

**HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG**

**KHOA KỸ THUẬT ĐIỆN TỬ I**

**----****----**

**BÁO CÁO**

**HỆ ĐIỀU HÀNH NHÚNG**

**Đề tài: “ỨNG DỤNG HỆ ĐIỀU HÀNH NHÚNG TRONG HỆ THỐNG GIÁM SÁT NHIỆT ĐỘ, ĐỘ ẨM & ÁNH SÁNG”**

|  |  |
| --- | --- |
| **Giảng viên hướng dẫn:** | **Th.S LƯƠNG CÔNG DUẨN** |
| **Sinh viên thực hiện :** | **HOÀNG QUỐC TOÀN**  **VƯƠNG TUẤN MINH**  **ĐÀO BÁ THỌ**  **TẠ QUANG TRƯỜNG** |
|  | **D21DTMT01** |
| **Khóa :** | **2021 – 2026** |
| **Hệ :** | **ĐẠI HỌC CHÍNH QUY** |

**HÀ NỘI - 5/2025**

NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN

**Điểm**: (bằng chữ:…………………..)

……… , ngày … tháng … năm 2025

**CÁN BỘ - GIẢNG VIÊN HƯỚNG DẪN**

(ký, họ tên)

Lương Công Duẩn

NHẬN XÉT CỦA GIÁO VIÊN PHẢN BIỆN

**Điểm**: (bằng chữ:…………………..)

……… , ngày … tháng … năm 2025

**CÁN BỘ - GIẢNG VIÊN PHẢN BIỆN**

(ký, họ tên)

BẢNG PHÂN CÔNG VIỆC

|  |  |
| --- | --- |
| Tên sinh viên | Công việc |
| Hoàng Quốc Toàn | Viết application,gửi tin lên MQTT broker, viết word và thuyết trình |
| Đào Bá Thọ | Viết driver LED, làm backend và database |
| Tạ Quang Trường | Viết driver DHT11 và làm PP |
| Vương Tuấn Minh | Viết driver BH1750, làm frontend và viết báo cáo |

Link source project: <https://github.com/hoangtoan232003/HDH-BeagleBone-Black.git>

LỜI CẢM ƠN

Để hoàn thành tốt đề tài này, ngoài sự nỗ lực của bản thân, em còn nhận được rất nhiều sự giúp đỡ quý báu từ các thầy cô cùng các bạn.

Chúng em xin chân thành cảm ơn các thầy cô trong khoa Kỹ thuật Điện tử 1 - Học viện Công nghệ Bưu chính Viễn thông đã tận tình giảng dạy và truyền đạt kiến thức cho em trong suốt quá trình học tập tại trường. Những kiến thức và kỹ năng em có được là nhờ sự hướng dẫn nhiệt tình của thầy cô, và đó là nền tảng giúp em hoàn thành đồ án tốt nghiệp này.

Chúng em đặc biệt cảm ơn thầy **Lương Công Duẩn** đã dành thời gian chỉ bảo, hỗ trợ và đưa ra những góp ý quý báu trong quá trình thực hiện đề tài "Ứng dụng hệ điều hành nhúng trong hệ thống giám sát nhiệt độ, độ ẩm & ánh sáng". Những lời khuyên của thầy đã giúp em có định hướng rõ ràng và khắc phục các khó khăn gặp phải.

Chúng em xin trân trọng cảm ơn tất cả những sự giúp đỡ quý báu đó!

Hà Nội, tháng 5 năm 2025

Hoàng Quốc Toàn

Vương Tuấn Minh

Đào Bá Thọ

Tạ Quang Trường

# TÓM TẮT

Hệ điều hành nhúng (Embedded Operating System) là một loại hệ điều hành được thiết kế tối ưu để chạy trên các thiết bị phần cứng có tài nguyên giới hạn, phục vụ cho một mục đích chuyên biệt, chẳng hạn như điều khiển cảm biến, xử lý tín hiệu hay giao tiếp với thiết bị ngoại vi. Khác với hệ điều hành thông thường trên máy tính cá nhân, hệ điều hành nhúng thường nhỏ gọn, thời gian đáp ứng nhanh và tích hợp chặt chẽ với phần cứng. Chúng được ứng dụng rộng rãi trong các lĩnh vực như điện tử tiêu dùng (smart TV, router), ô tô (hệ thống ABS, ECU), công nghiệp (PLC, HMI), thiết bị y tế, và đặc biệt là trong các hệ thống IoT – nơi đòi hỏi thiết bị phải hoạt động ổn định, tiết kiệm năng lượng và có khả năng giao tiếp mạng.

Đề tài tập trung vào việc xây dựng một hệ thống giám sát môi trường sử dụng nền tảng BeagleBone, với khả năng thu thập và xử lý dữ liệu từ các cảm biến nhiệt độ, độ ẩm và ánh sáng. Hệ thống nhằm theo dõi điều kiện môi trường theo thời gian thực và có thể ứng dụng trong các mô hình như trạm quan trắc thời tiết mini, nhà kính thông minh hoặc không gian sống trong nhà. Dữ liệu được xử lý bởi BeagleBone, có thể hiển thị lên màn hình, lưu trữ cục bộ hoặc truyền lên nền tảng web để phục vụ giám sát, cảnh báo, hoặc điều khiển thiết bị tự động như quạt, đèn hoặc bơm nước. Hệ thống hoạt động trên nền tảng Ubuntu Linux được cài đặt trong BeagleBone, cho phép tận dụng sức mạnh mã nguồn mở và dễ dàng lập trình với Python hoặc Shell Script. Việc giao tiếp với các cảm biến được thực hiện thông qua các driver hệ thống đã được hỗ trợ sẵn hoặc tự viết, giúp truy cập các giao diện như GPIO, I2C, UART một cách linh hoạt và hiệu quả. Đây là một mô hình IoT cơ bản nhưng có khả năng mở rộng cao và mang tính ứng dụng thực tế.

MỤC LỤC

[BẢNG PHÂN CÔNG VIỆC 3](#_Toc199930095)

[LỜI CẢM ƠN 4](#_Toc199930096)

[TÓM TẮT 5](#_Toc199930097)

[MỤC LỤC 6](#_Toc199930098)

[MỞ ĐẦU 11](#_Toc199930099)

[Tính cấp thiết của đề tài 11](#_Toc199930100)

[Ý nghĩa khoa học và thực tiễn 11](#_Toc199930101)

[Đối tượng và phương pháp nghiên cứu 12](#_Toc199930102)

[Nội dung nghiên cứu 13](#_Toc199930103)

[CHƯƠNG 1: CƠ SỞ LÝ THUYẾT 14](#_Toc199930104)

[1.1. Giới thiệu Linux Embedded. 14](#_Toc199930105)

[1.2. Giới thiệu Driver và vai trò 14](#_Toc199930106)

[1.3. Ứng dụng công nghệ. 16](#_Toc199930107)

[1.4. Tổng quan kiến trúc và thành phần hệ điều hành nhúng 16](#_Toc199930108)

[1.4.1. Các thành phần của một Embedded OS 16](#_Toc199930109)

[1.4.2. Các thành phần bên trong 17](#_Toc199930110)

[1.5. Embedded Linux Build System 19](#_Toc199930111)

[1.6.1 Giao tiếp I2C. 20](#_Toc199930112)

[1.6.2 Giao thức MQTT. 23](#_Toc199930113)

[1.7. Cơ sở dữ liệu 26](#_Toc199930114)

[1.8. Kết luận. 26](#_Toc199930115)

[CHƯƠNG 2: XÂY DỰNG HỆ THỐNG 28](#_Toc199930116)

[2.1. Thiết kế phần cứng. 28](#_Toc199930117)

[2.2. Phần cứng sử dụng. 29](#_Toc199930118)

[2.2.1. BeagleBone Black 29](#_Toc199930119)

[2.2.2. Một số module khác 30](#_Toc199930120)

[2.3. Cách sử dụng build system với Buildroot 31](#_Toc199930121)

[2.4. Phát triển application cho hệ thống 33](#_Toc199930122)

[2.5. Thiết lập auto service cho hệ thống 34](#_Toc199930123)

[2.6. Thiết kế firmware 35](#_Toc199930124)

[2.7. Cơ sở dữ liệu MySQL 36](#_Toc199930125)

[2.8. Thiết kế và xây dựng hệ thống backend 38](#_Toc199930126)

[2.7. Kết luận. 41](#_Toc199930127)

[CHƯƠNG 3: KẾT QUẢ CHẠY MÔ PHỎNG 43](#_Toc199930128)

**DANH MỤC KÝ HIỆU VÀ CHỮ VIẾT TẮT**

|  |  |  |
| --- | --- | --- |
| **Từ viết tắt** | **Thuật ngữ tiếng anh** | **Thuật ngữ tiếng việt** |
| ACK/NACK | Acknowledged / Not Acknowledged | Đã xác nhận / Chưa xác xhận |
| CSS | Cascading Style Sheets | Bảng kiểu dạng nén |
| DB | DataBase |  |
| EEPROM | Erasable Programmable ReadOnly Memory | Bộ nhớ chỉ đọc có thể lập trình có thể xóa |
| EPROM | Electrically  ErasableProgrammable Read-OnlyMemory | Bộ nhớ chỉ đọc có thể lập trình được xóa điện tử |
| GND | Ground | Điểm nối đất |
| GUI | Graphical User Interface | Giao diện đồ họa người dùng |
| I2C | Inter Integrated Circuit | Giao tiếp tích hợp nối tiếp |
| IC | Integrated circuit | Chip |
| IoT | Internet of Things | Internet kết nối vạn vật |
| LED | Light-Emitting Diode | Diode phát sáng |
| LSB | Least Significant Bit | Bit có trọng số thấp nhất |
| LCD | Liquid Crystal Display | Màn hình tinh thể lỏng |
| M2M | Machine-To-Machine | Máy đến máy |
| MISO | Master In Slave Out | Truyền dữ liệu từ slave đến master |
| MOSI | Master Out - Slave In |  |
| MQTT | Message Queueing TelemetryTransport | Giao thức truyền thông hàng đợi đo lường từ xa |
| MSB | Most Significant Bit | Bit có trong số cao nhất |
| PWM | Pulse Width Modulation | Điều chỉnh Mô-đun độ rộng xung |
| QoS | Quality of Service | Chất lượng dịch vụ |
| ROM | Read Only Memory | Bộ nhớ chỉ đọc |
| RSSI | Received Signal Strength Indicator |  |
| RTC | Real Time control | Điều khiển thời gian thực |
| SNR | Signal-to-Noise Ratio |  |
| SCL | Serial Clock | Tín hiệu xung đồng hồ |
| SDA | Source Data Automation |  |
| SPI | Serial Peripheral Interface | Giao tiếp ngoại vi nối tiếp |
| TCP | Transmission Control Protocol | Giao thức điều khiển truyền vận |

MỞ ĐẦU

## Tính cấp thiết của đề tài

Trong bối cảnh biến đổi khí hậu và đô thị hóa ngày càng gia tăng, nhu cầu giám sát môi trường một cách chính xác và liên tục trở nên ngày càng quan trọng, đặc biệt tại các khu vực nhà kính, không gian sống trong nhà, hoặc các cơ sở sản xuất có yêu cầu kiểm soát nhiệt độ và độ ẩm. Việc xây dựng một hệ thống giám sát nhiệt độ, độ ẩm và ánh sáng sử dụng nền tảng nhúng như BeagleBone giúp cung cấp giải pháp tiết kiệm chi phí, linh hoạt, dễ mở rộng và có khả năng hoạt động ổn định trong thời gian dài. Bên cạnh đó, ứng dụng hệ điều hành Linux nhúng vào hệ thống không chỉ giúp tăng tính linh hoạt trong phát triển phần mềm, mà còn phù hợp với xu hướng chuyển đổi số, nơi mà các thiết bị IoT đóng vai trò thu thập dữ liệu và kết nối thông minh với hệ thống điều khiển trung tâm. Do đó, đề tài không chỉ mang tính thực tiễn cao mà còn phù hợp với xu thế phát triển công nghệ hiện nay, góp phần tạo nền tảng cho các hệ thống tự động hóa, nhà thông minh, và giải pháp môi trường thông minh trong tương lai.

## Ý nghĩa khoa học và thực tiễn

Đề tài góp phần nghiên cứu và ứng dụng hệ điều hành nhúng – một lĩnh vực quan trọng trong kỹ thuật điều khiển và tự động hóa hiện đại. Thông qua việc triển khai hệ thống trên nền tảng BeagleBone sử dụng Ubuntu Linux nhúng, đề tài tạo điều kiện để người học tiếp cận với các khái niệm khoa học như quản lý tiến trình, lập trình driver thiết bị, giao tiếp phần cứng cấp thấp (I2C, UART, GPIO), xử lý tín hiệu từ cảm biến theo thời gian thực, và lập trình điều khiển thiết bị ngoại vi. Đây là những thành phần cốt lõi của một hệ điều hành nhúng hiện đại, giúp làm rõ vai trò của hệ điều hành trong việc tối ưu tài nguyên phần cứng, đảm bảo tính ổn định, độ trễ thấp và khả năng mở rộng của hệ thống.

Về mặt thực tiễn, đề tài hướng đến xây dựng một hệ thống giám sát môi trường có tính ứng dụng cao, dễ triển khai trong thực tế như giám sát nhà kính, phòng máy, kho lạnh, hoặc không gian sống thông minh. Việc sử dụng BeagleBone giúp giảm chi phí, đồng thời tận dụng được các tính năng mạnh mẽ của nền tảng nhúng để thu thập, xử lý và truyền dữ liệu theo thời gian thực. Hệ thống này có khả năng mở rộng, tích hợp thêm điều khiển tự động hoặc kết nối mạng (IoT), phù hợp với xu hướng công nghiệp 4.0 và nhu cầu quản lý môi trường thông minh hiện nay.

## Đối tượng và phương pháp nghiên cứu

* Hệ điều hành nhúng (Embedded Linux), cụ thể là Ubuntu được cài đặt trên nền tảng phần cứng BeagleBone.
* Cách hệ điều hành nhúng giao tiếp với phần cứng: bao gồm cách truy xuất các chân GPIO, giao tiếp I2C/UART với cảm biến.
* Các cảm biến môi trường như nhiệt độ, độ ẩm (DHT11/DHT22), và ánh sáng (BH1750 hoặc tương đương).
* Cơ chế xử lý dữ liệu cảm biến trong hệ điều hành nhúng, bao gồm thu thập, xử lý theo thời gian thực, và hiển thị dữ liệu.
* Lập trình ứng dụng nhúng trên nền tảng Linux: sử dụng ngôn ngữ C phát triển ứng dụng để thao tác với phần cứng và xây dựng ứng dụng giám sát.

**Phương pháp nghiên cứu**

* **Tổng quan tài liệu và khảo sát công nghệ**: Nghiên cứu các khái niệm về hệ điều hành nhúng, đặc biệt là Ubuntu Linux trên BeagleBone, tìm hiểu về các giao thức giao tiếp phần cứng như GPIO, I2C, UART, và các loại cảm biến nhiệt độ, độ ẩm, ánh sáng phổ biến.
* **Lắp ráp và thiết kế phần cứng**: Kết nối các cảm biến đo nhiệt độ, độ ẩm và ánh sáng với BeagleBone qua các giao tiếp phù hợp (I2C, GPIO, UART), đảm bảo kết nối điện và truyền tín hiệu ổn định.
* **Phát triển phần mềm nhúng**: Viết driver hoặc sử dụng driver có sẵn để giao tiếp với cảm biến trên nền Ubuntu Linux, lập trình thu thập dữ liệu theo thời gian thực bằng Python hoặc C/C++, xử lý và lưu trữ dữ liệu, đồng thời xây dựng giao diện hiển thị hoặc truyền dữ liệu lên hệ thống giám sát từ xa.
* **Kiểm thử và hiệu chỉnh hệ thống**: Thử nghiệm hệ thống trong các điều kiện môi trường khác nhau, kiểm tra độ chính xác của dữ liệu thu thập, hiệu suất xử lý, độ ổn định của hệ thống và khả năng hoạt động liên tục.
* **Phân tích và đánh giá kết quả**: So sánh dữ liệu thực tế với các thiết bị đo chuẩn, đánh giá hiệu quả của hệ thống, xác định ưu điểm, hạn chế và đề xuất hướng cải tiến.
* **Tổng hợp và báo cáo**: Thu thập tài liệu, phân tích kết quả nghiên cứu và viết báo cáo hoàn chỉnh về đề tài.

## Nội dung nghiên cứu

Hệ thống giám sát môi trường sử dụng BeagleBone là một mô hình nhúng được xây dựng để đo và theo dõi các thông số môi trường như nhiệt độ, độ ẩm và ánh sáng theo thời gian thực. Hệ thống bao gồm 3 phần chính: phần cứng, phần mềm và giao diện hiển thị.

* Phần cứng: BeagleBone đóng vai trò là bộ xử lý trung tâm, kết nối với các cảm biến môi trường (DHT11 để đo nhiệt độ, độ ẩm; BH1750 để đo ánh sáng). Các cảm biến này giao tiếp với BeagleBone thông qua các giao thức phổ biến như GPIO, I2C hoặc UART.
* Phần mềm: Ubuntu Linux (hệ điều hành nhúng) được cài đặt trên BeagleBone để quản lý tài nguyên và điều khiển các tiến trình. Người dùng lập trình bằng Python hoặc C để tạo các đoạn mã đọc dữ liệu từ cảm biến, xử lý thông tin, và có thể hiển thị ra màn hình, lưu trữ vào file hoặc truyền qua mạng.

Sau khi khởi động, hệ điều hành Ubuntu trên BeagleBone sẽ tự động chạy chương trình giám sát.

* Các cảm biến gửi dữ liệu môi trường về theo chu kỳ định sẵn (ví dụ 1 lần/giây).
* Chương trình nhận dữ liệu, xử lý và hiển thị kết quả lên terminal hoặc giao diện đồ họa (nếu có).
* Tùy theo thiết kế, hệ thống có thể lưu dữ liệu vào bộ nhớ, gửi cảnh báo khi vượt ngưỡng hoặc điều khiển thiết bị như quạt, đèn.
* Tính năng mở rộng: Hệ thống có thể phát triển thêm các tính năng như gửi dữ liệu qua mạng LAN/Wi-Fi, tích hợp MQTT để đẩy dữ liệu lên cloud, hoặc thêm giao diện web để giám sát từ xa.

# CƠ SỞ LÝ THUYẾT

## 1.1. Giới thiệu Linux Embedded.

Linux Embedded (Linux nhúng) là phiên bản tùy chỉnh của hệ điều hành Linux được tối ưu hóa để chạy trên các thiết bị phần cứng có tài nguyên hạn chế như vi điều khiển, vi xử lý nhúng (như ARM Cortex-A), thường không có ổ cứng, RAM nhỏ và yêu cầu thời gian thực.

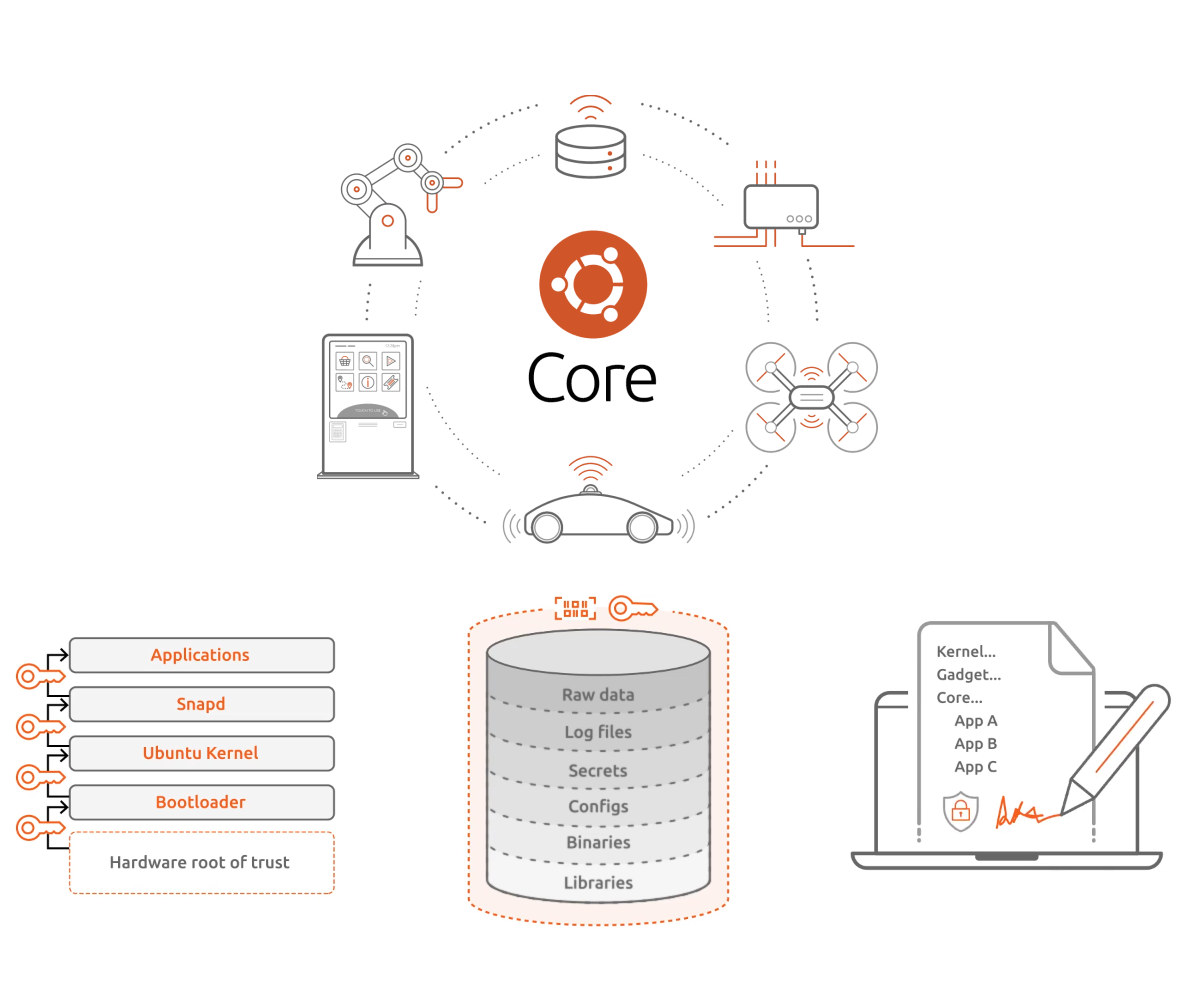
Nó cung cấp đầy đủ các chức năng như quản lý tiến trình, driver thiết bị, giao tiếp phần cứng và hỗ trợ lập trình mạnh mẽ (C, Python...), rất phù hợp cho các ứng dụng trong IoT, tự động hóa, thiết bị y tế, robot và hệ thống nhúng công nghiệp. Linux nhúng thường được dùng trên các nền tảng như BeagleBone, Raspberry Pi, hoặc các SoC chuyên dụng.

## 1.2. Giới thiệu Driver và vai trò

Trong hệ điều hành Linux nhúng, driver thiết bị (device driver) là một thành phần quan trọng thuộc tầng nhân (kernel), đóng vai trò trung gian giữa phần cứng và phần mềm. Driver giúp hệ điều hành "hiểu" cách giao tiếp và điều khiển từng loại thiết bị cụ thể như cảm biến, LED, UART, I2C, SPI, v.v. mà phần cứng cung cấp.

Khi người dùng viết chương trình ở không gian người dùng (user space) để đọc dữ liệu cảm biến hoặc điều khiển thiết bị, chương trình đó không tương tác trực tiếp với phần cứng, mà sẽ gọi đến driver tương ứng trong kernel space. Driver sẽ thực hiện các tác vụ cấp thấp như đọc/ghi thanh ghi, cấu hình chân GPIO, hoặc truyền dữ liệu qua bus giao tiếp. Nhờ có tầng driver, hệ điều hành đảm bảo tính trừu tượng phần cứng, dễ bảo trì và mở rộng khi thay đổi thiết bị.

Trong phát triển hệ thống nhúng, đặc biệt trên nền tảng như BeagleBone, việc viết hoặc tùy chỉnh driver giúp người phát triển làm chủ hoàn toàn quá trình giao tiếp phần cứng – từ điều khiển LED, đọc cảm biến đến xây dựng hệ thống I/O tùy biến theo yêu cầu thực tế.



Hình 1 Linux Embedded trong hệ thống nhúng

Một số đặc điểm nổi bật của Linux Embedded gồm:

* Mã nguồn mở, tùy biến linh hoạt: Cho phép thay đổi, tối ưu kernel, driver hoặc thư viện hệ thống để phù hợp với phần cứng và ứng dụng cụ thể.
* Hỗ trợ mạnh mẽ phần cứng: Tích hợp sẵn rất nhiều driver cho các chuẩn giao tiếp như GPIO, I2C, SPI, UART, USB…, dễ dàng tương thích với nhiều loại cảm biến và thiết bị ngoại vi.
* Khả năng phát triển driver thiết bị riêng: Cho phép lập trình viên viết driver để điều khiển các phần cứng đặc biệt chưa được hỗ trợ sẵn.
* Chạy đa nhiệm, quản lý tiến trình tốt: Có thể xử lý nhiều tác vụ đồng thời: đọc cảm biến, ghi dữ liệu, hiển thị giao diện… một cách mượt mà.
* Hỗ trợ nhiều ngôn ngữ lập trình: Phát triển ứng dụng dễ dàng bằng C, C++, Python, Shell script,…
* Cộng đồng lớn, tài liệu phong phú: Dễ học, dễ tra cứu, được dùng rộng rãi trong giáo dục và công nghiệp.

Nhờ các đặc điểm trên, Linux Embedded là lựa chọn lý tưởng để xây dựng các hệ thống nhúng hiện đại như giám sát môi trường, điều khiển công nghiệp, thiết bị IoT thông minh.

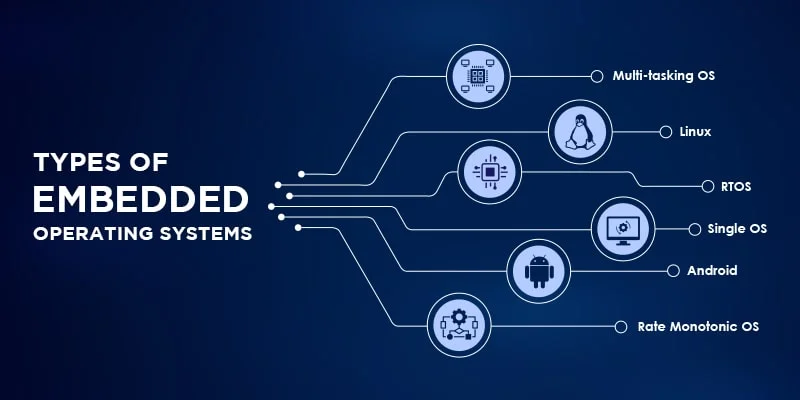
## 1.3. Ứng dụng công nghệ.

* Hệ điều hành nhúng (Embedded OS) đóng vai trò cốt lõi trong việc vận hành các thiết bị thông minh và hệ thống nhúng hiện đại. Nhờ khả năng quản lý tài nguyên hiệu quả, hỗ trợ giao tiếp phần cứng và khả năng xử lý thời gian thực, hệ điều hành nhúng được ứng dụng rộng rãi trong nhiều lĩnh vực như:
* Điện tử tiêu dùng: Smart TV, router Wi-Fi, máy ảnh kỹ thuật số, loa thông minh,...
* Tự động hóa công nghiệp: PLC, hệ thống SCADA, bộ điều khiển dây chuyền sản xuất,...Ô tô và giao thông: Hệ thống điều khiển động cơ, camera lùi, màn hình giải trí trên ô tô,...Y tế: Máy đo nhịp tim, thiết bị theo dõi bệnh nhân,...
* Internet of Things (IoT): Nhà thông minh, nông nghiệp thông minh, hệ thống giám sát môi trường,...
* Hệ điều hành nhúng (cụ thể là Linux Embedded chạy trên BeagleBone) giúp quản lý cảm biến nhiệt độ, độ ẩm và ánh sáng, điều phối quá trình đọc dữ liệu, xử lý và hiển thị thông tin theo thời gian thực. Việc sử dụng hệ điều hành nhúng giúp hệ thống hoạt động ổn định, dễ mở rộng, và có khả năng giao tiếp với các thiết bị hoặc nền tảng IoT khác.

## 1.4. Tổng quan kiến trúc và thành phần hệ điều hành nhúng

### 1.4.1. Các thành phần của một Embedded OS

Hệ điều hành nhúng là một loại hệ điều hành được thiết kế tối ưu và gọn nhẹ, chuyên dùng cho các thiết bị nhúng có tài nguyên phần cứng hạn chế (CPU yếu, RAM ít, dung lượng thấp). Ví dụ: BeagleBone Black, STM32, ESP32...



Hình 2 Tổng quan một EOS

### 1.4.2. Các thành phần bên trong

|  |  |
| --- | --- |
| **Thành phần** | **Mô tả** |
| **Kernel** | Nhân hệ điều hành – xử lý lịch trình, ngắt, quản lý bộ nhớ |
| **Driver** | Giao tiếp phần cứng (I2C, SPI, GPIO...) |
| **File System** | Quản lý dữ liệu lưu trữ (EXT4, FAT32...) |
| **Networking Stack** | Hỗ trợ giao tiếp mạng (TCP/IP, MQTT, BLE...) |
| **Libraries** | Thư viện lập trình hệ thống và ứng dụng |

**1.4.3. Board Support Package**

BSP là một tập hợp phần mềm trung gian nằm giữa phần cứng và hệ điều hành, giúp hệ điều hành có thể khởi động và hoạt động đúng trên một phần cứng cụ thể.

|  |  |
| --- | --- |
| **Thành phần** | **Mô tả** |
| **Bootloader** | Nạp và khởi động kernel (thường dùng: U-Boot) |
| **Device Tree** | Mô tả cấu trúc phần cứng (CPU, RAM, thiết bị ngoại vi) cho kernel Linux |
| **Linux Kernel Config** | Cấu hình các tính năng và driver cần thiết cho phần cứng cụ thể |
| **Device Drivers** | Trình điều khiển cho từng phần cứng: LED, nút nhấn, cảm biến... |
| **Toolchain** | Bộ biên dịch chéo giúp build code trên PC nhưng chạy trên board ARM |

**1.4.4. Embedded Middleware Layer (EML) – Kiến trúc phần mềm nhúng**

Mục tiêu là tách biệt ứng dụng khỏi phần cứng để dễ phát triển, bảo trì và mở rộng. Cho phép một ứng dụng có thể chạy trên nhiều nền tảng phần cứng khác nhau nếu có tầng EML phù hợp.

Cấu trúc phân tầng gồm:

[ Application Layer ] ← tầng viết app, giao diện người dùng

[ Middleware Layer ] ← các stack giao tiếp: TCP/IP, MQTT, GUI API,

[ OS Abstraction Layer (OAL) ] ← lớp trừu tượng hóa hệ điều hành

[ Device Drivers ] ← giao tiếp trực tiếp phần cứng

[ Hardware ] ← thiết bị thật: cảm biến, cổng I/O, MCU/SoC

|  |  |
| --- | --- |
| **Tầng** | **Vai trò** |
| **Application Layer** | Tầng viết chương trình logic chính (giao diện, xử lý dữ liệu, thuật toán) |
| **Middleware Layer** | Cung cấp API sẵn dùng (giao tiếp mạng, sensor, hiển thị...) |
| **OAL** | Che giấu chi tiết hệ điều hành cụ thể, giúp viết app dễ dàng port sang nền tảng khác |
| **Driver Layer** | Điều khiển phần cứng cụ thể thông qua API dưới kernel hoặc user space |
| **Hardware** | Thiết bị thật như BBB, STM32, cảm biến, màn hình... |

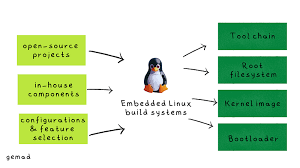
## 1.5. Embedded Linux Build System

Đây là **hệ thống xây dựng (build system)** chuyên dụng để:

* Tạo ra **bản phân phối Linux nhúng tùy chỉnh** (custom embedded Linux distro).
* Bao gồm kernel, driver, root filesystem, bootloader, thư viện, ứng dụng, cấu hình...
* Đặc biệt tối ưu cho các **thiết bị tài nguyên hạn chế** như: BeagleBone, Raspberry Pi, i.MX6, STM32MP1...

Một số hệ thống build phổ biến như:

* Yocto Project: Mạnh mẽ, linh hoạt, dùng cho sản phẩm công nghiệp. Có meta-layer, hỗ trợ cross-build rất tốt.
* Buildroot: Nhẹ, dễ dùng, đơn giản. Phù hợp với embedded nhỏ, thời gian build nhanh.
* OpenWrt: Tối ưu cho thiết bị mạng (router, AP). Có sẵn GUI quản lý qua web.



Hình 3 Cấu tạo cơ bản của build system

**1.6. Các giao thức**

**1.6.1 Giao tiếp I2C.**

*Cấu tạo:*

A diagram of a computer network

Description automatically generatedI2C (Inter – Integrated Circuit) là một giao thức giao tiếp nối tiếp đồng bộ được phát triển bởi Philips Semiconductors, sử dụng để truyền nhận dữ liệu giữa các IC với nhau chỉ sử dụng hai đường truyền tín hiệu.

Hình 4 Giao tiếp I2C

Các bit dữ liệu sẽ được truyền từng bit một theo các khoảng thời gian đều đặn được thiết lập bởi 1 tín hiệu đồng hồ. Bus I2C thường được sử dụng để giao tiếp ngoại vi cho rất nhiều loại IC khác nhau như các loại vi điều khiển, cảm biến, EEPROM.

Từ hình 1.10 cho ta thấy cấu tạo của giao tiếp I2C, sử dụng 2 đường truyền tín hiệu:

* SCL - Serial Clock Line: Tạo xung nhịp đồng hồ do master phát đi.
* SDA - Serial Data Line: Đường truyền nhận dữ liệu.

Giao tiếp I2C bao gồm quá trình truyền nhận dữ liệu giữa các thiết bị chủ tớ, hay Master - Slave. Thiết bị master là một vi điều khiển, nó có nhiệm vụ điều khiển đường tín hiệu SCL và gửi nhận dữ liệu hay lệnh thông qua đường SDA đến các thiết bị khác. Các thiết bị nhận các dữ liệu lệnh và tín hiệu từ thiết bị Master được gọi là các thiết bị Slave. Các thiết bị Slave thường là các IC, hoặc thậm chí là vi điều khiển. Master và Slave được kết nối với nhau như hình trên. Hai đường bus SCL và SDA đều hoạt động ở chế độ Open Drain, nghĩa là bất cứ thiết bị nào kết nối với mạng I2C này cũng chỉ có thể kéo 2 đường bus này xuống mức thấp (LOW), nhưng lại không thể kéo được lên mức cao. Vì để tránh trường hợp bus vừa bị 1 thiết bị kéo lên mức cao vừa bị 1 thiết bị khác kéo xuống mức thấp gây hiện tượng ngắn mạch. Do đó cần có 1 điện trờ (từ 1 – 4,7 kΩ) để giữ mặc định ở mức cao.

*Khung truyền I2C:*

Khung truyền của I2C bao gồm 7 khối, mỗi khối thực hiện một chức năng riêng biệt như sau:

A white rectangular object with black text

Description automatically generated

Hình 5 Khung truyền I2C

* Khối bit địa chỉ: Thông thường quá trình truyền nhận sẽ diễn ra với rất nhiều thiết bị, IC với nhau. Do đó để phân biệt các thiết bị này, chúng sẽ được gắn 1 địa chỉ vật lý 7-bit cố định.
* Bit Read/Write: Bit này dùng để xác định quá trình là truyền hay nhận dữ liệu từ thiết bị Master. Nếu Master gửi dữ liệu đi thì ứng với bit này bằng ‘0’, và ngược lại, nhận dữ liệu khi bit này bằng ‘1’.
* Bit ACK/NACK: Viết tắt của Acknowledged / Not Acknowledged. Dùng để so sánh bit địa chỉ vật lý của thiết bị so với địa chỉ được gửi tới. Nếu trùng thì Slave sẽ được đặt bằng ‘0’ và ngược lại, nếu không thì mặc định bằng ‘1’.
* Khối bit dữ liệu: Gồm 8-bit và được thiết lập bởi thiết bị gửi truyền đến thiết bị nhân. Sau khi các bit này được gửi đi, lập tức 1-bit ACK/NACK được gửi ngay theo sau để xác nhận rằng thiết bị nhận đã nhận được dữ liệu thành công hay chưa. Nếu nhận thành công thì bit ACK/NACK được set bằng ‘0’ và ngược lại.

*Quá trình truyền nhận dữ liệu:*

Quá trình giao tiếp I2C bắt đầu khi thiết bị Master gửi một xung Start bằng cách lần lượt kéo các đường SDA và SCL từ mức cao xuống mức thấp. Sau đó, Master tiếp tục gửi 7-bit địa chỉ của thiết bị Slave mà nó muốn giao tiếp, kèm theo một bit xác định chế độ Read/Write. Khi nhận được địa chỉ, Slave sẽ so sánh với địa chỉ vật lý của mình; nếu trùng khớp, Slave xác nhận bằng cách kéo đường SDA xuống mức thấp và set bit ACK/NACK thành '0'. Nếu địa chỉ không khớp, đường SDA và bit ACK/NACK sẽ giữ ở mức cao.

Tiếp theo, Master sẽ truyền hoặc nhận khung dữ liệu. Nếu Master gửi dữ liệu đến Slave, bit Read/Write sẽ ở mức '0'; nếu Master nhận dữ liệu từ Slave, bit này sẽ ở mức '1'. Khi một khung dữ liệu được truyền thành công, bit ACK/NACK sẽ được set ở mức '0' để báo hiệu cho Master tiếp tục. Cuối cùng, khi toàn bộ dữ liệu đã được truyền đi, Master phát tín hiệu Stop để báo hiệu quá trình truyền kết thúc, bằng cách lần lượt chuyển SCL và SDA từ mức thấp lên mức cao.

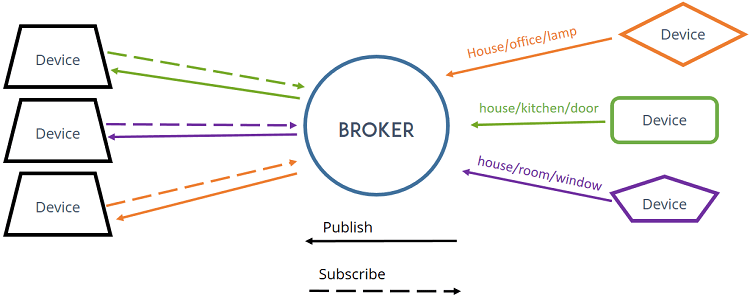
*Các cơ chế hoạt động của I2C:*

Chế độ chuẩn (standard mode) với tốc độ 100 kBit/s và chế độ tốc độ thấp (low speed mode) với tốc độ 10 kBit/s.

Ngoài ra, khác với giao tiếp SPI chỉ có thể có 1 Master, giao tiếp I2C cho phép chế độ truyền nhận dữ liệu giữa nhiều thiết bị Master khác nhau với thiết bị Slave. Tuy nhiên quá trình này có hơi phức tạp vì thiết bị Slave có thể nhận một lúc nhiều khung dữ liệu từ các thiết bị Master khác nhau, điều đó đôi khi dẫn đến xung đột hoặc sai sót dữ liệu nhận được. Để tránh điều đó, khi làm việc ở chế độ này, mỗi thiết bị Master cần phát hiện xem đường SDA đang ở trạng thái nào. Nếu SDA ở mức 0, nghĩa là đang có 1 thiết bị Master khác đang có quyền điều khiển và phải chờ đến khi truyền xong. Ngược lại nếu SDA ở mức 1, nghĩa là đường truyền SDA đã an toàn và có sử dụng.

**1.6.2 Giao thức MQTT.**

MQTT (Message Queueing Telemetry Transport) là một giao thức mạng kích thước nhỏ (lightweight), hoạt động theo cơ chế publish – subscribe theo tiêu chuẩn ISO (ISO/IEC 20922) và OASIS mở để truyền tin nhắn giữa các thiết bị.



Hình 6 Giao thức MQTT

Giao thức này hoạt động trên nền tảng TCP/IP. MQTT được thiết kế cho các kết nối cho việc truyền tải dữ liệu cho các thiết bị ở xa, các thiết bị hay vi điều khiển nhỏ có tài nguyên hạn chế hoặc trong các ứng dụng có băng thông mạng bị hạn chế.

MQTT là lựa chọn lý tưởng trong các môi trường như:

* Những nơi mà giá mạng viễn thông đắt đỏ hoặc băng thông thấp hay thiếu tin cậy.
* Khi chạy trên thiết bị nhúng bị giới hạn về tài nguyên tốc độ và bộ nhớ.
* Bởi vì giao thức này sử dụng băng thông thấp trong môi trường có độ trễ cao nên nó là một giao thức lý tưởng cho các ứng dụng M2M (Machine to Machine).

1. *Tính năng, đặc điểm nổi bật.*

Tính năng và đặc điểm nổi bật của giao thức MQTT thể hiện sự linh hoạt và hiệu quả trong truyền thông dữ liệu. Giao thức này áp dụng mô hình truyền thông điệp Pub/Sub, cho phép việc truyền tin phân tán một chiều và tách biệt với phần ứng dụng, giúp đơn giản hóa quá trình xử lý. Việc truyền thông điệp diễn ra ngay lập tức mà không cần quan tâm đến nội dung được truyền, đồng thời sử dụng TCP/IP làm giao thức nền để đảm bảo tính ổn định và đáng tin cậy.

Đặc biệt, MQTT hỗ trợ ba mức độ tin cậy (QoS: Quality of Service) cho việc truyền dữ liệu. Ở *QoS 0*, dữ liệu được gửi đúng một lần và xác nhận thông qua giao thức TCP/IP. Với *QoS 1*, dữ liệu được gửi kèm ít nhất một lần xác nhận từ đầu nhận, có thể dẫn đến việc gửi lại nhiều lần. Trong khi đó, *QoS 2* đảm bảo dữ liệu được gửi và nhận đúng một lần duy nhất, thông qua quá trình bắt tay gồm 4 bước. Ngoài ra, phần bao bọc dữ liệu của MQTT được thiết kế nhỏ gọn và tối ưu hóa để giảm tải cho đường truyền, phù hợp với các ứng dụng yêu cầu băng thông thấp và hiệu quả cao.

1. *Ưu điểm của giao thức MQTT.*

Giao thức MQTT mang lại nhiều ưu điểm vượt trội, giúp cải thiện hiệu suất và tính hiệu quả trong truyền thông dữ liệu. Một số ưu điểm đáng chú ý bao gồm khả năng truyền thông tin hiệu quả, tăng cường khả năng mở rộng, và giảm tiêu thụ băng thông mạng. Giao thức này tối đa hóa băng thông có sẵn, đi kèm chi phí thấp nhưng vẫn đảm bảo an toàn và bảo mật cao. Ngoài ra, MQTT giúp tiết kiệm thời gian phát triển và với mô hình publish/subscribe, nó có thể thu thập nhiều dữ liệu hơn trong khi sử dụng ít băng thông hơn so với các giao thức truyền thống.

1. *Mô hình Publish/Subscribe trong giao thức MQTT.*

Thành phần mô hình Publisher/Subscriber trong giao thức MQTT bao gồm: MQTT Broker & MQTT Client

* ***MQTT Broker***

MQTT Broker hay máy chủ mô giới được coi như trung tâm, nó là điểm giao của tất cả các kết nối đến từ Client (Publisher/Subscriber).

|  |  |
| --- | --- |
| Nhiệm vụ chính của Broker là nhận thông điệp (message) từ Publisher, xếp vào hàng đợi rồi chuyển đến một địa điểm cụ thể. Nhiệm vụ phụ của Broker là nó có thể đảm nhận thêm một vài tính năng liên quan tới quá trình truyền thông như: bảo mật message, lưu trữ message, logs, …. | A diagram of a bee  Description automatically generated  Hình 7 Mô hình Publish/Subscriber |

MQTT Broker được cung cấp dưới dạng mã nguồn mở hoặc các phiên bản thương mại giúp người dùng có thể tự cài đặt và tạo broker riêng. Ngoài ra cũng có thể sử dụng Broker trên điện toán đám mây với các nền tảng IOT như hive broker, amazone….

* ***MQTT Client***

Là các thiết bị/ứng dụng Client kết nối đến Broker để thực hiện truyền nhận dữ liệu. Client thì được chia thành hai nhóm là Publisher và Subscriber. Một Client có thể có 1 trong 2 nhiệm vụ hoặc cả 2.

Publisher là thiết bị gửi bản tin lên broker

Subscriber là người nhận bản tin mỗi khi có bản tin mới gửi lên Broker.

1. *Cơ chế hoạt động của giao thức MQTT*

Cơ chế hoạt động của giao thức MQTT bao gồm bốn giai đoạn chính: kết nối, xác thực, giao tiếp và kết thúc. Quá trình bắt đầu khi Client (máy khách) tạo kết nối TCP/IP tới broker thông qua cổng tiêu chuẩn hoặc cổng tùy chỉnh, mà các nhà phát triển broker xác định. Cổng tiêu chuẩn cho giao tiếp không mã hóa là 1883 và cổng 8883 dành cho giao tiếp được mã hóa qua SSL/TLS. Trong quá trình giao tiếp qua SSL/TLS, máy khách cần thực hiện kiểm chứng và xác thực máy chủ để đảm bảo an toàn.

Tiếp theo, nếu máy khách là Publisher, nó sẽ gửi bản tin lên broker; ngược lại, nếu là Subscriber, nó nhận bản tin từ broker. Quá trình kết nối giữa Client và broker sẽ được duy trì cho đến khi kết thúc kết nối. Sau khi kết thúc, quá trình truyền nhận MQTT có thể bắt đầu lại từ các bước trên, tiếp tục hỗ trợ việc truyền thông tin giữa các thiết bị.

## 1.7. Cơ sở dữ liệu

Cơ sở dữ liệu (CSDL) là một tập hợp có tổ chức của dữ liệu được lưu trữ dưới dạng số, nhằm phục vụ việc truy xuất, quản lý và cập nhật một cách hiệu quả. Trong các ứng dụng hệ thống nhúng, cơ sở dữ liệu đóng vai trò cực kỳ quan trọng trong việc lưu trữ thông tin thu thập từ các thiết bị phần cứng như cảm biến. Dữ liệu này sau đó được phân tích để đưa ra các hành động phù hợp, giúp hệ thống có khả năng phản hồi thông minh với môi trường thực tế.

Trong đồ án này, hệ thống được thiết kế để thu thập dữ liệu từ các cảm biến (nhiệt độ, độ ẩm, ánh sáng), truyền dữ liệu qua giao thức MQTT, xử lý bằng phần mềm backend viết bằng Python, và lưu trữ trong cơ sở dữ liệu MySQL để dễ dàng truy xuất và phân tích sau này.

MySQL là một hệ quản trị cơ sở dữ liệu quan hệ (Relational Database Management System – RDBMS) mã nguồn mở được sử dụng rộng rãi trong công nghiệp cũng như trong các dự án nghiên cứu, học tập. MySQL sử dụng ngôn ngữ SQL (Structured Query Language) để thực hiện các thao tác như tạo bảng, thêm/sửa/xoá dữ liệu và truy vấn thông tin.

Trong đồ án này, em sử dụng 2 thành phần chính của MySQL:

* **MySQL Server**: là thành phần chính của hệ thống quản trị cơ sở dữ liệu, chạy dưới dạng dịch vụ nền (service) trên máy tính, chịu trách nhiệm lưu trữ, quản lý và xử lý các truy vấn từ phía ứng dụng.
* **MySQL Workbench**: là công cụ giao diện đồ họa hỗ trợ quản lý cơ sở dữ liệu, cho phép thiết kế sơ đồ dữ liệu (ERD), thực hiện các truy vấn SQL, kiểm tra dữ liệu một cách trực quan.

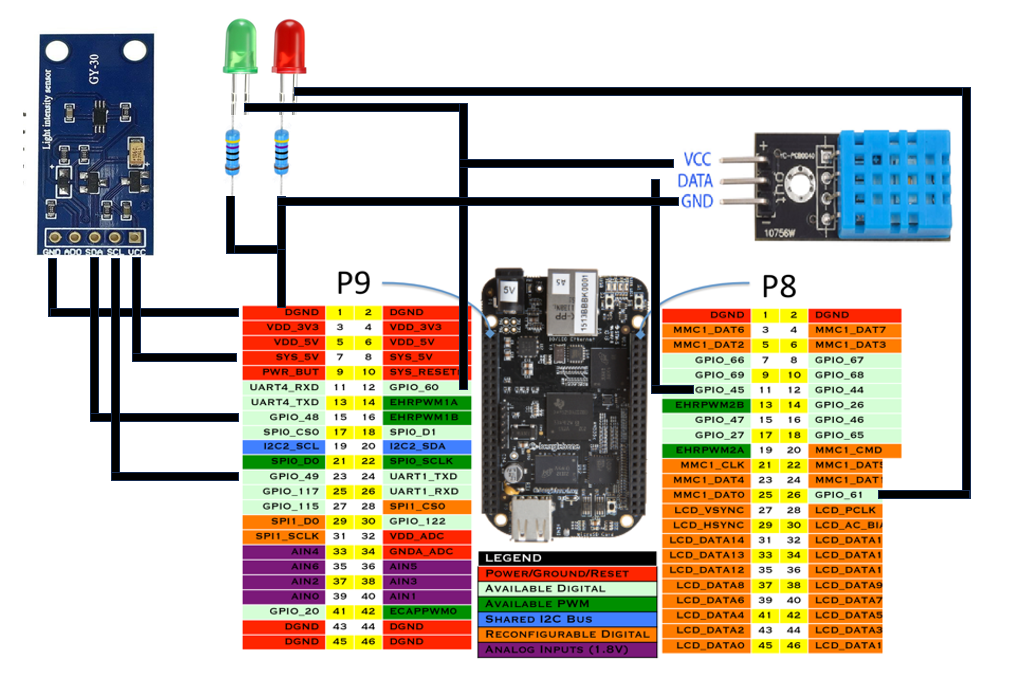
## 1.8. Kết luận.

Chương 1 đã tập trung nghiên cứu cơ sở lý thuyết về công nghệ LoRa, vi điều khiển và các giao thức truyền nhận, đồng thời trình bày tổng quan về các cơ sở lý thuyết được sử dụng trong đồ án này. Những nội dung này là nền tảng quan trọng cho việc triển khai hệ thống. Trong chương tiếp theo, em sẽ tiếp tục với quá trình thiết kế phần cứng và phần mềm, đồng thời xây dựng ý tưởng cụ thể để tạo nên một hệ thống hoàn chỉnh.

# XÂY DỰNG HỆ THỐNG

Sau khi nghiên cứu và tìm hiểu tổng quan về cơ sở lý thuyết cũng như các kiến thức căn bản ở CHƯƠNG 1. Chương này sẽ chủ yếu tìm hiểu về thiết kế, xây dựng hệ thống phần cứng và hệ thống phần mềm, các thành phần sử dụng để thực hiện đề tài.

## 2.1. Thiết kế phần cứng.



Hình 8 Sơ đồ kết nối chân các thiết bị

* **Khối xử lý trung tâm – BeagleBone Black:** Là thiết bị nhúng chạy hệ điều hành Linux (Debian/Ubuntu). Đóng vai trò thu thập dữ liệu từ các cảm biến, xử lý và truyền tải. Hỗ trợ các giao tiếp phổ biến như GPIO, I2C, UART, SPI.
* **Khối cảm biến (bao gồm nhiệt độ và độ ẩm DHT11, Cảm biến ánh sáng BH1750:** Giao tiếp qua 1 wire (single digital pin) cho phép đo nhiệt độ và độ ẩm môi trường theo thời gian thực & giao tiếp qua I2C đo cường độ ánh sáng môi trường (đơn vị: lux), phù hợp trong giám sát môi trường.
* **Khối điều khiển – LED:** Điều khiển thông qua GPIO output của BeagleBone.
* Có thể thiết lập điều kiện bật/tắt theo ngưỡng nhiệt độ, độ ẩm, ánh sáng...
* **Khối truyền thông – MQTT:** Giao tiếp nhẹ, phù hợp với hệ thống IoT.
* BeagleBone đóng vai trò **MQTT publisher**, đẩy dữ liệu cảm biến định kỳ lên broker (VD: Mosquitto). WebServer sẽ đóng vai trò **subscriber**, nhận và hiển thị dữ liệu theo thời gian thực. A diagram of a computer network

  AI-generated content may be incorrect.

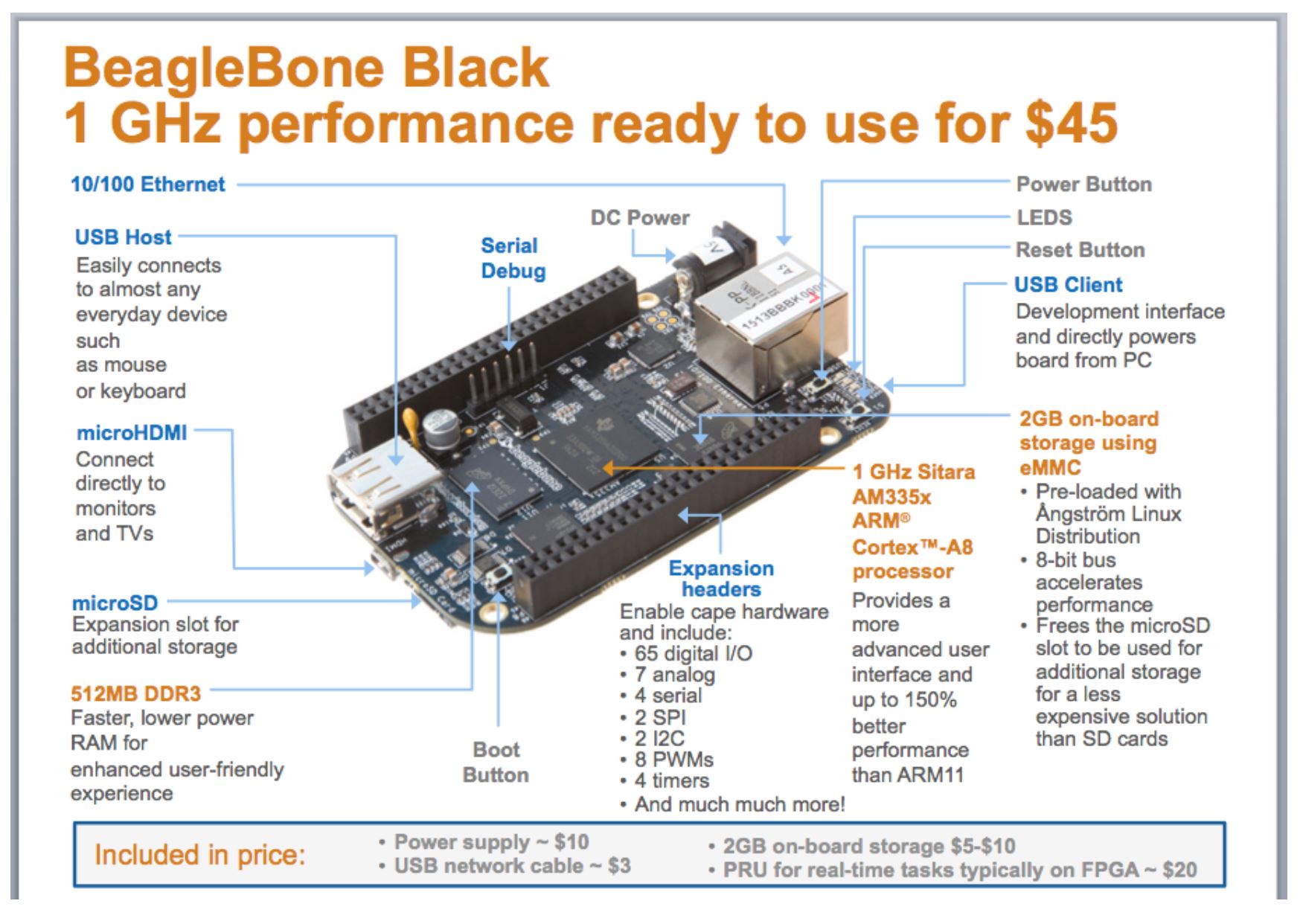
Hình 9 Ý tưởng hệ thống

## 2.2. Phần cứng sử dụng.

**2.2.1. BeagleBone Black**

[BeagleBone Black Rev.C](https://www.thegioiic.com/beaglebone-black-rev-c-) là board phát triển dựa trên bộ vi xử lý AM335x tích hợp lõi ARM Cortex-A8 tốc độ 1GHz và có nhiều thiết bị ngoại vi. BB Black cung cấp rất nhiều giao tiếp mở rộng như Ethernet, USB host và OTG, TF card, serial, JTAG (không có kết nối theo mặc định), HDMI D, Emmc, ADC, I2C, SPI, PWM và LCD.

Board phát triển này được thiết kế để đáp ứng các yêu cầu khác nhau như thiết bị chơi game, gia đình và tự động hóa công nghiệp, thiết bị y tế tiêu dùng, máy in, hệ thống tolling thông minh, hệ thống cân của máy bán hàng thông minh, thiết bị đầu cuối giáo dục và đồ chơi cao cấp.



Hình 10 BeagleBone Black

**Tính năng nổi bật:**

* Có bộ nhớ eMMC để chạy hệ điều hành độc lập không cần thẻ nhớ ngoài.
* Hỗ trợ cài đặt Linux Embedded đầy đủ với kernel tùy chỉnh.
* Đa dạng giao tiếp ngoại vi để phát triển hệ thống nhúng và IoT.
* Có cộng đồng hỗ trợ mạnh và hệ sinh thái phần mềm mở rộng.

Thông tin chi tiết về BeagleBone Black (Phụ lục 2)

**2.2.2. Một số module khác**

|  |  |  |
| --- | --- | --- |
| Cảm Biến Cường Độ Ánh Sáng Lux BH1750 - Nshop Hình 11.1 Module đo nhiệt độ BH1750 | Module Cảm Biến Độ Ẩm, Nhiệt Độ DHT11 Hình 11.2 DHT11. | LM393 Cảm Biến Âm Thanh V1, Nguồn:3.3-5VDC  Hình 11.3 Cảm biến âm thanh. |

Cảm biến ánh sáng BH1750 là cảm biến đo ánh sáng kỹ thuật số, sử dụng một quang điốt nhạy sáng bên trong để đo cường độ ánh sáng môi trường theo đơn vị lux (đơn vị chuẩn đo độ sáng mà mắt người cảm nhận được), sử dụng giao tiếp I2C (gồm 2 chân: SDA, SCL), rất tiện để kết nối với ESP32, Arduino, STM32,...

Cảm biến nhiệt độ - độ ẩm DHT11 là linh kiện tích hợp thermistor để đo nhiệt độ, cảm biến điện dung để đo độ ẩm, vi điều khiển nhỏ để xử lý dữ liệu và gửi qua 1 chân duy nhất.

Là cảm biến dùng microphone điện dung nhỏ để bắt âm thanh môi trường.  
Mạch tích hợp bộ khuếch đại và IC so sánh điện áp LM393 để xuất ra mức logic (0 hoặc 1) khi âm thanh vượt ngưỡng. Digital output (DO): mức HIGH/LOW khi có tiếng động. Một số loại có thêm Analog output (AO) nếu cần đọc cường độ âm thanh.

## 2.3. Cách sử dụng build system với Buildroot

Buildroot là một công cụ mã nguồn mở giúp bạn tạo nhanh một hệ điều hành nhúng Linux tùy chỉnh, bao gồm:

* Target Options:
  + Target Architecture: ARM (little endian).
  + Target Architecture Variant: Cortex-A8 (AM335x).
  + Target ABI: EABIhf (hiệu quả hơn EABI).
  + Default Settings: ELF format, VFPv3-D16, ARM instruction set.
* Build Options : Giữ mặc định, xem qua các tùy chọn có sẵn để tham khảo.
* Toolchain
  + Toolchain Type: Buildroot toolchain (xây dựng toolchain nội bộ).
  + Thư viện: Sử dụng glibc
* System Configuration
  + Tùy chỉnh: System hostname, System banner, Root password.
  + Các tùy chọn khác giữ mặc định.
* Kernel
  + Enable Linux Kernel: Bật.
  + Kernel Version: Custom, sử dụng 6.6.58
  + Defconfig: omap2plus\_defconfig (hỗ trợ AM335x).
  + Kernel Binary Format: zImage.
  + Device Tree: Bật, sử dụng ti/omap/am335x-boneblack.dts.
  + Host Requirement: Bật Needs host OpenSSL.
* Target Packages
  + Mặc định bật BusyBox, tùy chọn thêm gói khác sau
  + Networking application: Bật MQTT,dhcpd,iproute2 để hỗ trợ ethernet
* Filesystem Images :
  + Chỉ bật tar the root filesystem (xử lý SD card riêng)
* Bootloaders
  + Bootloader: U-Boot, phiên bản 2025.02.
  + Build System: Kconfig.
  + Board Defconfig: am335x\_evm\_defconfig.
  + Custom Make Options: DEVICE\_TREE=am335x-boneblack.
  + Binary Format: u-boot.img, bật SPL (MLO).
* Sau khi chạy lệnh make hoàn tất, Buildroot sẽ tạo ra:
  + Toolchain: Cross-compiler ARM (dùng glibc/uClibc) trong thư mục output/host.
  + Các file boot: Bao gồm zImage (kernel), am335x-boneblack.dtb (Device Tree), MLO và u-boot.img (U-Boot) trong output/images.
  + Root filesystem: Tệp rootfs.tar trong output/images, sẵn sàng để triển khai lên thẻ SD cho BeagleBone Black.
* Boot BeagleBone Black bằng thẻ nhớ và đăng nhập

Sau khi build xong, ghi các tệp từ Buildroot (zImage, am335x-boneblack.dtb, MLO, u-boot.img, rootfs.tar) vào thẻ SD:

* Chuẩn bị thẻ SD: Phân vùng thẻ SD với 1 phân vùng boot (FAT32) và 1 phân vùng rootfs (ext4).
* Copy file boot: Sao chép MLO, u-boot.img, zImage, am335x-boneblack.dtb vào phân vùng boot.
* Giải nén rootfs: Giải nén rootfs.tar vào phân vùng rootfs.
* Boot BBB: Cắm thẻ SD vào BBB, cấp nguồn. U-Boot tự động nạp kernel và Device Tree, khởi động hệ thống.
* Đăng nhập: Kết nối qua cổng serial (minicom/PuTTY, 115200 baud) hoặc SSH (nếu cấu hình mạng), đăng nhập bằng tài  
  khoản root (mật khẩu tùy chỉnh trong Buildroot). Giao diện dòng lệnh của BBB sẽ hiển thị, sẵn sàng sử dụng.

Ta thấy buildroot công dụng của buildroot :

* Tùy chỉnh hệ điều hành nhúng phù hợp với phần cứng.
* Lược bỏ các thành phần không cần thiết để giảm kích thước hệ thống.
* Tích hợp sẵn ứng dụng, driver hoặc thư viện theo yêu cầu.
* Hỗ trợ đóng gói hệ thống vào các định dạng như .ext4, .tar, .sdcard.img để dễ dàng nạp vào thiết bị.
  + 1. **Application**

Ứng dụng (app.c) có nhiệm vụ thu thập dữ liệu từ cảm biến DHT11 (nhiệt độ, độ ẩm) và BH1750 (ánh sáng), xử lý dữ liệu, gửi thông tin lên MQTT, và điều khiển LED dựa trên ngưỡng thiết lập.

Chức năng chính của app.c

* Thu thập dữ liệu cảm biến, cập nhật liên tục theo chu kỳ nhất định.
* Xử lý và truyền dữ liệu lên MQTT để hiển thị trên web dashboard.
* Điều khiển LED dựa trên giá trị cảm biến hoặc thao tác từ người dùng.

Giao tiếp và điều khiển thiết bị

* Sử dụng file /dev/dht11 và /dev/bh1750 để đọc dữ liệu từ cảm biến.
* Gửi lệnh đến /dev/led để bật/tắt thiết bị thông qua GPIO.
* Tích hợp MQTT, cho phép hệ thống gửi dữ liệu cảm biến lên nền tảng giám sát từ xa.
  + 1. **Driver DHT11**

Driver DHT11 giúp hệ thống nhúng giao tiếp với cảm biến nhiệt độ và độ ẩm thông qua GPIO, thu thập và xử lý dữ liệu để hiển thị lên giao diện giám sát.

Cấu trúc và nguyên lý hoạt động

* Giao tiếp 1-wire: DHT11 sử dụng một dây duy nhất để truyền dữ liệu theo chu kỳ HIGH/LOW, yêu cầu xử lý tín hiệu chính xác.
* Giao tiếp GPIO: Driver điều khiển chân GPIO của BeagleBone Black để gửi tín hiệu khởi động và nhận phản hồi từ cảm biến.
* Sử dụng bit-banging: Kỹ thuật đọc từng bit từ cảm biến bằng cách kiểm soát thời gian chính xác.

Chức năng chính của driver

* Khởi tạo giao tiếp bằng cách đặt GPIO ở chế độ input hoặc output.
* Đọc dữ liệu cảm biến, kiểm tra checksum để đảm bảo giá trị không bị sai lệch.
* Chuyển đổi giá trị nhiệt độ và độ ẩm, hiển thị theo dạng "Temp: XX°C, Hum: XX%".
* Cung cấp giao diện thiết bị (/dev/dht11) để cho phép ứng dụng người dùng đọc dữ liệu trực tiếp.

Độ tin cậy và tối ưu hóa

* Mutex bảo vệ quá trình đọc, tránh xung đột dữ liệu khi có nhiều yêu cầu cùng lúc.
* Cơ chế timeout, giúp ngăn việc treo hệ thống nếu cảm biến không phản hồi.
* Kiểm tra checksum, đảm bảo dữ liệu đọc được là chính xác, tránh lỗi đo đạc.

|  |
| --- |
| obj-m += dht11.o  # Đường dẫn tới kernel source hoặc kernel headers của BBB (nơi có Makefile của kernel)  KDIR=~/buildroot-2025.02/output/build/linux-6.6.58  PWD := $(shell pwd)  # Đặt CROSS\_COMPILE cho toolchain ARM, ví dụ arm-linux-gnueabihf  CROSS\_COMPILE=~/buildroot-2025.02/output/host/bin/arm-buildroot-linux-gnueabihf-  all:  $(MAKE) -C $(KDIR) M=$(PWD) ARCH=arm CROSS\_COMPILE=$(CROSS\_COMPILE) modules  clean:  $(MAKE) -C $(KDIR) M=$(PWD) clean |

* + 1. **Driver BH1750**

Driver BH1750 cho phép hệ thống nhúng đọc dữ liệu ánh sáng thông qua giao tiếp I2C, giúp giám sát môi trường theo thời gian thực.

Cấu trúc và nguyên lý hoạt động

* Giao tiếp I2C: BH1750 sử dụng bus I2C để trao đổi dữ liệu với hệ thống nhúng.
* Độ phân giải cao: Cảm biến xuất giá trị 16-bit, cho biết cường độ ánh sáng theo đơn vị lux.
* Cách hoạt động: Hệ thống gửi lệnh yêu cầu dữ liệu, BH1750 phản hồi bằng một giá trị ánh sáng đo được.

Chức năng chính của driver

* Khởi tạo giao tiếp với cảm biến BH1750 trên bus I2C.
* Gửi lệnh đọc dữ liệu, xử lý và chuyển đổi sang dạng dễ hiểu.
* Cung cấp giao diện thiết bị (/dev/bh1750), cho phép ứng dụng truy xuất dữ liệu trực tiếp.

|  |
| --- |
| obj-m += bh1750.o  # Đường dẫn tới kernel source hoặc kernel headers của BBB (nơi có Makefile của kernel)  KDIR=~/buildroot-2025.02/output/build/linux-6.6.58  PWD := $(shell pwd)  # Đặt CROSS\_COMPILE cho toolchain ARM, ví dụ arm-linux-gnueabihf  CROSS\_COMPILE=~/buildroot-2025.02/output/host/bin/arm-buildroot-linux-gnueabihf-  all:  $(MAKE) -C $(KDIR) M=$(PWD) ARCH=arm CROSS\_COMPILE=$(CROSS\_COMPILE) modules  clean:  $(MAKE) -C $(KDIR) M=$(PWD) clean |

* + 1. **Driver LED**

Driver LED đóng vai trò trung gian giữa phần cứng và hệ điều hành, giúp điều khiển LED thông qua GPIO trên nền tảng BeagleBone Black. Driver này được thiết kế để nhận lệnh từ hệ thống, điều khiển trạng thái bật/tắt LED dựa trên điều kiện môi trường hoặc thao tác từ giao diện web.

* Cấu trúc và nguyên lý hoạt động
* Sử dụng GPIO để điều khiển LED, cụ thể GPIO1\_28 (LED Web) và GPIO1\_29 (LED Temp).
* Ghi trạng thái LED thông qua ioread32() và iowrite32() để cập nhật giá trị trên thanh ghi DATAOUT.
* Giao tiếp với người dùng bằng cách tạo character device (/dev/led), cho phép đọc hoặc ghi trạng thái LED.
* Các chức năng chính
* led\_set(gpio, state): Bật/tắt LED bằng cách thay đổi giá trị trong thanh ghi DATAOUT.
* led\_get(gpio): Đọc trạng thái hiện tại của LED từ thanh ghi GPIO.
* device\_read(): Cho phép ứng dụng người dùng đọc trạng thái LED qua /dev/led.
* device\_write(): Cho phép ghi trạng thái LED từ ứng dụng, sử dụng cú pháp như "1:1" để bật LED1.
* Khởi tạo và huỷ driver
* Khi module được tải, LED sẽ được cấu hình là output và đảm bảo trạng thái ban đầu tắt (OFF).
* Khi module được gỡ bỏ, LED sẽ được đặt về trạng thái tắt để tránh tiêu thụ điện năng không cần thiết.

|  |
| --- |
| obj-m += led.o  # Đường dẫn tới kernel source hoặc kernel headers của BBB (nơi có Makefile của kernel)  KDIR=~/buildroot-2025.02/output/build/linux-6.6.58  PWD := $(shell pwd)  # Đặt CROSS\_COMPILE cho toolchain ARM, ví dụ arm-linux-gnueabihf  CROSS\_COMPILE=~/buildroot-2025.02/output/host/bin/arm-buildroot-linux-gnueabihf-  all:  $(MAKE) -C $(KDIR) M=$(PWD) ARCH=arm CROSS\_COMPILE=$(CROSS\_COMPILE) modules  clean:  $(MAKE) -C $(KDIR) M=$(PWD) clean |

## 2.4. Phát triển hệ điều hành cho hệ thống

Mục tiêu: Tích hợp driver DHT11, BH1750, LED; gửi dữ liệu qua MQTT; điều khiển LED dựa trên ngưỡng và lệnh.

Các bước bao gồm

* Thiết kế: Vòng lặp chính đọc dữ liệu, gửi MQTT, điều khiển LED. Thread phụ đọc DHT11 và giám sát treo.
* Tích hợp driver:
* DHT11: Đọc /dev/dht11\_driver (nhiệt độ, độ ẩm), retry 3 lần, tắt sau 5 lỗi.
* BH1750: Đọc /dev/bh1750 (lux).
* LED: Gửi lệnh "LED:STATE" tới /dev/led\_driver, nháy LED theo ngưỡng.
* MQTT: Kết nối broker, gửi dữ liệu (sensor/\*), nhận lệnh (control/led/\*), tái kết nối sau 5s nếu mất.
* Giám sát: Thread kiểm tra treo, khởi động lại DHT11 nếu cần. Watchdog ping định kỳ (tùy chọn).
* Log: Ghi trạng thái, lỗi vào /var/log/sensor\_system.log (giới hạn 1MB).
* Tín hiệu: Xử lý SIGINT, SIGTERM để thoát an toàn.

Kết quả: Ứng dụng ổn định, tích hợp driver, MQTT, và chống treo hiệu quả.

## 2.5. Thiết lập auto service cho hệ thống

BusyBox init thường sử dụng:

* File /etc/inittab để xác định các hành động khi khởi động.
* Thư mục /etc/init.d/ để chạy các script khởi động (tương tự SysVinit, nhưng đơn giản hơn).
* File /etc/rcS hoặc /etc/rc.local (nếu có) để chạy các lệnh khởi động.

Phương pháp phù hợp :

* Tạo một script trong /etc/init.d/ để nạp driver và chạy ứng dụng.
* Liên kết script này để chạy khi khởi động.

Tạo script khởi động:

* + File: /etc/init.d/S99bbb\_drivers
  + Nội dung:
* start: Nạp driver (insmod /root/\*.ko), chạy /root/app &.
* stop: Dừng ứng dụng (kill -9), gỡ driver (rmmod).
  + Lệnh: chmod +x /etc/init.d/S99bbb\_drivers
  + Kích hoạt tự động: /etc/init.d/rcS tự chạy các script S\* khi khởi động.
  + S99bbb\_drivers được gọi nhờ tiền tố S99.

## 2.6. Thiết kế firmware

Trong hệ thống này, **driver** đóng vai trò trung gian giữa phần cứng (cảm biến, LED) và nhân hệ điều hành (Linux kernel). Mỗi thiết bị ngoại vi được truy cập thông qua **tầng driver**, nhằm đảm bảo việc giao tiếp chuẩn hóa, bảo mật và ổn định. Mô hình thiết kế driver trong hệ thống bao gồm:

* **Driver cho cảm biến DHT11 (One-Wire GPIO)**  
  DHT11 sử dụng giao thức một dây không chuẩn, nên thường được điều khiển thông qua **bit-banging** tại tầng kernel hoặc bằng một driver người dùng (user-space driver). Driver chịu trách nhiệm tạo thời gian trễ chính xác và đọc tín hiệu số theo chuẩn thời gian cụ thể.
* **Driver cho cảm biến BH1750 (I2C)**  
  BH1750 sử dụng giao tiếp I2C tiêu chuẩn. Trong Linux, có thể sử dụng **kernel I2C subsystem** để viết driver thiết bị (device driver) dạng i2c\_client, dùng hàm đọc/ghi để lấy dữ liệu ánh sáng. Có thể viết dưới dạng platform\_driver nếu gắn tĩnh vào thiết bị cụ thể.
* **Driver điều khiển LED (GPIO)**  
  LED được điều khiển thông qua **GPIO sysfs hoặc GPIO character driver**. Trong Linux mới, nên dùng gpiod API trong kernel hoặc libgpiod từ người dùng. Driver chịu trách nhiệm xuất mức logic để bật/tắt đèn theo logic hệ thống hoặc từ tín hiệu điều kiện.
* **Tầng giao tiếp người dùng (User Space Interface)**  
  Các driver có thể tạo **node trong /dev** (với character driver) hoặc sử dụng /sys/class, để người dùng tương tác qua ứng dụng như Python/C, shell script hoặc các service đọc giá trị cảm biến và publish qua MQTT.

## 2.7. Cơ sở dữ liệu MySQL

Hệ thống backend viết bằng Python nhận dữ liệu cảm biến thông qua giao thức MQTT, sau đó xử lý và lưu trữ vào cơ sở dữ liệu. Việc lưu trữ này có nhiều mục đích:

* **Lưu vết lịch sử**: mọi giá trị đo được từ cảm biến đều được lưu lại để phục vụ việc giám sát thời gian thực và phân tích sau này.
* **Phản hồi điều khiển**: trạng thái đèn LED được ghi nhận theo thời gian, giúp biết được hệ thống đã từng phản ứng ra sao trong từng điều kiện cụ thể.
* **Phân tích thống kê**: có thể thống kê các thông số môi trường theo thời gian để đưa ra các quyết định tối ưu hơn trong vận hành.
* **Hiển thị dữ liệu**: giao diện frontend có thể truy vấn dữ liệu để hiển thị cho người dùng mà không cần kết nối trực tiếp với phần cứng.

Trong đồ án này, hệ thống sử dụng một cơ sở dữ liệu có tên là **sensor\_system**, được xây dựng trên nền tảng **MySQL** nhằm lưu trữ toàn bộ thông tin liên quan đến cảm biến và điều khiển thiết bị. Cơ sở dữ liệu đóng vai trò trung tâm trong quá trình lưu trữ, truy xuất và phân tích dữ liệu, đồng thời là nơi giao tiếp giữa phần mềm backend và giao diện người dùng.

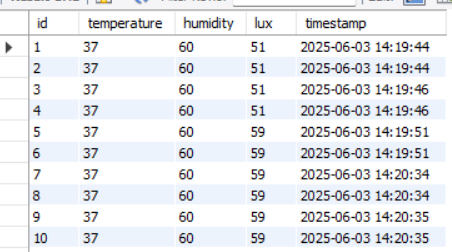
Cấu trúc cơ sở dữ liệu gồm **hai bảng chính**, tương ứng với hai nhóm thông tin cần thiết trong hệ thống:

1. **Bảng lưu dữ liệu cảm biến**

Đây là bảng dùng để lưu trữ toàn bộ dữ liệu thu thập được từ các cảm biến gắn trên thiết bị (ví dụ như cảm biến nhiệt độ, cảm biến độ ẩm, cảm biến ánh sáng). Mỗi khi thiết bị đo được thông số mới, dữ liệu sẽ được gửi về backend qua giao thức MQTT, sau đó được phần mềm backend xử lý và lưu vào bảng này.

Cụ thể, bảng sensor\_data bao gồm các trường sau:

|  |
| --- |
| CREATE TABLE sensor\_data (  id INT AUTO\_INCREMENT PRIMARY KEY,  temperature FLOAT NOT NULL,  humidity FLOAT NOT NULL,  lux FLOAT NOT NULL,  timestamp DATETIME DEFAULT CURRENT\_TIMESTAMP  ); |

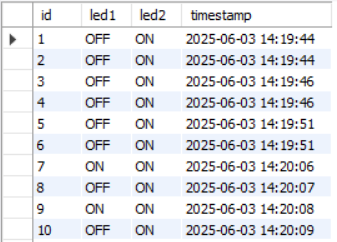


Hình 12 Bảng dữ liệu cảm biến

1. **Bảng lưu trạng thái led**

Bảng này có nhiệm vụ lưu trữ trạng thái hoạt động của các đèn LED trong hệ thống. Các đèn này được điều khiển tự động (dựa vào nhiệt độ, độ sáng...) hoặc thủ công (thông qua giao diện người dùng). Mỗi khi có sự thay đổi trạng thái, backend sẽ ghi nhận trạng thái mới và thời điểm thay đổi vào bảng này. Cấu trúc bảng led\_status gồm:

|  |
| --- |
| CREATE TABLE led\_status (  id INT AUTO\_INCREMENT PRIMARY KEY,  led1 VARCHAR(3) NOT NULL,  led2 VARCHAR(3) NOT NULL,  timestamp DATETIME DEFAULT CURRENT\_TIMESTAMP  ); |



Hình 13 Bảng trạng thái thiết bị

## 2.8. Thiết kế và xây dựng hệ thống backend

Phần backend đóng vai trò quan trọng trong việc xử lý dữ liệu, kết nối với cơ sở dữ liệu và giao tiếp với thiết bị thông qua giao thức MQTT. Backend được xây dựng sử dụng ngôn ngữ **Python** kết hợp với framework **Flask** – một framework nhẹ, dễ triển khai và phù hợp với mô hình ứng dụng Web phục vụ API.

**2.8.1 Kiến trúc tổng thể**

Hệ thống backend có các chức năng chính như sau:

* Tiếp nhận dữ liệu từ cảm biến qua MQTT.
* Xử lý và lưu trữ dữ liệu cảm biến vào MySQL.
* Tự động điều khiển đèn LED dựa trên ngưỡng nhiệt độ (ví dụ: LED2 bật nếu nhiệt độ > 27°C).
* Phục vụ API RESTful cho giao diện frontend để truy xuất dữ liệu hiện tại và lịch sử.
* Cho phép điều khiển LED từ giao diện web và gửi lệnh điều khiển qua MQTT.

**2.8.2 Giao tiếp với MQTT**

Hệ thống sử dụng thư viện paho-mqtt để kết nối với **MQTT broker**. Dữ liệu từ cảm biến được gửi đến topic bbb/sensors, và hệ thống sẽ đăng ký (subscribe) để nhận dữ liệu từ topic này.

Khi nhận được dữ liệu mới, hệ thống backend:

* Phân tích giá trị cảm biến (temperature, humidity, lux).
* Ghi dữ liệu vào bảng sensor\_data.
* Kiểm tra và cập nhật trạng thái LED (ví dụ: tự động bật LED2 nếu nhiệt độ cao).
* Gửi trạng thái LED mới xuống thiết bị thông qua topic bbb/led.

|  |
| --- |
| def on\_message(client, userdata, msg):  payload = msg.payload.decode()  data = json.loads(payload)  temperature = data.get("temperature", 0)  humidity = data.get("humidity", 0)  lux = data.get("lux", 0)  led2\_status = "ON" if temperature > 27 else "OFF"  db = get\_db\_connection()  cursor = db.cursor()  cursor.execute(  "INSERT INTO sensor\_data (temperature, humidity, lux) VALUES (%s, %s, %s)",  (temperature, humidity, lux)  )  cursor.execute(  "INSERT INTO led\_status (led1, led2) VALUES (%s, %s)",  (led1\_status, led2\_status)  )  db.commit() |

**2.8.3. API cung cấp cho frontend**

Backend cung cấp một số **endpoint RESTful** thông qua Flask để frontend truy xuất hoặc gửi dữ liệu:

* GET /api/latest: Trả về dữ liệu cảm biến và trạng thái LED mới nhất.
* POST /api/led: Cho phép frontend điều khiển LED1 và LED2. Backend sẽ ghi nhận trạng thái vào cơ sở dữ liệu và gửi lệnh điều khiển MQTT.
* GET /api/history\_sensors: Trả về lịch sử 10 bản ghi cảm biến gần nhất.
* GET /api/history\_led: Trả về lịch sử 10 lần thay đổi trạng thái LED gần nhất.

Các API đều trả dữ liệu dưới dạng **JSON**, giúp frontend dễ dàng xử lý và hiển thị thông tin lên giao diện.

|  |
| --- |
| @app.route('/api/latest', methods=['GET'])  def get\_latest():  db = get\_db\_connection()  cursor = db.cursor(dictionary=True)  cursor.execute("""  SELECT s.temperature, s.humidity, s.lux, s.timestamp,  COALESCE(l.led1, 'OFF') as led1,  COALESCE(l.led2, 'OFF') as led2  FROM sensor\_data s  LEFT JOIN led\_status l  ON l.timestamp = (  SELECT MAX(timestamp)  FROM led\_status  WHERE timestamp <= s.timestamp  )  ORDER BY s.id DESC LIMIT 1  """)  row = cursor.fetchone()  return jsonify({  "temperature": int(round(row['temperature'])),  "humidity": int(round(row['humidity'])),  "lux": int(round(row['lux'])),  "led1": row['led1'],  "led2": row['led2'],  "timestamp": row['timestamp'].strftime('%Y-%m-%d %H:%M:%S')  }) |

**2.8.4 Kết nối cơ sở dữ liệu**

Backend sử dụng thư viện mysql-connector-python để kết nối và thao tác với cơ sở dữ liệu **MySQL**. Một hàm get\_db\_connection() được thiết kế để mở kết nối tới CSDL khi cần thiết, và các thao tác đọc/ghi dữ liệu được thực hiện qua các truy vấn SQL phù hợp.

Để tăng độ tin cậy và dễ bảo trì:

* Tất cả thao tác với CSDL đều có xử lý **đóng cursor và connection** sau khi dùng xong.
* **Giá trị thời gian (timestamp)** trong bảng được hệ thống tự động chèn theo thời gian thực tại khi ghi nhận dữ liệu, giúp dễ dàng truy xuất theo thời gian.

|  |
| --- |
| def get\_db\_connection():  return mysql.connector.connect(  host="localhost",  user="root",  password="12345",  database="sensor\_system"  ) |

**2.8.5 Tự động điều khiển LED**

Một điểm nổi bật trong backend là khả năng **điều khiển LED tự động theo điều kiện môi trường**, cụ thể:

* LED2 sẽ được tự động bật (ON) nếu nhiệt độ vượt quá 35°C.
* LED1 sẽ được giữ nguyên theo trạng thái gần nhất được người dùng điều khiển từ frontend.

Điều này giúp hệ thống có thể hoạt động ở **chế độ tự động hoặc bán tự động**, góp phần nâng cao khả năng ứng dụng thực tế.

## 2.7. Kết luận.

Hệ điều hành nhúng (Embedded OS) là thành phần phần mềm quan trọng điều khiển hoạt động của hệ thống nhúng, giúp quản lý tài nguyên phần cứng và thực thi các tác vụ theo thời gian thực hoặc bán thời gian thực. Các hệ điều hành nhúng phổ biến như Embedded Linux, FreeRTOS được ứng dụng rộng rãi trong thiết bị IoT, điều khiển công nghiệp, ô tô, và robot.

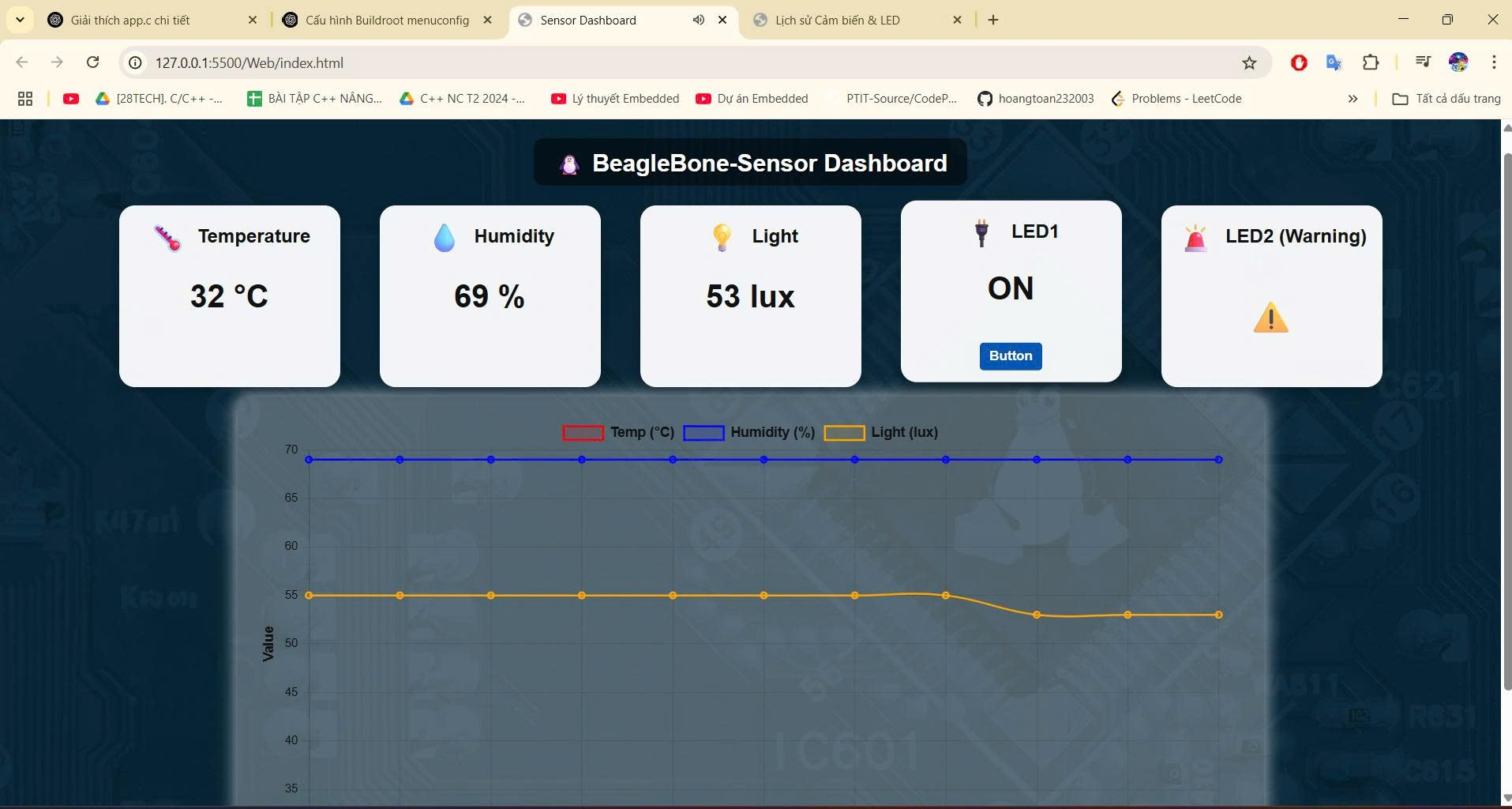
Embedded Linux, điển hình là các bản phân phối như Buildroot hoặc Yocto, cung cấp một môi trường linh hoạt, mã nguồn mở, phù hợp cho các hệ thống yêu cầu hiệu năng tốt và dễ tùy chỉnh. Trong đó, BeagleBone Black là một nền tảng phát triển mạnh mẽ, hỗ trợ chạy Embedded Linux, tích hợp nhiều cổng giao tiếp như GPIO, I2C, UART… thuận tiện để phát triển ứng dụng nhúng.

Trong mô hình đề tài, các thành phần chính gồm BeagleBone Black, cảm biến DHT11 (đo nhiệt độ/độ ẩm), BH1750 (đo ánh sáng), LED điều khiển cảnh báo, và Web server nhận dữ liệu thông qua giao thức MQTT. Các driver thiết bị được thiết kế để làm cầu nối giữa phần cứng và nhân hệ điều hành, giúp xử lý tín hiệu cảm biến và điều khiển thiết bị ngoại vi một cách chính xác, ổn định.

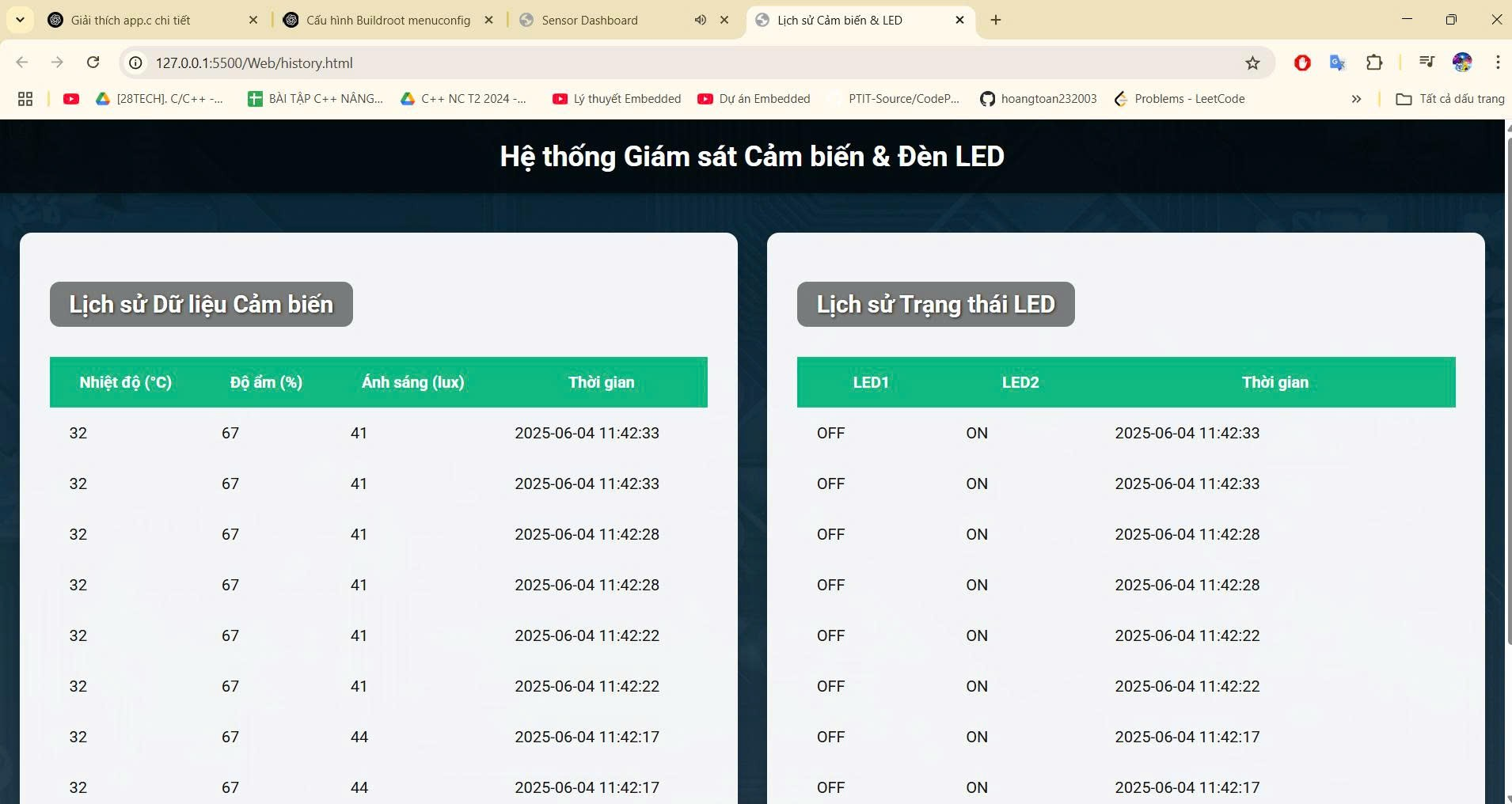
Các công cụ như Buildroot giúp build kernel và hệ thống file tối ưu cho BeagleBone, đồng thời hỗ trợ tích hợp sẵn các thư viện cần thiết để phát triển ứng dụng. Toàn bộ kiến trúc hệ thống được tổ chức theo mô hình từ BSP (Board Support Package) đến EML (Embedded Middleware Layer), đảm bảo tính phân tách rõ ràng giữa tầng phần cứng và phần mềm ứng dụng.

# KẾT QUẢ CHẠY MÔ PHỎNG

Hệ thống hoạt động ổn định, dữ liệu từ các cảm biến được ghi nhận và truyền lên web theo thời gian thực.

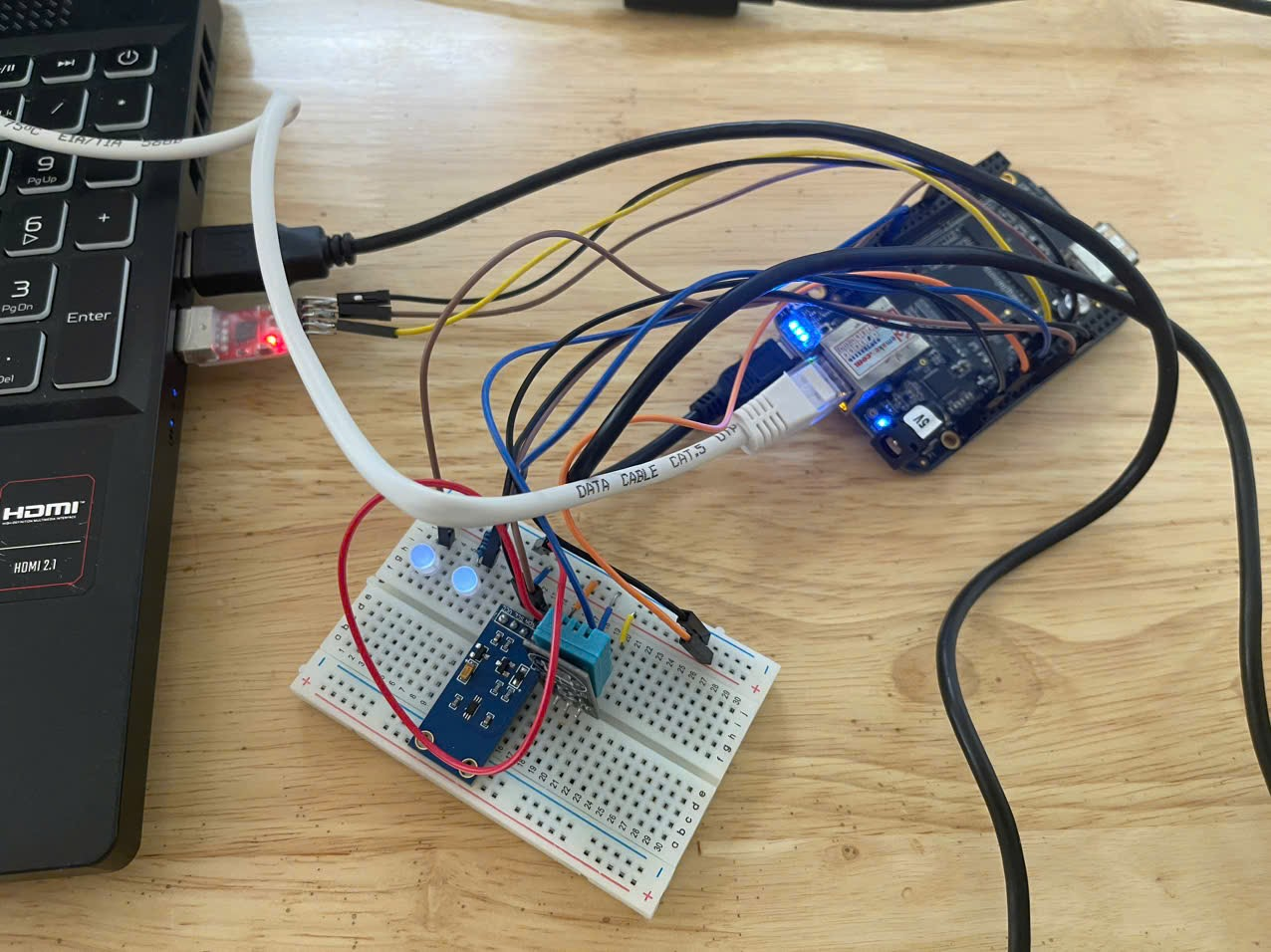


Hình 14 Giao diện giám sát và điều khiển



Hình 15 Bảng dữ liệu cảm biến và trạng thái thiết bị

* Dữ liệu cảm biến được thu thập theo chu kỳ đều đặn với các giá trị nhiệt độ, độ ẩm, và ánh sáng nhất quán trong các mốc thời gian.
* Trạng thái LED cũng được cập nhật liên tục, phản ánh điều kiện môi trường hoặc thao tác từ người dùng trên giao diện web.
* Nhiệt độ dao động nhẹ, nhưng duy trì mức ổn định khoảng 30°C.
* Độ ẩm ít biến đổi, giữ mức 60-67%, chứng tỏ môi trường thử nghiệm có điều kiện ổn định.
* Ánh sáng giảm xuống 41 lux, có thể do điều kiện ánh sáng yếu hoặc thay đổi trong môi trường thử nghiệm.
* LED1 luôn TẮT trong lịch sử theo dõi, có thể do không có điều kiện kích hoạt hoặc cần kiểm tra logic điều khiển.
* LED2 luôn BẬT, có thể phản hồi theo ngưỡng nhiệt độ hoặc được thiết lập để cảnh báo.



Hình 16 Phần cứng thực tế