TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN &

TRUYỀN THÔNG VIỆT-HÀN

Khoa Kĩ Thuật Máy Tính & Điện Tử

Chuyên đề 4(CE) Lập trình Ô Tô

Sinh viên thực hiện: **Võ Văn Tuấn**

Trần Anh Tuấn

Hoàng Minh Nghĩa

Trương Đắc Trường

Nguyễn Duy Đại Thạch

Lớp: 21CE

Giảng viên hướng dẫn: TS.Nguyễn Vũ Anh Quang

KS. Trần Viết An

Đà Nẵng, tháng 5 năm 2025

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN &

TRUYỀN THÔNG VIỆT-HÀN

Khoa Kĩ Thuật Máy Tính & Điện Tử

Chuyên đề 4(CE) Lập trình Ô Tô

Thiết kế và triển khai hệ thống giám sát tốc độ, vòng tua máy và trạng thái trên xe ô tô

Sinh viên thực hiện: **Võ Văn Tuấn**

Trần Anh Tuấn

Nguyễn Duy Đại Thạch

Hoàng Minh Nghĩa

Trương Đắc Trường

Lớp: 21CE

Giảng viên hướng dẫn: TS.Nguyễn Vũ Anh Quang

KS. Trần Viết An

Đà Nẵng, tháng 4 năm 2025

LỜI CẢM ƠN

Nhờ sự hướng dẫn tận tình của Thầy TS. Nguyễn Vũ Anh Quang và Kỹ sư Trần Viết An, em đã hiểu rõ hơn về kiến thức chuyên ngành trong lĩnh vực lập trình ô tô, cách áp dụng lý thuyết vào thực tiễn, và đặc biệt là cách tiếp cận các vấn đề kỹ thuật phức tạp một cách khoa học. Những chỉ dẫn quý báu của Thầy và Kỹ sư không chỉ giúp em hoàn thành tốt môn học Chuyên đề 4 – Lập trình ô tô, mà còn là hành trang quan trọng trên con đường học tập và phát triển sự nghiệp của em trong tương lai.

Trước hết, em xin gửi tới các thầy cô trường Đại học Công nghệ Thông tin và Truyền thông Việt – Hàn lời chào trân trọng, lời chúc sức khỏe và lời cảm ơn sâu sắc. Với sự quan tâm, giảng dạy và chỉ bảo tận tình, các giảng viên đã trang bị cho chúng em những kỹ năng và kiến thức cần thiết để hoàn thành tốt nội dung học tập của học phần này.

Đặc biệt, em xin gửi lời cảm ơn chân thành nhất tới Thầy TS. Nguyễn Vũ Anh Quang và Kỹ sư Trần Viết An đã luôn quan tâm, hỗ trợ, và tạo điều kiện tốt nhất để em có thể tiếp cận kiến thức một cách hiệu quả và thực tế.

Vì điều kiện thời gian cũng như kinh nghiệm còn hạn chế, bài báo cáo này chắc chắn không thể tránh khỏi những thiếu sót. Em rất mong nhận được sự chỉ bảo, góp ý của các thầy cô để có thể bổ sung, hoàn thiện hơn trong tương lai, phục vụ tốt hơn cho công việc thực tiễn sau này.

Em xin chân thành cảm ơn!

Đà Nẵng, ngày tháng 4 năm 2025

Sinh viên

NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Chữ ký của GVHD

MỤC LỤC

CÁC CỤM TỪ VIẾT TẮT

Viết tắt	Ý nghĩa đầy đủ
ESP32	Espressif Systems 32-bit
MCU	Microcontroller Unit
UART	Universal Asynchronous Receiver Transmitter
EEPROM	Electrically Erasable Programmable ROM
ODO	Odometer
TRIP	Trip Distance
RPM	Revolutions Per Minute
GPIO	General Purpose Input/Output
ADC	Analog-to-Digital Converter
QML	Qt Modeling Language
GUI	Graphical User Interface
QTimer	Qt Timer
QSerialPort	Qt Serial Port
QSettings	Qt Settings
Q_PROPERTY	Qt Property Macro
MS	Milliseconds

DANH MỤC HÌNH VẼ

Chương 1: Giới thiệu

1.1 Bối cảnh và đặt vấn đề

Trong bối cảnh công nghiệp ô tô hướng tới "Xe thông minh" và "Xe kết nối", việc giám sát thời gian thực các thông số vận hành như tốc độ, vòng tua máy và trạng thái các thiết bị điện trên xe không còn là một lựa chọn mà đã trở thành yêu cầu bắt buộc. Bảng đồng hồ cơ truyền thống với kim đo analog tuy có ưu điểm bền bỉ, đơn giản nhưng lại thiếu tính linh hoạt và công suất hiển thị hạn chế. Việc phân tích xu hướng vận hành dài hạn, lưu trữ dữ liệu hoặc truyền tín hiệu cảnh báo sớm cho bộ phận bảo trì hầu như không khả thi.

Trong khi đó, các giải pháp giám sát công nghiệp có sẵn thường đắt đỏ, phụ thuộc vào giao thức CAN-bus chuẩn OBD-II hoặc phải tích hợp sâu vào ECU của xe, gây khó khăn khi triển khai cho các dòng xe đời cũ hoặc xe tự chế. Với sự phát triển mạnh mẽ của các nền tảng vi điều khiển như ESP32, khả năng đo đếm xung cao cấp, tích hợp Wi-Fi/Bluetooth cùng với các thư viện giao diện đồ họa Qt trên Linux, chúng có thể tạo thành một hệ thống "vừa đủ" về mặt chức năng, chi phí hợp lý và dễ tùy biến cho nhu cầu nghiên cứu, giảng dạy hoặc ứng dụng thực tế.

Hệ thống giám sát được xây dựng gồm hai thành phần chính: phần cứng thu thập và xử lý tín hiệu xung từ cảm biến, và phần mềm giao diện trên máy tính dùng Qt để hiển thị, lưu trữ và xuất báo cáo. Mục tiêu của đồ án là phát triển một giải pháp tổng thể, từ khâu thiết kế mạch, bố trí PCB đến lập trình firmware trên ESP32 và triển khai ứng dụng Qt trên môi trường Linux, đảm bảo đáp ứng được các tiêu chí về độ chính xác, thời gian thực và khả năng mở rộng.

1.2 Mục tiêu nghiên cứu

Xây dựng mạch phần cứng sử dụng ESP32–WROOM32 làm lõi, kết hợp với cảm biến Hall-effect để thu nhận xung quay bánh xe, cảm biến từ hoặc tín hiệu xung từ bộ đánh lửa cho vòng tua máy.

Triển khai mạch lọc nhiễu đơn giản với tụ điện và trở kéo, có thể cách ly quang-điện để bảo vệ vi điều khiển trong môi trường nhiễu điện áp cao trên xe.

Thiết kế PCB hai lớp, quy chuẩn hóa kích thước và chân kết nối (UART, nguồn, I/O), đảm bảo tính cơ động khi lắp ráp trên xe hoặc mô hình bench-top.

Phát triển firmware ESP32 sử dụng hardware timer để đếm xung với độ phân giải cao, tính toán tốc độ (km/h) và vòng tua (rpm) theo thời gian thực, đóng gói dữ liệu và truyền qua UART với định dạng dễ parse.

Xây dựng ứng dụng GUI bằng Qt 5.15 trên Ubuntu 20.04 sử dụng QSerialPort để đọc dữ liệu từ ESP32, thực hiện phân tích cú pháp, thể hiện trực quan qua đồng hồ analog/digital, biểu đồ thời gian thực và bảng trạng thái thiết bị.

Tích hợp cơ chế ghi nhật ký theo ngày, lưu file CSV/PDF để phục vụ công tác hậu kiểm và báo cáo.

Thử nghiệm hệ thống với tín hiệu xung giả lập từ function generator, so sánh kết quả hiển thị với bộ đo tham chiếu để đánh giá sai số, thời gian đáp ứng và độ ổn định.

1.3 Phạm vi và ý nghĩa đề tài

Phạm vi nghiên cứu tập trung vào hai mảng chính là đo xung tốc độ/vòng tua và hiển thị trạng thái digital (on/off). Đề tài chưa đề cập đến:

Giao tiếp CAN-bus chuẩn OBD-II hay tích hợp trực tiếp vào ECU của xe.

Chức năng kết nối Internet, di động hoặc lưu trữ đám mây.

Các cảm biến analog phức tạp như áp suất dầu, nhiệt độ động cơ (trong giai đoạn tiếp theo có thể mở rộng).

Từ góc độ ý nghĩa, đồ án:

Cung cấp giải pháp mẫu cho các dự án nghiên cứu, giảng dạy, workshop về nhúng và giao diện đồ họa.

Giúp sinh viên nắm vững quy trình thiết kế phần cứng, layout PCB, lập trình firmware real-time và phát triển ứng dụng desktop.

Tạo nền tảng mở để dễ dàng thêm cảm biến mới, chuyển nền tảng Qt sang Windows/macOS hoặc xây dựng phiên bản di động.

1.4 Cơ sở lý thuyết và tổng quan công nghệ

Hệ thống giám sát phương tiện:

Hệ thống giám sát phương tiện (Vehicle Monitoring System) là một giải pháp công nghệ dùng để thu thập, phân tích và hiển thị các thông số quan trọng như:

Tốc độ (km/h),

Vòng tua động cơ (RPM),

Quãng đường đã đi (ODO và Trip),

Trạng thái hộp số (tiến/lùi).

Các hệ thống này giúp tài xế kiểm soát phương tiện hiệu quả hơn, đồng thời là cơ sở để phân tích hành vi lái xe, bảo trì xe hoặc tích hợp vào hệ thống hỗ trợ lái tự động.

Tốc độ ,vòng tua máy,ODO và TRIP

- **Tốc độ (Speed)** là đại lượng đo tốc độ di chuyển của xe tính theo km/h. Trong hệ thống này, giá trị tốc độ được mô phỏng bằng điện áp đọc từ chiết áp (analog), sau đó quy đổi về tốc độ thực.
- **Vòng tua máy (RPM - Round Per Minute)** biểu thị số vòng quay mỗi phút của trục khuỷu động cơ. Mặc dù hệ thống không dùng cảm biến thực tế, RPM cũng được mô phỏng qua chiết áp để phục vụ hiển thị.
- **ODO (Odometer)** là tổng số km xe đã đi kể từ khi bắt đầu hoạt động. Đây là giá trị quan trọng, cần lưu lại vĩnh viễn, kể cả khi tắt nguồn.
- **Trip** là quãng đường đi được trong một hành trình cụ thể. Người dùng có thể chủ động reset Trip để theo dõi hành trình mới.

Trạng thái hộp số

Hệ thống sử dụng một nút gạt vật lý để chuyển đổi giữa hai trạng thái: **Drive (D)** và **Reverse (R)**. Việc xử lý chuyển đổi này cần tránh nhiễu, rung phím và đảm bảo đồng bộ với giao diện người dùng.

Tổng quan công nghệ sử dụng

ESP32–WROOM32 là module vi điều khiển mạnh mẽ, tích hợp CPU dual-core, Wi-Fi/Bluetooth, và các peripheral như timer, GPIO, UART, I²C, SPI. Đặc biệt, hardware timer cho phép đếm xung ngoại vi với độ chính xác đến micro-giây, rất phù hợp cho nhiệm vụ đo tốc độ xe và vòng tua máy.

Trong đề tài này, ESP32 dùng để:

- Đọc chiết áp và nút nhấn.
- Tính toán tốc độ, RPM, ODO và Trip.
- Lưu dữ liệu vào EEPROM.
- Giao tiếp với phần mềm máy tính qua Serial.

Giao tiếp Serial và QtSerialPort

- Dữ liệu từ ESP32 được gửi liên tục qua UART (Serial) đến máy tính.
- Phía máy tính, QtSerialPort được dùng để đọc dữ liệu, phân tích chuỗi và hiển thị lên GUI.
- Giao thức truyền đơn giản: speed,rpm,odo,trip,gear\n.

Giao diện người dùng bằng Qt/QML

- **Qt** là framework phát triển ứng dụng đa nền tảng.
- **QML** (Qt Modeling Language) hỗ trợ thiết kế giao diện hiện đại, mượt mà, dễ kết hợp logic với C++.
- Các thành phần được hiển thị gồm:
    - Đồng hồ tốc độ và RPM.
    - Hiển thị số liệu ODO, Trip, tốc độ, số.
    - Kết nối với backend (SerialHandler C++) để nhận dữ liệu.

Phần mềm Qt là một framework phát triển ứng dụng đa nền tảng, hỗ trợ widget truyền thống (Qt Widgets) và giao diện khai báo kiểu QML. Lớp QSerialPort trong Qt cung cấp API bất đồng bộ (asynchronous) để đọc/ghi dữ liệu qua cổng serial, có callback khi nhận dữ liệu đến. Đối với đồ thị và đồng hồ analog, Qt hỗ trợ vẽ trên QPainter hoặc dùng các thư viện đồ họa vector tích hợp sẵn, giúp tạo giao diện mượt mà và dễ dàng tùy biến.

EEPROM

- EEPROM là bộ nhớ không mất điện.
- ESP32 hỗ trợ lưu trữ ODO và Trip vào EEPROM để đảm bảo dữ liệu không mất khi thiết bị tắt nguồn.
- Dữ liệu được ghi định kỳ (mỗi 5 giây) hoặc khi người dùng nhấn nút reset.

Các linh kiện phần cứng khác:

- **Chiết áp (Potentiometer):** Giả lập cảm biến tốc độ và vòng tua bằng điện áp thay đổi.
- **Nút nhấn:** Giả lập thao tác của người lái: gạt số tiến/lùi, reset ODO, reset Trip.
- **Kết nối USB-TTL:** Cho phép ESP32 truyền dữ liệu sang máy tính qua cổng COM.

1.5 Phương pháp nghiên cứu

Khảo sát tài liệu: Thu thập datasheet ESP32, hướng dẫn sử dụng module Hall-effect, IC nguồn AMS1117, các bài báo và dự án tương tự.

Thiết kế sơ đồ khối: Vẽ diagram mô tả luồng tín hiệu từ cảm biến → ESP32 → UART → Qt, đồng thời luồng nguồn 12 V trên xe → mạch hạ áp → 3.3 V cấp cho ESP32 và cảm biến.

Mô phỏng sơ đồ nguyên lý: Sử dụng Proteus để vẽ schematic, kiểm tra nguyên lý lọc nhiễu, cấu hình chân IO và giả lập tín hiệu xung.

Layout PCB và kiểm tra 3D: Dùng Altium hoặc Proteus PCB để bố trí linh kiện, thiết kế ground-plane, kiểm tra khoảng cách track, tạo mô hình 3D để đánh giá độ vướng víu khi lắp ráp.

Lập trình firmware: Dựa trên ESP-IDF, cấu hình timer làm bộ đếm xung, viết ISR (Interrupt Service Routine) để đếm xung, thuật toán tính vận tốc và vòng tua, định dạng gói dữ liệu và gửi qua UART.

Phát triển ứng dụng Qt: Thiết kế UML, flowchart cho luồng dữ liệu, viết module giao tiếp serial, parser, cập nhật GUI, lưu log và chức năng xuất báo cáo.

Thử nghiệm và đánh giá: Thiết lập bench-top với function generator mô phỏng xung tốc độ/vòng tua, đo song song với bộ đo tham chiếu, thu thập dữ liệu, phân tích sai số, thời gian đáp ứng và độ ổn định theo biểu đồ.

1.6 Kết cấu báo cáo

Chương 1 – Giới thiệu: Trình bày bối cảnh, lý do chọn đề tài, mục tiêu, phạm vi, cơ sở lý thuyết, phương pháp và kết cấu báo cáo.

Chương 2 – Cơ sở lý thuyết và phân tích yêu cầu: Nghiên cứu nguyên lý cảm biến, phân tích tín hiệu đầu vào, yêu cầu chức năng và phi chức năng của hệ thống.

Chương 3 – Thiết kế phần cứng: Trình bày sơ đồ khối, sơ đồ nguyên lý Proteus, layout PCB 2 lớp, lựa chọn linh kiện và mô hình 3D.

Chương 4 – Thiết kế phần mềm: Mô tả kiến trúc ứng dụng Qt, flowchart xử lý dữ liệu, cấu trúc code, giao diện và cơ chế lưu trữ log.

Chương 5 – Sản phẩm và mã nguồn.

Chương 6 – Kết luận và hướng phát triển: Tổng kết kết quả đạt được, chỉ ra hạn chế và đề xuất hướng mở rộng như tích hợp GPS, truyền dữ liệu không dây, phát triển phiên bản di động.

Tài liệu tham khảo: Liệt kê datasheet, sách, bài báo, tài liệu hướng dẫn và mã nguồn tham khảo.

Phụ lục: Đính kèm mã nguồn chính, bảng số liệu chi tiết và hình ảnh thực tế khi lắp ráp hệ thống.

Chương 2: Cơ sở lý thuyết và phân tích yêu cầu

2.1 Cơ sở lý thuyết

2.1.1 Tốc độ và nguyên lý đo tốc độ xe

Tốc độ xe thường được đo bằng cảm biến Hall gắn trên trục bánh xe hoặc hộp số. Tín hiệu điện áp từ cảm biến này được chuyển đổi sang tốc độ bằng cách đếm số xung trên mỗi vòng quay.

Trong đề tài, do chưa sử dụng cảm biến thật, tốc độ được mô phỏng bằng chiết áp, chuyển giá trị analog 0–4095 sang tốc độ 0–220 km/h bằng hàm map().

speed = map(analogRead(POT_PIN), 0, 4095, 0, 220);

2.1.2 Vòng tua máy (RPM)

RPM là chỉ số đo tốc độ quay của trục khuỷu động cơ. Thông thường, các cảm biến Hall hoặc từ kế được dùng để đo số vòng quay trong một phút.

Trong đề tài, RPM cũng được mô phỏng bằng chiết áp, quy đổi từ điện áp thành số vòng tua trong khoảng 0–8000 vòng/phút (RPM x1000).

rpm = map(analogVal, 0, 4095, 0, 8);

2.1.3 Odometer (ODO) và Trip Meter

ODO đo tổng quãng đường xe đã đi kể từ khi khởi động lần đầu.

Trip đo quãng đường đi trong một hành trình cụ thể, có thể reset bất kỳ lúc nào.

Cả hai giá trị được tính bằng công thức vật lý:

float elapsedTime = (now - lastTime) / 1000.0;

*float distance = speed * (elapsedTime / 3600.0); // km*

Chúng được lưu vào bộ nhớ EEPROM để không mất dữ liệu khi mất điện.

2.1.4 Trạng thái hộp số (Gear)

Trong thực tế, trạng thái hộp số được nhận từ công tắc cơ điện trong cần số. Trong đề tài, ta dùng một nút nhấn để chuyển đổi giữa hai trạng thái:

- D (Drive – số tiến),
- R (Reverse – số lùi).

Việc gạt nút sẽ đảo trạng thái gearForward = !gearForward; và thay đổi dữ liệu hiển thị tương ứng.

2.1.5 Các component được dùng

Qobject

Lớp gốc của mọi object trong Qt (ngoại trừ các lớp tiện ích nhỏ như QString, QList, v.v).

Cung cấp:

- Cơ chế signal-slot.
- Tree ownership: nếu A là cha của B, thì khi A bị xóa, B cũng sẽ bị xóa tự động.
- Cơ chế phản ánh (meta-object system) → dùng cho Q_PROPERTY, signal-slot, dynamic property, v.v.

Qtimer

Dùng để gọi một hàm sau một khoảng thời gian nhất định, hoặc lặp lại nhiều lần.

Có 2 cách dùng:

- Single-shot: chạy 1 lần duy nhất.
- Interval: chạy lặp lại liên tục.

Signals – Slots

- Là hệ thống truyền thông điệp giữa các object.
- Một signal có thể kết nối với nhiều slot.
- Khi signal phát (emit), slot được thực thi.

Mục đích: Dùng để kết nối dữ liệu từ C++ sang QML, hỗ trợ binding 2 chiều.

Phân tích: Nếu bạn muốn thuộc tính từ C++ được binding trực tiếp trong QML (ví dụ: speed, rpm), bạn phải dùng Q_PROPERTY và emit signal khi giá trị thay đổi.

Q_PROPERTY

Cho phép expose các thuộc tính từ C++ sang QML.

Qsettings

- Mục đích: Lưu trữ và phục hồi giá trị ODO, Trip khi đóng/mở app.
- Phân tích: Dữ liệu lưu dưới dạng key-value, tự động ghi vào hệ thống (Windows Registry hoặc file .ini).

Connections

- Mục đích: Kết nối QML với signal từ SerialHandler.
- Phân tích: Khi SerialHandler emit signal updateData(...), dữ liệu sẽ được QML cập nhật.

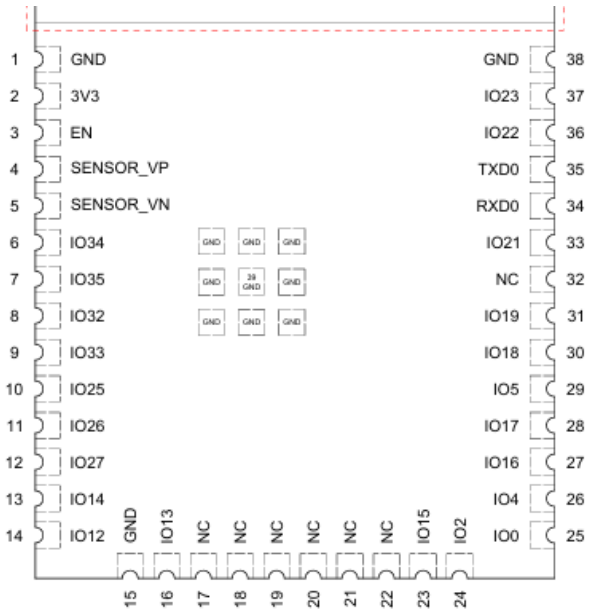
QserialPort

- Mục đích: Giao tiếp với ESP32 qua cổng UART.
- Phân tích: Nhận dữ liệu từ ESP32 gửi lên (speed, rpm, odo, trip, gear).

2.2 Mô tả linh kiện chính

2.2.1 ESP32-WROOM32

Module chứa vi điều khiển dual-core Tensilica LX6, chạy tốc độ lên đến 240 MHz, tích hợp Wi-Fi 802.11 b/g/n và Bluetooth v4.2. Các ngõ pulse counter và hardware timer được sử dụng để đếm xung Hall-effect và pickup coil. UART 0 được dùng cho debug và nạp firmware, UART 1 cho giao ti ếp với ứng dụng Qt.



Hình ảnh 1 ESP32-Wroom32

2.2.2 IC ổn áp AMS1117

AMS1117-3.3 là IC ổn áp tuyến tính, hạ áp đầu vào 12 V xuống mức 3.3 V ổn định, dòng tối đa 1 A. Để giữ ổn định, cần hai tụ hóa 10 µF đặt trước và sau IC, giảm dao động điện áp thấp tần; tụ ceramic 0.1 µF gần chân nguồn ESP32 lọc nhiễu cao tần.

Hình ảnh 2 IC AMS1117

2.2.3 Tụ và điện trở

- Tụ hóa 10 µF (C9, C10) cho ổn áp; 10 µF (C5) lọc nguồn ngõ ra.
- Tụ ceramic 0.1 µF (C6) đặt sát chân ESP32.
- Tụ lọc rung 3 µF (C1) cho nút nhấn chính; 1 nF (C2–C4, C7, C8) cho các nút/công tắc khác.
- Điện trở pull-up/pull-down 10 kΩ (R1–R5) ổn định mức logic.
- Biến trở 1 kΩ (RV1, RV2) điều chỉnh ngưỡng so sánh tín hiệu pickup coil.

2.2.4 Nút nhấn và công tắc

- Nút BUTTON (U2, U3, U5, U6) cho các chức năng Start/Stop, bật/tắt ghi log, chọn chế độ
- Công tắc slide SPST (U4) loại 100SP1T1B4M2QE để chuyển nguồn phụ hoặc chế độ vận hành.
- LED-RED (D1) báo lỗi hoặc mất kết nối, kèm điện trở hạn dòng 10 kΩ.

2.3 Phân tích yêu cầu chức năng

2.3.1 Ghi nhận và xử lý tín hiệu mô phỏng từ phần cứng

Đọc giá trị tốc độ và vòng tua máy từ chiết áp:

- Chiết áp đóng vai trò như một cảm biến đầu vào.
- Vi điều khiển ESP32 đọc giá trị analog và ánh xạ sang giá trị thực tế cho tốc độ (0–220 km/h) và vòng tua (0–8000 RPM).

Ghi nhận trạng thái số tiến/lùi từ nút nhấn:

- Nút gạt chuyển đổi giữa số tiến (D) và số lùi (R).
- Mỗi lần nhấn, trạng thái hộp số đảo chiều.

Ghi nhận nút nhấn reset ODO và Trip:

- Khi người dùng nhấn nút tương ứng, ODO hoặc Trip sẽ được đặt về 0.
- Sau đó hệ thống lưu lại giá trị mới vào EEPROM để không bị mất sau khi tắt nguồn.

2.4 Phân tích yêu cầu phi chức năng

2.4.1 Độ chính xác

Sai số đo vận tốc/vòng tua không vượt quá ±1% so với thiết bị tham chiếu. Cho phép hiệu chỉnh thông số cơ bản (chu vi, số xung/vòng) để bù sai lệch thực tế.

2.4.2 Thời gian đáp ứng

Tổng thời gian từ cạnh xung đến GUI hiển thị ≤ 100 ms. ESP32 cập nhật định kỳ, Qt xử lý dữ liệu bất đồng bộ.

2.4.3 Độ ổn định và độ tin cậy

Hệ thống chạy liên tục ≥ 8 giờ:

Sử dụng watchdog timer tự reset firmware khi treo.

Qt phát hiện mất kết nối, tự reconnect mỗi 1 s.

2.4.4 Khả năng mở rộng

Firmware và ứng dụng thiết kế module hóa:

- Cho phép thêm kênh đo analog (nhiệt độ, áp suất).
- Dễ tích hợp Wi-Fi/Bluetooth, truyền dữ liệu lên server.

2.5 Yêu cầu giao diện người dùng

2.5.1 Đồng hồ analog

Hiển thị rõ ràng thang đo:

Vận tốc: 0–200 km/h, vạch chia 10 km/h.

Vòng tua: 0–8 000 rpm, vạch chia 500 rpm.

Đồng hồ vẽ bằng QPainter hoặc widget chuyên dụng.

2.5.2 Bảng trạng thái

Cột tên tín hiệu (ON/OFF), biểu tượng LED màu: xanh (bình thường), vàng (cảnh báo), đỏ (lỗi). Hiển thị vị trí công tắc và nút nhấn.

2.5.3 Chức năng cấu hình

Dialog cho phép nhập:

- Chu vi bánh xe (m)
- Số xung/vòng quay
- Cổng serial (COM/tty)
- Thời gian gửi gói (ms)

Chương 3: Thiết kế hệ thống

3.1. Giới thiệu tổng quan

Trong chương này, hệ thống giám sát tốc độ, vòng tua máy và trạng thái hoạt động trên xe ô tô sẽ được trình bày chi tiết về mặt thiết kế phần cứng, phần mềm điều khiển (firmware) và phần mềm giao diện (ứng dụng Qt). Thiết kế hướng đến tính ổn định, khả năng xử lý thời gian thực và dễ dàng mở rộng.

3.2. Thiết kế tổng thể hệ thống

3.2.1. Sơ đồ khối chính của hệ thống

Sơ đồ khối hệ thống thể hiện các thành phần chính trong mô hình giám sát xe ô tô và cách các thành phần này tương tác với nhau.

Hệ thống bao gồm hai khối chính: khối phần cứng (ESP32 và các cảm biến) và khối phần mềm (ứng dụng Qt trên máy tính).

Khối phần cứng sử dụng ESP32 WROOM làm bộ điều khiển trung tâm, thu thập tín hiệu từ chiết áp (điều chỉnh tốc độ và vòng tua máy),nút chuyển số tiến/lùi, nút reset quãng đường (ODO và Trip).

ESP32 đọc dữ liệu và xử lý, sau đó truyền kết quả qua cổng UART đến phần mềm giao diện người dùng trên PC.

Giao diện Qt đọc dữ liệu qua cổng Serial, hiển thị dưới dạng đồng hồ tốc độ, vòng tua, ODO, Trip và trạng thái số (P/R).

Việc lưu trữ ODO và Trip được thực hiện bằng EEPROM trên ESP32 để đảm bảo dữ liệu không bị mất khi mất nguồn.

Hình ảnh 3 Sơ đồ khối chính

Mã liên quan (ESP32 - phần xử lý):

- **Chiết áp đưa vào POT_PIN, tính ra tốc độ và vòng tua.**

```
int analogVal = analogRead(POT_PIN);
speed = map(analogVal, 0, 4095, 0, 220);
rpm = map(analogVal, 0, 4095, 0, 8);
```

- **Nút chuyển số đọc tại GEAR_PIN.**

```
bool currentGearState = digitalRead(GEAR_PIN);
```

if (currentGearState != lastGearState) {

gearForward = !gearForward; }

Gửi dữ liệu qua UART.

Serial.printf("%d,%d,%.0f,%.0f,%s\n", speed, rpm, odo, trip, gearForward ? "D" : "R");

3.2.2 Sơ đồ luồng hoạt động

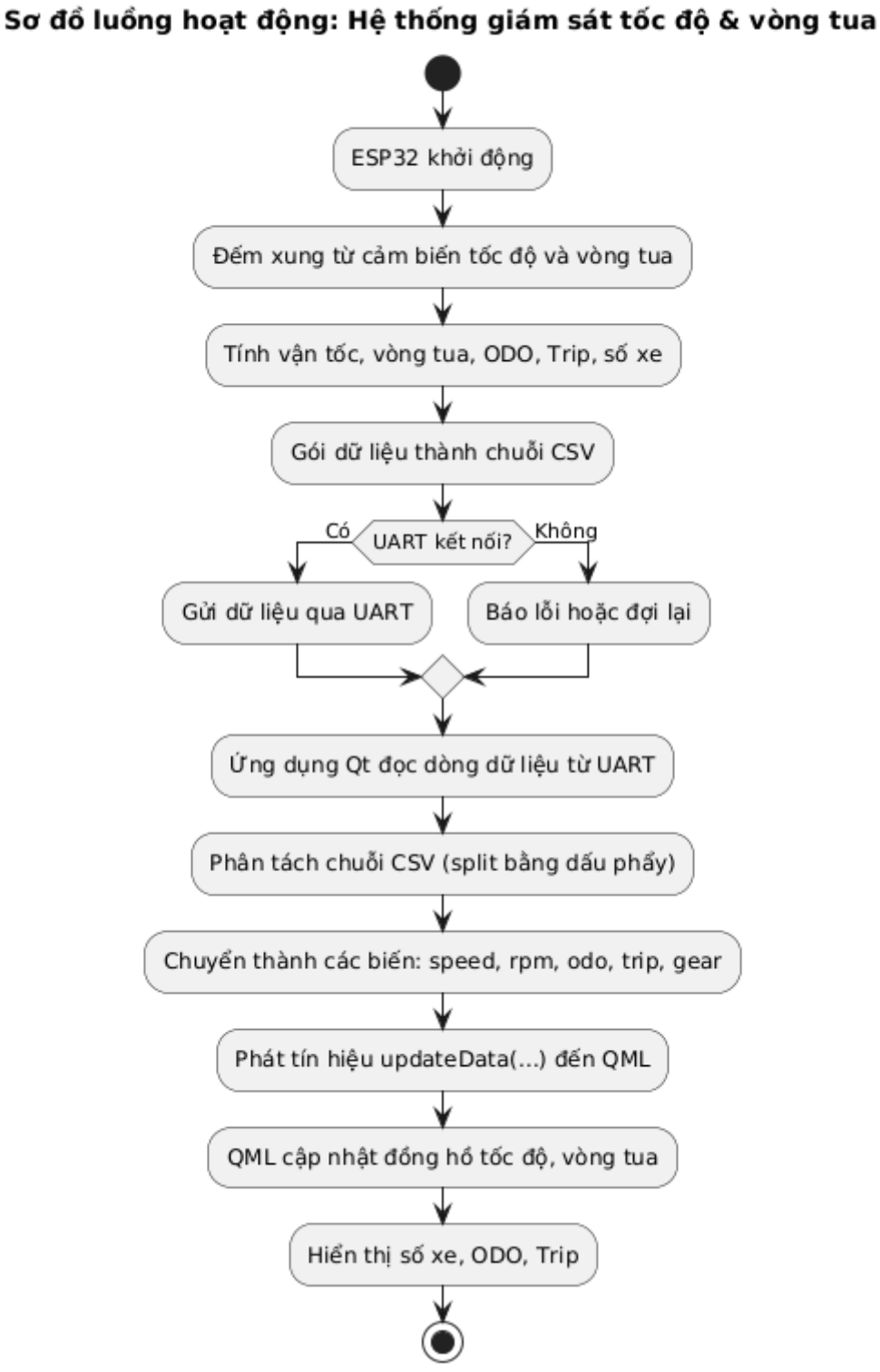
Sơ đồ luồng hoạt động mô tả tiến trình hoạt động tuần hoàn của hệ thống, bao gồm luồng xử lý trên ESP32 và trên ứng dụng Qt.

Trên ESP32:

- Khởi động hệ thống, đọc lại giá trị ODO và Trip từ EEPROM.
- Liên tục đọc chiết áp để tính tốc độ (km/h) và vòng tua (rpm).
- Tính toán quãng đường đi được dựa vào thời gian và tốc độ.
- Cập nhật giá trị ODO, Trip và lưu vào EEPROM mỗi 5 giây.
- Đọc trạng thái nút gạt số tiến/lùi và các nút reset nếu được nhấn.
- Gửi dữ liệu đã xử lý (speed, rpm, odo, trip, gear) lên PC qua Serial mỗi 100ms.

Trên ứng dụng Qt:

- Khởi chạy ứng dụng, mở kết nối Serial.
- Đặt Timer để đọc dữ liệu liên tục mỗi 100ms.
- Khi nhận được dữ liệu, parse chuỗi và cập nhật các thông số giao diện:
  đồng hồ tốc độ, vòng tua, ODO, Trip và trạng thái số.



Hình ảnh 4 Sơ đồ luồng

Luồng hoạt động chính:

- ESP32 đọc tín hiệu từ chiết áp và các nút.
- Tính toán tốc độ, RPM, ODO, Trip.

- Gửi dữ liệu qua UART.
- Qt đọc UART qua SerialHandler::readData().
- Giao diện Qt hiển thị các thông số.

Mã liên quan (Qt – đọc UART):

```
void SerialHandler::readData() {
    QByteArray data = serial.readAll();
    QList<QByteArray> values = data.split(',');

    if (values.size() == 5) {
        emit updateData(values[0].toInt(), values[1].toInt(),
                        values[2].toFloat(), values[3].toFloat(),
                        QString(values[4]).trimmed());
    }
}
```

Mã liên quan (QML – cập nhật giao diện):

```
Connections {
    target: SerialHandler
    onUpdateData: function(speedVal, rpmVal, odoVal, tripVal, gearVal) {
        speed = speedVal
        rpm = rpmVal
        odo = odoVal
        trip = tripVal
        gear = gearVal
    }
}
```

Connections là một QML type dùng để bắt tín hiệu (signal).

target: SerialHandler có nghĩa là bạn đang nghe các tín hiệu phát ra từ đối tượng C++ có tên là SerialHandler.

SerialHandler phải là **context property** đã được set từ C++ bằng QQmlContext:

```
QQmlApplicationEngine engine;
SerialHandler serialHandler;
engine.rootContext()->setContextProperty("SerialHandler", &serialHandler);
```

3.2.3 Sơ đồ lớp :Ứng dụng Qt hiện thị dữ liệu từ ESP32

Trong ứng dụng Qt, lớp xử lý chính là SerialHandler, thực hiện chức năng giao tiếp với ESP32 qua Serial và phát tín hiệu cập nhật dữ liệu.

Lớp SerialHandler có cấu trúc như sau:

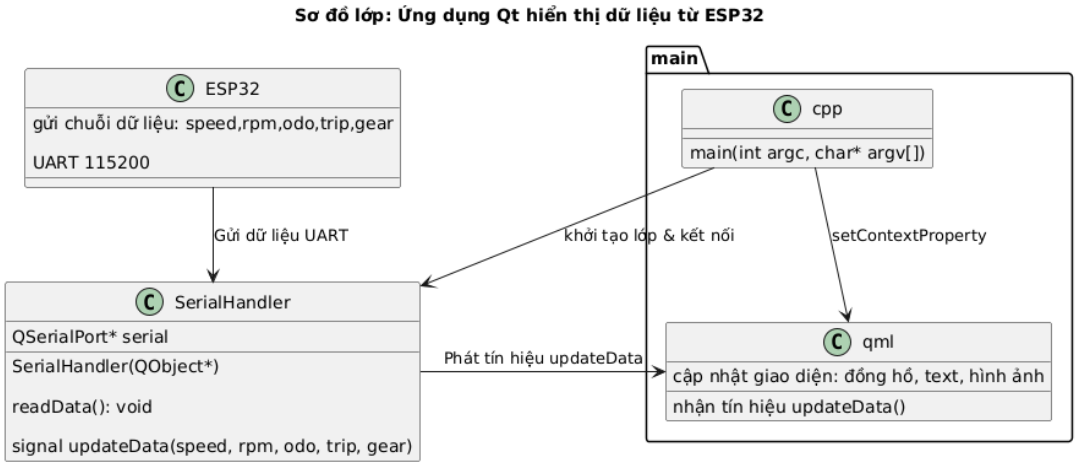
```
class SerialHandler : public QObject {
    Q_OBJECT
public:
    explicit SerialHandler(QObject *parent = nullptr);
    Q_INVOKABLE void readData();
signals:
    void updateData(int speed, int rpm, float odo, float trip, QString gear);
private:
    QSerialPort serial;
};
```

Hàm readData() được gọi theo chu kỳ 100ms từ QML để đọc dữ liệu mới từ ESP32.

Dữ liệu nhận được là một chuỗi dạng “speed,rpm,odo,trip,gear”, được parse và truyền sang QML thông qua tín hiệu updateData().

QML nhận tín hiệu này để cập nhật tốc độ, vòng tua, quãng đường và trạng thái số trên giao diện.

Thiết kế này giúp tách biệt logic xử lý dữ liệu với phần giao diện, tuân thủ mô hình MVVM (Model - View - ViewModel) trong Qt.



Hình ảnh 5 Sơ đồ lớp

Mã constructor:

```
SerialHandler::SerialHandler(QObject *parent) : QObject(parent) {
    serial.setPortName("COM3"); // chỉnh tùy máy
    serial.setBaudRate(QSerialPort::Baud115200);
    serial.open(QIODevice::ReadOnly);
}
```

SerialHandler::SerialHandler(QObject *parent) : QObject(parent)

- Đây là **constructor** của lớp SerialHandler, kế thừa từ QObject.
- Tham số QObject *parent cho phép Qt quản lý bộ nhớ thông qua **object tree** (cấu trúc cây đối tượng).
- : QObject(parent) là **initialization list**, dùng để gọi constructor của lớp cha (QObject).

serial.setPortName("COM3");

- Thiết lập cổng COM mà chương trình sẽ kết nối đến, ở đây là "COM3".

- Trong Windows, các thiết bị nối tiếp thường được gán tên kiểu COMx (COM1, COM2, COM3,...).
- **Lưu ý:** Cổng này cần được chỉnh lại theo đúng cổng mà ESP32 đang kết nối trên máy bạn.

serial.setBaudRate(QSerialPort::Baud115200);

- Thiết lập tốc độ truyền dữ liệu là **115200 baud**, phổ biến cho các thiết bị như ESP32, Arduino.

serial.open(QIODevice::ReadOnly);

- Mở cổng nối tiếp chỉ để đọc dữ liệu (không ghi).
- Nếu bạn muốn **gửi và nhận**, nên dùng QIODevice::ReadWrite.

3.2.2. Nguyên lý hoạt động chung

Tổng quan:

Dự án mô phỏng một hệ thống giám sát thông số xe hơi như: tốc độ (Speed), vòng tua máy (RPM), quãng đường đã đi (ODO), quãng đường chuyến đi hiện tại (Trip), và trạng thái hộp số (Gear - D/R). Dữ liệu được thu thập từ phần cứng ESP32 WROOM, xử lý và truyền qua cổng UART đến phần mềm giao diện viết bằng Qt C++/QML.

Nguyên lý hoạt động chi tiết

1. **Thu thập dữ liệu từ cảm biến và nút nhấn**

- Chiết áp (Potentiometer) kết nối vào chân GPIO34:

→ Mô phỏng tín hiệu từ cảm biến vận tốc và vòng tua máy.

- Nút nhấn chuyển số (GPIO32):

→ Gạt qua lại giữa số tiến (D) và số lùi (R), mỗi lần nhấn đảo trạng thái.

- Nút RESET ODO (GPIO25) và RESET TRIP (GPIO26):

→ Khi nhấn sẽ xóa dữ liệu quãng đường tương ứng.

2. **Tính toán quãng đường**

- Quãng đường = Vận tốc × Thời gian

→ Do chương trình chạy lặp liên tục, nên mỗi chu kỳ sẽ tính quãng đường nhỏ và cộng dồn vào ODO và Trip.

3. Lưu dữ liệu vào EEPROM

Mỗi 5 giây, dữ liệu ODO và Trip sẽ được lưu vào bộ nhớ EEPROM để tránh mất dữ liệu khi tắt nguồn.

```
if (now - lastSend >= 5000) {
   EEPROM.put(0, odo);
   EEPROM.put(10, trip);
   EEPROM.commit();
}
```

4. Truyền dữ liệu qua UART

ESP32 định kỳ gửi các giá trị qua Serial UART đến máy tính, với tốc độ 115200 baud, dạng chuỗi CSV:

```
Serial.printf("%d,%d,%.0f,%.0f,%s\n", speed, rpm, odo, trip, gearForward ? "D" : "R");
```

3.3. Thiết kế phần cứng

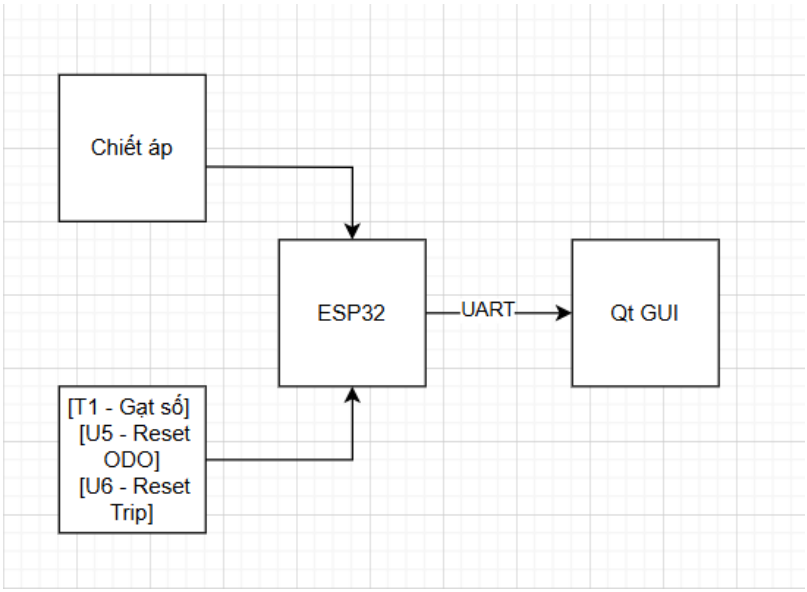
3.3.1 Tổng quan

Hệ thống phần cứng sử dụng **ESP32 WROOM** làm vi điều khiển trung tâm, kết nối với các thiết bị đầu vào (input) và xuất dữ liệu ra cổng UART để truyền về máy tính, nơi ứng dụng Qt xử lý và hiển thị.

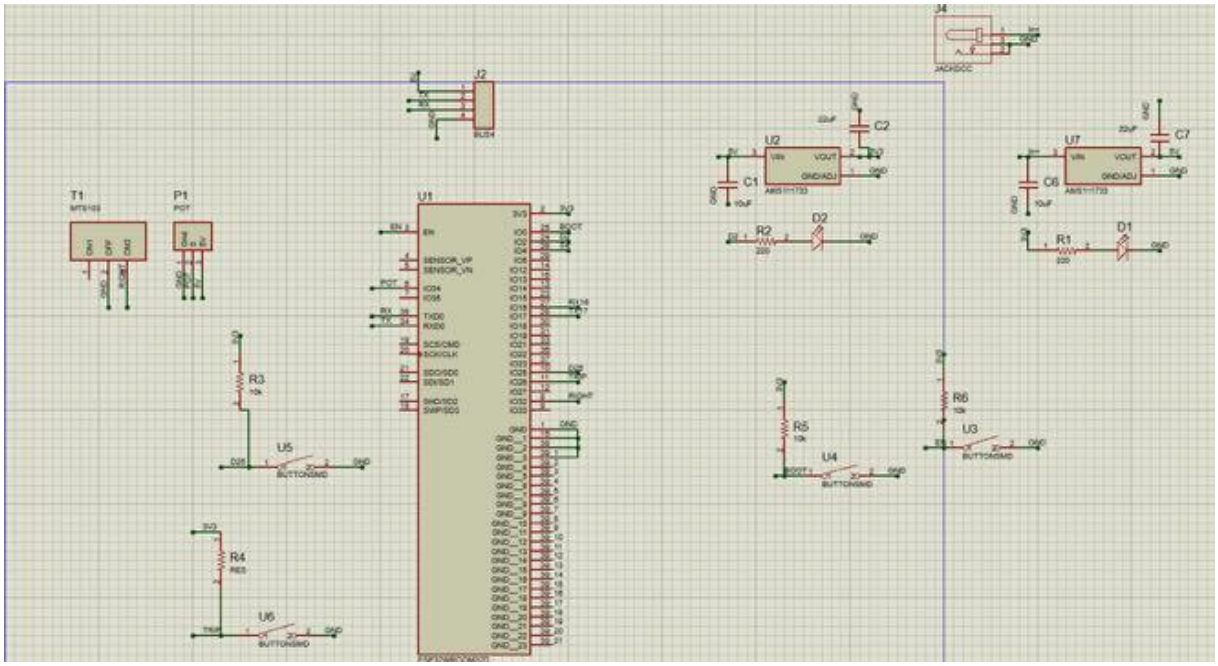
Thiết bị	**Mô phỏng chức năng**	**Kết nối với ESP32**
Chiết áp (potentiometer)	Mô phỏng cảm biến tốc độ và RPM	GPIO 34 (ADC)
Nút nhấn T1	Chuyển số tiến / lùi	GPIO 32
Nút nhấn U5	Reset ODO	GPIO 25
Nút nhấn U6	Reset Trip	GPIO 26

Thiết bị	Mô phỏng chức năng	Kết nối với ESP32
UART TTL USB	Giao tiếp với máy tính	TX/RX, Baud: 115200

Sơ đồ khối phần cứng:

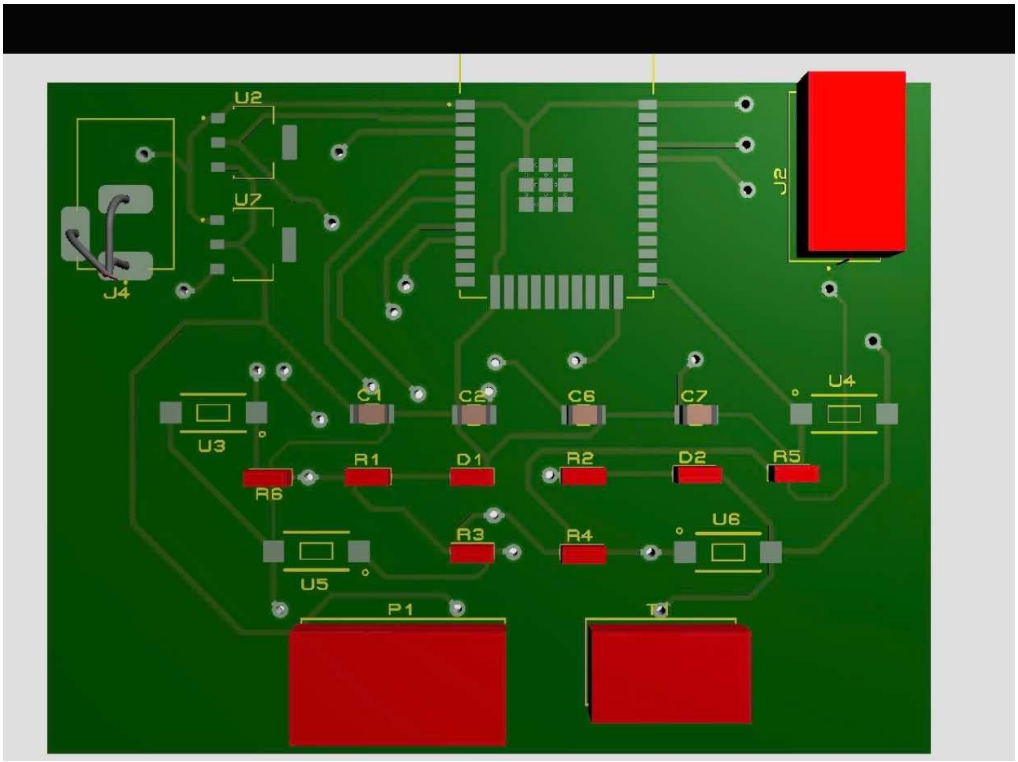


Sơ đồ mạch:



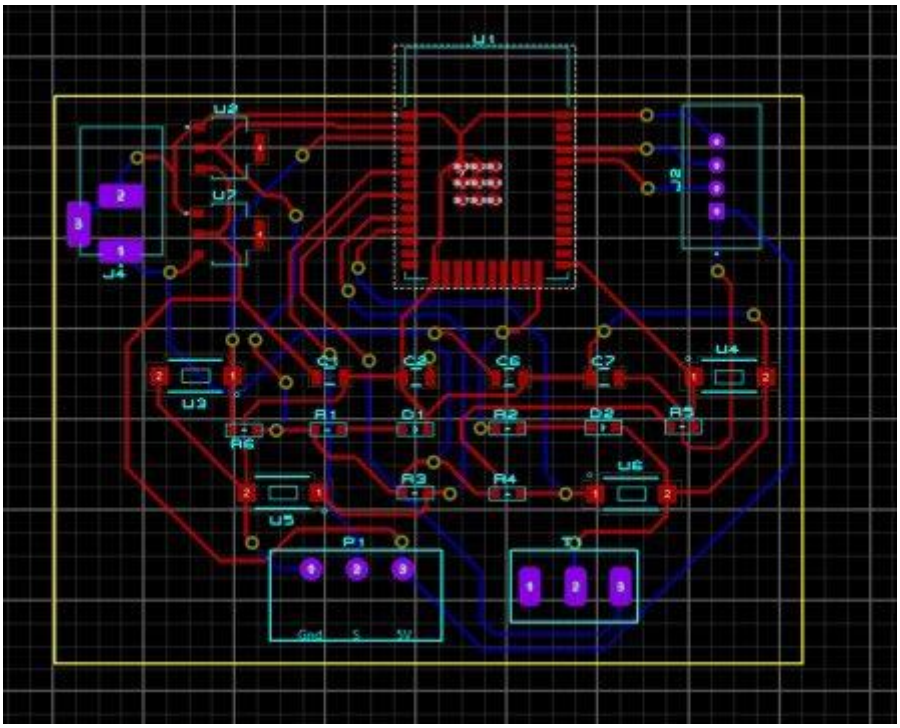
Hình ảnh 6 Sơ đồ mạch

Mô hình 3D:



Hình ảnh 7 Mô hình 3D

Mạch PCB:



Hình ảnh 8 Mạch PCB

CHƯƠNG 4: SẢN PHẨM VÀ MÃ NGUỒN

4.1 Tổng quan sản phẩm

Hệ thống được xây dựng hoàn chỉnh bao gồm hai thành phần:

Phần cứng: Một vi điều khiển ESP32 kết nối với chiết áp,nút chuyển đổi để đo tốc độ và vòng tua máy,trạng thái số, đồng thời truyền dữ liệu UART về máy tính.

Phần mềm: Ứng dụng giao diện người dùng (GUI) viết bằng Qt/QML, có khả năng nhận dữ liệu thời gian thực từ ESP32, hiển thị vận tốc, vòng tua máy, quãng đường (ODO, Trip) và trạng thái số xe (D/R) qua các đồng hồ số trực quan.

4.2 Mã nguồn chính

4.2.1 Mã ESP32 (esp32_vehicle_monitor.ino)

```
#include <EEPROM.h>
#define POT_PIN      34  // Chiết áp
#define GEAR_PIN     32  // Nút gạt tiến/lùi
#define RESET_ODO    25  // Nút reset ODO
#define RESET_TRIP   26  // Nút reset Trip
int speed = 0;
int rpm = 0;
float odo = 0.0;
float trip = 0.0;
bool gearForward = true;
bool lastGearState = HIGH;
unsigned long lastSend = 0;
unsigned long lastTime = 0;
void setup() {
  Serial.begin(115200);
  pinMode(GEAR_PIN, INPUT_PULLUP);
  pinMode(RESET_ODO, INPUT_PULLUP);
  pinMode(RESET_TRIP, INPUT_PULLUP);
```

```
  EEPROM.begin(512);
  EEPROM.get(0, odo);
  if (isnan(odo)) odo = 0;
  EEPROM.get(10, trip);
  if (isnan(trip)) trip = 0;
  lastTime = millis();
void loop() {
  // Đọc giá trị chiết áp
  int analogVal = analogRead(POT_PIN);
  speed = map(analogVal, 0, 4095, 0, 120);   // Giới hạn tốc độ 0-220 km/h
  rpm = map(analogVal, 0, 4095, 0, 8);       // RPM x1000
  // Tính quãng đường đi được
  unsigned long now = millis();
  float elapsedTime = (now - lastTime) / 1000.0; // giây
  lastTime = now;
  float distance = speed * (elapsedTime / 3600.0); // km = km/h * h
  odo += distance;
  trip += distance;
  // Lưu ODO & Trip sau mỗi 5 giây
  if (now - lastSend >= 5000) {
    EEPROM.put(0, odo);
    EEPROM.put(10, trip);
    EEPROM.commit();
    lastSend = now;
  }
  // Đọc nút reset ODO
  if (digitalRead(RESET_ODO) == LOW) {
```

```
    odo = 0;
    EEPROM.put(0, odo);
    EEPROM.commit();
    delay(300);
  }
  // Đọc nút reset Trip
  if (digitalRead(RESET_TRIP) == LOW) {
    trip = 0;
    EEPROM.put(10, trip);
    EEPROM.commit();
    delay(300);
  }
  // Đọc trạng thái số tiến/lùi
  bool currentGearState = digitalRead(GEAR_PIN);
  if (currentGearState != lastGearState) {
    gearForward = !gearForward;
    lastGearState = currentGearState;
    delay(200);
  }
  // Gửi dữ liệu lên Qt qua Serial
  Serial.printf("%d,%d,%.0f,%.0f,%s\n", speed, rpm, odo, trip, gearForward ? "D" : "R");
  delay(100);
}
```

4.2.2 Mã nguồn giao diện Qt

(1) File main.cpp:

```
#include <QGuiApplication>
#include <QQmlApplicationEngine>
#include <QQmlContext>
```

```
#include "serialhandler.h"
int main(int argc, char *argv[])
{
    QGuiApplication app(argc, argv);
    SerialHandler serialHandler;
    QQmlApplicationEngine engine;
    engine.rootContext()->setContextProperty("SerialHandler", &serialHandler);
    engine.load(QUrl(QStringLiteral("qrc:/main.qml")));
    return app.exec();
}
```

(2) File serialhandler.h:

```
#ifndef SERIALHANDLER_H
#define SERIALHANDLER_H
#include <QObject>
#include <QSerialPort>
class SerialHandler : public QObject
{
    Q_OBJECT
public:
    explicit SerialHandler(QObject *parent = nullptr);
    Q_INVOKABLE void readData();
signals:
    void updateData(int speed, int rpm, int odo, int trip, QString gear);
private:
    QSerialPort *serial;
};
#endif // SERIALHANDLER_H
```

(3) File serialhandler.cpp:

```
#include "serialhandler.h"
#include <QDebug>
SerialHandler::SerialHandler(QObject *parent) : QObject(parent)
{
    serial = new QSerialPort(this);
    serial->setPortName("COM3"); // Cập nhật lại COM nếu khác
    serial->setBaudRate(QSerialPort::Baud115200);
    serial->open(QIODevice::ReadOnly);
}
void SerialHandler::readData()
{
    while (serial->canReadLine()) {
        QByteArray line = serial->readLine().trimmed();
        QList<QByteArray> parts = line.split(',');
        if (parts.size() == 5) {
            int speed = parts[0].toInt();
            int rpm = parts[1].toInt();
            int odo = parts[2].toInt();
            int trip = parts[3].toInt();
            QString gear = QString(parts[4]);
            emit updateData(speed, rpm, odo, trip, gear);
        }
    }
}
}
```

(4) File main.qml (giao diện):

```
import QtQuick 2.15
import QtQuick.Controls 2.15
```

```
import QtQuick.Layouts 1.15
Window {
    visible: true
    width: 800
    height: 480
    title: qsTr("Vehicle Monitor")
    property int speed: 0
    property int rpm: 0
    property int odo: 0
    property int trip: 0
    property string gear: "D"
    Rectangle {
        anchors.fill: parent
        color: "#202020"
        // Mặt đồng hồ tốc độ
        Image {
            id: speedImage
            source: "qrc:/images/SpeedMetter.png"
            width: 300
            height: 300
            anchors.left: parent.left
            anchors.leftMargin: 20
            anchors.top: parent.top
            anchors.topMargin: 20
        }
        // Mặt đồng hồ vòng tua
        Image {
            id: rpmImage
```

```
    source: "qrc:/images/TachoMetter.png"
    width: 300
    height: 300
    anchors.right: parent.right
    anchors.rightMargin: 20
    anchors.top: parent.top
    anchors.topMargin: 20
}
// Kim tốc độ
Image {
    id: speedNeedle
    source: "qrc:/images/Indicator.png"
    width: 110
    height: 20
    anchors.centerIn: speedImage
    transform: Rotation {
        origin.x: 36
        origin.y: speedNeedle.height / 4
        angle: speed *1.1 + 143  // scale và dịch về góc bắt đầu = 0
    }
}
Image {
    id: rpmNeedle
    source: "qrc:/images/Indicator.png"
    width: 110
    height: 20
    anchors.centerIn: rpmImage
    transform: Rotation {
```

```
        origin.x: 36
        origin.y: rpmNeedle.height / 4
        angle: rpm * 30 + 143   // RPM 0 → -120°, RPM 8 → 120°
    }
}
Rectangle {
    width: parent.width
    height: 140
    anchors.bottom: parent.bottom
    color: "#303030"
    radius: 10
    border.color: "#505050"
    border.width: 1
    RowLayout {
        anchors.fill: parent
        anchors.margins: 20
        spacing: 40
        // Gear
        Column {
            spacing: 4
            Layout.preferredWidth: 100
            Text {
                text: "Gear"
                font.pixelSize: 20
                color: "#AAAAAA"
            }
            Text {
                text: gear
```

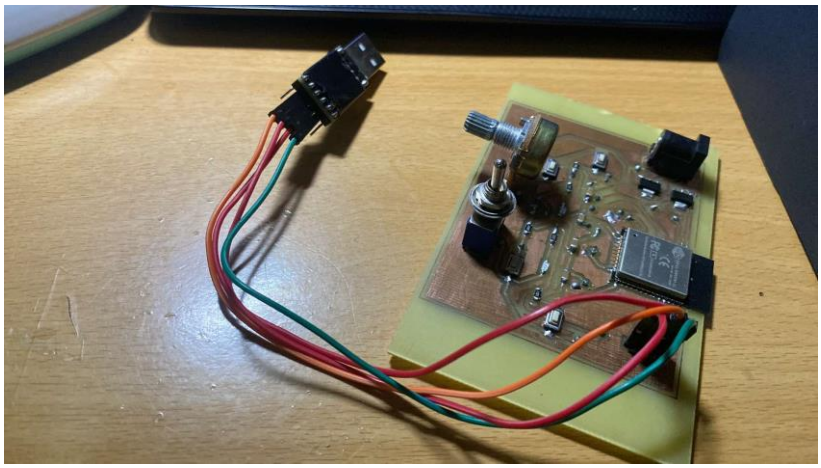
```
        font.pixelSize: 26
        font.bold: true
        color: "#00BFFF"
    }
}
// Speed
Column {
    spacing: 4
    Layout.preferredWidth: 150
    Text {
        text: "Speed (km/h)"
        font.pixelSize: 20
        color: "#AAAAAA"
    }
    Text {
        text: speed
        font.pixelSize: 26
        font.bold: true
        color: "white"
    }
}
// RPM
Column {
    spacing: 4
    Layout.preferredWidth: 150
    Text {
        text: "RPM (x1000)"
        font.pixelSize: 20
```

```
        color: "#AAAAAA"
    }
    Text {
        text: rpm
        font.pixelSize: 26
        font.bold: true
        color: "white"
    }
}
// ODO
Column {
    spacing: 4
    Layout.preferredWidth: 150
    Text {
        text: "ODO (km)"
        font.pixelSize: 20
        color: "#AAAAAA"
    }
    Text {
        text: odo
        font.pixelSize: 26
        font.bold: true
        color: "lightgreen"
    }
}
// Trip
Column {
    spacing: 4
```

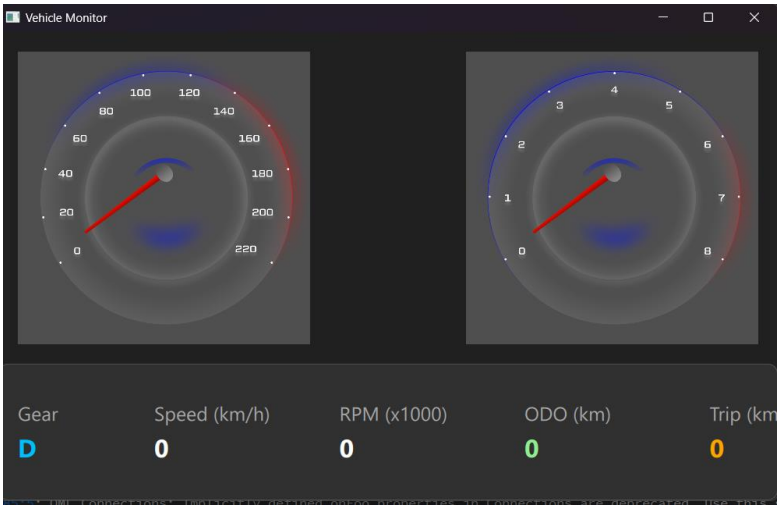
```
            Layout.preferredWidth: 150
            Text {
                text: "Trip (km)"
                font.pixelSize: 20
                color: "#AAAAAA"
            }
            Text {
                text: trip
                font.pixelSize: 26
                font.bold: true
                color: "orange"
            }
        }
    }
}
// Đọc dữ liệu từ ESP32
Timer {
    interval: 100
    running: true
    repeat: true
    onTriggered: SerialHandler.readData()
}
// Nhận tín hiệu cập nhật từ SerialHandler
Connections {
    target: SerialHandler
    onUpdateData: function(speedVal, rpmVal, odoVal, tripVal, gearVal) {
        speed = speedVal
```

```
            rpm = rpmVal
            odo = odoVal
            trip = tripVal
            gear = gearVal
        }
    }
}
```

5.3 Giao diện sản phẩm



Hình ảnh 9 Sản phẩm phần cứng



Hình ảnh 10 Giao diện sản phẩm

Chương 5: Kết luận và hướng phát triển

Kết luận

Sau một quá trình nghiên cứu, thiết kế và triển khai, nhóm đã hoàn thành đề tài "Thiết kế và triển khai hệ thống giám sát tốc độ, vòng tua máy và trạng thái trên xe ô tô". Hệ thống đã đáp ứng được các mục tiêu đề ra ban đầu, bao gồm:

- Thu thập dữ liệu từ các cảm biến tốc độ và vòng tua bằng tín hiệu xung số.
- Xử lý và tính toán thông tin vận tốc và vòng tua máy thông qua vi điều khiển ESP32.
- Truyền dữ liệu qua giao tiếp UART với tốc độ ổn định, thời gian thực.
- Hiển thị trực quan các thông số trên phần mềm Qt chạy trên Linux, với giao diện thân thiện và dễ sử dụng.
- Báo trạng thái hoạt động, lỗi hoặc kết nối của hệ thống để người dùng có thể dễ dàng giám sát.

Hệ thống được đánh giá là hoạt động ổn định, có thể ứng dụng vào các tình huống thực tế như lắp đặt thử nghiệm trên xe mô hình, hoặc tích hợp trong hệ thống giáo dục, học tập, nghiên cứu về nhúng và IoT trong ngành công nghệ ô tô.

Thông qua quá trình làm đồ án, nhóm đã rèn luyện được các kỹ năng thiết kế phần cứng, lập trình vi điều khiển, thiết kế phần mềm giao diện người dùng và quan trọng nhất là khả năng làm việc nhóm và giải quyết vấn đề trong thực tiễn.

Hạn chế của hệ thống

Mặc dù hệ thống đã vận hành đúng yêu cầu, nhưng vẫn còn tồn tại một số điểm hạn chế như:

- Độ chính xác đo vận tốc và vòng tua còn phụ thuộc nhiều vào độ nhạy của cảm biến và điều kiện môi trường.
- Phần giao diện người dùng hiện tại mới dừng ở mức hiển thị cơ bản, chưa có tính năng phân tích dữ liệu hoặc lưu trữ dài hạn.
- Hệ thống chỉ hoạt động qua giao tiếp dây UART, chưa hỗ trợ không dây (WiFi/Bluetooth).

- Cần hiệu chỉnh thủ công biến trở khi thay đổi cảm biến hoặc thay đổi loại xe.

Hướng phát triển trong tương lai

- Để nâng cao hiệu quả và mở rộng phạm vi ứng dụng của hệ thống, một số hướng phát triển khả thi được đề xuất như sau:
- Nâng cấp giao tiếp sang sử dụng Bluetooth hoặc WiFi để tiện lợi hơn trong kết nối với điện thoại hoặc máy tính bảng.
- Thiết kế phần mềm Qt đa nền tảng, có thể chạy trên cả Windows, Linux và Android.
- Tích hợp lưu trữ dữ liệu vào thẻ nhớ SD để hỗ trợ xem lại lịch sử hoạt động, thuận tiện cho việc bảo trì và đánh giá vận hành phương tiện.
- Bổ sung thêm cảm biến khác như cảm biến nhiệt độ động cơ, cảm biến nhiên liệu để mở rộng hệ thống giám sát.
- Ứng dụng AI hoặc xử lý tín hiệu số nâng cao để phân tích dữ liệu thu thập được và dự đoán các trạng thái hỏng hóc.
- Thiết kế mạch PCB nhỏ gọn hơn, chuẩn công nghiệp để có thể thương mại hóa hoặc đưa vào thực nghiệm thực tế.

Lời kết

Việc hoàn thành đề tài không chỉ giúp nhóm hiểu sâu hơn về hệ thống nhúng, lập trình vi điều khiển và phát triển phần mềm giao diện người dùng, mà còn là cơ hội để vận dụng lý thuyết vào thực tế. Đây là bước đệm quan trọng để nhóm tiếp tục các nghiên cứu sâu hơn trong lĩnh vực IoT, tự động hóa và công nghệ ô tô trong tương lai.

TÀI LIỆU THAM KHẢO

Tiếng Anh

1. **Espressif Systems**

ESP32 Technical Reference Manual

URL: https://www.espressif.com/en/products/socs/esp32/resources

2. **Qt Documentation**

Official Qt 6 Documentation

URL: https://doc.qt.io/

3. **QML and Qt Quick**

Qt Quick Controls & QML Basics

URL: https://doc.qt.io/qt-6/qtquick-index.html

4. **Serial Communication**

Qt SerialPort Module Documentation

URL: https://doc.qt.io/qt-6/qtserialport-index.html