# CAN2.0

## [CẤU TRÚC VÀ CÁC THÀNH PHẦN TRONG BUS CAN](https://nguyenquanicd.blogspot.com/2017/06/cancontroller-area-netwwork-cau-truc-va.html)

### Cấu trúc mạng CAN

Mạng CAN gồm các thành phần cơ bản sau:

|  |
| --- |
|  |
| **Hình 1.** Cấu trúc bus và Node của mạng CAN |

Bus CAN là bus vi sai gồm:

* Dây cáp gồm hai dây riêng biệt được gọi là CAN\_H (CAN High) và CAN\_L (CAN Low). Tùy vào mỗi ứng dụng mà hai dây này có thể là dây xoắn kép hay cáp quang. Ngoài ra, dây cáp còn có vỏ bọc chống nhiễu, vỏ này được nối đến một điện áp tham chiếu hoặc GND.
* CAN\_H có điện áp cao, thường là 0V khi bus không truyền, nhận dữ liệu. CAN\_L có điện áp thấp, thường là +5V khi bus không truyền, nhận dữ liệu. Như đã nói, bus CAN là bus vi sai, khi dữ liệu được truyền trên bus, điện áp trên CAN\_H và CAN\_L sẽ thay đổi, mức logic 0/1 sẽ được xác định căn cứ trên sự chênh lệch (sai lệch) điện áp giữa CAN\_H và CAN\_L.
* Điện trở đầu cuối của đường dây là **120Ω**.
* Node: là một thành phần kết nối đến Bus CAN thông quan 2 dây CAN\_H và CAN\_L. Node này là các bo mạch hoặc module điều khiển.

### Cấu trúc Node

|  |
| --- |
|  |
| **Hình 2.** Cấu trúc Node của mạng CAN |

Mỗi một nút CAN yêu cầu phải có:

* Một **Vi điều khiển** (microcontroller - MCU) kết nối với một **Bộ điều khiển CAN** (CAN – Controller).
* **Bộ điều khiển CAN** sẽ được kết nối với **Bộ truyền-nhận CAN** (CAN – Transceiver) thông qua một đường ra dữ liệu nối tiếp (Tx) và một đường vào dữ liệu nối tiếp (Rx).
* Đường Vref là điện áp ra tham khảo cung cấp một mức điện áp danh định bằng 0.5×Vcc = 0.5×5 = 2.5V.

***Chức năng:***

* **Microcontroller – MCU**: Thực thi các chức năng chính, điều khiển chính của một Node. Thực hiện cấu hình hoạt động cho CAN Controller, phân phối dữ liệu cần truyền đến CAN controller, lấy dữ liệu nhận từ CAN Controller để sử dụng cho hoạt động của Node.
* **CAN Controller**: Thực thi các xử lý về truyền nhận dữ liệu, báo lỗi, tính toán thời gian bit, ... theo chuẩn CAN quy định; phát dữ liệu cần truyền dạng số (theo mức logic 0/1) ra chân TX; nhận dữ liện dạng số qua chân RX.
* **CAN Transceiver**: Hoạt động như bộ chuyển đổi từ tín hiệu số (mức logic 0/1) trên đường TX thành tín hiệu tương tự trên bus CAN và ngược lại, chuyển đổi từ tín hiệu tương tự trên bus CAN (CAN\_H và CAN\_L) thành tín hiệu số trên đường RX

### Tín hiệu trên bus CAN

CAN Transceiver chuyển đổi qua lại giữa tín hiệu số (giao tiếp với CAN Controller) và tương tự (giao tiếp với Bus CAN). Trong giao thức CAN,

* Mức **dominant**, nghĩa là "mức trội" là mức logic 0
* Mức **recessive**, nghĩa là "mức lặn" là mức 1

Điều này vì bus CAN hoạt động theo cơ chế AND-wire, khi một logic 0 và một logic khác cùng được truyền từ CAN Controller qua CAN Transceiver đến Bus CAN thì trên Bus CAN sẽ thể hiện mức điện áp biểu diễn cho bit 0 (bit trội).

Mức điện áp trên Bus CAN là 0V trên CAN\_L và +5V trên CAN\_H khi dùng ở tốc tộ cao 1Mbit/s.

* Đường CAN\_H có mức áp +5V khi ở trạng thái nghỉ và sẽ sụt áp còn +3.5V khi đang hoạt động. Lúc này, mức +3.5V được quy định là mức ‘dominant’ và +2.5V được quy định là mức ‘recessive’.
* Đường CAN\_L có mức áp 0V khi ở trạng thái nghỉ và sẽ tăng lên +1.5V khi hoạt động. Lúc này, mức +1.5V được quy định là mức ‘dominant’ và +2.5V được quy định là mức ‘recessive’.

|  |
| --- |
|  |
| **Hình 5.** Mức điện áp trên đường CAN\_H và CAN\_L khi truyền dữ liệu |

Trạng thái bus dựa vào sự sai lệch áp giữa CAN\_H và CAN\_L. Cụ thể, nếu sai lệch điện áp thấp hơn điệp áp ngưỡng tối thiểu thì đó là mức recessive (mức 1), thường dưới +0.5V. Nếu sai lệch điện áp cao hơn điện áp ngưỡng tối thiểu thì đó là mức dominant (mức 0), thường trên +0.9V. Nếu sai lệch áp rơi vào khoảng từ 0.5V đến ).9V thì không thể phân biệt được mức logic và gây sai dữ liệu.

|  |
| --- |
|  |
| **Hình 6.** Xác định mức logic thông qua sai lệch áp giữa CAN\_H và CAN\_L |

|  |
| --- |
|  |
| **Hình 7.** Một phần cấu trúc của chip CAN transceiver gồm bộ lái bus Driver và bộ nhận từ bus Receiver. |

## CÁC KHÁI NIỆM CƠ BẢN VÀ CẤU TRÚC PHÂN LỚP CỦA CAN

### Các khái niệm cơ bản và lưu ý quan trọng

* **Node** : là những thành phần độc lập có thể xử lý truyền nhận dữ liệu trên Bus CAN, một Node thường có 3 thành phần cơ bản là: Vi điều khiển MCU, bộ điều khiển CAN (CAN Controller) và bộ thu-phát (CAN Transceiver)
* **Thông điệp (Message)**: Thông tin trên bus CAN được gửi dưới dạng các thông điệp có định dạng cố định. Các thông điệp có thể khác nhau nhưng độ dài (số bit trong một thông điệp) là có giới hạn và được giới hạn về độ dài. Khi bus rảnh (IDLE) thì bất kỳ Node nào đều cũng có thể bắt đầu truyền một thông điệp mới. Thông điệp được truyền thông qua 4 loại khung (frame) khác nhau là khung dữ liệu (Data frame), khung điều khiển hay khung yêu cầu (Remote frame), khung báo lỗi (Error frame) và khung báo quá tải (Overload frame)
* **Tốc độ bit (Bit rate):** Tốc độ bit của CAN có thể khác nhau trong các hệ thống khác nhau nhưng trong một hệ thống cho trước thì tốc độ bit đồng nhất và cố định. Tốc độ bit còn tùy thuộc vào chiều dài đường truyền. Tốc độ tối đa có thể lên đến 1 Mbit/s. Sau đây là bảng một số thông số tham khảo thực tế:

|  |  |  |
| --- | --- | --- |
| **Tốc độ bit**  **(Bit Rate –** kbit/s**)** | **Chiều dài bus**  **(Bus Length -** m**)** | **Thời gian bit danh định**  **(Nominal Bit-time -** µs**)** |
| 1000 | 30 | 1 |
| 800 | 50 | 1.25 |
| 500 | 100 | 2 |
| 250 | 250 | 4 |
| 125 | 500 | 8 |
| 62.5 | 1000 | 20 |
| 20 | 2500 | 50 |
| 10 | 5000 | 100 |

* **Tính ưu tiên (Priorities)**: Mỗi khung dữ liệu hoặc khung yêu cầu sẽ chứa một định danh IDENTIFIER (hay số ID). ID sẽ định nghĩa một mức ưu tiên cố định cho thông điệp trong suốt quá trình truy cập bus.
* **Yêu cầu dữ liệu từ xa (Remote Data Request):** Bằng cách gửi một khung yêu cầu (Remote Frame), một Node đang chờ dữ liệu có thể yêu cầu một Node khác gửi một khung dữ liệu (Data Frame) tương ứng với khung yêu cầu đó. Khung dữ liệu và khung yêu cầu tương ứng phải có IDENTIFIER giống nhau.
* **Đa master (Multimaster):** Khi bus rảnh, mỗi Node đều có thể truyền thông điệp nhưng chỉ Node nào truyền thông điệp có mức ưu tiên cao nhất mới chiếm quyền truy cập bus, tức thông điệp của Node đó được truyền trước. Các thông điệp từ các Node khác phải tạm dừng và chờ truyền sau.
* **Giá trị bus (Bus values):** Khi hoạt động, bus có thể mang một trong hai giá trị logic là ‘dominant’ tương ứng với mức thấp – bit 0 hay ‘recessive’ tương ứng mức cao – bit 1. Trong cùng một thời điểm nếu ‘dominant’ và ‘recessive’ cùng được truyền lên bus thì giá trị bus sẽ là ‘dominant’. Đây là cơ chế hoạt động AND-wire của bus.
* **Sự phân xử (Arbitration):** 
  + Bất cứ khi nào bus rảnh, mỗi Node đều có thể bắt đầu truyền thông điệp. Như vậy, nếu có hai hay nhiều Node truyền thông điệp trong cùng một thời gian thì sẽ xảy ra xung đột khi truy cập bus. Để tránh xung đột này, cơ chế phân xử từng bit của IDENTIFIER được sử dụng để đảm bảo thông tin không bị mất và thời gian truyền cũng không bị mất. Thông tin không bị mất là vì Node nào thua khi phân xử thì phải dừng lại, thông điệp của Node đó sẽ được phát lại khi bus rảnh. Thời gian không bị mất là vì sự phân xử thực hiện thời gian thực, Node nào thắng trong sự phân xử sẽ tiếp tục truyền tiếp các bit còn lại chứ không phải dừng để truyền lại từ đầu.
  + Nếu một khung dữ liệu và một khung yêu cầu có IDENTIFIER giống nhau được bắt đầu cùng một thời gian thì khung dữ liệu chiếm ưu thế hơn khung yêu cầu.
  + Mỗi bộ truyền thông điệp lên bus phải giám sát bus để so sánh mức bit mà nó truyền ra và mức bit thể hiện trên bus. Nếu giống nhau, nó tiếp tục truyền bit tiếp theo. Nếu bit truyền ra là reccessive và giá trị giám sát được trên bus là dominant thì Node thua phân xử, phải dừng truyền ngay lập tức từ bit tiếp theo. Nếu bit truyền ra là dominant nhưng giá trị giám sát được là recessive thì đó là lỗi bit.
* **Tính an toàn (Safety):** Để có được sự an toàn cao cho việc truyền/nhận dữ liệu thì giao thức có quy định các biện pháp phát hiện lỗi, báo hiệu lỗi và tự kiểm tra được thực thi cho mỗi nút CAN.
  + Phát hiện lỗi (Error detection): sử dụng cơ chế giám sát (Monitoring – so sánh mức bit được truyền với mức bit trên bus), sử dụng mã kiểm tra CRC – Cyclic Redundancy Check), Chèn bit (Bit stuffing) và kiểm tra khung thông điệp.
  + Hiệu quả phát hiện lỗi: Với việc áp dụng các cơ chế phát hiện lỗi, tất cả các lỗi toonge thể (lỗi xảy ra trên toàn bus) được phát hiện, tất cả các lỗi cục bộ ở các bộ truyền được phát hiện, có khả năng phát hiện đến 5 lỗi ngẫu nhiên khác nhau trong 1 thông điệp, các lỗi nhóm với độ dài nhỏ hơn 15 trong một thông điệp, các lỗi với số lẻ lần được phát hiện.
  + Xác xuất lỗi không phát hiện được nhỏ hơn: (Tốc độ lỗi thông điệp)×4.7×10^(-11)
* **Hạn chế lỗi (Fault Confinement):** Các nút CAN có thể nhận biết các nhiễu ngắn từ các sai cố định. Các nút hỏng có thể được tắt khỏi bus.
* **Sự kết nối (Connection):** Trong bus CAN, số lượng Node nối đến bus là không giới hạn (theo lý thuyết). Trên thực tế, tổng số đơn vị kết nối đến bus bị giới hạn do thời gian trễ và giới hạn về tải trên đường bus.
* **Báo nhận (Acknowledgment):** Tất cả các bộ nhận sẽ kiểm tra tính phù hợp của thông điệp được nhận và sẽ báo ACK nếu thông điệp được nhận đúng và phù hợp.
* **Hoạt động Sleep và Wake-up:** Để giảm công suất tiêu thụ hệ thống, một linh kiện CAN (hoặc một Node) có thể được thiết lập chế độ Sleep để không có bất cứ hoạt động nội nào và ngắt bộ lái bus của nó. Chế độ Sleep có thể được thoát ra với cơ chế wake-up từ bất kỳ hoạt động nào trên bus hoặc các điều kiện nội của hệ thống. Khi wake-up, hoạt động nội của Node được khởi động lại và chờ cho nguồn xung clock ổn định. Sau đó là chờ cho đến khi nó tự đồng bộ lại với bus bằng cách kiểm tra có sự xuất hiện của 7 bit recessive liên tiếp trước khi bật lại bộ lái bus. Để wake-up các Node của hệ thống đang trong chế độ Sleep thì một thông điệp đặc biệt được sử dụng với một ID có ưu tiên thấp nhất có thể.
* **Bộ truyền hay bộ phát (transmitter):** Khi một Node phát đi một thông điệp, Node đó trở thành bộ truyền của thông điệp đó. Node sẽ là bộ truyền cho đến khi nó truyền xong thông điệp này (lúc này bus về trạng thái rảnh) hoặc cho đến khi thua phân xử.
* Bộ nhận hay bộ thu (receiver): một Node là "bộ nhận" của một thông điệp khi nó không là bộ truyền của thông điệp đó và bus không rảnh. Ví dụ, bus đang rảnh (IDLE), lúc này không có thông điệp nào đang lan truyền trên bus, tất cả các Node đều không phải là bộ truyền hoặc bộ nhận. Tại một thời điểm, có 2 Node A và B cùng truyền, lúc này cả A và B đều là bộ truyền, các Node khác trong bus đều là bộ nhận. Sau một khoảng thời gian, Node B thua phân xử và ngưng truyền, Node A thắng phân xử và tiếp tục truyền, lúc này Node B trở thành bộ nhận và Node A là bộ truyền cho đến khi nó truyền xong thông điệp.
* Định tuyến thông tin trong CAN: Trong các hệ thống CAN, một Node không cần sử dụng bất kỳ thông tin nào về cấu hình hệ thống. Điều này làm cho các Node trong một mạng CAN có sự độc lập và linh động cao. Từ đó, CAN mang một số đặc điểm quan trọng sau đây:
  + Tính linh động (Flexibility): Các Node CAN có thể được kết nối thêm vào mạng mà không cần phải thay đổi phần mềm hay phần cứng của bất kỳ Node nào khác và lớp ứng dụng.
  + Định tuyến thông điệp: Mỗi thông điệp (khung dữ liệu hoặc khung yêu cầu) có một ID (IDENTIFIER). ID không chỉ ra đích đến của thông điệp mà chỉ ra ý nghĩa của dữ liệu mà thông điệp đó mang theo. Vì vậy, tất cả các Node trong mạng đều có thể nhận và quyết định có sử dụng nội dung thông điệp đó hay không. Việc quyết định có sử dụng thông điệp đó hay không là nhờ cơ chế LỌC THÔNG ĐIỆP (MESSAGE FILTERING) có ở mỗi Node CAN
  + Đa điểm (Multicast): Như đã nói, bất kỳ Node nào cũng có thể nhận thông điệp. Thông qua cơ chế lọc thông điệp, tại một thời điểm, nhiều Node khác nhau có thể cùng sử dụng và xử lý cùng một thông điệp.
  + Tính đồng nhất dữ liệu (Data Consistency): Một mạng CAN luôn đảm bảo một thông điệp sẽ được nhận đồng thời bởi một hoặc nhiều Node hoặc không có Node nào. Do đó, tính đồng nhất của dữ liệu được đảm bảo bởi cơ chế đa điểm và kiểm soát lỗi.

### Cấu trúc phân lớp của giao thức CAN

Chuẩn CAN Specification 2.0 đã được chuẩn hóa trong ISO 11898, chuẩn CAN quy định chi tiết các chức năng và hoạt động của mạng CAN ở hai lớp Data Link và Physical. Đối với lớp ứng dụng (Application) không được quy định trong chuẩn nhưng hiện nay có hai giao thức phổ biến và CANopen và DeviceNet.

Các chức năng của hai lớp này đều do phần cứng (vi mạch, chip) chuyên dụng tự thực hiện theo cấu hình được nạp vào. Các chức năng lớp Data Link do chip chuyên dụng CAN controller (chip số) hoặc ngoại vị CAN được tích hợp trong vi điều khiển MCU thực hiện. Chức năng lớp vật lý (Physical) do chip CAN transceiver (chip tương tự) thực hiện.

|  |
| --- |
|  |
| **Hình 1.** Cấu trúc phân lớp của giao thức CAN |

## CẤU TRÚC KHUNG DỮ LIỆU VÀ KHUNG YÊU CẦU

### Các loại khung truyền của giao thức CAN

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

* Khung dữ liệu (Data frame): là khung mang dữ liệu từ một bộ truyền đến các bộ nhận. Khung này có vùng để mang các byte dữ liệu.
* Khung yêu cầu hay khung điều khiển (Remote frame): là khung được truyền từ một Node để yêu cầu Node khác truyền khung dữ liệu có ID (IDENTIFIER) trùng với khung yêu cầu.
* Khung báo lỗi (Error frame): là khung được truyền bởi bất kỳ Node nào khi Node đó phát hiện lỗi bus.
* Khung báo quá tải (Overload frame): được sử dụng để tạo thêm độ trễ giữa giữa các khung dữ liệu hoặc khung yêu cầu. Mỗi Node trong bus CAN có thể truyền bất kỳ khi nào nếu phát hiện bus rảnh, nêu một Node nhận quá nhiều dữ liệu, nó có thể dùng khung này để ngăn sự truyền tiếp theo.

Chỉ có khung dữ liệu và khung yêu cầu là có ID, cơ chế phân xử sẽ áp dụng cho hai loại khung này khi chúng được truyền trên bus.

Khung dữ liệu và khung yêu cầu có hai định dạng khác nhau là định dạng chuẩn (Standard) và định dạng mở rộng (Extended).

* Định dạng khung chuẩn sử dụng ID có độ dài 11 bit
* Định dạng khung mở rộng sử dụng ID có độ dài 29 bit

Chuẩn CAN Specification 2.0-Part A (gọi tắt 2.0A) chỉ quy định sử dụng loại khung tiêu chuẩn. Chuẩn CAN Specification 2.0-Part B (gọi tắt 2.0B) sử dụng cả loại khung tiêu chuẩn và khung mở rộng. Như vậy, khi sử dụng CAN controller, bạn nên quan tâm đến việc nó tương thích chuẩn nào.

Ngoài ra, chuẩn CAN còn một loại định dạng thứ 5 là khoảng liên khung (Interframe Spacing). Nó có vai trò tạo khoảng ngăn cách giữa các khung truyền trên bus CAN.

### Khung dữ liệu (Data frame)

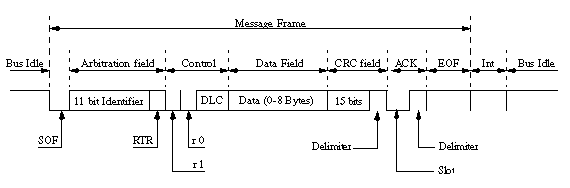
Một khung dữ liệu bao gồm bảy vùng bit khác nhau theo thứ tự là:

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

Mỗi vùng có chức năng nhiệm vụ khác nhau, số lượng bit khác nhau và có quy định cụ thể về mức logic và ý nghĩa các bit. Trong hình minh họa có chú thích rõ số lượng bit từng vùng.

|  |
| --- |
|  |
| **Hình 1.** Định dạng khung dữ liệu chuẩn (Standard Data frame) |

|  |
| --- |
|  |
| **Hình 2.** Định dạng khung dữ liệu mở rộng (Extended Data Frame) |



#### Vùng bắt đầu khung SOF (Start Of Frame)

SOF đánh dấu sự bắt đầu của một khung dữ liệu hoặc khung yêu cầu. Nó chỉ có 1 bit logic 0.

Một Node chỉ được cho phép truyền khi bus rảnh. Khi bus rảnh thì trạng thái trên bus đang là logic 1. Lúc này, nếu có một Node bắt đầu truyền thì nó sẽ truyền SOF trước. Ngay khi xuất hiện cạnh xuống (chuyển từ logic 1 xuống logic 0) của SOF thì tất cả các Node trên bus được đồng bộ theo cạnh này. Nghĩa là, các Node bắt đầu tính toán tính toán thời gian để lấy mẫu các bit trên bus CAN.

Việc đồng bộ theo cạnh gọi là đồng bộ cứng (Hard Synchronization).

#### Vùng phân xử (Arbitration field)

Định dạng vùng phân xử là khác nhau đối với dạng khung chuẩn và dạng khung mở rộng.

* ***Định dạng chuẩn***: vùng phân xử có độ dài 12 bit, bao gồm 11 bit ID và 1 bit RTR. Vùng này có quy định khác nhau đối với chuẩn 2.0A và 2.0B.
* ***Định dạng mở rộng***: vùng phân xử có độ dài 32 bit, bao gồm có 29 bit ID, 1 bit SRR, 1 bit IDE và 1 bit RTR

**IDENTIFIER (ID)**

* Đối với 2.0A:
  + Định dạng chuẩn: 11 bit ID được đánh thứ tự từ .

**Lưu ý**:

là bit LSB.

là 7 bit MSB không được cùng là logic 1.

* Đối với 2.0B:
  + Định dạng chuẩn: 11 bit ID được đánh số thứ tự từ . Việc quy định thứ tự này là vì bản 2.0B có thêm khung mở rộng được thêm vào 18 bit ID từ .

**Lưu ý**:

là 7 bit MSB không được cùng là logic 1.

* + Định dạng mở rộng: gồm 29 bit ID chia làm 2 vùng
    - Base ID: 11 bit từ , tương ứng với vị trí của 11 bit ID ở khung chuẩn)
    - Extended ID: 18 bit từ .

**Bit Remote Transmission Request (RTR)**

Đây là bit dùng để phân biệt khung dữ liệu và khung yêu cầu. Chú ý, bit RTR là nằm sau bit với khung chuẩn và sau với khung mở rộng.

* Bit này luôn bằng 0 đối với khung dữ liệu.
* Bit này luôn bằng 1 đối với khung yêu cầu.

**Bit Substitute Remote Request (SRR)**

* Bit này chỉ có ở khung mở rộng
* Đây là bit logic 1.
* So sánh với khung chuẩn, vị trí của bit này trùng với vị trí của bit RTR nên có tên gọi là bit thay thế (Subsitute) cho bit RTR ở khung chuẩn

Nếu có hai Node cùng truyền, một Node truyền khung dữ liệu chuẩn, một Node truyền khung dữ liệu mở rộng có ID giống nhau thì Node truyền khung chuẩn sẽ thắng phân xử vì đến vị trí sau Base ID, khung chuẩn là bit RTR = 0, còn khung mở rộng là bit SRR = 1. Như vậy, khung chuẩn chiếm ưu thế hơn so với khung mở rộng khi có Base ID như nhau.

**Bit IDentifier Extension (IDE)**

* Đây chính là bit phân biệt giữa loại khung chuẩn và khung mở rộng.
  + IDE bằng logic 0 thì là khung chuẩn.
  + IDE bằng logic 1 thì là khung mở rộng.
* Bit này thuộc:
  + Vùng phân xử nếu là khung mở rộng
  + Vùng điều khiển nếu là khung chuẩn

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

* Nếu bit ngay sau Base ID là bit logic 0 thì đây chắc chắn là khung dữ liệu định dạng chuẩn.
* Nếu bit ngay sau Base ID là bit logic 1 thì đây là khung yêu cầu định dạng chuẩn hoặc khung mở rộng (khung dữ liệu hoặc khung yêu cầu). Node sẽ nhận tiếp bit tiếp theo là bit IDE để phân biệt.
  + Nếu IDE bằng 0 thì bit trước đó là bit RTR và đây là khung yêu cầu dạng chuẩn
  + Nếu bit IDE bằng 1 thì bit trước đó là bit SRR và đây là khung mở rộng nhưng chưa biết là khung dữ liệu hay khung yêu cầu. Node sẽ nhận tiếp Extended ID và đến bit RTR mới phân biệt được.

#### Vùng điều khiển (Control Field)

|  |
| --- |
|  |
| **Hình 3.** Định dạng vùng điều khiển |

Định dạng vùng này ở khung chuẩn và mở rộng là khác nhau.

* Khung chuẩn gồm IDE, r0 và DLC (Data Length Code).
* Khung mở rộng gồm r1, r0 và DLC.

**Bit IDE**

Đã trình bày ở trên.

**Bit r1 và r0:**

* Đây là hai bit dự trữ.
* Tuy hai bit này phải được truyền là bit logic 1 bởi bộ truyền nhưng bộ nhận không qua tâm đến giá trị 2 bit này. Bộ nhận có thể nhận được các tổ hợp 00, 01, 10 hoặc 11 của r1 và r0 nhưng không coi đó là lỗi mà bỏ qua và nhận thông điệp bình thường.

**Data Length Code (DLC)**

* Có độ dài 4 bit quy định số byte của vùng dữ liệu của khung dữ liệu
* Chỉ được mang giá trị từ 0 đến 8 tương ứng với vùng dữ liệu có từ 0 đến 8 byte dữ liệu. Như vậy, khung dữ liệu có thể không có byte dữ liệu nào khi DLC = 0.

**Lưu ý:** Giá trị lớn hơn 8 không được phép sử dụng.

|  |
| --- |
|  |
| **Hình 4.** Bảng giá trị của DLC (r = recessive = 1, d = dominant = 0) |

#### Vùng dữ liệu (Data field)

Vùng này có độ dài từ 0 đến 8 byte tùy vào giá trị của DLC ở vùng điều khiển.

Mỗi byte có 8 bit dữ liệu với bit MSB được truyền trước.

#### Vùng CRC

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

|  |
| --- |
|  |
| **Hình 5.** Vùng CRC |

**CRC Sequence**: gồm 15 bit CRC tuần tự. Mọi tính toán cho CRC sequence thực chất là phép chia đa thức (hay chia bằng các bit nhị phân) và đều dùng modulo-2.

Ta thực hiện tính toán cho CRC sequence như sau:

* Đa thức sinh (đa thức chia) là:

H(X) = X^15 + X^14 + X^10 + X^8 + X^7 + X^4 + X^3 + 1

* Đa thức của chuỗi bit được kiểm tra CRC (đa thức bị chia) nằm từ vùng SOF đến vùng dữ liệu, đa thức chia không tính đến các bit được chèn thêm (quy luật chèn thêm bit sẽ được trình bày sau). Gọi đa thức bị chia là:

Z(X)

* Dịch trái đa thức bị chia 15 bit, tức là thực hiện phép nhân đa thức sau:

X^15.Z(X)

* Lấy đa thức bị chia đã được dịch trái chia cho đa thức sinh:

X^15.Z(X)/H(X)

* Gọi số dư của phép chia trên là R(X), R(X) chính là 15 bit tuần tự chứa trong chuỗi CRC.
* Sau đây là đoạn mã giả minh họa cách tính toán chuỗi CRC

CRC\_RG = 0; // initialize shift registerREPEAT CRCNXT = NXTBIT EXOR CRC\_RG(14);

CRC\_RG(14:1) = CRC\_RG(13:0); // shift left by

CRC\_RG(0) = 0; // 1 position

IF CRCNXT THEN

CRC\_RG(14:0) = CRC\_RG(14:0) EXOR (4599hex);

ENDIF

UNTIL (CRC SEQUENCE starts or there is an ERROR condition)

Mã kiểm tra CRC phù hợp nhất cho các khung mà chuỗi bit được kiểm tra có chiều dài dưới 127 bit, mã này thích hợp cho việc phát hiện các trường hợp sai nhóm (burst error). Ở đây, tổng bit từ vùng SOF đến vùng dữ liệu tối đa là 83 bit (khung định dạng chuẩn) và 103 bit (khung định dạng mở rộng).

Bộ nhận cũng sẽ tính toán CRC như bộ truyền khi đã nhận dữ liệu và so sánh kết quả đó với CRC sequence mà nó đã nhận được, nếu khác nhau tức là đã có lỗi, nếu giống nhau tức là đã nhận đúng từ vùng SOF đến vùng dữ liệu.

**CRC delimiter***:* theo ngay sau CRC sequence, nó là một bit logic 1 làm nhiệm vụ phân cách vùng CRC với vùng ACK.

#### Vùng ACK

Vùng báo nhận hay vùng ACK có độ dài 2 bit và bao gồm hai phần là ACK Slot và phần phân cách ACK Delimiter.

|  |
| --- |
|  |
| **Hình 6.** Vùng ACK |

**ACK Slot**: có độ dài 1 bit, một Node truyền dữ liệu sẽ thiết lập bit này là logic 1. Khi một hoặc nhiều Node nhận chính xác giá trị thông điệp (không lỗi và so sánh CRC sequence trùng khớp) thì nó sẽ báo lại cho bộ truyền bằng cách truyền ra một bit logic 0 ngay vị trí ACK slot để ghi đè lên bit logic 1 của bộ truyền.

**ACK Delimiter**: có độ dài 1 bit, nó luôn là một bit logic 1. Như vậy, ta thấy rằng ACK slot luôn được đặt giữa hai bit logic 1 là CRC delimiter và ACK delimiter.

#### Vùng kết thúc khung EOF (End Of Frame)

Vùng EOF là vùng thông báo kết thúc một khung dữ liệu hay khung yêu cầu. Vùng này gồm 7 bit logic 1.

### Khung yêu cầu (Remote frame)

Một Node hoạt động như một bộ nhận đang mong muốn một dữ liệu, để nhận dữ liệu này từ một Node nguồn thì bộ nhận sẽ gửi một khung yêu cầu. Khung yêu cầu này có tác dụng thông báo cho Node nguồn biết để Node này truyền dữ liệu.

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

|  |
| --- |
|  |
| **Hình 7.** Khung yêu cầu dạng chuẩn |

|  |
| --- |
|  |
| |  | | --- | | **Hình 8.** Khung yêu cầu dạng mở rộng | |

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

* Khung yêu cầu không có vùng dữ liệu (Data Field). Chính vì vậy, khung yêu cầu không phụ thuộc vào giá trị của mã chỉ độ dài dữ liệu DLC, DLC có thể nhận bất kỳ giá trị nào từ 0 đến 8. Nhưng giá trị DLC của khung yêu cầu lại chỉ ra độ dài dữ liệu trong khung dữ liệu tương ứng sẽ trả về, tức DLC của khung dữ liệu trả về giống DLC của khung yêu cầu đã gửi. Và khung dữ liệu trả về có số byte bằng với giá trị DLC này.
* Bit RTR (thuộc vùng phân xử) của khung yêu cầu là bit logic 1. Nó khác với khung dữ liệu vì ở khung dữ liệu nó là một bit logic 0.

### Khung báo lỗi (Error frame)

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

|  |
| --- |
|  |
| **Hình 1.** Khung báo lỗi |

#### Cờ lỗi (Error Flag)

Cờ lỗi có 2 dạng là cờ lỗi chủ động (Active Error Flag) và cờ lỗi bị động (Passive Error Flag)

* Cờ lỗi chủ động có 6 bit logic 0 liên tiếp.
* Cờ lỗi bị động có 6 bit logic 1 liên tiếp.

Một Node đang trong trạng thái chủ động với lỗi (Error Active Station) sẽ truyền cờ lỗi chủ động khi phát hiện một điều kiện lỗi. Dạng cờ lỗi chủ động sẽ vi phạm luật chèn bit (bit stuffing) được quy định bởi chuẩn này. Luật chèn bit quy định cứ sau 5 bit liên tiếp có mức logic giống nhau thì bộ truyền phải tự động chèn thêm một bit khác mức logic của 5 bit trước đó. Luật này áp dụng từ vùng SOF đến hết vùng CRC của khung dữ liệu và khung yêu cầu. Đồng thời, dạng cờ lỗi chủ động cũng không giống bất kỳ định dạng nào của vùng ACK hay EOF.

Tất cả các Node phát hiện thấy một điều kiện lỗi đều phải truyền cờ báo lỗi. Vì có thể có nhiều Node cùng phát hiện lỗi và truyền cờ lỗi nên các cờ lỗi được phát ra bus sẽ đè hay chồng lên nhau tạo thành vùng chồng chập các cờ lỗi. Với cờ lỗi chủ động, vùng chồng chập này sẽ tạo ra một chuỗi biT logic 0 có số bit tối thiểu là 6 và tối đa là 12 bit.

Một Node ở trong trạng thái lỗi bị động (Passive Error Station) sẽ cố gắng báo hiệu bằng một cờ lỗi bị động khi nó phát hiện điều kiện lỗi. Cờ lỗi bị động có thể bị ghi đè bởi cờ lỗi chủ động được phát từ các Node ở trạng thái chủ động với lỗi. Vì vậy, Node truyền cờ lỗi bị động sẽ chờ cho đến khi phát hiện được 6 bit liên tiếp cùng cực tính từ lúc bắt đầu phát cờ lỗi bị động của nó. Việc truyền cờ lỗi bị động được xem là hoàn thành khi 6 bit bằng nhau liên tiếp đã được phát hiện.

#### Vùng phân cách khung báo lỗi (Error Delimiter)

Vùng này gồm 8 bit recessive liên tiếp.

Sau khi truyền xong cờ lỗi, các Node bắt đầu phát các bit recessive và phải giám sát cho đến khi bắt đầu nhận được một bit recessive thì bit này được tính là bit đầu tiên của vùng phân cách khung báo lỗi. Lúc này, các Node tiếp tục phát thêm 7 bit recessive nữa để hoàn thành vùng này.

#### Giải thích về sự truyền khung báo lỗi

Khung báo lỗi được phát theo dạng nào là tùy vào trạng thái lỗi hiện tại của Node. Trạng thái lỗi của Node sẽ được trình bày trong 1 bài khác.

Khung báo lỗi được phát khi nào?

* Bất cứ khi nào một Node phát hiện điều kiện lỗi bus khi khung dữ liệu hoặc khung yêu cầu đang được phát thì nó sẽ phát ngay khung báo lỗi của nó. Khung báo lỗi được phát có thể ngắt sự truyền ngay lập tức hoặc có thể không gây ngắt sự truyền.
* Chú ý, trường hợp phát hiện lỗi CRC thì việc truyền cờ báo lỗi phải sau vùng phân cách ACK (ACK DELIMITER) trừ khi đã có một cờ lỗi khác đã phát trước đó.

Khi nào khung báo lỗi gây ngắt sự truyền hiện tại trên bus và khi nào không?

* Khi khung báo lỗi được phát ra là loại khung báo lỗi chủ động vì cờ lỗi chủ động có 6 bit dominant liên tiếp sẽ phá vỡ các quy định về việc truyền dữ liệu như luật chèn bit hoặc phá vỡ định dạng chung của các vùng cấu tạo nên khung dữ liệu hoặc khung yêu cầu. Vì vậy, việc truyền một cờ lỗi chủ động từ bất kỳ Node nào cũng sẽ được tất cả các Node trong mạng phát hiện được
* Khi khung báo lỗi được phát ra là loại khung báo lỗi bị động và được phát ra bởi chính bộ truyền thông điệp trên bus. Một bộ truyền đang ở trạng thái lỗi bị động phát hiện lỗi bus thì nó sẽ ngừng truyền và việc phát 6 bit recessive liên tiếp sẽ vi phạm luật chèn bit và báo hiệu cho các Node khác.
* Một hoặc nhiều bộ nhận trong trạng thái lỗi bị động khi phát hiện lỗi sẽ cố gắng phát khung báo lỗi bị động lên bus. Nếu đó là lỗi cục bộ của riêng của các Node này và không có Node nào truyền khung báo lỗi chủ động thì sự truyền vẫn tiếp tục bởi vì khung báo lỗi bị động toàn bit recessive nên không ảnh hưởng đến dữ liệu đang truyền trên bus. Lúc này, các Node phát khung báo lỗi bị động sẽ ngừng nhận dữ liệu và gí giát bus để chờ nhận được 6 bit liên tiếp bằng nhau như quy định trước khi phát tiếp các bit phân cách khung báo lỗi.

Tại sao độ dài vùng chồng chập cờ lỗi chủ động là từ 6 đến 12 bit?

* Giả sử tại thời điểm phát hiện lỗi bus, đây là một lỗi toàn cục, tất cả các Node trên bus đều đồng thời nhận ra và phát khung báo lỗi thì các cờ lỗi chủ động chồng chập lên nhau và có độ dài ngắn nhất là 6 bit.
* Nếu một lỗi xuất hiện nhưng chỉ có một hoặc một vài Node nhận ra thì chỉ các các Node này phát khung báo lỗi. Các Node còn lại chưa giám sát bus thấy có 6 bit dominant liên tiếp của cờ lỗi chủ động trên bus thì ngay lập tức phát hiện đây là một lỗi và bắt đầu phát khung báo lỗi của nó. Lúc này sự chồng chập các cờ lỗi chủ động sẽ là 6 bit dominant của các Node phát trước đó cộng với 6 bit dominant của các Node phát sau này là 12 bit

Việc khung báo lỗi bị động bị ghi đè bởi khung báo lỗi chủ động?

* Đối với các trạng phát khung báo lỗi chủ động, trường hợp này sẽ không nhận ra có khung báo lỗi bị động và xử lý bình thường.
* Đối với Node phát khung báo lỗi bị động, như đã trình bày, khi phát cờ lỗi bị động, trong trường hợp này bị ghi đè, nó giám sát thấy 6 bit dominant. Đây chính là là 6 bit cùng cực (cùng mức logic) thì coi như nó đã phát xong cờ báo lỗi. Lúc này nó bắt đầu gửi các bit recessive của vùng phân tách khung báo lỗi và chờ đến khi phát hiện bit recessive đầu tiên thì tiếp tục truyền 7 bit recesssive và kết thúc việc truyền khung báo lỗi cùng với các Node truyền khung báo lỗi chủ động.

### Khung báo quá tải (Overload frame)

#### Vai trò và cấu trúc

Khung báo quá tải được các Node trong mạng CAN sử dụng để tạo thêm độ trễ giữa các khung dữ liệu và khung điều khiển. Một Node có thể phát nhiều khung báo quá tải liên tiếp nhau để tạo ra độ trễ bus phù hợp.

|  |
| --- |
|  |
| **Hình 1.** Khung báo quá tải (Overload frame) |

Khung báo quá tải có cấu tạo gồm hai vùng là:

* Vùng chồng lấn các cờ báo quá tải (superposition of Overload flags) là vùng các Node trên bus cùng phát cờ báo quá tải (Overload flag). Vùng này tương tự như ở khung báo lỗi chủ động với độ dài không cố định.
* Cờ báo quá tải gồm 6 bit dominant
* Vùng phân cách khung báo quá tải (Overload delimiter) gồm 8 bit recessive

Khi gặp điều kiện quá tải thì một Node phát khung báo quá tải ở các vị trí sau:

* Ngay sau EOF của một khung dữ liệu hoặc khung yêu cầu. Nếu trên bus đang truyền khung dữ liệu hoặc khung yêu cầu, một Node phải nhận và chờ đến hết vùng EOF mới được phép phát khung báo quá tải.
* Ngay sau vùng phân cách (DELIMITER) của một khung báo lỗi hoặc một khung báo quá tải khác.

Có thể thấy khung báo quả tải và khung báo lỗi chủ động có định dạng giống nhau nhưng hai loại khung này sẽ không bị các Node hiểu nhầm vì vị trí phát chúng là khác nhau.

* Vị trí phát khung báo lỗi là bất cứ khi nào phát hiện lỗi bus khi đang truyền khung dữ liệu và khung yêu cầu. Vì vậy, khi một Node đang giám sát bus, nó phát hiện định dạng như đã mô tả ở trên trong khi khung dữ liệu hay khung yêu cầu chưa truyền xong (chưa hết EOF) thì đó là khung lỗi chủ động
* Vị trí phát khung báo quá tải là sau EOF và sau vùng phần cách của một khung báo lỗi hoặc báo quá tải. Vì vậy, các Node phát hiện định dạng như đã mô tả ở trên ở các vị trị này thì đó là khung báo quá tải.

Trong điều kiện hoạt động không xuất hiện sự quá tải tại các Node bus, sau các vùng trên phải là khoảng liên khung (Interframe spacing, được trình bày ở cuối bài này). Vì vậy, việc phát khung báo quá tải ở vị trí này sẽ không giống định dạng của khoảng liên khung và tất cả các Node khác trong mạng dễ dàng nhận biết đây là khung báo quá tải và cũng bắt đầu truyền cờ báo quá tải của chúng chồng lấn với cờ báo quá tải hiện tại đang phát.

#### Điều kiện báo quá tải

Một Node có thể phát khung này khi gặp một trong các điều kiện sau:

* Điều kiện 1: Do điều kiện hoạt động nội bộ (cục bộ) của bộ nhận, một bộ nhận có thể không xử lý kịp các thông điệp được gửi đến có thể phát khung báo quá tải để tạm dừng truyền tiếp các khung dữ liệu và khung yêu cầu.
* Điều kiện 2: Phát hiện bit dominant (bit 0) tại vị trí bit thứ nhất và bit thứ hai của vùng INTERMISSION trong khoảng liên khung. Vùng INTERMISSION được quy định gồm 3 bit recessive nhưng Node lại phát hiện bit dominant xuất hiện ở vị trí bit thứ nhất hoặc thứ 2 của vùng này thì nó phải phát khung báo quá tải để ngăn chặn sự truyền.
* Điều kiện 3: Nếu một Node phát hiện một bit dominant ở vị trí bi thứ 8 (bit cuối cùng) của ở vùng DELIMITER của khung báo lỗi hay khung báo quá tải, nó sẽ truyền ngay một khung báo quá tải. Chú ý, trong trường hợp này các Node sẽ xem khung phát trên bus là khung báo quá tải chứ không phải là khung báo lỗi chủ động cho dù định dạng hai loại giống nhau. Vì vậy, các Node sẽ không coi đây là điều kiện lỗi và không tăng giá trị bộ đếm lỗi, đây là bộ đếm được thiết kế chạy tự động trong các phần cứng chip CAN controller để giám sát trạng thái lỗi của Node.

Việc phát khung báo quá tải trên bus xảy ra như thế nào?

* Phần lớn việc khởi tạo phát khung báo quá tải là từ hoạt động nội tại của một Node (điều kiện 1).
* Một Node bắt đầu phát cờ báo quá tải (phát bit dominant) phá vỡ cấu trúc cố định thông thường của vùng INTERMISSION.
* Các Node khác trong mạng phát hiện thấy có bit dominant ở bit thứ 1 và thứ 2 ở vị trí đáng lẽ là các bit recessive của vùng INTERMISSION (điều kiện 2). Lúc này, tất cả các Node còn lại sẽ bắt đầu phát cờ báo quá tải chồng lấp lên nhau.
* Tất cả các Node, sau khi phát đủ 6 bit dominant của cờ báo quá tải sẽ chờ cho đến khi phát hiện 1 bit recessive đầu tiên, sau đó phát tiếp 7 bit recessive nữa để hoàn thành vùng phân cách của khung báo quá tải.
* Nếu một Node trong mạng vẫn cần làm trễ đường truyền thì nó có thể bắt đầu phát cờ báo quá tải ở bit thứ 8 của vùng phân cách khung quá tải hoặc sau bit thứ 8 này.
* Các Node khác phát hiện thấy bit thứ 8 của vùng phân cách khung báo quá tải là dominant (điều kiện 3) thì bắt đầu phát cờ báo quá tải cúa nó ở bit tiếp theo.

|  |
| --- |
|  |
| **Hình 2.** Ví dụ về sự truyền khung báo quá tải (Overload Frame) |

#### Khoảng liên khung (Interframe Space)

#### Vai trò và cấu trúc

Khoảng liên khung dùng để phân tách giữa các khung dữ liệu và khung yêu cầu với khung vừa phát trước đó. Khung trước đó có thể là khung dữ liệu, khung yêu cầu, khung báo lỗi hoặc khung báo quá tải.

Khoảng liên khung không bao giờ đặt ngay trước các khung báo lỗi và khung báo quá tải vì:

* Khung báo lỗi phát ngay khi phát hiện lỗi trên bus lúc đang truyền khung dữ liệu và khung mở rộng nên trước nó không thể là một khoảng liên khung.
* Khung báo quá tải thì phát chiếm chỗ vị trí của khoảng liên khung, tức là phát ngay sau EOF hoặc sau vùng phân cách của một khung báo lỗi hoặc khung báo quá tải trước đó nên trước nó cũng không thể là khoảng liên khung
* Mặt khác, sau một khoảng liên khung là trạng thái bus IDLE (bus rảnh) và đây không phải là vị trí phát khung báo lỗi hay khung báo quá tải

Nhiều khung báo quá tải liên tiếp nhau thì sẽ truyền liên tục như đã trình bày ở trên chứ không bị phân tách bởi khoảng liên khung.

Có hai cấu trúc khoảng liên khung như sau:

* Đối với Node **không** trong trạng thái lỗi bị động (passive error) hoặc Node là bộ nhận và vừa mới nhận xong một thông điệp trước đó thì khoảng liên khung gồm vùng INTERMISSION và BUS IDLE

|  |
| --- |
|  |
| **Hình 2.** Định dạng khoảng liên khung (Interframe Space) cho Node không trong trạng thái lỗi bị động hoặc là bộ nhận vừa nhận xong thông điệp |

* Đối với Node trong trạng thái lỗi bị động và vừa mới truyền xong một thông điệp ngay trước đó thì khoảng liên khung gồm vùng INTERMISSION, vùng SUSPEND TRANSMISSION và vùng BUS IDLE

|  |
| --- |
|  |
| Hình 3. Định dạng khoảng liên khung cho Node đang trong trạng thái lỗi bị động và vừa mới truyền xong khung thông điệp trước đó |

#### Cấu tạo các vùng

*Vùng INTERMISSION là vùng ngưng truyền* chứa 3 bit recessive liên tiếp. Trong vùng này, không Node nào được phép bắt đầu truyền khung dữ liệu hoặc khung điều khiển. Chỉ một hoạt động được phép thực hiện là báo hiệu quá tải bằng cách truyền khung quá tải.

Chú ý, như đã trình bày ở trên, việc xuất hiện bit dominant ở vị trí bit thứ 1 hoặc thứ 2 của vùng INTERMISSION sẽ được xem như sự xuất hiện của khung báo quá tải.

Đối với bit thứ 3 của vùng INTERMISSION, nếu một bit dominant xuất hiện thì đó xem như là SOF của khung dữ liệu hoặc khung yêu cầu. Như vậy, một Node có thể bắt đầu truyền thông điệp mới ngay từ bit thứ 3 của vùng INTERMISSION mà không cần chờ đến khi vùng này kết thúc. Lúc này, nếu có các Node khác trong mạng cũng có thông điệp cần truyền thì sau khi phát hiện bit dominant ở vị trí bit thứ 3 của INTERMISSION, các Node này cũng sẽ bắt đầu phát thông điệp của nó bắt đầu từ bit đầu tiên của vùng IDENTIFIER. Tiếp theo, sự phân xử bus sẽ quyết định Node nào thắng và được truyền tiếp.

*Vùng bus rảnh (BUS IDLE)* là vùng có độ dài không xác định. Lúc này, bus được xem là tự do, bất kỳ Node nào có thông điệp cần truyền đều có thể phát bắt đầu với SOF.

*Vùng cấm truyền (SUSPEND TRANSMISSION)* là vùng gồm 8 bit recessive được thêm vào dành cho Node ở trạng thái bị động và vừa truyền xong mộ thông điệp. Sau thông điệp vừa truyền, Node này phải truyền 3 bit recessive của vùng INTERMISSION, tiếp theo phải truyền 8 bit recessive trước khi bắt đầu được phép truyền một thông điệp khác hoặc ghi nhận trạng thái bus rảng (BUS IDLE). Như vậy, nếu giám sát thấy sự xuất hiện của một bit dominant ở bit thứ 3 của INTERMISSION hoặc trong vùng SUSPEND TRANSMISSION thì đó là một SOF nhưng Node này không được phép phát thông điệp của nó như cách đã nói phía trên mà Node này sẽ trở thành bộ nhận thông điệp đang phát bởi các Node khác trên bus.

Có thể thấy một Node đang ở trong trạng thái lỗi bị động, sau khi truyền xong một thông điệp sẽ phải chịu một khoảng cấm truyền và trở nên kém ưu tiên hơn các Node khác trong mạng CAN.

## QUY ĐỊNH VỀ TRẠNG THÁI LỖI CỦA CAN NODE VÀ TÍNH HỢP LỆ CỦA THÔNG ĐIỆP

### Các trạng thái lỗi của CAN Node

Để giám sát và kiểm soát lỗi, mỗi Node đều tự động xác định trạng thái của chính nó dựa trên số lượng lỗi mà nó phát hiện được. Tại một thời điểm, một Node có thể thuộc một trong ba trạng thái sau:

* *Error active - Trạng thái chủ động với lỗi (gọi tắt là trạng thái lỗi chủ động)*. Đây là trạng thái hoạt động thông thường và tốt nhất của một Node. Node có thể tham gia truyền nhận dữ liệu trên bus một cách bình thường, nếu phát hiện lỗi thì sẽ phát khung báo lỗi với cờ báo lỗi chủ động (ACTIVE ERROR FLAG). Khi cờ báo lỗi chủ động được phát thì đường truyền bị ngắt ngay và các Node đều nhận biết được sự báo lỗi này vì cờ báo lỗi chủ động gồm 6 bit dominant sẽ phá vỡ cấu trúc thông thường của khung dữ liệu hoặc khung yêu cầu và không tuân theo luật chèn bit.
  + *Error warning - Trạng thái cảnh báo lỗi* là trạng thái lỗi chủ động nhưng Node CAN lúc này giám sát thấy số lượng lỗi phát hiện được quá nhiều đến ngưỡng cần phải báo động để Node có cơ chế xử lý thích hợp. Node trong trạng thái này hoạt động như trong trạng thái lỗi chủ động. Đây là trạng thái không được quy định trong chuẩn CAN Specification 2.0 nhưng hều hết các CAN controller hiện tại đều có.
* *Error Passive - Trạng thái bị động với lỗi (gọi tắt là trạng thái lỗi bị động)*. Một Node ở trạng thái này sẽ có một vài hoạt động bị giới hạn và kém ưu tiên hơn các Node ở trạng thái lỗi chủ động. Cụ thể, trong khi truyền nhận, nếu phát hiện một lỗi bus thì Node sẽ truyền khung báo lỗi bị động với cờ báo lỗi bị động (PASSIVE ERROR FLAG) toàn là bit recessive nên kém ưu tiên và thông điệp đang truyền có thể vẫn truyền tiếp. Mặt khác nó có thẻ bị ghi đè bởi các Node phát cờ báo lỗi chủ động. Một hạn chế nữa là, Node trong trạng thái lỗi bị động, sau khi vừa truyền xong một thông điệp thì phải chờ thêm một khỏng thời gian của vùng SUSPEND TRANSMISSION trước khi được phép truyền tiếp. Trong khoảng thời gian chờ này, nếu có Node khác truyền dữ liệu thì nó trở thành bộ nhận.
* *Bus-off - Trạng thái ngắt khỏi bus CAN (gọi tắt là trạng thái ngắt bus)* là trạng thái Node không được phép có bất cứ táng động nào trên bus. Bộ lái ngõ ra bus, nằm trong chip CAN transceiver, bị ngắt.

### Cách xác định trạng thái lỗi

Mỗi Node sẽ có 2 bộ đếm được dùng để xác định trạng thái lỗi của Node là:

* Bộ đếm lỗi truyền (TRANSMIT ERROR COUNT)
* Bộ đếm lỗi nhận (RECEIVE EROR COUNT)

Hai bộ đếm lỗi này được tích hợp trong phần cứng CAN controller và chạy tự động khi Node CAN hoạt động.

Giá trị của hai bộ đếm lỗi là bộ đếm lỗi truyền (Tx\_count) và bộ đếm lỗi nhận (Rx\_count) sẽ quyết định Node đang trong trạng thái nào. Các khoảng giá trị của chúng được quy định như sau:

* Trạng thái lỗi chủ động ‘Error active’ nếu Tx\_count ≤ 127 và Rx\_count ≤ 127.
* Trạng thái lỗi bị động ‘Error passive’ nếu Tx\_count > 127 hoặc Rx\_count > 127 nhưng Tx\_count ≤ 255.
* Trạng thái ngắt bus ‘Bus off’ nếu Tx\_count > 255.
* Trạng thái cảnh báo lỗi 'Error Warning' nếu Tx\_count >= 96 hoặc Rx\_count >= 96. Đây là trạng thái không được quy định trong CAN Specification 2.0 nhưng được tách ra dựa trên một chú ý (Note) trong chuẩn này. Cụ thể, giá trị của một bộ đếm lỗi lớn hơn "khoảng" 96 sẽ chỉ ra việc bus bị nhiễu nặng và có thể cần kiểm tra điều kiện này.

|  |
| --- |
|  |
| Hình 1. Các trạng thái của một CAN NODE |

### Các loại lỗi mà một Node CAN có thể phát hiện

Có 5 loại lỗi khác nhau và chúng không loại trừ lẫn nhau, nghĩa là một lỗi xảy ra có thể thuộc hơn 2 loại.

1. Lỗi bit (BIT ERROR) Một Node khi gửi một bit lên bus thì phải giám sát bit đó. Một lỗi bit được phát hiện ngay trong khoảng thời gian truyền bit đó nếu giá trị được lấy mẫu khi giám sát khác với giá trị bit được truyền ra. Nghĩa là truyền bit dominant nhưng giám sát thấy recessive và ngược lại. Ngoại trừ một số trường hợp dù giá trị gửi ra và giá trị bit giám sát khác nhau nhưng không được xem là lỗi bit, cụ thể như sau:
   1. Gửi một bit recessive trong vùng ACK SLOT nhưng giám sát thấy một bit dominant.
   2. Node gửi cờ báo lỗi bị động nhưng giám sát thấy có bit dominant xuất hiện
2. Lỗi chèn bit (STUFF ERROR) Một Node giám sát thấy chuỗi 5 bit liên tiếp giống nhau thì sẽ kiểm tra bit thứ 6 có khác cực với 5 bit trước hay không. Nếu không khác thì đây là lỗi chèn bit vì sau 5 bit dominant liên tiếp phải được chèn 1 bit recessive, sau 5 bit recessive liên tiếp phải được chèn 1 bit dominant.
3. Lỗi CRC (CRC ERROR) CRC SEQUENCE đi kèm theo thông điệp được tính bởi bộ truyền. Các bộ nhận phải tính lại chuỗi CRC giống với cách tính của bộ truyền. Kết quả tính toán này sẽ được dùng để so sánh với chuỗi CRC từ bộ truyền. Nếu hai chuỗi khác nhau thì đó là lỗi CRC.
4. Lỗi định dạng (FORM ERROR) Một Node giám sát thấy các vùng có định dạng bit cố định (các vùng quy định rõ mức logic cố định của bit là 1 hoặc 0) nhưng chứa 1 hoặc nhiều bit vi phạm quy định thì đó là lỗi định dạng. Ngoại trừ:
   1. Một bộ nhận phát hiện thấy một bit dominant ở bit cuối cùng vùng EOF (bit thứ 7 của EOF) thì không xem đây là lỗi định dạng
5. Lỗi ACK (ACKNOWLEDGMENT ERROR) là lỗi được phát hiện bởi bộ truyền khi nó không thấy một bit dominant xuất hiện tròng vùng ACK SLOT.

|  |
| --- |
|  |
| Hình 2. Các vùng được giám sát lỗi |

Một Node phát hiện thấy một trong các lỗi trên thì phải phát cờ báo lỗi chủ động hoặc bị động tùy vào trạng thái hiện tại của Node.

Đối với loại lỗi BIT, lỗi chèn bit, lỗi định dạng và lỗi ACK thì việc truyền cờ báo lỗi bắt đầu ngay ở bit kế tiếp ngay sau khi phát hiện lỗi.

Đối với loại lỗi CRC thì cờ báo lỗi chỉ được truyền sau vùng ACK DELIMITER trừ khi có một cờ báo lỗi sinh ra bởi điều kiện khác đang được truyền. Ví dụ, giả sử Node A và Node B đều ở trạng thái lỗi chủ động. Node A là transmitter phát hiện lỗi bit tại bit cuối của CRC Sequence bị lỗi (lỗi Bit). Node A phát cờ báo lỗi bắt đầu từ vùng đáng lẽ là CRC Delimiter và làm vi phạm định dạng bit của vùng CRC Delimiter. Node B không phát hiện ra lỗi ở bit cuối CRC Seuqence nhưng sau khi nhận bit cuối cùng của CRC Sequence thì Node B kiểm tra thấy lỗi CRC. Tuy nhiên, Node B giám sát thấy vùng CRC Delimiter bị lỗi (lỗi định dạng - FORM ERROR) nên Node B phát cờ báo lỗi của nó bắt đầu từ vùng đáng lẽ là ACK SLOT chứ không chờ sau vùng ACK DELIMITER mới phát cờ báo lỗi để đáp ứng cho lỗi CRC.

|  |
| --- |
|  |
| Hình 3. Ví dụ về việc phát hiện lỗi và báo lỗi |

**4. Quy định về việc tăng/giảm giá trị các bộ đếm lỗi**

Hai bộ đếm lỗi Tx\_count và Rx\_count khi nào tăng? khi nào giảm? giá trị tăng hoặc giảm là bao nhiêu? được quy định rõ ràng bởi chuẩn. Trong suốt quá trình truyền một thông điệp, các quy tắc tăng giảm bộ đếm lỗi sau đây sẽ được áp dụng:

1. Node là bộ nhận phát hiện một lỗi thì Rx\_count tăng 1, ngoại trừ phát hiện lỗi bit khi phát cờ báo lỗi chủ động và cờ báo quá tải
2. Node là bộ nhận phát hiện một bit dominant ở bit đầu tiên sau khi phát một cờ báo lỗi thì Rx\_count tăng 1
3. Node là bộ truyền phát một cờ báo lỗi thì Tx\_count tăng 8. Tuy nhiên, Tx\_count không thay đổi trong hai trường hợp ngoại lệ sau:
   1. Ngoại lệ 1: Bộ truyền trong trạng thái lỗi bị động phát hiện lỗi ACK và không phát hiện một bit dominant nào trong khi gửi cờ báo lỗi bị động của nó.
   2. Ngoại lệ 2: Nếu bộ truyền gửi một cờ báo lỗi sinh ra bởi lỗi chèn bit xuất hiện trong vùng phân xử mà bit chèn này đáng lẽ là recessive và bộ truyền đã truyền đi bit recessive nhưng lại được giám sát thấy là dominant
4. Node là bộ truyển phát hiện lỗi bit khi đang truyền cờ báo lỗi chủ động hoặc cờ báo quá tải thì Tx\_count tăng 8
5. Node là bộ nhận phát hiện một lỗi bit trong khi gửi một cờ báo lỗi chủ động hoặc cờ báo quá tải thì Rx\_count tăng 8
6. Sau khi gửi một cờ báo lỗi chủ động, cờ báo lỗi bị động hoặc cờ báo quá tải, nếu một Node phát hiện thêm 7 bit dominant liên tiếp thì:
   1. Trong trường hợp của cờ báo lỗi chủ động và cờ báo quá tải, sau khi phát hiện bit dominant thứ 14 liên tiếp và sau mỗi chuỗi 8 bit dominant tiếp theo thì Tx\_count và Rx\_count tăng lên 8.
   2. Trong trường hợp của cờ báo lỗi bị động, sau khi phát hiện bit dominant thứ 8 liên tiếp và sau mỗi chuỗi 8 bit dominant tiếp theo thì Tx\_count và Rx\_count tăng lên 8.
7. Sau khi truyền thành công một thông điệp (phát hiện được ACK và không có lỗi cho đến khi EOF kết thúc), Tx\_count giảm 1 trừ khi Tx\_count đang là 0 thì không giảm.
8. Sau khi nhận thành công một thông điệp (nhận không lỗi đến ACK SLOT và gửi thành công bit ACK) thì:
   1. Rx\_count giảm 1 nếu giá trị 1 ≤ Rx\_count ≤ 127.
   2. Rx\_count giữa giá trị 0 nếu nó đang bằng 0
   3. Rx\_count được gán một giá trị giữa 119 và 127 nếu Rx\_count ≥ 127

Chú ý, trong quá trình khởi động (Start-up), chỉ có một Node được phép hoạt động. Nếu Node này truyền một số thông điệp, nó sẽ không nhận được ACK nên phát hiện ra lỗi và lặp lại việc truyền thông điệp. Node này có thể rơi vào trạng thái lỗi bị động nhưng không được phép bị bus-off vì lý do này.

### Tính hợp lệ của thông điệp

Thời điểm xác nhận sự hợp lệ của thông điệp của bộ truyền và bộ nhận là khác nhau.

Với bộ truyền, thông điệp gọi là hợp lệ khi không có bất kỳ lỗi nào cho đến khi EOF kết thúc. nếu thông điệp bị lỗi thì thông điệp sẽ được tự động truyền lại dựa trên sự ưu tiên. Để có thể cạnh trang với sự truy cập bus từ các thông điệp khác thì việc truyền lại phải được bắt đầu sớm nhất có thể khi bus rảnh (BUS IDLE).

Với bộ nhận, thông điệp gọi là hợp lệ nếu nó không lỗi cho đến bit cuối của EOF. Giá trị bit cuối EOF, được bộ nhận xem như don't care. Bộ nhận phát hiện bit cuối này là dominant cũng không xem là một lỗi định dạng (FORM ERROR) mà đáp ứng lại bằng một khung báo quá tải.

## LUẬT TRUYỀN CHUỖI BIT CỦA CAN - BIT STREAM CODING

### Loại mã hóa chuỗi bit

Phương pháp mã hóa được sử dụng trong giao thức CAN là phương pháp NRZ mà cụ thể hơn là NRZ-L (Non-Return-to-Zero Level). Đây là dạng mã hóa theo mức, tức là mỗi trạng thái dominant và recesive sẽ được quy định mang một mức áp khác nhau. Cụ thể, dominant là mức thấp tương ứng với logic 0, recessive là mức cao tương ứng với logic 1. Trong suốt thời gian phát một bit, không xảy ra sự chuyển mức.

|  |
| --- |
|  |
| Hình 1. Mã hóa NRZ |

**2. Luật chèn bit - Bit stuffing**

Vấn đề của loại mã hóa NRZ là có thể dẫn đến mất đồng bộ và nhận sai giá trị bit khi truyền một chuỗi bit cùng mức logic quá dài. Vì vậy, CAN quy định cứ sau 5 bit liên tiếp cùng mức logic thì bộ truyền phải tự động chèn một bit trái mức logic, gọi là bit chèn (STUFF BIT).

Luật chèn bit này áp dụng cho vùng SOF, vùng phân xử, vùng điều khiển, vùng dữ liệu và cùng CRC SEQUENCE. Các cùng còn lại là CRC DELIMITER, ACK FIELD và EOF không áp dụng luật này.

Bộ nhận tự động giám sát để loại bỏ bit chèn khi nhận dữ liệu.

Ví dụ:

* Chuỗi bit gốc: 0100000 00000 1111 10001
* Chuỗi bit được chèn: 0100000**1**00000**1**1111**0**10001

|  |
| --- |
|  |
| Hình 2. (a) chuỗi bit gốc (b) chuỗi bit sau khi áp dụng luật chèn bit với S là vị trị bit được chèn thêm |

Vai trò của các bit chèn là tạo ra sự chuyển mức (cạnh lên hoặc cạnh xuống) để bộ nhận có thể sử dụng tái đồng bộ. Việc đồng bộ khi truyền nhận dữ liệu sẽ được trình bày sau.

Khung báo lỗi và khung báo quá tải có định dạng cố định và không áp dụng luật chèn bit.

**3. Luật phân xử**

Như đã trình bày trong các phần trước, luật phân xử bit giải quyết vấn đề xung đột khi nhiều Node cùng truyền dữ liệu. Đồng thời, luật phân xử cũng đảm bảo việc truyền thông điệp không bị bất cứ gián đoạn nào khi có xung đột xảy ra.

Sử phân xử này xảy ra trong vùng IDENTIFIER của thông điệp với bit dominant luôn chiếm ưu thế.

|  |
| --- |
|  |
| Hình 3. Luật phân xử của CAN bus |

Sự phân xử và chèn bit được thực hiện tự động bởi chip (vi mạch) thực hiện giao thức CAN.

### [Quy định về thời gian bit - Bit Timing](https://nguyenquanicd.blogspot.com/2017/06/can20controller-area-networkbai-8-quy.html)

Tác giả [Nguyễn Quân](https://www.blogger.com/profile/11739939602738380602) at 00:59 [CAN](https://nguyenquanicd.blogspot.com/search/label/CAN?&max-results=5), [CAN2.0](https://nguyenquanicd.blogspot.com/search/label/CAN2.0?&max-results=5), [Controller Area Network](https://nguyenquanicd.blogspot.com/search/label/Controller%20Area%20Network?&max-results=5), [Kiến Thức Cơ Bản](https://nguyenquanicd.blogspot.com/search/label/Ki%E1%BA%BFn%20Th%E1%BB%A9c%20C%C6%A1%20B%E1%BA%A3n?&max-results=5), [Protocol](https://nguyenquanicd.blogspot.com/search/label/Protocol?&max-results=5) [0 bình luận](https://nguyenquanicd.blogspot.com/2017/06/can20controller-area-networkbai-8-quy.html#comment-form)

**1. Tốc độ bit danh định (Nominal Bit Rate)**

Tốc độ bit danh định (**fn**) là số bit được truyền trong một giây (bps - bit per second) bằng bộ truyền lý tưởng và không có sự tái đồng bộ xảy ra. Sự tái đồng bộ cho phép hiệu chỉnh lại ví trí lấy mẫu bit khi xuất hiện sự lệch bit trên đường truyền.

Tốc độ bit danh định có thể hiểu là tốc độ bit cao nhất mà đường truyền có thể đạt được trên lý thuyết.

**2. Thời gian bit danh định (Nominal Bit Time)**

Thời gian bit danh định (Nominal Bit Time) được định nghĩa như sau:

**Thời gian bit danh định (Tn) = 1/(Tốc độ bit danh định - fn)**

Thời gian bit danh định được chia thành 4 đoạn riêng biệt (4 khoảng thời gian) không chồng lấn lên nhau là

* Đoạn đồng bộ (Synchronization Segment) ký hiệu là Sync\_seg
* Đoạn lan truyền (Propagation Time Segment) ký hiệu là Prop\_seg
* Đoạn đệm pha thứ nhất (Phase Buffer Segment 1) ký hiệu là Phase\_seg1
* Đoạn đệm pha thứ hai (Phase Buffer Segment 2) ký hiệu là Phase\_seg2.

|  |
| --- |
|  |
| Hình 1. Các phân đoạn thời gian của 1 bit |

*Sync\_seg:* Đoạn này dành cho việc đồng bộ giữa các Node trong 1 bus CAN. Một cạnh bit xuất hiện trên bus sẽ được mong chờ nằm trong đoạn này. Một Node luôn giám sát và tính toán thời gian bit. Trong khi truyền dữ liệu, bất cứ khi nào xuất hiện sự chuyển mức logic (cạnh bit) thì Node phải đồng bộ lại hoặc hiệu chỉnh cách tính toán thời gian bit để việc lấy mẫu bit được chính xác. Khi một cạnh bit xuất hiện mà Node gimas sát thấy nằm đúng trong đoạn Sync\_seq thì đường truyền đang ổn định và các Node đang có sự đồng bộ tốt.

*Prop\_seg:* Đoạn này được dùng để bù cho các thời gian trễ vật lý trong mạng. Nó gấp hai lần khoảng thời gian lan truyền tín hiệu trên bus, độ trễ mạch so sánh (comparator) ngõ vào và độ trễ mạch lái (driver) ngõ ra.  
  
*Phase\_seq1 và Phase\_seq2:* Hai đoạn này dành cho việc bù lỗi pha của cạnh bit. Thời gian các đoạn này có thể được kéo dài hoặc rút ngắn khi thực thi tái đồng bộ.

*Vị trí lấy mẫu (Sample point)* là thời điểm tại đó mức trạng thái của bus CAN được đọc để xác định giá trị logic của bit. Ví trí lấy mẫu là ở cuối đoạn Phase\_seg1. Trong quá trình truyền nhận dữ liệu, đoạn Phase\_seg1 và Phase\_seg2 được điều chỉnh để tái đồng bộ nhưng vị trí lấy mẫu này luôn ở cuối Phase\_seg1.  
  
*Thời gian xử lý thông tin Tifp (Information processing time)* là thời gian dự trữ tính từ vị trí lấy mẫu dành cho việc tính toán mức bit tiếp theo trong chuỗi. Độ dài của nó được xác định tùy vào phần cứng (chip) CAN controller.  
  
**3. Tính toán thời gian bit**  
  
Đơn vị thời gian sử dụng trong chuẩn CAN là TQ (Time Quantum) được tính từ chu kỳ bộ dao động **Tos** (chu kỳ xung clock) của CAN controller, datasheet hoặc user manual của CAN controller sẽ quy định rõ nguồn clock này. Bộ dao động ở đây có thể là thạch anh hay bộ tạo dao động bên trong chip cấp cho CAN controller.  
  
TQ là đơn vị thời gian cố định trong suốt quá trình hoạt động của Node nhưng có thể lập trình được. Một TQ sẽ bằng m lần TQmin. TQmin (MINIMUM TIME QUANTUM) là đơn vị thời gian tối thiểu được tính từ Tos, n không có giới hạn cụ thể.

**TQ = m x TQmin = m x n x Tos**

Trong đó m là giá trị nguyên trong khoảng từ 1 đến 32.

|  |
| --- |
|  |
| Hình 2. Thời gian của 1 bit với n = 4, m = 3 |

Thời gian từng đoạn chia của 1 bit được quy định cụ thể như sau:

* Sync\_seg: 1 TQ
* Prop\_seg: từ 1 đến 8 TQ
* Phase\_seg1: từ 1 đến 8 TQ
* Phase\_seg2 = max(Phase\_seg1, Tifp) .Nghĩa là, Phase\_seg1 và Tifp cái nào lớn hơn thì sử dụng giá trị đó cho Phase\_seg2.
  + Trong đó, thời gian xử lý thông tin Tifp nhỏ hơn hoặc bằng 2 TQ
* Tổng thời gian một bit ít nhất là 8 TQ và tối đa là 25 TQ

Thông thường việc lập trình thời gian bit là cần thiết để tất cả các Node trong mạng có tốc độ bit và thời gian bit giống nhau. Giống nhau ở đây nghĩ là có sự sai lệch ít nhất. Thường các Node này sẽ đi kèm với một vi xử lý hoặc vi điều khiển. Tuy nhiên, CAN có thể được thiết kế mà không cần cấu hình thời gian bit, nghĩa là nó có thể tự điều chỉnh đến tốc độ bit thích hợp chỉ cần cung cấp nguồn xung clock phù hợp. Đối với Node loại này thì vị trí điểm lấy mẫu phải tương thích cho tất cả các Node và thời gian bit theo định nghĩa sau đây:

* Sync\_seg: 1 TQ
* Prop\_seg: 1 TQ
* Phase\_seg1: 4 TQ
* Phase\_seg2: 4 TQ

|  |
| --- |
|  |
| Hình 3. Thời gian bit cho các linh kiện CAN không lập trình được thời gian bit |

**4. Độ sai số cho phép của bộ dao động**

Độ sai số cho phép tối đa của bộ dao động là 1.58% và sử dụng một bộ cộng hưởng gốm (ceramic resonator) ở tốc độ bus lên đến 125 Kbits/s như nguyên tắc cơ bản. Tham khảo bài báo sau để có thông tin chính xác hơn:  
  
Dais, S; Chapman, M;  
“Impact of Bit Representation on Transport Capacity and Clock  
Accuracy in Serial Data Streams”,  
SAE Technical Paper Series 890532, Multiplexing in Automobiles SP-773 March 1989

Một Node muốn sử dụng được đầy đủ các tốc độ bus của giao thức CAN (full speed) thì cần sử dụng bộ dao động thạch anh. Chip CAN yêu cầu cao nhất về độ chính xác của bộ tạo dao động. Việc xác định độ chính xác của các bộ tạo dao động sử dụng trong các Node của mạng CAN phải được thực hiện.

**5. Tính toán thời gian trễ lan truyền và Prop\_seg**  
  
Như đã trình bày ở trên, đoạn Prop\_seg gấp hai lần khoảng thời gian lan truyền tín hiệu trên bus, độ trễ mạch so sánh (comparator) ngõ vào và độ trễ mạch lái (driver) ngõ ra. Độ trễ mạch lái ngõ ra và độ trễ mạch so sánh ngõ vào thuộc chip CAN transceiver.

|  |
| --- |
|  |
| Hình 4. Giải thích độ trễ giữa 2 Node trong bus CAN |

Theo hình minh họa trên, các độ trễ được tính như sau:

* Độ trễ từ Node A đến Node B:
  + Độ trễ mạch lái ngõ ra tại Node A: Txa
  + Độ trễ trên bus: Tbus
  + Độ trễ mạch so sánh ngõ vào tại Node B: Trb
  + Độ trễ lan truyền từ A đến B:
    - Tprop(A->B) = Txa + Tbus + Txb
* Độ trễ từ Node B đến Node A:
  + Độ trễ mạch lái ngõ ra tại Node B: Txb
  + Độ trễ trên bus: Tbus
  + Độ trễ mạch so sánh ngõ vào tại Node A: Tra
  + Độ trễ lan truyền từ B đến A:
    - Tprop(B->A) = Txb + Tbus + Txa

Thời gian đoạn Prop\_seg:

Tprop\_seg = Tprop(A->B) + Tprop(B->A)

Nếu CAN transceiver ở cả 2 Node là cùng loại thì Txa = Txb và Tra = Trb. Vì vậy, thời gian lan truyền từ A đến B và từ B đến A là như nhau.

Tprop\_seg = 2 x (Tx + Tbus + Tr)

Số TQ cấu hình cho Prop\_seg được tính như sau:

Prop\_seg = Tprop\_seg/Ttq

Nếu không phải là một số nguyên thì giá trị này được làm tròn lên đến số nguyên gần nhất. Chú ý, thông số cấu hình này được tính trên trường hợp xấu nhất là 2 Node nằm cách xa nhau nhất (nằm ở hai đầu cuối) trong bus CAN.  
  
*Lịch sử cập nhật:*  
1) 2019.10.18 - Chỉnh sửa link hình minh họa

### [Quy định về đồng bộ trong bus CAN](https://nguyenquanicd.blogspot.com/2017/06/can20controller-area-networkbai-10-quy.html)

Tác giả [Nguyễn Quân](https://www.blogger.com/profile/11739939602738380602) at 04:36 [CAN](https://nguyenquanicd.blogspot.com/search/label/CAN?&max-results=5), [CAN2.0](https://nguyenquanicd.blogspot.com/search/label/CAN2.0?&max-results=5), [Controller Area Network](https://nguyenquanicd.blogspot.com/search/label/Controller%20Area%20Network?&max-results=5), [Kiến Thức Cơ Bản](https://nguyenquanicd.blogspot.com/search/label/Ki%E1%BA%BFn%20Th%E1%BB%A9c%20C%C6%A1%20B%E1%BA%A3n?&max-results=5), [Protocol](https://nguyenquanicd.blogspot.com/search/label/Protocol?&max-results=5) [0 bình luận](https://nguyenquanicd.blogspot.com/2017/06/can20controller-area-networkbai-10-quy.html#comment-form)

**1. Đồng bộ cứng - Hard Synchronization**

Đồng bộ cứng làm cho thời gian bit đang được giám sát tại một Node CAN khởi động lại vị trí Sync\_seg.

Đồng bộ cứng xảy ra khi có cạnh bit, cạnh lên hoặc cạnh xuống. Như vậy, một cạnh bit xuất hiện sẽ làm việc tính toán thời gian bit của một Node khởi động lại về Sync\_seg.

Khi Node phát hiện cạnh xuống của SOF trên đường truyền, Node bắt đầu tính toán thời gian bit bắt đầu từ Sync\_seg.

|  |
| --- |
|  |
| Hình 1. Các phân đoạn của 1 bit |

**2. Độ rộng bước tái đồng bộ - Resynchronization Jump Width**

Như đã trình bày về [định thời bit](http://nguyenquanicd.blogspot.com/2017/06/can20controller-area-networkbai-8-quy.html), độ dài đoạn Phase\_seg1 và Phase\_seg2 không cố định mà Phase\_seg1 có thể bị kéo dài và Phase\_seg2 có thể bị rút ngắn trong quá trình một Node hoạt động.

Độ rộng bước tái đông bộ (RJW hoặc SJW) sẽ quy định đố lớn của đoạn được kéo dài hoặc rút ngắn. RJW có thể được lập trình trong khoảng từ 1 đến Min(4, Phase\_seg1).

**3. Lỗi pha của một cạnh bit - Phase Error**

Lỗi pha của một cạnh, cạnh lên hoặc cạnh xuống, được xác định bởi vị trí của cạnh bit so với đoạn Sync\_seg. Lỗi pha được đo bằng đơn vị TQ. Ký hiệu lỗi pha được quy định như sau:

* e = 0: cạnh bit xuất hiện trong vùng Sync\_seg và không xảy ra lỗi pha
* e > 0: cạnh bit xuất hiện sau Sync\_seg và trước điểm lấy mẫu (Sample point) gọi là lỗi pha dương
* e < 0: cạnh bit xuất hiện sau điểm lấy mẫu và trước Sync\_seg của bit tiếp theo gọi là lỗi pha âm

|  |
| --- |
|  |
| Hình 2. Vị trí xác định lỗi pha của cạnh bit |

Như vậy việc xác định lỗi pha chỉ khi có sự xuất hiện cạnh bit. Việc truyền liên tiếp nhiều bit cùng mức logic sẽ không xác định được lỗi pha. Mỗi Node đều giám sát thời gian từng bit được truyền trên bus. Khi một cạnh bit xuất hiện. Một Node sẽ kiểm tra xem cạnh này rơi vào đoạn thời gian nào của bit mà nó đang giám sát. Từ đó, nó xác định có lỗi pha hay không.

CAN truyền theo dạng NRZ nên việc truyền nhiều bit cùng mức logic liên tiếp sẽ có thể gây ra mất đồng bộ dẫn đến việc nhận dữ liệu bị sai. Vì vậy, chuẩn CAN đã quy định về việc chèn bit để giải quyết vấn đề này. Việc chèn một bit trái dấu sẽ làm các Node đang nhận đồng bộ lại việc tính toán thời gian bit của chúng.

**4. Tái đồng bộ - Resynchronization**

Sự tái đồng bộ chỉ có thể xảy ra khi xuất hiện cạnh bit. Sự tái đồng bộ xảy ra như đồng bộ cứng (hard synchronization) nếu lớn lỗi pha có xảy ra nhưng độ lớn lỗi pha nhỏ hơn hoặc bằng độ rộng bước tái đồng bộ RJW.

Nếu độ lớn lỗi pha lớn hơn RJW thì việc tái đồng bộ được thực hiện như sau:

* Nếu lỗi pha dương thì Phase\_seg1 được kéo dài thêm một đoạn đúng bằng RJW.
* Nếu lỗi pha âm thì Phase\_seg2 được rút ngắn một đoạn đúng bằng RJW

**5. Quy luật đồng bộ - Synchronization rules**

Đồng bộ cứng và Tái đồng bộ là hai dạng của sự đồng bộ. Chúng tuân theo quy luật sau đây:

1. Chỉ một quy luật đồng bộ được áp dụng cho một bit. Một bit nếu đã áp dụng đồng bộ cứng thì không áp dụng tái đồng bộ.
2. Sự đồng bộ chỉ áp dụng cho một cạnh nếu giá trị được phát hiện tại điểm lấy mẫu trước đó (giá trị bus được đọc trước đó) khác giá trị bus ngay sau cạnh bit.
3. Đồng bộ cứng xảy ra khi xuất hiện cạnh chuyển từ bit recessive sang bit dominant khi bus IDLE. Như vậy, việc xuất hiện cạnh của một SOF sẽ làm tất cả các Node trong mạng CAN được đồng bộ.
4. Tất cả các cạnh chuyển từ recessive thành dominant khác theo luật 1 và luật 2 sẽ sử dụng Tái đồng bộ. Ngoại trừ, một Node truyền một bit dominant sẽ không thực hiện Tái đồng bộ khi cạnh chuyển từ recessive sang dominant với lỗi pha dương, nếu chỉ có các cạnh recessive-dominant được sử dụng cho việc Tái đồng bộ.

**6. Giải thích về các yêu cầu và quy luật đồng bộ**

Kết hợp quy tắc Đồng bộ cứng và Quy luật đồng bộ thì Đồng bộ cứng chỉ xảy ra khi bắt đầu bit SOF khi xét trong một thông điệp.

Sự tái đồng bộ có hai cơ chế thực hiện tùy vào độ lớn của lỗi pha:

* Lỗi pha nhờ hơn hoặc bằng RJW thì Tái đồng bộ khởi động lại việc tính thời gian bit vào đoạn Sync\_seg như đồng bộ cứng
* Lỗi pha lớn hơn RJW thì Tái đồng bộ kéo dài đoạn Phase\_seg1 hoặc rút ngắn đoạn Phase\_seg2 theo giá trị cấu hình RJW

|  |
| --- |
|  |
| Hình 3. Sự tái đồng bộ khi xuất hiện lỗi pha dương nhỏ hơn RJW (trên) và lớn hơn RJW (dưới) |

|  |
| --- |
|  |
| Hình 4. Sự tái đồng bộ khi xuất hiện lỗi pha âm nhỏ hơn RJW (trên) và lớn hơn RJW (dưới) |

### [][Bài 11] Ví dụ về quan sát một bus CAN thực tế bằng dao động ký](https://nguyenquanicd.blogspot.com/2017/06/can20controller-area-networkbai-11-vi.html)

Tác giả [Nguyễn Quân](https://www.blogger.com/profile/11739939602738380602) at 23:39 [CAN](https://nguyenquanicd.blogspot.com/search/label/CAN?&max-results=5), [CAN2.0](https://nguyenquanicd.blogspot.com/search/label/CAN2.0?&max-results=5), [CANFD](https://nguyenquanicd.blogspot.com/search/label/CANFD?&max-results=5), [Controller Area Network](https://nguyenquanicd.blogspot.com/search/label/Controller%20Area%20Network?&max-results=5), [Kiến Thức Cơ Bản](https://nguyenquanicd.blogspot.com/search/label/Ki%E1%BA%BFn%20Th%E1%BB%A9c%20C%C6%A1%20B%E1%BA%A3n?&max-results=5), [Protocol](https://nguyenquanicd.blogspot.com/search/label/Protocol?&max-results=5) [0 bình luận](https://nguyenquanicd.blogspot.com/2017/06/can20controller-area-networkbai-11-vi.html#comment-form)

Tham khảo: Introduction to the Controller Area Network (CAN), Application Report, TI

Nội dung bài này tham khảo từ một kết quả đo đạc của TI nhằm cho bạn đọc có cái nhìn cụ thể về sự truyền nhận dữ liệu trên 1 bus CAN và quan sát hoạt động 1 bus CAN trên thực tế.

Chuẩn CAN định nghĩa một giao thức mạng thông tin cho phép tất cả các Node trong mạng có thể thông tin được với nhau. Mạng có thể có hoặc không có Node điều khiển trung tâm. Các Node có thể được thêm vào mạng bất kỳ lúc nào kể cả lúc mạng đang hoạt động (hot-plugging).

Việc quan sát được thực hiện trên:

1. Bus CAN ở Ch 1 sử dụng đầu đo vi sai
2. Đường TX của Node A, nối từ vi xử lý DSP có tích hợp CAN controller đến CAN transceiver, ở Ch 2
3. Đường TX của Node B, nối từ vi xử lý DSP có tích hợp CAN controller đến CAN transceiver, ở Ch 3
4. Đường TX của Node C, nối từ vi xử lý DSP có tích hợp CAN controller đến CAN transceiver, ở Ch 4

|  |
| --- |
|  |
| Hình 1. Minh họa bus CAN được kết nối và đo đạc bằng dao động ký |

Đầu tiên Node A truyền một thông điệp hợp lệ và được Node B và C xác nhận bằng 1 bit dominant ở vùng ACK slot. Sau đó, Node B và Node C cùng phát thông điệp. Sự xung đột xảy ra và Node C thắng phân xử nên truyền tiếp cho đến khi kết thúc thông điệp của nó. Node B ngừng truyền ngay khi thua phân xử và trở thành bộ nhận. Sau đó, cả Node A và Node B đều phát ACK báo hiệu Node C truyền một thông điệp hợp lệ. Cuối cùng, Node B phát lại thông điệp của nó khi bus rảnh và được Node A và Node C báo ACK.

Chú ý, cực của dạng sóng trên bus đo bằng đầu dò vi sai ở hình sau ngược với dạng sóng đo trên các đường TX, đồng thời cũng là ngõ vào bộ lái bus CAN của CAN transceiver.

|  |
| --- |
|  |
| Hình 2. Dạng sóng thể hiện trên dao động ký của bus CAN |

### [AN2.0][Controller Area Network][Bài 12] Lọc thông điệp trong giao thức CAN](https://nguyenquanicd.blogspot.com/2017/06/can20controller-area-networkbai-12-loc.html)

Tác giả [Nguyễn Quân](https://www.blogger.com/profile/11739939602738380602) at 22:30 [CAN](https://nguyenquanicd.blogspot.com/search/label/CAN?&max-results=5), [CAN2.0](https://nguyenquanicd.blogspot.com/search/label/CAN2.0?&max-results=5), [Controller Area Network](https://nguyenquanicd.blogspot.com/search/label/Controller%20Area%20Network?&max-results=5), [Kiến Thức Cơ Bản](https://nguyenquanicd.blogspot.com/search/label/Ki%E1%BA%BFn%20Th%E1%BB%A9c%20C%C6%A1%20B%E1%BA%A3n?&max-results=5), [Protocol](https://nguyenquanicd.blogspot.com/search/label/Protocol?&max-results=5) [0 bình luận](https://nguyenquanicd.blogspot.com/2017/06/can20controller-area-networkbai-12-loc.html#comment-form)

**1. Lọc thông điệp**

Việc lọc thông điệp thực hiện trên các thông điệp được nhận bởi Node. Một Node lọc thông điệp để biết nó có phải là thông điệp của nó hay không.

Việc lọc thông điệp thực hiện trên toàn bộ vùng ID (Identifier) của thồn điệp. Việc lọc thông điệp là sự so sánh giữa ID thông điệp với một hoặc nhiều giá trị ID được cấu hình bên trong một Node.

Mỗi bit được so sánh có thể được che bởi các thanh ghi mặt nạ (mask register) để thiết lập bit này là don't care, tức là không quan tâm đến giá trị bit này là bao nhiêu khi so sánh. Các bit của thanh ghi mặt nạ phải lập trình được. Độ dài của thanh ghi mặt nạ có thể bằng độ dài của toàn bộ vùng ID hoặc chỉ bằng một phần vùng này.

Sau đây là một ví dụ về việc lọc thông điệp được thực hiện trong chip CAN controller mcp2515 của Microchip. Từng bit ID của thông điệp nhận được (Message Assembly Buffer Identifier) sẽ được so sánh với Thanh ghi lưu giá trị ID được chấp nhận (Acceptance Filter register) bằng cổng XOR. Nếu giá trị giống nhau, ngõ ra cồng XOR là 0; nếu khác nhau thì ngõ ra cổng XOR là 1. Giá trị này được lấy bù để OR với giá trị bù của của các bit thanh ghi mặt nạ (Acceptance Mask register). Nếu tất cả các ngõ ra cổng OR là 1 thì tín hiệu báo thông điệp hợp lệ với Node (RxRqst) sẽ tích cực. Nếu 1 hoặc nhiều ngõ ra cổng OR bằng 0, nghĩa là có 1 hoặc nhiều bit ID không phù hợp với cấu hình của Node thì RxRqst sẽ không tích cự và Node không nhận thông điệp này.

|  |
| --- |
|  |
| Hình 1. Cấu tạo thành phần lọc thông điệp của chip CAN controller mcp2515 của Microchip |

Đối với mạch nguyên lý trên, nếu bit thanh ghi mặt nạ bằng 0 thì giá trị so sánh tại vị trí đó, ngõ ra cổng XOR, sẽ là don’t care và giá trị bit ID tại vị trí này luôn phù hợp cho dù là 0 hay 1.

|  |
| --- |
|  |
| Hình 2. Bảng giá trị minh họa mối liên hệ giữa bit mặt nạ (Mask), bit cấu hình lọc IC (Filter) và giá trị bit ID của thông điệp |

Việc sử dụng thanh ghi mặt nạ cho phép một Node có thể được cấu hình 1 khoảng giá trị ID thay vì từng giá trị đơn lẻ. Theo mạch nguyên lý đã trình bày trên đây, với 29 bit ID, giả sử các giá trị cấu hình là:

* Thanh ghi lưu các bit Filter: h0000\_005A
* Thanh ghi lưu các bit Mask: hFFFF\_FFF0

thì khoảng giá trị ID mà Node sẽ nhận là từ h0000\_0050 đến h0000\_005F.

**2. Minh họa việc truyền nhận thông điệp trên bus CAN**

Giả sử ta có bốn Node CAN như sau:

* Node 1: muốn truyền dữ liệu với IDENTIFIER 01101000110 (838).
* Node 2: muốn truyền một dữ liệu khác với IDENTIFIER 01101001000 (840) và cho phép chỉ nhận dữ liệu với IDENTIFIER 01101000110 (838).
* Node 3: không truyền dữ liệu nào và có khoảng ID cho phép là 0110100xxxx (‘x’ hiểu là don't care, tức tầm giá trị của khoảng cho phép là [832; 847]).
* Node 4: muốn truyền dữ liệu có IDENTIFIER 01110010010 (914) và cho phép nhận ID là 0111001xxx1 (tức tầm của khoảng ID cho phép là các số lẻ trong đoạn [913; 927])

Giả sử, Node 1, Node 2, Node 4 cùng truyền dữ liệu của mình trong cùng một thời gian. Như vậy, cả 3 Node sẽ chuyển từ trạng thái Idle thành bộ truyền và sự xung đột sẽ xuất hiện nên trên bus sẽ xảy ra sự phân xử.

|  |
| --- |
|  |
| Hình 3. Sự phân xử khi truyền thông điệp trên bus CAN |

Sự phân xử thực hiện như sau, ở ba bit đầu tiên, cả 3 Node đều truyền cùng mức và giá trị bus ở vị trí tương ứng cũng vậy. Đến bit thứ 4, Node 1 và Node 2 truyền dominant nên giá trị bus theo ‘AND-wire’ là dominant. Cùng lúc này, node 4 truyền recessive nhưng khi giám sát bus lại thấy là dominant, nó hiểu là có nút khác cũng đang truyền dữ liệu. Node 4 sẽ ngưng truyền và trở thành bộ nhận. Ta nói rằng nút 4 thua trong sự phân xử. Tương tự như vậy, đến bit thứ 8, Node 2 cũng thua trong sự phân xử và trở thành bộ nhận.

Kết quả của cuối cùng của trình phân xử trên là chỉ có Node 1 truyền dữ liệu với IDENTIFIER 838 của nó trên bus và nó là bộ truyền cho đến khi bus Idle. Các Node còn lại đều trở thành bộ nhận dữ liệu từ Node 1. Trong 3 Node nhận là Node 2, Node 3, Node 4 thì Node 4 sẽ không nhận dữ liệu này vì 838 không thuộc khoảng ID [913; 927] mà nó được cấu hình. Node 2 và Node 3 sẽ cho phép dữ liệu qua lớp phụ LLC để lên lớp cao hơn.

|  |
| --- |
|  |
| Hình 3. Hoạt động truyền nhận của các Node trong bus CAN |

*Lịch sử cập nhật:*  
1) 2019.10.18 - Chỉnh sửa link hình minh họa

### [D][Controller Area Network][Bài 4] CANFD - Sự khác nhau giữa CAN FD và CAN 2.0 - phần 4](https://nguyenquanicd.blogspot.com/2017/07/canfdcontroller-area-networkbai-4-canfd.html)

Tác giả [Nguyễn Quân](https://www.blogger.com/profile/11739939602738380602) at 07:45 [CAN](https://nguyenquanicd.blogspot.com/search/label/CAN?&max-results=5), [CANFD](https://nguyenquanicd.blogspot.com/search/label/CANFD?&max-results=5), [Controller Area Network](https://nguyenquanicd.blogspot.com/search/label/Controller%20Area%20Network?&max-results=5), [Kiến Thức Cơ Bản](https://nguyenquanicd.blogspot.com/search/label/Ki%E1%BA%BFn%20Th%E1%BB%A9c%20C%C6%A1%20B%E1%BA%A3n?&max-results=5), [Protocol](https://nguyenquanicd.blogspot.com/search/label/Protocol?&max-results=5) [0 bình luận](https://nguyenquanicd.blogspot.com/2017/07/canfdcontroller-area-networkbai-4-canfd.html#comment-form)

CAN FD hỗ trợ hai tốc độ bit trong một khung truyền là tốc độ bit danh định (Nominal Bit Rate) dùng cho pha phân xử và tốc độ bit dữ liệu (Data Bit Rate) dùng cho pha dữ liệu. Phần này sẽ phân tích rõ hiện tượng chuyển đổi giữa hai tốc độ bit này trong giao thức CAN FD.

|  |
| --- |
|  |
| Hình 1. Pha phân xử và pha dữ liệu trong khung yêu cầu (trên) và khung dữ liệu (dưới) |

Việc tốc độ bit có bị chuyển đổi hay không phụ thuộc vào giá trị bit BRS (Bit Rate Switch).

* BRS = 1 là có chuyển đổi
* BRS = 0 là không chuyển đổi

Có hai điểm chuyển tốc độ bit là tại vị trí bit BRS và vị trí bit CRC delimiter. Tốc độ bit sẽ được chuyển đổi ngay sau điểm lấy mẫu (sample point) của 2 bit này. Điều này làm cho 2 bit này mang cùng lúc 2 cấu hình tốc độ bit khác nhau. Tổng độ dài của 2 bit này sẽ bằng độ dài 1 bit thuộc cấu hình thời gian bit danh định (Nominal Bit Time) và một bit thuộc cấu hình thời gian bit dữ liệu (Data Bit Time).

Trong trường hợp phát hiện lỗi, Node phải truyền khung báo lỗi nhưng khung này theo tốc độ bit danh định (Nominal Bit Rate). Nếu Node đang truyền trong pha dữ liệu thì nó phải chuyển đổi từ tốc độ bit dữ liệu sang tốc độ bit danh định để phát khung báo lỗi nhưng thời điểm chuyển đổi là sau điểm lấy mẫu của bit hiện tại một khoảng thời gian bằng thời gian xử lý thông tin (Information Processing Time - IPT).

Ví dụ, thời gian bit danh định được cấu hình dựa trên m(N) = 2, đây là hệ số tính độ dài TQ(N). Thời gian các đoạn trong 1 bit gồm:

* Prop\_seg(N) = 6 TQ
* Phase\_seg1(N) = 4 TQ
* Phase\_seg2(N) = 4 TQ

Thời gian dữ liệu được cấu hình dựa trên hệ số m(D) = 1 và có độ dài các đoạn trong 1 bit như sau:

* Prop\_seg(D) = 1 TQ
* Phase\_seg1(D) = 4 TQ
* Phase\_seg2(D) = 4 TQ

|  |
| --- |
|  |
| Hình 2. Ví dụ về thời gian bit danh định và thời gian bit dữ liệu |

Với cấu hình trên, thời gian của bit BRS và CRC Delimiter được thể hiện như sau:

* Bit BRS: trước điểm lấy mẫu theo cấu hình tốc độ bit danh định, sau điểm lấy mẫu theo tốc độ bit dữ liệu
* Bit CRC Delimiter: trước điểm lấy mẫu theo cấu hình tốc độ bit dữ liệu, sau điểm lấy mẫu theo tốc độ bit danh định

|  |
| --- |
|  |
| Hình 3. Độ dài bit BRS và CRC Delimiter |

*Lịch sử cập nhật:*  
1) 2019.10.18 - Chỉnh sửa link hình minh họa

### [- Sự khác nhau giữa CAN FD và CAN 2.0 - phần 5](https://nguyenquanicd.blogspot.com/2017/07/canfdcontroller-area-networkbai-5-canfd.html)

Tác giả [Nguyễn Quân](https://www.blogger.com/profile/11739939602738380602) at 08:33 [CAN](https://nguyenquanicd.blogspot.com/search/label/CAN?&max-results=5), [CANFD](https://nguyenquanicd.blogspot.com/search/label/CANFD?&max-results=5), [Controller Area Network](https://nguyenquanicd.blogspot.com/search/label/Controller%20Area%20Network?&max-results=5), [Kiến Thức Cơ Bản](https://nguyenquanicd.blogspot.com/search/label/Ki%E1%BA%BFn%20Th%E1%BB%A9c%20C%C6%A1%20B%E1%BA%A3n?&max-results=5), [Protocol](https://nguyenquanicd.blogspot.com/search/label/Protocol?&max-results=5) [0 bình luận](https://nguyenquanicd.blogspot.com/2017/07/canfdcontroller-area-networkbai-5-canfd.html#comment-form)

**1. Vấn đề đồng bộ trong CAN FD**

CAN FD có các quy định về đồng bộ cứng (hard synchronization), tái đồng bộ (Resynchronous), lỗi pha (Phase error) tương tự như CAN 2.0. Bên cạnh đó CAN FD cũng quy định thêm một số điểm khác biệt về vấn đề này.

Ở độ rộng bước tái đồng bộ RJW (Resynchronous Jump Width), nó còn được ký hiệu là SJW, RJW quy định cho pha phân xử và pha dữ liệu là khác nhau:

* Pha phân xử, RJW được lập trình trong khoảng từ 1 đến min (16, Phase\_seg1(N)).
* Pha phân xử, RJW được lập trình trong khoảng từ 1 đến min(4, Phase\_seg1(D)) -> như chuẩn CAN 2.0

Các quy định về sự đồng bộ (Synchronization Rules) cũng có những khác biệt như sau:

1. Giống CAN 2.0
2. Giống CAN 2.0
3. Cả CAN FD và CAN 2.0 đều quy định đồng bộ cứng được áp dụng khi xuất hiện cạnh chuyển từ recessive sang dominant ở vùng Bus Idle. Nhưng CAN FD quy định thêm, điều này còn áp dụng ở vùng Suspend Transmission (xem khoảng liên khung), ở bit thứ 2 hoặc bit thứ 3 của vùng Intermission. Đồng bộ cứng còn được áp dụng tại cạnh chuyển từ recessive sang dominant khi chuyển từ bit EDL thành bit r0 trong khung truyền định dạng CAN FD
4. Giống CAN 2.0
5. Một bộ truyền sẽ không thực thi tái đồng bộ trong khi truyền pha dữ liệu trong khung truyền CAN FD (Quy đình này thêm vào và dành riêng cho CAN FD)

**2. Dung sai của bộ dao động**

Bộ dao động ở đây là nguồn xung clock được cấp cho CAN controller sử dụng tính toán thời gian bit cho các khung truyền. Dung sai là độ sai số cho phép của bộ dao động để đảm bảo mỗi Node có thể hoạt động chính xác nhất.

CAN FD quy định rõ ràng vấn đề này hơn so với CAN 2.0. Cụ thể, dung sai df cho một bộ dao động tần số *fosc* sẽ quanh tần số danh định *fnom* như sau:

*(1-df).fnom <= fosc <= (1+df).fnom*

nhưng phụ thuộc vào tỷ lệ của Phase\_seg1, Phase\_seg2, SJW và thời gian bit. Dung sai df tối đa được định nghĩa bởi 5 điều kiện trong đó điều kiện I và II đã được áp dụng cho chuẩn CAN 2.0. Đối với CAN FD, tất cả 5 điều kiện đều phải được áp dụng.

|  |
| --- |
|  |
| Hình 1. Những điều kiện ràng buộc giá trị dung sai bộ dao động |

Ở đây, N là ký hiệu chỉ thông số thuộc cấu hình tốc độ bit danh định và D là ký hiệu chỉ thông số thuộc cấu hình tốc độ bit dữ liệu. Nguyên tắc rất đơn giản, tất cae các thông số cấu hình thời gian bit cần được tính toán bằng các điều kiện ràng buộc trên để chọn bộ dao động có dung sai phù hợp cấu hình. Hoặc khi đã có nguồn dao động và dung sai cụ thể của nó thì việc cấu hình thời gian bit và các thông số khác phải đáp ứng theo dung sai đó.

Cần lưu ý, SJW không được lớn hơn giá trị của Phase\_seg1 và Phase\_seg2. Đoạn Prop\_seg có thể được sử dụng cho các đoạn Phase\_seg1 và Phase\_seg2. Tổ hợp cấu hình Prop\_seg(N)=1 và Phase\_seg1(N)=Phase\_seg2(N)=SJW(N) cho phép dung sai lớn nhất lên đến 1.58%. Đây cũng là dung sai cho phép trong xử lý của CAN 2.0. Việc tổ hợp Prop\_seg khi nó chỉ chiếm 10% thời gian 1 bit là không thích hợp cho các thời gian bit ngắn. Nó có thể được sử dụng cho tốc độ tối đa là 125 kbit/s, tương ứng với thời gian bit là 8 us, với độ dài bus 40 m. Thời gian bit danh định này có thể tổ hợp với một thời gian dữ liệu ngắn hơn.

**3. Bù trễ Transceiver**

Bù trễ cho CAN transceiver là một khái niệm không có trong CAN 2.0.

Ở CAN FD, nếu không có cơ chế bù trễ, khi một Node là bộ truyền, nó có thể không giám sát được (không nhận được) giá trị bit do chính nó truyền ra tại điểm lấy mẫu (Sample point) như hình 2. Điều này sẽ giới hạn tốc độ của pha dữ liệu, thời gian bit của pha này không được phép quá ngắn dẫn đến tốc độ bit không được phép quá cao.

|  |
| --- |
|  |
| Hình 2. Trễ trên Transceiver làm giới hạn tốc độ truyền dữ liệu |

Các Node CAN FD sẽ hỗ trợ tùy chọn thực thi cơ chế bù trễ cho Transceiver để sử dụng cho các ứng dụng cần thời gian bit trong pha dữ liệu ngắn hơn giới hạn thời gian được quy định trong thời gian trễ của Transceiver.

Cơ chế bù trễ sẽ định nghĩa một điểm lấy mẫu thứ 2 (Secondary Sample Point). Điểm lấy mẫu này được sử dụng để so sánh giá trị bit được truyền ra so với bit được nhận vào theo thứ tự để kiểm tra lỗi bit. Cơ chế này chỉ sử dụng cho bộ truyền và trong pha dữ liệu của khung định dạng CAN FD. Khi sử dụng cơ chế này, bộ truyền sẽ không quan tâm đến giá trị bit ở vị trí lấy mẫu thông thường (Sample point).

Độ trễ bù được ký hiệu là TRV\_DELAY. Nó được đo trong từng khung truyền tại cạnh chuyển từ bit EDL sang bit r0 bằng cách đo khoảng cách giữa cạnh bit truyền ra trên TX và cạnh bit nhận được trên RX của CAN controller. Giá trị đo được tính theo đơn vị TQ và sẽ được làm tròn xuống đến số nguyên gần nhất.

Vị trí của điểm lấy mẫu thứ hai sẽ được tính từ điểm lấy mẫu thông thường cộng bù thêm TRV\_DELAY, ví dụ như nửa thời gian của 1 bit. Và nó có thể nằm sau điểm cuối của bit truyền.

|  |
| --- |
|  |
| Hình 3. Bù trễ transceiver trong CAN FD |

Nếu một lỗi bit (BIT ERROR) được phát hiện tại điểm lấy mẫu thứ 2, bộ truyền sẽ đáp ứng lại lỗi bit này này ở điểm lấy mẫu (sample point) tiếp theo.

# CAN-FD

## SỰ KHÁC NHAU GIỮA CAN FD VÀ CAN 2.0

### CAN FD là gì?

CAN FD như tên gọi là giao thức CAN với tốc độ dữ liệu linh động. Đây có thể xem là một giao thức mở rộng của giao thức CAN.

Việc ngày càng nhiều ứng dụng sử dụng các giao thức truyền thông nối tiếp dẫn đến nhu cầu bằng thông sử dụng trong giao thức CAN ngày càng tăng. Giao thức CAN truyền thống (CAN Specification 2.0) khó đáp ứng được trong một số ứng dụng nhất định. Để tăng khả năng sử dụng của giao thức CAN, CAN FD ra đời cho phép tốc độ truyền dữ cao hơn 1Mbit/s và số byte dữ liệu nhiều hơn 8 byte trong một khung truyền dữ liệu.

CAN FD sử dụng chung lớp vật lý với giao thức CAN quy định trong CAN Specification 2.0 (CAN 2.0).

Định dạng khung truyền của CAN FD khác với CAN 2.0. Nó có thêm 2 bit điều khiển mới. Một bit cho phép định dạng khung truyền với mã độ dài dữ liệu (DLC) mới và một bit cho phép tùy chọn chuyển sang tốc độ bit nhanh hơn sau khi sự phân xử đã được thực hiện xong.

Các đa thức CRC mới cùng được giới thiệu để đảm bảo cho các khung CAN FD dài hơn với khoảng cách Hamming giống như giao thức CAN đã chứng minh.

Định dạng khung CAN FD được định nghĩa để có thể cùng hoạt động với định dạng CAN 2.0 trong cùng một mạng. Giao thức CAN 2.0 vẫn hợp lệ và không có bất cứ điều chỉnh nào như một giao thức độc lập. Việc hai giao thức CAN 2.0 và CAN FD có thể cùng tồn tại trong một mạng được đảm bảo bởi yêu cầu là một thành phần CAN FD được xây dựng buộc phải tương thích với cả CAN FD và CAN 2.0.

Bên cạnh đó, để tương thích với CAN FD thì một thành phần CAN FD được xây dựng phải tương thích với mô tả của chuẩn CAN FD và với ISO 11898-1.

**Chú ý**: CAN FD có thể giao tiếp với CAN 2.0 miễn là không sử dụng định dạng khung của CAN FD. Nghĩa là, sử dụng định dạng khung của CAN 2.0.

### Định dạng khung truyền

Chuẩn CAN FD quy định 4 định dạng khung truyền là:

* (1) Định dạng CAN cơ bản (CAN Base Format): 11 bit ID + tốc độ bit hằng
* (2) Định dạng CAN mở rộng (CAN Extended Format): 29 bit ID + tốc độ bit hằng
* (3) Định dạng CAN FD cơ bản (CAN FD Base Format): 11 bit ID + tốc độ bit kép
* (4) Định dạng CAN FD mở rộng (CAN FD Extended Format): 29 bit ID + tốc độ bit kép

***Từ đây về sau, khi nói đến định dạng CAN thì đó là CAN 2.0.***

Tốc độ bit hằng tốc độ bit không thay đổi trong suốt quá trình truyền thông điệp. Tốc độ bit kép là trong một khung truyền Node CAN có thể truyền 2 tốc độ bit khác nhau.

Định dạng (1) và (2) như định dạng chuẩn (Standard Format) và định dạng mở rộng (Extended Format) của CAN 2.0.

### Loại khung truyền

CAN FD có 4 loại khung truyền như CAN 2.0 là khung dữ liệu, khung yêu cầu, khung báo lỗi và khung báo quá tải. CAN FD khác CAN 2.0 ở các điểm sau:

Ngoài 2 định dạng "CAN cơ bản" và "CAN mở rộng" như CAN 2.0, khung dữ liệu có thêm 2 dạng:

* Khung dữ liệu dạng CAN FD cơ bản
* Khung dữ liệu dạng CAN FD mở rộng

Khung yêu cầu chỉ có hai định dạng như CAN 2.0 chứ không có định dạng CAN FD cơ bản hay CAN FD mở rộng.

Khung báo quá tải ngoài chức năng tạo thêm trễ giữa các khung dữ liệu và khung yêu cầu thì còn thêm chức năng "đồng bộ sự phát hiện trạng thái bus rảng - IDLE".

### Cấu tạo khung dữ liệu

Cấu tạo của loại khung này tạo nên sự khác biệt giữa CAN FD và CAN 2.0. Khác biệt ở đây được hiểu là sự mở rộng của CAN FD so với CAN 2.0 chứ không phải đối lập.

#### Khung dữ liệu

#### Vùng phân xử

Vị trí bit RTR (Remote Transmission Request) dùng để xác định khung dữ liệu (dominant) hoặc khung yêu cầu (recessive) chỉ dành cho định dạng CAN cơ bản, nó tương ứng với định dạng khung chuẩn - Standard Format của CAN 2.0.

Với CAN FD cơ bản, bit RTR chuyển thành bit r1 và luôn là bit dominant. Điều này tương ứng với việc định dạng CAN FD chỉ dành cho khung dữ liệu, không dành cho khung yêu cầu.

#### Vùng điều khiển

Vùng điều khiển của định dạng CAN FD có cấu tạo rất khác so với CAN 2.0

CAN FD có thêm 3 bit mới là:

* EDL (Extended Data Length) là bit recessive. Đây là bit cho phép độ dài dữ liệu mở rộng. Bit này dùng để phân bit dữ định dạng khung CAN 2.0 và CAN FD.
* BRS (Baud Rate Switch) là bit dùng để quyết định việc thay đổi tốc độ bit trong định dạng CAN FD. Nếu bit này là recessive thì sau "pha phân xử" (Arbitration Phase), tốc độ bit hiện tại sẽ được chuyển thành một tốc độ bit khác đã được cấu hình trước, tốc độ bit mới được dùng cho pha dữ liệu (Data Phase). Chú ý, ở CAN FD, có thêm 2 khái niệm mới là pha phân xử và pha dữ liệu. Hai khái niệm này khác với hai khái niệm "vùng phần xử" (Arbitration Field) và vùng dữ liệu (Data Field).
* ESI (Error State Indicator) là cờ chỉ trạng thái lỗi được truyền là dominant nếu Node đang trong trạng thái tích cực với lỗi (Active Error) và truyền là recessive nếu Node đang trong trạng thái lỗi bị động (Passive Error)

|  |
| --- |
|  |
| **Hình 1.** So sánh giữa vùng phân xử và vùng điều khiển trong khung dữ liệu của CAN 2.0 định dạng cơ bản và CAN FD định dạng cơ bản |

|  |
| --- |
|  |
| **Hình 2.** So sánh giữa vùng phân xử và vùng điều khiển trong khung dữ liệu của CAN 2.0 định dạng mở rộng và CAN FD định dạng mở rộng |

Với bit SRR, CAN FD cho phép Node là bộ nhận cho phép bit SRR là dominant. Điều này không được nhắc đến trong CAN 2.0.

Bốn bit độ dài dữ liệu DLC trong CAN FD được mở rộng thêm. Trong khi CAN 2.0 chỉ sử dụng các giá trị từ 0 đến 8 tương ứng từ 0 đến 8 byte dữ liệu, DLC lớn hơn 8 cũng được xem như là 8 byte, thì CAN FD sử dụng các giá trị từ 9 đến 15 để tăng số byte vùng dữ liệu lên đến 64 byte cho phép truyền được nhiều dữ liệu hơn trong 1 lần phát.

|  |
| --- |
|  |
| **Hình 2.** Bảng mã DLC của CAN 2.0 và CAN FD |

#### Vùng dữ liệu

Định dạng CAN FD cho phép tối đa có thể lên đến 64 byte.

#### Vùng CRC:

Vì độ dài vùng dữ liệu tăng lên nên tương ứng vùng CRC cũng được điều chỉnh lại cho phù hợp

Vùng CRC không có độ dài cố định như CAN 2.0 mà thay đổi tùy theo độ dài vùng dữ liệu. Cụ thể:

* Khung theo định dạng CAN sẽ dùng CRC\_15 có đa thức sinh như chuẩn CAN 2.0 => độ dài CRC SEQUENCE là 15 bit
* Khung theo định dạng CAN FD có độ dài từ 16 byte trở xuống sử dụng CRC\_17 => độ dài CRC SEQUENCE là 17 bit
* Khung theo định dạng CSN FD có độ dài trên 16 byte sẽ sử dụng CRC\_21 => độ dài CRC SEQUENCE là 21 bit

|  |
| --- |
|  |
| Hình 3. Các đa thức dinh trong chuẩn CAN FD |

Mỗi đa thức sinh đều cho khoảng cách mã Hamming HD = 6.

Tại thời điểm bắt đầu của một khung, tất cả các Node đều phải tính cả 3 loại CRC cùng lúc, kể cả bộ truyền dữ liệu. Một Node thắng phân xử sẽ chọn CRC phù hợp với cấu hình bit EDL và độ dài dữ liệu DLC của khung truyền. Các bộ nhận cũng sẽ dựa trên thông tin của khung truyền để chọn và kiểm tra CRC phù hợp.

Trong định dạng CAN FD, CRC được tính cho chuỗi bit đã được áp dụng luật chèn bit. Đây là điểm khác biệt so với định dạng CAN vì CRC của định dạng này không bao gồm các bit chèn.

Đoạn mã giả minh họa việc tính CRC như sau:

CRC\_RG = 0; // initialize shift register

REPEAT

CRCNXT = NXTBIT EXOR CRC\_RG(nCRC-1);

CRC\_RG(nCRC-1:1) = CRC\_RG(nCRC-2:0); // shift left by 1 position

CRC\_RG(0) = 0;

IF CRCNXT THEN

CRC\_RG(nCRC-1:0) = CRC\_RG(nCRC-1:0) EXOR (CRC polynomial);

ENDIF

UNTIL (CRC SEQUENCE starts or there is an ERROR condition)

CRC Delimiter chỉ là 1 bit recessive đối với định dạng CAN nhưng nó có thể là 1 hoặc 2 bit recessive trong định dạng CAN FD. Cụ thể, trong định dạng CAN FD, bộ truyền chỉ truyền 1 bit recessive nhưng phải chấp nhận có 2 bit recessive trước khi phát hiện cạnh từ recessive chuyển sang dominant của ACK SLOT. Một bộ nhận sẽ vẫn gửi ACK của nó sau bit recessive đầu tiên của CRC Delimiter.

|  |
| --- |
|  |
| Hình 4. vùng CRC |

**Chú ý: CAN FD controller, bộ điều khiển thực hiện giao thức CAN FD, chuyển từ pha dữ liệu trở lại pha phân xử khi đên điểm lấy mẫu bit đầu tiên của CRC Delimiter**

**e) Vùng ACK**

|  |
| --- |
|  |
| **Hình 5.** Vùng ACK |

Định dạng CAN FD chấp nhận vùng ACK SLOT có độ dài tói đa 2 bit để bù sự dịch pha giữa các bộ nhận trong khi định dạng CAN chỉ chấp nhận 1 bit ACK và coi bit ACK thứ 2 là một lỗi định dạng.

ACK Delimiter cũng được tăng lên 2 bit recessive trong định dạng CAN FD.

### Pha phân xử và pha dữ liệu

Đây là hai khái niệm mới trong CAN FD. Trong đó:

* Pha dữ liệu (Data Phase) được tính từ điểm lấy mẫu được bit BRS (Bit Rate Switch) đến điểm lấy mẫu được bit đầu tiên của CRC Delimiter. Đây là vùng sẽ được chuyển sang tốc độ bit khác nếu bit BRS là recessive
* Pha phân xử (Arbitration Phase) là các vùng còn lại, bao gồm từ SÒ đến điểm lấy mẫu bit BRS và từ điểm lấy mẫu bit đầu tiên của CRC Delimiter đến kết thúc khung.

*\*Xem lại các hình mình họa phía trên để hình dung rõ hơn về các pha này*

### Cấu tạo khung yêu cầu

Tương tự CAN 2.0

### Cấu tạo khung báo lỗi và báo quá tải

Tương tự CAN 2.0

### Cấu tạo khoảng liên khung

Tương tự CAN 2.0

### Tính nhất quán của dữ liệu

Cấu trúc phân lớp của CAN FD theo mô hình tham khảo OSI.

|  |
| --- |
|  |
| **Hình 1.** Cấu trúc phân lớp của CAN FD trong mô hình tham khảo OSI |

Tính nhất quán của dữ liệu được thể hiện qua việc dữ liệu được đảm bảo đồng nhất khi truyền qua các lớp này. Đồng nhất (consistency) được hiểu là dữ liệu được phát đi là dữ liệu đúng, đủ và kịp thời như mong muốn.

Như đã trình bày ở các bài [cấu trúc phân lớp của CAN](http://nguyenquanicd.blogspot.com/2017/06/cancontroller-area-networkbai-3-cac.html), CAN controller là thành phần sẽ thực hiện các xử lý ở lớp LLC (Logical Link Control) và lớp MAC (Medium Access Control).

Trong một [Node](http://nguyenquanicd.blogspot.com/2017/06/cancontroller-area-netwwork-cau-truc-va.html), các thông điệp để truyền được chuẩn bị trước bởi host, host ở đây là đơn vị điều khiển chính của Node, có thể là CPU, FPGA,... Các thông điệp này được chuyển đến lớp phụ LLC (Logical Link Control) thông qua giao tiếp giữa host và thành phần CAN controller để đến lớp phụ MAC (Medium Access Control). Lớp MAC chịu trách nhiệm tạo ra khung thông điệp theo đúng chuẩn. Nếu host và CAN controller là 2 chip độc lập thì giao tiếp giữa chúng có thể là SPI, UART, parallel port,… Nếu CAN controller được tích hợp trong một MCU thì giao tiếp giữa chúng là giao tiếp nội thông qua bus hệ thống như APB, AHB, AXI, …

Các thông điệp có thể được lưu trữ trong một bộ nhớ được chia sẻ (shared memory), ví dụ như RAM. Việc truy xuất dữ liệu từ RAM có thể làm sai dữ liệu nên cần có cơ chế để đảm bảo tính đúng đắn của dữ liệu cần truyền lấy từ RAM. Tính đồng nhất của các thông điệp được truyền lấy từ một bộ nhớ được chia sẻ phải được đảm bảo bởi ít nhất bằng một trong 2 cách sau:

* Lớp MAC sẽ lưu trữ toàn bộ thông điệp được truyền đi trong một bộ đệm tạm trước khi bắt đầu truyền. Nghĩa là, dữ liệu đúng đã được chứa đầy đủ tại lớp MAC sẵn sàng cho việc truyền.
* Lớp LLC sẽ kiểm tra các lỗi dữ liệu trong khi thông điệp được gửi đến lớp MAC. Nếu lỗi xuất hiện, thông điệp sẽ không được truyền. Nếu một lỗi được phát hiện khi việc truyền đã bắt đầu thì Node phải được chuyển sang chế độ giám sát bus (Bus Monitoring Mode) và các Node nhận sẽ không thể thấy được một thông điệp hợp lệ để nhận.

Lỗi dữ liệu được nhắc đến ở bên trên có thể là lỗi parity khi kiểm tra dữ liệu đọc ra từ RAM, lỗi không cung cấp dữ liệu kịp thời cho CAN controller xử lý, hoặc một phần dữ liệu đã được cập nhật mới trong khi đang truyền, …

Có thể thấy, CAN FD làm rõ hơn vấn đề thực thi ở phần cứng CAN controller với những quy định cụ thể về kiểm soát tính đồng nhất khi truyền dữ liệu.

### Các trạng thái hoạt động

Một Node CAN FD có bốn trạng thái hoạt động là *Integrating (trạng thái chờ tích hợp), Idle (Rảnh), Receiver (Nhận) và Transmitter (Truyền)*. Việc quy định về bốn trạng thái hoạt động không xung đột với chuẩn CAN 2.0 mà chỉ làm rõ hơn hoạt động cơ bản của một Node.

*Integrating* là trạng thái chờ tích hợp. Việc gọi là chờ tích hợp vì đây là trạng thái mà một Node chờ để phát hiện chuỗi có 11 bit recessive liên tiếp sau khi Node này được khởi động hoặc Node này đang trong quá trình khôi phục hoạt động từ [bus-off sang Idle](http://nguyenquanicd.blogspot.com/2017/06/canbai7errorstatus.html).

Sau khi một Node mới được khởi động, hoặc khởi động lại (reset) hoặc được cấu hình bởi host để khơi động lạ thì nó không tham gia ngay vào các hoạt động trên mạng mà cần thời gian để đồng bộ với các hoạt động trên mạng, Node phải chờ sai khi phát hiện được một chuỗi 11 bit recessive liên tiếp thì mới bắt đầu tham gia vào hoạt động của mạng. Quy định 11 bit recessive liên tiếp đảm bảo lúc này bus CAN đã IDLE. Xem lại cấu trúc các khung của CAN, phần khung có kết thúc dài nhất là [khung báo lỗi hoặc báo quá tải](http://nguyenquanicd.blogspot.com/2017/06/cancontroller-area-networkbai-6-khung.html) với 8 bit recessive. Trường hợp xấu nhất là ngay sau đó là khoảng liên khung với độ dài phần Intermission là 3 bit recessive. Như vậy, việc chờ 11 bit recessive sẽ đảm bảo bus đang IDLE. Chú ý, phần Suspend Transmission của khoảng liên khung chỉ dành cho bộ truyền đang trong trạng thái lỗi bị động nên một Node mới khởi động.

|  |
| --- |
|  |
| **Hình 2.** Giải thích trường hợp Node phải chờ 11 bit recessive trước khi tham gia hoạt động của bus |

Một Node đang hoạt động trên bus khi rơi vào trạng thái bus-off thì phải giám sát bus cho đến khi phát hiện được 128 chuỗi 11 bit recessive thì mới được tham gia trở lại vào hoạt động của bus.

*Idle* là trạng thái rảnh. Node trong trạng thái này sẵn sàng truyền hoặc nhận SOF. Node sẽ chuyển từ trạng thái này sang trạng thái là bộ truyền (Transmitter) hoặc nhận (Receiver).

*Receiver* là trạng thái Node phát hiện hoạt động trên bus CAN ví dụ như SOF nhưng nó không đang truyền bất cứ thông điệp nào.

*Transmitter* là trạng thái mà Node khởi tạo truyền thông điệp trên bus. Node sẽ giữ trạng thái này cho đến khi nó thua phân xử hoặc đến khi bus Idle.

|  |
| --- |
|  |
| Hình 3. Các trạng thái hoạt động cơ bản của một Node CAN FD |

*Lịch sử cập nhật:*  
1) 2019.10.18 - Chỉnh sửa link hình minh họa

[Gửi email bài đăng này](https://www.blogger.com/share-post.g?blogID=6059866044676831944&postID=8871056198472755033&target=email)[BlogThis!](https://www.blogger.com/share-post.g?blogID=6059866044676831944&postID=8871056198472755033&target=blog)[Chia sẻ lên Twitter](https://www.blogger.com/share-post.g?blogID=6059866044676831944&postID=8871056198472755033&target=twitter)[Chia sẻ lên Facebook](https://www.blogger.com/share-post.g?blogID=6059866044676831944&postID=8871056198472755033&target=facebook)

#### Bài viết khác:

* [[CANFD][Controller Area Network][Bài 1] CANFD - Sự khác nhau giữa CAN FD và CAN 2.0 - phần 1](https://nguyenquanicd.blogspot.com/2017/06/can20controller-area-networkbai-12.html)Tham khảo: CAN with Flexible Data-Rate Specification Version 1.0 (released April 17th, 2012) by BOSCH Nội dung phần này chỉ nói về điểm khác biệt của CAN FD so với CAN Specification 2.0 (CAN 2.0). Bạn đọc hãy tham khảo về… [Đọc thêm](https://nguyenquanicd.blogspot.com/2017/06/can20controller-area-networkbai-12.html)
* [[CAN2.0][Controller Area Network][Bài 8] Luật truyền chuỗi bit của CAN - Bit Stream Coding](https://nguyenquanicd.blogspot.com/2017/06/can20controller-area-networkbai-8-luat.html)1. Loại mã hóa chuỗi bit Phương pháp mã hóa được sử dụng trong giao thức CAN là phương pháp NRZ mà cụ thể hơn là NRZ-L (Non-Return-to-Zero Level). Đây là dạng mã hóa theo mức, tức là mỗi trạng thái dominant và recesive sẽ đ… [Đọc thêm](https://nguyenquanicd.blogspot.com/2017/06/can20controller-area-networkbai-8-luat.html)
* [[CANFD][Controller Area Network][Bài 2] CANFD - Sự khác nhau giữa CAN FD và CAN 2.0 - phần 2](https://nguyenquanicd.blogspot.com/2017/06/can20controller-area-networkbai-12_30.html)Tham khảo: CAN with Flexible Data-Rate Specification Version 1.0 (released April 17th, 2012) by BOSCH Nội dung phần này chỉ nói về điểm khác biệt của CAN FD so với CAN Specification 2.0 (CAN 2.0). Bạn đọc hãy tham kh… [Đọc thêm](https://nguyenquanicd.blogspot.com/2017/06/can20controller-area-networkbai-12_30.html)
* [[CAN2.0][Controller Area Network][Bài 9] Quy định về thời gian bit - Bit Timing](https://nguyenquanicd.blogspot.com/2017/06/can20controller-area-networkbai-8-quy.html)1. Tốc độ bit danh định (Nominal Bit Rate) Tốc độ bit danh định (fn) là số bit được truyền trong một giây (bps - bit per second) bằng bộ truyền lý tưởng và không có sự tái đồng bộ xảy ra. Sự tái đồng bộ cho phép hiệu chỉn… [Đọc thêm](https://nguyenquanicd.blogspot.com/2017/06/can20controller-area-networkbai-8-quy.html)
* [[CANFD][Controller Area Network][Bài 3] CANFD - Sự khác nhau giữa CAN FD và CAN 2.0 - phần 3](https://nguyenquanicd.blogspot.com/2017/06/canfdcontroller-area-networkbai-3-canfd.html)Tham khảo: CAN with Flexible Data-Rate Specification Version 1.0 (released April 17th, 2012) by BOSCH Nội dung phần này chỉ nói về điểm khác biệt của CAN FD so với CAN Specification 2.0 (CAN 2.0). Bạn đọc hãy tham kh… [Đọc thêm](https://nguyenquanicd.blogspot.com/2017/06/canfdcontroller-area-networkbai-3-canfd.html)

### [ANFD][Controller Area Network][Bài 3] CANFD - Sự khác nhau giữa CAN FD và CAN 2.0 - phần 3](https://nguyenquanicd.blogspot.com/2017/06/canfdcontroller-area-networkbai-3-canfd.html)

Tác giả [Nguyễn Quân](https://www.blogger.com/profile/11739939602738380602) at 20:33 [CAN](https://nguyenquanicd.blogspot.com/search/label/CAN?&max-results=5), [CANFD](https://nguyenquanicd.blogspot.com/search/label/CANFD?&max-results=5), [Controller Area Network](https://nguyenquanicd.blogspot.com/search/label/Controller%20Area%20Network?&max-results=5), [Kiến Thức Cơ Bản](https://nguyenquanicd.blogspot.com/search/label/Ki%E1%BA%BFn%20Th%E1%BB%A9c%20C%C6%A1%20B%E1%BA%A3n?&max-results=5), [Protocol](https://nguyenquanicd.blogspot.com/search/label/Protocol?&max-results=5) [0 bình luận](https://nguyenquanicd.blogspot.com/2017/06/canfdcontroller-area-networkbai-3-canfd.html#comment-form)

Tham khảo: CAN with Flexible Data-Rate Specification Version 1.0 (released April 17th, 2012) by BOSCH

**Nội dung phần này chỉ nói về điểm khác biệt của CAN FD so với CAN Specification 2.0 (CAN 2.0).** [**Bạn đọc hãy tham khảo về CAN2.0 trước khi đọc bài này**](http://nguyenquanicd.blogspot.com/search/label/CAN?&max-results=5)**.**

**1. Luật chèn Bit (Stuffing)**

Khác với CAN 2.0, luật chèn bit áp dụng cho các vùng từ SOF đến CRC Sequence. Định dạng khung truyền CAN FD có sự thay đổi về luật chèn bit cho vùng CRC Sequence. Các bit được chèn thêm ở vùng này chỉ nằm ở các vị trí cố định.

Một bit cố định được chèn ở trước bit đầu tiên của CRC Sequence cho dù các bit cuối cùng của vùng trước đó chưa đáp ứng đúng theo điều kiện của luật chèn bit, nghĩa là chưa xuất hiện 5 bit cùng mức logic liên tiếp nhau. Sau đó, mỗi bit chèn sẽ được thêm sau mỗi nhóm 4 bit của chuỗi CRC Sequence. Giá trị của bit chèn là đảo của giá trị bit đứng ngay trước nó.

Bộ nhận sẽ bỏ qua các bit chèn khi nhận CRC. Bộ nhận sẽ phát hiện ra lỗi chèn bit (Stuff Error) nếu phát hiện ra bit được chèn có giá trị giống giá trị bit ngay trước nó. Số bit chèn cố định trong CRC định dạng CAN FD bằng với số tối đa của các bit chèn khi áp dụng luật chèn bit của định dạng CAN (5 bit cùng mức thì chèn 1 bit đảo).

|  |
| --- |
|  |
| Hình 1. So sánh số bit chèn trên CRC15 giữa CAN FD và một trường hợp chèn tối đa theo luật chèn bit CAN 2.0 |

**2. Các loại lỗi**

Các loại lỗi được phát hiện trong CAN FD tương tự như CAN 2.0 nhưng lỗi định dạng (Form Error) bổ sung thêm lỗi khi phát hiện lỗi ở bit chèn cố định của vùng CRC Sequence.

**3. Quy định về thời gian bit**

Do các khung truyền định dạng CAN FD có thể hỗ trợ 2 tốc độ bit khác nhau, 1 cho pha phân xử (Arbitration phase) và 1 cho pha dữ liệu (Data phase), nên trong phần cứng CAN controller sẽ có 2 bộ thanh ghi cấu hình về thời gian bit cho 2 tốc độ này.

Tương tự CAN 2.0, thời gian của một bit vẫn luôn gồm 4 đoạn là Sync\_seg, Prop\_seg, Phase\_seg1 và Phase\_seg2. Điểm lấy mẫu (sample point) vẫn ở cuối Phase\_seg1. Nhưng vì CAN FD hỗ trợ 2 tốc độ bit nên định nghĩa thêm thuật ngữ để phân biệt 2 tốc độ bit này.

Thời gian bit danh định (Nominal Bit Time) là khái niệm dành cho pha phân xử. Đối với pha dữ liệu thì thời gian lý tưởng của 1 bit gọi là thời gian bit dữ liệu (Data Bit Time).

TIME QUANTUM(N) = m(N) \* MINIMUM TIME QUANTUM

TIME QUANTUM(D) = m(D) \* MINIMUM TIME QUANTUM

* MINIMUM TIME QUANTUM (TQ tối thiểu): là giá trị thời gian tối thiểu được chia từ xung clock cấp cho CAN controller
* m: là giá trị của bộ đếm số TQ tối thiểu
* TQ: Thời gian dùng để tính độ dài 1 bit

Như vậy, độ dài đơn vị thời gian TQ của pha phân xử TQ(N) và pha dữ liệu TQ(D) có thể khác hoặc bằng nhau.

Tương tự, tốc độ bit danh định (Norminal Bit Rate) dùng để nói về tốc độ bit của pha phân xử ở CAN FD còn đối với pha dữ liệu, tốc độ này được gọi là tốc độ bit dữ liệu (Data Bit Rate).  
  
Đối với tốc độ bit danh định, độ dài các đoạn thời gian của một bit được quy định như sau:

* Sync\_seg: 1 TQ(N)
* Prop\_seg: 1 đến 32 TQ(N)
* Phase\_seg1: 1 đến 32 TQ(N)
* Phase\_seg2: max(Phase\_seg1, Thời gian lan truyền thông tin IPT)

Đối với tốc độ bit dữ liệu, độ dài các đoạn thời gian của 1 bit giống quy định của chuẩn CAN 2.0 như sau:

* Sync\_seg: 1 TQ(D)
* Prop\_seg: 1 đến 8 TQ(D)
* Phase\_seg1: 1 đến 8 TQ(D)
* Phase\_seg2: max(Phase\_seg1, Thời gian lan truyền thông tin IPT)

*Lịch sử cập nhật:*

1) 2019.10.18 - Chỉnh sửa link hình minh họa

[Gửi email bài đăng này](https://www.blogger.com/share-post.g?blogID=6059866044676831944&postID=4089419343842975161&target=email)[BlogThis!](https://www.blogger.com/share-post.g?blogID=6059866044676831944&postID=4089419343842975161&target=blog)[Chia sẻ lên Twitter](https://www.blogger.com/share-post.g?blogID=6059866044676831944&postID=4089419343842975161&target=twitter)[Chia sẻ lên Facebook](https://www.blogger.com/share-post.g?blogID=6059866044676831944&postID=4089419343842975161&target=facebook)

#### Bài viết khác:

* [[CAN2.0][Controller Area Network][Bài 4] Cấu trúc khun](https://nguyenquanicd.blogspot.com/2017/06/cancontroller-area-networkbai-4-cac.html)

# CANFD INTERFACE (RS-CANFD)

This section contains a generic description of the CANFD interface (RS-CANFD).

The first part of this section describes all specific properties, such as the number of units, register base addresses, etc. The remainder of the section describes the functions and registers of RS-CANFD.

### Features

#### Number of Units and Channels

#### Functional Overview

**Table 102.1 RS-CANFD Module Specifications**

|  |  |
| --- | --- |
| **Item** | **Specification** |
| Communication | CAN functionality conform to CAN-FD ISO 11898-1 (2015) |
| Protocol engine Version | RS-CANFD\_PE V3.0 |
| Gateway Function | CAN 2.0 <> CAN 2.0 CAN 2.0 <> CAN-FD Gateway (Only 8 Byte Payload) CAN-FD <> CAN-FD |
| Data transfer rate | up to 1Mbps for arbitration phase and up to 8Mbps for data phase , individually for each CAN channel |
| Proposed min. operation frequency peripheral clock/APB clock | 133.33MHz |
| Data Link Layer clock (DLLC) | 40MHz(Max) |
| Input/Output pins | TX/RX |
| CAN channels | 8 |
| Selectable ID type | 11-bit Standard ID |
|  | 11-bit Standard ID + 18-bit Extended ID |
| Selectable Frame Type | Data Frame (RTR = 0) (CAN and CAN-FD frames) |
|  | Remote Frame (RTR = 1) (only CAN frames) |
| Variable Data Byte Count for Data Frames | DLC range: 0 to F |
| Message Buffer | 64 transmit message buffers per channel \*\*1 4 transmission queue per channel Automatic message transfer into transmission queues supported |
|  | 256 shared buffers for RXMB and FIFO buffers per channel \*\*1 - RXMB: - Up to 16\*(n+1) reception message buffers, shared among all the CAN channels - FIFO buffers: - 8 Reception FIFO Buffers - Up to 3\*(n+1) FIFOs individually configurable as Reception FIFO / Transmission FIFO / CAN to CAN Gateway FIFO |
| Automatic delay interval timer for transmission | The delay timer can be applied to: -Transmission FIFO -CAN to CAN Gateway FIFO |
| Enhanced reception filtering | support of 11bit and 29bit CAN identifier |
|  | programmable 29 bit CAN identifier acceptance filter mask for each entry |
|  | programmable GW routing capability for each channel (up to 8 routing destinations) |
|  | RTR and IDE masking |
|  | DLC filter |
|  | Message buffer payload overload protection |
|  | Payload filter |
|  | Updating AFL entry during communication |
| General SW Support | Automatic label information added to receive message (for upper SW layer support) |
| Timer | TX and RX Time Stamp function |
| Power down function | Module start stop function for each CAN node (Channel & Global Sleep Mode) |
| RAM | RAM ECC protected |
| Bus traffic measurement | CAN bus traffic measurement of each Channel is possible |

#### Features of RS-CANFD

RS-CANFD has two interface modes (classical CAN mode and CAN FD mode), and uses different registers in each  
mode. The different register names RSCFD{unit}XXX and RSCFD{unit}CFDXXX are used in the different interface  
modes (XXX can be any character). In this document, the registers common to the two modes are indicated as  
RSCFD{unit}CFDXXX.

##### Number of Units and Channels

This microcontroller has the following number of RS-CANFD units.

**Number of Units and Channels**

|  |  |
| --- | --- |
| **Product Name** | **R-Car V4H** |
| Number of Units | 1 |
| Number of Channels | 8 |
| Number of Channel per Unit | 8 |
| Transmission Message Buffers per Channel | 64 |
| Shared Buffers for per Channel | 256 |
| Total Message Buffer | 2560 |

**Table 102.1 Index**

|  |  |
| --- | --- |
| **Index** | **Description** |
| unit | Throughout this section, the individual RS-CANFD units are generically indicated by the index “unit”. For example, RSCFD[unit]CFDGCTR is the global control register of the RSCFDunit unit. |
| n | Throughout this section, the individual channels of RS-CANFD units are generically indicated by the index “n”. For example, RSCFD[unit]CFDCnSTS is the channel n status register. |
| m | The individual channels of RSCFD unit is generically indicated by the index “m”. For example, CANFDm\_RX is receive data input. |
| d | The individual common FIFO buffers are generically indicated by the index “d”. For example, RSCFD[unit]CFDCFCCd is the common FIFO buffer configuration/control register. |
| a | The individual receive FIFO buffers are generically identified by the index “a”. For example, RSCFD[unit]CFDRFSTSa is the receive FIFO buffer status register. |
| p | Data field registers of common FIFO buffers and receive FIFO buffers are identified by “p”. For example, RSCFD[unit]CFDCFDFp is the common FIFO access data field register. |
| i | The individual transmit control buffers are generically indicated by the index “i”. For example, RSCFD[unit]CFDTMCi is transmit message buffer control register. |
| j | The individual transmit status buffers are generically indicated by the index “j”. For example, RSCFD[unit]CFDTMSTSj is transmit message buffer status register. |
| k | The individual RAM tests for CAN are generically indicated by the index “k”. For example, RSCFD[unit]CFDRPGACCk is the RAM test page access register. |
| f | The transmit message buffer transmission are generically indicated by index “f”. For example, RSCFD[unit]CFDTMTRSTSf is the transmit message buffer transmission request status register. |
| w | A global register which setting for channels indicated by the index "w". For example, RSCFD[unit]CFDGAFLCFGw is global Acceptance Filter List configuration register. |
| v | A global register which setting for 04 channels indicated by the index “v”. For example, RSCFD[unit]CFDGTINTSTSv is the global transmit interrupt status register. |
| t | The receive message buffer new data are generically indicated by index “t”. For example, RSCFD[unit]CFDRMNDt is a receive message buffer new data register. |

Note 1. The functions and descriptions of registers in this section are for the RS-CANFDs that has 8 channels (m = 0 to 7).  
When referring to information with indices, regard the index values as the ones corresponding to the target product.  
Note 2. In some figure or table, this section use acronyms: CH, Ch, Ch instead of "channel".

**Table 102.2 Indices for Individual Products**

|  |  |
| --- | --- |
| **Index Correspondence of Each Product** | **R-CarV4H** |
| unit | 0 |
| n | 0 to 7 |
| m | 0 to 7 |
| d | 0 to 23 |
| a | 0 to 7 |
| p | 0 to 15 |
| i | 0 to 511 |
| j | 0 to 511 |
| k | 0 to 63 |
| f | 0 to 15 |
| w | 0 to 3 for RSCFD[unit]CFDGAFLCFGw 0 to 1 for RSCFD[unit]CFDGPFLCFGw |
| v | 0 to 1 |
| t | 0 to 3 |

#### **Interface Mode**

RS-CANFD has the following two interface modes.  
• Classical CAN mode: Only classical CAN frames are handled.  
• CAN FD mode: both the classical CAN frames and CAN FD frames are handled.  
Two modes use different register maps with the same base address, and the register maps are switched by switching the modes.  
Interface modes can be switched using the CLOE bit in the RSCFD{unit}CFDCnFDCFG register.

#### Block Diagram

The block diagram which shows connection with a peripheral module is shown

A diagram of a computer

Description automatically generated

**Figure 102.1 RS-CANFD Module Block Diagram (in classical CAN mode)**

CANFDɸ must be set as follows.  
R-Car V4H: 80(MHz)

Peripheral bus clock (pclk): SASYNCPERD2ɸ  
clkc: CANFDɸ  
clk\_xincan: CAN\_CLK  
clk\_ram: SASYNCPERD1ɸ

#### External Pins

Table 102.3 shows the CAN module pin.  
Pin switching is required for pins which are multiplexed with other Function. For details, see section 6, Pin Function  
Controller (PFC).  
**Table 102.3 Pin Configuration**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Abbreviation** | **I/O** | **Function** |
| CANFDm\_RX | — | Input | Pins for receiving data |
| CANFDm\_TX | — | Output | Pins for transmitting data |
| CAN\_CLK | — | Input | Input pin used for external clock input. |

#### Connected module

**Table 102.4 Connected module**

|  |  |  |
| --- | --- | --- |
| Module name | Connected module name | function of the related module |
| CANFD | AXI | Register access of CPU |
| CPG | Clock output |  |
| PFC | Selection of external pins |  |
| Module Standby | Clock stop control |  |
| Software Reset | Soft reset execution |  |
| INTC | Interruption |  |
| SYS-DMAC | DMA transmission |  |
| CRC | CRC module |  |
| MFIS | MFIS modul |  |

##### Register Base Address

RSCFD{unit} base addresses are listed in the following table.  
RSCFD{unit} register addresses are given as offsets from the base addresses in general.  
**Table 102.5 Register Base Address**

|  |  |
| --- | --- |
| **Base Address Name** | **Base Address** |
| <RSCFD0\_base> | H'E666 0000 |

#### Register Configuration

This section describes all registers of RS-CANFD

##### List of Registers

RSCFD registers are listed in the table below.

For details about Register Base Addresses.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Register Name** | **Symbol RSCFD0+{registers of RS-CANFD}** | **R/W (Refer to Figure 4.2)** | **Address** | **Access Size** |
| Channel n Nominal Bitrate Configuration Register | RSCFD0CFDCnNCFG | - | <RSCFD0\_base> + 0000h + n\*0010h | 8, 16, 32 |
| Channel n Control Registers | RSCFD0CFDCnCTR | - | <RSCFD0\_base> + 0004h + n\*0010h | 8, 16, 32 |
| Channel n Status Registers | RSCFD0CFDCnSTS | - | <RSCFD0\_base> + 0008h + n\*0010h | 8, 16, 32 |
| Channel n Error Flag Registers | RSCFD0CFDCnERFL | - | <RSCFD0\_base> + 000Ch + n\*0010h | 8, 16, 32 |
| Global IP Version Register | RSCFD0CFDGIPV | - | <RSCFD0\_base> + 0080h | 8, 16, 32 |
| Global Configuration Register | RSCFD0CFDGCFG | - | <RSCFD0\_base> + 0084h | 8, 16, 32 |
| Global Control Register | RSCFD0CFDGCTR | - | <RSCFD0\_base> + 0088h | 8, 16, 32 |
| Global Status Register | RSCFD0CFDGSTS | - | <RSCFD0\_base> + 008Ch | 8, 16, 32 |
| Global Error Flag Register | RSCFD0CFDGERFL | - | <RSCFD0\_base> + 0090h | 8, 16, 32 |
| Global Timestamp Counter Register | RSCFD0CFDGTSC | - | <RSCFD0\_base> + 0094h | 16, 32 |
| Global Acceptance Filter List Entry Control Register | RSCFD0CFDGAFLECTR | - | <RSCFD0\_base> + 0098h | 8, 16, 32 |
| Global Acceptance Filter List Configuration Register w | RSCFD0CFDGAFLCFGw | - | <RSCFD0\_base> + 009Ch + w\*0004h | 8, 16, 32 |
| RX Message Buffer Number Register | RSCFD0CFDRMNB | - | <RSCFD0\_base> + 00Ach | 8, 16, 32 |
| RX Message Buffer New Data Register t | RSCFD0CFDRMNDt | - | <RSCFD0\_base> + 00B0h + t\*0004h | 8, 16, 32 |
| RX FIFO Configuration / Control Registers a | RSCFD0CFDRFCCa | - | <RSCFD0\_base> + 00C0h + a\*0004h | 8, 16, 32 |
| RX FIFO Status Registers a | RSCFD0CFDRFSTSa | - | <RSCFD0\_base> + 00E0h + a\*0004h | 8, 16, 32 |
| RX FIFO Pointer Control Registers a | RSCFD0CFDRFPCTRa | - | <RSCFD0\_base> + 0100h + a\*0004h | 8, 16, 32 |
| Common FIFO Configuration / Control Registers d | RSCFD0CFDCFCCd | - | <RSCFD0\_base> + 0120h + d\*0004h | 8, 16, 32 |
| Common FIFO Configuration / Control Enhancement Registers d | RSCFD0CFDCFCCEd | - | <RSCFD0\_base> + 0180h + d\*0004h | 8, 16, 32 |
| Common FIFO Status Registers d | RSCFD0CFDCFSTSd | - | <RSCFD0\_base> + 01E0h + d\*0004h | 8, 16, 32 |
| Common FIFO Pointer Control Registers d | RSCFD0CFDCFPCTRd | - | <RSCFD0\_base> + 0240h + d\*0004h | 8, 16, 32 |
| FIFO Empty Status Register | RSCFD0CFDFESTS | - | <RSCFD0\_base> + 02A0h | 8, 16, 32 |
| FIFO Full Status Register | RSCFD0CFDFFSTS | - | <RSCFD0\_base> + 02A4h | 8, 16, 32 |
| FIFO Message Lost Status Register | RSCFD0CFDFMSTS | - | <RSCFD0\_base> + 02A8h | 8, 16, 32 |
|  |  |  |  |  |
|  |  |  |  |  |
| RX FIFO Interrupt Flag Status Register | RSCFD0CFDRFISTS | - | <RSCFD0\_base> + 02Ach | 8, 16, 32 |
| Common FIFO RX Interrupt Flag Status Register | RSCFD0CFDCFRISTS | - | <RSCFD0\_base> + 02B0h | 8, 16, 32 |
| Common FIFO TX Interrupt Flag Status Register | RSCFD0CFDCFTISTS | - | <RSCFD0\_base> + 02B4h | 8, 16, 32 |
| Common FIFO One Frame RX Interrupt Flag Status Register | RSCFD0CFDCFOFRISTS | - | <RSCFD0\_base> + 02B8h | 8, 16, 32 |
| Common FIFO One Frame TX Interrupt Flag Status Register | RSCFD0CFDCFOFTISTS | - | <RSCFD0\_base> + 02BCh | 8, 16, 32 |
| Common FIFO Message Over Write Status Register | RSCFD0CFDCFMOWSTS | - | <RSCFD0\_base> + 02C0h | 8, 16, 32 |
| FIFO FDC Full Status Register | RSCFD0CFDFFFSTS | - | <RSCFD0\_base> + 02C4h | 8, 16, 32 |
| TX Message Buffer Control Registers i | RSCFD0CFDTMCi | - | <RSCFD0\_base> + 02D0h + i\*0001h | 8 |
| TX Message Buffer Status Registers j | RSCFD0CFDTMSTSj | - | <RSCFD0\_base> + 07D0h + j\*0001h | 8 |
| TX Message Buffer Transmission Request Status Register f | RSCFD0CFDTMTRSTSf | - | <RSCFD0\_base> + 0CD0h + f\*0004h | 8, 16, 32 |
| TX Message Buffer Transmission Abort Request Status Register f | RSCFD0CFDTMTARSTSf | - | <RSCFD0\_base> + 0D70h + f\*0004h | 8, 16, 32 |
| TX Message Buffer Transmission Completion Status Register f | RSCFD0CFDTMTCSTSf | - | <RSCFD0\_base> + 0E10h + f\*0004h | 8, 16, 32 |
| TX Message Buffer Transmission Abort Status Register f | RSCFD0CFDTMTASTSf | - | <RSCFD0\_base> + 0EB0h + f\*0004h | 8, 16, 32 |
| TX Message Buffer Interrupt Enable Configuration Register f | RSCFD0CFDTMIECf | - | <RSCFD0\_base> + 0F50h + f\*0004h | 8, 16, 32 |
| TX Queue Configuration / Control Registers 0 [n] | RSCFD0CFDTXQCC0[n] | - | <RSCFD0\_base> + 1000h + n\*0004h | 8, 16, 32 |
| TX Queue Status Registers 0 [n] | RSCFD0CFDTXQSTS0[n] | - | <RSCFD0\_base> + 1020h + n\*0004h | 8, 16, 32 |
| TX Queue Pointer Control Registers 0 [n] | RSCFD0CFDTXQPCTR0[n] | - | <RSCFD0\_base> + 1040h + n\*0004h | 8, 16, 32 |
| TX Queue Configuration / Control Registers 1 [n] | RSCFD0CFDTXQCC1[n] | - | <RSCFD0\_base> + 1060h + n\*0004h | 8, 16, 32 |
| TX Queue Status Registers 1 [n] | RSCFD0CFDTXQSTS1[n] | - | <RSCFD0\_base> + 1080h + n\*0004h | 8, 16, 32 |
| TX Queue Pointer Control Registers 1 [n] | RSCFD0CFDTXQPCTR1[n] | - | <RSCFD0\_base> + 10A0h + n\*0004h | 8, 16, 32 |
| TX Queue Configuration / Control Registers 2 [n] | RSCFD0CFDTXQCC2[n] | - | <RSCFD0\_base> + 10C0h + n\*0004h | 8, 16, 32 |
| TX Queue Status Registers 2 [n] | RSCFD0CFDTXQSTS2[n] | - | <RSCFD0\_base> + 10E0h + n\*0004h | 8, 16, 32 |
| TX Queue Pointer Control Registers 2 [n] | RSCFD0CFDTXQPCTR2[n] | - | <RSCFD0\_base> + 1100h + n\*0004h | 8, 16, 32 |
| TX Queue Configuration / Control Registers 3 [n] | RSCFD0CFDTXQCC3[n] | - | <RSCFD0\_base> + 1120h + n\*0004h | 8, 16, 32 |
| TX Queue Status Registers 3 [n] | RSCFD0CFDTXQSTS3[n] | - | <RSCFD0\_base> + 1140h + n\*0004h | 8, 16, 32 |
| TX Queue Pointer Control Registers 3 [n] | RSCFD0CFDTXQPCTR3[n] | - | <RSCFD0\_base> + 1160h + n\*0004h | 8, 16, 32 |
| TX Queue Empty Status Register | RSCFD0CFDTXQESTS | - | <RSCFD0\_base> + 1180h | 8, 16, 32 |
| TX Queue Full Interrupt Status Register | RSCFD0CFDTXQFISTS | - | <RSCFD0\_base> + 1184h | 8, 16, 32 |
| TX Queue Message Lost Status Register | RSCFD0CFDTXQMSTS | - | <RSCFD0\_base> + 1188h | 8, 16, 32 |
| TX Queue Message Overwrite Status Register | RSCFD0CFDTXQOWSTS | - | <RSCFD0\_base> + 118Ch | 8, 16, 32 |
| TX Queue Interrupt Status Register | RSCFD0CFDTXQISTS | - | <RSCFD0\_base> + 1190h | 8, 16, 32 |
| TX Queue One Frame TX Interrupt Status Register | RSCFD0CFDTXQOFTISTS | - | <RSCFD0\_base> + 1194h | 8, 16, 32 |
| TX Queue One Frame RX Interrupt Status Register | RSCFD0CFDTXQOFRISTS | - | <RSCFD0\_base> + 1198h | 8, 16, 32 |
| TX Queue Full Status Register | RSCFD0CFDTXQFSTS | - | <RSCFD0\_base> + 119Ch | 8, 16, 32 |
| TX History List Configuration / Control Register n | RSCFD0CFDTHLCCn | - | <RSCFD0\_base> + 1200h + n\*0004h | 8, 16, 32 |
| TX History List Status Register n | RSCFD0CFDTHLSTSn | - | <RSCFD0\_base> + 1220h + n\*0004h | 8, 16, 32 |
| TX History List Pointer Control Registers n | RSCFD0CFDTHLPCTRn | - | <RSCFD0\_base> + 1240h + n\*0004h | 8, 16, 32 |
| Global TX Interrupt Status Register v | RSCFD0CFDGTINTSTSv | - | <RSCFD0\_base> + 1300h + v\*4 | 8, 16, 32 |
| Global Test Configuration Register | RSCFD0CFDGTSTCFG | - | <RSCFD0\_base> + 1308h | 8, 16, 32 |
| Global Test Control Register | RSCFD0CFDGTSTCTR | - | <RSCFD0\_base> + 130Ch | 8, 16, 32 |
| Global FD Configuration register | RSCFD0CFDGFDCFG | - | <RSCFD0\_base> + 1314h | 8, 16, 32 |
| Global FD CRC Configuration register | RSCFD0CFDGCRCCFG | - | <RSCFD0\_base> + 1318h | 8, 16, 32 |
| Global Lock Key Register | RSCFD0CFDGLOCKK | - | <RSCFD0\_base> + 131Ch | 16, 32 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| Global AFL Ignore Entry Register | RSCFD0CFDGAFLIGNENT | - | <RSCFD0\_base> + 1324h | 8, 16, 32 |
| Global AFL Ignore Control Register | RSCFD0CFDGAFLIGNCTR | - | <RSCFD0\_base> + 1328h | 16, 32 |
| DMA Transfer Control Register | RSCFD0CFDCDTCT | - | <RSCFD0\_base> + 1330h | 8, 16, 32 |
| DMA Transfer Status Register | RSCFD0CFDCDTSTS | - | <RSCFD0\_base> + 1334h | 8, 16, 32 |
| DMA TX Transfer Control Register | RSCFD0CFDCDTTCT | - | <RSCFD0\_base> + 1340h | 8, 16, 32 |
| DMA TX Transfer Status Register | RSCFD0CFDCDTTSTS | - | <RSCFD0\_base> + 1344h | 8, 16, 32 |
| Global RX Interrupt Status Register n | RSCFD0CFDGRINTSTSn | - | <RSCFD0\_base> + 1350h + n\*0004h | 8, 16, 32 |
| Pretended Network Filter List Entry control Register | RSCFD0CFDGPFLECTR | - | <RSCFD0\_base> + 1370h | 8, 16, 32 |
| Pretended Network Filter List Entry Configuration Register u | RSCFD0CFDGPFLCFGu | - | <RSCFD0\_base> + 1374h + u \* 0004h | 8, 16, 32 |
| Global SW reset Register | RSCFD0CFDGRSTC | - | <RSCFD0\_base> + 1380h | 16, 32 |
| Global Flexible CAN mode Configuration Register | RSCFD0CFDGFCMC | - | <RSCFD0\_base> + 1384h | 8, 16, 32 |
| Global Flexible transmission buffer assignment Configuration Register | RSCFD0CFDGFTBAC | - | <RSCFD0\_base> + 138Ch | 8, 16, 32 |
| Global Virtual Machine Mode configuration Register | RSCFD0CFDGFFIMC | - | <RSCFD0\_base> + 1390h | 16, 32 |
| Global Virtual Machine Error Interrupt Select Register | RSCFD0CFDGVMEIS | - | <RSCFD0\_base> + 1394h | 8, 16, 32 |
| Global Virtual Machine Common FIFO TXQ configuration Register n | RSCFD0CFDVMCFGn | - | <RSCFD0\_base> + 13A0h + n\*0004h | 8, 16, 32 |
| Global Virtual Machine RX FIFO configuration Register | RSCFD0CFDVMRFCFG | - | <RSCFD0\_base> + 13C0h | 8, 16, 32 |
| Virtual Machine Interrupt Status Register n | RSCFD0CFDVMISTSn | - | <RSCFD0\_base> + 13E0h + n\*0004h | 8, 16, 32 |
| Channel n Data Bitrate Configuration Register | RSCFD0CFDCnDCFG | - | <RSCFD0\_base> + 1400h + n\*0020h | 8, 16, 32 |
| Channel n CAN-FD Configuration Register | RSCFD0CFDCnFDCFG | - | <RSCFD0\_base> + 1404h + n\*0020h | 8, 16, 32 |
| Channel n CAN-FD Control Register | RSCFD0CFDCnFDCTR | - | <RSCFD0\_base> + 1408h + n\*0020h | 8, 16, 32 |
| Channel n CAN-FD Status Register | RSCFD0CFDCnFDSTS | - | <RSCFD0\_base> + 140Ch + n\*0020h | 8, 16, 32 |
| Channel n CAN-FD CRC Register | RSCFD0CFDCnFDCRC | - | <RSCFD0\_base> + 1410h + n\*0020h | 8, 16, 32 |
| Channel n Bus load Control Register | RSCFD0CFDCnBLCT | - | <RSCFD0\_base> + 1418h + n\*0020h | 8, 16, 32 |
| Channel n Bus load Status Register | RSCFD0CFDCnBLSTS | - | <RSCFD0\_base> + 141Ch + n\*0020h | 8, 16, 32 |
| Global Acceptance Filter List ID Registers r = [1…10]h | RSCFD0CFDGAFLIDr | - | <RSCFD0\_base> + 1800h + (r-1)\*0010h | 8, 16, 32 |
| Global Acceptance Filter List Mask Registers r = [1…10]h | RSCFD0CFDGAFLMr | - | <RSCFD0\_base> + 1804h + (r-1)\*0010h | 8, 16, 32 |
| Global Acceptance Filter List Pointer 0 Registers r = [1…10]h | RSCFD0CFDGAFLP0r | - | <RSCFD0\_base> + 1808h + (r-1)\*0010h | 8, 16, 32 |
| Global Acceptance Filter List Pointer 1 Registers r = [1…10]h | RSCFD0CFDGAFLP1r | - | <RSCFD0\_base> + 180Ch + (r-1)\*0010h | 8, 16, 32 |
| Global Pretended Network Filter List ID Registers s = [1…4] | RSCFD0CFDGPFLIDs | - | <RSCFD0\_base> + 1A00h + (s-1) \* 0040h | 8, 16, 32 |
| Global Pretended Network Filter List Mask Registers s = [1...4] | RSCFD0CFDGPFLMs | - | <RSCFD0\_base> + 1A04h + (s-1) \* 0040h | 8, 16, 32 |
| Global Pretended Network Filter List Pointer 0 Registers s = [1...4] | RSCFD0CFDGPFLP0s | - | <RSCFD0\_base> + 1A08h + (s-1) \* 0040h | 8, 16, 32 |
| Global Pretended Network Filter List Pointer 1 Registers s = [1...4] | RSCFD0CFDGPFLP1s | - | <RSCFD0\_base> + 1A0Ch + (s-1) \* 0040h | 8, 16, 32 |
| Global Pretended Network Filter List Filter Payload Type Registers s = [1...4] | RSCFD0CFDGPFLPTs | - | <RSCFD0\_base> + 1A10h + (s-1) \* 0040h | 8, 16, 32 |
| Global Pretended Network Filter List Payload Data 0 Registers s = [1...4] | RSCFD0CFDGPFLPD0s | - | <RSCFD0\_base> + 1A14h + (s-1) \* 0040h | 8, 16, 32 |
| Global Pretended Network Filter List Payload Mask 0 Registers s = [1...4] | RSCFD0CFDGPFLPM0s | - | <RSCFD0\_base> + 1A18h + (s-1) \* 0040h | 8, 16, 32 |
| Global Pretended Network Filter List Payload Data 1 Registers s = [1...4] | RSCFD0CFDGPFLPD1s | - | <RSCFD0\_base> + 1A1Ch + (s-1) \* 0040h | 8, 16, 32 |
| Global Pretended Network Filter List Payload Mask 1 Registers s = [1...4] | RSCFD0CFDGPFLPM1s | - | <RSCFD0\_base> + 1A20h + (s-1) \* 0040h | 8, 16, 32 |
| Channel n TX History List Access Registers 0 | RSCFD0CFDTHLACC0[n] | - | <RSCFD0\_base> + 8000h + n\*0008h | 8, 16, 32 |
| Channel n TX History List Access Registers 1 | RSCFD0CFDTHLACC1[n] | - | <RSCFD0\_base> + 8004h + n\*0008h | 8, 16, 32 |
| RAM Test Page Access Registers k | RSCFD0CFDRPGACCk | - | <RSCFD0\_base> + 8400h + k\*0004h | 8, 16, 32 |
| RX Message Buffer ID Registers | RSCFD0CFDRMID | - | <RSCFD0\_base> + Refer to Figure 4.2 | 8, 16, 32 |
| RX Message Buffer Pointer Registers | RSCFD0CFDRMPTR | - | <RSCFD0\_base> + Refer to Figure 4.2 | 8, 16, 32 |
| RX Message Buffer CAN-FD Status Register | RSCFD0CFDRMFDSTS | - | <RSCFD0\_base> + Refer to Figure 4.2 | 8, 16, 32 |
|  |  |  |  |  |
|  |  |  |  |  |
| RX Message Buffer Data Field p Registers | RSCFD0CFDRMDFp | - | <RSCFD0\_base> + Refer to Figure 4.2 | 8, 16, 32 |
| RX FIFO Access ID Registers | RSCFD0CFDRFID | - | <RSCFD0\_base> + Refer to Figure 4.2 | 8, 16, 32 |
| RX FIFO Access Pointer Register | RSCFD0CFDRFPTR | - | <RSCFD0\_base> + Refer to Figure 4.2 | 8, 16, 32 |
| RX FIFO Access CAN-FD Status Register | RSCFD0CFDRFFDSTS | - | <RSCFD0\_base> + Refer to Figure 4.2 | 8, 16, 32 |
| RX FIFO Access Data Field p Registers | RSCFD0CFDRFDFp | - | <RSCFD0\_base> + Refer to Figure 4.2 | 8, 16, 32 |
| Common FIFO Access ID Registers | RSCFD0CFDCFID | - | <RSCFD0\_base> + Refer to Figure 4.2 | 8, 16, 32 |
| Common FIFO Access Pointer Registers | RSCFD0CFDCFPTR | - | <RSCFD0\_base> + Refer to Figure 4.2 | 8, 16, 32 |
| Common FIFO Access CAN-FD Control/Status Register | RSCFD0CFDCFFDCSTS | - | <RSCFD0\_base> + Refer to Figure 4.2 | 8, 16, 32 |
| Common FIFO Access Data Field p Registers | RSCFD0CFDCFDFp | - | <RSCFD0\_base> + Refer to Figure 4.2 | 8, 16, 32 |
| TX Message Buffer ID Registers | RSCFD0CFDTMID | - | <RSCFD0\_base> + Refer to Figure 4.2 | 8, 16, 32 |
| TX Message Buffer Pointer Registers | RSCFD0CFDTMPTR | - | <RSCFD0\_base> + Refer to Figure 4.2 | 8, 16, 32 |
| TX Message Buffer CAN-FD Control Register | RSCFD0CFDTMFDCTR | - | <RSCFD0\_base> + Refer to Figure 4.2 | 8, 16, 32 |
| TX Message Buffer Data Field p Registers | RSCFD0CFDTMDFp | - | <RSCFD0\_base> + Refer to Figure 4.2 | 8, 16, 32 |

### Register Description

#### Details of Interface Mode Related Registers

Refer to the section 4 of “102\_uciaprcn0140\_IPSpec\_v1p06.pdf”

### Operation

Refer to the section 5 to 18 of “102\_uciaprcn0140\_IPSpec\_v1p06.pdf”

### Usage Notes

No description on this section.

### Safety Mechanisms

#### ECC for CAN-FD (IP uciapecc7w10)

##### Overview

ECC IP – Function block diagram:

A diagram of a program code

Description automatically generated

**Figure 102.2 ECC Block diagram**EDC protection is applied to internal buffer (Acceptance filter list RAM, Message Buffer RAM) of CAN-FD.  
**(1) Error Detection and Correction**-Seven-bit ECC data is appended to the 32-bit RAM data.  
-This ECC module provides 2-bit ECC error detection and 1-bit ECC error detection and correction.  
\*Note: This module is not capable of reliably detecting errors in three or more bits. If errors occur in three or more bits,  
the module may detect the errors as 1- or 2-bit ECC errors or not detect any errors. Depending on the settings, this may  
lead to the correction of a bit that was not actually inverted.  
**(2) Enabling or Disabling ECC Error Detection and Correction**-ECC error detection can be either enabled or disabled.  
-One-bit ECC error correction can be either enabled or disabled.  
-If all the bits of RAM output data are stuck to 0 or 1, it is detected as a 2-bit ECC error.  
**(3) Error Notification**-The ECM is notified when 2-bit ECC errors are detected (this can be enabled or disabled).  
-The ECM is notified when 1-bit ECC errors are detected (this can be enabled or disabled).  
-Once the ECM has been notified of an error, even if another ECC error is detected, the ECM is not notified until the error  
status bit corresponding to the initial error is cleared.  
**(4) Error Status**-Detection of 2-bit and 1-bit ECC errors can be monitored.  
-Special registers are provided to clear error status.  
**(5) Address Capture**-Only one address at which an ECC error has occurred can be captured.  
-When a 2-bit or 1-bit ECC error is detected, the error-causing address is captured. It is used to capture only when the  
first error is detected after the flag is cleared.  
**(6) Testing Function (Error Insertion)**-By setting the test mode, register values can be used as the data to be output to the RAM. When a peripheral module writes  
to the RAM, the ECEDB[31:0] register value can be written to the RAM data section, and the ECERDB[6:0] register  
value can be written to the ECC redundant bit section.  
-By setting the test mode, the ECC redundant bit section can be latched when RAM data is read, and the value can be  
confirmed.  
-By setting the test mode, the ECC redundant bit (encoding circuit) and syndrome code (decoding circuit), which are  
generated from the input data, can be confirmed.

##### Hardware Description

A diagram of a computer system

Description automatically generated

**Figure 102.3 EDC protection for CAN-FD**

**Table 102.6 Register Configuration**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Register Name |  | Abbreviation | R/W | Address | Initial Value | Size |
| ECC control register |  | EC710CTL | R/W | H’E66C\_8000 | H’0000 0000 | 32 |
| ECC test mode control register |  | EC710TMC | R/W | H’E66C\_8004 | H’0000 0000 | 32 |
| ECC redundant bit data control test register |  | EC710TRC | R | H’E66C\_8008 | H’0000 0000 | 32 |
| ECC encoder and decoder data test register |  | EC710TED | R/W | H’E66C\_800C | H’0000 0000 | 32 |
| ECC error address register |  | EC710EAD | R | H’E66C\_8010 | H’0000 0000 | 32 |