Lý thuyết CAN

1. Giới thiệu

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

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

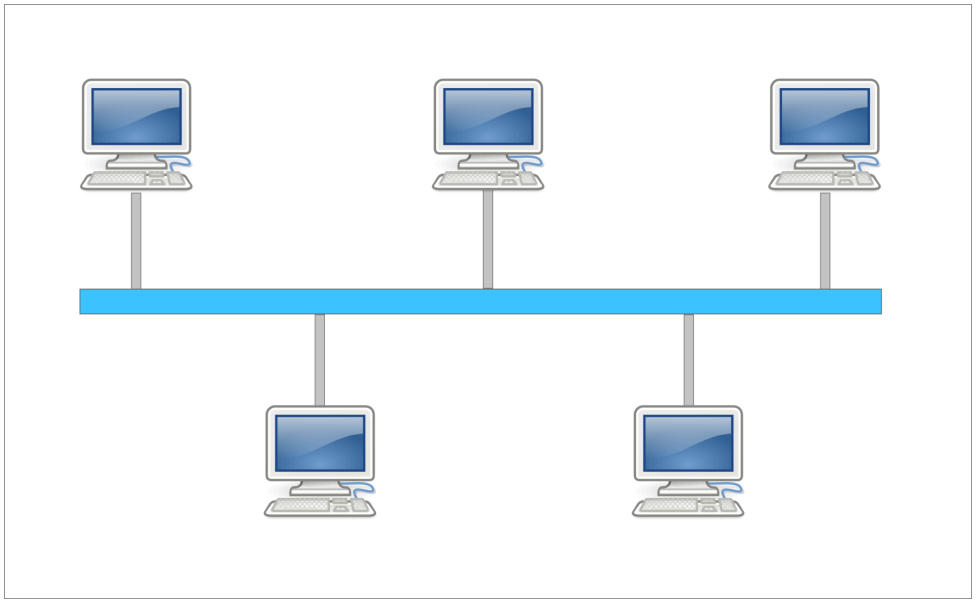
CAN được phát triển để giải quyết các vấn đề này bằng cách:

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

2. Kiến trúc

2.1 Bus topology

Giao thức CAN sử dụng **tô-pô bus** để kết nối các thiết bị với nhau, nghĩa là tất cả các thiết bị (node) đều được kết nối song song vào một cặp dây truyền thông chung được gọi là **CAN bus**.



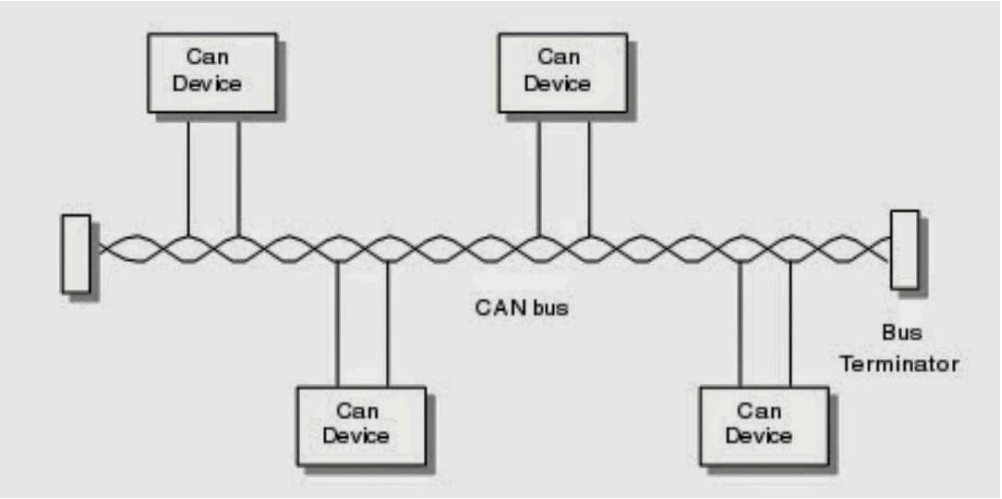
CAN bus này gồm hai dây tín hiệu chính, là:

- **CANH (CAN High)**: Dây tín hiệu cao.
- **CANL (CAN Low)**: Dây tín hiệu thấp.

Các tín hiệu truyền qua bus CAN là tín hiệu vi sai (differential signaling), nghĩa là thông tin được mã hóa dựa trên sự chênh lệch điện áp giữa hai dây **CANH** và **CANL**. Điều này giúp mạng CAN chống lại nhiễu từ môi trường và duy trì tín hiệu ổn định trên đường truyền.

Hai dây tín hiệu này được xoắn lại tạo thành đường dây xoắn đôi giúp:

- **Giảm thiểu nhiễu từ môi trường bên ngoài**: Khi các dây được xoắn lại, mỗi đoạn của cặp dây sẽ nhận nhiễu với cường độ khác nhau và theo các hướng ngược nhau, làm triệt tiêu phần lớn nhiễu điện từ.
- **Giảm thiểu nhiễu xuyên âm**: Việc xoắn đôi các dây giúp giảm hiện tượng này bằng cách phân tán nhiễu xuyên âm ra khắp chiều dài của cáp.



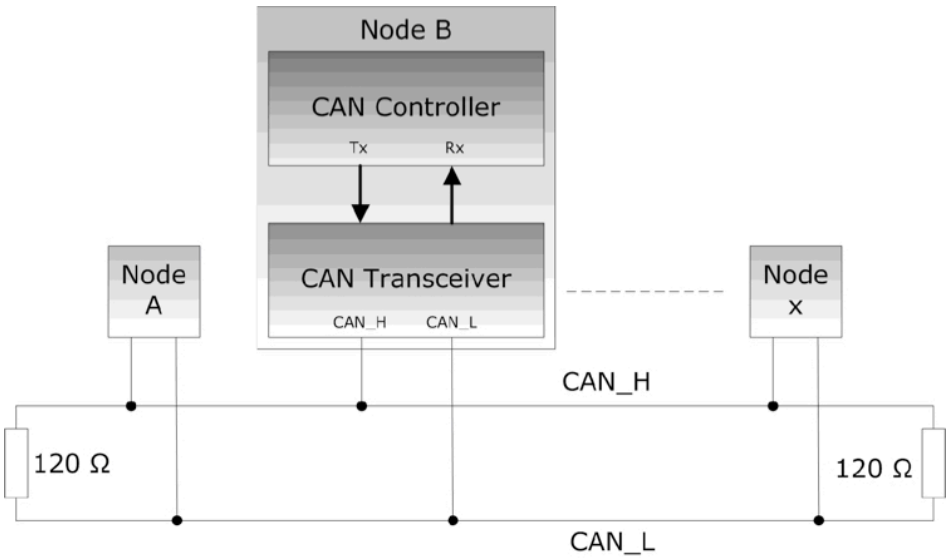
Đặc điểm của tô-pô bus:

- **Kết nối song song**: Tất cả các thiết bị trên bus CAN đều được kết nối song song với nhau. Mỗi thiết bị (node) có thể truy cập vào bus để truyền hoặc nhận dữ liệu bất cứ lúc nào, mà không cần một máy tính chủ (master) điều khiển.
- **Giảm số lượng dây dẫn**: Với tô-pô bus, tất cả các thiết bị chia sẻ chung một bus truyền dữ liệu, làm giảm đáng kể số lượng dây dẫn so với các mô hình khác. Điều này giúp giảm chi phí, tiết kiệm không gian và đơn giản hóa hệ thống dây dẫn trong các hệ thống nhúng.
- **Termination resistor (Điện trở kết cuối)**: Mỗi đầu của bus CAN cần một điện trở kết cuối với giá trị **120Ω** để ngăn chặn hiện tượng phản xạ tín hiệu. Nếu không có điện trở này, tín hiệu có thể bị phản xạ lại từ các đầu cuối mở, gây ra nhiễu và làm hỏng dữ liệu.

2.2 Các thiết bị trên bus CAN

Mạng CAN hỗ trợ nhiều loại thiết bị khác nhau trên cùng một bus, mỗi thiết bị được gọi là một **node**. Các thiết bị này có thể là:

- **Vi điều khiển (Microcontroller)**: Đây là các bộ điều khiển chính trong các hệ thống nhúng. Ví dụ, một vi điều khiển STM32 có thể kết nối với bus CAN để điều khiển các thiết bị khác hoặc thu thập dữ liệu từ các cảm biến.
- **Cảm biến (Sensors)**: Các cảm biến thu thập dữ liệu từ môi trường (như nhiệt độ, áp suất, tốc độ) và gửi dữ liệu này lên bus CAN để các thiết bị khác xử lý.
- **Actuator (Thiết bị kích động)**: Đây là các thiết bị đầu ra, nhận lệnh từ các vi điều khiển qua bus CAN để thực hiện các hành động vật lý, chẳng hạn như mở van, điều khiển động cơ hoặc bật đèn.



Các node nếu muốn gửi và nhận dữ liệu CAN thì bên trong các node cần có những thành phần sau:

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

2.3 Đặc điểm giao tiếp của CAN

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

- **Không cần máy tính chủ (No Master-Slave Architecture)**

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

- **Truyền thông quảng bá (Broadcast Communication)**

Khi một node gửi thông điệp, thông điệp đó sẽ được phát sóng đến tất cả các node khác trên bus. Tuy nhiên, không phải tất cả các node đều xử lý thông điệp này. Mỗi node sẽ sử dụng bộ lọc để kiểm tra xem thông điệp có phù hợp với mình hay không.

- **Tranh chấp quyền gửi (Arbitration)**

Một đặc điểm quan trọng của mạng CAN là khả năng giải quyết tranh chấp quyền gửi dữ liệu giữa các node. Nếu có nhiều node cùng muốn gửi dữ liệu lên bus cùng một lúc, cơ chế arbitration sẽ được thực hiện:

  - Mỗi thông điệp CAN có một ID ưu tiên. Node nào có thông điệp với ID ưu tiên thấp hơn (tức có độ ưu tiên cao hơn) sẽ chiếm quyền truy cập bus và gửi thông điệp trước.
  - Những node khác có ID ưu tiên cao hơn sẽ tự động dừng lại và chờ lượt tiếp theo để gửi thông điệp.
  - Quá trình arbitration diễn ra mà không gây mất dữ liệu hay làm gián đoạn các thiết bị khác, vì thế mạng CAN là một hệ thống non-destructive (không gây mất dữ liệu).

- **Giao tiếp song công (Full-duplex Communication)**

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

- **Phát hiện và xử lý lỗi tự động**

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

3. Khung dữ liệu CAN

Trong mạng CAN, dữ liệu được truyền thông qua các **khung dữ liệu** (CAN Frame). Mỗi khung dữ liệu bao gồm nhiều trường khác nhau, từ việc chỉ định ID của thiết bị gửi cho đến việc chứa dữ liệu và thông tin kiểm tra lỗi. Mạng CAN hỗ trợ

nhiều loại khung dữ liệu, mỗi loại có chức năng cụ thể. Dưới đây là chi tiết về các loại khung và cấu trúc của một khung dữ liệu trong mạng CAN.

3.1 Các loại khung dữ liệu trong CAN

3.1.1 Data Frame

Đây là loại khung phổ biến nhất và được sử dụng để truyền dữ liệu thực sự trên mạng CAN. **Data Frame** chứa thông tin về ID của node gửi và dữ liệu được truyền. Mỗi khung có thể chứa tối đa **8 byte dữ liệu**.

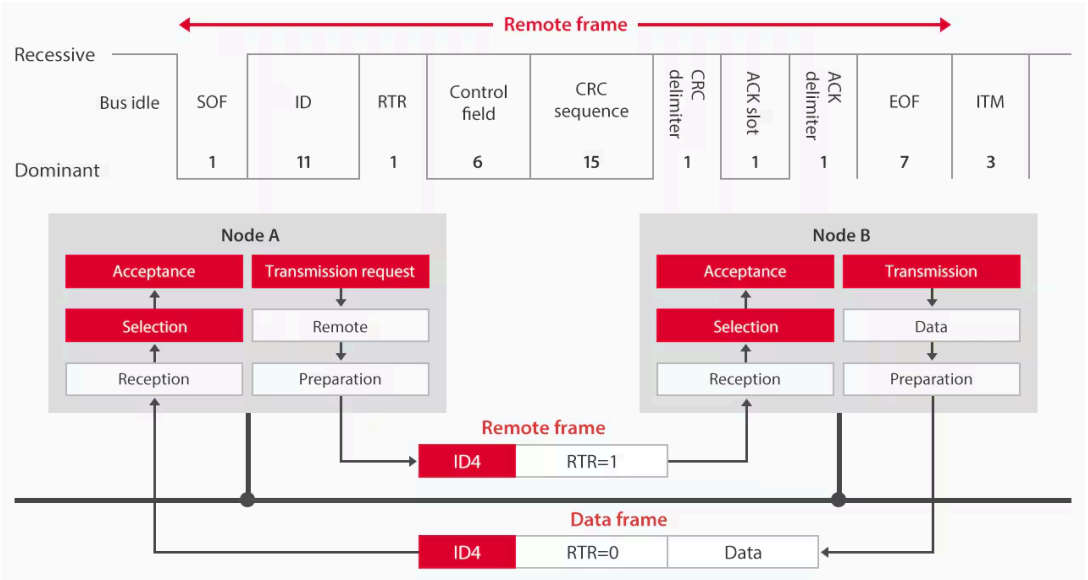
Cấu trúc của Data Frame:

- **ID (Identifier)**: Mỗi thông điệp trong mạng CAN đều có một ID, thể hiện mức độ ưu tiên của thông điệp. ID càng thấp thì thông điệp càng được ưu tiên cao hơn trong quá trình tranh chấp quyền gửi (arbitration).
- **Payload**: Đây là phần chứa dữ liệu chính của thông điệp, có thể chứa từ **0 đến 8 byte** dữ liệu.

3.1.2 Remote Frame

Remote Frame được sử dụng khi một node trên mạng CAN yêu cầu dữ liệu từ một node khác. Thay vì chứa dữ liệu thực, **Remote Frame** chứa ID của node cần yêu cầu, cùng với bit điều khiển **RTR (Remote Transmission Request)**.

Remote Frame thường được sử dụng trong các hệ thống mà một node có thể yêu cầu thông tin từ một node khác mà không có dữ liệu nào được truyền ngay lập tức. Node nhận yêu cầu sẽ trả lời bằng một **Data Frame** chứa dữ liệu cần thiết.

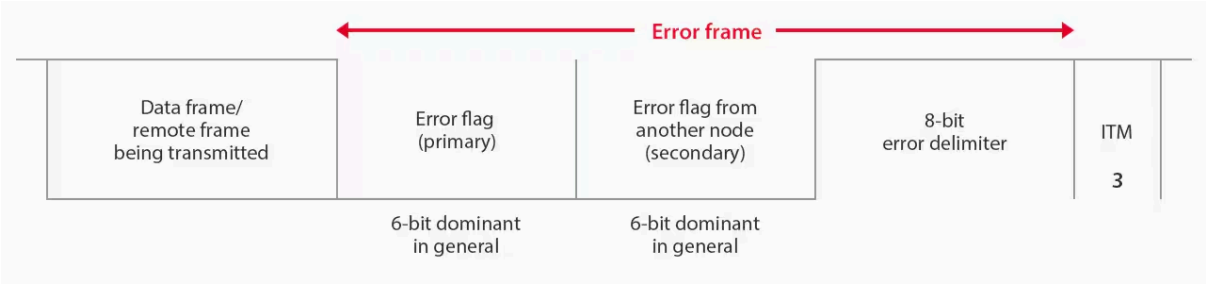


3.1.3 Error Frame

Error Frame được sử dụng khi một node phát hiện ra lỗi trong quá trình truyền dữ liệu. Nó được gửi để thông báo cho các node khác rằng có lỗi đã xảy ra trên bus. Bất kỳ node nào phát hiện ra lỗi đều có thể gửi **Error Frame**.

Error Frame có vai trò rất quan trọng trong việc duy trì độ tin cậy của mạng CAN. Khi một lỗi xảy ra, **Error Frame** sẽ báo hiệu để các node khác biết rằng thông điệp vừa được truyền không hợp lệ và cần được truyền lại.

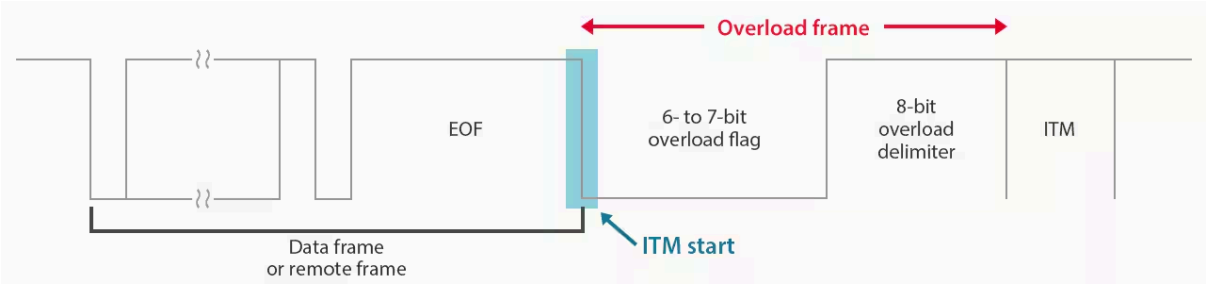
Error Frame gồm hai phần: **Error Flag** và **Error Delimiter**. **Error Flag** là chuỗi từ 6 đến 12 bit dominant, báo hiệu lỗi. **Error Delimiter** là chuỗi 8 bit recessive, kết thúc Error Frame.



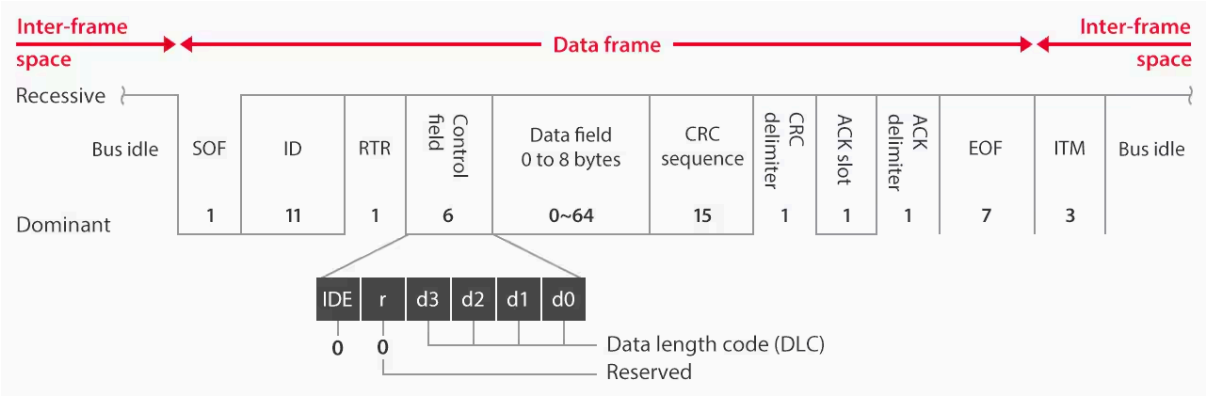
3.1.4 Overload Frame

Overload Frame được sử dụng để báo hiệu rằng một node đang trong trạng thái bận và không thể xử lý thêm thông điệp nào ngay lập tức. Điều này có thể xảy ra khi một node chưa xử lý xong thông điệp trước đó hoặc hệ thống quá tải.

Khi một node gửi **Overload Frame**, nó báo hiệu cho các node khác trên mạng rằng chúng cần dừng truyền thông trong một thời gian ngắn để giảm tải cho node đó.



3.2 Cấu trúc của một khung dữ liệu trong CAN



3.2.1 Start of Frame (SOF)

SOF là bit bắt đầu của khung dữ liệu, chỉ có giá trị dominant (0). Nó báo hiệu rằng một khung dữ liệu mới đang bắt đầu. Tất cả các node trên mạng sẽ nhận biết rằng đây là thời điểm để bắt đầu đọc dữ liệu.

3.2.2 Arbitration Field (Trường tranh chấp)

Trường này chứa ID của thông điệp và bit RTR (Remote Transmission Request).

- **ID**: Chứa định danh của thông điệp, ID này được sử dụng để xác định mức độ ưu tiên trong quá trình arbitration (tranh chấp quyền gửi).
- **RTR**: Đối với Data Frame, bit này sẽ có giá trị dominant (0). Đối với Remote Frame, bit này sẽ có giá trị recessive (1), báo hiệu rằng đây là một yêu cầu dữ liệu từ một node khác.

3.2.3 Control Field (Trường điều khiển)

Control Field chứa các thông tin về kích thước của phần dữ liệu.

DLC (Data Length Code): Đây là trường quan trọng trong Control Field, xác định độ dài của dữ liệu (từ 0 đến 8 byte).

3.2.4 Data Field (Trường dữ liệu)

Data Field là phần chứa dữ liệu chính của khung, có thể có từ 0 đến 8 byte dữ liệu. Trong Data Frame, đây là nơi chứa thông tin mà node gửi muốn truyền tải.

3.2.5 CRC Field (Trường kiểm tra lỗi)

CRC (Cyclic Redundancy Check): Đây là trường kiểm tra lỗi, giúp phát hiện các lỗi xảy ra trong quá trình truyền dữ liệu. Node nhận sẽ sử dụng CRC Field để kiểm tra xem dữ liệu đã được truyền chính xác hay chưa. Nếu phát hiện lỗi, một Error Frame sẽ được gửi đi.

3.2.6 ACK Field (Trường xác nhận)

ACK Field được sử dụng để xác nhận rằng một thông điệp đã được nhận thành công. Khi một node nhận được dữ liệu mà không phát hiện lỗi, nó sẽ gửi bit ACK dominant (0) vào trường ACK để thông báo cho node gửi rằng dữ liệu đã được nhận chính xác.

Nếu không có node nào gửi ACK, điều này báo hiệu rằng có lỗi xảy ra hoặc thông điệp không được nhận đúng cách, và node gửi sẽ phải truyền lại thông điệp.

3.2.7 End of Frame (EOF)

EOF là trường kết thúc của khung dữ liệu, chứa một chuỗi các bit recessive (1). Trường này báo hiệu rằng toàn bộ khung dữ liệu đã được truyền và quá trình truyền thông cho khung này đã kết thúc.

4. Abitration trong CAN

Arbitration (tranh chấp quyền gửi) là một cơ chế quan trọng của giao thức CAN, cho phép nhiều node trên bus có thể cố gắng truyền thông điệp đồng thời mà không gây xung đột hoặc mất dữ liệu. Cơ chế này đảm bảo rằng **node có mức độ ưu tiên cao nhất** sẽ chiếm quyền truy cập bus, trong khi các node có ưu tiên thấp hơn sẽ đợi lượt tiếp theo. Điều đặc biệt là quá trình này diễn ra **một cách không phá hủy (non-destructive)**, tức là không có bất kỳ dữ liệu nào bị mất khi có xung đột về quyền gửi.

4.1 Cơ chế ưu tiên

Trong mạng CAN, **ID của thông điệp** đóng vai trò quan trọng trong việc xác định **mức độ ưu tiên**. Mỗi thông điệp CAN đều có một **ID định danh (identifier)**, và giá trị của ID này quyết định mức độ ưu tiên khi có nhiều node cố gắng gửi dữ liệu cùng một lúc.

- **ID thấp hơn tương ứng với mức độ ưu tiên cao hơn**. Nghĩa là, khi nhiều node cùng muốn truyền dữ liệu, node có ID nhỏ hơn (giá trị nhị phân thấp hơn) sẽ được ưu tiên và thắng quá trình arbitration.
- Mỗi bit trong ID của thông điệp có thể ở trạng thái **dominant (trạng thái ưu tiên - giá trị 0)** hoặc **recessive (trạng thái không ưu tiên - giá trị 1)**. Khi hai node hoặc nhiều node cùng gửi dữ liệu, CAN sử dụng quy tắc **wire-AND logic** để quyết định node nào được ưu tiên.

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

Nguyên lý hoạt động:

- Khi nhiều node muốn truyền dữ liệu, chúng đều bắt đầu gửi thông điệp của mình lên bus. Tín hiệu được gửi đồng thời và mỗi node sẽ kiểm tra từng bit của dữ liệu trên bus.
- Mỗi bit trong ID sẽ được truyền từng cái một (từ bit MSB - Most Significant Bit). Nếu một node gửi **bit recessive (1)** nhưng nhận thấy trên bus có **bit dominant (0)**, nghĩa là có một node khác có ưu tiên cao hơn đang chiếm quyền truyền dữ liệu. Lúc này, node này sẽ ngừng truyền và chuyển sang chế độ nghe (listen).
- Node có ID thấp hơn (tức là có nhiều bit dominant hơn ở đầu) sẽ tiếp tục quá trình truyền cho đến khi toàn bộ ID được gửi đi, trong khi các node khác ngừng gửi và chuyển sang chế độ chờ.
- Các node không thắng quá trình arbitrage sẽ không bị mất dữ liệu mà chỉ đơn giản là đợi lượt tiếp theo để cố gắng truyền lại thông điệp của mình.

4.2 Non-destructive Arbitration (Tranh chấp không phá hủy)

Cơ chế **non-destructive arbitration** có nghĩa là quá trình arbitrage diễn ra mà **không làm mất dữ liệu** của các node thua. Điều này có được nhờ vào tính năng **multi-master** và cơ chế **wire-AND logic** của CAN.

- Khi một node thua trong quá trình arbitration, nó sẽ **tạm dừng việc truyền** nhưng không xóa dữ liệu của mình.
- Node thua sẽ **chuyển sang trạng thái chờ** và lắng nghe bus. Khi bus không còn bận (tức là node thắng đã gửi xong thông điệp), node thua sẽ **thử lại** và tham gia tranh chấp quyền gửi ở lần tiếp theo.
- Quá trình này đảm bảo rằng không có dữ liệu bị mất trong quá trình tranh chấp, vì các node thua sẽ tiếp tục gửi thông điệp của mình vào thời điểm thích hợp.

5. Lỗi trong giao thức CAN

Trong CAN, cơ chế phát hiện và sửa lỗi là một tính năng quan trọng để đảm bảo độ tin cậy và tính ổn định của quá trình truyền dữ liệu. Hệ thống CAN có thể tự động phát hiện nhiều loại lỗi khác nhau trong quá trình truyền thông qua các công cụ

như kiểm tra bit, CRC và ACK. Khi lỗi được phát hiện, CAN có khả năng xử lý và sửa lỗi một cách tự động mà không làm gián đoạn toàn bộ quá trình giao tiếp.

5.1 Các loại lỗi trong CAN

5.1.1 Bit Error

Bit Error xảy ra khi một node gửi một bit (dominant hoặc recessive) lên bus và nhận lại một bit khác với giá trị mong đợi. Trong mạng CAN, mỗi node không chỉ gửi dữ liệu mà còn tự lắng nghe các tín hiệu trên bus để kiểm tra sự đồng bộ.

- **Bit dominant (0)**: Tín hiệu ưu tiên trên bus.
- **Bit recessive (1)**: Tín hiệu không ưu tiên trên bus.

Nguyên nhân:

- Nếu một node gửi một **bit recessive (1)** nhưng nhận lại **bit dominant (0)** từ bus, node này sẽ phát hiện ra lỗi.
- Điều này có thể xảy ra khi một node khác có ưu tiên cao hơn trên bus đang truyền dữ liệu, hoặc do tín hiệu bị nhiễu.

5.1.2 Stuff Error

Stuff Error xảy ra khi có **hơn 5 bit liên tiếp cùng giá trị** (tất cả đều là 0 hoặc tất cả đều là 1) trên bus CAN. Điều này vi phạm quy tắc **bit stuffing** của giao thức CAN.

- **Quy tắc bit stuffing**: Trong mạng CAN, sau mỗi chuỗi 5 bit giống nhau liên tiếp, một bit **ngược giá trị** (ngược với giá trị của các bit trước đó) phải được thêm vào để đảm bảo tính đồng bộ và tránh nhiễu tín hiệu. Nếu quy tắc này bị vi phạm, lỗi sẽ xảy ra.
- **Nguyên nhân**: Vi phạm quy tắc bit stuffing có thể do lỗi trong quá trình truyền tín hiệu hoặc do thiết bị không tuân theo quy chuẩn CAN.

5.1.3 CRC Error

CRC Error xảy ra khi có sai lệch trong quá trình kiểm tra **CRC (Cyclic Redundancy Check)**, được sử dụng để phát hiện lỗi trong dữ liệu truyền qua bus.

Cơ chế CRC:

- Trong mỗi khung dữ liệu CAN, có một **CRC Field** được sử dụng để kiểm tra tính toàn vẹn của dữ liệu. Trường này chứa giá trị CRC, được tính toán dựa trên nội dung của thông điệp.
- Node nhận sẽ tính toán lại giá trị CRC của dữ liệu nhận được và so sánh với CRC trong trường CRC Field. Nếu hai giá trị này không khớp, một **CRC error** sẽ được phát hiện.

Nguyên nhân: Lỗi CRC có thể xảy ra do nhiễu tín hiệu trong quá trình truyền dữ liệu hoặc do lỗi phần cứng trong node gửi hoặc nhận.

5.1.4 Form Error

Form Error xảy ra khi **cấu trúc khung dữ liệu** không tuân theo quy chuẩn của giao thức CAN. Mỗi khung dữ liệu trong CAN phải tuân theo một cấu trúc định sẵn, bao gồm **Start of Frame (SOF)**, **Arbitration Field**, **Control Field**, **Data Field**, **CRC Field**, **ACK Field**, và **End of Frame (EOF)**.

Nguyên nhân: Nếu một node nhận thấy có lỗi trong định dạng của bất kỳ trường nào trong khung dữ liệu, đặc biệt là các bit trong **EOF** hoặc **ACK Field**, nó sẽ phát hiện **Form Error**.

5.1.5 Acknowledgment Error

Acknowledgment Error (ACK Error) xảy ra khi node gửi thông điệp lên bus mà **không nhận được bit ACK** từ bất kỳ node nào trên mạng.

Cơ chế ACK trong CAN:

- Khi một node gửi thành công một khung dữ liệu, các node nhận phải gửi một **bit ACK dominant (0)** để xác nhận rằng dữ liệu đã được nhận chính xác.
- Nếu không có node nào gửi **bit ACK**, node gửi sẽ phát hiện **ACK Error** và phải truyền lại thông điệp.

Nguyên nhân:

- Thiết bị nhận có thể không hoạt động đúng cách hoặc không kết nối đúng vào bus CAN.
- Tín hiệu ACK có thể bị nhiễu hoặc lỗi phần cứng.

5.2 Cơ chế phát hiện lỗi trong mạng CAN

Mạng CAN sử dụng nhiều cơ chế để phát hiện lỗi, giúp duy trì tính ổn định và tin cậy của dữ liệu truyền tải trên bus. Các cơ chế này bao gồm:

- **Kiểm tra bit**: Mỗi node gửi sẽ tự lắng nghe dữ liệu mà nó vừa gửi để đảm bảo rằng dữ liệu đó đã được truyền đúng cách. Nếu có sự khác biệt giữa bit gửi đi và bit nhận lại, node sẽ phát hiện **bit error**.
- **Kiểm tra CRC**: Mỗi thông điệp CAN chứa một giá trị **CRC** được tính toán dựa trên dữ liệu. Node nhận sẽ tính toán lại giá trị CRC và so sánh với CRC của thông điệp để phát hiện lỗi.
- **Kiểm tra định dạng (Form Check)**: Các bit trong **EOF** và **ACK Field** phải tuân theo một định dạng chuẩn. Nếu không, node nhận sẽ phát hiện **form error**.
- **Xác nhận (Acknowledgment)**: Node gửi sẽ kiểm tra xem có bất kỳ node nào trên bus gửi bit **ACK** để xác nhận rằng dữ liệu đã được nhận thành công. Nếu không, **ACK error** sẽ được phát hiện.

5.3 Cơ chế sửa lỗi tự động trong mạng CAN

Khi lỗi được phát hiện, mạng CAN có khả năng sửa lỗi một cách tự động thông qua quá trình phát **Error Frame** và truyền lại thông điệp.

Cơ chế sửa lỗi trong CAN:

- **Error Frame**: Khi một node phát hiện lỗi (bit error, CRC error, form error, stuff error, hoặc ACK error), nó sẽ gửi một **Error Frame** để thông báo cho tất cả các node khác trên bus rằng có lỗi đã xảy ra.
- **Truyền lại thông điệp**: Sau khi **Error Frame** được phát, các node sẽ dừng giao tiếp và node gửi ban đầu sẽ cố gắng **truyền lại thông điệp** bị lỗi. Việc này sẽ tiếp tục cho đến khi thông điệp được truyền đi thành công hoặc node gửi bị đưa vào trạng thái **bus off** nếu lỗi quá nhiều.

5.4 Các trạng thái lỗi của node

Khi phát hiện lỗi, các node trong mạng CAN sẽ tự động chuyển đổi giữa ba trạng thái lỗi để đảm bảo hệ thống hoạt động ổn định và không gây gián đoạn cho bus.

5.4.1 Error Active

Trong trạng thái **Error Active**, node vẫn có khả năng tham gia đầy đủ vào quá trình truyền thông và có thể phát hiện lỗi. Nếu node phát hiện lỗi, nó sẽ gửi một **Error Frame** để thông báo cho các node khác trên bus rằng đã xảy ra lỗi.

5.4.2 Error Passive

Nếu một node phát hiện quá nhiều lỗi, nó sẽ chuyển sang trạng thái **Error Passive**. Trong trạng thái này, node vẫn có thể tham gia truyền thông, nhưng nếu phát

hiện lỗi, nó sẽ **không gửi Error Frame mạnh mẽ** như trong trạng thái **Error Active**. Điều này giúp tránh gây gián đoạn lớn cho bus khi node gặp sự cố thường xuyên.

Trong trạng thái **Error Passive**, node vẫn có thể nhận và gửi thông điệp nhưng sẽ hạn chế việc can thiệp vào quá trình truyền thông của các node khác. Node chỉ gửi **Error Frame yếu hơn** để thông báo lỗi, và không ảnh hưởng đến quá trình truyền thông của các node khác.

5.4.3 Bus Off

Khi một node gặp quá nhiều lỗi nghiêm trọng, nó sẽ chuyển sang trạng thái **Bus Off**. Trong trạng thái này, node sẽ hoàn toàn **ngắt kết nối khỏi bus CAN** và không thể tham gia vào quá trình truyền hay nhận dữ liệu. Node chỉ có thể được kết nối lại vào bus sau khi được **khởi động lại** (restart) hoặc **reset** bởi phần mềm.

Bus Off là trạng thái an toàn, ngăn chặn một node bị lỗi nặng gây ra sự cố nghiêm trọng cho toàn bộ hệ thống CAN.

6. Tốc độ truyền và giới hạn vật lý của CAN

Giao thức **CAN** được thiết kế để hoạt động hiệu quả trong các hệ thống nhúng với khả năng truyền thông dữ liệu tin cậy và độ trễ thấp. Một số yếu tố quan trọng ảnh hưởng đến hiệu suất của mạng CAN bao gồm **tốc độ truyền (baud rate)**, **chiều dài của bus**, và **điện trở kết cuối (termination resistor)**. Các yếu tố này liên quan chặt chẽ với nhau, ảnh hưởng đến khả năng truyền tín hiệu trên bus CAN.

6.1 Tốc độ baud của CAN

Tốc độ baud là tốc độ truyền dữ liệu trên bus CAN, thường được đo bằng **kbps (kilobits per second)** hoặc **Mbps (megabits per second)**. Tốc độ baud quyết định tốc độ truyền thông giữa các thiết bị trên mạng và phụ thuộc vào khả năng xử lý của hệ thống cũng như chiều dài của bus.

Dải tốc độ baud của CAN:

Mạng CAN hỗ trợ **dải tốc độ baud từ 10 kbps đến 1 Mbps**.

- **10 kbps**: Tốc độ thấp nhất, thường được sử dụng cho các hệ thống có yêu cầu truyền thông chậm, nhưng cần truyền xa.
- **1 Mbps**: Tốc độ cao nhất, thường được sử dụng trong các ứng dụng yêu cầu truyền thông nhanh, chẳng hạn như trong hệ thống ô tô hoặc robot.

Ảnh hưởng của tốc độ baud:

- **Chiều dài tối đa của bus**: Tốc độ truyền càng cao, chiều dài tối đa của bus càng ngắn do ảnh hưởng của thời gian lan truyền tín hiệu trên bus. Điều này có nghĩa là khi cần truyền dữ liệu với tốc độ cao, hệ thống phải chấp nhận giảm chiều dài của bus để đảm bảo tín hiệu truyền chính xác và đồng bộ.
- **Độ trễ**: Tốc độ baud càng cao, độ trễ của việc truyền thông tin trên mạng càng giảm, giúp cải thiện khả năng đáp ứng của hệ thống.

6.2 Chiều dài tối đa của bus trong CAN

Chiều dài của bus trong mạng CAN bị giới hạn bởi tốc độ baud và chất lượng của dây dẫn (bus). Sự kết hợp giữa tốc độ truyền và chiều dài của bus quyết định khả năng truyền tín hiệu đúng cách và độ tin cậy của mạng.

- **Tốc độ truyền càng cao, chiều dài bus càng ngắn**: Điều này do thời gian lan truyền tín hiệu trên dây dẫn cần phải nhỏ hơn một khoảng thời gian nhất định để đảm bảo tất cả các node trên bus có thể nhận được tín hiệu đồng bộ.
- Khi tốc độ baud tăng lên, thời gian bit ngắn lại, nghĩa là tín hiệu phải đến các node nhận nhanh hơn. Do đó, chiều dài tối đa của bus phải giảm để đảm bảo thời gian lan truyền tín hiệu phù hợp với tốc độ baud.

7. CAN và các phiên bản mở rộng

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

7.1 CAN 2.0A

Đặc điểm chính của CAN 2.0A:

- **11-bit identifier (ID)**: Phiên bản **CAN 2.0A** sử dụng định dạng ID dài **11 bit**. Đây là một trong những yếu tố quan trọng để xác định mức độ ưu tiên của thông điệp trong quá trình truyền dữ liệu. Với 11 bit, có thể biểu diễn **2^11 = 2048** ID khác nhau.
- **Khả năng ưu tiên**: Mỗi thông điệp trong mạng CAN đều có một ID xác định mức độ ưu tiên của nó. ID càng thấp, mức độ ưu tiên càng cao.
- **Truyền dữ liệu**: Dữ liệu có thể truyền đi qua bus CAN với kích thước tối đa là **8 byte** mỗi khung. Điều này là đặc trưng của CAN 2.0A, giúp quản lý hiệu quả băng thông và độ trễ.

7.2 CAN 2.0B (Extended CAN)

Đặc điểm chính của CAN 2.0B:

- **29-bit identifier (ID)**: Phiên bản **CAN 2.0B** mở rộng định dạng ID từ **11 bit** trong CAN 2.0A lên **29 bit**. Với 29 bit, có thể biểu diễn **2^29** ID khác nhau, cho phép phân bổ số lượng ID lớn hơn và hỗ trợ các hệ thống phức tạp với nhiều node hơn.
- **Tương thích ngược**: **CAN 2.0B** vẫn tương thích ngược với **CAN 2.0A**, có nghĩa là các node sử dụng **CAN 2.0B** có thể hiểu được và giao tiếp với các node sử dụng **CAN 2.0A**. Các node CAN 2.0B có thể nhận diện giữa các khung dữ liệu tiêu chuẩn và khung dữ liệu mở rộng thông qua bit điều khiển **IDE (Identifier Extension)**.
- **Khả năng truyền dữ liệu**: Giống như CAN 2.0A, **CAN 2.0B** cũng giới hạn khung dữ liệu tối đa là **8 byte**, nhưng sự khác biệt chính nằm ở số lượng ID có thể được sử dụng để định danh các thiết bị trên mạng.

Khung dữ liệu CAN 2.0B bao gồm các trường sau:

- **Start of Frame (SOF)**:
  - Đây là một bit duy nhất đánh dấu bắt đầu của một frame.
- **Identifier (ID)**:
  - **Standard Frame (11-bit ID)**: Đây là phần chứa 11-bit để nhận diện thông điệp.
  - **Extended Frame (29-bit ID)**: 29-bit được chia thành hai phần:
    - **Base ID**: 11-bit giống như trong Standard Frame.
    - **Extended ID**: 18-bit mở rộng để cung cấp tổng cộng 29-bit nhận dạng.
- **Remote Transmission Request (RTR)**:
  - Bit này chỉ định liệu frame là một **Data Frame** hay **Remote Frame**.
  - `RTR = 0`: Đây là **Data Frame**.
  - `RTR = 1`: Đây là **Remote Frame** (yêu cầu dữ liệu từ một node khác).
- **Identifier Extension (IDE)**:
  - Bit này phân biệt giữa **Standard Frame (IDE = 0)** và **Extended Frame (IDE = 1)**.
- **Reserved Bit (r0, r1)**:
  - Những bit dự phòng để phục vụ cho các phiên bản tương lai của chuẩn CAN.
- **Data Length Code (DLC)**:
  - 4-bit chỉ định số lượng byte dữ liệu trong **Data Field** (từ 0 đến 8 byte).
- **Data Field**:
  - Chứa dữ liệu thực tế mà bạn muốn truyền. Độ dài từ 0 đến 8 byte (mỗi byte 8-bit).

- **Cyclic Redundancy Check (CRC)**:
    - Trường này chứa 15-bit CRC và 1-bit CRC Delimiter. CRC được dùng để kiểm tra lỗi trong frame.
- **ACK Slot**:
    - 1-bit dành cho node nhận phát tín hiệu xác nhận (ACK) nếu frame được nhận mà không có lỗi.
- **End of Frame (EOF)**:
    - Kết thúc frame, gồm 7 bit liên tiếp có giá trị "1".
- **Intermission Frame Space (IFS)**:
    - 3-bit dành cho thời gian nghỉ giữa hai frame truyền liên tiếp.

7.3 CAN FD (Flexible Data-rate)

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

Đặc điểm chính của CAN FD:

- **Truyền dữ liệu với tốc độ cao hơn**: **CAN FD** cho phép tốc độ truyền dữ liệu nhanh hơn nhiều so với **CAN tiêu chuẩn**, với tốc độ có thể lên tới **8 Mbps** trong giai đoạn truyền dữ liệu (Data Phase). Trong **CAN tiêu chuẩn**, tốc độ truyền tối đa bị giới hạn ở **1 Mbps**. Điều này giúp tăng đáng kể tốc độ truyền thông cho các hệ thống đòi hỏi xử lý nhanh.
- **Kích thước khung dữ liệu lớn hơn**: Trong khi **CAN tiêu chuẩn** chỉ hỗ trợ tối đa **8 byte** dữ liệu trong mỗi khung, **CAN FD** có thể truyền tới **64 byte** dữ liệu trong một khung. Điều này giúp giảm số lượng khung cần thiết để truyền một lượng lớn dữ liệu, từ đó giảm độ trễ và tăng hiệu suất.
- **Tốc độ truyền linh hoạt**: **CAN FD** sử dụng hai tốc độ truyền khác nhau trong cùng một khung: một tốc độ chậm hơn cho **Arbitration Phase** (giai đoạn tranh chấp quyền gửi), và một tốc độ nhanh hơn cho **Data Phase** (giai đoạn truyền dữ liệu). Điều này giúp đảm bảo tính tương thích và hiệu quả của hệ thống.
- **Tương thích ngược**: **CAN FD** vẫn giữ được khả năng tương thích ngược với các phiên bản CAN cũ như **CAN 2.0A** và **CAN 2.0B**, giúp các hệ thống sử dụng cả CAN tiêu chuẩn và CAN FD có thể hoạt động song song.

Cấu trúc khung dữ liệu CAN FD:

1. **Start of Frame (SOF)**:
    - Một bit đơn, luôn có giá trị `0`, đánh dấu bắt đầu của một khung dữ liệu.
2. **Identifier (ID)**:

    - **Standard Frame (11-bit ID)**: Khung chuẩn với 11-bit định danh.
    - **Extended Frame (29-bit ID)**: Khung mở rộng với 29-bit định danh, tương tự như trong CAN 2.0B.
3. **Extended Identifier (IDE)**:
    - Bit này cho biết khung dữ liệu là chuẩn (Standard Frame) hay mở rộng (Extended Frame).
    - `IDE = 0`: Khung chuẩn (11-bit).
    - `IDE = 1`: Khung mở rộng (29-bit).
4. **Remote Transmission Request (RTR)**:
    - Bit này chỉ có trong CAN 2.0B để yêu cầu dữ liệu từ node khác. Trong CAN FD, **RTR** đã bị loại bỏ.
5. **FD Format (FDF)**:
    - Đây là bit mới được thêm vào trong CAN FD, cho biết khung dữ liệu là **CAN 2.0** hay **CAN FD**.
    - `FDF = 0`: Khung CAN 2.0.
    - `FDF = 1`: Khung CAN FD.
6. **Bit Rate Switch (BRS)**:
    - Bit này cho phép điều chỉnh tốc độ truyền dữ liệu.
    - Nếu `BRS = 1`, tốc độ truyền trong phần Data Phase (phần chứa dữ liệu) sẽ cao hơn tốc độ truyền trong phần Arbitration Phase (phần xác định ưu tiên).
7. **Error State Indicator (ESI)**:
    - Bit này cho biết trạng thái lỗi của node.
    - `ESI = 0`: Node đang trong trạng thái active (hoạt động bình thường).
    - `ESI = 1`: Node đang trong trạng thái passive (chờ phục hồi sau lỗi).
8. **Data Length Code (DLC)**:
    - **CAN FD** vẫn sử dụng trường **DLC** để chỉ định số byte trong **Data Field** (trường dữ liệu). Tuy nhiên, CAN FD cho phép số byte dữ liệu lớn hơn nhiều so với CAN 2.0:
        - **CAN 2.0**: Tối đa 8 byte.
        - **CAN FD**: Tối đa 64 byte.
9. **Data Field**:
    - Đây là trường chứa dữ liệu thực tế.
    - **CAN FD** cho phép truyền tối đa 64 byte dữ liệu (trong khi CAN 2.0 chỉ cho phép 8 byte).
10. **Cyclic Redundancy Check (CRC)**:

- Trường **CRC** trong CAN FD dài hơn so với CAN 2.0 để đảm bảo độ tin cậy khi truyền dữ liệu lớn hơn.
- **CAN 2.0**: CRC là 15-bit.

- **CAN FD**: CRC có thể là 17-bit hoặc 21-bit, tùy thuộc vào độ dài của dữ liệu.

11. **ACK Slot**:
- Khung ACK để các node trên bus xác nhận rằng frame đã được nhận thành công.
- Giống với CAN 2.0, node nhận sẽ ghi bit ACK nếu nhận được khung mà không gặp lỗi.

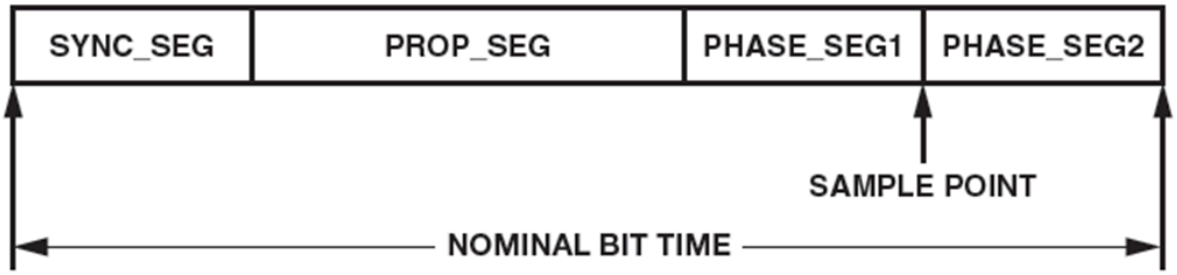
12. **End of Frame (EOF)**:
- Khung kết thúc giống với CAN 2.0, bao gồm 7 bit liên tiếp có giá trị "1".

8. Quá trình cấu hình và thiết lập CAN

Để mạng CAN hoạt động hiệu quả và ổn định, việc **cấu hình và thiết lập** hệ thống đúng cách là rất quan trọng. Các bước cấu hình liên quan đến việc thiết lập **tốc độ baud**, **bộ lọc CAN**, và **kiểm soát lỗi**. Dưới đây là chi tiết về quá trình thiết lập từng phần trong giao thức CAN, kèm theo các khái niệm quan trọng và cách thức hoạt động.

8.1 Thiết lập tốc độ baud trong CAN

Tốc độ baud (baud rate) là tốc độ truyền dữ liệu trên bus CAN, được tính bằng số bit truyền trên giây (bps - bit per second). Việc thiết lập tốc độ baud chính xác là một bước quan trọng vì nó ảnh hưởng trực tiếp đến khả năng truyền dữ liệu và chiều dài của bus.



Các yếu tố ảnh hưởng đến tốc độ baud:

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

- **Bộ chia thời gian (Time Segment)**: là một thành phần quan trọng để đồng bộ hóa và điều chỉnh thời gian truyền thông giữa các node trên bus CAN. Một bit trong mạng CAN được chia thành các **phân đoạn thời gian** (time segment), bao gồm:
    - **Sync Segment**: là đoạn đầu tiên của mỗi bit và có độ dài cố định là **1 time quanta (TQ)**. Đoạn này giúp **đồng bộ hóa** tất cả các node trên bus. Nó đảm bảo rằng tất cả các node đều nhận biết sự bắt đầu của một bit tại cùng một thời điểm. Node nhận sẽ phát hiện ra cạnh thay đổi (rising edge hoặc falling edge) của tín hiệu CAN tại đoạn này để điều chỉnh thời gian của chính nó, đảm bảo đồng bộ với các node khác trên bus.
    - **Propagation Segment**: Đoạn này bù đắp thời gian cần thiết để tín hiệu lan truyền qua bus CAN từ node gửi đến node nhận. Tín hiệu cần thời gian để di chuyển từ một node đến một node khác, và thời gian này phụ thuộc vào **chiều dài của bus** và **tốc độ truyền**. Propagation Segment được cấu hình sao cho nó bao gồm **độ trễ lan truyền** và **thời gian trễ xử lý** của cả phần cứng CAN.
    - **Phase Segment 1 (PS1)** và **Phase Segment 2 (PS2)**: Hai phân đoạn này được sử dụng để **đồng bộ hóa** tín hiệu và bù đắp cho các sai lệch về thời gian hoặc độ trễ nhỏ trong quá trình truyền.
        - **PS1** là đoạn thời gian trước **điểm lấy mẫu**. Đoạn này cho phép node điều chỉnh thời gian đọc tín hiệu để đồng bộ với tín hiệu thực tế trên bus.
        - **PS2** là đoạn thời gian sau **điểm lấy mẫu**. Nó có thể được kéo dài hoặc thu ngắn để bù trừ sự sai lệch nhỏ giữa các node, giữ cho tất cả node đồng bộ với nhau. Đây là đoạn để kết thúc 1 bit.
        - Cả PS1 và PS2 đều có thể được điều chỉnh linh hoạt tùy thuộc vào sự thay đổi thời gian cần thiết để đảm bảo việc lấy mẫu tín hiệu một cách chính xác.
- **Đơn vị của phân đoạn thời gian: Time Quanta (TQ)**
    - **Time Quanta (TQ)** là đơn vị thời gian nhỏ nhất mà một bit trong giao thức CAN được chia thành. Mỗi phân đoạn thời gian trong 1 bit được tính bằng số lượng **TQ**.
    - Mỗi node trên bus CAN sử dụng cùng một đơn vị **TQ**, và tổng số **TQ** trong một bit sẽ quyết định **tốc độ baud** của hệ thống.

Vậy, tổng thời gian của một bit trong giao thức CAN là tổng của các phân đoạn thời gian (**Sync Segment**, **Propagation Segment**, **PS1**, và **PS2**), tính bằng **TQ**:

$$Bit\ Time = Sync\ Segment + Propagation\ Segment + PS1 + PS2$$

Và **tốc độ baud** (bit rate) được tính như sau:

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

8.2 Bộ lọc CAN

Trong CAN, các node có thể nhận rất nhiều thông điệp, nhưng không phải tất cả thông điệp đều liên quan đến tất cả các node. **Bộ lọc CAN** cho phép các node lựa chọn và chỉ nhận những thông điệp cần thiết, dựa trên **ID** của thông điệp hoặc các tiêu chí khác.

Vai trò của bộ lọc CAN:

- **Lựa chọn thông điệp**: Bộ lọc CAN giúp các node lọc ra những thông điệp cần thiết và bỏ qua những thông điệp không liên quan. Điều này giúp giảm tải cho vi điều khiển, vì nó chỉ xử lý những dữ liệu mà nó cần.
- **Giảm băng thông**: Bằng cách chỉ nhận những thông điệp có ID cụ thể, node có thể giảm khối lượng dữ liệu cần xử lý, giúp mạng hoạt động hiệu quả hơn.

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

- **Mask (Mặt nạ)**: Quy định những bit nào trong **ID của thông điệp** cần được kiểm tra.
- **Filter (Bộ lọc)**: Được dùng để so sánh các bit của **ID thông điệp** với giá trị quy định trong bộ lọc. Nếu các bit này khớp với **mask**, thông điệp sẽ được chấp nhận và xử lý.

8.2.1 Mask

Mask là một dãy bit mà các bit có giá trị **1** sẽ được kiểm tra, còn các bit có giá trị **0** sẽ bị bỏ qua. Điều này cho phép chỉ định cụ thể những phần của ID thông điệp mà node sẽ quan tâm, trong khi bỏ qua các phần không quan trọng. Mask giúp xác định phạm vi ID mà node quan tâm.

8.2.2 Filter

Filter là giá trị mà các bit trong ID của thông điệp phải khớp với, dựa trên mask. Các bit được phép kiểm tra thông qua mask sẽ được so sánh với filter. Nếu ID thông điệp trùng khớp với giá trị của bộ lọc (sau khi áp dụng mask), thông điệp sẽ được chấp nhận. Nếu không trùng khớp, thông điệp sẽ bị bỏ qua, node sẽ không xử lý nó.

Ví dụ: Giả sử trong một hệ thống CAN, chúng ta có một node cần nhận thông điệp có **ID** trong khoảng từ **0x100** đến **0x1FF**. Điều này có nghĩa là node chỉ quan tâm đến

các thông điệp có giá trị từ **0x100** đến **0x1FF**, và không quan tâm đến các thông điệp có ID nằm ngoài phạm vi này.

Để đạt được điều này, chúng ta có thể cấu hình bộ lọc CAN như sau:

Cấu hình bộ lọc:

- **Mask (Mặt nạ)**: `0x700` – chỉ kiểm tra **3 bit đầu tiên** của ID.
- **Filter (Bộ lọc)**: `0x100` – chỉ nhận thông điệp có ID bắt đầu bằng **0x001**.

Phân tích cấu hình:

- **Mask** `0x700` (111 0000 0000) có nghĩa là chỉ có **3 bit đầu tiên** của ID thông điệp sẽ được so sánh với **filter**. Các bit khác (bit từ 8 trở xuống) sẽ không được kiểm tra.
- **Filter** `0x100` (001 0000 0000) có nghĩa là node sẽ chấp nhận những thông điệp có **3 bit đầu tiên là 001**, tức là thông điệp có ID nằm trong khoảng từ **0x100** đến **0x1FF**.

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

Minh họa với ví dụ cụ thể:

Giả sử có các thông điệp trên bus CAN với các ID sau: **0x0F0**, **0x100**, **0x180**, **0x200**. Khi sử dụng cấu hình bộ lọc như trên:

- **ID 0x0F0**: Bị bỏ qua vì **3 bit đầu tiên là 000**, không trùng khớp với **001** trong filter.
- **ID 0x100**: Được chấp nhận vì **3 bit đầu tiên là 001**, trùng khớp với filter.
- **ID 0x180**: Được chấp nhận vì **3 bit đầu tiên là 001**, trùng khớp với filter.
- **ID 0x200**: Bị bỏ qua vì **3 bit đầu tiên là 010**, không trùng khớp với filter.