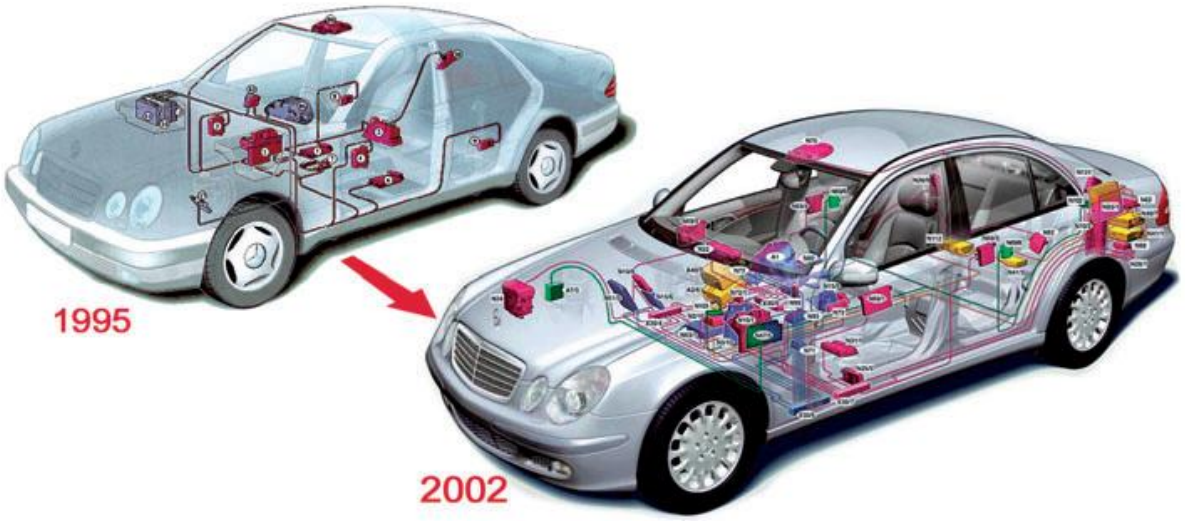
CAN-BUS PROTOCOL THEORIE

ALGEMEEN

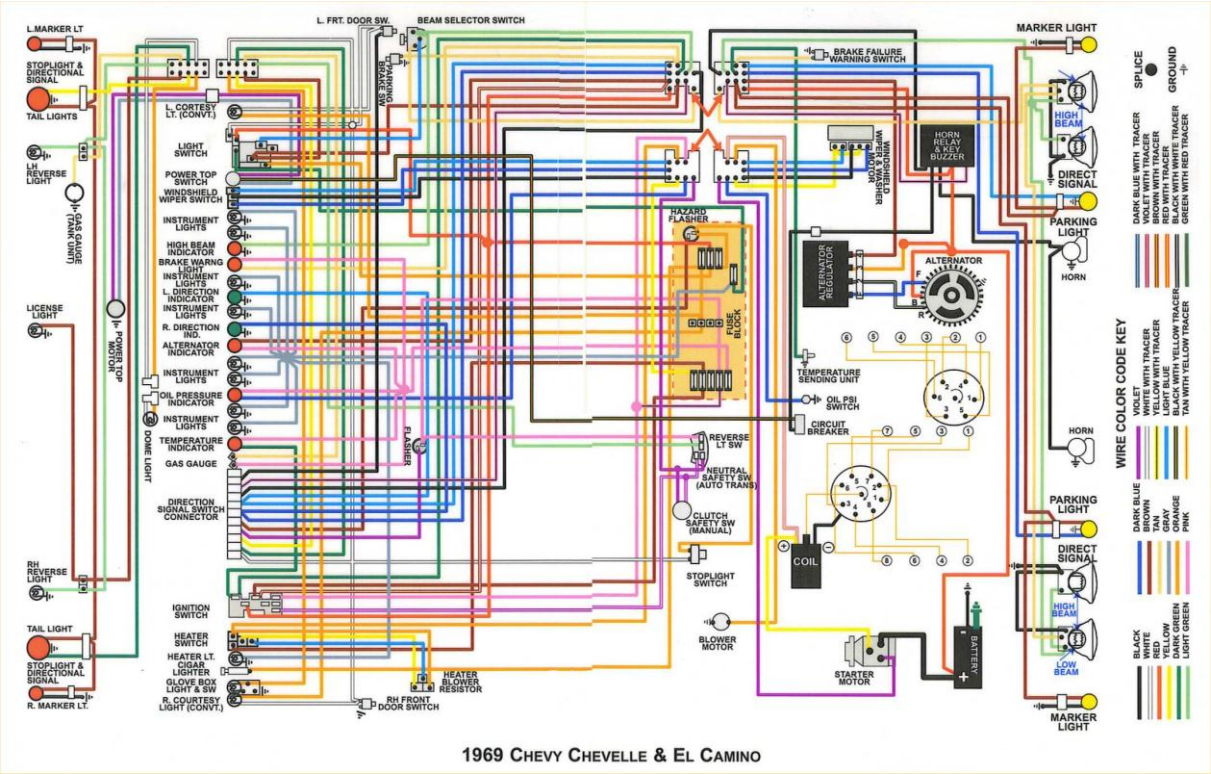


CAN BUS staat voor "Controller Area Network". Dit bussysteem is in 1986 door BOSCH speciaal ontworpen voor gebruik in wagens, vrachtwagens en bussen en later ook voor landbouwvoertuigen. CAN is ondertussen door de meeste grote automerken als algemene standaard aangenomen. CAN BUS is echter mede door z'n grote storings-ongevoeligheid en relatief goedkope componenten ook populair geworden als industrieel bussysteem.

EIGENSCHAPPEN:

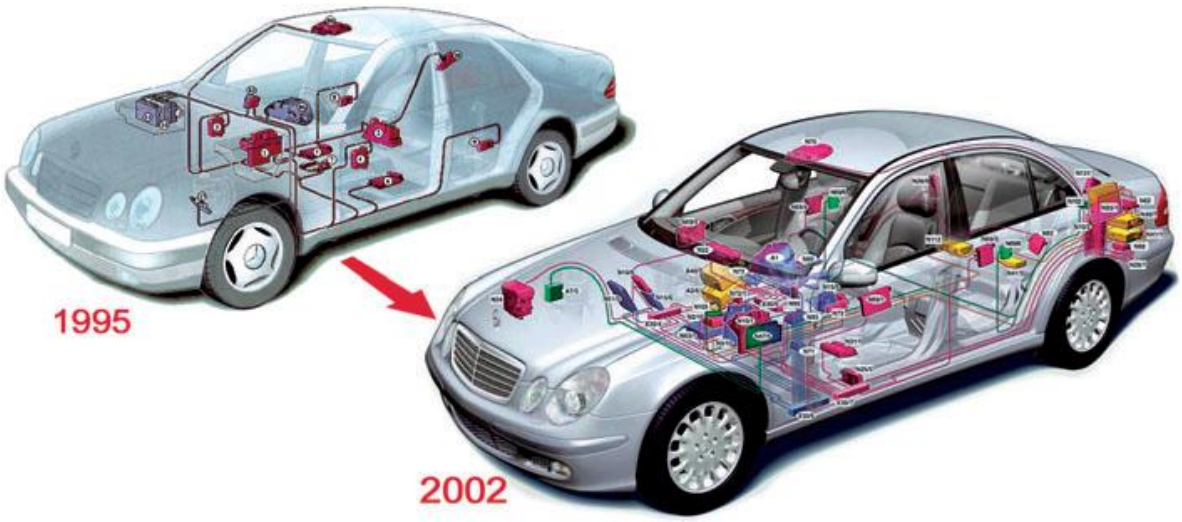
- CAN: Controller Area Network
- Differentiëel signaal
- NRZ (Non return to zero) + Bit stuffing
- 2 draads verbinding (STP, UTP)
- Multi master – iedereen op de bus is de baas…
- Speed: Max 1Mbps (40m) – 50kbps (1km) - Can bus E-block = standaard 125kbps (300m)
- Datalengte: 0-8 bytes
- 16 bit CRC check wat er samen met andere foutdetectiesystemen voor zorgt dat deze bus extreem bedrijfszeker is.
- Geen klassieke adressering maar wel een "Identifier" van 11 bits of 29 bits bij Extended CAN

WAAROM CAN BUS IN AUTOMOTIVE

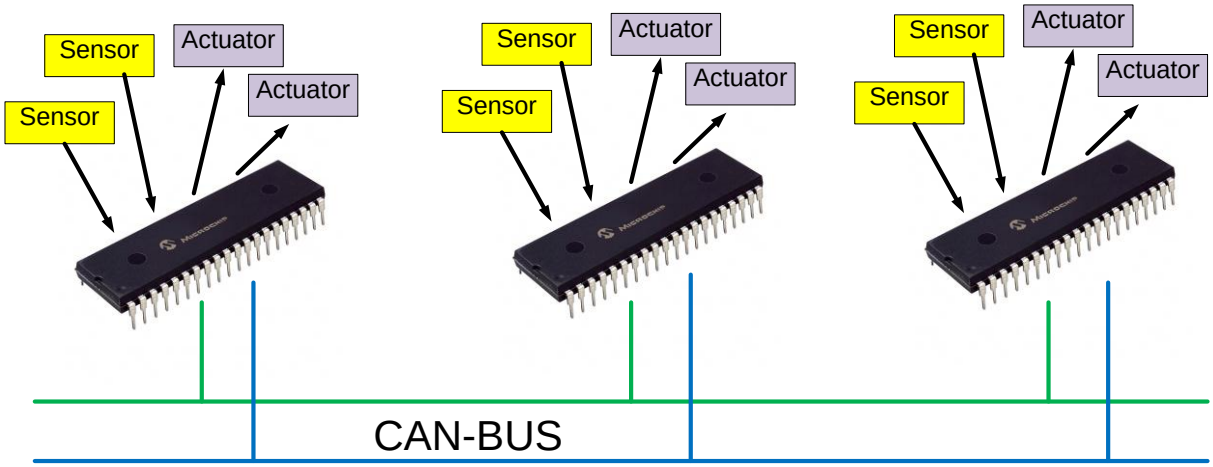


1969 CHEVY CHEVELLE & EL CAMINO

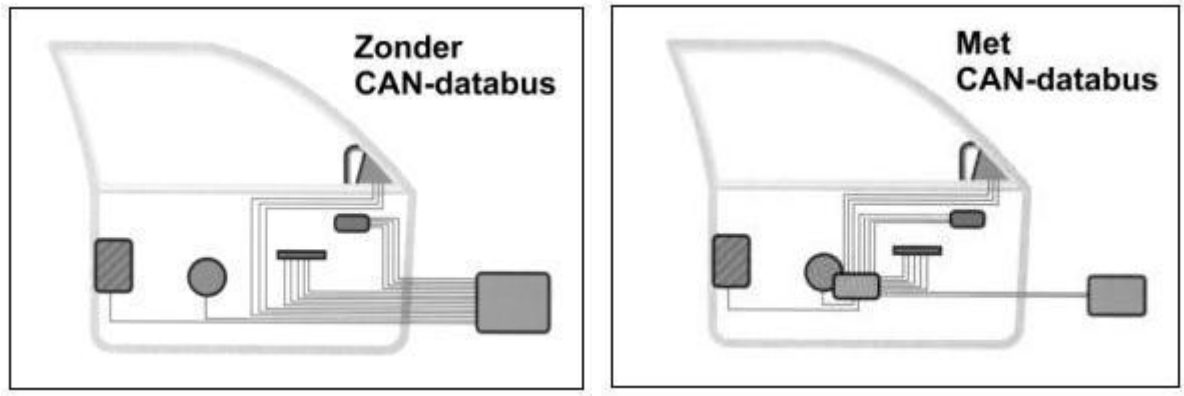
Een moderne wagen heeft enorm veel contacten – schakelaars – lampjes en sensoren die allemaal op één of andere manier met elkaar en met de centrale boordcomputer moeten verbonden worden. Dat zorgt voor een niet te onderschatten gewicht aan koperdraad welke mee het verbruik bepaalt en dus voor vervuiling zorgt. Ook de prijs van het koper is mee bepalend. De stijgende vraag naar steeds meer comfort en veiligheid in een wagen zouden dit gewicht en ook de prijs aan koperdraad alleen nog maar laten stijgen. U ziet hier het bedradingsschema van een wagen uit 1970. Kan u zich al voorstellen hoe een gelijkaardig schema voor een moderne wagens met zo veel meer comfort en motor management er uit zou zien?



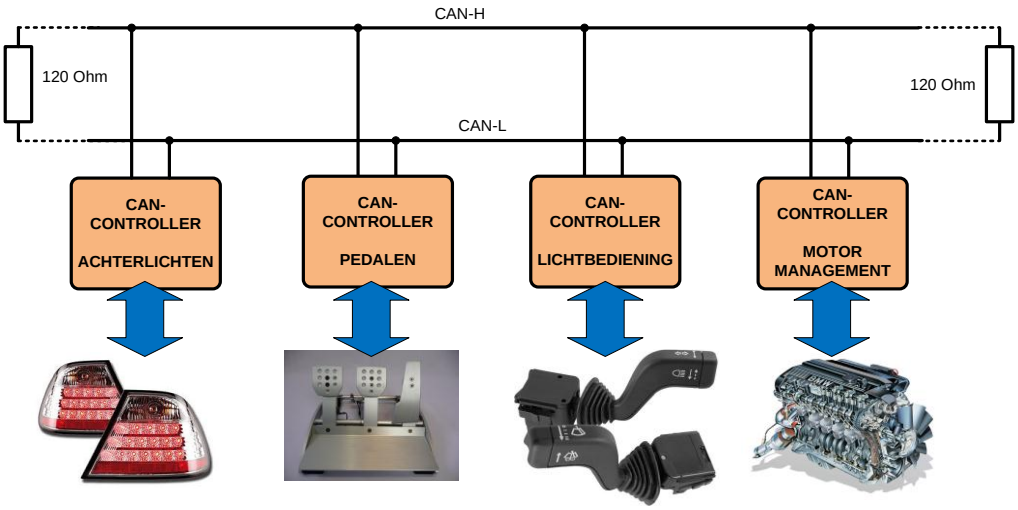
Automerken ontwikkelden daarom eerst elk een eigen bussysteem, maar uiteindelijk is het CAN bus systeem van BOSCH overgebleven.



Bij een CAN bus systeem worden er enkele tientallen microcontrollers over de verschillende delen van de wagen verdeeld. Elke actuator en sensor wordt met korte draden met de dichtstbijzijnde microcontroller verbonden. Alle microcontrollers worden via een 2 draads CAN bus systeem met elkaar verbonden en kunnen zo de noodzakelijke gegevens uitwisselen. Dit can bus netwerk heeft slechts 2 draden nodig en zo besparen we heel wat honderden meters draad en dus gewicht en kostprijs.

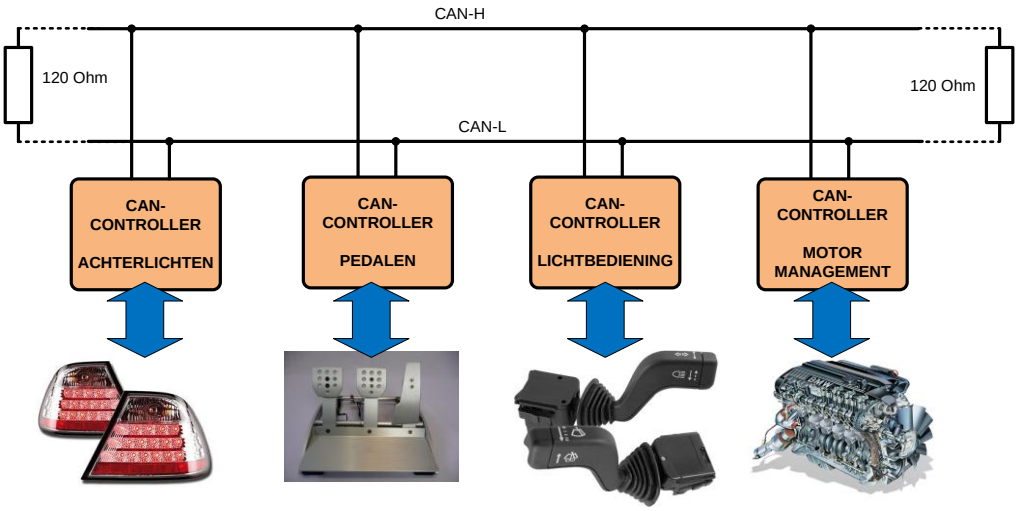


Bekijk als voorbeeld deze deur. Zonder een busnetwerk zouden alle sensoren, knoppen en actuatoren van deze deur met lange dikke koperdraden verbonden moeten worden met de centrale boordcomputer. Met een bus-systeem worden al deze sensoren en actuatoren enkel nog verbonden met de lokale microcontroller in de deur zelf. Deze microcontroller staat via een tweedraads CAN bus in communicatie met alle andere microcontrollers in de wagen. De functionaliteit blijft dezelfde – het gewicht aan koperdraad is wel gereduceerd.



De verschillende microcontrollers of nodes in eenzelfde can bus netwerk takken éénder waar op de twee CAN draden af. Er dient wel op gelet te worden dat de twee uiteinden afgesloten worden met weerstanden van 120 Ohm om reflecties in de kabels te vermijden.

IDENTIFIER



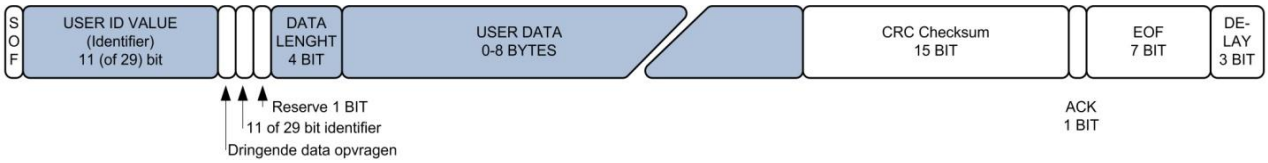
Het CAN systeem werkt niet met adressering. De verschillende nodes in het systeem hebben geen uniek adres zoals we dat kennen van I2C of IP adressen. Wat er wel wordt gedaan is dat elk bericht een unieke 11 of 29 bit identifier mee krijgt. (Bij Trucks en landbouwvoertuigen wordt er in de 29 bit identifier echter wel een soort bron een doeladres meegestuurd) Het bericht dat van de sensor onder de rempedaal komt krijgt zo een unieke identifier mee zodat elke node op de bus weet dat dit bericht van de rempedaal komt. Elke node beslist zelf of dit bericht met deze identifier nut heeft voor deze node. Een bericht dat van de rempedaal komt zal zo onder andere worden opgepikt door de microcontrollers van de achterlichten, van het derde stoplicht en van het ABS remsysteem. De microcontroller voor het motor management heeft op het eerste zicht niets met dit bericht te maken, maar toch haalt deze ook dit bericht binnen zodat de bestuurder niet gas kan geven en remmen tegelijk. Dat zou voor oververhitting van de remmen en ook voor vervuiling zorgen. Een deurmodule bijvoorbeeld zal dit bericht niet binnen pakken.

Zo'n identifier kon oorspronkelijk bestaan uit 11 bits wat betekent dat er binnen één can bus systeem maximaal 2048 verschillende identifiers mogelijk waren. Voor vrachtwagens en landbouwvoertuigen wilde men toch werken met een bron en een doeladres om alle gegevens tussen de trekker en de opligger op een uniforme manier te kunnen communiceren. Daarom heeft men de extended CAN bus standaard ontworpen die 29 bit identifiers toelaat wat betekent dat er 536.870.912 verschillende identifiers mogelijk zijn binnen dit can bus systeem.

De identifier bepaalt ook de arbitrage als er twee nodes gelijktijdig data op de bus willen zetten. Vermits bij CAN bus de 0 het dominante niveau is zal het bericht met de numeriek laagste identifier de bus kunnen claimen, de andere node moet het op een later tijdstip dan nog eens terug proberen.

OPBOUW CAN BUS BERICHT

Hieronder zien we de principiële opbouw van een CAN bus bericht. De reële opbouw verschilt licht met de 11 of 29 bit identifier en enkele verschoven datablokjes, maar het principe blijft hetzelfde.



SOF : Een CAN bericht start met een START OF FRAME bit

Identifier : De identifier bepaalt wat voor een bericht het is. Alle can nodes kunnen dit bericht lezen, maar enkel de nodes die interesse hebben in een bericht met deze identifier zullen dit bericht ook binnen-halen. Deze identifier bestaat standaard uit 11 bits, en in Extended CAN bestaat deze uit 29 bits.

3 Config bits : Met de eerste RTR-bit (remote transmission request) bit kan je aangeven of het een data- of een remote frame is om dringend bepaalde data op te vragen, de tweede bit geeft aan of het bericht met de standaard 11 bit of de extended 29 bit identifier werkt. De derde bit heeft tot op heden nog geen functie gekregen.

Data Length : Deze 4 bits geven aan uit hoeveel dataBYTES dit can bericht zal bestaan. 0b0000 betekent 0 databytes en 0b1000 betekent 8 databytes. Een Can bericht kan bestaan uit 0 tot 8 databytes – 1 byte is 8 bits!

User data : Hier komt de data te staan — minimaal 0 en maximaal 8 blokken van 8 bits.

CRC : Een 16 bit Cyclic redundancy check verzekert een zeer hoge mate van foutdetectie.

ACK : Alle busdeelnemers zenden een ACK bit om aan te geven dat de data goed op de bus is gezet.

EOF : Een 7 bit End of Frame geeft het einde van een CAN bericht aan

Delay : Na elk CAN bericht wordt er minimaal 3 bits gewacht alvorens het volgende bericht gestuurd wordt.

ERROR DETECTIE

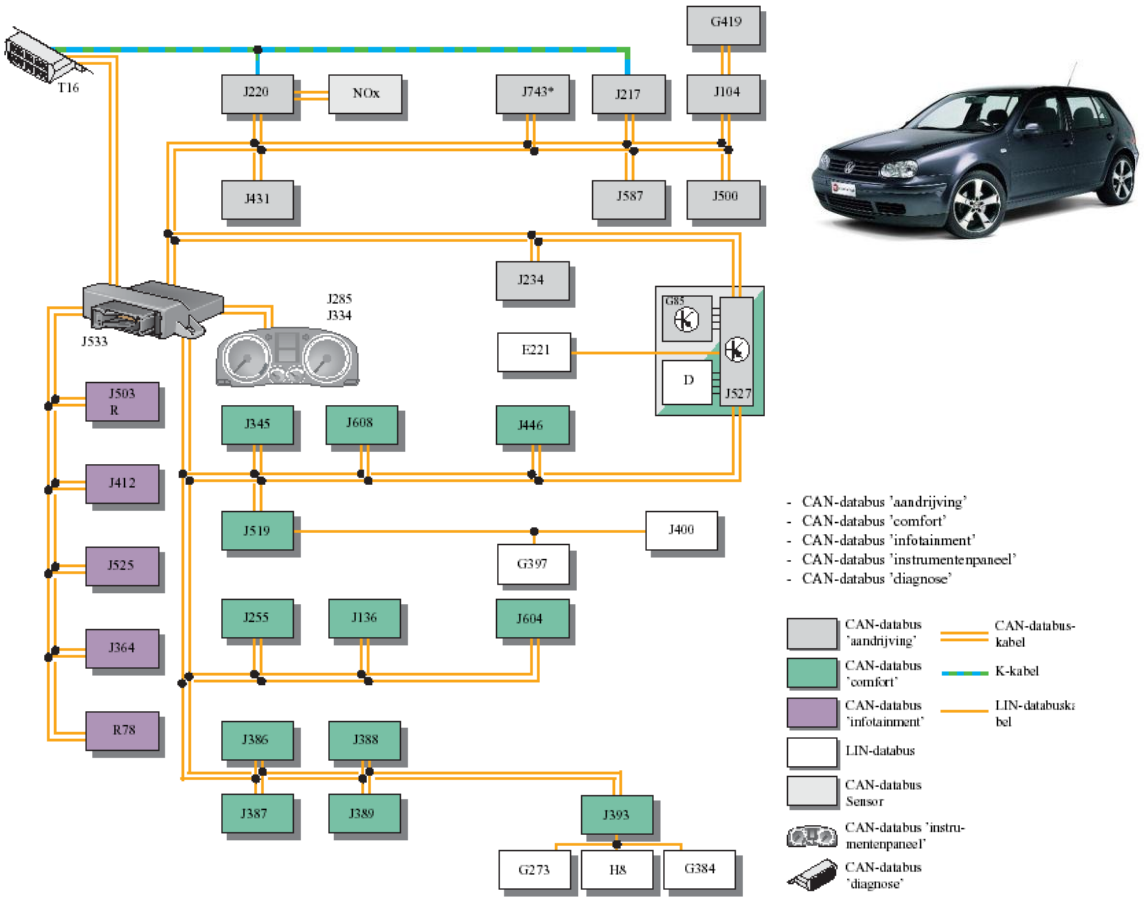
Een bussysteem dat gebruikt mag worden in wagens moet echter een extreem hoge bedrijfszekerheid hebben. Mogelijke fouten in de dataoverdracht van bijvoorbeeld de rempedaal naar het ABS systeem zouden onnoemelijke schadeclaims als gevolg kunnen hebben.

CAN-BUS heeft zo verschillende controlesystemen die er samen voor zorgen dat CAN bus één van de meest bedrijfszekere bussystemen ter wereld is.

De foutcontrole bij CAN bus gebeurt op twee niveau's:

- Op bitniveau zijn er twee controles:
    - **Bit monitoring**: Er wordt steeds door de ontvanger van de zendende module gecontroleerd of het bitniveau op de bus wel gelijk is aan het bitniveau dat uitgezonden wordt. Zo controleert de zender meteen z'n eigen signaal.
    - **Bit stuffing**: Bij CAN wordt er bitstuffing toegepast. Elke 5e opeenvolgende identieke bit wordt automatisch vervangen door z'n tegengestelde waarde. Aan de ontvangende zijde wordt er ook gecontroleerd of er niet meer dan 5 dezelfde bits elkaar opvolgen, hetgeen in een stuffbit error resulteert. Errorframes zijn een uitzonderling op deze regel, dit zijn wel 6 dominante bits na elkaar.
- Op berichtniveau zijn er drie controles:
    - **CRC-check**: Hiermee wordt er gecontroleerd of er tijdens de overdracht geen bits door storingen veranderd zijn. CAN bus is ontworpen met een 15 bit CRC. Deze CRC detecteert tot zelfs 5 random bit errors in het CAN bericht. Dit wordt uitgedrukt in een 'Hamming distance' van 6
    - **Frame check**: Hier gaat men kijken of de lengte van het frame klopt. Ook is er een controle van de waarde van bepaalde bits zoals de 'end of frame'.
    - **Acknowledge check**: Door middel van controle gaat men kijken of tenminste één deelnemer het bericht heeft ontvangen.

VERSCHILLENDE CAN NETWERKEN BINNEN 1 WAGEN



Can bus systeem in een VW GOLF 4

Om overbelasting van het CAN bus netwerk te voorkomen bestaan er in moderne wagens zelfs meerdere CAN bus netwerken. Meestal zijn die opgedeeld in 3 typische CAN netwerken:

- **CAN-driveline**: snel en zeer stabiel CAN netwerk dat de communicatie tussen de meest kritische modules regelt: de motorregeling, ABS, Airbag, 4x4, Servostuur …

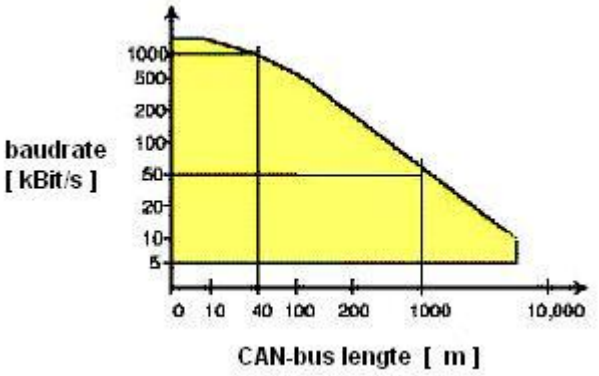
- **CAN-Comfort**: verbindt minder kritische comfort-apparaten zoals deurvergrendeling, automatische ramen, parkeerhulp, verwarming, airco, en nog andere.

- **Can-Infotainment**: verbindt de media-modules zoals navigatie, CD, radio, telefoon, versterker, … Let wel: CAN wordt enkel gebruikt om deze apparaten aan te sturen, CAN is niet ontworpen om audio en video te streamen.

SNELHEID CAN BUS

De maximale snelheid die haalbaar is hangt af van de lengte van de bus. Bij buslengtes van minder dan 40 meter kan men in principe 1Mbps halen. Bij zeer lange bussen – tot 6 km zelfs kan men maximaal snelheden tot 10 kbps behalen. Bij personenwagens wordt heel vaak 500 kbit/s gebruikt. Bij vrachtwagens gebruikt men 250kbit/s.

Baudrate (kBit/s)	Maximale Bus Lengte (m)
1000	40
500	100
250	200
125	500
62.5	1000
10	6000

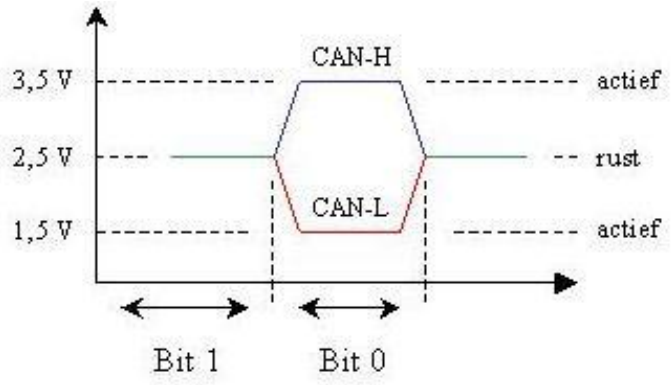


SIGNAALNIVEAU'S

In de Fysieke laag van CAN bus zijn er twee soorten CAN signalen gedefinieerd,

CAN HIGH SPEED OF HSCAN

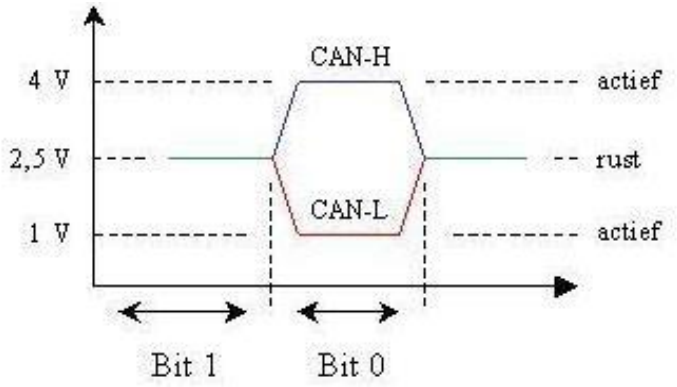
Signal	recessive state (rust)			dominant state (actief)			unit
	min	nominaal	max	min	nominaal	max	
CAN-High	2.0	2.5	3.0	2.75	3.5	4.5	Volt
CAN-Low	2.0	2.5	3.0	0.5	1.5	2.25	Volt



Bij deze High speed CAN bus is het zeer belangrijk dat beide uiteinden van de bus worden afgesloten met 120 weerstanden om reflecties op deze snelle bus te voorkomen.

CAN LOW SPEED OF FTCAN

Signal	recessive state (rust)			dominant state (actief)			unit
	min	nominaal	max	min	nominaal	max	
CAN-High	1.6	1.75	1.9	3.85	4.0	5.0	Volt
CAN-Low	3.1	3.25	3.4	0	1.0	1.15	Volt



Het fysisch CAN protocol dat zonder afsluitweerstanden werkt heet FTCAN (fault tolerant CAN).

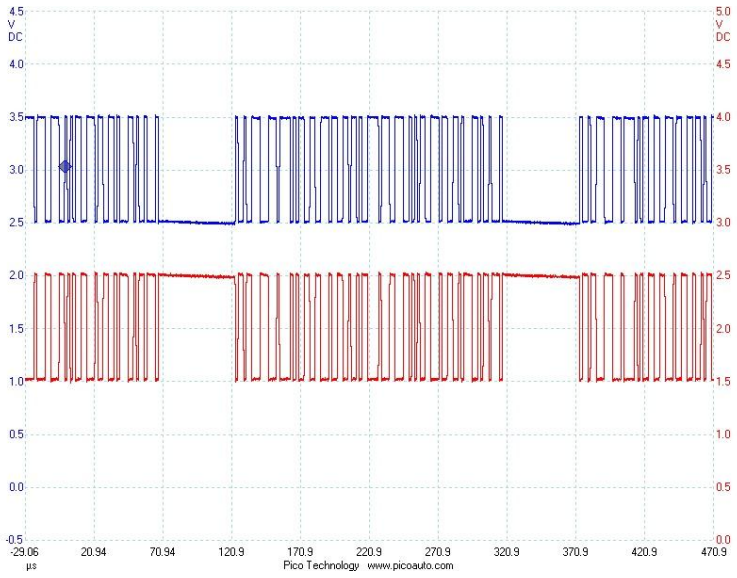
4 verschillen in vergelijking met het High Speed CAN protocol.

- Heeft geen afsluitweerstanden
- De max baudrate is 125 kbps
- Het is fout tolerant (kan overschakelen naar 1 draadsmodus als één van de 2 CAN draden een defect vertoont)
- Spanningsniveaus CANH 0-4 V, CANL 5-1 V

FTCAN vindt men bij een beperkt aantal merken (bvb Peugeot, Citroen,…..) voor functies zoals lichten, ruitbediening,….. Sommige merken gebruiken zelfs SWCAN (single wire CAN). FTCAN en SWCAN zullen binnen de komende jaren echter opnieuw verdwijnen en vervangen worden door HSCAN

SCOOPBEELD CAN HIGH SPEED

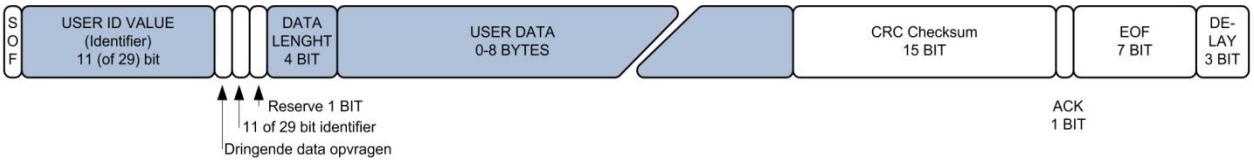
Dit is een scoopbeeld van de twee CAN High speed signalen. De spanningsniveau's zijn duidelijk, maar ook de bitlengte die we hier snel schatten op minder dan 5 usec/bit resulteert in een snelheid van meer dan 200Kbps. High speed can dus!



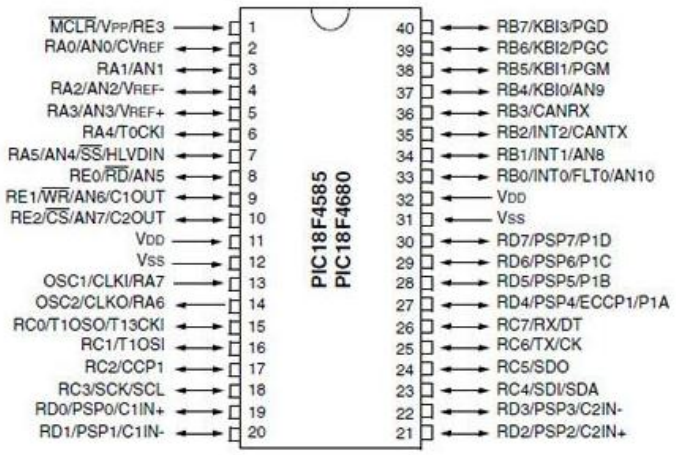
CAN BUS PRAKTISCH

CAN STACK IC

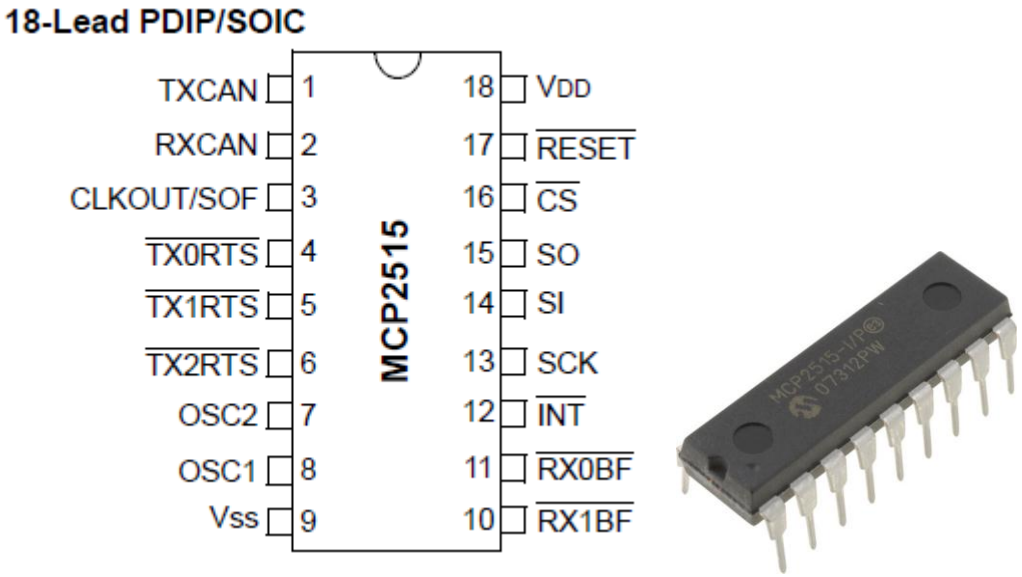
Het CAN bericht is, in tegenstelling tot RS232, I2C en SPI, veel langer dan de gebruikelijke 8 bits die we met onze 8 bit microcontroller rechtsreeks kunnen verwerken.



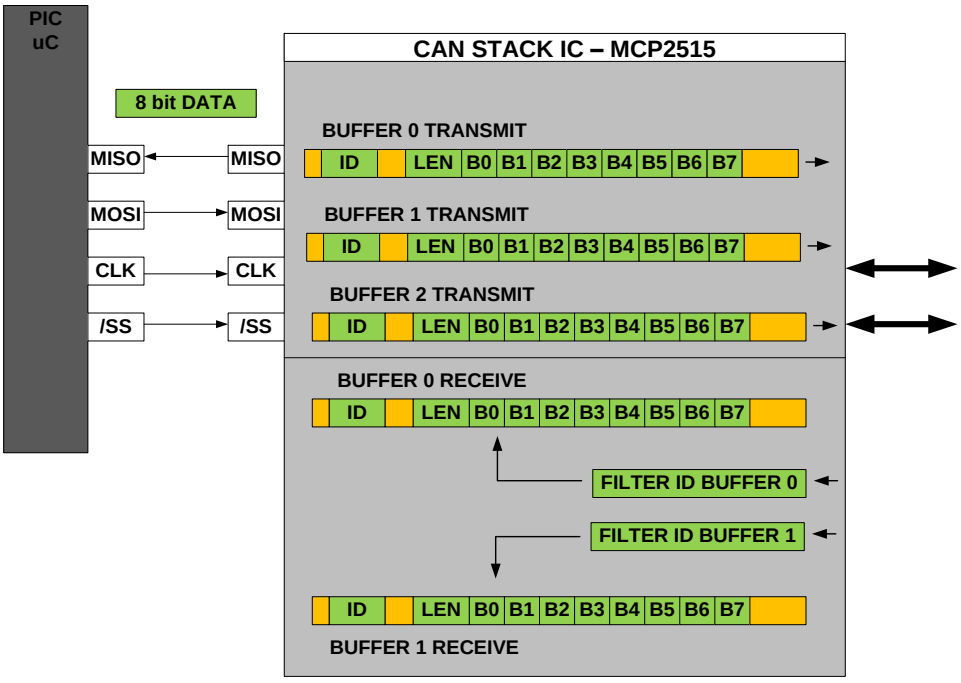
Voordat we dit bericht, dat uit vele tientallen bits bestaat, kunnen verzenden, moeten we dit eerst ergens – byte voor byte - opbouwen.



Hiervoor bestaan er enkele microcontrollers die een CAN stack hardware aan boord hebben. Zoals bij RS232 en I2C zorgt een hardware module in de uC dan dat het CAN bericht correct wordt opgebouwd, verzonden en ontvangen met alle complexe error detecties inclusief, maar microcontrollers met CAN bus module zijn niet in veel varianten beschikbaar en zijn niet goedkoop, al kan er hier wel snel verandering in komen.



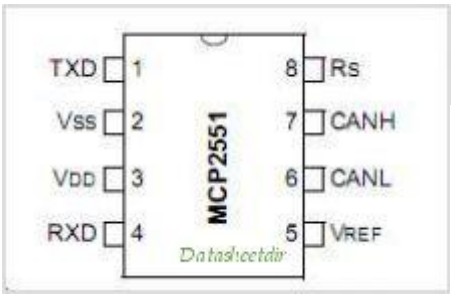
Onze voorkeur gaat uit naar de MCP2515. Dit is een 18 pins CAN STACK IC van Microchip die speciaal ontworpen is om deze taak op zich te nemen.

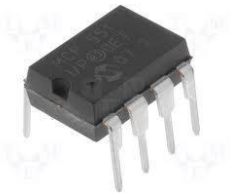


Onze microcontroller moet via SPI enkel de Identifier, de lengte en de data – in blokjes van 8 bits aanleveren aan deze CAN stack IC. Deze STACK IC zet al deze blokjes op de correcte plaats in het CAN bericht, berekent zelf de CRC, voegt start en stopbits toe en doet al het nodige om een bericht correct te verzenden. Deze CAN STACK heeft zelfs 3 verschillende zend buffers zodat we veel voorkomende berichten niet telkens terug helemaal moeten opbouwen.

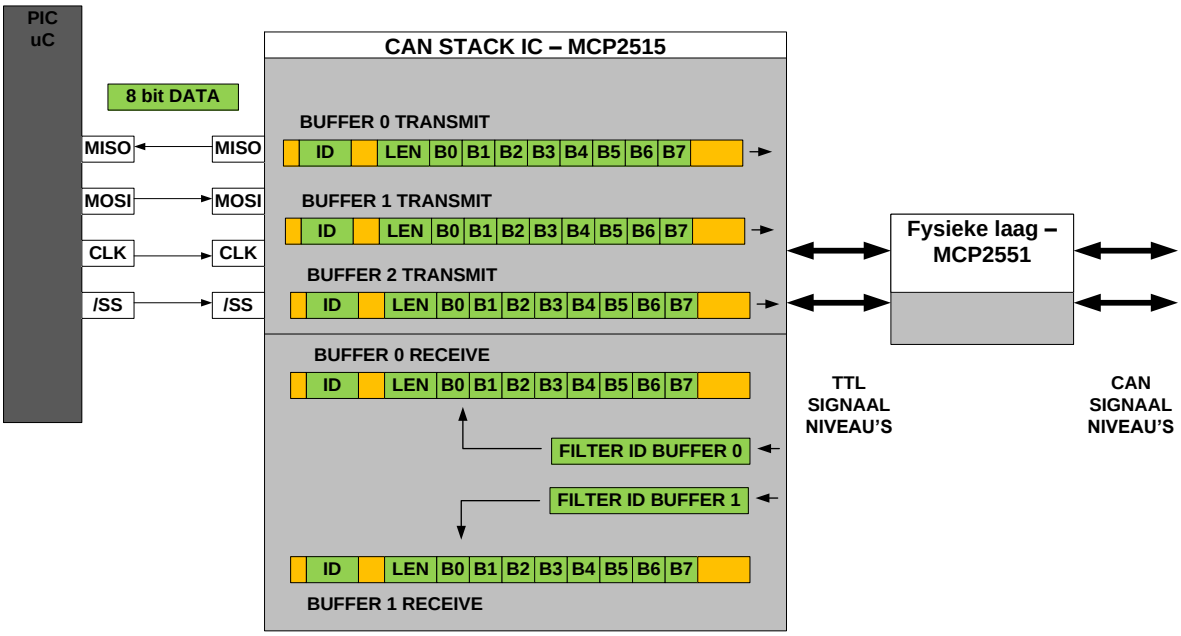
Ook het ontvangen van data wordt volledig verzorgd door deze CAN STACK IC – inclusief de complexe error controle. Er zijn twee verschillende ontvangstbuffers en instelbare filters die afhankelijk van de identifier het can bericht zullen doorlaten naar één van de twee receive buffers of niet . De uC krijgt een bericht wanneer nieuwe data ontvangen is zodat deze de data byte per byte via SPI kan binnenhalen.

FYSIEKE LAAG IC

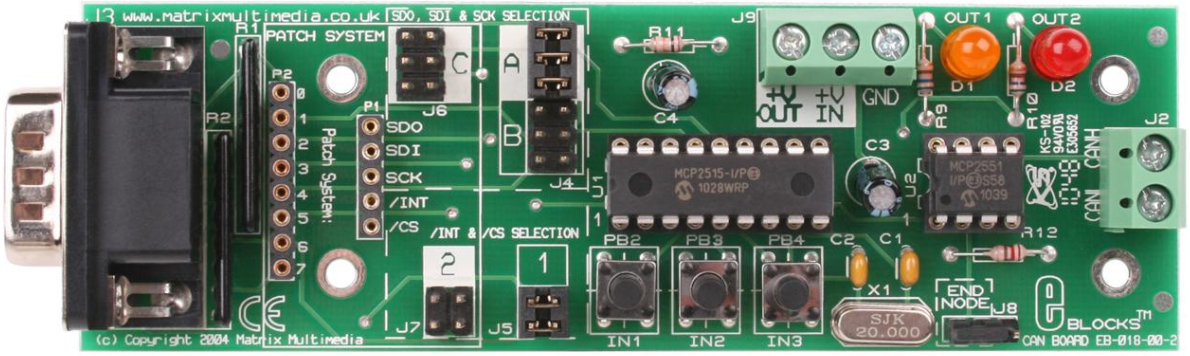


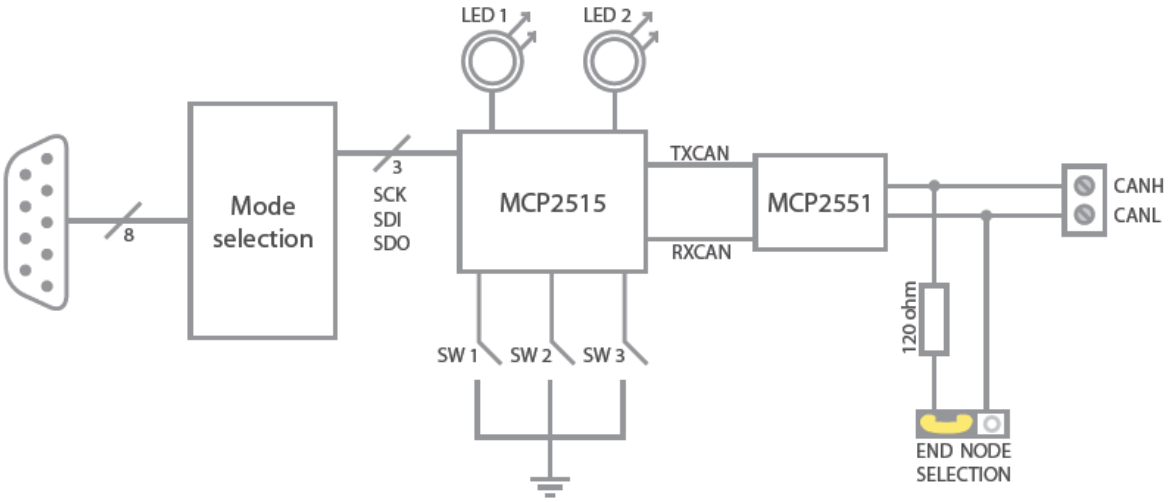


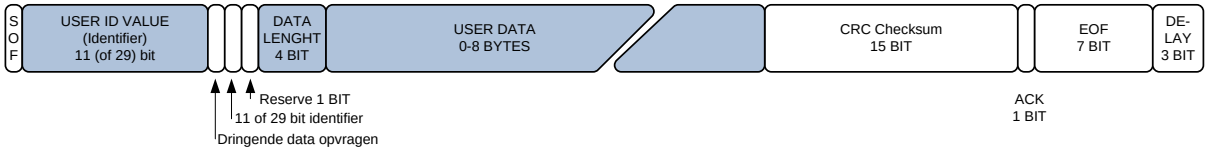
De MCP2551 is een 8 pins Fysieke Laag IC die de TTL signalen van de CAN STACK IC omvormt naar echte differentiële CAN bus niveau's en omgekeerd.



Matrixmultimedia heeft een CAN bus E-blocks waarop zowel de MCP2515 stack IC als de MCP2551 fysieke laag IC reeds correct aangesloten staan. Je kan dit bordje zeer eenvoudig gebruiken als link tussen eender welke uC en de CAN bus. Met een jumper bepaal je of een 120 Ohm weerstand de bus moet afsluiten.

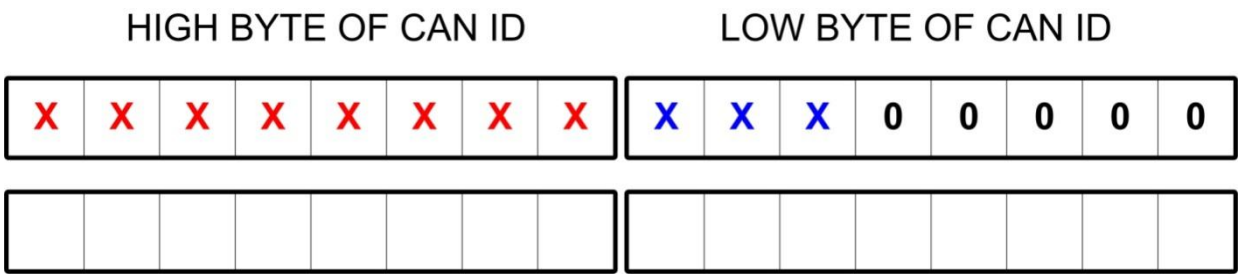






Zoals we reeds gezien hebben is zo'n CAN bus bericht samengesteld uit verschillende delen. Wij moeten in ons programma enkel de donkere delen doorsturen naar de MCP2515 can stack. Vanaf dat moment neemt die het over. Wat moeten we nu allemaal doorsturen :

Als eerste de 11 bit identifier:

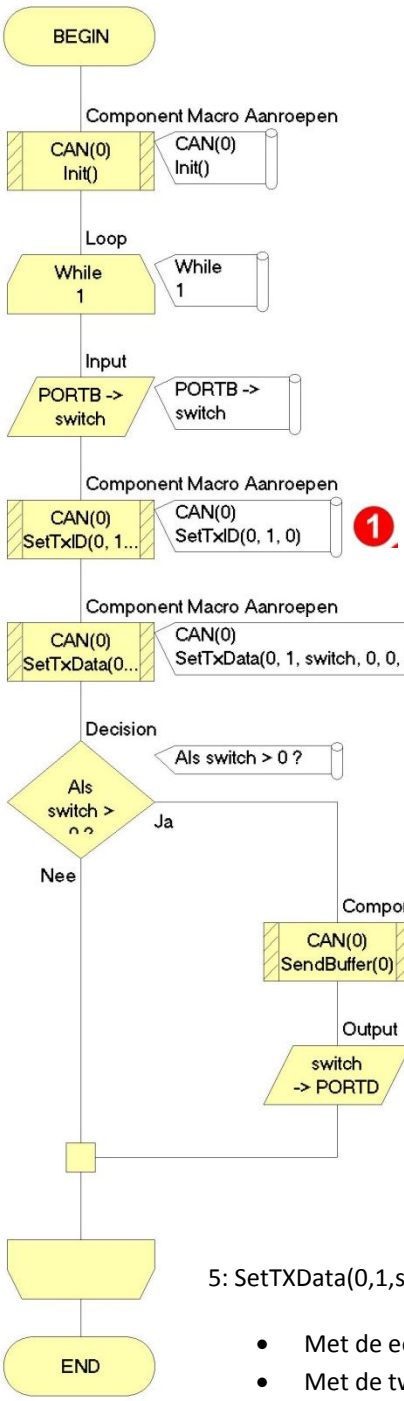


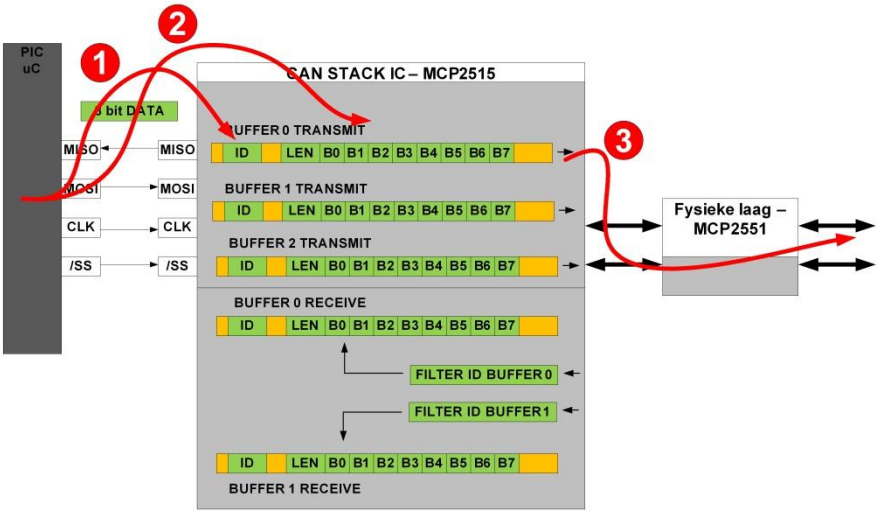
Vermits dit een 11 bit waarde is moeten we die in 2 bytes doorsturen. Een High en een Low byte. Let er op dat de hoogste 3 bits van de low byte de 3 LSB's zijn van de identifier. Stel dat we als identifier "9" hebben, dan zal de High byte 0b00000001 bevatten en de low byte zal 0b00100000 moeten bevatten.

Vervolgens sturen we de data lengte door. Met een waarde tussen 0 en 8 geeft je aan hoeveel bytes data dit bericht zal bevatten

Tot slot het bericht – dit kan bestaan uit minimaal 0 tot maximaal 8 bytes. Lege bytes vullen we met een 0.

ZENDEN VAN EEN CAN BERICHT





<u>Uitleg programma CAN SEND</u>

1: Om CAN berichten op deze manier te verzenden moeten we aan de externe eigenschappen van de CAN component niets veranderen.

2: CAN INIT: We initialiseren de CAN component in dit programma.

3: PORTB -> Switch: Lees de waarde van de schakelaars aan PORTB in in de variabele "switch"

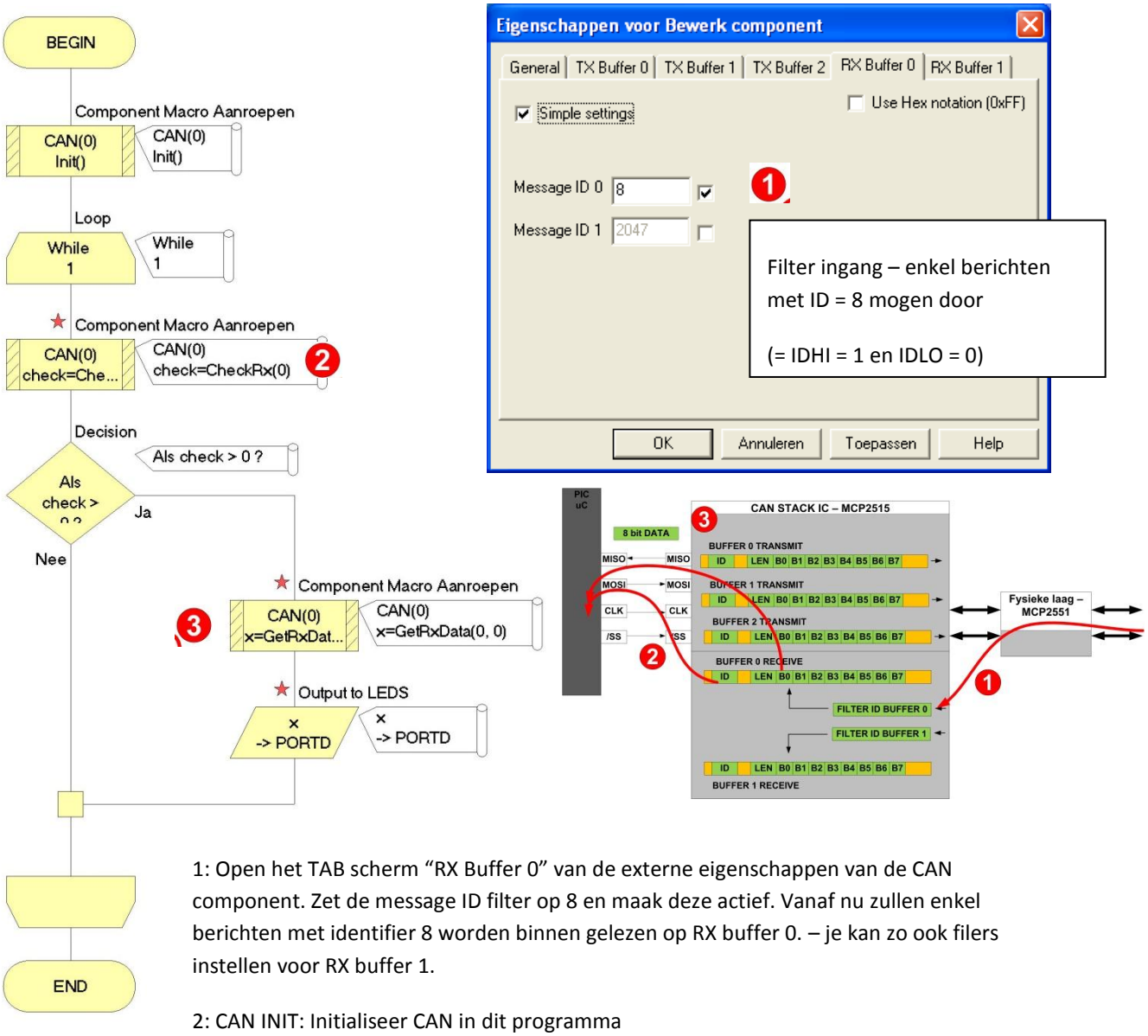
4: SetTXID(**0**,1,0): Zet de Identifier van het CAN bericht van **buffer 0** op 1,0. 1 is dan adres Hi, 0 is adres LO – samen maakt dit: **<u>0000 0001 000</u>**0 0000. Samen maakt dit een 16 bit adres waarvan enkel de 11 MSB's het eigenlijke adres bepalen – in dit geval is dit het adres "8" – (1000 = 8)

5: SetTXData(0,1,switch,0,0,0,0,0,0,0) .

- Met de eerste 0 maken we duidelijk dat we in buffer 0 schrijven.
- Met de tweede byte geven we aan hoeveel databytes zullen volgen (0-8) – in dit geval zullen we maar één byte data mee sturen – we vullen hier 1 in.
- Als data kunnen we 0 tot 8 bytes data mee geven met 1 CAN bericht. Wij geven met dit bericht enkel de variabele "switch" mee en maken de 7 resterende bytes 0

6: Sendbuffer: commando aan de MCP2515 om het volledige CAN bericht uit buffer 0 te verzenden.

ONTVANGEN VAN EEN CAN BERICHT:



1: Open het TAB scherm "RX Buffer 0" van de externe eigenschappen van de CAN component. Zet de message ID filter op 8 en maak deze actief. Vanaf nu zullen enkel berichten met identifier 8 worden binnen gelezen op RX buffer 0. – je kan zo ook filers instellen voor RX buffer 1.

2: CAN INIT: Initialiseer CAN in dit programma

3: CHECKCAN 0: Controleer of er nieuwe data staat in BUFFER 0 RECEIVE. Indien dit zo is dan wordt de variabele "check" 1 gemaakt.

4: Als "check" 1 is en er dus nieuwe data klaar staat zullen we deze data ook gaan binnenlezen.

5: GetRXDATA(0,0) : haal de data van RXBuffer0 binnen –de Index 0 geeft aan dat we de eerste byte binnenhalen. Met GetRXData(0,1) zouden we bijvoorbeeld de tweede byte van RXbuffer 0 binnen halen. We zetten deze data in variabele x.

6: Als laatste stap zetten we de data in variabele x – de data die we ontvangen hebben via CAN – op de leds aan POORTD.

CAN INSTRUCTIES IN FLOWCODE

zie help file!!

Init

Sets up the CAN component. Needs to be placed at the beginning of the program for the CAN component to work.

SendBuffer

This macro sends the information currently held in the specified <Tx_buffer> onto the CAN bus network.

CheckRx

This macro returns a value to indicate whether a message has been received by the specified <Rx_buffer>. A zero value indicates that a message has not been received, while a non-zero value indicates that a message has been received.

GetRxDataCount(buffer)

This macro returns a value that specifies how many data bytes are contained in the message received via the specified receive buffer. This macro would only be used if the user did not know how many bytes were contained in the message or if the message had a varying length.

GetRxData(buffer, index)

Returns a data byte from the location index in the specified buffer.

SetTxData(buffer, data_cnt, d0, d1, d2, d3, d4, d5, d6, d7)

This macro configures the Data for the specified buffer.

buffer is the Tx buffer to be used (0-2).

data_cnt is the number of bytes of data in the message.

IMPORTANT - This macro changes property values. The values listed on the property pages will then no longer be accurate. You will need to track the changed values yourself to ensure they are correct.

SetTxID(buffer, hi, lo)

Sets the Message ID value.

The Message ID value is a number between 0 and 2047 (0x00 and 0x7FF). However PICmicro's use 8 bit numbers 0-255 (0x00-0xFF) and so must use a set of two ID values to send the whole ID value.

The hi parameter is used for High byte of the Message ID value.

The lo parameter is used for Low byte of the Message ID value.

char GetRxIDLo(buffer)

Gets the Low byte of the Message ID.

This can be used along with the Hi byte of the Message ID to determine if the Message ID is to be enacted upon.

char GetRxIDHi(buffer)

Gets the High byte of the Message ID.

This can be used along with the Low byte of the Message ID to determine if the Message ID is to be enacted upon.

UITDAGINGEN:

Theorie:

- Teken blokschematisch hoe een CAN bus bericht er uit ziet.
- Wat is het verschil tussen HSCAN en FTCAN?
- Waarom is CAN bus ontworpen?
- Wat is de functie van de stack IC?
- Wat is de functie van de fysieke laag IC?
- Bespreek alle ingebouwde controlesystemen die CAN bus zo'n grote betrouwbaarheid geven
- Wat is het verschil tussen een bussysteem met klassieke adressering en een bus systeem met identifiers.
- Bij 29 bit CAN identifiers wordt er gedeeltelijk van deze identifiers afgestapt en wordt er toch terug een systeem van adressering toegepast. Raadpleeg het internet om dit 29 bit CAN protocol tot op bitniveau uit te spitten.

Praktisch:

- Ontwerp een CAN bus systeem met minimaal 4 nodes waarbij elk systeem een bepaalde controller in de wagen voorstelt. Denk bij elke functie goed na welke sensor invloed heeft om welke actuator of indicator:
    - A: dashboard
        - Indicatie standlichten aan
        - Knop om standlichten aan te schakelen
        - Indicatie mistlichten
        - Knop om mistlicht in te schakelen
        - Indicatie motor aan
        - Knop om motor aan te schakelen
    - B: achterlichten
        - Remlichten
        - Standlichten
        - Mistlichten
        - Eventueel spanning over serieweerstanden lichten meten als indicatie dat de lampen niet stuk zijn…
    - C: motor management
        - Temperatuurmeting motor (simuleren met potmeter)
        - Motor aan/uit zetten (simuleren met DC motor of leds)
        - Meten of motor aan staat
    - D: pedalen
        - Potmeter om gaspedaal te simuleren
        - Potmeter om rempedaal te simuleren

Deze cursus werd gerealiseerd met de ondersteuning van het **"Laagdrempelig Expertise- en Dienstverleningscentrum" "Netwerken in Voertuigen"** van de **bachelor opleiding autotechnologie**, departement technologie en informatica **(VHTI)** van **KATHO** te Kortrijk.

Via een netwerk van Laagdrempelige Expertise- en Dienstverleningscentra (afgekort LED) wil de Provincie West-Vlaanderen de kennis en expertise die binnen de drie West-Vlaamse hogescholen HOWEST, KATHO en KHBO aanwezig is vlot ter beschikking stellen van ondernemingen en non-profit organisaties. Het LED netwerk wordt gesteund door de provincie West-Vlaanderen, de Europese Unie, het Europees Fonds voor Regionale Ontwikkeling en de Vlaamse overheid.

http://www.lednetwerk.be

Het laagdrempelig expertise- en dienstverleningscentrum (LED) "**Netwerken in voertuigen**" bestudeert en onderzoekt netwerken in personenwagens, bedrijfsvoertuigen, landbouwvoertuigen en –machines.

Bedrijven en organisaties kunnen beroep doen op dit expertisecentrum voor advies, opleidingen, probleemoplossing, onderzoek, metingen, begeleiding van eindwerken, enzovoort.

Contact : Frans Devolder, Jeremy Lebon, Jurgen Van Mossevelde, docenten autotechnologie. frans.devolder@katho.be

jeremy.lebon@katho.be
jurgen.vanmossevelde@katho.be

http://netwerken-in-voertuigen.katho.be

Met de steun van: Provincie West-Vlaanderen Door mensen gedreven · Europese Unie · EFRO · Vlaanderen In Actie · Agentschap Ondernemen