

# *De Can-bus (controller area network)*

## *intelligente datacommunicatie voor in de praktijk*

### **DE CAN-ONTWIKKELING**

Begin jaren '90 kristalliseerden zich voor de internationale automobiellindustrie twee principiële probleemgebieden uit. Deze hadden te maken met technische ontwikkelingen op het gebied van personenauto's en vrachtwagens in de toekomst. Zo namen de eisen wat betreft comfort in voertuigen steeds meer toe: van elektronisch bediende ruiten, stoel- en spiegelverstelling, stoelverwarming, airconditioning tot aan audiovisuele presentaties en GPS-routebegeleidings-systemen. De markt eiste een steeds groter spectrum aan 'gemaksproducten'. Aan de andere kant werd natuurlijk ook de gebruiker zich steeds meer bewust van veiligheidsaspecten; de daarvoor noodzakelijke systeemuitbreidingen reiken van elektronische centraalvergrendeling, startblokkering, ABS-voorzieningen tot economisch en ecologisch motormanagement. Het onvermijdelijke gevolg van deze elektronificatie van motorvoertuigen is dat de noodzaak tot communicatie tussen afzonderlijke intelligente units binnen het chassis enorm zal toenemen. In het jaar 2005 zullen naar schatting tot 100 afzonderlijke microcontrollers onder het blik hun taken verrichten en onder elkaar gegevens moeten uitwisselen.

Het aantal communicatiewegen (in goed Nederlands: de kabelboom in het voertuig) moet daarvoor enorm worden uitgebreid: in auto's in de hogere prijsklasse zit tegenwoordig zo'n 2000 meter draad met een totaal gewicht van ruim 100 kg. Voor de diverse modellen van een fabrikant moeten soms wel 600 verschillende kabelbomen worden gemaakt. Uit het oogpunt van kosten en baten is daarmee het einde van de traditionele bedradingstechniek bereikt.

De automobiellindustrie, vooral de Duitse, Franse en Amerikaanse (VS), begon naar nieuwe, moderne communicatiemethodes te zoeken en kwamen terecht bij de bustechniek, die echter wel aan de bijzondere eisen van de voertuigtechniek moest voldoen zoals:

- datatransport met lage en hoge snelheid, met transmissiesnelheden van 5 kbit/s tot 1 Mbit/s voor luxe- en veiligheidselektronica,
- zeer betrouwbare data-overdracht met een 'Hamming-Distance' (HD) groter dan 4,
- geoptimaliseerd voor de overdracht van kleine data-hoeveelheden, zoals die vooral bij sensor/actuator-toepassingen voorkomen, m.a.w. nuttige overdracht per blok 0... 8 bytes,
- protocolcomponenten die door massafabricage heel goedkoop moesten zijn en makkelijk in het gebruik,
- een busopbouw die zo eenvoudig mogelijk moest zijn (bus-media, bus-topologie) voor integratie in het chassis.

Maar jammer genoeg ontwikkelde iedere grote automobiellfabrikant een eigen busconcept (natuurlijk niet compatibel met dat van de concurrent) en probeerde vervolgens deze huisstandaard internationaal aanvaard te krijgen. M.a.w. men trachtte zijn eigen concept in een internationale norm om te zetten met de bedoeling uit dit bussysteem zelf economische voordelen te halen.

Maar niet alles liet zich probleemloos tot standaard verheffen. In principe zijn er slechts vier belangrijke ontwikkelingen overgebleven: CAN (Controller Area Network) in een lage-en hogesnelheidsversie, VAN, J1850 SCP en J1850 DLC

Het weliswaar reeds genormeerde VAN-concept en vele andere nietgenormeerde ontwerpen van fabrikanten werden in de eerste helft van de jaren'90 opgegeven ten gunste van CAN, zodat CAN tegenwoordig wereldwijd toonaangevend is op het gebied van 'automobiel-bussen'.

In de auto-praktijk wordt CAN als low en highspeed data-overdrachtssysteem sinds 1992 in duurdere personenauto's van Mercedes (S-klasse) gebruikt. BMW, Porsche en Jaguar volgden en tegenwoordig gebruiken ook VW, Renault, Fiat e.a. de CAN-bus in hun middenklasse auto's.

Bijna gelijktijdig ontdekte ook de 'normale' industrie en wel in het bijzonder de automatiserings- en productiebranche de eerder genoemde voordelen van de automobiel-bus (vooral die van het CAN-concept) voor meten, sturen en regelen. Men vindt deze bus nu als communicatie-ruggegraat in PLC-systemen, robot- en motorsturingen, in gebouwen, liften, in de automatisering van laboratoria, in sensor/actuatorsystemen, kortom te veel om op te noemen.

Het CAN-protocol is inmiddels in **hardware 'gegoten'**, m.a.w. verkrijgbaar als IC's, en de gebruiker hoeft zich het hoofd niet meer te breken over eventuele details betreffende de communicatie: CAN-chips worden simpelweg als intelligente periferiebouwstenen in een bestaand of nieuw te ontwikkelen micro-controllersysteem geïntegreerd en klaar is het CAN-product.

De eigenschap van het nagenoeg probleemloze gebruik én het feit dat CAN-chips steeds goedkoper worden, maken de CAN-bus ook voor toepassingen die niet direct met de industrie te maken hebben, erg interessant. Bijvoorbeeld om kleine, decentrale communicatienetwerken (veldbus-systemen) op te bouwen.

## **DE NORMERING**

Als men wil proberen om de structuur van een communicatiesysteem in zijn algemeenheid vast te leggen, dan is het zinvol om eerst over vier fundamentele zaken duidelijkheid te verschaffen. en deze dan in overeenkomstige normen vast te leggen:

Hoe moeten de individuele deelnemers qua structuur op het communicatienetwerk worden aangesloten? M.a.w. hoe ziet de topologie van het netwerk eruit?

Hoe worden de deelnemers aan het netwerk gekoppeld en hoe vindt het datatransport van en naar het medium (kabel, glasvezel, draadloos voor radiografische en infrarood communicatie) plaats? M.a.w. hoe worden de signaalniveaus, connectors enz. gedefinieerd?

Wat zijn de regels voor informatieuitwisseling tussen de deelnemers? Hoe moeten overdrachtsfouten vermeden, herkend, gecorrigeerd worden? M.a.w. hoe moet het overdrachtsprotocol eruit zien?

Hoe kunnen deelnemers die willen zenden toegang tot het medium krijgen? Hoe worden conflicten opgelost als meerdere stations tegelijk willen zenden?

Voor uitsluitend ontvangst zijn er in het algemeen geen problemen te verwachten. In het kader van de specificaties kan een willekeurig aantal ontvangers, die op het overdrachtsmedium zijn aangesloten, gelijktijdig alle berichten afluisteren.

In zijn algemeenheid geldt voor een communicatiesysteem dat niet meer dan één zender tegelijkertijd actief mag zijn, maar dat wel meerdere ontvangers tegelijkertijd actief kunnen zijn.

Deze punten (en nog meer zaken) moeten dus eenduidig worden vastgelegd, wil een communicatiesysteem zinvol bruikbaar zijn en internationaal worden aanvaard. Met dat doel voor ogen begon begin jaren 90 de International Standardization Organisation (ISO) met de wereldwijde normering van de automobielbus, waarbij de CAN-bus steeds meer op de voorgrond trad.

De basis voor de normeringsactiviteiten op het reusachtige gebied van de open, fabrikant-onafhankelijke datacommunicatie is het uit 7 lagen bestaande ISO/OSI referentiemodel. Bij de veldbussystemen, waartoe ook de auto-bus hoort, bleven (tot nu toe) de layers (lagen) 3 t/m 6 leeg, zodat voor de CAN-bus alleen de lagen 1, 2 en 7 nader werden gespecificeerd.

<b>Layer 8</b> Application: "Device on Bus"	CANopen	DeviceNet	Smart Distributed System (SDS)
<b>Layer 7</b> "Application Layer"	CAL: CAN Application layer for industrial Applications	DeviceNet Specifications	SDS Specifications
<b>Layer 3 - 6</b>	Empty		
<b>Layer 2</b> "Data Link Layer"	LLC: Logical Link Control MAC: Medium Access Control acc. to ISO 11898 Result: CAN 2.0 A Specifications CAN 2.0 B Specifications		
<b>Layer 1</b> "Physical Layer"	"Low-Speed CAN" ISO 11519-2	"High-Speed CAN" ISO 11898	

**Layer 1:** Physical Layer (data-overdrachtsmedium).

Hierin vinden de definities plaats voor het data-overdrachtsmedium (in eerste instantie de buskabel), voor de connectors, voor de data-overdrachtniveau's en voor de zend-en ontvangbouwstenen. De twee hieruit volgende CAN-normen zijn:

*ISO11519-2:* Low-Speed-CAN. De basis hiervoor is een ontwikkeling waarmee de firma Bosch begin jaren '80 in Duitsland is begonnen en waarbij de ondersteuning van INTEL (omzetting van de protocollen in hardwarechips) van groot belang was. Low-Speed betekent hier dat data-overdrachtsnelheden van 5 kbit/s tot 125 kbit/s mogelijk zijn.

*ISO11898:* High-Speed-CAN. Hierbij worden data-overdrachtsnelheden tot 1 Mbit/s ondersteund.

**Layer 2:** Data Link Layer (bustoeegang en'foutherkenning).

Op deze laag wordt vastgelegd hoe (bij wens tot zenden) toegang wordt verkregen tot het data-overdrachtmedium, hoe het bericht is opgebouwd (adres-, data-, besturings- en foutherkennings/correctie-velden) en hoe het data-overdrachtprotocol is gestructureerd.

De definities hiervoor zijn ook in *ISO11898* te vinden.

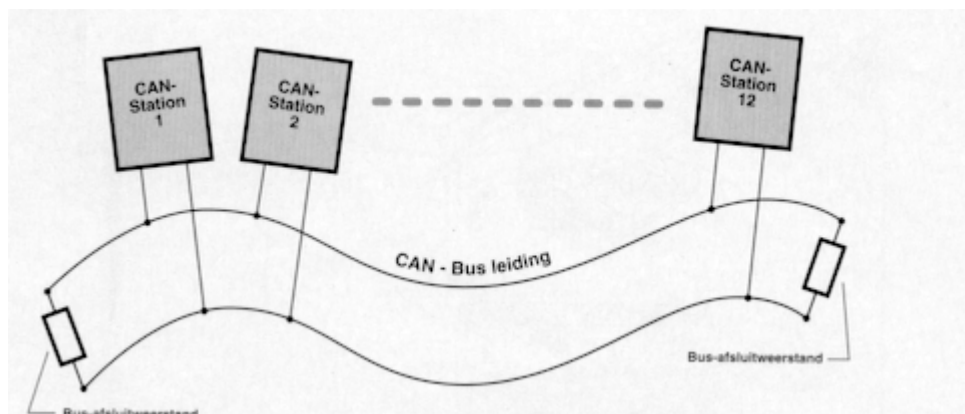
Daarnaast werden in 1991 de CANspecificaties voor Layer 2 nog uitgebreid, zodat men tegenwoordig van de twee parallel naast elkaar bestaande versies CAN 2.0A en CAN 2.0B spreekt. (Op gemeenschappelijkheden en verschillen tussen deze versies zie verder.) Omdat het CAN-concept zich ondertussen ook op veel andere industriegebieden heeft doorgezet, ziet de situatie op de Application Layer (laag 7) er complex en wat onoverzichtelijk uit. Deze laag is immers de directe interface met de eigenlijke toepassing (Layer 8), naar het feitelijke 'industrie apparaat aan de bus'. Drie grote CAN-vertakkingen voor verschillende toepassingsgebieden worden momenteel verder ontwikkeld: CANopen, DeviceNet en Smart Distributed System (SDS).

## SPECIFICATIES

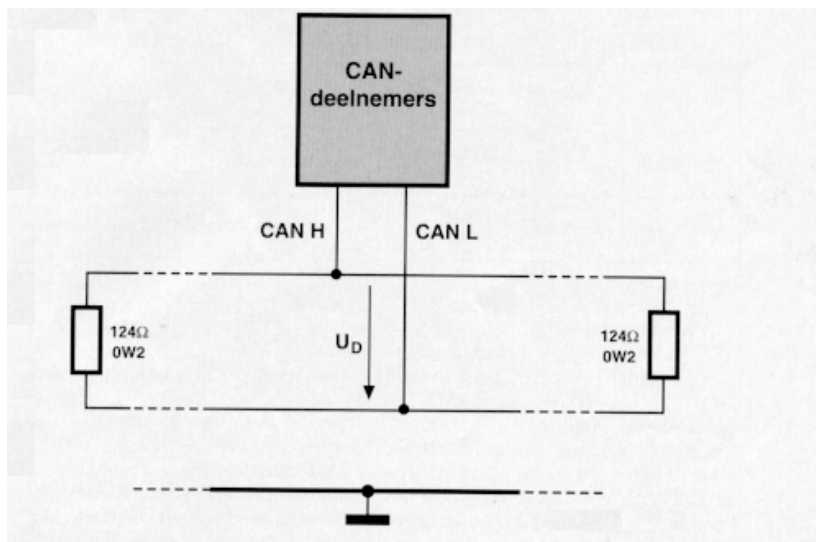
De CAN-definities, die in normen zijn vastgelegd moeten nu verder verduidelijkt worden. De koppeling aan de bus (Physical Layer) omvat de netwerk-topologie van de CAN-bus en de koppeling aan het busmedium.

Achter het begrip netwerk-topologie gaat de fysieke opbouw van het netwerk schuil, m.a.w 'hoe is de ordening van de stations aan busmedium'?

De CAN-bus gebruikt de zogenaamde bus-topologie: alle deelnemers zijn aan een enkele tweedraads leiding (twisted pair kabel, met of zonder afscherming) aangesloten, met aan beide einden bus-afsluitweerstand.



Ieder station kan hierbij onbeperkt met ieder ander station communiceren. Om aan het busmedium aan te koppelen wordt de zend/ontvangtrap van een CAN-busdeelnemer via twee aansluitingen, CAN-High (CANH) en CAN-Low (CANL) op de buskabel aangesloten



Voor de feitelijke data overdracht worden verschillspanningen gebruikt omdat die minder storingsgevoelig zijn. Hierbij is dus het spanningsverschil maatgevend. In ISO11898 worden twee verschillspanningsbereiken voor de representatie van data op de bus gedefinieerd: de recessieve en de dominante bustoestand. Dat men hier geen gebruik maakt van gewone logische '0'- en '1'-toestanden heeft een bijzondere reden, waarop later wordt ingegaan. In de eerste plaats geldt nu: Is het spanningsverschil  $U_D$  tussen CANH en CANL hoogstens 0,5 V (dus = 0,5 V), dan is er sprake van een recessieve toestand.

Is het spanningsverschil  $U_D$  tussen CANH en CANL minstens 0,9 V (dus = 0,9 V), dan is er sprake van een dominante toestand.

De nominale absolute niveaus van de busleiding, dat betekent het niveau van de individuele leidingen t.o.v. (locale) massa.

spanning op... bustoestand	recessief	dominant
CANH	2,5 V	3,5 V
CANL	2,5 V	1,5 v
toelaatbaar spanningsverschil: $U_d = \text{CANH} - \text{CANL}$	0... 0,5V	0,9...2,0V

De hier aangegeven absolute waarden zijn uiteraard van toleranties voorzien. In de praktijk zijn de onderaan in de tabel gegeven spanningsverschillen mogelijk. De definities voor ISO11519-2 (Low-Speed-CAN) zien er iets anders uit, maar omdat ISO11898 voor High- en Low-Speed-CAN geschikt is, worden tegenwoordig bijna uitsluitend ISO11898 buskoppelingen gebruikt.

De praktisch ingestelde gebruiker hoeft zich echter het hoofd niet te breken over de opbouw van een dergelijke zend/ontvangtrap, want van veel halfgeleiderfabrikanten zijn kant-en-klare transceiver-bouwstenen verkrijgbaar. Deze bouwstenen zijn voor wat betreft EMC-gedrag, afmetingen en thermische overbelastingsbeveiliging (bij kortsluiting van CANH en CANL) geoptimaliseerd en de signaalniveaus voldoen uiteraard aan de specificaties. Men hoeft alleen nog maar de busleiding aan te sluiten en klaar is de CANKoppeling.

Men moet alleen wel letten op de norm waarmee de bouwstenen werken: ISO11519-2 of ISO11898, waarbij de laatste de voorkeur verdient (opmerking: ook andere verschillspanning-methodes zijn voor CAN-bedrijf bruikbaar bijv. RS485. Belangrijke vragen die met betrekking tot de CAN~bus nog moeten worden gesteld, zijn:

Hoever kan de bus worden uitgebreid en welke data-overdrachtsnelheden zijn dan mogelijk?

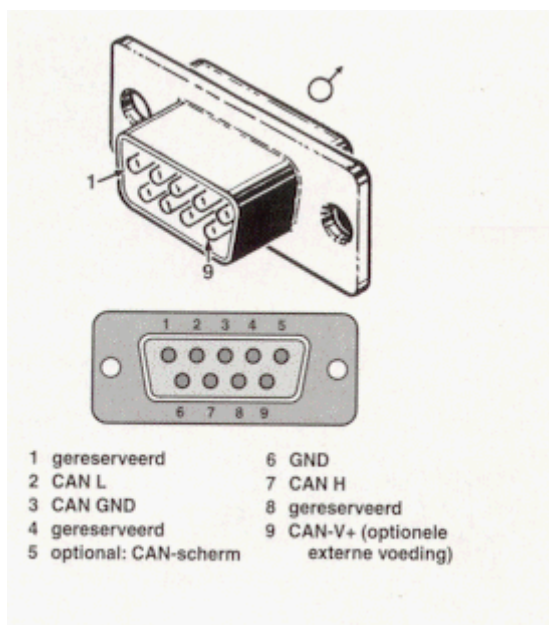
Hoeveel deelnemers kunnen op de bus worden aangesloten?

Busuitbreiding	bus	kabel	bus-afsluit-	max. overdrachtssnelheid
	categorie	doorsnede	Weerstand	
0... 40m	70mΩ/m	0,25...0,34mm <sup>2</sup> AWG23,AWG22	124Ω(1%)	1Mbit/s bij 40m
40...300m	<60mΩ/m	0,34...0,6mm <sup>2</sup> AWG22,AWG20	127Ω(1%)	500Kbit/s bij 100m
300...600m	<40mΩ/m	0,5...0,6mm <sup>2</sup> AWG20	150Ω tot 300Ω	100Kbit/s bij 500m
600...1Km	<26mΩ/m	0,75...0,8mm <sup>2</sup> AWG18	150Ω tot 300Ω	50Kbit/s bij 1Km

De antwoorden op deze vragen zijn in principe afhankelijk van de gebruikte buskabel. Bovestaande tabel toont het antwoord op de eerste vraag voor 32 aan de CAN-bus aangesloten stations (het maximale aantal volgens ISO11898). Als data-overdrachtskabel dient de voorkeur te worden gegeven aan wel of niet afgeschermd twisted-pair kabel.

Een voorbeeld: Men wil op een CAN-bus van 80 m een data-overdrachtsnelheid van 100 kbit/s halen. Dan moet twisted-pair kabel met een aderdoorsnede van 0,34 ... 0,6 mm<sup>2</sup> worden gebruikt en de bus-afsluitweerstand moeten 127Ω zijn. Verder moet de kabelweerstand kleiner zijn dan 60 m Ω /m, hetgeen echter bij doorsneden groter dan 0,30 mm<sup>2</sup> al het geval is. Voor de toevoerleidingen - in het geval deelnemers niet direct aan de CAN bus zijn aangesloten - gelden ook enkele regels. Deze mogen per deelnemer tot 250 kbit/s niet langer dan 2 m zijn en bij hogere bitrates niet langer dan 0,3 m. De lengte van alle toevoerleidingen samen mag niet groter zijn dan 30 m.

Ter afsluiting van de beschouwingen met betrekking tot de Physical-Layer kan nog worden opgemerkt dat ook de connectors en de aansluitingen genormeerd zijn.



Naast de koppeling aan de bus (Physical Layer) is het belangrijkste deel van de definitie van de bus het protocol voor de data-overdracht (Data Link Layer). De essentiële betekenis van een eenduidig genormeerd en erkend data-overdrachtsprotocol zal aan de hand van een eenvoudig voorbeeld worden verduidelijkt:

U wilt uw vriend opbellen en hem een boodschap meedelen. Het standaard data-overdrachtsprotocol ziet er ~ zoals we weten - ongeveer als volgt uit:

Hoorn van de haak nemen en op de kiestoon wachten. Als er geen kiestoon komt: de telefoon (of de aansluiting) is kapot; communicatie is niet mogelijk. De 'foutbehandelingsroutine' moet worden aangeroepen, in dit geval moet de storingsdienst worden verwittigd.

Als de kiestoon wel komt: nummer kiezen en wachten tot wordt opgenomen. Als de lijn in gesprek is, of als – nadat de bel meerdere keren is overgegaan er aan de andere kant niet wordt gereageerd: hoorn erop leggen en later opnieuw bij 1 beginnen.

Als de hoorn wordt opgenomen, kan met de vriend worden gesproken waarbij er beslist op het volgende moet worden gelet: praten en luisteren moeten elkaar steeds afwisselen, anders is het niet mogelijk om zinvol gegevens uit te wisselen.

Bij fouten in de data-overdracht (er werd iets niet goed begrepen), verifiëren en de data (boodschap) laten herhalen. Beëindiging van de communicatie geschiedt door de hoorn op de haak te leggen.

U ziet hier al dat de regels voor een zinvol verloop van de communicatie heel complex kunnen zijn (probeer het bovenstaande eens uit te leggen aan iemand die niet weet wat een telefoon is) en dat overtreding van de regels (bijv. kiezen zonder eerst de hoorn van de haak te hebben genomen) er toe leidt dat er geen communicatie tot stand komt.

Het is daarom noodzakelijk om aan het data-overdrachtsprotocol bijzondere aandacht te schenken. We beginnen daarom met het verklaren van enkele belangrijke begrippen:

## **BERICHTUITWISSELING**

De uitwisseling van berichten tussen de bus-deelnemers kan in principe op twee totaal verschillende manieren plaatsvinden:

### *Deelnemer-geöriënteerde berichtuitwisseling*

Hierbij spreekt de berichtzender de berichtontvanger heel concreet aan met zijn ontvangadres, bijv. 'station 25 stuurt een bericht naar station 37'. Er wordt dus over de bus een 'virtuele' (schijnbare) verbinding tussen zender en ontvanger opgebouwd, het bericht is slechts voor één station bestemd. In het verzonden datapakket staan daarom het adres van de ontvanger en het adres van de afzender (37 en 25). Alle andere stations aan de bus negeren dit datapakket van de zender, omdat het niet aan hen is geadresseerd.

De ontvanger verwerkt het bericht en bevestigt normalerwijs de correcte ontvangst. Bij fouten tijdens de data-overdracht (negatieve bevestiging van de kant van de ontvanger), herhaalt de zender het bericht.

### *Object-geöriënteerde berichtuitwisseling*

Hierbij wordt door de berichtzender aan zijn bericht een uniek berichtnummer toegekend (Identifier) en bericht en Identifier worden op de bus gezet, bijv. 'station A stuurt een spanningsmeetwaarde met Identifier 978', ontvang- en zendadressen worden niet aangegeven.

Dit bericht is daarom gelijktijdig voor 'ieder willekeurig aantal' ontvangers bestemd (broadcasting-principe) en het motto van de zender aan de ontvangers is dien overeenkomstig: 'pak van de bus, wat je nodig hebt'.

Alle aangesloten stations moeten nu zelf (aan de hand van hun interne software) beslissen of dit bericht voor hen van belang is.

## Communicatieverloop

Het verloop van de communicatie tussen de individuele stations aan de CAN-bus geschiedt nu in de vorm van een broadcast-uitwisseling van boodschappen (of te wel '(communicatie)objecten') ten gevolge van gebeurtenissen tussen de busdeelnemers (= object-geïoriënteerde berichtoverdracht). De nummering van de berichten is volgens hun prioriteit.

*Dominante en recessieve bustoestandenbits.* De eigenlijke data-overdracht op het data-overdrachtsmedium geschiedt hier niet zoals gebruikelijk in de vorm van logische nullen en enen, maar door dominante (overheersende) en recessieve (onderdanige) bits. Daarbij wordt door recessief een bustoestand gekarakteriseerd die door een tweede, dominante bustoestand kan worden overschreven. Als dus een station op de bus een recessief bit uitzendt en een ander station gelijktijdig een dominant bit zendt, dan overschrijft het dominante bit het recessieve, m.a.w. de dominante toestand (het dominante bit) gaat voor de hele bus gelden. De toewijzing van de logische toestanden voor deze bustoestanden geschiedt over het algemeen zodanig dat een logische nul de dominante en een logische één de recessieve toestand aangeeft.

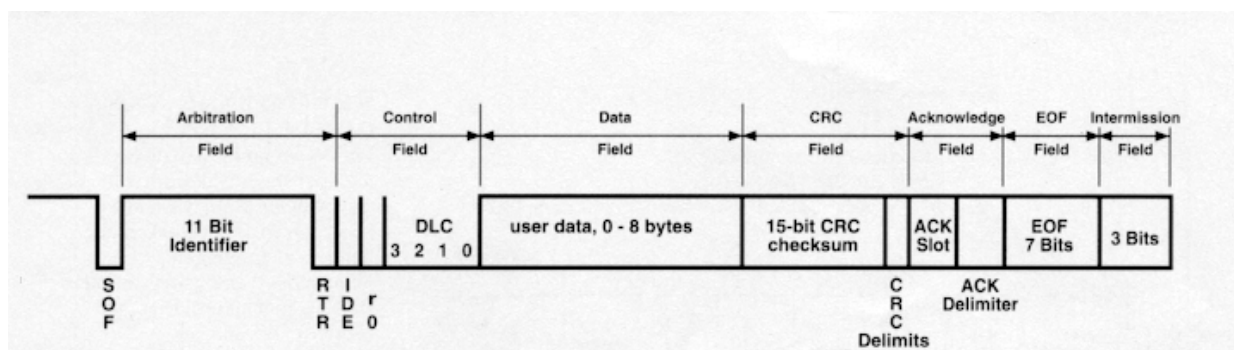
Deze afspraken vormen een wezenlijk kernpunt van de CAN-specificaties en worden verderop nog nader verklaard.

### *Communicatie-objecten*

Voor de uitwisseling van data op de bus gebruikt men bij CAN vier soorten communicatie-objecten, die ook frames (kaders) worden genoemd: data-frame, remote-frame, error-frame, overload-frame.

## Data-frame

Hiermee verzenden de CAN-stations naar eigen goeddunken (overeenkomstig hun software) hun data. De opbouw van zo'n, uit velden (fields) bestaand, frame is te zien in onderstaande figuur (standard-frame-format volgens CAN2.0A-specificatie).



De betekenis is als volgt:

**SOF (Start Of Frame-bit, altijd dominant (logisch nul)**

Hiermee wordt het begin van een frame aangegeven en alle busdeelnemers synchroniseren hun inwendige ontvangmodule met de neergaande flank van dit bit.

**Arbitration-field (12 bits)**

Dit veld bevat de informatie voor de regeling van de bustoegang

**11 bit identifier**

In dit deel staat de identifier (ID) van het verzonden communicatie-object. Met de 11 bits zijn  $2^{11} = 2048$  verschillende ID's mogelijk, waarvan echter slechts 2032 ID's vrij beschikbaar zijn, omdat de overige 16 ID's voor bijzondere functies zijn gereserveerd. Met andere woorden: in een enkelvoudig CAN-bus-systeem kan met 2032 verschillende objecten (meerwaarden, schakelstanden, lampfuncties, e.d.) worden gewerkt. Dat lijkt weliswaar erg veel, maar voor een hele reeks toepassingen is dat niet voldoende. Daarom heeft men het ExtendedFrame-Format met 29 identifier-bits gedefinieerd (CAN 2.0B), waarbij met  $2^{29} = 536.870.912$  verschillende objecten kan worden gewerkt (u leest het goed: vijfhonderd zes en dertig miljoen ... ).

**RTR (Remote Transmission Requestbit)**

Hiermee kan een station een ander station - heel doelgericht - opdragen om direct, zonder vertraging, zijn data (communicatie-objecten) te verzenden, bijvoorbeeld omdat ze ergens anders dringend nodig zijn (meer daarover later). Bij een data-frame is dit bit altijd dominant (logisch nul).

**Control-field (6 bits)**

Hier staat informatie over de opbouw van de data-frames.

**IDE (Identifier Extension-bit)**

Met dit bit wordt aangegeven of een StandardFrame-Format met een 11-bit-identifier (IDE = dominant, logisch nul) wordt verzonden of met een Extended-Frame-Format met een 29-bit-identifier (IDE = recessief, logisch één).

**r0 (reserve-bit 0)**

Dit dominant verzonden bit dient als reserve-bit voor toekomstige uitbreidingspecificaties.

**DLC (Data Length Code, 4 bits)**

Met deze vier bits wordt aangegeven hoeveel databytes er in het navolgende dataveld (data-field) staan. De CANspecificatie laat hierbij dataveldlengtes van 0... 8 bytes toe, m.a.w. een dataframe kan maximaal 8 gebruikersdatabytes bevatten.

**Data-field (0-8 bytes)**

In dit veld staan de te verzenden gebruikersdatabytes, 0 tot 8 stuks.

**Het CRC-veld (16 bits)**

In dit veld staat informatie ten behoeve van de beveiliging van de te verzenden data tegen bijv. storing. Aan de kant van de zender wordt daarbij - volgens bepaalde regels - een 15-

bits CRCcontrole(check)-som gemaakt, die wordt meegezonden in dit CRC-veld. De ontvanger berekent dan volgens dezelfde regels uit de ontvangen data ook een CRC-controlesom en vergelijkt deze met de ontvangen CRC-waarde van de zender. Als de controlesommen gelijk zijn, is het zo goed als zeker dat er geen data-overdrachtsfouten zijn. Als de waarden niet gelijk zijn, dan is een data-overdrachtsfout opgetreden en de , 'fout-behandelings-routine' begint (zie verder). Het CRC-veld wordt begrensd door het CRC-delimiter-bit, dat altijd recessief wordt verzonden.

**Acknowledge-Field (2 bits)**

Dit veld dient voor de overdracht van (positieve) ontvangstbevestigingen van de verzonden data.

**ACK-slot (1 bit)**

Dit bit wordt door de zender als recessief bit verzonden, het kan dus door een dominant bit van een ander station worden overschreven.

In dit bit-tijdvenster kunnen de aan de bus aangesloten ontvangers een positieve melding verzenden, als teken dat ze het data-frame foutvrij hebben ontvangen. Dit positieve bevestigingssignaal is een dominant verzonden bit dat ieder ontvanger bij foutvrije ontvangst uitzendt en dat dus het recessieve ACK-slot-bit van de zender overschrijft. Met andere woorden: als de zender tijdens het ACK-slot-tijdvenster een dominant bit ontvangt (in plaats van het door hem verzonden recessieve bit), dan weet hij dat tenminste één station zijn data-frame foutloos heeft ontvangen.

Het acknowledge-field wordt door het recessief verzonden ACK-delimiter-bit begrensd. Het complete Data-Frame wordt nu afgesloten met de EOF(End Of Frame)bitcombinatie, die uit zeven recessief verzonden bits bestaat.

Voordat een volgend frame kan worden verzonden, moet voor de ontvangers een kleine 'rustpauze' op de bus worden ingelast, zodat ze in staat zijn om de eerder ontvangen data te verwerken of op te slaan. Deze tijdvertraging wordt bereikt door na het verzenden van een frame een recessieve bustoestand van minstens 3 bittijden aan te houden, alvorens het volgende dominante startbit (SOF) van een frame te verzenden. Deze minimale wachttijd wordt door het intermission-field gevormd.

## **VERMIJDING VAN CONFLICTEN**

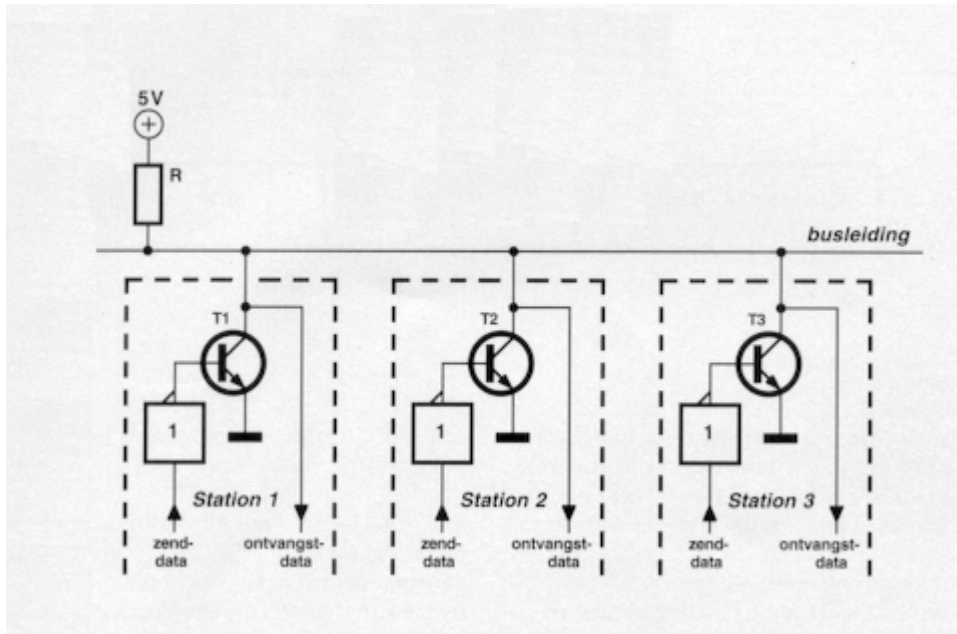
We komen nu toe aan de twee, tot nog toe openstaande, kernproblemen van de CAN-bus.

Aangezien alle CAN-stations aan de bus gelijke zendrechten hebben, zult u vragen: - Wat gebeurt er eigenlijk als meerdere stations tegelijkertijd iets willen verzenden?

- Welk station mag eerst zenden, welk station moet wachten?

Om deze conflicten op te lossen is er bij de CAN-bus een speciale bus-toegangsmethode (bus-arbitrage), waaraan alle stations zich moeten houden als ze wat willen verzenden.

Hierbij spelen de recessieve en de dominante bits van het arbitration-field een heel speciale rol. Principeel geldt hier: iedere zender 'luistert' naar zijn eigen uitzending op de bus: hij verzendt een bit, ontvangt het weer en vergelijkt of beide bits identiek zijn. Als dat het geval is, dan is de 'uitzending' nog in orde. Als een zender echter een andere bittoestand ontvangt dan degene die hij heeft verzonden, dan is er een probleem. We weten al dat een recessief bit (logisch één) door een dominant bit (logisch nul) kan worden overschreven. Laten we eens kijken naar, waarin de buskoppeltrappen van de CAN-stations vereenvoudigd zijn afgebeeld.



In principe gaat het hier om open-collector-uitgangen die gezamenlijk 'geWIRED-AND-ed' zijn. We kijken eens naar station 1: een recessief verzonden bit (logisch nul) zorgt ervoor dat transistor T1 gesperd blijft. Op de bus staat dus een recessief niveau (logisch één). Na het zenden van dit bit leest station 1 de bustoestand en herkent het zelf verzonden bit. Als nu een dominante toestand (logisch nul) wordt verzonden, dan is T1 in geleiding en de busleiding ligt aan massa. Nu bestaat er een dominante bustoestand. Ook hier leest station 1 de correct verzonden waarde terug.

Laten we nu eens alle drie stations bekijken, die op de bus zijn aangesloten. Het is dan gemakkelijk in te zien dat, zodra ook maar een enkel station een dominant bit (logisch nul) verzendt, de busleiding vast op het dominante niveau wordt gezet en alle andere stations lezen dit niveau terug. Na deze beschouwing kan men nu aan de hand van het volgende voorbeeld nagaan hoe de automatische toegang tot de bus, de arbitrage, bij de CAN-bus functioneert.

bit-tijdstip	a	b	c	d	e	f	g	h	i	j	k	l	
S		Identifiser											
O		←----->											
F													
station 1	0	0	0	1	0	1	1	0	1	1	1	1	367
station 2	0	0	0	0	1	1	1	0	1	0	0	0	0 0 1 232
station 3	0	0	0	0	1	1	1	0	1	1	1	1	239
bustoeestand	0	0	0	0	1	1	1	0	1	0	0	0	

**station 2 wint en zendt verder**

log.'0' = dominant bit (bustoeestand)  
 log.'1' = recessief bit (bustoeestand)

De drie in figuur getoonde stations willen hun data-frames met drie verschillende identifiers verzenden:

Station 1: berichtobject met identifier = 367

Station 2: berichtobject met identifier = 232

Station 3: berichtobject met identifier = 239.

Op bittijdstip a willen alle drie stations tegelijk toegang tot de bus om hun frames te verzenden. Ze beginnen alle met de arbitrage(bustoegang-)fase, door het verzenden van het SOF-bit. Dit SOF-bit is een dominant bit en ieder station leest nu eerst de juiste (eigen) waarde terug. Nu worden door de stations de identifiers verzonden: op tijdstip b verzenden ze alle een dominant bit en nòg is alles in orde. Op tijdstip c is er ook nog geen probleem. Op tijdstip d verzendt station 1 een recessief bit, stations 2 en 3 echter beide een dominant bit. Bij het teruglezen merkt station 1 dat zijn recessieve bit is overschreven en dat tenminste één station hem de toegang tot de bus heeft 'afgesnoept'. Vanaf dit tijdstip (d) stopt station 1 met zenden en gaat over op ontvangst (station 1 zal op een later tijdstip opnieuw proberen zijn data te verzenden). Stations 2 en 3 daarentegen zenden onverdroten verder.

Op tijdstippen d tot i zetten stations 2 en 3 hun data parallel op de bus en niets bijzonders gebeurt. Op tijdstip j echter verzendt station 3 een recessief niveau, dat nu door het dominante niveau van station 2 wordt overschreven. Dat merkt station 3 bij het teruglezen en concludeert daaruit dat vanaf nu de toegang tot de bus door minstens één ander station is gewonnen. Station 3 stopt daarom vanaf tijdstip j met zenden, gaat over op ontvangst en zal later nogmaals proberen zijn data te verzenden.

De 'bus-winnaar' is hier station 2, die nu onbelemmerd zijn complete dataframe verder verzenden kan. Hierbij nog de opmerkingen dat de arbitrage ook het RTR-bit omvat en dat altijd het motto geldt: 'Er kan er maar één winnen!'.

Als we nu de identifier wat beter bekijken, dan zien we dat het station - of beter het bericht - met de kleinste identifier altijd de toegang tot de bus wint en dus de hoogste zendprioriteit bij de data-overdracht heeft.

Zo wordt dus door middel van de identifier bij de CAN-bus een automatische berichtprioriteit verwezenlijkt: het bericht met identifier 0 bereikt direct en zonder belemmeringen de ontvangers (hoogste prioriteit), het bericht met identifier 2032 moet soms heel lang wachten voordat het de ontvangers bereikt (laagste prioriteit).

Dan komen we nu toe aan een ander belangrijk CAN-berichtobject (Frame):

#### Remote-Request-Frame

Laten we ons eens de volgende situatie voorstellen: Station D aan de CAN-bus verzendt regelmatig (om de vijf minuten) drie temperatuur-meetwaarden met identifier 598 (dataveldlengte dus 3 bytes), die door andere stations worden ontvangen en verwerkt.

Station G heeft nu (waarom doet er niet toe) onmiddellijk de nieuwste temperatuurmetingen nodig en kan absoluut geen vijf minuten wachten. Nu heeft station G de mogelijkheid om bij station D meteen de meerwaarden op te vragen, m.a.w. G kan op de dataoverdrachtscyclus inbreken. Daartoe verzendt G een "data-opvraagtelegram", een remote-request-frame. Dit is precies zo opgebouwd als een DataFrame, echter met vier kleine verschillen:

In het identifier-veld wordt de identifier ingevuld van het gewenste, dus van het berichtobject dat van het andere station wordt verlangd. In dit geval dus identifier = 598.

In het lengte-veld (DLC-veld) van het remote-frame staat altijd het aantal vereiste gebruikers-databytes, in dit geval dus de waarde 3.

Het bij een data-frame dominant (logisch nul) verzonden RTR(Remote Transmission Request)-bit wordt nu logisch één gemaakt en dus recessief verzonden. Dat is het teken dat een station doelbewust en direct gegevens van een ander station aanvraagt.

In het remote-frame zelf wordt geen dataveld verzonden (het dataveld bestaat niet), na het DLC-veld wordt meteen het CRC-veld verzonden. Het remote-frame is dus net zo opgebouwd als een data-frame met 0 bytes data.

Het verzonden remote-frame werkt nu als volgt: alle stations ontvangen dit frame en zien aan het RTR-bit (dat logisch één is) dat een station van een ander station data verlangt.

Station D merkt dat de identifier van het remoteframe overeenstemt met de identifier van zijn eigen data-pakket en verzendt als antwoord direct een data-frame met de gewenste data.

## **FOUTHERKENNING EN -AFHANDELING**

Een wezenlijk en zeer positief kenmerk van het CAN-bus-concept is dat het in staat is om een veelheid aan fouten bij de data-overdracht te herkennen en daarop op adequate wijze te reageren. Laten we eerst eens kijken hoe het met die foutherkenning is gesteld.

Bij de CAN-bus wordt bij de data-overdracht een Hamming-Distance van  $HD = 6$  bereikt. In de praktijk betekent dat het volgende:

In een CAN-bus-systeem wordt continu data verzonden met een overdrachtsnelheid van 500 kbit/s. Iedere 0,7 s treedt door uitwendige storingen een bitfout op (wat al op een extreem gestoorde omgeving wijst). Deze CANbus heeft een normale 'arbeidstijd' van 8 uur per dag, ze werkt 365 dagen per jaar. De ingebouwde 'foutzekerheid' van de CAN-bus staat er nu voor garant dat in een bedrijfstijd van circa 1000 jaar slechts één fout niet wordt herkend. Overigens treden er altijd fouten op, maar fouten die worden gecorrigeerd zijn geen fouten meer.

Gevaarlijk zijn fouten die niet worden herkend en die er toe kunnen leiden dat er met verkeerde gegevens wordt gewerkt.

Door welke maatregelen wordt bij de CAN-bus deze hoge betrouwbaarheid bereikt?

### Herkenning van data-overdrachtsfouten

Hiervoor worden bij de CAN-bus vijf verschillende concepten gelijktijdig gebruikt:

#### *Bitfoutherkenning*

Iedere deelnemer luistert mee naar zijn eigen uitzending. Als hij nu, na de arbitreringsfase, zijn data als enige op de bus verzenden mag en hij ontvangt een andere bittoestand dan degene die hij heeft verzonden, dan is dat voor hem een teken dat er een fout op het data-overdrachtsmedium (bus) is opgetreden. Hij stopt dan met zenden en gaat over op de fout-afhandelingsroutine (zie verderop).

#### *Stuffbit-foutherkenning*

Wat dit betreft is de CAN-norm heel duidelijk: als in een CAN-Frame meer dan vijf bits in dezelfde (logische) toestand na elkaar moeten worden verzonden (bijv. zeven maal een logische nul), dan wordt automatisch na iedere reeks van vijf identieke bits een complementair bit (hier dus '1') tussengevoegd en verzonden. Dit toegevoegde bit, dat natuurlijk geen informatieinhoud bezit, wordt stuff-bit genoemd (to stuff = opvullen). De ontvanger verwijdert deze bits uit de datastroom, zodat het oorspronkelijke bericht weer wordt hersteld. Deze stuff-bit-eigenschap kan heel goed worden gebruikt om fouten te herkennen: als de ontvanger in een frame meer dan vijf bits achter elkaar met dezelfde logische toestand ontvangt (behalve het EOFveld); dan weet hij dat dit niet mogelijk is en er dus sprake moet zijn van een data-overdrachtsfout (doordat één of meer bits werden geïnverteerd). De ontvanger gaat dan over op de foutafhandelingsroutine.

### *CRC-foutherkenning*

Hierbij gaat het, zoals reeds eerder vermeld, om het bepalen van de CRCcontrolesom door de ontvanger. Als de ontvangen en zelf berekende controlesommen verschillen, dan gaat de ontvanger ook hier over naar de foutafhandelingsroutine.

### *Acknowledge-foutherkenning*

Bij de beschrijving van het Frame-Format hebben we ook het ACK-slot-bit leren kennen, dat door de datazender als recessief bit wordt verzonden. Alle ontvangers die het uitgaande frame correct hebben ontvangen, overschrijven dit bit met een dominant bit. De zender ziet dat en weet dat minstens één zender zijn data goed heeft ontvangen.

Als de zender echter vaststelt dat zijn ACK-slot-bit helemaal niet is overschreven, dan weet hij dat geen enkele ontvanger zijn data goed heeft ontvangen. Nu gaat de zender over naar de foutafhandelings-routine.

### *Format-foutherkenning*

Hierbij maken de stations gebruik van het feit, dat er in het CAN-Frame-Format een aantal velden zijn, die altijd een vaste inhoud moeten hebben: de CRCdelimiter, de acknowledge-delimiter en het EOF-veld bestaan altijd uit recessieve bits. Wordt hier een dominant bit aangetroffen, dan kan dat alleen maar door een data-overdrachtsfout komen. Ook hier wordt dan de fout-afhandelings-routine aangeroepen.

## **Behandeling van data-overdrachtsfouten**

De fout-afhandelingsroutine die bij data-overdrachtsfouten in werking treedt, werkt tweeledig. Ten eerste wordt een frame dat als foutief wordt herkend, door het station onmiddellijk weggegooid en niet verwerkt.

De tweede stap dient er toe om bij de CAN-bus de totale zekerheid van alle data over het hele systeem te garanderen. Zodra een station een fout herkent, verzendt het direct een zogeheten error-frame, dat uit 6 dominante bits (= error-flag) en een error-delimiter (8 recessieve bits) bestaat. Dit heeft dan verstrekkende gevolgen: door de 6 dominante bits worden alle recessieve bits op de bus overschreven en er blijven 6 dominante bits over. Dat is echter in strijd met de bit-stuffingregel, die zegt dat maar 5 bits achter elkaar met dezelfde logische toestand op de bus mogen worden gezet.

Alle andere deelnemers op de bus herkennen deze foutcode en constateren dat het zojuist gezonden frame fout was, gooien het weg en verzenden zelf ook een error-frame. Met andere woorden: een station dat een fout herkent, stoort bewust het hele verzonden frame, zodat alle andere stations ook een fout frame ontvangen. Een lokaal door een station herkende fout wordt dus gemeenschappelijk gemaakt voor alle stations.

Bij de behandeling van fouten komt het motto van de CAN-bus wel heel mooi tot uiting: 'Ofwel alle stations ontvangen gemeenschappelijk juiste gegevens die verder worden verwerkt, ofwel alle stations ontvangen gemeenschappelijk foute gegevens die niet verder worden verwerkt.' Op die manier wordt dan de betrouwbaarheid van de data voor het hele systeem gegarandeerd. De verzenden van de data merkt zelf natuurlijk ook dat zijn frame werd gestoord, stopt met zenden en probeert het na een tijdje opnieuw.

### *Inwendige stationsfouten*

Wat gebeurt er nu als een CAN-busstation inwendig defect is of bijvoorbeeld met een verkeerde (onnauwkeurige) data-bitrate werkt, of als het station als enige lokaal wordt gestoord?

Zo'n station zou dan continu (onterechte) error-frames uitzenden en daarmee de hele bus plat leggen. Ook hier voorziet het CAN-bus-concept in oplossingen, waarop we echter uit plaatsruimtegebrek niet verder in kunnen gaan.

### SAMENVATTING

In tabel 1 zijn tenslotte de gegevens van de beide actuele CAN-versies naast elkaar gezet: CAN2.OA (Standard-Frame-Format) en CAN2.OB (Extended-Frame-Format).

	CAN 2.OA	CAN 2.OB
max aantal beschikbare identifiers (boodschappen) per bussysteem	$2^{11}$	$2^{29}$
Aantal stations(knooppunten) per bussysteem	max 32	max 32
Data-overdrachtsnelheid	5kbit/s tot 125kbit/s	5kbit/s tot 1Mbit/s
hoeveelheid gebruikersdata per frame	0 tot 8 byte	0 tot 8 byte
Max lengte van een fame	117bit	136bit

De CAN-bus werd voorgesteld als een veldbus-systeem voor data-overdracht met hoge prestaties en een hoge mate van foutzekerheid. De man in de praktijk zal zich mede in verband met het complexe data-overdrachtsprotocol wellicht afvragen: hoe moet men zoiets nu in concreto realiseren? Er zijn dominante en recessieve bits, een 11-bits identifier, 15-bits CRC-controlesom, 1bit delimiter, 7-bits EOF-velden, 6-bits fout-Frames enzovoort. Van de bekende en vertrouwde 8- of 16-bitdatastructuur van microcontrollers is niets meer te herkennen. Als men dus het CAN-protocol zou willen programmeren, dan zou dat neerkomen op een flinke portie werk op machinetaal-niveau. Maar daar hoeft men niet bang voor te zijn. Een groot voordeel van het CAN-concept is namelijk dat een groot aantal fabrikanten reeds kant-en-klare (en goedkope) protocolcomponenten maakt, die alles doen wat in dit artikel is beschreven. Deze ondersteuning van de kant van halfgeleiderfabrikanten heeft de CAN-bus al zeer populair gemaakt.