ĐẠI HỌC QUỐC GIA TP.HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA ĐIỆN – ĐIỆN TỬ
BỘ MÔN ĐIỆN TỬ
----------o0o----------

BÁO CÁO
ĐỒ ÁN 2 (KỸ THUẬT ĐIỆN TỬ - VIỄN THÔNG)

Đề tài:

Hệ thống giám sát áp suất lốp xe ô tô

(Tire-Pressure Monitoring System)

GVHD: PGS. TS. Trương Quang Vinh
SVTH: Võ Thành Danh
MSSV: 2110902

TP. HỒ CHÍ MINH, THÁNG 8 NĂM 2024

LỜI CẢM ƠN

Lời đầu tiên, em xin chân thành cảm ơn thầy, PGS.TS. Trương Quang Vinh đã đồng ý cho em bảo vệ bài Đồ án "Hệ thống giám sát áp suất lốp xe ô tô" của em, đồng thời, thầy đã giúp đỡ em hết mình, tạo cho em những điều kiện tốt nhất cũng như đề xuất những hướng đi hợp lý và đúng đắn trong quá trình 3 tháng thực hiện Đồ án qua.

Em xin chân thành cảm ơn!

Tp. Hồ Chí Minh, ngày tháng 12 năm 2024 .

Sinh viên

TÓM TẮT ĐỒ ÁN

Trong đa số các dòng xe ô tô hiện nay, các hệ thống giám sát áp suất lốp xe ô tô đang càng ngày càng phổ biến. Các hệ thống này giúp cho các tài xế có thể dễ dàng kiểm soát tình trạng lốp xe của mình, giúp bảo quản lốp xe tốt hơn và đảm bảo an toàn cho tài xế trong trường hợp lốp xe bị thủng.

Đề tài *"Hệ thống giám sát áp suất lốp xe ô tô"* (từ nay xin được viết tắt là TPMS) là một hệ thống mô phỏng quá trình đọc và cập nhật áp suất lốp trong xe ô tô. Hệ thống sẽ đọc liên tục áp suất không khí bên trong lốp xe và sẽ gửi không dây liên tục đến bộ ECU áp suất lốp sử dụng giao thức BLE và sẽ được điều khiển bởi ESP32-C6. ECU áp suất lốp sau khi nhận được thông số từ cảm biến sẽ gửi cho bộ ECU màn hình tài xế sử dụng CAN. ECU màn hình sẽ sử dụng màn hình OLED 0.96in để hiển thị các thông tin nhận được từ ECU áp suất lốp, giúp cho tài xế có thể theo dõi liên tục tình trạng của lốp xe. Hai ECU lấy vi điều khiển dòng STM32F0 để điều khiển các luồng hoạt động của hệ thống và sẽ được nuôi bởi nguồn 12V từ adapter.

Đề tài TPMS chỉ mang tính chất mô phỏng hoạt động của 1 bộ TPMS ngoài thị trường nên chỉ có thể đáp ứng được 70 – 80% chức năng của các TPMS ngoài thị trường. Đồng thời, vì lý do ngân sách không quá cao nên đề tài TPMS sẽ cắt giảm một số yếu tố mà một xe ô tô cần như màn hình lớn với hệ điều hành hay bình ắc quy và sẽ được thử nghiệm hoạt động trong không gian của một ống tiêm.

MỤC LỤC

DANH SÁCH HÌNH MINH HỌA **5**

DANH SÁCH BẢNG SỐ LIỆU **7**

1. GIỚI THIỆU **8**

1.1 Định nghĩa đề tài ***8***

1.1.1 TPMS trực tiếp 9

1.1.2 TPMS gián tiếp 10

1.2 Nhiệm vụ đề tài ***12***

1.3 Phạm vi đề tài ***13***

1.4 Ứng dụng của đề tài ***13***

2. CƠ SỞ LÝ THUYẾT **14**

2.1 Control Area Network ***14***

2.1.1 Giới thiệu 14

2.1.2 Kiến trúc 14

2.1.3 Đặc điểm 17

2.1.4 Tốc độ truyền dữ liệu 17

2.1.5 Cơ chế phân quyền 19

2.1.6 Cơ chế bộ lọc 20

2.1.7 Các khung dữ liệu trong CAN 21

2.1.8 Các loại lỗi và cơ chế tự phát hiện lỗi 24

2.1.9 Các phiên bản của CAN 25

2.2 Bluetooth Low Energy ***28***

2.2.1 Giới thiệu BLE 28

2.2.2 Kiến trúc BLE 28

2.3 Cảm biến áp suất ***33***

2.4 Kỹ thuật decoupling ***34***

2.5 Buck converter cơ bản 36

3. THỰC HIỆN ĐỀ TÀI 38

3.1 Lựa chọn phần cứng 38

3.1.1 Hệ thống cảm biến 38

3.1.2 ECU TPMS 40

3.1.3 ECU màn hình 41

3.2 Thiết kế phần cứng 41

3.2.1 Hệ thống cảm biến 41

3.2.2 ECU TPMS 42

3.2.3 ECU màn hình 54

3.3 Lập trình firmware 56

3.3.1 ESP32C6 trong hệ thống cảm biến 57

3.3.2 ESP32C6 trong ECU TPMS 58

3.3.3 STM32 trong TPMS 60

3.3.4 STM32 trong ECU màn hình 61

4. KẾT QUẢ THỰC HIỆN ĐỀ TÀI 63

5. TÀI LIỆU THAM KHẢO 66

6. PHỤ LỤC 67

6.1 Điều khiển màn hình OLED SSD1306 67

6.2 Mã nguồn hệ thống 67

DANH SÁCH HÌNH MINH HỌA

Hình 1-1: Hình ảnh báo hiệu lốp xe có vấn đề 8
Hình 1-2: Màn hình hiển thị các dữ liệu của các lốp xe của 1 TPMS 9
Hình 1-3: So sánh phổ tín hiệu tần số cộng hưởng giữa lốp xe bình thường và lốp xe non 11
Hình 2-1: Sơ đồ kết nối các node và CAN bus 15
Hình 2-2: Hình dạng thực tế của CAN Bus 15
Hình 2-3: Trạng thái bit dominat và recessive trong CAN 16
Hình 2-4: Các phân đoạn trong thời gian tồn tại của một bit 18
Hình 2-5: Cơ chế phân quyền góc nhìn từng bit 20
Hình 2-6: Cấu trúc Data Frame 21
Hình 2-7: Cấu trúc và hoạt động của Remote Frame 22
Hình 2-8: Cấu trúc của Error Frame 23
Hình 2-9: Cấu trúc của Overload Frame 24
Hình 2-10: Cấu trúc Data Frame của CAN 2.0B 26
Hình 2-11: Cấu trúc Data Frame của CAN FD so với CAN 2.0A 27
Hình 2-12: Các lớp trong kiến trúc của BLE 29
Hình 2-13: Cấu trúc dữ liệu và hoạt động mẫu trong lớp GATT 33
Hình 2-14: Sơ đồ của một tụ điện thực với các yếu tố ký sinh 35
Hình 2-15: Trở kháng của các tụ 100µF khác nhau theo tần số 36
Hình 2-16: Schematic mạch buck converter cơ bản 37
Hình 3-1: Mạch phát triển ESP32-C6 Super Mini của Nologo 40
Hình 3-2: Module màn hình OLED 0.96in SSD1306 hỗ trợ giao tiếp I^2C 41
Hình 3-3: Schematic mạch vi điều khiển 42
Hình 3-4: Schematic mạch decoupling digital và analog 43
Hình 3-5: Schematic mạch nút reset, mạch chọn chế độ boot và mạch dao động ngoại 44
Hình 3-6: Ảnh hưởng của nhiệt độ đến độ chính xác của bộ dao động nội tần số cao 46
Hình 3-7: Đặc tính của bộ dao động ngoại tần số cao 46

Hình 3-8: Schematic mạch CAN transceiver 47
Hình 3-9: Schematic mạch nguồn 48
Hình 3-10: Sơ đồ khối của TPS54331 50
Hình 3-11: Schematic của các mạch khác của ECU TPMS 51
Hình 3-12: Mạch nạp ST-Link V2 52
Hình 3-13: Layout mạch ECU TPMS 53
Hình 3-14: Mặt trước của PCB ECU TPMS (mô hình 3D) 53
Hình 3-15: Mặt sau của PCB ECU TPMS (mô hình 3D) 54
Hình 3-16: Schematic của các mạch khác trong ECU màn hình 55
Hình 3-17: Layout của mạch ECU màn hình 55
Hình 3-18: Mặt trước của PCB ECU màn hình (mô hình 3D) 56
Hình 3-19: Mặt sau của PCB ECU màn hình (mô hình 3D) 56
Hình 3-20: Sơ đồ giải thuật của hệ thống cảm biến 58
Hình 3-21: Sơ đồ giải thuật của ESP32C6 trong ECU TPMS 59
Hình 3-22: Sơ đồ giải thuật của STM32 trong ECU TPMS 61
Hình 3-23: Sơ đồ giải thuật của STM32 trong ECU màn hình 62
Hình 4-1: Tổng quan về phần cứng của đề tài 63
Hình 4-2: Quan sát Serial Monitor khi ESP32C6 trong hệ thống cảm biến đang hoạt động 63
Hình 4-3: Quan sát Serial Monitor khi ESP32C6 trong ECU TPMS đang hoạt động 64
Hình 4-4: Quan sát debug dữ liệu STM32 trong ECU TPMS sau khi nhận SPI 64
Hình 4-5: Quan sát debug dữ liệu STM32 trong ECU TPMS đóng khung và sẽ gửi cho ECU màn hình 64
Hình 4-6: Các kết quả cuối cùng in ra màn hình OLED 65

DANH SÁCH BẢNG SỐ LIỆU

Bảng 1-1: So sánh TPMS trực tiếp và TPMS gián tiếp 12

Bảng 2-1: So sánh giữa BLE và Bluetooth 28

Bảng 2-2: Các phương thức làm việc với các thuộc tính trong lớp ATT 32

Bảng 3-1: Sơ đồ kết nối các chân của XGZP6847A500KPG[8] 41

Bảng 3-2: Quy đổi giá trị 4 bit tình trạng và % so với tình trạng ban đầu 57

1. GIỚI THIỆU

1.1 Định nghĩa đề tài

Xã hội ngày càng phát triển, kèm theo đó là nhu cầu sống của mỗi người cũng tăng lên, cho nên các tiện nghi xung quanh ta từ đó phải phát triển không ngừng để theo kịp với nhu cầu của người dùng. Các dòng xe ô tô cũng không phải ngoại lệ, chúng phải không ngừng phát triển để đáp ứng được nguồn cầu của thị trường hiện nay. Xu hướng các tài xế có thể tiếp cận được các thông số quan trọng của xe một cách thuận tiện nhất đang được các dòng xe để ý tới. Ví dụ như các tài xế có thể quan sát được mức xăng, tốc độ, các lỗi đang xảy ra với xe,… chỉ qua chiếc màn hình trước mặt họ.

Với xu hướng như thế, các nhà phát triển đã sáng tạo ra một hệ thống giúp các tài xế có thể quan sát được tình trạng lốp xe hiện tại của họ để giúp họ có thể bao trì nó tốt hơn và cũng như giúp phát hiện nhanh chóng những sự cố của lốp xe khi đang di chuyển. Khi hệ thống phát hiện lốp xe hơi non hơn so với tiêu chuẩn thì sẽ báo hiệu bằng cách cho sáng ký tự như bên dưới hình 1-1. Hệ thống được gọi với cái tên tiếng Anh là **Tire-Pressure Monitoring System** (thường được gọi là **TPMS**), tạm thời được dịch là “Hệ thống giám sát áp suất lốp xe ô tô”.

Hình 1-1: Hình ảnh báo hiệu lốp xe có vấn đề

TPMS bao gồm 2 dạng chính là TPMS trực tiếp và TPMS gián tiếp.

1.1.1 TPMS trực tiếp

TPMS trực tiếp sử dụng hệ thống cảm biến áp suất không khí đặt ở vành lốp xe để đo trực tiếp áp suất không khí bên trong lốp xe. Dữ liệu đo được kèm với các tình trạng của hệ thống như thời lượng pin, tình trạng kết nối, tình trạng cảm biến,… sẽ được truyền không dây đến một bộ thu nằm ở thân xe. Bộ thu này sau đó sẽ xử lý dữ liệu nhận được và sẽ đem dữ liệu đó đi đến những hệ thống khác cần sử dụng, một trong số đó chính là hệ thống màn hình trước mặt tài xế. Khi hệ thống màn hình nhận dữ liệu, nó sẽ hiển thị lên cho tài xế và thực hiện một số lệnh được gửi kèm theo dữ liệu ví dụ như lệnh bật đèn báo hiệu lốp xe bị vấn đề.

Hình 1-2: Màn hình hiển thị các dữ liệu của các lốp xe của 1 TPMS

TPMS bao gồm 2 thành phần là hệ thống cảm biến và ECU TPMS.

- **Hệ thống cảm biến** được đặt bên trong lốp xe, thường sẽ là một phần trong hệ thống van lốp xe để đo áp suất không khí bên trong. Giá trị đo được sẽ được gửi đến ECU TPMS thông qua một giao thức không dây như Wifi, Bluetooth, Thread, BLE,… Hệ thống cảm biến bao gồm những thành phần cơ bản sau:
    - Cảm biến đo áp suất không khí
    - Vi điều khiển có thể đọc và xử lý các giá trị từ cảm biến, đồng thời có thể truyền không dây dữ liệu sau xử lý cho ECU TPMS
    - Mạch nguồn để cấp nguồn cho 2 mạch ở trên.

  Hệ thống cảm biến trong các hệ thống TPMS trực tiếp phải đáp ứng được những yêu cầu như tiết kiệm năng lượng, chịu được áp suất, nhiệt độ cao hay có thể theo dõi tình trạng pin của cảm biến.

- **ECU TPMS** (viết tắt của Electronic Control Unit Tire-Pressure Monitoring System) là bộ điều khiển trung tâm của TPMS, là "bộ não" của TPMS và bộ trung gian giúp cho dữ liệu về áp suất lốp có thể đi đến tài xế. **ECU – Electronic Control Unit** là một hệ thống mạch điện tử trong một hệ thống nhúng để điều khiển một chức năng cụ thể nào đó trong xe ô tô. ECU lấy ngõ vào từ các cảm biến hoặc các ECU khác để xử lý và điều khiển và điều phối các hoạt động trong xe ô tô[1], ví dụ là bộ ECU điều khiển động cơ lấy ngõ vào từ cảm biến bàn đạp ga, cảm biến tải trọng, ECU phanh,…, xử lý và sử dụng các bộ truyền động để điều khiển động cơ xe ô tô. ECU TPMS nhận ngõ vào từ hệ thống cảm biến và điều phối thông tin này đến ECU màn hình tài xế. Một ECU TPMS sẽ bao gồm những thành phần cơ bản sau:
    - Vi điều khiển vừa để nhận thông tin không dây từ hệ thống cảm biến, vừa điều khiển hoạt động giao tiếp với các ECU khác sử dụng giao tiếp CAN.
    - CAN Transceiver giúp "biên dịch" tín hiệu điện áp từ vi điều khiển sang tín hiệu vi sai trên bus CAN.
    - Mạch nguồn nhận nguồn điện dùng để chạy đa số các hệ thống khác trong ô tô, thường là 12V từ ắc quy. Mạch này phải giảm áp xuống 3.3V để có thể nuôi cho 2 mạch được nhắc đến ở trên.

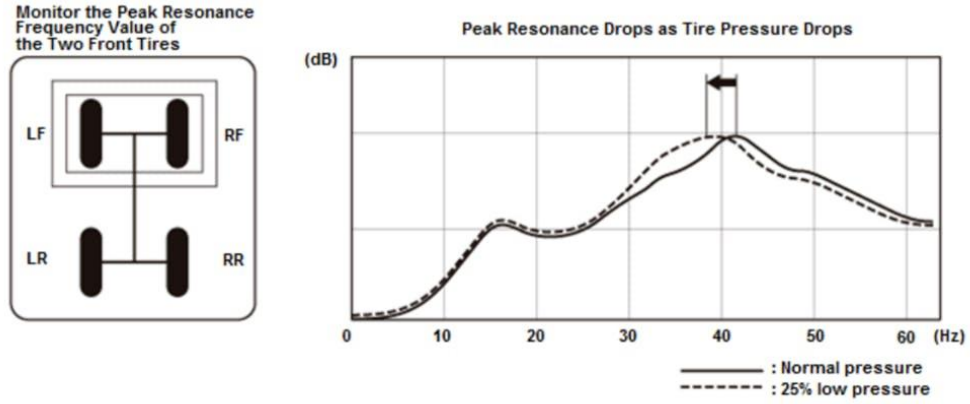
1.1.2 TPMS gián tiếp

TPMS gián tiếp không có các cảm biến đặt bên trong lốp xe để đo trực tiếp áp suất lốp xe. TPMS gián tiếp sử dụng các đầu vào khác, sau đó tính toán để xem rằng lốp xe đó có đủ căng hay không[2].

Ở các TPMS gián tiếp trong các dòng xe cũ, TPMS sẽ theo dõi tốc độ của bánh xe bằng cảm biến tốc độ được đặt trong mỗi bánh xe thuộc hệ thống ABS (Anti-lock Braking System). Khi áp suất trong lốp xe giảm, hình dạng và độ phồng của lốp thay đổi, điều này làm cho chu vi của lốp giảm, nghĩa là mỗi vòng quay của bánh xe bao phủ ít khoảng cách hơn so với lốp ở trạng thái bình thường. Vì vậy để xe di chuyển cùng một khoảng cách, bánh xe có áp suất thấp sẽ quay nhanh hơn so với các bánh xe có áp suất bình thường. TPMS gián tiếp sẽ từ các dữ liệu tốc độ bánh xe từ hệ thống ABS sẽ nhận ra nếu có bánh có tốc quay cao hơn so với các bánh xe còn lại sẽ thông báo rằng có bánh xe đang non, nhưng không chỉ rõ là bánh xe nào đang non như TPMS trực tiếp.

Ở các TPMS gián tiếp hiện đại thì chúng lấy rất nhiều đầu vào khác nhau như cảm biến tốc độ bánh xe, cảm biến gia tốc, áp suất phanh, góc lái, nhiệt độ bên ngoài,… để phán đoán áp suất các lốp xe. Các TPMS hiện đại sử dụng 2 cách tiếp cận chính để phán đoán.

- Cách tiếp cận đầu tiên được lấy từ các TPMS gián tiếp cũ là sử dụng cảm biến tốc độ bánh xe để phán đoán tình trạng lốp xe. Tuy nhiên, ở các TPMS hiện đại thì nó sẽ so sánh chéo các dữ liệu đó với các bánh xe khác, ví dụ như so sánh 2 bánh xe trước với nhau, 2 bánh sau với nhau hay 2 bánh bên trái với nhau,… Khi so sánh kiểu này, ta có thể kết luận rằng sẽ có bánh xe quay nhanh hơn bánh xe còn lại, kết hợp các phép so sánh lại với nhau, hệ thống có thể đưa ra kết luận rằng bánh xe nào đang non.
- Cách tiếp cận thứ hai sử dụng một thông số là tần số cộng hưởng của lốp xe. Khi lốp quay, nó tạo ra các dao động với một tần số nhất định, gọi là tần số cộng hưởng. Tần số này phụ thuộc vào áp suất lốp, kích thước, và điều kiện vận hành. Lốp có áp suất bình thường sẽ tạo ra một tần số dao động cố định, gọi là tần số cộng hưởng đỉnh khi xe di chuyển. Khi lốp mất áp suất, cấu trúc lốp thay đổi, làm giảm độ cứng, dẫn đến tần số cộng hưởng của lốp giảm. TPMS sẽ phân tích phổ tín hiệu từ dữ liệu của cảm biến tốc độ bánh xe để xác định tần số cộng hưởng đỉnh của từng lốp. Nếu trong quá trình đo, TPMS thấy tần số giảm dưới một ngưỡng nhất định, hệ thống sẽ xác nhận lốp xe đã bị non.



Hình 1-3: So sánh phổ tín hiệu tần số cộng hưởng giữa lốp xe bình thường và lốp xe non

Bảng 1-1 sau đây sẽ so sánh các tiêu chí quan trọng của 2 loại TPMS.

Tiêu chí	TPMS trực tiếp	TPMS gián tiếp
Nguyên lý hoạt động	Đo trực tiếp áp suất không khí bên trong lốp xe rồi gửi không dây đến	Lấy các thông số đầu vào khác để chuẩn đoán tình trạng lốp xe

	ECU	
Độ chính xác	Rất chính xác, cung cấp áp suất cụ thể của lốp xe	Chính xác nhưng thấp hơn TPMS trực tiếp
Khả năng bảo trì	Khó vì hệ thống cảm biến lấy từ bên trong lốp xe	Dễ vì chỉ có bộ ECU gắn trong thân xe
Khả năng phát hiện lốp xe sự cố	Phát hiện ngay lập tức khi áp suất giảm	Có thể gặp khó nếu cả 4 bánh đều giảm áp suất đều nhau
Tương thích	Với gần như tất cả các dòng xe	Chỉ các dòng xe có hệ thống ABS
Chi phí	Cao vì có các cảm biến chuyên dụng	Thấp hơn vì sử dụng cảm biến tốc độ bánh xe có sẵn trong ABS

Bảng 1-1: So sánh TPMS trực tiếp và TPMS gián tiếp

1.2 Nhiệm vụ đề tài

Với đề tài TPMS của môn Đồ án 2 (Kỹ thuật Điện tử - Viễn thông), hệ thống TPMS mà em thực hiện sẽ mô phỏng lại một **TPMS trực tiếp** ngoài thị trường, bao gồm việc:

- Đo áp suất không khi bên trong một môi trường giả lập lốp xe ô tô, cụ thể là bên trong ống tiêm.
- Giao tiếp không dây sử dụng giao tiếp BLE.
- Giao tiếp dữ liệu giữa các ECU sử dụng CAN.
- Xuất các thông tin đã qua xử lý ra màn hình OLED SSD1306 0.96in.

Bên cạnh hai thành phần là hệ thống cảm biến và ECU TPMS thì đề tài TPMS sẽ thiết kế thêm ECU màn hình, giúp có thể hiển thị những thông tin lấy được từ hệ thống cảm biến. Vậy đề tài TPMS sẽ phải thiết kế 3 mạch khác nhau, bao gồm mạch hệ thống cảm biến, mạch ECU TPMS và mạch ECU màn hình.

Đề tài TPMS sẽ sử dụng nguồn 12V để nuôi cho cả hệ thống, giống với một TPMS ngoài trường. Tuy nhiên vì giá cả cũng như là kích thước của một bình ắc quy không quá phù hợp với mô hình của một đồ án chuyên ngành nên sẽ đề tài sẽ sử dụng nguồn 12V – 2A từ adapter AC – DC.

1.3 Phạm vi đề tài

Với phạm vi của môn Đồ án chuyên ngành thì rất khó để có thể tạo ra một TPMS có thể chạy trong môi trường thực tế hay có những tính năng cao cấp. Đề tài TPMS chỉ mô phỏng các hoạt động chính của một TPMS ngoài thị trường đã nêu ở phần 1.2. Đề tài TPMS sẽ chỉ hoạt động trong phạm vi sau:

- Đề tài chỉ mô phỏng trong áp suất lốp xe giả lập, là áp suất từ piston trong ống tiêm.
- Đề tài sẽ không sử dụng bus CAN quá dài như trong xe ô tô, sẽ chỉ sử dụng cặp dây khoảng 15 – 20cm.
- Đề tài sẽ chỉ sử dụng module màn hình OLED 0.96in thay vì sử dụng một màn hình có hệ điều hành trước mặt tài xế.
- Đề tài sử dụng nguồn 12V từ adapter AC – DC thay vì sử dụng ắc quy.
- Đề tài sẽ không có những tính năng nâng cao cho TPMS mặc dù những tính năng này là cần thiết đối với một hệ thống TPMS ngoài thị trường như tính năng tiết kiệm năng lượng ở hệ thống cảm biến, tính năng bảo vệ quá nhiệt hay khả năng chịu được áp suất cao và chịu được môi trường quay nhanh như trong bánh xe của hệ thống cảm biến,…

1.4 Ứng dụng của đề tài

Đề tài TPMS nếu có thể được phát triển thêm về thành phần bao phủ hệ thống cảm biến giúp cho nó có thể gắn vào vòi lốp xe thì có thể dùng để quan sát áp suất lốp của các phương tiện có cấu trúc không quá phức tạp như xe đạp, xe đạp điện, xe máy,… Ngoài ra, TPMS có thể dùng để ứng dụng cho các ứng dụng ngoài lề như quan sát áp suất không khí của môi trường xung quanh hay là một phần trong một trạm thời tiết.

2. CƠ SỞ LÝ THUYẾT

2.1 Control Area Network

2.1.1 Giới thiệu

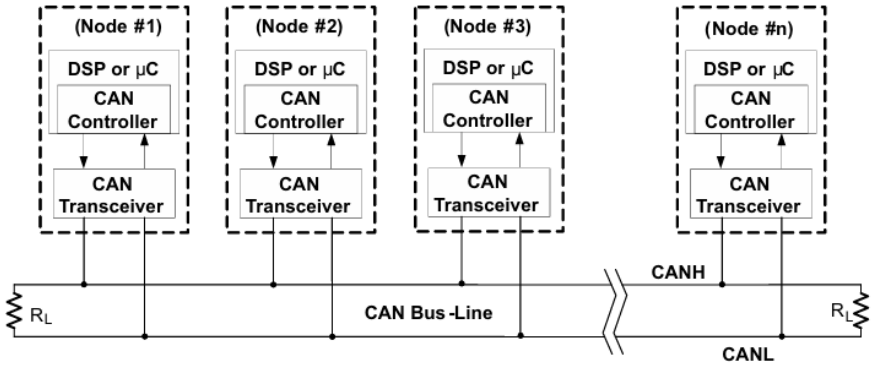
Giao thức **CAN (Controller Area Network)** là một giao thức truyền thông được sử dụng rộng rãi trong các hệ thống nhúng, đặc biệt là trong lĩnh vực ô tô và các ứng dụng công nghiệp. CAN cho phép các vi điều khiển và các thiết bị khác nhau giao tiếp với nhau mà không cần có máy tính chủ.

Trước khi có CAN, các hệ thống điều khiển trong xe (như hệ thống động cơ, phanh, và các hệ thống điều khiển khác) thường sử dụng các giao tiếp song song hoặc nối tiếp đơn giản. Điều này gây ra sự phức tạp khi phải kết nối nhiều hệ thống với nhau, do cần nhiều dây dẫn, chi phí cao, và nguy cơ xảy ra lỗi trong hệ thống lớn. CAN được phát triển để giải quyết các vấn đề này bằng cách:

- Cho phép các ECU giao tiếp với nhau trên một bus truyền thông chung.
- Giảm số lượng dây dẫn, giúp tiết kiệm chi phí và tăng độ tin cậy của hệ thống.
- Không yêu cầu master để điều phối các thiết bị. Các thiết bị trên bus CAN có thể truyền dữ liệu bất cứ lúc nào, với cơ chế phân quyền tự động để tránh xung đột dữ liệu.
- Độ tin cậy cao, đảm bảo việc phát hiện lỗi tự động thông qua cơ chế kiểm tra và sửa lỗi. Điều này cực kỳ quan trọng trong các hệ thống ô tô yêu cầu an toàn cao.

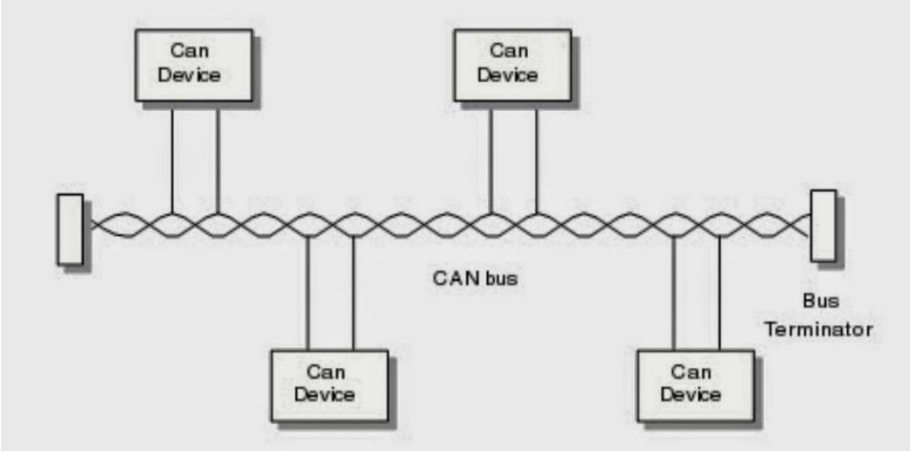
2.1.2 Kiến trúc

Những thiết bị giao tiếp trong CAN gọi là các **node**. Các node được kết nối song song vào cùng một đường bus gọi là **CAN bus** (xem Hình 2-1), bao gồm 2 dây **CANH (CAN High) và CANL (CAN Low)**. Các tín hiệu truyền qua bus CAN là tín hiệu vi sai, nghĩa là thông tin được mã hóa dựa trên sự chênh lệch điện áp giữa hai dây CANH và CANL.



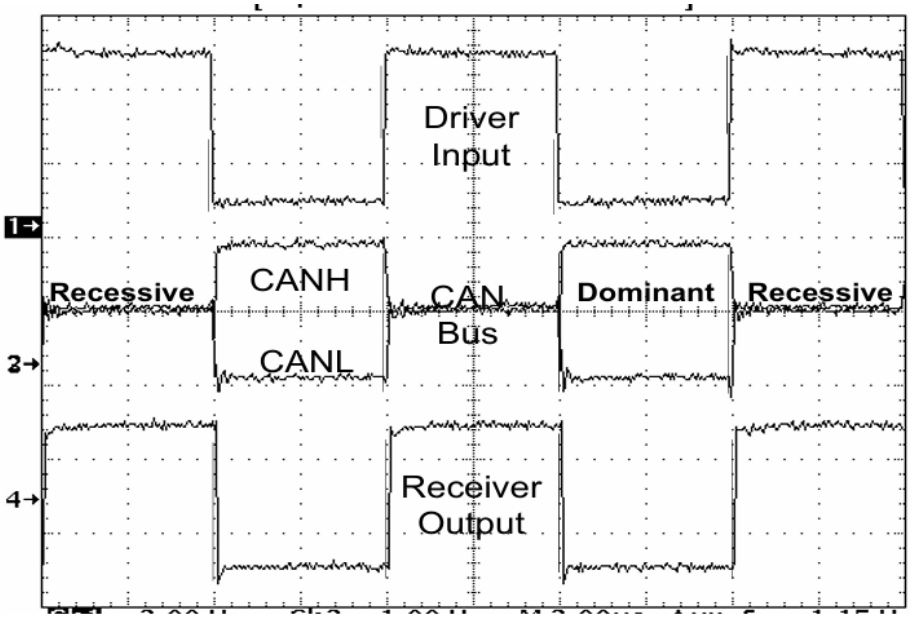
Hình 2-1: Sơ đồ kết nối các node và CAN bus

Hai dây tín hiệu này được xoắn lại tạo thành đường dây xoắn đôi bởi vì khi các dây được xoắn lại, mỗi đoạn của cặp dây sẽ nhận nhiễu với cường độ khác nhau và theo các hướng ngược nhau, làm triệt tiêu phần lớn nhiễu điện từ và giảm nhiễu xuyên âm bằng cách phân tán nhiễu xuyên âm ra khắp chiều dài của cáp. Ở hai đầu cuối của CAN bus, ta mắc thêm điện trở 120Ω (còn được gọi là điện trở kết cuối) để ngăn chặn hiện tượng phản xạ tín hiệu. Nếu không có điện trở này, tín hiệu có thể bị phản xạ lại từ các đầu cuối mở, gây ra nhiễu và làm hỏng dữ liệu.



Hình 2-2: Hình dạng thực tế của CAN Bus

Tuỳ vào các điều kiện khác nhau như tốc độ truyền, loại CAN transceiver hay bus CAN thì giá trị tín hiệu vi sai trên CAN bus sẽ khác nhau. Các ứng dụng CAN ngoài thị trường sẽ cố gắng sử dụng chuẩn ISO 11898 để định nghĩa tín hiệu tren CAN bus. Theo chuẩn ISO 11898, khi điện áp trên CANH lớn hơn điện áp trên CANL gần bằng 2.5V thì tín hiệu đang được truyền trên CAN bus sẽ tương ứng với bit 0 trong vi điều khiển, tín hiệu này được gọi là **"dominant"**[3]. Khi điện áp trên CANH gần bằng (thường là lớn hơn 0.5V) điện áp trên CANL thì tín hiệu đang được truyền trên CAN bus sẽ tương ứng với bit 0 trong vi điều khiển, tín hiệu này được gọi là **"recessive"**[3] (xem Hình 2-3).



Hình 2-3: Trạng thái bit dominat và recessive trong CAN

Các node nếu muốn gửi và nhận dữ liệu CAN thì bên trong các node cần có những thành phần sau (xem Hình 2-1):

- **CAN controller** là thành phần chính trong node CAN, có nhiệm vụ xử lý toàn bộ giao tiếp CAN như:
  - Gửi và nhận thông điệp CAN.
  - Điều khiển truy cập vào bus CAN (arbitration).
  - Phát hiện và xử lý các lỗi truyền thông CAN.
  - Kiểm soát việc truyền lại thông điệp khi gặp lỗi.
  - Cung cấp giao diện giữa các vi điều khiển và bus CAN.
- **CAN transceiver** như một "người phiên dịch" viên giúp:
  - Chuyển đổi tín hiệu số từ bộ điều khiển CAN thành tín hiệu điện áp dạng vi sai để gửi lên bus CAN và ngược lại
  - Đảm bảo tín hiệu truyền và nhận trên bus CAN có độ chính xác và tốc độ cao.
- **Vi điều khiển** là thành phần trung tâm điều khiển hoạt động của node CAN như:
  - Đọc và xử lý thông điệp CAN.
  - Tạo ra thông điệp CAN để truyền đi.
  - Quản lý các khung dữ liệu, bit arbitration và quá trình xử lý lỗi.
  - Điều khiển hành vi của node (ví dụ: bật/tắt node, reset node khi gặp lỗi bus-off).

2.1.3 Đặc điểm

CAN có những đặc điểm giao tiếp nổi bật giúp nó trở thành một giao thức truyền thông đáng tin cậy và hiệu quả trong các hệ thống nhúng.

Thứ nhất, CAN không tuân theo kiến trúc master-slave. Tất cả node đều có quyền bình đẳng trong việc truyền dữ liệu mà không cần phải có thiết bị master điều khiển. Điều này cho phép mạng hoạt động linh hoạt hơn, khi bất kỳ node nào cũng có thể truyền hoặc nhận thông tin bất cứ lúc nào.

Thứ hai, khi một node gửi thông điệp, thông điệp đó sẽ được phát sóng đến tất cả các node khác trên bus. Tuy nhiên, không phải tất cả các node đều xử lý thông điệp này. Mỗi node sẽ sử dụng bộ lọc để kiểm tra xem thông điệp có phù hợp với mình hay không (xem chi tiết ở mục **2.1.6**).

Thứ ba, một đặc điểm quan trọng của mạng CAN là khả năng giải quyết tranh chấp quyền gửi dữ liệu giữa các node. Nếu có nhiều node cùng muốn gửi dữ liệu lên bus cùng một lúc, CAN sẽ sử dụng cơ chế phân quyền để quản lý dữ liệu truyền trên bus (xem chi tiết ở mục **2.1.5**).

Thứ tư, mặc dù chỉ sử dụng một bus với hai dây tín hiệu, mạng CAN vẫn cho phép các node vừa gửi vừa nhận dữ liệu đồng thời. Điều này giúp mạng CAN hoạt động hiệu quả và không bị nghẽn khi có nhiều thiết bị cùng giao tiếp.

Thứ năm, một tính năng quan trọng khác của mạng CAN là khả năng tự động phát hiện và xử lý lỗi. Nếu một node phát hiện ra lỗi trong quá trình truyền hoặc nhận dữ liệu, node đó sẽ gửi một Error Frame để thông báo cho các node khác rằng dữ liệu bị lỗi. Sau đó, thông điệp sẽ được truyền lại.

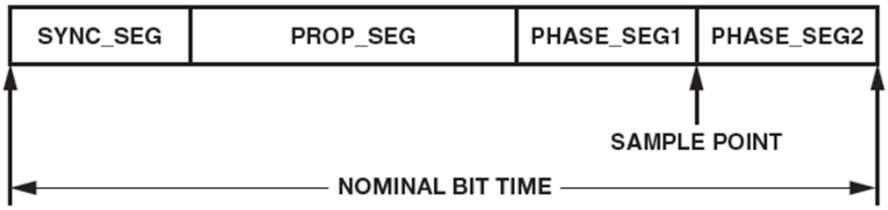
2.1.4 Tốc độ truyền dữ liệu

Giao thức CAN được thiết kế để hoạt động hiệu quả trong các hệ thống nhúng với khả năng truyền thông dữ liệu tin cậy và độ trễ thấp. Hai yếu tố quan trọng ảnh hưởng đến hiệu suất của mạng CAN là tốc độ truyền và chiều dài của CAN bus.

Tốc độ truyền (baud rate) là tốc độ truyền dữ liệu trên bus CAN, thường được đo bằng kbps hoặc Mbps. Baud rate quyết định tốc độ truyền thông giữa các thiết bị trên mạng và phụ thuộc vào khả năng xử lý của hệ thống cũng như chiều dài của bus. Tuỳ vào phiên bản CAN mà baud rate tối đa nó có thể hỗ trợ sẽ khác nhau (xem chi tiết ở mục **2.1.9**).

Baud rate bị ảnh hưởng mật thiết bởi chiều dài của CAN bus. Khi khoảng cách của 2 node gần nhau, dữ liệu truyền từ node này qua node sẽ nhanh bởi thời gian lan truyền trên bus lúc này là ngắn, đồng thời thời gian dữ liệu tiếp xúc với các nhiễu cũng ngắn. Điều này có nghĩa là khi 2 node muốn truyền dữ liệu với tốc độ cao, hệ thống phải chấp nhận giảm chiều dài của bus để đảm bảo tín hiệu truyền đủ nhanh và chính xác.

Để có thể cài đặt được tốc độ truyền mong muốn, ta sẽ chỉnh sửa thời gian mà 1 bit sẽ tồn tại trên bus. Thời gian tồn tại của 1 bit được chia thành 4 đoạn (xem Hình 2-4). Đơn vị của từng phân đoạn là time quanta (TQ). TQ là đơn vị thời gian nhỏ nhất mà một bit trong giao thức CAN được chia thành. TQ sẽ phụ thuộc vào tần số xung clock mà MCU cấp cho CAN controller.



Hình 2-4: Các phân đoạn trong thời gian tồn tại của một bit

- **Sync Segment** là đoạn đầu tiên của mỗi bit và có độ dài cố định là 1 TQ. Đoạn này giúp đồng bộ hóa tất cả các node trên bus. Nó đảm bảo rằng tất cả các node đều nhận biết sự bắt đầu của một bit tại cùng một thời điểm. Node nhận sẽ phát hiện ra cạnh thay đổi (cạnh lên hoặc cạnh xuống) của tín hiệu CAN tại đoạn này để điều chỉnh thời gian của chính nó, đảm bảo đồng bộ với các node khác trên bus.
- **Propagation Segment** là đoạn bù đắp thời gian cần thiết để tín hiệu lan truyền qua bus CAN từ node gửi đến node nhận. Tín hiệu cần thời gian để di chuyển từ một node đến một node khác, và thời gian này phụ thuộc vào chiều dài của bus và tốc độ truyền. Propagation Segment được cấu hình sao cho nó bao gồm độ trễ lan truyền và thời gian trễ xử lý của cả phần cứng CAN.
- **Phase Segment 1 (PS1) và Phase Segment 2 (PS2)** là 2 phân đoạn này được sử dụng để đồng bộ hóa tín hiệu và bù đắp cho các sai lệch về thời gian hoặc độ trễ nhỏ trong quá trình truyền.
    - PS1 là đoạn thời gian trước điểm lấy mẫu. Đoạn này cho phép node điều chỉnh thời gian đọc tín hiệu để đồng bộ với tín hiệu thực tế trên bus.

- PS2 là đoạn thời gian sau điểm lấy mẫu. Nó có thể được kéo dài hoặc thu ngắn để bù trừ sự sai lệch nhỏ giữa các node, giữ cho tất cả node đồng bộ với nhau. Đây là đoạn để kết thúc 1 bit.
- Cả PS1 và PS2 đều có thể được điều chỉnh linh hoạt tùy thuộc vào sự thay đổi thời gian cần thiết để đảm bảo việc lấy mẫu tín hiệu một cách chính xác.

Điểm lấy mẫu là thời điểm mà tín hiệu trên bus CAN được đọc để xác định giá trị của một bit (dominant hoặc recessive). Mẫu thường được lấy ở vị trí cuối mỗi bit để đảm bảo tín hiệu đã ổn định. Vị trí của điểm mẫu được tính toán dựa trên tỷ lệ phần trăm trong một khoảng thời gian bit. Điểm mẫu lý tưởng thường nằm ở khoảng 75% đến 90% thời gian của một bit.

Vậy, tổng thời gian của một bit trong giao thức CAN là tổng của các đoạn thời gian đơn vị là TQ:

$$\text{Bit Time} = \text{Sync Segment} + \text{Propagation Segment} + \text{PS1} + \text{PS2}$$

Và baud rate được tính như sau:

$$\text{Tốc độ baud} = 1/\text{Bit Time (bps)}$$

2.1.5 Cơ chế phân quyền

Phân quyền là một cơ chế quan trọng của giao thức CAN, cho phép nhiều node trên bus có thể cố gắng truyền thông điệp đồng thời mà không gây xung đột hoặc mất dữ liệu. Cơ chế này đảm bảo rằng node có mức độ ưu tiên cao nhất sẽ chiếm quyền truy cập bus, trong khi các node có ưu tiên thấp hơn sẽ đợi lượt tiếp theo. Quá trình này giúp cho không có bất kỳ dữ liệu nào bị mất khi có xung đột về quyền gửi.

Trong CAN, mỗi thông điệp CAN đều có một ID định danh, và giá trị của ID này quyết định mức độ ưu tiên khi có nhiều node cố gắng gửi dữ liệu cùng một lúc. Khi nhiều node muốn truyền dữ liệu, chúng đều bắt đầu gửi thông điệp của mình lên bus. Tín hiệu được gửi đồng thời và mỗi node sẽ kiểm tra từng bit của dữ liệu trên bus. CAN sử dụng phép logic AND để quyết định node nào được ưu tiên. Mỗi bit trong ID sẽ được truyền từng cái một (MSB truyền trước). Nếu một node gửi bit recessive nhưng nhận thấy trên bus có bit dominant, nghĩa là có một node khác có ưu tiên cao hơn đang chiếm quyền truyền dữ liệu. Lúc này, node này sẽ ngừng truyền và chuyển sang chế độ nghe. Node có ID thấp hơn sẽ tiếp tục quá trình truyền cho đến khi toàn bộ ID được gửi đi, trong khi các node khác ngừng gửi

và chuyển sang chế độ chờ. Các node không thắng quá trình phân quyền sẽ không bị mất dữ liệu mà chỉ đơn giản là đợi lượt tiếp theo để cố gắng truyền lại thông điệp của mình.

	Start bit	ID bits											The rest of the frame
		10	9	8	7	6	5	4	3	2	1	0	
Node 15	0	0	0	0	0	0	0	0	1	1	1	1	
Node 16	0	0	0	0	0	0	0	1	Stopped Transmitting				
CAN data	0	0	0	0	0	0	0	0	1	1	1	1	

Hình 2-5: Cơ chế phân quyền góc nhìn từng bit

2.1.6 Cơ chế bộ lọc

Trong CAN, các node có thể nhận rất nhiều thông điệp, nhưng không phải tất cả thông điệp đều liên quan đến tất cả các node. Bộ lọc CAN cho phép các node lựa chọn và chỉ nhận những thông điệp cần thiết, dựa trên ID của thông điệp hoặc các tiêu chí khác, giúp giảm tải cho vi điều khiển, vì nó chỉ xử lý những dữ liệu mà nó cần.

Bộ lọc CAN hoạt động dựa trên hai thành phần chính:

- Mask là một dãy bit mà các bit có giá trị 1 sẽ được kiểm tra, còn các bit có giá trị 0 sẽ bị bỏ qua. Mask quy định những bit nào trong ID của thông điệp cần được kiểm tra, giúp xác định phạm vi ID mà node quan tâm.

- Filter là giá trị mà các bit trong ID của thông điệp đã được chọn bởi mask phải khớp với. Các bit được phép kiểm tra thông qua mask sẽ được so sánh với filter. Nếu các bit đã được chọn bởi mask của ID thông điệp trùng khớp với giá trị của filter, thông điệp sẽ được chấp nhận. Nếu không trùng khớp, thông điệp sẽ bị bỏ qua, node sẽ không xử lý nó. Filter được dùng để so sánh các bit của ID thông điệp với giá trị quy định trong bộ lọc.

Ví dụ: Giả sử trong một hệ thống CAN, chúng ta có một node cần nhận thông điệp có ID trong khoảng từ 0x100 đến 0x1FF. Điều này có nghĩa là node chỉ quan tâm đến các thông điệp có giá trị từ 0x100 đến 0x1FF, và không quan tâm đến các thông điệp có ID nằm ngoài phạm vi này. Để đạt được điều này, chúng ta có thể cấu hình bộ lọc CAN như sau:

- Mask: 0x700 (111 0000 0000). Có nghĩa là chỉ có 3 bit đầu tiên của ID thông điệp sẽ được so sánh với filter. Các bit khác (bit từ 8 trở xuống) sẽ không được kiểm tra.

- Filter: 0x100 (001 0000 0000). Có nghĩa là node sẽ chỉ chấp nhận những thông điệp có 3 bit đầu tiên là 001, tức là thông điệp có ID nằm trong khoảng từ 0x100 đến 0x1FF.

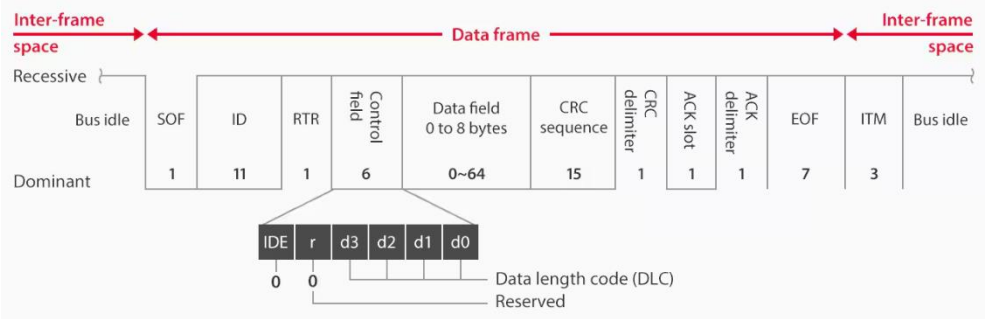
Với cấu hình này, node sẽ chỉ nhận những thông điệp có ID từ 0x100 đến 0x1FF, giúp lọc bỏ các thông điệp không liên quan và giảm tải cho vi điều khiển.

2.1.7 Các khung dữ liệu trong CAN

Khi giao tiếp với nhau, các node trong CAN sẽ truyền dữ liệu theo các khung truyền đã được định nghĩa trước bởi chuẩn ISO 11898. Có 4 loại khung truyền trong CAN để thực hiện 4 chức năng khác nhau, bao gồm Data Frame, Remote Frame, Error Frame và Overload Frame.

2.1.7.1 Data Frame

Data Frame là khung phổ biến nhất được sử dụng trong giao thức CAN. Nó được sử dụng để truyền dữ liệu thực tế giữa các node trên mạng CAN. Data Frame bao gồm các thông tin về địa chỉ của node gửi và nhận, kích thước dữ liệu, và chính dữ liệu cần truyền. Frame này giúp đảm bảo rằng dữ liệu sẽ được truyền đúng cách và bảo vệ khỏi lỗi bằng cách sử dụng mã kiểm tra CRC. Data Frame được chia làm nhiều trường khác nhau (xem Hình 2-6).



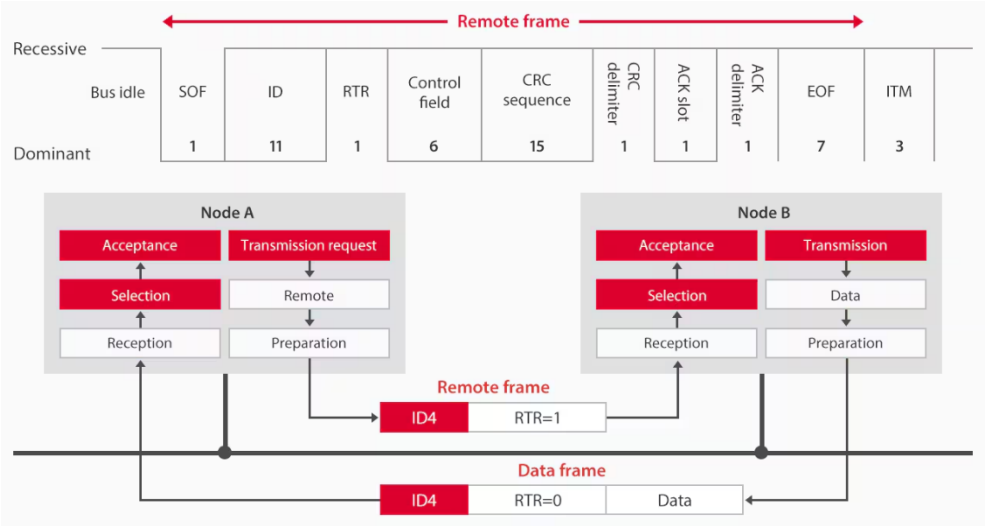
Hình 2-6: Cấu trúc Data Frame

- **SOF (Start Of Frame)**: 1 bit dominant để báo hiệu bắt đầu của khung truyền.
- **Arbitration Field**: là nhân tố sẽ được so sánh trong cơ chế phân quyền.
  - Chứa ID của thông điệp cần gửi, có thể là 11-bit (khung tiêu chuẩn) hoặc 29-bit (khung mở rộng), giúp phân biệt giữa các node.
  - Trường này cũng chứa bit **RTR** để xác định loại khung là Data Frame (dominant) hay Remote Frame (recessive).

- **Control Field**: Chứa bit **DLC,** chỉ ra số byte dữ liệu sẽ truyền đi, từ 0 đến 8 byte và bit **IDE** để xác định kiểu khung là khung tiêu chuẩn hay khung mở rộng.
- **Data Field**: Chứa dữ liệu thực tế cần truyền. Độ dài từ 0 đến 8 byte tùy thuộc vào giá trị của DLC.
- **CRC Field**: Dùng để phát hiện lỗi trong quá trình truyền thông qua mạng.
- **ACK Field**: Trường này sẽ để 1 bit ACK Slot có giá trị là recessive để node nhận dữ liệu kéo xuống dominant khi nhận thành công dữ liệu từ node gửi.
- **EOF (End Of Frame)**: 7 bit recessive kết thúc khung.

2.1.7.2 Remote Frame

Remote Frame được sử dụng khi một node trên CAN yêu cầu dữ liệu từ một node khác. Thay vì chứa dữ liệu thực, Remote Frame chứa ID của node cần yêu cầu, cùng với bit điều khiển **RTR (Remote Transmission Request)**. Remote Frame thường được sử dụng trong các hệ thống mà một node có thể yêu cầu thông tin từ một node khác mà không có dữ liệu nào được truyền ngay lập tức. Node nhận yêu cầu sẽ trả lời bằng một Data Frame với ID giống với ID của Remote Frame trước đó và chứa dữ liệu cần thiết.



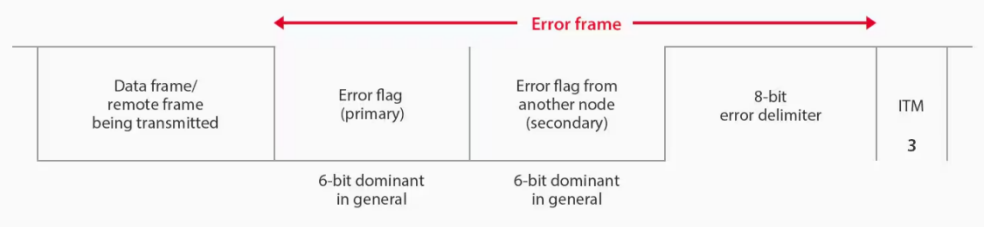
Hình 2-7: Cấu trúc và hoạt động của Remote Frame

Remote Frame có cấu trúc gần giống với Data Frame và chức năng của các trường cũng giống nhau, chỉ khác tại bit RTR của Remote Frame sẽ là recessive và Remote Frame không có Data Field, tức là Remote Frame không chứa dữ liệu cần truyền mà chỉ dùng để yêu cầu dữ liệu từ node khác.

2.1.7.3 Error Frame

Error Frame được sử dụng khi một node phát hiện ra lỗi trong quá trình truyền dữ liệu. Nó được gửi để thông báo cho các node khác rằng có lỗi đã xảy ra trên bus. Bất kỳ node nào phát hiện ra lỗi đều có thể gửi Error Frame ra bus CAN. Error Frame có vai trò rất quan trọng trong việc duy trì độ tin cậy của mạng CAN. Khi một lỗi xảy ra, Error Frame sẽ báo hiệu để các node khác biết rằng thông điệp vừa được truyền không hợp lệ để các node khác có thể hành động để giúp node lỗi có thể sửa.

Error Frame gồm hai phần: Error Flag và Error Delimiter. Error Flag là chuỗi từ 6 đến 12 bit dominant, báo hiệu lỗi. Error Delimiter là chuỗi 8 bit recessive, kết thúc Error Frame. Tuỳ thuộc vào trạng thái lỗi của node mà Error Flag được gửi ra bus sẽ khác (chi tiết ở mục 2.1.8).

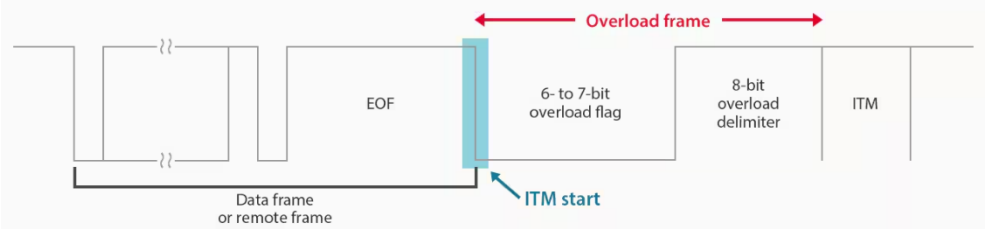


Hình 2-8: Cấu trúc của Error Frame

2.1.7.4 Overload Frame

Overload Frame là một loại khung đặc biệt trong giao thức CAN được sử dụng để trì hoãn việc truyền dữ liệu khi một node trong CAN cần thêm thời gian để xử lý. Khung này không chứa dữ liệu, mà chỉ báo hiệu rằng một node đang quá tải và cần thời gian trước khi tiếp tục giao tiếp. Mục tiêu của Overload Frame là ngăn không cho các khung khác được truyền quá nhanh, giúp node bị quá tải có đủ thời gian để xử lý các khung trước đó. Overload Frame không phải do người dùng phát ra, mà được tự động phát ra bởi phần cứng CAN khi một trong các điều kiện sau xảy ra:

- Node không thể xử lý tiếp dữ liệu do buffer đã đầy hoặc cần thêm thời gian xử lý dữ liệu.
- Node không thể nhận khung mới do có quá trình xử lý nội bộ cần hoàn thành trước.



Hình 2-9: Cấu trúc của Overload Frame

2.1.8 Các loại lỗi và cơ chế tự phát hiện lỗi

Trong CAN, cơ chế phát hiện là một tính năng quan trọng để đảm bảo độ tin cậy và tính ổn định của quá trình truyền dữ liệu. CAN sử dụng nhiều cơ chế để phát hiện lỗi, giúp duy trì tính ổn định và tin cậy của dữ liệu truyền tải trên bus. Các cơ chế này bao gồm:

- Mỗi node gửi sẽ tự lắng nghe dữ liệu mà nó vừa gửi để đảm bảo rằng dữ liệu đó đã được truyền đúng cách. Nếu có sự khác biệt giữa bit gửi đi và bit nhận lại, node sẽ phát hiện **bit error**.
- Mỗi thông điệp CAN chứa một giá trị CRC được tính toán dựa trên dữ liệu. Node nhận sẽ tính toán lại giá trị CRC và so sánh với CRC của thông điệp để phát hiện **CRC error**.
- Khung dữ liệu nhận được khung không tuân theo định dạng đúng của giao thức CAN, node nhận sẽ phát hiện **form error**.
- Node gửi sẽ kiểm tra xem có bất kỳ node nào trên bus gửi bit ACK để xác nhận rằng dữ liệu đã được nhận thành công. Nếu không, **ACK error** sẽ được phát hiện.
- Trong CAN, sau mỗi chuỗi 5 bit giống nhau liên tiếp, một bit ngược giá trị phải được thêm vào để đảm bảo tính đồng bộ và tránh nhiễu tín hiệu. Nếu quy tắc này bị vi phạm, **stuff error** sẽ xảy ra.

Node nào phát hiện được 1 trong 5 lỗi trên thì sẽ tự động phát ra 1 Error Frame để thông báo cho các node khác rằng đang có lỗi và các node khác nên hành động để giúp cho node phát hiện lỗi khởi động lại để sửa lỗi.

Khi phát ra Error Frame, có một bộ đếm lỗi ở mỗi node sẽ đếm lên mỗi khi node phát hiện ra lỗi. Dựa vào giá trị của bộ đếm lỗi này, CAN controller sẽ đưa node đó vào 3 trạng thái khác nhau:

- **Error Active:** Trong trạng thái Error Active, node vẫn có khả năng tham gia đầy đủ vào quá trình truyền thông và có thể phát hiện lỗi. Nếu node phát hiện lỗi, nó sẽ gửi

một Error Frame để thông báo cho các node khác trên bus rằng đã xảy ra lỗi. Ở trạng thái này, node sẽ phát ra một Error Frame với một Error Flag gồm 6 bit dominant liên tiếp.

- **Error Passive:** Nếu một node phát hiện quá nhiều lỗi, nó sẽ chuyển sang trạng thái Error Passive. Trong trạng thái Error Passive, node vẫn có thể nhận và gửi thông điệp nhưng sẽ hạn chế việc can thiệp vào quá trình truyền thông của các node khác. Nhưng nếu phát hiện lỗi, nó sẽ không gửi Error Frame mạnh mẽ như trong trạng thái Error Active mà chỉ gửi 1 Error Frame yếu hơn. Điều này giúp tránh gây gián đoạn lớn cho bus khi node gặp sự cố thường xuyên. Ở trạng thái này, node sẽ phát ra Error Flag với 12 bit (thay vì 6 bit trong Active Error Frame), bao gồm 6 bit dominant và 6 bit recessive.

- **Bus Off:** Khi một node gặp quá nhiều lỗi nghiêm trọng, nó sẽ chuyển sang trạng thái Bus Off. Trong trạng thái này, node sẽ hoàn toàn ngắt kết nối khỏi bus CAN và không thể tham gia vào quá trình truyền hay nhận dữ liệu. Node chỉ có thể được kết nối lại vào bus sau khi được sửa bởi người phát triển hoặc reset bởi phần mềm. Bus Off là trạng thái an toàn, ngăn chặn một node bị lỗi nặng gây ra sự cố nghiêm trọng cho toàn bộ hệ thống CAN.

2.1.9 Các phiên bản của CAN

Giao thức CAN đã phát triển qua nhiều phiên bản để đáp ứng nhu cầu ngày càng cao trong các ứng dụng nhúng, đặc biệt là trong ngành công nghiệp ô tô và tự động hóa. Các phiên bản mở rộng của CAN bao gồm CAN 2.0A, CAN 2.0B, và CAN FD. Mỗi phiên bản có những cải tiến để hỗ trợ các yêu cầu khác nhau về độ ưu tiên, dung lượng dữ liệu và tốc độ truyền tải.

2.1.9.1 CAN 2.0A

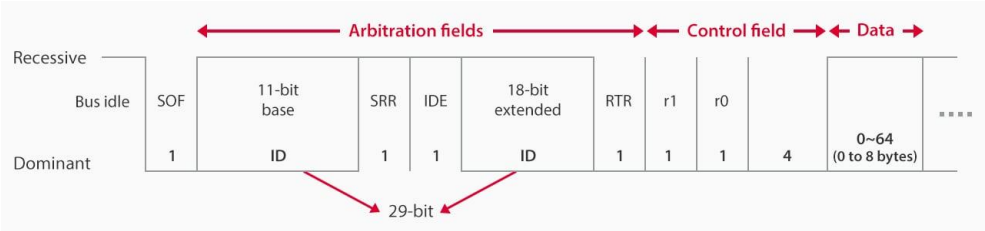
Đây là phiên bản tiêu chuẩn của CAN. Phiên bản CAN 2.0A sử dụng định dạng ID dài 11 bit. Đây là một trong những yếu tố quan trọng để xác định mức độ ưu tiên của thông điệp trong quá trình truyền dữ liệu. Với 11 bit, có thể biểu diễn 2^11 = 2048 ID khác nhau. Dữ liệu có thể truyền đi qua bus CAN với kích thước tối đa là 8 byte mỗi khung.

2.1.9.2 CAN 2.0B

Đây là phiên bản mở rộng so với CAN 2.0A. Phiên bản CAN 2.0B mở rộng định dạng ID từ 11 bit trong CAN 2.0A lên 29 bit. Với 29 bit, có thể biểu diễn 2^29 ID khác nhau, cho phép phân bổ số lượng ID lớn hơn và hỗ trợ các hệ thống phức tạp với nhiều node hơn. Tuy nhiên, giống như CAN 2.0A, CAN 2.0B cũng chỉ giới hạn khung dữ liệu tối đa là 8 byte.

CAN 2.0B vẫn tương thích ngược với CAN 2.0A, có nghĩa là các node sử dụng CAN 2.0B có thể hiểu được và giao tiếp với các node sử dụng CAN 2.0A. Các node CAN 2.0B có thể nhận diện giữa các khung dữ liệu tiêu chuẩn và khung dữ liệu mở rộng thông qua bit điều khiển IDE.

Khung dữ liệu CAN 2.0B cũng sẽ khác tại một số điểm nhất định. Trong Abitration Field thì sẽ có thêm 18 bit mở rộng, cùng với 11 bit ID tiêu chuẩn tạo thành 29 bit ID. Ngoài ra sẽ có thêm 1 bit **SRR** luôn có giá trị recessive, điều này khiến cho khung mở rộng sẽ có mức ưu tiên thấp hơn khung tiêu chuẩn (do tại vị trí của bit SRR trong khung tiêu chuẩn là bit RTR dominant khi là Data Frame).



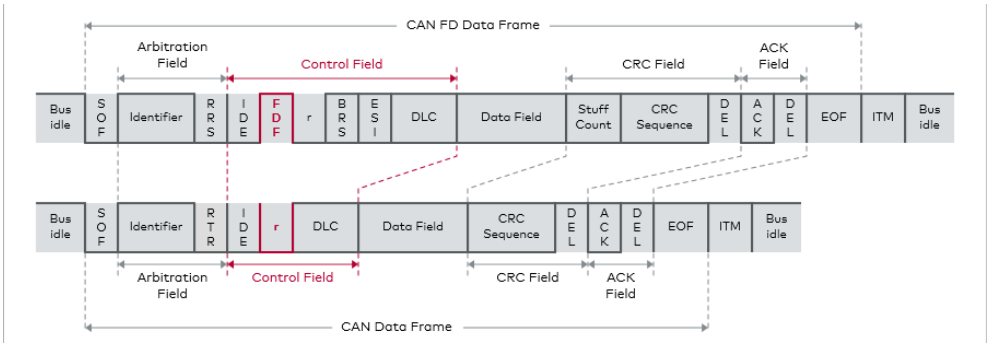
Hình 2-10: Cấu trúc Data Frame của CAN 2.0B

2.1.9.3 CAN FD

CAN FD là một phiên bản cải tiến của giao thức CAN tiêu chuẩn, được phát triển để giải quyết các hạn chế về tốc độ truyền dữ liệu và kích thước khung dữ liệu trong các phiên bản trước đó. CAN FD là viết tắt của Flexible Data-rate, tức là tốc độ dữ liệu linh hoạt, và có những cải tiến lớn so với CAN 2.0A và 2.0B.

CAN FD cho phép tốc độ truyền dữ liệu nhanh hơn nhiều so với CAN tiêu chuẩn, với tốc độ có thể lên tới 8 Mbps trong giai đoạn truyền dữ liệu. Trong CAN tiêu chuẩn, tốc độ truyền tối đa bị giới hạn ở 1 Mbps. Trong khi CAN tiêu chuẩn chỉ hỗ trợ tối đa 8 byte dữ liệu trong mỗi khung, CAN FD có thể truyền tới 64 byte dữ liệu trong một khung. Điều này giúp giảm số lượng khung cần thiết để truyền một lượng lớn dữ liệu, từ đó giảm độ trễ và tăng hiệu suất.

CAN FD sử dụng hai tốc độ truyền khác nhau trong cùng một khung: một tốc độ chậm hơn cho giai đoạn phân quyền, và một tốc độ nhanh hơn cho giai đoạn truyền dữ liệu. Điều này giúp đảm bảo tính tương thích và hiệu quả của hệ thống. CAN FD vẫn giữ được khả năng tương thích ngược với các phiên bản CAN cũ như CAN 2.0A và CAN 2.0B, giúp các hệ thống sử dụng cả CAN tiêu chuẩn và CAN FD có thể hoạt động song song.



Hình 2-11: Cấu trúc Data Frame của CAN FD so với CAN 2.0A

Data Frame của CAN FD sẽ phức tạp hơn so với 2 phiên bản CAN 2.0.

- Bit **FDF** ở trong Control Field sẽ chỉ định khung được truyền đi sẽ sử dụng các thông số của phiên bản CAN nào.
    - Nếu FDF là recessive, dữ liệu có thể được truyền đi nhanh hơn trong giai đoạn truyền dữ liệu (tuỳ vào giá trị của bit BRS) và có thể truyền lên đến 64 byte
    - Nếu FDF là dominant, dữ liệu được truyền đi sẽ giống như CAN 2.0.
- Bit **BRS** trong Control Field để chỉ định rằng giai đoạn truyền dữ liệu trong khung CAN FD có được truyền ở tốc độ cao hơn so với giai đoạn phân quyền hay không (nếu có thì BRS là recessive).
- Bit **ESI** trong Control Field để chỉ định trạng thái lỗi của node. Khi ESI là dominant, node đang ở trạng thái Error Active, ngược lại khi ESI là recessive thì node đang ở trạng thái Error Passive.
- **Stuff Count** trong CRC Field được dùng để đếm số lượng bit giống nhau liên tiếp để thực hiện cơ chế bit stuffing. Trong 2 phiên bản CAN 2.0 thì cơ chế bit stuffing được áp dụng lên cả frame, khiến cho có nhiều bit stuff hơn, khiến cho việc giải mã phức tạp hơn, nhất là ở Data Field. Trong CAN FD, cơ chế bit stuffing chỉ áp dụng trong CRC và phân quyền nên sẽ cần 1 bộ đếm các bit giống nhau liên tiếp để chèn stuff bit ngay sau trong trường CRC.

2.2 Bluetooth Low Energy

2.2.1 Giới thiệu BLE

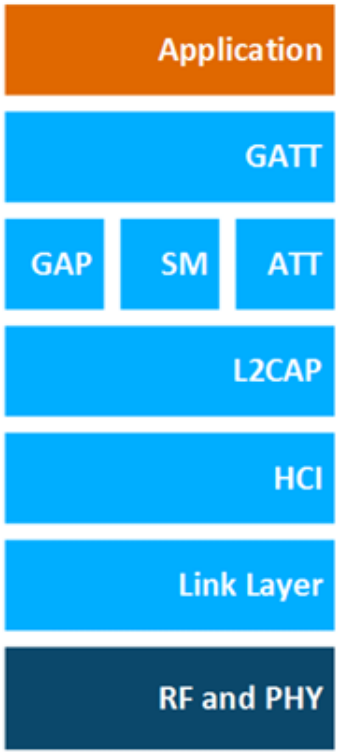
Từ phiên bản Bluetooth 4.0 trở đi, một phiên bản khác của Bluetooth được giới thiệu. Phiên bản này sử dụng các tính năng cơ bản của Bluetooth nhưng tốn ít năng lượng hơn. Phiên bản này là **Bluetooth Low Energy (BLE)**, tạm dịch là Bluetooth năng lượng thấp[4]. Giống như Bluetooth, BLE là một công nghệ không dây cho phép các thiết bị kết nối và trao đổi dữ liệu qua sóng radio, nhưng được tối ưu hóa cho các ứng dụng tiêu thụ năng lượng thấp và truyền dữ liệu ngắn.

Đặc điểm	BLE	Bluetooth
Mục tiêu	Tối ưu hóa cho các ứng dụng IoT cần tiêu thụ năng lượng thấp và truyền dữ liệu ngắn.	Tối ưu hóa cho việc truyền luồng dữ liệu chất lượng cao liên tục như âm thanh, video,…
Ứng dụng	Các ứng dụng IoT, thiết bị cảm biến, thiết bị đeo, thiết bị y tế, các ứng dụng cần thời lượng pin dài.	Truyền âm thanh không dây (tai nghe, loa), kết nối thiết bị ngoại vi (chuột, bàn phím).
Cách kết nối	Kết nối ngắn, không liên tục, cho phép truyền dữ liệu theo đợt ngắn.	Kết nối liên tục, tầm ngắn.
Mạng lưới	Mạng broadcast (một thiết bị gửi đến nhiều thiết bị) và mạng kết nối (một thiết bị kết nối với một thiết bị trung tâm).	Mạng kết nối point-to-point, tức là một thiết bị chỉ liên kết trực tiếp đến 1 thiết bị.
Tốc độ truyền dữ liệu	Thấp hơn so với BR/EDR trong Bluetooth.	Cao hơn.
Vai trò của các thiết bị	Hỗ trợ nhiều vai trò khác nhau như Advertiser, Scanner, Peripheral, Central.	Không rõ ràng các vai trò như BLE.

Bảng 2-1: So sánh giữa BLE và Bluetooth

2.2.2 Kiến trúc BLE

Kiến trúc của BLE bao gồm các lớp và giao thức khác nhau, phối hợp với nhau để cung cấp kết nối không dây tiết kiệm năng lượng (xem Hình 2-12).



Hình 2-12: Các lớp trong kiến trúc của BLE

2.2.2.1 Lớp vật lý (RF and PHY)

Lớp vật lý là lớp thấp nhất trong kiến trúc BLE, chịu trách nhiệm chính trong việc truyền và nhận tín hiệu radio. BLE hoạt động trong băng tần ISM (Industrial, Scientific, Medical) 2.4 GHz.

BLE sử dụng 40 kênh RF với khoảng cách 2 MHz giữa các kênh. Trong số 40 kênh này, có 3 kênh quảng bá được sử dụng để phát hiện thiết bị, thiết lập kết nối và phát sóng dữ liệu. 37 kênh còn lại là các kênh dữ liệu, được dùng cho giao tiếp hai chiều giữa các thiết bị đã kết nối. Các kênh này cũng có thể được sử dụng làm kênh quảng bá phụ trong Bluetooth 5. Các kênh quảng bá được chọn tần số để giảm thiểu nhiễu từ các kênh 1, 6 và 11 của IEEE 802.11, là các kênh thường dùng trong Wi-Fi.

2.2.2.2 Lớp liên kết (Link Layer)

Lớp liên kết đóng vai trò quan trọng trong việc quản lý các hoạt động truyền thông cơ bản và đảm bảo sự tin cậy của kết nối không dây. Lớp này định nghĩa cấu trúc gói tin, máy trạng thái và điều khiển radio, cũng như cung cấp mã hóa cấp lớp liên kết, chịu trách nhiệm về các hoạt động như quảng bá, quét và thiết lập kết nối.

Các hoạt động chính trong lớp liên kết:

- Thiết bị phát các gói tin quảng bá để thông báo sự hiện diện của mình. Các gói tin quảng bá có thể chứa dữ liệu về tên thiết bị, các dịch vụ hỗ trợ hoặc mức công suất phát. Các thiết bị khác có thể quét để tìm các gói quảng bá này.
- Thiết bị lắng nghe các gói tin quảng bá để khám phá các thiết bị khác. Có 2 dạng quét:
    - **Quét thụ động:** Thiết bị chỉ lắng nghe các gói tin quảng bá, tuần tự quét qua từng kênh quảng bá.
    - **Quét chủ động:** Thiết bị lắng nghe các gói tin quảng bá và gửi thêm yêu cầu quét để nhận thêm thông tin từ thiết bị quảng bá.
- Cho phép các thiết bị trao đổi dữ liệu một cách đáng tin cậy bằng cách sử dụng CRC, xác nhận và truyền lại các gói tin bị mất. Kết nối bắt đầu khi thiết bị quét nhận được gói tin quảng bá cho phép kết nối.

Các vai trò của các thiết bị trong BLE bao gồm:

- **Advertiser:** Thiết bị chỉ phát gói quảng bá.
- **Scanner:** Thiết bị chỉ lắng nghe gói quảng bá.
- **Peripheral:** Thiết bị kết nối với một hoặc nhiều thiết bị central.
- **Central:** Thiết bị kết nối với nhiều thiết bị peripheral.
- **Hybrid:** Thiết bị có thể vừa quảng bá vừa quét cùng lúc.

2.2.2.3 Lớp Generic Access Profile (GAP)

GAP là một trong những lớp đầu tiên mà các nhà phát triển BLE tiếp xúc, vì nó kiểm soát cách thiết bị hiển thị, kết nối với các thiết bị khác, cũng như cách khám phá và kết nối với các thiết bị từ xa.

GAP cung cấp quyền truy cập vào các hoạt động của lớp liên kết liên quan đến khám phá thiết bị, thiết lập và chấm dứt kết nối, cũng như kiểm soát thời gian kết nối. Cụ thể hơn, GAP quản lý các hoạt động như quảng bá, quét và thiết lập kết nối, vốn được định nghĩa trong lớp liên kết. Ngoài ra, GAP định nghĩa các vai trò thiết bị, mỗi vai trò có các yêu cầu riêng về bộ điều khiển. Các vai trò này xác định xem một thiết bị có khả năng chỉ truyền, chỉ nhận, hay cả vừa truyền vừa nhận tín hiệu radio. Một thiết bị có thể hỗ trợ nhiều vai trò, nhưng chỉ có thể đảm nhận một vai trò tại một thời điểm nhất định. Các vai trò đó bao gồm:

- **Broadcaster**: Chỉ gửi các sự kiện quảng bá và dữ liệu quảng bá, không thể nhận.
- **Observer**: Chỉ lắng nghe các sự kiện quảng bá và dữ liệu quảng bá, không thể truyền.

- **Peripheral**: Luôn là thiết bị ngoại vi, có thể kết nối và quảng bá. Được thiết kế cho các thiết bị đơn giản sử dụng một kết nối duy nhất với thiết bị có vai trò Central.
- **Central**: Luôn là thiết bị trung tâm, không bao giờ quảng bá. Được thiết kế cho các thiết bị chịu trách nhiệm khởi tạo và quản lý nhiều kết nối.

GAP cũng định nghĩa các chế độ và quy trình cho khám phá, kết nối và liên kết.

2.2.2.4 Lớp Attribute Protocol (ATT)

ATT là một giao thức quan trọng để truyền dữ liệu giữa các thiết bị BLE, hoạt động trên một kênh L2CAP chuyên dụng. Nó định nghĩa cách các thiết bị đóng vai trò là server và client tương tác với nhau để trao đổi dữ liệu thông qua các thuộc tính.

ATT cung cấp phương tiện để truyền dữ liệu giữa các thiết bị BLE. Nó dựa trên kết nối BLE và cung cấp các thủ tục để đọc, ghi, chỉ định và thông báo các giá trị thuộc tính. ATT định nghĩa hai vai trò chính:

- **Server**: Thiết bị lưu trữ dữ liệu dưới dạng một hoặc nhiều thuộc tính.
- **Client**: Thiết bị thu thập thông tin từ một hoặc nhiều server.

Các thuộc tính là đơn vị cơ bản dùng để lưu trữ dữ liệu trong BLE. Mỗi thuộc tính có một handle để định địa chỉ. Client sử dụng handle để truy cập các thuộc tính của server. Các thuộc tính có kiểu dữ liệu được mô tả bằng một UUID (Universally Unique Identifier). UUID xác định ý nghĩa của giá trị thuộc tính. Kiểu dữ liệu và handle của thuộc tính là thông tin công khai, nhưng quyền truy cập vào các thuộc tính là không công khai. Client có thể truy cập các thuộc tính của server bằng cách gửi các yêu cầu. Server sẽ trả lời các yêu cầu này bằng các thông báo phản hồi. Client cũng có thể gửi lệnh đến server để thay đổi các giá trị của thuộc tính.

ATT hỗ trợ nhiều phương thức để làm việc với các thuộc tính, xem Bảng 2-2.

Phương thức	Mô tả	Hướng đi của dữ liệu
Find Information	Tìm handle và kiểu dữ liệu của thuộc tính	Client -> Server
Find By Type Value	Tìm handle của các thuộc tính có kiểu dữ liệu và giá trị phù hợp	Client -> Server
Read By Group Type	Đọc giá trị của các thuộc tính có cùng kiểu dữ liệu trong một phạm vi	Client -> Server

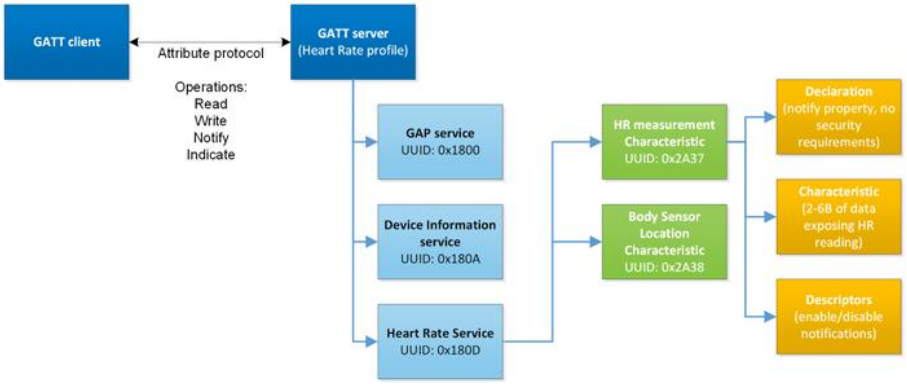
Read	Đọc giá trị của một handle nhất định	Client -> Server
Read Blob	Đọc các thuộc tính dài hơn 250 byte	Client -> Server
Read Multiple	Đọc nhiều giá trị cùng một lúc	Client -> Server
Write	Ghi giá trị vào một handle, không có phản hồi	Client -> Server
Prepare Write/Execute	Chuẩn bị một thủ tục ghi, được xếp hàng đợi ở server	Client -> Server
Handle Value Notification	Server thông báo cho client về một thuộc tính với giá trị mới	Server -> Client
Handle Value Indication	Server chỉ định cho client một thuộc tính với giá trị mới và yêu cầu client xác nhận	Server -> Client
Error response	Phản hồi lỗi khi có yêu cầu không thành công	Server -> Client

Bảng 2-2: Các phương thức làm việc với các thuộc tính trong lớp ATT

2.2.2.5 Lớp Generic Attribute Profile (GATT)

GATT đóng vai trò là lớp trung gian giữa giao thức ATT và các ứng dụng, cung cấp một cấu trúc có tổ chức và dễ hiểu để trao đổi dữ liệu. GATT được xây dựng dựa trên giao thức thuộc tính trong lớp ATT và thiết lập một khuôn khổ chung cho dữ liệu được truyền tải và lưu trữ bởi ATT. GATT quy định định dạng dữ liệu được chứa trên GATT server, giúp các thiết bị BLE khác nhau có thể hiểu và tương tác với nhau một cách nhất quán. GATT định nghĩa cách các thuộc tính, được vận chuyển bởi giao thức ATT, được định dạng thành các dịch vụ và các đặc tính.

- **Đặc tính** trong lớp GATT là giá trị được sử dụng trong một dịch vụ để hiển thị và/hoặc trao đổi dữ liệu và/hoặc điều khiển thông tin. Đặc tính chứa thông tin về cách giá trị có thể được truy cập, các yêu cầu bảo mật cần được đáp ứng và cách giá trị được hiển thị hoặc diễn giải. Đặc tính có thể chứa các mô tả giá trị hoặc cho phép cấu hình các chỉ định hoặc thông báo.
- **Dịch vụ** trong lớp GATT là tập hợp các đặc tính được sử dụng để thực hiện một chức năng cụ thể của thiết bị, ví dụ như theo dõi pin hoặc dữ liệu nhiệt độ. Một dịch vụ có thể chứa một hoặc nhiều đặc tính.



Hình 2-13: Cấu trúc dữ liệu và hoạt động mẫu trong lớp GATT

2.3 Cảm biến áp suất

Áp suất là tổng lực tác động lên một đơn vị diện tích. Ta thường đo áp suất của không khí hoặc chất lỏng trong một phạm vi nào đó, ví dụ như áp suất không khí hay áp suất nước biển. Có khá nhiều đơn vị để đo áp suất, phổ biến nhất là Pa, mmHg, psi, bar,…

Cảm biến áp suất là một linh kiện điện tử có thể đo và hiển thị được áp suất bằng một phương pháp nào đó[5]. Có 2 loại ngõ ra phổ biến của các loại cảm biến áp suát nói riêng và tất cả các cảm biến nói chung là ngõ ra dòng điện từ 4 – 20mA và ngõ ra điện áp từ 0 – 5V.

Đa số các cảm biến áp suất sử dụng hiệu ứng **piezoelectric** thuận (tạm dịch là hiệu ứng áp điện thuận). Hiệu ứng áp điện thuận là một hiện tượng vật lý trong đó một số loại vật liệu tạo ra một điện áp khi chúng bị nén, kéo dãn hoặc chịu ứng suất cơ học. Khi một lực cơ học như nén, kéo dãn, xoắn, bẻ, rung,… được áp dụng lên vật liệu, nó gây ra sự phân cực điện trong vật liệu, làm xuất hiện một điện áp giữa các bề mặt của nó. Điều này xảy ra do sự dịch chuyển của các ion trong mạng tinh thể của vật liệu, làm thay đổi sự cân bằng điện tích. Cảm biến sẽ tính được áp suất bằng cách đo điện áp thay đổi trên vật liệu.

Có 3 loại cảm biến áp suất phổ biến là cảm biến áp suất Gauge, cảm biến áp suất tuyệt đối và cảm biến áp suất vi sai.

- **Cảm biến áp suất Gauge** là một loại cảm biến áp suất đo áp suất của một chất lỏng hoặc khí so với áp suất không khí xung quanh (thường rơi vào khoảng 14.7 psi). Cảm biến Gauge có 2 đầu đo, một đầu đo môi trường cần đo, một đầu tiếp xúc với môi trường khí quyển.
- **Cảm biến áp suất tuyệt đối** là một loại cảm biến đo áp suất của chất lỏng hoặc khí so với áp suất chân không tuyệt đối (áp suất bằng 0). Cảm biến có một màng áp suất

được đặt trong một buồng kín, nơi một mặt tiếp xúc với môi trường cần đo, và mặt còn lại được cách ly bởi chân không.

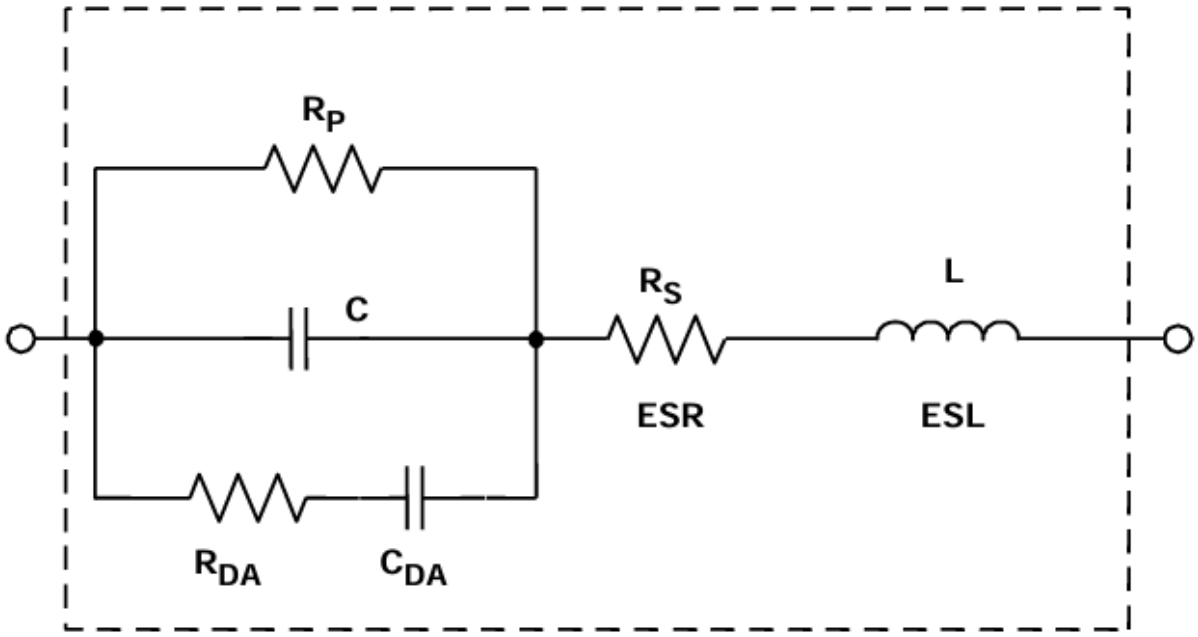
- **Cảm biến áp suất vi sai** là cảm biến đo độ chêch lệch áp suất của 2 môi trường khác nhau. Cảm biến thường có hai cổng kết nối áp suất, một cổng dành cho áp suất cao và một cổng dành cho áp suất thấp. Một màng áp suất nằm giữa hai cổng này, chịu tác động của sự chênh lệch áp suất.

2.4 Kỹ thuật decoupling

Đa số các IC sẽ bị giảm hiệu suất hoạt động nếu nguồn cấp vào chúng bị nhiễu hoặc không ổn định. Với các IC số cần phải có hiệu suất cao như các vi điều khiển hay FPGA thì việc có một nguồn đầu vào ổn định và ít nhiễu sẽ rất quan trọng [6]. Các IC sẽ có khả năng ngăn nhiễu của nguồn cấp ảnh hưởng đến tín hiệu đầu ra, được đo bằng thông số là **PSRR - Power Supply Rejection Ratio**. Đối với các nhiễu ở tần số thấp (1kHz đến 10kHz) thì PSRR của các IC đều khá cao, tín hiệu đầu ra gần như không bị ảnh hưởng bởi nhiễu ở tần số thấp. Tuy nhiên, với các nhiễu có tần số cao hơn, chỉ số PSRR lại giảm, tức là các IC khó có thể ngăn nhiễu tần số cao ảnh hưởng đến tín hiệu đầu ra. Đã có rất nhiều cách để giảm ảnh hưởng của nhiễu, nhưng cách phổ biến và thông dụng nhất mà các kĩ sư thường tin tưởng là sử dụng mạch decoupling.

Mạch decoupling được dùng để tách lọc các nhiễu ở nhiều dải tần số khác nhau. Trong đa số các mạch decoupling thông dụng thì sẽ sử dụng tập hợp các loại tụ điện khác nhau, thông dụng nhất là tụ điện hoá (lọc nhiễu ở tần số thấp) và tụ điện gốm (lọc nhiễu ở tần số cao). Người ta sẽ mắc tụ điện ở giữa tín hiệu nguồn và đất (GND). Ở đây, tụ sẽ đóng vai trò như một điện trở ở các tần số tín hiệu khác nhau. Khi trở kháng của tụ điện là thấp ở một dải tần số nào đó thì tín hiệu nằm trong dải tần số đó sẽ đi thẳng xuống đất, những tín hiệu có tần số ở dải khác, lúc này trở kháng của tụ sẽ cao, khiến tín hiệu không thể xuống đất và sẽ đi tiếp vào IC. Tụ điện sẽ có trở kháng nhỏ nhất khi đang hoạt động ở **tần số tự cộng hưởng (Self-Resonant Frequency – SRF)**. Đây là tần số mà dung kháng và cảm kháng bên trong tụ điện bằng nhau.

Đương nhiên là ngoài thị trường sẽ không có tụ điện lý tưởng và sẽ luôn có những yếu tố ký sinh trong cấu trúc của tụ. Trong đó, **ESL** và **ESR** là hai yếu tố quan trọng quyết định hiệu suất của tụ trong các ứng dụng thực tế.

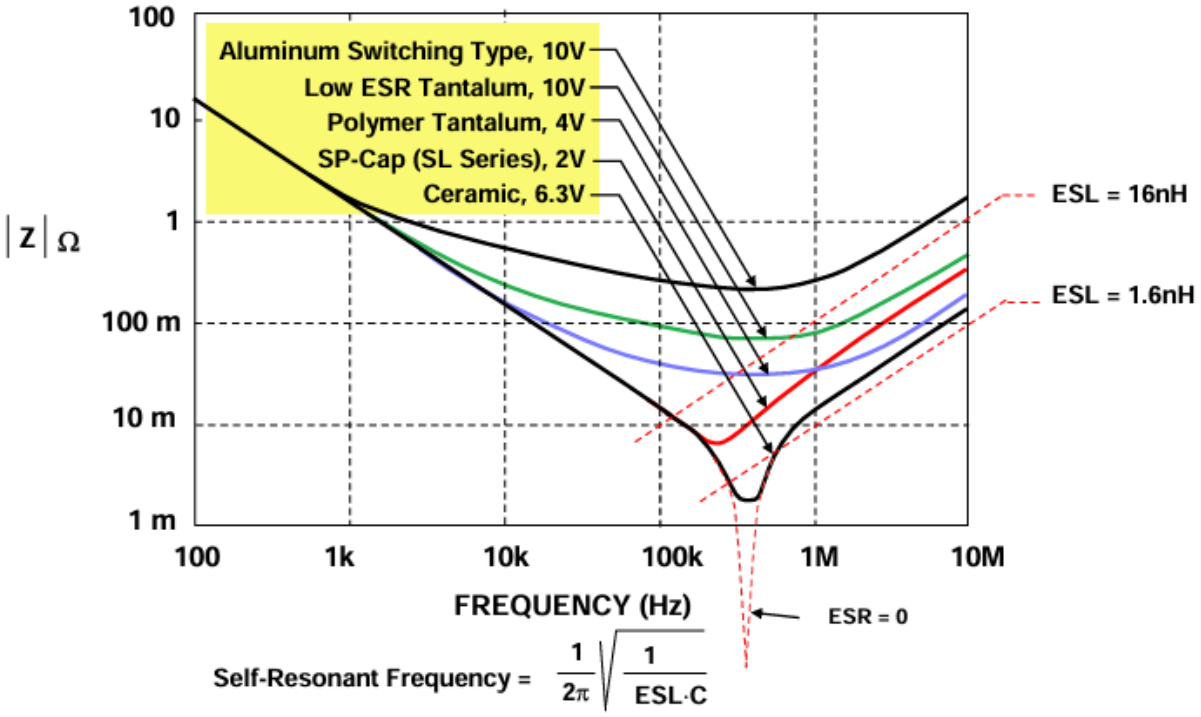


Hình 2-14: Sơ đồ của một tụ điện thực với các yếu tố ký sinh

ESL (Equivalent Series Inductance) là điện cảm ký sinh của tụ, được tạo ra do các chân nối và cấu trúc cuộn bên trong của tụ. Tại tần số cao, dung kháng bên trong tụ càng giảm và cảm kháng càng tăng. Cảm kháng bên trong tụ gây ra là do ESL. Khi tần số lớn hơn SRF, cảm kháng sẽ càng tăng, khiến cho trở kháng tổng của tụ điện tăng lên lại so với lúc ở SRF. Điều này khiến ở dải tần số lớn hơn SRF, tụ không thể đem nhiễu xuống đất. Vì thế, ta cần những tụ có ESL càng nhỏ càng tốt để có thể lọc nhiễu ở tần số cao hơn. Một trong các phương pháp là sử dụng tụ dán, giúp giảm thiểu chiều dài dây nối của tụ. Với cấu trúc trong tụ, tụ gốm sẽ là lựa chọn tối ưu nhất để giảm thiểu tối đa ESL mà phổ biến ngoài thị trường.

ESR (Equivalent Series Resistance) là điện trở ký sinh trong tụ điện đại diện cho tổn hao năng lượng bên trong tụ. ESR phát sinh từ các nguồn như điện trở của dây dẫn, chân tụ, điện trở của lớp điện môi hay tiếp xúc giữa các vật liệu bên trong tụ. ESR khiến cho trở kháng tổng của tụ tăng lên và khiến tổn hao năng lượng, chủ yếu ở dạng nhiệt. Để gia tăng khả năng lọc nhiễu, ta phải chọn tụ điện có ESR càng thấp như tụ gốm hay tụ polymer.

Hình 2-15 sẽ cho ta cái nhìn tổng quan hơn về ảnh hưởng của ESR và ESL lên trở kháng của tụ tại các dải tần số khác nhau.



Hình 2-15: Trở kháng của các tụ 100µF khác nhau theo tần số

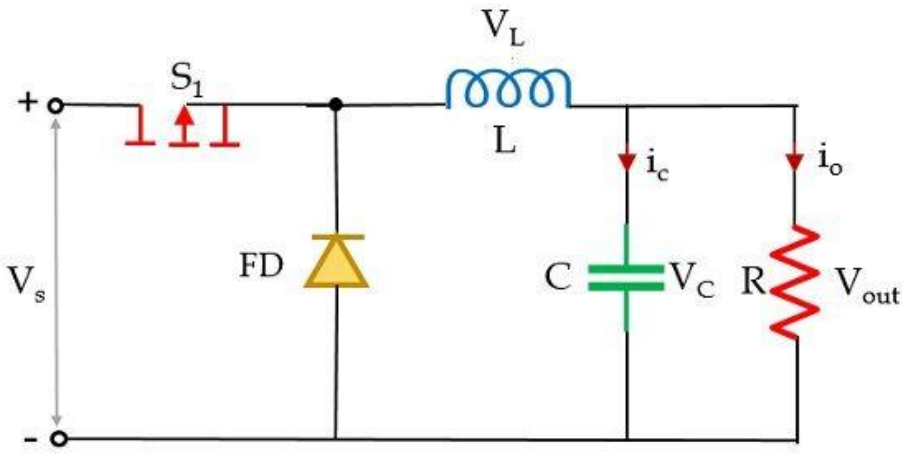
Trong một mạch decoupling cơ bản thì có 2 loại tụ điện chính được sử dụng.

- Một tụ có điện dung cao (vào khoảng 10µF - 100µF) đặt cách không quá 5cm so với chân nguồn của IC để làm một bể chứa dòng điện tức thời cho IC mỗi khi cần.
- Loại tụ điện thứ hai là một tụ lọc nhiễu với mỗi chân nguồn của IC. Vì các chân nguồn ở IC là nằm ở các vị trí khác nhau trên PCB nên nếu chỉ sử dụng 1 tụ lọc cho toàn IC thì đối với những chân nguồn xa tụ lọc thì chúng sẽ bị ảnh hưởng bởi cảm kháng trên đường dây. Ngoài ra, đường dây xa đấy sẽ như một anten hút các nhiễu điện từ vào, khiến cho nguồn vào chân nguồn đó bị nhiễu. Nên với mỗi chân nguồn, ta sẽ gắn một tụ lọc nhiễu (điện dung vào khoảng 10nF – 100nF) càng gần chân nguồn càng tốt để đem nhiễu xuống đất.
- Các tụ nên được nối với một vùng GND lớn, ít cảm kháng bằng một đường kết nối càng ngắn càng tốt để giảm tối đa cảm kháng đường dây.

2.5 Buck converter cơ bản

Có rất nhiều phương pháp khác nhau để giảm điện áp đầu vào và cho ra được điện áp ngõ ra nhỏ hơn như sử dụng cầu phân áp, sử dụng diode Zener hay mạch biến đổi điện áp tuyến tính. Những mạch nói trên giảm điện áp ngõ ra bằng cách chuyển chênh lệch điện áp thành nhiệt, điều này khiến hiệu suất cuuar mạch không cao.

Một mạch buck converter có thể giảm điện áp đầu vào, tuy nhiên phần chêch lệch điện áp sẽ được bù bằng cách tăng dòng điện ngõ ra lên, khiến mạch có công suất ngõ ra giữ nguyên so với đầu vào[7]. Dưới Hình 2-16 là một mạch buck converter cơ bản.



Hình 2-16: Schematic mạch buck converter cơ bản

MOSFET S_1 đóng vai trò như một công tắc. Khi S_1 dẫn, dòng điện từ V_s sẽ đi vào mạch. Xét cuộn cảm L, khi dòng đi vào mạch, đồng nghĩa với việc dòng đi qua cuộn cảm I_L tăng, khiến cho mật độ từ thông xung quanh L tăng. Lúc này, L sẽ hấp thụ năng lượng từ dòng ngõ vào và phân cực với cực dương hướng về nguồn đầu vào. Khi có sự phân cực giữa 2 đầu L, nó sẽ tạo cho mình điện áp V_L. Điện áp V_L làm giảm điện áp giữa 2 đầu R, tức là làm giảm điện áp ngõ ra của mạch.

$$V_{out} = V_s - V_L$$

Khi dòng điện qua L, dòng điện cũng đi vào điện trở R và tụ điện C, giúp nạp điện cho C. Lúc này diode FD đang ở chế độ phân cực ngược nên đang tắt, ngăn cản dòng điện đi qua FD.

Khi S_1 không dẫn, tức là mạch hở, I_L bắt bầu giảm dần, khiến cho mật độ từ thông xung quanh L giảm dần. Lúc này, L sẽ phân cực ngược lại so với lúc S_1 dẫn và sẽ hoạt động như một nguồn điện. Nếu V_L lớn hơn V_C thì lúc này L sẽ nạp điện cho C. Khi $V_L = V_C$, cả L và C đều sẽ xả dòng vào R và đi vòng ngược lại và đi qua FD. Lúc này FD phân cực thuận nên sẽ mở cho dòng điện đi qua.

Khi S_1 dẫn, I_R bắt nguồn từ dòng từ nguồn đầu vào. Khi S_1 đóng thì I_R sẽ bắt nguồn từ I_L. Khi cộng 2 trạng thái lại thì đương nhiên là là tổng I_R sẽ lớn hơn dòng điện của nguồn đầu

vào. Những điều trên chứng minh được là buck converter có thể giúp giảm điện áp ngõ ra, đồng thời tăng dòng điện nguồn đầu ra.

Điện áp ngõ ra dựa vào 2 thành phần chính là điện áp đầu vào và chu kỳ nhiệm vụ của S_1 (ký hiệu là D) – là tỉ lệ thời gian S_1 dẫn so với tổng chu kỳ hoạt động của S_1 (đơn vị là %). Vì khi S_1 dẫn thì V_{out} giảm còn khi S_1 đóng thì V_{out} dần tăng trở lại gần với V_{in}, nên điện áp ngõ ra được tính toán như sau:

$$V_{out} = V_{in}.D$$

3. THỰC HIỆN ĐỀ TÀI

3.1 Lựa chọn phần cứng

Như đã đề cập ở phần 1.2, đề tài TPMS sẽ bao gồm 3 mạch là mạch hệ thống cảm biến, mạch ECU TPMS và mạch ECU màn hình.

3.1.1 Hệ thống cảm biến

Mạch hệ thống cảm biến bao gồm một cảm biến áp suất không khí và một vi điều khiển hỗ trợ BLE. Để gần giống với thực tế nhất thì mạch hệ thống cảm biến phải có những yêu cầu sau:

- Có tầm đo bao quát được áp suất không khí thường thấy bên trong lốp xe.
- Có thể đọc và xử lý tín hiệu tương tự từ cảm biến áp suất.
- Có khả năng truyền thông tin không dây sử dụng BLE.
- Càng tiết kiệm năng lượng càng tốt do hệ thống sử dụng pin để nuôi.
- Mạch càng nhỏ gọn càng tốt, lý tưởng là dưới 10x10cm.

Theo như các dữ liệu thu thập được và thống kê ở [6], đối với các dòng xe dân dụng và nổi tiếng như Mazda, Toyota, Mercedes, Audi,… thì **áp suất không khí lý tưởng** bên trong lốp xe trước rơi vào khoảng 28 – 33 psi, còn đối với lốp xe sau thì vào khoảng 30 – 38 psi. Vì thế, cảm biến áp suất không khí của chúng ta phải có tầm đo bao quát được các khoảng nói trên.

Trong các lựa chọn khả thi ngoài thị trường, đề tài TPMS chọn cảm biến **XGZP6847A500KPG**, là một cảm biến thuộc dòng XGZP6847A của CFSensor. Dòng cảm biến XGZP6874A là dòng cảm biến đo áp suất không khí và xuất ra tín hiệu tương tự tỉ lệ với

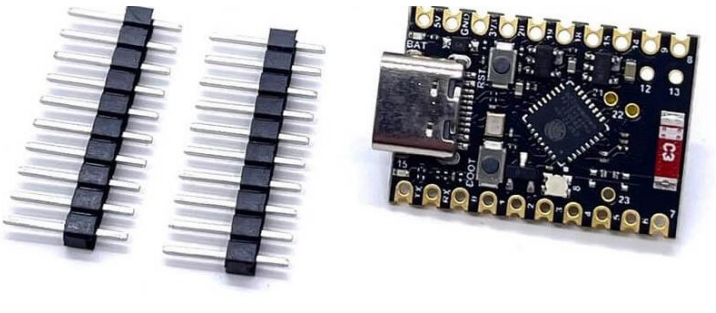
giá trị trong tầm đo của nó[9]. Dòng cảm biến này có thể tuỳ chỉnh một số tham số như tầm đo, điện áp hoạt động, điện áp ngõ ra,… Cảm biến XGZP6847A500KPG mặc định có tầm đo là từ 0 ~ 500kPa, hoặc là từ 0 ~ 72.5psi, hoạt động tại 5V và điện áp ngõ ra có tầm là từ 0.5 ~ 4.5V.

Tầm đo của cảm biến đã phù hợp với ngữ cảnh đề tài. Tuy nhiên, điện áp hoạt động của cảm biến lại không quá phù hợp. Trong thực tế, mạch hệ thống cảm biến được đặt bên trong lốp xe và phải sử dụng pin để duy trì hoạt động cho hệ thống. Vì thế, hệ thống phải tiêu thụ càng ít năng lượng càng tốt để kéo dài thời gian hoạt động theo tuổi thọ của pin. Để làm được điều này, ta sẽ sử dụng khả năng thay đổi mức điện áp hoạt động của cảm biến để giảm điện áp hoạt động của cảm biến từ 5V xuống còn 3.3V. Điện áp ngõ ra của cảm biến từ đó cũng giảm theo, từ 0.2 ~ 2.7V.

Về phần xử lý các thông tin của cảm biến, ta cần một vi điều khiển có thể thực hiện những tác vụ như đọc và xử lý tín hiệu tương tự hay truyền thông tin không dây. Đề tài TPMS sẽ sử dụng vi điều khiển **ESP32-C6** thông qua mạch phát triển **ESP32-C6 Super Mini** của Nogolo.

Dòng **ESP32-C6** là SoC (System on Chip) hỗ trợ Wi-Fi 6, Bluetooth 5, BLE 3.0 và Thread 1.3[10]. ESP32-C6 bao gồm rất nhiều ngoại vi bao gồm tới 30 GPIO, 7 kênh ADC 12-bit, UART, SPI, I2C, I2S, USB, PWM,… Dòng ESP32-C6 hoạt động ở mức điện áp từ 3.0 ~ 3.6V và dòng tối thiểu cấp vào SoC là 0.5A. Theo như bảng 5-9, mục 5.6.1 của tài liệu [10], khi cấp cho ESP32-C6 điện áp 3.3V và hoạt động ở 25^{o}C để truyền đi tín hiệu RF chuẩn 802.15.4, lượng dòng điện SoC sẽ tiêu thụ tối đa là 305mA. Đây đều là những thông số khá phù hợp với yêu cầu của hệ thống cảm biến.

Đối với mạch phát triển ESP32-C6 Super Mini thì đây chỉ là mạch được xây dựng để người dùng có thể dễ dàng giao tiếp với ESP32-C6. Mạch này có kích thước là 26x18mm, rất phù hợp với yêu cầu của mạch hệ thống cảm biến.



Hình 3-1: Mạch phát triển ESP32-C6 Super Mini của Nologo

3.1.2 ECU TPMS

Mạch ECU TPMS sẽ bao gồm 1 vi điều khiển để nhận dữ liệu BLE từ hệ thống cảm biến, 1 vi điều khiển để xử lý dữ liệu để truyền cho ECU màn hình thông qua CAN, mạch CAN transceiver và một mạch nguồn để giảm điện áp đầu vào là 12V xuống 3.3V để nuôi cho 2 vi điều khiển nói trên.

Giống như hệ thống cảm biến, mạch ECU TPMS sẽ sử dụng mạch ESP32-C6 Super Mini để nhận tín hiệu BLE, đồng thời sẽ truyền dữ liệu nhận được sang cho vi điều khiển còn lại để xử lý thông qua SPI. Vi điều khiển dùng để truyền dữ liệu sang ECU màn hình mà đề tài này sử dụng là STM32F042K6T6. Vì đề tài này không sử dụng quá nhiều ngoại vi và công suất mà đề tài này yêu cầu là không quá cao nên đề tài sẽ chọn những vi điều khiển dòng STM32F0 hoặc STM8 để xử lý các thông tin. Vì các dòng STM8 có hỗ trợ CAN giá thành khá đắt (khoảng 3 – 4USD) và khó tìm ở Việt Nam hay Trung Quốc nên đề tài sẽ chọn vi điều khiển dòng STM32F0. STM32F042K6T6 có giá thành khá rẻ (vào khoảng 1.2USD), có hỗ trợ CAN cũng như nhiều ngoại vi thông dụng khác và có thể tìm thấy trong các thư viện linh kiện của Trung Quốc. CPU của STM32F042K6T6 là thuộc dòng Cortex-M0 với tần số xung nhịp hoạt động là 48MHz, đủ nhanh với yêu cầu của đề tài. Để vi điều khiển có thể nạp firmware và hoạt động được thì cần nhiều mạch xung quanh.

Đối với mạch CAN transceiver thì đề tài sẽ sử dụng IC CAN transceiver TJA1050, là một giao diện giữa CAN controller và bus CAN[11] với 1 kênh nối ra bus CAN. Lý do chọn IC này vì nó khá phổ biến, có nhiều tài liệu tham khảo để cấu hình hay thiết kế phần cứng xung quanh nó.

Đối với mạch nguồn, đề tài sẽ sử dụng IC TPS54331 làm trung tâm. TPS54331 là một bộ giảm áp DC, có thể lấy nguồn đầu vào lên đến 28V, 3A[12]. Trong [12] đã có sơ đồ mạch ứng dụng IC này và ta chỉ cần lựa chọn đúng linh kiện sao cho điện áp ngõ ra gần với 3.3V nhất có thể, cộng với giá thành của TPS54331 cũng khá rẻ (1.3USD). Ngoài ra, TPS54331 có

các tính năng rất quan trọng như đóng ngắt sử dụng MOSFET, giúp tiết kiệm năng lượng hơn so với BJT hay bo vệ quá áp, quá nhiệt, ngắn mạch,…

3.1.3 ECU màn hình

Mạch ECU màn hình cơ bản là giống 90% so với mạch ECU TPMS, chỉ khác là ECU màn hình chỉ cần 1 vi điều khiển là STM32F042K6T6 để xử lý thông tin và thêm 1 màn hình để hiển thị thông tin nhận được từ ECU TPMS. Vì đề tài đang ở quy mô của môn Đồ án 2 nên đề tài sẽ chỉ sử dụng màn hình OLED 0.96in SSD1306 và hỗ trợ giao tiếp I^2C.

Hình 3-2: Module màn hình OLED 0.96in SSD1306 hỗ trợ giao tiếp I^2C

3.2 Thiết kế phần cứng

3.2.1 Hệ thống cảm biến

Cảm biến XGZP6847A500KPG có sơ đồ kết nối như Bảng 3-1.

PIN1	**PIN2**	**PIN3**	**PIN4**	**PIN5**	**PIN6**
N/C	**VDD**	**GND**	**VDD**	**OUT**	**GND**
N/C	No connect, không kết nối với bất cứ mạch ở ngoài nào hay GND				
VDD	Nguồn cấp				
GND	Đất				
OUT	Điện áp ngõ ra				

Bảng 3-1: Sơ đồ kết nối các chân của XGZP6847A500KPG[9]

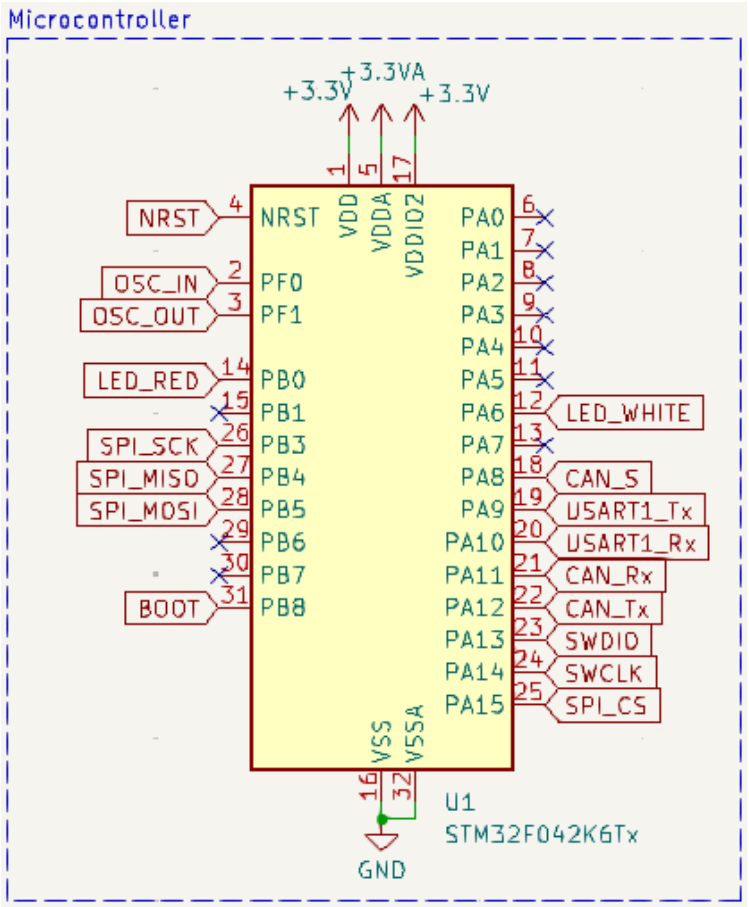
Vì bên dưới cảm biến đã có linh kiện thực hiện nhiệm vụ đọc ngõ ra từ cảm biến và xử lý tín hiệu số cho ra ngõ ra nên để đọc cảm biến này thì không cần thêm bất cứ linh kiện ngoại nào. Ngõ ra PIN5 của cảm biến sẽ được nối vào 1 kênh ADC của ESP32C6 để xử lý tín hiệu đọc vào. Đề tài này sẽ sử dụng kênh ADC tích hợp trong GPIO 5 của ESP32C6.

3.2.2 ECU TPMS

Mạch ECU TPMS bao gồm nhiều thành phần nhỏ ghép lại với nhau:

- Mạch vi điều khiển
- Mạch decoupling cho tín hiệu digital và analog
- Mạch hệ thống và cấu hình cho vi điều khiển
- Mạch CAN transceiver
- Mạch nguồn
- Thành phần khác

3.2.2.1 Mạch vi điều khiển



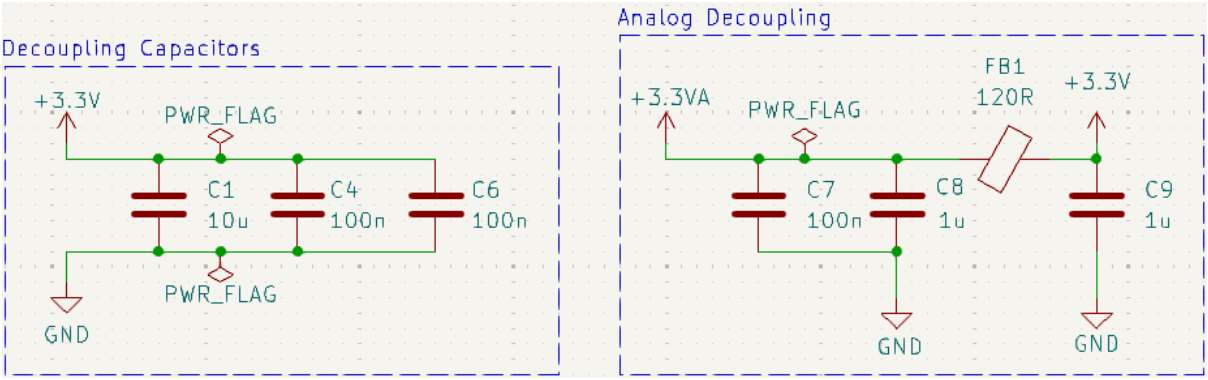
Hình 3-3: Schematic mạch vi điều khiển

Mạch này chủ yếu chỉ đánh nhãn cho những chân cần phải sử dụng của vi điều khiển và cấp nguồn cho vi điều khiển. Theo như mục 6.1.2 của [11], mức điện áp tiêu biểu cấp vào các chân VDD và VDDA là 3.3V. Trong STM32F042K6T6, ta có 2 chân để cấp nguồn digital cho vi điều khiển, chân VDD dùng để cấp nguồn cho hệ thống của cả vi điều khiển, còn chân VDDIO2 dùng để cấp nguồn cho các chân từ **PA8 đến PA15**. Nguồn VDDIO2

hoàn toàn độc lập so với VDD và VDDA. Nếu muốn những chân nói trên có một nguồn điện áp digital riêng thì ta sẽ cấp điện áp khác cho chân VDDIO2. Ở đề tài TPMS, các chân nói trên không cần nguồn ngoài để hoạt động nên chân VDDIO2 sẽ được cấp mức điện áp giống như VDD.

Chân VDDA dùng để cấp nguồn analog, phụ thuộc cho các hoạt động liên quan đến xử lý các tín hiệu tương tự. Ta có thể sử dụng cùng nguồn 3.3V của VDD để cấp cho chân VDDA, tuy nhiên vì tín hiệu analog cần được phải tách lọc sâu hơn nên ở mạch này, nguồn VDD và VDDA là khác nhau.

3.2.2.2 Mạch decoupling

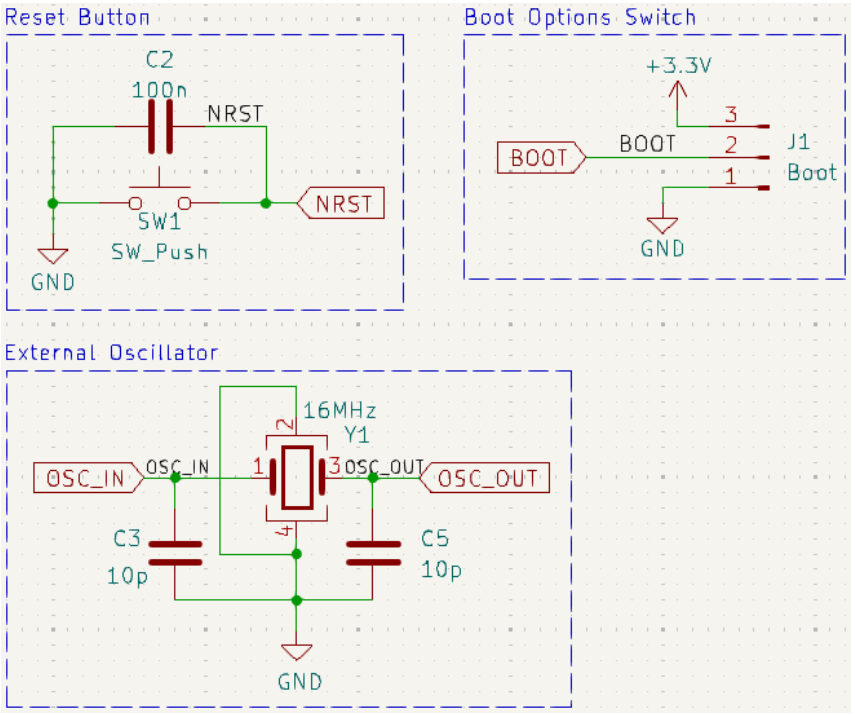


Hình 3-4: Schematic mạch decoupling digital và analog

Trong STM32F042K6T6, có tổng cộng 2 chân nguồn digital (VDD và VDDIO2) và 1 chân nguồn analog (VDDA). Với 2 chân nguồn digital, mỗi chân sẽ có 1 tụ gốm 100nF đặt ở gần và một tụ gốm 10µF để cấp dòng điện tức thời cho vi điều khiển.

Đối với nguồn analog, là một nguồn cần phải xử lý kĩ hơn vì các bộ ADC hay DAC rất nhạy cảm. Ở mạch này, một ferrite bead được sử dụng để chặn nhiễu cao tần từ nguồn cấp. Ferrite bead được mắc nối tiếp giữa nguồn digital và analog để tách biệt hai nguồn này với nhau. Nguồn digital thường sẽ có nhiễu cao tần, các tần số thoát được khỏi mạch decoupling digital. Các mạch analog yêu cầu nguồn cấp vào phải sạch để tránh sai lệch tín hiệu đầu vào. Ferrite bead có trở kháng cao khi tần số cao nên sẽ chặn các nhiễu cao tần và chỉ cho tín hiệu DC hoặc tần số thấp đi qua. Hai tụ 100nF và 1µF trước ferrite bead dùng để lọc tín hiệu trung tần bước đầu tới chân VDDA. Khi tín hiệu qua ferrite bead, nó sẽ tiếp tục được "dọn dẹp" bởi tụ 1µF ngay sau. Tụ này còn có tác dụng cấp dòng điện tức thời cho tải khi cần.

3.2.2.3 *Mạch hệ thống và cấu hình cho vi điều khiển*



Hình 3-5: Schematic mạch nút reset, mạch chọn chế độ boot và mạch dao động ngoại

Trong hầu như tất cả các vi điều khiển đều sẽ có 1 chân dùng để nhận tín hiệu bên ngoài, làm chức năng khởi động lại hệ thống vi điều khiển. Tuỳ vào dòng vi điều khiển mà ta phải đưa mức điện áp của chân reset xuống LOW hoặc HIGH. Đối với STM32F042K6T6, để khởi động lại hệ thống thì ta phải đưa chân reset (cụ thể là NRST) xuống LOW. Trong mục 6.3.15 của [11], STMicroelectronics đã để cho ta một mạch nút nhấn để reset vi điều khiển bằng cách nối song song với một tụ điện 100nF xuống GND. Tụ điện ở đây sẽ như một phương pháp chống rung cho nút nhấn. Khi nhấn nút, chân NRST sẽ nối trực tiếp với GND, khiến cho vi điều khiển sẽ reset.

Trong các vi điều khiển dòng STM32, chúng đều sẽ có 1 chân GPIO để giúp cho người dùng chọn chế độ boot cho vi điều khiển. Chế độ boot ở các dòng vi điều khiển STM32 sẽ giúp cho vi điều khiển chọn được vùng nhớ khởi đầu trong bộ nhớ để tiến hành thực thi chương trình trong vùng nhớ đó. Có 3 chế độ boot, bao gồm boot từ bộ nhớ Flash của người dùng, boot từ bộ nhớ hệ thống và boot từ vùng SRAM được tích hợp. Chỉ có 2 chế độ được sử dụng thông dụng và thường xuyên là boot từ bộ nhớ Flash của người dùng, boot từ bộ nhớ hệ thống.

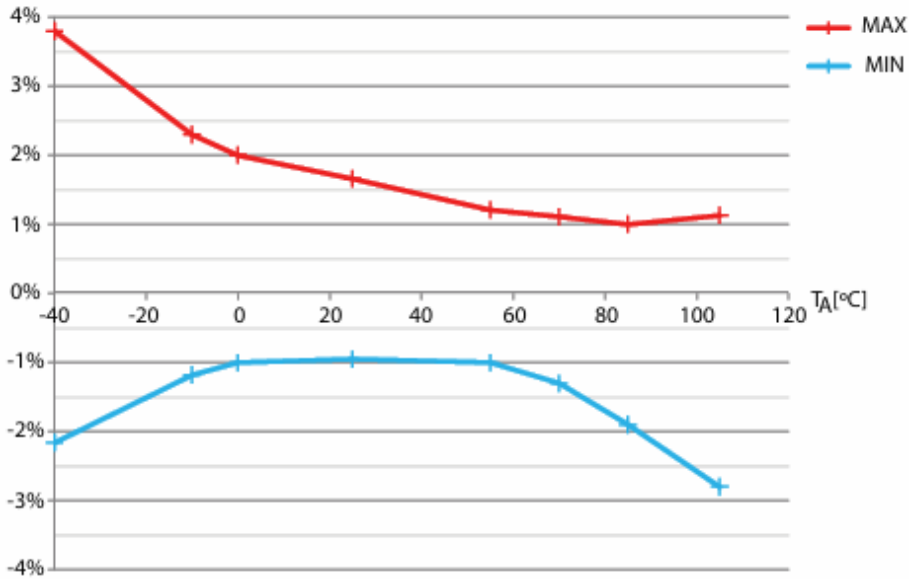
- Boot từ vùng nhớ Flash: Khi vi điều khiển được cấp nguồn hay reset, nó sẽ chạy vào địa chỉ đầu tiên của vùng Flash (thường sẽ là địa chỉ 0x08000000) để thực thi các lệnh

tại đó. Vùng nhớ Flash là nơi chứa chương trình mà người dùng lập trình. Vì vậy, đây là chế độ được chọn khi người dùng muốn vi điều khiển thực hiện chương trình đã nạp cho.

- Boot từ vùng nhớ hệ thống: Khi này, vi điều khiển sẽ chọn vùng nhớ của hệ thống để thực thi đoạn chương trình ở trong vùng nhớ đó. Trong vùng nhớ này chứa sẵn đoạn chương trình do nhà sản xuất phát triển dùng để khởi tạo các ngoại vi như USART, I2C hay USB để sẵn sàng nhận các file mã máy từ máy chủ. Các mã máy trong file đó sẽ được đoạn chương trình này ghi vào bộ nhớ Flash để sẵn sàng thực thi trong chế độ boot từ vùng nhớ Flash. Chế độ boot này thường dùng để nạp chương trình qua các phương thức khác ngoài việc sử dụng phương thức nạp code chính của các vi điều khiển dòng STM32 là SWD (Serial Wire Debug).

Khi chân Boot có mức điện áp là LOW thì sẽ hoạt động ở chế độ boot từ vùng nhớ Flash, khi HIGH thì sẽ hoạt động ở chế độ boot từ vùng nhớ hệ thống. Để chọn mức điện áp cho chân Boot, mạch này đơn giản sẽ sử dụng hàng rào 3 chân đực, 1 chân nối với nguồn, 1 chân nối với GND, chân chính giữa nối với chân Boot. Mạch này sử dụng 1 jumper để nối chân Boot với 1 trong 2 lựa chọn ở hai bên.

Trong đa số các vi điều khiển đều sẽ có ít nhất một bộ dao động tần số cao tích hợp nội bên trong (**HSI** – High-Speed Internal oscillator) dùng để cung cấp xung nhịp cho CPU và các ngoại vi khác. Tuy nhiên, HSI thường sẽ không chính xác vì tác động từ nhiệt độ (xem **Error! Reference source not found.**). Vì thế, mạch này sẽ sử dụng một bộ dao động ngoại tần số cao (**HSE** – High-Speed External oscillator) để cung cấp xung nhịp cho vi điều khiển. Theo Hình 3-7 từ [11], tần số của HSE có thể đi từ 4MHz đến 32MHz. Với yêu cầu là cung cấp nhịp đủ cao để phục vụ cho phương thức cần tốc độ nhanh như CAN và phổ biến ngoài thị trường thì mạch này sẽ chọn HSE có tần số là 16MHz.



Hình 3-6: Ảnh hưởng của nhiệt độ đến độ chính xác của bộ dao động nội tần số cao

Symbol	Parameter	Conditions[1]	Min[2]	Typ	Max[2]	Unit
f_{OSC_IN}	Oscillator frequency	-	4	8	32	MHz
R_F	Feedback resistor	-	-	200	-	kΩ
I_{DD}	HSE current consumption	During startup[3]	-	-	8.5	mA
		V_{DD} = 3.3 V, Rm = 30 Ω, CL = 10 pF@8 MHz	-	0.4	-	
		V_{DD} = 3.3 V, Rm = 45 Ω, CL = 10 pF@8 MHz	-	0.5	-	
		V_{DD} = 3.3 V, Rm = 30 Ω, CL = 5 pF@32 MHz	-	0.8	-	
		V_{DD} = 3.3 V, Rm = 30 Ω, CL = 10 pF@32 MHz	-	1	-	
		V_{DD} = 3.3 V, Rm = 30 Ω, CL = 20 pF@32 MHz	-	1.5	-	
g_m	Oscillator transconductance	Startup	10	-	-	mA/V
$t_{SU(HSE)}$[4]	Startup time	V_{DD} is stabilized	-	2	-	ms

Hình 3-7: Đặc tính của bộ dao động ngoại tần số cao

Schematic của mạch dao động ở trên Hình 3-5 sử dụng nguyên lý của mạch dao động Pierce, là mạch sử dụng rất phổ biến trong các vi điều khiển hay các IC cần dao động chính xác vì tính chính xác, dễ tích hợp và hiệu quả cao. Thạch anh trung tâm có vai trò như phần tử chọn tần số dao động của mạch và sẽ dao động tại tần số cộng hưởng của nó. Trong thạch anh có một điện cảm tương đương và một điện dung ký sinh trong phần tử, hai thành phần sẽ tạo thành tần số cộng hưởng của thạch anh. Tuy nhiên, do thành phần điện dung ký sinh là thông số không lý tưởng và khó kiểm soát, nên mạch sẽ mắc thêm 2 tụ điện để phối hợp với

điện cảm tương đương tạo thành tần số cộng hưởng chính xác với tần số dao động mong muốn. Ta chọn giá trị 2 tụ điện theo công thức sau được cung cấp trong [14].

$$C_L = \frac{C_{L1}.C_{L2}}{C_{L1} + C_{L2}} + C_S$$

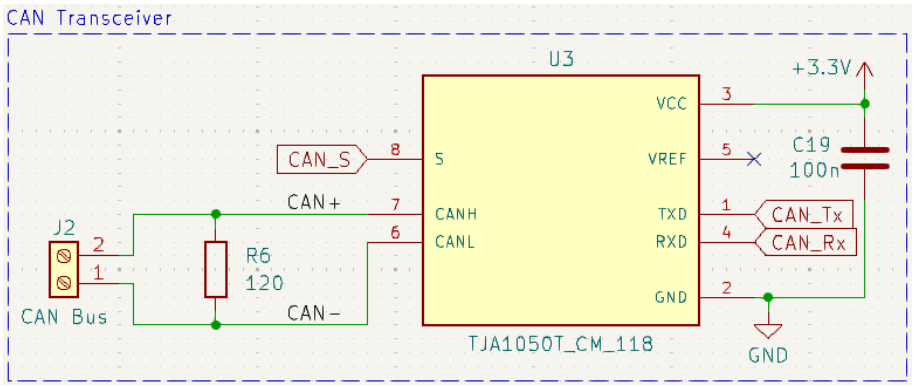
Với: **C_L** là giá trị điện dung cần phải tìm để tạo thành số cộng hưởng mong muốn

C_{L1} và **C_{L2}** là giá trị của 2 tụ điện trong mạch Pierce

C_S là giá trị điện dung ký sinh trong thạch anh

Sau các quá trình tính toán và tham khảo, mạch sẽ chọn 2 tụ điện gốm có giá trị 10µF.

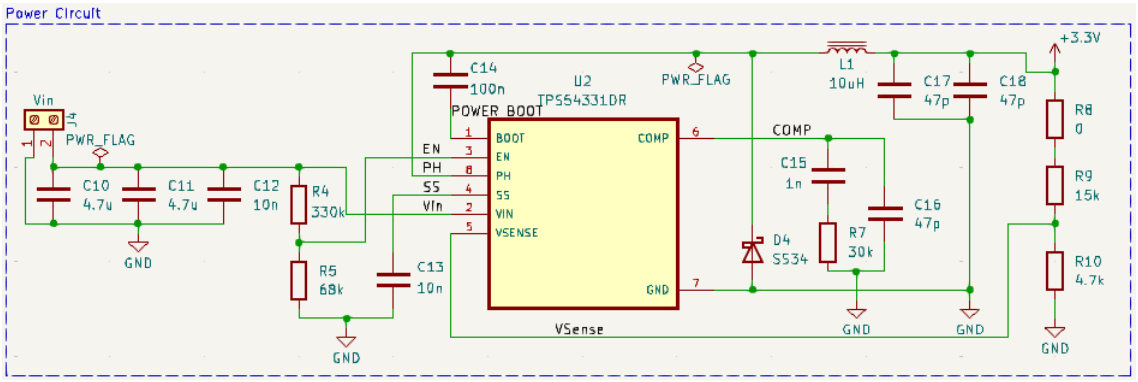
3.2.2.4 Mạch CAN transceiver



Hình 3-8: Schematic mạch CAN transceiver

Điện áp cấp vào chân VCC tối đa 6V nên mạch này sử dụng nguồn 3.3V từ mạch nguồn. Nguồn đưa vào sẽ được lọc nhiễu sử dụng tụ gốm 100nF. Chân TXD và RXD dùng để giao tiếp với CAN controller bên trong vi điều khiển. Chân S dùng để chọn chế độ cho node CAN. Mặc định, TJA1050 sẽ ở chế độ tốc độ cao (high-speed mode) và họ khuyến nghị nên nối chân S xuống GND khi sử dụng chế độ này. Khi chân S nối lên VCC thì TJA1050 sẽ hoạt động ở chế độ im lặng (silent mode), là chế độ chỉ nhận tín hiệu từ những node khác, không tha gia truyền tín hiệu. Để có thể linh hoạt chuyển chế độ cho TJA1050, chân S sẽ được nối với 1 chân GPIO trong vi điều khiển. 2 chân CANH và CANL sẽ được đưa ra một terminal block để có thể gắn cặp dây giao tiếp với các node khác. Vì tín hiệu trong CAN bus là tín hiệu vi sai nên khi vẽ layout thì phải sử dụng tính năng “Differential pair” trong KiCAD. Giữa hai đường CANH và CANL sẽ có 1 điện trở 120Ω để hấp thụ sóng phản xạ.

3.2.2.5 Mạch nguồn



Hình 3-9: Schematic mạch nguồn

Mạch nguồn này sẽ có nhiệm vụ chính là giảm áp nguồn đầu vào từ 12V xuống 3.3V ổn định. Mạch này sẽ lấy IC TPS54331 làm trung tâm. TPS54331 là một bộ giảm áp DC, có thể lấy nguồn đầu vào lên đến 28V, 3A[12]. Mạch ứng dụng từ TPS54331 được lấy từ mục 8.2 của [12].

Nguồn điện áp đầu ra có thể được chỉnh sửa nhờ vào việc chọn 2 điện trở R9 và R10 trong Hình 3-9 với cơ chế cầu phân áp, ta có công thức tính được V_{OUT} như sau:

$$V_{OUT} = V_{SENSE} \cdot \left(\frac{R_9}{R_{10}} + 1 \right)$$

Với V_{SENSE} = 0.8V và V_{OUT} là 3.3V, ta có thể tính được tỉ lệ điện trở R_9 và R_{10}, từ đó, chọn được giá trị 2 điện trở sao cho giá trị đó được sản xuất thông dụng và phải đảm bảo V_{OUT} gần với 3.3V nhất. Ở mạch này, với R_9 = 15kΩ và R_{10} = 4.7kΩ, V_{OUT} ≈ 3.35V. Một mạch buck converter như trên hoạt động dựa trên vòng điều khiển phản hồi. Vòng điều khiển này sẽ lấy V_{OUT} về để so sánh với điện áp tham chiếu V_{SENSE}, từ đó điều chỉnh tín hiệu đầu vào giúp cho V_{OUT} được ổn định hơn. Việc để điện trở R_8 = 0Ω là làm một địa điểm để ngắt vòng điều khiển và tiến hành kiểm tra hoạt động mạch. R_8 được đặt ngay dưới V_{OUT} để khi kiểm tra thì ta có thể kiểm tra ngay đường chạy về của V_{OUT}. Giá trị điện trở 0Ω giúp cho việc chèn thiết bị đo vào dễ hơn mà không cần phải thay đổi cấu trúc mạch.

Do TPS54331 không có cuộn cảm bên trong IC nên ta phải sử dụng một cuộn cảm ngoại để tích trữ năng lượng và giảm điện áp đầu ra của mạch. Cuộn cảm lựa chọn phải có chỉ số dòng điện RMS tối đa và dòng điện bão hoà tối đa lớn hơn dòng RMS ($I_{L(RMS)}$) và dòng tối qua có thể đi qua cuộn cảm ($I_{L(PK)}$) để tránh gây hư hỏng cuộn cảm trong quá trình

hoạt động. Cách tính 2 thông số nói trên đã được đề cập trong [12] *(Note: I_{LPP} là biên độ dòng điện dao động trong cuộn cảm).*

$$I_{LPP} = \frac{V_{OUT}.(V_{IN(MAX)} - V_{OUT})}{V_{IN(MAX)}.L_{OUT}.F_{SW}.0{,}8}$$

$$I_{L(RMS)} = \sqrt{I_{OUT(MAX)}^2 + \frac{1}{12}.I_{LPP}^2}$$

$$I_{L(PK)} = I_{OUT(MAX)} + \frac{I_{LPP}}{2}$$

Sau các quá trình tham khảo thì $I_{L(RMS)}$ = 3.01A và $I_{L(PK)}$ = 3.47A. Theo [12] thì giá trị điện cảm trong cuộn cảm nên nằm trong khoảng 6.8µH đến 47µH, nếu điện cảm càng cao thì dòng AC trong ngõ ra và biên độ áp ngõ ra sẽ nhỏ hơn, dẫn tới nguồn đầu ra sẽ ổn định hơn. Mạch này sẽ sử dụng SWPA8040S100MT có điện cảm là 10µH, thoả được điều kiện được đề ở trên và dễ tìm trong thị trường cũng như có sẵn trong các phần mềm thiết kế mạch điện.

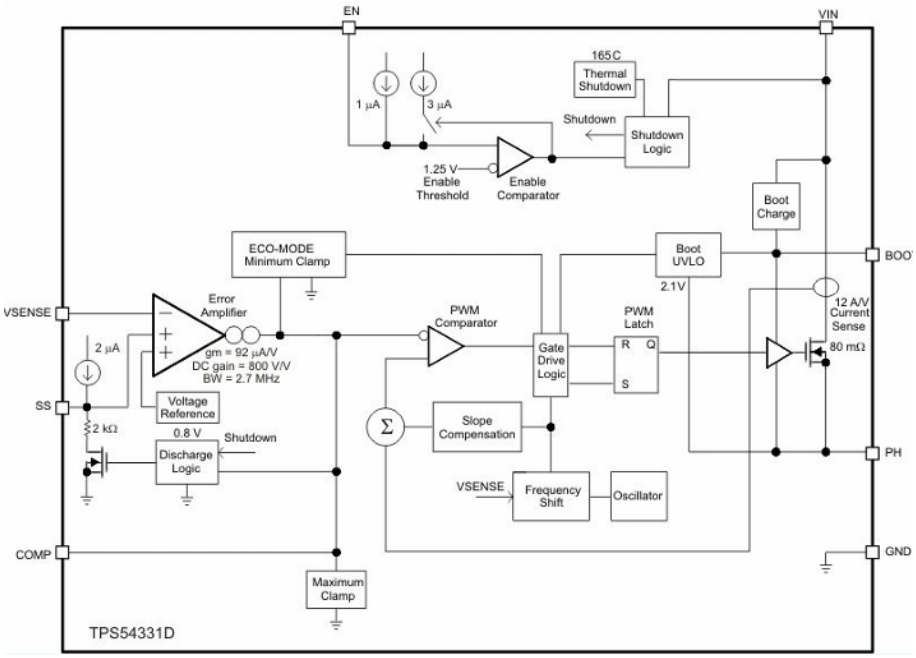
Ở V_{OUT} ngoài cuộn cảm L1 thì còn 2 tụ điện 47pF nữa để giúp lọc những gợn sóng và nhiễu cao tần. 2 tụ này hoạt động gần giống như tụ decoupling khi đem những gợn sóng và nhiễu ở tần số tự cộng hưởng của riêng nó xuống GND. Giá trị điện dung của 2 tụ này không quá quan trọng, tuy nhiên phải chọn loại tụ điện sao cho có thể chịu được những thông số của áp ngõ ra như gai nhọn của áp và dòng hay ESR. Tuy nhiên sẽ có giới hạn cho giá trị điện dung này, nó phụ thuộc chính vào tần số đóng vòng (closed-loop crossover frequency). Tần số đóng vòng là tốc độ phản hồi của vòng điều khiển với các thay đổi ở tải và dòng. Tần số đóng vòng theo [12] phải nhỏ hơn 1/5 so với tần số chuyển mạch. TPS54331 có tần số chuyển mạch là 570kHz cho nên mạch này sẽ chọn tần số đóng vòng là 25kHz. Từ tần số đóng vòng này, ta tìm giá trị điện dung tối thiểu cho 2 tụ này như sau:

$$C_{O(MIN)} = \frac{1}{2\pi.R_O.F_{CO(MAX)}}$$

Theo như mục 8.2.2.8 của [12] thì bắt buộc phải có 1 tụ gốm có điện dung là 0.1µF nằm giữa chân BOOT và PH. Tụ điện này được gọi là **tụ bootstrap**. Ngoài ra, cầm phải có 1 diode nằm giữa chân PH và GND. Trong mạch điều khiển buck converter, MOSFET bên trong TPS54331 nằm giữa nguồn đầu vào và cuộn cảm (MOSFET bên phải trong Hình 3-10) cần một điện áp đủ lớn cho bộ điều khiển cực gate để chuyển sang trạng thái ON. Tuy nhiên,

nguồn điều khiển bên trong IC không thể trực tiếp tạo ra điện áp này nên sẽ cần sự phối hợp giữa tụ bootstrap và diode để giúp duy trì điện áp cho bộ điều khiển cực gate.

Chân BOOT là chân điều khiển hoạt động cực gate của MOSFET bên trong TPS54331 còn chân PH là chân nối thẳng với cực source của MOSFET (xem Hình 3-10). Khi diode ON, chân PH sẽ nối với GND và tụ bootstrap sẽ được nạp từ V_{IN} khiến nó sẽ lưu lượng điện áp gần bằng V_{IN}. Khi MOSFET ON, chân PH sẽ có mức điện áp gần bằng V_{IN}, lúc này điện áp của tụ bootstrap sẽ xả ra bộ điều khiển cực gate MOSFET. Điều này đảm bảo bộ điều khiển cực gate luôn có mức điện áp đủ lớn kể cả PH có gần bằng V_{IN} đi chăng nữa. Diode phải có thông số điện áp ngược lớn hơn điện áp của chân PH và dòng tối đa mà diode chịu được phải lớn hơn dòng tối đa ở ngõ ra. Mạch này sử dụng diode SS34 với điện áp ngược là 40V và dòng điện tối đa chịu được là 3A.



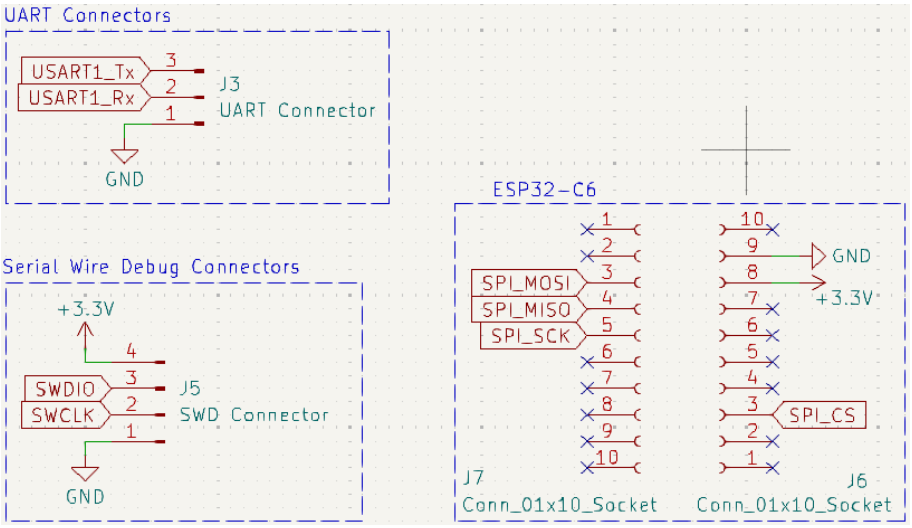
Hình 3-10: Sơ đồ khối của TPS54331

TPS54331 cần một nguồn đầu vào không có quá nhiều gai nhọn và nhiễu nên sẽ cần các tụ decoupling để lọc bớt gai áp và nhiễu. Mạch này sẽ sử dụng 2 tụ gốm 4.7µF và có giá trị điện áp đánh thủng cao hơn mức điện áp đầu vào (12V). Mạch sẽ sử dụng thêm 1 tụ gốm 10nF để lọc nhiễu ở những tần số cao hơn.

TPS54331 sử dụng vòng điều khiển để ổn định tín hiệu đầu ra, tuy nhiên vòn điều khiển có thể hoạt động không ổn định do tác động của các tụ điện, cuộn cảm bên trong TPS54331 và cả những thay đổi của tải. Vì thế các buck converter sử dụng vòng điều khiển phải có cơ chế bù (compensation) cho vòng điều khiển. Bù là cơ chế thiết kế và sử dụng một

số thành phần trong vòng điều khiển để giúp nó ổn định khi bị tác động bởi các linh kiện bên trong IC và điều kiện thay đổi liên tục của tải. Theo như mục 8.2.2.5 của [12] thì có đề cập đến việc TPS54331 cần một mạch bù bên ngoài, kết nối với chân COMP. Mạch bù ở trên sử dụng mạch bù loại II, tức là sử dụng một tụ gốm mắc song song với 1 tụ gốm khác nối tiếp với 1 điện trở.

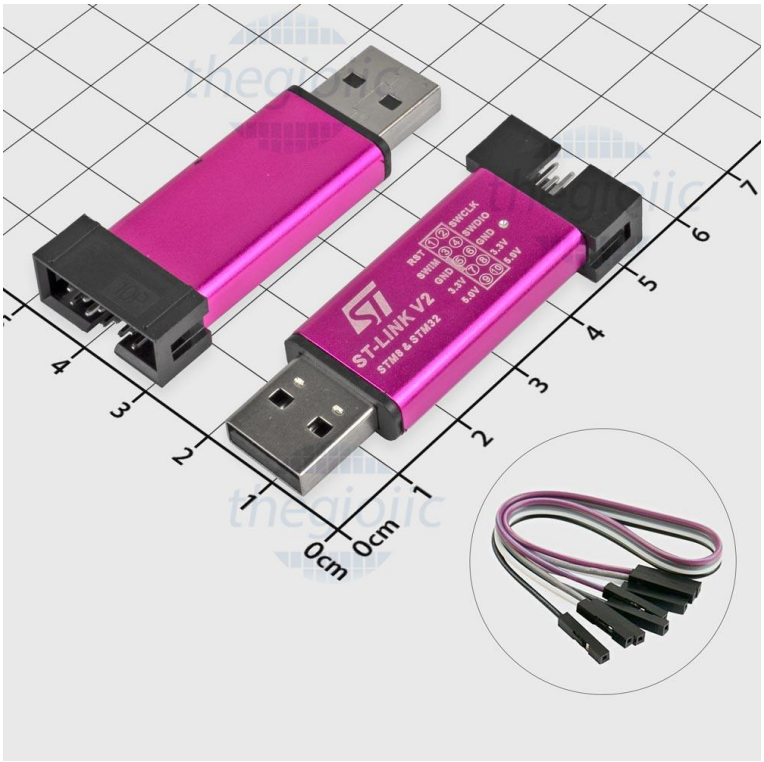
3.2.2.6 Các mạch khác



Hình 3-11: Schematic của các mạch khác của ECU TPMS

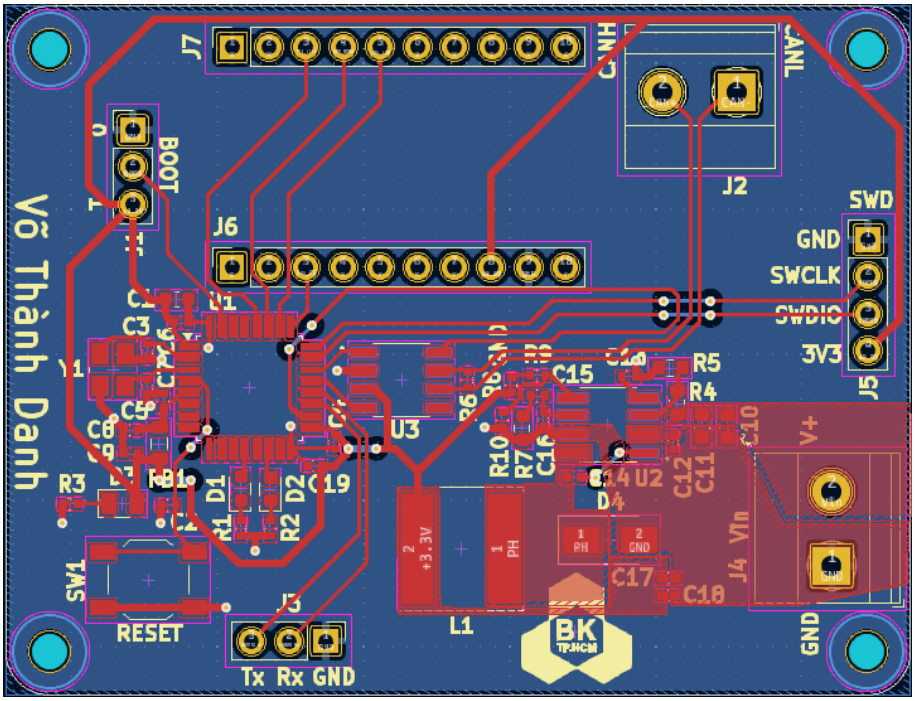
Mạch ECU TPMS sẽ có bao gồm 3 pin header dùng để giao tiếp UART với các ngoại vi bên ngoài. Mạch sử dụng UART để quan sát hoạt động của chương trình và là một lựa chọn để nạp code cho vi điều khiển khi BOOT = 1.

Lựa chọn nạp code chính của mạch này là sử dụng nạp qua giao thức **SWD (Serial Wire Debuug)**. Đây là giao thức giúp cho những mã máy được viết thẳng vào bộ nhớ Flash mà không cần thông qua bootloader. Để có thể nạp code cho vi điều khiển sử dụng SWD, ta cần một mạch nạp (programmer) để giúp các mã máy đi vào Flash và giúp ta có thể sử dụng tính năng debuug của các IDE. Đề tài này sẽ sử dụng mạch nạp ST-Link V2 (xem Hình 3-12). Ngoài ra, mạch sẽ có chỗ để cắm module ESP32-C6 và ra các chân SPI để giao tiếp với STM32F042K6T6.

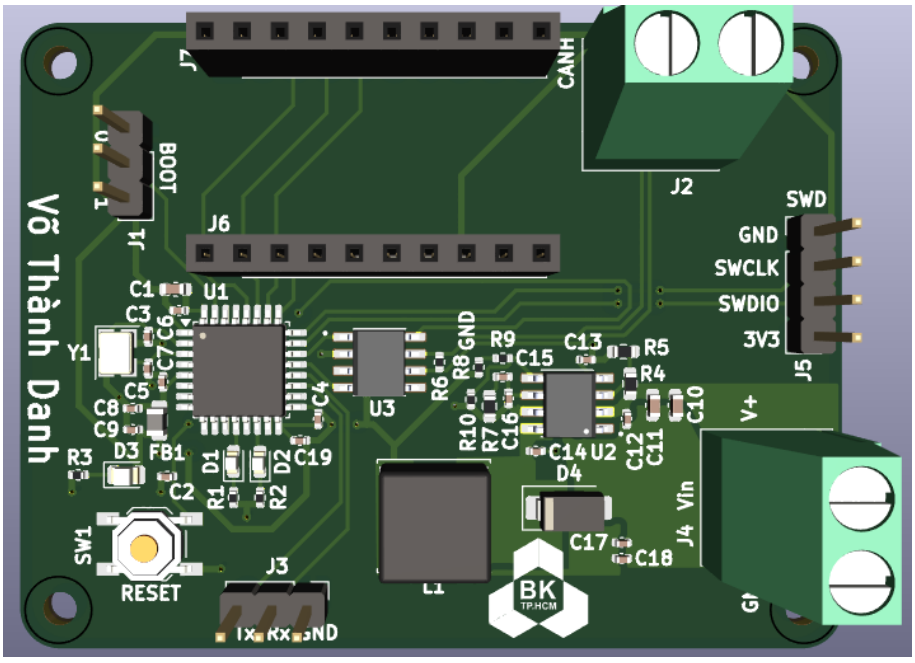


Hình 3-12: Mạch nạp ST-Link V2

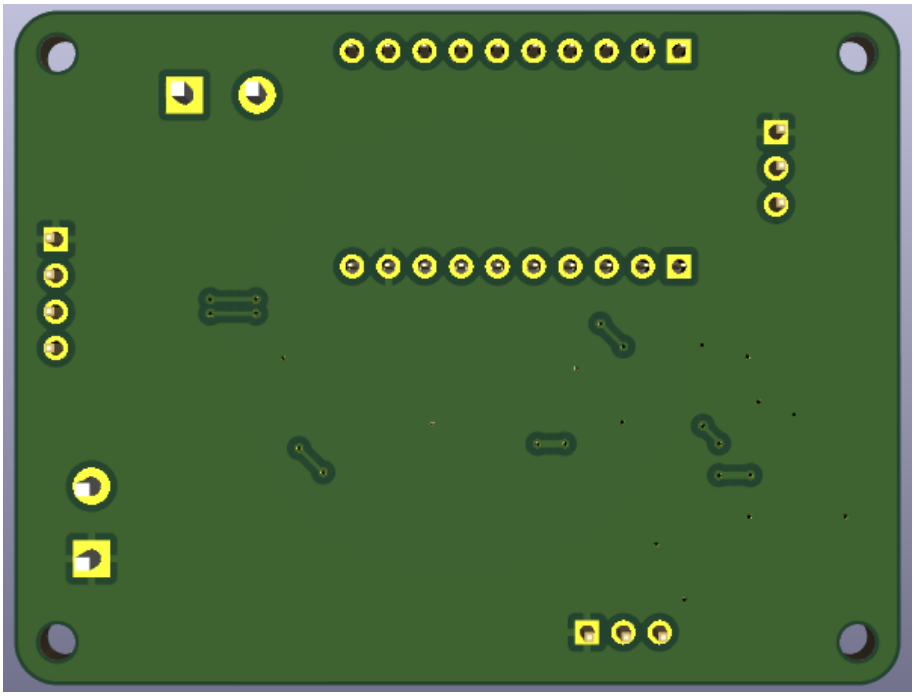
Sau khi hoàn thành schematic của ECU TPMS, đề tài tiến hành vẽ layout cho mạch. Mạch PCB này sẽ gồm 2 lớp, lớp trên sẽ dùng để đi các tín hiệu số và nguồn, lớp dưới sẽ là mảng GND để giảm cảm kháng của mạch. Các đường tín hiệu sẽ và GND sẽ có độ dày là 0.5mm còn đường tín hiệu số sẽ dạy 0.3mm. Các lỗ liên thông giữa 2 lớp mạch sẽ có đường kính tín hiệu là 0.7mm và đường kính lỗ là 0.3mm. Xem layout hoàn chỉnh ở Hình 3-13 và mô hình 3D của mạch ở Hình 3-14 và Hình 3-15.



Hình 3-13: Layout mạch ECU TPMS



Hình 3-14: Mặt trước của PCB ECU TPMS (mô hình 3D)



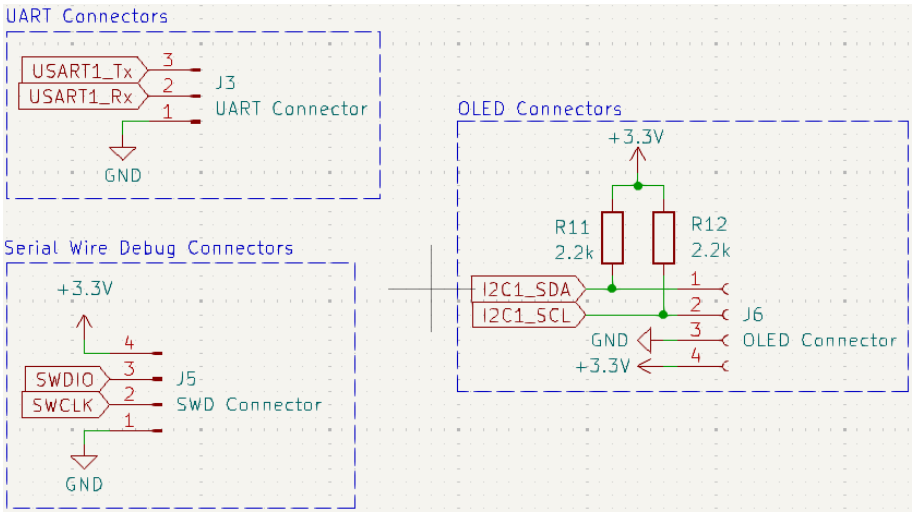
Hình 3-15: Mặt sau của PCB ECU TPMS (mô hình 3D)

3.2.3 ECU màn hình

Mạch ECU TPMS bao gồm nhiều thành phần nhỏ ghép lại với nhau:

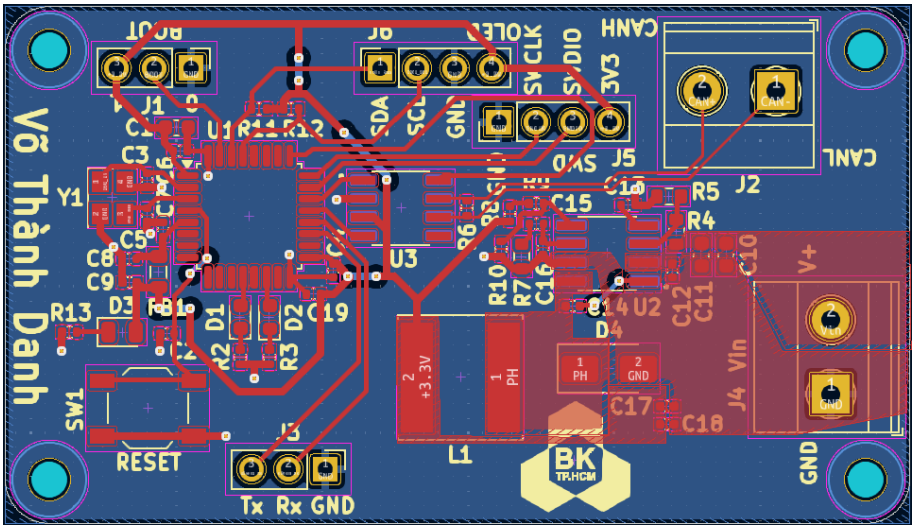
- Mạch vi điều khiển
- Mạch decoupling cho tín hiệu digital và analog
- Mạch hệ thống và cấu hình cho vi điều khiển
- Mạch CAN transceiver
- Mạch nguồn
- Thành phần khác

Ngoài thành phần mạch khác thì tất cả các mạch còn lại đều giống với mạch ECU TPMS. Ở các thành phần mạch khác thì mạch ECU màn hình sẽ bỏ nơi gắn ESP32-C6 và thay vào nơi gắn màn hình OLED 0.96 SSD1306. Những pin header loại cái này sẽ giao tiếp I^2C với vi điều khiển và kèm theo 2 điện trở kéo lên nguồn ở mỗi đường SDA và SCL.

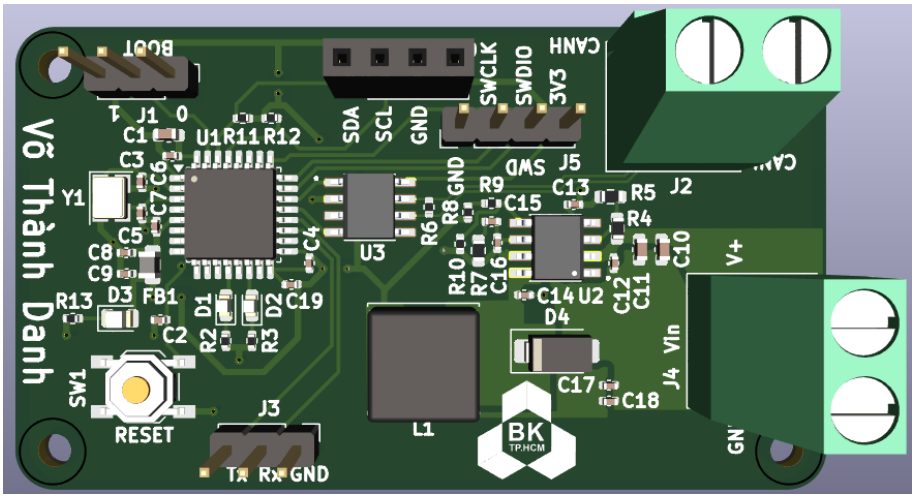


Hình 3-16: Schematic của các mạch khác trong ECU màn hình

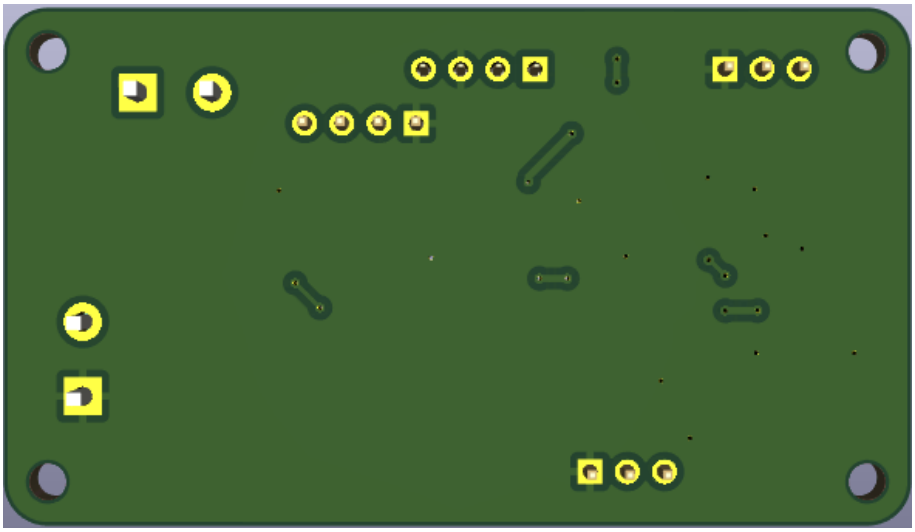
Trong quá trình vẽ layout, chỉ cần bỏ 2 hàng pin header cái của ESP32-C6 và thay bằng 1 hàng pin header cái cho OLED. Tất cả các thành phần khác đều giữ nguyên (có thể di dời vị trí của một vài thành phần sao cho phù hợp với mạch). Xem layout hoàn chỉnh của mạch ở Hình 3-17 và mô hình 3D của PCB ở Hình 3-18 và Hình 3-19.



Hình 3-17: Layout của mạch ECU màn hình



Hình 3-18: Mặt trước của PCB ECU màn hình (mô hình 3D)



Hình 3-19: Mặt sau của PCB ECU màn hình (mô hình 3D)

3.3 Lập trình firmware

Đề tài sẽ phải lập trình cho 2 dòng vi điều khiển hoàn toàn khác nhau là ESP32 và STM32. Đối với ESP32, đề tài sẽ sử dụng ngôn ngữ C++ và theo chuẩn **"Arduino friendly"**, tạm dịch là chuẩn phù hợp với thư viện Arduino. Đối với STM32, đề tài sẽ lập trình theo ngôn ngữ C và sử dụng thư viện **SPL**.

SPL (Standard Peripherals Library) là một gói firmware bao gồm các driver cho tất cả các ngoại vi tiêu chuẩn, dành cho các vi điều khiển dòng STM32[14]. Thư viện SPL là một tập hợp các API, cấu trúc dữ liệu, macro,… phục vụ cho các tính năng của các ngoại vi của STM32. Thư viện SPL chứa đầy đủ các mô tả của các API, cấu trúc dữ liệu, macro cũng như các ví dụ sử dụng chúng. Thư viện giúp chúng ta có thể sử dụng ngoại vi của STM32 mà không cần phải tìm hiểu quá sâu về các tính năng, hoạt động và nguyên lý của ngoại vi, đặc

biệt là các thanh ghi để điều khiển các ngoại vi. Việc sử dụng SPL giúp ta rút ngắn được quy trình làm việc với một ngoại vi mà không đi quá xa so với bản chất của ngoại vi. Trong các định nghĩa của các API, chúng được phát triển lên từ việc cấu hình các thanh ghi liên quan nên việc sử dụng SPL không khiến ta rời quá xa so với cấu trúc thanh ghi, từ đó hiệu suất chương trình cũng được giữ ở mức rất ổn.

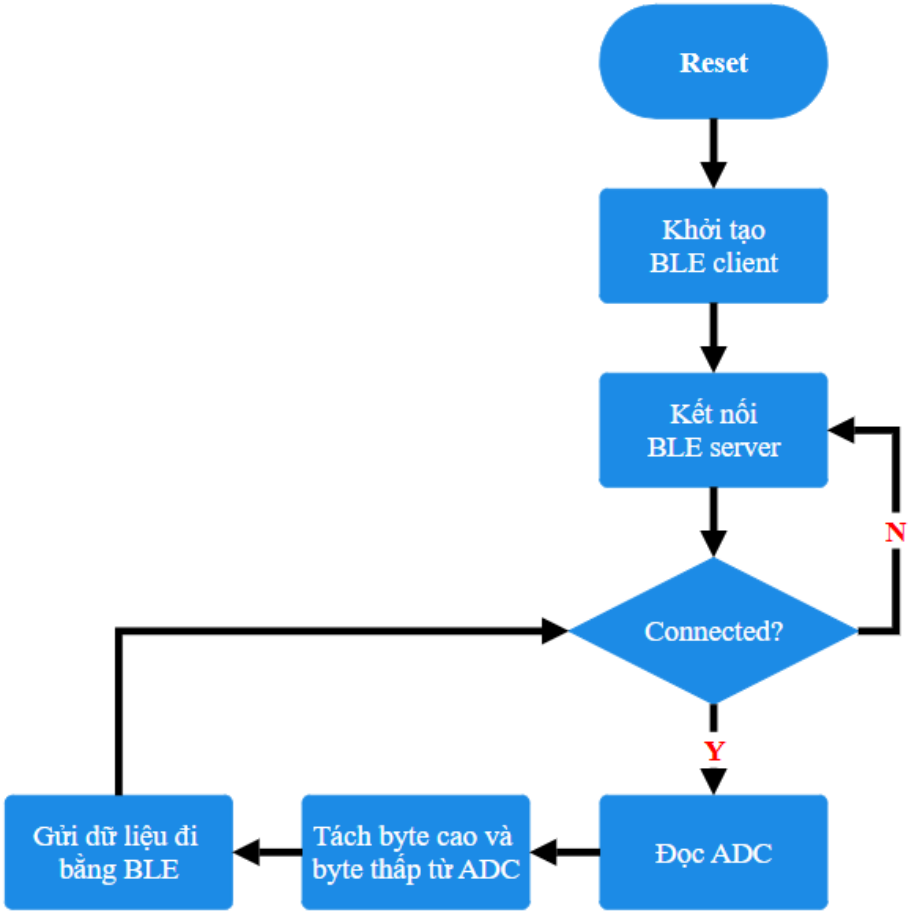
3.3.1 ESP32C6 trong hệ thống cảm biến

ESP32C6 trong hệ thống cảm biến có nhiệm vụ đọc tín hiệu analog từ cảm biến và gửi dữ liệu nhận được cho ECU TPMS thông qua BLE. Vì bộ ADC của ESP32C6 có độ phân giải là 12 bit, nên giá trị mà vi điều khiển đọc được sẽ được chứa trong 2 biến có kiểu dữ liệu 8 bit (uint8_t). Vì còn dư 4 bit trống, đề tài quyết định lắp đầy 4 bit đó bằng tình trạng pin hiện tại của hệ thống cảm biến. Với 4 bit, hệ thống có thể phản hồi 16 mức pin khác nhau trải dài từ 0 – 100% so với tình trạng pin ban đầu. Cụ thể xem Bảng 3-2.

Dữ liệu 4 bit mức pin	Tình trạng pin theo %
0000	0%
0001	6.25%
0010	12.5%
0011	18.75%
0100	25%
0101	31.25%
0110	37.5%
0111	43.75%
1000	50%
1001	56.25%
1010	62.5%
1011	68.75%
1100	75%
1101	81.25%
1110	87.5%
1111	93.75%

Bảng 3-2: Quy đổi giá trị 4 bit tình trạng và % so với tình trạng ban đầu

Tuy nhiên, vì trong TPMS có tận 4 client từ 4 bánh xe nên phải có phương pháp để phân biệt dữ liệu từ các client. BLE có hỗ trợ nhiều phương pháp khác nhau như sử dụng địa chỉ MAC, connection handle, GATT,… Đề tài này sẽ đơn giản là gửi thêm 1 byte dữ liệu là ID của client và đi chung với dữ liệu từ cảm biến XGZP6847A500KPG. Mỗi client sẽ có 1 ID riêng, có thể là từ đặc tính của client hay người phát triển tự đặt. Khi server nhận dữ liệu, nó sẽ đọc byte đầu tiên trong khung dữ liệu để kiểm tra xem dữ liệu này là của client nào, từ đó xử lý chúng và gửi cho STM32.



Hình 3-20: Sơ đồ giải thuật của hệ thống cảm biến

Giải thích: Chương trình trước tiên sẽ cấu hình ESP32C6 để trở thành một BLE client và sẽ tìm một BLE server để kết nối dựa vào địa chỉ MAC của server được cung cấp. Sau khi đã kết nối, client tiến hành đọc giá trị từ bộ ADC, phân tách dữ liệu 12 bit thành 2 byte 8 bit, thêm 4 bit tình trạng pin vào đầu byte cao, thêm byte ID vào đầu khung. Khung dữ liệu gồm 3 byte sẽ được gửi đi qua BLE cho server nhận. Sau mỗi lần gửi dữ liệu đi sẽ kiểm tra kết nối với server để đảm bảo vẫn còn giao tiếp với server. Quá trình sẽ lặp lại từ bước đọc ADC.

3.3.2 ESP32C6 trong ECU TPMS

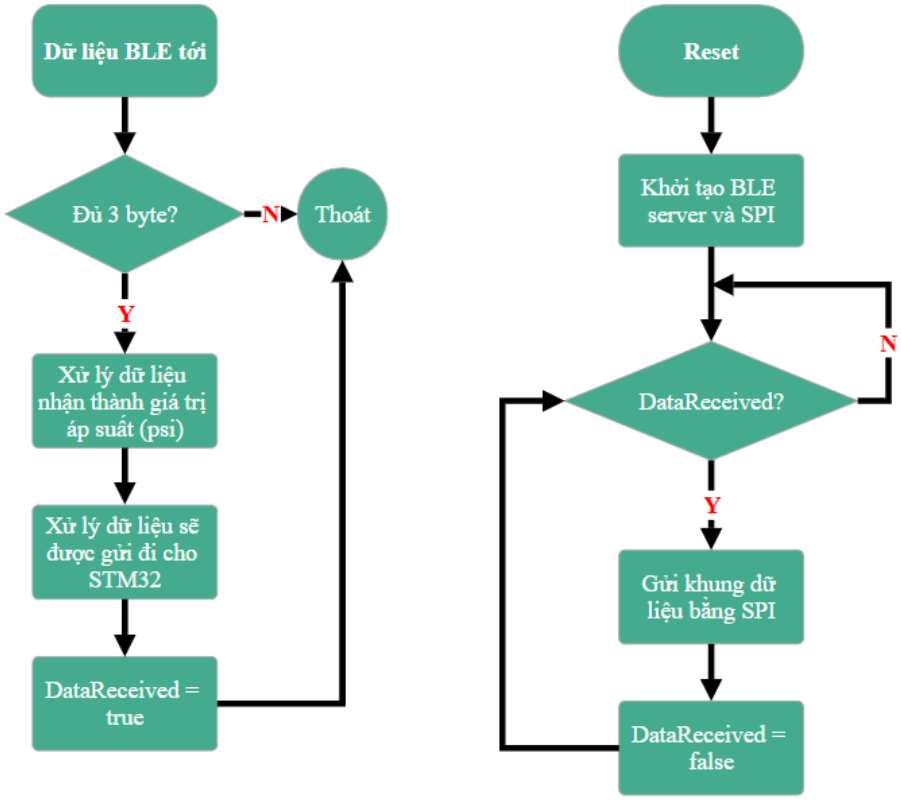
ESP32C6 trong ECU TPMS có nhiệm vụ là BLE server nhận dữ liệu từ các client, xử lý dữ liệu đó để gửi cho STM32 sử dụng SPI. Dữ liệu mà ESP32C6 sẽ gửi cho STM32 sẽ bao gồm 6 byte, cụ thể như sau:

- **Byte 1, 2, 3, 4: Áp suất không khí của 4 lốp xe**
  Hai byte này sẽ chứa áp suất không khí của 4 lốp xe theo đơn vị là psi. Theo thứ từ từ byte 1 đến byte 4 là áp suất của lốp trái trước, phải trước, trái sau, phải sau.
- **Byte 5, 6: Tình trạng pin của cảm biến trong 4 lốp xe**

Byte này chứa phần trăm pin hiện tại của 4 cảm biến của 4 lốp xe so với tình trạng pin ban đầu.

- Byte 4 bao gồm 4 bit đầu sẽ là tình trạng pin của lốp trước trái, 4 bit sau là của lốp trước phải.
- Byte 5 bao gồm 4 bit đầu sẽ là tình trạng pin của lốp sau trái, 4 bit sau là của lốp sau phải.

Việc tổ chức khung dữ liệu giúp cho các dữ liệu được ngăn nắp và dễ dàng xử lý qua nhiều vi điều khiển khác nhau.



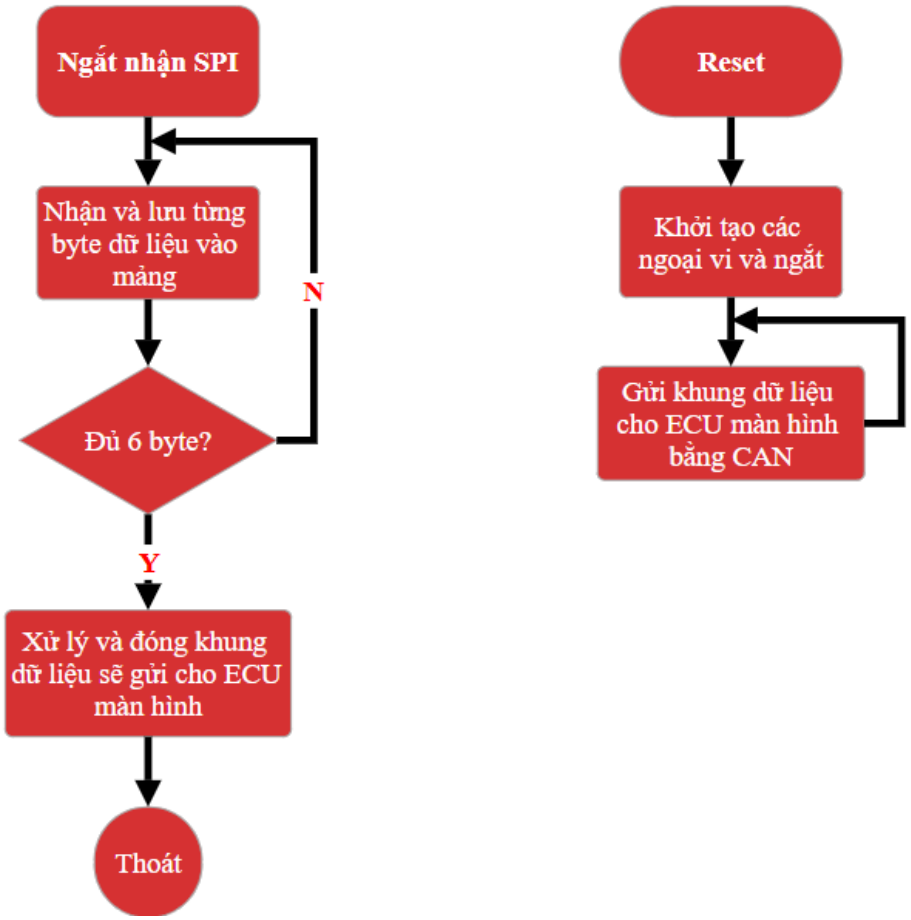
Hình 3-21: Sơ đồ giải thuật của ESP32C6 trong ECU TPMS

Giải thích: Đầu tiên, thiết bị sẽ khởi tạo với vai trò là BLE server, khởi tạo thêm SPI. Hoạt động chính của thiết bị là kiểm tra liên tục 1 biến để lưu tình trạng của dữ liệu từ các client gửi đến. Nếu nhận được dữ liệu từ client, chương trình chính sẽ tạm dừng để thực hiện hàm callback xử lý dữ liệu. Callback trước tiên sẽ kiểm tra dữ liệu nhận được có đủ 3 byte không, nếu không thì thoát khỏi callback để quay lại chương trình chính, nếu đủ 3 byte thì tiến hành xử lý dữ liệu đọc được thành dạng 6 byte khung dữ liệu để gửi cho STM32. Trước khi quay trở lại chương trình chính thì callback sẽ cập nhật trạng thái của dữ liệu nhận được để được gửi đi bằng SPI ở chương trình chính.

3.3.3 STM32 trong TPMS

STM32 trong TPMS làm nhiệm vụ nhận dữ liệu từ ESP32C6 qua SPI, xử lý dữ liệu nhận được, kiểm tra tình trạng hoạt động của hệ thống và đóng khung dữ liệu mới sẽ gửi đi cho ECU màn hình. CAN trong STM32F0 là CAN tiêu chuẩn 2.0B và có thể truyền nhận tối đa 8 byte dữ liệu trong 1 lần gửi. Đề tài sẽ sử dụng 8 byte dữ liệu để chứa các thông tin của hệ thống. Cụ thể như sau:

- **Byte 1: Tình trạng của hệ thống**
  Byte này được sử dụng để giúp phản hồi lại các lỗi mà hệ thống có thể gặp phải. Các lỗi này bao gồm.
  - Bit 7: ECU TPMS có còn nhận được dữ liệu từ hệ thống cảm biến (recessive) hay không (dominant).
  - Bit [6:4]: Trống
  - Bit [3:0]: Cảm biến của 4 bánh xe (bánh trước trái, bánh trước phải, bánh sau trái, bánh sau phải) có truyền dữ liệu cho ECU TPMS (recessive) hay không (domimant).
- **Byte 2, 3, 4, 5: Áp suất không khí của 4 lốp xe**
  Hai byte này sẽ chứa áp suất không khí của 4 lốp xe theo đơn vị là psi. Theo thứ từ từ byte 2 đến byte 5 là áp suất của lốp trái trước, phải trước, trái sau, phải sau.
- **Byte 6, 7: Tình trạng pin của cảm biến trong 4 lốp xe**
  Byte này chứa phần trăm pin hiện tại của 4 cảm biến của 4 lốp xe so với tình trạng pin ban đầu.
  - Byte 6 bao gồm 4 bit đầu sẽ là tình trạng pin của lốp trước trái, 4 bit sau là của lốp trước phải.
  - Byte 7 bao gồm 4 bit đầu sẽ là tình trạng pin của lốp sau trái, 4 bit sau là của lốp sau phải.
- **Byte 8: Báo động cho trường hợp lốp xe có vấn đề**
  Byte này sẽ dùng để phản hồi vấn đề của các lốp xe. 4 bit đầu sẽ báo hiệu các lốp xe có bị quá áp (recessive) hay không (dominant). 4 bit sau sẽ báo hiệu các lốp xe có bị non (recessive) hay không (dominant).

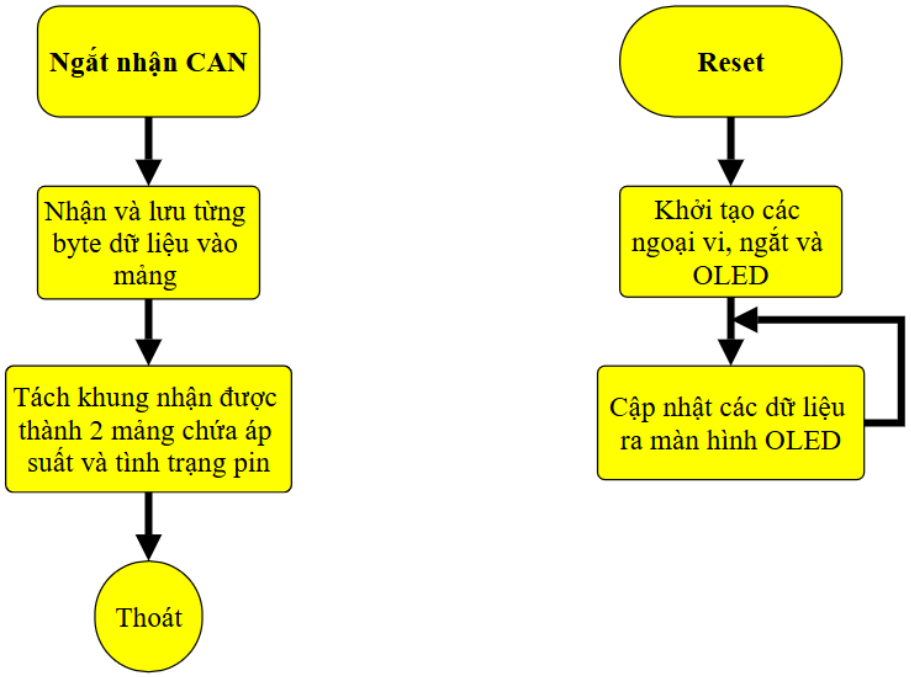


Hình 3-22: Sơ đồ giải thuật của STM32 trong ECU TPMS

Giải thích: Trước tiên, vi điều khiển sẽ cấp xung nhịp, khởi tạo các ngoại vi sẽ sử dụng như GPIO, SPI, CAN và timer. Chương trình chính chỉ đơn giản là sẽ gửi khung dữ liệu đã được xử lý trong ngắt nhận SPI cho ECU màn hình. Mỗi khi có dữ liệu tới thanh ghi nhận dữ liệu của SPI thì sẽ báo ngắt cho CPU để chạy đến ISR ngắt nhận SPI. Trong ISR này, chương trình sẽ nhận lần lượt từng byte được gửi đến cho đến khi đủ 6 byte. Khi đã đủ 6 byte sẽ tiến hành phân tách, xử lý và đóng khung thành mảng dữ liệu gồm 8 byte để gửi đi qua CAN. Khi xử lý xong dữ liệu sẽ reset cờ ngắt, thoát khỏi ISR và thực thi nơi chương trình chính đang làm dở.

3.3.4 STM32 trong ECU màn hình

STM32 trong ECU màn hình có nhiệm vụ nhận khung dữ liệu từ CAN, xử lý nó và hiển thị các dữ liệu cần thiết ra màn hình OLED. Dữ liệu hiển thị ra OLED sẽ được tổ chức trong 4 góc, tương ứng với vị trí của 4 bánh xe trong xe ô tô.



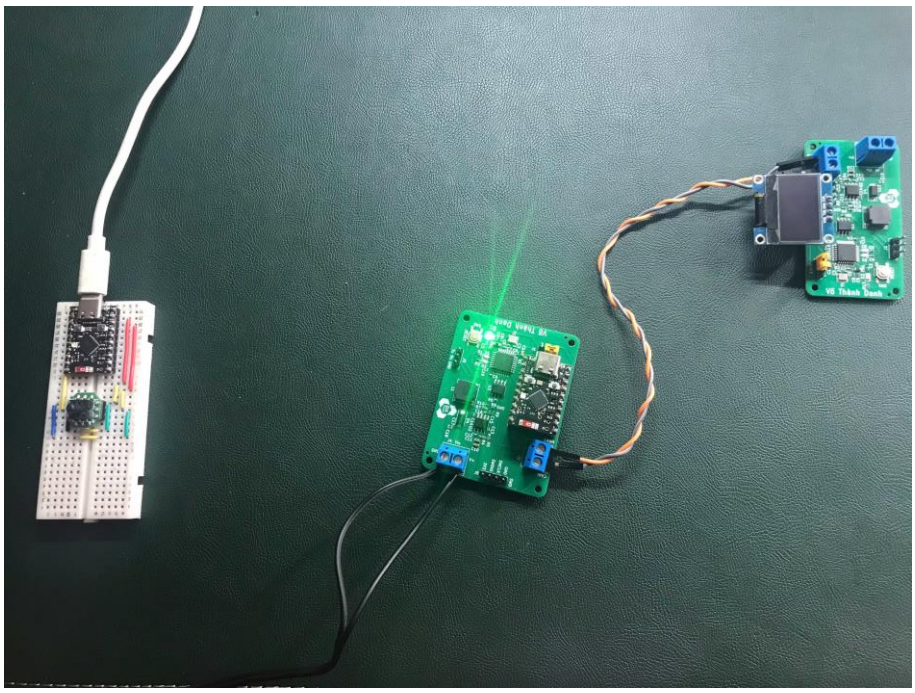
Hình 3-23: Sơ đồ giải thuật của STM32 trong ECU màn hình

Giải thích: Trước tiên, STM32 sẽ cấp xung nhịp và khởi tạo các ngoại vi cần thiết. Ngoài ra, tiến hành khởi tạo OLED bằng cách gửi liên các ID lệnh cần thiết để khởi tạo nó (xem chi tiết ở mục). Chương trình chính chỉ đơn giản là cập nhật các dữ liệu từ 2 mảng chứa áp suất và tình trạng pin của từng cảm biến ra màn hình OLED. Khi có dữ liệu được truyền tới qua CAN, phần cứng CAN sẽ báo ngắt cho CPU để nó thực thi ISR nhận CAN. Trong ISR này, hệ thống sẽ lưu dữ liệu được lưu trong một bộ nhớ FIFO vào 1 mảng. Sau đó sẽ phân tách mảng này thành 2 mảng nhỏ, một mảng chứa áp suất của 4 lốp xe, một mảng chứa tình trạng pin của 4 hệ thống cảm biến. Chương trình chính sẽ sử dụng 2 mảng này để cập nhật dữ liệu ra màn hình. Phân tách mảng hoàn tất thì sẽ thoát khỏi ISR và thực thi tiếp chương trình chính.

4. KẾT QUẢ THỰC HIỆN ĐỀ TÀI

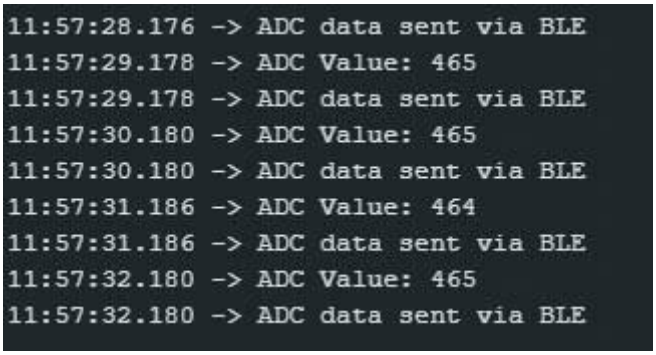
Sau 2.5 tháng làm việc, các kết quả đạt được bao gồm:

- Thiết kế thành công mạch phát triển từ vi điều khiển STM32F042K6T6 và chạy thành công một số ngoại vi cơ bản như GPIO, timer, SPI, UART,…
- Mạch nguồn được thiết kế thành công và vi điều khiển chạy bình thường với nguồn đầu vào là 12V.



Hình 4-1: Tổng quan về phần cứng của đề tài

- Các tính năng cơ bản của hệ thống chạy thành công, tuy nhiên chỉ chạy thành công khi thay 2 mạch phần cứng bằng 2 mạch phát triển Bluepill STM32F103C8T6.



Hình 4-2: Quan sát Serial Monitor khi ESP32C6 trong hệ thống cảm biến đang hoạt động

```
11:57:39.221 -> Sent via SPI: 0x08 0x00 0x00 0x00 0xF0 0xFF
11:57:40.233 -> Data received and processed via BLE:
11:57:40.233 -> Raw ADC: 465
11:57:40.233 -> Scaled Value: 8.23
11:57:40.233 -> Buffer to send: 0x08 0x00 0x00 0x00 0xF0 0xFF
11:57:40.233 -> Sent via SPI: 0x08 0x00 0x00 0x00 0xF0 0xFF
```

Hình 4-3: Quan sát Serial Monitor khi ESP32C6 trong ECU TPMS đang hoạt động

Watch 1

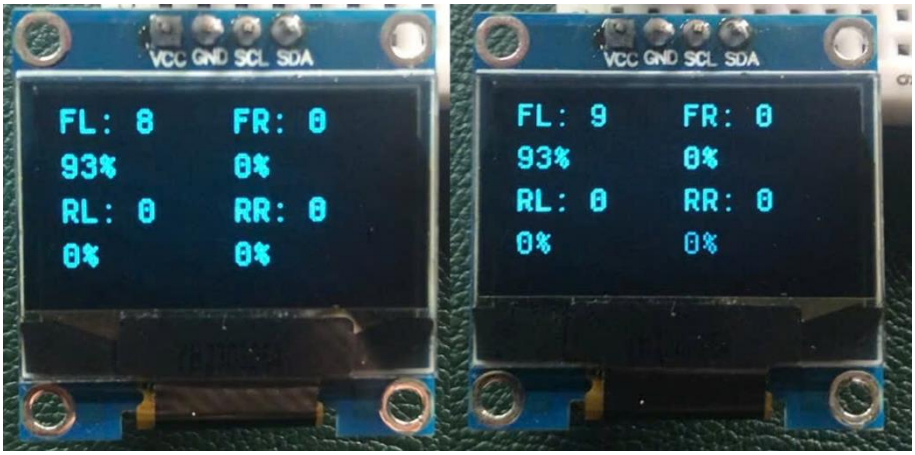
Name	Value	Type
sensorData	0x20000000 sensorDat...	unsigned char[6]
[0]	0x08	unsigned char
[1]	0x00	unsigned char
[2]	0x00	unsigned char
[3]	0x00	unsigned char
[4]	0xF0 'ð'	unsigned char
[5]	0xFF 'ÿ'	unsigned char

Hình 4-4: Quan sát debug dữ liệu STM32 trong ECU TPMS sau khi nhận SPI

Watch 1

Name	Value	Type
CANTransmitData	0x20000006 CANTrans...	unsigned char[8]
[0]	0x8F ''	unsigned char
[1]	0x08	unsigned char
[2]	0x00	unsigned char
[3]	0x00	unsigned char
[4]	0x00	unsigned char
[5]	0xF0 'ð'	unsigned char
[6]	0x00	unsigned char
[7]	0x0F	unsigned char

Hình 4-5: Quan sát debug dữ liệu STM32 trong ECU TPMS đóng khung và sẽ gửi cho ECU màn hình



Hình 4-6: Các kết quả cuối cùng in ra màn hình OLED

Ghi chú: Bên trái các vị trị của lốp xe là giá trị áp suất đo được theo đơn vị psi, bên dưới là tình trạng pin hiện tại của hệ thống cảm biến của từng lốp xe.

- FL: Front Left – lốp xe trái trước
- FR: Front Right – lốp xe phải trước
- RL: Rear Left – lốp xe trái sau
- RR: Rear Right – lốp xe phải sau

Kết quả đạt được của đề tài còn rất nhiều vấn đề và nhược điểm:

- Mạch tự phát triển rất hay bị lỗi Hard fault khi chạy những chương trình hơn mức cơ bản.
- Không thể sử dụng giao thức CAN (yếu tố quan trọng nhất) trên mạch tự phát triển.
- Chưa thiết kế mạch nguồn cho hệ thống cảm biến, dẫn đến chưa thể phản hồi tình trạng pin của hệ thống.
- Mô hình khá đắt so với những gì mà nó làm được.

5. TÀI LIỆU THAM KHẢO

[1] Md Swawibe Ul Alam (09/2018), *"Securing Vehicle Electronic Control Unit (ECU) Communications and Stored Data"*, www.researchgate.net

[2] Tanner Brandt (01/03/2022), *"Direct versus indirect TPMS: How they work and how to diagnose them"*, www.vehicleservicepros.com

[3] Texas Instruments (05/2016), *"Introduction to the Controller Area Network (CAN)"*, www.ti.com

[4] Silicon Labs, *"Bluetooth® LE Fundamentals"*, www.silabs.com

[5] Kevin Cope (22/04/2019), *"What is a Pressure Sensor?"*, www.realpars.com

[6] Analog Devices (03/2009), *"Decoupling Technique"*, www.analog.com

[7] The Organic Chemistry Tutor (20/06/2020), *"Buck Converter"*, www.youtube.com

[8] Tire Pressure, *"Tire Pressure by Vehical"*, tirepressure.com

[9] CFSensor (12/2022), *"XGZP6847A PRESSURE SENSOR"*, cfsensor.com

[10] Espressif (23/08/2024), *"ESP32-C6 Series Datasheet Version 1.2"*, www.espressif.com

[11] Philips Semiconductors (22/10/2003), *"TJA1050 High speed CAN transceiver datasheet"*, www.nxp.com

[12] Texas Intruments (09/2023), *"TPS54331 3-A, 28-V Input, Step Down DC-DC Converter With Eco-mode"*, www.ti.com

[13] STMicroelectronics (10/01/2017), *"STM32F042x4 STM32F042x6 Datasheet - production data"*, www.st.com

[14] STMicroelectronics (12/11/2024), *"Guidelines for oscillator design on STM8AF/AL/S and STM32 MCUs/MPUs"*, www.st.com

[15] STMicroelectronics (18/03/2014), *"STM32F0xx Standard Peripherals library"*, www.st.com

6. PHỤ LỤC

6.1 Điều khiển màn hình OLED SSD1306

Trong quá trình học môn Thiết kế và Phát triển hệ thống IoT, em đã làm một bài báo cáo để thực hiện khởi tao, in ký tự và cập nhật màn hình OLED SSD1306. Xem chi tiết tại link: drive.google.com. (Lưu ý: chỉ có thể mở file sử dụng tài khoản google có miền hcmut.edu.vn).

6.2 Mã nguồn hệ thống

Tất cả mã nguồn của 4 firmware được nạp trong 4 vi điều khiển đều đã được đăng lên GitHub, xem tại: github.com.