# Giới Thiệu

## Course Contents

Chào mừng bạn đến với bài giảng này, trong đó tôi sẽ giới thiệu và dẫn bạn qua nội dung khóa học.

Vậy, trong khóa học này, bạn sẽ học về giao thức CAN một cách toàn diện.

Bạn sẽ làm chủ nó và cả giao thức CAN FD so với giao thức CAN, sau đó sẽ tìm hiểu cách thực hiện giao thức CAN trong ngành công nghiệp, tức là các nguyên tắc cơ bản của nó và cách kiểm tra tính hợp lệ của giao thức CAN.

Chúng tôi sẽ trình bày một bản demo của công cụ Canalyzer.

Khóa học có tổng cộng khoảng 18 bài giảng, bao gồm các phần kỹ thuật.

Chúng ta sẽ bắt đầu với phần giới thiệu về CAN.

Trong phần giới thiệu về CAN, chúng tôi sẽ giải thích giao thức CAN là gì và so sánh nó với các giao thức khác như thế nào.

Chúng tôi sẽ giải thích lý do tại sao CAN lại là giao thức tốt nhất cho các ứng dụng trong ô tô, v.v., sau đó chúng tôi sẽ giải thích các tính năng của CAN, bao gồm tốc độ CAN là bao nhiêu, nó có thể mang bao nhiêu dữ liệu, các đặc tính vật lý của Canbus và phần logic của CAN?

Chúng tôi cũng sẽ giới thiệu về các bộ điều khiển CAN. Chúng làm gì?

CAN transceivers, chúng làm gì?

Và về kiến trúc mạng của CAN. Chúng ta sẽ hiểu về các tính năng của CAN như vậy, và sau đó chúng ta sẽ tìm hiểu các thuộc tính của CAN.

Đặc điểm của CAN, như nó là một giao thức bán song công, song công, có dây, không dây, nối tiếp, song song, đồng bộ hay không đồng bộ, nếu là giao thức điểm-điểm hay phát sóng?

Giao thức này là giao thức dựa trên thông điệp, v.v. Tất cả các thuộc tính này sẽ được giải thích.

Chúng ta sẽ hiểu về các thuộc tính của CAN, nơi mà CAN thuộc về và các thuộc tính khác của nó trong phần đầu tiên, đó là kiến thức cơ bản về giao thức CAN.

Sau khi phần này hoàn thành, chúng ta sẽ đi sâu vào chi tiết của các khung CAN.

Khung CAN có những loại nào?

Chúng ta sẽ tìm hiểu về điều đó và sau đó tìm hiểu về định dạng của khung dữ liệu chuẩn và tôi đã tạo ra một công cụ Excel để giúp bạn làm quen với định dạng khung dữ liệu của CAN và tất cả những thứ liên quan. Chúng tôi cũng sẽ giới thiệu bạn.

Sau đó, bạn sẽ hiểu về định dạng khung mở rộng. Các khung mở rộng, khung chuẩn, v.v. Mọi thứ sẽ được giải thích trong các bài giảng này.

Sau đó, chúng ta sẽ tiếp tục với phần cơ bản của giao thức CAN như phân đoạn bit là gì?

Phân xử bus là gì?

Một nút trở thành máy phát hay máy thu như thế nào và nhồi bit (bit stuffing) là gì? Tại sao lại cần nhồi bit?

Sau đó là tính toán checksum, logic sử dụng và cách tính toán checksum cho giao thức CAN.

Những phương pháp này sẽ được giải thích ở đây và sau đó là xử lý lỗi của CAN.

Nếu có lỗi xảy ra trong CAN, chúng ta sẽ xử lý như thế nào?

Có những loại lỗi CAN nào và định dạng khung lỗi là gì?

Cờ lỗi và các trạng thái lỗi sẽ được giải thích ở đây. Một trong những lỗi mà bạn sẽ thường gặp là bus-off trong CAN.

Điều này sẽ được giải thích trong phần này.

Và đến đây, chúng ta sẽ tìm hiểu về giao thức CAN và khung CAN.

Vậy giao thức CAN là gì? Và định dạng khung CAN là gì? Sự khác biệt giữa CAN và CAN FD, tất cả những điều này bạn sẽ hiểu trong phần này.

Đến đây thì là phần lý thuyết. Sau đó, sẽ đến phần thực hành, trong đó chúng ta sẽ tìm hiểu về việc thực hiện giao thức CAN.

Thực hiện giao thức CAN có nghĩa là gì? Làm thế nào nó được thực hiện và kiểm tra, xác minh như thế nào?

Các nguyên tắc cơ bản này sẽ được giải thích ở đây, và chúng tôi sẽ sử dụng Canalyzer – một công cụ nổi tiếng trong ngành công nghiệp ô tô hiện nay của Vector Tools.

Chúng tôi sẽ trình diễn chi tiết về cách gửi khung trên bus, cách nhận các khung trên bus, cách đo và theo dõi giao thông trên bus, và cách sử dụng Canalyzer để kiểm tra các khung CAN với ECU.

Cách thực hiện điều này trong các công ty thực tế cũng sẽ được trình bày.

Bạn sẽ có cơ hội trải nghiệm điều đó.

Cuối cùng, chúng ta sẽ có phần đánh giá, bài kiểm tra cuối cùng, và sau đó bạn sẽ nhận chứng chỉ Udemy về giao thức CAN.

Vậy là về nội dung khóa học.

Chúc bạn học tốt và bắt đầu với các bài giảng, thư giãn và thực hành sau khi học. Cố gắng viết lại những gì bạn đã học bằng từ ngữ của chính mình để có thể ghi chú lại.

Hy vọng khóa học này sẽ thực sự giúp ích cho bạn.

Cảm ơn bạn.

# Can Protocol Basics

## Giới thiệu Can

Đây là bài học đầu tiên về giao thức CAN, bắt đầu từ phần kỹ thuật của giao thức.

Vậy, trong bài học này, chúng ta sẽ cố gắng hiểu giao thức là gì? Tại sao chúng ta lại sử dụng giao thức Racan, giao thức này thực tế được sử dụng ở đâu và sự so sánh giữa nó với các giao thức tương tự khác để giúp chúng ta hiểu được vị trí của nó?

Trong suốt khóa học này, tôi chỉ muốn chia sẻ rằng CAN là một giao thức rất thú vị, và trong bài học thứ hai, chúng ta sẽ tìm hiểu về giao thức truyền thông của nó.

Vậy CAN là gì? Nó đại diện cho mạng điều khiển khu vực của chúng ta.

Hãy để tôi lấy một cây bút khác để làm nổi bật điều này.

Mạng CAN là một dạng giao thức truyền thông.

Nó được phát triển và sáng tạo bởi Bosch Germany GMBH.

Nếu bạn biết Bosch là một công ty ô tô, thì nó là một công ty nổi tiếng, và hiện nay nó sẽ giúp kết nối nhiều ECU (Electronic Control Units) trong ô tô.

Các ECU này là các đơn vị điều khiển điện tử.

Nếu bạn đến từ ngành công nghiệp ô tô, thì chắc chắn bạn biết về chúng. Khi vấn đề này xuất hiện, chúng có một số công việc đặc biệt để làm.

Những thiết bị này là các hệ thống nhúng, đúng không?

Chúng nhận đầu vào và đưa ra đầu ra.

Trong ô tô, chúng hiện là các mô-đun độc lập, thực hiện nhiệm vụ của mình một cách liền mạch.

Các mô-đun này phải truyền một số dữ liệu với nhau. Chúng ta gọi những tín hiệu này là các tín hiệu dữ liệu, và chúng cần phải giao tiếp.

Ví dụ, nếu bạn thấy điều khiển túi khí, nó phải lấy giá trị từ bộ phân loại người ngồi, như là người ngồi có mặt trên ghế hay không, và chỉ khi đó túi khí mới triển khai.

Do đó, nếu bạn và túi khí được điều khiển bởi hệ thống điều khiển túi khí, thì liệu dữ liệu người ngồi có được xác định bởi bộ phân loại người ngồi hay không sẽ được tính toán hoặc phát hiện.

Dữ liệu phải đến từ bộ điều khiển túi khí.

Vì vậy, giao tiếp giữa các hệ thống sẽ được thực hiện.

Chúng đề xuất một giao thức truyền thông gọi là giao thức CAN, được thiết kế đặc biệt cho mục đích này.

Trong hình ảnh này, bạn có thể thấy các mô-đun khác nhau trong ô tô.

Chúng được kết nối với nhau bằng các dây dẫn, với các màu khác nhau như dây vàng, xanh, cam,... Đây là những mô-đun khác nhau, kết nối với nhau qua mạng. Mạng này được gọi là mạng CAN.

Có các nút trong mạng, và mỗi nút có thể truyền dữ liệu và nhận dữ liệu. Mỗi nút là một đơn vị đang giao tiếp trong mạng, và phương tiện truyền tải này có thể là có dây hoặc không dây.

Có nhiều giao thức truyền thông khác nhau mà chúng ta có thể sử dụng, bao gồm có dây. Phương thức có dây này được gọi là "bus CAN".

Giao thức CAN được thiết kế chủ yếu cho ngành công nghiệp ô tô, nhưng nó cũng trở nên rất phổ biến trong các ứng dụng khác, như các thang máy, các bộ phận trong máy pha cà phê, hoặc các thiết bị khác có thể sử dụng giao thức này để giao tiếp.

Vì vậy, CAN được thiết kế để đáp ứng các nhu cầu đặc biệt của ngành ô tô, và hiện nay, nó đã trở nên rất phổ biến trong nhiều ứng dụng khác.

Nếu bạn hiểu được yêu cầu trước đây của giao tiếp giữa các nút, bạn sẽ hiểu lý do tại sao CAN tồn tại, vì nó được thiết kế để giải quyết những vấn đề đặc thù trong ngành ô tô.

Trong mạng CAN, các nút có thể bắt đầu giao tiếp mà không cần phải xin phép một nút khác. Nếu dữ liệu đã sẵn sàng và bus (mạng) không bị bận, nút có thể bắt đầu gửi một khung dữ liệu (data frame).

CAN có khả năng hỗ trợ các vấn đề điện tử với mức độ nhiễu điện rất cao, điều này rất quan trọng trong môi trường ô tô, nơi có rất nhiều nhiễu từ điện và điện tử.

Khi chúng ta nói về các đặc tính của giao thức CAN, bạn sẽ thấy nó có khả năng chống lại nhiễu điện tử và có độ trễ thấp, với mỗi khung dữ liệu có thể chứa tối đa 8 byte dữ liệu.

Giao thức CAN cũng hỗ trợ khả năng mở rộng, ví dụ nếu các yêu cầu trong tương lai cần truyền tải dữ liệu lớn hơn, sẽ có những cập nhật để hỗ trợ những thay đổi này, ví dụ như giao thức CAN FD (Flexible Data-rate).

Giao tiếp CAN là không đồng bộ, nghĩa là các nút không đồng bộ với nhau, và chỉ khi bus không bận, một nút có thể bắt đầu truyền dữ liệu.

Tóm lại, CAN là giao thức truyền thông rất phù hợp với ngành ô tô, vì nó có thể xử lý các vấn đề giao tiếp giữa các thiết bị điện tử trong môi trường có nhiều nhiễu điện và yêu cầu độ trễ thấp.

Vậy là bạn đã có cái nhìn cơ bản về giao thức CAN. Trong bài học tiếp theo, chúng ta sẽ đi sâu vào các tính năng của CAN và các định dạng dữ liệu, các trạng thái khác nhau và những trách nhiệm của CAN.

## Chức năng của CAN

Giảng viên: Và cuối cùng là về **mạng topology**.

Vậy đó là chương trình học của bài giảng này.

Bây giờ, chúng ta sẽ đi vào chủ đề chính. **Tốc độ CAN**, đúng không?

Tốc độ CAN có nghĩa là gì, đúng không?

Chúng ta biết rằng trong CAN,

nhiều ECU được kết nối trong mạng

và dữ liệu phải được truyền tải

dưới dạng các frame, đúng không?

Các frame CAN,

chúng sẽ đi trên bus và truyền tải qua bus.

Vậy tốc độ

mà các frame được truyền tải qua bus là bao nhiêu, đúng không?

Đó chính là ý nghĩa của tốc độ CAN.

Các frame CAN có thể truyền với tốc độ lên đến 1 mbps

với chiều dài bus lên tới 40 mét, đúng không?

Khi chiều dài bus tăng lên,

tốc độ tối đa sẽ giảm đi, đúng không?

Các bạn có thể thấy biểu đồ này, đúng không?

Trong biểu đồ này, bạn có thể thấy

trục tung nói về **tốc độ bit**,

tức là 10 kbps, 20 kbps, 50 kbps,

100 kbps, 200 kbps và 1 mbps.

Trục hoành nói về chiều dài bus, đúng không?

Vì vậy, như bạn có thể thấy,

với tốc độ 1,000 kbps, tức là 1 mbps,

với chiều dài bus 40 mét,

ta có thể đạt được tốc độ 1,000 kbps, tức là 1 mbps.

Và khi chiều dài bus tăng lên, chiều dài dây cũng tăng lên,

tốc độ tối đa được phép sẽ giảm theo.

Ví dụ, nếu bạn muốn đạt tốc độ 500 kbps,

với chiều dài bus là 100 mét,

thì tốc độ tối đa bạn có thể có là 500 kbps, đúng không?

Đó là ý nghĩa của biểu đồ này, đúng không?

Vậy đó là về **tốc độ CAN**,

và hãy cùng hiểu rõ hơn về tốc độ CAN này.

Khi chúng ta nói về 1 mbps và 500 kbps, chúng có nghĩa là gì?

Kbps, nếu bạn nói về ở đây, thì chữ K là viết tắt của kilo và "bits per second" có nghĩa là "bit mỗi giây".

Vì đây là một bus nối tiếp, chúng ta sẽ truyền tải từng bit

sau từng bit một.

Vậy bps có nghĩa là **bit mỗi giây**, đúng không?

Vậy 500 kilobits mỗi giây có nghĩa là gì?

Nó có nghĩa là trong một giây,

500 kilobits có thể được truyền tải.

Tức là 500,000 bits có thể được truyền tải, được chưa?

Và một frame đơn có thể mất đến 125 bits,

thường là với 8 bytes dữ liệu

và độ dài dữ liệu tối đa,

các bit thông thường sẽ là 111 bit.

Tôi sẽ giải thích thêm về khái niệm bit

khi tôi giải thích về định dạng frame và các chi tiết,

lúc đó bạn sẽ hiểu rõ hơn về số lượng bits.

Nhưng giả sử, ví dụ,

một frame sẽ mất khoảng 125 bits mỗi frame, đúng không?

Giảng viên: Sau đó, trong một giây,

bus có thể chứa tối đa khoảng 2,000 frame.

Tức là, với tốc độ 500 Kbps và mỗi frame có 125 bits,

ta có thể gửi tối đa 2,000 frame

trên bus CAN trong một giây.

Vậy trong một giây, chúng ta có thể gửi được 2,000 frame, đúng không?

Phép tính này sẽ được sử dụng

để tính toán độ quá tải của bus,

lưu lượng trên bus và tất cả những yếu tố liên quan,

như số lượng các node có thể kết nối,

số lượng frame trong một mạng,

và chu kỳ các thông điệp chu kỳ của chúng,

tất cả những phép tính này,

bạn sẽ được tìm hiểu trong các bài giảng tiếp theo.

Đây là những kiến thức cơ bản, đúng không?

Vì vậy, dựa trên tốc độ CAN này,

chúng ta có thể quyết định số lượng ECU,

số lượng node

có thể hiện diện trong mạng CAN,

và baud rate là bao nhiêu,

cũng như chu kỳ, tức là có thể có bao nhiêu frame.

Tất cả những tính toán này sẽ phụ thuộc vào tốc độ này.

Vậy tôi hy vọng phần giải thích này đã giúp bạn

hiểu rõ về tốc độ CAN, đúng không?

Và tiếp theo, chúng ta sẽ nói về **giới hạn dữ liệu**, đúng không?

Vậy tại sao chúng ta lại sử dụng CAN?

Giả sử có nhiều ECU,

và trong ví dụ này, chúng ta sẽ lấy hai ECU,

ECU 1 và ECU 2.

Có hai ECU ở đây, đang giao tiếp với nhau.

Và ECU 1 có ứng dụng.

Ứng dụng 1 đang gửi dữ liệu.

Vậy, ECU 1 sẽ là nguồn truyền

và ứng dụng 2 là nhận dữ liệu.

Vậy việc truyền tải dữ liệu đang diễn ra.

Vậy CAN có thể chỉ truyền dữ liệu

lên bus không? Có thể không, vì chúng ta không biết dữ liệu là gì,

cũng như ID của thông điệp và kích thước dữ liệu.

Tất cả những thông tin đó là cần thiết

và chúng phải được xác định bởi giao thức CAN, đúng không?

Vậy ngoài dữ liệu,

có những thông tin khác cũng

cần thiết để giao tiếp.

Vì vậy, toàn bộ gói tin,

gửi qua bus, được gọi là **frame**.

Và trong frame, dữ liệu là một phần trong đó.

Các bit khác hoặc phần khác

có trong frame,

ngoài dữ liệu, đó là các thông tin kiểm soát giao thức,

chúng ta gọi là **PCI (Protocol Control Information)**.

Vậy bạn có thể nói rằng frame có hai phần.

Một là PCI, đó là tất cả các trường

liên quan đến giao thức, ở đây là giao thức CAN,

và phần còn lại là dữ liệu, dữ liệu thực tế

được truyền tải giữa các node, đúng không?

Vậy trong một frame, bạn có thể gửi bao nhiêu dữ liệu?

Bạn có thể gửi hàng ngàn byte hay chỉ một bit,

hay có thể gửi 10 bit?

Bạn có thể gửi ba byte, bốn byte, mười byte hay megabyte không?

Câu hỏi là, dữ liệu tối đa có thể gửi trong một frame là bao nhiêu

và trong giao thức CAN, đã quyết định

rằng lượng dữ liệu tối đa

có thể gửi mỗi frame là 8 byte.

Vậy trong một frame, tối đa là 8 byte,

và bạn có thể gửi cả 0 byte nữa.

Bạn có thể gửi một byte, hai byte,

nhưng bạn không thể gửi 10 bit,

vì 10 bit là hai byte và hai bit, đúng không?

Vậy bạn không thể gửi 10 bit,

bạn phải gửi dữ liệu theo bội số của byte.

Đó là quy tắc do giao thức CAN đưa ra.

Vậy bạn có thể gửi 0 byte, 1 byte, 2 byte, 3 byte,

4 byte, 5 byte, 6 byte, 7 byte,

hoặc tối đa 8 byte, đúng không?

Vậy đó là giới hạn về dữ liệu,

giới hạn tối đa mà giao thức CAN đưa ra.

Vậy nếu bạn có dữ liệu lớn hơn 8 byte,

thì sẽ có các giao thức khác hỗ trợ

gọi là giao thức CANTp,

theo tiêu chuẩn ISO 15765,

ở đó dữ liệu sẽ được chia ra

và gửi qua nhiều frame khác nhau,

nhiều frame CAN, đúng không?

Đó là cách xử lý.

Còn bây giờ, bạn có thể hiểu rằng

nếu đây là khóa học về giao thức CAN,

thì trong mỗi frame, dữ liệu tối đa là 8 byte.

Và giả sử bạn có 10 bit dữ liệu cần gửi,

bạn sẽ làm thế nào để thực hiện?

Bạn không có đủ, đó là hai byte dữ liệu.

Tức là,

giả sử, hai byte là 16 bit,

16 bit và một phần sáu bit khác.

Vậy bạn sẽ gửi hai byte

với sáu bit cuối là **dummy bits**.

Ứng dụng sẽ phải biết rằng sáu bit cuối là dummy bits.

Và bạn sẽ gửi hai byte thực tế trong trường hợp đó.

Vậy đó là cách bạn sẽ gửi dữ liệu trên bus CAN, đúng không?

Vậy số byte, như tôi đã nói, có thể là một byte, hai byte,

ba byte, bất kỳ số byte nào,

nhưng tối đa là 8 byte.

Vậy làm thế nào mà frame CAN biết được

số byte dữ liệu có trong đó?

Làm thế nào mà ECU biết điều đó?

Để làm điều đó, bạn cần hiểu định dạng frame của CAN.

Khi tôi giải thích định dạng frame của CAN,

sẽ có một trường gọi là **DLC (Data Length Code)**,

nó sẽ cho biết có bao nhiêu byte

dữ liệu trong frame này.

Tôi sẽ giải thích chi tiết về phần này

khi tôi giảng về định dạng frame,

nhưng ngay bây giờ bạn có thể hiểu rằng

số byte trong frame, từ 0 đến 8,

có thể là bao nhiêu byte, tối đa là 8 byte,

và số byte có trong đó, bạn sẽ biết

thông qua một phần trong thông tin điều khiển giao thức,

được gọi là **DLC**.

Được chứ? Tốt.

Vậy là chúng ta đã đề cập đến tốc độ CAN và giới hạn dữ liệu.

Tiếp theo, chúng ta sẽ đến với phần lớp vật lý của giao thức CAN, đó chính là bus CAN.

Như bạn có thể thấy,

đây là cách mà bus CAN được thể hiện, đúng không?

Bạn có thể thấy có những cặp dây xoắn.

Vậy là có một dây và một dây khác.

Bạn lấy hai dây này và xoắn chúng lại với nhau.

Chúng được quấn vào nhau.

Bạn có thể tưởng tượng chúng giống như những con rắn xoắn lại với nhau.

Bus CAN cũng tương tự như vậy.

Bus CAN chủ yếu bao gồm hai dây,

chúng được xoắn lại với nhau,

được gọi là cặp dây xoắn, đúng không?

Hai dây này, một dây được gọi là CAN high,

dây kia được gọi là CAN low,

và chúng được xoắn với nhau,

và kết thúc bằng một điện trở.

Đây là một điện trở,

và đây là một điện trở, đúng không?

Như vậy, giá trị điện trở luôn là 120 ohm,

với điện trở này, đúng không?

Bạn có hai dây.

Một dây được gọi là CAN high,

dây kia được gọi là CAN low,

và hai dây này đều được kết thúc bằng các điện trở 120 ohm ở hai đầu, đúng không?

Bus CAN là một cặp dây xoắn kép,

gọi là CAN high, CAN low,

và chúng có điện áp vi sai, đúng không?

Trước tiên, chúng ta cần hiểu tại sao lại sử dụng cặp dây xoắn.

Để hiểu được điều này, bạn cần hiểu các lý thuyết cơ bản, đúng không?

Bạn biết đây là giao tiếp bus nối tiếp,

có nghĩa là các bit được truyền đi.

Vậy tại một thời điểm nào đó, bus CAN sẽ chỉ có một giá trị bit, đúng không?

Có thể là một bit logic một hoặc logic không, đúng không?

Tùy thuộc vào các điện áp

có trên CAN high và CAN low,

delta V (sự chênh lệch điện áp) sẽ được tính toán.

Delta V thực ra là sự khác biệt giữa điện áp

trên CAN high và CAN low.

Cụ thể là điện áp trên CAN high,

điện áp trên CAN high trừ điện áp trên CAN low.

Chênh lệch điện áp này được gọi là

điện áp vi sai, đúng không?

Nếu bạn muốn đặt logic một trên bus CAN,

thì bạn sẽ làm cho giá trị CAN high là 2.5 volt,

CAN low là 2.5 volt.

Nếu cả hai dây có cùng điện áp,

là 2.5 volt, thì delta V,

sự khác biệt giữa chúng, điện áp vi sai,

sẽ là không.

Và nếu bạn muốn đặt logic không,

thì CAN high sẽ là 3.5 volt

và CAN low sẽ là 1.5 volt, đúng không?

Và sự khác biệt giữa chúng sẽ là 2 volt, đúng không?

Vậy bạn có thể thấy sự kết nối giữa logic và delta V,

nếu delta V là 0, thì đó là logic một.

Nếu delta V là 2 volt, thì đó là logic không.

Bạn có thể thấy trong sơ đồ này, đúng không?

Vì vậy, logic một, cả hai đều là 2.5 volt.

Logic không, CAN high là 3.5 volt, CAN low là 1.5 volt.

Nếu bạn muốn nhớ, bạn có thể tưởng tượng như vậy.

Hãy tưởng tượng CAN high là môi trên của bạn,

CAN low là môi dưới của bạn.

Và nếu bạn muốn đặt logic không, bạn có thể làm như thế này,

nó giống như số không, đúng không?

Còn đây là logic một trong vị trí ngang, đúng không?

Vì vậy, với logic một, ở vị trí ngang,

cả hai đều có 2.5 volt, còn CAN high

và CAN low có điện áp khác nhau.

CAN high là 3.5, CAN low là 1.5.

Vậy đây là logic không và đây là logic một, đúng không?

Đó là cách bạn có thể nhớ, đúng không?

Vì vậy, logic không có điện áp khác nhau

giữa CAN high và CAN low,

nên logic không được gọi là bit thống trị.

Tôi sẽ giải thích tại sao nó được gọi là bit thống trị

và bit thống trị là gì,

nhưng bạn chỉ cần nhớ rằng trong giao thức CAN,

logic không được gọi là bit thống trị,

logic một được gọi là bit thoái lui, đúng không?

Vì vậy, trạng thái mặc định của bus CAN sẽ là thoái lui,

vì điện áp CAN high và CAN low bằng nhau ở đó.

Tức là, điện áp chênh lệch là không, đúng không?

Và bit thống trị là logic không,

vì điện áp CAN high

và CAN low khác nhau.

Điện áp vi sai không bằng không, đúng không?

Logic không được gọi là bit thống trị,

vì giả sử, nếu hai nút đang cố gắng

thay đổi giá trị của bus

và một nút cố gắng đặt logic một trên bus

và nút kia cố gắng đặt logic không trên bus,

thì bus sẽ lấy logic không

chứ không phải logic một.

Vì vậy, khi cạnh tranh giữa hai logic này,

logic không thắng và bus sẽ nhận giá trị logic không,

vì vậy logic không được gọi là thống trị,

vì nó đã chiếm ưu thế và đã thay đổi giá trị bus thành logic không.

Vậy logic không được gọi là bit thống trị,

logic một được gọi là bit thoái lui.

Hãy nhớ điều này, vì khái niệm thống trị

và thoái lui sẽ được sử dụng

trong các lỗi của CAN và trong quá trình phân xử,

cách mà các nút sẽ giành quyền kiểm soát bus.

Đây là một trong những khái niệm thú vị của CAN,

đây là khái niệm cơ bản,

logic không là thống trị

và logic một là thoái lui, đúng không?

Tại sao nó gọi là thống trị?

Bởi vì nếu logic một và logic không đang cạnh tranh

để chiếm quyền kiểm soát bus,

thì nút cố gắng đặt logic không sẽ thắng,

vì bus sẽ nhận logic không trên bus, đúng không?

Vậy đó là ý nghĩa của logic không và logic một

và đó là cách CAN high và CAN low hoạt động.

Và vì chúng ta có điện áp vi sai ở đây,

do việc sử dụng điện áp vi sai,

và vì có cặp dây xoắn, bus CAN có khả năng chống nhiễu điện rất cao, đúng không?

Bạn biết bus CAN này?

Chúng ta sẽ đặt bus CAN ở đâu trong xe, đúng không?

Giữa các ECU khác nhau,

chúng ta đặt nó trong xe,

nơi có rất nhiều động cơ,

các bộ phận cơ khí, khung gầm, có độ rung,

và môi trường có mức độ nhiễu điện rất cao,

vậy điện áp có trên bus,

do nhiễu điện, sẽ có một số biến dạng

và có thể dẫn đến lỗi.

Vì vậy, rất quan trọng là bất kỳ bus và giao thức nào chúng ta triển khai

trong môi trường có nhiễu điện cao,

giao thức và bus phải có khả năng chống nhiễu rất tốt, đúng không?

Bây giờ, CAN rất phù hợp cho việc này.

Hãy để tôi giải thích tại sao.

Giả sử chúng ta đang cố gắng gửi các bit trên bus CAN,

và CAN high có giá trị điện áp VH, đúng không?

Và CAN low có giá trị điện áp VL, đúng không?

Tùy thuộc vào việc đó là logic không hay logic một,

VH và VL sẽ là 3.5 volt,

VH sẽ là 3.5 volt hoặc 2.5 volt,

VL sẽ là 1.5 volt hoặc 2.5 volt, đúng không?

Và do nhiễu,

một điện áp phụ sẽ được chèn vào bus.

Vì đây là cặp dây xoắn, chúng rất gần nhau

và nếu nhiễu được chèn vào một dây, nó sẽ chèn cùng một điện áp vào dây còn lại, đúng không?

Vậy giá trị điện áp nhiễu,

nó là một điện áp ngẫu nhiên, sẽ được chèn vào bus.

Bạn có thể thấy rằng bus CAN được nối đất ở cả hai đầu với một điện trở 120 ohm, vì sự khớp trở kháng. Giả sử có một điện áp đang di chuyển, và nếu bạn quen thuộc với khái niệm tín hiệu và hệ thống, nếu bạn đến từ nền tảng điện tử, thì bạn sẽ biết rằng các tín hiệu di chuyển qua dây sẽ bị phản xạ lại ở cuối dây, và có một thứ gọi là sóng phản xạ sẽ can thiệp với sóng hiện có. Vì vậy, sự phản xạ của sóng cần phải được giảm thiểu. Để giảm thiểu sóng này, chúng ta phải kết nối dây với một điện trở phù hợp. Tùy thuộc vào giá trị của các điện trở khác nhau, mức độ hoặc độ phản xạ của sóng sẽ thay đổi. Có một công thức gọi là một chia cho bốn epsilon nought, mu nought, hoặc một chia cho bốn epsilon mu, trong đó epsilon là hằng số điện thẩm, và mu là hằng số từ thẩm. Vì vậy, tùy thuộc vào công thức này, người ta tính toán rằng đối với giao thức CAN với dây sử dụng, 120 ohm nếu bạn kết nối ở cuối mỗi đầu của bus CAN, thì sự khớp trở kháng này sẽ xảy ra. Sóng bị phản xạ lại sẽ rất nhỏ, và vì lý do này, để đảm bảo khớp trở kháng, chúng ta kết nối 120 ohm ở cuối bus CAN.

Vậy đây là những điều cơ bản của giao thức CAN, tại sao chúng ta sử dụng cặp xoắn, tại sao chúng ta sử dụng điện áp vi sai, tại sao chúng ta sử dụng 120 ohm và các logic CAN, CAN high và CAN low. Nhưng có một vấn đề ở đây. Vấn đề đó là gì? Bạn biết rằng các vi điều khiển mà bạn đang làm việc, đây không phải là logic mà bạn đang thấy, đúng không? Chúng ta luôn có logic 0 được biểu diễn bằng 0 volt. Logic 1 được biểu diễn bằng, ví dụ, 3.3 volt hoặc 5 volt. Logic đó gọi là TTL (logic transistor-transistor). Nhưng vì TTL không chống nhiễu tốt và các vấn đề liên quan, chúng ta sử dụng logic CAN ở đây. Tuy nhiên, làm thế nào mà CAN được vi điều khiển hiểu? Các vi điều khiển làm việc với TTL, không phải logic CAN. Bus đang sử dụng logic CAN, nhưng các vi điều khiển, các ECU lại sử dụng TTL logic. Vậy làm thế nào chúng ta có thể kết nối giữa chúng? Chúng ta cần một thiết bị để chuyển đổi logic TTL từ vi điều khiển sang logic CAN và đưa nó vào bus CAN.

Và ngược lại, lấy logic CAN từ bus và chuyển nó thành TTL logic để vi điều khiển có thể hiểu. Chúng ta cần một người trung gian hoặc một bộ chuyển đổi làm công việc chuyển đổi tín hiệu, chuyển đổi công nghệ, đúng không? Đó là lúc chúng ta gặp khái niệm gọi là bộ chuyển đổi CAN (CAN transceiver). Bạn có thể thấy rằng, bộ chuyển đổi này, bạn sẽ thấy phía kết nối với bus có logic CAN, CAN high và CAN low. Và ở đây chúng ta có logic TTL, Rx và Tx. Vì vậy, chúng ta có gì? Bạn phải biết rằng có những khái niệm như sau: Đây là vi điều khiển của chúng ta, hệ thống nhúng, và sau đó chúng ta có một bộ điều khiển CAN, bộ điều khiển này thực hiện toàn bộ giao thức CAN bên trong phần cứng. Sau đó chúng ta có một bộ chuyển đổi, bộ chuyển đổi này sẽ xử lý việc chuyển đổi logic.

Bộ điều khiển CAN này cũng thực hiện giao thức CAN bên trong nó, nhưng nó làm việc với TTL, đúng không? Vì vậy, TTL logic, tức là 0 volt và 5 volt, còn ở đây là logic CAN, có nghĩa là 1.5 volt, 2.5 volt và 3.5 volt cho logic 1. Việc chuyển đổi này được thực hiện bởi bộ chuyển đổi CAN. Vì vậy, mọi dữ liệu phải được chuyển đổi, dữ liệu đó sẽ được vi điều khiển chuyển cho bộ điều khiển CAN. Bộ điều khiển CAN sẽ tạo ra frame CAN trong các bit TTL, và toàn bộ frame CAN này sẽ được chuyển đến bộ chuyển đổi. Bộ chuyển đổi CAN, mọi thứ có trong đó, sẽ chuyển từ TTL sang logic CAN và gửi lên bus CAN. Và ngược lại, khi frame CAN đến, frame này sẽ được lấy, chuyển từ logic CAN sang logic TTL. Frame với logic TTL sẽ được đưa cho bộ điều khiển CAN. Bộ điều khiển CAN sẽ loại bỏ tất cả các trường dữ liệu liên quan đến giao thức và chỉ lấy phần dữ liệu cần thiết để gửi cho vi điều khiển, và vi điều khiển sẽ tiếp tục xử lý cho ứng dụng của nó.

Đây là cách mà việc triển khai thực tế của CAN hoạt động. Chúng ta có vi điều khiển, sau đó là bộ điều khiển CAN, và bộ chuyển đổi CAN. Vi điều khiển này sẽ làm việc với phần mềm, phần cứng cộng với phần mềm sẽ ở đó. Logic sẽ được triển khai bằng phần mềm được nạp vào và nó hoạt động, nhưng bộ điều khiển CAN không có phần mềm. Đây hoàn toàn là phần cứng. Toàn bộ giao thức CAN được triển khai trong phần cứng. Không có mã lệnh trong bộ điều khiển CAN này. Có một số cấu hình có thể thực hiện như cấu hình bộ lọc, nhưng giao thức CAN được triển khai trong phần cứng và bộ chuyển đổi CAN cũng hoàn toàn là phần cứng.

Đây là cách mà toàn bộ hệ thống giao tiếp CAN tại mỗi nút hoạt động. Nếu có 10 nút, thì ít nhất sẽ có 10 bộ điều khiển CAN. Số lượng có thể nhiều hơn, nhưng ít nhất sẽ có 10 bộ điều khiển CAN. Hy vọng bạn hiểu phần cơ bản về CAN, bộ chuyển đổi CAN và bộ điều khiển CAN, và lý do tại sao chúng ta cần chúng. Bây giờ, hãy xem phần cuối về cấu trúc CAN.

Nếu có nhiều ECU, chúng ta sẽ có một mạng CAN. Và cấu trúc chúng ta đang sử dụng gọi là cấu trúc bus. Cấu trúc bus nghĩa là gì? Giả sử mạng có N ECU, ECU 1, ECU 2, ECU 3 đến ECU N. Và đây là bus CAN. Bạn có thể thấy CAN high, đây là CAN high, và đây là CAN low. Ở đây tôi đã hiển thị song song, nhưng chúng sẽ được xoắn lại. CAN high và CAN low sẽ được xoắn như bạn đã thấy trước đó. Tại sao? Vì để chống nhiễu. Và chúng kết thúc bằng 120 ohm ở cả hai đầu, ở cuối này và cuối kia. Và ở đây có các kết nối, CAN high, của ECU 1. Vì vậy, ở ECU 1, khi tôi nói có nghĩa là ở đây vi điều khiển, bộ điều khiển CAN, và CAN high. Hai dây này sẽ kết nối CAN high với CAN high của bus. CAN low sẽ kết nối với CAN low của bus cho từng ECU, và vậy là giao tiếp xảy ra, đó là cách mà các ECU được kết nối.

Vì tất cả các ECU này đều kết nối vào cùng một bus CAN, chúng ta gọi đây là tất cả các ECU nằm trong cùng một mạng CAN. Vì vậy, toàn bộ hệ thống này là mạng CAN, với số lượng ECU kết nối vào bus CAN high và CAN low.

Lưu ý rằng tốc độ baud, tức là tốc độ truyền, nếu một ECU được cấu hình là 500 kbps, thì tất cả các ECU này cũng phải là 500 kbps. Tốc độ, hay baud rate mà chúng ta gọi, được cấu hình cho mạng và không phải cho từng ECU riêng biệt. Cấu hình này phải được thực hiện trên từng ECU, nhưng nó phải giống nhau. Không thể có 500 kbps ở đây, 200 kbps ở kia, 1 mbps ở chỗ khác. Không thể có điều đó. Nếu mạng có baud rate là 500 kbps, thì tất cả các ECU kết nối vào mạng CAN này đều phải được cấu hình với tốc độ 500 kbps.

Vậy đó là các tính năng cơ bản của mạng CAN. Trong bài giảng này, chúng ta đã hiểu về tốc độ CAN, tức là CAN có thể đạt đến 1 mbps tối đa, với chiều dài tối đa là 4 mét. Khi chiều dài bus tăng lên, tốc độ tối đa sẽ giảm. Và trong công nghiệp, tiêu chuẩn thường sử dụng 500 kbps. Vì vậy, chiều dài bus có thể lên tới 100 mét. Dữ liệu CAN, frame CAN có một phần PCI, phần liên quan đến giao thức, và phần dữ liệu thực sự cần được truyền tải. Mỗi frame, chúng ta có thể truyền tối đa 8 byte dữ liệu. Chúng ta cũng có thể có ít hơn 8 byte. Và mỗi byte có bao nhiêu byte? Nó thường được hiển thị trong trường DLC của frame, mà bạn sẽ hiểu rõ hơn trong các định dạng frame.

Chúng ta không thể có 10 bit, 12 bit, tất cả phải là bội số của byte để truyền tải trong frame CAN. Sau đó, chúng ta hiểu về logic bus CAN, đó là cặp xoắn. Chúng ta sử dụng điện áp vi sai. Các logic CAN và lý do tại sao chúng ta cần bộ điều khiển CAN, bộ chuyển đổi CAN và kết nối bus CAN.

## Thuộc tính của giao thức Can

Trong bài giảng này, chúng ta sẽ cùng tìm hiểu về các đặc tính của CAN.

Các giao thức truyền thông có nhiều đặc tính khác nhau, và CAN sẽ thuộc vào một trong những loại đặc tính này, được phân loại như thế nào.

Ví dụ, giao tiếp có thể là có dây hoặc không dây, có thể là giao tiếp nối tiếp hoặc song song, đồng bộ hoặc không đồng bộ, có thể là peer-to-peer (điểm-điểm) hoặc phát sóng (broadcast), và cũng có thể là simplex, half-duplex, hoặc full-duplex. Như vậy, có rất nhiều loại giao thức truyền thông khác nhau.

Trong bài giảng này, chúng ta sẽ tìm hiểu về từng loại giao thức này và xác định CAN thuộc vào loại nào. Đặc tính của CAN sẽ được xác định theo loại giao thức mà nó thuộc về.

Cụ thể, chúng ta sẽ cùng xem xét các nội dung sau:

* Giao tiếp có dây hay không dây
* Giao tiếp nối tiếp hay song song
* Giao tiếp đồng bộ hay không đồng bộ
* Peer-to-peer và phát sóng là gì?
* Giao thức CAN là giao thức dựa trên tin nhắn, vậy ý nghĩa của điều này là gì?
* CAN có tính năng lọc tin nhắn
* Simplex, half-duplex, full-duplex là gì, và CAN thuộc loại nào?
* Phương pháp xác nhận trong CAN
* CAN là một giao thức CSMA-CA, ý nghĩa của CSMA-CA là gì?
* CAN có phương pháp xử lý lỗi tạm thời và vĩnh viễn cho các nút (node)

Chúng ta sẽ cùng tìm hiểu tất cả những điều này, vậy hãy bắt đầu.

**1. Giao tiếp có dây và không dây**

Giao tiếp có dây là khi các nút giao tiếp với nhau thông qua các dây dẫn. Dây dẫn có thể là sợi quang, cáp đồng, nhôm, cặp xoắn, không xoắn, hoặc nhiều loại dây khác nhau. Nói chung, tất cả các giao thức truyền thông mà các nút được kết nối qua dây dẫn và môi trường truyền thông là dây dẫn sẽ được gọi là giao tiếp có dây.

Còn giao tiếp không dây là khi các nút không kết nối với nhau qua dây mà sử dụng môi trường điện từ để truyền thông. Ví dụ, điện thoại cố định là một ví dụ điển hình của giao tiếp có dây. Còn điện thoại di động là ví dụ của giao tiếp không dây, vì bạn không cần dây để kết nối với người khác nhưng vẫn có thể thực hiện cuộc gọi.

Một số giao thức không dây điển hình là Bluetooth, Wi-Fi. Còn Ethernet và CAN là ví dụ của giao thức có dây.

CAN là một giao thức có dây, sử dụng cặp dây xoắn để truyền dữ liệu. Khi sử dụng giao tiếp có dây, các nút phải cố định tại chỗ, không thể di chuyển tự do. Ngược lại, giao tiếp không dây cho phép các mô-đun giao tiếp di chuyển tự do. Trong các ứng dụng như xe ô tô, các ECU (Electronic Control Units) thường được cố định và gần nhau, vì vậy CAN sử dụng giao thức có dây, điều này cũng giúp nó đáng tin cậy hơn so với giao tiếp không dây.

**2. Giao tiếp nối tiếp và song song**

Giao tiếp song song là khi nhiều dây được sử dụng để truyền nhiều bit cùng một lúc. Ví dụ, một bus song song 8 bit có thể truyền 8 bit cùng lúc. Trong khi đó, giao tiếp nối tiếp chỉ truyền một bit một lần. Dù trong CAN có hai dây (CAN High và CAN Low), nhưng CAN vẫn là một giao thức nối tiếp vì mỗi lần chỉ truyền một bit dữ liệu.

Giao tiếp song song thường dùng trong các bus dữ liệu, bus địa chỉ, bus điều khiển trong các vi điều khiển, và các mô-đun phải rất gần nhau để đảm bảo độ tin cậy. Ví dụ, nếu vi điều khiển được gọi là 8-bit, điều này có nghĩa là bus địa chỉ của nó có thể mang 8 bit tại một thời điểm.

**3. Giao tiếp đồng bộ và không đồng bộ**

Giao tiếp đồng bộ là khi các nút cần đồng bộ với nhau qua một tín hiệu đồng hồ để quyết định thời gian bắt đầu truyền dữ liệu. Nếu một giao thức là đồng bộ, khi một nút có dữ liệu sẵn sàng để gửi, nó phải đợi đến lượt mình trong một khe thời gian cụ thể. Trong khi đó, giao tiếp không đồng bộ cho phép nút gửi dữ liệu bất kỳ lúc nào khi nó sẵn sàng.

Với CAN, nó là một giao thức không đồng bộ, có nghĩa là một nút có thể bắt đầu truyền tin khi dữ liệu đã sẵn sàng mà không cần phải đợi đến một thời điểm nhất định trong quy trình đồng bộ.

**CAN không phải chờ một khoảng thời gian nào hoặc tham chiếu đồng hồ, đúng không?**

Tôi đã chuẩn bị sẵn khung tin (frame) và có thể bắt đầu truyền tải bất kỳ lúc nào. Tôi có thể bắt đầu, tôi có thể truyền tải vào bất kỳ thời điểm nào.

Nếu bus (mạng) đang bận, tôi sẽ phải đợi, nhưng việc chờ đợi là vì bus bận, chứ không phải vì có một ràng buộc về thời gian, đúng không?

Vì vậy, nếu bus không bận, và nếu tôi có một khung tin sẵn, tôi có thể bắt đầu truyền tải mà không cần phải lo lắng về thời gian hay phải đồng bộ với một đồng hồ nào khác. Tôi chỉ cần truyền tải khung tin mà không cần phải quan tâm đến đồng hồ hay thời gian.

Tất cả các nút (node) khác sẽ tự động đồng bộ với nút phát (transmitter) chỉ bằng dữ liệu trong khung tin đó, nếu là giao tiếp không đồng bộ.

Trong giao tiếp đồng bộ, tất cả các nút nhận (Rx) và nút phát (Tx) sẽ đồng bộ nhờ một tín hiệu đồng hồ riêng biệt hoặc chu kỳ đồng hồ.

Vậy nguồn gốc của sự đồng bộ sẽ quyết định xem đó là giao tiếp đồng bộ hay không đồng bộ. CAN là giao thức giao tiếp không đồng bộ, đúng không?

Vì vậy, CAN không có tín hiệu đồng hồ đi kèm. Khi có khung tin sẵn, nó có thể bắt đầu giao tiếp, và khung tin đó sẽ tự nó trở thành nguồn đồng bộ hóa giữa các nút phát và nút nhận.

**Peer-to-peer và Broadcast:**

Peer-to-peer giống như một chiếc điện thoại di động, trong khi Broadcast giống như sóng radio FM.

Peer-to-peer có nghĩa là giao tiếp chỉ có thể diễn ra giữa một người và một người khác. Ví dụ trong sơ đồ này, nếu một nút muốn giao tiếp với nút khác, thì đây là giao tiếp peer-to-peer.

Khi cần giao tiếp với một người khác, một kết nối trực tiếp sẽ được tạo ra. Các nút khác không liên quan gì đến việc truyền tải, và họ không biết dữ liệu được truyền là gì.

Broadcast giống như sóng radio FM. Một trạm radio FM phát sóng tín hiệu và mọi người trong mạng sẽ nhận được tín hiệu đó. Trạm phát không biết ai là người nhận, và người nhận cũng không biết ai là người phát. Tuy nhiên, vì họ cùng ở trong mạng, nên họ sẽ nhận ra rằng có một khung tin đang đến.

Ethernet là một ví dụ về giao thức peer-to-peer, trong đó các nút được kết nối thông qua một công tắc (switch). Còn các giao thức broadcast là CAN và FlexRay.

CAN là một giao thức multicast, vì một nút có thể truyền khung tin đến nhiều nút khác, nhưng không phải tất cả các nút đều cần nhận khung tin đó. Một số nút có thể lựa chọn bỏ qua khung tin này bằng cách lọc, mà tôi sẽ giải thích chi tiết hơn sau.

**Giao thức CAN là một giao thức dựa trên thông điệp, không phải là giao thức dựa trên nút.**

Một giao thức có thể là giao thức dựa trên thông điệp hoặc giao thức dựa trên nút. Nếu bạn nghĩ về điện thoại di động, bạn có thể thấy rằng giao tiếp qua điện thoại di động là một giao thức dựa trên nút, vì số điện thoại di động là một ID duy nhất cho mỗi thiết bị.

Trong CAN, khung tin được xác định bởi một ID thông điệp, không phải bằng địa chỉ nút phát hay nút nhận. Điều này có nghĩa là bạn không cần biết ai là nút phát hoặc nút nhận, mà chỉ cần biết ID của thông điệp.

Mỗi khung tin trong CAN đều có một ID thông điệp, và ID này có ba mục đích:

1. **Xác định loại dữ liệu**: Ví dụ, nếu ID thông điệp là 5F2, thì đây có thể là thông tin về số km mà xe đã di chuyển.
2. **Quyết định độ ưu tiên của thông điệp**: ID thông điệp nhỏ hơn sẽ có độ ưu tiên cao hơn.
3. **Lọc thông điệp**: Nếu một nút không muốn nhận một loại thông điệp cụ thể, nó có thể lọc thông điệp đó dựa trên ID thông điệp.

**Lọc thông điệp trong CAN**:

Lọc thông điệp là một tính năng quan trọng trong CAN. Cụ thể, mỗi nút trong CAN có thể lọc những khung tin mà nó không muốn nhận. Quá trình này dựa trên ID thông điệp.

Mỗi nút có thể có một bộ lọc, và nó sẽ kiểm tra ID của khung tin trước khi quyết định có nhận hay không. Nếu một nút không muốn nhận thông tin về số km, ví dụ, nó có thể bỏ qua khung tin chứa ID thông điệp 5F2.

Các giá trị cần thiết để lọc bao gồm **mask** và **filter value**. Nếu sau khi áp dụng **mask**, ID thông điệp và giá trị bộ lọc trùng khớp, thì thông điệp sẽ được nhận.

**CAN sử dụng giao tiếp bán song công (half-duplex)**:

Trong CAN, tại một thời điểm, một nút có thể hoạt động như một nút phát hoặc nút nhận, nhưng không thể đồng thời phát và nhận. Điều này giống như khi hai người nói chuyện, một người nói, người kia lắng nghe, rồi sau đó đổi lại.

CAN là một giao thức bán song công (half-duplex), tức là một nút chỉ có thể phát hoặc nhận tại một thời điểm, không thể đồng thời thực hiện cả hai.

Và có một số giao thức không sử dụng cơ chế **xác nhận** (acknowledgement).

Người gửi sẽ gửi một thông điệp, không cần phải có xác nhận, và tiếp tục gửi thông điệp tiếp theo, giả sử rằng thông điệp trước đã được nhận đúng.

Tiếp tục gửi thông điệp tiếp theo mà không cần phải xác nhận, đúng không? Không có khái niệm xác nhận trong giao thức này.

Các giao thức trên Internet và các giao thức khác cũng hoạt động theo cách này. Các gói tin sẽ được gửi liên tiếp, không cần xác nhận. Nếu có gói tin bị mất, một thông điệp phản hồi sẽ được gửi yêu cầu gửi lại thông điệp đó.

Tuy nhiên, nếu thông điệp đã được nhận đúng, sẽ không có xác nhận, thông điệp sẽ cứ thế được gửi liên tiếp mà không cần phản hồi.

Vấn đề với hệ thống này là không có đảm bảo rằng thông điệp đã được truyền đi sẽ được nhận đúng.

Vấn đề nữa là cần phải có một khung xác nhận riêng biệt, điều này sẽ làm tăng tải cho bus, tạo ra một khung dữ liệu không mang thông tin gì cả. Nó chỉ đơn giản thông báo rằng "vâng, tôi đã nhận thông điệp đó đúng" và điều này sẽ là một sự lãng phí thời gian.

Tuy nhiên, trong giao thức CAN, một cách thức xác nhận mới và sáng tạo đã được giới thiệu. Đây là một phương thức xác nhận rất, rất tuyệt vời.

Điều gì xảy ra là khi một khung dữ liệu được gửi từ người gửi đến các nút nhận, trước khi khung dữ liệu kết thúc, trong một khoảng thời gian rất ngắn của quá trình truyền tải, việc xác nhận sẽ diễn ra.

Có nhiều trường (field) trong khung dữ liệu mà tôi sẽ giải thích chi tiết khi trình bày về các định dạng khung trong giao thức CAN. Nhưng bạn chỉ cần biết rằng trong quá trình truyền tải dữ liệu, có một trường đặc biệt gọi là **trường xác nhận**, trong đó khi phần dữ liệu đang được truyền tải, tất cả các nút nhận sẽ thực hiện xác nhận xem dữ liệu có được nhận đúng hay không.

Để làm điều này, trước khi khung dữ liệu được truyền xong, tất cả các nút cần phải biết liệu dữ liệu đã được truyền đúng chưa. Sau đó, khi đến trường xác nhận, các nút sẽ xác nhận hoặc không xác nhận. Nếu dữ liệu đúng, họ sẽ xác nhận, nếu không đúng, họ sẽ không xác nhận.

Vì vậy, trước khi người truyền tải hoàn thành việc truyền khung dữ liệu, họ sẽ nhận được thông tin về việc có nhận được xác nhận hay không. Quy trình xác nhận này đã hoàn tất ngay cả trước khi khung dữ liệu hoàn toàn được truyền tải.

Đây là một khái niệm rất đẹp và chưa từng tồn tại trước khi giao thức CAN ra đời. Những người phát triển giao thức CAN, đặc biệt là những người từ Bosch, là những chuyên gia thực thụ và có một trí tuệ tuyệt vời.

Tôi sẽ giải thích cách thức xác nhận này hoạt động, cũng như cách mà người truyền tải nhận biết việc có xác nhận hay không khi tôi giải thích chi tiết về các trường xác nhận trong định dạng khung của giao thức CAN. Nhưng hiện tại, bạn chỉ cần biết rằng có một trường xác nhận trong chính khung dữ liệu, và người truyền tải sẽ nhận được xác nhận ngay cả trước khi khung dữ liệu được hoàn thành.

Đây là một phương thức xác nhận rất sáng tạo của giao thức CAN, và nó là đặc trưng riêng của giao thức này.

Tiếp theo, chúng ta sẽ nói về giao thức **CSMA-CA**.

Bạn có thể đã nghe đến thuật ngữ này, hoặc có thể đây là một thuật ngữ hoàn toàn mới đối với bạn. CSMA-CA là viết tắt của **Carrier Sense Multiple Access Collision Avoidance** (Giao thức Truy cập Nhiều Qua Cảm biến Tín hiệu mang và Tránh Va chạm).

Giao thức CSMA-CA là gì? **CSMA** là nơi mà người truyền tải cố gắng xác định liệu có một truyền tải khác đang diễn ra trước khi bắt đầu truyền tải dữ liệu, sử dụng cơ chế cảm biến tín hiệu mang (carrier sense mechanism).

Như tôi đã nói, giao thức CAN là một giao thức **asynchronous** (không đồng bộ), vì vậy không có đồng bộ thời gian, không có thời gian cụ thể dành cho mỗi nút. Khi mỗi nút sẵn sàng truyền tải một khung dữ liệu, trước tiên nó phải đảm bảo rằng không có khung dữ liệu nào khác đang truyền trên bus.

Và nếu bus đang rảnh, thì nó mới có thể bắt đầu truyền tải khung dữ liệu.

Làm sao mỗi nút biết được liệu có khung nào đang được truyền hay không? Bằng cách **lắng nghe** bus. Nếu có một tín hiệu mang hiện hữu, tức là có một khung dữ liệu đang được truyền, thì nó sẽ không bắt đầu truyền tải. Nếu không có tín hiệu mang nào trên bus, nó sẽ bắt đầu truyền tải.

Đây là tính năng CSMA. Vì CAN là giao thức **asynchronous**, nên CAN cũng là một giao thức thuộc loại CSMA.

Về phần xử lý va chạm, có ba loại: **Tránh Va chạm** (Collision Avoidance - CA), **Phát hiện Va chạm** (Collision Detection - CD), và **Giải quyết Va chạm** (Collision Resolution - CR).

**Phát hiện Va chạm** là khi có sự va chạm giữa các khung dữ liệu. Khi va chạm xảy ra, giao thức cần nhận biết được và thực hiện hành động phù hợp.

**Giải quyết Va chạm** là khi va chạm xảy ra giữa các khung dữ liệu, giao thức sẽ giải quyết tình huống đó tại thời điểm xảy ra va chạm.

**Tránh Va chạm** là khi các khung dữ liệu đang được truyền, giao thức thiết kế sao cho va chạm không thể xảy ra. Nếu có tình huống nào có thể gây ra va chạm, giao thức sẽ tránh nó.

Vì giao thức CAN sử dụng tính năng tránh va chạm (Collision Avoidance - CA), nên chúng ta nói rằng CAN là một giao thức CSMA-CA.

Vậy CAN tránh va chạm như thế nào nếu nhiều khung dữ liệu được gửi cùng lúc từ nhiều nút khác nhau trên bus? Nếu bus rảnh và nhiều nút cùng cố gắng gửi khung dữ liệu, làm sao chúng không bị va chạm với nhau?

Câu hỏi này sẽ được giải thích trong các bài giảng sau, khi tôi giải thích về **quá trình phân quyền** (arbitration process), trong đó khung dữ liệu có **ưu tiên cao hơn** sẽ giành quyền sử dụng bus, đảm bảo không có va chạm xảy ra. Quá trình phân quyền này đảm bảo rằng va chạm sẽ không xảy ra.

Quá trình này sẽ được giải thích chi tiết trong các bài giảng sau. Hiện tại, bạn chỉ cần nhớ rằng CAN là một giao thức CSMA-CA, và bạn đã hiểu ý nghĩa của CSMA và CA.

Tiếp theo, chúng ta sẽ nói về **lỗi tạm thời và lỗi vĩnh viễn của nút**.

Trong giao thức CAN, có sự phân biệt giữa lỗi **tạm thời** và lỗi **vĩnh viễn** của nút. Nếu một nút gặp lỗi tạm thời, nghĩa là trong khi truyền tải một khung dữ liệu, khung đó bị lỗi tạm thời do nút gặp sự cố, nhưng nút này không bị hỏng hoàn toàn. Khi lỗi tạm thời xảy ra, khung bị lỗi sẽ bị hủy và sẽ được gửi lại từ người truyền tải.

Với lỗi **vĩnh viễn**, nếu nút gặp lỗi và tiếp tục phá hủy các khung dữ liệu trên bus, nút đó sẽ bị đưa vào trạng thái lỗi vĩnh viễn, không thể gửi hoặc nhận khung dữ liệu nữa, và sẽ không thể phá hủy khung dữ liệu nào trên bus.

Có ba trạng thái trong CAN liên quan đến lỗi: **lỗi hoạt động** (error active), **lỗi thụ động** (error passive), và **bus off**. Giao thức CAN cũng có các cơ chế để phục hồi sau lỗi. Các cơ chế này sẽ được giải thích chi tiết trong các bài giảng sau.

Tóm lại, giao thức CAN phân biệt giữa lỗi tạm thời và lỗi vĩnh viễn của nút, cũng như các cơ chế phục hồi sau lỗi.

**Tóm tắt nhanh về các đặc điểm của CAN**:

* CAN có hai loại: **có dây** và **không dây**.
* Giao thức CAN là **asynchronous**, tức là không có đồng hồ đồng bộ giữa các nút.
* **CSMA-CA**: CAN sử dụng cơ chế **Carrier Sense Multiple Access (CSMA)** để kiểm tra tín hiệu mang trên bus và tránh va chạm bằng cách sử dụng **Collision Avoidance (CA)**.
* CAN có một cơ chế **xác nhận** rất sáng tạo trong khung dữ liệu, nơi quá trình xác nhận được hoàn thành trước khi khung dữ liệu được truyền hoàn toàn.
* CAN có thể xử lý **lỗi tạm thời** và **lỗi vĩnh viễn** của nút với các cơ chế phục hồi lỗi.

# Khung của Can

## Các loại khung của Can

Giao thức CAN có bốn loại khung.

Có bốn loại khung trong giao thức CAN.

Được rồi.

Chúng gồm: khung dữ liệu (data frame), khung yêu cầu (remote frame), khung lỗi (error frame) và khung quá tải (overload frame).

Trong bài giảng này, chúng ta sẽ xem qua các khái niệm cơ bản của mỗi loại khung, và trong các bài giảng sau, chúng ta sẽ tìm hiểu chi tiết về định dạng của chúng, lý do tại sao chúng được sử dụng và tất cả các khía cạnh khác liên quan.

Vậy thì, đầu tiên chúng ta sẽ tìm hiểu về **khung dữ liệu**.

Đây là khung chính, đúng không?

Khi chúng ta nói về các khung CAN trong các bài giảng của mình, thì mặc định chúng ta đang nói đến **khung dữ liệu**.

Khung dữ liệu là khung mang dữ liệu từ nút phát (transmitter node) đến tất cả các nút nhận (receiver nodes).

Đây chính là khung phục vụ mục đích chính của giao thức CAN.

Và trong khung dữ liệu, có hai loại khung: **khung tiêu chuẩn** và **khung mở rộng**.

Sự khác biệt giữa chúng là gì?

Khung tiêu chuẩn có **ID tin nhắn** dài 11 bit, trong khi khung mở rộng có **ID tin nhắn** dài 29 bit, được chia thành 11 bit cho ID cơ bản và 18 bit cho ID mở rộng.

Để hiểu rõ hơn về sự phân biệt giữa khung tiêu chuẩn và khung mở rộng, chúng ta cần quay lại nhu cầu của giao thức CAN và một chút lịch sử về nó.

Ban đầu, khi giao thức CAN được thiết kế và định dạng các khung dữ liệu được phát triển, họ đã dành 11 bit cho **ID tin nhắn**. Giao thức CAN là giao thức dựa trên tin nhắn, và **ID tin nhắn** là thứ giúp xác định loại dữ liệu trong mỗi khung.

Ví dụ, nếu khung dữ liệu chứa thông tin về **tuổi xe** (KM age) hoặc **trạng thái túi khí** (airbag status), hoặc **trạng thái dây an toàn** (seatbelt status), mỗi loại dữ liệu sẽ có một ID tin nhắn khác nhau. Cách để phân biệt các khung dữ liệu này là chúng có ID tin nhắn khác nhau.

Với 11 bit cho ID tin nhắn, có thể có **2^11** (512) loại tin nhắn khác nhau trong mạng CAN. Tuy nhiên, một số bit được dự trữ, mà tôi sẽ giải thích sau.

Vậy với ID tin nhắn dài 11 bit, bạn có thể có tối đa 512 tin nhắn khác nhau trong mạng CAN. Tuy nhiên, khi ngành công nghiệp ô tô phát triển và nhu cầu cần nhiều hơn 512 loại tin nhắn, thì 11 bit cho ID tin nhắn không còn đủ.

Vì vậy, họ đã quyết định tăng số bit cho ID của khung dữ liệu từ 11 lên 29 bit. Nhưng họ không muốn từ bỏ hoàn toàn khái niệm 11 bit, vì vậy họ giữ khung cũ với **ID 11 bit** và đồng thời bổ sung khung mới với **ID 29 bit**.

Như vậy, khung tiêu chuẩn sử dụng 11 bit cho ID tin nhắn và khung mở rộng sử dụng 29 bit. Trong khung mở rộng, 11 bit đầu tiên là giống như ID của khung tiêu chuẩn, còn 18 bit còn lại là phần mở rộng.

Mục đích của việc chia như vậy là để đảm bảo rằng cả hai loại khung có thể cùng tồn tại trong một mạng CAN. Cách thức chúng có thể cùng tồn tại sẽ được giải thích rõ hơn khi tôi trình bày về định dạng khung dữ liệu trong các bài giảng sau.

Tạm thời, bạn chỉ cần nhớ rằng khung tiêu chuẩn có ID tin nhắn dài 11 bit, còn khung mở rộng có ID tin nhắn dài 29 bit.

Và trong giao thức CAN, khi chúng ta thường nói về các khung, chúng ta đang đề cập đến **khung dữ liệu**.

Khung dữ liệu trong CAN có một số trường, và tôi sẽ giải thích về định dạng chi tiết của khung dữ liệu trong bài giảng tiếp theo.

Tiếp theo là **khung yêu cầu** (remote frame).

Khung yêu cầu là khung dùng để yêu cầu một khung dữ liệu.

Giả sử có nhiều nút trong mạng và mỗi nút có các khung dữ liệu riêng để truyền tải. Nhưng giả sử, tôi có một nút và tôi cần một khung dữ liệu về trạng thái dây an toàn, xem có người lái có thắt dây an toàn hay không. Nhưng trách nhiệm cung cấp trạng thái này là của một nút khác.

Ví dụ, tôi là một nút trong mạng CAN, và có một nút khác là hệ thống **phát hiện người ngồi** và một nút khác nữa là hệ thống **phanh**. Giả sử tôi muốn biết khi nào phanh được áp dụng và mức độ áp suất phanh. Tuy nhiên, tôi không có cảm biến nào để biết trạng thái dây an toàn. Trách nhiệm đó là của một nút khác, và nó sẽ gửi thông tin về trạng thái dây an toàn theo chu kỳ.

Giả sử rằng nút đó có thể đã bỏ lỡ việc gửi khung chứa dữ liệu về trạng thái dây an toàn. Nhưng ngay lúc này, tôi cần khung dữ liệu đó. Làm sao tôi có thể có được dữ liệu này?

Để làm được điều đó, tôi sẽ gửi một **khung yêu cầu** (remote frame) với cùng ID tin nhắn trên bus. Khi khung yêu cầu này được gửi, tất cả các nút trong mạng đều nghe thấy nó và hiểu rằng tôi yêu cầu một khung dữ liệu về trạng thái dây an toàn, với ID tin nhắn ví dụ là **7F1**.

Nút cung cấp thông tin về dây an toàn sẽ nhận thấy khung yêu cầu này và gửi lại khung dữ liệu chứa thông tin trạng thái dây an toàn.

Như vậy, **khung yêu cầu** là khung dùng để yêu cầu một khung dữ liệu, và khi khung yêu cầu hoàn tất việc truyền tải, nó sẽ được theo sau bởi khung dữ liệu mà nó yêu cầu.

Đó là những khái niệm cơ bản về các loại khung trong giao thức CAN. Trong các bài giảng sau, chúng ta sẽ đi sâu vào các chi tiết về cách các khung này được định dạng và sử dụng trong mạng CAN.

**Tóm tắt nhanh**:

* **Khung dữ liệu**: Là khung chính trong giao thức CAN, mang dữ liệu từ nút phát đến các nút nhận. Có hai loại: khung tiêu chuẩn (11 bit ID) và khung mở rộng (29 bit ID).
* **Khung yêu cầu**: Là khung yêu cầu dữ liệu từ một nút khác trong mạng. Khi nhận được khung yêu cầu, nút có dữ liệu sẽ phản hồi bằng khung dữ liệu tương ứng.
* Lưu ý rằng trong trường hợp này, ABS là khung gửi khung remote, nhưng nó không biết ai sẽ là người gửi khung dữ liệu thực tế, bởi vì đây là một mạng dạng bus. Mọi nút trong mạng sẽ đều nghe thấy khung remote này, và khi khung remote được gửi đi, bất kỳ nút nào là người phát khung dữ liệu đó sẽ tiếp tục gửi và chia sẻ khung dữ liệu đó.
* Tuy nhiên, nếu bạn nhìn vào định dạng của khung remote, bạn sẽ thấy rằng nó tương tự như khung khi mạng không có tín hiệu (idle bus). Các trường như arbitration, điều khiển (control field), CRC, VFS, và interframe space (IFS) đều giống như trong khung dữ liệu. Điểm khác biệt duy nhất là trường dữ liệu (data field) trong khung remote bị thiếu. Nếu chúng ta loại bỏ trường dữ liệu này, định dạng của khung remote và khung dữ liệu gần như giống nhau. Tuy nhiên, vẫn có một vài sự khác biệt trong các trường arbitration và control, nhưng nhìn chung về cấu trúc, chúng khá tương tự.
* Khung dữ liệu và khung remote có cấu trúc gần như giống nhau, chỉ có một hoặc hai điểm khác biệt. Tuy nhiên, cách xử lý khung remote có thể khác nhau giữa các bộ điều khiển khác nhau. Chính vì lý do này, khái niệm khung remote không còn được sử dụng nhiều trong ngành công nghiệp hiện nay. Ban đầu, khi giao thức CAN ra đời, khung remote rất hữu ích, nhưng hiện tại khung remote đã trở nên lỗi thời và không còn được sử dụng nhiều. Tuy nhiên, nếu bạn muốn hiểu lý do tại sao khung remote bị loại bỏ, bạn có thể tham khảo báo cáo CIA 8.2, được gọi là CIA Scanning Automation, một tổ chức chịu trách nhiệm phát triển và cải tiến giao thức CAN. Bạn có thể tìm báo cáo này trên Google để hiểu lý do khung remote bị loại bỏ.
* Tiếp theo, chúng ta sẽ tìm hiểu về loại khung thứ ba, đó là **khung lỗi (error frame)**. Khung lỗi dùng để báo hiệu tình trạng lỗi trong khung dữ liệu đang được truyền trên bus. Khi một khung dữ liệu đang được truyền, nếu nút phát hiện thấy có lỗi trong khung đó, nó sẽ gửi khung lỗi để phá huỷ khung dữ liệu đang truyền. Tương tự, nếu một nút nhận dữ liệu phát hiện có lỗi trong khung dữ liệu, nó cũng có thể gửi khung lỗi để huỷ khung dữ liệu hiện tại.
* Mục đích của khung lỗi là để báo hiệu cho tất cả các nút trong mạng rằng khung dữ liệu hiện tại hoặc khung remote hiện tại đang có lỗi, và khung đó sẽ bị huỷ bỏ. Sau khi khung lỗi được truyền, nút đã phát khung dữ liệu có thể gửi lại khung dữ liệu đó. Định dạng của khung lỗi bao gồm cờ lỗi (error flag) và trình phân biệt lỗi (error delimiter), cùng với khoảng cách giữa các khung (interframe space). Khi khung lỗi được truyền, nó sẽ phá huỷ bất kỳ khung dữ liệu nào chưa hoàn tất, và sau khi khung lỗi truyền xong, khung dữ liệu sẽ được gửi lại.
* Có nhiều lý do gây ra lỗi, ví dụ như lỗi của nút phát (transmitter errors) và lỗi của nút nhận (receiver errors). Trong các bài học sau, chúng ta sẽ tìm hiểu chi tiết về các loại lỗi trong giao thức CAN và định dạng của khung lỗi, cũng như ý nghĩa của các cờ lỗi và các bit trong khung lỗi. Tuy nhiên, hiện tại bạn chỉ cần nhớ rằng khung lỗi có mặt trong giao thức CAN để xử lý các tình huống lỗi trong quá trình truyền dữ liệu.
* Cuối cùng, loại khung thứ tư trong giao thức CAN là **khung quá tải (overload frame)**. Khung quá tải được truyền đi khi một nút bị quá tải và cần thêm thời gian để xử lý khung dữ liệu đã nhận trước đó. Ví dụ, giả sử bạn là một nút trong mạng CAN và vừa nhận được một khung dữ liệu. Bạn đã đưa dữ liệu đó vào bộ đệm, nhưng vì bị quá tải, bạn cần thêm thời gian để xử lý khung dữ liệu đó. Nếu bạn không giữ bus bận, một nút khác có thể nhận thấy bus trống và bắt đầu truyền khung dữ liệu của nó.
* Để ngăn điều này xảy ra, bạn có thể gửi một khung quá tải để làm cho bus luôn bận, tránh để các nút khác bắt đầu truyền khung dữ liệu mới khi bạn chưa sẵn sàng. Khung quá tải có định dạng giống với khung lỗi. Tuy nhiên, khung quá tải sẽ được gửi sau khi khung dữ liệu đã được truyền và thường được sử dụng để giữ bus bận trong khi nút xử lý khung dữ liệu đã nhận. Trong thực tế, các bộ điều khiển hiện đại thường có tốc độ xử lý rất nhanh, và vì vậy khung quá tải rất hiếm khi xảy ra trên bus CAN.
* Trong bốn loại khung của giao thức CAN, khung remote và khung quá tải hiện nay rất hiếm khi được sử dụng, trong khi khung lỗi cũng không xuất hiện nhiều. Do đó, các khung chính mà chúng ta thường gặp trong mạng CAN chủ yếu là khung dữ liệu với các ID thông điệp khác nhau.
* Nhưng theo giao thức CAN, bạn cần hiểu rằng có bốn loại khung này và mục đích của chúng. Trong bài học tiếp theo, chúng ta sẽ tìm hiểu chi tiết về định dạng của khung dữ liệu, bao gồm cả định dạng khung chuẩn (standard frame) và khung mở rộng (extended frame). Chúng ta cũng sẽ tìm hiểu sự khác biệt giữa định dạng của khung dữ liệu và khung remote.
* Cảm ơn các bạn đã theo dõi bài học này, và hẹn gặp lại các bạn trong bài học tiếp theo.
* Tạm biệt!

## Định dạng khung dữ liệu tiêu chuẩn

**Giảng viên**: Chào các bạn sinh viên.

Chào mừng các bạn đến với bài giảng này.

Trong bài giảng hôm nay, chúng ta sẽ học về định dạng khung (frame) của khung dữ liệu CAN.

Hãy chú ý kỹ trong bài giảng này, vì toàn bộ giao thức CAN xoay quanh định dạng khung, và hầu hết các kiến thức quan trọng sẽ được trình bày trong bài giảng này.

Khi không có khung nào trên bus, bus sẽ ở trạng thái nhàn rỗi (idle), và nếu bạn đo bus, bạn sẽ thấy nó có một bit recessive.

Ngoài ra, nhớ rằng bit recessive là logic một, còn bit dominant là logic không, như bạn đã học trước đó.

Vậy, mỗi khung dữ liệu bắt đầu bằng một bit dominant gọi là **bắt đầu khung (start of frame)**.

Trường bắt đầu khung luôn có độ dài 1 bit và là bit dominant.

Mục đích của trường này là thông báo cho tất cả các nút (nodes) rằng một khung mới bắt đầu trên bus, và bus không còn ở trạng thái nhàn rỗi nữa.

Trường này sẽ được sử dụng để đồng bộ hóa tất cả các nút nhận (Rx) với nút truyền (Tx) nữa.

Khi khung đã bắt đầu, chúng ta cần giới thiệu khung dữ liệu, đúng không?

Vì CAN là một giao thức dựa trên thông điệp (message-based protocol), mỗi khung được xác định bằng một **ID thông điệp (message ID)**.

Trong khung chuẩn (standard frame), **message ID** có độ dài 11 bit.

Trước tiên, chúng ta sẽ học về định dạng khung chuẩn, sau đó sẽ tìm hiểu về định dạng khung mở rộng.

Hầu hết các driver của CAN đã che giấu bit thứ 11 và bit thứ 4 vì một lý do phức tạp nào đó.

Vì vậy, trong thực tế, chỉ có 9 bit trong tổng số 11 bit của **message ID** là khác nhau trong các khung CAN.

Với 9 bit trong tổng số 11 bit của **message ID**, chúng ta có thể có 2^9 = 512 khung CAN khác nhau có thể có trên một mạng CAN với định dạng khung chuẩn.

**Message ID** có thể có giá trị từ **HEXA decimal 000** đến **HEXA decimal 7FF**.

Sau **message ID**, chúng ta cần cho bus biết đó là khung dữ liệu hay khung yêu cầu dữ liệu, vì cả khung dữ liệu và khung yêu cầu dữ liệu đều có cùng **message ID**.

Để phân biệt chúng, trường tiếp theo sẽ là một bit đơn gọi là **RTR (Remote Transmit Request)**.

Nếu bit **RTR** là 1, thì khung đó là khung yêu cầu dữ liệu (remote frame), và nếu là 0, thì đó là khung dữ liệu (data frame).

Các bạn có thể dễ dàng nhớ điều này: bit dominant dành cho khung dữ liệu và bit recessive dành cho khung yêu cầu dữ liệu. Bạn có thể nhớ qua cách đơn giản là **D** cho Dữ liệu (Data) và **R** cho Yêu cầu (Request).

Sau bit **RTR**, tiếp theo là hai bit dự phòng gọi là **R1** và **R0**.

Các bit này không có mục đích đặc biệt khi giao thức CAN ban đầu được thiết kế. Tuy nhiên, giống như các nhà thiết kế đã biết trước rằng họ sẽ nâng cấp giao thức CAN trong tương lai, họ đã để lại không gian cho việc này bằng cách thêm hai bit dự phòng.

Những bit này luôn là bit recessive.

Như đã dự đoán, hiện nay hai bit này được sử dụng cho các mục đích khác nhau.

Một bit được sử dụng để phân biệt giữa khung chuẩn và khung mở rộng, bit còn lại được sử dụng để phân biệt giữa khung CAN và CAN-FD.

Tuy nhiên, chúng ta sẽ chỉ nói về chúng trong các bài giảng sau khi cần thiết.

Hiện tại, hãy coi những bit này như là bit dự phòng, luôn luôn recessive.

Trường tiếp theo là **DLC**, là mã độ dài dữ liệu (Data Length Code).

Trường này chứa mã hoặc số để thông báo cho các nút về độ dài dữ liệu chứa trong khung này.

Như bạn đã học trong các bài giảng trước, khung CAN có thể chứa dữ liệu với độ dài thay đổi. Nó có thể có từ 0 byte đến 8 byte trong một khung CAN duy nhất.

Số byte dữ liệu trong khung CAN này được chỉ ra bởi **DLC**.

Vì chúng ta cần 4 bit để biểu diễn số 8 (mã nhị phân là 1000), nên trường **DLC** có độ dài 4 bit.

Vậy nên **DLC** là trường 4 bit, với phạm vi từ 0 đến 8.

Sau **DLC**, dữ liệu thực sự sẽ được gửi trong khung.

Dữ liệu mà các nút muốn truyền tải sẽ được gửi tại đây.

Tối đa 8 byte dữ liệu có thể được gửi trong một khung, và tối thiểu là 0 byte, nhưng luôn luôn là bội số của 1 byte.

Tiếp theo là trường **CRC**, là trường kiểm tra lỗi.

Bất kỳ bit nào đã được gửi cho đến lúc này mà bị thay đổi do sự cố trên bus, chúng ta cần một cách để phát hiện nó.

Vì vậy, nút Tx sẽ tính toán một mã CRC dài 15 bit và gửi nó sau trường dữ liệu.

Giao thức CAN sử dụng phương pháp BCH để tính toán mã này.

Nút Rx cũng sẽ tính toán mã CRC riêng của mình cho tất cả các bit từ SOF đến hết trường dữ liệu và so sánh mã CRC đã tính toán với mã CRC trong trường **CRC**.

Nếu hai mã CRC này giống nhau, khung được coi là hợp lệ; ngược lại, khung sẽ bị coi là không hợp lệ.

Tất cả các phép tính và so sánh này để xác thực khung xảy ra trong một cửa sổ thời gian 1 bit, và bit đó là **CRC delimiter** (bit phân tách CRC), theo sau các bit CRC.

Mục đích duy nhất của bit phân tách CRC là cung cấp thời gian cho tất cả các nút nhận để xác thực dữ liệu và khung nhận được và chuẩn bị cho việc xác nhận.

Tiếp theo bit phân tách CRC là **slot xác nhận (acknowledgement slot)**, nơi tất cả các nút nhận có thể xác nhận rằng khung đã được nhận và hợp lệ.

Khung này hoạt động như thế nào?

Nút truyền (Tx) sẽ đưa vào **bit recessive** trong slot xác nhận.

Trong khi đó, tất cả các nút nhận mà cho rằng dữ liệu và khung là hợp lệ sẽ thay đổi bit recessive này thành bit dominant trên bus.

Như chúng ta đã biết, nếu một nút gửi bit recessive và một nút khác gửi bit dominant cùng lúc trên bus, bus sẽ chọn giá trị bit dominant.

Khi nút truyền kiểm tra giá trị trên bus trong slot xác nhận, nếu nó tìm thấy bit dominant, nghĩa là nó đã nhận được xác nhận và khung đã được xác thực.

Ngược lại, nếu nó thấy rằng slot xác nhận vẫn chỉ có bit recessive, điều đó có nghĩa là nó không nhận được xác nhận, gây ra lỗi gọi là lỗi xác nhận.

Chúng ta sẽ tìm hiểu về lỗi xác nhận này trong các bài giảng sau về lỗi CAN.

Vì vậy, **slot xác nhận** là một bit duy nhất, nút Tx đưa vào bit recessive trong đó, và nút Rx muốn xác nhận sẽ thay đổi nó thành bit dominant.

Cần ít nhất một nút Rx để gửi xác nhận về nút Tx qua bit này, nhưng có thể có nhiều nút gửi xác nhận, và tất cả sẽ được gộp vào bit này.

Sau **slot xác nhận**, chúng ta có **bit phân tách xác nhận**, cũng là một bit duy nhất và luôn recessive.

Trường cuối cùng của khung dữ liệu là **end of frame** (kết thúc khung), là một chuỗi gồm bảy bit recessive liên tiếp. Điều này đánh dấu sự kết thúc của khung.

Sau đó, có ba bit **intermission** hoặc **interframe space (IFS)**. Chúng ta có thể tranh luận liệu nó có phải là một phần của khung hay không, vì mỗi khung đều kết thúc với ít nhất ba bit IFS.

Vì vậy, nếu bạn chú ý, bạn sẽ thấy rằng mỗi khung dữ liệu kết thúc với 11 bit recessive liên tiếp: một bit phân tách xác nhận, bảy bit kết thúc khung, và ba bit IFS, tổng cộng 11 bit recessive.

Khi IFS kết thúc, bus lại trở về trạng thái nhàn rỗi và bất kỳ nút nào khác có thể bắt đầu truyền khung của nó trên bus.

Đây là định dạng khung của khung dữ liệu CAN chuẩn.

Tiếp theo, các phần này có thể được kết hợp và xem như các trường khác nhau tùy thuộc vào mục đích của chúng.

* **SOF**, **message ID** và **RTR** tạo thành **trường phân bổ (arbitration field)**.
* **Reserve bits** và **DLC** tạo thành **trường điều khiển (control field)**.
* **Data**, **CRC** và **ACK** tạo thành **trường dữ liệu (data field)**.
* **End of frame** và **IFS** tạo thành **trường kết thúc (end frame)**.

Khung CAN cũng có thể chứa thông tin bổ sung hoặc đặc biệt như **khung CAN-FD** hoặc **khung mở rộng**, nhưng chúng ta sẽ không đề cập đến trong bài giảng này.

Vậy là chúng ta đã học xong về định dạng khung của khung dữ liệu CAN.

Hy vọng bạn đã hiểu rõ về các trường và chức năng của chúng trong khung CAN.

Cảm ơn các bạn đã chú ý!

## Data and remote frame format with Excel tool

Giảng viên: Chào các bạn, trong bài giảng trước,

Tôi đã giải thích cho các bạn về định dạng khung dữ liệu của một khung dữ liệu chuẩn, đúng không?

Trong đó, các bạn đã được làm quen và hiểu về các trường khác nhau, phải không?

Các trường mà các bạn đã hiểu như sau:

Trong phần điều phối (arbitration), bạn có SOF (Start of Frame), mã nhận diện tin nhắn (Message ID), RTR (Remote Transmission Request),

và trong phần điều khiển (Control Field), bạn có hai trường dự trữ, đó là R1 và R0, cùng với mã chiều dài dữ liệu (DLC, Data Length Code).

Sau đó, bạn có trường dữ liệu (Data Field), tiếp theo là trường CRC (Cyclic Redundancy Check), trong đó có phần kiểm tra tổng (checksum), với 15 bit cho phần checksum và phần phân cách checksum (checksum delimiter).

Sau đó, bạn sẽ thấy trường xác nhận (Acknowledgement Field), trong đó có một bit cho vị trí xác nhận và phân cách xác nhận (acknowledgement delimiter),

và cuối cùng là trường kết thúc khung (End of Frame), trong đó có 7 bit cho kết thúc khung và 3 bit IFS (Interframe Space).

Chúng ta cũng đã thấy tất cả các khung dữ liệu hợp lệ và với 11 bit liên tiếp nhận được, đúng không?

Vậy là, khi các bạn đã hiểu được tất cả những điều này,

Tôi muốn các bạn thực sự cảm thấy thoải mái với nó,

hãy thuộc lòng định dạng khung này,

và các bạn nên làm quen với nó một cách rất kỹ lưỡng, đúng không?

Để làm được điều đó, tôi đã tạo ra một công cụ nhỏ như một hoạt hình,

không phải là hoạt hình thật sự mà là một công cụ chơi với Excel để giúp các bạn hiểu về định dạng khung dữ liệu, được chứ?

Với công cụ này, bạn có thể nhập các mã CAN khác nhau, mã tin nhắn của CAN,

và nhập các DLC khác nhau,

và các byte dữ liệu yêu cầu.

Ví dụ, ở đây, bạn có thể chọn 8 byte,

sau đó bạn phải nhập các byte dữ liệu 8 bit ở đây,

và xác định xem nó là khung dữ liệu (Data Frame) hay khung yêu cầu (Remote Frame).

Bạn nhập các giá trị này và khung dữ liệu sẽ được hiển thị ở đây, được chứ?

Vậy các trường khác nhau, mỗi bit sẽ có giá trị gì,

và các sóng mang (waveforms) sẽ trông như thế nào,

tất cả sẽ được thể hiện rõ ràng ở đây,

và ở đây, checksum cũng được tính toán chính xác.

Có một khái niệm gọi là "bit stuffing" trong CAN.

Vậy sau khi nhồi bit (stuffing), dòng bit sẽ như thế nào,

và bạn sẽ biết được cho một khung dữ liệu duy nhất có bao nhiêu bit.

Từ đó, bạn có thể tính toán được thời gian mà khung dữ liệu cần để di chuyển trên bus và các yếu tố khác tùy thuộc vào tốc độ truyền và các yếu tố khác, được chứ?

Tôi sẽ đính kèm file Excel này trong phần tài nguyên,

bạn hãy thử nghiệm với nó, được chứ?

Bạn có thể thay đổi mã CAN, ví dụ, tôi thay đổi mã CAN thành 5F1, được chứ?

Nếu bạn thay đổi mã đó, thì mã tin nhắn (Message ID) sẽ thay đổi và sóng mang cũng thay đổi.

Nếu đây là khung dữ liệu (Data Frame), thì đó là trường hợp bình thường.

Và nếu là khung yêu cầu (Remote Frame), thì tôi sẽ giải thích về phần khung yêu cầu.

Tôi không chú trọng nhiều vào khung yêu cầu trong các bài giảng của mình và chủ yếu tập trung vào khung dữ liệu, vì ngày nay, khung yêu cầu đã trở nên lỗi thời.

Tuy nhiên, vì đây là khóa học CAN đầy đủ,

bạn cũng nên có một cái nhìn cơ bản về khung yêu cầu, được chứ?

Khung yêu cầu có định dạng rất giống với khung dữ liệu.

Hai điểm khác biệt duy nhất là:

Đối với khung yêu cầu, bit RTR sẽ là 1.

Còn đối với khung dữ liệu, bit RTR sẽ là 0, được chứ?

Tôi đã cho bạn một mẹo ghi nhớ:

R cho Remote Frame,

và D cho Data Frame, D và R, đúng không?

Vì vậy, đối với khung yêu cầu, không có trường dữ liệu, được chứ?

Giữ nguyên mã CAN và giá trị DLC sẽ giống như khung dữ liệu mà khung yêu cầu này đang yêu cầu.

Chúng ta sẽ giữ lại mã CAN và DLC như nhau,

chỉ khác là không có trường dữ liệu trong khung yêu cầu.

Vậy khi tôi đổi từ khung dữ liệu sang khung yêu cầu,

thì chỉ có mỗi bit RTR thay đổi từ 0 thành 1.

Sau DLC, trực tiếp sẽ có checksum, vì khung yêu cầu không có dữ liệu, đúng không?

Khung yêu cầu là một khung đang yêu cầu dữ liệu từ một khung dữ liệu khác, được chứ?

Vì vậy, sau DLC, trực tiếp là checksum,

nhưng nếu tôi làm nó thành khung dữ liệu,

thì sẽ có trường dữ liệu và sau đó mới là checksum, được chứ?

Giá trị checksum sẽ thay đổi giữa khung dữ liệu và khung yêu cầu,

bởi vì đối với khung dữ liệu, ngay cả các byte dữ liệu cũng sẽ được tính vào để tính checksum.

Cách tính checksum là như thế nào?

Trong video trước, tôi chỉ nói qua về phương pháp BCS để tính checksum,

nhưng trong bài giảng sau, tôi sẽ giải thích chi tiết về phương pháp BCS này và cách tính checksum chính xác với một ví dụ cụ thể, được chứ?

Và sau đó bạn sẽ rất quen thuộc với checksum,

nhưng bây giờ, bạn hiểu sự khác biệt giữa khung dữ liệu và khung yêu cầu rồi.

Mục đích của khung yêu cầu là yêu cầu một khung dữ liệu,

và với khung yêu cầu, bit RTR sẽ thay đổi trong định dạng,

và trường dữ liệu sẽ không có trong khung yêu cầu.

Ngoài ra, theo định dạng của khung, mọi thứ khác đều giống hệt nhau.

Khi không có byte dữ liệu trong khung yêu cầu,

do không có dữ liệu nên DLC sẽ không phải là 0, được chứ?

Đó là một trong những sự hiểu nhầm lớn mà mọi người có.

DLC sẽ giống hệt như DLC của khung dữ liệu mà nó yêu cầu.

Nếu nó yêu cầu một khung dữ liệu 5F1,

và đó là khung yêu cầu, được chứ?

Nếu khung yêu cầu này yêu cầu một khung dữ liệu 5F1,

và khung dữ liệu 5F1 có 8 byte dữ liệu,

thì DLC của khung yêu cầu cũng phải là 8,

chỉ có điều trường dữ liệu sẽ không có, được chứ?

Đó là sự khác biệt. DLC sẽ giống nhau.

Vì lý do DLC này, một trong những lý do,

chúng ta không thể xác định chính xác DLC,

vì vậy chúng ta gặp phải sự xung đột và đó cũng là một trong những lý do khiến khung yêu cầu trở nên lỗi thời, được chứ?

Vì nếu khung yêu cầu tồn tại,

sẽ có câu hỏi về khung dữ liệu,

và khung dữ liệu có thể có nhiều DLC khác nhau,

nên khi yêu cầu khung dữ liệu từ khung yêu cầu,

chúng ta không biết chính xác DLC sẽ là bao nhiêu.

Đó là lý do tại sao Hội đồng tự động hóa CAN đã quyết định

rằng chúng ta sẽ không sử dụng khung yêu cầu nữa, được chứ?

Vậy dù thế nào đi nữa,

bạn chỉ cần hiểu rằng,

và bạn có thể thay đổi các byte dữ liệu.

Tôi sẽ thay đổi byte dữ liệu từ 2 sang 3,

và làm nó thành khung dữ liệu, được chứ?

Hãy tiếp tục thay đổi những giá trị này

và làm quen với tất cả các byte dữ liệu

và các trường khác nhau của khung dữ liệu

và cách nó hoạt động ở mức logic TTL

và cách nhồi bit, có bao nhiêu bit,

bạn hãy làm quen với các khái niệm này

và sử dụng sheet Excel này

và chơi với nó, được chứ?

Vậy là hết bài giảng này,

và trong bài giảng tiếp theo, chúng ta sẽ tìm hiểu

về định dạng khung mở rộng (Extended Frame).

Nó giống với khung chuẩn,

chỉ khác biệt sẽ xuất hiện trong phần điều phối.

Vậy trong phần điều phối,

sự khác biệt sẽ xảy ra như thế nào,

chúng ta sẽ xem xét trong bài giảng tiếp theo về định dạng khung mở rộng.

Được rồi, vậy là hết bài giảng này,

chúng ta đã hiểu về công cụ này,

bài giảng này giúp các bạn làm quen với định dạng khung,

và chúng ta cũng đã hiểu sự khác biệt giữa khung dữ liệu và khung yêu cầu,

cũng như lý do tại sao khung yêu cầu trở nên lỗi thời.

Ok? Hẹn gặp lại các bạn trong bài giảng tiếp theo.

## Extended Data Frame Format

Chào các bạn, chào mừng đến với bài giảng này. Chúng ta sẽ tìm hiểu về định dạng khung mở rộng (extended frame format). Chúng ta đã nắm vững định dạng khung dữ liệu chuẩn (standard data frame format), và giờ đây chúng ta sẽ thử tìm hiểu về định dạng khung dữ liệu mở rộng (extended data frame format).

Trước hết, để hiểu rõ kiến thức này, các bạn phải thực sự hiểu rõ những gì chúng ta sẽ xử lý. Chúng ta sẽ hiểu lý do tại sao khi hoạt động trên hiện trường, chúng ta lại cần đến định dạng khung dữ liệu mở rộng, mặc dù đã có định dạng khung dữ liệu chuẩn. Sau đó, chúng ta sẽ hiểu về các hành động và dữ liệu trong khung.

Chúng ta không đi vào toàn bộ bức tranh mà chỉ tập trung vào khung chuẩn. Dù có sự khác biệt, chúng ta sẽ hiểu các phần và sự kết nối giữa các phần này, từ khung chuẩn đến các phần giải thích. Tiếp theo là các bit bổ sung hoặc các trường bổ sung, chúng ta sẽ tìm hiểu chi tiết về những điều này. Chúng là những phần bổ sung, và chúng có nghĩa gì, v.v.

Được rồi, chúng ta bắt đầu nhé.

Tại sao hành động và dữ liệu có thể là một giao thức dựa trên thông điệp (message-based protocol)? Chúng ta đã hiểu rằng, đối với định dạng khung chuẩn, thông điệp là một phần của giao thức. Những thông điệp này được che đậy bởi các bộ điều khiển (controllers), và chúng có một số giới hạn, ví dụ như mỗi mạng có thể có tối đa 512 khung dữ liệu (frames). Tuy nhiên, trong lĩnh vực ô tô, số lượng dữ liệu cần chia sẻ giữa các hệ thống ngày càng tăng.

Vấn đề là, trong nhiều trường hợp, số lượng khung này là không đủ. Một số người đã tìm cách giải quyết vấn đề này bằng cách thêm vào nhiều mạng hơn. Tuy nhiên, giải pháp này không phải lúc nào cũng khả thi. Một giải pháp khác xuất hiện, đó chính là giao thức CAN (Controller Area Network), với sự xuất hiện của khung mở rộng. Trong trường hợp dữ liệu không đủ, thông điệp trong khung CAN được mở rộng từ 11 bit lên 29 bit.

Vậy là, chúng ta đã quen với cách phản ứng và xử lý khi khung dữ liệu mở rộng này được áp dụng. Khi khung mở rộng được sử dụng, tổng số khung có thể lên đến 512 triệu khung.

Nhưng vấn đề là, con số này không phải lúc nào cũng đủ cho các yêu cầu hiện tại. Vì vậy, đã có các đề xuất mở rộng thông điệp tới 29 bit, giúp giữ được tính tương thích ngược (backward compatibility). Do đó, khung dữ liệu chuẩn và khung mở rộng có thể cùng tồn tại trong một mạng mà không xung đột với nhau. Đây là lý do tại sao chúng ta gọi đây là "khung hỗn hợp" (hybrid frame).

Nếu bạn đang nghiên cứu về an ninh truyền thông, có thể bạn đã nghe về thuật ngữ "khung chuẩn" và "khung hỗn hợp". Khung chuẩn là khung dữ liệu CAN truyền thống, còn khung hỗn hợp bao gồm cả khung chuẩn và khung mở rộng.

Như vậy, cả hai loại khung này có thể tồn tại cùng nhau trong một mạng. Khi chúng ta quyết định định dạng của một khung dữ liệu, chúng ta phải nhớ rằng sẽ có sự kết hợp giữa các khung chuẩn và khung mở rộng. Đây là điều chúng ta sẽ tìm hiểu trong bài giảng tiếp theo.

Tất cả các khung dữ liệu này có sự khác biệt. Khi bạn hiểu về định dạng khung, bạn sẽ nhận thấy sự phân biệt này giữa các loại khung.

Do đó, ngoài khung chuẩn, khung mở rộng cũng tồn tại và để giữ được tính tương thích ngược, mạng sẽ sử dụng cả hai loại khung này.

Tiếp theo là việc phân tích các hành động và các bit trong khung. Khung mở rộng sẽ có các bit bổ sung như "Request to Send" (RTR), dùng để thay thế các yêu cầu điều khiển từ xa (remote transmission requests). Khi làm việc với khung mở rộng, bạn sẽ thấy một số điều chỉnh và thay đổi trong các bit đánh dấu, giúp nhận diện khung chuẩn hay khung mở rộng.

Các bước này sẽ được thực hiện như thế nào, ví dụ: khi bạn có một ID, nếu ID này được mở rộng, khung đó sẽ là khung mở rộng. Khi bạn có 29 bit ID, khung dữ liệu sẽ được phân chia thành các phần riêng biệt, với một phần dành cho ID cơ bản và phần còn lại cho các thông tin mở rộng.

Mỗi phần trong khung mở rộng sẽ thay thế các bit mặc định trong khung chuẩn, như các bit dành cho "remote request" (yêu cầu từ xa). Nếu khung có các bit này, thì nó được coi là một khung mở rộng và có khả năng sử dụng các chỉ dẫn mới cho việc kiểm tra mạng.

Lý do tại sao tất cả các điều này cần phải được thực hiện là để đảm bảo rằng khung mở rộng có thể tiếp tục hoạt động trong hệ thống mà không phá vỡ khả năng tương thích ngược với các khung chuẩn.

Cuối cùng, khung dữ liệu chuẩn và khung mở rộng có thể coexisted trong mạng, và sự lựa chọn của mỗi loại khung sẽ tùy thuộc vào yêu cầu cụ thể của hệ thống và ứng dụng.

Vậy là chúng ta đã kết thúc bài giảng này. Cảm ơn các bạn đã dành thời gian và hy vọng các bạn học được nhiều kiến thức bổ ích từ bài học này. Hẹn gặp lại các bạn trong những bài giảng sau!

# Khái niệm giao thức CAN

## Bit Segmentation

Giảng viên: Chào các bạn sinh viên.

Chào mừng các bạn đến với bài giảng về phân đoạn bit trong giao thức CAN.

Trong bài giảng này, chúng ta sẽ tìm hiểu về giám sát bit, phân đoạn bit trong giao thức CAN và các phần tử của nó, như phần đồng bộ (sync segment), phần lan truyền (propagation segment), phần pha 1 (phase 1 segment) và phần pha 2 (phase 2 segment), sau đó chúng ta sẽ trực quan hóa độ trễ lan truyền và những yếu tố liên quan.

Vậy chúng ta bắt đầu ngay nhé!

Trước tiên, chúng ta sẽ tìm hiểu về giám sát bit. Giám sát bit là gì?

Giám sát bit có nghĩa là khi các nút (nodes) giám sát bus để lấy giá trị bit hiện tại, tức là lấy mẫu giá trị điện áp trên bus và xác định giá trị của bit hiện tại. Quá trình này diễn ra cho mỗi bit và được gọi là giám sát bit.

Trên màn hình, các bạn có thể thấy có các mũi tên và hình vuông này. Mỗi hình vuông này đại diện cho một bit. Ví dụ, đây là bit đầu tiên, đây là bit thứ hai, thứ ba, thứ tư... đến bit thứ chín.

Các mũi tên này đại diện cho thời gian mà quá trình lấy mẫu bit diễn ra, tức là các nút sẽ giám sát bus vào thời điểm nào để lấy mẫu. Khi bus có giá trị logic 1, nút sẽ xác định giá trị là logic 1, và khi có giá trị logic 0, nó sẽ xác định là logic 0.

Điều quan trọng cần lưu ý ở đây là, ví dụ, bit thứ tám, thời gian của bit này kéo dài một khoảng như vậy. Và trong khoảng thời gian đó, việc lấy mẫu có thể xảy ra tại nhiều điểm khác nhau. Cụ thể, quá trình lấy mẫu có thể xảy ra tại điểm giữa của bit, hoặc ở gần cuối của bit. Việc xác định thời gian lấy mẫu sẽ ảnh hưởng đến cách các nút quyết định giá trị của bit đó.

Để hiểu rõ hơn về giám sát bit, chúng ta cần tìm hiểu về phân đoạn bit.

Phân đoạn bit trong giao thức CAN như thế nào?

Khi một nút truyền (Tx node) gửi một bit lên bus, nó sẽ giám sát lại bus ngay sau khi gửi để kiểm tra xem bit đó đã được đưa lên bus một cách chính xác hay chưa. Nếu nút truyền gửi một bit thống trị (dominant bit) lên bus, nó sẽ giám sát xem bit thống trị đó có đúng trên bus hay không. Tương tự, nếu nó gửi một bit phục hồi (recessive bit), nó cũng sẽ giám sát xem bit phục hồi đó có được gửi đúng hay không.

Mỗi khoảng thời gian của một bit, nút truyền sẽ gửi bit lên bus và sau đó sẽ giám sát lại bus để đảm bảo bit đó đã được truyền chính xác. Điều này sẽ xảy ra trong cùng một khoảng thời gian của bit.

Còn đối với nút nhận (Rx node), nó không cần phải gửi bit, vì nó chỉ nhận bit từ bus. Nó sẽ giám sát thời điểm mà giá trị trên bus được lấy mẫu.

Để hiểu rõ về việc giám sát bit, chúng ta cần hiểu về phân đoạn bit.

Phân đoạn bit là gì?

Nếu đây là một bit đơn của bus CAN, bit này được chia thành các phân đoạn khác nhau. Cụ thể, mỗi bit trong giao thức CAN được chia thành 4 phân đoạn, bao gồm phân đoạn đồng bộ (sync segment), phân đoạn lan truyền (propagation segment), phân đoạn pha 1 (phase segment 1), và phân đoạn pha 2 (phase segment 2). Điểm giao nhau giữa phân đoạn pha 1 và pha 2 là nơi việc lấy mẫu bit xảy ra, tức là nơi giám sát bit diễn ra.

Vậy mỗi phân đoạn có ý nghĩa gì?

Phân đoạn đồng bộ (sync segment) là phân đoạn dùng để đồng bộ các nút với nhau. Chúng ta biết rằng giao thức CAN là giao thức không đồng bộ, tức là không có tín hiệu đồng hồ riêng để đồng bộ các nút nhận (Rx nodes) với các nút truyền (Tx nodes). Do đó, chính các khung dữ liệu trong giao thức CAN sẽ đảm nhiệm vai trò đồng bộ các nút với nhau. Phân đoạn đồng bộ là phân đoạn nơi mà nút truyền sẽ đưa bit lên bus.

Phân đoạn lan truyền (propagation segment) được thêm vào để bù đắp cho độ trễ lan truyền tín hiệu trên bus. Giả sử một bus có chiều dài 40 mét và có hai nút A và B. Nếu một bit được đưa lên bus tại thời điểm t0 ở nút A, thì nút B sẽ không nhận được tín hiệu ngay lập tức. Tín hiệu cần một thời gian để lan truyền từ A đến B, và độ trễ này được gọi là độ trễ lan truyền.

Các phân đoạn pha 1 và pha 2 được sử dụng để xác định thời điểm lấy mẫu của bit trên bus. Tùy vào cách điều chỉnh các phân đoạn pha 1 và pha 2, chúng ta có thể thay đổi vị trí lấy mẫu bit sang trái hoặc sang phải trong thời gian bit.

Kết hợp các phân đoạn này lại, phân đoạn đồng bộ giúp các nút đồng bộ với nhau, phân đoạn lan truyền bù đắp cho độ trễ lan truyền, còn các phân đoạn pha 1 và pha 2 quyết định thời điểm lấy mẫu của bit.

Vậy độ trễ lan truyền là gì? Đó là khoảng thời gian giữa khi bit được đưa lên bus và khi nút nhận nhận được bit đó.

Cuối cùng, phân đoạn lan truyền giúp bù đắp độ trễ này, đảm bảo rằng các nút nhận sẽ lấy mẫu bit đúng thời điểm và giá trị đúng với bit mà nút truyền đã gửi lên bus.

Như vậy, tất cả bốn phân đoạn của một bit trong giao thức CAN – phân đoạn đồng bộ, phân đoạn lan truyền, phân đoạn pha 1 và pha 2 – đều đóng vai trò quan trọng trong việc đồng bộ và điều khiển quá trình truyền và nhận dữ liệu trên bus.

Hy vọng các bạn đã nắm được các khái niệm cơ bản về phân đoạn bit và giám sát bit trong giao thức CAN.

## Bus Arbitration

Giảng viên: Chào các bạn sinh viên!

Chào mừng các bạn đến với bài giảng về phân giải quyền truy cập bus trong giao thức CAN.

Trong bài giảng này, chúng ta sẽ cùng tìm hiểu:

* Quy trình phân giải quyền truy cập bus là gì?
* Phân giải quyền truy cập bus hoạt động như thế nào?
* Chúng ta sẽ xem một ví dụ minh họa về phân giải quyền truy cập bus thông qua phần mềm Excel animation mà tôi đã tạo ra.

Vậy phân giải quyền truy cập bus là gì?

Khi nhiều nút (nodes) trong mạng CAN muốn truyền một frame (khung dữ liệu) cùng một lúc trên bus, thì xung đột này sẽ được giải quyết thông qua một quá trình gọi là phân giải quyền truy cập bus.

Hãy nhớ rằng, trong một mạng CAN, vào một thời điểm nhất định, chỉ có một nút CAN có thể truyền khung dữ liệu, và các nút còn lại phải là nút nhận (receiver). Vì vậy, chỉ có thể truyền một frame duy nhất trên bus, nhưng nếu có nhiều nút, ví dụ như có 10 nút trên mạng CAN và nút A cùng nút C muốn truyền khung dữ liệu trên bus đồng thời, thì làm sao để quyết định nút nào sẽ được phép truyền khung dữ liệu?

Để giải quyết xung đột này, chúng ta sử dụng một quy trình gọi là phân giải quyền truy cập bus. Trong định dạng khung CAN, các trường như bắt đầu khung (start of frame), mã định danh (message ID) và bit RTR tham gia vào quá trình phân giải quyền truy cập bus.

Vì các trường này tham gia vào phân giải quyền truy cập bus, chúng được gọi chung là trường phân giải (arbitration field). Nút nào thắng trong quá trình phân giải quyền truy cập bus sẽ trở thành nút truyền (transmitter node) và được phép gửi khung dữ liệu của mình, trong khi các nút khác sẽ trở thành nút nhận.

Các nút thua trong quá trình phân giải sẽ thử truyền lại khung dữ liệu của mình khi bus trở nên rỗi. Như vậy, không có mất mát khung dữ liệu.

Khi hai nút cố gắng gửi các bit khác nhau lên bus, một nút gửi bit "recessive" (bit thấp) và nút khác gửi bit "dominant" (bit cao) cùng lúc, bus sẽ lấy giá trị bit "dominant" và bỏ qua giá trị bit "recessive". Đây chính là sự khác biệt giữa "dominant" và "recessive". Khi hai bit xuất hiện, bus sẽ lấy bit "dominant" làm giá trị chính.

Quy trình phân giải quyền truy cập bus cần thiết khi nhiều nút cố gắng truyền dữ liệu cùng lúc. Chỉ một nút truyền được phép truyền tại một thời điểm. Các nút sẽ chờ cho đến khi bus trở nên rỗi, và các nút có khung dữ liệu quan trọng hơn sẽ tiếp tục truyền dữ liệu.

Ví dụ:

Đây là một mạng CAN. Giả sử có ba nút đang cố gắng gửi dữ liệu cùng lúc. Một nút muốn gửi thông điệp quan trọng, một nút gửi thông điệp khẩn cấp, và nút còn lại cũng muốn gửi. Nếu một nút có quyền ưu tiên trong phân giải, nó sẽ thắng và được phép truyền. Trong khi đó, các nút khác sẽ phải trở thành nút nhận và chờ cơ hội tiếp theo.

Như vậy, bạn đã hiểu về phân giải quyền truy cập bus rồi đúng không? Đây là cơ chế giúp giải quyết các xung đột giữa các nút khi cố gắng trở thành nút truyền trong mạng cùng lúc.

Quá trình phân giải quyền truy cập bus hoạt động như thế nào?

Khi bus không có dữ liệu truyền tải, đó là khi bus "idle" (rỗi), một bit khởi đầu (SOF - Start of Frame) sẽ xuất hiện trên bus để báo hiệu rằng một khung dữ liệu mới sẽ bắt đầu. Tiếp theo là mã định danh (Message ID) dài 11 bit, và một bit RTR (Remote Transmission Request) sẽ cho biết đây là khung dữ liệu hay khung yêu cầu dữ liệu từ xa.

Tất cả các trường này cùng tham gia vào quá trình phân giải quyền truy cập bus. Nếu có ba nút đang cạnh tranh để truyền khung dữ liệu, mỗi nút sẽ gửi khung của mình lên bus và bắt đầu so sánh các giá trị bit trong quá trình phân giải. Các nút sẽ tiếp tục gửi dữ liệu cho đến khi có sự khác biệt giữa các giá trị bit mà chúng gửi. Nút nào có giá trị bit "dominant" sẽ thắng và tiếp tục truyền.

Trong một ví dụ khác, khi hai nút cạnh tranh để truyền dữ liệu, nếu một nút gửi bit "recessive" và các nút còn lại gửi bit "dominant", bus sẽ chọn bit "dominant" và bỏ qua bit "recessive". Do đó, nút gửi bit "recessive" sẽ thua và chuyển sang chế độ nhận (RX mode), trong khi các nút gửi bit "dominant" sẽ tiếp tục tranh giành quyền truyền.

Kết thúc quá trình phân giải, chỉ một nút duy nhất sẽ có quyền truyền dữ liệu. Các nút còn lại sẽ chờ cho đến khi bus trở nên rỗi một lần nữa.

Tóm lại, các quy tắc chính trong phân giải quyền truy cập bus là:

1. Bus được coi là "rỗi" sau khi một khung dữ liệu đã được truyền xong cùng với trường thời gian trống (IFS).
2. Nút truyền có mã định danh thấp nhất sẽ thắng trong phân giải quyền truy cập, vì mã định danh thấp nhất sẽ có bit "dominant" (bit 0).
3. Các nút thua sẽ bắt đầu phân giải lại khi bus trở nên rỗi.

Như vậy, bạn đã hiểu rõ cơ chế phân giải quyền truy cập bus trong giao thức CAN. Bây giờ, bạn có thể thử nghiệm và kiểm tra các khung dữ liệu khác nhau trong môi trường mô phỏng để hiểu rõ hơn cách mà phân giải quyền truy cập hoạt động.

Chúc các bạn học tốt!

## Bit Stuffing.

Chào các bạn, trong buổi học hôm nay, chúng ta sẽ tìm hiểu về một số khái niệm quan trọng.

Lịch trình của buổi học này sẽ bắt đầu với việc các bạn hiểu thế nào là một cấu hình nhân sự tốt. Sau đó, chúng ta sẽ cùng nhau tìm hiểu về cấu hình nhân sự.

Tiếp theo, chúng ta sẽ xem xét cấu hình nhân sự trong các khung dữ liệu, trong các khung truyền dữ liệu, và xác định phần nào hoặc trường nào trong khung dữ liệu cần thiết phải cấu hình nhân sự.

Chúng ta cũng sẽ hiểu được ý nghĩa của việc cấu hình nhân sự đối với Cơ quan Quản lý An toàn Vận tải, cũng như các ứng dụng của nó trong danh sách các máy bay của chúng ta.

Vậy, chúng ta sẽ bắt đầu thôi.

Được rồi, đầu tiên, chúng ta cần hiểu nhu cầu của nhân viên.

Như các bạn đã biết, giao thức CAN là một giao thức đồng bộ.

Vậy đồng bộ là gì?

Đồng bộ có nghĩa là không có tín hiệu đồng hồ riêng biệt để thông báo trạng thái bit hiện tại trên bus, hay không có tín hiệu đồng hồ để đồng bộ hóa giữa các thiết bị truyền (oryx) và mạng DSN.

Vì vậy, giao thức CAN kết nối tất cả các nút lại với nhau.

Nút nhận dữ liệu ban đầu sẽ được đồng bộ với nút truyền, và cạnh xuống của SCAF sẽ đồng bộ hóa với nút nhận.

Thông qua tín hiệu đồng hồ nội bộ, nút nhận có thể theo dõi các bit và đồng bộ hóa với mỗi cạnh của tín hiệu từ nút phát.

Nhưng tôi biết điều này có thể hơi khó hiểu, để tôi giải thích thêm.

Hãy xem biểu đồ dưới đây.

Trong biểu đồ này, bạn có thể thấy có bao nhiêu nút.

Có một nút, sau đó là các nút thứ ba, thứ tư, thứ năm, thứ sáu, thứ bảy, thứ tám.

Vì vậy, mặc dù các bit này không được ghi rõ, bạn có thể nhận biết rằng mỗi bit có một sự thay đổi nhỏ, vì vậy bạn biết rằng giá trị của bit đã thay đổi.

Hãy nhìn vào biểu đồ tiếp theo.

Bạn thấy đó, đây là lần ra giá thầu thứ hai.

Nhưng sau đó, trạng thái này tiếp tục không thay đổi cho đến khi có sự chuyển đổi.

Khi sự chuyển đổi này xảy ra, chúng ta có trạng thái 01010, với sự thay đổi liên tục của các bit, như các bạn thấy.

Tuy nhiên, khi không có sự thay đổi trong khoảng thời gian dài, sẽ rất khó để biết có bao nhiêu bit đang có trên bus.

Vậy điều gì sẽ xảy ra khi có một xung động, đúng không?

Vì vậy, nút nhận sẽ đồng bộ lại mỗi khi có một cạnh xuất hiện, nó biết rằng đó là một sự thay đổi trạng thái, điều này có nghĩa là nó biết rằng đây là sự bắt đầu của một năm tồi tệ.

Vì vậy, nút nhận sẽ lại đồng bộ lại đồng hồ nội bộ của mình, đặt lại giá trị ban đầu của nó về số không, rồi lại bắt đầu đếm lại từ đầu.

Vậy, đồng hồ sẽ bắt đầu "tích tắc", đúng không?

Cứ mỗi chu kỳ đồng hồ sẽ đếm tiếp và dựa trên sự thay đổi trạng thái.

Nó sẽ tính toán khi nào có sự thay đổi xảy ra và làm sao để tiếp tục đồng bộ lại.

Khi không có sự thay đổi, chúng ta sẽ phải sử dụng đồng hồ nội bộ để tính toán lại, để đảm bảo rằng dữ liệu được đồng bộ đúng với các bit tiếp theo.

Tuy nhiên, có những sai số nhỏ xảy ra trong mỗi chu kỳ đồng hồ do các yếu tố vật lý như các vấn đề về phần cứng hoặc sự khác biệt trong việc truyền tải tín hiệu.

Vì vậy, có thể xảy ra sai số, gọi là "delta error", khi không có sự thay đổi rõ rệt trong quá trình truyền tải.

Để khắc phục điều này, nút nhận sẽ phải điều chỉnh đồng hồ của mình mỗi khi có sự thay đổi mới để giảm thiểu sai số tích lũy.

Quá trình này được gọi là việc "tái đồng bộ hóa" (re-synchronization) của nút nhận.

Khi có quá nhiều bit liên tiếp có cùng giá trị (ví dụ: 5 bit liên tiếp có cùng một trạng thái), một bit giả (hay còn gọi là bit lấp đầy) sẽ được thêm vào để đảm bảo rằng đồng hồ của nút nhận vẫn đồng bộ.

Điều này là một phần trong việc bảo vệ tính chính xác của dữ liệu khi truyền trên bus.

Trong quá trình này, các bit không phải lúc nào cũng được tính là một phần của dữ liệu, mà thay vào đó chỉ là các bit điều chỉnh nhằm duy trì đồng bộ hóa giữa các nút trong mạng CAN.

Khi việc đồng bộ hóa gặp sự cố, hoặc khi có sự thay đổi không mong muốn trong trạng thái bit, các bit lấp đầy sẽ được chèn vào để tái thiết lập trạng thái ban đầu của bus và các nút sẽ tiếp tục hoạt động bình thường.

## CAN CRC Computation (BCH Method)

**Giảng viên:** Chào các bạn,

Trong bài giảng này, chúng ta sẽ không sử dụng bất kỳ bài thuyết trình PowerPoint nào.

Nhưng trong bài giảng này, chúng ta sẽ tập trung vào **trường CRC** hoặc **checksum** của một frame CAN.

Cách tính toán checksum này, đó là điều chúng ta sẽ tìm hiểu ở đây.

Trong giao thức CAN, trường CRC được tính toán bởi **CAN Controller** (bộ điều khiển CAN) thông qua một phương pháp gọi là phương pháp **BCH**.

Bạn đã thấy bảng tính Excel này trước đây khi bạn cố gắng hiểu định dạng frame CAN.

Vì vậy, đối với **CAN ID** 7F2, **DLC** là 1, tôi đã chọn giá trị DLC tối thiểu mà không phải là 0, vì DLC tối thiểu là 0. Nhưng tôi đã chọn DLC là 1 để trường dữ liệu sẽ nhỏ, giúp chúng ta có thể tính toán CRC nhanh chóng với một ví dụ. Và trường dữ liệu, tôi đã chọn là **AA**.

Nếu bạn sử dụng dữ liệu này, dạng sóng sẽ trông như thế này và các trường CAN, các khu vực frame CAN và các bit sẽ như sau.

Đúng vậy, bit đầu tiên là 0 **luôn luôn**.

Và **Message ID** như chúng ta biết là **7F2**.

Số **7** là **0111** trong hệ nhị phân, **F** là **1111** và **2** là **0010**, đúng không?

Vậy **Message ID** là 11 bit, vì đây là định dạng frame chuẩn và vì đây là **Data Frame**, bit **RTR** là 0, tức là **dominant** (thống trị), sau đó là các bit **reserved** và **DLC**. Dữ liệu là **AA**, giá trị nhị phân của AA là **1010**, tương ứng với **A** và **1010** nữa là **A**.

Với những dữ liệu này, chúng ta cần tính toán CRC.

Để tính CRC, dữ liệu đầu vào bắt đầu từ **start of frame** (SOF) cho đến hết **data field**, đoạn dữ liệu này được lấy ra.

Và đối với dữ liệu này, chúng ta thực hiện **bit stuffing**.

Vậy **bit stuffing** là gì? Đó là việc thêm bit vào để đảm bảo rằng không có quá 5 bit giống nhau liên tiếp.

Vì vậy, bit stuffing sẽ được thực hiện và sau khi thực hiện bit stuffing, chúng ta sẽ có chuỗi dữ liệu. Sau đó, chúng ta sẽ sử dụng một **polynomial** để tính toán checksum bằng phương pháp BCH.

Sau khi có checksum, chúng ta sẽ thêm nó vào dữ liệu này.

Cũng cần lưu ý rằng đối với checksum, chúng ta cũng thực hiện bit stuffing như đã đề cập. Bit stuffing sẽ được thực hiện từ phần **start frame** cho đến **end of checksum**, nơi mà các bit stuffing vẫn tiếp tục.

Chúng ta sẽ thấy một ví dụ về phương pháp BCH này với một frame cụ thể. Bây giờ, hãy xem xét dữ liệu này để xác định liệu có cần bit stuffing hay không.

Nhìn vào đây, bạn có thể thấy rằng có 5 bit **1** liên tiếp. Vì vậy, một bit **0** sẽ được thêm vào giữa những bit **1** này. Chỉ cần thêm một bit stuffing tại đây thôi.

Ngoài ra, không có các chuỗi 5 bit **0** hay **1** liên tiếp ở những chỗ khác ngoài nơi này.

Vậy hãy lấy dữ liệu này và thực hiện các phép tính, tôi đã làm sẵn các phép tính cho bạn. Đây là bắt đầu frame và 5 bit đầu của **Message ID**. Sau đó, một bit stuffing sẽ được thêm vào giữa, và sau đó là các **Message ID** còn lại, bit **RTR**, các bit **reserved**, **DLC**, và 8 bit dữ liệu **AA**.

Bước đầu tiên của phương pháp BCH đã được thực hiện.

Đây là **polynomial** (đa thức) chuẩn được sử dụng trong phương pháp BCH. Đây là đa thức được chọn sau rất nhiều nghiên cứu cho giao thức CAN: **1100010110011001**.

Đây là một đa thức chuẩn hóa, luôn luôn là đa thức này cho giao thức CAN.

Chúng ta thấy rằng đây là một đa thức có độ dài **16 bit**.

Độ dài của đa thức luôn luôn lớn hơn độ dài checksum (vì checksum của CRC có độ dài 15 bit). Vì vậy, đa thức phải có độ dài 16 bit.

Cách tính toán là bạn lấy dữ liệu và thực hiện **stuffing**. Sau khi thực hiện stuffing, bạn chèn **bit stuffing** vào và sau đó thêm vào đủ số lượng **0** sao cho số bit bằng với kích thước của CRC.

Checksum của chúng ta có độ dài 15 bit. Vì vậy, bạn sẽ chèn 15 bit **0** vào dòng dữ liệu đầu vào.

Sau khi chèn 15 bit **0** vào, toàn bộ dữ liệu sẽ trở thành chuỗi bit đầu vào mà chúng ta sẽ tính toán checksum.

Làm thế nào để tính toán checksum? Bạn phải ghi nhớ bảng **XOR** cho 2 bit.

Bảng XOR cho 2 bit như sau:

* 0 XOR 0 = 0
* 0 XOR 1 = 1
* 1 XOR 0 = 1
* 1 XOR 1 = 0

XOR có nghĩa là **exclusive OR** (hoặc duy nhất), có nghĩa là chỉ có một trong hai đầu vào là **1**, kết quả mới là **1**. Nếu cả hai đầu vào đều giống nhau, kết quả sẽ là **0**.

Sau khi thực hiện XOR cho tất cả các bit, bạn sẽ thu được kết quả.

Bây giờ, làm thế nào để bắt đầu với phương pháp BCH? Bạn phải bắt đầu với bit **MSB** (Most Significant Bit) không bằng 0. Nếu **SOF** (Start of Frame) là 0, bạn sẽ bắt đầu từ bit đầu tiên không phải là 0 của **Message ID**.

Trong trường hợp này, **Message ID** là **7F2**, vậy bạn bắt đầu từ bit đầu tiên của **7F2**. Còn nếu **Message ID** là **2F2**, bạn sẽ bắt đầu từ bit của **2F2**.

Vì vậy, khi bắt đầu với **7F2**, bạn sẽ bắt đầu từ bit đầu tiên.

Sau đó, bạn sẽ sao chép đa thức này và XOR với dữ liệu.

Các bước tiếp theo là tiếp tục XOR với đa thức cho đến khi bạn đạt được phần dư cuối cùng. Phần dư chính là checksum.

Và kết quả cuối cùng bạn có sẽ là **checksum 15 bit** cho dữ liệu này.

Sau khi có checksum, bạn thêm nó vào dữ liệu của bạn, sau đó có thể thực hiện thêm bit stuffing nếu cần thiết.

Và cuối cùng, bạn sẽ có **frame** hoàn chỉnh, bao gồm cả checksum.

**Kết luận:**

Hy vọng bạn đã hiểu cách tính toán checksum trong giao thức CAN bằng phương pháp BCH. Bạn có thể kiểm tra lại các bước trong bảng tính Excel mà tôi đã cung cấp trong phần Tài nguyên. Hãy thử tính toán thủ công và xem bạn có nhận được kết quả giống như kết quả tôi đã trình bày không.

**Chúc các bạn học tốt, tạm biệt!**

# CAN Xử lý lỗi

## Types of CAN errors

Giảng viên: Chào các bạn sinh viên.

Chào mừng các bạn đến với bài giảng về lỗi CAN.

Trong bài học này, chúng ta sẽ tìm hiểu về các điểm sau:

* Lỗi CAN là gì, chúng ta sẽ hiểu về điều này trước tiên.
* Sau đó, chúng ta sẽ tìm hiểu về các phản ứng khi một lỗi CAN xảy ra.
* Các phản ứng này sẽ được thể hiện như thế nào trên bus và các nút trong hệ thống.
* Các loại lỗi CAN, có 5 loại lỗi CAN khác nhau.
* Chúng ta sẽ tìm hiểu chi tiết về từng loại lỗi này.
* Cũng như các lỗi truyền (Tx errors) và lỗi nhận (Rx errors), chúng ta sẽ hiểu cách phân loại các lỗi này.
* Cuối cùng, chúng ta sẽ tìm hiểu về các tình huống khác nhau liên quan đến lỗi CAN, và nếu có bất kỳ ngoại lệ nào đối với quy tắc lỗi, chúng ta sẽ hiểu rõ về điều đó.

Hãy bắt đầu ngay thôi.

**Lỗi CAN là gì?**

Trước khi hiểu rõ về lỗi CAN, chúng ta cần hiểu về mục đích của giao thức CAN.

Khi bạn học về giao thức CAN, tôi đã nói điều này nhiều lần. Mục đích của giao thức CAN là để truyền tải dữ liệu từ một nút này đến các nút khác trong một mạng.

Trong lĩnh vực ô tô, mỗi nút là một ECU (Electronic Control Unit), đúng không? Vì vậy, CAN được sử dụng để truyền thông giữa các ECU. Dữ liệu cần truyền sẽ được đóng gói trong các trường dữ liệu và sau đó truyền đi dưới dạng bit, từng bit một.

Đây là cách truyền thông nối tiếp (serial communication) hoạt động. Trong định dạng khung CAN, chúng ta có dữ liệu cần truyền, và để làm điều đó, chúng ta sẽ thêm vào các trường như ID tin nhắn, DLC (Data Length Code), checksum để kiểm tra, và tất cả các trường liên quan đến khung này, rồi tạo thành một khung.

Sau đó, khung này sẽ được truyền đi từ SOF (Start of Frame) cho đến bit cuối cùng của khung dữ liệu, kết thúc bằng End of Frame.

Tất cả các nút nhận (Rx nodes) phải nhận đúng dữ liệu mà nút truyền (Tx node) mong muốn truyền đi. Dữ liệu này phải chính xác và hợp lệ, và khung dữ liệu phải được nhận nguyên vẹn tại nút nhận.

Tuy nhiên, điều này không phải lúc nào cũng xảy ra vì có thể có các vấn đề như nhiễu, hoặc sự cố phần cứng có thể làm gián đoạn quá trình truyền tải dữ liệu. Khi dữ liệu không thể truyền đạt chính xác, đó được gọi là lỗi CAN.

Lỗi CAN có thể xảy ra ở bất kỳ đâu trong quá trình truyền tải: có thể ở nút truyền (Tx), trên bus CAN, hoặc ở nút nhận (Rx). Khi một lỗi CAN xảy ra, giao thức CAN phải đủ mạnh mẽ để nhận diện lỗi đó và thực hiện các biện pháp khắc phục, đồng thời phản ứng đúng với lỗi.

Điều này định nghĩa cách xử lý lỗi trong giao thức CAN.

**Phản ứng với lỗi CAN**

Khi một nút (có thể là nút truyền hoặc nút nhận) phát hiện ra một lỗi CAN, có nghĩa là khung dữ liệu đang truyền có vấn đề hoặc bị lỗi.

Khi lỗi được phát hiện, nút đó (dù là nút truyền hay nút nhận) sẽ ngay lập tức gửi một cờ lỗi (error flag) lên bus CAN. Cờ lỗi này được thiết kế sao cho nó sẽ "hủy bỏ" khung dữ liệu hiện tại đang được truyền trên bus.

Ví dụ, nếu một khung dữ liệu đang được truyền và trong quá trình truyền, một lỗi CAN được phát hiện, ngay lập tức một cờ lỗi sẽ được gửi đi. Cờ lỗi này có chức năng ghi đè lên khung dữ liệu, làm cho nó không còn giá trị nữa.

Khi cờ lỗi được gửi, tất cả các nút khác trong mạng cũng có thể gửi cờ lỗi của riêng mình. Kết hợp tất cả các cờ lỗi này sẽ tạo thành một "khung lỗi" trên bus. Khung lỗi này sẽ làm cho khung dữ liệu bị hủy và không còn hợp lệ nữa. Sau khi khung lỗi được truyền, khung dữ liệu cần được gửi lại từ đầu.

**Các loại lỗi CAN**

Có 5 loại lỗi CAN mà chúng ta sẽ tìm hiểu trong bài giảng này. Các lỗi này có thể được phân loại theo cách mà nút phát hiện ra lỗi. Nếu lỗi được phát hiện ở nút truyền (Tx), đó là lỗi truyền (Tx error). Nếu lỗi được phát hiện ở nút nhận (Rx), đó là lỗi nhận (Rx error).

Mỗi lỗi này có thể xảy ra tại các điểm khác nhau trong hệ thống, bao gồm nút truyền, bus CAN, và nút nhận. Giao thức CAN cần phải có cơ chế xử lý lỗi để nhận diện và phản ứng với các lỗi này đúng cách.

**Đếm lỗi và xử lý lỗi**

Khi một lỗi xảy ra, số lượng lỗi sẽ được ghi nhận trong các bộ đếm lỗi (Error Counters). Có hai bộ đếm lỗi chính: **Transmit Error Counter (TEC)** và **Receive Error Counter (REC)**.

* **TEC (Transmit Error Counter)**: Đếm số lỗi xảy ra ở nút truyền.
* **REC (Receive Error Counter)**: Đếm số lỗi xảy ra ở nút nhận.

Các bộ đếm này giúp theo dõi các lỗi tạm thời và vĩnh viễn. Khi các lỗi xảy ra, giá trị của TEC và REC sẽ thay đổi, giúp hệ thống phân biệt được lỗi tạm thời và lỗi vĩnh viễn.

Trong các bài học sau, chúng ta sẽ đi sâu vào việc xử lý trạng thái lỗi của CAN, và cách các giá trị TEC và REC ảnh hưởng đến quá trình này.

Tóm lại, khi một lỗi CAN xảy ra, phản ứng là gửi cờ lỗi và tạo ra khung lỗi để hủy bỏ dữ liệu không hợp lệ. Các bộ đếm lỗi TEC và REC sẽ giúp theo dõi các lỗi này, và hệ thống sẽ có các biện pháp để xử lý chúng một cách hiệu quả.

Sau khi khung lỗi được truyền trên bus CAN, máy phát có thể tiếp tục gửi lại cùng một khung dữ liệu trên bus. Đây là phản ứng với lỗi CAN.

Vậy là bây giờ chúng ta đã hiểu lỗi CAN là gì và các nút mạng phản ứng thế nào với lỗi CAN, đúng không? Tiếp theo, chúng ta sẽ tìm hiểu các loại lỗi CAN khác nhau. Lỗi CAN có thể được chia thành nhiều loại, cụ thể là lỗi bit, lỗi xác nhận, lỗi CRC, lỗi đệm (stuff error) và lỗi định dạng (form error).

Chúng ta đã biết rằng các lỗi CAN có thể được phân loại thành lỗi truyền (Tx errors) và lỗi nhận (Rx errors). Hai lỗi đầu tiên, lỗi bit và lỗi xác nhận, được gọi là lỗi truyền (Tx errors), bởi vì chỉ có máy phát mới có thể nhận diện các lỗi này. Các lỗi còn lại như lỗi CRC, lỗi đệm và lỗi định dạng được gọi là lỗi nhận (Rx errors), vì chỉ có máy nhận mới có thể nhận diện các lỗi này trên bus CAN.

Do đó, lỗi truyền (Tx errors) và lỗi nhận (Rx errors) sẽ có ảnh hưởng khác nhau đến các bộ đếm TEC (Transmission Error Counter) và REC (Reception Error Counter). Lỗi bit và lỗi xác nhận sẽ làm tăng giá trị của bộ đếm TEC, trong khi lỗi CRC, lỗi đệm và lỗi định dạng sẽ làm tăng giá trị của bộ đếm REC.

Bây giờ, chúng ta sẽ tìm hiểu chi tiết về từng loại lỗi này, xem chúng là gì, nguyên nhân gây ra chúng và cách nhận diện chúng.

### Lỗi Bit (Bit Error)

Lỗi bit, hay còn gọi là lỗi giám sát bit, là một lỗi xảy ra khi giá trị của bit mà máy phát cố gắng truyền trên bus không trùng khớp với giá trị mà nó giám sát trên bus.

Trong một khung CAN, mỗi bit được chia thành bốn phần: đoạn đồng bộ (synchronization segment), đoạn truyền (propagation segment), đoạn pha 1 (phase segment 1) và đoạn pha 2 (phase segment 2). Quá trình giám sát bit diễn ra ở giao điểm giữa pha 1 và pha 2 của mỗi bit.

Trong đoạn đồng bộ, máy phát sẽ đưa giá trị bit lên bus và sau đó giám sát bit đó tại điểm lấy mẫu trong đoạn pha 1 và pha 2. Nếu máy phát đặt một bit lên bus, nó sẽ giám sát xem giá trị trên bus có khớp với giá trị mà nó đặt hay không. Nếu không khớp, đó là một lỗi bit.

Ví dụ, nếu máy phát cố gắng truyền một bit "dominant" (0), nhưng máy nhận thấy bit trên bus là "recessive" (1), thì đó là một lỗi bit. Điều này cũng áp dụng khi máy phát truyền bit "recessive" (1) mà máy nhận thấy trên bus là "dominant" (0).

Lỗi bit sẽ chỉ được nhận diện bởi máy phát, vì chỉ máy phát mới biết chính xác giá trị mà nó đã cố gắng truyền. Khi lỗi này xảy ra, máy phát sẽ gửi một "frame lỗi" (error frame) để thông báo lỗi.

#### Các Trường Hợp Đặc Biệt Của Lỗi Bit

1. **Trong trường hợp phân xử (arbitration)**: Nếu máy phát cố gắng truyền một bit "recessive" (1) nhưng nhận thấy một bit "dominant" (0), thì đây không phải là lỗi bit, mà là kết quả của việc mất phân xử. Khi đó, có thể có một máy phát khác đã thắng phân xử và truyền một bit "dominant".
2. **Trong trường hợp xác nhận (acknowledgement)**: Khi máy phát truyền một bit "recessive" và nhận thấy một bit "dominant" trong khoảng thời gian xác nhận, điều này là bình thường và không phải là lỗi bit, vì đây là cách xác nhận rằng máy nhận đã nhận dữ liệu thành công.

### Lỗi Xác Nhận (Acknowledgement Error)

Lỗi xác nhận xảy ra khi máy nhận không phản hồi với bit xác nhận đúng trong quá trình xác nhận. Nếu máy phát không nhận được phản hồi từ máy nhận sau khi truyền dữ liệu, nó sẽ biết rằng có lỗi trong quá trình truyền.

### Lỗi CRC (Cyclic Redundancy Check)

Lỗi CRC xảy ra khi có sự không khớp giữa giá trị CRC được tính toán và giá trị CRC thực sự nhận được từ dữ liệu. Lỗi này thường chỉ ra rằng có lỗi trong quá trình truyền dữ liệu và dữ liệu bị hỏng.

### Lỗi Đệm (Stuff Error)

Lỗi đệm xảy ra khi có số lượng bit liên tiếp giống nhau vượt quá giới hạn cho phép trong khung CAN. Ví dụ, nếu có sáu bit "recessive" liên tiếp trong dữ liệu mà không có sự thay đổi, thì đó là một lỗi đệm.

### Lỗi Định Dạng (Form Error)

Lỗi định dạng xảy ra khi khung CAN không tuân theo cấu trúc hoặc định dạng chuẩn. Điều này có thể liên quan đến việc các trường dữ liệu không đúng định dạng hoặc không tuân thủ các quy tắc về các trường trong khung.

### Tóm Tắt Các Lỗi

* **Lỗi truyền (Tx errors)**: Lỗi bit và lỗi xác nhận, ảnh hưởng đến bộ đếm TEC.
* **Lỗi nhận (Rx errors)**: Lỗi CRC, lỗi đệm và lỗi định dạng, ảnh hưởng đến bộ đếm REC.

Khi máy phát gặp lỗi, nó sẽ gửi một khung lỗi (error frame) để thông báo. Máy nhận sẽ có thể nhận diện lỗi nhận nhưng không thể nhận diện lỗi truyền, và ngược lại.

Hiểu rõ các lỗi này sẽ giúp chúng ta có thể phát hiện và khắc phục lỗi trong quá trình truyền thông qua CAN bus.

Vì đây là vi phạm một định dạng cố định, đó là 11 bit liên tiếp, nên có một ngoại lệ. Ngoại lệ thứ ba là ngoại lệ liên quan đến lỗi bit, nhưng không hoàn toàn là một ngoại lệ cho lỗi CAN. Chúng ta có một tên gọi riêng cho lỗi này, được hiểu rõ hơn qua các tình huống cụ thể.

Vậy nên, đây là lời giải thích về lỗi bit, các trường hợp sử dụng, và các ngoại lệ liên quan đến lỗi bit.

Sau khi tìm hiểu về lỗi bit, chúng ta sẽ tiếp tục với lỗi xác nhận (acknowledgement error). Vì cơ chế của lỗi xác nhận và lỗi CRC tương tự nhau, một cái là lỗi của người phát (transmitter), còn một cái là lỗi của người nhận (receiver). Vì vậy, chúng ta sẽ cố gắng hiểu hai loại lỗi này cùng nhau.

**Lỗi xác nhận (Acknowledgement Error):**

Khi nút phát (transmitter) giám sát một bit recessive trong khoảng thời gian xác nhận, điều này có nghĩa là nó không nhận được xác nhận từ nút nhận. Đây chính là lỗi xác nhận.

**Lỗi CRC:**

Lỗi CRC xảy ra khi nút nhận (receiver) không gửi xác nhận, nhưng vẫn giám sát một bit dominant trong khoảng thời gian xác nhận, thì đó là lỗi CRC. Vậy trước khi hiểu lỗi này, chúng ta cần nhắc lại cơ chế xác nhận trong giao thức CAN.

**Cơ chế xác nhận:**

Cách xác nhận được thực hiện trong giao thức CAN có liên quan mật thiết đến việc tính toán checksum (mã kiểm tra). Như tôi đã giải thích khi trình bày về định dạng của frame dữ liệu trong giao thức CAN, tôi đã nói rằng có một trường checksum. Mã kiểm tra này được tính toán bằng phương pháp BCH từ trường SOF (Start of Frame) đến hết trường dữ liệu.

Toàn bộ chuỗi bit sau khi đã qua thao tác bit stuffing, sẽ được tính toán để tạo thành một mã kiểm tra dài 15 bit, và mã kiểm tra này được đặt vào trường CRC trong frame dữ liệu.

Khi nhận frame, nút nhận cũng sẽ tính toán mã kiểm tra của chính nó bằng phương pháp BCH, sau đó so sánh mã kiểm tra đã tính toán với mã kiểm tra nhận được trong frame. Nếu hai mã kiểm tra trùng khớp, điều này có nghĩa là frame dữ liệu nhận được là hợp lệ. Nếu không, đó là một frame dữ liệu không hợp lệ.

**Xác nhận và không xác nhận:**

Các nút nhận sẽ gửi xác nhận nếu frame nhận được hợp lệ. Nếu frame không hợp lệ, chúng sẽ không gửi xác nhận. Việc xác nhận hoặc không xác nhận xảy ra trong một khoảng thời gian gọi là "acknowledgement slot," diễn ra sau "checksum delimiter."

Trong khoảng thời gian này, việc xác định frame có hợp lệ hay không được thực hiện, và việc xác nhận (hoặc không xác nhận) xảy ra trong một khoảng thời gian là một bit time.

**Ví dụ về các tình huống lỗi:**

Hãy cùng xem xét một trường hợp không có lỗi. Tất cả các frame và quá trình truyền tải đã diễn ra thành công. Nút phát gửi dữ liệu, và hai nút nhận (Rx Node 1 và Rx Node 2) đều nhận và giám sát CAN bus. Trong khoảng thời gian checksum delimiter (CD), các nút nhận sẽ so sánh mã kiểm tra đã tính toán của mình với mã kiểm tra nhận được từ frame. Nếu frame hợp lệ, các nút nhận sẽ gửi xác nhận bằng cách gửi một bit dominant trong thời gian xác nhận (acknowledgement slot). Khi nút phát giám sát khoảng thời gian này, nó sẽ nhận thấy bit dominant và hiểu rằng frame dữ liệu đã được xác nhận là hợp lệ.

**Trường hợp lỗi xác nhận:**

Trong trường hợp lỗi xác nhận, các nút nhận sẽ phát hiện mã kiểm tra tính toán không khớp với mã kiểm tra trong frame gửi. Khi đó, frame này bị coi là không hợp lệ. Trong thời gian xác nhận, các nút nhận sẽ gửi một bit recessive, điều này có nghĩa là không có xác nhận. Khi nút phát giám sát khoảng thời gian xác nhận và thấy chỉ có bit recessive, nó sẽ hiểu rằng không nhận được xác nhận, tức là frame này không hợp lệ. Đây là một lỗi của nút phát, gọi là lỗi xác nhận (acknowledgement error).

Khi nút phát nhận thấy không có xác nhận, nó sẽ bắt đầu gửi cờ lỗi xác nhận (acknowledgement error flag) ngay sau bit tiếp theo.

**Lỗi xác nhận (Acknowledgement Error)**

Lỗi này xảy ra khi một khung dữ liệu không nhận được xác nhận từ các nút nhận (Rx). Khi nút phát (Tx) gửi một khung dữ liệu nhưng không nhận được tín hiệu xác nhận từ bất kỳ nút nhận nào (Rx node), nó hiểu rằng khung dữ liệu mà nó đã gửi là không hợp lệ. Trong trường hợp này, nút phát sẽ gửi lại khung dữ liệu bắt đầu từ trường SOF (Start of Frame). Đây là lỗi xác nhận.

**Lỗi CRC (Cyclic Redundancy Check Error)**

Lỗi CRC là lỗi xảy ra do tính toán sai mã kiểm tra chu kỳ (checksum). Lỗi này là lỗi từ phía nút nhận (Rx). Giả sử có hai nút nhận. Nút nhận 1 tính toán mã kiểm tra và cho rằng khung dữ liệu hợp lệ vì mã kiểm tra khớp với mã kiểm tra trong khung. Tuy nhiên, nút nhận 2 lại tính toán sai và cho rằng khung dữ liệu không hợp lệ. Trong trường hợp này, nút nhận 1 sẽ gửi một bit xác nhận (acknowledge) mạnh mẽ (dominant bit) vào khe xác nhận, trong khi nút nhận 2 gửi một bit yếu (recessive bit). Bit mạnh sẽ ghi đè lên bit yếu, và nút phát (Tx) sẽ nghĩ rằng khung dữ liệu hợp lệ, nhưng thực tế lại có sự mâu thuẫn giữa các nút nhận.

Khi nút nhận 2 phát hiện ra lỗi trong tính toán mã kiểm tra của mình, nó hiểu rằng mình đã tính toán sai, và khung dữ liệu sẽ bị hủy bỏ. Đây là lỗi CRC.

**Lỗi Stuff (Stuff Error)**

Lỗi này xảy ra khi có quá nhiều bit cùng trạng thái (bit đồng nhất) liên tiếp trong một khung dữ liệu, vi phạm quy tắc "bit stuffing" trong giao thức CAN. Trong CAN, nếu một khung dữ liệu có năm bit liên tiếp cùng trạng thái (ví dụ: năm bit dominant hoặc năm bit recessive), một bit bổ sung sẽ được chèn vào để duy trì đồng bộ hóa giữa nút phát và các nút nhận. Nếu một nút nhận phát hiện ra sáu bit liên tiếp cùng trạng thái mà không có bit bổ sung, nó sẽ báo động về lỗi stuffing.

**Lỗi Form (Form Error)**

Lỗi này xảy ra khi các trường cố định trong khung dữ liệu CAN (chẳng hạn như trường kiểm tra CRC, trường xác nhận, và trường EOF) không tuân thủ định dạng cố định của CAN. Nếu một trong những trường này có bit dominant thay vì recessive, hoặc ngược lại, thì đây là lỗi Form. Khi một lỗi form được phát hiện, nó sẽ tạo ra một khung dữ liệu lỗi.

**Tóm Tắt Các Lỗi CAN**

1. **Lỗi bit** (Bit Error): Xảy ra khi bit trên bus CAN không khớp với bit mà nút phát dự tính.
2. **Lỗi xác nhận** (Acknowledgement Error): Khi không có nút nhận xác nhận rằng khung dữ liệu hợp lệ.
3. **Lỗi CRC** (CRC Error): Xảy ra khi mã kiểm tra CRC bị sai, thường là từ phía nút nhận.
4. **Lỗi stuffing** (Stuff Error): Xảy ra khi có sáu bit cùng trạng thái liên tiếp trong khung dữ liệu.
5. **Lỗi form** (Form Error): Xảy ra khi các trường cố định trong khung dữ liệu bị vi phạm định dạng.

**Kết luận**

Trong giao thức CAN, khi một lỗi được phát hiện, nó sẽ được báo cáo bằng cách gửi một khung lỗi, và khung dữ liệu bị hỏng sẽ bị hủy bỏ. Các lỗi có thể được phát hiện bởi nút phát (transmitter node) hoặc nút nhận (receiver node), và tùy thuộc vào loại lỗi, các hành động khác nhau sẽ được thực hiện.

## CAN Error Frame Format

**Giảng viên**: Chào các bạn sinh viên!

Chào mừng các bạn đến với bài giảng hôm nay, nơi chúng ta sẽ học về định dạng khung lỗi trong giao thức CAN.

Trong bài giảng trước, chúng ta đã tìm hiểu về các lỗi trong CAN, tức là năm loại lỗi trong CAN bao gồm lỗi xác nhận (acknowledgement error), lỗi bit (bit error), lỗi CRC (CRC error), lỗi khung (frame error) hoặc lỗi dạng (form error), và lỗi phần dư (stuff error).

Trong bài học này, chúng ta sẽ tìm hiểu về định dạng của khung lỗi. Trước tiên, chúng ta sẽ hiểu về khung lỗi, sau đó là cờ lỗi (error flag) và các loại cờ lỗi. Tiếp theo, chúng ta sẽ tìm hiểu về khung lỗi cơ bản và khung lỗi hoàn chỉnh, cách mà khung lỗi được hình thành trên mạng CAN, và các trường hợp sử dụng khác nhau để minh họa việc hình thành khung lỗi trên mạng.

Vậy, chúng ta bắt đầu luôn nhé!

Khung lỗi là một trong bốn loại khung trong giao thức CAN, bao gồm các loại khung: khung dữ liệu (data frame), khung yêu cầu (remote frame), khung lỗi (error frame) và khung quá tải (overload frame). Vì vậy, khung lỗi là một trong bốn loại khung của giao thức CAN.

**Mục đích của khung lỗi là gì?**  
Khung lỗi báo hiệu cho các nút khác trên mạng biết rằng khung dữ liệu hiện tại bị lỗi, tức là có vấn đề và không hợp lệ. Nếu bất kỳ nút nào phát hiện điều này, ví dụ như nút phát (transmitter node) nhận ra rằng khung dữ liệu mà nó gửi đi không hợp lệ hoặc có vấn đề, hoặc nút nhận (receiver node) nhận ra rằng khung dữ liệu mà nó nhận được hiện tại trên bus không hợp lệ, thì nó phải thông báo cho các nút khác rằng khung này bị sai và yêu cầu không xử lý khung đó nữa.

Vậy, khi một nút phát hiện một lỗi trong một trong năm loại lỗi CAN mà chúng ta đã tìm hiểu, nó sẽ chồng khung lỗi lên khung dữ liệu hiện tại. Khi khung lỗi chồng lên khung dữ liệu, khung dữ liệu bị hủy bỏ và không nút nào khác sẽ xử lý khung dữ liệu đó. Sau khi khung lỗi kết thúc, khung dữ liệu đó sẽ được truyền lại bởi nút phát.

Để hiểu định dạng của khung lỗi, chúng ta cần tìm hiểu ba yếu tố sau:

1. **Cờ lỗi (Error flag):** Bao gồm cờ lỗi chủ động (active error flag) và cờ lỗi thụ động (passive error flag).
2. **Khung lỗi cơ bản (Basic error frame):** Là định dạng khung lỗi của một nút riêng lẻ.
3. **Khung lỗi hoàn chỉnh (Complete error frame):** Là khung lỗi được hình thành trên bus CAN.

**Cờ lỗi là gì?**  
Khi một nút phát hiện bất kỳ lỗi CAN nào trong năm lỗi đã đề cập, nó biết rằng khung dữ liệu hiện tại không hợp lệ. Vì vậy, cờ lỗi sẽ được gửi đến các nút còn lại trên mạng để báo hiệu rằng khung này có lỗi.

Cờ lỗi phải được nhận biết bởi tất cả các nút khác và không được nhầm lẫn với các bit trong khung dữ liệu bình thường. Vì lý do này, cờ lỗi phải có định dạng đặc biệt, không thể xuất hiện trong bất kỳ khung dữ liệu bình thường nào.

Để đảm bảo điều này, các nhà phát triển giao thức CAN đã sử dụng quy tắc **bit stuffing**. Quy tắc này đảm bảo rằng không thể có chuỗi bit liên tiếp có cùng giá trị trong khung dữ liệu hợp lệ. Nếu có chuỗi bit liên tiếp cùng giá trị, sẽ có sự thay đổi bit sau một số bit nhất định. Cờ lỗi được thiết kế sao cho vi phạm quy tắc này, tức là nó sẽ có sáu bit liên tiếp cùng giá trị (6 bits of same polarity), điều này không thể xuất hiện trong khung dữ liệu hợp lệ.

Có hai loại cờ lỗi:

1. **Cờ lỗi chủ động (Active error flag)**: Là sáu bit liên tiếp với giá trị **dominant** (0).
2. **Cờ lỗi thụ động (Passive error flag)**: Là sáu bit liên tiếp với giá trị **recessive** (1).

**Khung lỗi cơ bản**:  
Khung lỗi cơ bản là định dạng khung lỗi của một nút riêng biệt. Ví dụ, khi một nút phát hiện lỗi trong khung dữ liệu, ngay lập tức nó sẽ chồng khung lỗi lên khung dữ liệu đó. Khung lỗi cơ bản bao gồm ba phần chính:

1. **Cờ lỗi**: Cờ lỗi có thể là cờ lỗi chủ động hoặc thụ động, tùy vào trạng thái của nút (nút ở trạng thái lỗi chủ động hoặc lỗi thụ động).
2. **Delimeter (Phân cách)**: Sau cờ lỗi sẽ là 8 bit phân cách (error delimiter), tất cả các bit này đều là bit **recessive**.
3. **Intermission (Gián đoạn)**: Sau delimiter, sẽ có ba bit gián đoạn (intermission), tất cả đều là **recessive**.

Tổng cộng, kích thước tối thiểu của khung lỗi cơ bản là 17 bit.

**Khung lỗi hoàn chỉnh**:  
Khung lỗi hoàn chỉnh là sự chồng lắp của nhiều khung lỗi cơ bản từ các nút khác nhau. Khi một nút phát hiện lỗi và gửi cờ lỗi, các nút khác cũng có thể phát hiện lỗi và gửi cờ lỗi của riêng mình. Tất cả các cờ lỗi này sẽ chồng lên nhau, tạo thành một khung lỗi hoàn chỉnh trên bus.

Thông thường, khung lỗi hoàn chỉnh có thể có từ 6 đến 12 bit cờ lỗi, tiếp theo là 8 bit delimiter và 3 bit intermission.

**Ví dụ về các tình huống hình thành khung lỗi**:  
Có thể có các tình huống khác nhau khi khung lỗi được hình thành, ví dụ như khi một nút phát hiện lỗi CRC, lỗi dạng, hay lỗi phần dư. Các nút sẽ bắt đầu gửi cờ lỗi của riêng mình khi phát hiện lỗi, và các cờ lỗi này sẽ chồng lên nhau trên bus, hình thành một khung lỗi hoàn chỉnh.



Dưới đây là bản dịch từ tiếng Anh sang tiếng Việt của đoạn văn bạn yêu cầu, với cách diễn đạt dễ hiểu và đầy đủ ý nghĩa:

Một, hai, ba, bốn, năm, sáu, bảy.

Bảy bit còn lại, một, hai, ba, bốn, năm, sáu, bảy bit phân cách sẽ được gửi đi, đúng không?

Trong các trường hợp khác, sau khi cờ lỗi (error flag) được gửi đi, khi bit phân cách đầu tiên được gửi bởi nút, nếu nó phát hiện một bit recessive, ví dụ như trong trường hợp của nút nhận (receiver) thứ hai, nó sẽ gửi bit phân cách đầu tiên, là recessive, và tiếp tục giám sát bit recessive trên bus. Sau đó, nó sẽ gửi các bit phân cách còn lại, đúng không?

Vậy trong trường hợp này, bit phân cách đầu tiên được đưa lên bus. Nó chỉ giám sát bit recessive trên bus, và sau đó gửi các bit phân cách còn lại. Vì lý do này, sẽ luôn xảy ra trong trường hợp mà bảy bit phân cách cuối cùng của nút cuối cùng (nút đã gửi cờ lỗi cuối cùng) và tất cả các nút còn lại. Vì vậy, bảy bit phân cách cuối cùng của tất cả các nút gửi cờ lỗi sẽ tương ứng với nhau, tức là chúng sẽ trùng nhau và đồng bộ với nhau, và vì thế, khoảng cách nghỉ (intermission space) cũng vậy.

Kết hợp lại, sẽ tạo thành một khung dữ liệu hoàn chỉnh còn lại trên bus. Có một khung hoàn chỉnh, sáu đến mười hai bit của cờ lỗi, tám bit phân cách, ba bit của IFS (Inter Frame Spacing), trong trường hợp này là sáu cộng sáu là mười hai, cộng với tám là hai mươi, hai mươi cộng ba là hai mươi ba. Vì vậy, mất hai mươi ba bit thời gian cho một khung lỗi ở mức tối đa.

Vậy đó là thời gian phục hồi lỗi, có nghĩa là nếu một khung dữ liệu bị lỗi trên bus và bị hủy, thì thời gian phục hồi lỗi là một thời gian của hai mươi ba bit. Một bit thời gian có thể được tính toán bằng cách biết tốc độ truyền của bộ điều khiển CAN, tùy vào cấu hình.

Đây là ví dụ về cờ lỗi cơ bản, các cờ lỗi và khung lỗi hoàn chỉnh được tìm thấy trên bus.

Bây giờ, chúng ta cũng có cờ lỗi hoạt động (active error flag) và cờ lỗi thụ động (passive error flag), đúng không?

Chúng ta sẽ xem xét các trường hợp cờ lỗi thụ động, một trường hợp mà cờ lỗi thụ động được gửi bởi nút nhận và trường hợp khác là cờ lỗi thụ động được gửi bởi nút truyền.

Sự khác biệt là gì, đúng không? Cái này.

Bây giờ, nút nhận là nút có cờ lỗi thụ động. Trong trường hợp này, đúng không?

Bây giờ, hãy xem trường hợp này. Như đã giải thích trước đó, có một nút truyền và hai nút nhận. Rx Node 1, Rx Node 2 và sóng hình cuối cùng trên bus CAN.

Vậy nút truyền đang gửi một khung lên bus và nút nhận. Giả sử vào thời điểm này, nơi cờ lỗi ED được nhắc đến, tại thời điểm bit này, nó đã phát hiện ra một lỗi. Một lỗi đã được phát hiện, đúng không?

Vậy nó sẽ gửi cờ lỗi thụ động. Cờ lỗi thụ động có nghĩa là sáu bit recessive liên tiếp, tức là 1, 1, 1, 1, 1, 1.

Nút truyền cũng đang gửi khung bình thường của nó và nút nhận đang gửi cờ lỗi của nó, đúng không? Vì đây là các bit nhận, khi nút truyền gửi một bit, nút nhận cũng gửi cờ lỗi của nó.

Bus sẽ nhận một bit duy nhất, nhưng ngược lại, khi nút truyền gửi một bit dominant, nhưng nút nhận lại gửi cờ lỗi thụ động với bit recessive, thì bus sẽ nhận một bit dominant.

Do đó, cái gì sẽ xảy ra là sóng hình tạo thành trên bus CAN sẽ giống hệt như sóng hình mà nút truyền phát đi. Vì vậy, như thể nút nhận đang gửi cờ lỗi thụ động không có ảnh hưởng gì lên bus CAN và nút nhận thứ hai cũng vậy, khi nó giám sát bus CAN, nó sẽ nhận được chính xác sóng hình mà nút truyền đang gửi.

Điểm tôi muốn nói ở đây là, vì trong một khung lỗi thụ động, cờ lỗi cũng bao gồm tất cả các bit recessive, các bit phân cách cũng có tất cả các bit recessive. Khoảng thời gian nghỉ cũng có tất cả các bit recessive.

Do đó, khi một nút nhận đang cố gắng gửi cờ lỗi thụ động hoặc một khung lỗi thụ động, nó sẽ không thể phá hủy khung dữ liệu đang được truyền tải trên bus.

Vì vậy, một cờ lỗi thụ động gửi bởi nút nhận không thể phá hủy khung dữ liệu và do đó không có tác động gì lên bus. Đây là điểm tôi muốn làm rõ.

Nếu bạn muốn, bạn có thể xem lại các quy tắc về bit phân cách và kiểm tra sóng hình này từng bit một.

Được chứ?

Đó là điểm cần nhớ.

Trường hợp tiếp theo là khi nút truyền đã phát hiện lỗi và nút truyền có cờ lỗi thụ động trong khi các nút nhận chỉ có cờ lỗi hoạt động.

Bây giờ, nút truyền đang gửi các khung. Vậy ở bit đầu tiên không có vấn đề gì, hoạt động bình thường. Bit thứ hai, hoạt động bình thường, bit thứ ba cũng vậy.

Giả sử ở bit thứ tư, nơi nút truyền đang cố gắng gửi bit dominant và vì một lý do nào đó trên bus CAN hoặc phần cứng của nút, bus lại nhận bit recessive, trong khi nút truyền đang gửi bit dominant.

Khi nút truyền gửi một bit dominant nhưng nhận được bit recessive, nó sẽ nhận diện một lỗi bit vì bit mà nó đang cố gắng gửi và bit mà nó giám sát là khác nhau.

Do đó, nó phát hiện ra một lỗi bit ở đây, đó là nơi mà lỗi bit được phát hiện.

Vậy tiếp theo, nó sẽ gửi cờ lỗi thụ động vì nó sẽ gửi khung lỗi. Cờ lỗi, vì đây là cờ lỗi của nút truyền, nên nó sẽ có cờ lỗi thụ động.

Cờ lỗi thụ động là sáu bit recessive liên tiếp, tức là 1, 1, 1, 1, 1, 1.

Vậy trên bus, trong trường hợp này, đã có ba bit recessive rồi. Đây là ba bit cờ lỗi thụ động. Các bit còn lại là ba bit cờ lỗi thụ động.

Với ba bit cờ lỗi thụ động đầu tiên, nút nhận chỉ nhận được bit thụ động mà thôi, nó không biết về lỗi.

Sau đó, trong các bit tiếp theo, cờ lỗi thụ động sẽ được gửi lên bus và tiếp theo là bit phân cách.

Chúng ta sẽ thấy rằng sau khi cờ lỗi thụ động đã được gửi xong, các bit phân cách sẽ được gửi. Cờ lỗi hoạt động của các nút nhận sẽ bắt đầu, vì vậy khung dữ liệu sẽ được đồng bộ hóa với các cờ lỗi và các bit phân cách.

## CAN Error States