiệm vụ một cách hiệu quả.



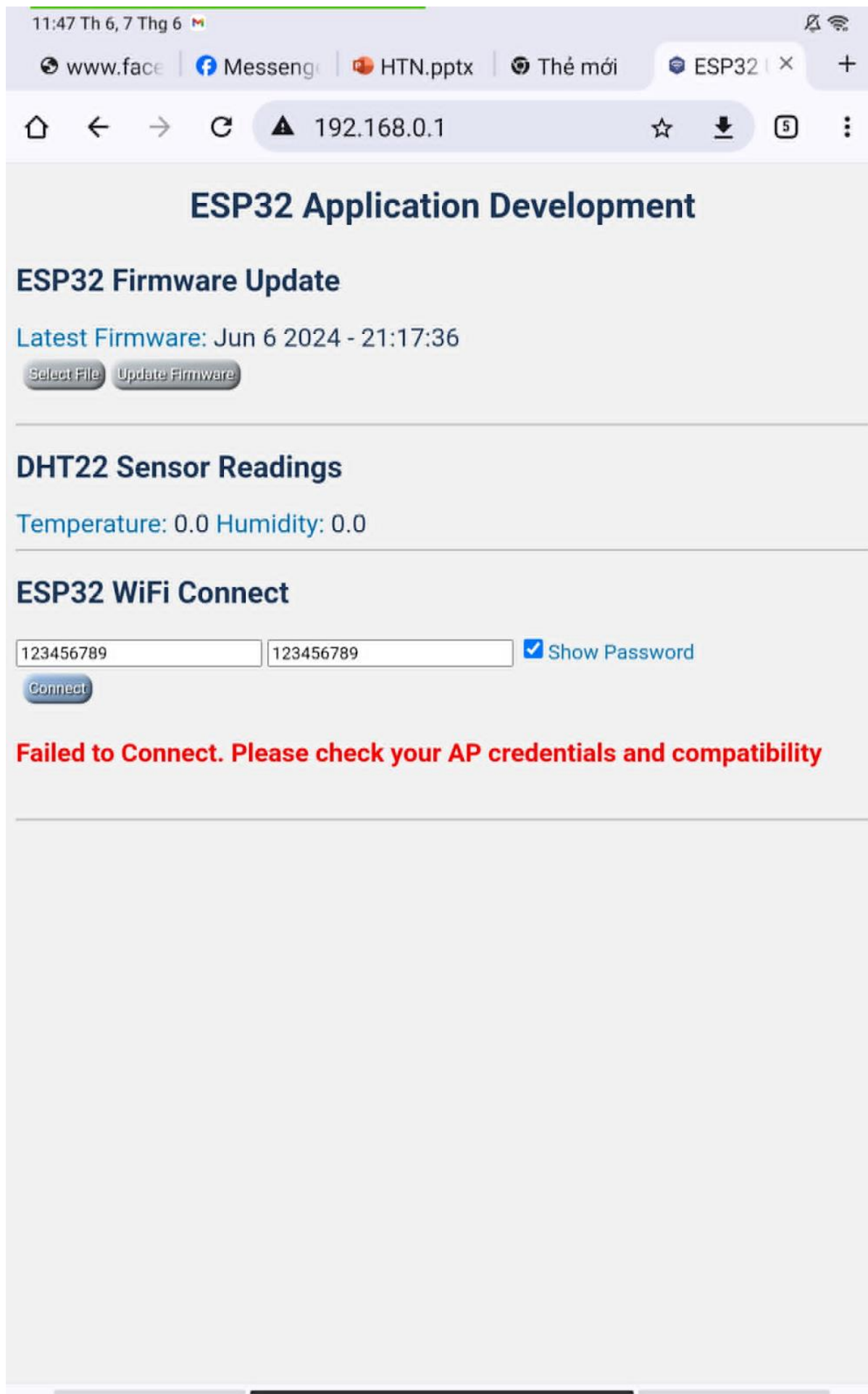
```

1  /**
2   * Application entry point.
3   */
4
5  #include "nvs_flash.h"
6
7  #include "DHT22.h"
8  #include "wifi_app.h"
9
10 void app_main(void)
11 {
12     // Initialize NVS
13     esp_err_t ret = nvs_flash_init();
14     if (ret == ESP_ERR_NVS_NO_FREE_PAGES || ret == ESP_ERR_NVS_NEW_VERSION_FOUND)
15     {
16         ESP_ERROR_CHECK(nvs_flash_erase());
17         ret = nvs_flash_init();
18     }
19     ESP_ERROR_CHECK(ret);
20
21     // Start Wifi
22     wifi_app_start();
23
24     // Start DHT22 Sensor task
25     DHT22_task_start();
26 }
27
28

```

## 5.7 Kết quả

Kết quả đã build thành công , giao diện có các chức năng chính như cập nhật firmware qua OTA , biế esp32 vào chế độ AP point , kết nối wifi với esp32 , hiển thị dữ liệu từ esp32 truyền lên



## KẾT LUẬN

Trong tình hình ô nhiễm không khí ngày càng trở nên nghiêm trọng, việc đo chất lượng không khí đã trở thành một việc quan trọng để đảm bảo sức khỏe con người và bảo vệ môi trường. Đề tài "Thiết kế mạch đo chất lượng không khí" tập trung vào nghiên cứu và thực hiện một hệ thống đo lường tiện ích, đáng tin cậy và dễ dàng sử dụng để đo chất lượng không khí.

Trong quá trình thực hiện đề tài, chúng em đã tiến hành nghiên cứu một loạt yêu cầu kỹ thuật cụ thể cho hệ thống, bao gồm yêu cầu chức năng và phi chức năng. Các yêu cầu này đã định hình hướng đi cho việc thiết kế phần cứng và phần mềm của hệ thống. Sau đó sử dụng nền tảng Blynk để xây dựng giao diện người dùng thân thiện và dễ dàng quản lý dữ liệu đo, đồng thời sử dụng nền tảng ESP32 nhằm mang lại lợi ích lớn về độ chính xác, tích hợp dữ liệu và khả năng tương thích, làm cho sản phẩm trở nên hữu ích và hiệu quả.

Trong phần thí nghiệm và kết quả, chúng em đã tiến hành kiểm thử các khả năng hoạt động của hệ thống, từ kiểm tra nguồn và cảm biến cho đến kiểm tra chức năng toàn bộ hệ thống. Kết quả thực tế đã cho thấy rằng hệ thống hoạt động đáng tin cậy và đáp ứng được các yêu cầu kỹ thuật đã đề ra.

Nhìn chung, qua quá trình nghiên cứu và thực hiện, đề tài đã đạt được những mục tiêu đề ra. Trong tương lai, nhóm sẽ tiếp tục phát triển đề tài với định hướng tối ưu hiệu suất cảm biến và tích hợp trí tuệ nhân tạo nhằm hoàn thiện sản phẩm và giúp hỗ trợ cải thiện chất lượng không khí, ảnh hưởng tích cực đến sức khỏe và môi trường, đóng góp vào cuộc sống bền vững.



## TÀI LIỆU THAM KHẢO

- [1] WHO, “[Ô nhiễm không khí ở Việt Nam](#)”, 2018
- [2] PGS.TS. Nguyễn Thị Nhật Thanh, “Kết quả nghiên cứu tác động ô nhiễm không khí do bụi mịn PM2.5 lên sức khỏe cộng đồng tại Hà Nội năm 2019”, 2021
- [3] WHO, “WHO global air quality guidelines: particulate matter (PM2.5 and PM10), ozone, nitrogen dioxide, sulfur dioxide and carbon monoxide”, 2021
- [4] Thanh Phương, “[Ô nhiễm không khí ở Hà Nội: Giải pháp không phải đơn giản](#)”, 2020
- [5] Châu Vũ, “[Cách cải thiện chất lượng không khí trong nhà](#)”, 2022
- [6] Expressif, “[ESP32 Series Datasheet](#)”, 2023
- [7] Mouser Electronics, “[DHT11 Humidity & Temperature Sensor](#)”
- [8] HANWEI ELECTRONICS, “[TECHNICAL DATA MQ-135 GAS SENSOR](#)”
- [9] Zhou Yong, “[PMS7003 series data manual](#)”, 2016
- [10] Riverdi, “[LCD TFT datasheet](#)”, 2016
- [11] Maxim Integrated, “[Extremely Accurate I2C-Integrated RTC/TCXO/Crystal](#)”, 2015
- [12] Khóa học esp32 trên udemy

## **PHỤ LỤC**

### **A. Mã nguồn hệ thống**

**[https://drive.google.com/drive/folders/1dIyPaZwfleHi\\_NpvY2wea-o2mKnDVJfI?usp=sharing](https://drive.google.com/drive/folders/1dIyPaZwfleHi_NpvY2wea-o2mKnDVJfI?usp=sharing)**