BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT

THÀNH PHỐ HỒ CHÍ MINH

KHÓA LUẬN TỐT NGHIỆP

NGÀNH CÔNG NGHỆ KỸ THUẬT Ô TÔ

NGHIÊN CỨU, THIẾT KẾ MÔ HÌNH MÔ PHỎNG HỆ THỐNG HIỂN THỊ THÔNG TIN SỬ DỤNG MẠNG CAN TRÊN Ô TÔ

GVHD: TS. LÊ THANH PHÚC

SVTH: NGUYỄN THỊ NHƯ THÙY

LÊ QUỐC TUẤN

SKL 007950

Tp. Hồ Chí Minh, tháng 01/2021

TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT THÀNH PHỐ HỒ CHÍ MINH

KHOA CƠ KHÍ ĐỘNG LỰC

ĐỒ ÁN TỐT NGHIỆP

NGHIÊN CỨU, THIẾT KẾ MÔ HÌNH MÔ PHỎNG HỆ THỐNG HIỂN THỊ THÔNG TIN SỬ DỤNG MẠNG CAN TRÊN Ô TÔ

SVTH : NGUYỄN THỊ NHƯ THÙY

MSSV: 16145538

SVTH : LÊ QUỐC TUẤN

MSSV: 16145562

GVHD: TS. LÊ THANH PHÚC

Tp. Hồ Chí Minh, tháng 1 năm 2021

TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT THÀNH PHỐ HỒ CHÍ MINH

KHOA CƠ KHÍ ĐỘNG LỰC

ĐỒ ÁN TỐT NGHIỆP

Chuyên ngành: Công nghệ kỹ thuật ô tô

Tên đề tài

NGHIÊN CỨU, THIẾT KẾ MÔ HÌNH MÔ PHỎNG HỆ THỐNG HIỂN THỊ THÔNG TIN SỬ DỤNG MẠNG CAN TRÊN Ô TÔ

SVTH : NGUYỄN THỊ NHƯ THÙY

MSSV: 16145538

SVTH : LÊ QUỐC TUẤN

MSSV: 16145562

GVHD: TS. LÊ THANH PHÚC

Tp. Hồ Chí Minh, tháng 1 năm 2021

TP. Hồ Chí Minh, ngày tháng năm 2021

NHIỆM VỤ ĐỒ ÁN TỐT NGHIỆP

TRƯỜNG ĐH SƯ PHẠM KỸ THUẬT
TP. HỒ CHÍ MINH
KHOA CƠ KHÍ ĐỘNG LỰC

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc Lập – Tự Do – Hạnh Phúc

Họ tên sinh viên: 1. NGUYỄN THỊ NHƯ THÙY MSSV: 16145538
2. LÊ QUỐC TUẤN MSSV: 16145562

Chuyên ngành: Công nghệ kỹ thuật ô tô Mã ngành đào tạo:
Hệ đào tạo: Sư phạm kỹ thuật ô tô Mã hệ đào tạo:
Khóa: 2016 Lớp: 169450

1. Tên đề tài:

Nghiên cứu, thiết kế mô hình mô phỏng hệ thống hiển thị thông tin sử dụng mạng CAN trên ô tô.

2. Nhiệm vụ đề tài:

Nghiên cứu, thiết kế mô hình mô phỏng hệ thống hiển thị thông tin sử dụng mạng CAN trên ô tô.

3. Sản phẩm của đề tài:

Mô hình mô phỏng hệ thống hiển thị thông tin sử dụng CAN trên ô tô và tài liệu về mạng giao tiếp CAN trên ô tô.

4. Ngày giao nhiệm vụ đề tài: 2/11/2020

5. Ngày hoàn thành nhiệm vụ: 25/01/2021

TRƯỞNG BỘ MÔN **CÁN BỘ HƯỚNG DẪN**

TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP. HCM

KHOA CƠ KHÍ ĐỘNG LỰC

Bộ môn Điện – Điện tử ô tô

CỘNG HOÀ XÃ HỘI CHỦ NGHĨA VIỆT NAM

Độc lập - Tự do – Hạnh phúc

PHIẾU NHẬN XÉT ĐỒ ÁN TỐT NGHIỆP

(Dành cho giảng viên hướng dẫn)

Họ và tên sinh viên NGUYỄN THỊ NHƯ THÙY MSSV: 16145538 Hội đồng……

Họ và tên sinh viên LÊ QUỐC TUẤN MSSV: 16145562 Hội đồng……

Tên đề tài: Nghiên cứu, thiết kế mô hình mô phỏng hệ thống hiển thị thông tin sử dụng mạng CAN trên ô tô.

Ngành đào tạo: Công nghệ kỹ thuật ô tô

Họ và tên GV phản biện: (Mã GV) .....

Ý KIẾN NHẬN XÉT

1. Kết cấu, cách thức trình bày ĐATN:

.....

.....

.....

2. Nội dung đồ án:

(Cơ sở lý luận, tính thực tiễn và khả năng ứng dụng của đồ án, các hướng nghiên cứu có thể tiếp tục phát triển)

.....

.....

.....

3. Kết quả đạt được:

.....

.....

.....

4. Những thiếu sót và tồn tại của ĐATN:

.....

.....

.....

.....

5. Câu hỏi:

.....

.....

.....

6. Đánh giá:

TT	Mục đánh giá	Điểm tối đa	Điểm đạt được
1.	***Hình thức và kết cấu ĐATN***	**30**	
	Đúng format với đầy đủ cả hình thức và nội dung của các mục	*10*	
	Mục tiêu, nhiệm vụ, tổng quan của đề tài	*10*	
	Tính cấp thiết của đề tài	*10*	
2.	***Nội dung ĐATN***	**50**	
	Khả năng ứng dụng kiến thức toán học, khoa học và kỹ thuật, khoa học xã hội…	*5*	
	Khả năng thực hiện/phân tích/tổng hợp/đánh giá	*10*	
	Khả năng thiết kế, chế tạo một hệ thống, thành phần, hoặc quy trình đáp ứng yêu cầu đưa ra với những ràng buộc thực tế.	*15*	
	Khả năng cải tiến và phát triển	*15*	
	Khả năng sử dụng công cụ kỹ thuật, phần mềm chuyên ngành…	*5*	

3.	***Đánh giá về khả năng ứng dụng của đề tài***	**10**	
4.	***Sản phẩm cụ thể của ĐATN***	**10**	
	Tổng điểm	**100**	

7. Kết luận:

☐ Được phép bảo vệ

☐ Không được phép bảo vệ

TP.HCM, ngày tháng năm 2021

Giảng viên hướng dẫn

(Ký, ghi rõ họ tên)

TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP. HCM

KHOA CƠ KHÍ ĐỘNG LỰC

Bộ môn Điện – Điện tử ô tô

CỘNG HOÀ XÃ HỘI CHỦ NGHĨA VIỆT NAM

Độc lập - Tự do – Hạnh phúc

PHIẾU NHẬN XÉT ĐỒ ÁN TỐT NGHIỆP

(Dành cho giảng viên phản biện)

Họ và tên sinh viên NGUYỄN THỊ NHƯ THÙY MSSV: 16145538 Hội đồng……

Họ và tên sinh viên LÊ QUỐC TUẤN MSSV: 16145562 Hội đồng……

Tên đề tài: Nghiên cứu, thiết kế mô hình mô phỏng hệ thống hiển thị thông tin sử dụng mạng CAN trên ô tô.

Ngành đào tạo: Công nghệ kỹ thuật ô tô

Họ và tên GV phản biện: (Mã GV) .....

Ý KIẾN NHẬN XÉT

1. Kết cấu, cách thức trình bày ĐATN:

.....

.....

.....

2. Nội dung đồ án:

(Cơ sở lý luận, tính thực tiễn và khả năng ứng dụng của đồ án, các hướng nghiên cứu có thể tiếp tục phát triển)

.....

.....

.....

3. Kết quả đạt được:

.....

.....

.....

4. Những thiếu sót và tồn tại của ĐATN:

.....

.....

.....

.....

5. Câu hỏi:

.....

.....

.....

6. Đánh giá:

TT	Mục đánh giá	Điểm tối đa	Điểm đạt được
1.	***Hình thức và kết cấu ĐATN***	**30**	
	Đúng format với đầy đủ cả hình thức và nội dung của các mục	*10*	
	Mục tiêu, nhiệm vụ, tổng quan của đề tài	*10*	
	Tính cấp thiết của đề tài	*10*	
2.	***Nội dung ĐATN***	**50**	
	Khả năng ứng dụng kiến thức toán học, khoa học và kỹ thuật, khoa học xã hội…	*5*	
	Khả năng thực hiện/phân tích/tổng hợp/đánh giá	*10*	
	Khả năng thiết kế, chế tạo một hệ thống, thành phần, hoặc quy trình đáp ứng yêu cầu đưa ra với những ràng buộc thực tế.	*15*	
	Khả năng cải tiến và phát triển	*15*	
	Khả năng sử dụng công cụ kỹ thuật, phần mềm chuyên ngành…	*5*	

3.	***Đánh giá về khả năng ứng dụng của đề tài***	**10**	
4.	***Sản phẩm cụ thể của ĐATN***	**10**	
	Tổng điểm	**100**	

7. Kết luận:

☐ Được phép bảo vệ

☐ Không được phép bảo vệ

TP.HCM, ngày tháng năm 2021

Giảng viên phản biện

(Ký, ghi rõ họ tên)

TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT

THÀNH PHỐ HỒ CHÍ MINH

KHOA CƠ KHÍ ĐỘNG LỰC

XÁC NHẬN HOÀN THÀNH ĐỒ ÁN

Tên đề tài: NGHIÊN CỨU, THIẾT KẾ MÔ HÌNH MÔ PHỎNG HỆ THỐNG HIỂN THỊ THÔNG TIN SỬ DỤNG MẠNG CAN TRÊN Ô TÔ.

Họ và tên sinh viên: NGUYỄN THỊ NHƯ THÙY MSSV: 16145538

LÊ QUỐC TUẤN MSSV: 16145562

Ngành: Công nghệ kỹ thuật ô tô

Sau khi tiếp thu và điều chỉnh theo góp ý của Giảng viên hướng dẫn, Giảng viên phản biện và các thành viên trong Hội đồng bảo vệ. Đồ án tốt nghiệp đã được hoàn chỉnh đúng theo yêu cầu về nội dung và hình thức.

Chủ tịch Hội đồng: ……..…….……..

Giảng viên hướng dẫn: ……...…….

Giảng viên phản biện: ……..……...……

Tp. Hồ Chí Minh, ngày tháng năm 2021

LỜI CẢM ƠN

Nhóm thực hiện xin chân thành cảm ơn quý thầy khoa Cơ Khí Động Lực, đặc biệt là quý thầy bộ môn Điện tử ô tô trường Đại học Sư phạm kỹ thuật Thành phố Hồ Chí Minh, những người đã dìu dắt nhóm thực hiện trong suốt thời gian học tập. Được giáo dục trong môi trường trang bị những thiết bị mới nhất, được truyền đạt kiến thức từ trong giáo trình đến thực tiễn, đó là những thứ mà nhóm thực hiện có được sau 4.5 năm học tập. Từ những nền tảng kiến thức và hiểu biết vững chắc đó đã giúp nhóm thực hiện hoàn thành đồ án này.

Hơn hết nhóm gửi lời cảm ơn sâu sắc đến TS. Lê Thanh Phúc là người thầy đã tận tình hướng dẫn, chỉ bảo kịp thời, tạo điều kiện, động viên và giúp đỡ nhóm rất nhiều về mặt tinh thần cũng như kiến thức để vượt qua khó khăn trong quá trình thực hiện đồ án này.

Bên cạnh đó nhóm thực hiện cũng xin gửi lời cảm ơn tới gia đình, bạn bè đã hết lòng ủng hộ, giúp đỡ và góp ý cho nhóm trong suốt quá trình thực hiện. Mặc dù đã rất cố gắng và nỗ lực, nhưng do kiến thức hạn chế cũng như thời gian nghiên cứu là có hạn nên những thành quả đạt được không tránh khỏi những thiếu sót. Do đó nhóm thực hiện rất mong nhận được những sự đóng góp, chỉ dạy của quý thầy cô để hoàn thiện đồ án được tốt hơn.

Nhóm thực hiện xin chân thành cảm ơn!

TPHCM, ngày tháng năm 2021

Nhóm sinh viên thực hiện

NGUYỄN THỊ NHƯ THÙY

LÊ QUỐC TUẤN

TÓM TẮT

Ô tô hiện đại có thể có tới 70 bộ điều khiển điện tử (ECU) cho các hệ thống khác nhau. Thông thường, bộ xử lý lớn nhất là bộ điều khiển động cơ. Các hệ thống khác là: hộp số, túi khí, chống bó cứng phanh ABS, kiểm soát hành trình, trợ lực lái điện, hệ thống âm thanh, cửa sổ chỉnh điện, cửa ra vào, điều chỉnh gương, hệ thống pin và sạc lại cho ô tô hybrid/điện, vv…Các hệ thống này có thể trao đổi thông tin liên lạc với nhau. Tiêu chuẩn CAN được tạo ra để đáp ứng nhu cầu này. Ưu điểm chính của CAN là kết nối giữa các hệ thống xe với nhau. Sự kết nối này cho phép thực hiện một loạt các tính năng an toàn, tiết kiệm và tiện lợi.

Chính vì sự phát triển mạnh mẽ của mạng giao tiếp CAN nên nhóm thực hiện quyết định chọn hệ thống hiển thị thông tin để mô phỏng quá trình truyền nhận dữ liệu của mạng CAN. Nghiên cứu và thiết kế mô hình mô phỏng hiển thị thông tin trên ô tô bằng việc giao tiếp giữa Arduino Uno và Raspberry Pi 3 qua MCP 2515 lên màn hình LCD là cách tiếp cận với mạng CAN một cách hiệu quả.

Từ những lí do trên nhóm thực hiện quyết định chọn đề tài *"**Nghiên cứu, thiết kế mô hình mô phỏng hệ thống hiển thị thông tin sử dụng mạng CAN trên ô tô**"* để làm đồ án tốt nghiệp.

MỤC LỤC

LỜI CẢM ƠN i
TÓM TẮT ii
MỤC LỤC iii
DANH MỤC CÁC TỪ VIẾT TẮT VÀ KÝ HIỆU v
DANH MỤC CÁC HÌNH vi
DANH MỤC CÁC BẢNG viii
Chương 1. TỔNG QUAN ĐỀ TÀI 1
1.1. Lịch sử vấn đề nghiên cứu 1
1.1.1. Tình hình nghiên cứu trong nước 1
1.1.2. Tình hình nghiên cứu ngoài nước 1
1.2. Tính cấp thiết của đề tài 1
1.3. Mục tiêu của đề tài 1
1.4. Đối tượng và phạm vi nghiên cứu 2
1.4.1. Đối tượng nghiên cứu 2
1.4.2. Phạm vi nghiên cứu 2
1.5. Phương pháp nghiên cứu 2
1.6. Giới hạn đề tài 2
1.7. Nội dung thực hiện 2
1.8. Bố cục của đồ án 3
Chương 2. CƠ SỞ LÍ THUYẾT 4
2.1. Tổng quan về mạng giao tiếp trên ô tô 4
2.1.1. Mạng giao tiếp ô tô 4
2.1.2. Các loại giao thức giao tiếp trên ô tô 4
2.1.3. Yêu cầu về mạng giao tiếp 5
2.1.4. Các ứng dụng trong ô tô 8
2.2. Tổng quan về mạng CAN 10
2.2.1. Lịch sử phát triển mạng CAN 10
2.2.2. Đặc điểm của mạng CAN 11
2.3. Hệ thống hiển thị thông tin trên ô tô 31
Chương 3. SƠ LƯỢC VỀ PHẦN CỨNG VÀ PHẦN MỀM 34
3.1. Sơ lược về phần mềm sử dụng 34
3.1.1. Giới thiệu về lập trình Python 34

3.1.2. Giới thiệu về Sublime Text 34
3.1.3. Giới thiệu về phần mềm Arduino 34
3.2. Sơ lược về phần cứng 34
3.2.1. Arduino Uno 34
3.2.2. MCP 2515 38
3.2.3. Raspberry Pi 3 39
3.2.4. LM 2596 41
3.2.5. Mạch chuyển đổi HDMI sang LCD 42
3.2.6. Màn hình LCD 42
3.2.7. Biến trở 43
Chương 4. THỰC NGHIỆM MÔ HÌNH HỆ THỐNG HIỂN THỊ THÔNG TIN TRÊN Ô TÔ 44
4.1. Thiết kế đồ họa trên Python 44
4.1.1. Các code cơ bản thiết kế giao diện táp lô 44
4.2. Thiết lập truyền tín hiệu 46
4.2.1. Thiết lập giao tiếp SPI 46
4.2.2. Thiết lập ADC 47
4.2.4. Thiết lập Bit Timing 52
4.2.5. Đọc giá trị biến trở và ghi vào data byte 53
4.3. Thiết kế mô hình thực nghiệm 53
4.3.1. Thiết kế khung mô hình 53
4.3.2. Kết quả mô hình mô phỏng 55
Chương 5. KẾT LUẬN VÀ KIẾN NGHỊ 57
5.1. Kết luận 57
5.1.1. Kết quả đạt được 57
5.1.2. Hạn chế của đề tài 57
5.2. Kiến nghị: 57
TÀI LIỆU THAM KHẢO 58
PHỤ LỤC 1 59
PHỤ LỤC 2 59

DANH MỤC CÁC TỪ VIẾT TẮT VÀ KÝ HIỆU

ECU	Electronic Control Unit
ABS	Anti-Lock Braking System
CAN	Controller Area Network
ECM	Engine Control Module
TCM	Transmission Control Module
BCM	Body Control Module
ESP	Electronic Stability Program
ACC	Adaptive cruise control
SOF	Start of Frame
CRC	Cyclic Redundancy Check
ACK	Acknowledge
EOF	End of Frame
RTR	Remote Transmission Request
IDE	Identifier Extension
ID	Identifier
SRR	Substitute Remote Request
SCK	Serial Clock
MISO	Master Input / Slave Output
MOSI	Master Output / Slave Input
SS	Slave Select
SPI	Serial Peripheral Bus
ADC	Analog to Digital Converter

DANH MỤC CÁC HÌNH

Hình 2.1. Điện áp trên hai dây CAN High và Low tốc độ cao. 12
Hình 2.2. Chuyển đổi tín hiệu CAN H và CAN L tốc độ cao. 13
Hình 2.3. Điện áp trên hai dây High và Low của CAN tốc độ cao khi bị nhiễu.....13
Hình 2.4. Điện áp trên hai dây CAN H và CAN L của CAN tốc độ thấp. 13
Hình 2.5. Chuyển đổi tín hiệu CAN H và CAN L tốc đô thấp.....14
Hình 2.6. Hai dây CAN H và CAN L. 15
Hình 2.7.Cấu trúc của mạng CAN.....16
Hình 2.8. Nguyên lí hoạt động của Gateway. 17
Hình 2.9. Cấu trúc phân lớp của mạng CAN. 18
Hình 2.10. Khung tiêu chuẩn.....19
Hình 2.11. Khung mở rộng.....19
Hình 2. 12. Vùng điều khiển. 21
Hình 2.13. Bảng giá trị DLC. 22
Hình 2.14. Vùng báo nhận. 23
Hình 2.15. Vùng kết thúc. 24
Hình 2.16. Khung lỗi. 25
Hình 2.17.Khung lỗi. 25
Hình 2. 18. Khung lỗi. 26
Hình 2.19. Khung quá tải. 26
Hình 2.20. Khung quá tải. 27
Hình 2.21. Sơ đồ nhận CAN message.. 27
Hình 2.22. Sơ đồ truyền CAN messages. 28
Hình 2.23. Đèn tín hiệu trên ô tô. 32
Hình 2.24. Tín hiệu Analog. 33
Hình 2.25. Tín hiệu Digital. 33
Hình 3.1. Arduino Uno. 35
Hình 3.2. ATmega 328. 36
Hình 3.3. Các chân của ATMega 328. 37
Hình 3.4. MCP 2515. 39
Hình 3.5. Raspberry Pi 3. 41

Hình 3.6. Sơ đồ các chân của Raspberry Pi 3. 41
Hình 3.7. Mạch giảm áp LM 2596. 42
Hình 3.8. Mạch chuyển đổi HDMI sang LCD. 42
Hình 3.9. Màn hìn LCD. 42
Hình 3.10. Biến trở. 43
Hình 4.1. Kết quả thiết kế giao diện hiển thị. 46
Hình 4.2. Code thiết lập giao tiếp SPI. 46
Hình 4.3. Thanh ghi DDRB. 46
Hình 4.4. Thanh ghi SPCR. 46
Hình 4.5. Thanh ghi ADMUX. 47
Hình 4.6. Điện áp tham chiếu. 48
Hình 4.7. Chế độ hoạt động ADC. 48
Hình 4.8. Thanh ghi ADCSRA. 48
Hình 4.9. Hệ số chia. 49
Hình 4.10. Code thiết lập ADC. 50
Hình 4.11. Thiết lập chế độ Configuration. 50
Hình 4.12. SPI instruction set của MCP 2515. 51
Hình 4.13. Địa chỉ thanh ghi trong MCP 2515. 51
Hình 4.14. Code thiết lập Bit timing MCP2515. 52
Hình 4.15. Chế độ Normal của MCP 2515. 52
Hình 4.16. Code xuất ID. 53
Hình 4.17. Code xuất số data byte. 53
Hình 4.18. Code đọc giá trị biến trở. 53
Hình 4.19. Khung mô hình hệ thống hiển thị thông tin trên Solidwork 54
Hình 4.20. Thiết kế khung mô hình trên Solidwork. 54
Hình 4.21. Thiết kế mô hình trên Solidwork. 54
Hình 4.22. Sơ đồ mô phỏng hệ thống hiển thị hệ thống hiển thị thông tin. 55
Hình 4.23. Mô hình thực tế. 56
Hình 4.24. Màn hình hiển thị thông tin. 56

DANH MỤC CÁC BẢNG

Bảng 2.1. Các loại giao thức truyền thông hay sử dụng trên ô tô. 5

Bảng 2.2. Giá trị điện áp trên hai dây CAN H và CAN L.....14

Bảng 4.1. Bảng chế độ MCP 2515. 50

Bảng 4.2. Bảng nối dây giữa các chân của Arduino Uno với MCP 2515 và biến trở.....55

Bảng 4.3. Bảng nối dây giữa các chân của Raspberry Pi 3 với MCP 2515.....55

Chương 1. TỔNG QUAN ĐỀ TÀI

1.1. Lịch sử vấn đề nghiên cứu:

1.1.1. Tình hình nghiên cứu trong nước:

Mạng truyền thông giao thức CAN không phải là một lĩnh vực kỹ thuật hoàn toàn mới mà thực chất là các công nghệ được kế thừa, chắt lọc và phát triển từ kỹ thuật truyền thống cho phù hợp với các yêu cầu trong nền công nghệ ô tô. Từ hơn một thập kỷ nay, mạng truyền thông đã trở nên không thể thiếu được trong các hệ thống điều khiển và giám sát hiện đại. Song, thực tế người vận hành thường gặp phải hàng loạt các vấn đề trong việc tìm kiếm nguồn tham khảo đáng tin cậy, chuyên sâu và đầy đủ như cách thức hoạt động cụ thể của một mạng CAN, các chuẩn khi giao tiếp, ứng dụng khác nhau trong xe.

1.1.2. Tình hình nghiên cứu ngoài nước:

Mạng CAN đã phát triển hầu như hoàn thiện, ứng dụng khác nhau trong nhiều lĩnh vực. Tuy nhiên lý thuyết CAN ô tô trừu tượng và chuyên ngành nên cũng cần nghiên cứu và thực hành.

1.2. Tính cấp thiết của đề tài:

Ngày nay, hệ thống mạng CAN ngày càng được trang bị trên nhiều xe ô tô hiện đại và đặc biệt ở Mỹ, hầu như tất cả các xe ô tô bán ra tại Mỹ đều được trang bị mạng giao tiếp này. Tuy nhiên, việc chú trọng đầu tư và giảng dạy về mạng CAN ở trường Đại học sư phạm kĩ thuật TPHCM chỉ dừng ở mức giới thiệu, lý thuyết chưa có mô hình mô phỏng quá trình giao tiếp của mạng CAN trong ô tô.

Từ những vấn đề trên nhóm thực hiện thấy cần thiết phải nghiên cứu và thiết kế một mô hình mô phỏng hệ thống hiển thị thông tin sử dụng CAN thu nhỏ để dễ quan sát quá trình giao tiếp truyền dữ liệu lên táp lô. Với việc vận dụng ngôn ngữ lập trình Python cho Raspberry Pi 3 để thiết kế ra giao diện hiển thị thông tin đơn giản và lập trình Arduino Uno, MCP 2515 giúp cung cấp phần nào kiến thức về mạng giao tiếp CAN trên ô tô.

1.3. Mục tiêu của đề tài:

- Hiểu và nắm vững kiến thức lý thuyết về giao thức CAN.
- Tìm hiểu ngôn ngữ lập trình Python.
- Lập trình hệ thống mạng giao tiếp CAN giữa các Arduino Uno, Raspberry Pi 3,

MCP 2515.

- Thiết kế giao diện hiển thị thông tin trên ô tô.
- Hoàn thiện mô hình hệ thống hiển thị thông tin.

1.4. Đối tượng và phạm vi nghiên cứu:

1.4.1. Đối tượng nghiên cứu:

Đề tài "Nghiên cứu, thiết kế mô hình mô phỏng hệ thống hiển thị thông tin sử dụng mạng CAN trên ô tô" được thực hiện với các đối tượng nghiên cứu sau:

- Các dạng tín hiệu hiển thị thông tin.
- Cách thức giao tiếp SPI.
- Sự truyền nhận tín hiệu mạng CAN trên nhiều node.
- Lập trình trên Arduino Uno, Raspberry Pi 3.
- Thiết kế mô hình.

1.4.2. Phạm vi nghiên cứu:

- Lập trình giao tiếp giữa Raspberry Pi 3 với Arduino Uno và giữa các MCP 2515 với nhau.
- Giả lập tín hiệu, lập trình truyền nhận dữ liệu điều khiển đồng hồ hiển thị.
- Đưa ra nhận xét và đề xuất hướng phát triển của đề tài.

1.5. Phương pháp nghiên cứu:

- Sử dụng các nguồn tài liệu trên Internet để tìm hiểu về Python, cách giao tiếp giao thức CAN giữa Raspberry Pi 3 và Arduino Uno.
- Ứng dụng Python để thiết kế ra màn hình giao diện.
- Ứng dụng kỹ thuật cơ khí hàn các board mạch.

1.6. Giới hạn đề tài:

- Thời gian thực hiện đề tài không nhiều nên người thực hiện chỉ tập trung vào việc truyền nhận dữ liệu của mạng CAN trên vài node.
- Thiết kế giao diện táp lô còn đơn giản.

1.7. Nội dung thực hiện:

- Các dạng tín hiệu hiển thị thông tin.
- Lý thuyết về truyền nhận mạng CAN.
- Cách thức hoạt động giao tiếp SPI.
- Lập trình Arduino Uno, Atmega 328, Raspberry Pi 3.

- Thiết kế hệ thống.

1.8. Bố cục của đồ án:

- Chương 1: Tổng quan.
- Chương 2: Cơ sở lý thuyết.
- Chương 3: Sơ lược về phần cứng và phần mềm.
- Chương 4: Thực nghiệm mô hình hệ thống hiển thị thông tin trên ô tô.
- Chương 5: Kết luận và kiến nghị.

Chương 2. CƠ SỞ LÍ THUYẾT

2.1. Tổng quan về mạng giao tiếp trên ô tô:

2.1.1. Mạng giao tiếp ô tô:

Mạng giao tiếp trên ô tô nói riêng hay mạng giao tiếp trên các phương tiện giao thông nói chung (Vehicle Bus) là một hệ thống các hộp điều khiển trên cùng một xe bao gồm các loại như ECM, TCM, BCM, ABS… Khi hoạt động, chúng có thể giao tiếp trao đổi thông tin qua lại với nhau mà không cần phải tăng thêm số lượng dây dẫn. Nhằm tối ưu cho việc điều khiển và hạn chế dây dẫn, ngày nay tất cả các phương tiện từ ô tô con, xe tải, đầu kéo, máy công trình, máy bay, xe quân sự, thậm chí cả xe máy cũng đều sử dụng mạng giao tiếp.

Ngày nay, với sự phát triển không ngừng của công nghệ ô tô, một chiếc xe châu Âu bình thường trung bình có khoảng 30 hộp điều khiển khác nhau chưa kể đến một chiếc xe sang thì con số hộp điều khiển có thể lên đến hàng trăm hộp. Ngay cả trên hệ thống điều khiển ghế ngồi, điều khiển mở cốp, điều khiển âm thanh đều có một hộp điều khiển riêng. Tất cả các hộp này được kết nối với nhau để lấy tín hiệu của nhau. Ví dụ: Hộp điều khiển hộp số TCM sẽ lấy tín hiệu tốc độ động cơ, tín hiệu bàn đạp ga để điều khiển sang số; tín hiệu tốc độ xe hiển thị trên đồng hồ táp lô lấy từ hộp điều khiển hộp số hoặc lấy từ hộp ABS cho thấy chúng có mối mối quan hệ với nhau.

Với sự ra đời của các hệ thống điều khiển điện tử đã làm giảm số lượng dây điện trên ô tô xuống mức tối đa, làm giảm chi phí sản xuất, tối ưu hóa không gian cho xe, tăng độ chính xác cho những khâu xử lý và đặc biệt rất ít lỗi trên hệ thống so với khi không sử dụng hệ thống điều khiển điện tử.

Khi nhu cầu về an toàn, tiện lợi và độ chính xác cao, đòi hỏi phải có một sự liên kết giữa tất cả các hộp điều khiển lại với nhau để có thể trao đổi thông tin giữa các hộp điều khiển một cách nhanh chóng, kịp thời và chính xác. Và mạng giao tiếp ô tô ngày nay là giải pháp tối ưu để giải quyết các vấn đề trên.

2.1.2. Các loại giao thức giao tiếp trên ô tô:

Các loại giao thức phổ biến trên ô tô hiện nay:

- CAN (Controller Area Network)
- Ethernet (K-line)
- Flexray

- LIN (Local Interconnect Network)

- Media Oriented Systems Transport (MOST)

Ngoài ra, còn có các loại giao thức khác như là J1979, J1850, ISO 9141,…Tuy nhiên, trong tương lai các xe tự hành hoàn toàn thì khối lượng dữ liệu và tốc độ truyền dẫn phải rất lớn. Để duy trì an toàn cho người lái, tốc độ các mạng như CAN và FlexRay đang áp dụng nhiều nhất hiện nay cũng không thể đáp ứng được. Do đó, mạng giao tiếp có thể được sử dụng phải kể đến Ethernet TSN (Time-Sensitive Networking), Ethernet AVB (Audio Video Bridging).

2.1.3. Yêu cầu về mạng giao tiếp:

2.1.3.1. Tốc độ truyền dữ liệu:

Là đơn vị cho biết khối lượng thông tin được truyền trong một đơn vị thời gian. Đơn vị truyền dữ liệu nhỏ nhất sẽ là bit và tốc độ truyền dữ liệu thường được chỉ định là bit/giây. Ngoài tên gọi "data transfer rate" thì còn có một số các tên gọi khác cùng biểu thị ý nghĩa tương tự như: transfer rate, data rate, bit rate hay baud rate. Tùy thuộc vào từng ứng dụng cụ thể mà ta sẽ có các yêu cầu khác nhau về tốc độ truyền. Tốc độ càng nhanh thì vấn đề về độ tin cậy, hạn chế lỗi càng gây ra nhiều áp lực cho quá trình phát triển đồng thời giá thành sẽ cao. Tốc độ càng chậm thì tồn tại vấn đề về mật độ băng thông và độ trễ truyền tin. Ngoài ra, tốc độ truyền dữ liệu còn phụ thuộc vào chiều dài đường truyền.

Bảng 2.1. Các loại giao thức truyền thông hay sử dụng trên ô tô.

Giao Thức	CAN-C High-speed CAN	CAN-B Low-speed CAN	LIN	MOST	FlexRay
Định nghĩa	Controller Area Network	Controller Area Network	Local Interconnect Network	Local Interconnect Network	Propnetary Name
Loại Bus	Bus thông thường	Bus thông thường	Bus thông thường	Bus quang học	Bus thông thường và bus quang học
Phạm vi ứng dụng	Truyền lực và hộp số	Thoải mái/tiện nghi	Thoải mái /tiện nghi	Đa phương tiện và infortainment	Tất cả các phạm vi

Các ứng dụng	Điều khiển động cơ, hệ thống mạng điều khiển truyền lực và ABS/ESP	Hệ thống mạng điện tử thân xe và thoải mái và tiện nghi	Chi phí thấp mở rộng của CAN bus cho các ứng dụng đơn giản trong phạm vi điện tử thoải mái và tiện nghi	Điều khiển truyền lực,hộp số, thông tin audio và video	Hệ thống sử dụng cho các ứng dụng liên quan tính an toàn và đơn giản
Cấu trúc liên kết	Tuyến bus	Tuyến bus	Tuyến bus	Cấu trúc không gian vòng	Cấu trúc không gian hình sao
Tốc độ truyền dữ liệu	125 kbit/s đến 1Mbit/s	Tối đa 125 kbit/s	Tối đa 20kbit/s	Tối đa 22.5Mbit/s	Tiêu biểu 10Mbit/s Tối đa 20Mbit/s
Số nút tối đa	10	24	16	64	Về mặt lý thiết lên đến 2048. Tối đa 22 trên passive bus/sao
Cơ cấu điều khiển	Dựa theo tác động	Dựa theo tác động	Dựa theo theo gian	Dựa theo thời gian và tác động	Dựa theo thời gian và tác động
Sự triển khai	Tất cả các xe	Tất cả các xe	Tất cả các xe	Dòng xe cao cấp của các nhà sản xuất xe châu Âu	Ứng dụng thí điểm
Sự phân loại SAE	Lớp C	Lớp B	Lớp A	Phương tiện di động	Drive by wire

2.1.3.2. Khả năng chống nhiễu:

Yêu cầu này chính là vấn đề về độ nhiễu khi truyền thông tin. Theo một cách lý tưởng nhất thì dữ liệu được truyền đi sẽ hoàn toàn không bị nhiễu hay mất mát, sai sót dữ liệu. Trong thực tế, môi trường làm việc của mạng trên ô tô là vô cùng phức tạp và chịu rất nhiều ảnh hưởng điện từ đến từ động cơ cũng như các hệ thống điện khác nên việc hoàn toàn không bị nhiễu là vấn đề không thể đạt được. Do đó, việc giảm ảnh hưởng của

sự nhiễu đến vấn đề truyền tin xuống mức thấp nhất vẫn là mối quan tâm hàng đầu, mức độ kháng nhiễu của hệ thống còn phụ thuộc vào mức độ an toàn cũng như mục đích mà hệ thống đó hướng đến.

Hiện nay có rất nhiều cách thức khác nhau để tạo nên khả năng kháng nhiễu cho hệ thống, có thể từ phần cứng hay tích hợp ngay trong phần mềm hoặc tích hợp vào bên trong cơ chế vận hành của các giao thức truyền tin. Có thể kể ra một số đại diện về phương pháp kháng nhiễu như sau: đôi dây cáp xoắn vào nhau, bit chẵn lẻ, checksum.

2.1.3.3. Khả năng đáp ứng thời gian thực:

Đây là một đặc tính có yêu cầu khắt khe hơn về mặt thời gian (thời gian truyền tin và thời gian đáp ứng), trong đó độ sai lệch là cực kì nhỏ. Yêu cầu về thời gian thực chỉ xuất hiện ở một vài hệ thống đặc biệt và thường còn đi kèm với yêu cầu về độ tin cậy.

Thời gian thực tương đối: Hệ thống sẽ tuân theo thời gian phản hồi được yêu cầu, nhưng đôi khi có thể bị vượt quá mà không gây ra bất kì ảnh hưởng nghiêm trọng nào (ví dụ như sự giật ảnh khi truyền hình ảnh, livestream,..).

Thời gian thực tuyệt đối: Các hệ thống áp đặt mức độ này thường là các hệ thống liên quan đến đặc tính an toàn trên ô tô. Đối với các hệ thống này, chỉ cần một chút sai sót trong quá trình truyền tin cũng như phản hồi thông tin thì kết quả sinh ra sẽ bị ảnh hưởng, gây ra sai sót và kết quả tính toán sẽ không thể sử dụng được. Từ đó, gây ra các vấn đề nghiêm trọng, đặc biệt là đối với các hệ thống an toàn và có khả năng gây nguy hiểm đến tính mạng người sử dụng.

2.1.3.4. Số node tham gia vào mạng:

Việc giới hạn số lượng node tối đa trong một mạng mang lại nhiều ý nghĩa đặc biệt:

- Ý nghĩa đầu tiên chính là góp phần giải quyết tình trạng độ trễ truyền tin. Xác suất về sự tranh chấp giữa các tin nhắn truyền cùng một thời điểm sẽ giảm khi tổng số lượng các node tham gia vào mạng giảm.

- Ý nghĩa tiếp theo đến từ việc một mạng nội bộ trên xe không thể (hoặc rất khó) sử dụng cùng một dạng giao thức cho toàn bộ mạng được (do các vấn đề về kĩ thuật, giá thành cũng như độ phù hợp đối với mục đích sử dụng).

- Ý nghĩa cuối cùng là tạo điều kiện thuận lợi nhất để các node thể hiện khả năng của mình một cách hiệu quả, dễ quản lý hơn cũng như dễ dàng nghiên cứu phát triển hơn.

Thông thường, để đưa ra được số node tối đa cho một giao thức bất kì, người ta thường dựa vào mục đích mà giao thức đó hướng đến nhằm suy ra khoảng độ trễ truyền tin có thể chấp nhận được cùng với tốc độ truyền của giao thức đó.

2.1.4. Các ứng dụng trong ô tô:

Hiện nay, mạng nội bộ trên ô tô hiện nay được phân chia thành 4 nhóm chính dựa trên chức năng và đặc tính của chúng:

- Hệ thống khung gầm
- Hệ thống truyền lực
- Hệ thống an toàn tiện nghi
- Hệ thống đa phương tiện

Tùy mỗi hệ thống mà có những vùng ứng dụng riêng. Có 3 ứng dụng cơ bản trên ô tô hiện nay:

2.1.4.1. Ứng dụng thời gian thực:

Xu hướng hiện tại của ngành công nghiệp ô tô chính là dần thay thế các hệ thống cơ khí hay cơ - thủy lực kiểu cũ bằng các hệ thống cơ - điện tử được điều khiển bằng máy tính.

Ví dụ như đối với các hệ thống như ABS hay X-by-wire, các yêu cầu của chúng đặt lên các hệ thống điều khiển điện tử là cực cao và gần như không cho phép bất cứ sai sót hay độ trễ nào có thể xảy ra, chính vì lẽ đó mà các giao thức sử dụng cho các hệ thống này phải không ngừng hoàn thiện và cần được thay thế bởi các giao thức mới hơn, hiệu quả hơn.

Khi càng có nhiều các hệ thống cơ - thủy lực được thay thế bằng các hệ thống cơ - điện tử thì sẽ có càng nhiều máy tính điều khiển. Do các cơ cấu chấp hành đa số nằm trong khoang động cơ nên các máy tính này cũng phải được đặt ở vị trí tương tự. Mà ta đã biết khoang động cơ chính là một trong những vùng chịu nhiễu điện từ trường lớn nhất trên xe. Do đó, tạo áp lực rất lớn đến các cơ chế chống nhiễu nhằm duy trì tính ổn định trong truyền thông tin.

Các hệ thống ứng dụng thời gian thực trên ô tô:

- Hệ thống điều khiển động cơ diesel (EDC).
- Điều khiển truyền lực.
- Điều khiển lực động của xe (hệ thống cân bằng điện tử - ESP)

- Các hệ thống điều khiển khung gầm ô tô (điều khiển thân xe, hệ thống phanh chống hãm - ABS).

- Các hệ thống hỗ trợ (điều khiển hành trình lái - ACC).

2.1.4.2. Ứng dụng đa dẫn:

Đây là mảng ứng dụng gần như phức tạp và có nhiều node nhất trên ô tô. Các node trong mảng này không chỉ là các máy tính điều khiển mà còn là các nút bấm, các công tắc, đèn báo và thường được bố trí theo cụm rải rác khắp xe.

Có thể kể ra một vài hệ thống điển hình như sau:

- Hệ thống hiển thị.
- Đèn chiếu sáng.
- Các thiết bị cảnh báo chống trộm.
- Hệ thống điều hòa.
- Điều chỉnh gương và ghế ngồi.
- Mô-đun cửa (bộ cửa sổ xe tự động, điều chỉnh gương cửa).
- Các gạt nước.
- Điều chỉnh đèn pha.

Đối với các hệ thống thuộc nhóm này thì yêu cầu về tính thời gian thực vẫn có, nhưng không quá khắt khe và thường không yêu cầu tốc độ cao mà chỉ quan tâm đến số lượng nút cho phép trong một mạng.

2.1.4.3. Ứng dụng đa phương tiện:

Các ứng dụng đa phương tiện trên xe có thể liệt kê như sau:

- Hệ thống âm thanh.
- Hệ thống lái.
- Các hệ thống giải trí.
- Điện thoại.
- Hệ thống video.
- Tín hiệu giọng nói.
- E-mai, internet.
- Camera sau.

Thiết kế của dạng mạng này thường tập trung vào một khu vực trung tâm với màn hình hiển thị và các đơn vị điều khiển tương ứng.

2.2. Tổng quan về mạng CAN:

2.2.1. Lịch sử phát triển mạng CAN:

CAN là một giao thức giao tiếp, hỗ trợ mạnh cho những hệ thống điều khiển theo thời gian thực với độ ổn định, bảo mật và khả năng chống nhiễu cực kì tốt.

CAN được Bosch GmbH phát triển từ năm 1983, sau đó đã chính thức ra mắt vào năm 1986 và được công nhận bởi SAE hiệp hội các kĩ sư ô tô Mỹ, có trụ sở đặt tại Detroit Michigan.

Vào những năm đầu tiên sau khi ra mắt, Intel và Philips là 02 nhà sản xuất đầu tiên sản xuất chip xử lý cho CAN (1987) và Mercedes-Benz W140 là chiếc ô tô thương mại đầu tiên được trang bị CAN. Ngày nay, gần như toàn bộ các dòng ô tô hiện đại đều có hỗ trợ CAN và hầu như tất các các nhà sản xuất chip lớn trên thế giới đều sản xuất ra các dòng chip có tích hợp CAN như Siemens, Motorola, NEO, Infineon, Mitsubishi, TI…

Ngoài nền công nghiệp ô tô, CAN còn được ứng dụng rộng rãi trong các ngành công nghiệp tự động hóa, đóng tàu, tàu ngầm, nông nghiệp, y khoa...nhờ vào các ưu điểm về độ tin cậy của mình.

Từ lúc giới thiệu lần đầu đến hiện tại, Bosch đã cho ra rất nhiều phiên bản khác nhau cho CAN, nhưng đại ý có thể tóm lại đơn giản như sau:

- CAN 1.0
- CAN 2.0

+ CAN 2.0 A : CAN tiêu chuẩn, 11-bit ID.

+ CAN 2.0 B tiêu chuẩn: CAN tiêu chuẩn, 11-bit ID.

+ CAN 2.0 B mở rộng: CAN mở rộng, 29-bit ID.

Vào năm 1993, hiệp hội ISO đã phát hành tài liệu tiêu chuẩn hóa cho CAN thông qua ISO 11898 (Cần lưu ý rằng tiêu chuẩn của ISO không phải là toàn bộ các đặc tính của CAN mà Bosch qui định).

ISO 11898-1: CAN lớp liên kết – dữ liệu_ CAN tốc độ cao.

ISO 11898-2: CAN lớp vật lý_ CAN tốc độ cao.

ISO 11898-3: CAN lớp vật lý_ CAN tốc độ thấp.

Theo sau ISO 11898, còn rất nhiều phiên bản khác nhằm tiêu chuẩn hóa CAN tính đến thời điểm hiện tại.

2.2.2. Đặc điểm của mạng CAN:

2.2.2.1. Một số khái niệm:

2.2.2.1.1. Node:

Là thành phần độc lập có thể xử lý truyền nhận dữ liệu trên bus CAN. Một node là một module có hệ thống xử lý cơ bản, thường có 3 thành phần là vi điều khiển MCU, chip điều khiển CAN (CAN controller) và chip thu-phát (CAN transceiver). Mạng CAN gồm nhiều node, mỗi node có thể giao tiếp với bất kì node nào khác trong mạng CAN nhờ việc truyền đi và nhận các gói dữ liệu gọi là message. Một node có thể nhận nhiều loại message.

2.2.2.1.2. Message:

Là gói dữ liệu được truyền trong CAN. Mỗi message được gán cho một ID. Tùy theo mức độ ưu tiên của message mà có một ID khác nhau. Một message có thể có nhiều node nhận. Dữ liệu trong message được truyền đi dưới dạng các khung (khung dữ liệu, khung yêu cầu dữ liệu, khung báo lỗi và khung báo quá tải).

2.2.2.2. Tốc độ truyền dữ liệu:

2.2.2.2.1. Truyền tốc độ cao:

CAN tốc độ cao (CAN High Speed): được định nghĩa trong tiêu chuẩn ISO 11898-2 và hoạt động với tốc độ 125kbit/s đến 1Mbit/s và được sử dụng cho mạng lưới của các hệ thống sau:

- Hệ thống kiểm soát động cơ (Motronic cho động cơ xăng và động cơ diesel điều khiển điện tử)
- Điều khiển hệ thống truyền lực bằng điện tử.
- Các hệ thống cân bằng xe (ESP).
- Các hệ thống hỗ trợ (điều khiển hành trình lái - ACC).

2.2.2.2.2. Truyền tốc độ thấp:

CAN tốc độ thấp (CAN Low Speed): được định nghĩa trong tiêu chuẩn ISO 11898-2 và hoạt động với tốc độ 5 đến 125kbit/s và được sử dụng cho mạng lưới của các hệ thống sau:

- Điều khiển hệ thống máy điều hòa
- Điều chỉnh ghế ngồi.
- Điều chỉnh gương.

- Hệ thống đèn.
- Hệ thống hiển thị
- Chống trộm.
- Gạt nước.

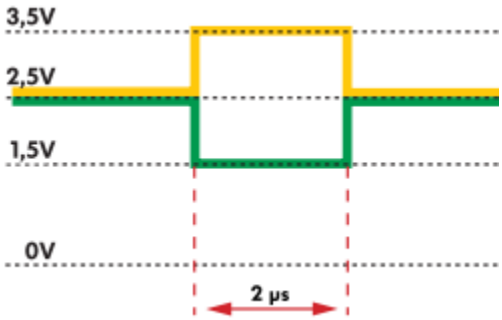
2.2.2.3. Giá trị của Bus CAN:

Bus có thể có một trong hai giá trị logic bổ sung: 'trội' hoặc 'lặn'. Trong quá trình truyền đồng thời các bit 'trội' và 'lặn', giá trị bus kết quả sẽ là 'trội'. Mức 'trội' sẽ được biểu thị bằng '0' và mức 'lặn' bằng '1'.

Giá trị trội: là giá trị điện áp của bus được chủ động thay đổi bởi nút CAN muốn truyền tín hiệu. Giá trị tương ứng là 0.

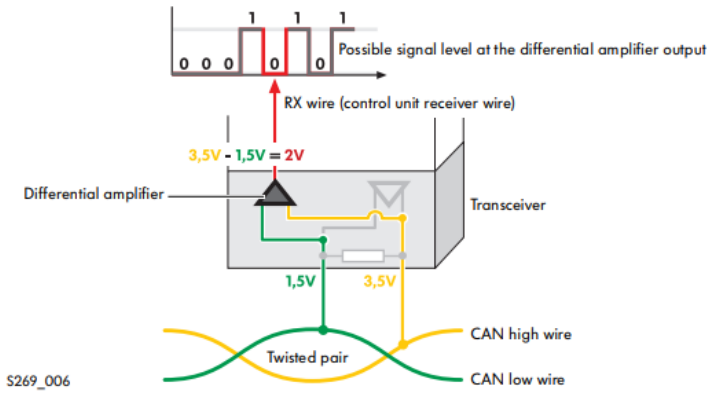
Giá trị lặn: là giá trị điện áp của bus bị động trả về giá trị mặc định bởi điện trở cuối. Giá trị lặn chỉ xảy ra khi không có bất kì nút CAN nào muốn truyền tín hiệu. Giá trị tương ứng là 1.

Đối với CAN tốc độ cao và CAN tốc độ thấp, giá trị trội và lặn là hoàn toàn khác nhau, nhưng cách định nghĩa vẫn giống nhau.



Hình 2.1. Điện áp trên hai dây CAN High và Low tốc độ cao.

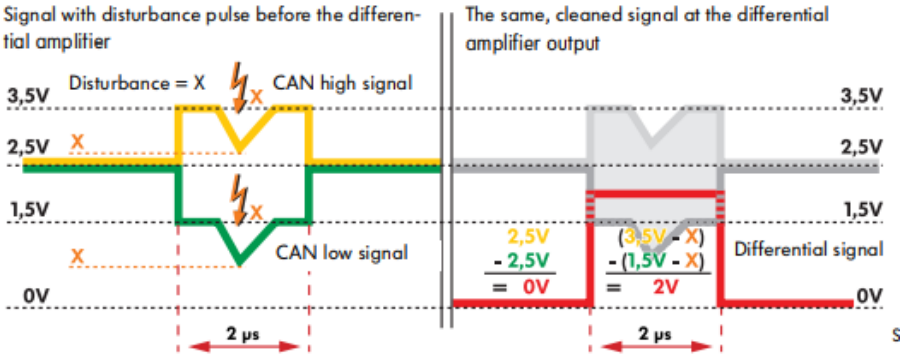
Ở trạng thái lặn, cả hai dây CAN đều có cùng cài đặt mức tín hiệu điện áp 2.5V. Ở trạng thái trội, điện áp của dây CAN H tăng lên theo một giá trị định trước (+1V). Đồng thời, điện áp của dây CAN L cũng giảm theo (- 1V). Điều này dẫn đến sự gia tăng điện áp của dây CAN H lên 3.5V (2.5V + 1V = 3.5V) ở trạng thái trội. Điện áp trong dây CAN L giảm xuống cực đại 1,5V (2,5V - 1V = 1,5V) ở trạng thái trội. Do đó, sự khác biệt điện áp giữa CAN H và CAN L trong trạng thái lặn là 0V và trong trạng thái trội là 2V. [1]



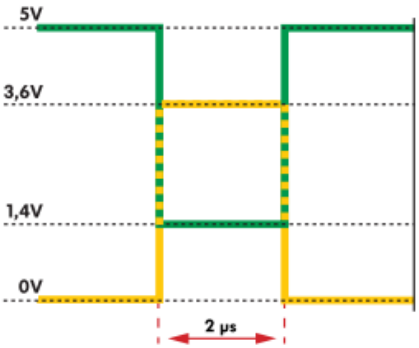
Hình 2.2. Chuyển đổi tín hiệu CAN H và CAN L tốc độ cao.

Nằm trong bộ thu phát là một bộ nhận. Bộ nhận này là bộ khuếch đại, có nhiệm vụ đánh giá các tín hiệu đầu vào từ hai dây CAN H và CAN L. Sau đó, truyền các tín hiệu đã chuyển đổi (3.5V – 1.5V = 2V) đến vùng nhận CAN của thiết bị điều khiển. Tín hiệu này gọi là điện áp đầu ra của bộ khuếch đại.

Vì hai dây CAN được xoắn vào nhau, do đó trong trường hợp bị nhiễu thì sẽ tác động như nhau đến cả hai dây.



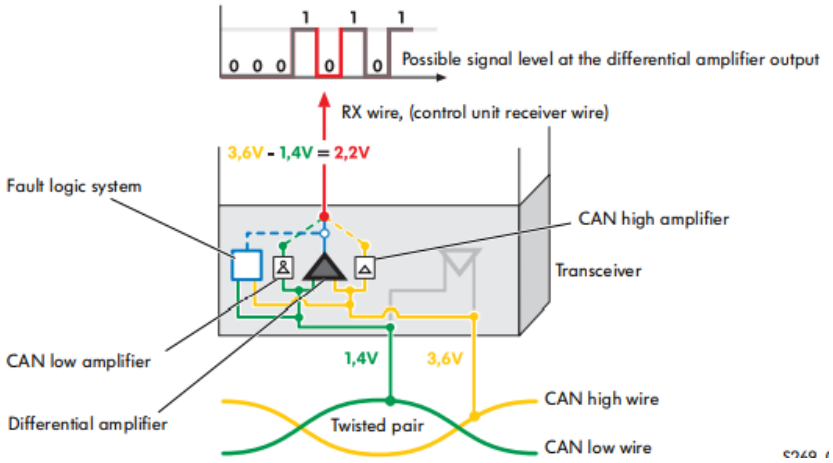
Hình 2.3. Điện áp trên hai dây High và Low của CAN tốc độ cao khi bị nhiễu.



Hình 2.4. Điện áp trên hai dây CAN H và CAN L của CAN tốc độ thấp.

Ở trạng thái lặn, điện áp của dây CAN H được cài đặt ở mức 0V, trong khi đó điện áp của dây CAN L ở mức 5V. Ở trạng thái trội, điện áp của dây CAN H tăng lên theo một

giá trị định trước (+3.6V). Đồng thời, điện áp của dây CAN L cũng giảm theo (- 3.6V). Điều này dẫn đến sự gia tăng điện áp của dây CAN H lên 3.6V (0V+ 3.6V = 3.6V) ở trạng thái trội. Điện áp trong dây CAN L giảm xuống cực đại 1,4V (5V – 3.6V = 1,4V) ở trạng thái trội. Do đó, sự khác biệt điện áp giữa CAN H và CAN L trong trạng thái lặn là 5V và trong trạng thái trội 2.2V.



Hình 2.5. Chuyển đổi tín hiệu CAN H và CAN L tốc đô thấp.

Tương tự như CAN tốc độ cao, CAN tốc độ thấp cũng có bộ nhận nằm trong bộ thu phát. Bộ nhận này là bộ khuếch đại, có nhiệm vụ đánh giá các tín hiệu đầu vào từ hai dây CAN H và CAN L. Sau đó, truyền các tín hiệu đã chuyển đổi (3.6V – 1.4V = 2.2V) đến vùng nhận CAN của thiết bị điều khiển. Tín hiệu này gọi là điện áp đầu ra của bộ khuếch đại.

Bảng 2.2. Giá trị điện áp trên hai dây CAN H và CAN L.

	CAN tốc độ cao		CAN tốc độ thấp	
	CAN H	CAN L	CAN H	CAN L
Tốc độ	125kb/s đến 1 Mb/s		125kb/s	
Trạng thái trội 0	3.5 V	1.5 V	3.6 V	1.4 V
Trạng thái lặn 1	2.5 V	2.5 V	0 V	5 V
Mức điện áp	5 V		5 V	

2.2.2.4. Cơ chế giao tiếp:

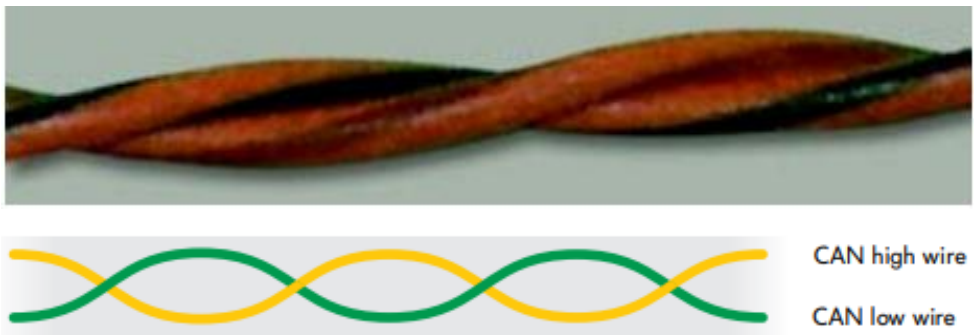
Đặc trưng của CAN là phương pháp định địa chỉ và giao tiếp hướng đối tượng, trong khi hầu hết các hệ thống bus khác đều giao tiếp dựa vào địa chỉ các trạm. Mỗi thông tin trao đổi trong mạng được coi như một đối tượng, được gán một số mã ID.

Thông tin được gửi đi trên bus theo kiểu truyền thông báo với độ dài khác nhau. Các thông báo không chỉ gửi tới một địa chỉ nhất định mà bất cứ trạm nào cũng có thể nhận theo nhu cầu. Nội dung mỗi thông báo được các trạm phân biệt qua một mã ID. Mã ID không nói lên địa chỉ đích của thông báo, mà chỉ biểu diễn ý nghĩa của dữ liệu trong tin nhắn. Vì thế, mỗi trạm trên mạng có thể tự quyết định tiếp nhận và xử lý tin nhắn hay không tiếp nhận tin nhắn qua phương thức lọc tin nhắn. Cũng nhờ sử dụng phương thức lọc tin nhắn, nhiều trạm có thể đồng thời cùng nhận một tin nhắn và có các phản ứng khác nhau. Một trạm có thể yêu cầu một trạm khác gửi dữ liệu bằng cách gửi một khung yêu cầu. Trạm có khả năng cung cấp nội dung thông tin đó sẽ gửi trả lại một khung chứa dữ liệu có cùng mã ID với khung yêu cầu. Cùng với tính năng đơn giản, cơ chế giao tiếp hướng đối tượng ở CAN còn mang lại tính linh hoạt và tính nhất quán dữ liệu của hệ thống. Một trạm CAN không cần biết thông tin cấu hình hệ thống (ví dụ địa chỉ trạm). Nên việc bổ sung hay bỏ đi một trạm trong mạng không đòi hỏi bất cứ một sự thay đổi nào về phần cứng hay phần mềm ở các trạm khác. Trong mạng CAN, có thể chắc chắn rằng một tin nhắn hoặc được tất cả các trạm quan tâm tiếp nhận đồng thời, hoặc không được trạm nào tiếp nhận. Tính nhất quán dữ liệu được đảm bảo qua các phương pháp gửi đồng loạt và xử lý lỗi.

2.2.2.5. Cấu trúc của mạng CAN:

Bus CAN bao gồm 3 thành phần chính [1]:

- Dây cáp gồm hai dây riêng biệt được gọi là CAN H (CAN High) và CAN L (CAN Low) xoắn vào nhau:

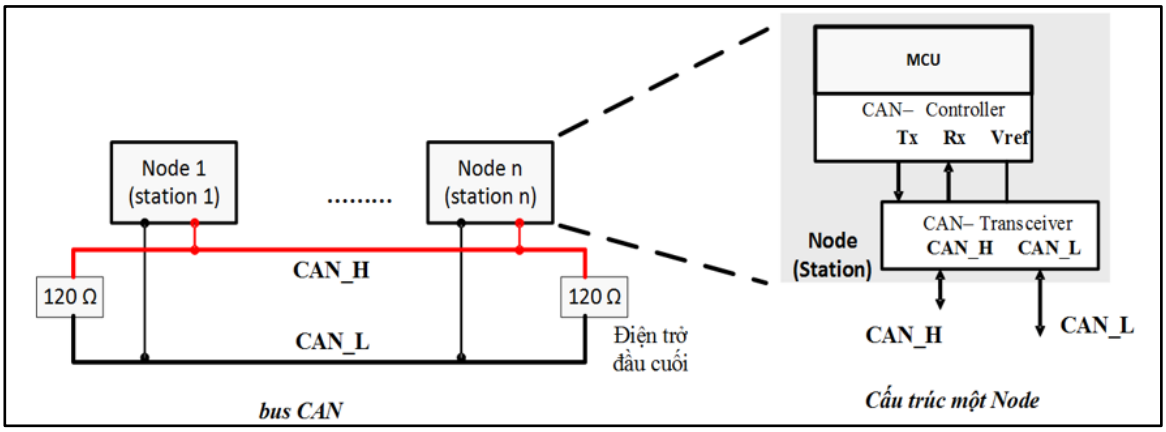


Hình 2.6. Hai dây CAN H và CAN L.

- Dây CAN H là dây có điện áp cao khi ở trạng thái trội.
- Dây CAN L là dây có điện áp thấp khi ở trạng thái trội.

- Điện trở đầu cuối của đường dây là 120Ω

- Node: là thành phần kết nối 2 dây CAN bao gồm: MCU, CAN Controller, CAN tranceiver.

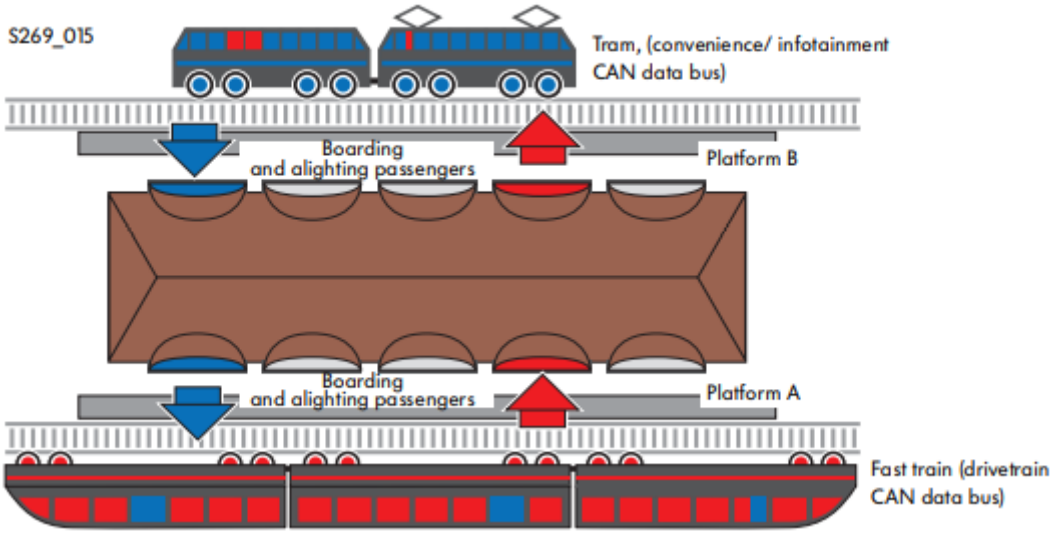


Hình 2.7.Cấu trúc của mạng CAN.

- MCU (Microcontroller): là một vi điều khiển, thực hiện hoạt động cho CAN controller, phân phối dữ liệu cần truyền đến CAN controller, lấy dữ liệu nhận từ CAN controller để sử dụng cho hoạt động của Node.
- CAN Controller: thực thi các xử lý về truyền nhận dữ liệu, báo lỗi, tính toán thời gian bit, ... theo chuẩn CAN quy định; phát dữ liệu cần truyền dạng số (theo mức logic 0/1) ra chân TX; nhận dữ liện dạng số qua chân RX.
- CAN tranceiver: bộ chuyển đổi từ tín hiệu số (mức logic 0/1) trên đường TX thành tín hiệu tương tự trên bus CAN, ngược lại, chuyển đổi từ tín hiệu tương tự trên bus CAN (CAN_H và CAN_L) thành tín hiệu số trên đường RX.

Vì không thể kết hợp CAN tốc độ cao với CAN tốc độ thấp do các mức tín hiệu khác nhau và cách bố trí điện trở. Hơn nữa, tốc độ truyền khác nhau của cả hai hệ thống bus dữ liệu khiến không thể đánh giá các tín hiệu khác nhau. Do đó, giữa hai hệ thống bus dữ liệu, một chuyển đổi là cần thiết. Việc chuyển đổi này được thực hiện trong Gateway. Vì Gateway có quyền truy cập vào tất cả thông tin thông qua bus dữ liệu CAN, nó cũng được sử dụng làm giao diện chẩn đoán.

- Nguyên lí hoạt động của Gateway:



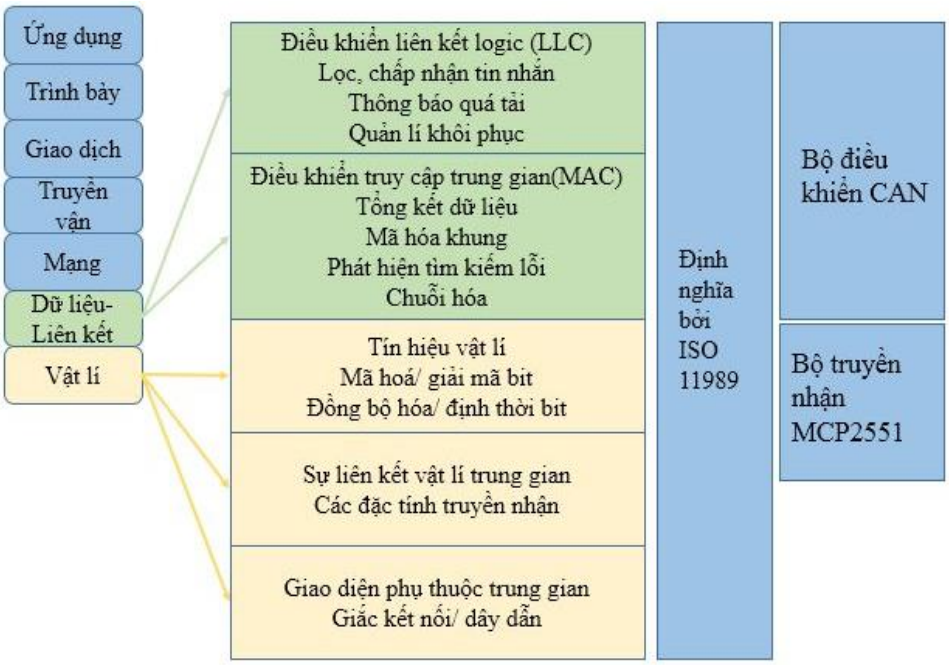
Hình 2.8. Nguyên lí hoạt động của Gateway.

Việc trao đổi dữ liệu từ CAN có tốc độ thấp với CAN có tốc độ cao được hình dung như việc trao đổi hành khách giữa 2 trạm qua sân ga Gateway. Điều này được mô tả như hình bên trên.

2.2.2.6. Cấu trúc phân lớp:

Lớp vật lý định nghĩa cách biểu diễn cũng như cách thu nhận bit 0 và 1, cách định thời và cách đồng bộ hóa. Là cấp thấp nhất trong bộ giao vận. Bao gồm các thành phần vật lý như: đường dây và điện áp dùng để gửi thông tin.

Lớp dữ liệu – liên kết: được chia làm 02 lớp con nhỏ hơn là điều khiển liên kết logic và điều khiển truy cập trung gian (MAC) nhằm định nghĩa khung truyền và những nguyên tắc về quyền ưu tiên.



Hình 2.9. Cấu trúc phân lớp của mạng CAN.

2.2.2.7. Cấu trúc bức điện:

Giao thức CAN sử dụng 4 loại khung (frame) khác nhau để truyền tải dữ liệu và điều khiển. Bốn loại khung này gồm:

- Khung dữ liệu (Data frame) là khung mang dữ liệu từ một bộ truyền đến các bộ nhận. Khung này có vùng để mang các byte dữ liệu.

- Khung yêu cầu hay khung điều khiển (Remote frame) là khung được truyền từ một node để yêu cầu node khác truyền khung dữ liệu có ID (IDENTIFIER) trùng với khung yêu cầu.

- Khung báo lỗi (Error frame) là khung được truyền bởi bất kỳ node nào khi node đó phát hiện lỗi bus.

- Khung báo quá tải (Overload frame) được sử dụng để tạo thêm độ trễ giữa các khung dữ liệu hoặc khung yêu cầu. Mỗi node trong bus CAN có thể truyền bất kỳ khi nào nếu phát hiện bus rảnh, nếu một node nhận quá nhiều dữ liệu, nó có thể dùng khung này để ngăn sự truyền tiếp theo.

Chỉ có khung dữ liệu và khung yêu cầu là có ID, cơ chế phân xử sẽ áp dụng cho hai loại khung này khi chúng được truyền trên bus. Khung dữ liệu và khung yêu cầu có hai định dạng khác nhau là định dạng chuẩn (Standard) và định dạng mở rộng (Extended):

- Định dạng khung chuẩn sử dụng ID có độ dài 11 bit.

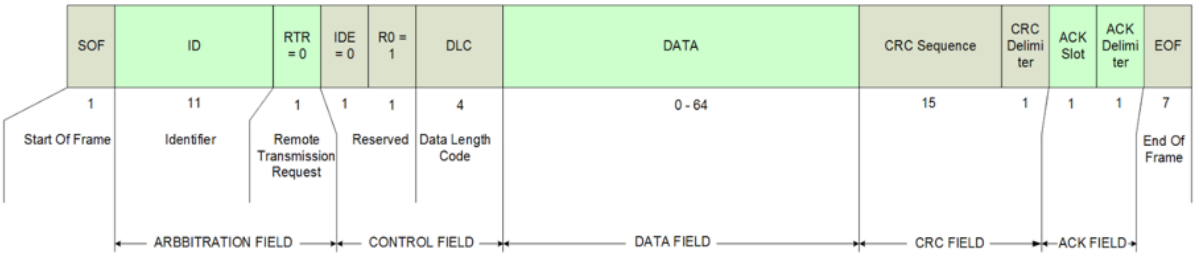
- Định dạng khung mở rộng sử dụng ID có độ dài 29 bit

Chuẩn CAN Specification 2.0-Part A (gọi tắt 2.0A) chỉ quy định sử dụng loại khung chuẩn. Chuẩn CAN Specification 2.0-Part B (gọi tắt 2.0B) sử dụng cả loại khung chuẩn và khung mở rộng. Như vậy, khi sử dụng CAN controller, bạn nên quan tâm đến việc nó tương thích chuẩn nào. Ngoài ra, chuẩn CAN còn một loại định dạng thứ 5 là khoảng liên khung (Interframe Spacing). Nó có vai trò tạo khoảng ngăn cách giữa các khung truyền trên bus CAN.

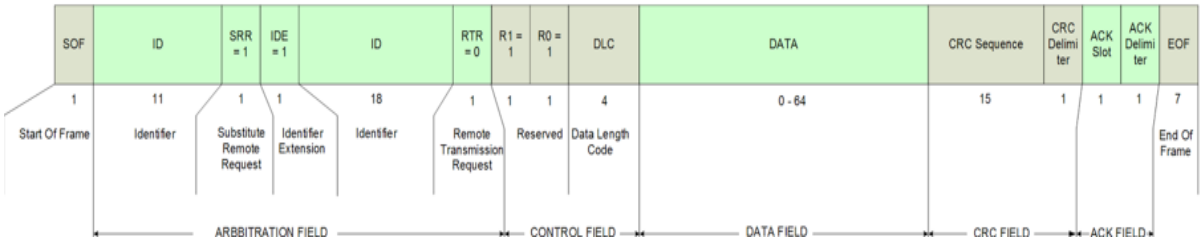
2.2.2.7.1. Khung dữ liệu:

Một khung dữ liệu bao gồm bảy vùng bit khác nhau theo thứ tự là:[4]

- Vùng bắt đầu khung (Start of frame – SOF)
- Vùng phân xử (Arbitration field)
- Vùng điều khiển (Control field)
- Vùng dữ liệu (Data field)
- Vùng kiểm tra (CRC field)
- Vùng báo nhận (ACK field)
- Vùng kết thúc khung (End of frame – EOF)



Hình 2.10. Khung tiêu chuẩn.



Hình 2.11. Khung mở rộng.

Vùng bắt đầu khung (Start of Frame - SOF): là vùng khởi đầu của khung dữ liệu, có độ dài là 1 bit và luôn là một bit trội. Một node chỉ được phép truyền dữ liệu khi Bus rảnh. Tất cả các trạm sẽ phải đồng bộ hóa dựa vào bit khởi đầu này. Khi một nút

CAN muốn gửi một tin nhắn nào đó thì nút đó sẽ đặt SOF trạng thái trội 0 để thông báo cho các nút khác. Ngoài ra, SOF còn có nhiệm vụ bắt đầu quá trình đồng bộ giữa các nút (tính từ thời điểm xuất hiện cạnh xuống chuyển từ trạng thái lặn sang trội) đồng thời báo hiệu quá trình phân xử sắp diễn ra.

Vùng phân xử (Arbitration field): là vùng giải quyết tranh chấp khi xảy ra việc một lúc có nhiều thông tin được truyền đi. Vùng phân xử được sử dụng là mức ưu tiên của bức điện ID CAN càng nhỏ thì có mức ưu tiên càng cao. Vùng phân xử có độ dài 12 bit bao gồm 11 bit ID và 1 bit RTR đối với khung tiêu chuẩn. Đối với khung mở rộng có độ dài 32 bit, bao gồm có 29 bit ID, 1 bit SRR, 1 bit IDE và 1 bit RTR.

Bit RTR (Remote Transmission Request) là bit dùng để phân biệt khung dữ liệu và khung yêu cầu. Bit này bằng 0 (bit dominant) đối với khung dữ liệu, bằng 1 (bit recessive) đối với khung yêu cầu. Vị trí bit này luôn đặt ngay sau ID. Chú ý, bit RTR là nằm sau bit ID -18 với khung chuẩn và sau ID -0 với khung mở rộng.

Bit SRR (Substitute Remote Request) là bit chỉ có ở khung mở rộng và luôn là bit recessive (bit 1). So sánh với khung chuẩn, vị trí của bit này trùng với vị trí của bit RTR nên có tên gọi là bit thay thế (Subsitute) cho bit RTR ở khung chuẩn. Giả sử có hai Node cùng truyền, một Node truyền khung dữ liệu tiêu chuẩn, một Node truyền khung dữ liệu mở rộng có ID giống nhau thì Node truyền khung chuẩn sẽ thắng phân xử vì đến vị trí sau Base ID, khung chuẩn là bit RTR = 0, còn khung mở rộng là bit SRR = 1. Như vậy, khung chuẩn chiếm ưu thế hơn so với khung mở rộng khi có Base ID như nhau.

Bit IDE là bit phân biệt giữa loại khung chuẩn và khung mở rộng. Nếu IDE = 0 (dominant) thì là khung chuẩn hoặc IDE = 1 (recessive) thì là khung mở rộng. Bit này có thể thuộc: vùng phân xử nếu là khung mở rộng hoặc thuộc vùng điều khiển nếu là khung chuẩn.

Dữ liệu trên bus CAN được truyền nối tiếp, các node nhận từng bit, khi một node nhận xong Base ID thì sẽ nhận tiếp bit tiếp theo, bit này có thể là SRR (nếu là khung mở rộng) hoặc RTR (nếu là khung chuẩn) vì đến đây node chưa xác định được loại khung. Node sẽ xác định như sau:

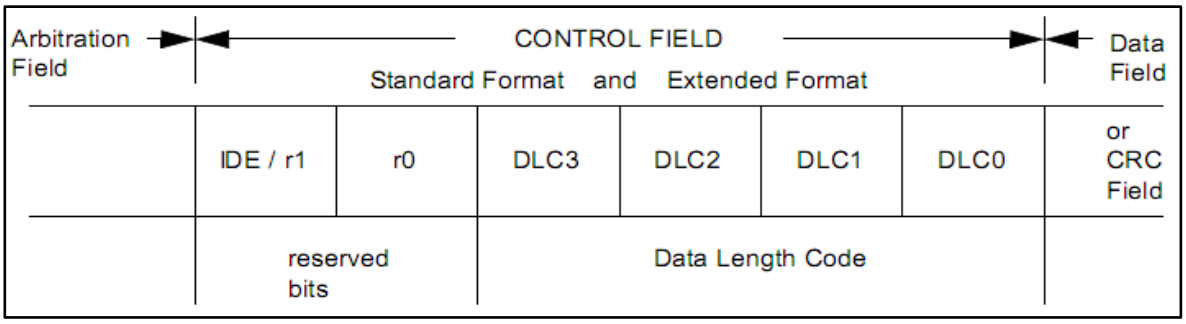
- Nếu bit ngay sau Base ID là bit dominant (bit 0) thì đây chắc chắn là khung dữ liệu định dạng chuẩn.

- Nếu bit ngay sau Base ID là bit recessive (bit 1) thì đây là khung điều khiển định dạng chuẩn hoặc khung mở rộng (khung dữ liệu hoặc khung yêu cầu). Node sẽ nhận tiếp bit tiếp theo là bit IDE để phân biệt.

- Nếu IDE = 0 thì bit trước đó là bit RTR và đây là khung yêu cầu dạng chuẩn.

- Nếu bit IDE = 1 thì bit trước đó là bit SRR và đây là khung mở rộng nhưng chưa biết là khung dữ liệu hay khung yêu cầu. Node sẽ nhận tiếp Extended ID và đến bit RTR mới phân biệt được.

Vùng điều khiển (Remote Field): có tổng chiều dài 6 bit, IDE bit trội nếu là khung tiêu chuẩn (bit 0), khung mở rộng bit lặn (bit 1) khi khung tiêu chuẩn và khung mở rộng cùng truyền một lúc thì khung tiêu chuẩn sẽ được ưu tiên (bit 0), r0 bit không sử dụng để sự dụng cho mục đích tương lai, 4 bít cuối mã hóa chiều dài dữ liệu. Có nhiệm vụ qui định số byte của trường dữ liệu.



Hình 2. 12. Vùng điều khiển.

DLC chỉ được mang giá trị (tính theo decimal) từ 0 đến 8, tương ứng với 8 byte dữ liệu. Có thể có trường hợp khung không chứa bất kì dữ liệu nào (DLC = 0).

Sự mã hóa số byte dữ liệu bởi mã độ dài dữ liệu.

Viết tắt : d = ‘dominant’ (trội)
r = ‘recessive’ (lặn)

Number of Data Bytes	Data Length Code			
	DLC3	DLC2	DLC1	DLC0
0	d	d	d	d
1	d	d	d	r
2	d	d	r	d
3	d	d	r	r
4	d	r	d	d
5	d	r	d	r
6	d	r	r	d
7	d	r	r	r
8	r	d	d	d

Hình 2.13. Bảng giá trị DLC.

Vùng dữ liệu (Data Field): Vùng dữ liệu có chiều dài từ 0 đến 8 byte, vùng dữ liệu chính là vùng chứa dữ liệu mà khung cần truyền, trong đó mỗi byte được truyền đi theo thứ tự từ bit có giá trị cao nhất (MSB) đến bít có giá trị thấp nhất (LSB). Độ dài của vùng dữ liệu phải tương ứng với giá trị của DLC. Nếu có sự khác biệt thì lỗi CAN sẽ được phát và dữ liệu truyền/nhận sẽ bị sai lệch.

Vùng kiểm tra (CRC field): Vùng kiểm tra hay vùng CRC gồm 16 bit và được chia làm hai phần là chuỗi CRC (CRC Sequence) và phần phân cách CRC (CRC Delimiter).

- Chuỗi CRC

- Gồm 15 bit.
- Được đánh dấu từ CRC-14 đến CRC-0.
- Giá trị của CRC sẽ ứng với một phép tính, thực chất là một phép chia đa thức (chia bit nhị phân).

- Gờ ngăn cách CRC

- Có độ dài 1 bit.
- Vị trí của bit ngăn cách CRC luôn nằm ngay sau CRC-0.
- Giá trị của bit ngăn cách CRC luôn là bit lặn nhằm làm nhiệm vụ phân cách vùng CRC với vùng liền kề nó.

Một chuỗi CRC cho phép hệ thống kiểm tra sự nguyên vẹn của dữ liệu truyền. Tất cả các nút nhận dữ liệu đều phải thực hiện bước này. Vùng dữ liệu được đưa vào để tính toán CRC được cung cấp từ:

- Khởi đầu khung.

- Trường phân xử.

- Vùng điều khiển dữ liệu chỉ đối với khung yêu cầu dữ liệu.

- Vùng dữ liệu chỉ đối với khung dữ liệu.

Quá trình tính toán CRC thực chất là một phép chia nhị phân sử dụng modulo -2 (là một dạng phép toán tìm phần dư của phép chia 2 số, còn gọi là modulus). Lưu ý rằng mã CRC chỉ phù hợp nhất đối với các chuỗi bit được kiểm tra có độ dài dưới 127 bit. Đối với khung CAN, tổng bit cần tính CRC tối đa là 83 bit (tiêu chuẩn) và 103 bit (mở rộng).

Vùng báo nhận (ACK Field) : Vùng xác nhận có độ dài 02 bit và được chia làm 02 phần:

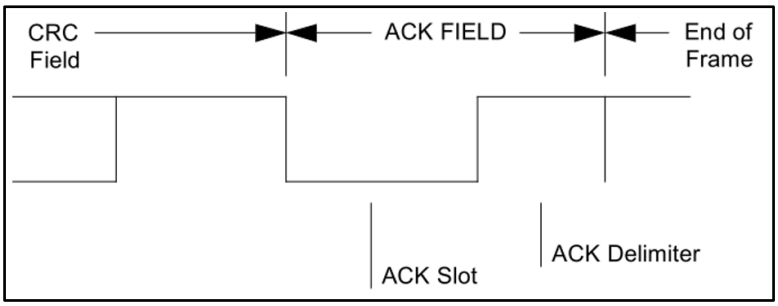
- Rảnh ACK

- Có độ dài 1 bit.
- Nút truyền khung CAN ban đầu sẽ đặt bit này ở giá trị lặn (ACK = 1).
- Khi một hoặc nhiều nút nhận khung hoàn tất đầy đủ các dữ liệu trong trường dữ liệu mà không trả về lỗi nào đồng thời tính toán CRC chính xác, thì các nút đó sẽ phản hồi lại đến nút nhận thông qua ACK Slot bit.
- Thông tin phản hồi được thể hiện qua việc các nút nhận khung sẽ kéo giá trị ACK Slot thành bit trội (ACK = 0).

- Gờ ngăn cách ACK

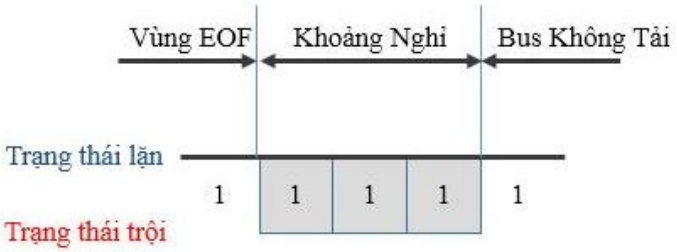
- Có độ dài 1 bit.
- Có nhiệm vụ phân cách giữa phần ACK với phần khác liền kề sau nó.

Lưu ý, bit ACK Slot luôn được đặt giữa 02 bit lặn là gờ ngăn cách CRC và gờ ngăn cách ACK.



Hình 2.14. Vùng báo nhận.

Vùng kết thúc khung (End of frame – EOF): là vùng thông báo kết thúc một khung dữ liệu hay một khung yêu cầu dữ liệu. EOF có độ dài 07 bit, giá trị của 07 bit này luôn là lặn.



Hình 2.15. Vùng kết thúc.

Ở khoảng giữa hai khung bất kỳ cũng phải giữ ít nhất 3 bit ở trạng thái "lặn" gọi là khoảng nghỉ của bus. Nếu sau đó không có nút nào muốn truyền dữ liệu thì bus tiếp tục giữ ở trạng thái "lặn", tức trạng thái bus không tải.

2.2.2.7.2. Khung yêu cầu:

Một node hoạt động như một bộ nhận đang mong muốn một dữ liệu, để nhận dữ liệu này từ một node nguồn thì bộ nhận sẽ gửi một khung yêu cầu (còn gọi là khung điều khiển). Khung yêu cầu có tác dụng thông báo cho node nguồn biết để node này truyền dữ liệu.

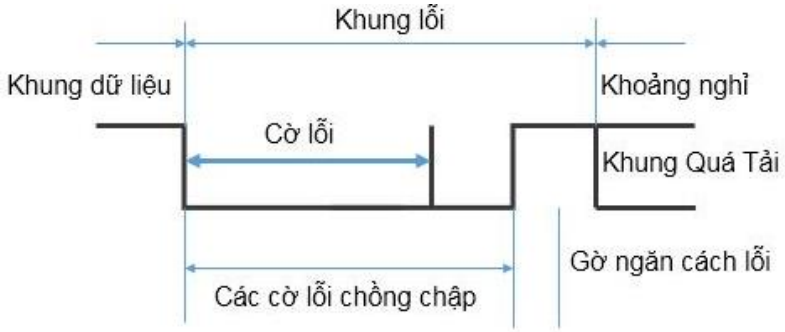
Một khung yêu cầu có thể là định dạng chuẩn hay định dạng mở rộng. Dù thuộc định dạng nào thì khung yêu cầu cũng gồm sáu vùng bit khác nhau là: Vùng bắt đầu khung (Start Of Frame), vùng phân xử (Arbitration Field), vùng điều khiển (Control Field), vùng kiểm tra (CRC Field), vùng ACK (ACK Field), vùng kết thúc khung (End Of Frame).

Cấu tạo và ý nghĩa các vùng trong khung yêu cầu giống như ở khung dữ liệu ngoại trừ một số điểm sau đây là khác so với khung dữ liệu:

Khung yêu cầu không có vùng dữ liệu (Data Field). Chính vì vậy, khung yêu cầu không phụ thuộc vào giá trị của mã chỉ độ dài dữ liệu DLC, DLC có thể nhận bất kỳ giá trị nào từ 0 đến 8. Nhưng giá trị DLC của khung yêu cầu lại chỉ ra độ dài dữ liệu trong khung dữ liệu tương ứng sẽ trả về, tức DLC của khung dữ liệu trả về giống DLC của khung yêu cầu đã gửi. Và khung dữ liệu trả về có số byte bằng với giá trị DLC này. Bit RTR (thuộc vùng phân xử) của khung yêu cầu là bit recessive. Nó khác với khung dữ liệu vì ở khung dữ liệu nó là một bit dominant.

2.2.2.7.3. Khung lỗi:

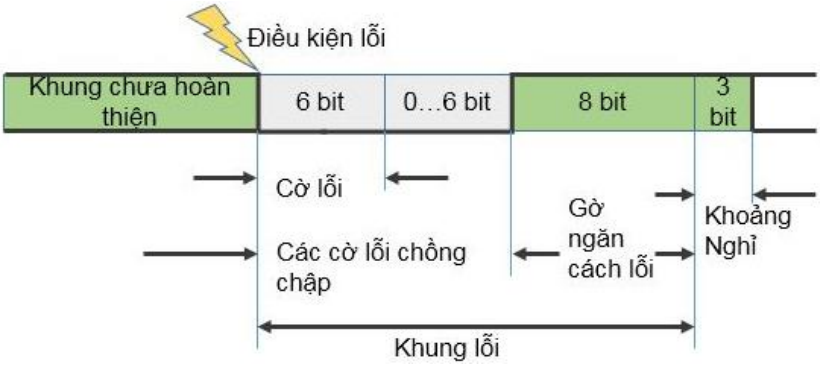
Khung báo lỗi (Error frame) được truyền trên bus bởi các node phát hiện thấy có điều kiện lỗi (các điều kiện lỗi sẽ được trình bày trong bài khác). Khung báo lỗi gồm hai vùng khác nhau. Vùng đầu tiên (superposition of Error Flags) là sự chồng chập của các cờ lỗi (Error Flag) được phát từ các node khác nhau trên bus. Tiếp theo đó là vùng phân cách khung báo lỗi (Error delimiter).



Hình 2.16. Khung lỗi.

Một khung lỗi được gửi từ bất kỳ trạm nào phát hiện lỗi trên bus. Khung lỗi bao gồm cờ lỗi và phân cách lỗi. CAN phân biệt hai loại lỗi là lỗi chủ động và lỗi bị động. Tương ứng với chúng là hai dạng cờ lỗi:

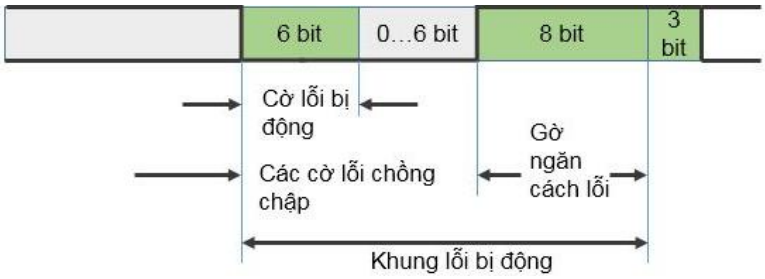
Dạng cờ lỗi chủ động bao gồm 6 bit trội liền nhau.



Hình 2.17.Khung lỗi.

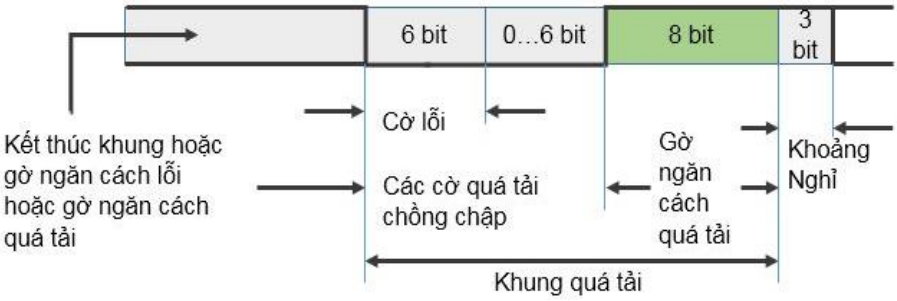
Dạng cờ lỗi bị động bao gồm 6 bit lặn liền nhau, trừ trường hợp nó bị ghi đè lên bởi các bit trội từ trạm khác. Một trạm lỗi chủ động khi phát hiện lỗi sẽ báo hiệu bằng cách gửi một cờ lỗi chủ động. Cờ lỗi chủ động vi phạm luật nhồi bit hoặc phá bỏ dạng cố định của trường ACK hay bit kết thúc khung. Chính vì vậy, tất cả các trạm khác cũng phát hiện ra lỗi và bắt đầu gửi cờ lỗi. Cuối cùng, dãy bit trội quan sát được trên bus thực tế là kết quả của sự xếp chồng nhiều cờ lỗi khác nhau phát riêng từ các trạm. Tổng chiều dài của dãy này dao động trong khoảng từ 6 đến 12 bit. Phân cách lỗi được đánh dấu

bằng 8 bit lặn liên tục. Sau khi gửi xong một cờ lỗi, mỗi trạm phải gửi tiếp một số bit lặn và đồng thời quan sát bus. Cho đến khi phát hiện ra một bit lặn (tức là khi các trạm khác đã gửi xong cờ lỗi chủ động), chúng sẽ phát tiếp bảy bit lặn.



Hình 2. 18. Khung lỗi.

2.2.2.7.4. Khung quá tải:



Hình 2.19. Khung quá tải.

Một khung quá tải có cấu trúc tương tự như ở khung lỗi bao gồm cờ quá tải và phân cách quá tải.

Cờ quá tải gồm sáu bit trội tương tự như cờ lỗi chủ động. Cờ quá tải xóa bỏ dạng cố định ở khoảng trống giữa hai khung. Chính vì vậy, tất cả các trạm khác cũng phát hiện tình trạng quá tải và bắt đầu gửi cờ quá tải. Cũng giống như khung lỗi, phân cách quá tải được đánh dấu bằng tám bít lặn liên tục.

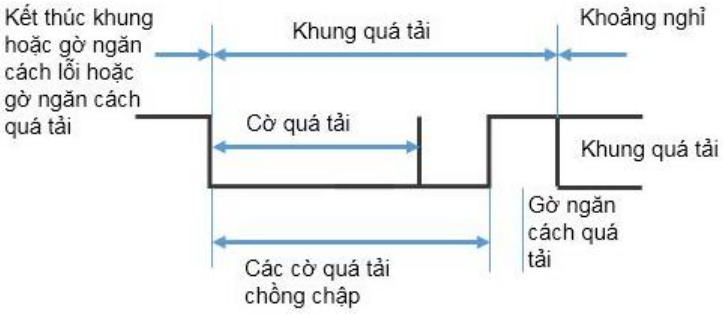
Có 3 kiểu tình trạng quá tải, cả 3 đều dẫn đến sự truyền của cờ báo quá tải.

Tình trạng bên trong của bộ thu nhận, nó đòi hỏi phải trì hoãn khung dữ liệu hay khung yêu cầu dữ liệu kế tiếp.

Phát hiện một bit "trội" tại bit thứ nhất hoặc bit thứ hai của khoảng nghỉ.

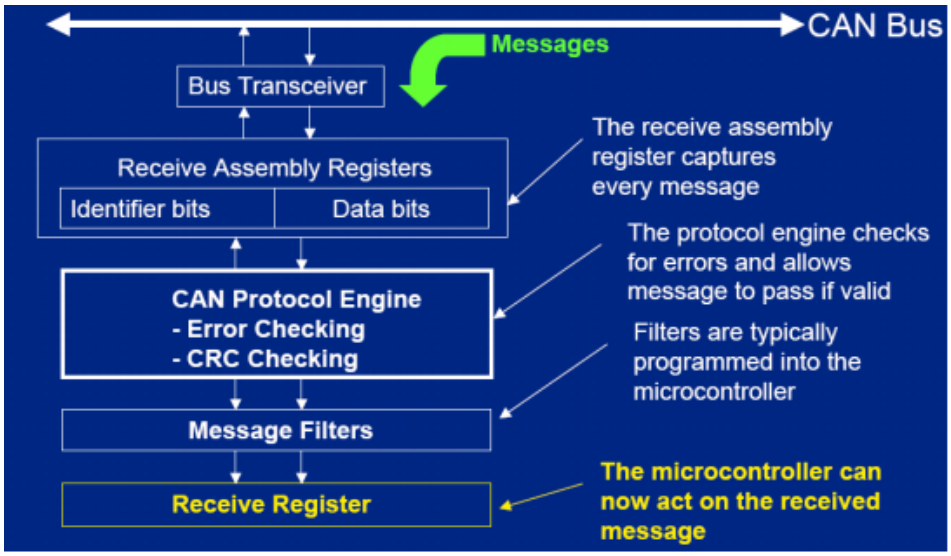
Nếu nút CAN lấy một bit trội ở bit thứ 8 (bit cuối cùng) của gờ phân cách lỗi hay gờ phân cách quá tải, nó sẽ bắt đầu truyền một khung quá tải chứ không truyền một khung lỗi, vì thế bộ đếm lỗi sẽ không bị gia tăng.

Nhiều nhất là hai khung quá tải có thể được tạo ra để trì hoãn các khung dữ liệu và khung yêu cầu dữ liệu kế tiếp tức là một nút chỉ có thể truyền tối đa 2 khung quá tải liên tiếp như hình dưới đây:



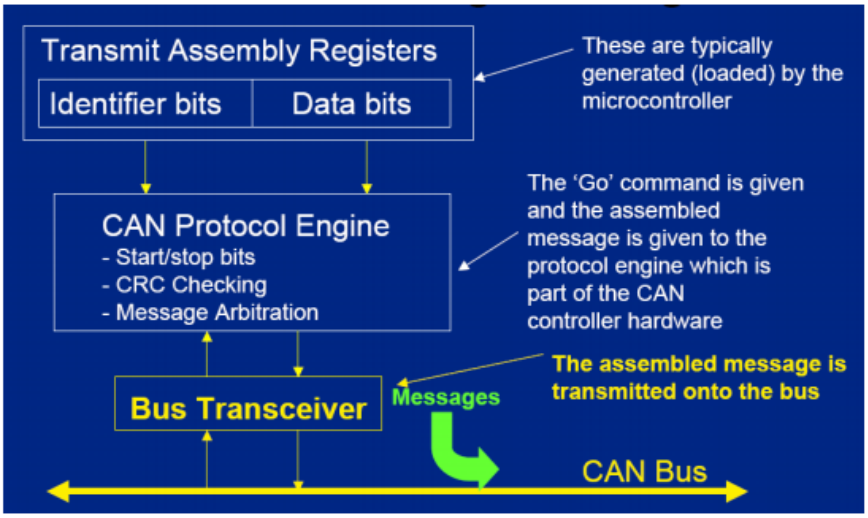
Hình 2.20. Khung quá tải.

2.2.2.8. Cách truyền dữ liệu:



Hình 2.21. Sơ đồ nhận CAN message..

Message được truyền bằng cách ghi vào các thanh ghi tương ứng. Trạng thái và các lỗi có thể phát hiện bằng đọc các thanh ghi tương ứng. Tất cả các message được kiểm tra lỗi và so sánh với các filter (thanh ghi lọc) để xem nó có được nhận và chứa vào trong 1 trong 2 thanh ghi nhận.



Hình 2.22. Sơ đồ truyền CAN messages.

Can hỗ trợ các kiểu frame sau:

- Kiểu frame chuẩn
- Kiểu mở rộng
- Remote frame
- Error frame
- Overload Frame Reception
- Khoảng giữa các frame truyền

Các tài nguyên trong module CAN[2]:

- 3 thanh ghi truyền: TXB0, TXB1 và TXB2
- 2 thanh ghi nhận: RXB0 và RXB1
- 2 mặt nạ nhận (filter mask), mỗi cái cho một thanh ghi nhận: RXM0, RXM1
- 6 thanh ghi lọc, 2cho RXB0 và 4 cho RXB1: RXF0, RXF1, RXF2, RXF3, RXF4, RXF5.

Module CAN sử dụng chân RB2/CANTX và RB3/CANRX để giao tiếp với bus CAN. Trình tự sau để thiết lập CAN module trước khi sử dụng để truyền hay nhận:

- Đảm bảo module trong chế độ thiết lập
- Thiết lập chế độ baud
- Thiết lập các thanh ghi lọc và mặt nạ
- Đưa module CAN về chế độ hoạt dộng bình thường hay các chế độ khác tùy theo áp dụng

Các mode hoạt động:

- Configuration mode : trong mode này, module CAN được khởi tạo trước khi hoạt động. Module CAN không cho phép vào mode này khi có một sự truyền hay nhận đang xảy ra, nó giống như cái khóa bảo vệ các thanh ghi khi hoạt động.
- Listen mode: Mode này dùng để quan sát trạng .
- Loop back mode: mode này cho phép truyền các message từ buffer truyền sang buffer nhận mà không thực sự truyền message ra ngoài CAN bus, sử dụng phát triển và kiểm tra hệ thống. Bit ACK không được kiểm tra và thiết bị cho phép các messages từ nó như những message từ các node khác.
- Disabled mode: trong mode này, module không truyền hay nhận, Mode này giống như tắt module, làm cho xung clock dừng.
- Normal mode: là mode hoạt dộng cho thiết bị. Trong mode này, thiết bị kiểm tra tất cả các message trên bus và tạo bit ACK, frame lỗi… và chỉ là mode duy nhất cho phép truyền nhận message lên bus CAN.
- Error recognition mode : mode có thể thiết lập để bỏ qua tất cả các lỗi và nhận bất cứ message nào.

Truyền message CAN:

Các buffer truyền:

Có 3 buffer truyền TXB0, TXB1, TXB2. Mỗi buffer chiếm 14 byte SRAM và bao gồm một thanh ghi điều khiển(TXBnCTRL), 4 thanh ghi ID (TXBnSIDL, TXBnSIDH, TXBnEIDL, TXBnEIDH), một thanh ghi đếm độ dài dữ liệu (TXBnDLC) và 8 thanh ghi dữ liệu (TXBnDm).

Thiết lập truyền:

Bit TXREQ phải được xóa để chỉ thị buffer đang rỗng hay message vừa mới gửi đi. Sau đó, các thanh ghi SIDH, SIDL, DLC và thanh dữ liệu được nạp. Nếu sử dụng frame mở rộng (ID mở rộng) thì thanh ghi EIDH:EIDL phải được ghi và bit EXIDE được set để báo hiệu sử dụng frame mở rộng.

Để bắt đầu truyền, ta set bit TXREQ cho mỗi buffer truyền. Để truyền thành công thì phải có ít nhất 1 node nhận biết được tốc độ baud trên mạng.

Set bit TXREQ không có nghĩa là truyền ngay, nó giống như báo hiệu buffer sẵn sàng truyền. Sự truyền chỉ bắt đầu khi thiết bị kiểm tra bus đã rảnh. Sau đó thiết bị sẽ truyền message nào có mức ưu tiên cao nhất. Khi truyền thành công, bit TXREQ sẽ xóa, cờ TXBnIF được set và ngắt sẽ xảy ra nếu bit cho phép ngắt TXBnIE được set.

Nếu truyền không thành công, bit TXREQ vẫn được set, báo hiệu message vẫn chưa giải quyết (pending) và một trong các cờ sẽ set. Nếu có lỗi, TXERR và IRXIF sẽ set và một ngắt sẽ sảy ra. Nếu message mất ưu tiên trên bus, bit TXLARB sẽ set.

Ưu tiên truyền:

Sự ưu tiên này không liên quan tới sự ưu tiên của message trên bus theo giao thức CAN. Đây chỉ là sự ưu tiên trong thiết bị xem message nào sẽ được truyền trước hay thứ tự truyền của 3 buffer. Buffer nào có mức ưu tiên cao nhất sẽ được truyền trước . Nếu 2 buffer có cùng mức ưu tiên, thì buffer nào có số kí hiệu cao hơn sẽ được truyền trước. Có 4 mức ưu tiên: nếu các bit TXP là ‘11’, thì buffer đó có mức ưu tiên cao nhất; nếu các bit TXP là ‘00’, thì buffer đó có mức ưu tiên thấp nhất.

Nhận message CAN:

Có 2 buffer nhận là RXB0 và RXB1. Mỗi buffer chiếm 14 byte SRAM và một thanh ghi điều khiển(RXBnCRTL), 4 thanh ghi ID (RXBnSIDL, RXBnSIDH, RXBnEIDL, RXBnEIDH), một thanh ghi đếm độ dài dữ liệu (RXBnDLC) và 8 thanh ghi dữ liệu (RXBnDm). Nó còn có một buffer riêng là Message Assembly Buffer (MAB) có vai trò là một buffer phụ. MAB luôn nhận message kế tiếp trên bus và không thể tác động trực tiếp bởi firmware. Buffer MAB tiếp nhận lần lượt tất cả các message tới. Message sau đó được truyền tới buffer nhận tương ứng chỉ khi nào ID của message đúng với bộ lọc.

MAB (message assembly buffer) được sử dụng để nhận message kế tiếp trên bus. MCU có thể tác động một buffer trong khi buffer kia nhận message hay giữ message vừa nhận.

Khi một message chuyển tới bất kì buffer nhận nào bit RXFUL được set. Bit này phải được xóa bởi MCU khi nó đã xử lý xong message trong buffer để cho phép message mới có thể nhận trong buffer. Bit này đảm bảo thiết bị đã xử lý xong message trước khi module cố gắng đưa message mới vào buffer nhận. Nếu một ngắt nhận cho phép, thì ngắt sẽ xảy ra báo hiệu một message đã được nhận thành công.

Khi message được đưa vào thanh ghi nhận, phần mềm có thể xác định chính xác bộ lọc nào cho phép sự nhận này bằng cách kiểm tra filter hit bits FILHIT<3:0> trong thanh ghi RXBnCON tương ứng. Message vừa nhận là message chuẩn nếu bit EXID trong thanh ghi RXBnSIDL được xóa. Ngược lại, bit EXID được set sẽ báo hiệu một message mở rộng.

Ưu tiên nhận:

RXB0 là buffer có mức ưu tiên cao nhất và có hai bộ lọc kết hợp với nó. RXB1 là buffer có mức ưu tiên thấp và có 4 bộ lọc. Hơn nữa, thanh ghi RXB0CON có thể thiết lập để khi RXB0 chứa một message hợp lệ, và một message hợp lệ khác khi được nhận, một error sẽ không xảy ra và message mới sẽ được đưa vào RXB1. Có 2 mặt nạ lọc cho mỗi bufer.

Message Acceptance Filtes and Masks được sử dụng để xác định xem message trong MAB có được chuyển vào các bufer nhận hay không. Khi một messge hợp lệ vừa được nhận vào MAB, vùng ID được so sánh với giá trị của bộ lọc. Nếu đúng, message sẽ được chuyển vào bufer tương ứng. Filter mask được sử dụng để xác định xem bit nào trong vùng ID sẽ được so sánh với bộ lọc. Bảng chân trị cho thấy mỗi bit trong ID được so sánh với mặt nạ và bộ lọc để xác định message có được chuyển vào buffer nhận hay không. Nếu bit nào được thiết lập bằng 0, bit đó sẽ được chấp nhận mà không cần xét đến bộ lọc.

2.3. Hệ thống hiển thị thông tin trên ô tô:

2.3.1. Tổng quát về hệ thống hiển thị thông tin trên ô tô:

Hệ thống thông tin trên xe bao gồm các bảng đồng hồ (tableau), màn hình và các đèn báo giúp tài xế và người sửa chữa biết được thông tin về tình trạng hoạt động của các hệ thống chính trong xe. Thông tin có thể truyền đến tài xế qua 2 dạng : tương tự (táp lô kim) và số (táp lô hiện số).

Hệ thống thông tin bao gồm các đồng hồ và đèn báo sau:

- Đồng hồ tốc độ xe (speedometer)
- Đồng hồ đo quãng đường (odometer)
- Đồng hồ tốc độ động cơ (tachometer)
- Đồng hồ áp lực nhớt
- Đồng hồ nhiệt độ nước làm mát.

- Đồng hồ báo nhiên liệu
- Đèn báo áp suất nhớt thấp
- Đèn báo nạp
- Đèn báo pha
- Đèn báo rẽ
- Đèn báo nguy hiểm hoặc ưu tiên
- Đèn báo mức nhiên liệu thấp
- Đèn báo hệ thống phanh
- Đèn báo cửa mở,…

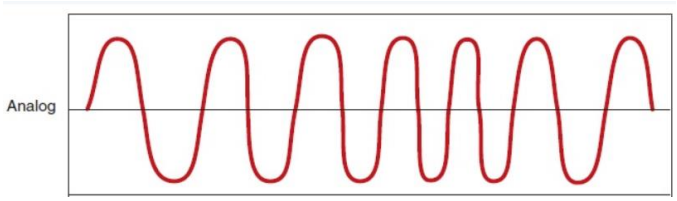
	Đèn báo phanh tay	**T-BELT**	Đèn báo thắt dây an toàn chưa đúng vị trí
	Đèn báo chưa thắt dây an toàn		Đèn báo lọc nhiên liệu bị bẩn, nghẹt
	Đèn báo nạp		Đèn báo mực nước làm mát thấp
	Đèn báo áp lực nhớt thấp		Đèn báo rẽ
	Đèn báo mực nhớt động cơ		Đèn báo nguy
	Đèn báo lỗi (điều khiển động cơ)		Đèn báo xông
	Đèn báo có cửa chưa đóng chặt		Đèn báo pha

Hình 2.23. Đèn tín hiệu trên ô tô.

2.3.1.1. Tín hiệu Analog:

Hệ thống thông tin dạng tương tự hay analog bao gồm các đồng hồ dạng kim và các đèn báo để kiểm tra và theo dõi hoạt động của một số bộ phận quan trọng của động cơ cũng như toàn xe.

Tín hiệu Analog là tín hiệu liên tục, đồ thị biểu diễn tín hiệu analog là một đường liên tục (ví dụ sin, cos, hoặc đường cong lên xuống bất kỳ), analog có nghĩa là: tương tự. Tương tự nghĩa là tín hiệu lúc sau cũng có dạng tương tự như lúc trước, giống nhau về bản chất tín hiệu nhưng sẽ khác về cường độ tín hiệu lúc sau so với lúc trước. Trong thiết bị điện tín hiệu analog là dòng điện, nó cũng có thể là tín hiệu âm thanh ta nghe, hình ảnh ta thấy, trong viễn thông là sóng điện từ (tức anh sáng không nhìn thấy).



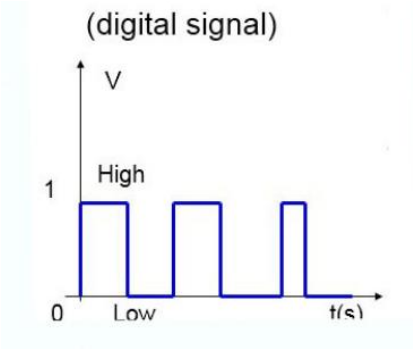
Hình 2.24. Tín hiệu Analog.

2.3.1.2. Tín hiệu Digital:

Tín hiệu digital là loại tín hiệu số loại tín hiệu lẻ rời rạc không nối tiếp nhau trong từng thời điểm. Đây là tín hiệu được thể hiện bằng hệ nhị phân điện thế 0-1; thể hiện ở 2 mức cao và thấp, trong đó 1 là mức điện thế cao và 0 là mức điện thấp.

Để hiển thị tín hiệu dạng digital thường sử dụng một VFD, một vài điốt đèn LED phát sáng hoặc một LCD. Kiểu VFD được sử dụng phổ biến trong các đồng hồ hiển thị số trong các xe đời mới. Đồng hồ hiển thị số có các đặc điểm sau:

- Dễ xem.
- Chính xác cao.
- Độ tin cậy cao nhờ hiển thị số, không có chi tiết chuyển động quay.
- Hiển thị tốt nhất cho mỗi đồng hồ.



Hình 2.25. Tín hiệu Digital.

Chương 3. SƠ LƯỢC VỀ PHẦN CỨNG VÀ PHẦN MỀM

3.1. Sơ lược về phần mềm sử dụng:

3.1.1. Giới thiệu về lập trình Python:

Python là một ngôn ngữ lập trình bậc cao cho các mục đích lập trình đa năng, do Guido van Rossum tạo ra và lần đầu ra mắt vào năm 1991. Python được thiết kế với ưu điểm mạnh là dễ đọc, dễ học và dễ nhớ. Python là ngôn ngữ có hình thức, cấu trúc rõ ràng, thuận tiện cho người mới học lập trình. Cấu trúc của Python còn cho phép người sử dụng viết mã lệnh với số lần gõ phím tối thiểu. Vào tháng 7 năm 2018, Van Rossum đã từ chức Leader trong cộng đồng ngôn ngữ Python sau 30 năm lãnh đạo. Python hoàn toàn tạo kiểu động và dùng cơ chế cấp phát bộ nhớ tự động; do vậy nó tương tự như Perl, Ruby, Scheme, Smalltalk, và Tcl. Python được phát triển trong một dự án mã mở, do tổ chức phi lợi nhuận Python Software Foundation quản lý.

3.1.2. Giới thiệu về Sublime Text:

Sublime Text là một trình soạn thảo mã nguồn đa nền tảng phần mềm chia sẻ với giao diện lập trình ứng dụng Python (API). Nó hỗ trợ nguyên bản nhiều ngôn ngữ lập trình và ngôn ngữ đánh dấu và người dùng có thể thêm các chức năng bằng các plugin, thường do cộng đồng xây dựng và duy trì theo giấy phép phần mềm miễn phí.

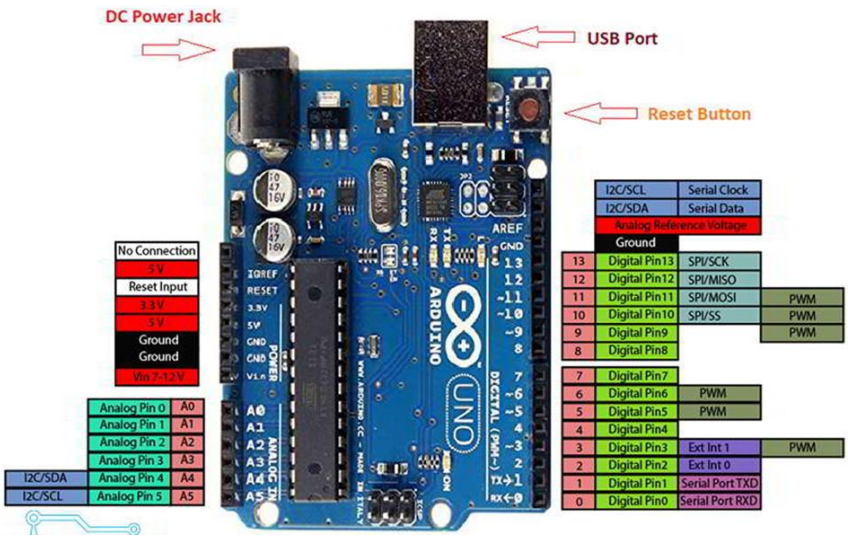
3.1.3. Giới thiệu về phần mềm Arduino:

Arduino bao gồm cả phần cứng và phần mềm. Để lập trình cho các board mạch Arduino bạn chỉ cần download và cài đặt phần mềm soạn thảo code là Arduino IDE. Phần mềm Arduino IDE có giao diện rất đơn giản, giúp bạn quản lý các file code và upload code vào board mạch Arduino dễ dàng. Ngôn ngữ lập trình của Arduino chính là C/C++ nhưng so với lập trình với vi điều khiển thì lập trình với Arduino đơn giản hơn nhiều vì bạn chỉ giao tiếp với phần cứng thông qua các thư viện và tất cả những sự phức tạp liên quan đến phần cứng đã được che dấu đi.

3.2. Sơ lược về phần cứng:

3.2.1. Arduino Uno :

Khi mới bắt đầu làm quen với lập trình Arduino thì mạch thường được sử dụng là dòng Arduino Uno. Hiện nay, dòng mạch này đã phát triển tới thế hệ thứ 3 (mạch Arduino Uno R3) cho phép lập trình cho các ứng dụng điều khiển và dễ sử dụng nhất trong các dòng Arduino hiện nay.



Hình 3.1. Arduino Uno.

Sơ lược về board[3]:

- Vi điều khiển ATmega328 họ 8 bit.
- Điện áp hoạt động 5V DC (chỉ được cấp qua cổng USB).
- Tần số hoạt động 16 MHz.
- Dòng tiêu thụ khoảng 30mA.
- Điện áp vào khuyên dùng 7-12V DC.
- Điện áp vào giới hạn 6-20V DC.
- Số chân Digital I/O14 (6 chân PWM).
- Ở mỗi chân đều có các điện trở kéo lên từ được cài đặt ngay trong vi điều khiển. ATmega328 (mặc định thì các điện trở này không được kết nối).
- Số chân Analog 6 (độ phân giải 10bit).
- Dòng tối đa trên mỗi chân I/O30 mA.
- Dòng ra tối đa (5V) 500 mA.
- Dòng ra tối đa (3.3V) 50 mA.
- Bộ nhớ flash32 KB (ATmega328) với 0.5KB dùng bởi bootloader.

SPI (Serial Peripheral Bus) là một chuẩn truyền thông nối tiếp tốc độ cao do hãng Motorola đề xuất. Đây là kiểu truyền thông Master-Slave, trong đó có 1 chip Master điều phối quá trình truyền thông và các chip Slaves được điều khiển bởi Master vì thế truyền thông chỉ xảy ra giữa Master và Slave. SPI là một cách truyền song công (full duplex) nghĩa là tại cùng một thời điểm quá trình truyền và nhận có thể xảy ra đồng thời. SPI đôi

khi được gọi là chuẩn truyền thông “4 dây” vì có 4 đường giao tiếp trong chuẩn này đó là SCK (Serial Clock), MISO (Master Input Slave Output), MOSI (Master Ouput Slave Input) và SS (Slave Select).

SCK: Xung giữ nhịp cho giao tiếp SPI, vì SPI là chuẩn truyền đồng bộ nên cần 1 đường giữ nhịp, mỗi nhịp trên chân SCK báo 1 bit dữ liệu đến hoặc đi.

MISO– Master Input / Slave Output: nếu là chip Master thì đây là đường Input còn nếu là chip Slave thì MISO lại là Output. MISO của Master và các Slaves được nối trực tiếp với nhau.

MOSI – Master Output / Slave Input: nếu là chip Master thì đây là đường Output còn nếu là chip Slave thì MOSI là Input. MOSI của Master và các Slaves được nối trực tiếp với nhau.

SS – Slave Select: SS là đường chọn Slave cần giao tiếp, trên các chip Slave đường SS sẽ ở mức cao khi không làm việc. Nếu chip Master kéo đường SS của một Slave nào đó xuống mức thấp thì việc giao tiếp sẽ xảy ra giữa Master và Slave đó. Chỉ có 1 đường SS trên mỗi Slave nhưng có thể có nhiều đường điều khiển SS trên Master, tùy thuộc vào thiết kế của người dùng.

+ MISO mang các dữ liệu từ các thiết bị SPI về Arduino.

+ MOSI mang các dữ liệu từ Arduino đến các thiết bị SPI.

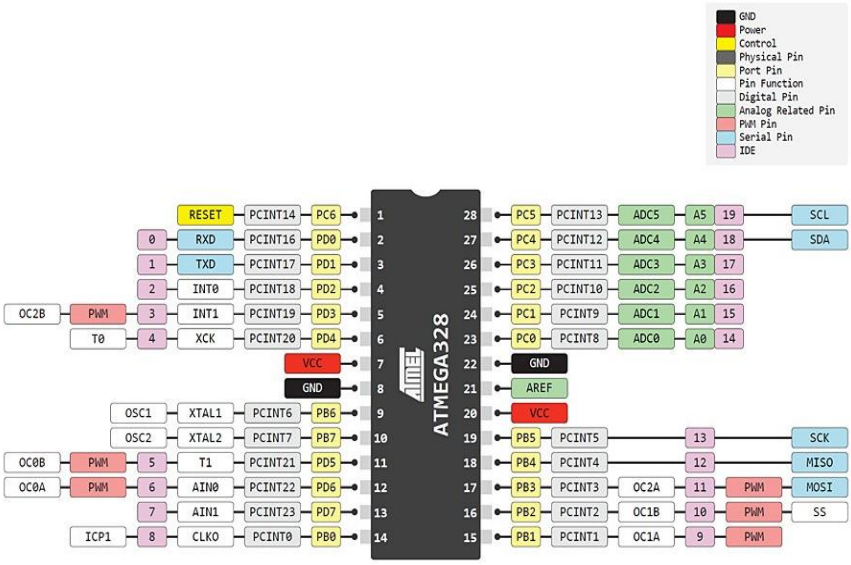
+ SS chọn thiết bị SPI cần làm việc.

+ SCK dòng đồng bộ.

Chip ATMega 328:

Atmega328 là một chíp vi điều khiển được sản xuất bởi hãng Atmel thuộc họ MegaAVR có sức mạnh hơn hẳn Atmega8. Atmega 328 là một bộ vi điều khiển 8 bít dựa trên kiến trúc RISC bộ nhớ chương trình 32KB ISP flash có thể ghi xóa hàng nghìn lần, 1KB EEPROM, một bộ nhớ RAM vô cùng lớn trong thế giới vi xử lý 8 bít (2KB SRAM)

Hình 3.2. ATmega 328.



Hình 3.3. Các chân của ATMega 328.

Chip Atmega 328P có thông số kỹ thuật như sau:

- Chip họ AVR 8-bit
- Điện áp hoạt động: 1.8V đến 5V
- Xung nhịp lớn nhất: 20Mhz
- Bộ nhớ EEPROM: 1KB
- Bộ nhớ RAM: 2KB
- Digital I/O pin: 23
- PWM pin: 6
- ADC pin: 6
- Nhiệt độ hoạt động: -40°C đến 85°C
- Hai bộ Timer/Counter 8-bit
- Một bộ Timer/Counter 16-bit
- Chiều dài: 34.798 mm
- Chiều rộng: 8.255 mm

Ý nghĩa các chân:

- Chân VCC: Chân số 7 là VCC cấp điện áp nguồn cho vi điều khiển. Nguồn điện cấp trong khoảng 1,8 – 5,5V.
- Chân GND: Chân số 8 và chân số 22 nối GND (hay nối mass). Khi thiết kế cần sử dụng một mạch ổn áp để bảo vệ cho vi điều khiển, cách đơn giản là sử dụng IC

7805.

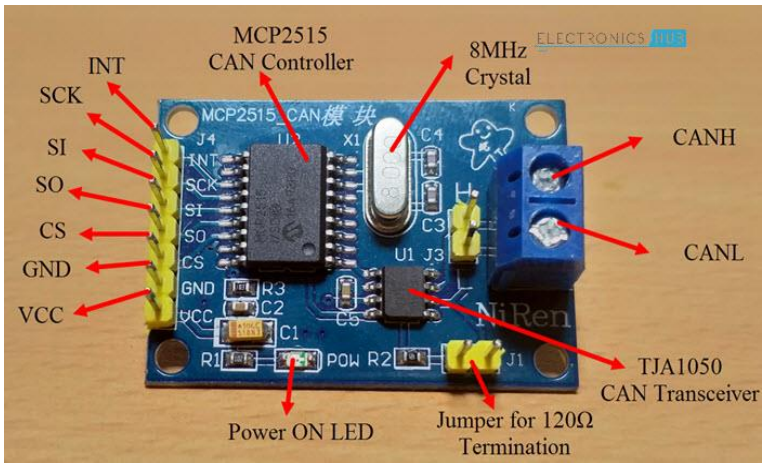
- Port B: Gồm 8 chân từ PB0 – PB7, các chân này có chức năng làm đường nhập/xuất và nhiều chức năng phụ khác.
- Port C: Gồm 7 chân từ PC0 – PC6, ngoài chức năng làm các đường nhập/xuất thì còn có 5 chân có chức năng đầu vào cho bộ chuyển đổi ADC.
- Port D: Gồm 8 chân từ PD0 – PD7, chức năng giống với các chân ở Port B.
- Chân RESET: Nằm ở chân số 1 của vi điều khiển dùng để thiết lập lại trạng thái ban đầu cho vi điều khiển.
- Chân XTAL[1..2]: Hai chân này nằm ở vị trí chân số 9 và 10 trên vi điều khiển được sử dụng để nhận nguồn xung clock từ bên ngoài để hoạt động, thường được ghép với thạch anh và các tụ để tạo nguồn xung clock ổn định.
- Chân AVCC: Nguồn cung cấp cho cổng C và bộ chuyển đổi ADC, chân này nên được nối với nguồn cấp VCC từ bên ngoài ngay cả khi bộ ADC không hoạt động.
- Chân AREF: AREF là chân chuẩn analog cho bộ chuyển đổi ADC

Các chức năng cơ bản của ATmega328P:

- Điều khiển động cơ: Chip ATmega328P sử dụng các chân PB1, PB2, PB3, PD3, PD5, PD6 cho phép xuất xung PWM dùng để điều khiển động cơ.
- Giao tiếp với các module ngoại vi thông qua 6 kênh ADC từ PC0 tới PC5.
- Truyền nhận dữ liệu thông qua các chuẩn giao tiếp: I2C, UART, SPI.

3.2.2. MCP 2515:

Vi điều khiển MCP2515 trong bộ điều khiển CAN quản lý gói dữ liệu, mã hóa khung dữ liệu, giải mã bit, phát hiện lỗi, đồng bộ dữ liệu,.. Hỗ trợ CPU quản lí tin nhắn, tương thích với các đặc điểm kỹ thuật của CAN 2.0B.



Hình 3.4. MCP 2515.

Module CAN dùng chip CAN Controller MCP2515 và bộ truyền nhận CAN TJA1050 là module mở rộng ngoại vi CAN.

Module còn tích hợp các Jump J1 J2, trong đó:

- J1 là jump chọn chế độ tốc độ giao tiếp, nếu Jump 1 được nối thì chip TJA1050 cho phép giao tiếp trên mạng CAN- BUS với tốc độ cao nhất có thể, lên tới 1M. Nếu Jump 1 không được kết nối thì TJA1040 giao tiếp với BUS CAN tốc độ thấp, < 10 kbps, ở chế độ này cho phép khoảng cách truyền xa hơn và nối được nhiều node mạng CAN trên bus vật lý hơn.

- J2 là jump nối điện trở liên kết. Mỗi BUS CAN có 2 điện trở 120 Ohm ở hai đầu bus. Nếu J2 nối thì module sẽ là nốt mạng đầu hoặc node mạng cuối. J2 không nối khi sử dụng ở module giao tiếp với mạng CAN BUS đã có sẵn điện trở ở 2 đầu.

Thông số kỹ thuật:

- Chuẩn giao tiếp SPI, có chân ngắt khi nhận được gói tin hợp lệ.

- Điện áp hoạt động: 4.75 ~5.25V. Tương thích đầy đủ với chuẩn ISO 11898 cho hệ thống hoạt động ổn định, chính xác.

3.2.3. Raspberry Pi 3:

Pi là một máy vi tính rất nhỏ gọn, kích thước hai cạnh chỉ cỡ một cái thẻ ATM. Người ta đã tích hợp mọi thứ cần thiết trong đó để bạn sử dụng như một cái máy vi tính. Rasbian là hệ điều hành được sử dụng phổ biến nhất do được hỗ trợ các công cụ lập trình giúp lập trình viên dễ dàng phát triển chương trình của mình, cung cấp một hệ điều hành thông thường cho người dùng.

Trên bo mạch của Pi có CPU, GPU, RAM, khe cắm thẻ microSD, Wi-Fi, Bluetooth và 4 cổng USB 2.0. Khi mua Pi về, bạn chỉ việc cài hệ điều hành, gắn chuột, bàn phím và màn hình là có thể sử dụng.

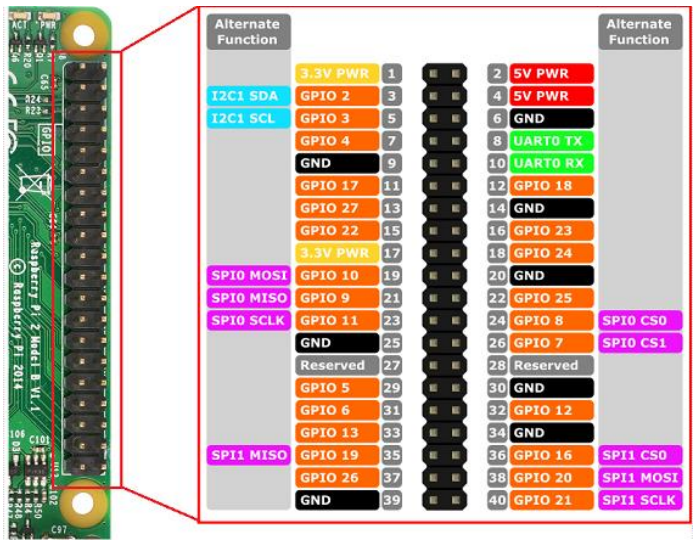
Bộ tứ lõi 64 bit 1,2 GHz CPU ARM Cortex-A53 (hiệu suất gấp 10 lần Raspberry Pi 1) tích hợp mạng LAN không dây 802.11n và Bluetooth 4.1 1 GB RAM 4x, cổng USB đầu ra kích thước đầy đủ. Khe cắm MicroSD cổng MicroUSB cho đầu ra âm thanh nổi GPIO 4 cực mở rộng 40 chân và video tổng hợp cổng camera CSI cổng để kết nối camera Raspberry Pi hoàn toàn tương thích với cổng hiển thị Raspberry Pi 1 và 2 DSI để kết nối màn hình cảm ứng Raspberry Pi 3.

Nguồn cung cấp Raspberry Pi 3 qua cổng micro usb với điện áp 5V và dòng xoay chiều 2A.

Thông số kĩ thuật:

- Vi xử lý: Broadcom BCM2837B0, quad-core A53 (ARMv8) 64-bit SoC @1.4GHz
- RAM: 1GB LPDDR2 SDRAM
- Kết nối: 2.4GHz and 5GHz IEEE 802.11 b/g/n/ac wireless LAN, Bluetooth 4.2, BLE, Gigabit Ethernet over USB 2.0 (Tối đa 300Mbps).
- Cổng USB: 4 x 2.0
- Mở rộng: 40-pin GPIO
- Video và âm thanh: 1 cổng full-sized HDMI, Cổng MIPI DSI Display, cổng MIPI CSI Camera, cổng stereo output và composite video 4 chân.
- Multimedia: H.264, MPEG-4 decode (1080p30), H.264 encode (1080p30); OpenGL ES 1.1, 2.0 graphics
- Lưu trữ: MicroSD
- Nguồn điện sử dụng: 5V/2.5A DC cổng microUSB, 5V DC trên chân GPIO, Power over Ethernet (PoE) (yêu cầu thêm PoE HAT)

Hình 3.5. Raspberry Pi 3.



Hình 3.6. Sơ đồ các chân của Raspberry Pi 3.

3.2.4. LM 2596:

Mạch giảm áp DC LM2596 3A nhỏ gọn có khả năng giảm áp từ 30V xuống 1.5V mà vẫn đạt hiệu suất cao (92%) . Thích hợp cho các ứng dụng chia nguồn, hạ áp, cấp cho các thiết bị như camera, motor, robot,…

Thông số kĩ thuật:

- Điện áp đầu vào: Từ 3V đến 30V.
- Điện áp đầu ra: Điều chỉnh được trong khoảng 1.5V đến 30V.
- Dòng đáp ứng tối đa là 3A.
- Hiệu suất: 92%
- Công suất: 15W
- Kích thước: 45 (dài) * 20 (rộng) * 14 (cao) mm

Hình 3.7. Mạch giảm áp LM 2596.

3.2.5. Mạch chuyển đổi HDMI sang LCD:

Hình 3.8. Mạch chuyển đổi HDMI sang LCD.

3.2.6. Màn hình LCD:

Hình 3.9. Màn hìn LCD.

3.2.7. Biến trở:

Biến trở là một điện trở mà giá trị của nó có thể điều chỉnh được. Biến trở gồm có 3 chân: 2 chân nguồn (chân âm GND và chân dương VCC 5V), 1 chân ở giữa là chân xuất ra giá trị điện áp thay đổi từ 0-5V khi xoay biến trở. Ở đây, nhóm thực hiện chọn biến trở 10kΩ.

Hình 3.10. Biến trở.

Chương 4. THỰC NGHIỆM MÔ HÌNH HỆ THỐNG HIỂN THỊ THÔNG TIN TRÊN Ô TÔ

4.1. Thiết kế đồ họa trên Python:

4.1.1. Các code cơ bản thiết kế giao diện táp lô:

4.1.1.1. Code tạo cửa sổ full màn hình:

window = Tk()

window.title("Testing Update Values")

window.attributes("-fullscreen", True)

4.1.1.2. Code bôi đen toàn bộ màn hình:

*canvas = tk.Canvas(master=window, width=1370*t, height=770*t2, bg='black')*

canvas.grid(row=0, column=0)

Với t = 0.478, t2 = 0.67 là tỉ lệ màn hình ngang và dọc.

4.1.1.3. Code vẽ hình oval:

*canvas.create_oval(92.5*t, 250*t2, 592.5*t, 750*t2, outline="blue", width=5)*

Để vẽ hình oval, ta dùng lệnh canvas.create_oval bằng cách xác định hai điểm A và C của hình chữ nhật ABCD. Với tọa độ điểm A là x1 = 92.5*t, y1 = 250*t2 và tọa điểm C là x2 = 592.5*t, y2 = 750*t2. (Nếu hai tọa độ ABCD là hình vuông thì hình oval vẽ được sẽ là hình tròn.)

4.1.1.4. Code vẽ hình quạt:

canvas.create_arc(371.5, 30, 610.5, 361, start=40, extent=100, outline="green", style=tk.ARC, width=5)

Để vẽ hình quạt, ta dùng lệnh canvas.create_arc bằng cách xác định hai điểm A và C của hình chữ nhật ABCD. Với tọa độ điểm A là x1 = 371.5, y1 = 30 và tọa điểm C là x2 = 610.5, y2 = 361. Bắt đầu vẽ vị trí ở với trục nằm ngang một góc 40 độ và góc quay 100.

4.1.1.5. Code vẽ vạch cho đường tròn:

line2 = canvas.create_line(400, 85, 410, 95, fill="snow", width=3)

Để vẽ đường thẳng bất kỳ ta dùng lệnh canvas.create_line bằng cách xác định tọa độ hai điểm đầu và điểm cuối (x1= 400, y1 = 85), (x2 = 410, y2 = 95).

4.1.1.6. Code viết chữ lên giao diện:

canvas.create_text(235, 530, fill= "snow", font="VNI-Dom 18", text="0")

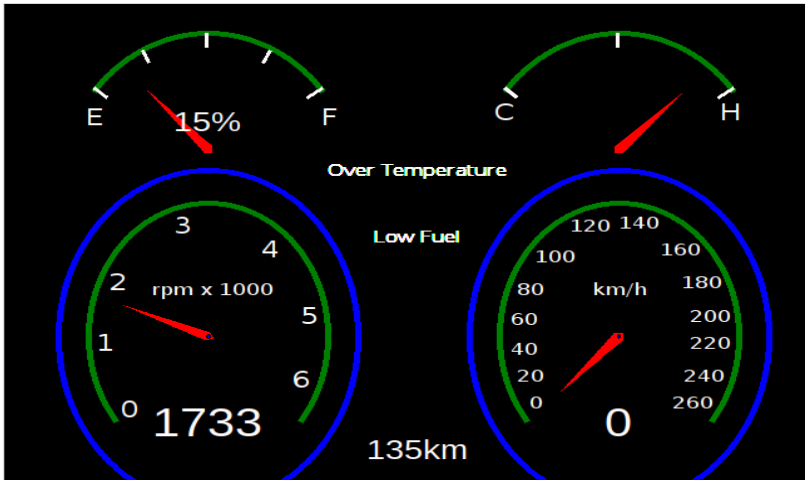
Để viết chữ lên giao diện ta dùng lệnh canvas.create_text với (x = 235, y = 530) là vị trí chữ hiển thị, fill = “snow” là chữ trắng và font = “VNI-Dom 18’’ là font chữ VNI kích cỡ 18 , text = “0” là kí tự mong muốn xuất hiện trên giao diện.

4.1.1.7. Code vẽ kim đồng hồ:

```
xT1 = 342*t
yT1 = 500*t2
R = 160*t
x1 = x*(math.pi)/180
xN = xT1 - R*math.cos(x1)
yN = yT1 - R*math.sin(x1)
r = 10*t
xM = xT1 - r*math.cos(x1+math.pi/2)
yM = yT1 - r*math.sin(x1+math.pi/2)
xP = xT1 - r*math.cos(x1+math.pi)
yP = yT1 - r*math.sin(x1+math.pi)
xQ = xT1 - r*math.cos(x1-math.pi/2)
yQ = yT1 - r*math.sin(x1-math.pi/2)
tdkim = [xN, yN, xM, yM, xP, yP, xQ, yQ]
kim = canvas.create_polygon(tdkim, fill="red")
```

Để vẽ hình kim đồng hồ, ta vẽ hình thoi MNPQ với N có tọa độ (xN, yN), M(xM,yM), P(xP,yP), Q(xQ,yQ) với x là góc hợp bởi đường chéo lớn nhất của hình thoi và trục nằm ngang. T là điểm giao nhau của hai đường chéo của hình thoi. R, r lần lượt là khoảng cách từ tâm tới hai điểm N và Q.

4.1.1.8. Kết quả thiết kế được:



Hình 4.1. Kết quả thiết kế giao diện hiển thị.

4.2. Thiết lập truyền tín hiệu:

Tìm hiểu giao tiếp SPI và việc truyền tín hiệu trong mạng CAN. Giả lập tín hiệu đầu vào là sự thay đổi của điện áp biến trở thông qua Arduino Uno và Raspberry Pi 3 sẽ là mạch nhận. Cả hai phần này sẽ được nối với nhau qua 2 dây CAN của MCP 2515.

4.2.1. Thiết lập giao tiếp SPI:

Để thiết lập giao tiếp SPI đầu tiên ta cần phải cài đặt trạng thái cho các chân SCK, MOSI, MISO, SS. Thanh ghi DDRB có 8 bit dùng để xác định input hay output cho các chân này. Ở đây quy định bit 0 sẽ là input và bit 1 sẽ là output.

```
void SPI_init()
{
        DDRB = (0 << MISO) | (1 << MOSI) | (1 << SS) | (1 << SCK);
        SPCR = (1 << SPE) | (1 << MSTR) | (1 << SPR0);
}
```

Hình 4.2. Code thiết lập giao tiếp SPI.

7	6	5	4	3	2	1	0
DDRB7	DDRB6	DDRB5	DDRB4	DDRB3	DDRB2	DDRB1	DDRB0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Hình 4.3. Thanh ghi DDRB.

Ta biết, SPI trên dòng AVR được vận hành bởi 3 thanh ghi bao gồm thanh ghi điều khiển SPCR , thanh ghi trạng thái SPSR và thanh ghi dữ liệu SPDR. Thanh ghi SPCR (SPI Control Register): là 1 thanh ghi 8 bit điều khiển tất cả hoạt động của SPI.

7	6	5	4	3	2	1	0	
SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

Hình 4.4. Thanh ghi SPCR.

Bit 7 – SPIE (SPI Interrupt Enable): Cho phép ngắt SPI. Nếu bit này có giá trị bằng 1 thì 1 ngắt sẽ xảy ra sau khi một gói dữ liệu được truyền hoặc nhận.

Bit 6 – SPE (SPI Enable): nếu bit này được set bằng 1 cho phép SPI hoạt động. Nếu SPE = 0 thì module SPI dừng hoạt động.

Bit 5 – DORD (Data Order) bit này chỉ định thứ tự dữ liệu các bit được truyền và nhận trên các đường MISO và MOSI, khi DORD=0 bit có trọng số lớn nhất của dữ liệu được truyền trước (MSB) ngược lại khi DORD=1 bit LSB được truyền trước.

Bit 4 – MSTR (Master/Slave Select) : Nếu MSTR = 1 thì chip được nhận diện là Master, ngược lại MSTR = 0 thì chip là Slave.

Bit 3 và 2 – CPOL và CPHA đây chính là 2 bit xác lập cực của xung giữ nhịp và cạnh sample dữ liệu mà chúng ta đã khảo sát trong phần đầu. Sự kết hợp 2 bit này tạo thành 4 chế độ hoạt động của SPI.[5]

4.2.2. Thiết lập ADC:

Thiết lập bộ chuyển đổi Analog to Digital Converter để chuyển đổi dữ liệu tương tự sang dữ liệu số. Chip AVR ATmega328 của Atmel có tích hợp sẵn các bộ chuyển đổi ADC với độ phân giải 10 bit. Có tất cả 8 kênh đơn (các chân ADC0 đến ADC7), 16 tổ hợp chuyển đổi dạng so sánh, trong đó có 2 kênh so sánh có thể khuyếch đại.

Có 4 thanh trong bộ ADC trên AVR trong đó có 2 thanh ghi data chứa dữ liệu sau khi chuyển đổi, 2 thanh ghi điều khiển và chứa trạng thái của ADC:

- ADMUX (ADC Multiplexer Selection Register): là 1 thanh ghi 8 bit điều khiển việc chọn điện áp tham chiếu, kênh và chế độ hoạt động của ADC. Chức năng của từng bit trên thanh ghi này sẽ được trình bày cụ thể như sau.

7	6	5	4	3	2	1	0	
REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	ADMUX
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
0	0	0	0	0	0	0	0	

Hình 4.5. Thanh ghi ADMUX.

- Bit 7:6- REFS1:0 (Reference Selection Bits): là các bit chọn điện áp tham chiếu cho ADC, 1 trong 3 nguồn điện áp tham chiếu có thể được chọn là: điện áp ngoài từ chân VREF, điện áp tham chiếu nội 2.56V hoặc điện áp AVCC. Bảng 2 tóm tắt giá trị các bit và điện áp tham chiếu tương ứng.

Bảng 2: Chọn điện áp tham chiếu

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, Internal Vref turned off
0	1	AVCC with external capacitor at AREF pin
1	0	Reserved
1	1	Internal 2.56V Voltage Reference with external capacitor at AREF pin

Hình 4.6. Điện áp tham chiếu.

- Bits 4:0-MUX4:0 (Analog Channel and Gain Selection Bits): là 5 bit cho phép chọn kênh, chế độ và cả hệ số khuyếch đại cho ADC. Do bộ ADC trên AVR có nhiều kênh và cho phép thực hiện chuyển đổi ADC kiểu so sánh (so sánh điện áp giữa 2 chân analog) nên trước khi thực hiện chuyển đổi, chúng ta cần set các bit MUX để chọn kênh và chế độ cần sử dụng. Bảng 3 tóm tắt các chế độ hoạt động của ADC thông qua các giá trị của các bit MUX. Trong bảng này, ứng với các giá trị từ 00000 đến 00111 (nhị phân), các kênh ADC được chọn ở chế độ đơn kênh (tín hiệu input lấy trực tiếp từ các chân analog và so sánh với 0V), giá trị từ 01000 đến 11101 tương ứng với chế độ chuyển đổi so sánh.

MUX4..0	Single Ended Input	Positive Differential Input	Negative Differential Input	Gain
00000	ADC0	N/A		
00001	ADC1			
00010	ADC2			
00011	ADC3			
00100	ADC4			
00101	ADC5			
00110	ADC6			
00111	ADC7			

Hình 4.7. Chế độ hoạt động ADC.

- ADCSRA (ADC Control and Status RegisterA): là thanh ghi chính điều khiển hoạt động và chứa trạng thái của module ADC.

7	6	5	4	3	2	1	0	
ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
0	0	0	0	0	0	0	0	

Hình 4.8. Thanh ghi ADCSRA.

- Bit 7 - ADEN(ADC Enable): viết giá trị 1 vào bit này tức bạn đã cho phép module ADC được sử dụng. Tuy nhiên khi ADEN=1 không có nghĩa là ADC đã hoạt

động ngay, bạn cần set một bit khác lên 1 để bắt đầu quá trình chuyển đổi, đó là bit ADSC.

- Bit 6 - ADSC(ADC Start Conversion): set bit này lên 1 là bắt đầu khởi động quá trình chuyển đổi. Trong suốt quá trình chuyển đổi, bit ADSC sẽ được giữ nguyên giá trị 1, khi quá trình chuyển đổi kết thúc (tự động), bit này sẽ được trả về 0
- Bit 4 – ADIF(ADC Interrupt Flag): cờ báo ngắt. Khi một chuyển đổi kết thúc, bit này tự động được set lên 1, vì thế người dùng cần kiểm tra giá trị bit này trước khi thực hiện đọc giá trị chuyển đổi để đảm bảo quá trình chuyển đổi đã thực sự hoàn tất.
- Bit 3 – ADIE(ADC Interrupt Enable): bit cho phép ngắt, nếu bit này được set bằng 1 và bit cho phép ngắt toàn cục (bit I trong thanh ghi trạng thái của chip) được set, một ngắt sẽ xảy ra khi một quá trình chuyển đổi ADC kết thúc và các giá trị chuyển đổi đã được cập nhật (các giá trị chuyển đổi chứa trong 2 thanh ghi ADCL và ADCH).
- Bit 2:0 – ADPS2:0(ADC Prescaler Select Bits): các bit chọn hệ số chia xung nhịp cho ADC. ADC, cũng như tất cả các module khác trên AVR, cần được giữ nhịp bằng một nguồn xung clock. Xung nhịp này được lấy từ nguồn xung chính của chip thông qua một hệ số chia. Các bit ADPS cho phép người dùng chọn hệ số chia từ nguồn clock chính đến ADC.[6]

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Hình 4.9. Hệ số chia.

```
uint16_t adc_read(void)
{
        ADCSRA |= (1<<ADSC);
        while(ADCSRA & (1<<ADSC));
        return ADC;
}


void adc_init()
{
       ADMUX = (1<<REFS0)|(1<<MUX0)|(1<<MUX2);
       ADCSRA = (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);
}
```

Hình 4.10. Code thiết lập ADC.

4.2.3. Thiết lập các chế độ cho MCP 2515:

Như đã trình bày thì MCP 2515 có 6 chế độ: Configuration mode, Listen mode, Loop back mode, Configuration mode, Disabled mode, Normal mode, Error recognition mode. Trước khi MCP 2515 hoạt động thì phải nó phải ở mode Configuration. Khi hoạt động thì MCP 2515 chuyển tới mode Normal.

```
PORTB = (0 << SS);
SPDR = 2;
while(!(SPSR &(1<<SPIF)));
SPDR =15;
while(!(SPSR &(1<<SPIF)));
SPDR =128;
while(!(SPSR &(1<<SPIF)));
PORTB = (1 << SS);
_delay_ms (1);
```

Hình 4.11. Thiết lập chế độ Configuration.

bit 7 – bit 5 trả ra theo quy ước thì MCP 2515 có chế độ đó.

Bảng 4.1. Bảng chế độ MCP 2515.

Bit 7	Bit 6	Bit 5	Mode
0	0	0	Normal
0	0	1	Sleep
0	1	0	Loopback
0	1	1	Listen-only
1	0	0	Configuration

Phía trên là 1 đoạn lệnh dùng để gửi lệnh từ master ATMEGA328P sang MCP2515 bằng giao tiếp SPI trong 1 chu kỳ.

- Lệnh PORTB = (0 << SS); đưa chân SS về LOW bắt đầu quá trình thực hiện giao tiếp.

- Lệnh SPDR đầu tiên gửi qua sẽ là instruction cho MCP2515 ở đây 2 sẽ là lệnh write (tùy theo các instruction khác nhau các SPDR tiếp theo sẽ mang ý nghĩa khác nhau sau đây chỉ là ví dụ khi instruction là write).

- Lệnh SPDR tiếp theo là địa chỉ thanh ghi.

- Lệnh SPDR tiếp theo là giá trị thanh ghi.

- Lệnh while(!(SPSR &(1<<SPIF))); chờ cho SPI truyền xong.

- Lệnh PORTB = (1 << SS); đẩy chân SS lên cao kết thúc quá trình truyền nhận.

Instruction Name	Instruction Format	Description
RESET	1100 0000	Resets internal registers to the default state, sets Configuration mode.
READ	0000 0011	Reads data from the register beginning at selected address.
READ RX BUFFER	1001 0nm0	When reading a receive buffer, reduces the overhead of a normal READ command by placing the Address Pointer at one of four locations, as indicated by 'n,m'. **Note:** The associated RX flag bit, RXnIF (CANINTF), will be cleared after bringing $\overline{CS}$ high.
WRITE	0000 0010	Writes data to the register beginning at the selected address.
LOAD TX BUFFER	0100 0abc	When loading a transmit buffer, reduces the overhead of a normal WRITE command by placing the Address Pointer at one of six locations, as indicated by 'a,b,c'.
RTS (Message Request-to-Send)	1000 0nnn	Instructs controller to begin message transmission sequence for any of the transmit buffers. 1000 0nnn Request-to-Send for TXB2 — Request-to-Send for TXB1 — Request-to-Send for TXBO

Hình 4.12. SPI instruction set của MCP 2515.[2]

Lower Address Bits	Higher Order Address Bits							
	0000 xxxx	0001 xxxx	0010 xxxx	0011 xxxx	0100 xxxx	0101 xxxx	0110 xxxx	0111 xxxx
0000	RXF0SIDH	RXF3SIDH	RXM0SIDH	TXB0CTRL	TXB1CTRL	TXB2CTRL	RXB0CTRL	RXB1CTRL
0001	RXF0SIDL	RXF3SIDL	RXM0SIDL	TXB0SIDH	TXB1SIDH	TXB2SIDH	RXB0SIDH	RXB1SIDH
0010	RXF0EID8	RXF3EID8	RXM0EID8	TXB0SIDL	TXB1SIDL	TXB2SIDL	RXB0SIDL	RXB1SIDL
0011	RXF0EID0	RXF3EID0	RXM0EID0	TXB0EID8	TXB1EID8	TXB2EID8	RXB0EID8	RXB1EID8
0100	RXF1SIDH	RXF4SIDH	RXM1SIDH	TXB0EID0	TXB1EID0	TXB2EID0	RXB0EID0	RXB1EID0
0101	RXF1SIDL	RXF4SIDL	RXM1SIDL	TXB0DLC	TXB1DLC	TXB2DLC	RXB0DLC	RXB1DLC
0110	RXF1EID8	RXF4EID8	RXM1EID8	TXB0D0	TXB1D0	TXB2D0	RXB0D0	RXB1D0
0111	RXF1EID0	RXF4EID0	RXM1EID0	TXB0D1	TXB1D1	TXB2D1	RXB0D1	RXB1D1
1000	RXF2SIDH	RXF5SIDH	CNF3	TXB0D2	TXB1D2	TXB2D2	RXB0D2	RXB1D2
1001	RXF2SIDL	RXF5SIDL	CNF2	TXB0D3	TXB1D3	TXB2D3	RXB0D3	RXB1D3
1010	RXF2EID8	RXF5EID8	CNF1	TXB0D4	TXB1D4	TXB2D4	RXB0D4	RXB1D4
1011	RXF2EID0	RXF5EID0	CANINTE	TXB0D5	TXB1D5	TXB2D5	RXB0D5	RXB1D5
1100	BFPCTRL	TEC	CANINTF	TXB0D6	TXB1D6	TXB2D6	RXB0D6	RXB1D6
1101	TXRTSCTRL	REC	EFLG	TXB0D7	TXB1D7	TXB2D7	RXB0D7	RXB1D7
1110	CANSTAT	CANSTAT	CANSTAT	CANSTAT	CANSTAT	CANSTAT	CANSTAT	CANSTAT
1111	CANCTRL	CANCTRL	CANCTRL	CANCTRL	CANCTRL	CANCTRL	CANCTRL	CANCTRL

Hình 4.13. Địa chỉ thanh ghi trong MCP 2515.[2]

4.2.4. Thiết lập Bit Timing:

Normal Bit time là độ dài của một bit trên bus. Mỗi node trên bus phải được điều chỉnh cùng nhịp với normal bit time để có thể nhận và phát chính xác. Normal bit time là nghịch đảo của dung lượng trên bus.

Thanh ghi cấu hình (CNF3, CNF2, CNF1) điều khiển thời gian bit cho giao diện bus CAN.

```
PORTB = (0 << SS);
SPDR = 2;
while(!(SPSR &(1<<SPIF)));
SPDR =40;
while(!(SPSR &(1<<SPIF)));
SPDR =129;
while(!(SPSR &(1<<SPIF)));
PORTB = (1 << SS);
_delay_ms (1);

PORTB = (0 << SS);
SPDR = 2;
while(!(SPSR &(1<<SPIF)));
SPDR =41;
while(!(SPSR &(1<<SPIF)));
SPDR =209;
while(!(SPSR &(1<<SPIF)));
PORTB = (1 << SS);
_delay_ms (1);
```

Hình 4.14. Code thiết lập Bit timing MCP2515.

4.2.5. Xuất ID tin nhắn:

Để bắt đầu việc truyền tin nhắn thì MCP chuyển sang chế độ Normal mới có thể gửi tin nhắn.

```
PORTB = (0 << SS);
SPDR = 2;
while(!(SPSR &(1<<SPIF)));
SPDR =15;
while(!(SPSR &(1<<SPIF)));
SPDR =7;
while(!(SPSR &(1<<SPIF)));
PORTB = (1 << SS);
_delay_ms (1);
```

Hình 4.15. Chế độ Normal của MCP 2515.

Thanh ghi TXB chịu trách nhiệm việc truyền tin nhắn. Thanh ghi ID TXBnSIDL.

```
PORTB = (0 << SS);
SPDR = 2;
while(!(SPSR &(1<<SPIF)));
SPDR =49;
while(!(SPSR &(1<<SPIF)));
SPDR =0;
while(!(SPSR &(1<<SPIF)));
PORTB = (1 << SS);
_delay_ms (1);

PORTB = (0 << SS);
SPDR = 2;
while(!(SPSR &(1<<SPIF)));
SPDR =50;
while(!(SPSR &(1<<SPIF)));
SPDR =96;
while(!(SPSR &(1<<SPIF)));
PORTB = (1 << SS);
_delay_ms (1);
```

Hình 4.16. Code xuất ID.

Sau khi xuất ID thì sẽ có thanh ghi số databyte của tin nhắn chứa dữ liệu cần truyền.

```
PORTB = (0 << SS);
SPDR = 2;
while(!(SPSR &(1<<SPIF)));
SPDR =53;
while(!(SPSR &(1<<SPIF)));
SPDR =8;
while(!(SPSR &(1<<SPIF)));
PORTB = (1 << SS);
_delay_ms (1);
```

Hình 4.17. Code xuất số data byte.

4.2.5. Đọc giá trị biến trở và ghi vào data byte:

Giá trị Arduino đọc được khi xoay điện trở chạy từ 0 đến 1023. Ở đây ta muốn giá trị biến trở chạy từ 0 đến 255 nên ta chia cho 4.

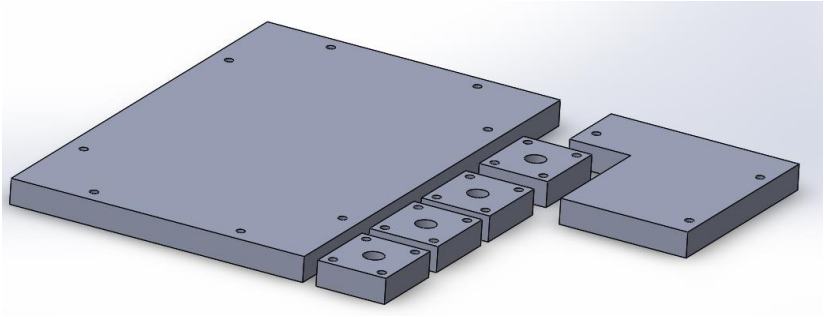
```
giatriADC = (adc_read()/4);

PORTB = (0 << SS);
SPDR =65;
while(!(SPSR &(1<<SPIF)));
SPDR =giatriADC;
while(!(SPSR &(1<<SPIF)));
PORTB = (1 << SS);
_delay_ms (1);
```

Hình 4.18. Code đọc giá trị biến trở.

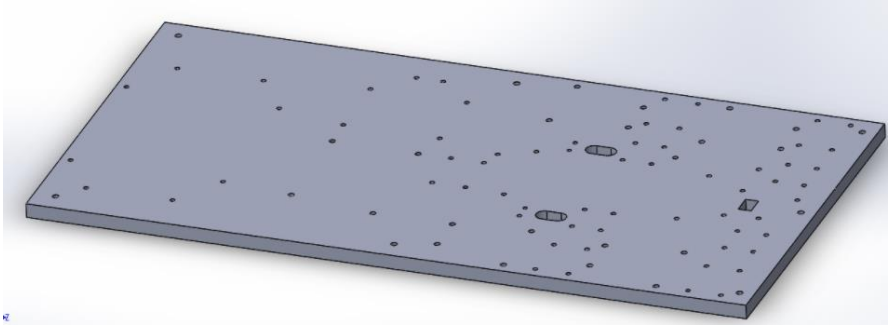
4.3. Thiết kế mô hình thực nghiệm:

4.3.1. Thiết kế khung mô hình:



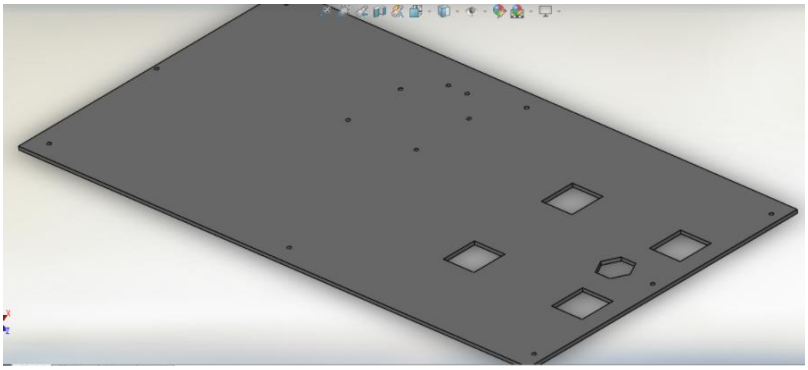
Thiết kế tấm chắn cho LCD, Raspberry Pi3 và các biến trở bằng mica 3mm nhựa trong.

Hình 4.19. Khung mô hình hệ thống hiển thị thông tin trên Solidwork



Hình 4.20. Thiết kế khung mô hình trên Solidwork.

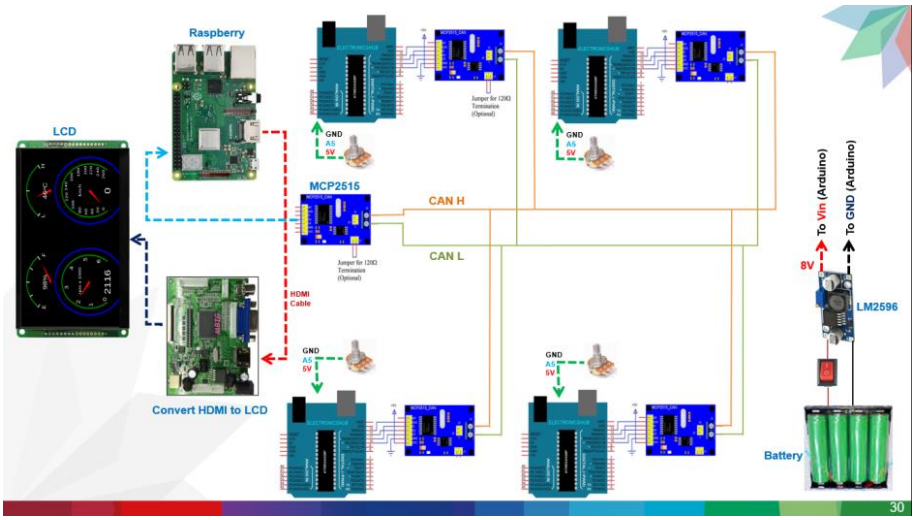
Thiết kế đế cho mô hình sử dụng mica đen 5 mm giúp cố định vị trí các linh kiện trên mô hình.



Hình 4.21. Thiết kế mô hình trên Solidwork.

Thiết kế tấm chắn phía trên bằng mica nhựa trong 3mm cho mô hình, giúp bảo vệ các kinh kiện khỏi các tác động bên ngoài.

4.3.2. Kết quả mô hình mô phỏng:



Hình 4.22. Sơ đồ mô phỏng hệ thống hiển thị hệ thống hiển thị thông tin.

Trên đây là sơ đồ thể hiện sự kết nối giữa các thiết bị phần cứng với nhau. Arduino Uno giao tiếp với MCP 2515 bằng chuẩn giao tiếp SPI thông qua 4 chân SCK, SO, SI, CS. Ở mô hình này, ta có 4 node gửi (mỗi node bao gồm 1 mạch Arduino Uno và 1 MCP 2515). Giả lập tín hiệu đầu vào là giá trị thay đổi của biến trở. Biến trở sẽ gửi tín hiệu điện áp đến chân A5 của Arduino Uno. Tín hiệu này sẽ truyền đến MCP 2515 rồi sau đó gửi lên đường truyền giao tiếp CAN. Ở node nhận (bao gồm Raspberry Pi 3 và MCP 2515) sẽ lấy dữ liệu từ hai dây CAN về xử lý rồi xuất lên màn hình LCD thông qua mạch chuyển đổi HDMI sang LCD.

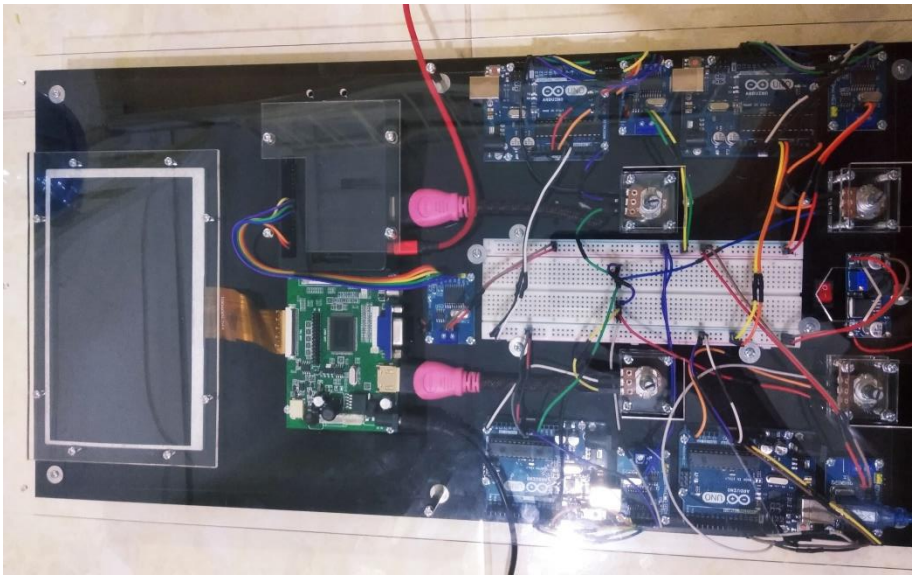
Bảng 4.2. Bảng nối dây giữa các chân của Arduino Uno với MCP 2515 và biến trở.

Arduino Uno	MCP 2515
5V	VCC
GND	GND
10	CS
11	SI
12	SO
13	SCK

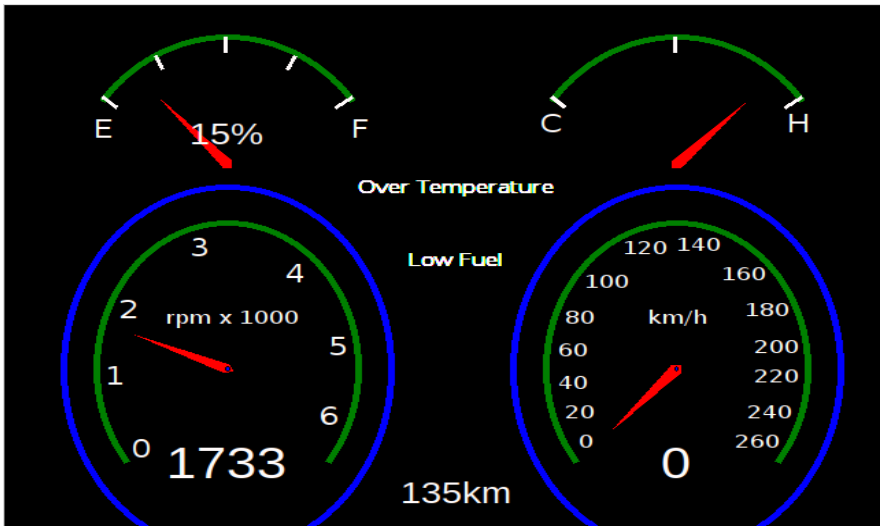
Arduino Uno	Biến trở
5V	1
GND	3
A5	2

Bảng 4.3. Bảng nối dây giữa các chân của Raspberry Pi 3 với MCP 2515.

Raspberry Pi 3	MCP 2515
5V	VCC
GND	GND
GPIO 8	CS
GPIO 9	SO
GPIO 10	SI
GPIO 11	SCK



Hình 4.23. Mô hình thực tế.



Hình 4.24. Màn hình hiển thị thông tin.

Màn hình được thiết kế bằng ngôn ngữ lập trình Python bao gồm 4 đồng hồ: đồng hồ tốc độ xe, đồng hồ tốc độ động cơ, đồng hồ báo nhiên liêu, đồng hồ báo nhiệt độ nước làm mát động cơ. Ngoài ra, màn hình còn hiển thị giá trị ước tính quãng đường đi được với lượng nhiên liệu còn lại. Đồng thời, đưa ra cảnh báo người dùng khi lượng nhiên liệu dưới mức 20% so với dung tích bình xăng hay khi nhiệt độ nước làm mát động cơ quá cao.

Chương 5. KẾT LUẬN VÀ KIẾN NGHỊ

5.1. Kết luận:

5.1.1. Kết quả đạt được:

Trong quá trình thực hiện đồ án, nhóm đã nghiên cứu và tìm hiểu về ngôn ngữ Python để lập trình cho Raspberry Pi 3 cũng như thiết kế ra giao diện hiển thị. Bên cạnh đó, còn vận dụng ngôn ngữ lập trình C để điều khiển cho Arduino Uno.

Tìm hiểu về mạng giao tiếp CAN và giao tiếp SPI để truyền và nhận dữ liệu giữa các module với nhau.

Giả lập tín hiệu cảm biến bằng biến trở và gửi tín hiệu lên táp lô thông qua các mạng giao tiếp. Thực hiện việc giao tiếp giữa nhiều node trên mạng CAN.

Xây dựng bộ tài liệu về kiến thức cơ bản mạng giao tiếp CAN trên ô tô.

Tính toán thiết kế khung mô hình bằng Solidwork. Hoàn thiện bộ mô hình mô phỏng hệ thống hiển thị thông tin sử dụng mạng CAN trên ô tô.

5.1.2. Hạn chế của đề tài:

- Thời gian để tìm hiểu và nghiên cứu đề tài còn hạn chế.
- Tài liệu khó kiếm.
- Thiếu trang thiết bị để thử nghiệm.
- Chưa thực hiện được việc hiển thị nhiều tín hiệu lên táp lô.

5.2. Kiến nghị:

- Thử nghiệm trực tiếp trên xe.
- Thực hiện được việc hiển thị nhiều tín hiệu lên táp lô.
- Hiển thị camera lên màn hình.
- Sử dụng cảm biến để làm tín hiệu đầu vào.

TÀI LIỆU THAM KHẢO

[1] Volkswagen AG, Data transfer on CAN data bus II.

[2] MCP 2515 datasheet, Internet: https://drive.google.com/file/d/1LsBm1mVzg3SHMJrnfvPywhsZO0otdi2O/view?fbclid=IwAR2SttVIAHq5ceJur7Nf2gEposwC1yO0RtvvWFoU3BL6fsSbbZjhuGccxIE

[3] Arduinokit, Mạch – arduino – uno là gì, Internet: http://arduinokit.vn/mach-arduino-uno-la-gi/

[4] Konrad Reif Ed, Automotive Mechatronics, Bosch Professional Automotive Information.

[5] Ho Tam, Giao tiếp SPI, Internet : http://www.hocavr.com/2018/06/bai-7-giao-tiep-spi.html?m=1&fbclid=IwAR0DiDEd74ea2GNMaWWM7ke0xzTWDOa9hphP8iMVCDJTk7cewHnbhQGGzfw, 2018.

[6] Ho Tam, Chuyển đổi ADC, Internet: http://www.hocavr.com/2018/06/bai-6-chuyen-oiadc.html?m=1&fbclid=IwAR1u20vgEk2zRzkJhFRKTHF93omnEano9LIB_6osLCSByXgdyAxNvkn1yv4, 2018

PHỤ LỤC 1

I. Cách cài Python và thư viện:

1. Cài đặt Python:

Để cài đặt Python, ta lên trang web www.pythonware.com để tải về phiên bản mới nhất.

2. Cài đặt Sublime Text:

Để cài đặt Sublime Text ta lên trang web www.sublimetext.com để tải về phiên bản mới nhất.

II. Cài đặt hệ điều hành cho Raspberry Pi 3.

Chuẩn bị thẻ nhớ tối thiểu 8GB để cài đặt hệ điều hành Raspberrian. Lên website raspberrypi.vn để tải hệ điều hành về laptop.

Link: https://raspberrypi.vn/huong-dan-cai-dieu-hanh-cho-raspberry-pi-2457.pi

Trong link này có hướng dẫn cách cài đặt hệ điều hành Raspberrian cho Raspberry Pi 3.

PHỤ LỤC 2

Code lập trình cho Raspberry Pi 3.

```
import tkinter as tk
import time
import math
import RPi.GPIO as GPIO
import spidev
#Gọi thư viện hình ảnh PIL
#from PIL import Image, ImageTk
GPIO.setmode(GPIO.BCM)
spi = spidev.SpiDev()
spi.open(0,0)
spi.max_speed_hz = 500000
def MCP2515_Setup():
    Value = [2,15,128]
    spi.xfer2(Value)
    Value = [2,12,60]
    spi.xfer2(Value)
```

```
    Value = [2,13,56]
    spi.xfer2(Value)
    Value = [2,40,129]
    spi.xfer2(Value)
    Value = [2,41,209]
    spi.xfer2(Value)
    Value = [2,42,0]
    spi.xfer2(Value)
    Value = [2,43,0]
    spi.xfer2(Value)
    Value = [2,96,102]
    spi.xfer2(Value)
    Value = [2,112,96]
    spi.xfer2(Value)
    Value = [2,15,7]
    spi.xfer2(Value)
def ThucHien():
    #Ham nay dung de cap nhat gia tri tu ben ngoai vao
    #Doc gia tri chua trong dia chi 114 cua MCP: RXB1SIDL, byte CAN ID Low
    address = [3,114,0]
    spi.xfer2(address)
    #Doc gia tri chua trong dia chi 118 cua MCP: RXB1D0, byte dau tien cua 8 byte
    Value = [3, 118, 0]
    print(address[2])
    print(Value[2])
    spi.xfer2(Value)
    if address[2] == 89:
        rpm=Value[2]/255*6500
        rpm2 = rpm/1000
        x = -50 + 43*rpm2
```

```
    #Code vẽ kim rpm
    xT1 = 342*t
    yT1 = 500*t2
    R = 160*t
    x1 = x*(math.pi)/180
    xN = xT1 - R*math.cos(x1)
    yN = yT1 - R*math.sin(x1)
    r = 10*t
    xM = xT1 - r*math.cos(x1+math.pi/2)
    yM = yT1 - r*math.sin(x1+math.pi/2)
    xP = xT1 - r*math.cos(x1+math.pi)
    yP = yT1 - r*math.sin(x1+math.pi)
    xQ = xT1 - r*math.cos(x1-math.pi/2)
    yQ = yT1 - r*math.sin(x1-math.pi/2)
    tdkim = [xN, yN, xM, yM, xP, yP, xQ, yQ]
    #kim = canvas.create_polygon(tdkim, fill="red")
    canvas.coords(kim, tdkim)
    speed_rpm=round(rpm)
    canvas.itemconfig(text_rpm, text=speed_rpm)

if address[2] == 88:
    #Nhập km/h
    kmh = Value[2]/255*260
    x2 = -50 + 1.07*kmh

    #Code vẽ kim km/h
    xT2 = 1027.5*t
    yT2 = 500*t2

    R2 = 160*t
    x3 = x2*(math.pi)/180
```

```
    xN2 = xT2 - (R2*math.cos(x3))
    yN2 = yT2 - (R2*math.sin(x3))

    r = 10*t
    xM2 = xT2 - r*math.cos(x3+math.pi/2)
    yM2 = yT2 - r*math.sin(x3+math.pi/2)
    xP2 = xT2 - r*math.cos(x3+math.pi)
    yP2 = yT2 - r*math.sin(x3+math.pi)
    xQ2 = xT2 - r*math.cos(x3-math.pi/2)
    yQ2 = yT2 - r*math.sin(x3-math.pi/2)

    tdkim2 = [xN2, yN2, xM2, yM2, xP2, yP2, xQ2, yQ2]
    #kim2 = canvas.create_polygon(tdkim2, fill="red")
    canvas.coords(kim2, tdkim2)
    speed=round(kmh)
    canvas.itemconfig(text_kmh, text=speed)

if address[2] == 90:
    #Nhap xang
    xang=Value[2]/255*115
    xang2=xang/115*100
    x5 = 34 + xang
    #Code vẽ kim xang
    xT6 = 342*t
    yT6 = 147
    R6 = 170*t
    x6 = x5*(math.pi)/180
    xN6 = xT6 - R6*math.cos(x6)
    yN6 = yT6 - R6*math.sin(x6)
    r6 = 10*t
    xM6 = xT6 - r6*math.cos(x6+math.pi/2)
```

```
    yM6 = yT6 - r6*math.sin(x6+math.pi/2)
    xP6 = xT6 - r6*math.cos(x6+math.pi)
    yP6 = yT6 - r6*math.sin(x6+math.pi)
    xQ6 = xT6 - r6*math.cos(x6-math.pi/2)
    yQ6 = yT6 - r6*math.sin(x6-math.pi/2)
    tdkimxang = [xN6, yN6, xM6, yM6, xP6, yP6, xQ6, yQ6]
    #kim6 = canvas.create_polygon(tdkimxang, fill="red")
    canvas.coords(kim6, tdkimxang)
    speed_xang=round(xang2)
    canvas.itemconfig(text_xang, text=str(speed_xang) + "%")

    quangduong_xang=speed_xang*9
    canvas.itemconfig(text_quangduongxang, text=str(quangduong_xang) + "km")

if address[2] == 91:
    #Nhập nhietdo
    temp=Value[2]/255*115
    temp2=temp/115*100+70
    x8 = 34 + temp

    #Code vẽ kim dong ho nhiet do
    xT7 = 1027.5*t
    yT7 = 147
    R7 = 170*t
    x7 = x8*(math.pi)/180
    xN7 = xT7 - (R7*math.cos(x7))
    yN7 = yT7 - (R7*math.sin(x7))
    r6 = 10*t
    xM7 = xT7 - r6*math.cos(x7+math.pi/2)
    yM7 = yT7 - r6*math.sin(x7+math.pi/2)
    xP7 = xT7 - r6*math.cos(x7+math.pi)
```

```
        yP7 = yT7 - r6*math.sin(x7+math.pi)
        xQ7 = xT7 - r6*math.cos(x7-math.pi/2)
        yQ7 = yT7 - r6*math.sin(x7-math.pi/2)

        tdkimtemp = [xN7, yN7, xM7, yM7, xP7, yP7, xQ7, yQ7]
        canvas.coords(kim7, tdkimtemp)

    if address[2] == 91:
        #Nhập nhietdo
        temp=Value[2]/255*115
        temp2=temp/115*255-40
        if temp2 > 150:
            canvas.create_text(690*t, 250*t2, fill= "snow", font="VNI 12", text="Over
Temperature")
        else :
            canvas.create_rectangle(530*t, 200*t2, 850*t, 300*t2,fill="black")

    if address[2] == 90:
        #Nhap xang
        xang=Value[2]/255*115
        xang2=xang/115*100
        if xang2 < 20:
            canvas.create_text(690*t, 350*t2, fill= "snow", font="VNI 12", text="Low Fuel")
        else :
            canvas.create_rectangle(600*t, 300*t2, 780*t, 400*t2,fill="black")

    #Xoa gia tri dang chua trong RXB cua MCP de cap nhat gia tri moi
    Value = [2,44,0]
    spi.xfer2(Value)
    time.sleep(0.01)
    #Sau 10ms, thi hanh ThucHien, tao thanh loop tu dong
```

```
    btn_action.after(10,ThucHien)

MCP2515_Setup()
#Code tạo cửa sổ full màn hình
window = tk.Tk()
window.title("Testing Update Values")
window.attributes("-fullscreen", True)
t = 0.478
t2 = 0.67
#t = 1
#t2 = 1
#Code bôi đen toàn bộ cửa sổ
canvas = tk.Canvas(master=window, width=1370*t, height=770*t2, bg='black')
canvas.grid(row=0, column=0)
#viết chữ lên đồng hồ
canvas.create_text(210*t, 610*t2, fill= "snow", font="VNI 17", text="0")
canvas.create_text(170*t, 510*t2, fill= "snow", font="VNI 17", text="1")
canvas.create_text(190*t, 420*t2, fill= "snow", font="VNI 17", text="2")
canvas.create_text(300*t, 335*t2, fill= "snow", font="VNI 17", text="3")
canvas.create_text(445*t, 370*t2, fill= "snow", font="VNI 17", text="4")
canvas.create_text(510*t, 470*t2, fill= "snow", font="VNI 17", text="5")
canvas.create_text(495*t, 565*t2, fill= "snow", font="VNI 17", text="6")
canvas.create_text(350*t, 430*t2, fill= "snow", font="VNI 13", text="rpm x 1000")

#vẽ hình tròn trái
canvas.create_oval(92.5*t, 250*t2, 592.5*t, 750*t2, outline="blue", width=5)
# vẽ hình quạt trái
canvas.create_arc((192.5-50)*t, 300*t2, (492.5+50)*t, (600+100)*t2, start=-40,
extent=260, outline="green",style=tk.ARC, width=5)
# ve hinh quat xang
```

```
#canvas.create_arc(510, -150, 810, 150, start=180, extent=90, outline="green",
style=tk.ARC, width=5)
#ve hinh quat xang
canvas.create_arc(44, 30, 283, 361, start=40, extent=100, outline="green", style=tk.ARC,
width=5)


canvas.create_text(1030*t, 430*t2, fill= "snow", font="VNI 13", text="km/h")
canvas.create_text(890*t, 600*t2, fill= "snow", font="VNI 13", text="0")
canvas.create_text(880*t, 560*t2, fill= "snow", font="VNI 13", text="20")
canvas.create_text(870*t, 520*t2, fill= "snow", font="VNI 13", text="40")
canvas.create_text(870*t, 475*t2, fill= "snow", font="VNI 13", text="60")
canvas.create_text(880*t, 430*t2, fill= "snow", font="VNI 13", text="80")
canvas.create_text(920*t, 380*t2, fill= "snow", font="VNI 13", text="100")
canvas.create_text(980*t, 335*t2, fill= "snow", font="VNI 13", text="120")
canvas.create_text(1060*t, 330*t2, fill= "snow", font="VNI 13", text="140")
canvas.create_text(1130*t, 370*t2, fill= "snow", font="VNI 13", text="160")
canvas.create_text(1165*t, 420*t2, fill= "snow", font="VNI 13", text="180")
canvas.create_text(1180*t, 470*t2, fill= "snow", font="VNI 13", text="200")
canvas.create_text(1180*t, 510*t2, fill= "snow", font="VNI 13", text="220")
canvas.create_text(1170*t, 560*t2, fill= "snow", font="VNI 13", text="240")
canvas.create_text(1150*t, 600*t2, fill= "snow", font="VNI 13", text="260")

# vẽ hình tròn phải
canvas.create_oval(777.5*t, 250*t2, 1277.5*t, 750*t2, outline="blue", width=5)
# vẽ hình quạt phải
canvas.create_arc((877.5-50)*t, 300*t2, (1177.5+50)*t, (600+100)*t2, start=-40,
extent=260, outline="green", style=tk.ARC, width=5)
# ve hinh quat nhiet do
#canvas.create_arc(-150, -150, 150, 150, start=-90, extent=90, outline="green",
style=tk.ARC, width=5)
```

```
#ve hinh quat nhiet do
canvas.create_arc(371.5, 30, 610.5, 361, start=40, extent=100, outline="green",
style=tk.ARC, width=5)

#viết chữ lên đồng hồ nhiệt độ
canvas.create_text(400, 110, fill= "snow", font="VNI 17", text="C")
canvas.create_text(580, 110, fill= "snow", font="VNI 17", text="H")

#ve vach C
line2 = canvas.create_line(400, 85, 410, 95, fill="snow", width=3)
#ve vach giua
line2 = canvas.create_line(1024.5*t, 30, 1024.5*t, 45, fill="snow", width=3)
#ve vach H
line2 = canvas.create_line(583, 85, 570, 95, fill="snow", width=3)


#viết chữ lên đồng hồ xăng
canvas.create_text(73, 115, fill= "snow", font="VNI 17", text="E")
canvas.create_text(260, 115, fill= "snow", font="VNI 17", text="F")
#ve vach dong ho xang
#ve vach E
line2 = canvas.create_line(72, 85, 83, 95, fill="snow", width=3)
#vach giua E va giữa
line2 = canvas.create_line(111, 47, 116, 60, fill="snow", width=3)
#vach giua giua va F

line2 = canvas.create_line(213, 47, 207, 60, fill="snow", width=3)
#ve vach giua
line2 = canvas.create_line(339*t, 30, 339*t, 45, fill="snow", width=3)
#ve vach F
line2 = canvas.create_line(255, 85, 242, 95, fill="snow", width=3)
```

```
rpm=1000
rpm2 = rpm/1000
x = -50 + 43*rpm2
#Code vẽ kim rpm
xT1 = 342*t
yT1 = 500*t2
R = 160*t
x1 = x*(math.pi)/180
xN = xT1 - R*math.cos(x1)
yN = yT1 - R*math.sin(x1)
r = 10*t
xM = xT1 - r*math.cos(x1+math.pi/2)
yM = yT1 - r*math.sin(x1+math.pi/2)
xP = xT1 - r*math.cos(x1+math.pi)
yP = yT1 - r*math.sin(x1+math.pi)
xQ = xT1 - r*math.cos(x1-math.pi/2)
yQ = yT1 - r*math.sin(x1-math.pi/2)
tdkim = [xN, yN, xM, yM, xP, yP, xQ, yQ]
kim = canvas.create_polygon(tdkim, fill="red")

#Nhập km/h
kmh = 100
x2 = -50 + 1.07*kmh

#Code vẽ kim km/h
xT2 = 1027.5*t
yT2 = 500*t2

R2 = 160*t
x3 = x2*(math.pi)/180
```

```
xN2 = xT2 - (R2*math.cos(x3))
yN2 = yT2 - (R2*math.sin(x3))

r = 10*t
xM2 = xT2 - r*math.cos(x3+math.pi/2)
yM2 = yT2 - r*math.sin(x3+math.pi/2)
xP2 = xT2 - r*math.cos(x3+math.pi)
yP2 = yT2 - r*math.sin(x3+math.pi)
xQ2 = xT2 - r*math.cos(x3-math.pi/2)
yQ2 = yT2 - r*math.sin(x3-math.pi/2)

tdkim2 = [xN2, yN2, xM2, yM2, xP2, yP2, xQ2, yQ2]
kim2 = canvas.create_polygon(tdkim2, fill="red")
#Nhap xang
xang = 0
x5 = 34 + xang
#Code vẽ kim xang
xT6 = 342*t
yT6 = 147
R6 = 170*t
x6 = x5*(math.pi)/180
xN6 = xT6 - R6*math.cos(x6)
yN6 = yT6 - R6*math.sin(x6)
r6 = 10*t
xM6 = xT6 - r6*math.cos(x6+math.pi/2)
yM6 = yT6 - r6*math.sin(x6+math.pi/2)
xP6 = xT6 - r6*math.cos(x6+math.pi)
yP6 = yT6 - r6*math.sin(x6+math.pi)
xQ6 = xT6 - r6*math.cos(x6-math.pi/2)
yQ6 = yT6 - r6*math.sin(x6-math.pi/2)
tdkimxang = [xN6, yN6, xM6, yM6, xP6, yP6, xQ6, yQ6]
```

```
kim6 = canvas.create_polygon(tdkimxang, fill="red")

#Nhập nhietdo
temp = 0
x8 = 34 + temp

#Code vẽ kim dong ho nhiet do
xT7 = 1027.5*t
yT7 = 147

R7 = 170*t
x7 = x8*(math.pi)/180
xN7 = xT7 - (R7*math.cos(x7))
yN7 = yT7 - (R7*math.sin(x7))

r6 = 10*t
xM7 = xT7 - r6*math.cos(x7+math.pi/2)
yM7 = yT7 - r6*math.sin(x7+math.pi/2)
xP7 = xT7 - r6*math.cos(x7+math.pi)
yP7 = yT7 - r6*math.sin(x7+math.pi)
xQ7 = xT7 - r6*math.cos(x7-math.pi/2)
yQ7 = yT7 - r6*math.sin(x7-math.pi/2)

tdkimtemp = [xN7, yN7, xM7, yM7, xP7, yP7, xQ7, yQ7]
kim7 = canvas.create_polygon(tdkimtemp, fill="red")

#Text chi so rpm
xs=342*t # Toa do x
ys=630*t2 # Toa do y
text_rpm=canvas.create_text(xs, ys, font="Arial 30", fill = "snow", text="0")
```

```
#Text chi so km/h
xs1=1027*t # Toa do x
ys1=630*t2 # Toa do y
text_kmh=canvas.create_text(xs1, ys1, font="Arial 30", fill = "snow", text="0")

#Text chi so xang
xs2=342*t # Toa do x
ys2=120 # Toa do y
text_xang=canvas.create_text(xs2, ys2, font="Arial 20", fill = "snow", text= "50")

#Text quang duong di duoc voi luong xang con lai
xs2=690*t # Toa do x
ys2=450 # Toa do y
text_quangduongxang=canvas.create_text(xs2, ys2, font="Arial 20", fill = "snow", text=
"0")

#Tâm hình tròn trái
canvas.create_oval(339*t, 497*t2, 345*t, 503*t2, fill="blue")
#Tâm hình tròn phải
canvas.create_oval(1024.5*t, 497*t2, 1030*t, 503*t2, fill="blue")

#pilImage19 = Image.open("camera.jpg")
#pilImage19 = pilImage19.resize((670*t, 210*t), Image.ANTIALIAS)
#image19 = ImageTk.PhotoImage(pilImage19)
#imagesprite19 = canvas.create_image(685*t,125*t,image=image19)

#Thu thuat tao ra 1 nut ao (khong hien thi) de gan 1 hanh dong nao do
btn_action = tk.Button(master=window,text="Action")
btn_action.after(100,ThucHien)
```

```
window.mainloop()
```

Code lập trình cho Arduino Uno 1 (gửi giá trị nhiệt độ nước làm mát động cơ).

```
#include <avr/io.h>
#ifndef F_CPU
#define F_CPU 2000000UL
#endif
#include <util/delay.h>
#define SS PORTB2
#define MOSI PORTB3
#define MISO PORTB4
#define SCK PORTB5

void SPI_init()
{
  DDRB = (0 << MISO) | (1 << MOSI) | (1 << SS) | (1 << SCK);
  SPCR = (1 << SPE) | (1 << MSTR) | (1 << SPR0);//Thiet lap SPI Master cua
ATmega328P
}

void adc_init()
{
  ADMUX = (1<<REFS0)|(1<<MUX0)|(1<<MUX2);  //Chon chan A5 la input cua bien
tro. MUX quy định chân input A0=>A5
  //ADMUX = (1<<REFS0)|(1<<MUX2);   //Chon chan A4 la input cua bien tro
  ADCSRA = (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);//Ti le=128
}

uint16_t adc_read(void)
{
  ADCSRA |= (1<<ADSC);
```

```
  while(ADCSRA & (1<<ADSC));
  return ADC;
}
void MCP_setup()
{
  //Thiet lap che do MCP Configuration de cai dat cac Register (Tham khao Datasheeet
cua MCP2115
  PORTB = (0 << SS);
  SPDR = 2;// Moi lan gan gia tri cho SPDR, ATmega tu dong kich hoat che do truyen
SPI
  while(!(SPSR &(1<<SPIF)));//Cho cho SPI truyen xong
  SPDR =15;
  while(!(SPSR &(1<<SPIF)));
  SPDR =128;
  while(!(SPSR &(1<<SPIF)));
  PORTB = (1 << SS);
  _delay_ms (1);

  PORTB = (0 << SS);
  SPDR = 2;
  while(!(SPSR &(1<<SPIF)));
  SPDR =40;
  while(!(SPSR &(1<<SPIF)));
  SPDR =129;
  while(!(SPSR &(1<<SPIF)));
  PORTB = (1 << SS);
  _delay_ms (1);

  PORTB = (0 << SS);
  SPDR = 2;
  while(!(SPSR &(1<<SPIF)));
```

```
  SPDR =41;
  while(!(SPSR &(1<<SPIF)));
  SPDR =209;
  while(!(SPSR &(1<<SPIF)));
  PORTB = (1 << SS);
  _delay_ms (1);
}

void MCP__setNormal()
{
  //Set MCP o che do Normal
  PORTB = (0 << SS);
  SPDR = 2;
  while(!(SPSR &(1<<SPIF)));
  SPDR =15;
  while(!(SPSR &(1<<SPIF)));
  SPDR =7;
  while(!(SPSR &(1<<SPIF)));
  PORTB = (1 << SS);
  _delay_ms (1);

}

volatile uint8_t giatriADC = 0;
int main(void)
{
  adc_init();
  SPI_init();
  MCP_setup();
  MCP__setNormal();
  while (1) {
```

```
// Doc gia tri cam bien
giatriADC = (adc_read()/4);

// Xuat ID cua CAN
PORTB = (0 << SS);
SPDR = 2;
while(!(SPSR &(1<<SPIF)));
SPDR =49;
while(!(SPSR &(1<<SPIF)));
SPDR =0;
while(!(SPSR &(1<<SPIF)));
PORTB = (1 << SS);
_delay_ms (1);

PORTB = (0 << SS);
SPDR = 2;
while(!(SPSR &(1<<SPIF)));
SPDR =50;
while(!(SPSR &(1<<SPIF)));
SPDR =91;
while(!(SPSR &(1<<SPIF)));
PORTB = (1 << SS);
_delay_ms (1);

//Khai bao do dai CAN 8 byte
PORTB = (0 << SS);
SPDR = 2;
while(!(SPSR &(1<<SPIF)));
SPDR =53;
while(!(SPSR &(1<<SPIF)));
SPDR =8;
```

```
    while(!(SPSR &(1<<SPIF)));
    PORTB = (1 << SS);
    _delay_ms (1);

    //Load TX Buffer: Load 8 byte du lieu len Buffer 0, trong do co 1 byte giatriADC duoc
cap nhat tu cam bien
    PORTB = (0 << SS);
    SPDR =65;
    while(!(SPSR &(1<<SPIF)));
    SPDR =giatriADC;
    while(!(SPSR &(1<<SPIF)));
    PORTB = (1 << SS);
    _delay_ms (1);

    // Request to send de gui tin hieu luu tren Buffer di len CAN
    PORTB = (0 << SS);
    SPDR = 129;
    while(!(SPSR &(1<<SPIF)));
    PORTB = (1 << SS);
    _delay_ms (1);

    _delay_ms (10);
  }

}
```

Code lập trình cho Arduino Uno 2 (gửi giá trị tốc độ xe).

```
#include <avr/io.h>
#ifndef F_CPU
#define F_CPU 2000000UL
#endif
```

```
#include <util/delay.h>
#define SS PORTB2
#define MOSI PORTB3
#define MISO PORTB4
#define SCK PORTB5

void SPI_init()
{
  DDRB = (0 << MISO) | (1 << MOSI) | (1 << SS) | (1 << SCK);
  SPCR = (1 << SPE) | (1 << MSTR) | (1 << SPR0);//Thiet lap SPI Master cua
ATmega328P
}

void adc_init()
{
  ADMUX = (1<<REFS0)|(1<<MUX0)|(1<<MUX2);  //Chon chan A5 la input cua bien
tro. MUX quy định chân input A0=>A5
  //ADMUX = (1<<REFS0)|(1<<MUX2);   //Chon chan A4 la input cua bien tro
  ADCSRA = (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);//Ti le=128
}

uint16_t adc_read(void)
{
  ADCSRA |= (1<<ADSC);
  while(ADCSRA & (1<<ADSC));
  return ADC;
}
void MCP_setup()
{
  //Thiet lap che do MCP Configuration de cai dat cac Register (Tham khao Datasheeet
cua MCP2115
```

```
  PORTB = (0 << SS);
  SPDR = 2;// Moi lan gan gia tri cho SPDR, ATmega tu dong kich hoat che do truyen
SPI
  while(!(SPSR &(1<<SPIF)));//Cho cho SPI truyen xong
  SPDR =15;
  while(!(SPSR &(1<<SPIF)));
  SPDR =128;
  while(!(SPSR &(1<<SPIF)));
  PORTB = (1 << SS);
  _delay_ms (1);

  PORTB = (0 << SS);
  SPDR = 2;
  while(!(SPSR &(1<<SPIF)));
  SPDR =40;
  while(!(SPSR &(1<<SPIF)));
  SPDR =129;
  while(!(SPSR &(1<<SPIF)));
  PORTB = (1 << SS);
  _delay_ms (1);

  PORTB = (0 << SS);
  SPDR = 2;
  while(!(SPSR &(1<<SPIF)));
  SPDR =41;
  while(!(SPSR &(1<<SPIF)));
  SPDR =209;
  while(!(SPSR &(1<<SPIF)));
  PORTB = (1 << SS);
  _delay_ms (1);
}
```

```
void MCP__setNormal()
{
  //Set MCP o che do Normal
  PORTB = (0 << SS);
  SPDR = 2;
  while(!(SPSR &(1<<SPIF)));
  SPDR =15;
  while(!(SPSR &(1<<SPIF)));
  SPDR =7;
  while(!(SPSR &(1<<SPIF)));
  PORTB = (1 << SS);
  _delay_ms (1);

}

volatile uint8_t giatriADC = 0;
int main(void)
{
  adc_init();
  SPI_init();
  MCP_setup();
  MCP__setNormal();
  while (1) {
    // Doc gia tri cam bien
    giatriADC = (adc_read()/4);

    // Xuat ID cua CAN
    PORTB = (0 << SS);
    SPDR = 2;
    while(!(SPSR &(1<<SPIF)));
```

```
SPDR =49;
while(!(SPSR &(1<<SPIF)));
SPDR =0;
while(!(SPSR &(1<<SPIF)));
PORTB = (1 << SS);
_delay_ms (1);

PORTB = (0 << SS);
SPDR = 2;
while(!(SPSR &(1<<SPIF)));
SPDR =50;
while(!(SPSR &(1<<SPIF)));
SPDR =92;
while(!(SPSR &(1<<SPIF)));
PORTB = (1 << SS);
_delay_ms (1);

//Khai bao do dai CAN 8 byte
PORTB = (0 << SS);
SPDR = 2;
while(!(SPSR &(1<<SPIF)));
SPDR =53;
while(!(SPSR &(1<<SPIF)));
SPDR =8;
while(!(SPSR &(1<<SPIF)));
PORTB = (1 << SS);
_delay_ms (1);

//Load TX Buffer: Load 8 byte du lieu len Buffer 0, trong do co 1 byte giatriADC duoc cap nhat tu cam bien
PORTB = (0 << SS);
```

```
    SPDR =65;
    while(!(SPSR &(1<<SPIF)));
    SPDR =giatriADC;
    while(!(SPSR &(1<<SPIF)));
    PORTB = (1 << SS);
    _delay_ms (1);

    // Request to send de gui tin hieu luu tren Buffer di len CAN
    PORTB = (0 << SS);
    SPDR = 129;
    while(!(SPSR &(1<<SPIF)));
    PORTB = (1 << SS);
    _delay_ms (1);
    _delay_ms (10);

  }

}
```

Code lập trình Arduino Uno 3 (gửi tín hiệu tốc độ động cơ).

```
#include <avr/io.h>
#ifndef F_CPU
#define F_CPU 2000000UL
#endif
#include <util/delay.h>
#define SS PORTB2
#define MOSI PORTB3
#define MISO PORTB4
#define SCK PORTB5

void SPI_init()
{
```

```
  DDRB = (0 << MISO) | (1 << MOSI) | (1 << SS) | (1 << SCK);
  SPCR = (1 << SPE) | (1 << MSTR) | (1 << SPR0);//Thiet lap SPI Master cua ATmega328P
}

void adc_init()
{
  ADMUX = (1<<REFS0)|(1<<MUX0)|(1<<MUX2);  //Chon chan A5 la input cua bien tro. MUX quy định chân input A0=>A5
  //ADMUX = (1<<REFS0)|(1<<MUX2);   //Chon chan A4 la input cua bien tro
  ADCSRA = (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);//Ti le=128
}

uint16_t adc_read(void)
{
  ADCSRA |= (1<<ADSC);
  while(ADCSRA & (1<<ADSC));
  return ADC;
}
void MCP_setup()
{
  //Thiet lap che do MCP Configuration de cai dat cac Register (Tham khao Datasheeet cua MCP2115
  PORTB = (0 << SS);
  SPDR = 2;// Moi lan gan gia tri cho SPDR, ATmega tu dong kich hoat che do truyen SPI
  while(!(SPSR &(1<<SPIF)));//Cho cho SPI truyen xong
  SPDR =15;
  while(!(SPSR &(1<<SPIF)));
  SPDR =128;
  while(!(SPSR &(1<<SPIF)));
```

```
  PORTB = (1 << SS);
  _delay_ms (1);

  PORTB = (0 << SS);
  SPDR = 2;
  while(!(SPSR &(1<<SPIF)));
  SPDR =40;
  while(!(SPSR &(1<<SPIF)));
  SPDR =129;
  while(!(SPSR &(1<<SPIF)));
  PORTB = (1 << SS);
  _delay_ms (1);

  PORTB = (0 << SS);
  SPDR = 2;
  while(!(SPSR &(1<<SPIF)));
  SPDR =41;
  while(!(SPSR &(1<<SPIF)));
  SPDR =209;
  while(!(SPSR &(1<<SPIF)));
  PORTB = (1 << SS);
  _delay_ms (1);
}

void MCP__setNormal()
{
  //Set MCP o che do Normal
  PORTB = (0 << SS);
  SPDR = 2;
  while(!(SPSR &(1<<SPIF)));
  SPDR =15;
```

```
  while(!(SPSR &(1<<SPIF)));
  SPDR =7;
  while(!(SPSR &(1<<SPIF)));
  PORTB = (1 << SS);
  _delay_ms (1);

}

volatile uint8_t giatriADC = 0;
int main(void)
{
  adc_init();
  SPI_init();
  MCP_setup();
  MCP__setNormal();
  while (1) {
    // Doc gia tri cam bien
    giatriADC = (adc_read()/4);

    // Xuat ID cua CAN
    PORTB = (0 << SS);
    SPDR = 2;
    while(!(SPSR &(1<<SPIF)));
    SPDR =49;
    while(!(SPSR &(1<<SPIF)));
    SPDR =0;
    while(!(SPSR &(1<<SPIF)));
    PORTB = (1 << SS);
    _delay_ms (1);

    PORTB = (0 << SS);
```

```
SPDR = 2;
while(!(SPSR &(1<<SPIF)));
SPDR =50;
while(!(SPSR &(1<<SPIF)));
SPDR =93;
while(!(SPSR &(1<<SPIF)));
PORTB = (1 << SS);
_delay_ms (1);

//Khai bao do dai CAN 8 byte
PORTB = (0 << SS);
SPDR = 2;
while(!(SPSR &(1<<SPIF)));
SPDR =53;
while(!(SPSR &(1<<SPIF)));
SPDR =8;
while(!(SPSR &(1<<SPIF)));
PORTB = (1 << SS);
_delay_ms (1);

//Load TX Buffer: Load 8 byte du lieu len Buffer 0, trong do co 1 byte giatriADC duoc cap nhat tu cam bien
PORTB = (0 << SS);
SPDR =65;
while(!(SPSR &(1<<SPIF)));
SPDR =giatriADC;
while(!(SPSR &(1<<SPIF)));
PORTB = (1 << SS);
_delay_ms (1);

// Request to send de gui tin hieu luu tren Buffer di len CAN
```

```
    PORTB = (0 << SS);
    SPDR = 129;
    while(!(SPSR &(1<<SPIF)));
    PORTB = (1 << SS);
    _delay_ms (1);
    _delay_ms (10);
  }

}
```

Code lập trình Arduino Uno 4 (gửi tín hiệu mức xăng).

```
#include <avr/io.h>
#ifndef F_CPU
#define F_CPU 2000000UL
#endif
#include <util/delay.h>
#define SS PORTB2
#define MOSI PORTB3
#define MISO PORTB4
#define SCK PORTB5

void SPI_init()
{
  DDRB = (0 << MISO) | (1 << MOSI) | (1 << SS) | (1 << SCK);
  SPCR = (1 << SPE) | (1 << MSTR) | (1 << SPR0);//Thiet lap SPI Master cua ATmega328P
}

void adc_init()
{
  ADMUX = (1<<REFS0)|(1<<MUX0)|(1<<MUX2);  //Chon chan A5 la input cua bien tro. MUX quy định chân input A0=>A5
```

```
  //ADMUX = (1<<REFS0)|(1<<MUX2);   //Chon chan A4 la input cua bien tro
  ADCSRA = (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);//Ti le=128
}

uint16_t adc_read(void)
{
  ADCSRA |= (1<<ADSC);
  while(ADCSRA & (1<<ADSC));
  return ADC;
}
void MCP_setup()
{
  //Thiet lap che do MCP Configuration de cai dat cac Register (Tham khao Datasheeet
cua MCP2115
  PORTB = (0 << SS);
  SPDR = 2;// Moi lan gan gia tri cho SPDR, ATmega tu dong kich hoat che do truyen
SPI
  while(!(SPSR &(1<<SPIF)));//Cho cho SPI truyen xong
  SPDR =15;
  while(!(SPSR &(1<<SPIF)));
  SPDR =128;
  while(!(SPSR &(1<<SPIF)));
  PORTB = (1 << SS);
  _delay_ms (1);

  PORTB = (0 << SS);
  SPDR = 2;
  while(!(SPSR &(1<<SPIF)));
  SPDR =40;
  while(!(SPSR &(1<<SPIF)));
  SPDR =129;
```

```
  while(!(SPSR &(1<<SPIF)));
  PORTB = (1 << SS);
  _delay_ms (1);

  PORTB = (0 << SS);
  SPDR = 2;
  while(!(SPSR &(1<<SPIF)));
  SPDR =41;
  while(!(SPSR &(1<<SPIF)));
  SPDR =209;
  while(!(SPSR &(1<<SPIF)));
  PORTB = (1 << SS);
  _delay_ms (1);
}

void MCP__setNormal()
{
  //Set MCP o che do Normal
  PORTB = (0 << SS);
  SPDR = 2;
  while(!(SPSR &(1<<SPIF)));
  SPDR =15;
  while(!(SPSR &(1<<SPIF)));
  SPDR =7;
  while(!(SPSR &(1<<SPIF)));
  PORTB = (1 << SS);
  _delay_ms (1);

}

volatile uint8_t giatriADC = 0;
```

```
int main(void)
{
  adc_init();
  SPI_init();
  MCP_setup();
  MCP__setNormal();
  while (1) {
    // Doc gia tri cam bien
    giatriADC = (adc_read()/4);

    // Xuat ID cua CAN
    PORTB = (0 << SS);
    SPDR = 2;
    while(!(SPSR &(1<<SPIF)));
    SPDR =49;
    while(!(SPSR &(1<<SPIF)));
    SPDR =0;
    while(!(SPSR &(1<<SPIF)));
    PORTB = (1 << SS);
    _delay_ms (1);

    PORTB = (0 << SS);
    SPDR = 2;
    while(!(SPSR &(1<<SPIF)));
    SPDR =50;
    while(!(SPSR &(1<<SPIF)));
    SPDR =94;
    while(!(SPSR &(1<<SPIF)));
    PORTB = (1 << SS);
    _delay_ms (1);
```

```
    //Khai bao do dai CAN 8 byte
    PORTB = (0 << SS);
    SPDR = 2;
    while(!(SPSR &(1<<SPIF)));
    SPDR =53;
    while(!(SPSR &(1<<SPIF)));
    SPDR =8;
    while(!(SPSR &(1<<SPIF)));
    PORTB = (1 << SS);
    _delay_ms (1);

    //Load TX Buffer: Load 8 byte du lieu len Buffer 0, trong do co 1 byte giatriADC duoc
cap nhat tu cam bien
    PORTB = (0 << SS);
    SPDR =65;
    while(!(SPSR &(1<<SPIF)));
    SPDR =giatriADC;
    while(!(SPSR &(1<<SPIF)));
    PORTB = (1 << SS);
    _delay_ms (1);

    // Request to send de gui tin hieu luu tren Buffer di len CAN
    PORTB = (0 << SS);
    SPDR = 129;
    while(!(SPSR &(1<<SPIF)));
    PORTB = (1 << SS);
    _delay_ms (1);
    _delay_ms (10);
  }
}
```

TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT THÀNH PHỐ HỒ CHÍ MINH

www.hcmute.edu.vn