Shape, square

Description automatically generated**ĐẠI HỌC BÁCH KHOA HÀ NỘI**

VIỆN ĐIỆN TỬ VIỄN THÔNG

-----□□&□□-----

**Icon

Description automatically generated with low confidence**

**BÁO CÁO BÀI TẬP LỚN**

**KỸ THUẬT VI XỬ LÝ**

**Đề tài: Hệ thống đo lường chất lượng không khí**

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

**Nhóm sinh viên thực hiện: Nhóm 5**

|  |  |
| --- | --- |
| **Họ và tên** | **MSSV** |
| Nguyễn Văn Long | 20182665 |
| Nguyễn Minh Đan | 20183876 |
| Ngô Đức Việt | 20183858 |
| Hoàng Anh Quân | 20180159 |
| Diêm Mạnh Hiếu | 20183529 |

**Lớp: CTTN Điện tử truyền thông K63**

Hà Nội, 07-2021

**LỜI NÓI ĐẦU**

Trong những năm trở lại đây, biến đổi khí hậu đã và đang là một vấn đề nhức nhối không chỉ ở Việt Nam mà còn trên toàn thế giới. Đặc biệt tại nước ta, với vị thế là một quốc gia đang phát triển, chúng ta không chỉ phải chú trọng kế hoạch phát triển về kinh tế mà còn phải quan tâm đến vấn đề môi trường. Đó chính là nền tảng của một nền kinh tế bền vững với sức khỏe của mọi người dân được đảm bảo.

Hiện nay, các chỉ số về chất lượng không khí tại Hà Nội thường xuyên ở mức báo động và xếp hạng ở mức tệ nhất trong khu vực. Từ thực trạng đó, những sản phẩm công nghệ ứng dụng đo lường và cảnh báo về chất lượng không khí là rất cần thiết, với mục đích giúp người sử dụng nắm bắt từ đó có những điều chỉnh trong sinh hoạt để cải thiện sức khỏe bản thân.

Trong quá trình học học phần “Kỹ thuật vi xử lý”, nhóm chúng em đã nhận thấy rằng có thể áp dụng được những kiến thức đã học vào một sản phẩm thực tiễn với đề tài “Đo lường chất lượng không khí”. Đây là cơ hội tốt để chúng em vận dụng vào thực tế và củng cố thêm những lý thuyết đã học trên lớp, ngoài ra còn có thể tìm hiểu nhiều kiến thức vận dụng liên quan khác trong quá trình làm việc.

Nhóm chúng em xin được gửi lời cảm ơn chân thành đến TS Hàn Huy Dũng - Viện Điện tử - Viễn thông đã nhiệt tình hướng dẫn và hỗ trợ nhóm trong suốt quá trình tìm hiểu và hoàn thành bài tập lớn này!

**MỤC LỤC**

[CHƯƠNG 1. GIỚI THIỆU 7](#_Toc77292055)

[1.1 Đặt vấn đề 7](#_Toc77292056)

[1.2 Giới thiệu đề tài 7](#_Toc77292057)

[1.3 Phân công công việc 8](#_Toc77292058)

[CHƯƠNG 2. YÊU CẦU HỆ THỐNG 10](#_Toc77292059)

[2.1 Yêu cầu chức năng 10](#_Toc77292060)

[2.2 Yêu cầu phi chức năng 10](#_Toc77292061)

[2.3 Sơ đồ hệ thống 10](#_Toc77292062)

[2.4 Danh sách linh kiện cần sử dụng 11](#_Toc77292063)

[2.4.1 Giới thiệu về ESP32 11](#_Toc77292064)

[2.4.2 Giới thiệu về màn hình LCD 1602 với module I2C 14](#_Toc77292065)

[2.4.3 Giới thiệu về cảm biến nhiệt độ, độ ẩm DHT22 17](#_Toc77292066)

[2.4.4 Giới thiệu về cảm biến khí CO MQ7 20](#_Toc77292067)

[2.4.5 Giới thiệu về cảm biến bụi PM2.5 GP2Y1010AU0F 24](#_Toc77292068)

[CHƯƠNG 3. THIẾT KẾ HỆ THỐNG 28](#_Toc77292069)

[3.1 Thiết kế phần cứng 28](#_Toc77292070)

[3.1.1 Phân tích công suất 28](#_Toc77292071)

[3.1.2 Thiết kế mạch 30](#_Toc77292072)

[3.2 Thiết kế phần mềm 33](#_Toc77292073)

[3.2.1 Lưu đồ thuật toán 33](#_Toc77292074)

[3.2.2 Công nghệ sử dụng 33](#_Toc77292075)

[3.2.3 Kết nối vi xử lý ESP32 với các module 44](#_Toc77292076)

[CHƯƠNG 4. KẾT QUẢ THỰC HIỆN 47](#_Toc77292077)

[4.1 Kết quả mạch thực tế 47](#_Toc77292078)

[4.2 Kiểm thử 47](#_Toc77292079)

[4.3 Kết quả thực hiện hệ thống thực tế 48](#_Toc77292080)

[KẾT LUẬN 53](#_Toc77292081)

[TÀI LIỆU THAM KHẢO 54](#_Toc77292082)

[PHỤ LỤC 56](#_Toc77292083)

[A. Mã nguồn hệ thống 56](#_Toc77292084)

[B. Video demo sản phẩm 72](#_Toc77292085)

**DANH SÁCH HÌNH ẢNH**

[Hình 2.1 Sơ đồ hệ thống quan trắc môi trường sử dụng ESP32 11](#_Toc77929713)

[Hình 2.2 Sơ đồ chân ESP32 DEVKIT V1 DOIT 14](#_Toc77929714)

[Hình 2.3 Đường bus kết nối của chuẩn I2C 15](#_Toc77929715)

[Hình 2.4 Màn hình LCD 1602 15](#_Toc77929716)

[Hình 2.5 Module I2C 16](#_Toc77929717)

[Hình 2.6 Sơ đồ đấu nối ESP32 với LCD I2C 17](#_Toc77929718)

[Hình 2.7 Các chân của DHT22 18](#_Toc77929719)

[Hình 2.8 Sơ đồ thời gian khởi động 19](#_Toc77929720)

[Hình 2.9 Sơ đồ mô tả quá trình đọc dữ liệu 19](#_Toc77929721)

[Hình 2.10 Ví dụ khi DHT22 đọc nhiệu độ, độ ẩm 20](#_Toc77929722)

[Hình 2.11 Sơ đồ nối chân DHT22 với vi điều khiển 20](#_Toc77929723)

[Hình 2.12 Cảm biến khí CO MQ7 21](#_Toc77929724)

[Hình 2.13 Sơ đồ nguyên lý module cảm biến CO MQ-7 21](#_Toc77929725)

[Hình 2.14 Module cảm biến MQ-7 22](#_Toc77929726)

[Hình 2.15 Đồ thị Rs/R0 so với nồng độ khí (ppm) 22](#_Toc77929727)

[Hình 2.16 Sơ đồ đấu nối MQ-7 24](#_Toc77929728)

[Hình 2.17 Cảm biến bụi PM2.5 GP2Y1010AU0F 25](#_Toc77929729)

[Hình 2.18 Sơ đồ dây GP2Y1010AU0F 25](#_Toc77929730)

[Hình 2.19 Sơ đồ nối chân của GP2Y1010AU0F 26](#_Toc77929731)

[Hình 2.20 Nguyên lý hoạt động của cảm biến GP2Y1010AU0F 27](#_Toc77929732)

[Hình 3.1 Mạch nguyên lý nối MQ7 với vi xử lý 28](#_Toc77929733)

[Hình 3.2 Mạch nguyên lý nối DHT22 với vi xử lý 29](#_Toc77929734)

[Hình 3.3 Mạch nguyên lý nối I2C LCD với vi xử lý 29](#_Toc77929735)

[Hình 3.4 Mạch nguyên lý nối GP2Y1010AU0F với vi xử lý 30](#_Toc77929736)

[Hình 3.5 Mạch nguyên lý cuối cùng của hệ thống 31](#_Toc77929737)

[Hình 3.6 Thiết kế mạch PCB dạng 3D 31](#_Toc77929738)

[Hình 3.7 Đi dây mạch PCB 32](#_Toc77929739)

[Hình 3.8 Lưu đồ thuật toán của hệ thống 33](#_Toc77929740)

[Hình 3.9 Giao diện Arduino IDE 34](#_Toc77929741)

[Hình 3.10 Kiến trúc hoạt động giao thức MQTT 35](#_Toc77929742)

[Hình 3.11 Giao diện dashboard AdafruitIO 36](#_Toc77929743)

[Hình 3.12 Trang đăng nhập 37](#_Toc77929744)

[Hình 3.13 Trang DashBoard 37](#_Toc77929745)

[Hình 3.14 Các nguồn dữ liệu 38](#_Toc77929746)

[Hình 3.15 Tạo nguồn dữ liệu mới 38](#_Toc77929747)

[Hình 3.16 Tạo nguồn dữ liệu nhiệt độ 39](#_Toc77929748)

[Hình 3.17 Tạo nguồn dữ liệu độ ẩm 39](#_Toc77929749)

[Hình 3.18 Dánh sách các nguồn dữ liệu mới 40](#_Toc77929750)

[Hình 3.19 AIO\_Controller 40](#_Toc77929751)

[Hình 3.20 Các khối giao diện 41](#_Toc77929752)

[Hình 3.21 Chọn giao diện cho nhiệt độ 41](#_Toc77929753)

[Hình 3.22 Cài đặt khối giao diện 42](#_Toc77929754)

[Hình 3.23 Chọn giao diện cho độ ẩm 42](#_Toc77929755)

[Hình 3.24 Trang Dashboard 43](#_Toc77929756)

[Hình 3.25 Lấy mã AIO Key 43](#_Toc77929757)

[Hình 3.26 Hiển thị dữ liệu 44](#_Toc77929758)

[Hình 4.1 Mạch thực tế của nhóm 47](#_Toc77929759)

[Hình 4.2 Mạch chưa có thông tin Wi-Fi 48](#_Toc77929760)

[Hình 4.3 Mạch chuyển sang chế độ AP 49](#_Toc77929761)

[Hình 4.4 Nhập thông tin Wi-Fi từ thiết bị cá nhân 49](#_Toc77929762)

[Hình 4.5 Hệ thống đã nhận thông tin Wi-Fi mới 49](#_Toc77929763)

[Hình 4.6 Thông tin nhiệt độ, độ ẩm hiển thị trên LCD 50](#_Toc77929764)

[Hình 4.7 Thông tin về bụi PM2.5, khí CO trên LCD 50](#_Toc77929765)

[Hình 4.8 Thông tin về không khí hiển thị trên website 51](#_Toc77929766)

[Hình 4.9 Dashboard theo dõi chất lượng không khí 51](#_Toc77929767)

[Hình 4.10 Giá trị nhiệt độ, độ ẩm tăng khi sử dụng máy sấy 52](#_Toc77929768)

**DANH SÁCH BẢNG**

[Bảng 1.1 Bảng phân công công việc 10](#_Toc77293315)

[Bảng 2.1 Danh sách linh kiện 12](#_Toc77293316)

[Bảng 2.2 Thông số kỹ thuật của ESP32 14](#_Toc77293317)

[Bảng 2.3 Sơ đồ chân LCD I2C 1602 18](#_Toc77293318)

[Bảng 2.4 Sơ đồ chân module GP2Y1010AU0F 27](#_Toc77293319)

[Bảng 4.1 Bảng các bài kiểm thử hệ thống 49](#_Toc77293320)

# GIỚI THIỆU

## Đặt vấn đề

Một thực trạng đáng báo động hiện nay đó là chất lượng không khí tại nhiều thành phố lớn ở Việt Nam, đặc biệt là tại thành phố Hà Nội, đang suy giảm một cách trầm trọng [1, 2]. Một vài nguyên nhân dẫn đến tình trạng này có thể kể đến như quá trình đô thị hóa, nền công nghiệp phát triển với ngày càng nhiều nhà máy [3],… Bên cạnh đó, hàng loạt những công trình xã hội như đường xá, cầu vượt liên tục được thi công cũng là nhân tố không nhỏ gây trầm trọng thêm tình hình ô nhiễm không khí. Ngoài ra không thể không kể đến số lượng phương tiện cá nhân tham gia giao thông hàng ngày liên tục tăng lên khi một lượng lớn khí thải độc hại được thải trực tiếp ra môi trường. Chất lượng không khí tác động trực tiếp đến sức khỏe của mọi người sống trong khu vực, gây ra nhiều loại bệnh nghiêm trọng về hô hấp khi hít vào do tích tụ trong phổi [4] . Khó khăn hơn nữa, ô nhiễm không khí là sự kiện rất khó quan sát bằng mắt thường, điều này càng làm tình hình thêm phức tạp do người dân không tự nhận biết được mức độ ô nhiễm của môi trường xung quanh.

Để giải quyết vấn đề nhức nhối này, giải pháp cấp thiết quan trọng nhất đó chính là phải nắm bắt và theo dõi thông tin về chỉ số chất lượng không khí một cách kịp thời và chính xác. Yêu cầu cho lời giải này đó là dữ liệu phải được cập nhật liên tục và đầy đủ để người dùng có cái nhìn chính xác nhất về tình hình ô nhiễm hiện tại. Ngoài ra, phương tiện tiếp cận những thông tin trên cũng phải thuận tiện cho người dùng, từ đó họ mới có thể đưa ra những điều chỉnh trong sinh hoạt nhằm giảm thiểu những ảnh hưởng của ô nhiễm không khí lên sức khỏe của bản thân và gia đình.

## Giới thiệu đề tài

Từ thực trạng cùng nhu cầu nêu trên, nhóm 5 thuộc lớp Kỹ thuật vi xử lý, dưới sự hướng dẫn của TS Hàn Huy Dũng, đã quyết định thực hiện tìm hiểu và triển khai một dự án với đề tài “Đo lường chất lượng không khí”. Dự án với mục tiêu cụ thể là giám sát chất lượng không khí, hiển thị thông tin chi tiết trên website cùng với đó là đưa ra cảnh báo tới người dùng.

Chất lượng không khí được đo thông qua các chỉ số không khí như khí CO, CO2, hạt bụi mịn PM2.5, cùng với thông tin về nhiệt độ và độ ẩm. Các chỉ số này được đo lường thông qua các cảm biến chuyên dụng, trước khi được gửi về trung tâm xử lý qua giao thức truyền thông nhẹ MQTT cùng kết nối Wi-Fi và cuối cùng là hiển thị trên website cho người dùng truy cập.

Người dùng khi truy cập website sẽ có thể biết được chính xác các chỉ số không khí chi tiết cùng với cảnh báo từ hệ thống về chất lượng không khí hiện tại.

## Phân công công việc

Nhằm phát triển hệ thống “Đo lường chất lượng không khí” dựa trên những kiến thức đã học trong học phần Kỹ thuật vi xử lý cùng kiến thức tìm hiểu liên quan, cả nhóm đã thảo luận, lập kế hoạch và phân công công việc cụ thể cho từng thành viên như trong Bảng 1.1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **STT** | **Nhiệm vụ** | **Thành viên thực hiện** | **Ngày bắt đầu** | **Ngày kết thúc** | **Thời gian thực hiện (Ngày)** |
|  |
|  |
| **1** | **Xác định yêu cầu của dự án** |  |  |  |  |  |
| 1,1 | Yêu cầu chức năng | Quân, Hiếu | 13/05/2021 | 14/05/2021 | 2 |  |
| 1,2 | Yêu cầu phi chức năng | Việt | 13/05/2021 | 14/05/2021 | 2 |  |
| 1,3 | Sơ đồ hệ thống | Đan | 17/05/2021 | 18/05/2021 | 2 |  |
| 1,4 | Danh sách linh kiện | Long | 17/05/2021 | 18/05/2021 | 2 |  |
| **2** | **Tìm hiểu về linh kiện** |  |  |  |  |  |
| 2,1 | Vi điều khiển ESP32 | Long | 21/05/2021 | 25/05/2021 | 5 |  |
| 2,2 | Màn hình LCD 1602, I2C | Việt | 21/05/2021 | 25/05/2021 | 5 |  |
| 2,3 | Cảm biến DHT22 | Đan | 21/05/2021 | 25/05/2021 | 5 |  |
| 2,4 | Cảm biến MQ7 | Hiếu | 21/05/2021 | 25/05/2021 | 5 |  |
| 2,5 | Cảm biến GP2Y1010AU0F | Quân | 21/05/2021 | 25/05/2021 | 5 |  |
| **3** | **Thiết kế mạch** |  |  |  |  |  |
| 3,1 | Tính toán công suất | Hiếu | 31/05/2021 | 02/06/2021 | 3 |  |
| 3,2 | Sơ đồ nguyên lý | Long | 07/06/2021 | 08/06/2021 | 2 |  |
| 3,3 | Thiết kế mạch PCB | Quân | 09/06/2021 | 11/06/2021 | 3 |  |
| **4** | **Lập trình các cảm biến** |  |  |  |  |  |
| 4,1 | Cảm biến DHT22 | Đan | 14/06/2021 | 20/06/2021 | 7 |  |
| 4,2 | Cảm biến MQ7 | Hiếu | 14/06/2021 | 20/06/2021 | 7 |  |
| 4,3 | Cảm biến GP2Y1010AU0F | Quân | 14/06/2021 | 20/06/2021 | 7 |  |
| **5** | **Lập trình phần hiển thị** |  |  |  |  |  |
| 5,1 | Hiển thị trên LCD | Việt | 21/06/2021 | 23/06/2021 | 3 |  |
| 5,2 | Hiển thị trên web | Việt | 24/06/2021 | 27/06/2021 | 4 |  |
| 5,3 | Lưu dữ liệu trên cơ sở dữ liệu | Long | 21/06/2021 | 27/06/2021 | 7 |  |
| **6** | **Tổng hợp code** |  |  |  |  |  |
| 6,1 | Chia code thành các module | Đan | 29/06/2021 | 01/07/2021 | 3 |  |
| 6,2 | Tổng hợp, ghép nối code | Long | 02/07/2021 | 06/07/2021 | 5 |  |
| **7** | **Kiểm thử** |  |  |  |  |  |
| 7,1 | Kiểm tra từng linh kiện | Long | 08/07/2021 | 09/07/2021 | 2 |  |
| 7,2 | Kiểm tra hoạt động của mạch | Long | 10/07/2021 | 12/07/2021 | 3 |  |
| **8** | **Báo cáo** |  |  |  |  |  |
| 8,1 | Viết báo cáo tổng hợp | Cả nhóm | 13/05/2021 | 15/07/2021 | 64 |  |
| 8,2 | Quay video demo | Long | 13/07/2021 | 13/07/2021 | 1 |  |

Bảng 1.1 Bảng phân công công việc

# YÊU CẦU HỆ THỐNG

## Yêu cầu chức năng

Từ nhu cầu sử dụng thực tế của hệ thống quản lý chất lượng không khí, nhóm thực hiện đặt ra yêu cầu chức năng đối với sản phẩm cuối cùng như sau:

- Đo đạc các thông số của không khí: Nhiệt độ, độ ẩm, nồng độ khí CO, bụi PM2.5

- Hiển thị các thông số theo thời gian thực lên màn hình LCD

- Hiển thị thông tin đo được, đồ thị của các thông số lên website.

## Yêu cầu phi chức năng

Từ các đặc điểm của môi trường thực tế cũng như nhu cầu sử dụng, nhóm đưa ra các yêu cầu phi chức năng như sau:

- Nguồn cung cấp: 5V

- Độ chính xác: 95%

- Điều kiện hoạt động:

* Nhiệt độ hoạt động: -10 – 600C
* Độ ẩm hoạt động: 0 – 100%
* Dải nồng độ CO phát hiện: 10 – 1000 ppm
* Giá trị phát hiện hạt tối thiểu: 0.8 micron

- Thiết kế phần cứng:

* Hiển thị thông tin trực tiếp lên màn hình LCD các thông số đo được.
* Sản phẩm được thiết kế nhỏ gọn, có tính di động cao, kích thước 9 x 9 x 2 cm.
* Sản phẩm được thiết kế thành các module, dễ dàng tháo lắp, sửa chữa và nâng cấp sau này.

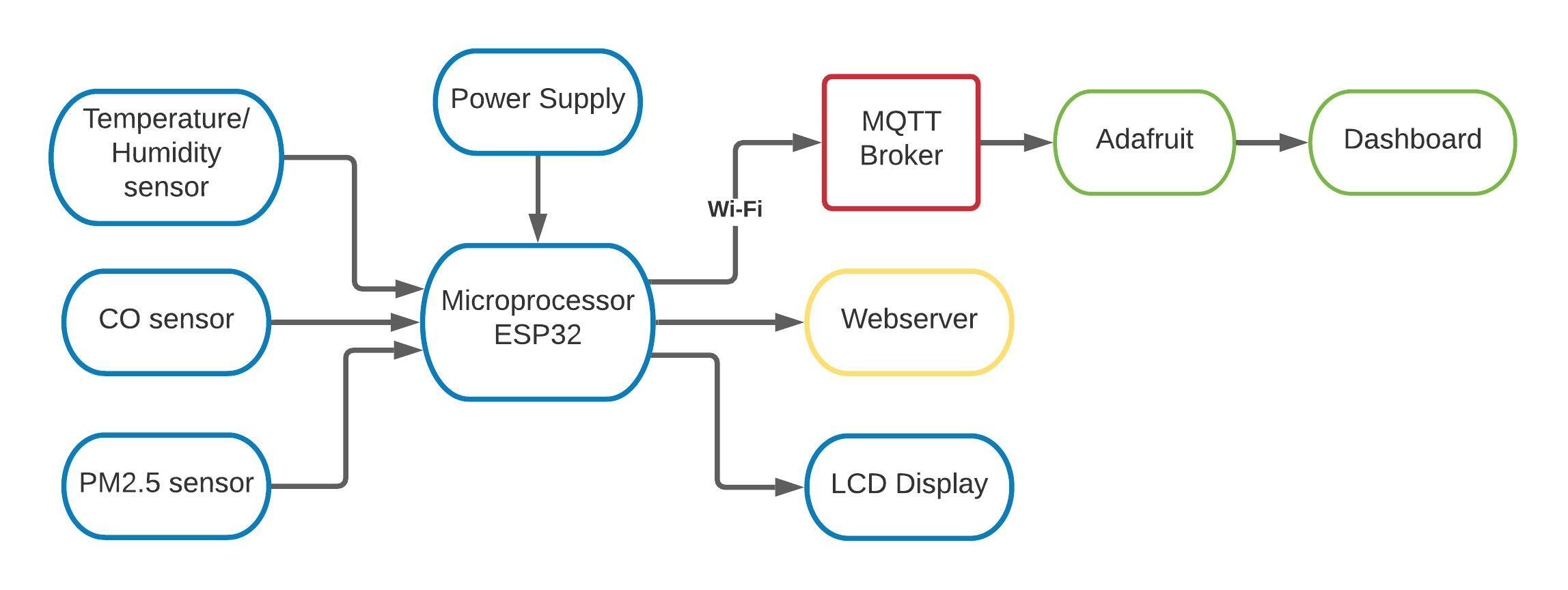
- Thiết kế website: Giao diện đơn giản, dễ sử dụng, hiển thị tối ưu cho các loại màn hình kích thước khác nhau.

- Cập nhật thông tin lên hệ thống trong khoảng thời gian 10 phút 1 lần.

- Ngôn ngữ sử dụng là tiếng Việt hoặc tiếng Anh.

## Sơ đồ hệ thống

Hình 2.1 là sơ đồ hệ thống quan trắc môi trường sử dụng ESP32. Qua sơ đồ này, ta hiểu được cách hoạt động của hệ thống. 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… Thông tin đó sẽ được hiển thị qua màn hình LCD, và được lưu trên server để hiển thị lên website.



Hình 2.1 Sơ đồ hệ thống quan trắc môi trường sử dụng ESP32

## Danh sách linh kiện cần sử dụng

|  |  |  |
| --- | --- | --- |
| **STT** | **Tên linh kiện** | **Số lượng** |
| 1 | ESP32 | 1 |
| 2 | LCD 1602 I2C | 1 |
| 3 | Cảm Biến Nhiệt Độ-Độ Ẩm DHT22 | 1 |
| 4 | Module Cảm Biến Khí CO MQ7 | 1 |
| 5 | Module cảm biến bụi PM2.5 GP2Y1010AU0F | 1 |

Bảng 2.1 Danh sách linh kiện

Bảng 2.1 là danh sách linh kiện cần sử dụng, không bao gồm các linh kiện như điện trở, tụ điện, dây dẫn… Datasheet của từng linh kiện được trích dẫn trong mục Tài liệu tham khảo của báo cáo này. Phần sau đây của báo cáo sẽ trình bày chi tiết về từng linh kiện.

### Giới thiệu về ESP32

ESP32 [5] là một series các vi điều khiển trên một vi mạch giá rẻ, năng lượng thấp có hỗ trợ WiFi và dual-mode Bluetooth (tạm dịch: Bluetooth chế độ kép). Dòng ESP32 sử dụng bộ vi xử lý Tensilica Xtensa LX6 ở cả hai biến thể lõi kép và lõi đơn, và bao gồm các công tắc antenna tích hợp, RF balun, bộ khuếch đại công suất, bộ khuếch đại thu nhiễu thấp, bộ lọc và module quản lý năng lượng. ESP32 được chế tạo và phát triển bởi Espressif Systems, một công ty Trung Quốc có trụ sở tại Thượng Hải, và được sản xuất bởi TSMC bằng cách sử dụng công nghệ 40 nm. ESP32 là sản phẩm kế thừa từ vi điều khiển ESP8266.

Ta có thể lập trình bằng nhiều ngôn ngữ khác nhau như C/C++, Python, NodeJs, Lua,… Ngoài ra, hiện tại Espressif Systems hỗ trợ chính thức SDK cho ESP32 với Arduino IDE, vì vậy ta có thể yên tâm xây dựng các ứng dụng với Arduino ESP32.

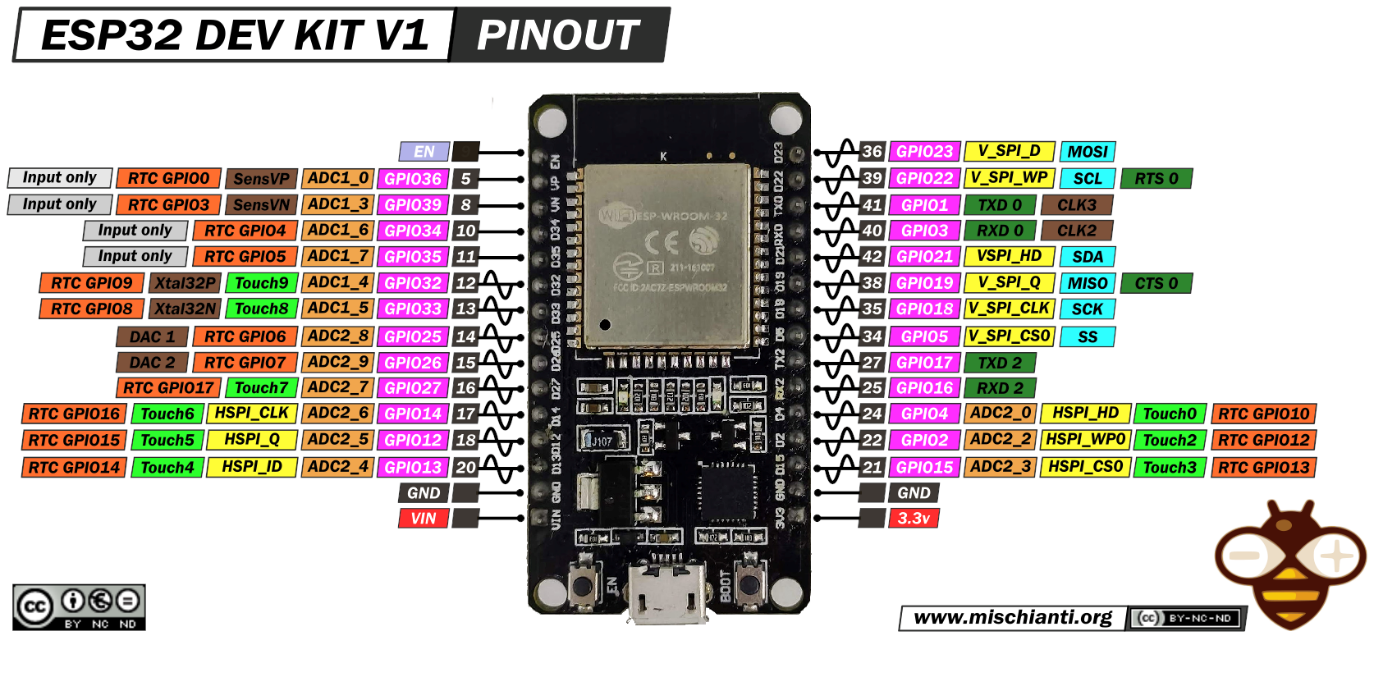
Bảng 2.2 là thông số kỹ thuật của ESP32 [6].

|  |  |
| --- | --- |
| Bộ xử lý | * CPU: Bộ vi xử lý Xtensa lõi kép (hoặc lõi đơn) 32-bit LX6,[4] hoạt động ở tần số 160 hoặc 240 MHz và hoạt động ở tối đa 600 DMIPS * Bộ đồng xử lý (co-processor) công suất cực thấp (Ultra low power, viết tắt: ULP) |
| Bộ nhớ nội | * 448 KB bộ nhớ ROM cho việc booting và các tính năng lõi * 520 KB bộ nhớ SRAM trên chip cho dữ liệu và tập lệnh |
| Kết nối không dây | * Wi-Fi: 802.11 b/g/n/e/i * Bluetooth: v4.2 BR/EDR and BLE |
| Các chuẩn giao tiếp | * ADC SAR 12 bit, 18 kênh * DAC 2 × 8-bit * 10 cảm biến cảm ứng (touch sensor) (GPIO cảm ứng điện dung) * 4 SPI * 2 giao diện I²S * 2 giao diện I²C * 3 UART (UART0, UART1, UART2) với tốc độ lên đến 5 Mbps[6] * SD/SDIO/CE-ATA/MMC/eMMC host controller * SDIO/SPI slave controller * Ethernet MAC interface cho DMA và IEEE 1588 Precision Time Protocol (tạm dịch: Giao thức thời gian chính xác IEEE 1588) * CAN bus 2.0 * Bộ điều khiển hồng ngoại từ xa (TX/RX, lên đến 8 kênh) * PWM cho điều khiển động cơ * LED PWM (lên đến 16 kênh) * Cảm biến hiệu ứng hall * Bộ tiền khuếch đại analog công suất cực thấp (Ultra low power analog pre-amplifier) |
| Bảo mật | * Hỗ trợ tất cả các tính năng bảo mật chuẩn IEEE 802.11, bao gồm WFA, WPA/WPA2 và WAPI * Secure boot * Mã hoá flash * 1024-bit OTP, lên đến 768-bit cho khách hàng * Tăng tốc mã hóa phần cứng: AES, SHA-2, RSA, elliptic curve cryptography (ECC, tạm dịch: mật mã đường cong ellip), trình tạo số ngẫu nhiên (random number generator, viết tắt: RNG). |
| Quản lý năng lượng | * Bộ ổn áp nội với điện áp rơi thấp (internal low-dropout regulator) * Miền nguồn riêng (individual power domain) cho RTC * Dòng 5 μA cho chế độ deep sleep * Trở lại hoạt động từ ngắt GPIO, timer, đo ADC, ngắt với cảm ứng điện dung |
| Nguồn điện hoạt động | * Nhiệt độ hoạt động -40 + 85C * Điện áp hoạt động: 2.2 – 5 V * Số cổng GPIOs : 34 |

Bảng 2.2 Thông số kỹ thuật của ESP32

Các tính năng ADC (bộ chuyển đổi tương tự sang kỹ thuật số) và DAC (bộ chuyển đổi kỹ thuật số sang tương tự) được gán cho các chân tĩnh cụ thể. Tuy nhiên, bạn có thể quyết định các chân nào là UART, I2C, SPI, PWM, v.v. - bạn chỉ cần gán chúng trong mã. Điều này có thể xảy ra do tính năng ghép kênh của chip ESP32.

Bo mạch ESP32 DEVKIT V1 DOIT có các chân được gán như Hình 2.2. Ngoài ra, có các chân với các tính năng cụ thể làm cho chúng phù hợp hoặc không cho một dự án cụ thể.



Hình 2.2 Sơ đồ chân ESP32 DEVKIT V1 DOIT

### Giới thiệu về màn hình LCD 1602 với module I2C

#### 2.4.2.1 Chuẩn giao tiếp I2C

I2C [7] là tên viết tắt của cụm từ tiếng anh “Inter-Integrated Circuit”. Nó là một giao thức giao tiếp được phát triển bởi Philips Semiconductors để truyền dữ liệu giữa một bộ xử lý trung tâm với nhiều IC trên cùng một board mạch chỉ sử dụng hai đường truyền tín hiệu. Do tính đơn giản của nó nên loại giao thức này được sử dụng rộng rãi cho giao tiếp giữa vi điều khiển và mảng cảm biến, các thiết bị hiển thị, thiết bị IoT, EEPROMs, v.v…

I2C sử dụng hai đường truyền tín hiệu [8] bao gồm:

* Một đường xung nhịp đồng hồ (SCL)
* Một đường dữ liệu theo 2 hướng (SDA)

Dữ liệu được truyền đi được gửi qua dây SDA và được đồng bộ với tín hiệu đồng hồ (clock) từ SCL. Tất cả các thiết bị hoặc IC trên mạng I2C được kết nối với cùng đường bus SCL và SDA như Hình 2.3:

Diagram

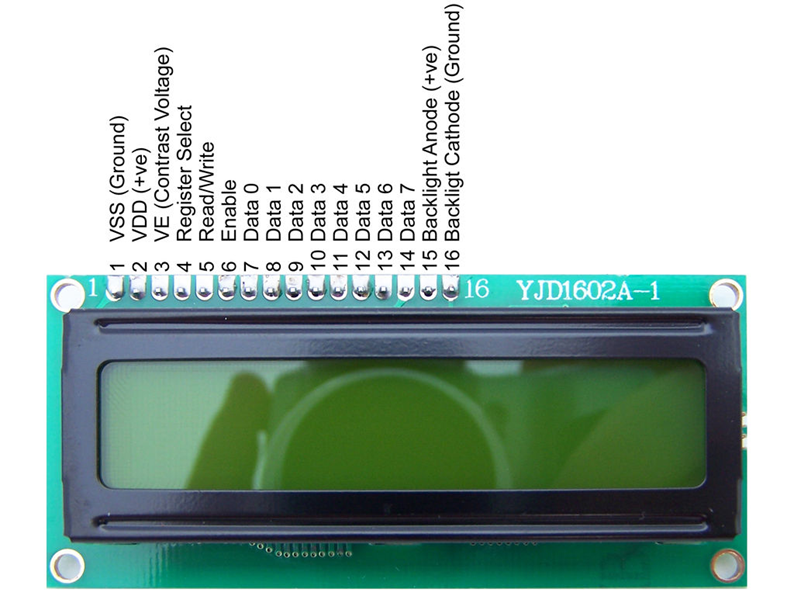
Description automatically generated

Hình 2.3 Đường bus kết nối của chuẩn I2C

Các thiết bị kết nối trên mạng I2C được phân loại hoặc là thiết bị Chủ (Master) hoặc là thiết bị Tớ (Slave). Ở bất cứ thời điểm nào thì chỉ có duy nhất một thiết bị Master ở trạng thái hoạt động trên bus I2C. Nó điều khiển đường tín hiệu đồng hồ SCL và quyết định hoạt động nào sẽ được thực hiện trên đường dữ liệu SDA.

Tất cả các thiết bị đáp ứng các hướng dẫn từ thiết bị Master này đều là Slave. Để phân biệt giữa nhiều thiết bị Slave được kết nối với cùng một bus I2C, mỗi thiết bị Slave được gán một địa chỉ vật lý 7-bit cố định.

#### 2.4.2.2 Màn hình LCD 1602



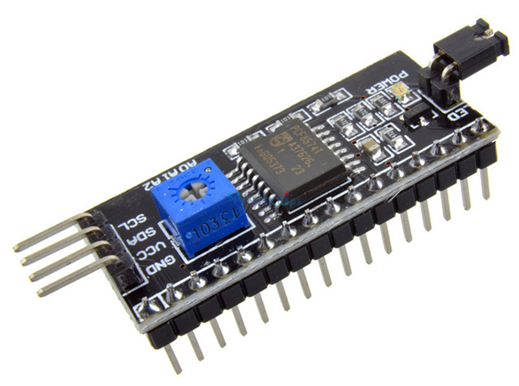
**Hình 2.4 Màn hình LCD 1602**

Màn hình LCD 1602 [9] (Hình 2.4) có khả năng hiển thị 2 dòng với mỗi dòng 16 ký tự, màn hình có độ bền cao, rất phổ biến.

Thông số kĩ thuật:

* Điện áp hoạt động: 5V DC.
* Kích thước: 80 x 36 x 12.5 mm
* Độ phân giải: 5x8 pixel
* Sử dụng 16 chân bao gồm 8 chân dữ liệu (D0 – D7), 3 chân điều khiển (RS, RW, EN) và 5 chân còn lại dùng để cấp nguồn.

#### 2.4.2.3 Module kết nối LCD1602 bằng chuẩn I2C



Hình 2.5 Module I2C

Module LCD I2C [10] (Hình 2.5) cho phép các vi điều khiển kết nối với màn hình LCD bằng 2 chân (SCL, SDA) của module qua giao tiếp I2C thay vì phải mất 6 chân của vi điều khiển để kết nối với màn LCD. Bộ phận quan trọng nhất trong module là một IC I/O mở rộng 8 bit PCF8574, nó cho phép chuyển đổi dữ liệu I2C từ vi điều khiển thành dữ liệu có thể hiển thị lên màn hình LCD.

Thông số kĩ thuật:

* Điện áp hoạt động: 2.5-6V DC
* Kích thước: 41.5 x 19 x 15.3 mm
* Hỗ trợ màn hình: LCD1602,1604,2004
* Giao tiếp: I2C

#### 2.4.2.4 Giao tiếp ESP32 và LCD 1602 I2C

Để kết nối giữa ESP32 và màn hình LCD 1602 I2C cần phải thiết lập một mạng I2C với 2 đường bus SCL và SDA được mô tả ở bên dưới. Trong mạng I2C, ESP32 đóng vai trò là thiết bị Chủ (Master) còn LCD 1602 I2C là thiết bị Khách (Slave) với địa chỉ là 0x27 hoặc 0x3F [11].

Hình 2.6 và Bảng 2.3 mô tả sơ đồ nối chân giữa màn hình LCD I2C với ESP32.

|  |  |
| --- | --- |
| **Module I2C LCD 1602** | **ESP 32** |
| GND | GND |
| VCC | VIN |
| SDA | GPIC 21 (D21) |
| SCL | GPIO 22 (D22) |

Bảng 2.3 Sơ đồ chân LCD I2C 1602

A picture containing text, electronics

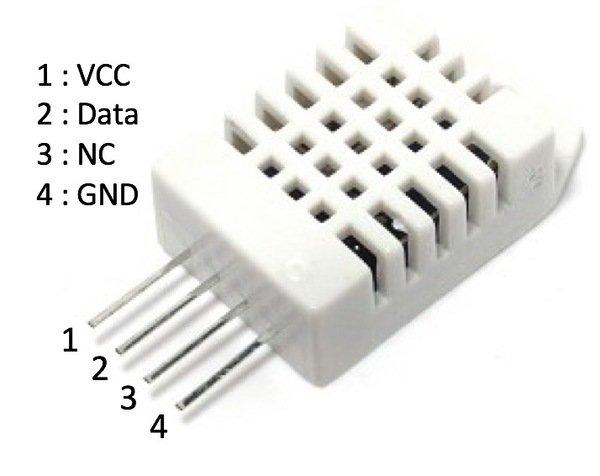
Description automatically generated

Hình 2.6 Sơ đồ đấu nối ESP32 với LCD I2C

### Giới thiệu về cảm biến nhiệt độ, độ ẩm DHT22

#### 2.4.3.1 Giới thiệu

Cảm biến độ ẩm và nhiệt độ DHT22 [12] Temperature Humidity Sensor (Hình 2.7) sử dụng giao tiếp 1 Wire dễ dàng kết nối và giao tiếp với vi điều khiển để thực hiện các ứng dụng đo nhiệt độ, độ ẩm môi trường, cảm biến có chất lượng tốt, kích thước nhỏ gọn, độ bền và độ ổn định cao.



**Hình 2.7 Các chân của DHT22**

Thông số kĩ thuật:

* Nguồn sử dụng: 3~5VDC.
* Dòng sử dụng: 2.5mA max (khi truyền dữ liệu).
* Đo tốt ở độ ẩm từ 0 đến 100%RH với sai số 2-5%.
* Đo tốt ở nhiệt độ từ -40 đến 80°C sai số ±0.5°C.
* Tần số lấy mẫu tối đa 0.5Hz (2 giây 1 lần)
* Kích thước 27mm x 59mm x 13.5mm (1.05" x 2.32" x 0.53")
* 4 chân, khoảng cách chân 0.1''

#### 2.4.3.2 Nguyên lý làm việc

Dữ liệu được đọc đến vi điều khiển từ chân Data. Vì không có chân đồng hồ như trong I2C và SPI, DHT22 phải hoạt động trong nối tiếp không đồng bộ, như RS-232. Điện trở kéo lên cần thiết cũng có nghĩa là cảm biến có thể hoạt động với cả nguồn 3.3 V và 5 V.

Sau đây là các bước để đo nhiệt độ, độ ẩm từ DHT22 [13]:

* **Bước 1.** Khởi động Module

Để bắt đầu yêu cầu DHT22 cung cấp dữ liệu về độ ẩm và nhiệt độ, bộ vi điều khiển phải thực hiện tín hiệu khởi động, tín hiệu này là mức logic 0 (thấp) trong ít nhất 500 micro giây sau đó là mức logic 1 (cao). Sau đó, DHT22 phải phản hồi bằng cách kéo bus xuống thấp trong 80 micro giây, sau đó kéo bus lên cao trong 80 micro giây nữa trước khi kéo xuống thấp trở lại.

Sơ đồ thời gian khởi động được minh hoạ như Hình 2.8

A picture containing timeline

Description automatically generated

**Hình 2.8 Sơ đồ thời gian khởi động**

Đường màu đỏ được bắt đầu bởi vi điều khiển trong khi đường màu xanh lam là tín hiệu từ DHT22. Chỉ sau khi khởi động đúng, DHT22 mới bắt đầu gửi dữ liệu.

* **Bước 2.** Đọc dữ liệu nhiệt độ và độ ẩm

DHT22 sẽ gửi giá trị độ ẩm trước tiên, sau đó là giá trị nhiệt độ. Cả hai đều là dữ liệu 16 bit, nhưng chỉ gửi 8 bit (1 byte) một lúc. Bit quan trọng nhất của byte cao được gửi đầu tiên. MSB của byte nhiệt độ cao là bit dấu; nếu bit này cao, nhiệt độ là âm. Nếu không, nhiệt độ là dương.

Mức logic 1 là xung thấp dài 50 micro giây tiếp theo là xung cao dài 70 micro giây. Trong khi đó, mức logic 0 là xung thấp dài 50 micro giây tiếp theo là xung cao dài 26 micro giây. Sơ đồ thời gian được minh hoạ như Hình 2.9:

A picture containing diagram

Description automatically generated

Hình 2.9 Sơ đồ mô tả quá trình đọc dữ liệu

Lưu ý rằng độ dài xung thực tế có thể không chính xác với giá trị trên. Trên thực tế, biểu dữ liệu nói rằng xung thấp trên cả logic 0 và 1 (50 us) là từ 48 đến 55 us. Hơn nữa, xung cao logic 1 là từ 68 đến 75 us trong khi xung cao logic 0 là từ 22 đến 30 us.

Sau khi gửi đi hai byte cho độ ẩm và hai byte cho nhiệt độ, thiết bị kết thúc quá trình truyền với một byte chẵn lẻ trước khi kéo bus lên cao. Byte chẵn lẻ chỉ là tổng của bốn byte dữ liệu:

parity\_byte = humidity.high\_byte + humidity.low\_byte + temp.high\_byte + temp.low\_byte

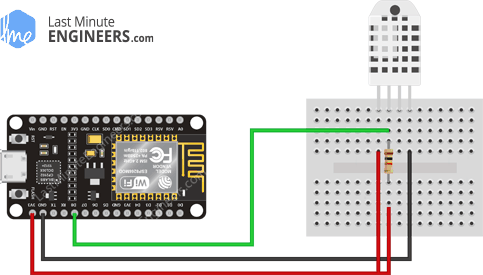
Hình 2.10 là một biểu đồ thời gian khi DHT22 gửi một số đọc nhiệt độ và độ ẩm:

Timeline

Description automatically generated

Hình 2.10 Ví dụ khi DHT22 đọc nhiệu độ, độ ẩm

#### 2.4.3.3 Sơ đồ kết nối với vi điều khiển



**Hình 2.11 Sơ đồ nối chân DHT22 với vi điều khiển**

Hình 2.11 là sơ đồ nối chân DHT22 với vi điều khiển. Trong đó chân 2 (Chân truyền dữ liệu) phải được nối với một điện trở kéo lên.

### Giới thiệu về cảm biến khí CO MQ7

#### 2.4.4.1 Giới thiệu

Cảm biến khí CO MQ-7 [13] (Hình 2.12) là cảm biến bán dẫn có giá rẻ có khả năng phát hiện khí carbon monoxide có nồng độ từ 20 đến 2000 ppm. Vật liệu tạo ra cảm biến là từ chất SnO2, có độ dẫn điện thấp trong không khí sạch.

A picture containing orange

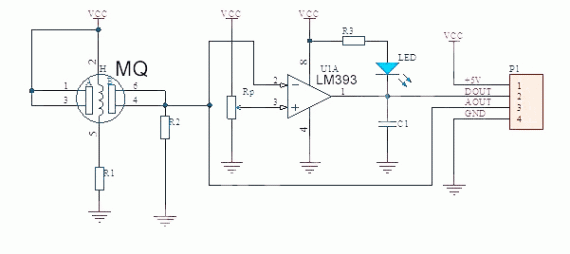
Description automatically generated

Hình 2.12 Cảm biến khí CO MQ7

Cảm biến khí CO MQ7 có độ nhạy cao và thời gian đáp ứng nhanh. Có 2 dạng tín hiệu ngõ ra là analog và digital. Cảm biến có thể hoạt động được ở nhiệt độ từ: -20 độ C đến 50 độ C và tiêu thụ dòng khoảng 150mA tại 5V. Tuổi thọ cao, chi phí thấp.

#### 2.4.4.2 Module cảm biến CO MQ7

Hình 2.13 là sơ đồ nguyên lý của module cảm biến CO MQ7 và hình 2.14 là hình ảnh thực tế của module trong thực tế.



Hình 2.13 Sơ đồ nguyên lý module cảm biến CO MQ-7

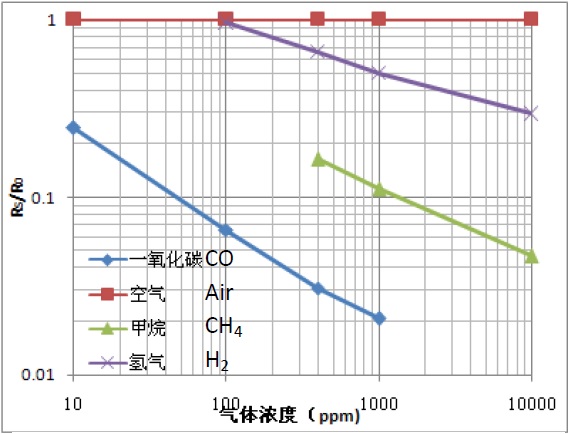
A close-up of a stethoscope

Description automatically generated with medium confidence

Hình 2.14 Module cảm biến MQ-7

#### 2.4.4.3 Nguyên lý làm việc

Theo datasheet cảm biến MQ-7 [14], cảm biến Carbon Monoxide MQ-7 phát hiện nồng độ khí CO từ 20 đến 2000 ppm trong không khí. Đây là đường cong đặc tính độ nhạy của nó:



Hình 2.15 Đồ thị Rs/R0 so với nồng độ khí (ppm)

Đây là đồ thị của Rs/R0 so với nồng độ khí tính bằng ppm. Rs là điện trở của cảm biến trong khí mục tiêu trong khi R0 là điện trở trong không khí sạch. Nhóm sẽ sử dụng biểu đồ này trong quá trình lập trình cảm biến này.

Có hai cách để đọc đầu ra từ MQ-7 [15]. Một là thông qua chân DOUT cho mức cao khi đạt đến ngưỡng nồng độ và thấp nếu không. Có thể thay đổi ngưỡng bằng cách điều chỉnh Rp.

Trong khi đó, chân AOUT cung cấp điện áp thay đổi đại diện cho nồng độ CO. Ta có thể chuyển đổi giá trị điện áp thành ppm nếu chúng ta nhìn vào đường đặc tính ở trên, đó là một biểu đồ log-log.

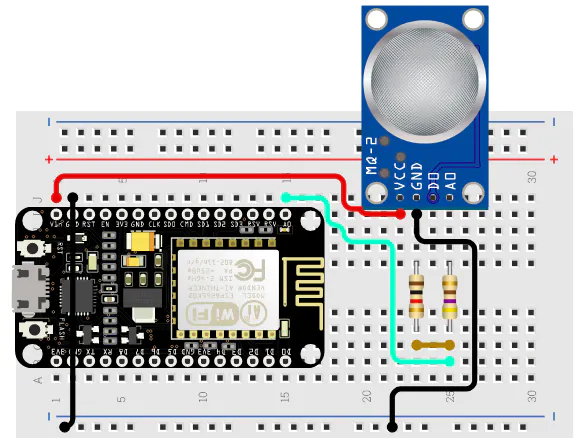
Ta chỉ quan tâm đến đường màu xanh lam trên biểu đồ cho biết nồng độ CO. Đồ thị của đường này được biểu diễn bởi phương trình

Trong đó , , , là 2 điểm bất kì trên đồ thị.

Từ đó, theo [16] tính toán được công thức tính nồng độ bụi là:

Trong đó Rs/R0 có thể được tính bằng công thức sau đây:

#### 2.4.4.4 Sơ đồ kết nối với vi điều khiển



Hình 2.16 Sơ đồ đấu nối MQ-7

Trong đó, chân AOUT nối vào chân ADC của vi điều khiển, còn chân DOUT nối vào chân vào của vi điều khiển

### Giới thiệu về cảm biến bụi PM2.5 GP2Y1010AU0F

#### 2.4.5.1 Giới thiệu

GP2Y1010AU0F Cảm Biến Bụi PM2.5 [17] (hình 2.17) là một diode phát tia hồng ngoại (IRED) và một phototransistor được bố trí theo đường chéo vào thiết bị này. Nó phát hiện ánh sáng phản xạ của bụi trong không khí. Đặc biệt, nó có hiệu quả phát hiện các hạt rất mịn như khói thuốc lá. Ngoài ra, nó có thể phân biệt khói với bụi nhà bằng dạng xung của điện áp đầu ra.

Mạch cảm biến này chủ yếu được sử dụng cho thiết bị báo động thiết bị loại bỏ bụi, thiết bị lọc không khí, rô bốt bụi, báo cháy,...



**Hình 2.17 Cảm biến bụi PM2.5 GP2Y1010AU0F**

Thông số kỹ thuật:

* Nguồn: 3.3 VDC
* Dòng tiêu thụ: 10mA
* Ngõ ra: analog với tỉ lệ 0.5V ~ 0.1mg/m3
* Nhiệt độ hoạt động: -40 ~ 85 độ C



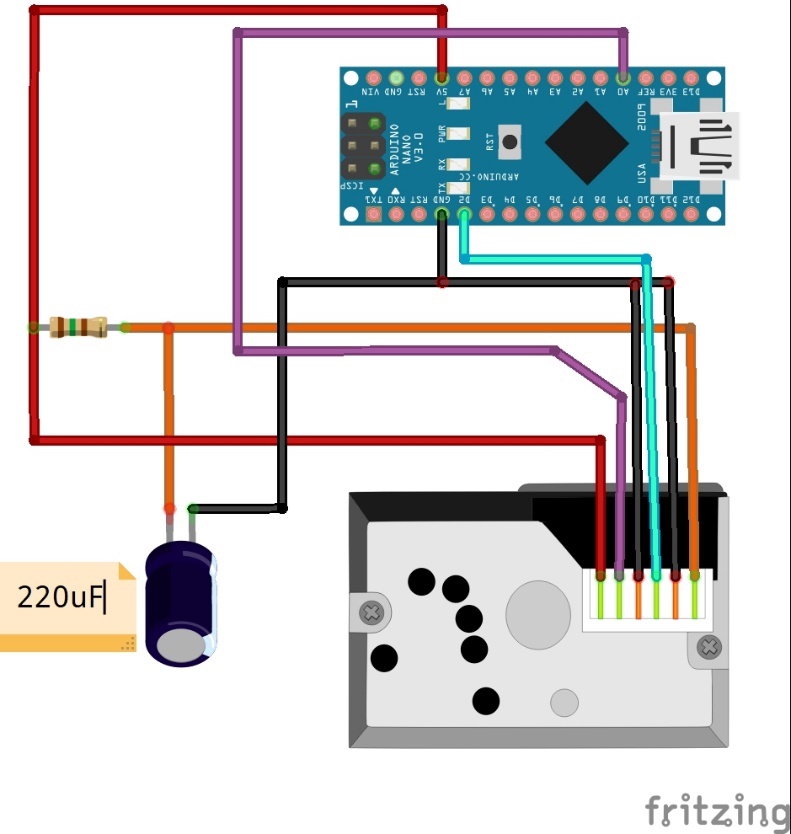
Hình 2.18 Sơ đồ dây GP2Y1010AU0F

Mỗi chân biểu thị như hình 2.18:

|  |  |
| --- | --- |
| 1 (V-LED) | 3.3V Pin (140 Ohm between) |
| 2 (LED-GND) | GND Pin |
| 3 (LED) | Digital Pin |
| 4 (S-GND) | GND Pin |
| 5 (Vo) | Analog Pin |
| 6 (Vcc) | 3.3V Pin (Direct) |

Bảng 2.4 Sơ đồ chân module GP2Y1010AU0F

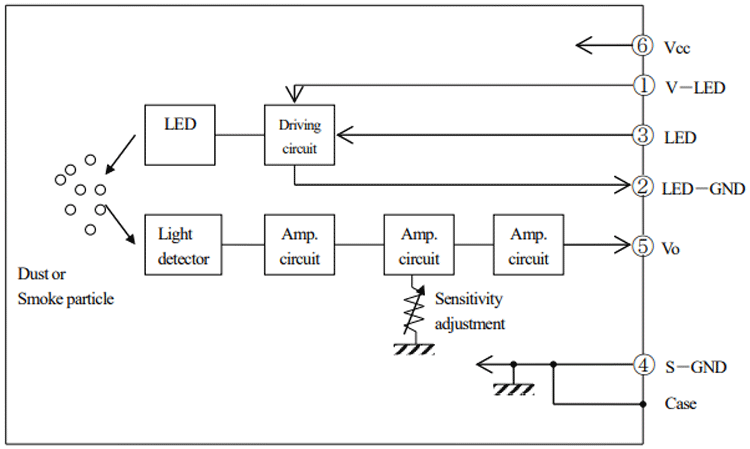
#### 2.4.5.3 Sơ đồ kết nối với vi điều khiển



Hình 2.19 Sơ đồ nối chân của GP2Y1010AU0F

Hình 2.19 là sơ đồ nối chân của cảm biến với vi điều khiển. Trong đó điện trở và tụ điện là 2 linh kiện bắt buộc, nếu không có 2 linh kiện này, thiết bị sẽ không hoạt động.

#### 2.4.5.2 Nguyên lý làm việc



Hình 2.20 Nguyên lý hoạt động của cảm biến GP2Y1010AU0F

Nguyên lý hoạt động cơ bản [18] của cảm biến này là quang học và cụ thể hơn là nguyên lý tán xạ ánh sáng. Không khí đi vào cảm biến thông qua một van nạp khí nơi đèn LED hồng ngoại bật tắt liên tục rơi vào không khí. Đèn hồng ngoại này chiếu sáng các hạt bụi có sẵn trong không khí. Đáp lại, một tín hiệu ánh sáng phân tán được tạo ra. Tín hiệu ánh sáng phân tán này được phát hiện bởi transistor quang của máy dò ánh sáng. Đầu ra của mạch tách sóng được khuếch đại bởi mạch khuếch đại nhiều tín hiệu. Sau đó, một tín hiệu ánh sáng khuếch đại được xử lý để có được nồng độ của các hạt bụi trong không khí. Cuối cùng, cảm biến bụi quang tạo ra tín hiệu điện áp tương tự trên chân Vo theo nồng độ của các hạt bụi hoặc khói trong không khí.

Điểm quan trọng nhất cần lưu ý ở đây là độ lớn của điện áp đầu ra phụ thuộc vào cường độ của ánh sáng tán xạ mà mạch dò ánh sáng phát hiện được. Hơn nữa, cường độ của ánh sáng tán xạ phụ thuộc vào nồng độ của các hạt bụi trong không khí.

Theo [19], và so sánh với các thiết bị đo thực tế, ta thu được công thức tính nồng độ bụi như sau:

# CHƯƠNG 3. THIẾT KẾ HỆ THỐNG

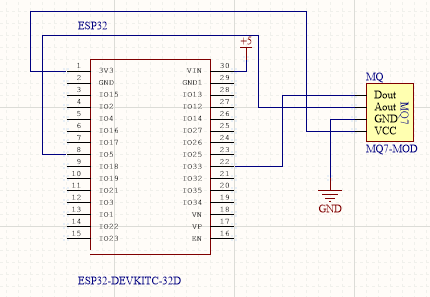
## 3.1 Thiết kế phần cứng

### 3.1.1 Phân tích công suất

#### 3.1.1.1 Mạch sensor MQ-7

Sensor MQ-7 [14] hoạt động dưới điện áp VC = 5±0.1V, VH = 5±0.1V, VL = 1,4±0.1V. Với công suất hoạt động trong vùng 350mW.

Sơ đồ đơn giản khi đấu nối với ESP32.

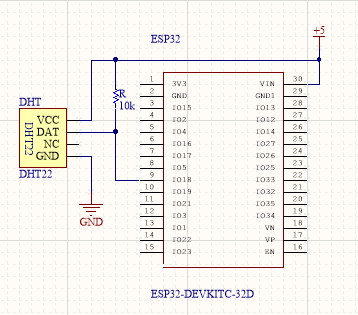


Hình 3.1 Mạch nguyên lý nối MQ7 với vi xử lý

#### 3.1.1.2 Mạch sensor DHT-22

Sensor DHT-22 [20] hoạt động dưới điện áp 3.3V – 5V với công suất hoạt động trong khoảng 10mW.

Sơ đồ đơn giản khi đấu với ESP32.

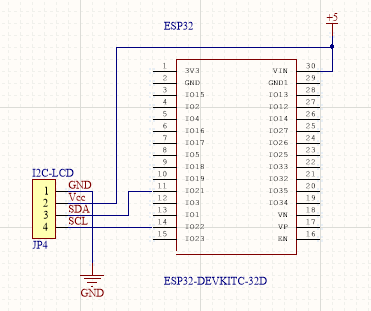


Hình 3.2 Mạch nguyên lý nối DHT22 với vi xử lý

#### 3.1.1.3 Màn hình LCD và chuẩn kết nối I2C

Với điện áp hoạt động 5V thì màn hình LCD và chuẩn kết nối I2C [21] tiêu thụ một lượng công suất khoảng 200mW

Mạch có sơ đồ kết nối đơn giản như sau :

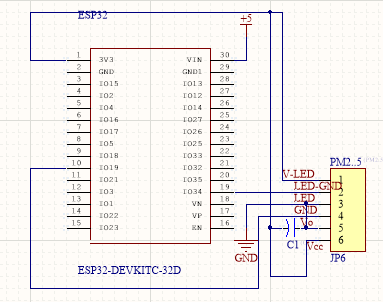


Hình 3.3 Mạch nguyên lý nối I2C LCD với vi xử lý

#### 3.1.1.4 Sensor đo bụi mịn 2.5P GP2Y1010AU0F

Điện áp hoạt động của sensor này trong khoảng từ 3.3 V đến 5V, tiêu thụ một lượng công suất trong khoảng 50mW [22].

Sơ đồ kết nối chân đơn giản của sensor.



Hình 3.4 Mạch nguyên lý nối GP2Y1010AU0F với vi xử lý

### 3.1.2 Thiết kế mạch

#### 3.1.2.1 Thiết kế mạch Schematic

Dựa vào hiệu điện thế của từng con và công suất hoạt động ổn định của tất cả con sensor và bộ hiển thị thì chúng ta có thể kết hợp cho các con sensor và màn hình hiển thị này vào bộ vi xử lý ESP 32. Chúng ta sẽ cần một vài điện trở và tụ điện để có thể nối các con sensor này để có kết quả trở nên chính xác hơn. Vì trong thiết kế Altium chưa có thư viện cho bộ I2C nên việc thiết kế mô phỏng và mạch sẽ dùng các chân jump nối ra để có thể nối với 1 con module I2C bên ngoài. Tổng công suất tiêu thụ của toàn bộ mạch sẽ rơi trên khoảng 650mW trong đó phần lớn công suất sẽ rơi trên Sensor và bộ hiển thị, vi xử lý, và một phần nhỏ công suất rơi trên tải và điện trở an toàn.

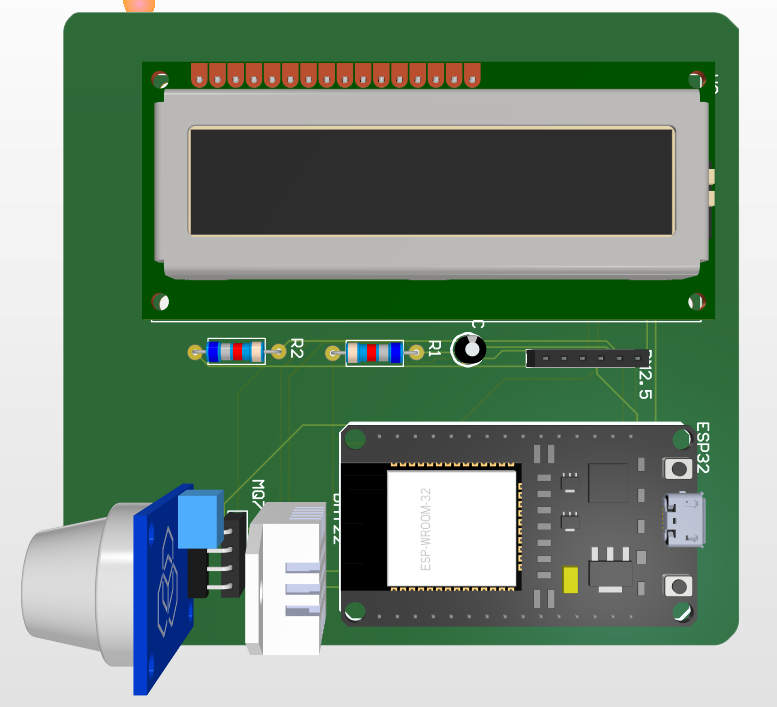
Cuối cùng mạch Schematic có sơ đồ như sau:



Hình 3.5 Mạch nguyên lý cuối cùng của hệ thống

#### 3.1.2.2 Thiết kế mạch PCB

Dựa vào sơ đồ schematic của mạch trên, sừ dụng phần mềm altium để có thể mô phỏng mạch và đây là kết quả:



Hình 3.6 Thiết kế mạch PCB dạng 3D

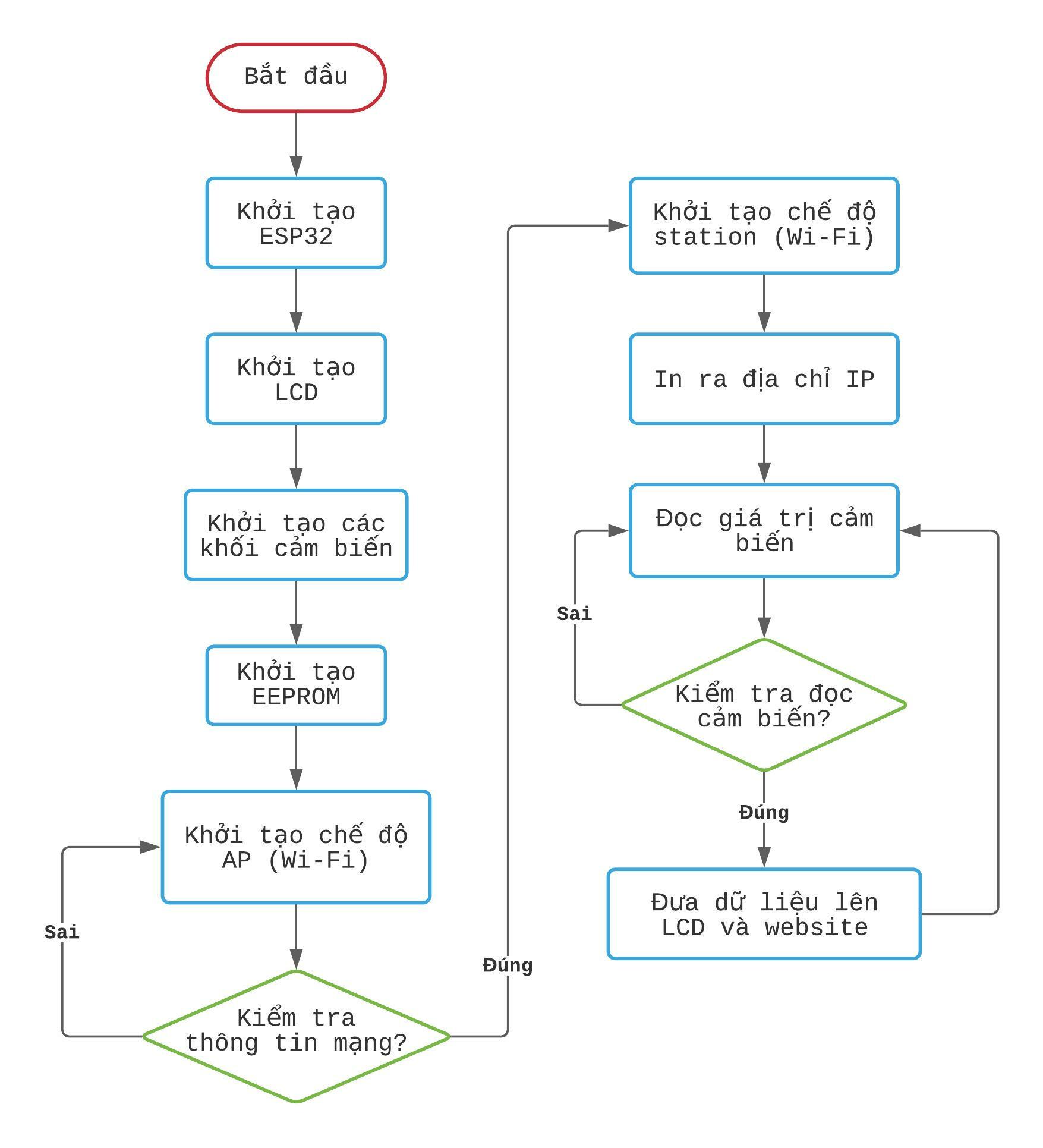


Mô tả được tạo tự động

Hình 3.7 Đi dây mạch PCB

## 3.2 Thiết kế phần mềm

### 3.2.1 Lưu đồ thuật toán



Hình 3.8 Lưu đồ thuật toán của hệ thống

### 3.2.2 Công nghệ sử dụng

#### 3.2.2.1 Arduino

Arduino [23], [24] là một nền tảng mã nguồn mở bao gồm cả phần mềm và phần cứng được thiết kế giúp làm việc với các board mạch điện tử trở nên dễ dàng hơn bao giờ hết.

Arduino che dấu đi sự phức tạp của điện tử bằng cách đơn giản hóa các khái niệm, che đi sự phức tạp của phần mềm bằng các thủ tục ngắn gọn thay phải làm việc xuống mức cấp thấp của hardware, nhưng với Arduino thì chỉ cần gọi 1 hàm.

Bởi vì tính phổ biến và dễ dùng cùng với các vô vàn thư viện được tích hợp sẵn để làm việc với rất nhiều module phần cứng khác nhau, bạn chỉ cần quan tâm đến tính năng sản phẩm mà bỏ qua các khái niệm phức tạp (protocol, datasheet …​) từ đó dễ dàng tiếp cận và làm ra các sản phẩm tuyệt vời mà không cần phải biết nhiều về điện tử.

Arduino bao gồm một phần mềm lập trình là Arduino IDE và một tập hợp rất nhiều các board mạch Arduino có thể lập trình được bằng phần mềm này với các biến thể khác nhau. Ban đầu phần lớn các board này đều dựa trên các chip họ AVR của Atmel sản xuất, nhưng sau này có rất nhiều nhà sản xuất sử dụng các chip khác nhau như ARM, PIC, STM32 gần đây nhất là ESP8266, ESP32… với năng lực phần cứng và phần mềm đi kèm mạnh mẽ hơn nhiều cũng phát hành các thư viện giúp làm việc được với Arduino như các board Arduino chính chủ.

Graphical user interface, text, application

Description automatically generated

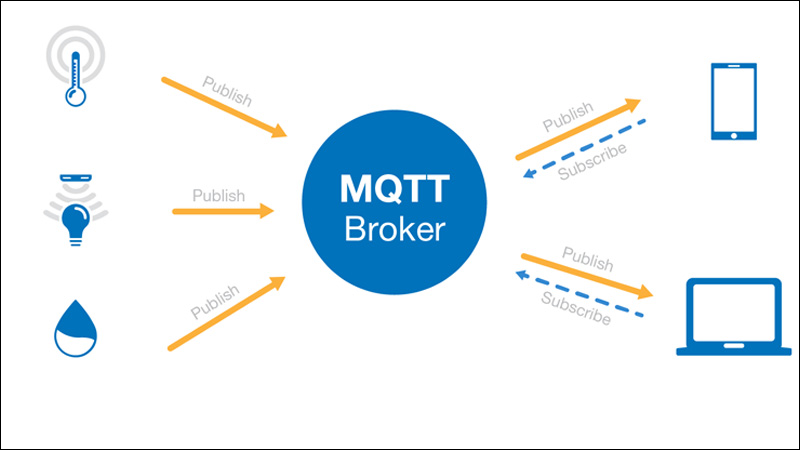
Hình 3.9 Giao diện Arduino IDE

#### 3.2.2.2 MQTT

MQTT [25], [26] - viết tắt của Message Queueing Telemetry Transport, là một **giao thức mạng mở**, dùng để **truyền thông điệp** giữa các thiết bị. MQTT được xem là giao thức nhắn tin tiêu chuẩn cho IoT bởi nó hoạt động truyền tải cực kỳ nhẹ, có độ tin cậy cao và kết nối với mức băng thông tối thiểu.

Xuất hiện đầu tiên vào năm 1999, cho đến nay, MQTT đã được sử dụng rộng rãi trong nhiều lĩnh vực. Được biết, phiên bản mới nhất của MQTT là **MQTT 5.0**với nhiều thay đổi tối ưu hơn so với các phiên bản trước đó.

Kiến trúc MQTT bao gồm 2 phần chính là **Broker**- có nhiệm vụ xuất bản và **Client**- có nhiệm vụ đăng ký.Trong đó, MQTT Broker có vai trò như một trung tâm lưu trữ thông tin, trong khi đó MQTT Client sẽ bao gồm 2 nhóm là Publisher (xuất bản) và Subscriber (đăng ký).



Hình 3.10 Kiến trúc hoạt động giao thức MQTT

Broker chính là cầu nối giữa các Publisher và Subscriber, Broker nhận thông tin từ Publisher, sau đó những Client nào có đăng ký topic (chủ đề) thông tin đó trên Broker sẽ nhận được thông tin.

Mô hình này được thiết kế để việc giao nhận thông tin diễn ra ngay cả khi đường truyền không ổn định, và là giao thức lý tưởng cho các ứng dụng M2M (Machine to Machine - Máy đến máy).

*3.2.2.3 Adafruit IO*

Adafruit IO [27] là nền tảng dựa trên dịch vụ đám mây khá nổi tiếng và phổ biến trong cộng đồng IoT, được thiết kế để đơn giản hóa các công việc liên quan đến dữ liệu trong các dự án.

Adafruit IO cho phép :

* Hiển thị dữ liệu theo thời gian thực
* Thực hiện các dự án có kết nối internet : Điều khiển động cơ, đọc dữ liệu từ xa.
* Kết nối dự án với các dịch vụ web như : Twitter, RSS feed, các dịch vụ thời tiết,…
* Kết nối dự án với các thiết bị internet khác

Ngoài ra, Adafruit IO cung cấp sẵn Dashboard, một giao diện GUI, cho phép người dùng dễ dàng tạo đồ thị, biểu đồ, ghi nhật ký và hiển thị dữ liệu lên trên web một các dễ dàng, không cần lập trình.

A screenshot of a computer

Description automatically generated with medium confidence

Hình 3.11 Giao diện dashboard AdafruitIO

Theo các bước dưới đây để thiết lập Adafruit IO

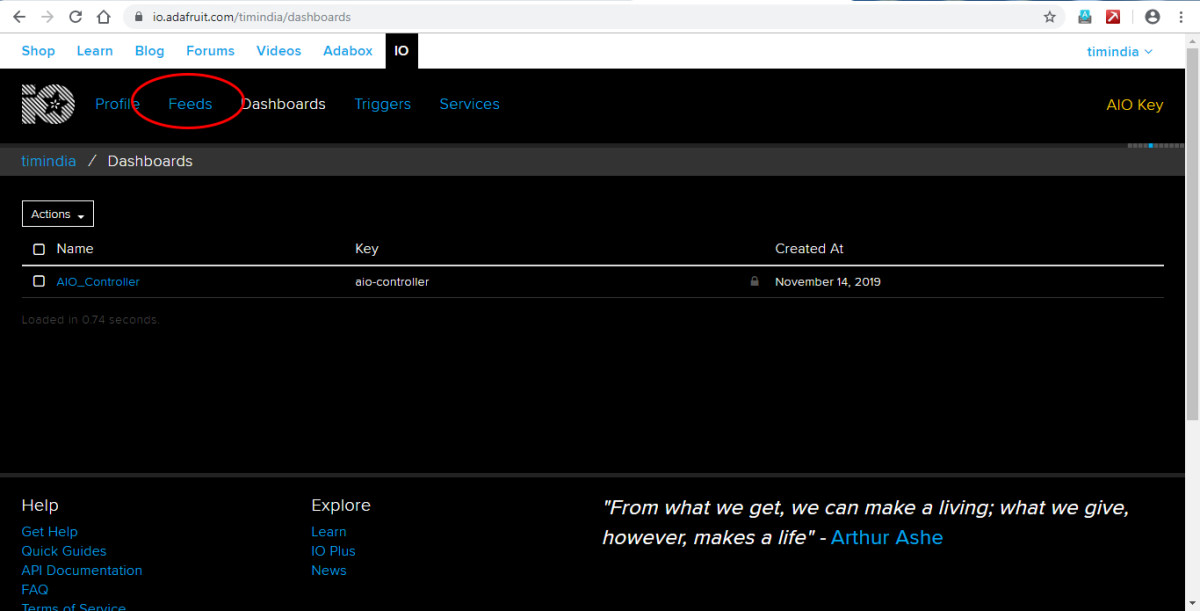
* Đầu tiên, ta phải tạo một tài khoản trên adafruit.io hoặc nếu bạn đã có tài khoản chỉ cần đăng nhập vào đó.

Graphical user interface, application

Description automatically generated

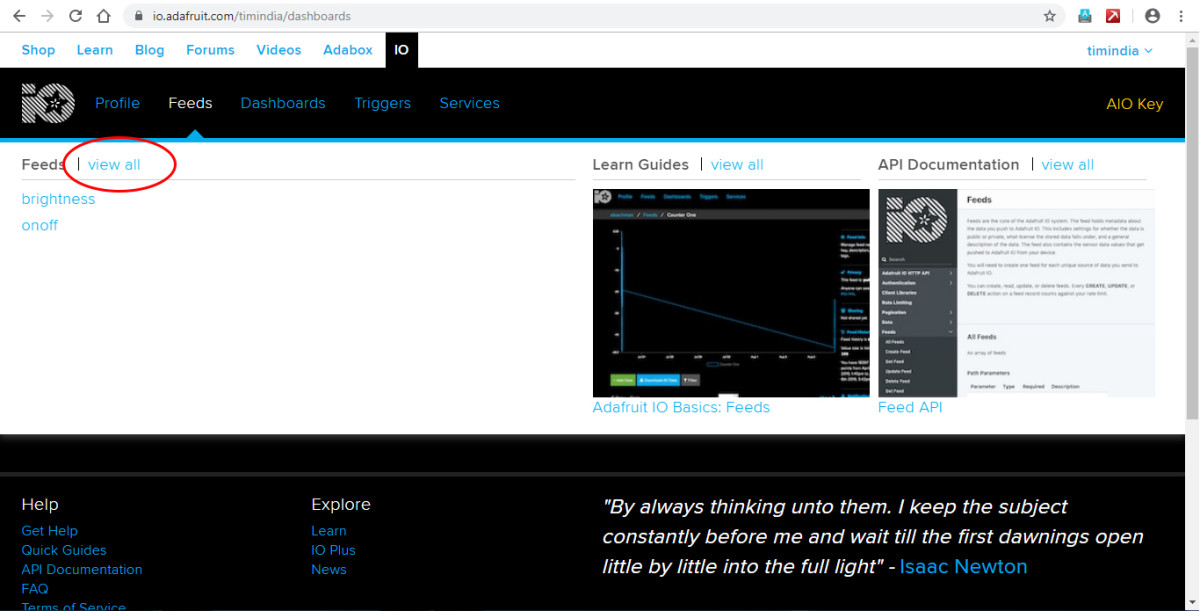
Hình Trang đăng nhập

* Sau khi đăng nhập, ta sẽ thấy trang **Dashboard AIO\_Controller** như hình. Tại trang Dashboard, ta cần tạo các nguồn dữ liệu (Feeds) bằng cách chọn **Feeds**.



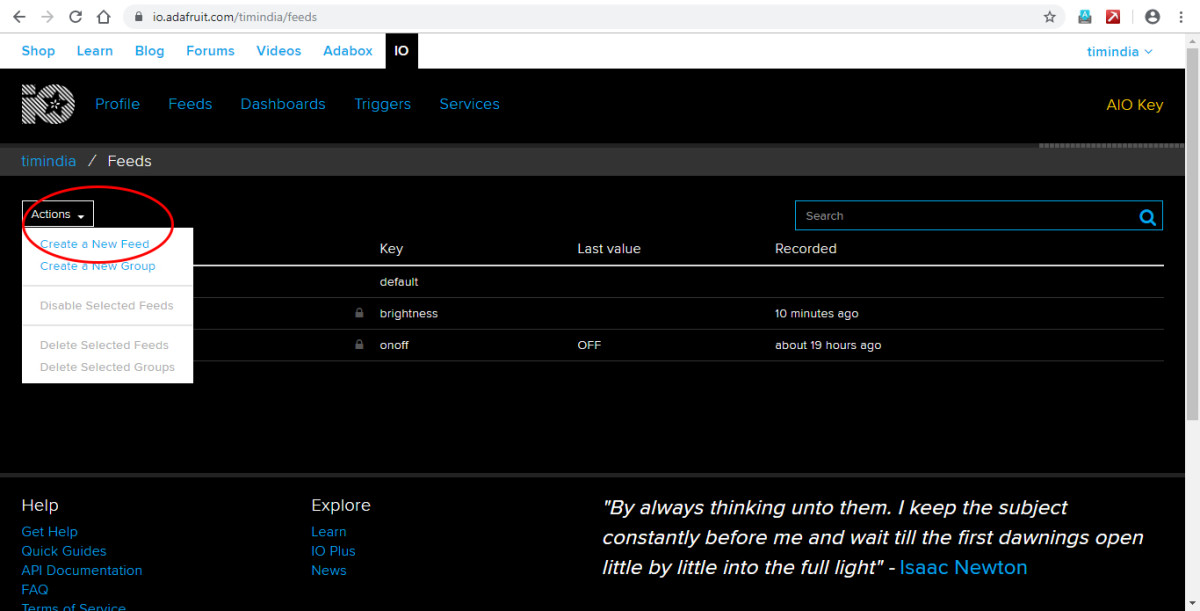
Hình Trang DashBoard

* Chọn **View All.**



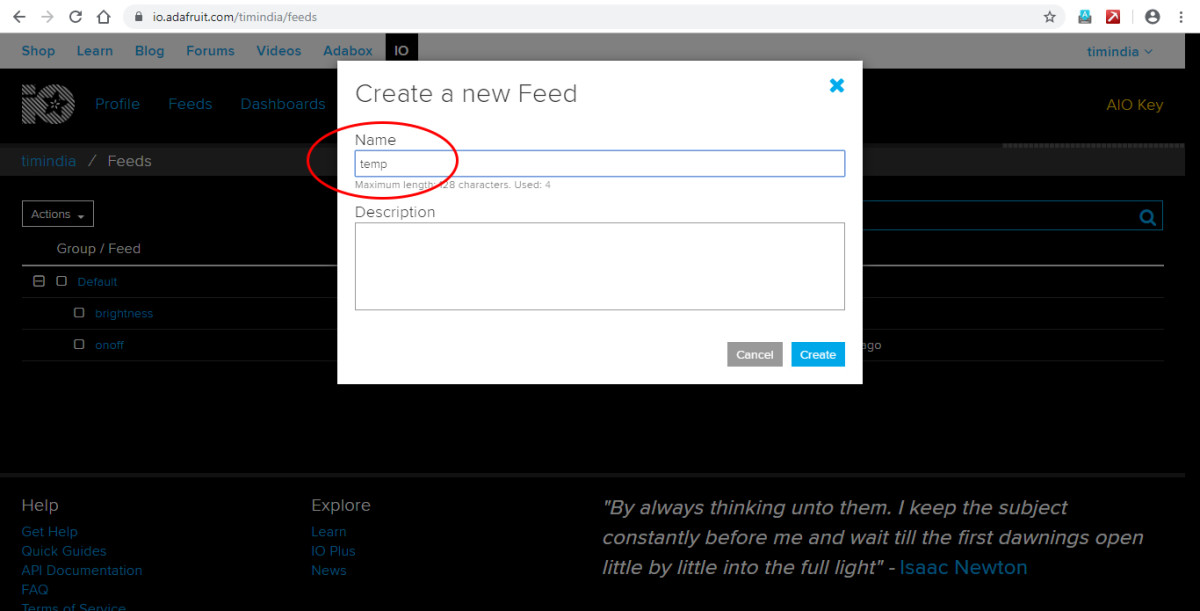
Hình Các nguồn dữ liệu

* Tiếp tục chọn **Actions -> Create a New Feed.**



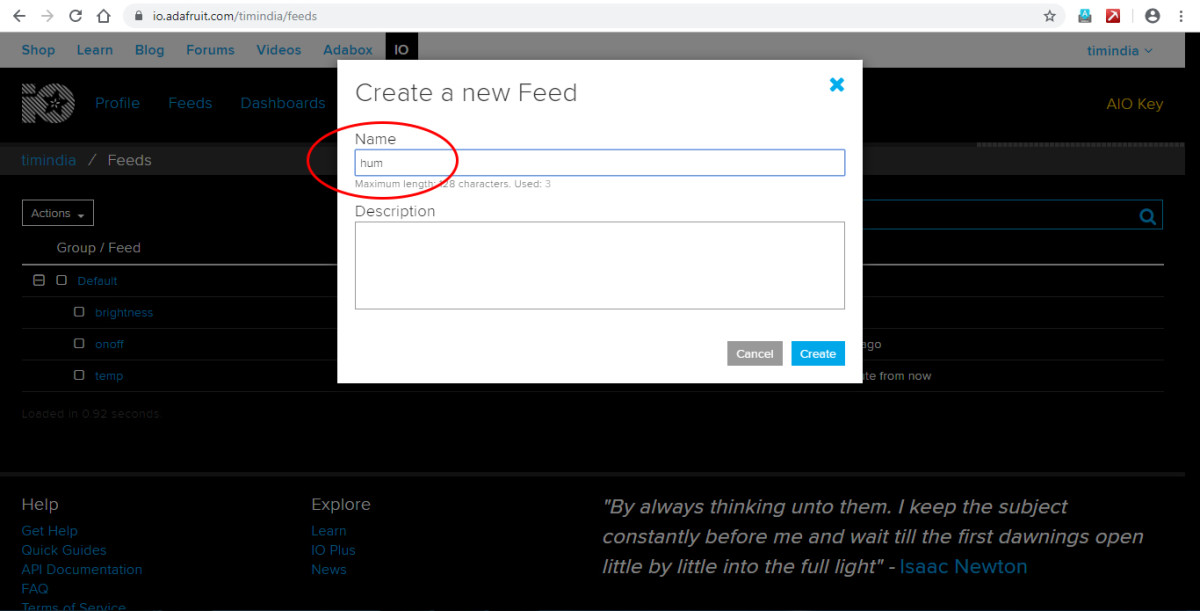
Hình Tạo nguồn dữ liệu mới

* Ta sẽ thấy màn hình *Create New Feed* hiện ra. Nhập tên nguồn dữ liệu là ‘temp’.



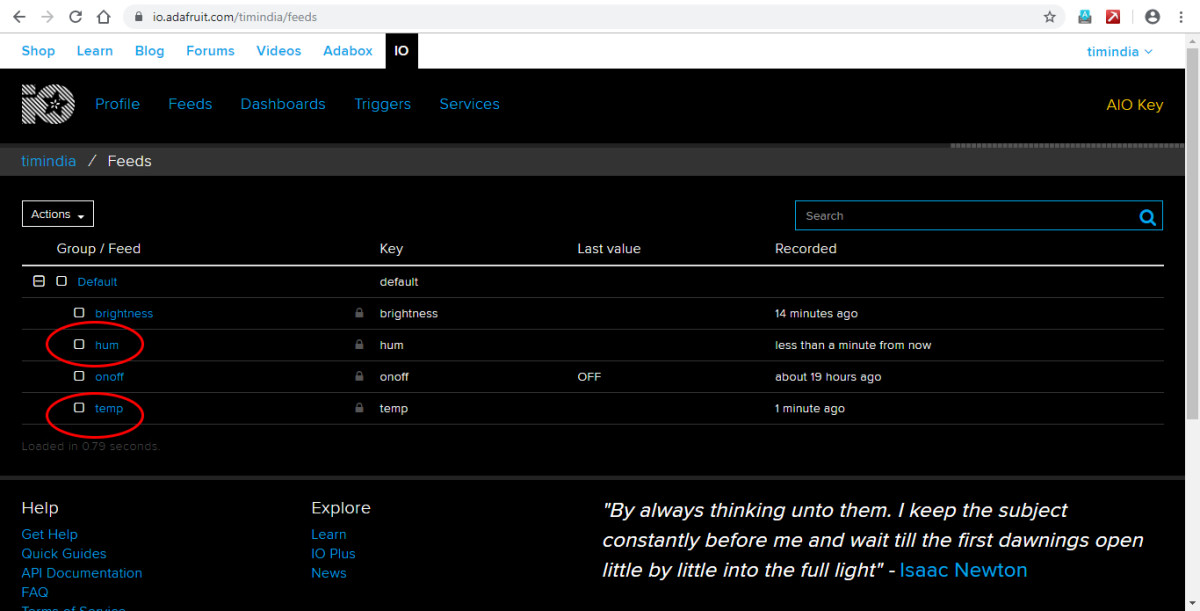
Hình Tạo nguồn dữ liệu nhiệt độ

* Làm tương tự với các nguồn dữ liệu khác.



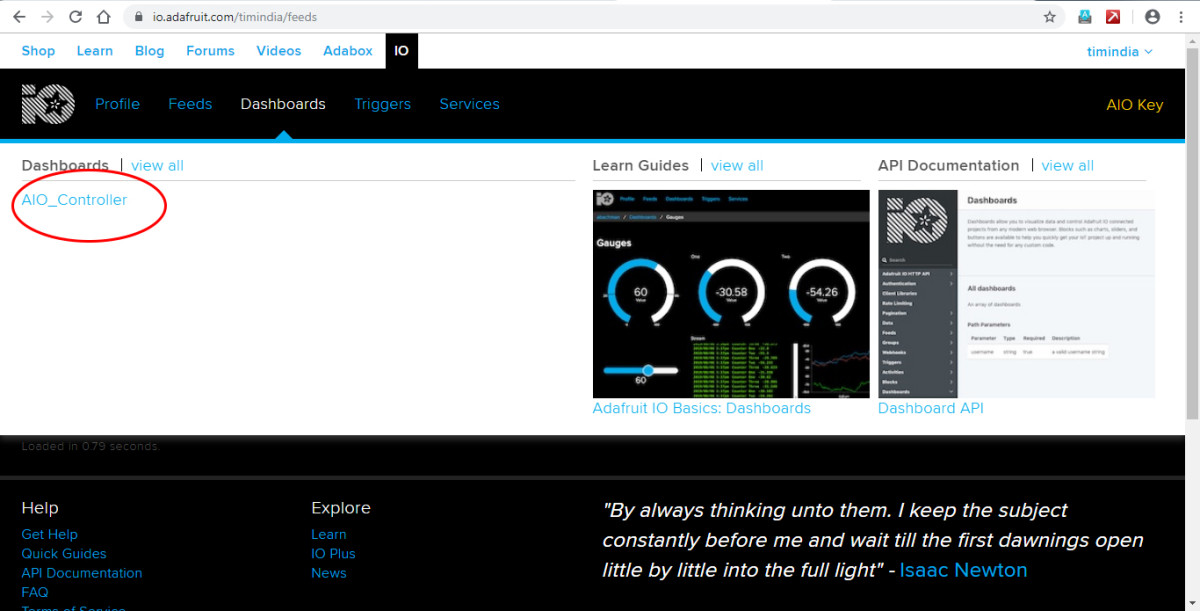
Hình Tạo nguồn dữ liệu độ ẩm

* Giờ ta đã có 2 nguồn dữ liệu ‘temp’ và ‘hum’ để lưu trữ dữ liệu nhiệt độ và độ ẩm.



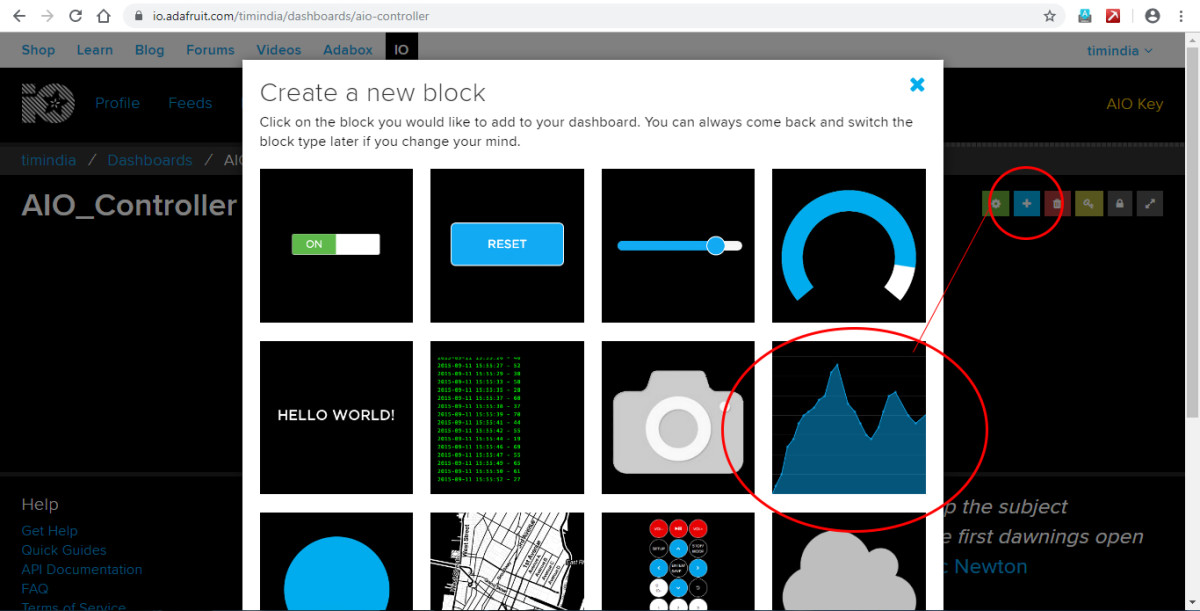
Hình Dánh sách các nguồn dữ liệu mới

* Trở về trang **Dashboard** và chọn **AIO\_Controller**



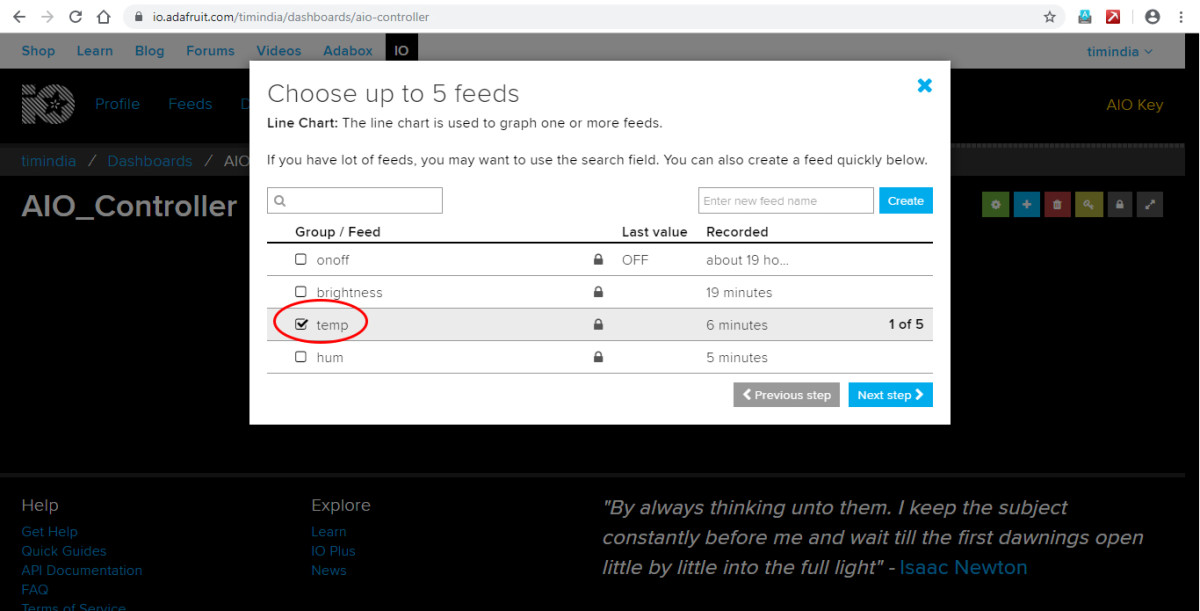
Hình AIO\_Controller

* Chọn nút dấu ‘+’ để tạo ra các khối giao diện (VD : biểu đồ ,…).



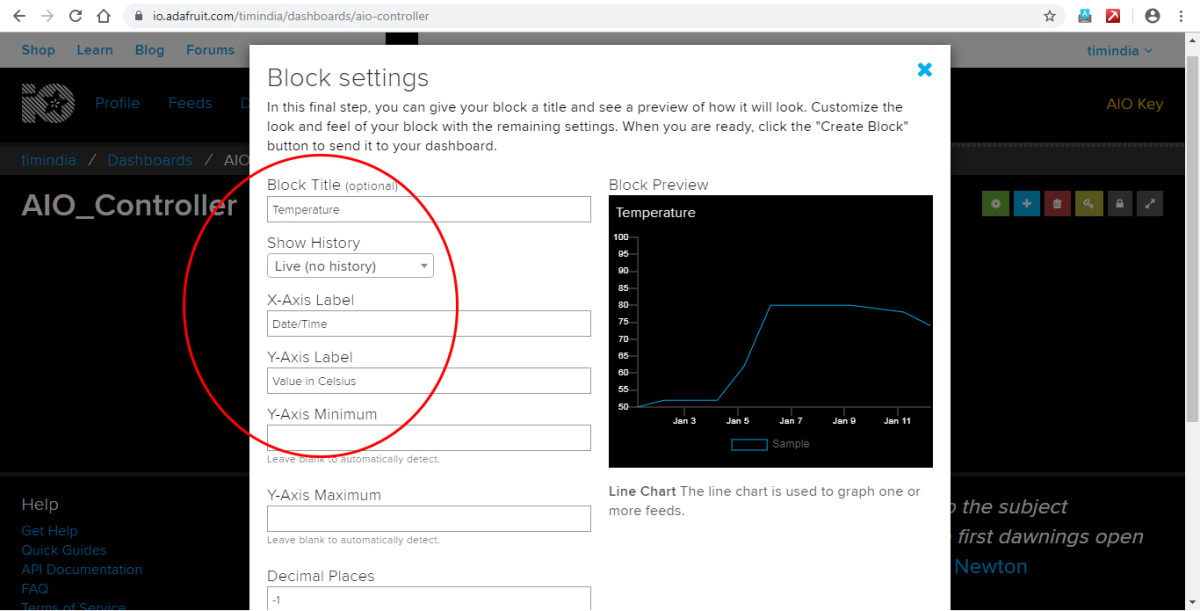
Hình Các khối giao diện

* Chọn giao diện Line Chart và chọn nguồn dữ liệu ‘temp’ trong danh sách các nguồn dữ liệu.



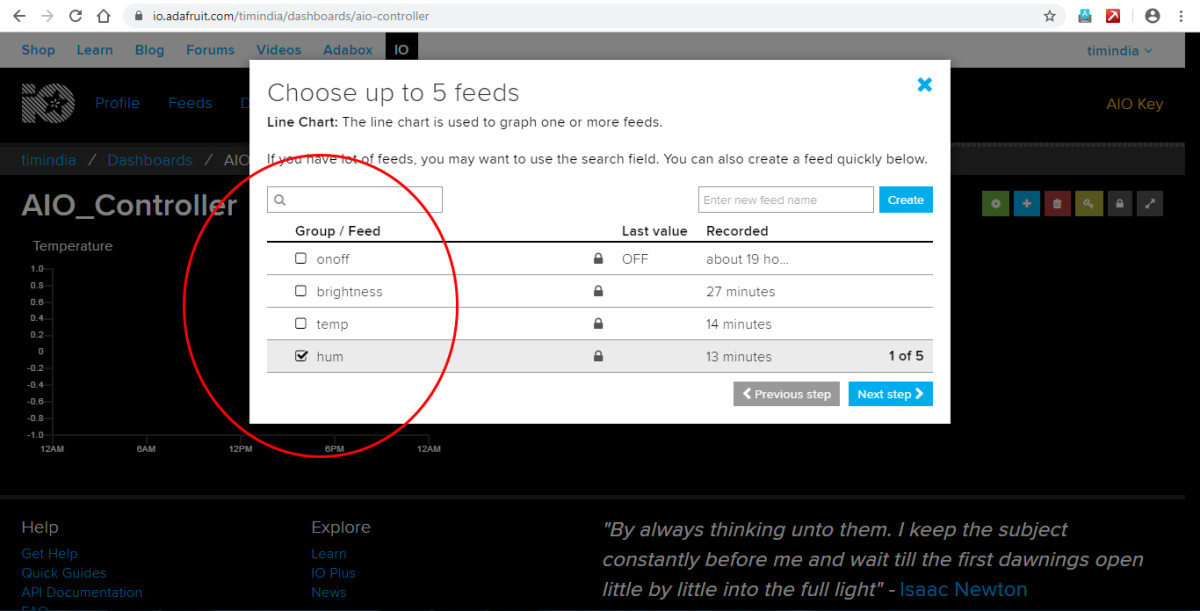
Hình Chọn giao diện cho nhiệt độ

* Tiếp tục thiết lập cho khối giao diện.



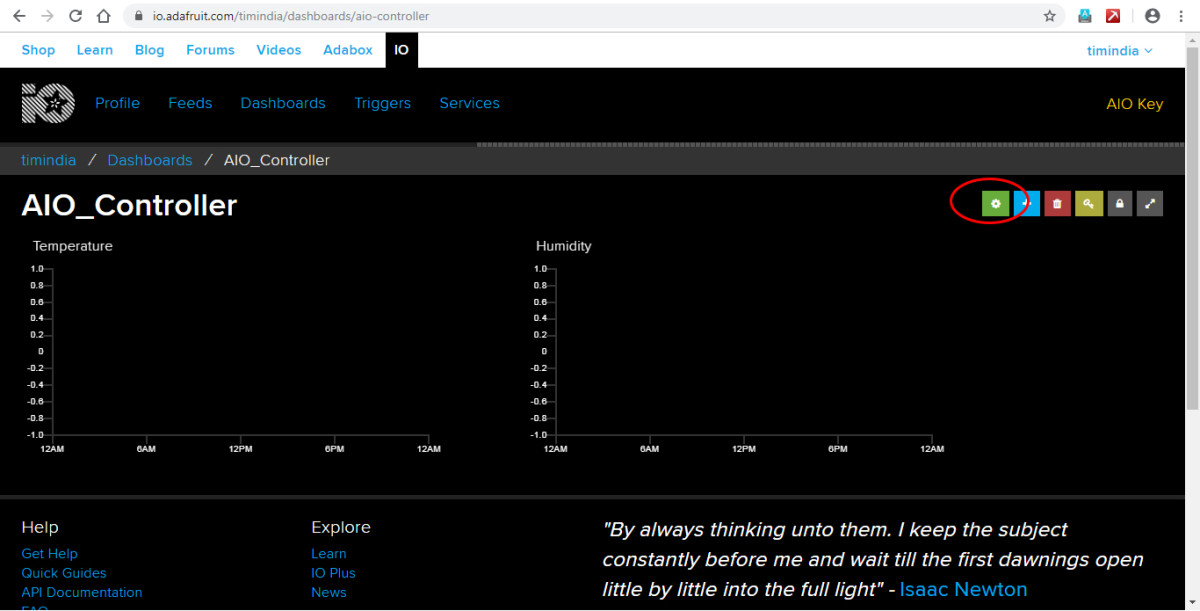
Hình Cài đặt khối giao diện

* Làm tương tự với nguồn dữ liệu độ ẩm.



Hình Chọn giao diện cho độ ẩm

* Sau đó ta được Dashboard chứa 2 khối đồ thị Line Chart như hình.



Hình Trang Dashboard

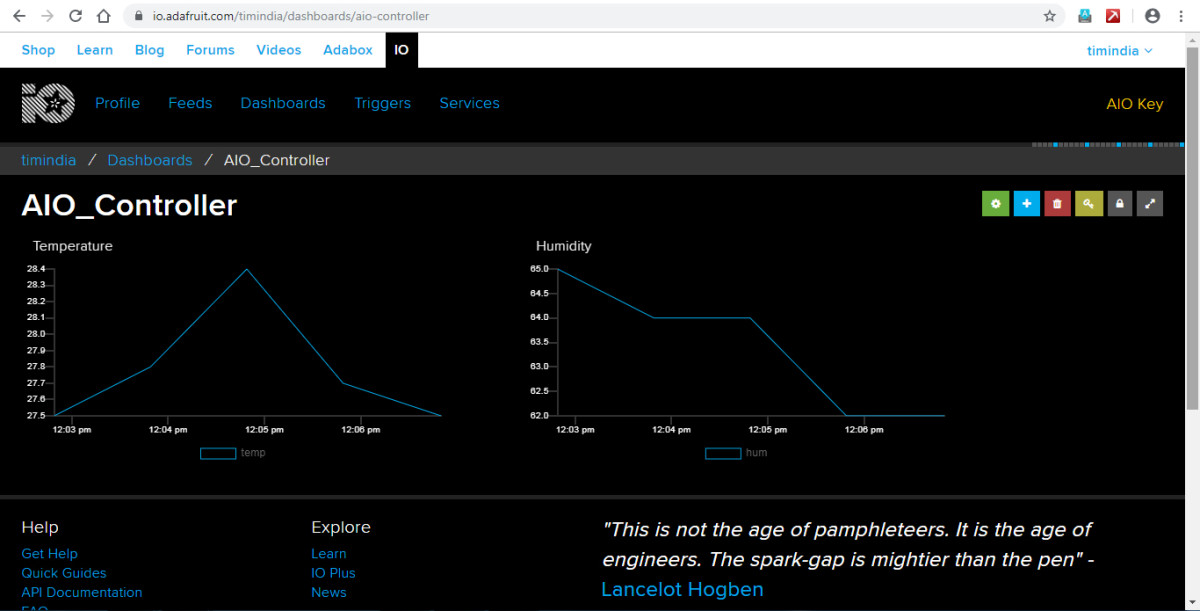
* Lấy mã AIO key

Graphical user interface, text, application, email

Description automatically generated

Hình Lấy mã AIO Key

* Kết nối tới các thiết bị IoT thông qua Username và AIO key để gửi các dữ liệu cho các nguồn “temp” và “hum” đến Adafruit IO.



Hình Hiển thị dữ liệu

### 3.2.3 Kết nối vi xử lý ESP32 với các module

#### 3.2.3. Kết nối với cảm biến nhiệt độ - độ ẩm DHT22

Đầu tiên cần cài đặt thư viện tuỳ chỉnh DHT.h mà nhóm lập trình:

#include “DHT.h”

Tiếp theo ta cần tạo một object DHT22 với hàm khởi tạo kèm theo vị trí chân mà module DHT22 kết nối với vi xử lý ESP32, cụ thể ở đây nhóm sử dụng chân D12 với số thứ tự trong sơ đồ chân ESP32 là 18 (Hình 2.2):

DHT22 dht(18);

Như vậy là kết nối giữa module DHT22 và vi xử lý ESP32 đã được thiết lập. Dữ liệu về nhiệt độ và độ ẩm được lấy ra bởi 2 phương thức getTemperature() và getHumidity() của object dht.

#### 3.2.3.2 Kết nối với cảm biến khí CO MQ7

Tương tự như cảm biến DHT22, đầu tiên ta cần cài đặt thư viện tuỳ chỉnh MQ7.h với câu lệnh:

#include “MQ7.h”

Kế tiếp, một object mq7 được khởi tạo với tham số truyền vào hàm khởi tạo chính là số thứ tự chân analog và digital mà module cảm biến MQ7 kết nối với vi xử lý. Ở đây, nhóm sử dụng chân D18 (35) và VP (5) tương ứng là chân digital và analog để kết nối với cảm biến MQ7.

MQ7 mq7(35, 5);

Sau khi cài đặt chế độ cho chân kết nối bằng phương thức begin(), ta có thể lấy giá trị nồng độ khí CO với phương thức getPPM() của lớp MQ7.

#### 3.2.3.3 Kết nối với cảm biến bụi PM2.5 GP2Y1010AU0F

Ban đầu, thư viện tuỳ chỉnh PM25.h được cài đặt:

#include “PM25.h”

Object pm25 cũng được khởi tạo với tham số truyền vào hàm khởi tạo là số thứ tự chân analog và chân đèn led trên vi xử lý ESP32.

PM25 pm25(34, 19);

Sau khi chạy phương thức begin() với mục đích cài đặt chế độ cho chân kết nối, giá trị của nồng độ bụi PM2.5 được đọc bởi phương thức getDustDensity() của lớp PM25.

Chi tiết về mã nguồn thư viện cũng như triển khai trong chương trình chính của module DHT22, MQ7, PM25 được ghi trong Phụ lục.

#### 3.2.3.4 Kết nối với LCD

Nhóm sử dụng thư viện mã nguồn mở LiquidCrystal\_I2C [28] trong Arduino IDE để kết nối với màn hình LCD. Địa chỉ I2C của LED được xác định là 0x27, từ đó ta có thể khai báo và thực thi điều khiển màn LCD qua các hàm print được cung cấp bởi thư viện LiquidCrystal\_I2C mà nhóm sử dụng.

Trước tiên, gọi thư viện LiquidCrystal\_I2C và thông báo hàm lcd sẽ dùng để điều khiển lcd:

LiquidCrystal\_I2C lcd(0x27,16,2);

Ở câu lệnh khai báo trên, nhóm sử dụng màn hình LCD 16x2 với địa chỉ 0x27.

Tiếp theo, phương thức init() và backlight() được sử dụng để khởi tạo và bật sáng màn hình:

lcd.init();

lcd.backlight();

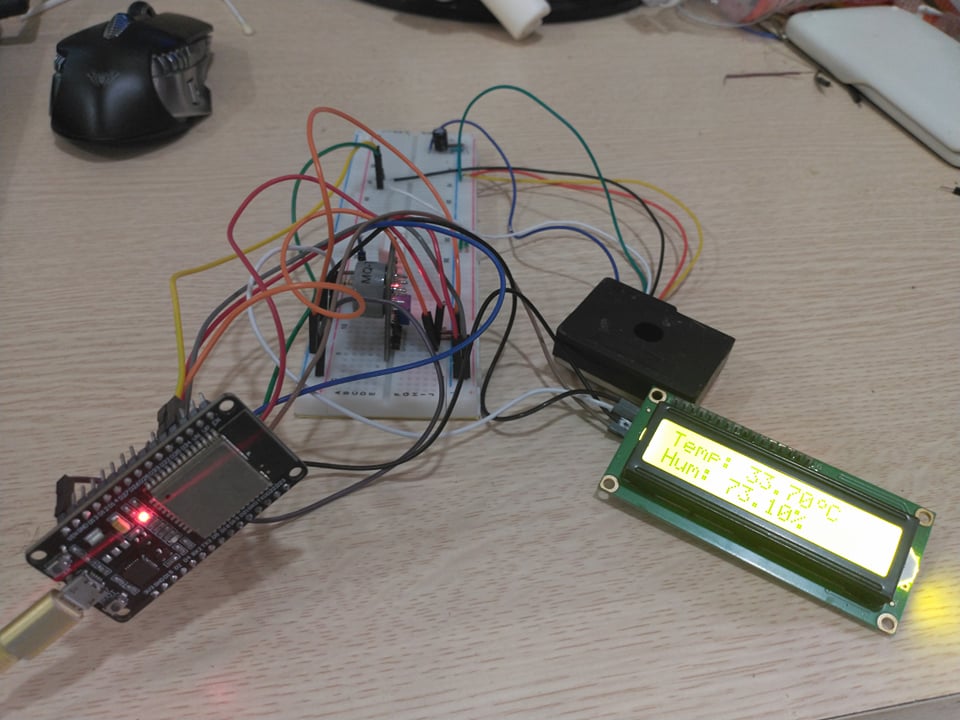
Từ đây ta chỉ cần dùng lệnh print() để in ra màn hình lcd một cách dễ dàng:

lcd.print(“myteam“);

# CHƯƠNG 4. KẾT QUẢ THỰC HIỆN

## 4.1 Kết quả mạch thực tế

Sau khi tiến hành mua linh kiện và lập trình trên vi xử lý, nhóm thực hiện hệ thống đo chất lượng không khí trên breadboard như hình 4.1.



Hình 4.1 Mạch thực tế của nhóm

## 4.2 Kiểm thử

Trong quá trình thực hiện nhóm đã thực hiện kiểm thử hệ thống theo các kịch bản sau:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **STT** | **Mục đích kiểm thử** | **Thủ tục kiểm thử** | **Kết quả mong muốn** | **Kết quả kiểm thử** |
| 1 | Kiểm thử hiển thị trên màn hình LCD | 1. Kết nối màn hình LCD với ESP32 2. Lập trình hiển thị các thông số nhiệt độ, độ ẩm lên màn hình | Màn hình LCD hiển thị các thông số nhiệt độ và độ ẩm | Đạt |
| 2 | Kiểm thử đầu ra của cảm biến DHT22 | 1. Kết nối cảm biến DHT22 với ESP32 2. Lập trình hiển thị các thông số nhiệt độ, độ ẩm lên màn hình LCD 3. Thay đổi nhiệt độ, độ ẩm xung quanh cảm biến bằng các nguồn nhiệt | Các thông số nhiệt độ và độ ẩm hiển thị trên màn hình thay đổi theo sự thay đổi của các nguồn nhiệt | Đạt |
| 3 | Kiểm thử chức năng hiển thị trên Web | 1. Đăng nhập vào mạng WiFi trên ESP32 2. Truy cập vào địa chỉ IP của trang Web | Các thông số hiển thị trên trang Web giống với các thông số trên màn hình LCD | Đạt |
| 4 | Kiểm thử chức năng đưa dữ liệu lên Adafruit | 1. Đăng nhập vào mạng WiFi trên ESP32 2. Truy cập vào Adafruit IO | Các thông số hiển thị trên Adafruit giống với các thông số trên màn hình LCD | Đạt |

Bảng 4.1 Bảng các bài kiểm thử hệ thống

## 4.3 Kết quả thực hiện hệ thống thực tế

Nhóm thực hiện chạy hệ thống khi chưa cài đặt thông tin Wi-Fi, lúc này hệ thống chuyển qua chế độ AP để người dùng có thể kết nối và cài đặt thông tin Wi-Fi có kết nối Internet một cách thủ công.

Graphical user interface, text, application

Description automatically generated

Hình 4.2 Mạch chưa có thông tin Wi-Fi

Graphical user interface, application

Description automatically generated

Hình 4.3 Mạch chuyển sang chế độ AP

Graphical user interface

Description automatically generated with medium confidence

Hình 4.4 Nhập thông tin Wi-Fi từ thiết bị cá nhân

Text

Description automatically generated with medium confidence

Hình 4.5 Hệ thống đã nhận thông tin Wi-Fi mới

Hệ thống đã có thể hiển thị thông tin về độ ẩm, nhiệt độ, nồng độ CO và bụi PM2.5 được thu thập từ các cảm biến. Kết quả cụ thể được hiển thị trong hình …

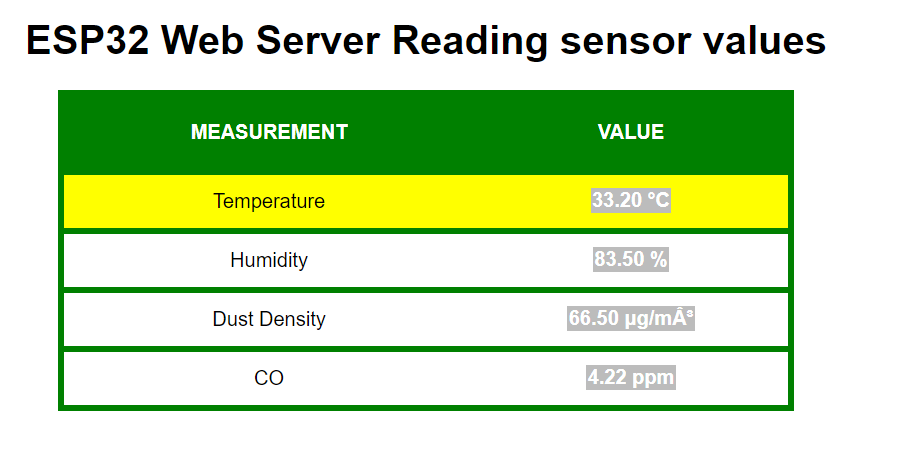


Hình 4.6 Thông tin nhiệt độ, độ ẩm hiển thị trên LCD

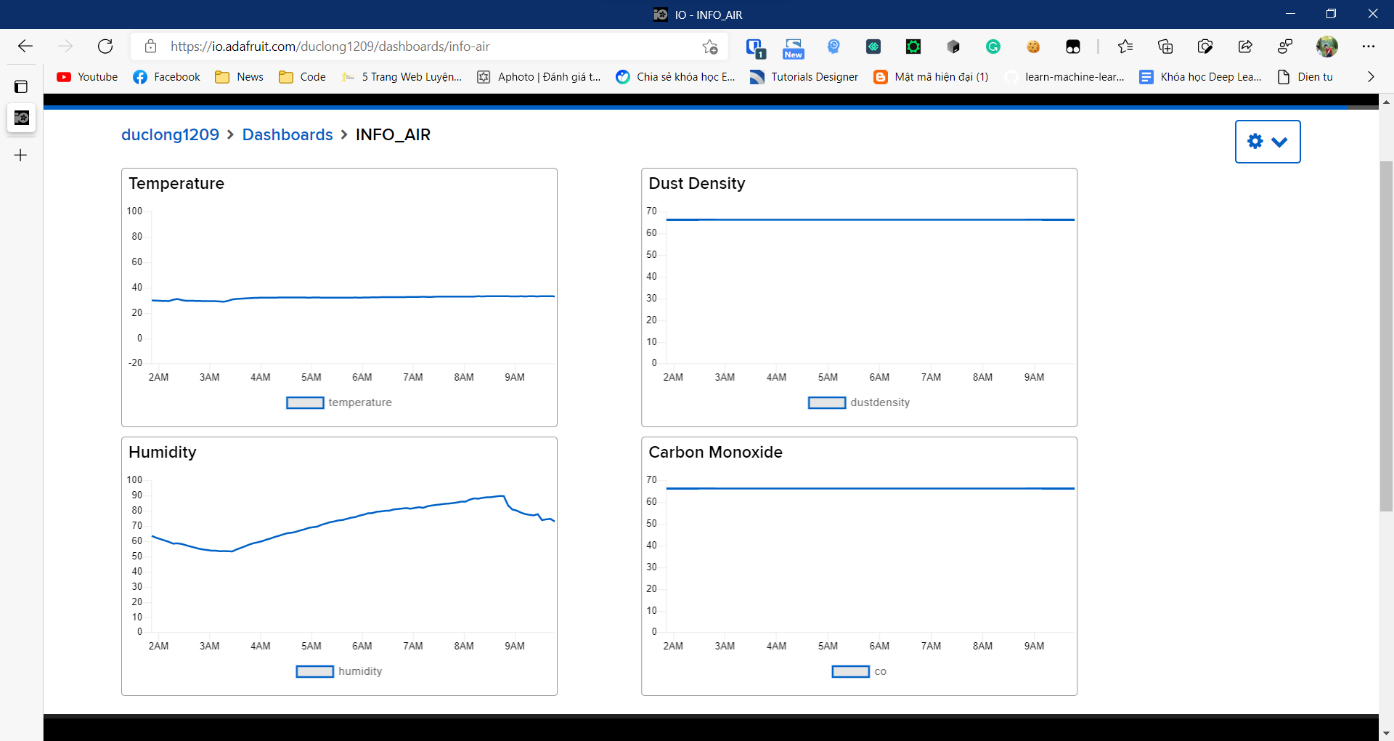


Hình 4.7 Thông tin về bụi PM2.5, khí CO trên LCD

Ngoài ra, kết quả thu thập từ cảm biến cũng được cập nhật lên webserver để hiển thị giá trị tức thời tại thời điểm đo như hình 4.8. Kèm theo đó, hình 4.9 đã cho thấy hệ thống có thể hiển thị dashboard với các đồ thị xây dựng từ các dữ liệu thu thập được theo thời gian trên hệ thống AdafruitIO.



Hình 4.8 Thông tin về không khí hiển thị trên website



Hình 4.9 Dashboard theo dõi chất lượng không khí

Cuối cùng, hình 4.10 thể hiện sự thay đổi nhiệt độ và độ ẩm đo được khi nhóm dùng máy sấy để thay đổi thông số môi trường đo. Từ kết quả có thể cho thấy rằng hệ thống đã đo được những số liệu phù hợp với môi trường.

A picture containing text, electronics

Description automatically generated

Hình 4.10 Giá trị nhiệt độ, độ ẩm tăng khi sử dụng máy sấy

# KẾT LUẬN

Với mục tiêu ban đầu mà nhóm đã đặt ra là xây xựng một mô hình giám sát chỉ số môi trường và hiển thị nó trên Websever, thông qua mạng dữ liệu để đưa các kết quả thu được từ các cảm biến đặt cố định trước. Khi đó người sử dụng hệ thống này để biết được các thông tin như là nhiệt độ, độ ẩm, bụi mịn và khí CO tại những điểm quan trọng đã được đặt các cảm biến từ trước. Từ mục tiêu như vậy thì nhóm đã cùng nhau tìm kiếm thông tin, cách thức và các kiến thức cần có để thiết kế được một sản phẩm thỏa mãn được yêu cầu mà nhóm đã đặt ra.

Để xây dựng được mô hình hệ thống này nhóm đã sử dụng ESP32, các cảm biến như DHT 22 , MQ7, cảm biến bụi PM2.5 , màn hình LCD, chuẩn giao tiếp I2C,…. Để giải quyết việc xây dựng hệ thống này thì nhóm tiến hành giao tiếp các cảm biến với vi điều khiển với chuẩn giao tiếp của từng con cảm biến. Sau khi nhận được dữ liệu đo được thì cần đẩy lên sever.

Hoạt động thực tế của hệ thống sẽ đo các chỉ số môi trường từ các cảm biến. Sau khi đo xong thì kết quả được trả về hiển thị trên màn hình LCD đồng thời cũng gửi tín hiệu lên sever thông qua wifi với mục đích là hiển thị thông tin tại các thời điểm thống kê trong ngày với các khoảng thời gian cách nhau 10 phút. Tuy vậy hệ thống hoạt động vẫn chưa đạt được độ ổn định cao, vẫn còn sai số so với thực tế và quá trình truyền tải lên sever chưa thực sự mượt mà.

Qua bài tập lớn này nhóm cũng đã học tập được rất nhiều kiến thức về chuẩn giao tiếp của các con cảm biến với bộ vi xử lý ESP32, nắm rõ thêm một số các thuật toán và lập trình và nhận thấy được phần quan trọng của vi xử lý đối với cuộc sống hiện tại. Nhóm xin được cảm ơn thầy Hàn Huy Dũng đã giúp đỡ giảng dạy cho nhóm về kiến thức về bộ môn Vi Xử lý để nhóm có thể dễ nắm bắt và hoản thiện được bài tập lớn này

# TÀI LIỆU THAM KHẢO

[1] “Hạn chế ô nhiễm không khí tại các đô thị.” https://nhandan.vn/khoa-hoc-thien-nhien/han-che-o-nhiem-khong-khi-tai-cac-do-thi-631210/.

[2] “Ô nhiễm không khí tại một số đô thị lớn có xu hướng gia tăng.” http://thainguyen.gov.vn/cong-nghe/-/asset\_publisher/L0n17VJXU23O/content/o-nhiem-khong-khi-tai-mot-so-o-thi-lon-co-xu-huong-gia-tang.

[3] “Ô nhiễm không khí ở Hà Nội: Giải pháp không phải đơn giản.” https://www.rfi.fr/vi/việt-nam/20200106-ô-nhiễm-không-khí-ở-hà-nội-giải-pháp-không-phải-đơn-giản.

[4] “Ô nhiễm không khí ở Việt Nam.” https://www.who.int/vietnam/vi/health-topics/air-pollution.

[5] “ESP32.” https://en.wikipedia.org/wiki/ESP32.

[6] “ESP32 Espressif.” https://www.espressif.com/en/products/socs/esp32.

[7] “I2C.” https://en.wikipedia.org/wiki/I2C.

[8] “Giới thiệu chuẩn giao tiếp I2C.” http://dammedientu.vn/gioi-thieu-chuan-giao-tiep-i2c/.

[9] “LED text LED1602 xanh lá,” [Online]. Available: https://hshop.vn/products/lcd-text-lcd1602-xanh-lo.

[10] “Mạch chuyển đổi I2C cho LCD.” https://iotmaker.vn/mach-chuyen-doi-i2c-cho-lcd.html.

[11] “How to Use I2C LCD with ESP32 on Arduino IDE (ESP8266 compatible).” https://randomnerdtutorials.com/esp32-esp8266-i2c-lcd-arduino-ide/.

[12] “Cảm biến độ ẩm và nhiệt độ DHT22.” https://hshop.vn/products/cam-bien-do-am-nhiet-do-dht22.

[13] “Cảm biến khí CO MQ7.” https://nshopvn.com/product/cam-bien-khi-co-mq-7/.

[14] Hanwei Electronics, “MQ-7 Gas Sensor Datasheet,” vol. 1, pp. 3–5, 2016.

[15] “How to use the MQ-7 Carbon Monoxide Sensor.” https://www.teachmemicro.com/use-mq-7-carbon-monoxide-sensor/.

[16] “MQ7-Library.” https://github.com/swatish17/MQ7-Library.

[17] “Cảm biến bụi Optical Dust Sensor PM2.5 GP2Y1010AU0F.” https://hshop.vn/products/cam-bien-bui-gp2.

[18] “GP2Y1014AU0F – Optical Dust Sensor Working Applications Datasheet.” https://microcontrollerslab.com/gp2y1014au0f-dust-sensor-pinout-working-applications-datasheet/.

[19] V. Tasić, M. Jovašević - Stojanović, D. Topalović, and M. Davidović, “18 - Measurement of PM2.5 Concentrations in Indoor Air Using Low-Cost Sensors and Arduino Platforms,” pp. 69–72, 2020, doi: 10.5162/6eunetair2016/18.

[20] T. Liu, “Digital-Output relative humidity & temperature sensor/module DHT22,” *New York Aosong Electron.*, vol. 22, pp. 1–10, 2015, [Online]. Available: https://www.sparkfun.com/datasheets/Sensors/Temperature/DHT22.pdf.

[21] I. interface for L. Datasheet, “I2C Interface for LCD,” p. 2, 2017, [Online]. Available: © Copyright 2017 , Mantech Electronics (Pty) Ltd. All rights reserved. A.

[22] SHARP, “GP2Y1010AU0F Compact Optical Dust Sensor,” *Datasheet*, pp. 1–11, 2017, [Online]. Available: http://www.sharpsme.com/download/gp2y1010au-epdf.

[23] “Arduino Homepage.” https://www.arduino.cc/.

[24] “Arduino.” https://en.wikipedia.org/wiki/Arduino.

[25] “MQTT.” https://en.wikipedia.org/wiki/MQTT.

[26] “MQTT: The Standard for IoT Messaging.” https://mqtt.org/.

[27] “Adafruit IO.” https://io.adafruit.com/.

[28] “LiquidCrystal I2C.” https://www.arduinolibraries.info/libraries/liquid-crystal-i2-c.

# PHỤ LỤC

## A. Mã nguồn hệ thống

Link github mã nguồn: [*https://github.com/longbaby5512/Microprocessor\_Group5*](https://github.com/longbaby5512/Microprocessor_Group5)

|  |
| --- |
| // File main.cpp  #include "DHT22.h"  #include "LiquidCrystal\_I2C.h"  #include "MQ7.h"  #include "PM25.h"  #include <Adafruit\_MQTT.h>  #include <Adafruit\_MQTT\_Client.h>  #include <EEPROM.h>  #include <String>  #include <WebServer.h>  #include <WiFi.h>  #define AIO\_SERVER "io.adafruit.com"  #define AIO\_SERVERPORT 1883  #define AIO\_USERNAME "duclong1209"  #define AIO\_KEY "4127c18370b6431cb863c82b0c075c3d"  const char char\_o[] = {6, 9, 9, 6, 0, 0, 0, 0};  const char char\_3[] = {6, 1, 2, 1, 6, 0, 0, 0};  const char INDEX\_HTML[] =  "<!DOCTYPE HTML>"  "<html>"  "<head>"  "<meta content=\"text/html; charset=ISO-8859-1\""  " http-equiv=\"content-type\">"  "<meta name = \"viewport\" content = \"width = device-width, initial-scale = 1.0, maximum-scale = 1.0, user-scalable=0\">"  "<title>ESP8266 Web Form Demo</title>"  "<style>"  "\"body { background-color: #808080; font-family: Arial, Helvetica, Sans-Serif; Color: #000000; text-align:center;}\""  "</style>"  "</head>"  "<body>"  "<h3>Enter your WiFi credentials</h3>"  "<form action=\"/\" method=\"post\">"  "<p>"  "<label>SSID:&nbsp;</label>"  "<input maxlength=\"30\" name=\"ssid\"><br>"  "<label>Key:&nbsp;&nbsp;&nbsp;&nbsp;</label><input maxlength=\"30\" name=\"password\"><br>"  "<input type=\"submit\" value=\"Save\">"  "</p>"  "</form>"  "</body>"  "</html>";  /\*\*\*\*\*\*\*\*\* Variables \*\*\*\*\*\*\*\*\*/  DHT22 dht(18);  PM25 pm25(34, 19);  MQ7 mq7(35, 5);  LiquidCrystal\_I2C lcd(0x27, 16, 2);  WiFiServer wifi(80);  WebServer server(80);  WiFiClient client;  Adafruit\_MQTT\_Client mqtt(&client, AIO\_SERVER, AIO\_SERVERPORT, AIO\_USERNAME, AIO\_KEY);  Adafruit\_MQTT\_Publish temperatureAdafruit = Adafruit\_MQTT\_Publish(&mqtt, AIO\_USERNAME "/feeds/air-info.temperature");  Adafruit\_MQTT\_Publish humidityAdafruit = Adafruit\_MQTT\_Publish(&mqtt, AIO\_USERNAME "/feeds/air-info.humidity");  Adafruit\_MQTT\_Publish dustDensityAdafruit = Adafruit\_MQTT\_Publish(&mqtt, AIO\_USERNAME "/feeds/air-info.dustdensity");  Adafruit\_MQTT\_Publish coAdafruit = Adafruit\_MQTT\_Publish(&mqtt, AIO\_USERNAME "/feeds/air-info.co");  float temperature = 0;  float humidity = 0;  float dustDensity = 0;  float co = 0;  /\*\*\*\*\*\*\*\*\* Function \*\*\*\*\*\*\*\*\*/  void readSensor(void \*);  void printClient(WiFiServer, float, float, float, float, int);  String sendHTML(float, float, float, int);  void MQTT\_connect();  void mqttTask(void \*);  void webServerTask(void \*);  void loadWiFi();  void setup()  {  Serial.begin(9600);  pm25.begin();  mq7.begin();  EEPROM.begin(400);  lcd.init();  lcd.backlight();  lcd.setBacklight(HIGH);  lcd.setCursor(0, 0);  lcd.print("Wellcome to");  lcd.setCursor(0, 1);  lcd.print("my team");  delay(1000);  lcd.clear();  lcd.setCursor(0, 0);  lcd.createChar(1, char\_o);  lcd.createChar(2, char\_3);  loadWiFi();  xTaskCreate(&readSensor, "Read sensor", 2048, NULL, 3, NULL);  xTaskCreate(&mqttTask, "Adafruit MQTT", 2048, NULL, 2, NULL);  xTaskCreate(&webServerTask, "Web Server", 2048, NULL, 2, NULL);  }  void loop()  {  }  void mqttTask(void \*pv)  {  for (;;)  {  MQTT\_connect();  if (!temperatureAdafruit.publish(temperature))  Serial.println(F("Temperature: Failed"));  else  Serial.println(F("Temperature: OK!"));  if (!humidityAdafruit.publish(humidity))  Serial.println(F("Humidity: Failed"));  else  Serial.println(F("Humidity: OK!"));  if (!dustDensityAdafruit.publish(dustDensity))  Serial.println(F("Dust Density: Failed"));  else  Serial.println(F("Dust Density: OK!"));  if (!coAdafruit.publish(dustDensity))  Serial.println(F("CO: Failed"));  else  Serial.println(F("CO: OK!"));  delay(5 \* 60 \* 1000);  }  }  void webServerTask(void \*pv)  {  for (;;)  {  printClient(wifi, temperature, humidity, dustDensity, co);  delay(5000);  }  }  /\*\*  \* @brief Read data from sensors  \*/  void readSensor(void \*pv)  {  for (;;)  {  temperature = dht.getTemperature(); // Gets the values of the temperature  humidity = dht.getHumidity(); // Gets the values of the humidity  dustDensity = pm25.getDustDensity(); // Gets the values of the dust density  co = mq7.getPPM(); // Gets the values of the CO  Serial.print("Temperature: ");  Serial.println(temperature);  Serial.print("Humidity: ");  Serial.println(humidity);  Serial.print("Dust Density: ");  Serial.println(dustDensity);  Serial.print("CO: ");  Serial.println(co);  lcd.clear();  lcd.setCursor(0, 0);  lcd.print("Temp: ");  lcd.print(temperature);  lcd.write(1);  lcd.print("C");  lcd.setCursor(0, 1);  lcd.print("Hum: ");  lcd.print(humidity);  lcd.print("%");  delay(2000);  lcd.clear();  lcd.setCursor(0, 0);  lcd.print("PM25: ");  lcd.print(dustDensity);  lcd.print("ug/m");  lcd.write(2);  lcd.setCursor(0, 1);  lcd.print("CO: ");  lcd.print(co);  lcd.print("ppm");  delay(2000);  }  }  /\*\*  \* @brief HTML with data had  \*/  String sendHTML(float temperature, float humidity, float dustDensity, float co)  {  String str = "<!DOCTYPE html><html>\n";  str += "<head><meta name=\"viewport\" content=\"width=device-width, initial-scale=1\">\n";  str += "<link rel=\"icon\" href=\"data:,\">\n";  str += "<style>body { text-align: center; font-family: \"Arial\", Arial;}\n";  str += "Bảng { border-collapse: collapse; width:40%; margin-left:auto; margin-right:auto;border-spacing: 2px;background-color: white;border: 4px solid green; }\n";  str += "th { padding: 20px; background-color: #008000; color: white; }\n";  str += "tr { border: 5px solid green; padding: 2px; }\n";  str += "tr:hover { background-color:yellow; }\n";  str += "td { border:4px; padding: 12px; }\n";  str += ".sensor { color:white; font-weight: bold; background-color: #bcbcbc; padding: 1px; }\n";  str += "</style></head><body><h1>ESP32 Web Server Reading sensor values</h1>\n";  str += "<Bảng><tr><th>MEASUREMENT</th><th>VALUE</th></tr>\n";  str += "<tr><td>Temperature</td><td><span class=\"sensor\">\n";  str += temperature;  str += " &deg;C</span></td></tr>\n";  str += "<tr><td>Humidity</td><td><span class=\"sensor\">\n";  str += humidity;  str += " %</span></td></tr>\n";  str += "<tr><td>Dust Density</td><td><span class=\"sensor\">\n";  str += dustDensity;  str += " &mu;g/m³</span></td></tr>\n";  str += "<tr><td>CO</td><td><span class=\"sensor\">\n";  str += co;  str += " ppm</span></td></tr>\n";  str += "</body>\n";  str += "<script>\n";  str += "setInterval(loadDoc,200);\n";  str += "function loadDoc() {\n";  str += "var xhttp = new XMLHttpRequest();\n";  str += "xhttp.onreadystatechange = function() {\n";  str += "if (this.readyState == 4 && this.status == 200) {\n";  str += "document.getElementById(\"webpage\").innerHTML =this.responseText}\n";  str += "};\n";  str += "xhttp.open(\"GET\", \"/\", true);\n";  str += "xhttp.send();\n";  str += "}\n";  str += "</script></html>";  return str;  }  void printClient(WiFiServer &sever, float temperature, float humidity, float dustDensity, float co)  {  WiFiClient client = sever.available();  if (client)  {  Serial.println("Web Client connected ");  String request = client.readStringUntil('\r');  client.println("HTTP/1.1 200 OK");  client.println("Content-type:text/html");  client.println("Connection: close");  client.println();  client.print(sendHTML(temperature, humidity, dustDensity, co));  client.println();  client.stop();  Serial.println("");  }  }  /\*\*  \* @brief Check MQTT is connected  \*/  void MQTT\_connect()  {  int8\_t ret;  //Stop if already connected  if (mqtt.connected())  return;  Serial.println("Connecting to MQTT...");  uint8\_t retries = 3;  while ((ret = mqtt.connect()) != 0)  {  Serial.println(mqtt.connectErrorString(ret));  Serial.println("Retrying MQTT connection in 5 seconds...");  mqtt.disconnect();  delay(5000);  retries--;  if (retries == 0)  {  while (1)  ;  }  }  Serial.println("MQTT connected");  }  /\*\*  \* @brief Connect to WiFi  \*/  void loadWiFi()  {  if (!checkWiFiCreds())  {  Serial.println("No WiFi credentials stored in memory. Loading form...");  while (loadWiFiCredsForm())  ;  }  }  /\*\*  \* @brief Function if don't generate web\_server  \*/  void handleNotFound()  {  String message = "File Not Found\n\n";  message += "URI: ";  message += server.uri();  message += "\nMethod: ";  message += (server.method() == HTTP\_GET) ? "GET" : "POST";  message += "\nArguments: ";  message += server.args();  message += "\n";  for (uint8\_t i = 0; i < server.args(); i++)  message += " " + server.argName(i) + ": " + server.arg(i) + "\n";  server.send(404, "text/plain", message);  }  /\*\*  \* @brief Function for writing WiFi creds to EEPROM  \* @param ssid ssid of WiFi to write to EEPROM  \* @param pass password of WiFi to write to EEPROM  \* @return true if save successful, else false  \*/  bool writeToMemory(String ssid, String pass)  {  char buff1[30];  char buff2[30];  ssid.toCharArray(buff1, 30);  pass.toCharArray(buff2, 30);  EEPROM.writeString(100, buff1);  EEPROM.writeString(200, buff2);  delay(100);  String s = EEPROM.readString(100);  String p = EEPROM.readString(200);  if (ssid == s && pass == p)  return true;  return false;  }  /\*\*  \* @brief Function for handling form  \*/  void handleSubmit()  {  String response\_success = "<h1>Success</h1>";  response\_success += "<h2>Device will restart in 3 seconds</h2>";  String response\_error = "<h1>Error</h1>";  response\_error += "<h2><a href='/'>Go back</a> to try again";  if (writeToMemory(String(server.arg("ssid")), String(server.arg("password"))))  {  server.send(200, "text/html", response\_success);  EEPROM.commit();  delay(3000);  ESP.restart();  }  else  server.send(200, "text/html", response\_error);  }  /\*\*  \* @brief Function for home page  \*/  void handleRoot()  {  if (server.hasArg("ssid") && server.hasArg("password"))  handleSubmit();  else  server.send(200, "text/html", INDEX\_HTML);  }  /\*\*  \* @brief Function for loading form  \* @return true if hagit ve WiFi creds in EEPROM, else false  \*/  bool loadWiFiCredsForm()  {  String s = EEPROM.readString(100);  String p = EEPROM.readString(200);  const char \*ssid = "ESP32 WiFi Manager";  Serial.println("Setting Access Point...");  WiFi.softAP(ssid);  IPAddress IP = WiFi.softAPIP();  Serial.print("AP IP address: ");  Serial.println(IP);  server.on("/", handleRoot);  server.onNotFound(handleNotFound);  server.begin();  Serial.println("HTTP server started");  while (s.length() <= 0 && p.length() <= 0)  {  server.handleClient();  delay(100);  }  return false;  }  /\*\*  \* @brief Function checking WiFi creds in memory  \* @return true if not empty, else if empty  \*/  bool checkWiFiCreds()  {  Serial.println("Checking WiFi credentials");  String s = EEPROM.readString(100);  String p = EEPROM.readString(200);  delay(5000);  if (s.length() <= 0 || p.length() <= 0)  return false;  Serial.print("Found credentials: ");  Serial.print(s);  Serial.print("/");  Serial.println(p);  delay(2000);  Serial.print("Connecting to WiFi Network ");  Serial.print(s);  WiFi.begin(s.c\_str(), p.c\_str());  int startTimeConnect = millis();  while (WiFi.status() != WL\_CONNECTED)  {  delay(500);  Serial.print(".");  if (millis() - startTimeConnect > 60 \* 1000)  {  Serial.println("False");  return false;  }  }  Serial.println("");  Serial.println("Successfully connected to WiFi.");  Serial.println("IP address of ESP32 is ");  Serial.println(WiFi.localIP());  Serial.println("Server started");  wifi.begin();  return true;  }  /\*\*  \* @brief Wipe EEPROM  \*/  void wipeEEPROM()  {  for (int i = 0; i < 400; i++)  {  EEPROM.writeByte(i, 0);  }  EEPROM.commit();  } |

|  |
| --- |
| /\*\*  \* @file DHT22.h  \* @author Group 5  \* @brief Library for temperature and humidity values by using DHT22 sensor  \* Datasheet: @see https://www.sparkfun.com/datasheets/Sensors/Temperature/DHT22.pdf  \* @version 1.0  \* @date 2021-06-08  \*  \* @copyright Copyright (c) 2021  \*  \*/  #ifndef DHT22\_H  #define DHT22\_H  #include <Arduino.h>  /\*\*  \* Class for DHT22 sensor.  \* Algorithm: https://www.teachmemicro.com/how-dht22-sensor-works/  \*/  class DHT22  {  public:  // Value returned when check errors during DHT22 processing  enum DHT\_ERROR\_t  {  ERROR\_NONE = 0, // No error  ERROR\_TIMEOUT, // Timeout error  ERROR\_CHECKSUM // Checksum errors  };  DHT22(int); // Construct a new DHT22::DHT22 object  void setup(int); // Setup pin connected microcontroller  void resetTimer(); // Reset timer for setup sensor  DHT\_ERROR\_t getStatus(); // Check the sensor is working properly  float getTemperature(); // Get the temperature measured by the sensor  float getHumidity(); // Get the humidity measured by the sensor  private:  DHT\_ERROR\_t error; // Error code  unsigned long lastReadTime; // Last time the sensor was read  float temperature; // Temperature  float humidity; // Humidity  uint8\_t pin; // Pin connected  void readSensor(); // Read the measured sensor value  };  #endif // DHT22\_H |

|  |
| --- |
| // File DHT22.cpp  #include "DHT22.h"  /\*\*  \* @brief Construct a new DHT22::DHT22 object  \* @param pin pin connected microcontroller  \*/  DHT22::DHT22(int pin) : pin(pin)  {  resetTimer();  }  /\*\*  \* @brief Setup pin connected microcontroller  \* @param pin pin connected microcontroller  \*/  void DHT22::setup(int pin)  {  this->pin = pin;  resetTimer();  }  /\*\*  \* @brief Check the sensor is working properly  \* @return ERROR\_NONE - No error, ERROR\_TIMEOUT - Timeout error, ERROR\_CHECKSUM - Checksum error  \*/  DHT22::DHT\_ERROR\_t DHT22::getStatus() { return error; }  /\*\*  \* @brief Reset timer for setup sensor  \*/  void DHT22::resetTimer()  {  lastReadTime = millis() - 3000;  }  /\*\*  \* @brief Read the measured sensor value  \*/  void DHT22::readSensor()  {  // Make sure we don't poll the sensor too often  // - Max sample rate DHT11 is 1 Hz (duty cicle 1000 ms)  // - Max sample rate DHT22 is 0.5 Hz (duty cicle 2000 ms)  unsigned long startTime = millis();  if ((unsigned long)(startTime - lastReadTime) < 1999L)  {  return;  }  lastReadTime = startTime;  temperature = NAN;  humidity = NAN;  uint16\_t rawHumidity = 0;  uint16\_t rawTemperature = 0;  uint16\_t data = 0;  // Request sample  digitalWrite(pin, LOW); // Send start signal  pinMode(pin, OUTPUT);  delay(2);  pinMode(pin, INPUT);  digitalWrite(pin, HIGH); // Switch bus to receive data  // We're going to read 83 edges:  // - First a FALLING, RISING, and FALLING edge for the start bit  // - Then 40 bits: RISING and then a FALLING edge per bit  // To keep our code simple, we accept any HIGH or LOW reading if it's max 85 usecs long  #ifdef ESP32  // ESP32 is a multi core / multi processing chip  // It is necessary to disable task switches during the readings  portMUX\_TYPE mux = portMUX\_INITIALIZER\_UNLOCKED;  portENTER\_CRITICAL(&mux);  #else  // cli();  noInterrupts();  #endif  for (int8\_t i = -3; i < 2 \* 40; i++)  {  byte age;  startTime = micros();  do  {  age = (unsigned long)(micros() - startTime);  if (age > 90)  {  error = ERROR\_TIMEOUT;  #ifdef ESP32  portEXIT\_CRITICAL(&mux);  #else  // sei();  interrupts();  #endif  return;  }  } while (digitalRead(pin) == (i & 1) ? HIGH : LOW);  if (i >= 0 && (i & 1))  {  // Now we are being fed our 40 bits  data <<= 1;  // A zero max 30 usecs, a one at least 68 usecs.  if (age > 30)  {  data |= 1; // we got a one  }  }  switch (i)  {  case 31:  rawHumidity = data;  break;  case 63:  rawTemperature = data;  data = 0;  break;  }  }  #ifdef ESP32  portEXIT\_CRITICAL(&mux);  #else  // sei();  interrupts();  #endif  // Verify checksum  if ((byte)(((byte)rawHumidity) + (rawHumidity >> 8) + ((byte)rawTemperature) + (rawTemperature >> 8)) != data)  {  error = ERROR\_CHECKSUM;  return;  }  // Store readings  humidity = rawHumidity \* 0.1;  if (rawTemperature & 0x8000)  rawTemperature = -(int16\_t)(rawTemperature & 0x7FFF);  temperature = ((int16\_t)rawTemperature) \* 0.1;  error = ERROR\_NONE;  }  /\*\*  \* @brief Get the temperature measured by the sensor  \* @return temperature (celsius)  \*/  float DHT22::getTemperature()  {  readSensor();  if (error == ERROR\_TIMEOUT)  readSensor();  return temperature;  }  /\*\*  \* @brief Get the humidity measured by the sensor  \* @return humidity (%)  \*/  float DHT22::getHumidity()  {  readSensor();  if (error == ERROR\_TIMEOUT)  readSensor();  return humidity;  } |

|  |
| --- |
| /\*\*  \* @file MQ7.h  \* @author Group 5  \* @brief Library for CO (Carbon Monoxide Gas) ppm value by using MQ7 gas sensor  \* Datasheet: @see https://www.sparkfun.com/datasheets/Sensors/Biometric/MQ-7.pdf  \* @version 1.0  \* @date 2021-06-18  \* @copyright Copyright (c) 2021  \*  \*/  #ifndef MQ7\_H  #define MQ7\_H  #include <Arduino.h>  #define COEFFICIENT\_A 19.32  #define COEFFICIENT\_B -0.64  #define R\_LOAD 10  /\*\*  \* The coefficients are estimated from the sensitivity characteristics graph  \* of the MQ7 sensor for CO (Carbon Monoxide) gas by using Correlation function.  \*  \*  \* Explanation:  \* The graph in the datasheet is represented with the function f(x) = a \* (x ^ b).  \* Where: f(x) = ppm, x = Rs/R0  \*  \* The values were mapped with this function to determine the coefficients a and b.  \*  \*/  class MQ7  {  private:  uint8\_t analogPin;  uint8\_t digitalPin;  float voltageConversion(int); // Returns voltage from the raw input value  public:  MQ7(uint8\_t, uint8\_t); // Constructor to initialize the analog pin and input voltage to MQ7  void begin(); // Setup pin  float getPPM(); // Return the ppm value of CO gas  float getSensorResistance(); // To find the sensor resistance Rs  float getRatio(); // This function is for the deriving the Rs/R0 to find ppm  int getLimit(); // Get limit  };  #endif // MQ7\_H |

|  |
| --- |
| // File MQ7.cpp  #include "MQ7.h"  /\*\*  \* @brief Constructor to initialize the analog pin and digital pin to MQ7  \* @param analogPin Input pin from MQ7 analog pin  \* @param digitalPin Input pin from MQ7 digital pin  \*/  MQ7::MQ7(uint8\_t analogPin, uint8\_t digitalPin)  {  this->digitalPin = digitalPin;  this->analogPin = analogPin;  }  /\*\*  \* @brief Setup pin  \*/  void MQ7::begin()  {  pinMode(digitalPin, INPUT);  }  /\*\*  \* @brief Function is used to return the ppm value of CO gas concentration  by using the parameter found using the function f(x) = a \* ((Rs/R0) ^ b)  \* @return ppm value of Carbon Monoxide concentration  \*/  float MQ7::getPPM()  {  if (analogRead(analogPin) == 0)  return NAN;  return (float)(COEFFICIENT\_A \* pow(getRatio(), COEFFICIENT\_B));  }  /\*\*  \* @brief This function returns voltage from the raw input value  \* Refer ADC Conversion for further reference  \* @param value : value from analogPin  \* @return voltage  \*/  float MQ7::voltageConversion(int rawData)  {  return (float)rawData \* (3.3 / 4095.0);  }  /\*\*  \* @brief This function is for the deriving the Rs/R0 to find ppm  \* @return The value of Rs/R\_Load  \*/  float MQ7::getRatio()  {  float voltage = voltageConversion(analogRead(analogPin));  return (3.3 - voltage) / voltage;  }  int MQ7::getLimit()  {  return digitalRead(digitalPin);  }  /\*\*  \* @brief To find the sensor resistance Rs  \* @return Rs value  \*/  float MQ7::getSensorResistance()  {  return R\_LOAD \* getRatio();  } |

|  |
| --- |
| /\*\*  \* @file PM25.h  \* @author Group 5  \* @brief Library for dust density value by using PM2.5 GP2Y1010AU0F sensor  \* Datasheet: @see https://www.sparkfun.com/datasheets/Sensors/gp2y1010au\_e.pdf  \* @version 1.0  \* @date 2021-06-08  \*  \* @copyright Copyright (c) 2021  \*  \*/  #ifndef PM25\_H  #define PM25\_H  #include <Arduino.h>  #define R\_LOAD 30.0  class PM25  {  public:  PM25(int, int); // Construct a new PM25::PM25 object  void begin(); //Init pin mode for sensor  float getDustDensity(); //Get dust density of air  int getRawValue(); //Get raw value  float getVoltage(); //Get voltage  private:  int analogPin; // Read value pin  int ledPin; // LED IR pin  float dustDensity = 0; // Dust density value  void readSensor(); // Read the measured sensor value  float voltageConversion(float); // Returns voltage from the raw input value  };  #endif //PM25GP2Y10\_H |

|  |
| --- |
| // File PM25.cpp  #include "PM25.h"  /\*\*  \* @brief Construct a new PM25::PM25 object  \* @param analogPin Read value pin  \* @param ledPin Led IR pin  \*/  PM25::PM25(int analogPin, int ledPin) : analogPin(analogPin), ledPin(ledPin) {}  /\*\*  \* @brief Init pin mode for sensor  \*/  void PM25::begin()  {  pinMode(ledPin, OUTPUT);  }  /\*\*  \* @brief Read the measured sensor value  \*/  void PM25::readSensor()  {  digitalWrite(ledPin, LOW); // Turn on IR  delayMicroseconds(280);  float rawData = analogRead(analogPin);  delayMicroseconds(40);  digitalWrite(ledPin, HIGH); // Turn off IR  delayMicroseconds(9680);  float voltage = voltageConversion(rawData); // Convert from raw value to voltage  dustDensity = (0.17 \* (voltage + 0.45) - 0.01) \* 1000; // Convert from voltage to dust density  if (dustDensity < 0)  dustDensity = 0;  }  /\*\*  \* @brief Get dust density of air  \* @return dust density (ug/m3)  \*/  float PM25::getDustDensity()  {  volatile int a;  if (dustDensity == 0)  readSensor();  return dustDensity;  }  /\*\*  \* @brief This function returns voltage from the raw input value  \* Refer ADC Conversion for further reference  \* @param value : value from analogPin  \* @return voltage  \*/  float PM25::voltageConversion(float rawData)  {  return (float)rawData \* (3.3 / 4096.0);  } |

## B. Video demo sản phẩm

Link video demo hệ thống:[*https://youtu.be/nD2iUrJQ9IU*](https://youtu.be/nD2iUrJQ9IU)