Tạ Đức Anh – 80200078 – Cơ Điện tử 2002 – PFIEV
Đại học Bách khoa thành phố Hồ Chí Minh

CONTROLLER AREA NETWORK (CAN)

1. Giới thiệu CAN

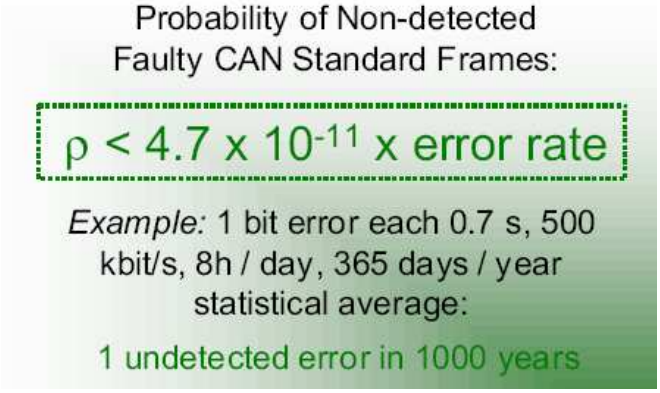
Controller Area Network (CAN) là giao thức giao tiếp nối tiếp hỗ trợ mạnh cho những hệ thống điều khiển thời gian thực phân bố (distributed realtime control system) với độ ổn định, bảo mật và đặc biệt chống nhiễu cực kỳ tốt.

CAN đầu tiên được phát triển bởi nhà cung cấp phụ tùng xe ôtô của Đức Robert Bosch vào giữa những năm 80. Để thỏa mãn yêu cầu ngày càng nhiều của khách hàng trong vấn để an toàn và tiện nghi, và để tuân theo yêu cầu việc giảm bớt ô nhiễm và tiêu thụ năng lượng, ngành công nghiệp ôtô đã phát triển rất nhiều hệ thống điện tử như hệ thống chống trượt bánh xe, bộ điều khiển động cơ, điều hòa nhiệt độ, bộ đóng cửa v.v…Với mục đích chính là làm cho những hệ thống xe ô tô trở nên an toàn, ổn định và tiết kiệm nhiên liệu trong khi đó giảm thiểu việc đi dây chằng chịt, đơn giản hóa hệ thống và tiết kiệm chi phí sản xuất, thì mạng CAN đã được phát triển.

Ngay từ khi mới ra đời, mạng CAN đã được chấp nhận và ứng dụng một cách rộng rãi trong các lĩnh vực công nghiệp, chế tạo ô tô, xe tải. Với thời gian, CAN càng trở nên thông dụng hơn vì tính hiệu quả, ổn định, đơn giản, mở và đặc biệt là chi phí rẻ. Nó được sử dụng với việc truyền dữ liệu lớn, đáp ứng thời gian thực và trong môi trường khác nhau. Cuối cùng, truyền tốc độ cao rất ổn định. Đó là lý do tại sao chúng được sử dụng trong nhiều ngành công nghiệp khác ngoài xe hơi như các máy nông nghiệp, tàu ngầm, các dụng cụ y khoa, máy dệt, v.v…

Ngày nay, CAN đã được chuẩn hóa thành tiêu chuẩn ISO11898. Hầu như mọi nhà sản xuất chip lớn như: Intel, NEC, siemens, Motorola, Maxim IC, Fairchild, Microchip, Philips, Texas Instrument, Mitsubishi, Hitachi, STmicro... đều có sản xuất ra chip CAN, hoặc có tích hợp CAN vào thành periperal của vi điều khiển. Việc thực hiện chuẩn CAN trở nên cực kỳ đơn giản nhờ sự hỗ trợ từ rất nhiều nhà sản xuất chip đó.

Điểm nổi trội nhất ở chuẩn CAN là tính ổn định và an toàn (reliability and safety). Nhờ cơ chế phát hiện và xử lý lỗi cực mạnh, lỗi CAN messages hầu như được phát hiện. Theo thống kê, xác suất để một message của CAN bị lỗi không được phát hiện là:

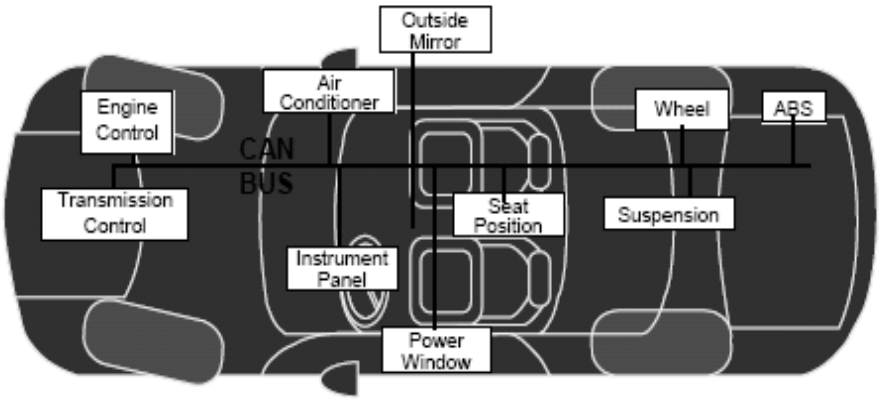


Hình 1.1. Tính ổn định của CAN (www.can-cia.org)

Ví dụ: cho rằng nếu giả sử cứ 0.7s thì môi trường tác động lên đường truyền CAN làm lỗi 1 bit. Và giả sử tốc độ truyền là 500kbits/s. Hoạt động 8h/ngày và 365ngày/ năm.

Thì trong vòng 1000 năm trung bình sẽ có một frame bị lỗi mà không phát hiện.

Miền ứng dụng của CAN trải rộng (from hight speed networks to low cost multiplex wiring) : hệ thống điện xe ô tô, xe tải, đơn vị điều khiển động cơ (engine control units), sensor, PLC comunication, thiết bị y tế…. Ngày nay CAN chiếm lĩnh trong ngành công nghiệp Ô tô. Trong những chiếc xe hơi đời mới thường có một mạng CAN high speed dùng điều khiển động cơ và thắng… một mạng CAN lowspeed dùng điều khiển những thiết bị khác như kiếng hậu, light…



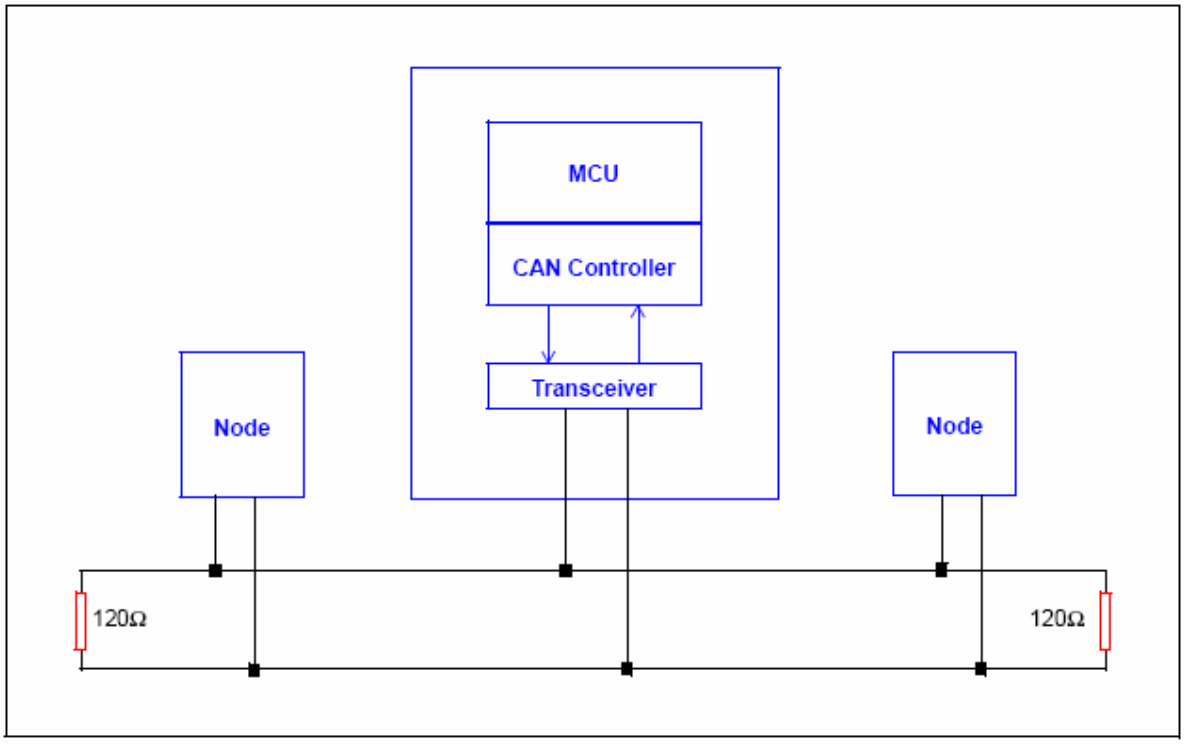
Hình 1.2. Ứng dụng mạng CAN trong điều khiển xe hơi

Chuẩn Field bus Device net, CANopen, J1939 thường dùng trong công nghiệp chính là chuẩn CAN mở rộng. (Physical layer và MAC sublayer của các chuẩn này là CAN).

2. CAN protocol overview

Chuẩn đầu tiên của CAN là chuẩn ISOP 11898-2 định nghĩa các tính chất của CAN High Speed.

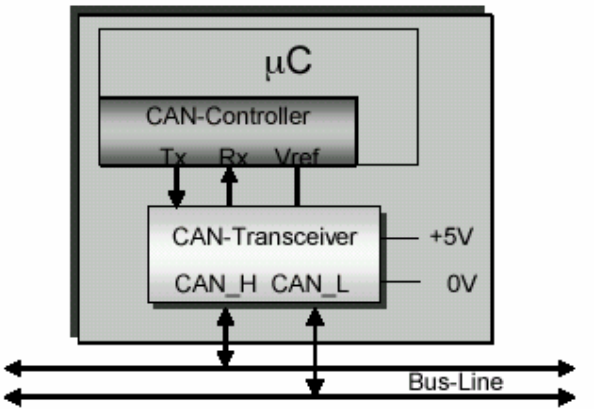
Một ví dụ về mạng CAN trong thực tế



Hình 2.1. Một ví dụ về mạng CAN

Công nghệ cáp của mạng CAN có đường dây dẫn đơn giản, giảm tối thiểu hiện tượng sự đội tín hiệu. sự truyền dữ liệu thực hiện nhờ cặp dây truyền tín hiệu vi sai, có nghĩa là chúng ta đo sự khác nhau giữa 2 đường (CAN H và CAN L). Đường dây bus kết thúc bằng điện trở 120 ohm (thấp nhất là 108 ohm và tối đa là 132 ohm) ở mỗi đầu

Mạng CAN được tạo thành bởi một nhóm các nodes. Mỗi node có thể giao tiếp với bất kỳ nodes nào khác trong mạng. Việc giao tiếp được thực hiện bằng việc truyền đi và nhận các gói dữ liệu - gọi là message. Mỗi loại message trong mạng CAN được gán cho một ID - số định danh - tùy theo mức độ ưu tiên của message đó.

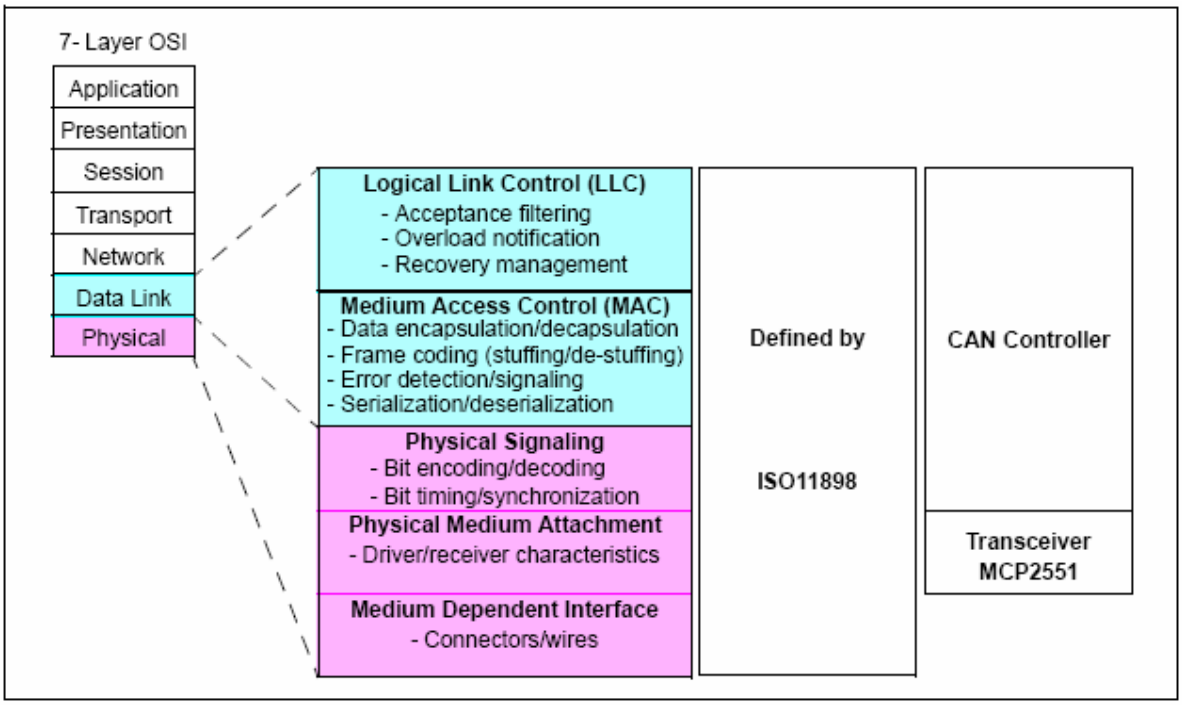


Hinh 2.2: Một nút mạng CAN

Mạng CAN thuộc loại message base system, khác với address base system, mỗi loại message được gán một ID. Những hệ thống address base thì mỗi node được gán cho một ID. Message base system có tính mở hơn vì khi thêm, bớt một node hay thay một nhóm node bằng một node phức tạp hơn không làm ảnh hưởng đến cả hệ thống. Có thể có vài node nhận message và cùng thực hiện một task. Hệ thống điều khiển phân bố dựa trên mạng CAN có tính mở, dễ dàng thay đổi mà không cần phải thiết kế lại toàn bộ hệ thống.

Mỗi node có thể nhận nhiều loại message khác nhau, ngược lại một message có thể được nhận bởi nhiều node và công việc được thực hiện một cách đồng bộ trong hệ thống phân bố.

ID của message phụ thuộc vào mức độ ưu tiên của message. Điều này cho phép phân tích response time của từng message. Ý nghĩa quan trọng trong việc thiết kế hệ thống nhúng thời gian thực. Trước khi có mạng CAN, lựa chọn duy nhất cho mạng giao tiếp trong hệ thống thời gian thực là mạng token ring chậm chạp.

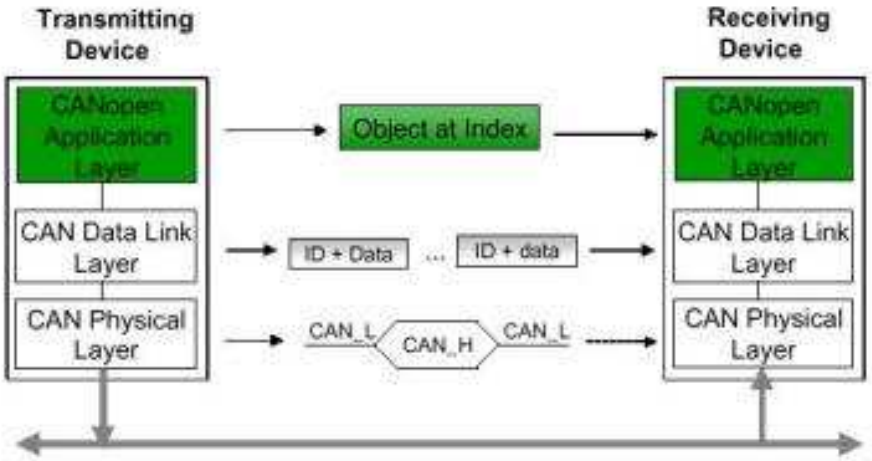


Hình 2.3: Mô hình mạng CAN

Tiêu chuẩn ISO11898 định nghĩa hai lớp Physical layer và Data link layer.

Lớp Physical layer định nghĩa cách biểu diễn/thu nhận bit 0 bit 1, cách định thời và đồng bộ hóa.

Lớp Data link layer được chia làm 2 lớp nhỏ là logical link control (LLC) và Medium Access Control (MAC): định nghĩa frame truyền và những nguyên tắc arbittration để tránh trường hợp cả hai Master cùng truyền đồng thời.



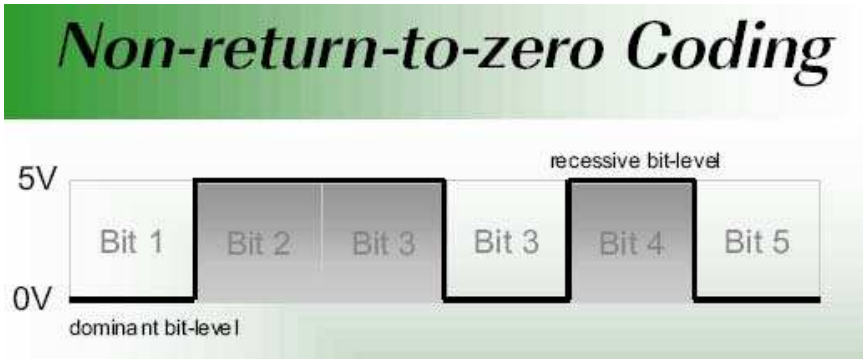
Hình 2.4: Các lớp layer giao tiếp

Ngoài ra, chuẩn CAN còn định nghĩa nhiều cơ chế khác để kiểm tra lỗi, xử lý lỗi… cơ chế kiểm tra và xử lý lỗi chia làm 5 loại lỗi: Bit error, Stuff error, CRC error, Form error, ACK error.

3. Lớp vật lý

3.1 None-return-to-zero

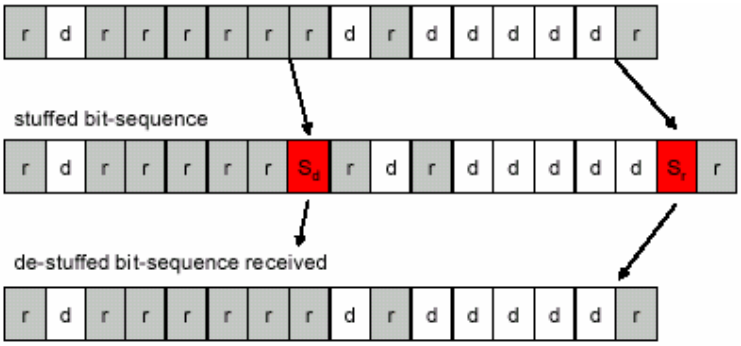
Mỗi bit trong mạng CAN được mã hóa bằng phương pháp None-return-to-zero (NRZ method). Trong suốt quá trình của một bit, mức điện áp của dây được giữ nguyên, có nghĩa trong suốt quá trình một bít được tạo, giá trị của nó giữ không đổi.



Hình 3.1: NRZ method

3.2. Bit stuffing

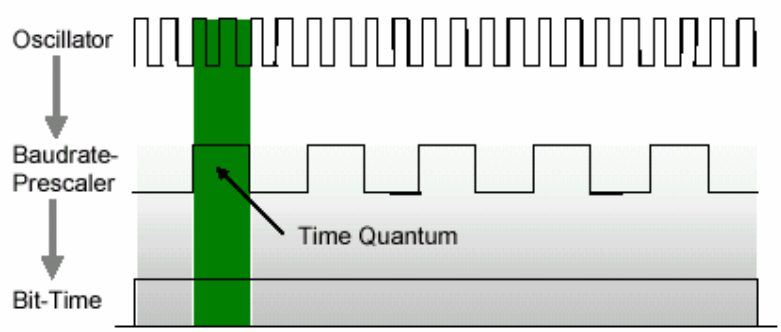
Một trong những ưu điểm của cách mã hóa NRZ là mức của bit được giữ trong suốt quá trình của nó. Điều này tạo ra vấn đề về độ ổn định nếu một lượng lớn bit giống nhau nối tiếp. Kỹ thuật Bit Stuffing áp đặt tự động một bit có giá trị ngược lại khi nó phát hiện 5 bit liên tiếp trong khi truyền.



Hình 3.2: Kỹ thuật Bit Stuffing

3.3. Bit timing

Ta định nghĩa thời gian đơn vị nhỏ nhất, là Time Quantum. Thời gian cơ bản này là một phân số của thời gian dao động của bus. Một bit khoảng 8 đến 25 quanta.



Hình 3.3: Giản đồ thời gian

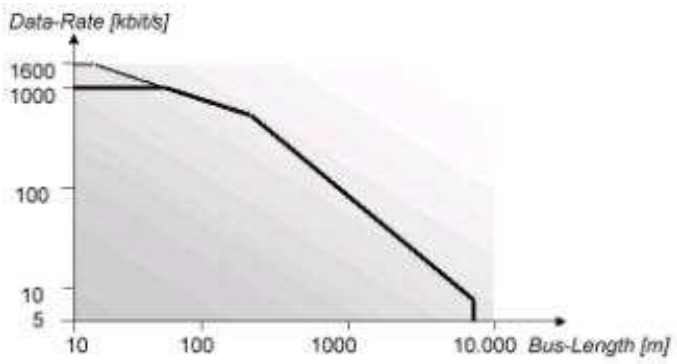
3.4. Độ dài của một bus

Độ dài của một bus phụ thuộc vào những thông số sau:

- Độ trễ lan truyền trên đường dây của bus

- Sự khác nhau của thời gian Time Quantum (định nghĩa ở trên), vì sự khác nhau của xung clock tại các nút

- Biên độ tín hiệu thay đổi theo điện trở của cáp và tổng trở vào của các nút



Hình 3.4: Tốc độ tỉ lệ nghịch với độ dài bus

Bit Rate	Bus Length	Nominal Bit-Time
1 Mbit/s	30 m	1 µs
800 kbit/s	50 m	1,25 µs
500 kbit/s	100 m	2 µs
250 kbit/s	250 m	4 µs
125 kbit/s	500 m	8 µs
62,5 kbit/s	1000 m	20 µs
20 kbit/s	2500 m	50 µs
10 kbit/s	5000 m	100 µs

Bảng 3.1 : Vận tốc – Độ dài – Bit time

Cần chú ý rằng bất cứ modul nào kết nối vào một bus CAN phải được hỗ trợ với tốc độ tối thiểu là 20kbit/s. Để sử dụng bus có độ dài hơn 200 m, cần thiết phải sử dụng một optocoupleur, và để sử dụng bus dài hơn 1 km, phải cần một hệ thống kết nối trung gian như repeater hoặc bridge.

3.5 Trạng thái "dominant" và "recessive"

Ở lớp vật lý, Bus CAN định nghĩa hai trạng thái là "dominant" và "recessive", tương ứng với hai trạng thái là 0 và 1. Trạng thái "dominant" chiếm ưu thế so với trạng thái "recessive". Bus chỉ ở trạng thái "reccessive" khi không có node nào phát đi trạng thái "dominant". Điều này tạo ra khả năng giải quyết chanh chấp khi nhiều hơn một Master cùng muốn chiếm quyền sử dụng bus.

Bởi tính chất vật lý của bus, cần thiết phải phân biệt 2 dạng truyền:

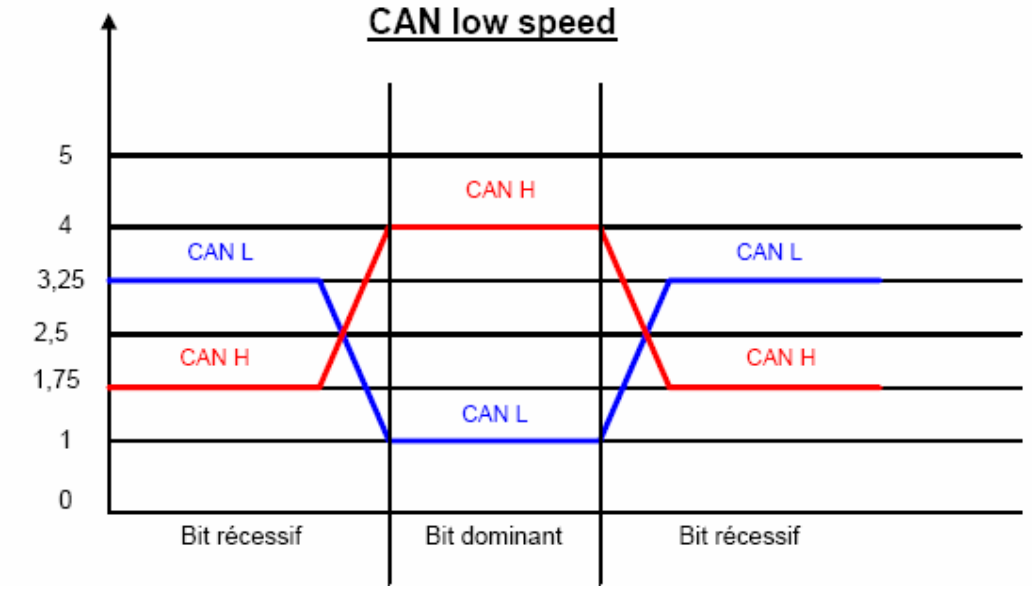
- Truyền CAN low speed
- Truyền CAN high speed

Bảng sau tổng kết những tính chất cơ bản khác nhau giữa 2 dạng, đặc biệt là tốc độ:

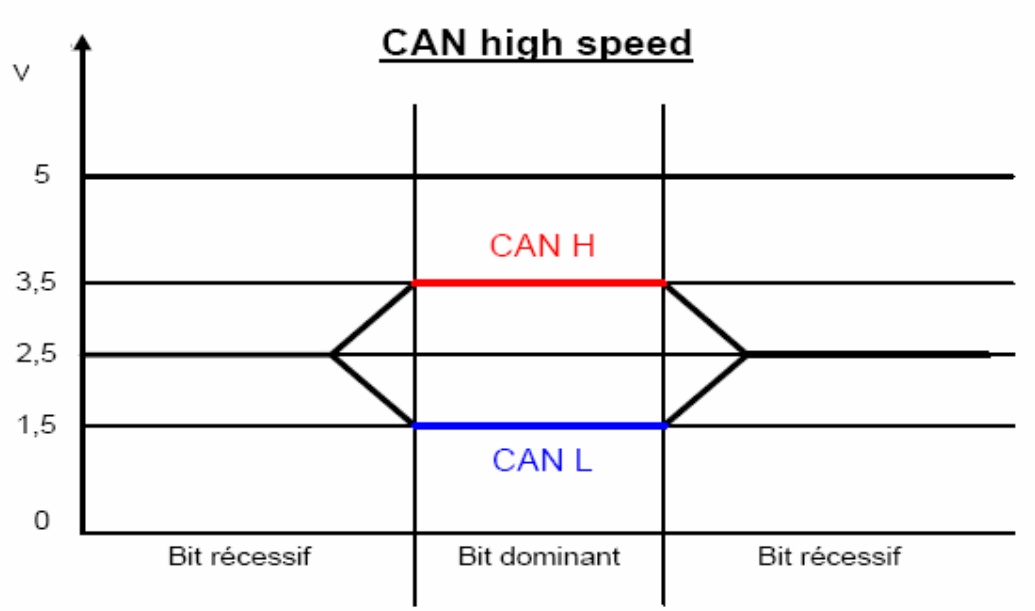
Thông số	CAN low speed	CAN high speed
Tốc độ	125 kb/s	125 kb/s tới 1Mb/s
số nút trên bus	2 tới 20	2 tới 30
Trạng thái dominant	CAN H = 4V ; CAN L = 1V	CAN H = 3,25V ; CAN L = 1,5V

Trạng thái recessive	CAN H = 1,75V ; CAN L = 3,25V	CAN H = 2,5V ; CAN L = 2,5V
tính chất của cap	30pF giữa cáp và dây	2*120 ohm
Mức điện áp cung cấp	5V	5V

Bảng 3.2: So sánh CAN low speed và CAN high speed

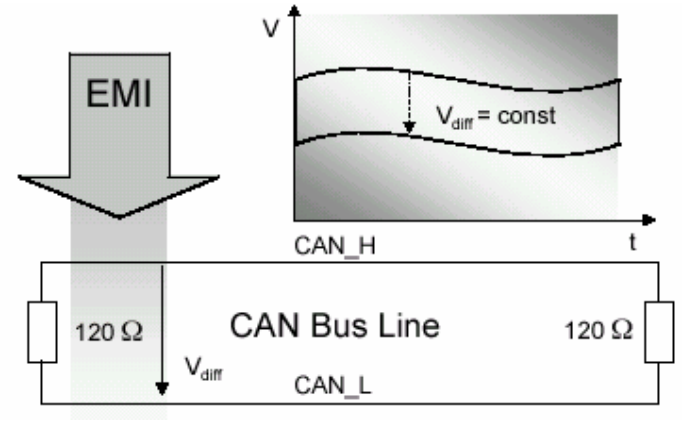


Hình 3.5: Điện áp của CAN low speed



Hình 3.6: Điện áp của CAN high speed

Vì tính chất vi sai trên đường truyền tín hiệu của bus CAN, sự miễn trừ tác động điện từ được bảo đảm vì 2 dây của bus đều bị tác động như nhau cùng một lúc bởi tín hiệu nhiễu.



Hình 3.7: Sự kháng nhiễu với ảnh hưởng của điện từ

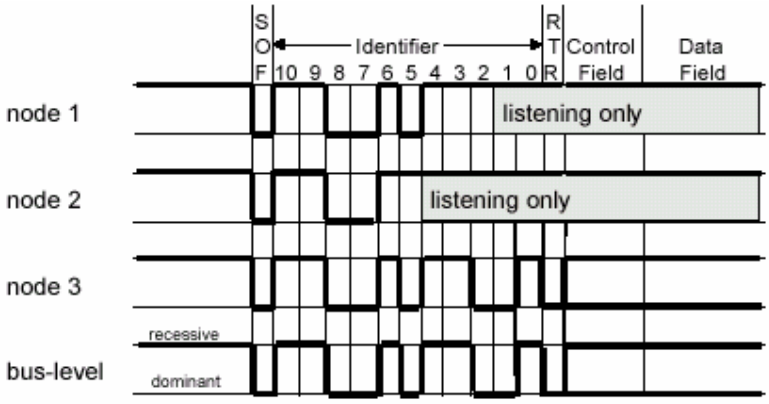
4. Giải quyết tranh chấp trên bus

Phương thức giao tiếp của bus CAN là sự phát tán thông tin (broadcast): mỗi điểm kết nối vào mạng thu nhận fame truyền từ nút phát. Sau đó, nỗi nút sẽ quyết định việc sử lý message, có trả lời hay không, có phản hồi hay không… Cách thức này giống như sự phát thông tin về đường đi của một trạm phát thanh,: khi nhận được thông tin về đường đi, người lái xe có thể thay đổi lộ trình của anh ta, dừng xe hay thay đổi tài xế hoặc chẳng làm gì cả…

Giao thức CAN cho phép các nút khác nhau đưa dữ liệu cùng lúc và một quá trình nhanh chóng , ổn định của cơ chế arbitration sẽ xác định xem nút nào được phát đầu tiên.

Để sử lý thời gian thực, dữ liệu phải được truyền nhanh. Điều này ảnh hưởng không chỉ đường truyền vật lý cho phép tới 1Mbit/s, mà còn đòi hỏi một sự cấp phát nhanh bus trong trường hợp xung đội, khi mà rất nhiều nút muốn truyền đồng thời. Khi trao đổi dữ liệu trên bus, thứ tự sẽ được xác định dựa vào loại thông tin. Ví du, các giá trị hay biến đổi nhanh, như trạng thái của một cảm biến, hay phản hồi của một động cơ, phải được truyền liên tục với độ trễ thấp nhất, hơn là các giá trị khác như nhiệt độ của động cơ, các giá trị thay đổi ít. Trong mạng CAN , phần ID của mỗi message, là một từ gồm 11 bit (version 2.0A) xác định mức ưu tiên. Phần ưu tiên này nằm ở đầu mỗi message. Mức ưu tiên được xác định bởi 7 bit cho verdion 2.0A, tới 127 mức vá mức 128 là 0000000 theo NMT(Netword Management)

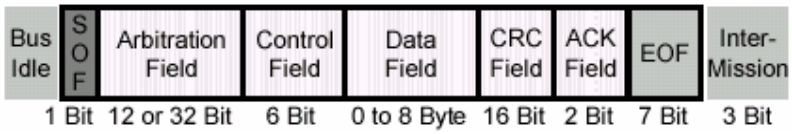
Quy trình arbitration của bus dựa trên phân giải từng bit, theo những nút đang tranh chấp, phát đồng thời trên bus. Nút nào mức ưu tiên thấp hơn sẽ mất sự cạnh tranh với nút có mức ưu tiên cao.



Hình 4.1: Giải quyết tranh chấp trên bus

5. CAN frame

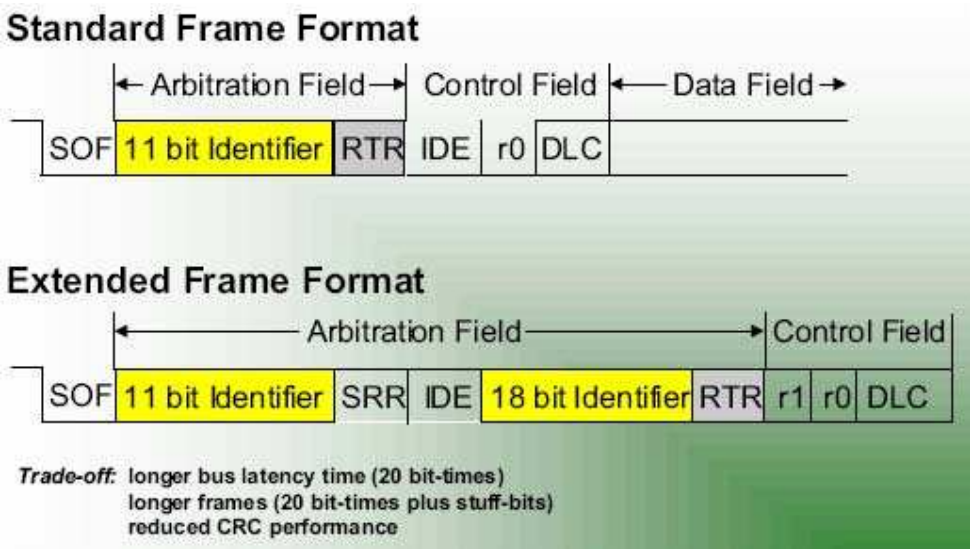
Một khung truyền có dạng sau:



Hình 5.1: Khung truyền

Chuẩn CAN định nghĩa bốn loại Frame: Data frame dùng khi node muốn truyền dữ liệu tới các node khác. Remote frame dùng để yêu cầu truyền data frame. Error frame và overload frame dùng trong việc xử lý lỗi.

- **Data frame:** dùng để truyền đi một message. Có hai dạng: standard frame và extended frame

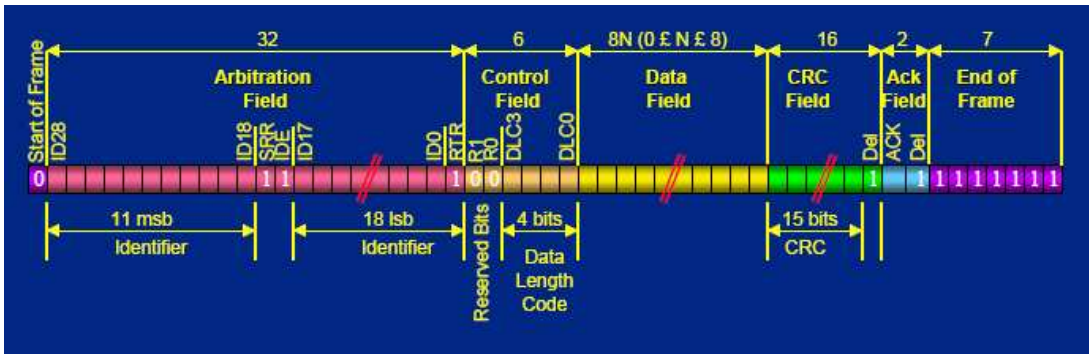


Hình 5.2. CAN data frame

Standard frame: bắt đầu bằng 1 bit start of frame (SOF) luôn ở trạng thái dominant, 11bit ID tiếp theo, 1 bit Remote Transmit Request (RTR) để phân biệt remote frame và data frame nếu bằng dominant nghĩa là data frame, nếu bằng recesive nghĩa là remote frame. Tiếp đến là 1 bit Identifier Extension (IDE) để phân biệt giữa Standard frame ("dominant") và extended frame ("reccesive"). Tiếp theo là 1 bit r0 luôn ở trạng thái dominant. Tiếp đến là 3 bit Data Length Control cho biết số lượng byte data cuả frame. Tiếp đến là 0 đến 8 bytes data. Tiếp đến là 15 bit CRC và 1bit CRC delimiter. tiếp đến là 1bit Acknoledge và 1 bit delimiter, tiếp theo là 7bits End of frame luôn ở trạng thái recesive. cuối cùng là khoảng cách tối thiểu giữa hai frame truyền inter-frame space (IFS).

Hình 5.3. CAN standard frame

Extended frame: gần giống như standard frame, và có 29 bit ID:

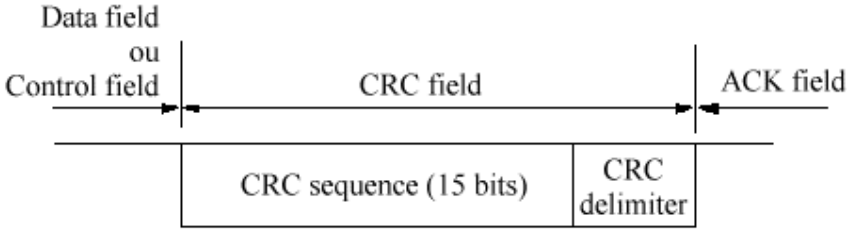


Hình 5.4. CAN extended frame

Chi tiết các phần khác nhau trong một khung truyền dữ liệu:

Start of Frame: nắm phần đầu của một farme dữ liệu hay Remote frame, luôn ở trạng thái dominant. Một nút có thể bắt đầu truyền dữ liệu nếu bus rãnh. Sau đó tất cả các nút đều đồng bộ sau SOF của nút bắt đầu truyền.

CRC Field:



Hình 5.5: CRC Field

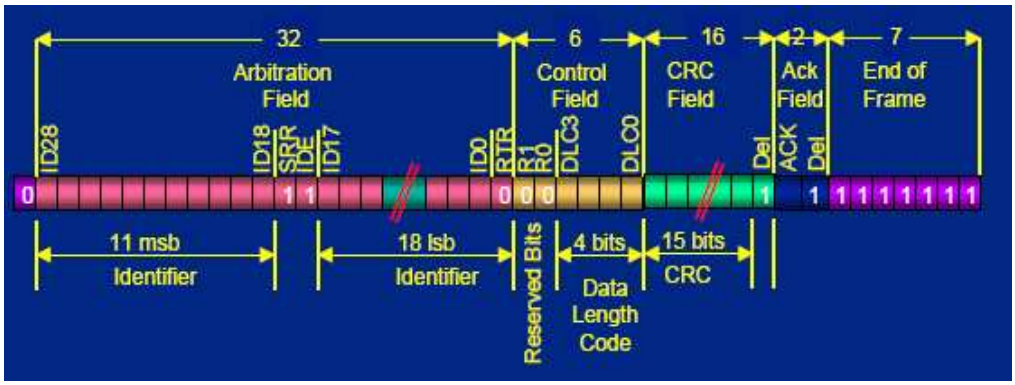
CRC Field bao gồm một chuỗi gồm 15 bit và CRC Delimiter (là 1 bit reccesive)
Một chuỗi CRC (Cyclic Redundancy Code) cho phép kiểm tra sự nguyên vẹn của dữ liệu truyền. Tất cả các nút nhận phải thực hiện quy trình kiểm tra này. Chỉ vùng SOF, vùng tranh chấp, vùng điều khiển và vùng dữ liệu được sử dụng để tính toán chuỗi CRC. Trên thực tế, độ dài cực đại của frame không vượt quá 2^{15} bit cho một chuỗi CRC 15 bit.
Trình tự tính toán :

- Các bit vừa nêu (trừ các bit-Stufing thêm vào), bao gồm các bit từ đầu frame tới cuối vùng dữ liệu (cho frame dữ liệu)hay cuối vùng điếu khiển (cho Remote frame) được coi như một hàm f(x) với hệ số là 0 và 1 theo sự hiên diện, số lượng của mỗi bit. Đa thức nhận được sẽ nhân với x^{15} sau đó chia cho hàm $g(x) = x^{15} + x^{14} + x^{10} + x^8 + x^7 + x^4 + x^3 + 1$. Chuỗi bit tương ứng với hàm này là : 1100010110011001.
- Số dư cửa phép chia modulo [2]hàm f(x) cho g(x) sẽ tạo thành chuỗi CRC 15 bit.
- Nếu sai số CRC được phát hiện khi kết quả gửi đi khác với kết quả nhận được, thì bên nhận sẽ gửi đi một message lỗi dưới dạng request frame.

ACK Field:
Gồm 2 bit : ACK slot và ACK Delimiter (là 1 bit recesive)

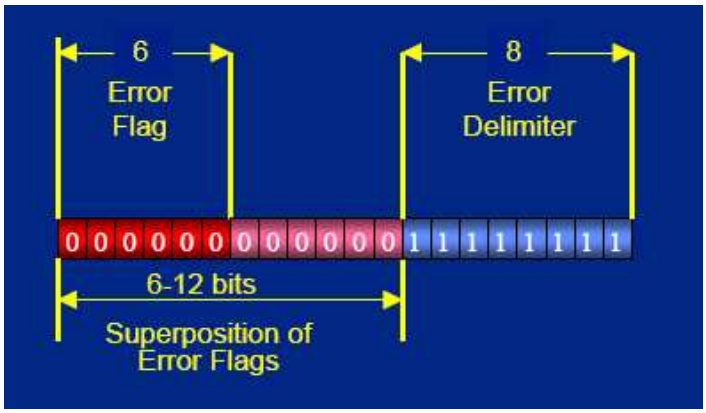
- một nút đang truyền sẽ gửi một bit recesive trong ACK slot
- một nút nhận đúng message thông báo cho nút truyền sẽ gửi 1 bit dominant trong ACK slot

- **Remote frame:** dùng để yêu cầu truyền data frame tới một nút khác. Gần giống data frame nhưng có DLC=0 và không có data field.



Hình 5.6: CAN remote frame

- **Error frame:** được phát ra khi node phát hiện lỗi



Hình 5.7: CAN error frame

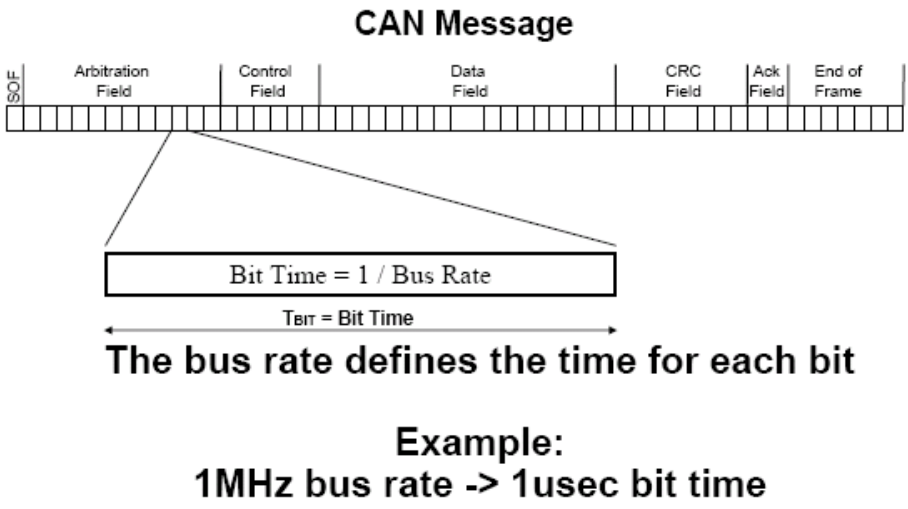
Frame lỗi bao gồm 2 phần:

- Cờ lỗi
- Phần delimiter

- **Overload frame:** dùng khi frame bị tràn bộ đệm

6. Nominal Bit Time

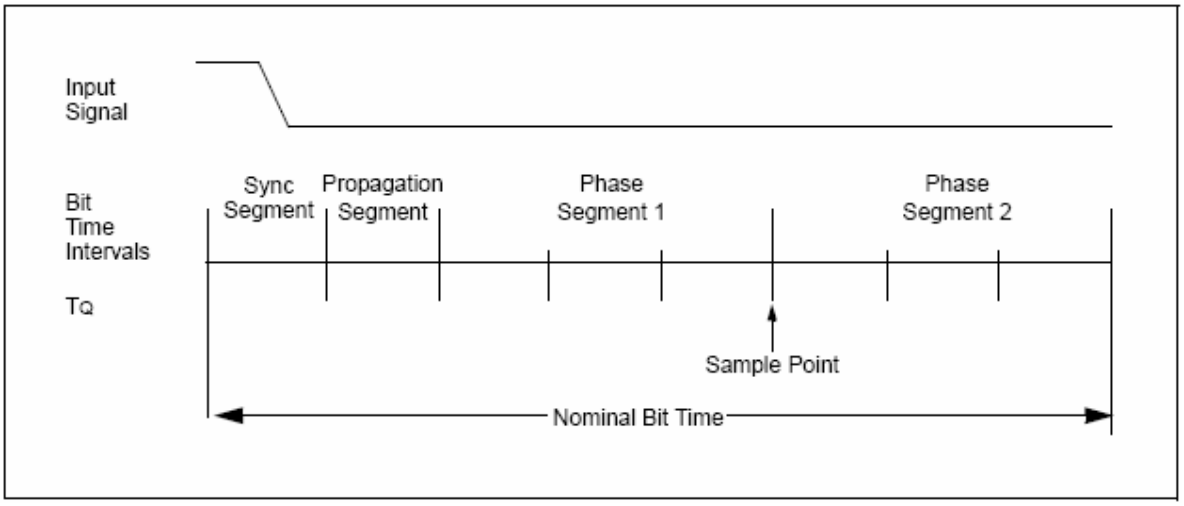
Nominal Bit Time là độ dài của một bit trên bus. Mỗi nút trên bus phải điều chỉnh nhịp cùng với Nominal Bit Time để có thể phát và nhận chính xác dữ liệu trện bus.



Hình 6.1. Baudrate định nghĩa thời gian cho 1 bit

Chuẩn BOSCH mô tả thành phần của Nominal Bit Time, được chia ra thành nhiều đoạn (segment):

- Đoạn đồng bộ (SYNC_SEG)
- Đoạn lan truyền (PROG_SEG)
- Đoạn pha buffer 1 (PHASE_SEG1)
- Đoạn pha buffer 2 (PHASE_SEG2)



Hình 6.2: Mỗi bít được cấu tạo bởi 4 segments

Nominal Bit Time, tính theo giây, là nghịch đảo của dung lượng trên bus:

$$Nominal\_Bit\_Time = \frac{1}{Nominal\_Bit\_Rate}.$$

6.1 Các segment khác nhau:

- Segment đồng bộ: sử dụng để đồng bộ các nút khác nhau trên bus. Một chuyển trạng thái (từ 0 xuống 1) phải được thực hiện trong phần này để cho phép đồng bộ xung nhịp lại của những nút khác nhau trong khi nhận frame

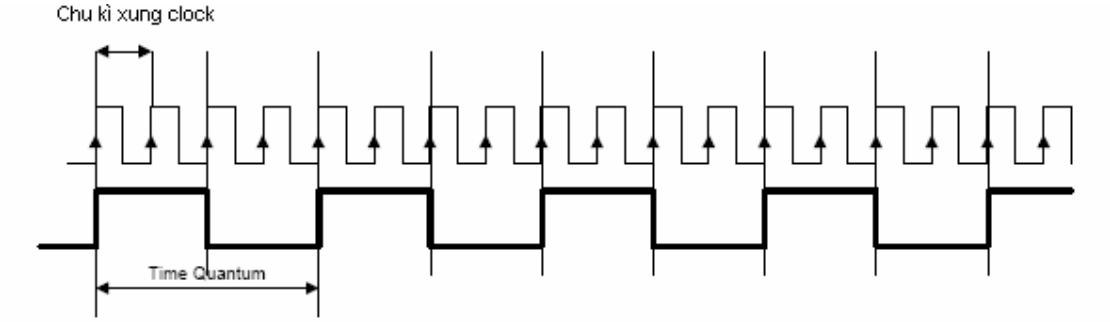
- Segment lan truyền: được sử dụng để bù trừ thời gian lan truyền trên bus.
- Segment bộ đệm pha 1 và 2: sử dụng để bù trừ lỗi của pha xác định khi truyền. Chúng ta sẽ thấy các segment thay đổi dài ngắn vì hiện tượng đồng bộ lại (resynchronisation)
- Điểm lấy mẫu: là điểm mà giá trị của bit được đọc bởi bus. Nó nằm cuối đoạn "buffer phase 1" và là điểm duy nhất cho mức của bit.

6.2 Khoảng thời gian khác nhau của các segment và Time Quantum

Time Quantum: là một đơn vị thời gian tạo thành từ chu kì dao động nội của mỗi nút. Time Quantum gồm rất nhiều xung clock của bộ dao động. Chu kì xung clock được gọi là minimum Time Quantum. Giá trị pre-scale là m thì:

$$TIME\_QUANTUM = m * MINIMUM\_TIME\_QUANTUM$$

Gái trị của m có thể dao động từ 1 đến 32, Hình mô tả cấu trúc của Time Quantum từ chu kì xung clock nội của 1 nút:



Hình 6.3: Cấu trúc của Time Quantum

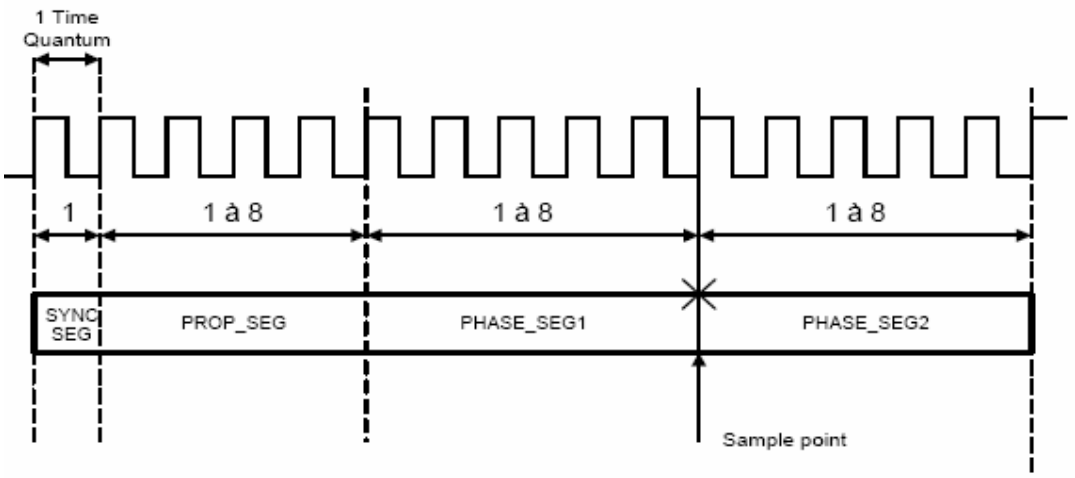
Trong ví dụ trên thì m bằng 2

- Thời gian của mỗi segment:

Segment	Thời gian của mỗi segment
Synchronisation - SYNC_SEG	1
Propagation - PROP_SEG	1 à 8
Buffer phase1 - PHASE_SEG1	1 à 8
Buffer phase1 - PHASE_SEG2	1 à 8

Bảng 6.1: Thời gian của mỗi segment

Số Time Quanta trong mỗi Nominal Bit Time thay đổi từ 8 đến 25. Hình đưa ra số lượng Time Quanta có thể cho mỗi segment của Nominal Bit Time:

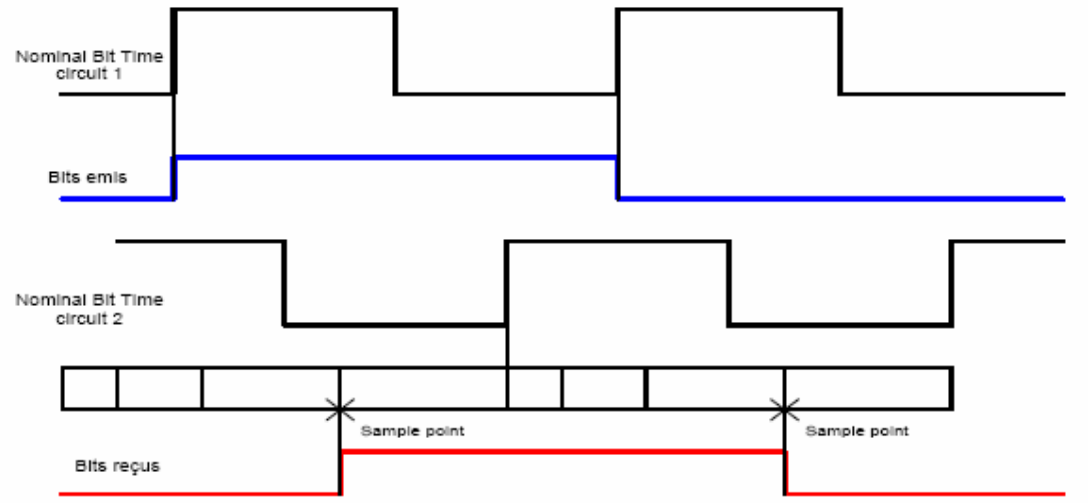


Hình 6.4 Số lượng Time Quanta có thể cho mỗi segment

Sự lựa chọn số lượng Time Quanta cho mỗi segment phụ thuộc vào tần số của bộ dao động. Một số lượng lớn Time Quanta cho segment sẽ tăng tính chính xác của sự đồng bộ của các nút trong bus.

7. Sự đồng bộ xung clock

Mỗi nút phải tạo một thời gian danh nghĩa Bit Time để có thể nhận và phát dữ liệu xuống bus với sự đồng bộ các nút khác. Thực tế, nếu Nominal Bit Time của mỗi nút không được đồng bộ với nhau, giá trị đọc từ bus tại thời điểm lấy mẫu có thể không là giá trị đúng với thời điểm mong muốn. Độ trễ này có thể làm ảnh hưởng trong nút nhận frame, khi mà có ít thời gian tính toán CRC và gởi 1 bit dominant trong ACK Slot để xác nhận rằng frame đã đúng.



Hình 7.1: Vấn đề đồng bộ

7.1 SJW (Synchronization Jump Width)

SJK điều chỉnh một bit clock đi 1-4 TQ (được khởi tạo trước trong thanh ghi và không đổi trong quá trình hoạt động) để thực hiện việc đồng bộ với message truyền.

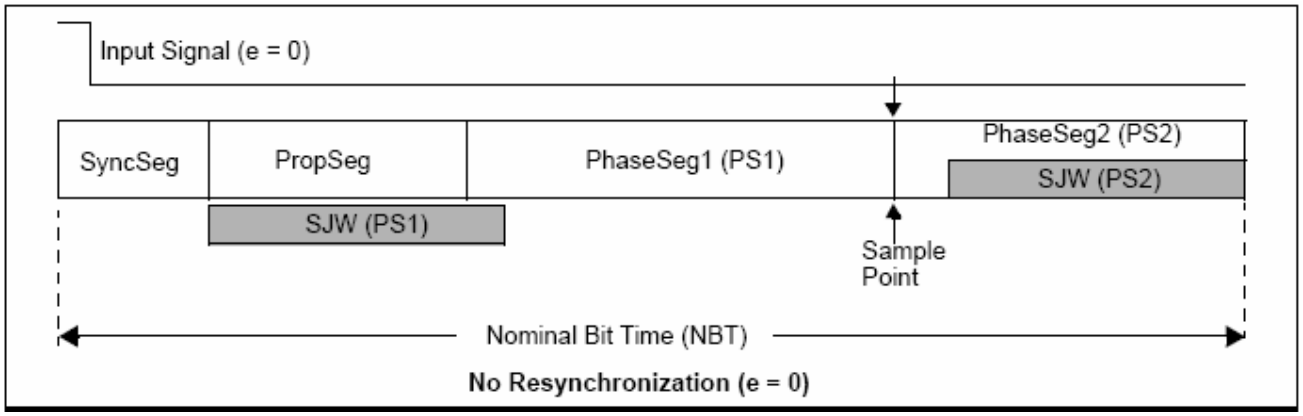
7.2 Lỗi pha

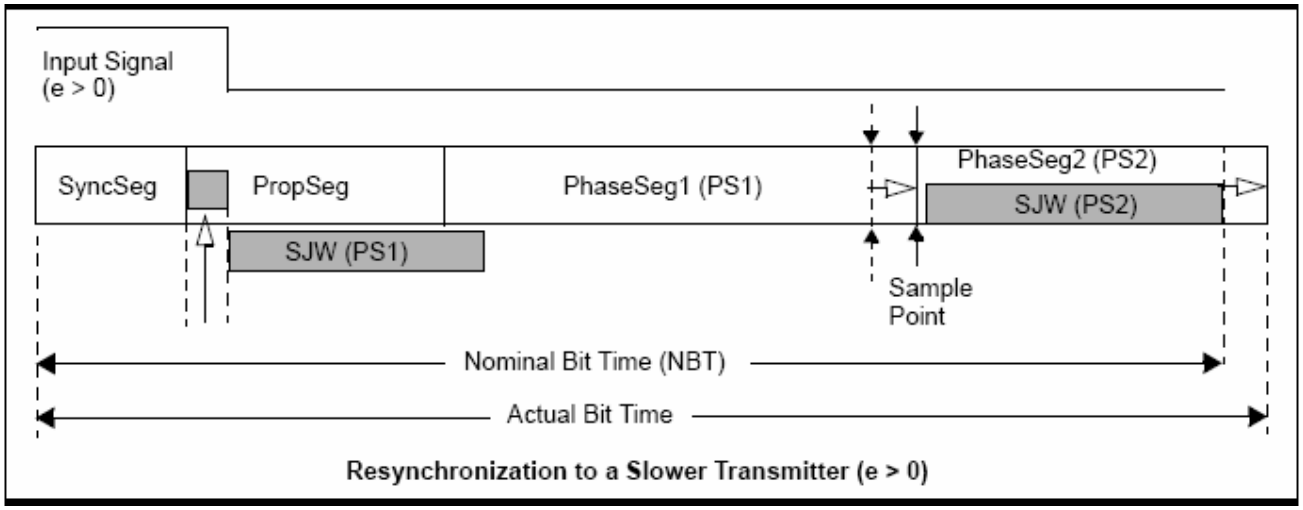
PHASE_ERROR được phát hiện khi sự thay đổi bit dominant thành recessive hay 1 bit recessive thành dominant không xảy ra bên trong segment đồng bộ. Một biến e được sư dụng để đánh giá lỗi này và đưa ra tín hiệu. Sự tính toán e được thực hiện như sau:

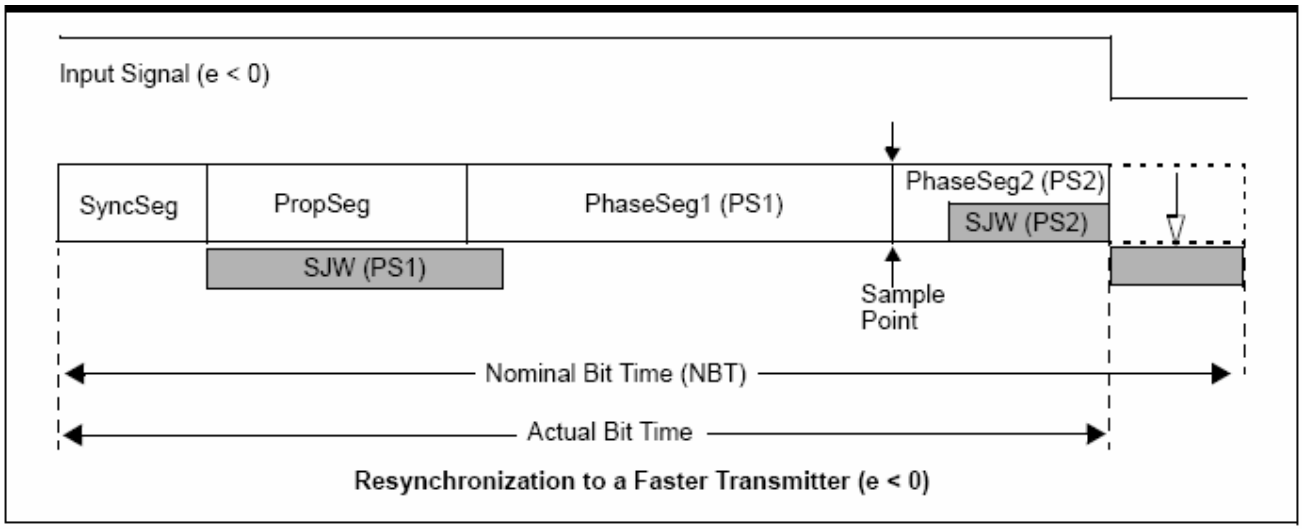
- e=0,khi sự thay đổi bit xảy ra bên trong segment đồng bộ (SYNC_SEG)
- e>0, khi sự thay đổi bit xảy ra trước thời điểm lấy mẫu
- e<0, khi sự thay đổi bit xảy ra sau thời điểm lấy mẫu

Cơ chế trên phục vụ cho việc đồng bộ lại những Nominal Bit Time khác nhau của mỗi nút trên bus. Cơ chế đồng bộ này cũng áp dụng cho sự chuyển bit recessive sang dominant hay ngược lại khi có 5 bit liên tiếp cùng loại theo cơ chế Bit-Stuffing.

Lỗi pha e tính toán so với thời điểm lấy mẫu để xác định PHASE_SEG 1 phải dài hơn hay PHASE_SEG 2 phải ngắn đi để lần chuyển trang thái bit tiếp theo sẽ vào segment đồng bộ. Hình đưa ra chuỗi dịch chuyển độ dài của segment của Nominal Bit Time:







7.3 Cơ chế đồng bộ

Đồng bộ cứng (Hard Synchronization): chỉ xảy ra khi chuyển cạnh bit đầu tiên từ recessive thành dominant (logic"1" thành "0") khi bus rảnh, báo hiệu 1 Start of Frame (SOF). Đồng bộ cứng làm cho bộ đếm bit timing khởi động lại, gây nên một chuyển cạnh trong SyncSeg. Tại thời điểm này, mọi nút nhận sẽ đồng bộ với nút phát. Đồng bộ cứng chỉ xảy ra một lần trong suốt một message. Và đồng bộ lại có thể không xảy ra trong cùng một bit (SOF) khi mà đồng bộ cứng đang xảy ra.

Đồng bộ lại (Resynchronization): được thực hiện để bảo toàn sự đồng bộ đã thực hiện bởi đồng bộ cứng. Thiếu đồng bộ lại, nút nhận không thể có được sự đồng bộ vì sự lệch pha của các bộ dao động tại mỗi nút.

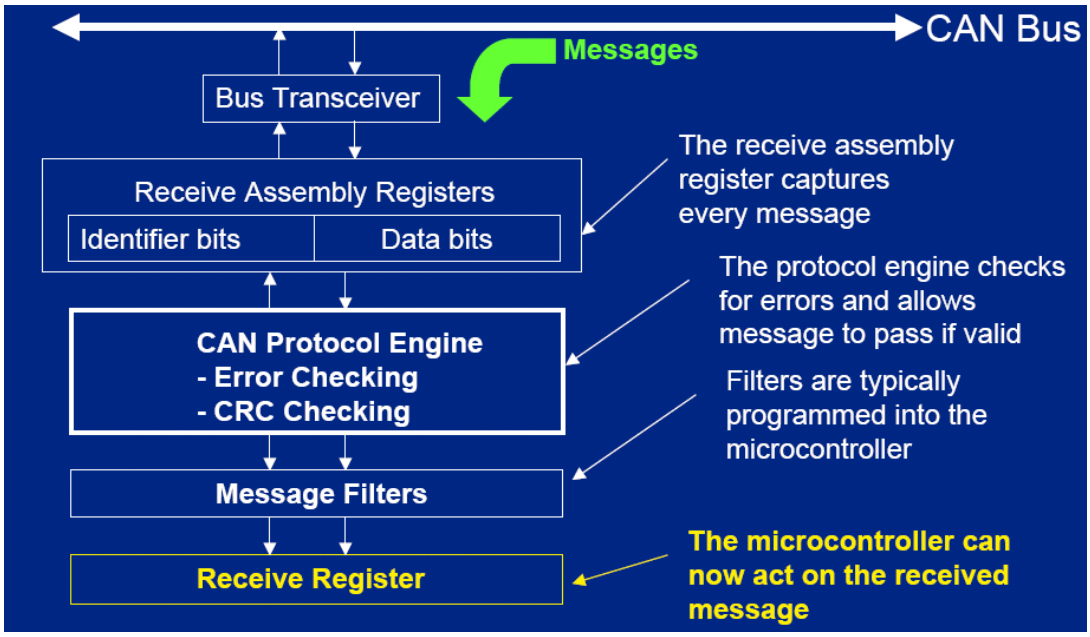
Sự tính toán và mức độ đồng bộ lại được đưa ra từ giá trị sai số pha e, và cũng phụ thuộc vào giá trị SJW:

- Nếu sai số pha e bằng 0 (e=0, chuyển cạnh trong Sync Seg), cơ chế đồng bộ lại cũng giống như đồng bộ cứng.
- Nếu sai số pha e dương và bé hơn giá trị tuyệt đối SJW (0<e<SJW) , PHASE_SEG 1 sẽ kéo dài thêm 1 đoạn e.
- Nếu sai số pha e âm nhưng nhỏ giá trị SJW về tuyệt đối (e<0 và |e|<SJW), PHASE_SEG 2 sẽ ngắn lại 1 đoạn e.
- Nếu sai số pha e dương và lớn hơn hay bằng SJW (e>0 và e>SJW), PHASE_SEG 1 sẽ kéo dài thêm 1 đoạn SJW
- Cuối cùng, Nếu sai số pha e âm nhưng lớn hơn giá trị SJW về tuyệt đối (e<0 và |e|>SJW), PHASE_SEG 2 sẽ ngắn lại 1 đoạn SJW.

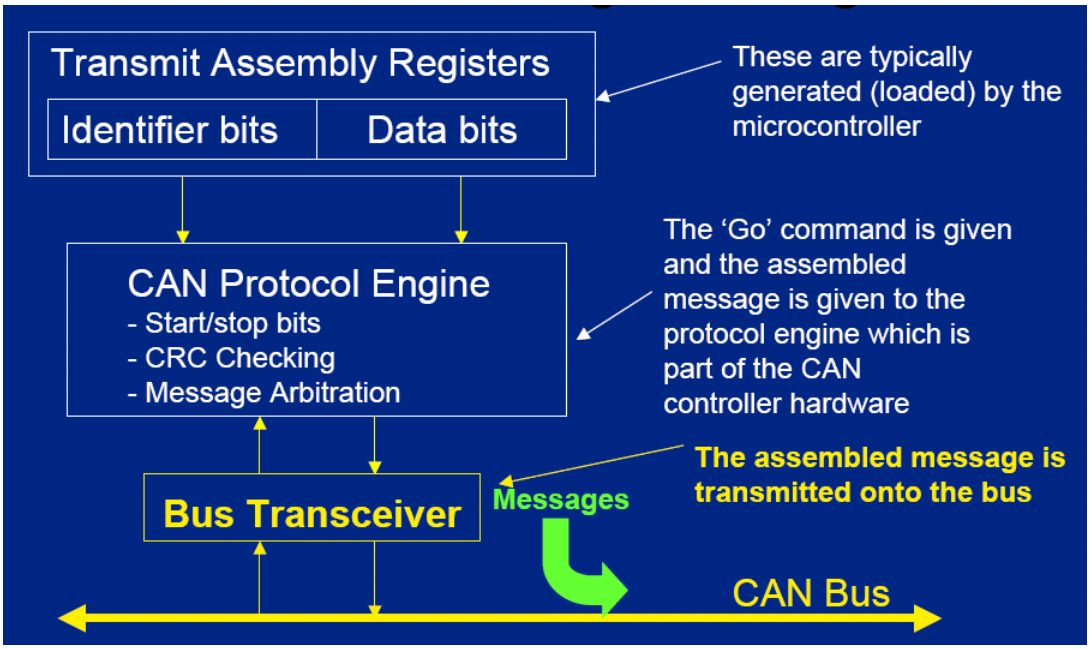
Bảng sau tóm tắt kết quả của cơ chế trên:

Lỗi pha	Tác động lên PHASE_SEG 1	Tác động lên PHASE_SEG 2
0<e<SJW	kéo dài thêm e	
e<0 và |e|<SJW		làm ngắn 1 đoạn e
e>0 và e>SJW	kéo dài thêm SJW	
e<0 và |e|>SJW		làm ngắn 1 đoạn SJW

8. Truyền nhận message



Hình 8.1: Sơ đồ khối bộ nhận CAN message



Hình 8.2: Sơ đồ khối bộ truyền CAN message

9. Sử lý lỗi

Khi truyền một frame trên bus, lỗi truyền có thể ảnh hưởng đến hoạt động của các nút trên bus. Lỗi có thể đến từ một nút, làm cho mạng không còn hoạt động chính xác, Vì vậy, nhiều cách phát hiện lỗi được sử dụng trong CAN

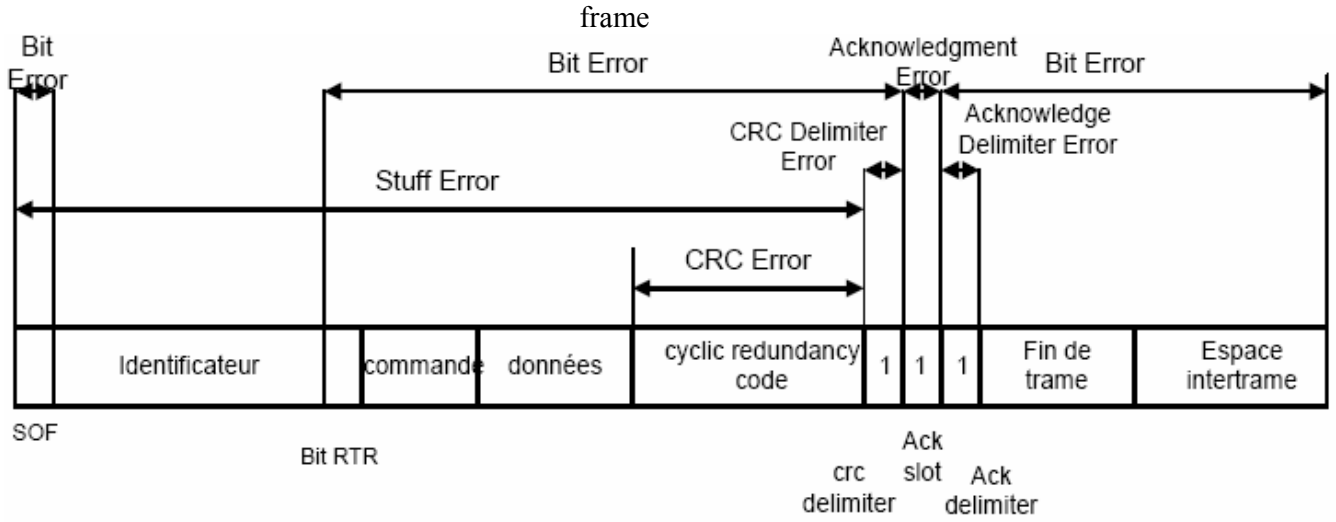
Các loại lỗi:

- Bit Error:mỗi khi nút truyền gửi một bit xuống bus, nó kiểm tra xem mức điện áp trên bus có đúng với bit cần gửi hay không. Nếu không đúng, nó sẽ báo hiệu bằng một Bit Error.

Tuy nhiên, Bit Error sẽ không báo hiệu trong những trường hợp sau:
- Không có Bit Error nào được tác động khi một bit dominant được gửi trong vùng ID thay thế cho một bit recessive. Cũng như vậy, trong vùng ACK Slot, thay cho một bit recessive.
- Một nút phát gửi một cờ lỗi (bit recessive) và nhận bit dominant, ko cần phải báo hiệu Bit error.

- Lỗi Stuffing(Stuff Error): Một lỗi Stuffing được phát hiện trong mỗi lần có 6 bit hay nhiều hơn liên tục trên một đường dây của Bus
  Tuy nhiên, lỗi Stuffing sẽ không báo trong vùng ID, vùng điều khiển và vùng CRC. Cơ chế Bit Stuffing không áp dụng sau CRC. Trong mọi trường hợp, lỗi Bit-Stuffing sẽ không báo trong đoạn kết thúc của frame hay trong vùng ACK
- Lỗi Cyclic Redundancy(CRC Error)
  Nếu giá trị CRC tính toán bởi nút nhận không giống với giá trị gửi đi bởi nút phát, Sẽ có một lỗi CRC(CRC Error).
- Lỗi ACK Delimiter
  Một lỗi ACK Delimiter được báo khi nút nhận không thấy một bit recessive trong vùng ACK Delimiter hay trong vùng CRC Delimiter.
- Lỗi Slot ACK (ACK Error)
  Một lỗi Slot ACK được báo bởi nút phát khi nó không đọc thấy bit dominant trong vùng Slot ACK.

Hình tổng hợp những loại lỗi khác nhau trong từng phần của một message



Hình 9.1: Các loại lỗi khác nhau

10. CAN MODULE trên PIC

Controller Area Network (CAN) là modul thực hiện các chuẩn giao tiếp CAN 2.0A hay B đã được định nghĩa bởi BOSCH. Modul hỗ trợ CAN 1.2, CAN 2.0A, CAN 2.0B, CAN 2.0A, CAN 2.0 B Passive và CAN 2.0 Active.

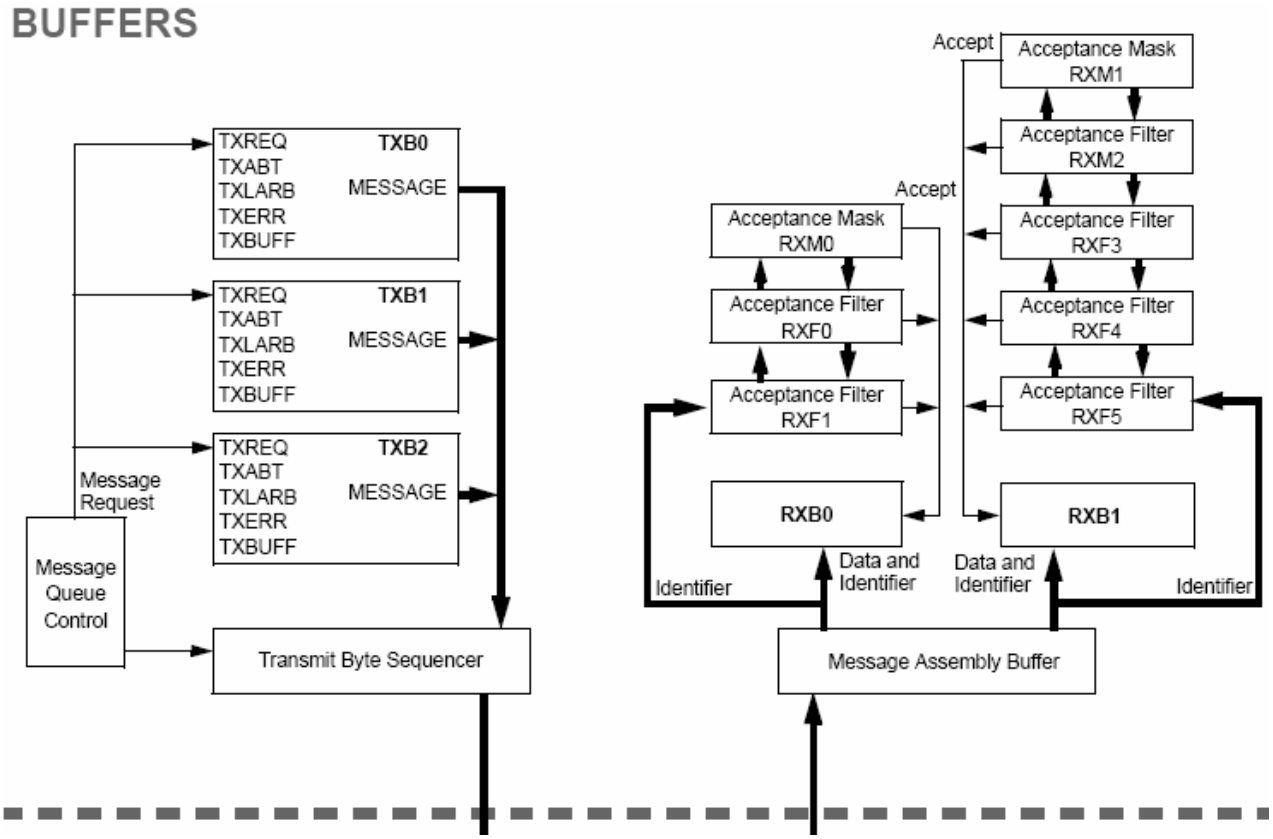
Module bao gồm:

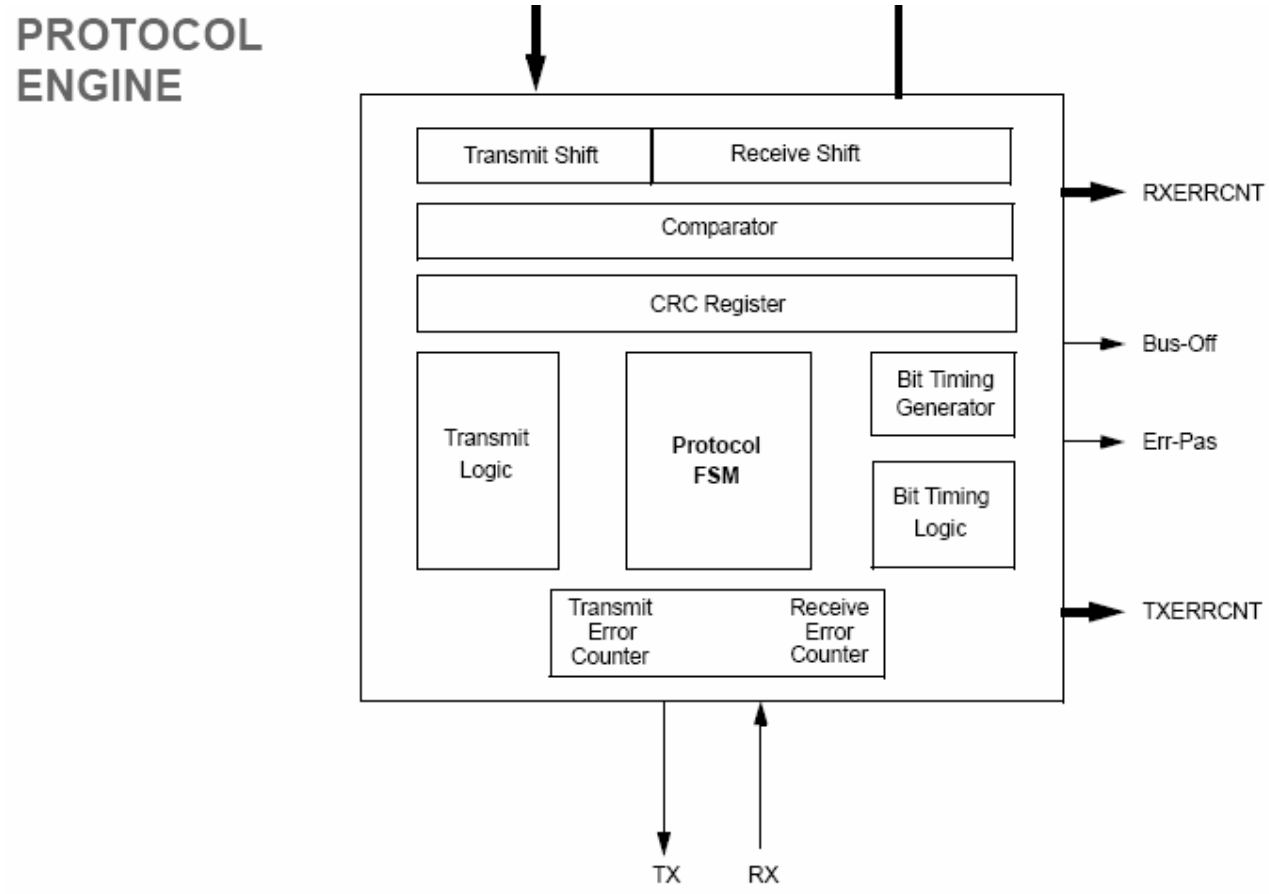
- Thực hiện các giao thức CAN 1.2, CAN 2.0A và CAN 2.0B
- Hỗ trợ các loại Frame chuẩn và mở rộng
- Độ dài dữ liệu từ 0-8 byte

- Lập trình tốc độ tới 1Mbit/s
- 2 buffer nhận với hai buffer chứa message với 2 mức ưu tiên
- 3 buffer truyền với chế độ ưu tiên và khả năng bỏ truyền.
- Các ngắt do lỗi truyền nhận.
- Lập trình xung clock.

10.1 Tổng quan về module

Module bao gồm một engine giao tiếp, các buffer message và điều khiển. Engine tự động làm tất cả các chức năng nhận và truyền dữ liệu. Message được truyền bằng cách ghi vào các thanh ghi tương ứng. Trạng thái và các lỗi có thể phát hiện bằng đọc các thanh ghi tương ứng. Tất cả các message được kiểm tra lỗi và so sánh với các filter (thanh ghi lọc) để xem nó có được nhận và chứa vào trong 1 trong 2 thanh ghi nhận.





Can hỗ trợ các kiểu frame sau:

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

Các tài nguyên trong module CAN:

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

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

1. Đảm bảo module trong chế độ thiết lập
2. Thiết lập chế độ baud
3. Thiết lập các thanh ghi lọc và mặt nạ
4. Đưa module CAN về chế độ hoạt động bình thường hay các chế độ khác tùy theo áp dụng.

10.2. Các mode hoạt động

1.Configuration mode : trong mode này, module CAN được khởi tạo trước khi hoạt động. Modul CAN không cho phép vào mode này khi có một sự truyền hay nhận đang xảy ra, nó giống như cái khóa bảo vệ các thanh ghi khi hoạt động.

2. Listen mode: Mode này dùng để quan sát trạng thái bus hay dùng để phân tích tốc độ baud trong trường hợp cắm nóng. Cho việc phân tích tốc độ Baud tự động, cần thiết phãi có 2 nút giao tiếp với nhau.

3. Loop back mode: mode này cho phép sự truyền các message từ buffer truyền sang buffer nhận mà không thực sự truyền message ra ngoài CAN bus, sử dụng phát triển và kiểm tra hệ thống. Bit ACK không được kiểm tra và thiết bị cho phép các messages từ nó như những message từ các nút khác.

4. Disabled mode: trong mode này, module không truyền hay nhận, Mode này giống như tắt module, làm cho xung clock dừng.

5. Normal mode: là mode hoạt dộng cho thiết bị. Trong mode này, thiết bị kiểm tra tất cả các message trên bus và tạo bit ACK, frame lỗi… và chỉ là mode duy nhất cho phép truyền nhận message lên bus CAN.

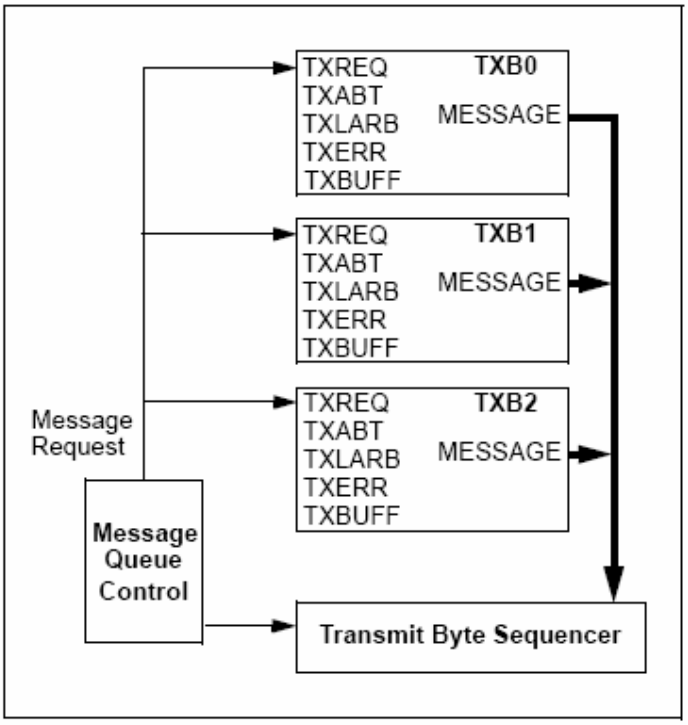
6. Error recognition mode : mode có thể thiết lập để bỏ qua tất cả các lỗi và nhận bất cứ message nào.

10.3. Truyền message CAN

a. Các buffer truyền:

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

FIGURE 19-2: TRANSMIT BUFFER BLOCK DIAGRAM



b. Thiết lập truyền:

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

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

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

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

c. Ưu tiên truyền:

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

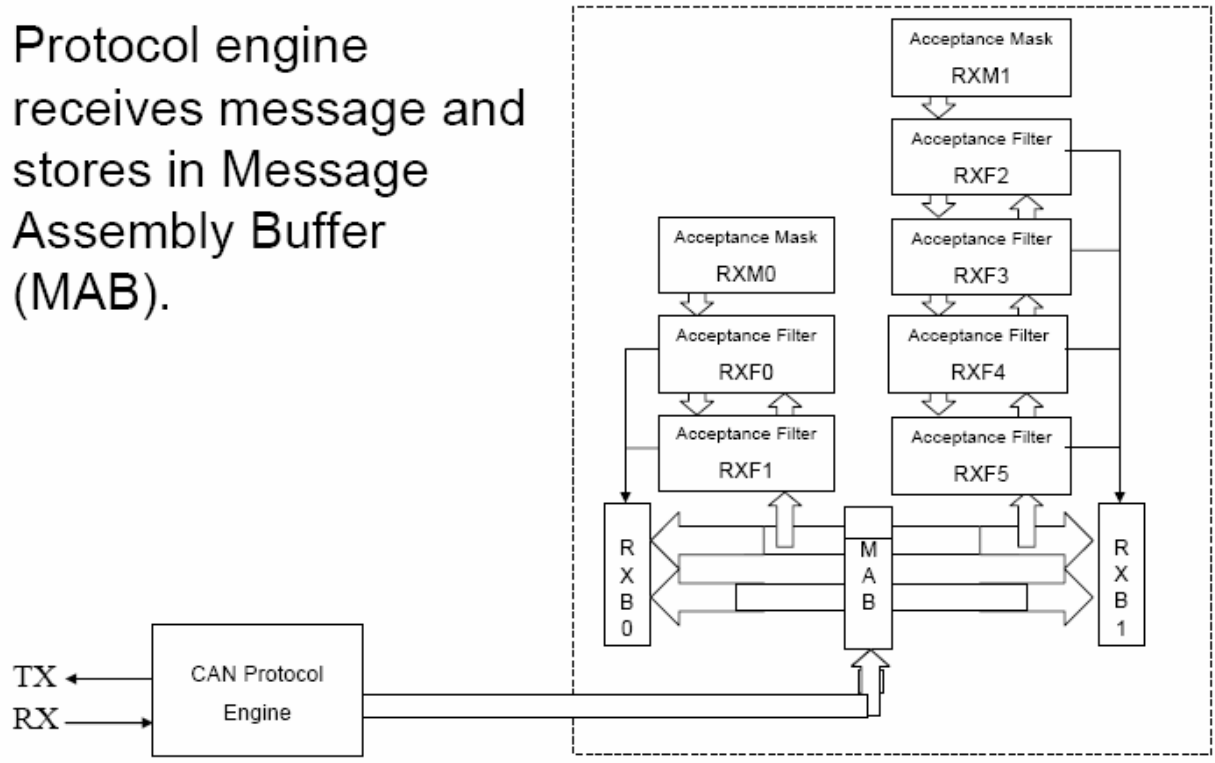
10.4. Nhận message:

a. Các buffer nhận:

Có 2 buffer nhận –RXB0 và RXB1. Mỗi buffer chiếm 14 byte SRAM và bao gồm một thanh ghi điều khiển(RXBnCON), 4 thanh ghi ID (RXBnSIDL, RXBnSIDH, RXBnEIDL, RXBnEIDH), một thanh ghi đếm độ dài dữ liệu (RXBnDLC) và 8 thanh ghi dữ liệu (RXBnDm).

Nó còn có một buffer riêng Message Assembly Buffer (MAB) có vai trò là một buffer phụ. MAB luôn nhận message kế tiếp trên bus và không thể tác động trực tiếp bởi firmware. Buffer MAB tiếp nhận lần lượt tất cả các message tới. Massage sau đó được truyền tới buffer nhận tương ứng chỉ khi nào ID của message đúng với bộ lọc.

b. Nhận một message:



Hình : Các bufer nhận

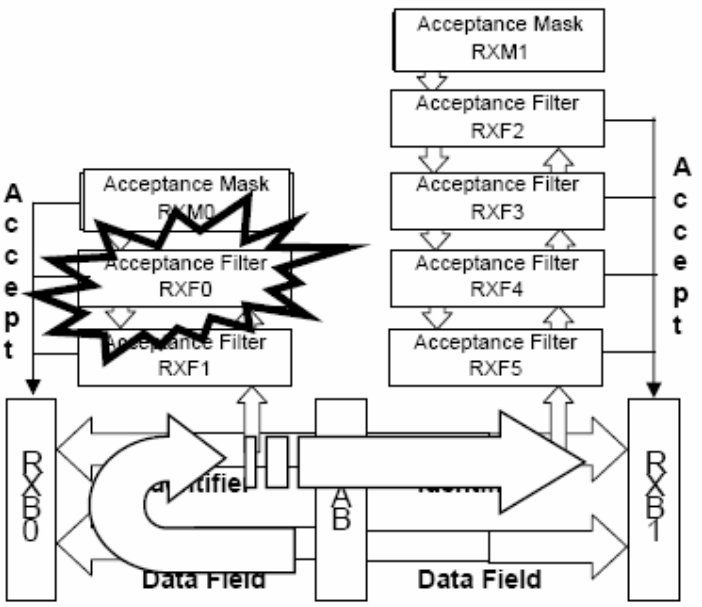
Cho tất cả các buffer, MAB (message assembly buffer) được sử dụng để nhận message kế tiếp trên bus. MCU có thể tác động một buffer trong khi buffer kia nhận message hay giữ message vừa nhận.

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

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

c. Ưu tiên nhận

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



Hình: RXB0 chứa một message hợp lệ, message khác khi được nhận sẽ được đưa vào RXB1

d. Message Acceptance Filtes and Masks.

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

TABLE 19-2: FILTER/MASK TRUTH TABLE

Mask bit n	Filter bit n	Message Identifier bit n001	Accept or Reject bit n
0	x	x	Accept
1	0	0	Accept
1	0	1	Reject
1	1	0	Reject
1	1	1	Accept

Legend: x = don't care

10.5. Baud Rate Setting:

$$\text{Nominal Bit Time} = T_Q * (\text{Sync\_Seg} + \text{Prop\_Seg} + \text{Phase\_Seg1} + \text{Phase\_Seg2})$$

Time Quantum được tính theo công thức:

$$T_Q\ (\mu s) = (2 * (BRP + 1))/F_{OSC}\ (MHz)$$

or

$$T_Q\ (\mu s) = (2 * (BRP + 1)) * T_{OSC}\ (\mu s)$$

trong đó F_{OSC} là tần số xung clock, T_{OSC} là chu kì dao động và BRP là số nguyên (từ 0 đến 63) theo giá trị các bit BRGCON1<5:0>.

11. Giới thiệu CAN trong trình dịch CCS

11.1. Các hàm hỗ trợ CAN

- void can_init(void);

Thiết lập modul CAN. Thiết lập bộ lọc RX và mask sao cho module CAN có thể nhận mọi ID tới. thiết lập ngõ ra 3 trạng thái cho pin B2 và B3 là ngõ vào.
Các thông số (CAN_USE_RX_DOUBLE_BUFFER, CAN_ENABLE_DRIVE_HIGH, CAN_ENABLE_CAN_CAPTURE) được thiết lập mặc định trong file can-18xxx8.h file.
Các giá trị mặc định này có thể thay đổi trong chương trình nhưng mọi ứng dụng đều chạy tốt với các mặc định này.

- void can_set_baud(void);

Thiết lập tốc độ baud 125kHz cho module. Tất cả thông số được định nghĩa mặc định trong file can-18xxx8.h file. Các giá trị mặc định này có thể thay đổi trong chương trình.
Hàm này được gọi trong hàm can_init() vì vậy không cần thiết phải gọi nữa

- void can_set_mode(CAN_OP_MODE mode);

Cho phép module CAN thay đổi các chế độ hoạt động:

- Configuration mode
- Listen mode
- Loop back mode.
- Disabled mode
- Normal mode.
- Error recognition mode

- void can_set_id(int* addr, int32 id, int1 ext);

Thiết lập các thanh ghi xxxxEIDL, xxxxEIDH, xxxxSIDL và xxxxSIDH để định nghĩa một ID cụ thể.
Thông số:
addr – là 1 pointer tới byte đầu tiên của thanh ghi ID, bắt đầu với xxxxEIDL.
Ví dụ :là một pointer tới RXM1EIDL
id - ID cần ghi vào
ext - Thiết lập là TRUE nếu sử dụng ID mở rộng , FALSE nếu là tiêu chuẩn

- int32 can_get_id(int * addr, int1 ext);

Trả về giá trị ID (ngược với hàm can_set_id())
Hàm này được sử dụng sau khi nhận một message, để biết được ID
Thông số:
addr - là 1 pointer tới byte đầu tiên cảu thanh ghi ID, bắt đầu với xxxxEIDL.
Ví dụ :là một pointer tới RXM1EIDL
ext - Thiết lập là TRUE nếu sử dụng ID mở rộng , FALSE nếu là tiêu chuẩn
Trả về: Giá trị ID của buffer

- int can_putd(int32 id, int * data, int len, int priority, int1 ext, int1 rtr);

Hàm này đặt dữ liệu vào bộ đệm truyền, vào lúc module CAN sẽ truyền khi bus sẵn sàng.
Thông số:
// id - ID của message

// data - pointer tới dữ liệu sẽ truyên
// len – độ dài dữ liệu truyền
// priority – mức ưu tiên của message. Số càng cao thì module CAN gửi đi càng sớm. Giá trị từ 0 tới 3.
// ext - Thiết lập là TRUE nếu sử dụng ID mở rộng , FALSE nếu là tiêu chuẩn
// rtr - TRUE để set RTR (request) bit trong ID, false nếu NOT
//
Trả về:
// Nếu thành công, tra về giá trị TRUE
// Nếu không thành công, tra về giá trị FALSE

- int1 can_getd(int32 & id, int * data, int & len, struct rx_stat & stat);

Lấy dữ liệu từ một bộ đệm nhận, nếu có data
Thông số:
// id - ID của message nhận
// data - pointer tới dãy dữ liệu
// len – độ dài của dữ liệu nhận.
// stat - structure holding some information (such as which buffer
// recieved it, ext or standard, etc)
//
Trả về:
// Hàm tra về TRUE nếu có data trong bộ đệm RX, FALSE nếu không có

- can_kbhit() (RXB0CON.rxful || RXB1CON.rxful)

Trả về giá trị TRUE nếu có data trong một trong những buffer nhận

- can_tbe() (!TXB0CON.txreq || !TXB1CON.txreq || !TXB2CON.txreq)

Trả về giá trị TRUE nếu buffer truyền sẵn sàng truyền dữ liệu

- can_abort() (CANCON.abat=1)

Bỏ tất cả những cuộc chưa truyền. Aborts all pending transmissions

11.2. Một ví dụ đơn giản:

```
////////////////////////////////////////////////////////////////////////
////                      EX_CAN.C                                  ////
////                                                                ////
//// Example of CCS's CAN library, using the PIC18Fxx8.  This       ////
//// example was tested using MCP250xxx CAN Developer's Kit.        ////
////                                                                ////
//// Connect pin B2 (CANTX) to the CANTX pin on the open NODE A of   ////
//// the developer's kit, and connect pin B3 (CANRX) to the CANRX    ////
//// pin on the open NODE A.                                        ////
////                                                                ////
//// NODE B has an MCP250xxx which sends and responds certan canned  ////
//// messages.  For example, hitting one of the GPX buttons on      ////
//// the development kit causes the MCP250xxx to send a 2 byte      ////
//// message with an ID of 0x290.  After pressing one of those      ////
//// buttons with this firmware you should see this message         ////
//// displayed over RS232.                                          ////
////                                                                ////
//// NODE B also responds to certain CAN messages.  If you send     ////
//// a request (RTR bit set) with an ID of 0x18 then NODE B will    ////
```

```
//// respond with an 8-byte message containing certain readings.     ////
//// This firmware sends this request every 2 seconds, which NODE B  ////
//// responds.                                                        ////
////                                                                  ////
//// If you install Microchip's CANKing software and use the         ////
//// MCP250xxx , you can see all the CAN traffic and validate all     ////
//// experiments.                                                     ////
////                                                                  ////
//// For more documentation on the CCS CAN library, see can-18xxx8.c ////
////                                                                  ////
////  Jumpers:                                                        ////
////     PCM,PCH    pin C7 to RS232 RX, pin C6 to RS232 TX           ////
////                                                                  ////
////  This example will work with the PCM and PCH compilers.          ////
//////////////////////////////////////////////////////////////////////////
////        (C) Copyright 1996,2003 Custom Computer Services         ////
//// This source code may only be used by licensed users of the CCS  ////
//// C compiler.  This source code may only be distributed to other  ////
//// licensed users of the CCS C compiler.  No other use,            ////
//// reproduction or distribution is permitted without written       ////
//// permission.  Derivative programs created using this software    ////
//// in object code form are not restricted in any way.              ////
//////////////////////////////////////////////////////////////////////////

#include <18F458.h>
#fuses HS,NOPROTECT,NOLVP,NOWDT
#use delay(clock=20000000)
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7)

#include <can-18xxx8.c>

int16 ms;

#int_timer2
void isr_timer2(void) {
  ms++; //keep a running timer that increments every milli-second
}

void main() {
  struct rx_stat rxstat;
  int32 rx_id;
  int in_data[8];
  int rx_len;

//send a request (tx_rtr=1) for 8 bytes of data (tx_len=8) from id 125 (tx_id=125)
  int out_data[8];
  int32 tx_id=125;
  int1 tx_rtr=1;
  int1 tx_ext=0;
  int tx_len=8;
  int tx_pri=3;
```

```
  int i;

  for (i=0;i<8;i++) {
    out_data[i]=0;
    in_data[i]=0;
  }

  printf("\r\n\r\nCCS CAN EXAMPLE\r\n");

  setup_timer_2(T2_DIV_BY_4,79,16);   //setup up timer2 to interrupt every 1ms if using 20Mhz
clock

  can_init();

  enable_interrupts(INT_TIMER2);   //enable timer2 interrupt
  enable_interrupts(GLOBAL);       //enable all interrupts (else timer2 wont happen)

  printf("\r\nRunning...");

  while(TRUE)
  {

    if ( can_kbhit() )   //if data is waiting in buffer...
    {
      if(can_getd(rx_id, &in_data[0], rx_len, rxstat)) { //...then get data from buffer
        printf("\r\nGOT: BUFF=%U ID=%LU LEN=%U OVF=%U ", rxstat.buffer, rx_id, rx_len,
rxstat.err_ovfl);
        printf("FILT=%U  RTR=%U  EXT=%U  INV=%U", rxstat.filthit, rxstat.rtr, rxstat.ext,
rxstat.inv);
        printf("\r\n    DATA = ");
        for (i=0;i<rx_len;i++) {
          printf("%X ",in_data[i]);
        }
        printf("\r\n");
      }
      else {
        printf("\r\nFAIL on GETD\r\n");
      }

    }

    //every two seconds, send new data if transmit buffer is empty
    if ( can_tbe() && (ms > 2000))
    {
      ms=0;
      i=can_putd(tx_id, out_data, tx_len,tx_pri,tx_ext,tx_rtr); //put data on transmit buffer
      if (i != 0xFF) { //success, a transmit buffer was open
        printf("\r\nPUT %U: ID=%LU LEN=%U ", i, tx_id, tx_len);
        printf("PRI=%U EXT=%U RTR=%U\r\n   DATA = ", tx_pri, tx_ext, tx_rtr);
        for (i=0;i<tx_len;i++) {
```

```
            printf("%X ",out_data[i]);
          }
          printf("\r\n");
        }
        else { //fail, no transmit buffer was open
          printf("\r\nFAIL on PUTD\r\n");
        }
      }
    }
}
```