## Can Driver

A computer screen shot of a computer

Description automatically generated

### Giới thiệu và Tổng quan chức năng

Tài liệu này xác định chức năng, API và cấu hình của mô-đun CAN Driver trong AUTOSAR Basic Software (gọi là "mô-đun Can" trong tài liệu này).

Mô-đun Can là một phần của tầng thấp nhất, thực hiện việc truy cập phần cứng và cung cấp một API độc lập với phần cứng cho tầng trên.

Tầng trên duy nhất có quyền truy cập vào mô-đun Can là mô-đun CanIf (xem thêm SRS\_SPAL\_12092).

Mô-đun Can cung cấp dịch vụ để khởi tạo truyền và gọi hàm gọi lại của mô-đun CanIf để thông báo sự kiện, độc lập với phần cứng. Hơn nữa, nó cung cấp dịch vụ để kiểm soát hành vi và trạng thái của CAN controllers thuộc cùng một CAN Hardware Unit.

Nhiều CAN controllers có thể được kiểm soát bởi một mô-đun Can duy nhất miễn là chúng thuộc cùng một CAN Hardware Unit

Để có mô tả chi tiết hơn về CAN controller và CAN Hardware Unit, xem chương Viết tắt và biểu đồ trong [5].

### Từ viết tắt và viết tắt

|  |  |
| --- | --- |
| **Thuật ngữ** | **Mô tả** |
| CAN Controller | CAN Controller phục vụ chính xác một kênh vật lý. |
| CAN Hardware Unit | CAN Hardware Unit có thể bao gồm một hoặc nhiều CAN Controllers cùng loại và một hoặc nhiều vùng CAN RAM. CAN Hardware Unit nằm trên chip hoặc thiết bị bên ngoài. CAN Hardware Unit được đại diện bởi một CAN Driver. |
| CAN L-PDU | Data Link Layer Protocol Data Unit. Bao gồm Mã định danh (ID), Độ dài dữ liệu và Dữ liệu (SDU). (xem [18]) |
| CAN L-SDU | Data Link Layer Service Data Unit. Dữ liệu được vận chuyển bên trong L-PDU. (xem [18]) |
| DLC | Data Length Code (một phần của Message CAN mô tả độ dài SDU) |
| Hardware Object | Một CAN Hardware Object được định nghĩa là bộ đệm PDU bên trong CAN RAM của CAN Hardware Unit/ CAN Controller. Một CAN Hardware Object được định nghĩa là bộ đệm L-PDU bên trong RAM CAN của CAN Hardware Unit. |
| Hardware Receive Handle (HRH) | Hardware Receive Handle (HRH) được xác định và cung cấp bởi CAN Driver. Mỗi HRH thường chỉ đại diện cho một Hardware Object. HRH có thể được sử dụng để tối ưu hóa Software Filtering |
| Hardware Transmit Handle (HTH) | Hardware Transmit Handle (HTH) được xác định và cung cấp bởi CAN Driver. Mỗi HTH thường chỉ đại diện cho một hoặc nhiều Hardware Object được cấu hình như Hardware Transmit Buffer |
| Inner Priority Inversion | Việc truyền L-PDU có mức ưu tiên cao bị ngăn chặn bởi sự hiện diện của L-PDU có mức ưu tiên thấp đang chờ xử lý trong cùng một Hardware Object. |
| ISR | Interrupt Service Routine |
| L-PDU Handle | L-PDU handle được xác định và đặt bên trong lớp mô-đun CanIf. Thông thường, mỗi bộ điều khiển đại diện cho một L-PDU, là một cấu trúc cố định chứa thông tin dành cho quá trình xử lý Tx/Rx. |
| MCAL | Microcontroller Abstraction Layer |
| Outer Priority Inversion | Khoảng cách thời gian xảy ra giữa hai lần truyền L-PDU liên tiếp. Trong trường hợp này, L-PDU có mức ưu tiên thấp hơn từ nút khác có thể ngăn việc gửi L-PDU có mức ưu tiên cao hơn của chính nó. Ở đây, L-PDU có mức ưu tiên cao hơn không thể tham gia phân xử trong quá trình truy cập mạng vì L-PDU có mức ưu tiên thấp hơn đã thắng trong quá trình phân xử. |
| Physical Channel | Kênh vật lý đại diện cho giao diện từ CAN Controller đến CAN Network. Các kênh vật lý khác nhau của CAN Hardware Unit có thể truy cập các mạng khác nhau. |
| Priority | Mức độ ưu tiên của CAN L-PDU được thể hiện bằng CAN Identifier. Giá trị số của mã định danh càng thấp thì mức độ ưu tiên càng cao. |
| SFR | Special Function Register. Thanh ghi phần cứng điều khiển hành vi của bộ điều khiển. |
| SPAL | Standard Peripheral Abstraction Layer |
| ICOM | Intelligent Communication Controller |

#### Priority Inversion - Đảo ngược ưu tiên

A diagram of a data link

Description automatically generated

"Nếu chỉ có một bộ đệm truyền đơn được sử dụng, có thể xảy ra sự đảo ngược ưu tiên nội bộ. Do ưu tiên thấp, một thông điệp được lưu trữ trong bộ đệm phải chờ đợi cho đến khi 'lưu lượng trên bus ổn định'. Trong thời gian chờ đợi này, thông điệp này có thể ngăn cản một thông điệp có ưu tiên cao hơn được tạo ra bởi cùng một vi điều khiển từ việc được truyền qua bus."

A data link system

Description automatically generated with medium confidence

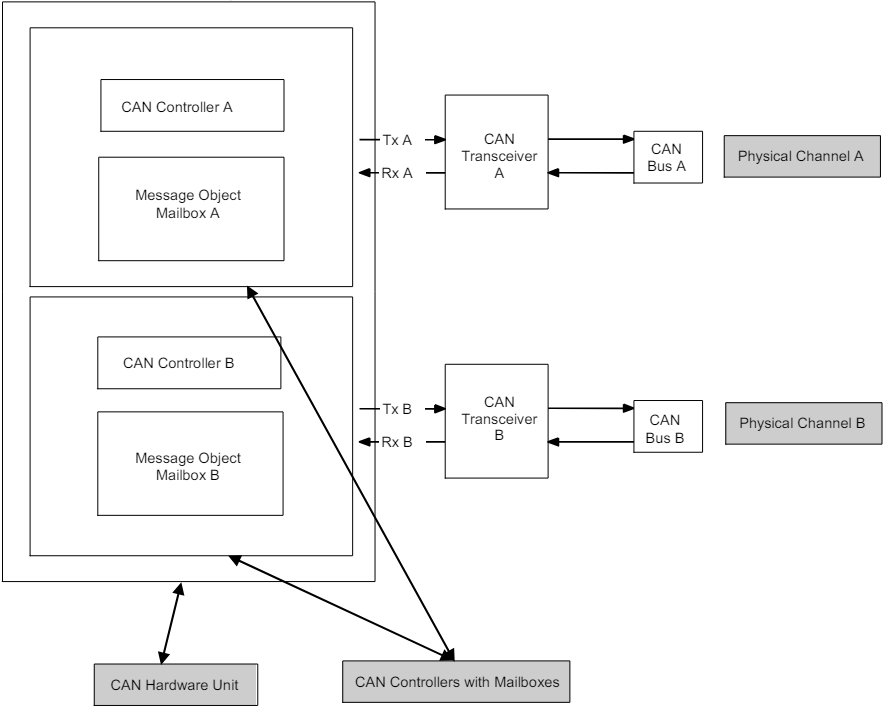
"Vấn đề của sự đảo ngược ưu tiên bên ngoài có thể xảy ra trong một số cài đặt CAN. Hãy giả sử rằng một nút CAN muốn truyền một gói tin của các thông điệp liên tiếp với ưu tiên cao, được lưu trữ trong các bộ đệm thông điệp khác nhau. Nếu khoảng thời gian giữa các thông điệp này trên mạng CAN dài hơn so với khoảng trống tối thiểu được xác định bởi tiêu chuẩn CAN, một nút thứ hai có thể bắt đầu truyền một thông điệp ưu tiên thấp hơn. Khoảng trống tối thiểu giữa các khung tin trên mạng CAN được xác định bởi trường Intermission, bao gồm 3 bit không quyết định. Một thông điệp, đợi trong quá trình truyền của một thông điệp khác, được bắt đầu trong khoảng thời gian Bus Idle, sớm nhất là trong bit tiếp theo sau trường Intermission. Ngoại lệ là một nút có thông điệp truyền đợi sẽ giải thích một bit động ở bit thứ ba của Intermission như một bit Start-of-Frame và bắt đầu truyền với bit nhận dạng đầu tiên mà không cần truyền trước một bit SOF. Thời gian xử lý nội bộ của một mô-đun CAN phải đủ ngắn để gửi các thông điệp liên tiếp với khoảng trống tối thiểu giữa các khung để tránh sự đảo ngược ưu tiên bên ngoài dưới tất cả các tình huống đã đề cập."

#### CAN Hardware Unit

CAN Hardware Unit kết hợp một hoặc nhiều CAN Controllers, có thể được đặt trên chip hoặc là các thiết bị độc lập bên ngoài cùng loại, với

Hardware Objects chung hoặc riêng lẻ.

Hình dưới đây thể hiện một CAN Hardware Unit bao gồm hai CAN controllers kết nối với hai Kênh Vật lý:



### Related documentation

### Constraints and assumptions

#### Limitations

Một CAN Controller luôn tương ứng với một kênh vật lý. Nó được phép kết nối các kênh vật lý ở phía bus. Mô-đun CanIf sẽ xử lý Bất kể các CAN Controllers liên quan một cách riêng biệt.

Một số ít đơn vị phần cứng CAN hỗ trợ khả năng kết hợp nhiều CAN Controllers bằng cách sử dụng CAN RAM, để mở rộng số lượng đối tượng thông điệp cho một CAN Controller. Các bộ điều khiển CAN kết hợp này được xử lý như một bộ điều khiển bởi mô-đun Can.

Mô-đun Can không hỗ trợ các khung CAN từ xa.

[SWS\_Can\_00237] ⌈ Mô-đun Can không được phép truyền các thông điệp được kích hoạt bởi yêu cầu truyền từ xa.⌋ (SRS\_Can\_01147)

[SWS\_Can\_00236] ⌈ Mô-đun Can phải khởi tạo CAN HW để bỏ qua bất kỳ yêu cầu truyền từ xa nào.⌋ (SRS\_Can\_01147)

#### Ứng dụng trong các lĩnh vực xe hơi

Mô-đun Can có thể được sử dụng cho bất kỳ ứng dụng nào, nơi giao thức CAN được sử dụng.

### Dependencies to other modules

#### Static Configuration

Các phần tử cấu hình mô tả trong chương 10 có thể được các mô-đun BSW khác tham chiếu đến cho cấu hình của chúng.

#### Driver Services

[SWS\_Can\_00238] ⌈ Nếu CAN controller nằm trên chip, mô-đun Can không được phép sử dụng bất kỳ dịch vụ của các trình điều khiển khác.⌋ (SRS\_BSW\_00005)

[SWS\_Can\_00239] ⌈ Hàm Can\_Init phải khởi tạo tất cả các tài nguyên phần cứng trên chip được sử dụng bởi CAN controller. Ngoại lệ duy nhất cho điều này là cấu hình chân I/O kỹ thuật số (của các chân được sử dụng bởi CAN), được thực hiện bởi trình điều khiển cổng.⌋ (SRS\_BSW\_00377)

[SWS\_Can\_00240] ⌈ Mô-đun Mcu (SPAL xem [8]) phải cấu hình các thiết lập của bộ đăng ký được 'chia sẻ' với các mô-đun khác.⌋ ()

Mẹo triển khai: Mô-đun Mcu phải được khởi tạo trước khi khởi tạo mô-đun Can.

[SWS\_Can\_00242] ⌈ Nếu CAN controller sử dụng, mô-đun Can phải sử dụng các dịch vụ của các trình điều khiển MCAL khác (ví dụ: SPI).⌋ (SRS\_BSW\_00005)

Mẹo triển khai: Nếu mô-đun Can sử dụng các dịch vụ của các trình điều khiển MCAL khác (ví dụ: SPI), phải đảm bảo rằng các trình điều khiển này đang hoạt động trước khi khởi tạo mô-đun Can. Thứ tự khởi tạo của các trình điều khiển khác nhau được chỉ định một phần trong [7].

[SWS\_Can\_00244] ⌈ Mô-đun Can phải sử dụng các API đồng bộ của các trình điều khiển MCAL cơ sở và không được cung cấp các hàm gọi lại có thể được gọi bởi các trình điều khiển MCAL.⌋ ()

Do đó, loại kết nối giữa vi điều khiển và Đơn vị Phần cứng CAN chỉ ảnh hưởng đến triển khai và không ảnh hưởng đến API.

#### System Services

[SWS\_Can\_00280] ⌈ Trong các trường hợp phần cứng đặc biệt, mô-đun Can phải đánh giá sự kiện của phần cứng.⌋ ()

[SWS\_Can\_00281] ⌈ Mô-đun Can phải sử dụng OsCounter được cung cấp bởi dịch vụ hệ thống để phát hiện timeout trong trường hợp phần cứng không phản ứng trong thời gian mong đợi (sự cố phần cứng) để ngăn chặn vòng lặp vô hạn.⌋ ()

Mẹo triển khai: Thời gian chặn của hàm mô-đun Can đang chờ phản ứng của phần cứng phải ngắn hơn chu kỳ kích hoạt của hàm chính CAN (tức là Can\_MainFunction\_Read), vì các chức năng chính CAN không thể được sử dụng cho mục đích đó.

#### Can module Users

[SWS\_Can\_00058] ⌈ Mô-đun Can tương tác với các mô-đun khác (ví dụ: Tracer Lỗi Mặc định (DET), Quản lý Trạng thái Ecu (ECUM)) với mô-đun CanIf một cách trực tiếp. Tài liệu này không bao giờ xác định nguồn gốc thực sự của một yêu cầu hoặc điểm đến thực sự của một thông báo. Trình điều khiển chỉ nhìn thấy mô-đun CanIf như là nguồn gốc và điểm đến.⌋ (SRS\_SPAL\_12092)

#### File structure

[SWS\_Can\_00436] ⌈ Can\_GeneralTypes.h phải chứa tất cả các loại và hằng số được chia sẻ giữa các mô-đun CAN AUTOSAR Can, CanIf và CanTrcv.⌋ ()

### Fsdaf

### Functional specification

### Dịch sang tiếng Việt:

Khi truyền L-PDU, mô-đun Can sẽ ghi L-PDU vào một bộ đệm thích hợp bên trong phần cứng của bộ điều khiển CAN. Xem chương 7.5 để biết mô tả chi tiết về việc truyền L-PDU.

Khi nhận L-PDU, mô-đun Can sẽ gọi hàm callback chỉ báo RX với các tham số là ID, Độ dài Dữ liệu và con trỏ đến L-SDU. Xem chương 7.6 để biết mô tả chi tiết về việc nhận L-PDU.

Mô-đun Can cung cấp một giao diện hoạt động như một hàm xử lý định kỳ, và phải được gọi bởi mô-đun Bộ lập lịch Phần mềm Cơ bản (Basic Software Scheduler) theo chu kỳ.

Hơn nữa, mô-đun Can cung cấp các dịch vụ để kiểm soát trạng thái của các bộ điều khiển CAN. Các sự kiện Bus-off và Wake-up được thông báo thông qua các hàm callback.

Mô-đun Can là một Mô-đun Phần mềm Cơ bản truy cập các tài nguyên phần cứng. Do đó, nó được thiết kế để đáp ứng các yêu cầu đối với các Mô-đun Phần mềm Cơ bản được quy định trong AUTOSAR\_SRS\_SPAL (xem [3]).

[SWS\_Can\_00033] ⌈ Mô-đun Can phải triển khai các hàm ngắt dịch vụ cho tất cả các ngắt Đơn vị Phần cứng CAN cần thiết. ⌋ (SRS\_BSW\_00164, SRS\_SPAL\_12129)

[SWS\_Can\_00419] ⌈ Mô-đun Can phải vô hiệu hóa tất cả các ngắt không sử dụng trong bộ điều khiển CAN. ⌋ ()

[SWS\_Can\_00420] ⌈ Mô-đun Can phải đặt lại cờ ngắt vào cuối hàm ISR (nếu không được tự động thực hiện bởi phần cứng). ⌋ ()

Gợi ý thực hiện: Mô-đun Can không được thiết lập cấu hình (ví dụ: ưu tiên) của bảng vectơ ngắt.

[SWS\_Can\_00079] ⌈ Mô-đun Can phải tuân thủ tất cả các hướng dẫn thiết kế và thực hiện được mô tả trong [2]. ⌋ (SRS\_BSW\_00007, SRS\_BSW\_00306, SRS\_BSW\_00308, SRS\_BSW\_00309, SRS\_BSW\_00330)

#### Driver scope

Một mô-đun Can cung cấp truy cập vào một Đơn vị Phần cứng CAN có thể bao gồm nhiều bộ điều khiển CAN.

[SWS\_Can\_00077] ⌈ Đối với các Đơn vị Phần cứng CAN của loại khác nhau, phải triển khai các mô-đun Can khác nhau. ⌋ (SRS\_BSW\_00347)

[SWS\_Can\_00284] ⌈ Trong trường hợp nhiều Đơn vị Phần cứng CAN (của cùng hoặc khác nhà sản xuất) được triển khai trong một ECU, tên hàm và biến toàn cục của các mô-đun Can phải được triển khai sao cho không có hai hàm cùng tên được tạo ra. ⌋ ()

Cách đặt tên tuân thủ như sau:

<Tên mô-đun Can>\_<vendorID>\_<Tên API cụ thể của nhà sản xuất><viết tắt trình điều khiển>()

SRS\_BSW\_00347 quy định cách đặt tên.

[SWS\_Can\_00385] ⌈ Cách đặt tên phải được sử dụng chỉ trong trường hợp, nếu cần hỗ trợ nhiều loại bộ điều khiển CAN khác nhau trên một ECU. ⌋ ()

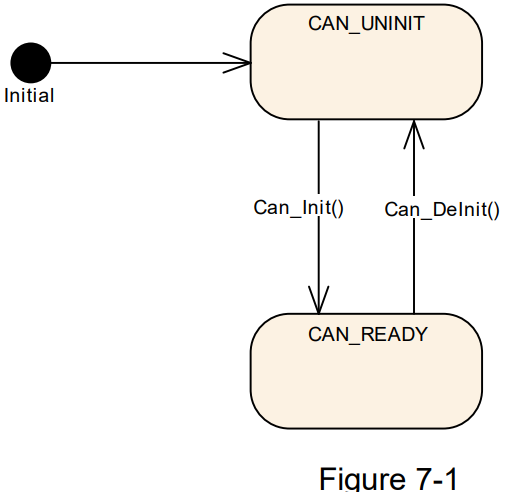
[SWS\_Can\_00386] ⌈ Nếu chỉ có một loại bộ điều khiển được sử dụng, các quy ước đặt tên ban đầu mà không có bất kỳ phần mở rộng viết tắt trình điều khiển nào cũng đủ.⌋ ()

Xem [5] để biết mô tả cách xử lý nhiều mô-đun Can bởi mô-đun CanIf.

#### Máy trạng thái Trình điều khiển

Mô-đun Can có một máy trạng thái rất đơn giản, với hai trạng thái CAN\_UNINIT và CAN\_READY. Hình 7.1 hiển thị máy trạng thái.

[SWS\_Can\_00103] ⌈ Sau khi mở nguồn/đặt lại, mô-đun Can phải ở trạng thái CAN\_UNINIT. ⌋ (SRS\_BSW\_00406)



[SWS\_Can\_00246] ⌈ Hàm Can\_Init phải thay đổi trạng thái của mô-đun thành CAN\_READY, sau khi khởi tạo tất cả các bộ điều khiển bên trong Đơn vị Phần cứng.⌋ (SRS\_SPAL\_12057, SRS\_Can\_01041)

[SWS\_Can\_00245] ⌈ Hàm Can\_Init phải khởi tạo tất cả các bộ điều khiển CAN theo cấu hình của chúng.⌋ (SRS\_SPAL\_12057, SRS\_Can\_01041)

Sau đó, mỗi bộ điều khiển CAN phải được khởi động riêng lẻ bằng cách gọi hàm Can\_SetControllerMode(CAN\_CS\_STARTED).

Gợi ý thực hiện:

Các thiết lập thanh ghi phần cứng có ảnh hưởng đến tất cả các bộ điều khiển CAN bên trong Đơn vị Phần cứng chỉ có thể được thiết lập trong hàm Can\_Init.

Gợi ý thực hiện:

Mô-đun Quản lý Trạng thái ECU phải gọi Can\_Init tối đa một lần trong quá trình chạy.

[SWS\_Can\_91009] ⌈ Hàm Can\_DeInit phải thay đổi trạng thái của mô-đun thành CAN\_UNINIT trước khi hủy khởi tạo tất cả các bộ điều khiển bên trong Đơn vị Phần cứng.⌋ (SRS\_Can\_01166)

Xem [SWS\_Can\_91010].

#### Máy trạng thái Bộ điều khiển CAN

Mỗi bộ điều khiển CAN có máy trạng thái phức tạp được triển khai trong phần cứng. Để đơn giản hóa, số lượng trạng thái được giảm xuống chỉ còn bốn trạng thái cơ bản sau đây trong mô tả này: UNINIT, STOPPED, STARTED và SLEEP.

Mọi truy cập phần cứng CAN được đóng gói bởi các hàm của mô-đun Can, nhưng mô-đun Can không ghi nhớ các thay đổi trạng thái.

Mô-đun Can cung cấp các dịch vụ Can\_Init, Can\_SetBaudrate và Can\_SetControllerMode. Các dịch vụ này thực hiện các thiết lập thanh ghi cần thiết gây ra sự thay đổi cần thiết của trạng thái bộ điều khiển CAN phần cứng.

Có hai cách để kích hoạt các thay đổi trạng thái bằng các sự kiện bên ngoài:

Sự kiện Bus-off

Sự kiện Wakeup phần cứng

Các sự kiện này được chỉ ra bằng một ngắt hoặc bằng một bit trạng thái được gạt trong hàm Can\_MainFunction\_BusOff hoặc Can\_MainFunction\_Wakeup.

Mô-đun Can thực hiện các thiết lập thanh ghi cần thiết để thực hiện hành vi yêu cầu (tức là không có khôi phục phần cứng trong trường hợp bus off).

Sau đó, nó thông báo cho mô-đun CanIf với hàm gọi lại tương ứng. Trạng thái phần mềm sau đó được thay đổi bên trong hàm gọi lại này.

Trong trường hợp lỗi phát triển được kích hoạt và có một chuyển đổi không được phép được yêu cầu bởi lớp trên, mô-đun Can phải tạo ra lỗi phát triển CAN\_E\_TRANSITION.

Mô-đun Can không kiểm tra trạng thái thực tế trước khi thực hiện Can\_Write hoặc gửi các hàm gọi lại.

##### Mô tả Trạng thái Bộ điều khiển CAN

Chương này mô tả hành vi phần cứng cần thiết cho các trạng thái bộ điều khiển khác nhau.

Trạng thái bộ điều khiển CAN UNINIT

Bộ điều khiển CAN không được khởi tạo. Tất cả các thanh ghi thuộc bộ điều khiển CAN đều ở trạng thái thiết lập lại, các ngắt CAN bị vô hiệu hóa. Bộ điều khiển CAN không tham gia vào bus CAN.

Trạng thái bộ điều khiển CAN STOPPED.

Trong trạng thái này, Bộ điều khiển CAN đã được khởi tạo nhưng không tham gia vào bus. Ngoài ra, các khung lỗi và xác nhận không được gửi đi.

(Ví dụ: Đối với nhiều bộ điều khiển, việc nhập vào chế độ 'khởi tạo' khiến cho bộ điều khiển bị dừng lại.)

Trạng thái của bộ điều khiển CAN STARTED

Bộ điều khiển đang ở chế độ hoạt động bình thường với đầy đủ chức năng, điều này có nghĩa là nó tham gia vào mạng. Đối với nhiều bộ điều khiển, việc rời khỏi chế độ 'khởi tạo' khiến cho bộ điều khiển được bắt đầu.

Trạng thái của bộ điều khiển CAN SLEEP

Các thiết lập phần cứng chỉ khác biệt so với trạng thái STOPPED đối với phần cứng CAN hỗ trợ chế độ ngủ (thức dậy qua bus CAN được hỗ trợ trực tiếp bởi phần cứng CAN).

[SWS\_Can\_00257] ⌈ Khi phần cứng CAN hỗ trợ chế độ ngủ và được kích hoạt để chuyển sang trạng thái SLEEP, mô-đun Can phải đặt bộ điều khiển vào trạng thái SLEEP từ đó phần cứng có thể được đánh thức qua Bus CAN. ⌋ (SRS\_SPAL\_12067)

[SWS\_Can\_00258] ⌈ Khi phần cứng CAN không hỗ trợ chế độ ngủ và được kích hoạt để chuyển sang trạng thái SLEEP, mô-đun Can phải mô phỏng một trạng thái SLEEP logic từ đó chỉ trở về khi được kích hoạt bởi phần mềm để chuyển sang trạng thái STOPPED. ⌋ ()

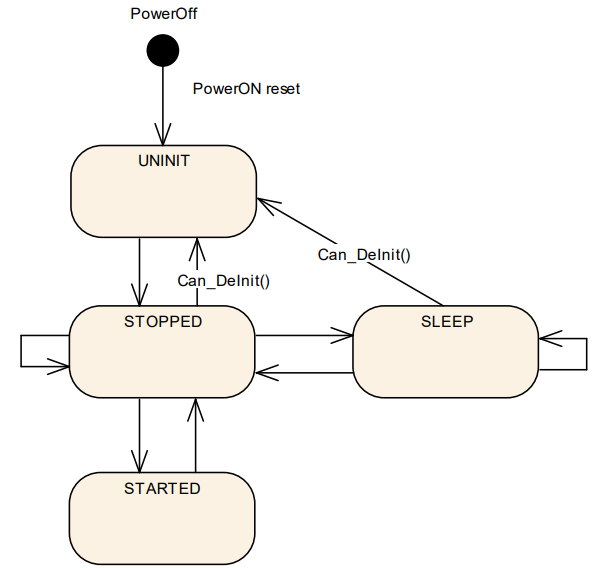
[SWS\_Can\_00404] ⌈ Phần cứng CAN phải duy trì ở trạng thái STOPPED trong khi trạng thái SLEEP logic đang hoạt động. ⌋ ()

##### Các Chuyển Đổi Trạng Thái Bộ Điều Khiển CAN

Một chuyển đổi trạng thái được kích hoạt bởi phần mềm với hàm Can\_SetControllerMode với chuyển đổi cần thiết làm tham số. Một chuyển đổi trạng thái thành công được kích hoạt bởi phần mềm được thông báo bằng hàm gọi lại (CanIf\_ControllerModeIndication). Việc theo dõi xem trạng thái yêu cầu đã đạt được là phần của một mô-đun lớp trên và không phải là phần của mô-đun Can.

Một số chuyển đổi được kích hoạt bởi các sự kiện trên bus (phần cứng). Những chuyển đổi này gây ra một thông báo thông qua một hàm gọi lại (CanIf\_ControllerBusOff, EcuM\_CheckWakeup).

Hành vi cho các chuyển đổi không hợp lệ trong mã sản xuất không xác định. Hình 7-2 hiển thị tất cả các chuyển đổi trạng thái hợp lệ.



Hình 7-2

##### Chuyển Đổi Trạng Thái Do Hàm Can\_Init Gây Ra

- UNINIT STOPPED (cho tất cả các bộ điều khiển trong đơn vị phần cứng)

- được kích hoạt bởi phần mềm thông qua cuộc gọi hàm Can\_Init

- thực hiện cấu hình cho tất cả các bộ điều khiển CAN bên trong Đơn vị Phần cứng

Tất cả các thanh ghi điều khiển được thiết lập theo cấu hình tĩnh.

[SWS\_Can\_00259] ⌈ Hàm Can\_Init phải đặt tất cả các bộ điều khiển CAN ở trạng thái STOPPED.⌋ ()

Khi hàm Can\_Init được gọi và mô-đun Can không ở trạng thái CAN\_UNINIT hoặc các bộ điều khiển CAN không ở trạng thái UNINIT, nó phải tạo ra lỗi CAN\_E\_TRANSITION (So sánh với SWS\_Can\_00174 và SWS\_Can\_00408).

##### Chuyển đổi trạng thái do hàm Can\_SetBaudrate gây ra

- STOPPED -> STOPPED; SLEEP -> SLEEP; STARTED -> STARTED

- được kích hoạt bởi phần mềm thông qua cuộc gọi hàm Can\_SetBaudrate

- thay đổi cấu hình của bộ điều khiển CAN

Các thanh ghi điều khiển của bộ điều khiển CAN được thiết lập theo cấu hình tĩnh.

[SWS\_Can\_00256] ⌈ Nếu cuộc gọi của Can\_SetBaudrate() gây ra một việc khởi tạo lại của Bộ điều khiển CAN và Bộ điều khiển CAN không ở trạng thái STOPPED, nó phải trả về E\_NOT\_OK.⌋ ()

[SWS\_Can\_00260] ⌈ Nếu việc khởi tạo lại cần thiết, hàm Can\_SetBaudrate phải duy trì Bộ điều khiển CAN ở trạng thái STOPPED.⌋ ()

[SWS\_Can\_00422] ⌈ Nếu việc khởi tạo lại cần thiết, hàm Can\_SetBaudrate phải đảm bảo rằng bất kỳ cài đặt nào sẽ khiến Bộ điều khiển CAN tham gia vào mạng không được thiết lập.⌋ ()

##### Chuyển đổi trạng thái do hàm Can\_SetControllerMode gây ra

Phần mềm có thể kích hoạt một chuyển đổi trạng thái của bộ điều khiển CAN với hàm Can\_SetControllerMode. Tùy thuộc vào phần cứng CAN, việc thay đổi cài đặt thanh ghi để chuyển sang một trạng thái mới của bộ điều khiển CAN có thể chỉ xảy ra sau một khoảng trễ. Mô-đun Can thông báo cho lớp trên (CanIf\_ControllerModeIndication) sau một chuyển đổi trạng thái thành công về trạng thái mới. Việc theo dõi xem trạng thái yêu cầu đã đạt được là phần của một mô-đun lớp trên và không phải là phần của mô-đun Can.

[SWS\_Can\_00370] ⌈ Hàm Can\_Mainfunction\_Mode phải điều tra một cờ của thanh ghi trạng thái CAN cho đến khi cờ báo hiệu rằng thay đổi có hiệu lực và thông báo cho lớp trên với hàm CanIf\_ControllerModeIndication về một chuyển đổi trạng thái thành công với thông tin về bộ điều khiển CAN tương ứng với CanIf ControllerId trừu tượng.⌋ ()

[SWS\_Can\_00398] ⌈ Hàm Can\_SetControllerMode phải sử dụng dịch vụ hệ thống GetCounterValue để theo dõi thời gian chờ để tránh chặn các hàm.⌋ ()

[SWS\_Can\_00372] ⌈ Trong trường hợp cờ báo hiệu rằng thay đổi không có hiệu lực và thời gian tối đa CanTimeoutDuration đã trôi qua, hàm Can\_SetControllerMode phải dừng và hàm Can\_Mainfunction\_Mode phải tiếp tục điều tra cờ báo hiệu.⌋ (SRS\_SPAL\_12077)

[SWS\_Can\_00373] ⌈ Hàm Can\_Mainfunction\_Mode phải gọi hàm CanIf\_ControllerModeIndication để thông báo cho lớp trên về một chuyển đổi trạng thái thành công của bộ điều khiển CAN tương ứng được chỉ định bởi CanIf ControllerId trừu tượng, trong trường hợp chuyển đổi trạng thái được kích hoạt bởi hàm Can\_SetControllerMode.⌋ ()

Chuyển đổi trạng thái do hàm Can\_SetControllerMode (CAN\_CS\_STARTED)

- STOPPED -> STARTED

- được kích hoạt bởi phần mềm

[SWS\_Can\_00261] ⌈ Hàm Can\_SetControllerMode(CAN\_CS\_STARTED) phải thiết lập các thanh ghi phần cứng một cách sao cho bộ điều khiển CAN tham gia vào mạng.⌋ ()

[SWS\_Can\_00262] ⌈ Hàm Can\_SetControllerMode(CAN\_CS\_STARTED) phải chờ một khoảng thời gian giới hạn cho đến khi bộ điều khiển CAN hoạt động hoàn toàn. So sánh với SWS\_Can\_00398.⌋ ()

Các yêu cầu truyền gửi được khởi tạo trước khi bộ điều khiển CAN hoạt động sẽ bị mất. Duy nhất chỉ có việc tiếp nhận các xác nhận truyền hoặc chỉ định nhận. Các thực thể gửi có thể nhận thấy một sự chậm trễ xác nhận và cần phải xử lý được điều đó.

[SWS\_Can\_00409] ⌈ Khi hàm Can\_SetControllerMode (CAN\_CS\_STARTED) được gọi và bộ điều khiển CAN không ở trạng thái STOPPED, nó phải phát hiện ra một chuyển đổi trạng thái không hợp lệ (So sánh với SWS\_Can\_00200).⌋ ()

Chuyển đổi trạng thái do hàm Can\_SetControllerMode (CAN\_CS\_STOPPED)

- STARTED -> STOPPED

- SLEEP -> STOPPED

- được kích hoạt bởi phần mềm

[SWS\_Can\_00263] ⌈ Hàm Can\_SetControllerMode(CAN\_CS\_STOPPED) phải thiết lập các bit bên trong phần cứng CAN sao cho bộ điều khiển CAN ngừng tham gia vào mạng.⌋ ()

[SWS\_Can\_00264] ⌈ Hàm Can\_SetControllerMode(CAN\_CS\_STOPPED) phải chờ một khoảng thời gian giới hạn cho đến khi bộ điều khiển CAN thực sự được tắt. So sánh với SWS\_Can\_00398.⌋ ()

[SWS\_Can\_00267] ⌈ Nếu phần cứng CAN không hỗ trợ chế độ ngủ, việc chuyển từ SLEEP sang STOPPED phải trở lại từ trạng thái ngủ logic, nhưng không có hiệu lực đối với trạng thái của bộ điều khiển CAN (vì bộ điều khiển đã ở trạng thái dừng).⌋ ()

[SWS\_Can\_00268] ⌈ Hàm Can\_SetControllerMode(CAN\_CS\_STOPPED) phải chờ một khoảng thời gian giới hạn cho đến khi bộ điều khiển CAN ở trạng thái STOPPED. So sánh với SWS\_Can\_00398.⌋ ()

[SWS\_Can\_00282] ⌈ Hàm Can\_SetControllerMode(CAN\_CS\_STOPPED) phải hủy bỏ các tin nhắn đang chờ. ⌋ ()

Chuyển đổi trạng thái do hàm Can\_SetControllerMode (CAN\_CS\_SLEEP)

- STOPPED -> SLEEP

- được kích hoạt bởi phần mềm

[SWS\_Can\_00265] ⌈ Hàm Can\_SetControllerMode(CAN\_CS\_SLEEP) phải đặt bộ điều khiển vào chế độ ngủ.⌋ ()

[SWS\_Can\_00266] ⌈ Nếu phần cứng CAN hỗ trợ chế độ ngủ, hàm Can\_SetControllerMode(CAN\_CS\_SLEEP) phải chờ một khoảng thời gian giới hạn cho đến khi bộ điều khiển CAN ở trạng thái NGỦ và đảm bảo rằng phần cứng CAN có thể đánh thức được. So sánh với SWS\_Can\_00398.⌋ ()

[SWS\_Can\_00290] ⌈ Nếu phần cứng CAN không hỗ trợ chế độ ngủ, hàm Can\_SetControllerMode(CAN\_CS\_SLEEP) phải thiết lập bộ điều khiển CAN vào chế độ ngủ logic.⌋ ()

[SWS\_Can\_00405] ⌈ Chế độ ngủ logic này chỉ được rời khỏi khi hàm Can\_SetControllerMode(CAN\_CS\_STOPPED) được gọi.⌋ ()

[SWS\_Can\_00411] ⌈ Khi hàm Can\_SetControllerMode(CAN\_CS\_SLEEP) được gọi và bộ điều khiển CAN không ở trạng thái STOPPED hoặc SLEEP, nó phải phát hiện ra một chuyển đổi trạng thái không hợp lệ (So sánh với SWS\_Can\_00200).⌋ ()

##### State transition caused by Hardware Events

Chuyển đổi trạng thái do Sự kiện Phần cứng Đánh thức (kích hoạt bởi sự kiện đánh thức từ BUS CAN)

- SLEEP -> STOPPED

- kích hoạt bởi các L-PDUs đến

- Mô-đun Quản lý Trạng thái ECU được thông báo bằng hàm EcuM\_CheckWakeup

Chuyển đổi trạng thái này chỉ xảy ra khi chế độ ngủ được hỗ trợ bởi phần cứng.

[SWS\_Can\_00270] ⌈ Khi có sự đánh thức phần cứng (được kích hoạt bởi một sự kiện đánh thức từ BUS CAN), bộ điều khiển CAN phải chuyển sang trạng thái DỪNG LẠI.⌋ ()

[SWS\_Can\_00271] ⌈ Khi có sự đánh thức phần cứng (được kích hoạt bởi một sự kiện đánh thức từ BUS CAN), Mô-đun Can phải gọi hàm EcuM\_CheckWakeup hoặc trong ngữ cảnh Can\_MainFunction\_Wakeup.⌋ (SRS\_BSW\_00375, SRS\_SPAL\_12069, SRS\_Can\_01054)

[SWS\_Can\_00269] ⌈ Mô-đun Can không được tiếp tục xử lý L-PDU gây ra sự đánh thức.⌋ ()

[SWS\_Can\_00048] ⌈ Trong trường hợp có sự đánh thức BUS CAN trong quá trình chuyển tiếp ngủ, hàm Can\_SetControllerMode(CAN\_CS\_STOPPED) phải trả về E\_NOT\_OK.⌋ (SRS\_Can\_01122)

Chuyển trạng thái do Bus-Off (kích hoạt bởi sự thay đổi trạng thái của bộ điều khiển CAN)

[SWS\_Can\_00020] ⌈

STARTED STOPPED

được kích hoạt bởi phần cứng nếu bộ điều khiển CAN đạt trạng thái bus-off

Mô-đun CanIf được thông báo bằng hàm CanIf\_ControllerBusOff sau khi trạng thái STOPPED được đạt đến liên quan đến bộ điều khiển CAN tương ứng với CanIf ControllerId trừu tượng.⌋ (SRS\_Can\_01055)

[SWS\_Can\_00272] ⌈ Sau khi phát hiện bus-off, bộ điều khiển CAN phải chuyển sang trạng thái DỪNG LẠI và mô-đun Can phải đảm bảo rằng bộ điều khiển CAN không còn tham gia vào mạng nữa. ⌋ (SRS\_Can\_01060)

[SWS\_Can\_00273] ⌈ Sau khi phát hiện bus-off, mô-đun Can phải hủy các tin nhắn đang chờ xử lý. ⌋ (SRS\_Can\_01060)

[SWS\_Can\_00274] ⌈ Mô-đun Can phải tắt hoặc ngăn chặn việc phục hồi bus-off tự động.⌋ (SRS\_Can\_01060)

##### Chuyển trạng thái do hàm Can\_DeInit

- STOPPED -> UNINIT; SLEEP -> UNINIT (cho tất cả các bộ điều khiển trong HW unit)

- được kích hoạt bởi phần mềm bằng cách gọi hàm Can\_DeInit

- chuẩn bị tất cả các bộ điều khiển CAN trong HW Unit để được cấu hình lại

[SWS\_Can\_91010] ⌈ Hàm Can\_DeInit phải đặt tất cả các bộ điều khiển CAN vào trạng thái UNINIT⌋ (SRS\_Can\_01166, SRS\_Can\_01053)

Khi hàm Can\_DeInit được gọi và mô-đun Can không ở trạng thái CAN\_READY hoặc bất kỳ bộ điều khiển CAN nào ở trạng thái STARTED, nó phải phát ra lỗi CAN\_E\_TRANSITION (Tham khảo [SWS\_Can\_91011] và [SWS\_Can\_91012]).

#### Khởi tạo mô-đun/Controller Can

Mô-đun Quản lý Trạng thái ECU phải khởi tạo mô-đun Can trong giai đoạn khởi động bằng cách gọi hàm Can\_Init trước khi sử dụng bất kỳ hàm nào khác của mô-đun Can.

[SWS\_Can\_00250] ⌈ Hàm Can\_Init phải khởi tạo:

biến tĩnh, bao gồm cờ,

Cài đặt chung cho toàn bộ đơn vị phần cứng CAN

Cài đặt cụ thể của bộ điều khiển CAN cho mỗi bộ điều khiển CAN.⌋ (SRS\_BSW\_00101)

[SWS\_Can\_00053] ⌈ Can\_Init không được thay đổi các thanh ghi của các tài nguyên phần cứng bộ điều khiển CAN mà không được sử dụng. ⌋ (SRS\_SPAL\_12125)

Mô-đun Can phải áp dụng các quy tắc sau đây về việc khởi tạo các thanh ghi của bộ điều khiển:

[SWS\_Can\_00407] ⌈ Nếu phần cứng chỉ cho phép sử dụng thanh ghi một lần, mô-đun Can thực hiện chức năng đó chịu trách nhiệm khởi tạo thanh ghi.

Nếu thanh ghi có thể ảnh hưởng đến nhiều mô-đun phần cứng và nếu nó là một thanh ghi I/O thì nó phải được khởi tạo bởi trình điều khiển PORT.

Nếu thanh ghi có thể ảnh hưởng đến nhiều mô-đun phần cứng và nếu nó không phải là một thanh ghi I/O thì nó phải được khởi tạo bởi trình điều khiển MCU.

Các thanh ghi chỉ có thể ghi một lần và cần phải được khởi tạo trực tiếp sau khi thiết lập lại phải được khởi tạo bởi mã khởi đầu.

Tất cả các thanh ghi khác phải được khởi tạo bởi mã khởi đầu.⌋

(SRS\_SPAL\_12461)

[SWS\_Can\_00056] ⌈ Các yếu tố cấu hình sau khi xây dựng được đánh dấu là 'nhiều' ('M' hoặc 'x') trong chương 10 có thể được chọn bằng cách chuyển con trỏ 'Config' vào hàm khởi tạo của mô-đun. ⌋ ()

[SWS\_Can\_00062] ⌈ Nếu Can\_SetBaudrate xác định rằng việc thay đổi cấu hình nhắm mục tiêu đòi hỏi một việc khởi tạo lại và bộ điều khiển CAN ở trạng thái STOPPED, hàm Can\_SetBaudrate phải khởi tạo lại bộ điều khiển CAN và các cài đặt cụ thể của bộ điều khiển.⌋ (SRS\_Can\_01139, SRS\_Can\_01042)

Nếu việc khởi tạo lại cần thiết, Bộ điều khiển CAN phải được chuyển sang trạng thái DỪNG LẠI trước khi Can\_SetBaudrate() có thể được thực hiện và cấu hình baud rate mới có thể được áp dụng.

[SWS\_Can\_00255] ⌈ Hàm Can\_SetBaudrate chỉ ảnh hưởng đến các khu vực thanh ghi chứa cấu hình cụ thể cho một bộ điều khiển CAN duy nhất. ⌋ ()

[SWS\_Can\_00021] ⌈ Cấu hình bộ điều khiển được mong muốn có thể được chọn với tham số Config. ⌋ (SRS\_BSW\_00344, SRS\_BSW\_00404, SRS\_BSW\_00405, SRS\_SPAL\_12263, SRS\_SPAL\_12265)

[SWS\_Can\_00291] ⌈ Config là một con trỏ vào mảng của cấu trúc dữ liệu cụ thể của việc triển khai được lưu trữ trong ROM. Các bộ cấu hình bộ điều khiển khác nhau được đặt dưới dạng các cấu trúc dữ liệu trong ROM.⌋ (SRS\_BSW\_00438)

Các giá trị có thể có cho Cấu hình được cung cấp bởi mô tả cấu hình (xem chương 10).

Cấu hình mô-đun Can xác định cài đặt Đơn vị CAN HW toàn cầu và

tham chiếu đến các bộ cấu hình bộ điều khiển CAN mặc định.

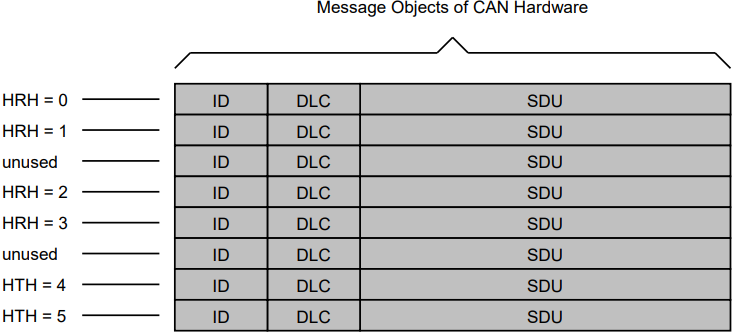
#### Truyền L-PDU

Trong quá trình truyền L-PDU, mô-đun Can chuyển đổi nội dung ID và Độ dài Dữ liệu của L-PDU sang một định dạng cụ thể của phần cứng (nếu cần) và kích hoạt việc truyền.

[SWS\_Can\_00059] ⌈ Bản đồ dữ liệu bởi CAN sang bộ nhớ được xác định theo cách mà byte dữ liệu CAN được gửi ra đầu tiên là phần tử mảng 0, byte dữ liệu CAN được gửi ra cuối cùng là phần tử mảng 7 hoặc 63 trong trường hợp của CAN FD.⌋ (SRS\_SPAL\_12063)

[SWS\_Can\_00427] ⌈ Nếu bản trình bày bên trong bộ đệm phần cứng CAN khác với định nghĩa AUTOSAR, mô-đun Can phải cung cấp một Bộ đệm SDU được điều chỉnh cho các lớp trên cùng.⌋ ()

[SWS\_Can\_00100] ⌈ Một số đối tượng phần cứng TX với các HTH duy nhất có thể được cấu hình. Mô-đun CanIf cung cấp HTH như tham số của yêu cầu TX. Xem Hình 7-3 cho một cấu hình có thể.⌋ (SRS\_Can\_01135)



Hình 7-3: Ví dụ về việc gán HTHs và HRHs cho các Đối tượng Phần cứng. Việc đánh số HTHs và HRHs là cụ thể cho từng triển khai. Việc đánh số được chọn chỉ là một ví dụ.

[SWS\_Can\_00276] ⌈ Hàm Can\_Write sẽ lưu trữ swPduHandle được đưa vào trong tham số PduInfo cho đến khi mô-đun Can gọi hàm CanIf\_TxConfirmation cho yêu cầu này, trong đó swPduHandle được đưa ra như là tham số.⌋ ()

Tính năng của SWS\_Can\_00276 được sử dụng để giảm thời gian tìm kiếm trong triển khai của mô-đun CanIf.

[SWS\_Can\_00016] ⌈ Mô-đun Can sẽ gọi CanIf\_TxConfirmation để chỉ ra việc truyền thành công. Nó có thể được gọi bởi dịch vụ ngắt TX của nguồn tài nguyên phần cứng tương ứng hoặc bên trong hàm Can\_MainFunction\_Write trong trường hợp chế độ đánh giá.⌋

##### Ưu tiên chéo (Priority Inversion)

Việc truyền đa kênh là cần thiết để ngăn chặn ưu tiên chéo (xem chương 2.1).

[SWS\_Can\_00277] ⌈ Mô-đun Can sẽ cho phép chức năng "Truyền đa kênh" có thể cấu hình tĩnh (BẬT | TẮT) tại thời điểm biên dịch trước.⌋ (SRS\_Can\_01134)

[SWS\_Can\_00401] ⌈ Nhiều đối tượng phần cứng truyền (được xác định bởi "CanHwObjectCount") sẽ được gán bởi một HTH để đại diện cho một thực thể truyền đến lớp trên cùng.⌋ (SRS\_Can\_01134)

[SWS\_Can\_00402] ⌈ Mô-đun Can sẽ hỗ trợ các cơ chế truyền đa kênh cho các thiết bị nơi mà:

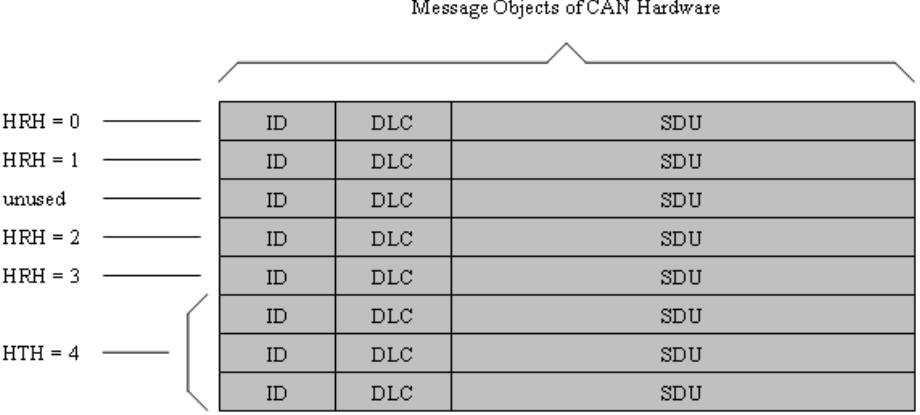
Nhiều đối tượng phần cứng truyền, được nhóm lại thành một thực thể truyền có thể được điền vào qua cùng một bộ đăng ký, và vi điều khiển lưu trữ L-PDU vào một bộ đệm trống tự động,

hoặc

Phần cứng cung cấp bộ đăng ký hoặc chức năng để xác định một đối tượng phần cứng truyền trống trong một thực thể truyền.⌋ (SRS\_Can\_01134)

[SWS\_Can\_00403] ⌈ Mô-đun Can sẽ hỗ trợ truyền đa kênh cho các thiết bị, mà gửi các L-PDU theo thứ tự ưu tiên của L-PDU.⌋ (SRS\_Can\_01134)

Lưu ý: Tránh mô phỏng phần mềm của việc xử lý ưu tiên, vì chi phí cơ hội sẽ hủy đi lợi ích của việc truyền đa kênh.



Hình 7-4: Ví dụ về việc gán HTHs và HRHs cho các Đối tượng Phần cứng với truyền thông đa luồng. Các số thứ tự của HTHs và HRHs là cụ thể cho việc triển khai. Các số thứ tự được chọn chỉ là một ví dụ.

##### Tính nhất quán của dữ liệu truyền

[SWS\_Can\_00011] ⌈ Module Can phải sao chép dữ liệu trực tiếp từ bộ đệm của tầng trên. Trách nhiệm của tầng trên là duy trì sự nhất quán của bộ đệm cho đến khi cuộc gọi hàm trả về (Can\_Write).⌋ (SRS\_SPAL\_12075, SRS\_Can\_01059)

#### Nhận L-PDU

[SWS\_Can\_00279] ⌈ Khi nhận được L-PDU, module Can sẽ gọi hàm gọi lại RX indication CanIf\_RxIndication với ID, Hoh, CanIf ControllerId trừu tượng trong tham số Mailbox, và chiều dài dữ liệu và con trỏ đến bộ đệm L-SDU trong tham số PduInfoPtr.⌋ (SRS\_Can\_01045)

[SWS\_Can\_00423] ⌈ Trong trường hợp của một khung CAN mở rộng, module Can sẽ chuyển đổi ID thành một định dạng chuẩn vì Tầng trên (CANIF) không biết rằng khung CAN nhận được là một khung CAN tiêu chuẩn hay mở rộng. Trong trường hợp của một khung CAN mở rộng, MSB của một khung CAN ID nhận được cần được đặt là '1' để đánh dấu khung CAN nhận được là Mở rộng.⌋ ()

[SWS\_Can\_00396] ⌈ RXTemporary-interrupt service routine của tài nguyên phần cứng tương ứng hoặc hàm Can\_MainFunction\_Read trong trường hợp chế độ đánh giá bằng cách bỏ phiếu sẽ gọi hàm gọi lại CanIf\_RxIndication.⌋ (SRS\_Can\_01045)

[SWS\_Can\_00060] ⌈ Ánh xạ dữ liệu bằng CAN vào bộ nhớ được xác định theo cách rằng byte dữ liệu CAN nhận được đầu tiên là phần tử mảng 0, byte dữ liệu CAN nhận được cuối cùng là phần tử mảng 7 hoặc 63 trong trường hợp CAN FD. Nếu bảng trình bày bên trong bộ đệm phần cứng CAN khác với định nghĩa AUTOSAR, module Can phải cung cấp một bộ đệm SDU được điều chỉnh cho các tầng trên.⌋ (SRS\_SPAL\_12063)

[SWS\_Can\_00501] ⌈ CanDrv sẽ chỉ ra liệu thông điệp nhận được là một khung CAN truyền thống hay một khung CAN FD như mô tả trong Can\_IdType.⌋ (SRS\_Can\_01162)

##### Tính nhất quán của dữ liệu nhận được

Để ngăn mất mát thông điệp nhận được, một số bộ điều khiển hỗ trợ một FIFO được xây dựng từ một tập hợp các đối tượng phần cứng, trong khi trên các bộ điều khiển khác, có thể cấu hình một đối tượng phần cứng khác với các thuộc tính tương tự mà hoạt động như một bộ đệm bóng bóng và bước vào khi đối tượng chính bận.

[SWS\_Can\_00489] ⌈ Trình điều khiển CAN phải hỗ trợ các bộ điều khiển mà triển khai một FIFO phần cứng. Kích thước của FIFO được cấu hình thông qua "CanHwObjectCount".⌋ ()

[SWS\_Can\_00490] ⌈ Các bộ điều khiển không hỗ trợ FIFO phần cứng thường cung cấp khả năng triển khai cơ chế bộ nhớ đệm bóng bóng, nơi các đối tượng phần cứng bổ sung tiếp quản khi đối tượng phần cứng chính bận rộn. Số lượng đối tượng phần cứng được cấu hình thông qua "CanHwObjectCount".⌋ ()

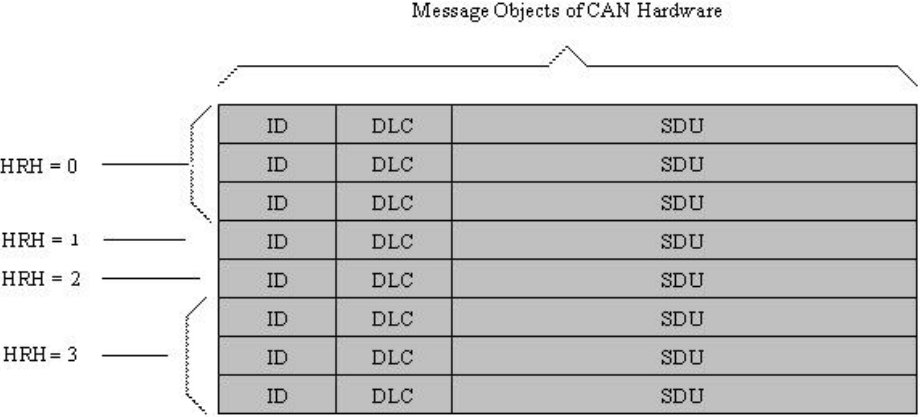


Figure 7-5: Ví dụ về việc gán các HRH giống nhau cho nhiều Đối tượng Phần cứng. Cách đánh số được chọn chỉ là một ví dụ.

[SWS\_Can\_00299] ⌈ Module Can sẽ sao chép L-SDU vào bộ đệm bóng sau khi nhận, nếu bộ đệm RX không thể được bảo vệ (khóa) bởi Phần cứng CAN chống ghi đè bởi một thông điệp mới được nhận.⌋ ()

[SWS\_Can\_00300] ⌈ Module Can sẽ sao chép L-SDU vào bộ đệm bóng, nếu Phần cứng CAN không có sẵn toàn cầu.⌋ ()

Quá trình RX hoàn chỉnh (bao gồm sao chép vào tầng đích, ví dụ: COM) được thực hiện trong ngữ cảnh của ngắt RX hoặc trong ngữ cảnh của Can\_MainFunction\_Read.

[SWS\_Can\_00012] ⌈ Module Can sẽ đảm bảo rằng cả các ISR lẫn hàm Can\_MainFunction\_Read không thể bị gián đoạn bởi chính nó. Bộ đệm Phần cứng CAN (hoặc bóng) luôn nhất quán, vì nó được ghi và đọc theo chuỗi trong một hàm duy nhất không bị gián đoạn bởi chính nó.⌋ (SRS\_Can\_01059)

Nếu Phần cứng CAN không thể được cấu hình để khóa đối tượng phần cứng RX sau khi nhận (tính năng phần cứng), có thể xảy ra tình trạng bộ đệm phần cứng bị ghi đè bởi một thông điệp mới đến. Trong trường hợp này, bộ điều khiển CAN phát hiện một sự kiện "ghi đè", nếu được hỗ trợ bởi phần cứng.

Nếu Phần cứng CAN có thể được cấu hình để khóa đối tượng phần cứng RX sau khi nhận, có thể xảy ra tình trạng thông điệp mới đến không thể được lưu vào bộ đệm phần cứng. Trong trường hợp này, bộ điều khiển CAN phát hiện một sự kiện "tràn", nếu được hỗ trợ bởi phần cứng.

[SWS\_Can\_00395] ⌈ Module Can sẽ tạo ra lỗi thời gian chạy CAN\_E\_DATALOST trong trường hợp phát hiện sự kiện "ghi đè" hoặc "tràn".⌋ ()

Gợi ý Triển khai:

Nhà thiết kế hệ thống phải đảm bảo rằng thời gian chạy cho việc nhận thông điệp (điều khiển ngắt hoặc đánh giá bằng cách bỏ phiếu) tương ứng với việc nhận nhanh nhất có thể trong hệ thống.

#### Khái niệm Wakeup

Module Can xử lý các sự thức dậy có thể được phát hiện bởi bản thân bộ điều khiển Can và không thông qua bộ chuyển mạch Can. Có hai kịch bản có thể xảy ra: thức dậy bằng ngắt và thức dậy bằng cách đánh giá bằng cách bỏ phiếu.

Đối với thức dậy bằng ngắt, một ISR của Module Can được gọi khi phần cứng phát hiện thức dậy.

[SWS\_Can\_00364] ⌈ Nếu ISR cho các sự kiện thức dậy được gọi, nó sẽ gọi EcuM\_CheckWakeup lần lượt. Tham số được truyền cho EcuM\_CheckWakeup sẽ là ID của nguồn thức dậy được tham chiếu bởi tham số cấu hình CanWakeupSourceRef.⌋ (SRS\_BSW\_00375, SRS\_SPAL\_12069, SRS\_Can\_01054)

Bộ điều khiển Trạng thái ECU sau đó sẽ thiết lập MCU và gọi lại Module Can thông qua Giao diện Can, dẫn đến việc gọi Can\_CheckWakeup.

Khi các sự kiện thức dậy được phát hiện bằng cách đánh giá bằng cách bỏ phiếu, Bộ điều khiển Trạng thái ECU sẽ gọi Can\_CheckWakeup theo chu kỳ qua Giao diện Can như trước đây. Trong cả hai trường hợp, Can\_CheckWakeup sẽ kiểm tra xem có sự thức dậy được phát hiện bởi một bộ điều khiển Can và trả về kết quả. Bộ điều khiển CAN sau đó sẽ thông báo về sự kiện thức dậy cho Bộ điều khiển Trạng thái ECU qua EcuM\_SetWakeupEvent.

Xác thực thức dậy để ngăn chặn các sự kiện thức dậy sai lầm sẽ được thực hiện bởi Bộ điều khiển Trạng thái ECU và Giao diện Can sau đó và không có sự giúp đỡ nào từ Module Can.

Để biết mô tả tổng quan về các cơ chế thức dậy và biểu đồ chuỗi thức dậy, xin tham khảo Chi tiết về Quản lý Trạng thái ECU [7].

#### Khái niệm Thông báo

Module Can chỉ cung cấp một giao diện thông báo được kích hoạt bởi sự kiện đến cho module CanIf. Mỗi thông báo được đại diện bởi một hàm gọi lại.

[SWS\_Can\_00099] ⌈ Các sự kiện phần cứng có thể được phát hiện bằng một ngắt hoặc bằng cách bỏ phiếu các cờ trạng thái của các đối tượng phần cứng. Các khả năng cấu hình liên quan đến việc bỏ phiếu là phụ thuộc vào phần cứng (tức là các sự kiện nào có thể được bỏ phiếu, các sự kiện nào cần được bỏ phiếu), và không bị hạn chế bởi tiêu chuẩn này.⌋ (SRS\_Can\_01132)

[SWS\_Can\_00007] ⌈ Phải có khả năng cấu hình trình điều khiển sao cho không có ngắt nào được sử dụng (toàn bộ bỏ phiếu). ⌋ (SRS\_Can\_01062)

Cấu hình của những gì được bỏ phiếu và không được bỏ phiếu bởi Module Can là nội bộ của trình điều khiển và không hiển thị bên ngoài module. Việc bỏ phiếu được thực hiện bên trong các chức năng chính CAN (Can\_MainFunction\_xxx). Các sự kiện được bỏ phiếu cũng được thông báo qua hàm gọi lại phù hợp. Sau đó, ngữ cảnh gọi không phải là ISR mà là hàm chính của CAN. Việc triển khai tất cả các hàm gọi lại phải được thực hiện như ngữ cảnh gọi là ISR.

Để biết thêm chi tiết, xem mô tả các chức năng chính CAN như Can\_MainFunction\_Read, Can\_MainFunction\_Write, Can\_MainFunction\_BusOff và Can\_MainFunction\_Wakeup.

#### Vấn đề Reentrancy

Một hàm phải thỏa mãn các điều kiện sau để có thể tái nhập:

- Nó sử dụng tất cả các biến chia sẻ theo cách nguyên tử, trừ khi mỗi biến được phân bổ cho một phiên bản cụ thể của hàm.

- Nó không gọi các hàm không tái nhập.

- Nó không sử dụng phần cứng theo cách không nguyên tử.

Các yêu cầu truyền gửi đơn giản được chuyển tiếp bởi mô-đun CanIf bên trong hàm CanIf\_Transmit.

Hàm CanIf\_Transmit là tái nhập. Do đó, hàm Can\_Write cần được triển khai an toàn cho luồng (ví dụ bằng cách sử dụng mutex):

Các cuộc gọi tiếp theo (phân biệt) sẽ trả về với CAN\_BUSY khi việc ghi không thể thực hiện tái nhập. (ví dụ: ghi vào các Xử lý TX phần cứng khác nhau được phép, ghi vào cùng Xử lý TX không được phép)

Trong trường hợp CAN\_BUSY, mô-đun CanIf sẽ xếp hàng yêu cầu đó. (hành vi giống như nếu tất cả các đối tượng phần cứng đều bận)

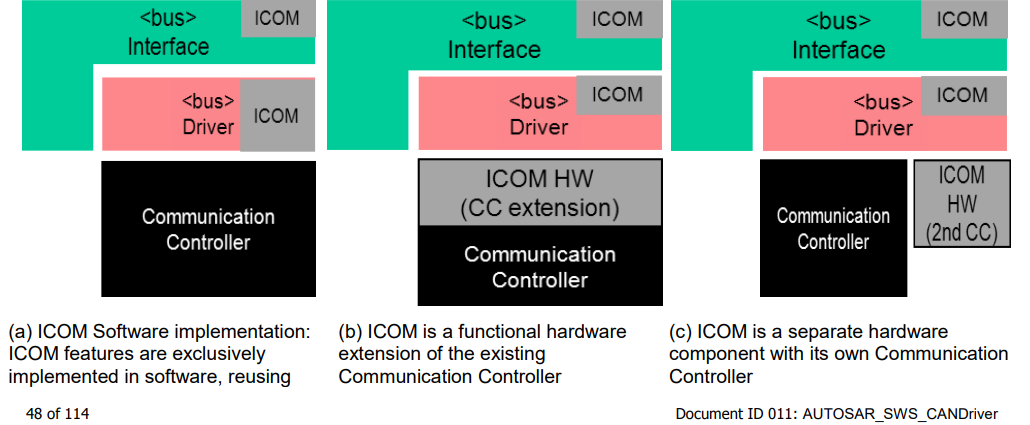
Các hàm Can\_EnableCanInterrupts và Can\_DisableCanInterrupts có thể được gọi trong các hàm tái nhập. Do đó, các hàm này cũng cần được triển khai tái nhập.

Tất cả các dịch vụ khác không cần được triển khai như các hàm tái nhập.

Các hàm chính CAN (ví dụ: Can\_MainFunction\_Read) không được phép bị gián đoạn bởi chính chúng. Do đó, các hàm chính CAN này không phải là hàm tái nhập.

#### Mạng ảo

Tối ưu hóa hiệu suất năng lượng đang trở nên ngày càng quan trọng trong tất cả các lĩnh vực ô tô vì tiêu thụ năng lượng có tác động trực tiếp đến tiêu thụ nhiên liệu, lượng khí CO2 phát thải và phạm vi của các xe hybrid hoặc toàn bộ điện. Khái niệm Mạng Ảo có tiềm năng cao để giảm năng lượng ở cấp độ ECU.



Dưới đây là bản dịch từ tiếng Anh sang tiếng Việt của đoạn văn trên, đảm bảo ngữ pháp, đầy đủ nội dung và logic:

Bộ điều khiển truyền thông hiện có

Hình 7-6: Các phương án thực hiện ICOM khả thi

Việc sử dụng các Bộ điều khiển Truyền thông Thông minh (ICOM) hỗ trợ các tính năng đó (không bắt buộc phải có phần cứng cụ thể). Nếu một số hoặc tất cả chức năng của một ECU tạm thời không cần thiết, ví dụ dựa trên trạng thái của xe, ECU có thể chuyển sang chế độ "Mạng giả". Trong chế độ này, MCU và/hoặc các ngoại vi sẽ chuyển sang chế độ tiêu thụ năng lượng thấp. Chỉ có ICOM và các bộ thu/phát kết nối với nó vẫn hoạt động. ICOM sẽ tạo ra một sự kiện đánh thức, gây ra bởi ví dụ như một tin nhắn bus được nhận, khi ECU cần khởi động lại hoạt động. Tùy thuộc vào cách thực hiện ICOM, ID tin nhắn và nội dung tin nhắn nhận được có thể được đánh giá và lọc hoàn toàn trong phần cứng, hoặc yêu cầu một cơ chế gọi lại trong phần mềm.

Như được thể hiện trong Hình 7-6, việc thực hiện ECU có thể được chia thành ba phương án khả thi - Phương án (a) là phương án phần mềm, không có phần cứng cụ thể để hỗ trợ Mạng giả. Phương án (b) mô tả một phần mở rộng phần cứng chức năng của bộ điều khiển truyền thông và phương án (c) cho thấy một biến thể phần cứng với bộ điều khiển truyền thông thứ hai mở rộng để xử lý đánh thức. Tất cả các biến thể và cách thực hiện ICOM đều phải được hỗ trợ bởi Mạng giả. Tùy thuộc vào cách thực hiện phần cứng, ICOM cũng có khả năng gửi tin nhắn. Bằng cách sử dụng ICOM để tiếp tục gửi, ví dụ như các tin nhắn trạng thái, các nút khác phụ thuộc vào tin nhắn đó sẽ không bị ảnh hưởng bởi một ECU ở chế độ Mạng giả. Hơn nữa, Mạng giả nhằm mục đích giảm thời gian phản hồi đánh thức, tức là khoảng thời gian giữa một sự kiện đánh thức và hành vi hợp lệ của một ECU. Bằng cách sử dụng ICOM để lưu các tin nhắn liên quan trong khi chế độ Mạng giả được kích hoạt, ứng dụng sẽ có quyền truy cập vào các giá trị tín hiệu hợp lệ cuối cùng ngay sau khi khởi động lại hoạt động. Do đó, ECU có thể phản hồi ngay lập tức với yêu cầu của người dùng sau khi đánh thức và không phải chờ đợi cho đến khi tin nhắn tương ứng được nhận lại.

Đây là một đoạn mô tả các yêu cầu phần mềm liên quan đến việc xử lý chế độ "Pretended Networking" trong driver CAN của AUTOSAR. Tôi sẽ dịch đoạn này sang tiếng Việt:

##### Hỗ trợ xử lý chế độ Pretended Networking [SWS\_Can\_00497]

⌈Trình điều khiển CAN phải vô hiệu hóa chế độ Pretended Networking sau khi khởi tạo bộ điều khiển CAN.⌋

() Kích hoạt chế độ Pretended Networking:

[SWS\_Can\_00462] ⌈Chế độ Pretended Networking sẽ được kích hoạt bằng cách gọi Can\_SetIcomConfiguration() với một ID cấu hình không được đặt là 0.⌋

[SWS\_Can\_00464] ⌈CanDrv chịu trách nhiệm thực hiện cấu hình lại bộ điều khiển CAN (bao gồm ICOM) theo các tham số CanIcomConfig cho cấu hình được chọn (CanIcomConfigId).⌋

[SWS\_Can\_00467] ⌈Nếu kích hoạt thành công, CanIf\_CurrentIcomConfiguration sẽ được gọi với tham số Error được đặt thành CAN\_OK.⌋

Trong đoạn này, các yêu cầu phần mềm được liệt kê như sau:

1. Sau khi khởi tạo bộ điều khiển CAN, trình điều khiển CAN phải vô hiệu hóa chế độ Pretended Networking.

2. Để kích hoạt chế độ Pretended Networking, phải gọi hàm Can\_SetIcomConfiguration() với một ID cấu hình không phải 0.

3. CanDrv có trách nhiệm cấu hình lại bộ điều khiển CAN (bao gồm ICOM) theo các tham số cấu hình CanIcomConfig cho cấu hình được chọn (CanIcomConfigId).

4. Nếu kích hoạt chế độ Pretended Networking thành công, hàm CanIf\_CurrentIcomConfiguration sẽ được gọi với tham số Error đặt thành CAN\_OK.

Các yêu cầu này xác định cách thức kích hoạt và xử lý chế độ Pretended Networking trong trình điều khiển CAN của AUTOSAR.

Đây là phần tiếp theo của các yêu cầu phần mềm liên quan đến xử lý chế độ Pretended Networking trong AUTOSAR. Tôi sẽ dịch phần này sang tiếng Việt:

Nếu kích hoạt thành công thì CanIf\_CurrentIcomConfiguration sẽ được gọi với tham số Error đặt thành ICOM\_SWITCH\_E\_OK tham chiếu đến bộ điều khiển CAN tương ứng với CanIf ControllerId trừu tượng. Nếu kích hoạt không thành công, CanIf\_CurrentIcomConfiguration sẽ được gọi với tham số Error đặt thành ICOM\_SWITCH\_E\_FAILED tham chiếu đến bộ điều khiển CAN tương ứng với CanIf ControllerId trừu tượng.⌋

[SWS\_Can\_00468] ⌈Nếu Pretended Networking được kích hoạt, CanDrv sẽ gọi CanIf\_RxIndication() nếu và chỉ nếu tin nhắn nhận được khớp với các điều kiện đánh thức của CanIcomConfig (xem CanIcomWakeupCauses).⌋

[SWS\_Can\_00470] ⌈Nếu Pretended Networking được kích hoạt, CanDrv sẽ từ chối yêu cầu Can\_Write() với giá trị trả về CAN\_BUSY.⌋

[SWS\_Can\_00498] ⌈Trình điều khiển CAN sẽ vô hiệu hóa chế độ Pretended Networking trước khi bộ điều khiển CAN được khởi động bằng SetControllerMode(CAN\_CS\_STARTED)⌋

Vô hiệu hóa chế độ Pretended Networking:

[SWS\_Can\_00471] ⌈Chế độ Pretended Networking sẽ được vô hiệu hóa (tức là CanDrv sẽ hoạt động giống như khi nó được cấu hình mà không hỗ trợ Pretended Networking) bằng cách gọi Can\_SetIcomConfiguration() với ID cấu hình = 0.⌋

[SWS\_Can\_00472] ⌈Nếu Pretended Networking bị vô hiệu hóa, CanDrv sẽ xử lý các tin nhắn bình thường như đã cấu hình trong cấu hình thông thường.⌋

[SWS\_Can\_00474] ⌈Trình điều khiển CAN sẽ thông báo cho CanIf về một sự chuyển đổi cấu hình bằng cách gọi CanIf\_CurrentIcomConfiguration tham chiếu đến bộ điều khiển CAN tương ứng với CanIf ControllerId trừu tượng. Tham số lỗi được đặt thành ICOM\_SWITCH\_E\_OK nếu vô hiệu hóa thành công và ICOM\_SWITCH\_E\_FAILED nếu không.⌋

[SWS\_Can\_00499] ⌈Trình điều khiển CAN sẽ vô hiệu hóa chế độ Pretended Networking trước khi bộ điều khiển CAN bị dừng bằng SetControllerMode(CAN\_CS\_STOPPED).⌋

##### Hỗ trợ gửi và nhận tin nhắn tự động

[SWS\_Can\_00477] ⌈Việc gửi tin nhắn tự động trong chế độ Pretended Networking chỉ được hỗ trợ nếu có phần cứng ICOM bổ sung. Một tham số cấu hình xác định liệu có hỗ trợ phần cứng hay không (Tham khảo CanIcomVariant).⌋

[SWS\_Can\_00478] ⌈Nếu ICOM được thực hiện bằng phần mềm, bộ điều khiển sẽ không gửi tin nhắn trong chế độ Pretended Networking.⌋

[SWS\_Can\_00479] ⌈CanDriver sẽ chuyển tiếp tất cả các tin nhắn nhận được trong Chế độ Pretended Networking đến CanIf.⌋

Phần này mô tả các yêu cầu về cách xử lý kích hoạt/vô hiệu hóa chế độ Pretended Networking, xử lý tin nhắn nhận/gửi trong chế độ này, và các điều kiện hỗ trợ gửi/nhận tin nhắn tự động khi ở chế độ Pretended Networking.

Đây là phần mô tả về phân loại lỗi trong mô-đun CAN của AUTOSAR. Tôi sẽ dịch nó sang tiếng Việt:

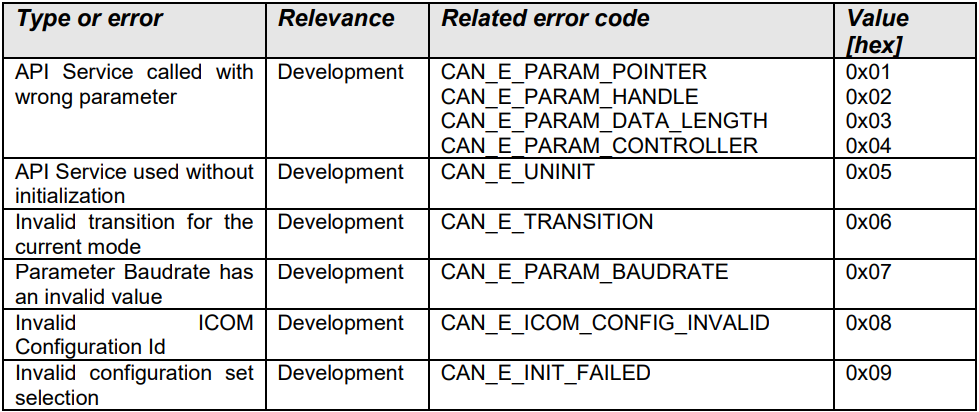
#### Phân loại lỗi

[SWS\_Can\_00104] ⌈Mô-đun CAN phải có khả năng phát hiện các lỗi và ngoại lệ sau đây tùy thuộc vào cấu hình của nó (mặc định/sản xuất)⌋ (SRS\_BSW\_00337, SRS\_BSW\_00385, SRS\_BSW\_00331)

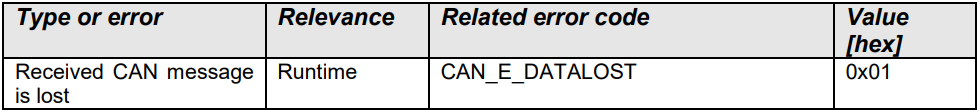
##### Lỗi trong quá trình phát triển

Phần này nêu ra rằng mô-đun CAN phải có khả năng phát hiện các lỗi và ngoại lệ khác nhau trong quá trình vận hành, tùy thuộc vào cấu hình của nó (cấu hình mặc định hay cấu hình sản xuất). Các yêu cầu phần mềm liên quan đến phân loại lỗi sẽ được liệt kê trong các phần tiếp theo.

Phần 7.11.1 sẽ đề cập đến các lỗi phát sinh trong quá trình phát triển phần mềm CAN, như lỗi cấu hình, lỗi tham số đầu vào không hợp lệ, v.v. Đây là dạng lỗi cần được phát hiện và xử lý trong giai đoạn phát triển để đảm bảo chất lượng phần mềm.



##### Runtime Errors



Đoạn văn bản này tiếp tục mô tả các loại lỗi khác nhau trong mô-đun CAN của AUTOSAR. Tôi sẽ dịch nó sang tiếng Việt:

[SWS\_Can\_00026] ⌈Mô-đun CAN phải báo các lỗi gây ra bởi việc sử dụng không đúng cách API của mô-đun CAN. Điều này bao gồm kiểm tra tham số API và lỗi tuần tự gọi hàm.⌋ (SRS\_BSW\_00337, SRS\_BSW\_00323, SRS\_SPAL\_00157)

[SWS\_Can\_00091] ⌈Sau khi trả về DET, hàm của mô-đun CAN gây ra lỗi phát triển sẽ trả về ngay lập tức.⌋ (SRS\_SPAL\_12448)

[SWS\_Can\_00089] ⌈Môi trường của mô-đun CAN chỉ báo các lỗi phát triển trong giá trị trả về của một hàm của mô-đun CAN khi DET được bật và hàm cung cấp giá trị trả về. Giá trị trả về là E\_NOT\_OK.⌋ (SRS\_BSW\_00369, SRS\_BSW\_00386, SRS\_SPAL\_12448)

##### Lỗi tạm thời

Không có lỗi tạm thời.

##### Lỗi sản xuất

Mô-đun CAN không gọi Trình quản lý Sự kiện Chẩn đoán, vì không có mã lỗi sản xuất được xác định cho mô-đun CAN.

##### Giá trị trả về

CAN\_BUSY được báo cáo thông qua giá trị trả về của hàm Can\_Write. Mô-đun CanIf phản ứng theo các sơ đồ trình tự được quy định cho mô-đun CanIf.

E\_NOT\_OK được báo cáo thông qua giá trị trả về trong trường hợp có sự đánh thức trong quá trình chuyển sang chế độ ngủ.

Các sự kiện Busoff và Đánh thức được chuyển tiếp thông qua các hàm gọi lại thông báo.

#### Hỗ trợ CAN FD

Vì lý do hiệu suất, một số bộ điều khiển CAN cho phép sử dụng tính năng Flexible Data-Rate gọi là CAN FD (xem "CAN with Flexible Data-Rate" specification). Được chỉ định trong giai đoạn tranh chấp, có thể chuyển sang tốc độ baud cao hơn trong khi truyền dữ liệu và CRC. Tốc độ baud thứ hai này phải được cấu hình bằng cách mở rộng CanControllerBaudrateConfig với CanControllerFdBaudrateConfig. Nếu một tốc độ baud đang hoạt động có cấu hình CAN FD (xem CanControllerFdBaudrateConfig), tính năng CAN FD sẽ được kích hoạt cho bộ điều khiển này. Tốc độ baud thứ hai đã chỉ định cần thiết để hỗ trợ việc nhận khung CAN FD với chuyển đổi tốc độ bit (BRS). Việc tốc độ baud thứ hai có được sử dụng để truyền hay không phụ thuộc vào tham số cấu hình CanControllerTxBitRateSwitch (xem CanControllerFdBaudrateConfig).

Tuy nhiên, có thể có trường hợp cần phải truyền các tin nhắn CAN 2.0 truyền thống trong mạng hỗ trợ tin nhắn CAN-FD, ví dụ để hỗ trợ đánh thức chọn lọc CAN. Trong những trường hợp này, cần hỗ trợ truyền xen kẽ các tin nhắn CAN truyền thống với tin nhắn CAN-FD. Điều này có thể đạt được ở cấp độ khung bằng cách sử dụng hai bit có trọng số lớn nhất của CanId (xem Can\_IdType, SWS\_Can\_00416) được truyền trong Can\_Write để chỉ định loại khung nào sẽ được sử dụng.

CAN FD cũng hỗ trợ dung lượng tải trọng mở rộng cho phép truyền lên đến 64 byte. Tính năng này cũng phụ thuộc vào cấu hình CAN FD (xem CanControllerFdBaudrateConfig). Do đó, nếu bộ điều khiển CAN ở chế độ CAN FD (có CanControllerFdBaudrateConfig hợp lệ) và cờ CAN FD được đặt trong CanId được truyền đến Can\_Write(), CanDrv hỗ trợ truyền PDU với độ dài lên đến 64 byte. Nếu có yêu cầu truyền một khung CAN FD và bộ điều khiển CAN không ở chế độ CAN FD (không có CanControllerFdBaudrateConfig), khung sẽ được gửi như khung CAN truyền thống miễn là độ dài PDU <= 8 byte.

Đoạn văn này mô tả cách xử lý các loại lỗi phát triển, lỗi tạm thời và lỗi sản xuất trong mô-đun CAN. Nó cũng giải thích về khả năng hỗ trợ CAN FD (Flexible Data-Rate) của CAN, cho phép truyền khung dữ liệu lớn hơn với tốc độ baud cao hơn.

### 8 Chỉ định API

Tiền tố của tên hàm có thể được thay đổi trong một triển khai với nhiều mô-đun CAN như được mô tả trong [SWS\_Can\_00284].

#### 8.1 Kiểu dữ liệu nhập

Trong chương này, tất cả các kiểu dữ liệu được nhập từ các mô-đun sau đây được liệt kê: [SWS\_Can\_00222] ⌈