ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

BÁO CÁO
ĐỒ ÁN MÔN HỌC KỸ THUẬT MÁY TÍNH

PHÁT TRIỂN THƯ VIỆN GIAO DIỆN CHO MẠCH NHÚNG ESP32 SỬ DỤNG THƯ VIỆN ĐỒ HỌA LVGL

Ngành: Kỹ thuật máy tính

HỘI ĐỒNG: **HỘI ĐỒNG X KỸ THUẬT MÁY TÍNH**
GVHD: **TS. LÊ TRỌNG NHÂN**
TKHĐ: **TS. LÊ TRỌNG NHÂN**

——————o0o——————

SVTH 1: **Nguyễn Trọng Vinh** **(2015070)**
SVTH 2: **Lê Thanh Dương** **(MSSV)**
SVTH 3: **Lê Đỗ Minh Thông** **(MSSV)**

TP. Hồ Chí Minh, 05/2025

Lời cam đoan

Chúng em xin cam đoan rằng báo cáo luận văn với đề tài "Phát triển thư viện giao diện cho mạch nhúng ESP32 sử dụng thư viện đồ họa LVGL" là kết quả của quá trình nghiên cứu và thực hiện nghiêm túc của nhóm, dưới sự hướng dẫn tận tình của thầy Lê Trọng Nhân. Toàn bộ nội dung, số liệu và kết quả trong báo cáo đều do nhóm tự triển khai, không sao chép từ bất kỳ nguồn tài liệu hay công trình nào khác. Mọi sự hỗ trợ và các tài liệu tham khảo đều được trích dẫn rõ ràng và minh bạch. Nhóm chúng em xin hoàn toàn chịu trách nhiệm nếu có bất kỳ vi phạm nào liên quan đến tính trung thực và bản quyền trong báo cáo này.

Thành phố Hồ Chí Minh, tháng 4 năm 2025.

Lời cảm ơn

Chúng em xin bày tỏ lòng biết ơn chân thành đến thầy Lê Trọng Nhân, người đã tận tình hướng dẫn và đồng hành cùng nhóm trong suốt quá trình thực hiện đề tài. Thầy luôn theo sát tiến độ, góp ý kịp thời và tạo mọi điều kiện thuận lợi để nhóm có thể hoàn thành tốt nhiệm vụ được giao.

Đồng thời, những ý kiến đóng góp từ thầy không chỉ giúp chúng em hoàn thiện nội dung báo cáo mà còn là nguồn động viên lớn trong suốt quá trình học tập và nghiên cứu.

Chúng em xin kính chúc thầy luôn mạnh khỏe, thành công trong sự nghiệp giảng dạy và tiếp tục truyền cảm hứng học tập, nghiên cứu cho các thế hệ sinh viên tiếp theo.

Mặc dù đã nỗ lực để hoàn thành đề tài trong phạm vi thời gian và kiến thức hiện có, nhưng chắc chắn vẫn còn những thiếu sót. Chúng em rất mong nhận được thêm ý kiến đóng góp từ quý thầy cô để hoàn thiện hơn trong tương lai.

Chúng em xin chân thành cảm ơn.

Thành phố Hồ Chí Minh, tháng 4 năm 2025.

Tóm tắt đồ án

Đề tài tập trung vào việc phát triển một thư viện giao diện đồ họa cho mạch nhúng ESP32, sử dụng thư viện mã nguồn mở LVGL (Light and Versatile Graphics Library). Mục tiêu là xây dựng một hệ thống giao diện người dùng trực quan, linh hoạt, tối ưu cho các ứng dụng nhúng trong điều kiện tài nguyên hạn chế. Thư viện được phát triển nhằm phục vụ cho việc thiết kế, lập trình và hiển thị các thành phần giao diện như nút nhấn, biểu đồ, thông báo trạng thái... trên các màn hình cảm ứng sử dụng với vi điều khiển ESP32.

Trong quá trình thực hiện, nhóm đã xây dựng cấu trúc phần mềm có khả năng mở rộng, đồng thời thiết kế mẫu giao diện thực tế nhằm thử nghiệm hiệu quả của thư viện trên màn hình cảm ứng 7 inch đi kèm ESP32-S3.

Các nội dung chính của đề tài bao gồm:

- **Nghiên cứu và tích hợp thư viện LVGL:** Làm rõ cách LVGL hoạt động, cách cấu hình và tích hợp với nền tảng ESP-IDF.
- **Xây dựng các thành phần giao diện:** Tạo các thành phần phổ biến như nút, slider, biểu đồ, widget cảm biến... tương thích với hệ thống điều khiển.
- **Thiết kế mô-đun thư viện:** Tổ chức mã nguồn thành các mô-đun có tính tái sử dụng cao, dễ bảo trì và tùy biến theo yêu cầu ứng dụng.
- **Thử nghiệm thực tế:** Ứng dụng thư viện trong một giao diện điều khiển thiết bị đơn giản, hiển thị dữ liệu từ cảm biến nhiệt độ, độ ẩm và trạng thái các thiết bị điều khiển.

Kết quả cho thấy thư viện hoạt động ổn định, có khả năng phản hồi nhanh và dễ dàng tích hợp vào các dự án thực tế sử dụng ESP32. Đề tài mang lại cái nhìn thực tế về việc xây dựng giao diện trong hệ thống nhúng, góp phần giúp sinh viên nâng cao năng lực lập trình, tư duy hệ thống và ứng dụng công nghệ hiện đại.

Thành phố Hồ Chí Minh, tháng 4 năm 2025.

Mục lục

Danh sách bảng

Danh sách hình vẽ

Chương 1

Tổng quan về đề tài

1.1 Giới thiệu

Trong thời đại công nghệ số phát triển mạnh mẽ, việc phát triển giao diện người dùng (GUI) cho các hệ thống nhúng đang trở thành một nhu cầu thiết yếu trong các ứng dụng công nghiệp, dân dụng và IoT. Giao diện trực quan không chỉ giúp người dùng tương tác thuận tiện hơn với hệ thống, mà còn nâng cao giá trị sử dụng và tính chuyên nghiệp của sản phẩm.

Với sự phổ biến của vi điều khiển ESP32-S3 và sự phát triển của các thư viện mã nguồn mở như LVGL (Light and Versatile Graphics Library), việc xây dựng các giao diện đồ họa phức tạp trên các hệ thống nhúng trở nên khả thi và hiệu quả. LVGL cung cấp đầy đủ các thành phần UI như nút bấm, slider, bảng thông tin... và hỗ trợ cảm ứng, animation, đa ngôn ngữ, rất phù hợp cho các ứng dụng có màn hình tích hợp.

Trong khuôn khổ đề tài **"Phát triển thư viện giao diện cho mạch nhúng ESP32 sử dụng thư viện đồ họa LVGL"**, nhóm chúng em tập trung nghiên cứu, thiết kế và hiện thực một thư viện giao diện đồ họa tùy biến, sử dụng ESP32-S3 kết hợp với màn hình cảm ứng 7 inch. Giao diện này sẽ hỗ trợ hiển thị và tương tác với các thông tin cảm biến như nhiệt độ, độ ẩm và trạng thái các thiết bị điều khiển trong mô hình thử nghiệm.Dự án cũng tích hợp kết nối với Web Server riêng, cho phép ESP32 đồng bộ dữ liệu cảm biến và nhận lệnh điều khiển thông qua giao thức HTTP hoặc WebSocket. Việc này hỗ trợ khả năng giám sát và điều khiển từ xa qua giao diện web một cách linh hoạt, không phụ thuộc vào nền tảng IoT bên thứ ba.

1.2 Mục tiêu

Mục tiêu chính của đề tài là xây dựng một thư viện giao diện người dùng trực quan dành cho hệ thống nhúng sử dụng vi điều khiển ESP32-S3 và thư viện LVGL. Thư viện này sẽ giúp đơn giản hóa quá trình phát triển ứng dụng trên nền tảng ESP32 có tích hợp màn hình cảm ứng. Các mục tiêu cụ thể bao gồm:

- **Tìm hiểu và tích hợp LVGL vào hệ thống ESP32-S3**: Cấu hình hệ thống phần mềm để chạy thư viện LVGL mượt mà trên màn hình cảm ứng 7 inch sử dụng giao tiếp SPI/Parallel.
- **Thiết kế thư viện giao diện tái sử dụng**: Phát triển các thành phần UI có thể tái sử dụng như các nút điều khiển, thẻ thông tin cảm biến, màn hình chính, màn hình cài đặt, v.v... phục vụ cho các hệ thống nhúng có màn hình.
- **Tích hợp cảm biến và hệ thống điều khiển**: Hiển thị thông tin từ các cảm biến nhiệt độ, độ ẩm (kết nối RS485 hoặc UART) và điều khiển thiết bị đầu ra (qua relay) thông qua giao diện đồ họa.
- **Kết nối và tương tác với Web Server**: ESP32 có khả năng gửi dữ liệu cảm biến đến Web Server thông qua HTTP hoặc WebSocket, đồng thời nhận lệnh điều khiển để cập nhật trạng thái giao diện và các thiết bị đầu ra. Hệ thống có thể được giám sát và điều khiển từ xa qua trình duyệt web.
- **Tối ưu hiệu năng**: Đảm bảo giao diện hoạt động mượt mà, tiêu tốn tài nguyên hợp lý và có khả năng mở rộng cho các ứng dụng thực tế khác.

1.3 Thực trạng

Việc phát triển giao diện người dùng trên các thiết bị nhúng trước đây thường gặp nhiều khó khăn do hạn chế về tài nguyên phần cứng, thiếu công cụ hỗ trợ và chi phí cao. Tuy nhiên, trong những năm gần đây, sự xuất hiện của các thư viện đồ họa nhẹ như LVGL đã thay đổi đáng kể cục diện.

Mặc dù LVGL đã được cộng đồng mã nguồn mở sử dụng rộng rãi, nhưng việc tích hợp và sử dụng thư viện này trên ESP32-S3 vẫn đòi hỏi kiến thức chuyên sâu về cấu trúc hệ thống nhúng, quản lý bộ nhớ, cảm ứng và hiển thị. Ngoài ra, hầu hết các giao diện hiện tại vẫn chưa tối ưu về tính trực quan, khả năng tái sử dụng và mở rộng.

Do đó, việc phát triển một thư viện giao diện chuyên dụng, dễ tùy chỉnh, thân thiện với người dùng và phù hợp với ứng dụng IoT thực tế là điều cần thiết.

1.4 Phạm vi dự án

Phạm vi nghiên cứu của đề tài tập trung vào việc phát triển và thử nghiệm thư viện giao diện đồ họa trên hệ thống phần cứng bao gồm:

- ESP32-S3 kết hợp với màn hình cảm ứng 7 inch.
- Giao diện được xây dựng bằng LVGL, hiển thị dữ liệu cảm biến (nhiệt độ, độ ẩm).
- Hỗ trợ điều khiển thiết bị đầu ra thông qua các nút điều khiển trên giao diện.
- Tích hợp truyền nhận dữ liệu cảm biến qua giao tiếp RS485.
- Giao tiếp với Web Server nội bộ hoặc trên mạng thông qua giao thức HTTP/WebSocket để truyền nhận dữ liệu và thực hiện điều khiển từ xa.

Đề tài không đi sâu vào phát triển phần cứng cấp thấp hay các thuật toán điều khiển phức tạp, mà tập trung vào phần giao diện, tính năng kết nối và khả năng mở rộng.

1.5 Ý nghĩa thực tiễn

Việc phát triển thư viện giao diện sử dụng LVGL trên ESP32-S3 không chỉ giúp chuẩn hóa và tối ưu quy trình thiết kế giao diện cho hệ thống nhúng, mà còn mang lại nhiều ý nghĩa thực tiễn như:

1. **Tăng tính tương tác người dùng**: Cung cấp giao diện trực quan, hiện đại cho các thiết bị IoT hoặc hệ thống nhúng, giúp người dùng dễ thao tác và theo dõi.
2. **Rút ngắn thời gian phát triển sản phẩm**: Thư viện giao diện có thể tái sử dụng, giúp đẩy nhanh quá trình tích hợp và triển khai các sản phẩm nhúng mới.
3. **Tăng cường tính chuyên nghiệp**: Giao diện đẹp và ổn định góp phần nâng cao giá trị của sản phẩm công nghệ và tăng tính cạnh tranh.
4. **Khả năng mở rộng cao**: Thư viện có thể được áp dụng cho nhiều loại thiết bị khác nhau như máy đo môi trường, bảng điều khiển, hệ thống giám sát từ xa, v.v.

Đề tài góp phần vào việc xây dựng nền tảng phần mềm mở cho các hệ thống nhúng hiện đại, đặc biệt là trong lĩnh vực nhà thông minh, nông nghiệp công nghệ cao và công nghiệp 4.0.

Chương 2

CƠ SỞ LÝ THUYẾT

2.1 Tổng quan về ESP32-S3-Touch-LCD-7

ESP32-S3-Touch-LCD-7 là một nền tảng phát triển mạnh mẽ, kết hợp giữa khả năng tính toán cao cấp của vi điều khiển ESP32-S3 và trải nghiệm giao diện người dùng vượt trội với màn hình cảm ứng điện dung 7 inch. Được thiết kế và phát triển bởi Waveshare, thiết bị này hướng tới các ứng dụng yêu cầu sự giao tiếp trực quan, khả năng xử lý dữ liệu nhanh chóng và tính linh hoạt cao trong môi trường IoT và công nghiệp.

Sự hội tụ giữa công nghệ hiển thị IPS tiên tiến, khả năng cảm ứng đa điểm nhạy bén, cùng với khả năng kết nối mạng không dây Wi-Fi/Bluetooth 5.0, đã biến ESP32-S3-Touch-LCD-7 trở thành lựa chọn lý tưởng cho các dự án giao diện người máy (HMI), hệ thống điều khiển tự động, các thiết bị điện tử tiêu dùng cao cấp và những giải pháp IoT cần tính năng tương tác trực quan.

Hình 2.1: ESP32-S3-Touch-LCD-7 (Giao diện mặt trước)

2.1.1 Cấu hình phần cứng tổng quan

Trung tâm của mô-đun ESP32-S3-Touch-LCD-7 là bộ vi điều khiển ESP32-S3R8, một phiên bản mạnh mẽ thuộc dòng ESP32-S3 với các tính năng chuyên biệt dành cho AI và xử lý đồ họa. Cấu trúc vi xử lý bao gồm:

- **CPU:** 2 nhân Xtensa LX7 tốc độ tối đa 240MHz, hỗ trợ tập lệnh SIMD tăng tốc xử lý song song.
- **RAM nội:** 512KB SRAM + 8MB PSRAM ngoài, đảm bảo khả năng lưu trữ bộ đệm lớn cho các tác vụ đồ họa nặng.
- **Flash:** 16MB Flash SPI tốc độ cao, đáp ứng nhu cầu lưu trữ chương trình và dữ liệu lớn.

- **Màn hình:** IPS LCD 7.0 inch, độ phân giải 1024x600 pixel, hiển thị màu sắc sống động với góc nhìn rộng lên tới 170 độ.
- **Cảm ứng điện dung:** Hỗ trợ đa điểm, sử dụng chip điều khiển cảm ứng GT911, đem lại trải nghiệm chạm, vuốt mượt mà.

Ngoài ra, mô-đun còn được tích hợp các phần tử giao tiếp ngoại vi mạnh mẽ như USB Type-C hỗ trợ chức năng OTG, cổng mở rộng GPIO đầy đủ chức năng (SPI, I2C, UART, ADC, PWM, SDIO), khe cắm thẻ nhớ microSD và giao tiếp Camera (DVP), mang lại khả năng mở rộng tối đa cho các dự án đa dạng.

2.1.2 Khả năng kết nối không dây và bảo mật

ESP32-S3-Touch-LCD-7 kế thừa đầy đủ sức mạnh kết nối không dây từ vi điều khiển ESP32-S3:

- **Wi-Fi 2.4GHz (802.11 b/g/n):** Cho phép mô-đun giao tiếp mạng LAN, Internet hoặc tạo mạng nội bộ (SoftAP) để kết nối các thiết bị khác.
- **Bluetooth 5.0 LE:** Hỗ trợ BLE Mesh, Extended Advertising giúp mở rộng phạm vi và tăng hiệu quả truyền dữ liệu.

Bên cạnh đó, hệ thống còn được tích hợp nhiều cơ chế bảo mật phần cứng như:
Secure Boot: Bảo vệ quá trình khởi động khỏi firmware trái phép.
Flash Encryption: Mã hóa nội dung bộ nhớ flash, đảm bảo dữ liệu và chương trình an toàn trước tấn công vật lý.
Bộ tăng tốc phần cứng: Hỗ trợ các thuật toán mã hóa AES, SHA-2, RSA, ECC phục vụ các ứng dụng bảo mật cao cấp.

2.1.3 Khả năng hiển thị đồ họa và tương tác cảm ứng

Điểm nhấn ấn tượng nhất của ESP32-S3-Touch-LCD-7 nằm ở khả năng hiển thị đồ họa sắc nét kết hợp với cảm ứng điện dung hiện đại. Sử dụng giao tiếp Parallel RGB tốc độ cao giữa vi xử lý và màn hình, thiết bị đảm bảo tốc độ làm tươi hình ảnh nhanh chóng, mang lại trải nghiệm hình ảnh mượt mà, không giật lag.

Màn hình IPS LCD 7 inch không chỉ mang lại độ phân giải cao (1024x600 pixel) mà còn có độ sáng lớn và độ tương phản cao, giúp hiển thị tốt ngay cả trong môi trường ánh sáng mạnh.

Cảm biến cảm ứng điện dung hỗ trợ đa điểm, cho phép thực hiện các thao tác chạm, kéo, vuốt, phóng to/thu nhỏ (multi-touch gestures) mượt mà, đáp ứng yêu cầu giao diện người dùng thân thiện và hiện đại.

2.1.4 Môi trường phát triển phần mềm

Để hỗ trợ lập trình và phát triển phần mềm, ESP32-S3-Touch-LCD-7 tương thích với nhiều công cụ đa dạng, cho phép lập trình viên lựa chọn theo nhu cầu:

- **ESP-IDF:** Bộ công cụ chính thức từ Espressif, hỗ trợ đầy đủ FreeRTOS, LVGL, giao thức mạng MQTT, HTTP, mDNS, OTA update, bảo mật TLS.
- **Arduino IDE:** Dễ sử dụng cho những người mới làm quen với ESP32, với thư viện hỗ trợ màn hình cảm ứng và kết nối mạng phong phú.
- **LVGL (Light and Versatile Graphics Library):** Thư viện đồ họa mã nguồn mở mạnh mẽ, tối ưu hóa cho ESP32-S3, hỗ trợ tạo giao diện người dùng đẹp mắt, chuyên nghiệp.
- **MicroPython:** Hỗ trợ lập trình nhanh gọn bằng ngôn ngữ Python.

Ngoài ra, thiết bị còn hỗ trợ các công cụ debug tiên tiến như OpenOCD, GDB Debugger, và ESP-Prog để hỗ trợ phân tích lỗi và tối ưu chương trình.

2.1.5 Ứng dụng thực tế

Nhờ sự kết hợp hài hòa giữa phần cứng mạnh mẽ, giao diện người dùng hiện đại và khả năng kết nối linh hoạt, ESP32-S3-Touch-LCD-7 có thể được ứng dụng trong rất nhiều lĩnh vực:

- **Thiết bị nhà thông minh:** Bảng điều khiển trung tâm cho hệ thống ánh sáng, điều hòa, an ninh.
- **HMI công nghiệp:** Giao diện điều khiển máy móc, giám sát trạng thái sản xuất theo thời gian thực.
- **Thiết bị chăm sóc sức khỏe:** Máy đo chỉ số sức khỏe, thiết bị y tế cá nhân thông minh.
- **Bảng điều khiển ô tô điện/xe máy điện:** Hiển thị tốc độ, quãng đường, trạng thái pin.
- **Các sản phẩm tiêu dùng cao cấp:** Bếp thông minh, hệ thống POS, máy bán hàng tự động.

2.2 Ngôn ngữ lập trình C/C++

2.2.1 Vai trò cốt lõi của C/C++ trong hệ thống nhúng

Ngôn ngữ C từ lâu đã trở thành tiêu chuẩn vàng trong phát triển hệ thống nhúng nhờ khả năng kiểm soát phần cứng sát cấp thấp, hiệu suất thực thi cao và khả năng biên dịch đa nền tảng. Các đặc điểm nổi bật như quản lý bộ nhớ thủ công, cú pháp đơn giản và khả năng tương thích rộng với trình biên dịch giúp C trở thành lựa chọn hàng đầu trên các nền tảng MCU như ARM Cortex-M, AVR, RISC-V và ESP32.

C++ bổ sung các tính năng mạnh mẽ của lập trình hướng đối tượng (OOP) như kế thừa, đa hình (polymorphism), trừu tượng hóa (abstraction) và đóng gói (encapsulation), cho phép cấu trúc mã nguồn rõ ràng, dễ mở rộng – đặc biệt hữu ích trong các dự án GUI có nhiều lớp giao diện, trạng thái và đối tượng tương tác.

Ví dụ: Trong một ứng dụng điều khiển thông minh, ta có thể định nghĩa một lớp cơ sở `Screen` với các phương thức ảo như `onLoad()`, `onExit()` và kế thừa cho các lớp cụ thể như `MainScreen`, `SettingsScreen`, giúp tách biệt giao diện và logic điều khiển.

2.2.2 Lý do chọn ngôn ngữ C/C++ cho phát triển giao diện với LVGL

2.2.2.1 LVGL được viết thuần bằng C

LVGL là một thư viện đồ họa mã nguồn mở được thiết kế cho vi điều khiển. Đặc điểm then chốt là nó được viết hoàn toàn bằng C thuần, không có bất kỳ *dependency* nào vào hệ điều hành hoặc thư viện ngoài, giúp:

- Dễ dàng *port* lên bất kỳ nền tảng phần cứng nào.
- Tương thích hoàn toàn với ESP-IDF (framework chính thức của ESP32).
- Tránh *overhead* không cần thiết, rất quan trọng trong các hệ thống thời gian thực (RTOS).

Việc phát triển ứng dụng sử dụng LVGL bằng ngôn ngữ C giúp lập trình viên có thể sử dụng trực tiếp các API như `lv_btn_create()`, `lv_label_set_text()` mà không cần tạo lớp trừu tượng hoặc *wrapper*, đảm bảo hiệu suất tối ưu và độ ổn định cao.

2.2.2.2 Quản lý bộ nhớ tối ưu

ESP32 có lượng RAM giới hạn (320KB SRAM, có thể thêm PSRAM ngoài), do đó việc kiểm soát bộ nhớ là sống còn. Ngôn ngữ C hỗ trợ ba dạng quản lý bộ nhớ:

- **Bộ nhớ tĩnh (static memory)**: sử dụng khi kích thước bộ nhớ đã biết tại thời điểm biên dịch.
- **Bộ nhớ tự động (stack memory)**: dùng cho biến cục bộ, giải phóng khi hàm thoát.
- **Bộ nhớ động (heap memory)**: cấp phát tại *runtime* với `malloc()` và giải phóng với `free()`.

Trong môi trường ESP-IDF, các hàm như `heap_caps_malloc()` cho phép chỉ định vùng bộ nhớ phù hợp: DRAM, PSRAM, DMA-capable... giúp tối ưu hóa hiệu suất và tránh *fragmentation*.

2.2.2.3 C++ hỗ trợ tổ chức GUI theo mô hình hướng đối tượng

Trong dự án GUI với LVGL, việc sử dụng OOP giúp giảm độ phức tạp và nâng cao tính bảo trì:

- **Kế thừa**: các màn hình GUI kế thừa từ một lớp cha có sẵn logic khởi tạo chung.
- **Đa hình**: mỗi màn hình có thể định nghĩa cách hiển thị, xử lý sự kiện khác nhau mà vẫn dùng cùng một *interface*.
- **Tái sử dụng mã**: các *widget* hoặc thành phần như `MessageBox`, `SliderPanel`, `LoadingOverlay` có thể được viết thành `class` và sử dụng lại ở nhiều nơi.

2.2.3 Ưu điểm chi tiết

Đặc điểm	Mô tả chuyên sâu
Tốc độ thực thi nhanh	C/C++ được biên dịch trực tiếp thành mã máy, không thông qua VM hay runtime, giúp giảm độ trễ xử lý – đặc biệt quan trọng trong GUI real-time (animation, touch response).
Tối ưu bộ nhớ	Lập trình viên có thể tinh chỉnh từng byte bộ nhớ sử dụng, giảm nguy cơ out-of-memory hoặc crash runtime.
Mô-đun hóa tốt với C++	Việc chia mã nguồn thành các module (class) giúp dễ bảo trì, tách biệt giao diện với logic, giảm lỗi phát sinh khi mở rộng hệ thống.
Hỗ trợ toolchain mạnh mẽ	Các công cụ như `idf.py`, `esp_log`, `heap_trace`, `perftest`, `OpenOCD` giúp giám sát chi tiết RAM, thời gian xử lý và hiệu suất toàn hệ thống.
Khả năng mở rộng	Có thể kết hợp với các thư viện xử lý logic khác như `protobuf`, `tinyxml`, `json-c` hoặc dùng C++ template để viết các class generic.

2.2.4 Hạn chế và biện pháp khắc phục

Hạn chế	Giải pháp kỹ thuật
Quản lý bộ nhớ thủ công dễ gây *leak*	Sử dụng RAII (Resource Acquisition Is Initialization) trong C++, kết hợp smart pointers nếu áp dụng được.
Runtime error khó phát hiện	Dùng `esp_log`, kiểm tra `null-pointer`, xác thực cấp phát trước khi sử dụng; bật tính năng heap poisoning hoặc heap tracing trong `menuconfig`.
Không có bảo vệ biên tự động	Viết *wrapper* kiểm tra *bound array*, dùng `assert` trong quá trình phát triển.
C++ có thể làm phình mã nhị phân	Tránh dùng các thành phần STL nặng như `<vector>`, `<map>`; thay bằng `struct`/`array` thủ công hoặc các thư viện nhẹ như `etl (Embedded Template Library)`.

2.2.5 Ứng dụng thực tế trong hệ thống dùng LVGL trên ESP32

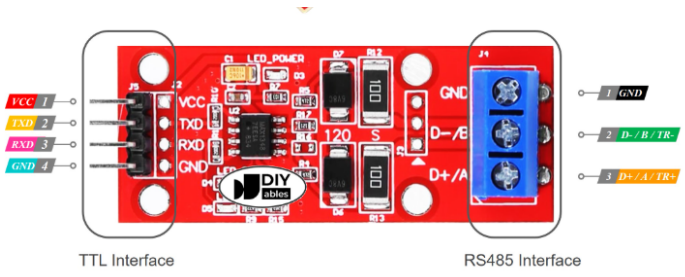
Tầng hệ thống	Ngôn ngữ đề xuất	Giải thích
Driver phần cứng (LCD, touch, backlight)	C	Tận dụng ESP-IDF driver: `esp_lcd_panel_io_spi`, `esp_lcd_touch`, dễ tối ưu, chạy được trong *interrupt context*.
Core xử lý GUI	C/C++	C để dùng API LVGL, C++ để xây dựng các màn hình logic theo hướng object.
Giao tiếp ngoại vi (WiFi, BLE, MQTT)	C	Các *stack* WiFi/Bluetooth/MQTT trong ESP-IDF đều là C thuần.
Quản lý trạng thái màn hình, hoạt cảnh	C++	FSM, State Pattern dễ viết bằng class, hỗ trợ constructor/destructor rõ ràng.
Hệ thống sự kiện (Event Queue)	C++	Có thể dùng template hoặc Event-Bus theo kiểu Observer Pattern.

2.3 Cơ sở lý thuyết về module RS485

2.3.1 Giới thiệu chung về module RS485

Module RS485 là một thiết bị trung gian giúp chuyển đổi tín hiệu UART (TTL) từ vi điều khiển sang chuẩn truyền thông RS485, cho phép giao tiếp trong các hệ thống truyền dữ liệu công nghiệp hoặc khoảng cách xa. Các module này thường được tích hợp sẵn mạch điện trở kéo, bảo vệ điện áp và mạch truyền nhận để đảm bảo tín hiệu ổn định và an toàn trong quá trình truyền thông.

RS485 hỗ trợ giao tiếp đa điểm (multi-point), cho phép kết nối nhiều thiết bị với nhau trên cùng một đường truyền dữ liệu (bus), giúp đơn giản hóa hệ thống dây và giảm chi phí triển khai trong các ứng dụng như hệ thống giám sát, tự động hóa tòa nhà, mạng cảm biến môi trường, v.v.



Hình 2.2: RS485 module

2.3.2 Nguyên lý hoạt động của module RS485

Module RS485 hoạt động dựa trên nguyên lý truyền vi sai (differential signaling), sử dụng hai dây tín hiệu chính là A và B. Tín hiệu được truyền dưới dạng chênh lệch điện áp giữa hai dây này thay vì so với mass như các chuẩn UART hay RS232 thông thường. Khi mức điện áp trên chân A cao hơn B (A > B), tín hiệu được xem là logic "1", ngược lại nếu A < B thì là logic "0". Điều này giúp loại bỏ nhiễu điện từ chung trên đường truyền và duy trì tính toàn vẹn dữ liệu.

Các module RS485 phổ biến hiện nay sử dụng IC chuyển đổi như MAX485, SP485, hoặc SN75176 để thực hiện quá trình mã hóa và giải mã tín hiệu vi sai. Các IC này thường hỗ trợ cả chế độ truyền và nhận (half-duplex), và cần thêm một chân điều khiển (thường là DE/RE) để chuyển đổi giữa hai chế độ.

2.3.3 Kết nối với vi điều khiển (ví dụ ESP32-S3)

Việc kết nối module RS485 với vi điều khiển như ESP32-S3 thường thông qua giao tiếp UART. Các chân TX và RX của vi điều khiển được nối với module RS485 thông qua mạch chuyển đổi mức tín hiệu. Ngoài ra, chân DE (Driver Enable) và RE (Receiver Enable) thường được điều khiển bằng phần mềm để bật chế độ truyền hoặc nhận dữ liệu. Trong nhiều module, DE và RE được nối với nhau và điều khiển chung qua một chân GPIO của vi điều khiển để đơn giản hóa việc lập trình.

Ví dụ, để gửi dữ liệu, chân DE/RE được đặt ở mức HIGH để bật chế độ truyền. Sau khi gửi xong, chân này sẽ được kéo xuống mức LOW để chuyển sang chế độ nhận. Thời điểm chuyển trạng thái cần được lập trình chính xác để tránh mất dữ liệu hoặc xung đột đường truyền.

2.3.4 Ứng dụng thực tế

Module RS485 được sử dụng rộng rãi trong nhiều ứng dụng yêu cầu truyền thông ổn định ở khoảng cách xa như:

- Giao tiếp giữa các bộ điều khiển khả trình (PLC) và cảm biến/thiết bị chấp hành.
- Mạng Modbus RTU trong hệ thống SCADA.
- Mạng cảm biến trong nhà máy, nông nghiệp thông minh, hoặc giám sát năng lượng.
- Hệ thống đo đạc từ xa như đồng hồ điện, nước, và hệ thống kiểm soát truy cập.

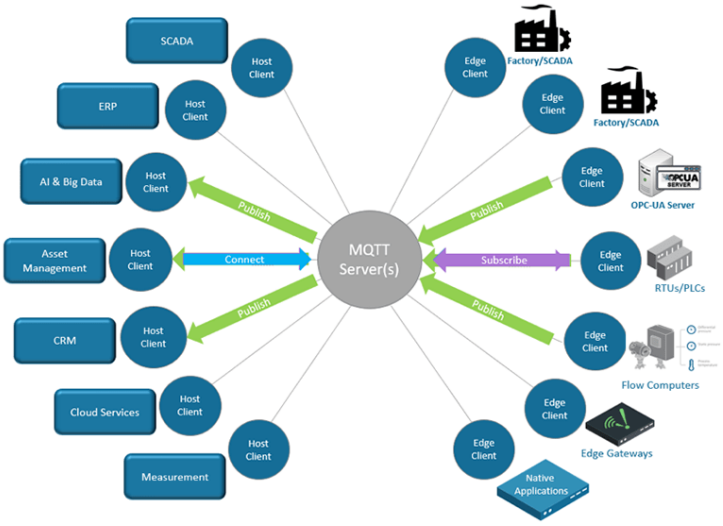
Với khả năng chống nhiễu tốt, chi phí thấp và triển khai dễ dàng, module RS485 là lựa chọn lý tưởng cho các hệ thống truyền thông công nghiệp và IoT.

2.4 MQTT trong IoT

2.4.1 Tổng quan về MQTT

MQTT (Message Queuing Telemetry Transport) là một giao thức truyền thông nhẹ, được thiết kế tối ưu cho các ứng dụng có băng thông thấp, độ trễ cao hoặc yêu cầu tiêu thụ năng lượng thấp — đặc điểm điển hình trong các hệ thống Internet of Things (IoT). Giao thức này hoạt động theo mô hình **publish/subscribe** (xuất bản/đăng ký), cho phép các thiết bị trao đổi dữ liệu thông qua một máy chủ trung gian gọi là **broker**.

MQTT ban đầu được phát triển bởi IBM vào năm 1999 và hiện nay là một chuẩn mở, được quản lý bởi tổ chức OASIS (Organization for the Advancement of Structured Information Standards).

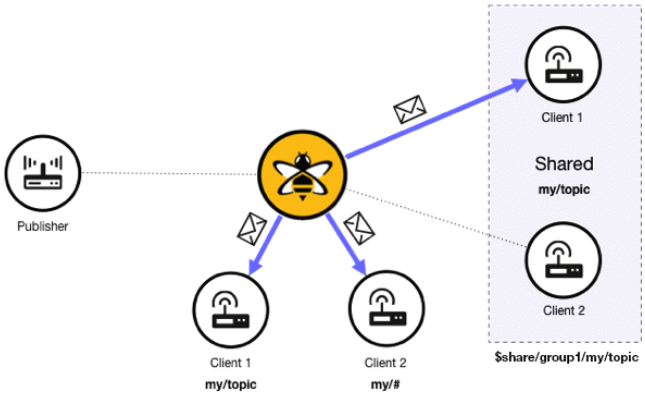


Hình 2.3: MQTT trong IoT

2.4.2 Kiến trúc MQTT

Kiến trúc MQTT bao gồm ba thành phần chính:

- **Publisher (Thiết bị xuất bản):** Gửi thông điệp đến một chủ đề (topic) cụ thể.
- **Subscriber (Thiết bị đăng ký):** Nhận thông điệp bằng cách đăng ký các chủ đề quan tâm.
- **Broker (Máy chủ trung gian):** Tiếp nhận các thông điệp từ publisher và phân phối chúng đến các subscriber phù hợp. Ví dụ broker phổ biến: Mosquitto, HiveMQ, EMQX.



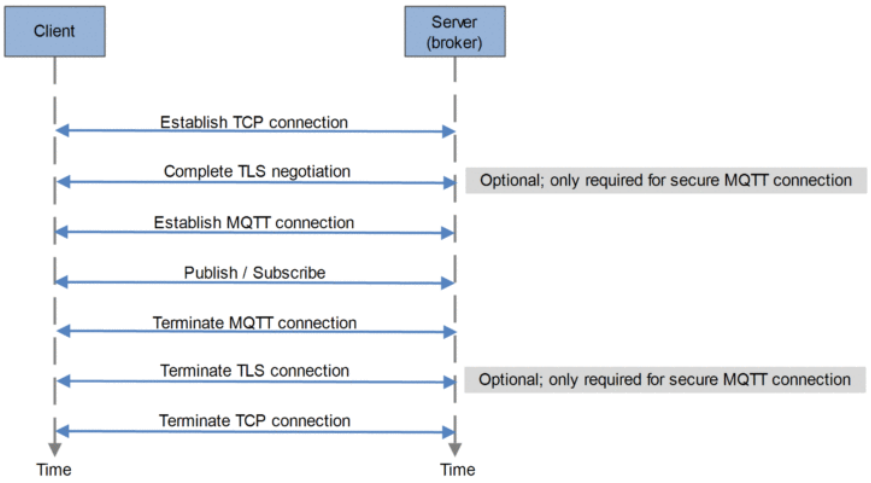
Hình 2.4: Mô hình Publish/Subscriber trong giao thức MQTT

2.4.3 Nguyên lý hoạt động

Khi một thiết bị **publisher** gửi một thông điệp (message) đến một **topic**, **broker** sẽ tiếp nhận thông điệp và phân phối nó đến tất cả các thiết bị **subscriber** đã đăng ký với topic đó. Giao thức MQTT cho phép hệ thống mở rộng dễ dàng và giảm đáng kể độ phụ thuộc giữa các thiết bị đầu cuối.

Ví dụ:

- Publisher gửi dữ liệu cảm biến đến topic "`/home/temperature`".
- Tất cả các subscriber quan tâm đến chủ đề này sẽ nhận được dữ liệu mà không cần biết thông tin về publisher.



Hình 2.5: Cơ chế hoạt động của giao thức MQTT

2.4.4 Cấu trúc topic

Topic trong MQTT được tổ chức theo dạng cây, sử dụng dấu gạch chéo `/` để phân tách các cấp.

Ví dụ:

- `/home/livingroom/temperature`
- `/device/esp32/status`

MQTT hỗ trợ ký tự đại diện:

- `+` đại diện cho một cấp (level).
- `#` đại diện cho nhiều cấp (từ vị trí hiện tại trở đi).

2.4.5 Chất lượng dịch vụ (QoS)

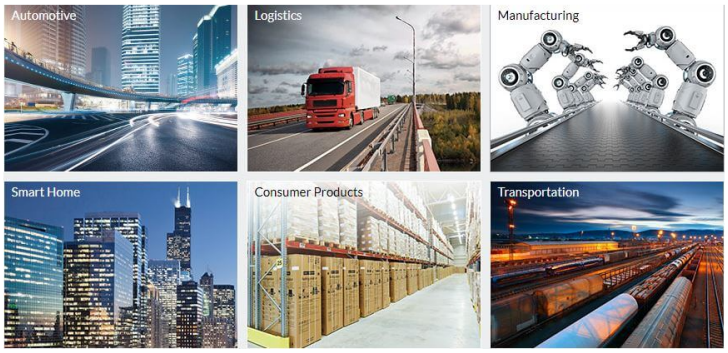
MQTT hỗ trợ ba mức độ đảm bảo chất lượng dịch vụ (Quality of Service - QoS):

- **QoS 0 - At most once:** Gửi một lần và không đảm bảo nhận được.
- **QoS 1 - At least once:** Gửi ít nhất một lần, có thể trùng lặp.
- **QoS 2 - Exactly once:** Gửi đúng một lần (đảm bảo không trùng và không mất).

2.4.6 Ứng dụng của MQTT trong IoT

MQTT được sử dụng rộng rãi trong nhiều hệ thống IoT nhờ vào tính nhẹ, đơn giản và hiệu quả:

- **Nhà thông minh (Smart home):** Kết nối thiết bị đèn, cảm biến, điều hòa.
- **Nông nghiệp thông minh:** Gửi dữ liệu độ ẩm, nhiệt độ, mực nước về máy chủ.
- **Giám sát công nghiệp:** Truyền dữ liệu cảm biến từ xa về trung tâm điều khiển.
- **Thiết bị đeo và sức khỏe:** Đồng bộ hóa dữ liệu sức khỏe với cloud server.



Hình 2.6: Ứng dụng của MQTT trong IoT

2.4.7 Ưu và nhược điểm của MQTT

Ưu điểm:

- Giao thức nhẹ, phù hợp với thiết bị tài nguyên hạn chế.
- Giao tiếp không đồng bộ và tách biệt giữa publisher và subscriber.
- Hỗ trợ nhiều mức QoS và cơ chế giữ kết nối (keep-alive).

Nhược điểm:

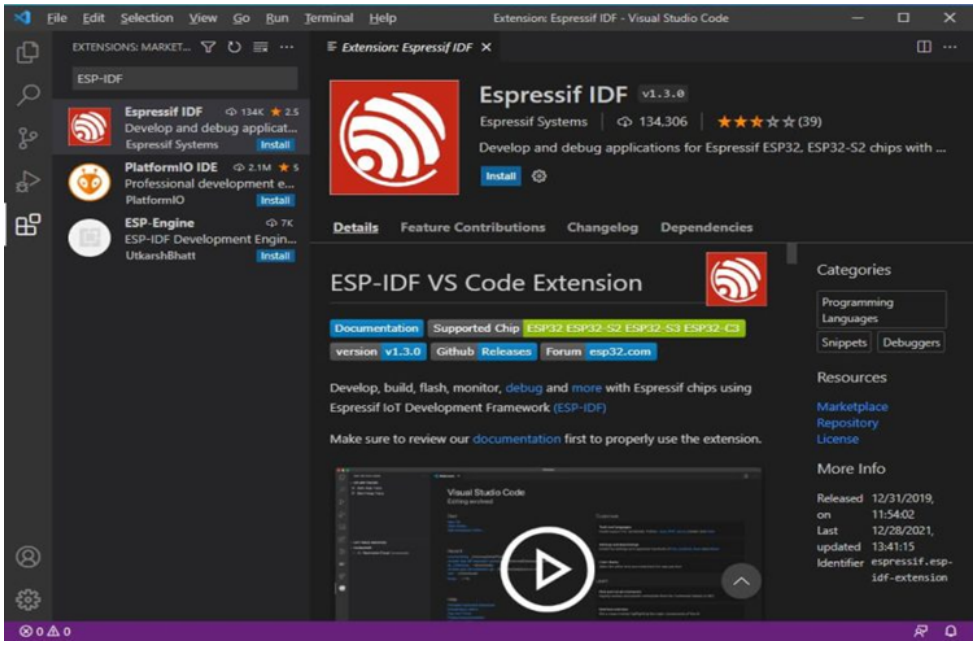
- Phụ thuộc vào broker trung gian.
- Không mã hóa mặc định, cần kết hợp với SSL/TLS để đảm bảo an toàn.
- Không phù hợp với các ứng dụng yêu cầu thời gian thực chính xác tuyệt đối.

2.5 Thư viện và Framework liên quan

2.5.1 ESP-IDF (Espressif IoT Development Framework)

2.5.1.1 Giới thiệu tổng quan về ESP-IDF

ESP-IDF là framework chính thức do Espressif Systems phát triển nhằm hỗ trợ lập trình hệ thống cho các vi điều khiển dòng ESP32. Đây là một môi trường phát triển toàn diện, bao gồm hệ thống build, driver phần cứng, thư viện giao tiếp mạng, công cụ quản lý dự án, và hỗ trợ tích hợp nhiều công cụ debug, phân tích hiệu suất.
ESP-IDF được viết chủ yếu bằng ngôn ngữ C và hoạt động theo mô hình cấu trúc thấp tầng (low-level), cho phép người lập trình kiểm soát chi tiết các thành phần phần cứng, rất phù hợp trong các dự án nhúng chuyên nghiệp.



Hình 2.7: ESP-IDF

2.5.1.2 Vai trò của ESP-IDF trong phát triển giao diện LVGL

Trong bối cảnh ứng dụng LVGL trên ESP32, ESP-IDF đóng vai trò "xương sống" giúp kết nối giữa thư viện giao diện (LVGL) với phần cứng như màn hình LCD, bộ cảm ứng chạm, hoặc các giao tiếp ngoại vi (I2C, SPI). Ngoài ra, nó còn cung cấp:

- Hệ điều hành thời gian thực FreeRTOS tích hợp sẵn, giúp xử lý song song các tác vụ: giao diện, WiFi, Bluetooth, sensor...
- ESP Timer giúp định kỳ gọi `lv_timer_handler()` — hàm cốt lõi của LVGL để xử lý giao diện.
- ESP LCD: tập hợp các driver hiển thị LCD (SPI, RGB, I80), có thể liên kết dễ dàng với LVGL thông qua `disp_flush_cb()`.
- ESP LCD Touch: thư viện hỗ trợ cảm ứng điện dung hoặc điện trở (FT6x36, XPT2046, GT911, v.v.), liên kết với `indev_read_cb()` của LVGL.

2.5.1.3 Các module và thành phần quan trọng trong ESP-IDF

Thành phần	Vai trò chính	Liên quan đến LVGL
esp_lcd_panel_io_spi	Giao tiếp SPI với LCD	Cần cấu hình chuẩn DMA, SPI tốc độ cao
esp_lcd_touch	Giao tiếp với cảm ứng	Trả về tọa độ nhấn chạm cho LVGL
esp_timer	Hẹn giờ định kỳ	Gọi `lv_timer_handler()` với độ trễ cấu hình
heap_caps	Phân tích sử dụng RAM	Dùng để theo dõi leak RAM do `lv_obj` gây ra
esp_log	Ghi log debug	Log lỗi, sự kiện UI, quản lý bộ nhớ
freertos/task	Quản lý task, priority	Tạo task riêng cho GUI, ưu tiên cao hơn các tác vụ nền

Bảng 2.1: Các thành phần ESP-IDF quan trọng liên quan đến LVGL

2.5.1.4 Cách tích hợp LVGL vào ESP-IDF

Tích hợp LVGL vào ESP-IDF yêu cầu thực hiện các bước cơ bản sau:

1. Thêm thư viện LVGL vào project:

   ```
   git submodule add https://github.com/lvgl/lvgl components/lvgl
   ```

2. Khai báo trong `CMakeLists.txt` hoặc `idf_component.yml`.

3. Cấu hình Timer và Display:

   - Sử dụng `esp_timer_create()` để tạo timer định kỳ 5–10ms gọi `lv_timer_handler()`.
   - Tạo driver LCD qua `esp_lcd_panel_io_spi_new()` và `esp_lcd_panel_new_st7789()`.

4. Cấu hình Touch:

   - Dùng `esp_lcd_touch_new_i2c()` để kết nối với driver cảm ứng.
   - Tạo callback `indev_read_cb()` để LVGL nhận dữ liệu từ cảm ứng.

2.5.1.5 Ưu điểm của ESP-IDF khi dùng với LVGL

Ưu điểm	Mô tả chi tiết
Tối ưu cho phần cứng ESP32	Tương thích hoàn toàn với các peripheral của ESP32: SPI, DMA, RGB, Touch
Quản lý bộ nhớ tốt	Cung cấp heap tracker, log phân tích fragment
Hệ thống log mạnh	Phân luồng log theo TAG, độ ưu tiên
Cấu hình linh hoạt	Qua `menuconfig`, dễ cấu hình SPI clock, DMA, PSRAM
Tích hợp OTA, WiFi, BLE	Có thể phát triển GUI cho hệ thống kết nối IoT

Bảng 2.2: Ưu điểm ESP-IDF khi tích hợp với LVGL

2.5.1.6 Nhược điểm và lưu ý khi sử dụng ESP-IDF

Nhược điểm	Biện pháp khắc phục
Cấu hình phức tạp cho người mới	Sử dụng template project, PlatformIO hỗ trợ
Dung lượng flash chiếm nhiều	Tối ưu `menuconfig`, tắt debug, loại bỏ unused component
Đòi hỏi hiểu rõ cấu trúc ESP-IDF	Tham khảo tài liệu chính thức, examples từ Espressif
Yêu cầu dùng CMake	Học cú pháp `CMakeLists.txt`, PlatformIO đã tự động hóa

Bảng 2.3: Một số lưu ý khi dùng ESP-IDF

2.5.2 Arduino Framework

2.5.2.1 Giới thiệu tổng quan về Arduino Framework

Arduino là một nền tảng mã nguồn mở nổi tiếng, cung cấp một lớp trừu tượng cao hơn để lập trình vi điều khiển. Arduino framework đơn giản hóa việc phát triển nhúng bằng cách ẩn đi phần lớn cấu hình phần cứng phức tạp và cung cấp một tập hợp API thân thiện.

Trên ESP32, Arduino hoạt động như một lớp wrapper bên trên ESP-IDF, nghĩa là người dùng vẫn có thể truy cập các API ESP-IDF nếu cần, nhưng trong đa số trường hợp, họ chỉ cần dùng các hàm Arduino như `digitalWrite()`, `analogRead()`, `WiFi.begin()`, v.v.

2.5.2.2 Lý do Arduino vẫn được lựa chọn khi phát triển giao diện với LVGL

Dù không mạnh bằng ESP-IDF về khả năng kiểm soát thấp tầng, Arduino framework vẫn được nhiều nhà phát triển lựa chọn trong các trường hợp sau:

- Phát triển nhanh prototype GUI nhờ cú pháp đơn giản và tài liệu dễ tra cứu.
- Đã có thư viện LVGL tích hợp sẵn: `lvgl_Arduino`, `TFT_eSPI + LVGL binding`.
- Thư viện cộng đồng mạnh: sensor, MQTT, Firebase...

2.5.2.3 Kiến trúc Arduino trên ESP32

Thành phần	**Vai trò**	**Ghi chú**
setup() và loop()	Hàm chính người dùng	Thay cho `main()` trong C truyền thống
Arduino Core for ESP32	Giao tiếp giữa API Arduino và ESP-IDF	Do Espressif phát triển
SPI.h, Wire.h	Giao tiếp SPI/I2C đơn giản	Wrapper của ESP-IDF
lvgl.h	API chính của GUI	Phải khởi tạo LVGL, buffer, flush...

Bảng 2.4: Kiến trúc Arduino trên ESP32

2.5.2.4 Ưu điểm của Arduino Framework

Ưu điểm	**Giải thích chi tiết**
Cú pháp đơn giản	Dễ học, dễ dùng, phù hợp sinh viên và người mới tiếp cận nhúng
Nhiều thư viện hỗ trợ	LVGL, TFT_eSPI, Touch, WiFi, MQTT, WebServer có sẵn và hoạt động ổn định
Tích hợp tốt với PlatformIO	Có thể dùng trên VS Code, dễ debug, tự động build/upload
Giao diện lập trình cao cấp	Viết nhanh hơn, dễ tổ chức code nhờ class đơn giản và lambda callback
Cộng đồng lớn	Dễ tìm ví dụ, diễn đàn hỗ trợ nhanh chóng

Bảng 2.5: Ưu điểm của Arduino framework

2.5.2.5 Hạn chế của Arduino khi dùng với LVGL

Hạn chế	Mô tả	Hướng khắc phục
Thiếu kiểm soát sâu	Không can thiệp DMA, SPI nâng cao	Dùng API ESP-IDF trực tiếp
Không hỗ trợ menuconfig	Không tùy biến thông số hệ thống	Dùng build flags với PlatformIO
Hạn chế đa task	`loop()` là một task duy nhất	Dùng `xTaskCreate()`
Không tối ưu RAM	Không giám sát fragmentation	Dùng `heap_caps_get_free_size()` nếu có kinh nghiệm

Bảng 2.6: Hạn chế của Arduino khi tích hợp với LVGL

2.5.2.6 Kịch bản sử dụng Arduino phù hợp

- Sinh viên, người mới bắt đầu học nhúng.
- Dự án nhỏ, thời gian gấp rút.
- Không cần tối ưu sâu về hiệu suất.
- Dự án tận dụng thư viện cộng đồng.

2.5.3 FreeRTOS

2.5.3.1 Giới thiệu tổng quan về FreeRTOS

FreeRTOS (Free Real-Time Operating System) là một hệ điều hành thời gian thực nhẹ, mã nguồn mở, được thiết kế đặc biệt cho hệ thống nhúng. ESP-IDF tích hợp sẵn FreeRTOS như một phần lõi, và tất cả các ứng dụng chạy trên ESP32 đều hoạt động theo mô hình đa nhiệm của FreeRTOS.
FreeRTOS cung cấp khả năng chạy đồng thời nhiều tác vụ (task), hỗ trợ ưu tiên, đồng bộ hóa bằng semaphore, mutex và queue, rất cần thiết trong ứng dụng giao diện người dùng, nơi cần xử lý song song cảm ứng, WiFi, và logic nền.

2.5.3.2 Vai trò của FreeRTOS trong hệ thống giao diện sử dụng LVGL

LVGL là thư viện GUI cần được cập nhật định kỳ bằng cách gọi hàm `lv_timer_handler()` (trước đây là `lv_task_handler()`). Do đó, việc sử dụng FreeRTOS giúp:

- Chạy `lv_timer_handler()` trong task riêng biệt, tránh ảnh hưởng đến các logic khác như WiFi, MQTT.

- Quản lý đa màn hình phức tạp, mỗi màn hình hoặc nhóm widget có thể liên kết với một task riêng để xử lý logic nội bộ.
- Đồng bộ hóa dữ liệu đa nguồn, ví dụ cập nhật nhiệt độ từ sensor và hiển thị lên giao diện thông qua queue hoặc message buffer.
- Ưu tiên xử lý GUI để đảm bảo trải nghiệm người dùng mượt mà, không bị lag hay giật.
- Tối ưu hóa tài nguyên hệ thống, tránh tình trạng tràn stack hoặc memory leak.

2.5.3.3 Cách tích hợp FreeRTOS với LVGL

- `lv_timer_handler()`: xử lý redraw GUI.
- `vTaskDelay()`: delay trong vòng lặp GUI.
- `xSemaphoreTake()`: đồng bộ touch ISR và GUI task.
- `xQueueSend()`, `xQueueReceive()`: truyền dữ liệu giữa các task.

Ví dụ:

```
void lvgl_task(void* param) {
    while (1) {
        lv_timer_handler(); // Xử lý GUI
        vTaskDelay(pdMS_TO_TICKS(5)); // Delay 5ms
    }
}
```

2.5.3.4 Ưu điểm của FreeRTOS

Ưu điểm	**Mô tả chi tiết**
Đa nhiệm thực thụ	Cho phép xử lý song song WiFi, cảm biến, touch, GUI... mà không block nhau
Quản lý ưu tiên	Có thể đặt task GUI với mức ưu tiên cao hơn để luôn đảm bảo trải nghiệm mượt
Tối ưu tài nguyên	Có thể cấu hình kích thước stack phù hợp cho từng task, tránh lãng phí RAM
Công cụ debug tốt	ESP-IDF có task monitor, log theo từng core, theo dõi task alive
Cấu trúc phần mềm rõ ràng	Phân chia task theo chức năng giúp code dễ bảo trì, mở rộng

Bảng 2.7: Ưu điểm của FreeRTOS

2.5.3.5 Hạn chế và cách khắc phục

Hạn chế	Giải pháp
Cần hiểu rõ lập trình đa nhiệm	Học kỹ về race condition, deadlock, semaphore, mutex
Dễ bị tràn stack task	Dùng `uxTaskGetStackHighWaterMark()` để kiểm tra stack thực tế
Có thể phức tạp với người mới	Bắt đầu từ 1-2 task, dần mở rộng; sử dụng wrapper hoặc layer trung gian để đơn giản hóa

Bảng 2.8: Hạn chế và cách khắc phục trong FreeRTOS

2.5.3.6 Tình huống sử dụng điển hình

Kịch bản	Vai trò FreeRTOS
Cập nhật GUI liên tục theo sensor	Dùng task riêng đọc sensor → gửi queue → GUI task hiển thị
Giao tiếp song song WiFi và touch	WiFi chạy trong task riêng, GUI có thể phản hồi ngay cả khi WiFi mất kết nối
Nhiều màn hình có hiệu ứng animation	GUI task xử lý hiệu ứng, logic chuyển trạng thái đặt trong các task phụ trợ

Bảng 2.9: Tình huống sử dụng điển hình của FreeRTOS

2.5.4 Thư viện đồ họa LVGL

LVGL (Light and Versatile Graphics Library) là một thư viện đồ họa mã nguồn mở, nhẹ nhàng và linh hoạt, được thiết kế đặc biệt cho các hệ thống nhúng với tài nguyên hạn chế. Thư viện này cung cấp tất cả các công cụ cần thiết để tạo ra giao diện người dùng đồ họa (GUI) hấp dẫn và chuyên nghiệp trên các vi điều khiển như ESP32, với hiệu suất tối ưu và yêu cầu bộ nhớ thấp.

2.5.4.1 Tổng quan về LVGL

LVGL được phát triển từ năm 2016 bởi Gábor Kiss-Vámosi, ban đầu với tên gọi LittlevGL. Qua nhiều năm phát triển, LVGL đã trở thành một trong những thư viện GUI phổ biến nhất cho các hệ thống nhúng, với các phiên bản chính từ v5 đến v9 hiện nay. Mỗi phiên bản đều mang đến những cải tiến đáng kể về hiệu suất, tính năng và khả năng tương thích.

LVGL có nhiều đặc điểm và ưu điểm nổi bật so với các thư viện đồ họa khác cho hệ thống nhúng. Thư viện này được thiết kế với kiến trúc module hóa, cho phép chỉ biên dịch các thành phần cần thiết, giảm thiểu kích thước mã nguồn. LVGL hỗ trợ đa nền

tảng, có thể chạy trên hầu hết các vi điều khiển 16, 32 hoặc 64 bit, từ các chip đơn giản như STM32 đến các nền tảng mạnh mẽ hơn như ESP32 hoặc Raspberry Pi.

Một trong những ưu điểm lớn nhất của LVGL là khả năng tạo ra giao diện người dùng hiện đại và hấp dẫn với yêu cầu tài nguyên tối thiểu. Thư viện có thể hoạt động với chỉ 64 KB flash và 16 KB RAM, mặc dù cấu hình được khuyến nghị là khoảng 180 KB flash và 48 KB RAM để sử dụng đầy đủ tính năng. LVGL cũng hỗ trợ hoạt động với chỉ một frame buffer, giảm thiểu yêu cầu bộ nhớ đồng thời vẫn duy trì hiệu ứng đồ họa mượt mà.

Về yêu cầu hệ thống và tương thích, LVGL có thể hoạt động trên hầu hết các vi điều khiển với tốc độ xung nhịp từ 16 MHz trở lên, mặc dù khuyến nghị tối thiểu 32 MHz cho hiệu suất tốt. Thư viện hỗ trợ nhiều loại màn hình với độ sâu màu khác nhau, từ màn hình đơn sắc đến màn hình TFT màu 16 hoặc 24 bit. LVGL cũng tương thích với nhiều loại thiết bị đầu vào như màn hình cảm ứng, nút bấm, encoder, và bàn phím.

2.5.4.2 Kiến trúc LVGL

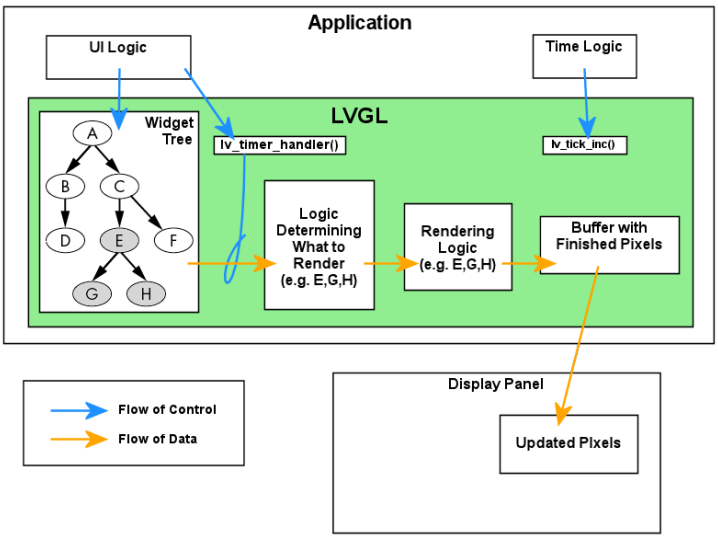
LVGL được xây dựng với kiến trúc module hóa, bao gồm các thành phần cốt lõi và các module chức năng. Cấu trúc này cho phép tùy chỉnh và mở rộng linh hoạt, đồng thời duy trì hiệu suất tối ưu cho các hệ thống nhúng.

Cấu trúc lõi của LVGL bao gồm các module cơ bản như hệ thống đối tượng, quản lý bộ nhớ, hệ thống sự kiện, và engine vẽ. Các module này tạo nền tảng cho toàn bộ thư viện và cung cấp các chức năng cơ bản như quản lý đối tượng, xử lý sự kiện, và render đồ họa. LVGL sử dụng kiến trúc hướng đối tượng mặc dù được viết bằng C, với các cấu trúc dữ liệu và hàm được tổ chức theo cách mô phỏng lập trình hướng đối tượng.

Hệ thống đối tượng và thừa kế trong LVGL cho phép tạo ra các widget phức tạp từ các thành phần cơ bản. Mỗi đối tượng trong LVGL đều kế thừa từ lớp cơ sở `lv_obj`, thừa hưởng các thuộc tính và phương thức của lớp cha, đồng thời có thể mở rộng với các chức năng riêng. Cơ chế thừa kế này giúp giảm thiểu mã lặp lại và tạo ra hệ thống widget nhất quán và dễ mở rộng.

Cơ chế quản lý bộ nhớ trong LVGL được thiết kế đặc biệt cho các hệ thống nhúng với bộ nhớ hạn chế. LVGL sử dụng bộ cấp phát bộ nhớ động riêng, cho phép kiểm soát chặt chẽ việc sử dụng bộ nhớ và tránh phân mảnh. Thư viện cũng hỗ trợ các cơ chế tái sử dụng bộ nhớ và giải phóng tự động, giảm thiểu rò rỉ bộ nhớ và tối ưu hóa hiệu suất.

Hệ thống sự kiện và callback trong LVGL cho phép xử lý tương tác người dùng một cách linh hoạt và hiệu quả. Mỗi đối tượng có thể đăng ký các hàm callback để phản hồi với các sự kiện như nhấn, kéo, hoặc thay đổi giá trị. Hệ thống sự kiện hỗ trợ lan truyền sự kiện qua cây đối tượng, cho phép xử lý sự kiện ở nhiều cấp độ khác nhau.



Hình 2.8: Tổng quan về luồng dữ liệu của LVGL

2.5.4.3 Các thành phần chính giao diện

LVGL cung cấp một bộ widget phong phú để xây dựng giao diện người dùng, từ các thành phần cơ bản đến các container phức tạp và hệ thống bố cục linh hoạt.
Hệ thống widget cơ bản của LVGL bao gồm các thành phần như Button (nút bấm), Label (nhãn), và Image (hình ảnh). Button cung cấp chức năng tương tác cơ bản, với hỗ trợ cho các trạng thái khác nhau như nhấn, thả, và vô hiệu hóa. Label cho phép hiển thị văn bản với nhiều tùy chọn về font, căn chỉnh, và cuộn. Image hỗ trợ nhiều định dạng hình ảnh, bao gồm cả hình ảnh nén và hình ảnh với kênh alpha.

Ngoài các widget cơ bản, LVGL còn cung cấp các thành phần phức tạp hơn như Chart (biểu đồ), Table (bảng), Dropdown (danh sách thả xuống), và Keyboard (bàn phím ảo). Các widget này cho phép xây dựng giao diện người dùng phức tạp với ít mã nguồn hơn, đồng thời duy trì hiệu suất tốt trên các hệ thống nhúng.

Hệ thống styles và themes trong LVGL cho phép tùy chỉnh giao diện một cách linh hoạt và nhất quán. Styles định nghĩa các thuộc tính như màu sắc, font, padding, và border cho các widget. LVGL sử dụng cơ chế CSS-like để áp dụng styles, cho phép kế thừa và ghi đè các thuộc tính. Themes là tập hợp các styles được định nghĩa sẵn, cung cấp giao diện nhất quán cho toàn bộ ứng dụng.

Bố cục và container trong LVGL giúp tổ chức các widget một cách linh hoạt và phản ứng. LVGL hỗ trợ các hệ thống bố cục hiện đại như Flex (dựa trên CSS Flexbox) và Grid (dựa trên CSS Grid), cho phép tạo ra giao diện phản ứng tự động điều chỉnh theo kích thước màn hình. Các container như Panel, Tabview, và Window giúp tổ chức nội dung thành các phần logic và cung cấp các chức năng như cuộn và chuyển tab.

Hiệu ứng đồ họa và animation trong LVGL mang lại trải nghiệm người dùng động và hấp dẫn. Thư viện hỗ trợ các hiệu ứng như fade, move, và scale, với khả năng tùy chỉnh thời gian, đường cong chuyển động, và callback. Animations có thể được áp dụng cho hầu hết các thuộc tính của widget, từ vị trí và kích thước đến màu sắc và độ trong suốt.

2.6 Công nghệ hiển thị cho hệ thống nhúng

Công nghệ hiển thị đóng vai trò quan trọng trong việc phát triển giao diện người dùng cho các hệ thống nhúng như ESP32. Các loại màn hình khác nhau cung cấp nhiều tùy chọn về kích thước, độ phân giải, chất lượng hiển thị và phương thức giao tiếp, cho phép nhà phát triển lựa chọn giải pháp phù hợp nhất với yêu cầu của dự án.

2.6.1 Màn hình LCD và TFT

Màn hình LCD (Liquid Crystal Display) và đặc biệt là công nghệ TFT (Thin Film Transistor) LCD đã trở thành lựa chọn phổ biến cho các ứng dụng nhúng nhờ khả năng hiển thị màu sắc sống động, độ phân giải cao và giá thành hợp lý.

Nguyên lý hoạt động của màn hình LCD dựa trên việc điều khiển các tinh thể lỏng để thay đổi cách ánh sáng đi qua chúng. Mỗi điểm ảnh (pixel) trên màn hình LCD bao gồm các tinh thể lỏng được đặt giữa hai tấm phân cực. Khi áp dụng điện áp, các tinh thể lỏng xoay và thay đổi cách ánh sáng đi qua, tạo ra các mức độ sáng tối khác nhau. Trong màn hình TFT LCD, mỗi pixel được điều khiển bởi một hoặc nhiều transistor màng mỏng, cho phép điều khiển độc lập và cải thiện đáng kể chất lượng hiển thị so với LCD thông thường.

Các loại màn hình phổ biến cho ESP32 bao gồm ILI9341 và ST7789. Màn hình ILI9341 thường có kích thước 2.8 inch với độ phân giải 240x320 pixel, trong khi ST7789 thường được sử dụng trong các màn hình có kích thước từ 1.3 đến 2.0 inch với độ phân giải tương tự. Cả hai loại controller này đều hỗ trợ giao tiếp SPI, cho phép kết nối dễ dàng với ESP32 và tiêu thụ ít chân GPIO.

Các đặc tính kỹ thuật quan trọng của màn hình TFT LCD bao gồm độ phân giải, độ sâu màu, góc nhìn và tốc độ làm mới. Độ phân giải thông thường cho các màn hình nhỏ dao động từ 240x240 đến 480x320 pixel. Độ sâu màu thường là 16-bit (65,536 màu) hoặc 18-bit (262,144 màu). Các yếu tố như góc nhìn và độ sáng cũng đóng vai trò quan trọng đối với trải nghiệm người dùng.

2.6.2 Màn hình OLED

Màn hình OLED (Organic Light Emitting Diode) là công nghệ hiển thị tiên tiến với nhiều ưu điểm so với LCD, đặc biệt phù hợp cho các ứng dụng nhúng yêu cầu tiết kiệm năng lượng và chất lượng hiển thị cao.

Nguyên lý hoạt động của màn hình OLED dựa trên các diode phát quang hữu cơ có khả năng tự phát sáng khi có dòng điện đi qua. Mỗi pixel là một diode phát quang độc lập, không cần đèn nền. Điều này cho phép tỷ lệ tương phản cao, thời gian đáp ứng nhanh, góc nhìn rộng và tiêu thụ năng lượng thấp khi hiển thị nội dung tối.

So với LCD/TFT, OLED có tỷ lệ tương phản và độ sống động màu sắc cao hơn, thời gian đáp ứng nhanh hơn (dưới 1ms), và tiêu thụ năng lượng thấp hơn. Tuy nhiên, OLED có tuổi thọ ngắn hơn, dễ bị hiện tượng burn-in và chi phí cao hơn.

Trong hệ thống nhúng, OLED phù hợp cho thiết bị đeo, thiết bị y tế cầm tay và các thiết bị IoT chạy pin. Nó cũng là lựa chọn lý tưởng trong môi trường ánh sáng yếu.

2.6.3 Giao tiếp với màn hình

Giao tiếp giữa ESP32 và màn hình là yếu tố ảnh hưởng đến hiệu suất và độ phức tạp của hệ thống. Các giao thức phổ biến gồm SPI, I2C và giao tiếp song song.

- **SPI (Serial Peripheral Interface):** phổ biến cho màn hình TFT và OLED. Giao thức này sử dụng các đường MOSI, MISO, SCK và CS, có tốc độ cao (lên đến 80MHz). ESP32 có thể dùng DMA để cải thiện hiệu suất truyền dữ liệu.
- **I2C (Inter-Integrated Circuit):** phù hợp với màn hình OLED nhỏ và controller cảm ứng. Chỉ sử dụng hai dây SDA và SCL, giúp tiết kiệm GPIO. Nhược điểm là tốc độ thấp hơn (100kHz đến 1MHz).
- **Giao tiếp song song (Parallel):** cung cấp băng thông cao nhất nhưng cần nhiều GPIO. Giao tiếp 8-bit hoặc 16-bit được hỗ trợ qua giao diện I8080/6800.

Về điều khiển cảm ứng, các controller như XPT2046 (cảm ứng điện trở) và FT6X36 (cảm ứng điện dung) sử dụng SPI hoặc I2C. Cảm ứng điện dung hỗ trợ multi-touch và có trải nghiệm tốt hơn, nhưng phức tạp và đắt hơn cảm ứng điện trở.

Việc lựa chọn công nghệ hiển thị và phương thức giao tiếp phù hợp là yếu tố then chốt để tạo ra giao diện người dùng hiệu quả trên nền tảng ESP32. Kết hợp với thư viện như LVGL sẽ nâng cao trải nghiệm người dùng.

2.7 Phân tích các giải pháp GUI cho ESP32

Khi phát triển ứng dụng với giao diện đồ họa người dùng (GUI) cho ESP32, nhà phát triển có nhiều lựa chọn về thư viện đồ họa. Mỗi thư viện có những ưu điểm, nhược điểm và trường hợp sử dụng riêng. Việc phân tích và so sánh các giải pháp này giúp lựa chọn công cụ phù hợp nhất cho dự án cụ thể.

2.7.1 Các thư viện đồ họa hiện có

Thị trường thư viện đồ họa cho ESP32 khá đa dạng, với nhiều lựa chọn từ các thư viện đơn giản, nhẹ nhàng đến các framework GUI toàn diện. Dưới đây là phân tích chi tiết về các thư viện phổ biến nhất.

2.7.1.1 TFT_eSPI (Bodmer)

TFT_eSPI là một thư viện đồ họa mạnh mẽ và được tối ưu hóa cao cho ESP32 và các vi điều khiển khác, được phát triển bởi Bodmer. Thư viện này được thiết kế đặc biệt cho màn hình TFT sử dụng giao tiếp SPI.

Ưu điểm chính của TFT_eSPI bao gồm hiệu suất cao nhờ tối ưu hóa mã assembly và sử dụng DMA, hỗ trợ nhiều loại controller màn hình (ILI9341, ST7789, ILI9488, ...), và tích hợp tốt với hệ sinh thái Arduino. Thư viện cung cấp các hàm vẽ cơ bản như điểm, đường, hình chữ nhật, hình tròn, và văn bản, cùng với hỗ trợ hiển thị hình ảnh và sprite (đối tượng đồ họa có thể di chuyển).

Tuy nhiên, TFT_eSPI có một số hạn chế. Thư viện không cung cấp các widget GUI cao cấp như nút bấm, thanh trượt, hoặc menu, khiến việc xây dựng giao diện người dùng phức tạp trở nên khó khăn hơn. Cấu hình thư viện cũng khá phức tạp, đòi hỏi chỉnh sửa file `User_Setup.h` để phù hợp với phần cứng cụ thể.

TFT_eSPI phù hợp nhất cho các ứng dụng yêu cầu hiệu suất cao, hiển thị đồ họa cơ bản, hoặc khi nhà phát triển muốn xây dựng GUI tùy chỉnh từ đầu.

2.7.1.2 Adafruit GFX

Adafruit GFX là một thư viện đồ họa phổ biến và dễ sử dụng, được phát triển bởi Adafruit Industries. Thư viện này cung cấp một API nhất quán cho nhiều loại màn hình khác nhau, từ OLED đơn sắc đến TFT màu.

Ưu điểm của Adafruit GFX bao gồm tính đơn giản và dễ học, hỗ trợ rộng rãi từ cộng đồng, và tương thích với nhiều loại màn hình thông qua các thư viện driver riêng (như `Adafruit_ILI9341`, `Adafruit_SSD1306`). Thư viện cung cấp các hàm vẽ cơ bản tương tự

như TFT_eSPI, cùng với hỗ trợ font và hiển thị bitmap.

Tuy nhiên, Adafruit GFX không được tối ưu hóa cho hiệu suất cao như TFT_eSPI, đặc biệt trên ESP32. Thư viện cũng thiếu các widget GUI và hệ thống quản lý sự kiện, đòi hỏi nhà phát triển phải xây dựng các thành phần này từ đầu.

Adafruit GFX là lựa chọn tốt cho người mới bắt đầu, các dự án đơn giản, hoặc khi cần tương thích với nhiều loại màn hình khác nhau trong cùng một codebase.

2.7.1.3 U8g2

U8g2 là một thư viện đồ họa monochrome (đơn sắc) được tối ưu hóa cho các màn hình OLED và LCD đơn sắc. Mặc dù chủ yếu tập trung vào hiển thị đơn sắc, U8g2 vẫn là một lựa chọn quan trọng trong hệ sinh thái ESP32.

Ưu điểm của U8g2 bao gồm kích thước nhỏ gọn, tiêu thụ bộ nhớ thấp, và hỗ trợ rộng rãi cho các controller màn hình đơn sắc (SSD1306, SH1106, ...). Thư viện cung cấp nhiều font với kích thước khác nhau, hỗ trợ nhiều ngôn ngữ, và có cơ chế buffer kép để tránh hiện tượng nhấp nháy khi cập nhật màn hình.

Hạn chế chính của U8g2 là chỉ hỗ trợ màn hình đơn sắc, không phù hợp cho các ứng dụng yêu cầu hiển thị màu. Thư viện cũng thiếu các widget GUI và hệ thống quản lý sự kiện như các thư viện khác.

U8g2 là lựa chọn tuyệt vời cho các ứng dụng sử dụng màn hình OLED đơn sắc, đặc biệt khi tài nguyên hệ thống hạn chế hoặc khi tiêu thụ năng lượng là ưu tiên hàng đầu.

2.7.1.4 LVGL

LVGL (Light and Versatile Graphics Library) là một thư viện GUI toàn diện, cung cấp không chỉ các hàm vẽ cơ bản mà còn bao gồm hệ thống widget phong phú, quản lý sự kiện, và nhiều tính năng cao cấp khác.

Như đã phân tích chi tiết trong phần trước, LVGL có nhiều ưu điểm nổi bật so với các thư viện khác. Thư viện cung cấp bộ widget phong phú (nút bấm, thanh trượt, bảng, biểu đồ, ...), hệ thống styles và themes linh hoạt, và hỗ trợ animation. LVGL được thiết kế để hoạt động hiệu quả trên các hệ thống nhúng với tài nguyên hạn chế, yêu cầu chỉ 64 KB flash và 16 KB RAM cho các tính năng cơ bản.

LVGL cũng có khả năng mở rộng cao, cho phép nhà phát triển tạo ra các widget tùy chỉnh hoặc mở rộng các widget có sẵn. Thư viện hỗ trợ nhiều loại thiết bị đầu vào (cảm ứng, nút bấm, encoder) và có thể hoạt động với nhiều loại màn hình khác nhau thông qua hệ thống driver linh hoạt.

Tuy nhiên, LVGL có đường cong học tập dốc hơn so với các thư viện đơn giản như Adafruit GFX, và cấu hình ban đầu có thể phức tạp. Thư viện cũng yêu cầu nhiều tài

nguyên hơn so với các giải pháp đơn giản hơn, mặc dù vẫn được tối ưu hóa cho hệ thống nhúng.

LVGL là lựa chọn lý tưởng cho các ứng dụng yêu cầu GUI phức tạp, chuyên nghiệp, với nhiều widget tương tác và hiệu ứng đồ họa.

2.7.2 So sánh và lựa chọn thư viện phù hợp

Khi lựa chọn thư viện đồ họa cho dự án ESP32, cần cân nhắc nhiều yếu tố khác nhau để đưa ra quyết định phù hợp nhất.

2.7.2.1 Hiệu suất và tài nguyên

Về hiệu suất render, TFT_eSPI thường dẫn đầu nhờ tối ưu hóa assembly và sử dụng DMA, tiếp theo là LVGL với engine render được tối ưu hóa. Adafruit GFX và U8g2 có hiệu suất thấp hơn nhưng vẫn đủ cho nhiều ứng dụng.

Về tiêu thụ bộ nhớ, U8g2 là nhẹ nhất, tiếp theo là TFT_eSPI và Adafruit GFX. LVGL yêu cầu nhiều bộ nhớ nhất, đặc biệt khi sử dụng nhiều widget và animation, nhưng vẫn được tối ưu hóa cho hệ thống nhúng.

2.7.2.2 Tính năng và khả năng mở rộng

LVGL dẫn đầu về tính năng với bộ widget phong phú, hệ thống styles, và animation. TFT_eSPI cung cấp các tính năng đồ họa cơ bản mạnh mẽ nhưng thiếu widgets. Adafruit GFX và U8g2 tập trung vào các chức năng vẽ cơ bản.

Về khả năng mở rộng, LVGL có kiến trúc module hóa cho phép mở rộng dễ dàng. TFT_eSPI cũng khá linh hoạt nhưng đòi hỏi nhiều công sức hơn để mở rộng. Adafruit GFX và U8g2 có khả năng mở rộng hạn chế hơn.

2.7.2.3 Độ phức tạp và đường cong học tập

Adafruit GFX có đường cong học tập thoải nhất, phù hợp cho người mới bắt đầu. U8g2 và TFT_eSPI có độ phức tạp trung bình, trong khi LVGL có đường cong học tập dốc nhất do cung cấp nhiều tính năng và khái niệm phức tạp hơn.

2.7.2.4 Hỗ trợ cộng đồng và tài liệu

Tất cả bốn thư viện đều có cộng đồng người dùng lớn và tài liệu tốt. Adafruit GFX có lợi thế về hướng dẫn và ví dụ từ Adafruit. LVGL có tài liệu toàn diện nhất với hướng dẫn, ví dụ, và tài liệu tham khảo API đầy đủ. TFT_eSPI và U8g2 cũng có tài liệu tốt và cộng đồng hỗ trợ tích cực.

2.7.2.5 Trường hợp sử dụng phù hợp

- **TFT_eSPI**: Phù hợp nhất cho các ứng dụng yêu cầu hiệu suất cao, hiển thị đồ họa cơ bản, hoặc khi tài nguyên hệ thống hạn chế nhưng vẫn cần hiển thị màu.
- **Adafruit GFX**: Lý tưởng cho người mới bắt đầu, các dự án đơn giản, hoặc khi cần tương thích với nhiều loại màn hình khác nhau.
- **U8g2**: Tốt nhất cho các ứng dụng sử dụng màn hình OLED đơn sắc, đặc biệt khi tiêu thụ năng lượng và bộ nhớ là ưu tiên.
- **LVGL**: Phù hợp nhất cho các ứng dụng yêu cầu GUI phức tạp, chuyên nghiệp, với nhiều widget tương tác và hiệu ứng đồ họa.

2.7.3 Tại sao chọn LVGL cho dự án phát triển thư viện giao diện

Sau khi phân tích các giải pháp GUI hiện có, LVGL nổi bật như một lựa chọn tối ưu cho việc phát triển thư viện giao diện cho ESP32 vì nhiều lý do.

- **Tính năng phong phú**: LVGL cung cấp một bộ widget phong phú và hệ thống styles linh hoạt, cho phép tạo ra giao diện người dùng hấp dẫn và chuyên nghiệp mà không cần phát triển từ đầu. Điều này giúp tiết kiệm thời gian phát triển đáng kể và mang lại kết quả chất lượng cao hơn.
- **Hiệu suất tối ưu**: LVGL được tối ưu hóa cho các hệ thống nhúng như ESP32, cân bằng giữa tính năng phong phú và hiệu suất. Thư viện có thể hoạt động hiệu quả với tài nguyên hạn chế của ESP32, đồng thời vẫn cung cấp trải nghiệm người dùng mượt mà.
- **Khả năng mở rộng**: Kiến trúc module hóa của LVGL cho phép mở rộng và tùy chỉnh dễ dàng, lý tưởng cho việc phát triển thư viện giao diện tùy chỉnh. Nhà phát triển có thể xây dựng các widget mới hoặc mở rộng các widget có sẵn để đáp ứng nhu cầu cụ thể của dự án.
- **Hỗ trợ cộng đồng và tài liệu**: LVGL có cộng đồng người dùng lớn, tài liệu toàn diện, và được cập nhật thường xuyên. Điều này đảm bảo rằng thư viện giao diện được phát triển sẽ có nền tảng vững chắc và hỗ trợ lâu dài.

Với những ưu điểm trên, LVGL là nền tảng lý tưởng để phát triển thư viện giao diện cho ESP32, cung cấp sự cân bằng tối ưu giữa tính năng, hiệu suất, và khả năng mở rộng.

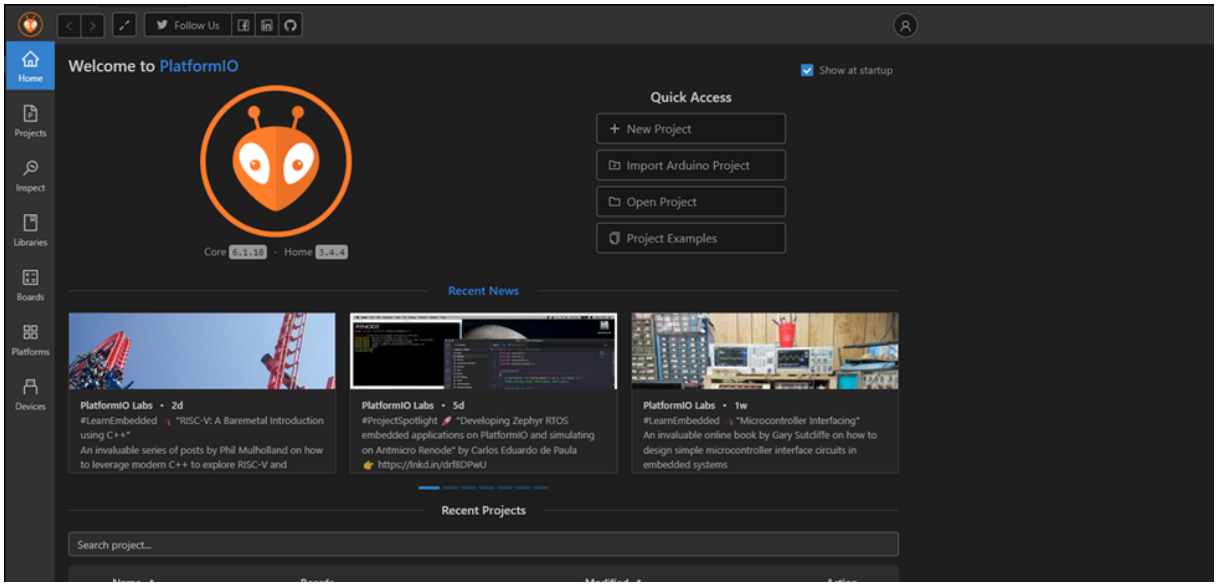
2.8 Công cụ hỗ trợ phát triển

2.8.1 PlatformIO

2.8.1.1 Giới thiệu tổng quan

PlatformIO là một môi trường phát triển tích hợp (IDE) hiện đại và mở rộng cho các hệ thống nhúng, đặc biệt phù hợp với các vi điều khiển như ESP32. Được phát triển như một plugin cho Visual Studio Code (VSCode), PlatformIO cho phép quản lý dự án, thư viện, board, và build system một cách dễ dàng, hỗ trợ cả framework ESP-IDF lẫn Arduino. Nó là công cụ lý tưởng cho những ai muốn phát triển giao diện với LVGL vì có thể:

- Quản lý thư viện trực tiếp từ CLI hoặc GUI.
- Tự động cấu hình các flag build, include path, và linker script.
- Hỗ trợ debug, monitor serial, và upload chỉ với một cú click.



Hình 2.9: PlatformIO trong workflow phát triển LVGL

2.8.1.2 Tính năng nổi bật

PlatformIO cung cấp nhiều tính năng nổi bật giúp tăng tốc quá trình phát triển và quản lý dự án. Dưới đây là một số tính năng chính:

Tính năng	Mô tả
Đa framework	Hỗ trợ ESP-IDF, Arduino, Zephyr, mbed, RT-Thread...
Hỗ trợ đa board	PlatformIO hỗ trợ sẵn hơn 900 board, trong đó có toàn bộ dòng ESP32 (ESP32-WROOM, S2, S3...)
Tự động build & flash	Không cần viết Makefile hay config phức tạp – chỉ cần `platformio run` và `upload`
Monitor serial tích hợp	Dễ dàng in log từ ESP32 để debug
Quản lý thư viện	Cài LVGL, touch driver, font renderer, JPEG decoder... từ CLI hoặc GUI
Tích hợp VSCode	Có intellisense, gợi ý mã, refactor code, phân tích lỗi cú pháp realtime

Bảng 2.10: Các tính năng nổi bật của PlatformIO

2.8.1.3 Lý do nên sử dụng PlatformIO cho phát triển giao diện LVGL

- Tăng tốc khởi tạo dự án: Có thể tạo dự án ESP32 chỉ với 1 dòng lệnh `pio project init`, giúp nhanh chóng bắt đầu coding mà không cần cấu hình thủ công.
- Dễ dàng tích hợp LVGL: PlatformIO cho phép thêm thư viện LVGL từ PlatformIO Registry, đảm bảo luôn có phiên bản mới nhất, dễ update.
- Tự động xử lý dependency: Không cần tải thư viện thủ công hoặc sửa include path trong `CMakeLists.txt` – PlatformIO tự động thực hiện việc đó.
- Hỗ trợ nhiều profile build: Dễ dàng build ở các mode như debug, release, hoặc custom.
- Cộng đồng đông đảo, tài liệu phong phú: Nhiều template và ví dụ ESP32 + LVGL sẵn có.

2.8.1.4 Hạn chế và cách khắc phục

Hạn chế	Cách giải quyết
Tốc độ build chậm hơn ESP-IDF gốc	Bật chế độ build incremental, tăng RAM cache, và tắt phần mềm diệt virus quét thư mục `.pio`
Debug bị hạn chế nếu không dùng board chính thức	Kết hợp OpenOCD để debug hoặc chuyển sang ESP-IDF nếu cần debug chuyên sâu
Không tối ưu mạnh về kích thước chương trình như ESP-IDF	Sử dụng `build_flags` để tinh chỉnh cờ biên dịch hoặc tối ưu file linker script

Bảng 2.11: Một số hạn chế của PlatformIO và cách khắc phục

2.8.1.5 Vai trò PlatformIO trong workflow tổng thể

Giai đoạn	Vai trò PlatformIO
Khởi tạo dự án	Tạo nhanh template ESP32, tự động cấu trúc thư mục
Tích hợp thư viện	Quản lý dễ dàng thư viện GUI như LVGL, driver touch
Build & Flash	Tự động build, upload, và theo dõi log serial
Phát triển UI	Soạn mã GUI trong VSCode, tận dụng intellisense & lỗi runtime
Tinh chỉnh hiệu suất	Sử dụng monitor heap/log, custom flag để tối ưu kích thước và tốc độ

2.8.2 SquareLine Studio

2.8.2.1 Giới thiệu tổng quan

SquareLine Studio là một công cụ phát triển giao diện người dùng (UI) dành cho các nền tảng nhúng như ESP32, ESP8266, và STM32. Được xây dựng với mục tiêu đơn giản hóa việc thiết kế giao diện, SquareLine Studio cung cấp một môi trường trực quan với tính năng kéo và thả (drag-and-drop), giúp người dùng dễ dàng tạo và tùy chỉnh giao diện mà không cần phải viết mã thủ công cho từng widget hay thành phần giao diện.

SquareLine Studio đặc biệt hữu ích khi kết hợp với LVGL, giúp tối ưu hóa quá trình phát triển và cải thiện hiệu quả công việc, đặc biệt khi phát triển giao diện cho các hệ thống nhúng như ESP32, nơi tài nguyên phần cứng bị hạn chế.

Hình 2.10: Squaline Studio giao diện thiết kế

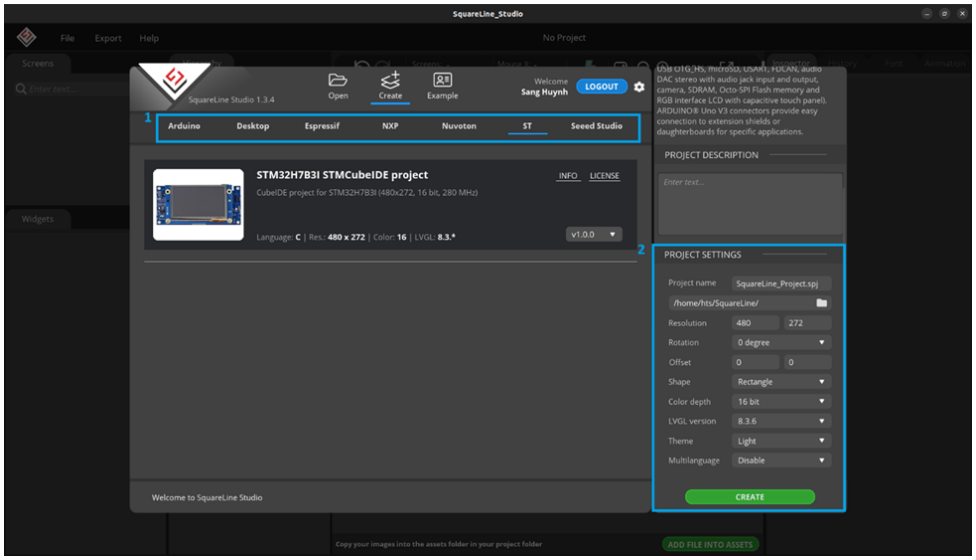
2.8.2.2 Các tính năng chính của SquareLine Studio

Tính năng	Mô tả
Kéo thả widget	Cho phép người dùng dễ dàng kéo và thả các widget (như buttons, labels, sliders...) vào giao diện mà không cần phải viết mã.
Chỉnh sửa style theo theme	Tùy chỉnh giao diện của các widget thông qua các theme và style có sẵn.
Tạo mã nguồn C/C++ đầy đủ	Tự động sinh mã nguồn C/C++ hoàn chỉnh từ thiết kế giao diện, dễ tích hợp vào ESP32 hoặc STM32.
Hỗ trợ cấu hình phần cứng	Cung cấp các lựa chọn cấu hình phần cứng (LCD, touch, SPI...).
Hỗ trợ LVGL	Làm việc trực tiếp với thư viện LVGL.
Preview trực tiếp	Xem trước giao diện trong quá trình thiết kế.

Bảng 2.12: Các tính năng chính của SquareLine Studio

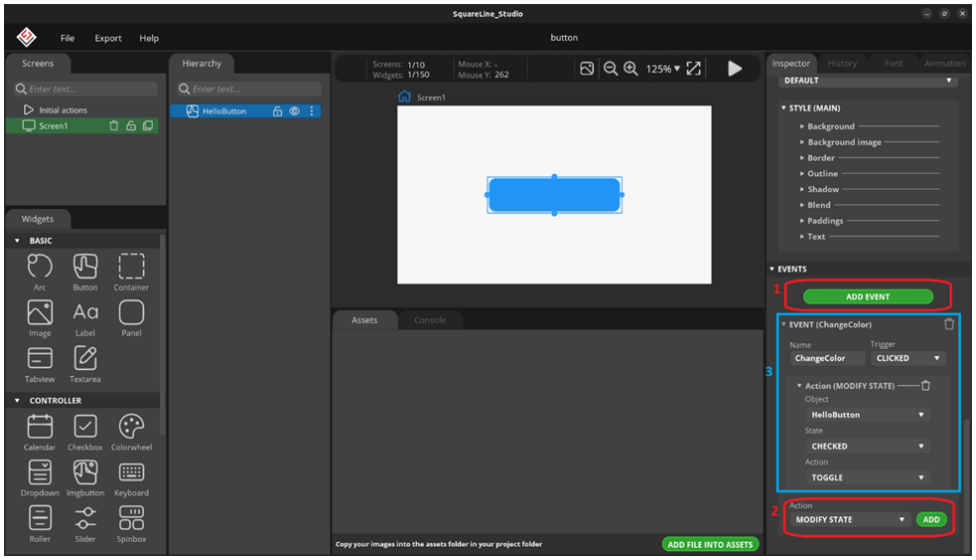
2.8.2.3 Quy trình phát triển giao diện với SquareLine Studio

1. **Tạo dự án mới:** Mở SquareLine Studio và chọn nền tảng phần cứng và màn hình sử dụng.



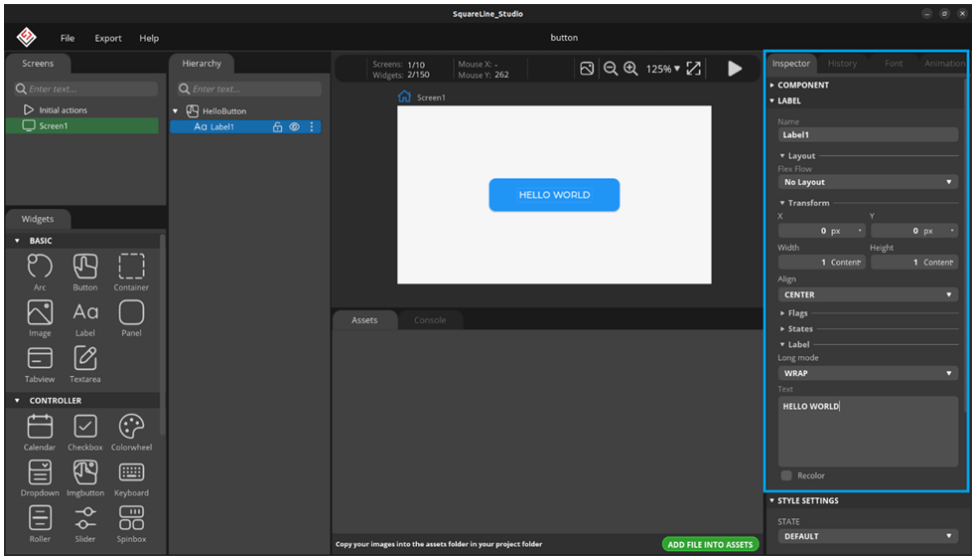
Hình 2.11: Tạo dự án mới trong SquareLine Studio

2. **Thiết kế giao diện:** Kéo thả các widget và tùy chỉnh thuộc tính giao diện.



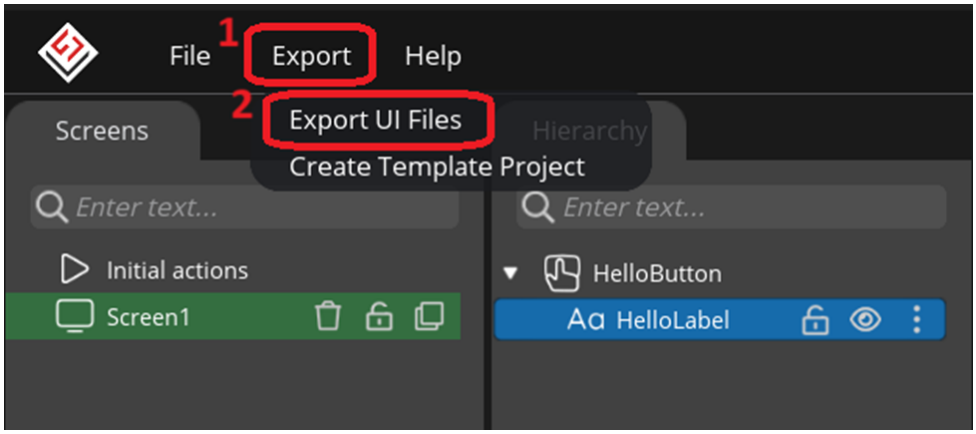
Hình 2.12: Thiết kế giao diện trong SquareLine Studio

3. **Tùy chỉnh style:** Áp dụng các theme có sẵn hoặc tùy chỉnh theo yêu cầu.



Hình 2.13: Tùy chỉnh style trong SquareLine Studio

4. **Sinh mã nguồn:** Tự động sinh mã C/C++ cho toàn bộ giao diện và xử lý sự kiện.



Hình 2.14: Sinh mã nguồn trong SquareLine Studio

5. **Tích hợp với ESP32:** Chèn mã vào dự án ESP32, cấu hình qua `menuconfig`, biên dịch bằng `idf.py`.

6. **Test và tối ưu hóa:** Kiểm thử trực tiếp hoặc trên phần cứng, điều chỉnh hiệu ứng và hiển thị.

2.8.2.4 Ưu điểm khi sử dụng SquareLine Studio

Ưu điểm	**Mô tả**
Tiết kiệm thời gian	Thiết kế kéo thả giúp tiết kiệm thời gian và công sức.
Chắc chắn và chính xác	Mã nguồn sinh ra tối ưu và hoàn chỉnh.
Dễ dàng tái sử dụng	Thiết kế và mã có thể dùng lại cho các dự án khác.
Tích hợp tốt với LVGL	Hỗ trợ đầy đủ tính năng đồ họa của LVGL.
Hỗ trợ đa nền tảng	Chạy được trên Windows, macOS, và Linux.

Bảng 2.13: Ưu điểm của SquareLine Studio

2.8.2.5 Hạn chế và biện pháp khắc phục

Hạn chế	**Giải pháp**
Không hỗ trợ toàn bộ tính năng LVGL	Có thể cần chỉnh sửa mã thủ công cho tính năng nâng cao.
Giới hạn trong thiết kế phức tạp	Phải can thiệp mã để xử lý logic phức tạp hoặc tuỳ chỉnh sâu.
Khả năng tùy biến giao diện hạn chế	Viết mã tùy chỉnh cho các hiệu ứng và hành vi đặc biệt.

Bảng 2.14: Hạn chế và cách khắc phục khi dùng SquareLine Studio

2.8.2.6 Ứng dụng thực tế với SquareLine Studio

SquareLine Studio hữu ích trong các dự án giao diện người dùng cho ESP32 với LVGL. Các ứng dụng phổ biến bao gồm:

- Thiết kế giao diện cho thiết bị điều khiển từ xa: điều khiển, giám sát trạng thái, hiển thị cảm biến.
- Giao diện người dùng cho hệ thống IoT: đồng hồ thông minh, nhà thông minh.
- Giao diện cảm ứng: hỗ trợ tương tác như chạm, vuốt, thay đổi cảnh.

Tài liệu tham khảo

[1] Karl Johan Astrom, Richard M. Murray, *Feedback Systems: An Introduction for Scientists and Engineers*, Princeton University Press, 2012.

[2] A Joukhadar, I Hasan, A Alsabbagh, M Alkouzbary, *Integral Lqr-Based 6dof Autonomous Quadrocopter Balancing System Control*, International Journal of Advanced Research in Artificial Intelligence, Vol. 4, No.5, 2015.

[3] Nur Hayati Sahrir, Mohd Ariffanan Mohd Basri, *Modelling and Manual Tuning PID Controlof Quadcopter*, Control, Instrumentation and Mechatronics: Theory and Practice (pp.346-357).

[4] Nguyễn Đình Huy, *Giáo trình giải tích 1*, Đại học Quốc gia TP. Hồ Chí Minh, 2020.

[5] Faisal Iqbal, Hussamud Din, Byeungleul Lee, *Single Drive Multi-Axis Gyroscope with High Dynamic Range, High Linearity and Wide Bandwidth*, Micromachines 2019, 10(6), 410.

[6] Heja Cengiz, *Quadcopter Modeling and Linear Quadratic Regulator Design Using Simulink*, Uppsala Universitet, 2024.

[7] Ha Quang Thinh Ngo, Thanh Phuong Nguyen, Hung Nguyen, "A COMPLETE COMPARISON TO DESIGN COMPLEMENTARY FILTER AND KALMAN FILTER FOR AERIAL VEHICLE", *International Journal of Mechanical Engineering and Technology*, 2018.

[8] Rio Ikhsan Alfian, Alfian Ma'arif, Sunardi Sunardi, "Noise Reduction in the Accelerometer and Gyroscope Sensor with the Kalman Filter Algorithm", *Journal of Robotics and Control (JRC)*, 2021.

[9] Luis E. Romero, David F. Pozo, Jorge A. Rosales, "Quadcopter Stabilization by Using PID Controllers", *ACADEMIA*, 2014.

[10] Yan Michalevsky, Dan Boneh, "Gyrophone: Recognizing Speech from Gyroscope Signals", *Proceedings of the 23rd USENIX Security Symposium*, 2014.

[11] Dennis Freeman, Kevin Chen "Linear quadratic regulator (LQR) control" Source: https://introcontrol.mit.edu/_static/spring23/lectures/lec08a-handout.pdf.

[12] "MPU-6000 and MPU-6050 Product Specification Revision 3.4", 2013. Source: www.invensense.com.

[13] "Hướng dẫn sử dụng cảm biến gia tốc MPU6050 với Arduino", 2023. Source: https://arduinokit.vn/huong-dan-su-dung-cam-bien-gia-toc-mpu6050-voi-arduino/.

[14] "tkinter — Python interface to Tcl/Tk". Source: https://docs.python.org/3/library/tkinter.html.

[15] "socket — Low-level networking interface". Source: https://docs.python.org/3/library/socket.html.

[16] "Arduino Documentation". Source: https://docs.arduino.cc/.

[17] "Distance Measurement with an Ultrasonic Sensor HY-SRF05", 2017. Source: https://projecthub.arduino.cc/Nicholas_N/distance-measurement-with-an-ultrasonic-sensor-hy-srf05-bf2923.

[18] "ESC Calibration". Source: https://ardupilot.org/plane/docs/common-esc-calibration.html.

[19] Wikipedia, "Quadcopter". Source: https://en.wikipedia.org/wiki/Quadcopter.

[20] STMicroelectronics, "Everything about STMicroelectronics' 3-axis digital MEMS gyroscopes". Source: https://www.elecrow.com/download/TA0343.pdf.

[21] Teppo Luukkonen, "Modelling and control of quadcopter". Source: https://sal.aalto.fi/publications/pdf-files/eluu11_public.pdf.

[22] "Accelerometer and Gyroscopes Sensors: Operation, Sensing, and Applications". Source: https://www.analog.com/en/resources/technical-articles/accelerometer-and-gyroscopes-sensors-operation-sensing-and-applications.html.

[23] "Single Drive Multi-Axis Gyroscope with High Dynamic Range, High Linearity and Wide Bandwidth". Source: https://www.mdpi.com/2072-666X/10/6/410.

[24] Michel van Biezen, "SPECIAL TOPICS 1 - THE KALMAN FILTER". Source: https://www.youtube.com/watch?v=CaCcOwJPytQ&list=PLX2gX-ftPVXU3oUFNATxGXY90AULiqnWT&index=1.

[25] Kevin Jordan, "Self-Stabilizing Quadcopter UAV Using PID Control: Full Control Systems Project Presentation". Source: https://www.youtube.com/watch?v=clyusOrMqbU.

[26] Curio Res, "How Gyroscope Sensor Works? | 3D Animated". Source: https://www.youtube.com/watch?v=HJ-C4Incgpw.

[27] Blue Butterfly, "How to design and implement a digital low-pass filter on an Arduino". Source: https://www.youtube.com/watch?v=REVp33SwwHE.

[28] Dr. KC Craig, "Quadrotor Equations of Motion and Control KCC Final 4 2023 Video". Source: https://www.youtube.com/watch?v=REVp33SwwHE.

[29] thegioiic, "Tìm hiểu về chuẩn giao tiếp I2C". Source: https://www.thegio iic.com/tin-tuc/tim-hieu-ve-chuan-giao-tiep-i2c.

[30] Timothy Hirzel, "Basics of PWM (Pulse Width Modulation)". Source: https://docs.arduino.cc/learn/microcontrollers/analog-output/.

[31] Carbon Aeronautics, "Carbon Aeronautics Quadcopter Manual". Source: https://github .com/CarbonAeronautics/Manual-Quadcopter-Drone/blob /main/-Carbon_Aeronautics_Quadcopter_Manual.pdf.

[32] LinhKienRC, "Combo điều khiển TX – RX Flysky FS i6". Source: https://bandochoi.net/combo-dieu-khien-tx-rx-flysky-fs-i6.html.

[33] nshopvn, "Module thu phát Wifi ESP8266 NodeMCU Lua CP2102". Source: https://nshopvn.com/product/module-thu-phat-wifi-esp8266-nodemcu-lua-cp2102/.

[34] Sam Market, "Teensy 4.0 (Headers)". Source: https://market.samm.com/teensy-4-0-headers-en.

[35] hshop, "Cảm biến GY-521 6DOF IMU MPU6050". Source: https://hshop.vn /cam-bien-6-dof-bac-tu-do-gy-521-mpu6050.

[36] nshopvn, "Cảm biến áp suất IIC I2C và nhiệt độ của BMP280 3.3 V". Source: https://nshopvn.com/product/cam-bien-ap-suat-iic-i2c-va-nhiet-do-cua-bmp280-3-3-v/.

[37] Nguyễn Hiền, "Mạch điều khiển tốc độ động cơ không chổi than ESC 30A 4V-16V (Pin 2S-4S) BEC 5V". Source: https://dientunguyenhien.vn/show/3252.

[38] nshopvn, "Động cơ không chổi than A2212". Source: https://nshopvn.com /product/dong-co-khong-choi-than-a2212/.

[39] FPT Jetking, "ESP8266 là gì? Tìm hiểu về module Wi-Fi phổ biến cho IoT". Source: https://jetking.fpt.edu.vn/esp8266/.

[40] pjrc, "Using the Hardware Serial Ports". Source: https://www.pjrc.com/teensy/td _uart.html

[41] x-engineer, "On-off control system". Source: https://x-engineer.org/on-off-control-system

[42] webOS TV Developer, "Sensor Data for Motion Sensor". Source: https:// webostv.developer.lge.com/develop/guides/motion-sensor-sensor-data.

[43] Yahya Tawil, "Towards understanding IMU: Basics of Accelerometer and Gyroscope Sensors and How to Compute Pitch, Roll and Yaw Angles". Source: https://atadiat.com/en/e-towards-understanding-imu-basics-of-accelerometer-and-gyroscope-sensors/.

[44] Nasser M. Abbasi, "Dynamics equations, kinematics, velocity and acceleration diagrams". Source: https://www.12000.org/my_notes/dynamics_cheat_sheet/ reportchapter2.htm.