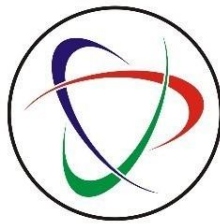


**TRƯỜNG ĐẠI HỌC BÁCH KHOA ĐÀ NẴNG**

**KHOA ĐIỆN TỬ - VIỄN THÔNG**



## **BÁO CÁO CUỐI KỲ**

**Môn học: Hệ Thống Thời Gian Thực**

**Đề tài: INDUSTRIAL AUTOMATION AND PROCESS  
CONTROL USING CAN PROTOCOL**

GVHD: TS. Đào Duy Tuấn

Nhóm 6: Lê Quang Hoàng – 106210214

Trần Đức Phát – 106210225

Ngô Nhật Minh – 106210222

Huỳnh Dích – 106210232

Đà Nẵng, 2025

# 1. Giới thiệu đề tài

## 1.1 Mục tiêu đề tài

Mục tiêu của đề tài là xây dựng một hệ thống tự động hóa, có khả năng điều khiển tốc độ động cơ dựa trên nhiệt độ môi trường và tích hợp chức năng chữa cháy, sử dụng giao thức truyền thông CAN để kết nối giữa các thiết bị. Hệ thống có khả năng hoạt động ổn định trong môi trường công nghiệp, đảm bảo tính phản ứng nhanh và đáng tin cậy trong quá trình giám sát và điều khiển.

## 1.2 Ứng dụng thực tế

Hệ thống được thiết kế hướng đến các ứng dụng giám sát và điều khiển thông minh trong:

- Công nghiệp: giám sát nhiệt độ trong nhà máy, điều khiển hệ thống làm mát tự động, cảnh báo khi phát hiện cháy.
- Nhà ở thông minh: kiểm soát môi trường trong nhà, tích hợp hệ thống chữa cháy an toàn.
- Tòa nhà, trung tâm dữ liệu: duy trì nhiệt độ phù hợp cho thiết bị điện tử, xử lý sớm khi có sự cố.

## 1.3 Lý do chọn đề tài

- Giao thức CAN là một chuẩn truyền thông phổ biến và đáng tin cậy, được sử dụng rộng rãi trong công nghiệp và ngành công nghiệp ô tô, có khả năng truyền dữ liệu chính xác trong môi trường có nhiễu điện từ.
- Việc tích hợp FreeRTOS giúp hệ thống xử lý song song nhiều tác vụ, tăng tính phản hồi nhanh và ổn định.
- Đề tài có tính ứng dụng thực tiễn cao, dễ mở rộng và phù hợp với định hướng phát triển hệ thống nhúng hiện đại.

# 2. Tổng quan hệ thống

Hệ thống bao gồm hai nút chính (Node) được kết nối với nhau thông qua bus CAN, đảm nhiệm các vai trò khác nhau trong quy trình giám sát và điều khiển:

### Node 1: Cảm biến và truyền dữ liệu

- Chức năng chính: Thu thập dữ liệu từ cảm biến môi trường và gửi lên mạng CAN.
- Các cảm biến sử dụng:
  - DHT11: Cảm biến nhiệt độ và độ ẩm.

- KY-026: Cảm biến phát hiện ngọn lửa hoặc khói (phát hiện cháy).
- Hoạt động: Node này sẽ định kỳ đọc nhiệt độ, độ ẩm và kiểm tra độ cháy (Analog) từ KY-026. Dữ liệu thu thập được đóng gói và gửi đi qua giao thức CAN đến Node 2.

## Node 2: Nhận dữ liệu, hiển thị và điều khiển









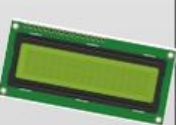


### Chức năng:

- Nhận dữ liệu từ Node 1 qua CAN.
- Hiển thị giá trị nhiệt độ, độ ẩm, và trạng thái cháy trên LCD.
- Phản ứng nhanh khi có tín hiệu cháy: bật máy bơm, tắt quạt, hiển thị cảnh báo.
- Điều khiển tốc độ quạt qua driver cầu H (L298N).
- Giám sát online bằng Grafana.
- Có thể đóng vai trò giám sát trung tâm hoặc giao diện người dùng.

### Phân cứng sử dụng:

- ESP32 Dev Board – điều khiển chính.
- Module MCP2515 CAN – giao tiếp mạng CAN.
- Màn hình LCD 16x2 để hiển thị.
- Module L298N – điều khiển động cơ/quạt.
- Quạt 12V.
- Bơm mini 5V.

### Linh kiện sử dụng trong hệ thống:

NODE 1						
NODE 2						

## 3. Giao thức CAN

CAN (Controller Area Network) là một giao thức truyền thông nối tiếp, được thiết kế để cho phép các thiết bị (node) giao tiếp với nhau một cách hiệu quả, đáng tin cậy và chống nhiễu tốt thông qua một đường truyền chung gọi là CAN bus. Giao thức này ban đầu được phát triển bởi Bosch cho ngành công nghiệp ô tô, nhưng hiện nay

được sử dụng rộng rãi trong nhiều lĩnh vực như tự động hóa công nghiệp, thiết bị y tế, và hệ thống nhúng.

### 3.1. Kiến trúc tổng quan của hệ thống CAN

Các thành phần chính của hệ thống CAN bao gồm:

- Host Controller: Là bộ vi điều khiển chính, chịu trách nhiệm xử lý dữ liệu đầu vào/đầu ra, thực hiện thuật toán điều khiển và giao tiếp với người dùng.
- CAN Controller: Là phần xử lý tín hiệu CAN, thực hiện đóng gói/giải mã và quản lý giao thức CAN như xác nhận (ACK), kiểm tra lỗi (CRC).
- CAN Transceiver: Là phần chịu trách nhiệm chuyển đổi dữ liệu số từ CAN controller thành tín hiệu điện vi sai (CAN\_H, CAN\_L) và ngược lại. Đây là phần giao tiếp vật lý với CAN bus.
- CAN Bus: Là đường truyền dữ liệu vật lý sử dụng cặp dây xoắn đôi gồm:
  - CAN\_H (CAN High)
  - CAN\_L (CAN Low)
  - Các đầu cuối của CAN bus thường được gắn điện trở 120  $\Omega$  để giảm phản xạ tín hiệu và chống nhiễu.

### 3.2 Ưu điểm của giao thức CAN

- Chống nhiễu tốt: Nhờ sử dụng truyền tín hiệu vi sai giữa CAN\_H và CAN\_L.
- Hỗ trợ giao tiếp đa thiết bị: Cho phép nhiều node (thiết bị) truyền và nhận dữ liệu với nhau, sử dụng địa chỉ ID riêng.
- Tốc độ và độ tin cậy cao: Có cơ chế phát hiện lỗi (CRC), xác nhận (ACK) và ưu tiên ID truyền.
- Hiệu quả và tiết kiệm dây dẫn: Nhiều thiết bị cùng chia sẻ một bus truyền.

### 3.3 Ứng dụng trong đề tài

Trong đề tài này, giao thức CAN đóng vai trò là cầu nối truyền thông giữa hai node chính:

- Node 1: Thu thập dữ liệu từ cảm biến nhiệt độ hoặc cảm biến cháy và gửi gói tin qua CAN.
- Node 2: Nhận dữ liệu từ CAN và điều khiển tốc độ động cơ tương ứng với nhiệt độ hoặc cảnh báo cháy.

Việc sử dụng CAN giúp đảm bảo tính ổn định, độ tin cậy và khả năng mở rộng của hệ thống trong môi trường công nghiệp có nhiều nhiễu điện từ và yêu cầu thời gian thực.

## 4. RTOS

Hệ điều hành thời gian thực RTOS được sử dụng để tổ chức và quản lý các chức năng chính của hệ thống một cách linh hoạt, an toàn và hiệu quả, cho phép ứng dụng chạy đa tác vụ và có thể đáp ứng được “deadline” theo thời gian thực.

RTOS bao gồm 2 loại:

- Soft Real-Time system: cho phép các tác vụ trong hệ thống trễ deadline. Nếu hệ thống không đáp ứng đúng thời hạn, có thể nhiều hơn một lần, hệ thống đó không được coi là bị lỗi.
- Hard Real-Time system: đảm bảo rằng các tác vụ thời gian thực được hoàn thành trong thời hạn deadline yêu cầu. Nếu không đáp ứng được một thời hạn có thể dẫn đến lỗi hệ thống nghiêm trọng.

Các thành phần quan trọng của RTOS được áp dụng như sau:

### 4.1. Task

- Task là một đoạn code được lập lịch bởi bộ scheduler để chạy. Các tác vụ được triển khai dưới dạng luồng trong RTOS, mỗi luồng chạy độc lập nhưng chia sẻ tài nguyên cho phép thực hiện nhiều tác vụ. Mỗi tác vụ có các đặc điểm về thời gian thực hiện, độ ưu tiên, deadline, trạng thái. Chương trình có thể có một hay nhiều task cùng chạy đồng thời.

- Task priority: Ở chế độ lập lịch preemptive thì task có độ ưu tiên cao hơn luôn được lập lịch để chạy trước task có priority thấp hơn. Điều này không đúng ở chế độ Cooperative. Task priority được lưu trữ trong Task Control Block của Task và được scheduler sử dụng để lập lịch.

- Task memory: Khi một task được tạo ra thì Task TCB và Task Stack sẽ được tạo ra trên vùng nhớ Heap của RTOS.

- Task TCB: Chứa tất cả các thông tin về task được tạo như: độ ưu tiên của task, Stack size, Stack pointer, task state ....
- Task Stack: Là vùng bộ nhớ stack riêng của task, chứa: các biến local, parameter được truyền vào task thông qua API...
- Task TCB, Task Stack chứa các thông tin quan trọng để task có thể hoạt động. Kích thước của nó có thể thay đổi dựa trên các config được thực hiện ở trong file RTOSConfig.h

- Task state (trạng thái) bao gồm:

- Idle: task đang không thực hiện tính toán tài nguyên.
- Ready: task được sẵn sàng để hoạt động, nhưng đang chờ để xử lý.

- Running: đang thực hiện các hoạt động liên quan.
  - Waiting: task cần phải chờ để task có độ ưu tiên cao hơn thực thi.
  - Suspended: task đang chờ tài nguyên.
- Các task có thể trao đổi dữ liệu với nhau thông qua: Queue, biến Global và Static.
- Đảm bảo đồng bộ hóa với nhau thông qua: Semaphore, Mutex, Notification.
- Phân chia task trong hệ thống để thực hiện các chức năng:
- Node 1 (STM32 – Cảm biến và truyền dữ liệu):
    - Task đọc cảm biến nhiệt độ & độ ẩm (DHT11).
    - Task đọc cảm biến phát hiện cháy (KY-026).
    - Task gửi dữ liệu cảm biến lên mạng CAN.
  - Node 2 (ESP32C3 – Hiển thị và giám sát):
    - Task nhận dữ liệu từ CAN.
    - Task điều khiển động cơ bơm và quạt.
    - Task hiển thị dữ liệu lên LCD.
    - Task gửi dữ liệu lên Grafana để giám sát.

## 4.2 Queue

Queue (hàng đợi) là một cấu trúc dữ liệu dùng để truyền thông giữa các task hoặc giữa task và interrupt theo cơ chế FIFO (First In, First Out) — nghĩa là dữ liệu nào đưa vào trước sẽ được lấy ra trước. Là cơ chế truyền dữ liệu an toàn giữa các task, giúp tránh xung đột và đảm bảo tính toàn vẹn dữ liệu.

Trong hệ thống: task nhận CAN lấy giá trị nhiệt độ, độ ẩm, phát hiện cháy truyền vào queue. Queue đầu tiên bao gồm 2 task là hiển thị và điều khiển. Queue thứ hai gồm task gửi dữ liệu lên Grafana để giám sát và tránh xung đột vì thời gian giao tiếp với server lâu hơn các task còn lại.

## 4.3 Scheduler

Lập lịch trong RTOS rất quan trọng để đảm bảo rằng các nhiệm vụ trong thời gian thực đáp ứng thời hạn, vì ngay cả một sự chậm trễ nhỏ trong việc thực hiện nhiệm vụ cũng có thể gây ra hậu quả nghiêm trọng.

Có hai loại thuật toán lập lịch tác vụ RTOS chính:

- Các thuật toán lập lịch ưu tiên cho phép RTOS làm gián đoạn một tác vụ đang chạy và chuyển sang tác vụ có mức độ ưu tiên cao hơn đảm bảo các tác vụ quan trọng đáp ứng đúng thời hạn.
- Các thuật toán lập lịch không ưu tiên không cho phép RTOS làm gián đoạn tác vụ đang chạy.

Tùy theo yêu cầu chức năng của hệ thống đang thực hiện để các thuật toán lập lịch cần xét đến: mức độ ưu tiên các tác vụ, task deadline, thời gian thực hiện tác vụ,...

Triển khai trong hệ thống điều khiển giám sát công nghiệp với các task sẽ được thực hiện lần lượt theo độ ưu tiên từ cao xuống thấp. Task nhận dữ liệu CAN được ưu tiên cao nhất vì cần dữ liệu thời gian thực, sau đó sẽ là task điều khiển, cuối cùng là task hiển thị và task gửi lên Grafana để giám sát.

## 4.4 Semaphore và Mutex

Semaphore và Mutex là các cơ chế đồng bộ hóa và bảo vệ tài nguyên trong RTOS.

- Semaphore được dùng để đồng bộ hoạt động giữa các task giúp một hoặc nhiều luồng thực thi có thể thực hiện hoặc giải phóng cho mục đích đồng bộ hóa hoặc loại trừ lẫn nhau.
- Mutex dùng để bảo vệ các tài nguyên dùng chung như bộ nhớ đệm (buffer), tránh tình trạng truy cập đồng thời gây lỗi. Ví dụ: task đọc cảm biến và task gửi CAN cùng truy cập một buffer chung. Mutex đảm bảo chỉ một task truy cập buffer tại một thời điểm. Trong hệ thống sử dụng mutex cho các task như đọc cảm biến, gửi CAN và hiển thị lên LCD.

Trong hệ thống được triển khai với thư viện FreeRTOS. FreeRTOS là một bộ thư viện RTOS mã nguồn mở, nhỏ gọn và phổ biến, được sử dụng rộng rãi trong các ứng dụng nhúng. Được phát triển bởi Richard Barry vào năm 2003 và hiện được duy trì bởi Amazon Web Services (AWS), FreeRTOS cung cấp các tính năng cốt lõi của một RTOS, bao gồm quản lý tác vụ, lập lịch, đồng bộ hóa, và truyền thông giữa các tác vụ.

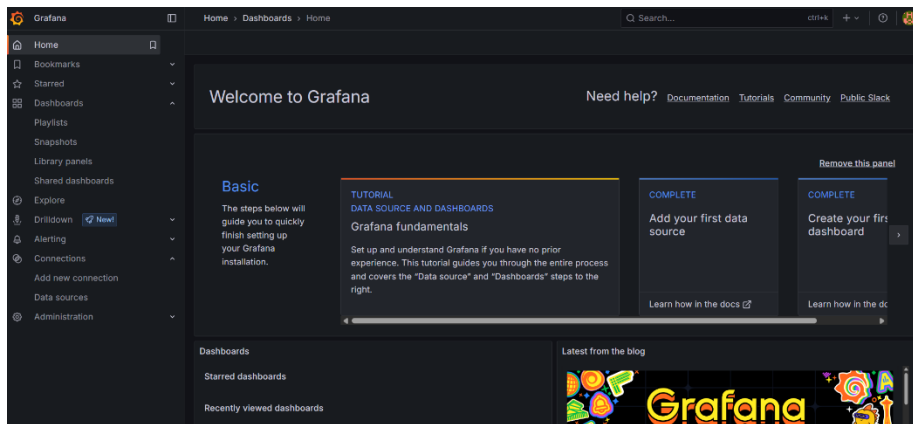
Thông qua việc sử dụng RTOS và các cơ chế như task, semaphore, queue..., được phối hợp chặt chẽ để đáp ứng các yêu cầu khắt khe về thời gian và tài nguyên. Hệ thống hoạt động ổn định, phản ứng nhanh với thay đổi từ môi trường, đồng thời dễ dàng mở rộng trong các ứng dụng giám sát và điều khiển công nghiệp.

## 5. GRAFANA

### 5.1 Grafana là gì ?

Grafana là một nền tảng được sử dụng để phân tích, giám sát và trực quan hóa dữ liệu. Nó cho phép tạo ra các bảng điều khiển tương tác từ nhiều nguồn dữ liệu khác nhau để dễ dàng theo dõi hệ thống, ứng dụng.

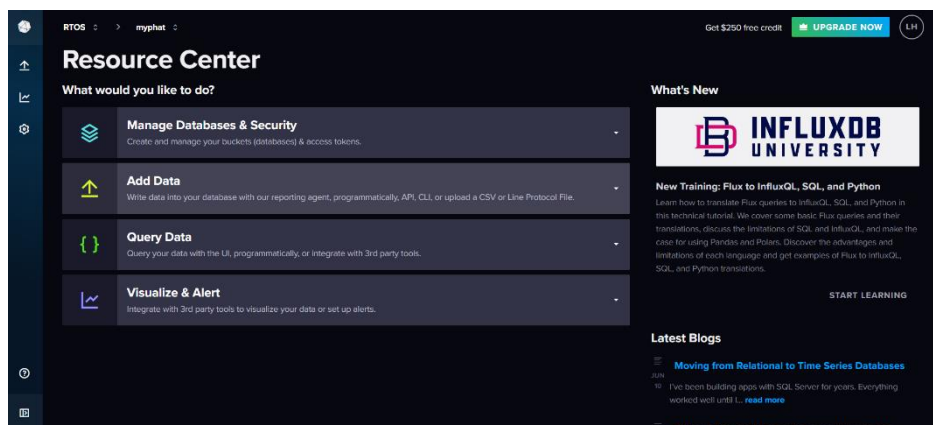
Grafana hoạt động bằng cách kết nối với nguồn dữ liệu có sẵn từ các bên khác, ví dụ như influxDB. Sau đó, cho phép truy vấn, chuyển đổi và trực quan hóa dữ liệu đó thông qua các bảng điều khiển và panel.



## 5.2 InfluxDB là gì ?

InfluxDB là một cơ sở dữ liệu theo chuỗi thời gian, được tối ưu hóa đặc biệt để lưu trữ và truy vấn dữ liệu dạng chuỗi thời gian. Dữ liệu chuỗi thời gian là các điểm dữ liệu được gắn thẻ thời gian, thường được thu thập liên tục từ các nguồn như cảm biến, hệ thống giám sát, ứng dụng,...

Đặc điểm nổi bật của InfluxDB là được thiết kế để xử lý dữ liệu được gắn thẻ thời gian, tức là dữ liệu sẽ được lưu liên tục theo thời gian. Điều này rất tốt trong việc quản lý, vận hành các hệ thống tự động hóa.



## 5.3 Kết hợp để hiển thị trực quan

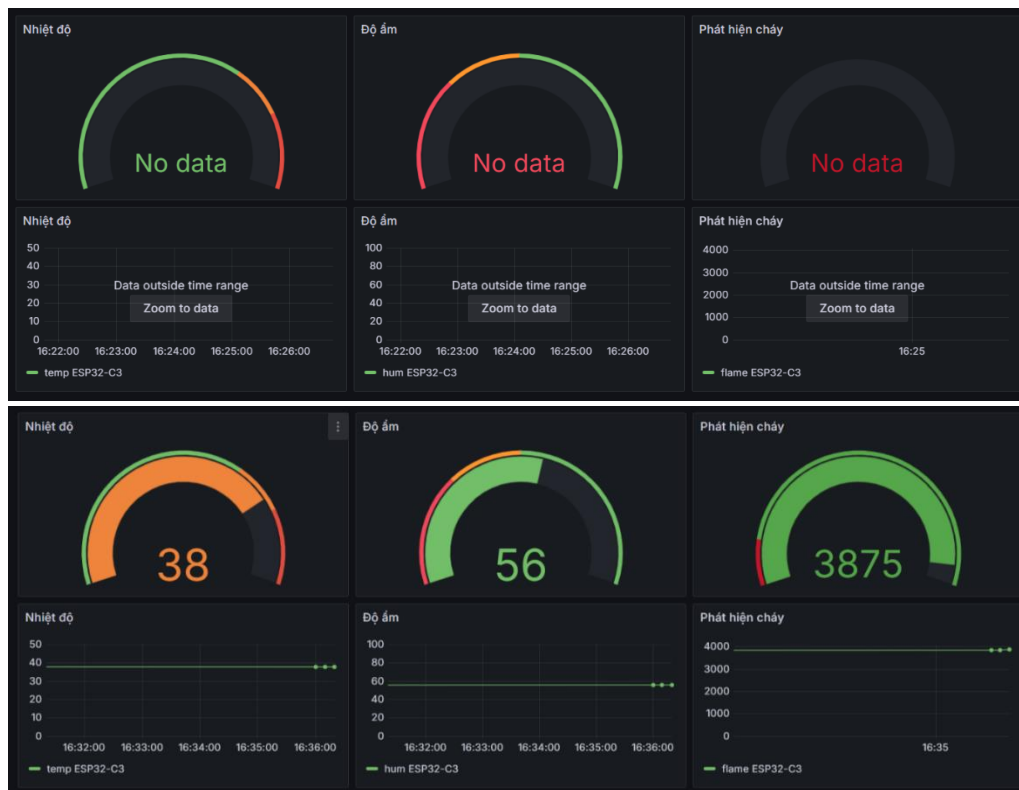
- Hệ thống hoạt động giúp đọc dữ liệu từ cảm biến sau đó gửi dữ liệu chuỗi thời gian đến InfluxDB. InfluxDB nhận và lưu trữ dữ liệu một cách hiệu quả, sắp xếp theo thời gian và theo các tag, field liên quan. Grafana được cấu hình để thêm InfluxDB làm một nguồn dữ liệu. Khi sử dụng, Grafana sẽ gửi các truy vấn đến InfluxDB và InfluxDB xử lý truy vấn, tổng hợp dữ liệu và trả về kết quả cho Grafana.
- Grafana nhận dữ liệu từ InfluxDB và hiển thị dưới dạng đồ thị và các panel trực quan trên bảng điều khiển.
- Trực quan hóa dữ liệu với Grafana:



- Truyền dữ liệu: Hệ thống gửi dữ liệu cảm biến (nhiệt độ, độ ẩm, ngọn lửa) từ cảm biến DHT11 và KY-026 qua InfluxDB bằng giao thức HTTP, với dữ liệu được cập nhật liên tục từ taskGrafana.
- Hiển thị thời gian thực: Các thông số như nhiệt độ, độ ẩm, và mức ngọn lửa được ghi nhận và hiển thị trên Grafana dưới dạng biểu đồ, bảng điều khiển, hoặc cảnh báo. Grafana giúp theo dõi xu hướng, phát hiện bất thường (như ngọn lửa vượt ngưỡng 500), và đánh giá hiệu suất điều khiển quạt/bơm.

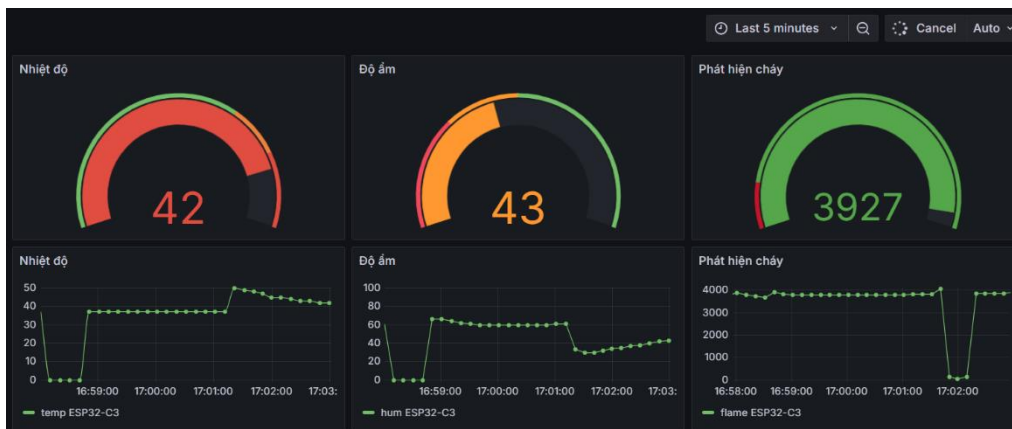
## 5.4 Kết quả đạt được

- Khi hệ thống chưa hoạt động thì grafana sẽ không có dữ liệu để hiển thị và giao diện sẽ là no data.
- Khi hệ thống đã hoạt động, dữ liệu hiển thị bao gồm đồ thị của nhiệt độ, độ ẩm và phát hiện cháy. Trong đó sẽ hiển thị điểm dữ liệu hiện tại đang đạt được của hệ thống và hiển thị toàn bộ dữ liệu từ lúc hệ thống bắt đầu hoạt động.



Kết quả mô phỏng của 2 trường hợp gồm thay đổi nhiệt độ và phát hiện cháy.

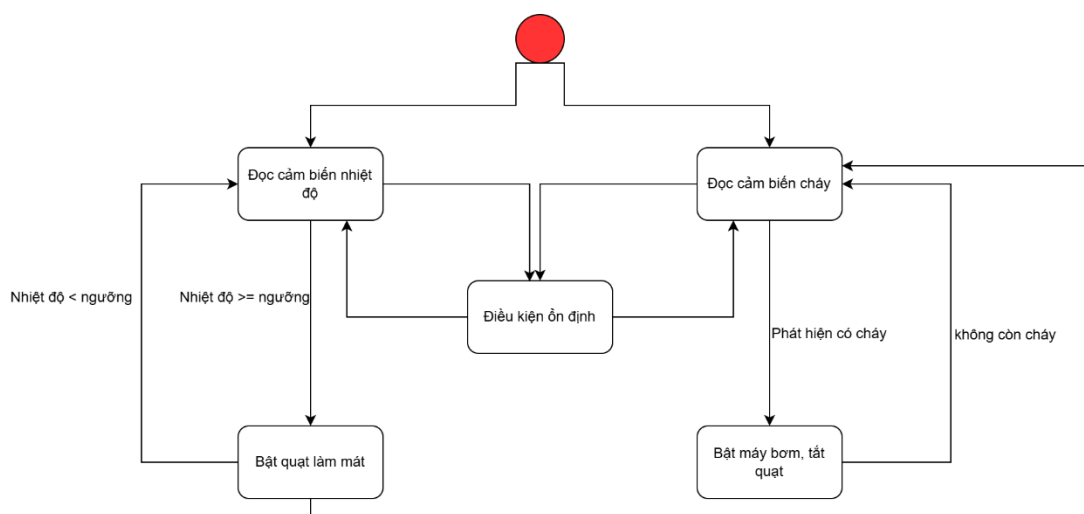
- Khi nhiệt độ tăng lên, grafana sẽ hiển thị điểm dữ liệu đang đạt được và đồng thời ở đồ thị hiển thị toàn bộ dữ liệu cũng sẽ thay đổi theo thời gian.
- Tương tự cho trường hợp mô phỏng phát hiện cháy. Khi phát hiện cháy điểm dữ liệu đang đạt được sẽ hiển thị ở mức thấp.



Tích hợp Grafana vào hệ thống không chỉ cung cấp giao diện thân thiện để giám sát mà còn hỗ trợ lưu trữ dữ liệu dài hạn, tạo báo cáo, và kích hoạt cảnh báo tự động khi các thông số vượt ngưỡng.

## 6. Sơ đồ mô tả hệ thống

### 6.1 Sơ đồ máy trạng thái



Sơ đồ máy trạng thái có vai trò mô tả quá trình hoạt động của hệ thống theo từng trạng thái cụ thể như trạng thái ổn định, làm mát, chữa cháy và các điều kiện chuyển đổi giữa chúng như nhiệt độ vượt ngưỡng, phát hiện cháy. Nhờ đó, giúp thiết kế sơ đồ một cách logic, điều khiển dễ hiểu, chính xác, đảm bảo ưu tiên chữa cháy khi cần, đồng thời hỗ trợ lập trình, bảo trì và mở rộng hệ thống hiệu quả.

### Cách hoạt động:

Trạng thái khởi đầu: hệ thống bắt đầu hoạt động với ký hiệu là dấu chấm đỏ.

Đọc cảm biến nhiệt độ:

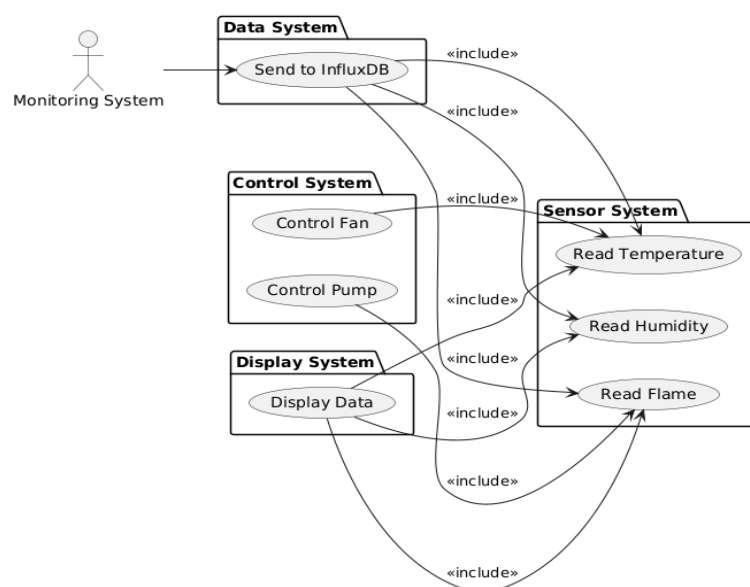
- Nếu nhiệt độ < ngưỡng: trở về trạng thái ổn định.
- Nếu nhiệt độ  $\geq$  ngưỡng: chuyển sang trạng thái Bật quạt làm mát.

Đọc cảm biến cháy:

- Nếu phát hiện có cháy: chuyển sang trạng thái Bật máy bơm, tắt quạt.
- Nếu không còn cháy: trở lại trạng thái ổn định.

Trạng thái ổn định: kiểm tra định kỳ cả hai cảm biến nhiệt và cháy để quyết định hành động tiếp theo.

## 6.2 Use Case Diagram



Tác nhân (Actor):

- Monitoring System: Tác nhân chính, đại diện cho người dùng hoặc hệ thống giám sát tương tác với các thành phần khác.
- Hệ thống chính (Systems).

Data System:

- Use Case: "Send to InfluxDB".
- Mô tả: Gửi dữ liệu đến InfluxDB.

Control System:

- Use Cases: "Control Fan", "Control Pump".
- Mô tả: Quản lý quạt và bơm dựa trên dữ liệu cảm biến.

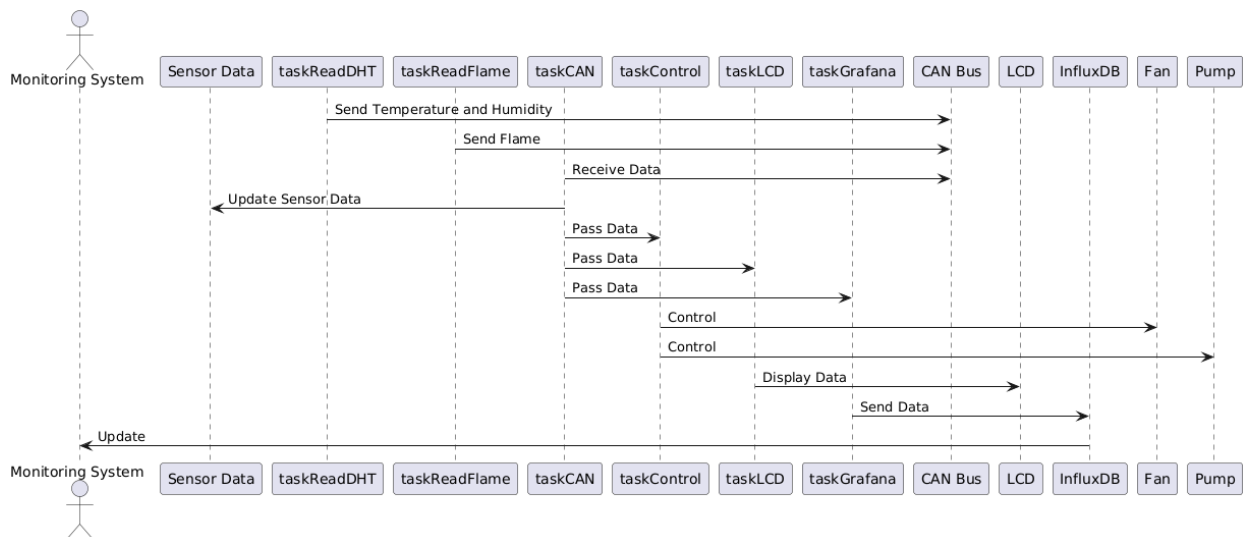
Display System:

- Use Case: "Display Data".
- Mô tả: Hiển thị dữ liệu cho người dùng.

Sensor System:

- Use Cases: "Read Temperature", "Read Humidity", "Read Flame".
- Mô tả: Thu thập dữ liệu nhiệt độ, độ ẩm, và tín hiệu ngọn lửa.
- Luồng hoạt động tổng quan.
- Monitoring System yêu cầu dữ liệu từ Data System.
- Data System gửi dữ liệu đến InfluxDB.
- Sensor System cung cấp dữ liệu cho Control System (điều chỉnh quạt/bơm) và Display System (hiển thị dữ liệu).

## 6.3 Sequence Diagram

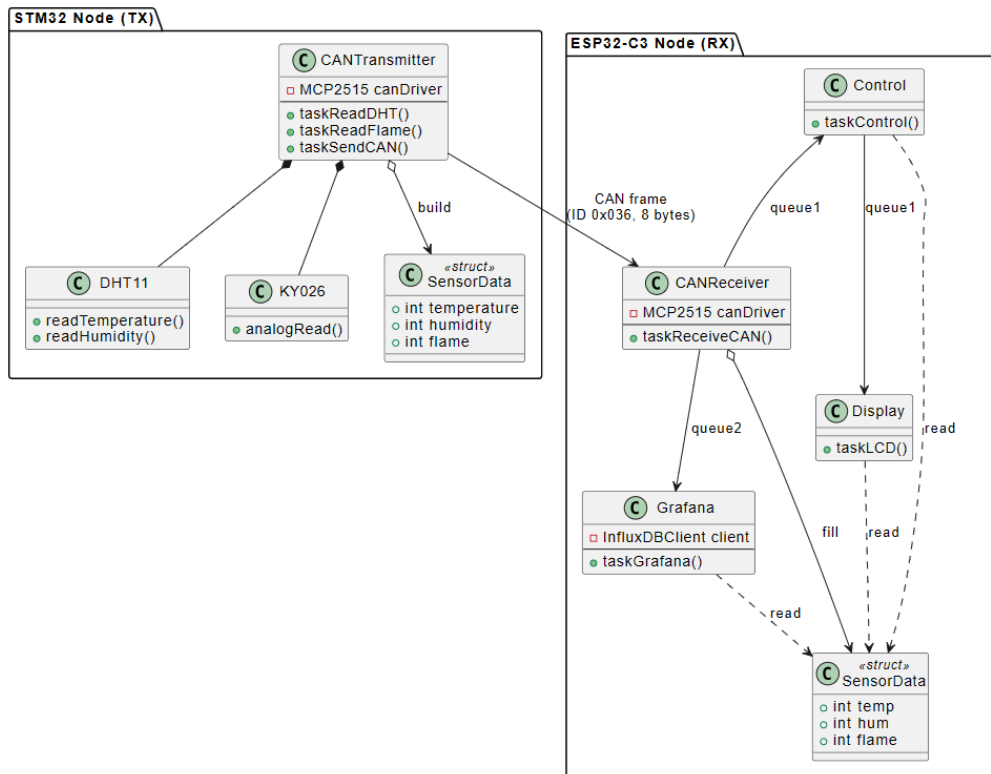


Mô tả luồng tương tác đơn giản giữa các thành phần:

- "taskReadDHT" và "taskReadFlame" gửi dữ liệu qua "CAN Bus",
- "taskCAN" nhận và phân phối dữ liệu đến "taskControl", "taskLCD", và "taskGrafana".
- "taskControl" điều khiển "Fan" và "Pump", "taskLCD" hiển thị trên "LCD",

- "taskGrafana" gửi dữ liệu đến "InfluxDB", cuối cùng cập nhật cho "Monitoring System"

## 6.4 Class Diagram

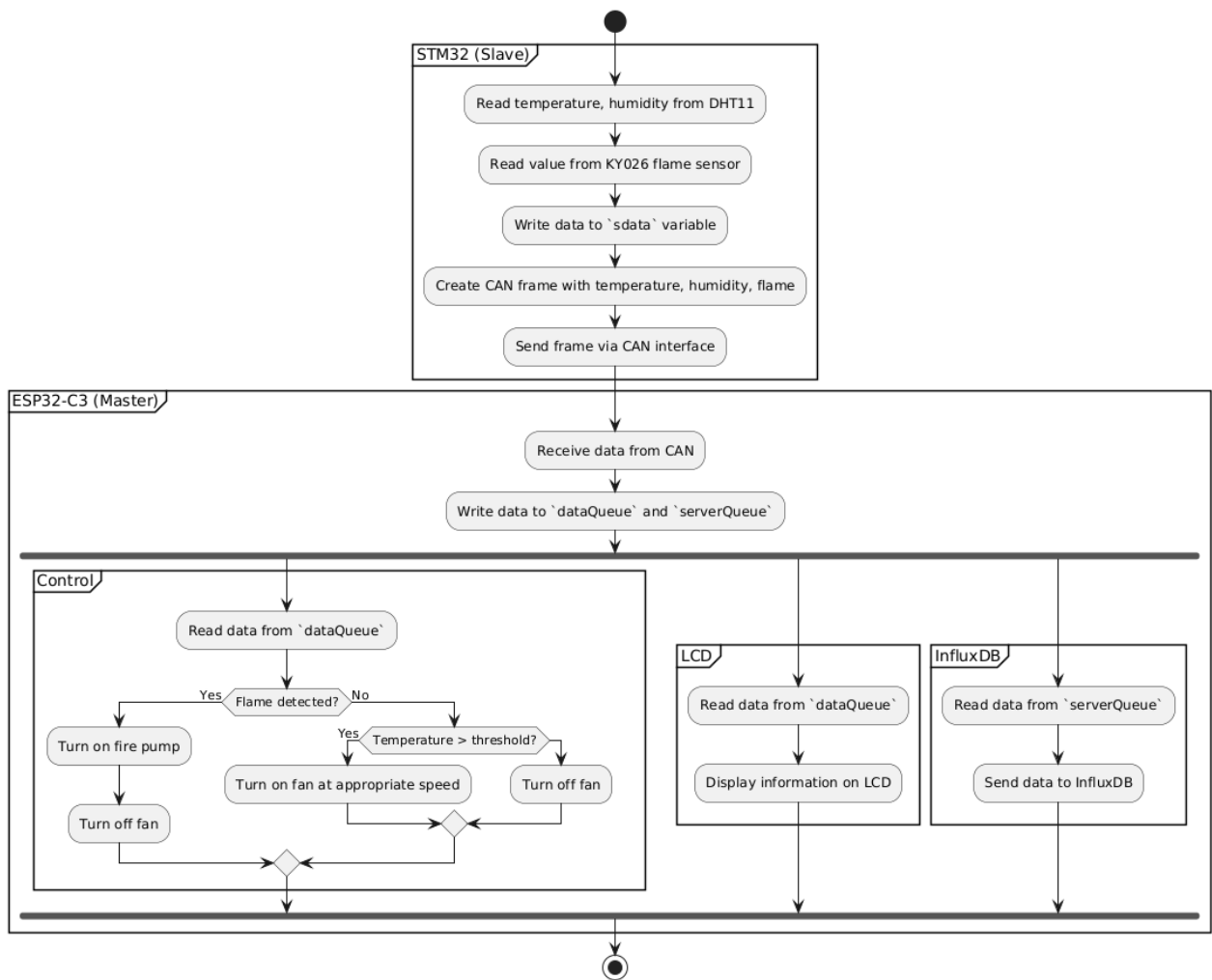


Class Diagram trên mô tả kiến trúc tổng thể của hệ thống truyền nhận dữ liệu qua giao tiếp CAN giữa hai vi điều khiển: STM32 (node truyền – TX) và ESP32-C3 (node nhận – RX).

Trong node STM32, lớp CANTransmitter đóng vai trò trung tâm, có quan hệ association với hai cảm biến DHT11 (đo nhiệt độ và độ ẩm) và KY026 (phát hiện lửa), cho phép thu thập dữ liệu định kỳ. Sau đó, CANTransmitter tạo ra một cấu trúc SensorData thông qua quan hệ aggregation và gửi nó qua giao tiếp CAN, sử dụng khung CAN với ID là 0x036 và độ dài dữ liệu là 8 byte.

Ở phía ESP32-C3, lớp CANReceiver có quan hệ aggregation với SensorData, khi tiếp nhận dữ liệu và điền vào cấu trúc này. SensorData sau đó được chia sẻ đến ba lớp khác thông qua các quan hệ dependency: Control để xử lý điều khiển động cơ (bơm và quạt), Display để hiển thị thông tin trên màn hình LCD, và Grafana để gửi dữ liệu lên nền tảng giám sát qua InfluxDB. Ngoài ra, các lớp này liên lạc nội bộ thông qua các hàng đợi của FreeRTOS thể hiện quá trình truyền thông tin tuần tự giữa các tác vụ.

## 6.5 Activity Diagram



Sơ đồ hoạt động (Activity Diagram) mô tả tổng quan toàn bộ quá trình vận hành của các tác vụ trên hệ thống, được chia thành hai node chính: node đọc cảm biến và node điều khiển.

- Tại node đọc cảm biến (STM32): Hoạt động đo đạc được thực hiện thông qua các cảm biến DHT11 và KY-026. STM32 thu thập dữ liệu từ các cảm biến này, sau đó đóng gói dữ liệu vào khung truyền CAN thông qua khối truyền (CAN Send), và gửi thông tin sang node điều khiển qua giao thức CAN.
- Tại node điều khiển: Dữ liệu từ khung CAN được tiếp nhận bởi khối nhận (CAN Receive), sau đó ghi vào hai hàng đợi: dataQueue và serverQueue. Từ đây, các tác vụ xử lý sẽ được kích hoạt và thực thi song song. Các tác vụ chính tại node điều khiển bao gồm:
  - TaskControl: Đọc dữ liệu từ dataQueue, thực hiện phân tích điều kiện môi trường. Nếu phát hiện có cháy, kích hoạt bơm chữa cháy. Ngược lại, khi không có cháy, hệ thống sẽ giám sát nhiệt độ để điều khiển quạt làm mát phù hợp.
  - TaskLCD: Đọc dữ liệu từ dataQueue và hiển thị thông tin môi trường (nhiệt độ, cảnh báo cháy) lên màn hình LCD giúp giám sát tại chỗ.

- TaskGrafana: Đọc dữ liệu từ serverQueue, sau đó gửi dữ liệu đến InfluxDB thông qua giao thức HTTP, sử dụng chức năng Wi-Fi tích hợp trên ESP32. Từ đó, dữ liệu sẽ được hiển thị trên nền tảng giám sát từ xa Grafana.

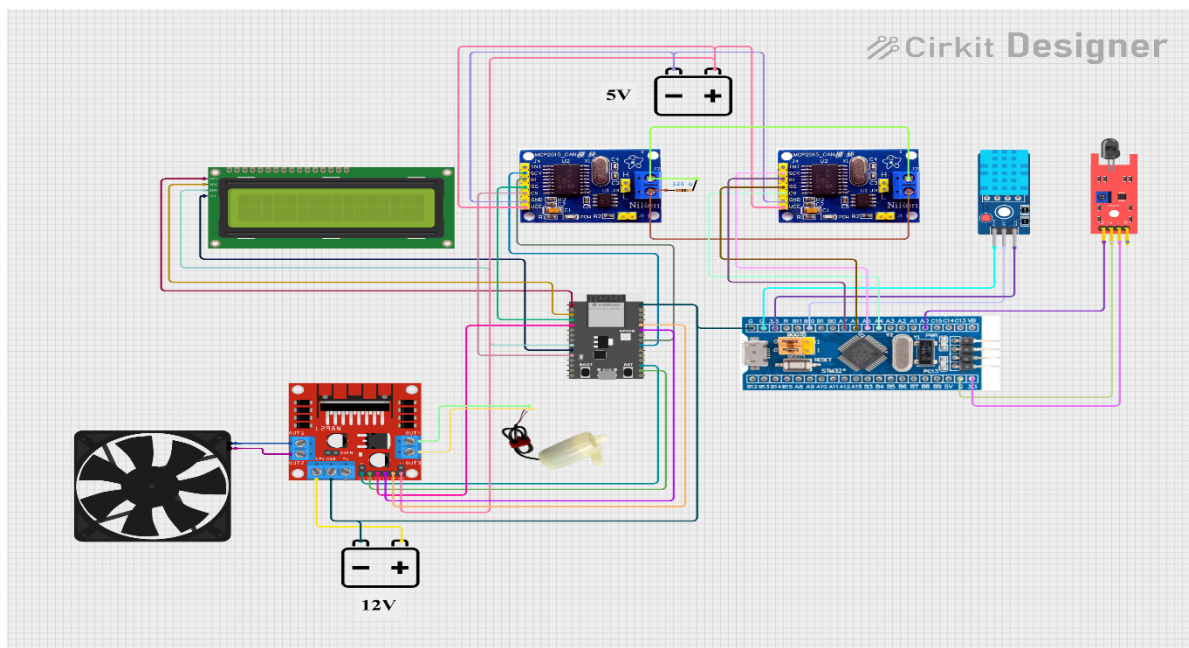
## 7. Triển khai hệ thống

### 7.1. Triển khai phần cứng

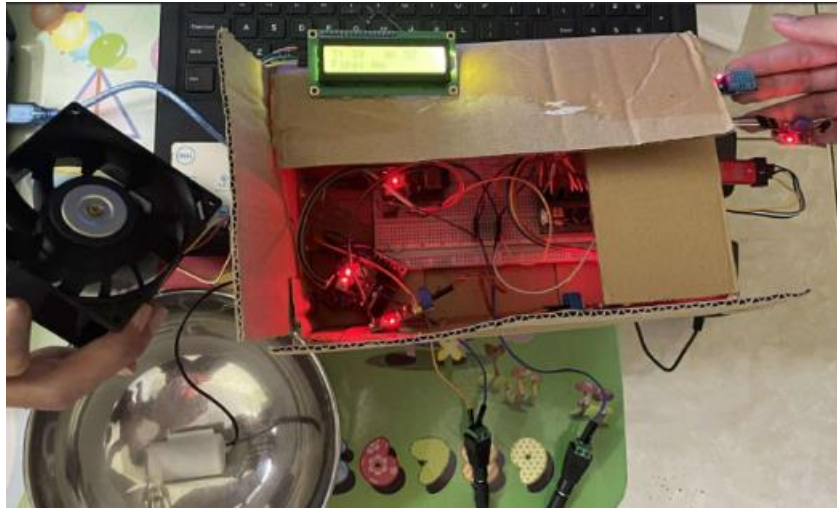
Từ các linh kiện và thiết bị ở phần tổng quan, tiến hành triển khai kết nối phần cứng với các bước:

- Kiểm tra chất lượng linh kiện được chuẩn bị: có nhận nguồn, các chân kết nối còn ổn định.
- Kiểm tra chức năng của từng nhóm linh kiện: đọc giá trị cảm biến so với thực tế và thực hiện điều chỉnh. Điều khiển cánh quạt, bơm và cố định các dây kết nối. Kiểm tra chất lượng hiển thị của LCD và điều chỉnh.
- Kết nối giao tiếp CAN và kiểm tra nguồn cũng như tín hiệu truyền giữa 2 con MCP2515.
- Kết nối các chân của linh kiện theo sơ đồ kết nối trên phần mềm cho toàn bộ hệ thống.
- Kết nối nguồn vào hệ thống và tiến hành thực nghiệm.
- Kiểm tra chức năng của hệ thống trước khi tiến hành đóng hộp sản phẩm.

Sơ đồ mạch mô phỏng:



Mạch thực tế:



## 7.2. Ngôn ngữ và phần mềm sử dụng

Phần mềm sử dụng là ArduinoIDE viết bằng ngôn ngữ C/C++ cho toàn bộ hệ thống. Trên đó sử dụng các thư viện sau để triển khai:

- SPI.h: Thư viện sử dụng để giao tiếp với các thiết bị hỗ trợ (MCP2515).
- mcp2515.h: Thư viện hỗ trợ mô-đun điều khiển CAN BUS sử dụng MCP2515.
- DHT.h: Thư viện hỗ trợ giao tiếp với các cảm biến nhiệt độ và độ ẩm dòng DHT như DHT11.
- Wire.h: Thư viện này hỗ trợ giao tiếp I2C.
- LiquidCrystal\_I2C.h: Thư viện điều khiển LCD sử dụng module I2C.
- MapleFreeRTOS900.h: Thư viện cho phép sử dụng FreeRTOS trên board STM32 (ESP32 đã tích hợp sẵn nên không cần thư viện).
- WiFiMulti.h: Thư viện giúp ESP32 kết nối với mạng WiFi đã khai báo sẵn
- InfluxDbClient.h và InfluxDbCloud.h: Thư viện giúp kết nối và giao tiếp với database InfluxDB (Grafana).

## 8. Kết quả thực nghiệm

- Kịch bản thử nghiệm:
  - Mô phỏng thay đổi giá trị nhiệt độ, độ ẩm môi trường bằng máy sấy.
  - Mô phỏng đám cháy bằng nguồn lửa gần cảm biến phát hiện cháy KY-206.
  - Quan sát dữ liệu truyền đi thông qua giao tiếp CAN trên ESP32C3.
  - Thực nghiệm điều khiển động cơ tại 3 mức tốc độ và điều khiển máy bơm trong trường hợp cháy.
  - Kiểm tra dữ liệu hiển thị trên grafana.
- Kết quả đạt được:
  - Dữ liệu cảm biến đọc đúng giá trị: nhiệt độ dao động từ 33°C - 45°C, độ ẩm ổn định từ 50-60%. Cảm biến cháy phát hiện chính xác tín hiệu.



- Dữ liệu CAN nhận được trên ESP32C3 ổn định không phát sinh lỗi.
- Quạt làm mát hoạt động ổn định, đúng logic tại 3 mức.
- Khi phát hiện cháy, máy bơm được kích hoạt với độ trễ khoảng 500ms.
- LCD hiển thị đúng trạng thái hệ thống. Đồng thời dữ liệu được gửi lên grafana hiển thị chính xác.
- Link video demo:  
[https://drive.google.com/file/d/1DI7Vf\\_E91BUE7fKcpmrPxDSzmLeT2q2Z/view?usp=drive\\_link](https://drive.google.com/file/d/1DI7Vf_E91BUE7fKcpmrPxDSzmLeT2q2Z/view?usp=drive_link)
- Link code:  
[https://drive.google.com/drive/folders/1BgqTlentid-AMxhPzUE9zF-0bh3\\_INdi?usp=sharing](https://drive.google.com/drive/folders/1BgqTlentid-AMxhPzUE9zF-0bh3_INdi?usp=sharing)