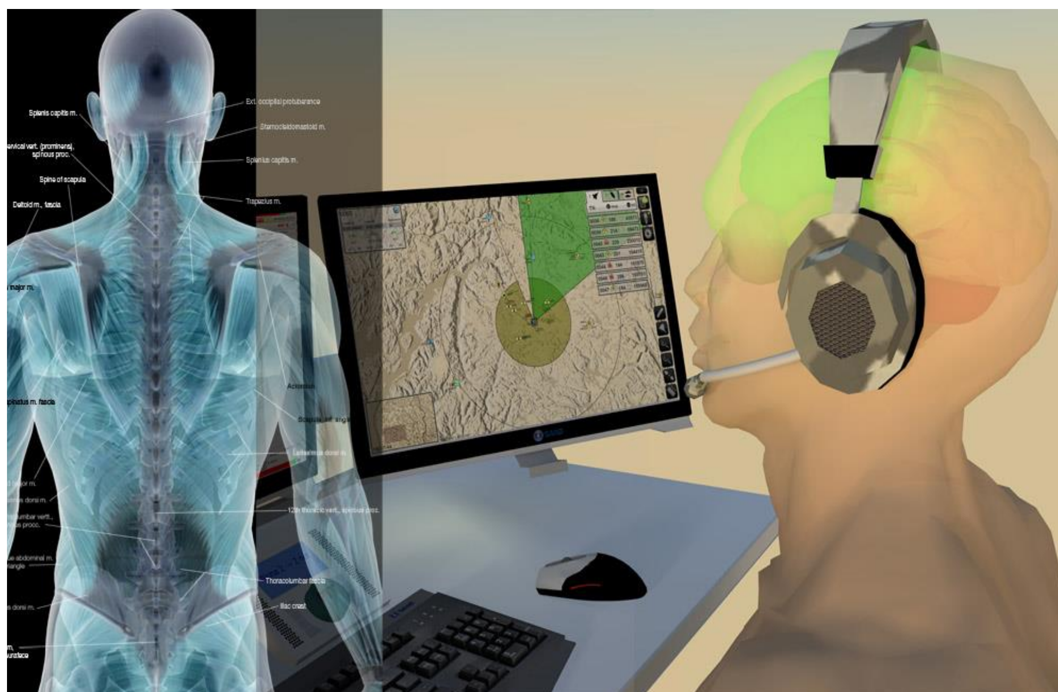
CHALMERS



Kommunikationsgränssnitt med CANopen

Communication interface with CANopen

Examensarbete inom högskoleingenjörsprogrammet Elektroingenjör

ANDERS KLAVMARK

TERJE VIKINGSSON

Examinator: Bill Karlström
Institutionen för Signaler och System
Avdelningen för Signalbehandling
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige, 2013

Förord

Examensarbetet genomfördes under 11 veckor, våren 2013 på Saab AB, Electronic Defense Systems, EDS i Kallebäck, Göteborg.

Denna rapport är ett examensarbete på 15 högskolepoäng och är utfört av Anders Klavmark och Terje Vikingsson från Chalmers Tekniska Högskola. Arbetet genomfördes på Saab AB, på avdelningen C3 (Command, Control and Communication) Training Systems & Computer Platforms. Tanken är att rapporten ska ge en inblick i möjligheten att använda CAN-buss kommunikation och genom att konfigurera systemets olika komponenter i ett nätverk kunna avläsa dess status. Under de tio veckor som spenderats på Saab har vi lärt oss mycket nytt och vi har fått en djupare förståelse för vad det innebär att arbeta som ingenjörer på ett företag som ligger i teknikutvecklingens framkant.

Vi skulle vilja rikta ett speciellt tack till vår handledare på Saab, Gustav Jansson, för all hjälp och guidning. Utan honom skulle vi aldrig kommit dit vi är idag. Ytterligare anställda på Saab som vi skulle vilja tacka är Per Örbäck, Lars-Olof Aridun, Björn Andersson, Jonas Granath och övriga anställda på avdelningen. Tack också till Manne Stenberg, vår handledare på Chalmers.

Sist men inte minst vill vi uttrycka ett stort tack till Anders Martinsson, sektionschef för Runtime Environments på Saab EDS, som gjorde detta examensarbete möjligt och snabbt såg till att vi kom in i gemenskapen.

Sammanfattning

Den här rapporten beskriver utvecklingen av en prototyp som visar funktionaliteten hos ett CANopen-nätverk med master-slave konfiguration. Till grund för arbetet ligger Saabs önskan om att implementera nätverket i sina markbaserade radarsystem. I dagsläget finns begränsade styr- och övervakningsmöjligheter, vilket är skälet till att man vill koppla samman ingående komponenter i ett nätverk. Tanken är att man ska övervaka dessa komponenter från en kontrollpanel med hjälp av CAN-buss via Ethernet, IEEE 802.3 till kontrollenheten.

En inledande studie av CAN-standarden ISO-11898 görs, där teknikens grunder förklaras och dess fördelar påvisas. Högnivåprotokollet CANopen utreds med en genomgång av funktionerna som karakteriserar det och dess relation till de lägre lagren i OSI-modellen. Hård- och mjukvara anskaffas och en fungerande prototyp utvecklas för en kommande demonstration. Avslutningsvis görs en genomgång av programdesignen och möjligheterna till fortsatt utveckling.

Abstract

This report describes the development of a prototype that demonstrates the functionality of a CANopen network with master-slave configuration. The basis of the work is Saab's wish to implement the network on their ground-based radar systems. Currently, the state control and monitoring capabilities of these units are limited, which is why their various electrical components need to be connected in a common network. The idea is to monitor these components from a control panel by CAN-bus via Ethernet, IEEE 802.3 to the control unit.

An initial study of the ISO-11898 CAN standard is made, where the basics of the technology are explained and its benefits demonstrated. The high-level CANopen protocol is investigated with an overview of the features that characterize it and its relation to the lower layers of the OSI-model. Hardware and software are acquired and a working prototype is developed for demonstration purposes. Finally, a review of the application design is done and the potential for further development is assessed.

Innehållsförteckning

Förord ... I
Sammanfattning ... II
Abstract ... III
Innehållsförteckning ... IV
Figurer ... VI
Ordlista ... VIII
1 Inledning ... 1
1.1 Bakgrund ... 1
1.2 Syfte ... 2
1.3 Precisering av arbetsuppgiften ... 2
1.4 Avgränsningar ... 4
2 Metod ... 5
2.1 Kravinsamling ... 5
2.2 Planering/tidsplan ... 5
2.3 Analys ... 5
2.4 Design ... 6
2.5 Integration ... 6
2.6 Demonstration ... 6
2.7 Rapport ... 7
2.8 Presentation ... 7
3 Teknisk bakgrund ... 8
3.1 Open System Interconnection, OSI ... 8
3.1.1 Lager 7, Applikationslagret ... 9
3.1.2 Lager 6, Presentationslagret ... 9
3.1.3 Lager 5, Sessionslagret ... 9
3.1.4 Lager 4, Transportlagret ... 9
3.1.5 Lager 3, Nätverkslagret ... 10
3.1.6 Lager 2, Datalänklagret ... 10
3.1.7 Lager 1, Fysiska lagret ... 10
3.2 Controller Area Network, CAN ... 11
3.2.1 Varför CAN? ... 12
3.2.2 Protokoll ... 13
3.2.3 Topologi (Fysiska Lagret) ... 14
3.2.4 Bitrepresentation (Fysiska Lagret) ... 16
3.2.5 Arbitrering (Datalänklagret) ... 18
3.2.6 Meddelandeformat (Datalänklagret) ... 20
3.2.7 Felhantering (Datalänklagret) ... 22
3.3 CANopen ... 24
3.3.1 Struktur ... 25
3.3.2 Objektlistan ... 27
3.3.3 Adressering ... 28
3.3.4 Kommunikationsmodeller ... 30
3.3.5 Service Data Object Protocol, SDO ... 31
3.3.6 Process Data Object Protocol, PDO ... 35
3.3.7 Network Management Protocol, NMT ... 40
3.3.8 Special Object Protocol ... 42
3.4 Ethernet ... 46
3.5 CANfestival ... 48

4 Genomförande 50
4.1 Undersökning och val av hårdvara och mjukvara 51
4.1.1 Hårdvara 51
4.1.2 Mjukvara 53
4.2 Programdesign 54
4.2.1 Programdesign, Master 56
4.2.2 Programdesign, Slav 57
4.3 Integration av systemet 59
4.4 Programkörning av systemet 61
5 Resultat 62
6 Slutsats 63
6.1 Kritisk diskussion 64
6.2 Fortsatt utveckling 65
6.2.1 CANopen över EtherCAT? 65
6.2.2 Intelligent Platform Management Interface, IPMI 65
6.2.3 SAE J1939 66
6.2.4 Befintligt CANopen-nätverk 66
7 Referenser 67
8 Bilagor 69
8.1 Appendix A – Artila Matrix 522 69
8.2 Appendix B – PEAK PCAN-USB 71
8.3 Appendix C – Kommunikationsgränssnittmanual 72

Figurer

Figur 1 *Arthur/Giraffe (Saab AB, 2013)* 1
Figur 2 *Översikt av hela nätverket* 2
Figur 3 *Översikt av arbetsuppgiften* 3
Figur 4 *Processindelning* 5
Figur 5 *OSI-modellens kommunikationsväg* 8
Figur 6 *CAN buss* 14
Figur 7 *Maximal bithastighet och busslängd (Pfeiffer, Ayre & Keydel, 2003)* 15
Figur 8 *Manchester och NRZ (Sony Corp.)* 16
Figur 9 *Nominell bittid (Mannisto & Dawson, 2003)* 17
Figur 10 *Arbitrering (Wells, 2001)* 20
Figur 11 *CAN meddelanderam (Wells, 2001)* 21
Figur 12 *CANopen sett från OSI-modellen (CAN in Automation 2002 CANopen Application Layer and Communication Profile)* 24
Figur 13 *Datagången mellan de olika lagren (CAN in Automation CANopen)* 25
Figur 14 *CANopen enhetsmodell (CAN in Automation CANopen)* 26
Figur 15 *Objektlistans struktur* 27
Figur 16 *Objektlistan index och delindex koncept* 28
Figur 17 *Meddelandetypernas struktur* 29
Figur 18 *Producer/Consumer-modell (CAN in Automation CANopen)* 30
Figur 19 *Server/Client-modell (CAN in Automation CANopen)* 30
Figur 20 *Master/Slave-modell (CAN in Automation CANopen)* 31
Figur 21 *SDO-struktur i ett nätverk (CAN in Automation CANopen)* 31
Figur 22 *Segmenterad SDO-överföring (CAN in Automation CANopen)* 32
Figur 23 *Påskyndad SDO-överföring vid uppladdning (CAN in Automation CANopen)* 33
Figur 24 *Segmenterad SDO-överföring vid uppladdning (CAN in Automation CANopen)* 34
Figur 25 *Definition av SDO-parametrar (CAN in Automation CANopen)* 34
Figur 26 *Objektlistans struktur med avseende på SDO (CAN in Automation CANopen)* 35
Figur 27 *PDO struktur (CAN in Automation CANopen)* 35
Figur 28 *Definition av kommunikationsparametrar* 36
Figur 29 *Definition av mappningsparametrar* 36
Figur 30 *Mappningsstruktur* 37
Figur 31 *Objektlistans struktur med avseende på PDO (CAN in Automation CANopen)* 37
Figur 32 *Olika överföringsmetoder (CAN in Automation CANopen)* 38
Figur 33 *Hur olika överföringstyper sätts* 38
Figur 34 *Överföring av PDO med fördröjningstid (CAN in Automation CANopen)* 39
Figur 35 *Mappningsexempel för PDO (CAN in Automation CANopen)* 39
Figur 36 *NMT-struktur (CAN in Automation CANopen)* 40
Figur 37 *CANopen tillståndsmaskin (CAN in Automation CANopen* 41
Figur 38 *NMT meddelandestruktur (CAN in Automation CANopen)* 41
Figur 39 *SYNC-signalens gång i nätverket (CAN in Automation CANopen)* 42
Figur 40 *SYNC-signalens tidsfönster och period (CAN in Automation CANopen)* 43
Figur 41 *Time Stamp-struktur (CAN in Automation CANopen)* 44
Figur 42 *Felmeddelandens gång i ett nätverk (CAN in Automation CANopen)* 44

Figur 43 *Felmeddelandets struktur (CAN in Automation CANopen)* 45
Figur 44 *Fördefinierade akuta felkoder (CAN in Automation CANopen)* 45
Figur 45 *Ethernet Meddelanderam (Singh, 2010)* 47
Figur 46 *CANfestival-implementation(CANfestival, 2001)* 48
Figur 47 *Artila Matrix 522 (Artila 2013)* 52
Figur 48 *PEAK PCAN-USB (PEAK SYSTEMS 2013)* 53
Figur 49 *Master/Slave-kommunikation (ICPDAS 2013)* 54
Figur 50 *Kommunikationsbegränsningar* 55
Figur 51 *Tillståndsmaskin (Pfeiffer, Ayre & Keydel, 2003)* 55
Figur 52 *Översikt av systemet* 61
Figur 53 *Programval vid körning av slavprogram* 61

Ordlista

ARM	Advanced RISC Machine, Processorarkitektur
Baudrate	Måttenhet för signalöverföring
CAN	Controller Area Network, Nätverksteknologi
CANfestival	CANopen ramverk av typen öppen källkod
CANopen	Kommunikationsprotokoll och enhetsprofilspecifikation för inbyggda system
CiA	CAN in Automation
COB	Communication Object
CRC	Cyclic Redundancy Check
CSMA	Carrier-Sense Multiple Access
DLC	Data Length Code
EOF	End Of Frame
Ethernet	Nätverksteknologi
GNU	General Public License Operativsystem
IA	Information Assurance, term för beskrivning av informationssäkerhet.
IDE	Identifier Extension
IEEE 802.3	Standard för Local Area Network, som reglerar Ethernet
IFS	Intermission Frame Space
IPMI	Intelligent Platform Management Interface
ISO 11898	Specifikation för seriella kommunikationsteknologin Controller Area Network
Linux	Unixliknande operativsystem
MAC	Media Access Control
MMU	Memory Management Unit, översätter virituella adresser till fysiska adresser i minnet.
NMT	Network Management
OS	Operating System
OSI	Open Systems Interconnection

PDO	Process Data Object, finns två typer: Receive och Transmit (RPDO & TPDO)
RTR	Remote Transmission Request
SAE	Society of Automotive Engineers
SDO	Service Data Object
SNMP	Simple Network Management Protocol
SOF	Start Of Frame
SYNC	Signal som används för synkronisering
Toolchain	Programmeringsverktyg
UCU	User Control Unit
UCP	User Control Panel
USB	Universal Serial Bus

1 Inledning

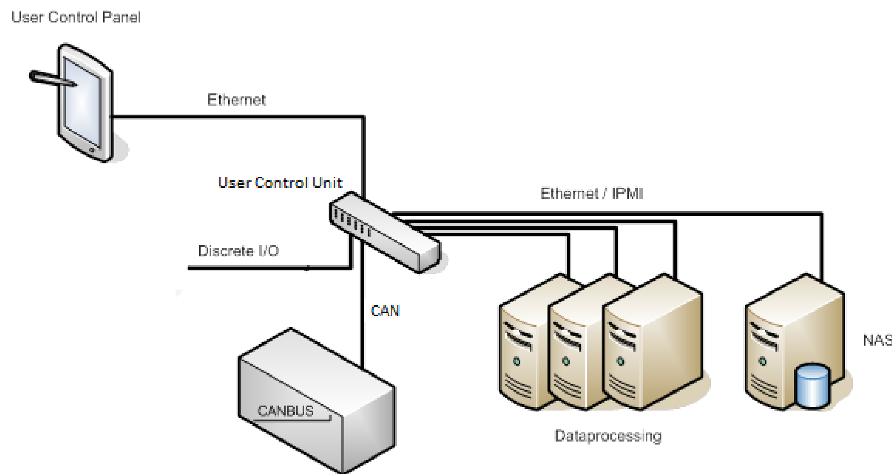
I takt med att elektroniska system blir mer och mer komplexa, ökar behovet att kunna övervaka status på ingående delar i ett service- och underhållssyfte. Ett sätt att styra och övervaka detta är att koppla ihop alla ingående komponenter i ett nätverk och låta en kontrollpanel övervaka status i systemet.

1.1 Bakgrund

Saab Electronic Defence Systems är en leverantör av mark- och flygbaserade radarsystem som används för att upptäcka, lokalisera och skydda mot hot. Här utvecklas bland annat radarsystemen Giraffe™ och Arthur™, se Figur 1, vilka tillhandahåller övervakning och stridsledning för närluftvärnsystem respektive lokalisering av fientligt artilleri. I dessa system ingår flera olika hårdvarukomponenter som i dagsläget har begränsade styr- och övervakningsmöjligheter. I nuvarande system avläses status på komponenterna med diskreta signaler, men förmågan att kommunicera med dem via exempelvis Ethernet till/från en User Control Panel, UCP, saknas. En UCP är en panel/bildskärm med touch-funktionalitet där man kan styra, övervaka och presentera status för systemet.

För att kunna möjliggöra detta behöver ingående noder konfigureras i ett nätverk. Tanken är att alla dessa komponenter ska kopplas samman i ett Controller Area Network, CAN, för att underlätta informationsöverföringen till UCP. Utöver kommunikationen med CAN-nätverket ska man från UCP kunna läsa av status, sätta diskreta signaler och även styra datorer/datorkort, se Figur 2.

Figur 1 *Arthur/Giraffe (Saab AB, 2013)*



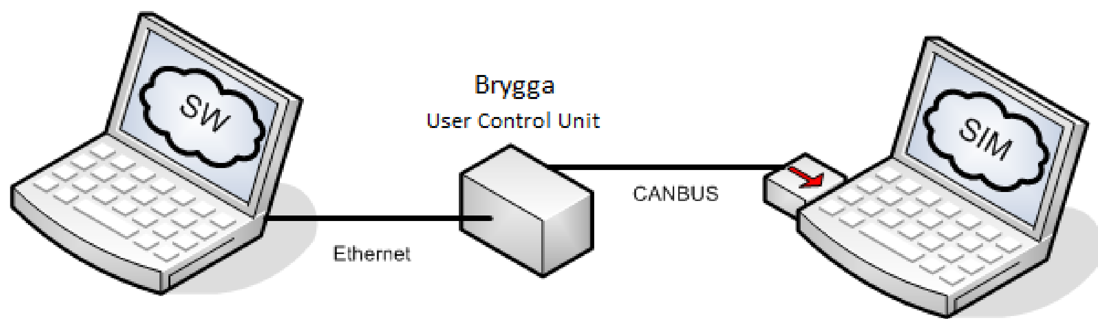
Figur 2 *Översikt av hela nätverket*

1.2 Syfte

Syftet med detta projekt är att ge ett förslag på kommunikationsgränssnitt som kan lösa systemets behov av övervakning och kontroll av ingående noder/komponenter, samt att ta fram en prototyp för att genomföra en demonstration. Den prototyp som tagits fram är ett första steg i att visa hur framtida lösningar skulle kunna utformas för system hos Saab.

1.3 Precisering av arbetsuppgiften

Arbetsuppgiften består i att undersöka om det finns produkter som uppfyller de krav som ställs och i så fall ge förslag på en lämplig enhet att utveckla en CAN-till-Ethernet-brygga på. Om ingen produkt kan tillfredsställa kraven skall en egen konstruktion tas fram. Vidare skall en simulerad CAN-buss miljö skapas. Två hårdvaruenheter behövs, en enhet för utveckling av bryggan och en CAN-till-USB-adapter som möjliggör kommunikation mellan bryggan och den simulerade CAN-miljön. Bryggan skall fungera som en master till den simulerade CAN-buss miljön och hela systemet skall kunna styras och övervakas via Ethernet, enligt Figur 3. Själva CAN-miljön skall realiseras med CANopen då Saab valt att använda denna standard i sina system.



Figur 3 *Översikt av arbetsuppgiften*

För att styra kommunikationen mellan komponenterna behövs en User Control Unit, UCU, som bryggar kommunikationen mellan nätverken. Systemen som skall använda sig av enheten ställer krav på den i form av bland annat interoperabilitet med befintliga nätverk, säkerhet och tolerans mot yttre faktorer.

Interoperabilitet: System som Arthur™ och Giraffe™ är ofta monterade på ett tyngre fordon, till exempel en lastbil. Elektroniken i dessa fordon kommunicerar redan via CAN, men med en standard för tyngre lastbilar och terrängfordon kallad J1939, vilken skiljer sig från den CANopen standard man ska implementera i systemet. Det är därför önskvärt att en UCU har minst två CAN-gränssnitt så att styrning och övervakning kan göras av systemet och dess transportplattform. Det innebär att UCU agerar gateway i systemet.

Säkerhet: För systemet finns det krav på Information Assurance, IA, som begränsar lösningsalternativen. En utredning av aktuella protokoll behövs göras. För ingående elektronik kan Simple Network Management Protocol version 3, SNMPv3, vara en lösning om det kan uppfylla krav på IT-säkerhet.

Styrning: För att kunna styra datorer/datorkort kan Intelligent Platform Management Interface, IPMI, vara en lösning, men då gäller att alla datorer/datorkort har stöd för detta. Minst två Ethernet-gränssnitt behövs därför, ett för kommunikation med UCP och ett för styrning av datorer/datorkort.

Tolerans mot yttre faktorer: Då enheten kan komma att användas i stort skiftande miljöer krävs att den har hög tolerans mot yttre faktorer såsom temperatur och vibrationer. Exakta specifikationer ges av standarderna; MIL-STD-810C och MIL-STD-810F.

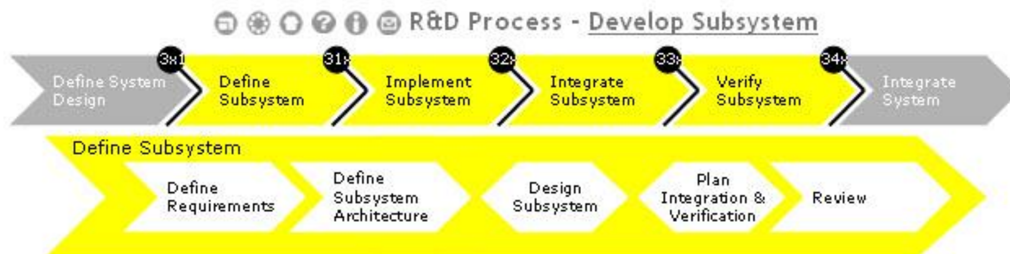
Övriga krav: Av enheten krävs att stöd finns för 28 V spänningsmatning, vilket ges av MIL-STD-1275D, men även stöd för mottagande av diskreta signaler och att en kort startup-tid kan erhållas.

1.4 Avgränsningar

Initialt övervägdes att från grunden utveckla helt egna hård- och mjukvarulösningar, såsom en egen Linux-distribution med tillhörande kärna och toolchain. Denna tanke övergavs till förmån för inköpt hårdvara och färdiga protokollstackar. Skälet till det var att den hårdvara som hittades ansågs gångbar, då de flesta krav uppfylldes och mjukvaran var tillräckligt avancerad för våra behov. Vidare var det från början tänkt att både CAN och Ethernet skulle behandlas, det visade sig dock tidigt att det skulle bli knappt om tid och all fokus lades på utvecklingen av CAN, vilket gjorde att Ethernet-aspekten lämnades till en möjlig fortsatt utveckling. I arbetet har vi även valt att inte lägga allt för stor tyngd på kraven på IT-säkerhet och tolerans mot yttre faktorer, utan först och främst sett till att en fungerande prototyp framställts. En annan begränsning som gjorts är att vi bara använder oss av två noder, en master och en slav. Enheten har dock utbyggnadsmöjligheter för många fler noder.

2 Metod

Forskning och utveckling kan på Saab beskrivas som en flerstegsprocess, Saab R&D process, se Figur 4, där varje steg är uppbyggt av flera underprocesser, med sina egna mål och delmål. Ett liknande tillvägagångssätt följdes under detta arbete.



Figur 4 *Processindelning*

2.1 Kravinsamling

Arbetet inleddes med flera möten med kollegor på Saab, där vi fick en mer ingående redogörelse om hur framtida system ska utformas och hur vår prototyp kommer att passa in i helheten. Tillsammans med de systemkrav och implementationsförutsättningar som specificerades togs en första skiss på designen fram.

2.2 Planering/tidsplan

Inledningsvis analyserades även vad/vilka delar som skulle ta mest tid i anspråk och en planeringsrapport, vilken användes till grund för det fortsatta arbetet, gjordes.

2.3 Analys

När planeringen av arbetet var gjord var nästa steg att bedöma huruvida det fanns lämplig hårdvara att köpa in eller om en egen konstruktion skulle tas fram. Vi var här tvungna att ta hänsyn till den funktionalitet vi ville uppnå och de krav som ställts. Parallellt med detta undersökte vi vilken typ av mjukvara som var lämplig att använda till CAN-buss miljön.

2.4 Design

Vi behövde någon form av industriell box-dator eller utvecklingskort att upprätta som master till en CAN-buss miljö samt som brygga till Ethernet. Utöver det var det nödvändigt med en CAN-till-USB-adapter för att kunna kommunicera med en CAN-buss miljö bestående av en eller flera slavar på en Linux-dator. Någon typ av CANopen protokollstack krävdes för att kunna bygga dessa master- och slavapplikationer. Det vore fördelaktigt om denna stöddes väl av hårdvaran och var förhållandevis billig. Det fanns ett antal olika alternativ på marknaden så vi tog beslutet att köpa in hårdvaran istället för att utveckla egen. Även en protokollstack av typen öppen källkod hittades. Blockschema över nätverket togs fram för att få en överblick hur systemet fysiskt skulle konfigureras. För att kunna styra signalerna i systemet togs även en switchpanel fram med tillhörande kablage.

2.5 Integration

Då produkterna levererats började vi installera allt vi behövde för att få upp en korrekt utvecklingsmiljö för båda sidor av CAN-buss miljön, det vill säga, master- och slavsidan. CAN-gränssnitten och bussen konfigurerades. Därefter kördes exempelprogram mellan enheterna och testkörning gjordes. Fokus lades sedan på att bygga vidare på dessa program och modifiera dem så att rätt funktionalitet kunde uppnås.

2.6 Demonstration

Verifiering av prototypens funktionalitet gjordes och verifierades med de krav man ställt på den. En manual som beskriver tillvägagångssättet för att få upp utvecklingsmiljö och köra prototypen skrevs, se kapitel 8.3.

2.7 Rapport

Rapportskrivandet påbörjades i mindre utsträckning när vi väntade på beställda produkter genom att vi strukturerade upp rapportens layout och påbörjade de inledande kapitlen. När produkterna levererades lades dock fokus på utveckling av hårdvara och mjukvara vilket gjorde att rapportskrivandet fick vänta. Parallellt med produktutvecklingen fördes dagbok/loggbok med anteckningar av olika designbeslut och implementationsförslag. Vid återupptagandet av rapportskrivandet hade vi stor nytta av dessa dagböcker och veckorapporter som kontinuerligt skrivits under arbetets gång.

2.8 Presentation

När arbetet kring rapporten var i sitt slutskede så inleddes förberedandet inför slutpresentationen. Presentationen genomförs med en muntlig framställan med stöd av en Power Point presentation

3 Teknisk bakgrund

Kommande underrubriker beskriver viktig och användbar information som ligger till grund för förståelse av senare delar av rapporten och arbetsuppgiften. Detta för att göra läsaren upplyst om termer och uttryck som används frekvent i senare delar av rapporten.

3.1 Open System Interconnection, OSI

OSI-modellen karakteriserar och kategoriserar de interna funktionerna hos ett kommunikationssystem. Modellens arkitektur består av sju lager och fungerar som en plattform för beskrivning av nätverksprotokoll. Varje lager tjänar det lager som ligger ovanför och blir tjänad av det lager som ligger under, se Figur 5. OSI-modellen är konstruerad på ett sådant sätt att även de mest komplexa nätverk kan beskrivas med den, dock använder sig många nätverk inte av alla sju lager. Då CAN-nätverk ofta är slutna och behandlar relativt små och enkla meddelandestrukturer med data från till exempel tryck- och temperaturgivare är endast de två lägsta lagren inkluderade i CAN-specifikationen, ISO-11898. (Microsoft Corp, 2002)

Figur 5 *OSI-modellens kommunikationsväg*

3.1.1 Lager 7, Applikationslagret

Applikationslagret är det högsta lagret och närmast användaren. Det kan ses som ett fönster som användare och applikationsprocesser öppnar för att få tillgång till nätverkstjänster. (Microsoft Corp, 2002)

3.1.2 Lager 6, Presentationslagret

Presentationslagret formaterar data som skall presenteras i applikationslagret. Här översätts data från formatet använt av applikationslagret till ett gemensamt format som kan användas av de lägre lagren och tvärtom. Detta lager tillhandahåller även kompression och kryptering av data. (Microsoft Corp, 2002)

3.1.3 Lager 5, Sessionslagret

Sessionslagret tillhandahåller information om hur sessioner upprättas och avslutas mellan processer som körs på olika stationer. (Microsoft Corp, 2002)

3.1.4 Lager 4, Transportlagret

Transportlagret ser till att hela meddelanden tas emot felfritt och i rätt sekvens utan databortfall eller dupliceringar. Storleken och komplexiteten på detta lager beror på hur tillförlitligt nätverkslagret under det är, ett opålitligt nätverkslager kräver ett transportprotokoll med stor feldetekteringsmöjlighet. Till skillnad från de lägre lagren vars protokoll berör omedelbart avgränsande noder så är transportlagret och de ovanstående lagren så kallade ”källa till destination”-lager. De är alltså endast bekymrade av kommunikationen från startnod till slutnod, inte de noder som passeras på vägen. (Microsoft Corp, 2002)

3.1.5 Lager 3, Nätverkslagret

Nätverkslagret styr driften för delnätet och bestämmer, baserat på bland annat nätverksförhållanden och prioritet, vilken väg skickad data skall ta. (Microsoft Corp, 2002)

3.1.6 Lager 2, Datalänklagret

Datalänklagret tillhandahåller felfri överföring av datapaket mellan noder på det fysiska lagret vilket gör att lagren ovanför kan anta en felfri informationsöverföring dem emellan. Här ligger bland annat funktioner såsom CSMA/CA vilka används av CAN. (Microsoft Corp, 2002)

3.1.7 Lager 1, Fysiska lagret

Fysiska lagret är modellens lägsta lager och behandlar sändning och mottagning av bitströmmar över fysiska delar av nätverket, såsom kablage. Det tillhandahåller bland annat funktioner och information rörande datakodning, hårdvaruanslutningar och sändningstekniker. (Microsoft Corp, 2002)

3.2 Controller Area Network, CAN

Controller Area Network, CAN, är ett seriellt bussystem ursprungligen utvecklat till fordonsapplikationer under tidigt 80-tal. Idag har i stort sett alla nyproducerade passagerarfordon i Europa minst ett CAN-nätverk. (CAN in Automation, CAN history, 2001)

Det var under tidigt 1980-tal som ingenjörer hos Bosch evaluerade möjligheten att använda seriella bussystem i passagerarfordon. Microprocessorer hade blivit så små och kraftfulla att de i allt större utsträckning användes i fordon och i takt med att fler och fler elektriska system introducerades ökade behovet av att på ett effektivt sätt få dem att kommunicera med varandra. (Mannisto & Dawson, 2003)

Då inga av de befintliga nätverksprotokollen ansågs uppfylla ingenjörernas krav fastslogs 1983 att man skulle utveckla ett helt nytt seriellt bussystem. Ett stort antal ingenjörer var involverade i utvecklingen, inte minst från biltillverkaren Mercedes-Benz och halvledartillverkaren Intel. I Detroit, under 1986 års Society of Automotive Engineers, SAE, kongress introducerades således vad man då kallade för ”Automotive Serial Controller Area Network” och dess multi-master nätverksprotokoll. Det sistnämnda byggde på en icke-destruktiv förlikningsmekanism som gav det meddelande med högst prioritet tillträde till bussen utan någon fördröjning. En extensiv felhanteringsmekanism hade även implementerats, bland annat en automatisk avkoppling av noder som uppvisade ett felaktigt beteende. När det kom till själva busskommunikationen stack CAN ut från de, på den tiden, tillgängliga bussystemen. Skickade meddelanden identifierades genom deras innehåll och inte avsändarens eller mottagarens adresser. Meddelandets identifierare specificerade även dess prioritet på bussen. Ett år senare levererade Intel det första CAN-kontroller-chipet, endast fyra år efter att idén kläckts. (CAN in Automation, *CAN history*, 2001)

CAN blev under ett tidigt skede mycket populärt i Nordeuropa och trots att man utvecklat det med inriktning på applikationer inom fordonsindustrin kom de första från helt andra marknadsområden. Det användes i allt från hissar till maskiner inom textil- och sjukvårdsindustrin. Trots att det inom dessa områden började dyka upp olika typer av mer eller mindre standardiserade högnivåprotokoll förhöll sig majoriteten av dessa CAN-pionjärer till ett monolitiskt tillvägagångssätt. Funktioner som berörde kommunikation, nätverkshantering och applikationer var en och samma mjukvara. Under tidigt 1990-tal gick användare och tillverkare därför ihop och etablerade en neutral plattform för teknisk förbättring av CAN och marknadsföringen av det seriella bussystemet. Det var början till vad som skulle komma till att bli CAN in Automation, CiA, en organisation av internationella användare och tillverkare, ledande i utveckling och stöd av CAN-baserade högnivålager. En av de första uppgifterna man åtog sig var specificeringen av ett CAN-applikationslager (CAL). Då CAN helt och hållet är en datalagerimplementation fanns inga standarder för hur utbytet av data skulle se ut på applikationsnivå. Tillvägagångssättet för utvecklingen av CAL var teoretiskt sett helt korrekt och resultatet användbart inom industriella applikationer men en stor nackdel var att varje användare blev tvungen att designa en ny kommunikationsprofil. Ett europeiskt konsortium lett av Bosch hade lösningen på problemet och utvecklade en prototyp som skulle bli känd som CANopen. (CAN in Automation, *CAN history*, 2001)

Applikationslagret, som senare kom att kallas ”Green Book” var bland det första man specificerade men utöver utvecklingen av specifikationer såsom denna var en av huvuduppgifterna att skapa ett informationsutbyte mellan experter och de som intresserade sig för CAN. En internationell konferens hålls därför årligen. (CAN in Automation, *CAN history*, 2001)

3.2.1 Varför CAN?

Det multi-master system som används av CAN skiljer sig från traditionella nätverk såsom USB och Ethernet. Istället för att sända stora datablock mellan noder under uppsikt av en central kontrollenhet så skickas flera korta meddelanden ut till alla noder på nätverket. Meddelanden är alltså inte riktade till någon specifik nod utan det

är noderna själva som avgör om det meddelande som ligger på bussen är av intresse eller inte. Det finns ingen unik nod utan alla är likvärdiga och kan, förutsatt att bussen är ledig, när som helst skicka information till vilka andra noder som helst. Alla meddelanden som skickas är tillgängliga för alla noder på nätverket men då varje meddelande har en unik identifierare väljer noden individuellt om informationen är relevant eller om den ska ignoreras. Om en nod skulle sluta fungera påverkar det inte systemet utan de fungerande noderna kan fortsätta sin kommunikation. Denna teknik leder till effektivare meddelandekontroll och reducerar drastiskt kabelåtgången. Service- och underhållsuppgifter förenklas då hårdvara kan tas bort, bytas ut eller läggas till utan att på ett negativt sätt påverka nätverket. (Mannisto & Dawson, 2003)

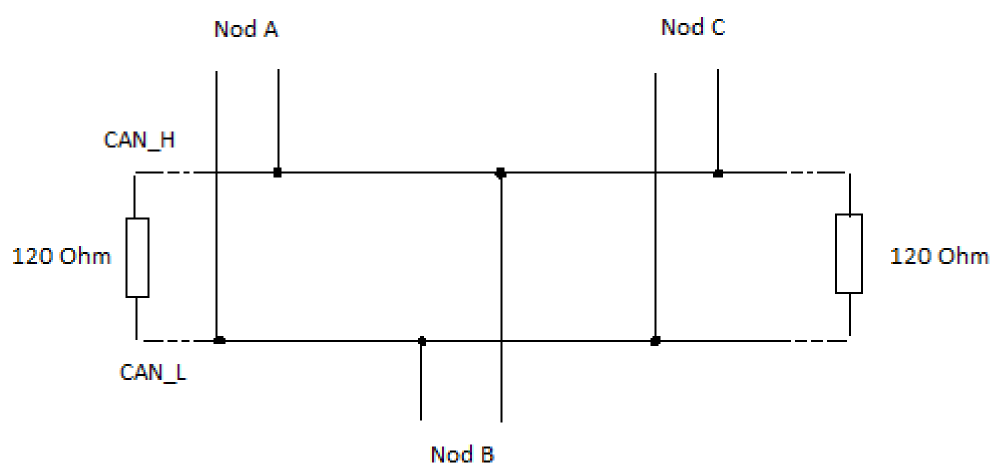
CAN-specifikationen, ISO-11898, beskriver hur information passerar mellan enheter på nätverket och använder sig av OSI-modellens två lägsta lager, datalänklagret och det fysiska lagret. Protokollet har vad man kallar ”*carrier-sense multiple access with collision avoidance*” förkortat CSMA/CA. I praktiken betyder det att alla noder måste vänta på en fördefinierad tid av bussinaktivitet innan de får sända ett meddelande och att eventuella busskonflikter löses genom bitvis arbitrering baserad på en fördefinierad prioritet av olika meddelandes identifierare. Det meddelande som har högst prioritet vinner tillträde till bussen. Ett meddelande som till exempel förmedlar en motors varvtal ges högre prioritet än ett som innehåller information om motors temperatur då temperaturen inte behöver uppdateras lika ofta. (Corrigan, 2008)

3.2.2 Protokoll

Ett protokoll är en uppsättning regler som rör kommunikationen på nätverket. Det specificerar bland annat parametrar såsom vilken typ av data som kan skickas, hur meddelanden identifieras, hur datapaket är uppbyggda och mycket mer. Ofta kan nätverkstekniker såsom CAN och Ethernet beskrivas med flera olika protokoll men som skiljer sig i någon eller några av parametrarna ovan. (Wildpackets Inc.)

3.2.3 Topologi (Fysiska Lagret)

Som kommunikationsnätverk är CAN mycket flexibelt då det endast implementerar delar av det fysiska lagret och datalänklagret. Ovanpå dessa lager kan sedan en mängd standardiserade högnivåprotokoll appliceras. En CAN-buss är ofta uppbyggd av ett tvinnat kabelpar som är terminerad med 120 Ω motstånd i båda ändar men det fysiska mediumet kan variera och vara av typen kraftledning, optisk fiber, med mera. Noder kopplas in på kablarna CAN_H respektive CAN_L (Hög/Låg), enligt Figur 6, och signalen utgörs av differensspänningen. (Pfeiffer, Ayre & Keydel, 2003)



Figur 6 *CAN buss*

En nod består av följande tre delar; *Processor, CAN controller* och *tranceiver*.

Host Processor

Nodens processor tolkar mottagna meddelanden men avgör även vilka meddelanden noden själv ska skicka. Till denna enhet kan sensorer, ställdon och styrenheter kopplas.

CAN controller

En nods kontroller har en inbyggd klocka och lagrar från bussen mottagna bitar i serie tills det att ett helt meddelande tagits emot och processorn kan hämta det. Vanligen sker detta efter det att kontrollern signalerat ett avbrott. Analogt för sändning, processorn lagrar meddelanden på kontrollern som sedan seriellt skickar bitarna ut på bussen.

Tranceiver (Transmitter/Receiver)

Anpassar signalnivåer från bussen till nivåer som förväntas av kontrollern och tvärtom. Den innehåller även kablage som skyddar kontrollern från överspänning.

Bussens läge beskrivs vanligen som antingen recessivt eller dominant där det förstnämnda inträffar när CAN_L och CAN_H har samma potential (2.5 V) och det sistnämnda när det är en potentialskillnad dem mellan (1.5 resp. 3.5 V). Ett recessivt läge på bussen motsvaras av CAN som en etta medans ett dominant läge motsvaras av en nolla. Den maximala hastigheten med vilken enheter på nätverket kan kommunicera med varandra beror av bussens fysiska längd och åskådliggörs i Figur 7 nedan. Långa kablar gör att det tar längre tid för en signal att färdas från en punkt i nätverket till en annan och tillbaka vilket resulterar i att bithastigheten måste sänkas för att undvika konflikt på bussen. När det gäller bithastighet benämns denna oftast i termer om tusentals bit per sekund (Kbps) eller miljoner bit per sekund (Mbps) och syftar på antal bitar som passerar en given punkt i nätverket per sekund. (Pfeiffer, Ayre & Keydel, 2003)

Bithastighet	Busslängd
1 Mbit/s	25 m
800 Kbit/s	50 m
500 Kbit/s	100 m
250 Kbit/s	250 m
125 Kbit/s	500 m
50 Kbit/s	1000 m

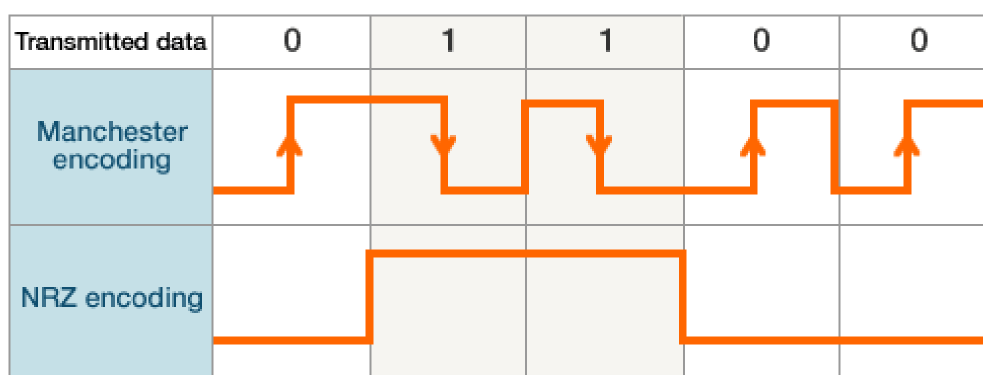
Figur 7 *Maximal bithastighet och busslängd (Pfeiffer, Ayre & Keydel, 2003)*

Som nämnts ovan är CAN ofta ett 2-kabel nätverk där endast CAN_H och CAN_L finns representerade. Beroende på applikation kan det dock behövas fler kablar/signaler såsom jord, spänningsmatning och sköldning. De kontakter som används kan variera och vara allt ifrån 9-pinnars D-Sub till 4-pinnars RJ10 och 8-pinnars RJ45. CAN är med andra ord inte särskilt kräset när det kommer till kontakter eller kablage, speciellt vid lägre hastigheter då toleransen är högre. (Pfeiffer, Ayre & Keydel, 2003)

3.2.4 Bitrepresentation (Fysiska Lagret)

Non-Return to Zero

CAN använder sig av bitkodningsmetoden NRZ, Non-Return to Zero. Denna metod kräver, till skillnad från bland annat Manchester-bitkodning, inte en signalövergång för att representera varje bit. Signalnivån ligger konstant över bittiden vilket gör att den för en sträng av ettor eller nollor kommer att hålla samma värde för så många bitar det behövs, se Figur 8. (Mannisto & Dawson, 2003)



Figur 8 *Manchester och NRZ (Sony Corp.)*

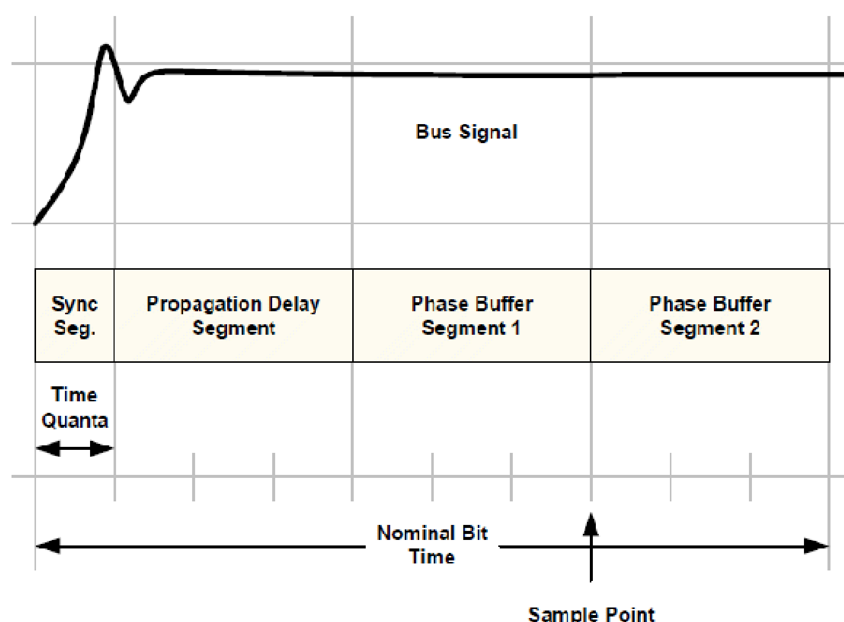
Med NRZ-kodning elimineras onödiga signalövergångar vilket gör att ett CAN-system kan kommunicera med nästan dubbla hastigheten för en given klockfrekvens jämfört med system som använder Manchester-kodning. Nackdelen är att signalen under långa tidsintervall kan ligga på ett konstant högt eller lågt värde vilket gör att nodernas interna klockor kan hamna ur synk. (Mannisto & Dawson, 2003)

Det kan leda till svårigheter i att veta när en bit slutar och nästa börjar om det är mer än två ettor eller nollor i rad. CAN-protokollet använder sig därför av en teknik som kallas *Bit Stuffing*. Den har funktionen att vid ett bitmönster med fem identiska bitar i rad sätta in en signalövergång, en så kallad *stuff bit*. Noder använder denna bit för att synkronisera sina klockor och då alla noder håller uppsikt efter bitmönster med fem identiska bitar i rad så ignoreras stuff-biten automatiskt av mottagaren. (Pfeiffer, Ayre & Keydel, 2003)

Bit-timing och Synkronisering

På det fysiska lagret använder sig CAN av synkron bitöverföring vilket ökar överföringskapaciteten men som i sin tur kräver sofistikerad bitsynkronisation. Noder i ett CAN-nätverk använder sig vanligen av två metoder för att synkronisera sina klockor, hårdsynkronisering och återsynkronisering. Hårdsynkronisering sker en gång vid en meddelandeöverföring, i början av ramen för ett nytt meddelande. Bussen är alltid i ett recessivt läge innan ett meddelande skickas så meddelanderamens första bit, SOF skickas alltid dominant. Noder på nätverket använder denna signalövergång för att synkronisera sina klockor men klarar ej av att behålla synkroniseringen genom ramens hela längd så det krävs kontinuerlig återsynkronisering. Detta sker inuti ramen för ett meddelande, varje gång bussen går från ett recessivt till ett dominant läge. (CAN in Automation, *CAN physical layer*, 2001)

Specificerat finns också en nominell bithastighet som indikerar antal bitar som skickas av en ideal överförare utan återsynkronisation. Alla noder i ett CAN-nätverk måste ha samma nominella bithastighet. Nominell bittid är den tiden som krävs för att skicka en enskild bit över nätverket och används för att säkerställa att alla noder samplar bussen vid rätt tillfälle för att avgöra om bussen är i ett recessivt eller dominant läge, se Figur 9. Denna bittid kan delas upp i segment där varje segment är indelat i mindre enheter, så kallade *time quanta*. (Mannisto & Dawson, 2003)



Figur 9 *Nominell bittid (Mannisto & Dawson, 2003)*

Den nominella bittiden har följande uppdelning:

Synchronization Segment är det första segmentet och det är här som signalövergången förväntas inträffa.
Propagation Delay Segment kompenserar för tiden det tar för signaler att färdas från en punkt i nätverket till en annan och elektriska komponenter att reagera på stimulus.
Phase Buffer Segment (1 och 2) används för att kompensera nodoscillatorers tendens att hamna ur synk. Dessa segment är justerbara och kan göras längre/kortare utifrån givet fel. Återsynkronisationen sker här.
Sample Point inträffar alltid mellan Phase buffer-segmenten och det är här som bussens läge samplas. (Mannisto & Dawson, 2003)

3.2.5 Arbitrering (Datalänklagret)

I OSI-modellens andra lager finns ett underliggande funktionslager kallat Media Access Control (MAC). Detta tillhandahåller kontrollmekanismer för adressering och åtkomst till noder inom nätverk med *multiple access*. Dessa mekanismer kan delas upp i två varianter; bestämd och slumpmässig åtkomstkontroll. Med bestämd åtkomstkontroll är tillträde till bussen satt innan en nod försöker nå den vilket garanterar att konflikt undviks. Detta involverar i de flesta fall en central kontrollenhet som styr åtkomsttilldelningen vilket ökar systemets sårbarhet då nätverket står och faller med denna. (Mannisto & Dawson, 2003)

Slumpmässig åtkomstkontroll bygger på att noder när som helst kan komma åt bussen så länge denna är i recessivt tillstånd. Vanligast är kontrollmetoder baserade på så kallad ”Carrier-Sense Multiple Access” (CSMA). Det betyder att alla noder lyssnar på bussen och de som har ett meddelande för sändning inväntar ett recessivt läge (bussen är ledig) innan de samtidigt börjar skicka sin data. Då endast en nod åt gången tillåts sända på bussen krävs åtgärder för att veta vilken nod som får prioritet. (Mannisto & Dawson, 2003)

Busskrockar kan nämligen ske trots att noder lyssnar på bussen för att försäkra sig om att ingen annan sänder något samtidigt då det alltid finns en liten fördröjning innan en nods bitar når en annan nod. (Mannisto & Dawson, 2003)

Carrier-Sense Multiple Access with Collision Detection, CSMA/CD

En vidareutveckling av ovanstående teknik är CSMA/CD där man lagt till en krock-detekteringsfunktion. Denna funktion tillåter meddelandekonflikt men ingriper när de inträffar. Som innan kommer varje nod att se om bussen är ledig innan de sänder men som sagt kan det, på grund av fördröjning i nätet, uppstå tillfällen där flera noder sänder samtidigt. Då dessa konstant lyssnar till vad som läggs på bussen upptäcks felet och alla noder avbryter sina sändningar. De väntar en individuellt slumpad tid och försöker därefter sända igen. Nackdelen med denna metod framträder när det är mycket tvister om vilken nod som ska sända då sändningar hela tiden avbryts vilket slösar bandbredd och skapar långa avbrott. (Mannisto & Dawson, 2003)

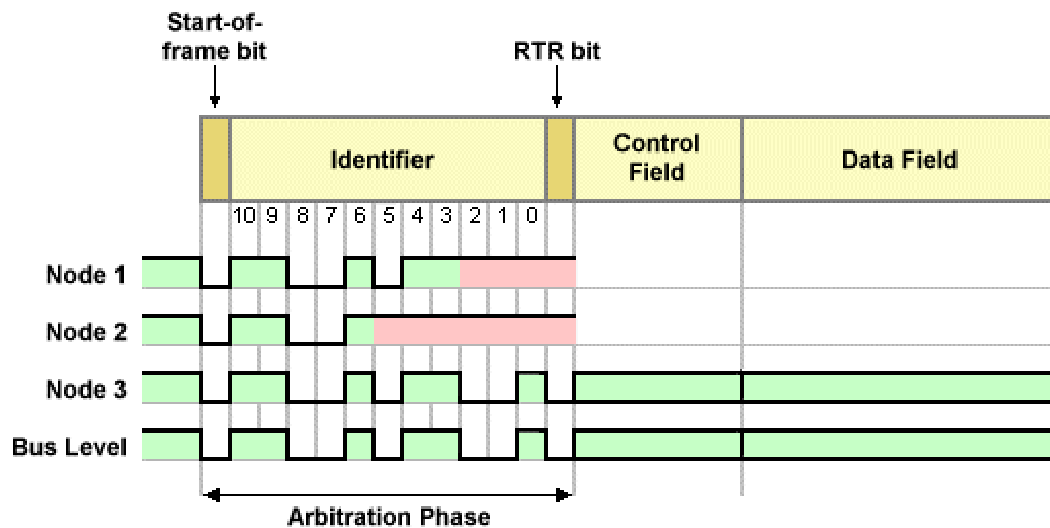
Carrier-Sense Multiple Access with Collision Avoidance, CSMA/CA

CAN-protokollet använder sig av en icke-destruktiv förlikningsmekanism, så kallad CSMA/CA. Principen är densamma som CSMA/CD med skillnaden att kollisioner undviks helt istället för att åtgärdas efter att de inträffat. Vidare är metoden icke-destruktiv, det vill säga, bussen kommer aldrig vara upptagen med datasändningar som konstant avbryts och återupptas. (Mannisto & Dawson, 2003)

Trafiken kontrolleras genom att ge meddelanden med hög prioritet tillträde till bussen före meddelanden med lägre prioritet. Som nämnts tidigare börjar alla CAN-meddelanden med ett arbitreringsfält som består av en SOF-bit, en 11 eller 29-bitars identifierare och en RTR-bit. Det är detta fält som identifierar meddelandet och bestämmer dess prioritet. Ett meddelande med en låg identifierare kommer oftare erhålla prioritet då identifieraren består av fler dominanta bitar (nollor). (Wells, 2001)

Arbitreringen sker då nodernas identifieringsfält sänds och börjar med den mest signifikanta biten, bit 10 för ett 11-bitars ID. En binär nolla ses som en dominant bit vilken alltid skriver över en binär etta, vilken ses som en recessiv bit. Detta medför att bussens läge alltid kommer reflektera det meddelande ID som har högst prioritet, se Figur 10. Skulle en nod upptäcka att bussen håller ett värde som inte stämmer överens med det noden skickar, avbryter den omedelbart sin arbitreringsprocess och väntar till dess att bussen är ledig innan den på nytt försöker sända sitt meddelande. På detta sätt erhåller meddelandet med högst prioritet rätten till fortsatt sändning, helt obehindrat

och utan fördröjning. Arbitreringsprocessen illustreras i bilden ovan, där nod 1 har prioritet över nod 2 men där nod 3 har prioritet över dem båda. Bussens innehåll speglar således nod 3. (Wells, 2001)



Figur 10 *Arbitrering (Wells, 2001)*

3.2.6 Meddelandeformat (Datalänklagret)

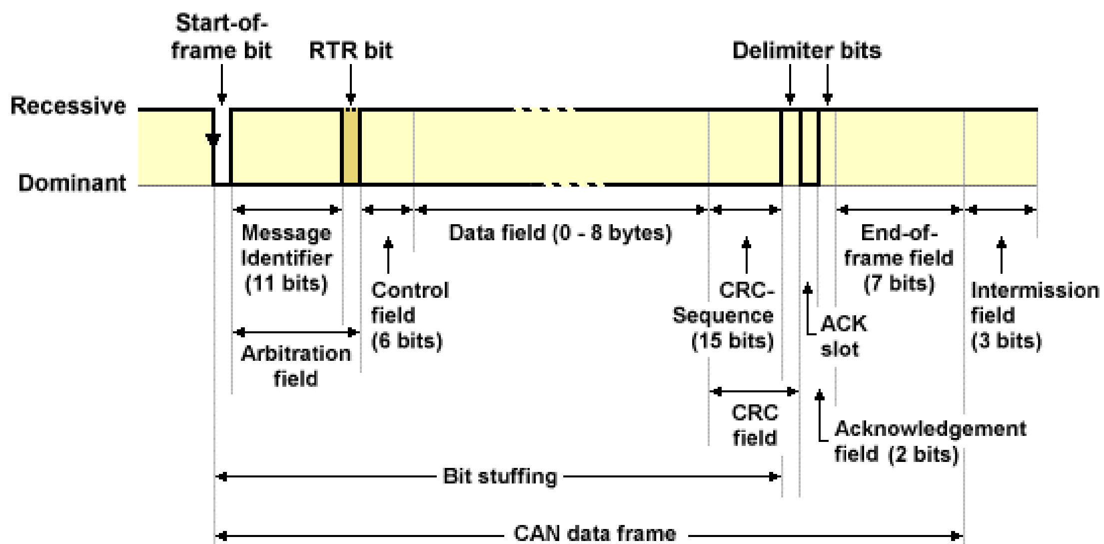
Ett CAN-system skickar data med hjälp av så kallade ”meddelanderamar” (*Frames)*. En meddelanderam kan ses som ett paket med information som innehåller ett komplett meddelande från en sändare. Det finns fyra olika typer av meddelanderamar:

- *Data Frame.* Används för överföring av data på nätverket
- *Remote Frame.* Skickas vid begäran av data från en nod till en annan.
- *Error Frame.* Skickas av mottagare om fel upptäckts i ett meddelande. Säger åt sändaren att skicka meddelandet igen. Kan skickas som aktiv eller passiv beroende på nodens tillstånd.
- *Overload Frame.* Skickas av mottagare för att be sändaren fördröja nästa meddelande. (Mannisto & Dawson, 2003)

Dessa meddelanden kan ha ett av två olika format beroende på vilket protokoll man använder sig av. I huvudsak skiljer de sig endast på längden av identifieraren som används. CAN version 2.0A (*Base Frame Format*) har en 11-bitars identifierare

medan version 2.0B (*Extended Frame Format*) har stöd för både 11 och 29-bitars identifierare. (Wells, 2001)

Ett generellt CAN meddelande börjar med en startbit (SOF) och följs sedan av ett arbitreringsfält som består av identifieraren och en Remote Transmission Request (RTR) bit, se Figur 11. Den sistnämnda skiljer på vad som är skickad data och vad som är en förfrågan om data. Därefter kommer ett kontrollfält som innehåller en IDentifier Extension (IDE) bit, för att göra skillnad på basformat och förlängt format och Data Length Code (DLC) som indikerar antal bytes som följer i Data-fältet. Skulle det aktuella meddelandet vara en förfrågan av information så innehåller detta fält antal efterfrågade data-bytes. Cyclic Redundancy Check (CRC) fältet innehåller en kontrollsumma som möjliggör utförandet av en integritetsförsäkran hos mottagaren. Fältet ACK står för Acknowledge och skickas alltid som en recessiv bit men skrivs över som en dominant bit av mottagaren om denne på ett korrekt vis mottagit meddelandet. För att indikera meddelandets slut används en End Of Frame (EOF) bit. Slutligen finns det en Intermission Frame Space (IFS) bit som indikerar det minsta antal bitar som skiljer fortlöpande meddelanden åt. (CAN in Automation, *CAN protocol*, 2001)



Figur 11 *CAN meddelanderam (Wells, 2001)*

3.2.7 Felhantering (Datalänklagret)

Datalänklagret i CAN-system har mycket effektiva mekanismer för detektering och hantering av fel:

Bit Check, görs av nod vid egen sändning och bygger på att noden övervakar sin sändning. Upptäcks en recessiv bit när en dominant förväntades så signaleras bitfel.

Frame Check, de olika meddelanderamarna innehåller fält som alltid håller samma värde, såsom SOF och EOF. Noder kontrollerar dessa värden för varje meddelande de tar emot, skulle ett fel upptäckas signaleras format-fel (*Form Error*).

Cyclic Redundancy Check, applicerar en polynomekvation på det datablock som ska skickas och det beräknade resultatet placeras i meddelandets CRC-fält. Mottagande nod utför samma beräkning och jämför resultatet med det i CRC-fältet, upptäcks ett fel skickas ett felmeddelande till sändaren.

Acknowledgement Check, de noder som skickat ett meddelande lyssnar efter ett bekräftelsemeddelande från minst en nod i nätverket. Tas inget sådant emot ses det som ett bekräftelsefel (*Acknowledgement error*) och noden kommer att fortsätta skicka sitt meddelande tills det att bekräftelse tagits emot.

Stuff Rule Check, noder håller aktivt koll på överträdelser av regeln mot mer än fem identiska bitar i rad. Skulle det upptäckas skickas ett felmeddelande till sändaren.

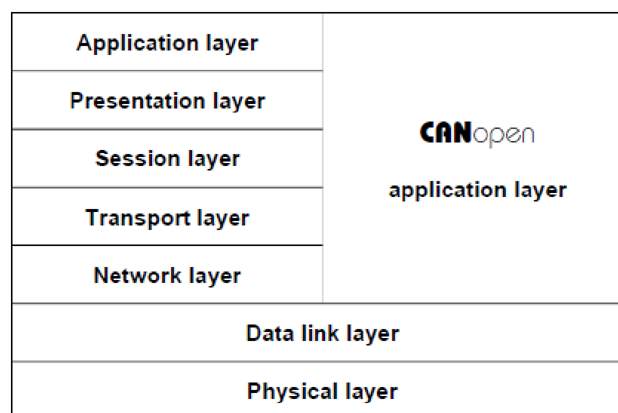
För att undvika att en defekt nod blockerar kommunikationen på hela nätverket har CAN designats för att automatiskt upptäcka felande noder och snabbt koppla bort dem. Alla noder har därför två felräknare, en för sändningsfel och en för mottagningsfel. Dessa räknas upp för respektive fel men ökningen per fel beror av vilken typ av fel det handlar om. En nod som ser att den är källan till ett fel ökar motsvarande räknare med 8 om det är sändningsfel och 9 om det är mottagningsfel. Upptäcks fel men källan inte är noden själv så ökas respektive räknare med 1. För varje lyckad sändning eller mottagning minskar respektive räknare med 1.

Dessa räknare placerar en nod i ett av tre tillstånd; ***aktiv***, ***passiv*** eller ***av***. En *aktiv* nod fungerar som den ska och får skicka felmeddelande till alla andra noder på nätverket. *Passiva* noder har en felräknare med ett värde som överstiger 127, de kan skicka och ta emot meddelanden men har bara tillåtelse att skicka passiva felmeddelanden om ett fel upptäcks. Skulle en *passiv* nod sända ett meddelande som innehåller fel blir den tvungen att vänta 8 bittider innan den får sända igen. För att en nod ska försättas i tillståndet *av* måste dess räknare för sändningsfel ha överstigit 255, det spelar ingen roll hur stort antal mottagningsfel noden har. En nod i detta tillstånd får inte på något sätt påverka bussen men kan fortfarande ta emot meddelanden. Denna respons på fel hjälper till att hålla nätverket i funktionsdugligt skick trots att en eller flera noder upphört att fungera normalt. Tack vare det faktum att noder kan återställa sig själva efter flera fel (minska sina räknare genom korrekt meddelandetransmission) har systemet goda återhämtningsmöjligheter. (Mannisto & Dawson, 2003)

3.3 CANopen

Det var ett europeiskt konsortium lett av Bosch som utvecklade en ny prototyp av ett CAN-baserat applikationslager som skulle komma till att bli CANopen. Efter färdigställandet skickades det till CiA för underhåll som år 1995 publicerade en helt reviderad kommunikationsprofil för CANopen. I standarden, CiA 301 specificeras den grundläggande CANopen-enheten och kommunikationsprofiler medan mer avancerade enheter och kommunikationsprofiler är skapade utifrån denna grundläggande profil och specificeras i andra, högre, standarder. Dessa standarder används med fördel som mallar vid skapande och anpassning av egen CANopen enhet. CANopen har mycket flexibla konfigureringsmöjligheter och lämpar sig väl till inbyggda nätverk i all form av maskinkontroll och det blev mycket använt under sent 1990-tal då hela industrisegment adopterade lösningen. (CAN in Automation CAN history) (RT-Labs AB 2009)

CANopen är en kommunikationsprotokollstack och enhetsprofilspecifikation, vilken representerar de högre lagren i OSI-modellen, ned till nätverkslagret, se Figur 12. De lägre lagren, datalänklagret och det fysiska lagret, är nästan alltid representerade av CAN. CANopen består av ett adresseringsschema, flera mindre kommunikationsprotokoll, en objektlista och ett applikationslager. (RT-Labs AB 2009) (Wikipedia 2013 CANopen)

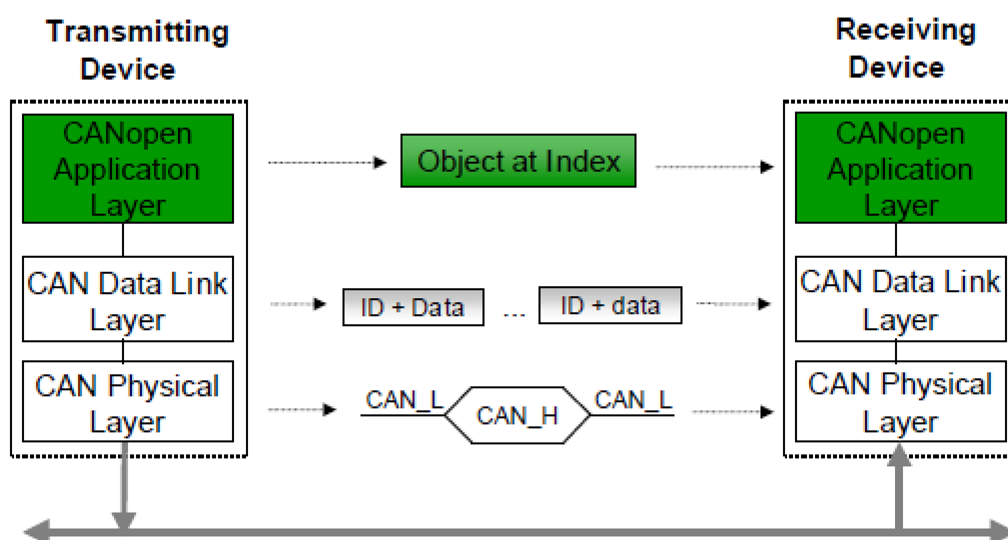


Figur 12 *CANopen sett från OSI-modellen (CAN in Automation 2002 CANopen Application Layer and Communication Profile)*

Kommunikationsprotokollstacken har som syfte att hantera de inkommande och utgående meddelandena, ansvara för enhetens signaler och dessutom sköta det logiska flödet hos enheten. Det finns en protokollstack i varje enhet och denna protokollstack kan anpassas för just den enhetens uppgift, men huvuddelen av protokollstacken är ändå likadan för samtliga enheter. (RT-Labs AB 2009) (Wikipedia 2013 *CANopen*)

3.3.1 Struktur

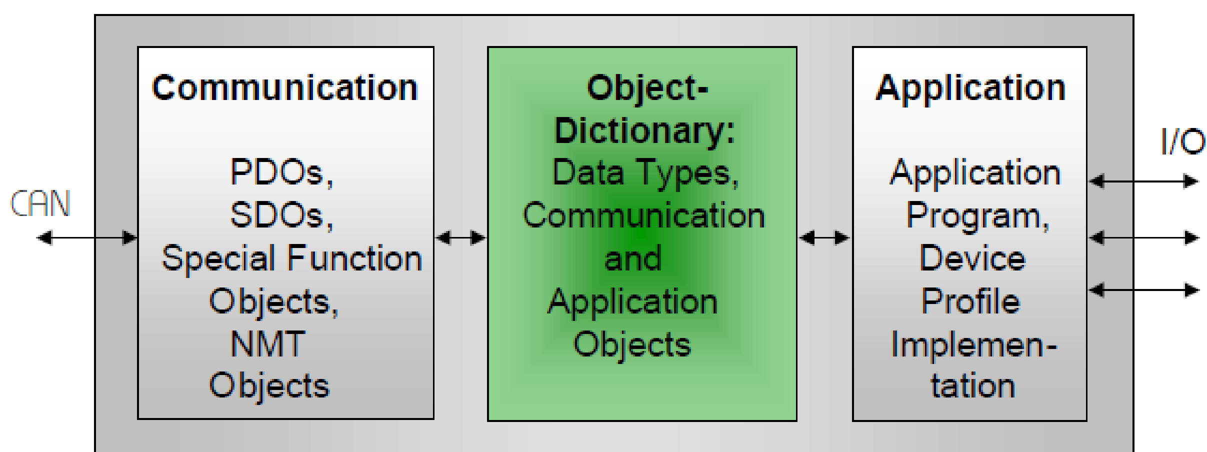
Kommunikationskonceptet bygger ju som sagt på OSI-modellen, där CANopen representerar de högre lagren med ett applikationslager och kommunikationsprofil och där de lägre lagren oftast representeras av CAN. Kommunikationsvägen mellan två enheter sker via de olika lagren, från applikationslagret på den sändande enheten ner till bussen och sedan vidare från bussen upp till applikationslagret på den mottagande enheten, enligt Figur 13. Kommunikationen som bildas mellan de olika lagren verkställs på olika sätt. På applikationslagret utbytes kommunikations- och applikationsobjekt. Dessa kan kommas åt från objektlistan via ett 16-bit index och ett 8-bit delindex. Mellan datalänklagren skickas information med nod-ID och data. Kommunikationen på det fysiska lagret specificerar bitnivå och bit-tidstyrning. (Pfeiffer, Ayre & Keydel, 2003) (CAN in Automation *CANopen*)



Figur 13 *Datagången mellan de olika lagren (CAN in Automation CANopen)*

En CANopen-enhet kan beskrivas i tre delar, vilka är kommunikation, objektlistan och applikation, se Figur 14. Utöver detta så behövs en tillståndsmaskin. Denna tillståndsmaskin ska innehålla tillstånden ”initiering”, ”före driftläge”, ”driftläge” och ”stoppad”.

Kommunikationsdelen innehåller flera olika kommunikationsprotokoll som bland annat används för att skicka och ta emot data, konfigurera objektlistan och ändra tillstånd i tillståndsmaskinen. Var och en av dessa måste stödjas av varje enhet i ett CANopen-nätverk. I kommunikationsdelen finns även passande funktionalitet för att tolka och transportera data via de lägre lagren. Objektlistan knyter samman de tre delarna med varandra och innehåller information om enheten. I objektlistan finns information som har inflytelse på enhetens kommunikations- och applikationsobjekt. Applikationsdelen utför den önskade funktionen hos enheten då tillståndsmaskinen är i sitt ”driftläge”. Det är applikationsdelen som tillhandahåller de eventuella processignalerna. (RT-Labs AB 2009) (Wikipedia 2013 *CANopen*) (CAN in Automation 2002 *CANopen Application Layer and Communication Profile*)



Figur 14 *CANopen enhetsmodell (CAN in Automation CANopen)*

3.3.2 Objektlistan

Den viktigaste delen i protokollstacken är objektlistan. Den är själva kärnan av varje CANopen-nod. Objektlistan fungerar som en uppslagstabell, där varje objekt adresseras genom ett 16-bit index och ett 8-bit delindex. Objekten kan kommas åt via att mottaga och sända Service Data Objects, SDO. Delar av objektlistan kan även mappas så att den kan kommas åt med Process Data Objects, PDO. Strukturen av objektlistan ges av att de 16-bitars indexen är indelade i olika sektioner, enligt Figur 15. (Pfeiffer, Ayre & Keydel, 2003) (CAN in Automation *CANopen*) (CAN in Automation 2002 *CANopen Application Layer and Communication Profile DS301*)

Indexområde	Beskrivning
0000h	Reserverad
0001h – 0FFFh	Datatyper
1000h – 1FFFh	Kommunikationsposter
2000h – 5FFFh	Tillverkarspecifik
6000h – 9FFFh	Enhetsprofilparametrar
A000h – FFFFh	Reserverad

Figur 15 *Objektlistans struktur*

Sektionen i indexområdet 0001h-0FFFh används för att definiera olika datatyper. Det finns två olika klasser av datatyper, standard och komplex. Standard innehåller definitioner för de vanliga datatyperna, såsom boolean, integer, sträng m.m. De komplexa datatyperna består av en eller flera ihopsatta standard-datatyper. Datatyperna definieras också olika beroende på om de är gemensamma för alla CANopen-enheter eller om de är specifika för en viss CANopen-enhet. (Pfeiffer, Ayre & Keydel, 2003) (CAN in Automation *CANopen*)

Nästa sektion i indexområdet 1000h-1FFFh innehåller kommunikationsparametrar som beskriver det mesta om hur kommunikationen sker hos noden. Här finns information om hur kommunikation ska ske med SDO, PDO, Network Management samt Special Object. Förutom detta så finns övrig information om enheten och PDO mappningen till den tillverkarspecifika sektionen här. (Pfeiffer, Ayre & Keydel, 2003) (CAN in Automation *CANopen*)

Den tillverkarspecifika sektionen i indexområdet 2000h-5FFFh används av applikationsdelen framförallt för processignaler. Med hjälp av mappning kan processignalerna hanteras med PDO. (Pfeiffer, Ayre & Keydel, 2003)

I nästa sektion i indexområdet 6000h-9FFFh definieras bland annat variablerna för de processignaler en enhet använder, men även dess standardkonfiguration och kommunikationsinställningar. (Pfeiffer, Ayre & Keydel, 2003)

Objektlistans objekt adresseras som sagt med ett 16-bit index och ett 8-bit delindex. Konceptet bygger på att objektlistans poster först adresseras med 16-bit index, där posten kan innehålla en enkel variabel, ett register eller en tabell. När det handlar om register eller tabell, så används 8-bit delindex för att komma åt de enskilda variablerna i tabellen. Överst i tabellen anges antal poster i tabellen. Figur 16 är ett exempel på hur det skulle kunna se ut vid indexet 2001, som tillhör den tillverkarspecifika sektionen, där applikationsprogrammets processignaler lagras. (Pfeiffer, Ayre & Keydel, 2003) (CAN in Automation *CANopen*)

Index	Delindex	Variabel	Datatyp	Värde
2001h	0h	Antal poster	Unsigned8	3
2001h	1h	Räknare	Unsigned16	50000
2001h	2h	Temperatur	Unsigned8	30
2001h	3h	Bränsle	Unsigned8	10

Figur 16 *Objektlistan index och delindex koncept*

3.3.3 Adressering

I ett CAN-meddelande ingår det en identifierare i varje meddelande. Identifieraren brukar benämnas som COB-ID eller CAN-ID. CANopen använder sig av en 11-bit identifierare, som är uppdelad i två delar; funktionskod och nod-ID. De fyra mest signifikanta bitarna är funktionskoden, vilket ger möjlighet till 16 olika funktionstyper, se Figur 17. Det är funktionskoden som definierar vad det är för typ av meddelande. De resterande sju bitarna är nod-ID, som används för adressering av noder. Det finns maximalt stöd för 127 enheter på en buss med CANopen och dessa

adresseras med nod-ID 1-127 medan nod-ID 0 adresserar samtliga enheter. (Pfeiffer, Ayre & Keydel, 2003) (CAN in Automation CANopen)

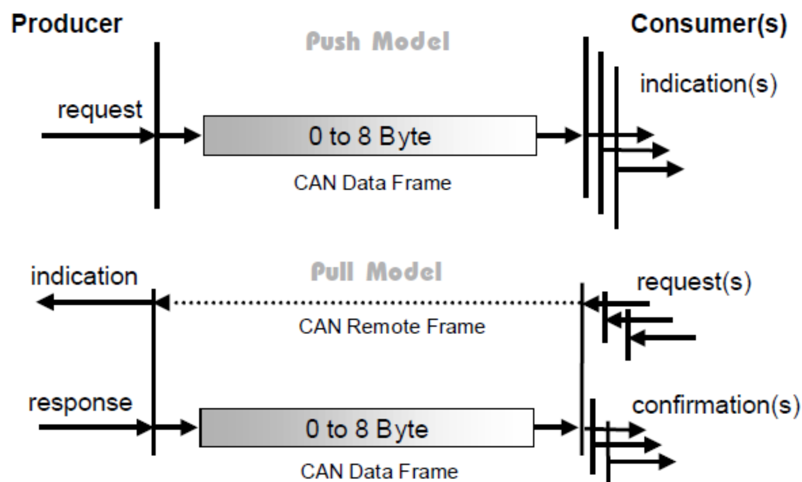
Meddelandetyp	Funktionskod Binär	Resulterande COB-ID/CAN-ID	Kommunikationsparametrar vid index
NMT	0000	0	-
SYNC	0001	80h	1005h, 1006h, 1007h
Emergency	0001	81h – FFh	1014, 1015h
TIME STAMP	0010	100h	1012, 1013h
PDO1 (tx)	0011	181h – 1FFh	1800h
PDO1 (rx)	0100	201h – 27Fh	1400h
PDO2 (tx)	0101	281h – 2FFh	1801h
PDO2 (rx)	0110	301h – 37Fh	1401h
PDO3 (tx)	0111	381h – 3FFh	1802h
PDO3 (rx)	1000	401h – 47Fh	1402h
PDO4 (tx)	1001	481h – 4FFh	1803h
PDO4 (rx)	1010	501h – 57Fh	1403h
SDO (tx)	1011	581h – 5FFh	1200h
SDO (rx)	1100	601h – 67Fh	1200h
NMT Error Control	1110	701h – 77Fh	1016h, 1017h

Figur 17 *Meddelandetypernas struktur*

Kommunikationen med CAN baseras på prioritet, där det är identifieraren som avgör hur hög prioritet ett meddelande har. Ju lägre värde på identifieraren desto högre prioritet har meddelandet. Det är därmed först och främst funktionskoden som avgör prioriteten för ett meddelande, eftersom den utgörs av de fyra mest signifikanta bitarna. Skulle däremot två likadana meddelandetyper skickas samtidigt är det nod-ID som avgör, där nod-ID 0 har högst prioritet och nod-ID 127 har lägst. Det finns således två olika prioritetskategorier att använda sig av, en efter meddelandetyp och en efter enhetstyp. (Pfeiffer, Ayre & Keydel, 2003) (CAN in Automation *CANopen*)

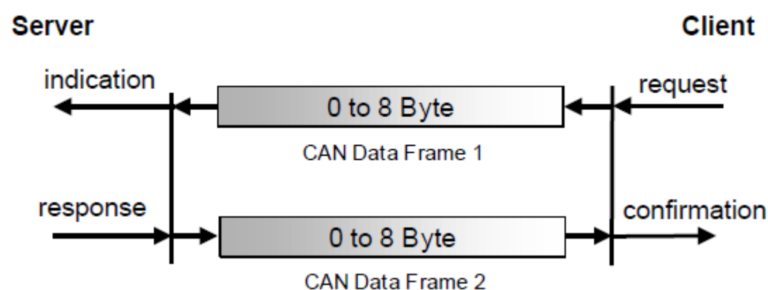
3.3.4 Kommunikationsmodeller

Modellen Producer/Consumer, se Figur 18, fungerar på så sätt att varje enhet på nätverket lyssnar på den sändande enheten och sedan själv bestämmer om den ska ta emot meddelandet eller inte. Det behövs således någon form av accepteringsfilter i enheten. Med denna modell kan man både sända meddelanden enligt push-modellen och begära ett meddelande enligt pull-modellen. (Pfeiffer, Ayre & Keydel, 2003) (CAN in Automation. *CANopen*)



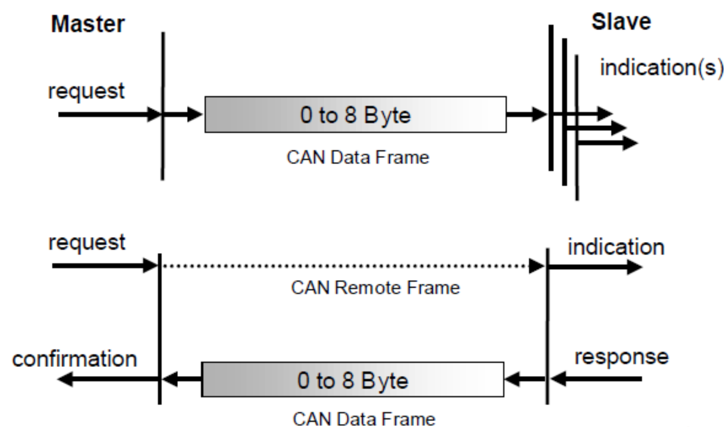
Figur 18 *Producer/Consumer-modell (CAN in Automation CANopen)*

Tekniken för Server/Client-modellen, se Figur 19, är att klienten överför ett meddelande till servern, som sedan svarar klienten med en bekräftelse. Denna modell används mycket vid överföring av stora mängder data, då man kan överföra data man vill sända segment för segment. Modellen inkluderar upp- och nedladdning samt överföringsavbrytning. (Pfeiffer, Ayre & Keydel, 2003) (CAN in Automation *CANopen*)



Figur 19 *Server/Client-modell (CAN in Automation CANopen)*

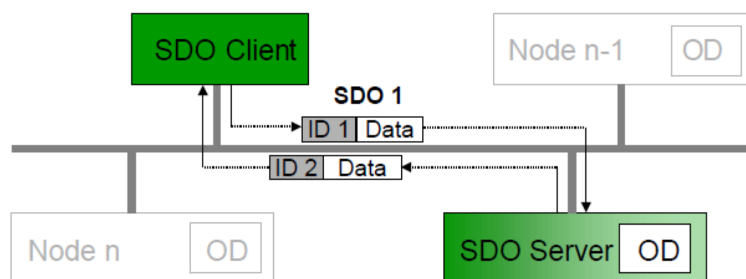
Master/Slave-modellen, se Figur 20, bygger på att endast kommunikation initierad från mastern är tillåten. Det är alltså helt och hållet mastern som sköter kommunikationen och kan både sända data till slaven och begära data från slaven. Slaven måste alltså alltid få en begäran från mastern innan den kan sända. (Pfeiffer, Ayre & Keydel, 2003) (CAN in Automation *CANopen*)



Figur 20 *Master/Slave-modell (CAN in Automation CANopen)*

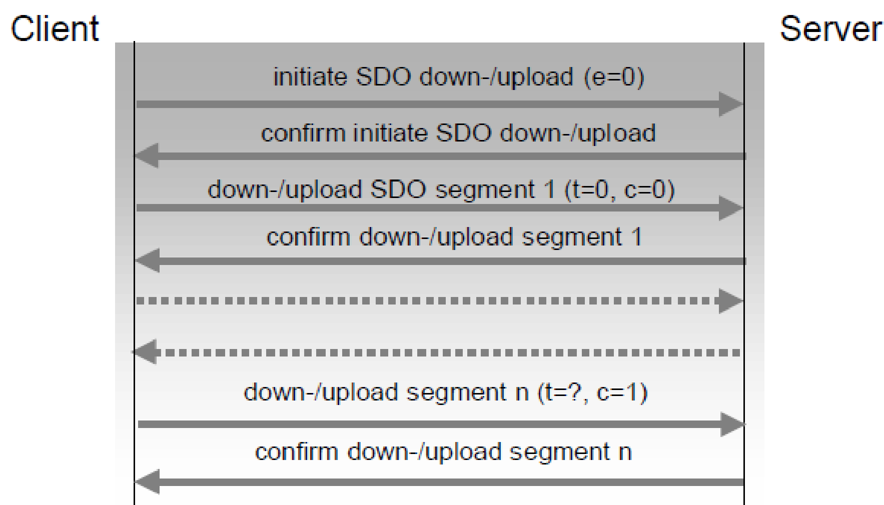
3.3.5 Service Data Object Protocol, SDO

SDO protokollet använder sig av en struktur enligt Server/Client-modellen och gör det möjligt att få tillgång till alla poster i en enhets objektlista. SDO används framförallt för att konfigurera en enhets objektlista. Vid kommunikation med SDO skickas alltid mottagningsbekräftelse, vilket gör att en SDO använder sig av två stycken dataramar med olika identifierare. Den som tillhandahåller objektlistan av enheterna i den upprättade kommunikationen är server och den andra är klient, se Figur 21. (Pfeiffer, Ayre & Keydel, 2003) (CAN in Automation *CANopen*)



Figur 21 *SDO-struktur i ett nätverk (CAN in Automation CANopen)*

Med SDO kan data av alla storlekar överföras, men man använder sig av olika metoder beroende på storleken. Om datalängden är högst fyra bytes används en påskyndad överföring med Initiate Down/Upload protokollen genom att dataöverföringen görs samtidigt som initieringen. Om datalängden är mer än fyra bytes måste däremot överföringen göras segment för segment, det vill säga, data delas upp över flera CAN-meddelanden, se Figur 22. Efter att initiering gjorts med det första meddelandet kan de följande meddelandena innehålla sju bytes av användbar data. När sista segmentet överförs indikeras detta genom en slutindikator i meddelandet. Initiativ till en överföring görs alltid av klienten och det är som sagt servern som tillhandahåller objektlistan, dock kan både klient och server avbryta överföringen. En annan metod för överföring av väldigt stora datamängder med SDO är att överföra en sekvens av block. Ett block kan innehålla en sekvens av upp till 127 segment. En fördel med detta är att varje block bara bekräftas en gång. I exemplet nedan visas hur en segmenterad SDO-överföring kan gå till. I initiering sätts e till 0, för att definiera att det är just en segmenterad överföring som ska ske och inte en påskyndad överföring. En växelbit, t, används för att försäkra sig om att ett meddelande inte läses av två gånger. Dessutom sätts slutindikatorn, c, till 1 vid sista överföringen för att definiera ett avslut på överföringen. (Pfeiffer, Ayre & Keydel, 2003) (CAN in Automation *CANopen*)



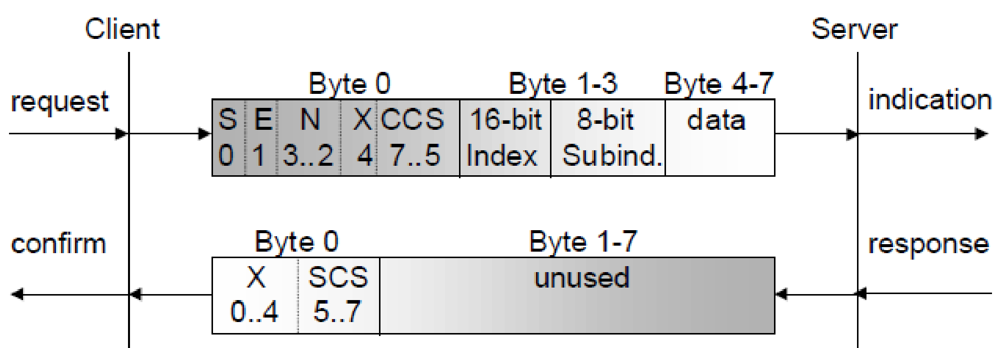
Figur 22 *Segmenterad SDO-överföring (CAN in Automation CANopen)*

Vid alla överföringsmetoder ingår alltid en kommandospecificerare, både för klienten, CCS, och för servern, SCS. Dessa definierar följande:

- Hämtning/Uppladdning
- Begäran/Svar eller bekräftan
- Segmenterad/Block/Påskyndad överföring
- Antal bytes
- Slutindikator
- Växelbit för varje följande segmentmeddelande

Vid användandet av SDO för skrivning och läsning i objektlistan används alltid Initiate Down/Upload protokollen, se Figur 23, först oavsett om det handlar om en påskyndad överföring, segmenterad överföring eller blocköverföring. Platsen i objektlistan där data ska hämtas är adresserad med ett 16-bit index och ett 8-bit index. Vid påskyndad överföring placeras data i datafältet i slutet av meddelandet och vid de andra två metoderna används meddelandet enbart till initiering för resten av överföringen. Meddelandet svaras med ett bekräftelsemeddelande. (Pfeiffer, Ayre & Keydel, 2003) (CAN in Automation *CANopen*)

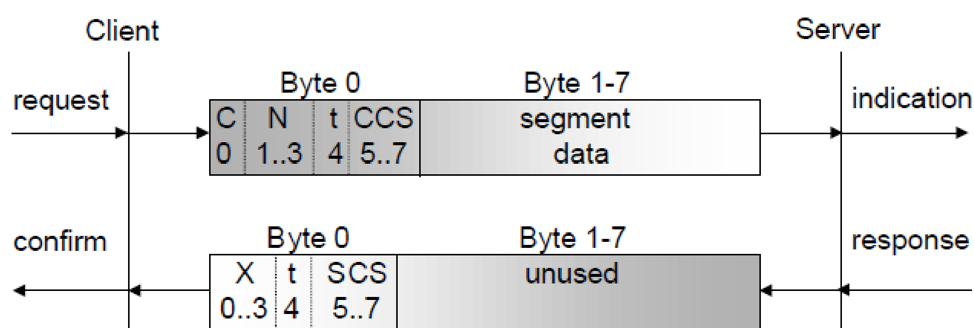
- S: Indikerar blockstorlek
- E: Indikerar påskyndad överföring
- N: Indikerar antal bytes
- X: Oanvända bitar
- CCS: Client Command Specifier
- SCS: Server Command Specifier



Figur 23 *Påskyndad SDO-överföring vid uppladdning (CAN in Automation CANopen)*

Vid segmenterad överföring börjar segment av data att överföras mellan klient och server efter att initiering gjorts, se Figur 24. Varje segment bekräftas med ett bekräftelsemeddelande. (Pfeiffer, Ayre & Keydel, 2003) (CAN in Automation *CANopen*)

- C: Indikerar sista segmentet
- N: Indikerar antal bytes
- T: Växelbit
- X: Oanvända bitar
- CCS: Client Command Specifier
- SCS: Server Command Specifier



Figur 24 *Segmenterad SDO-överföring vid uppladdning (CAN in Automation CANopen)*

Meddelandeparametrar för SDO definieras vid index 22h i objektlistan, se Figur 25, som används för att ange vilket COB-ID en klient-till-server-kommunikation har och vilket COB-ID en server-till-klient-kommunikation har samt nod-ID. SDO beskrivs sedan med hjälp av de definierade meddelandeparameterna med start vid 1200h för SDO-server och vid 1280h för klienten, enligt Figur 26. Det finns kapacitet för upp till 256 stycken SDO-kommunikationskanaler i ett enda CANopen-nätverk. (Pfeiffer, Ayre & Keydel, 2003) (CAN in Automation *CANopen*)

Index	Sub-Index	Description	Data Type
0022h	0h	number of entries	Unsigned8
	1h	COB-ID client -> server	Unsigned32
	2h	COB-ID client <- server	Unsigned32
	3h	node ID	Unsigned8

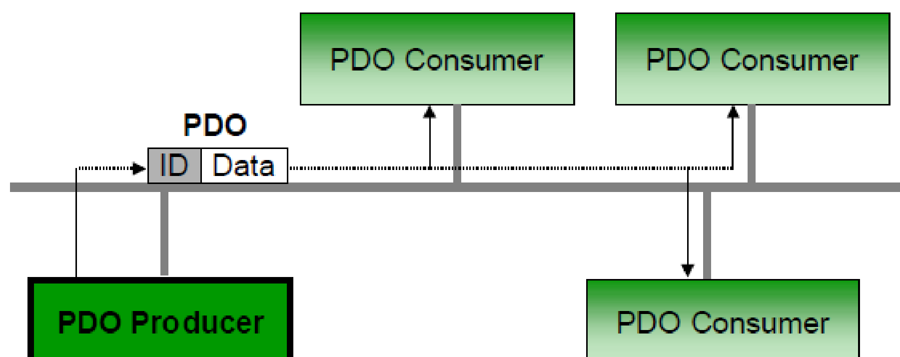
Figur 25 *Definition av SDO-parametrar (CAN in Automation CANopen)*

Server SDO Parameter (22H)					
1200	RECORD	1st Server SDO parameter	SDOParameter	ro	O
1201	RECORD	2nd Server SDO parameter	SDOParameter	rw	M/O**
.....	.....	.....	.....	.....	.....
127F	RECORD	128th Server SDO parameter	SDOParameter	rw	M/O**
Client SDO Parameter (22H)					
1280	RECORD	1st Client SDO parameter	SDOParameter	rw	M/O**
1281	RECORD	2nd Client SDO parameter	SDOParameter	rw	M/O**
.....	.....	.....	.....	.....	.....
12FF	RECORD	128th Server SDO parameter	SDOParameter	rw	M/O**
1300		reserved			
.....	.....	.....	.....	.....	.....
13FF		reserved			

Figur 26 *Objektlistans struktur med avseende på SDO (CAN in Automation CANopen)*

3.3.6 Process Data Object Protocol, PDO

CANopen har definierat PDO, som är ett mycket snabbare sätt att komma åt processignalerna än att använda sig av SDO. Kommunikation med PDO tillämpas enligt Producer/Consumer modellen, där processdata kan skickas från en enhet som agerar producent till en eller flera andra enheter som agerar konsument, enligt Figur 27. Man skiljer mellan skrivning av en PDO och läsning av en PDO, där Transmit PDO, TPDO, används för skrivning och Receive PDO, RPDO, används för läsning. Det är producenten som skickar en TPDO med en specifik identifierare, vilken matchar identifieraren för en RPDO av en eller flera konsumenter. En PDOs datafält kan som mest innehålla åtta bytes av processdata. (Pfeiffer, Ayre & Keydel, 2003) (CAN in Automation *CANopen*)



Figur 27 *PDO struktur (CAN in Automation CANopen)*

I Objektlistan motsvaras en PDO av ett antal poster, vilka avgör beteende och sändningsteknik samt ger gränssnittet till applikationsprogrammets processignaler. Det finns två typer av parametrar för en PDO. Kommunikationsparametrarna beskriver beteendet för en PDO. De definierar vilket COB-ID som används, hur triggning sker och andra sändningsvillkor för en specifik PDO. Kommunikationsparametrarna definieras i index 20h i objektlistan, se Figur 28. Mappningsparametrarna definierar vilka av objektslistans poster, det vill säga, poster där applikationsprogrammets processignaler lagras, som ska ingå i en specifik PDO. Mappningsparametrarna definieras i index 21h i objektlistan, se Figur 29. Det går maximalt att mappa 64 objekt. (Pfeiffer, Ayre & Keydel, 2003) (CAN in Automation *CANopen*)

Index	Delindex	Variabel	Datatyp
0020h	0h	Antal poster	Unsigned8
0020h	1h	COB-ID	Unsigned32
0020h	2h	Överföringstyp	Unsigned8
0020h	3h	Fördröjningstid	Unsigned16
0020h	4h	Reserverad	Unsigned8
0020h	5h	Händelsetimer	Unsigned16

Figur 28 *Definition av kommunikationsparametrar*

Index	Delindex	Variabel	Datatyp
0021h	0h	Antal poster	Unsigned8
0021h	1h	1:a objektet	Unsigned32
0021h	2h	2:a objektet	Unsigned32
0021h	3h	3:e objektet	Unsigned32
0021h	…	…	…
0021h	40h	64:e objektet	Unsigned32

Figur 29 *Definition av mappningsparametrar*

Varje objekt mappas med hjälp av en 32-bit variabel, där både adresseringen till korrekt plats i objektlistan, i form av ett 16-bit index och ett 8-bit delindex, och längden på objektet placeras, enligt Figur 30. (Pfeiffer, Ayre & Keydel, 2003) (CAN in Automation *CANopen*)

31	16	15	8	7	0
16-bit Index		8-bit Delindex		Längd	

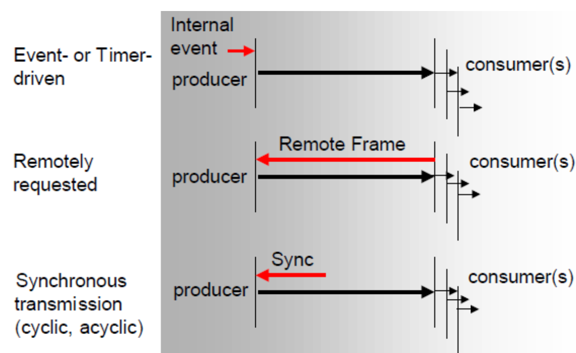
Figur 30 *Mappningsstruktur*

För TPDO beskrivs beteendet och sändningstekniken med start vid 1800h och mappningen med start vid 1A00h. RPDO beskrivs på motsvarande sätt, men med beteendet och sändningstekniken med start vid 1400h och mappningen med start vid 1600h. I ett enda CANopen nätverk kan 512 TPDO och 512 RPDO användas, se Figur 31 . (Pfeiffer, Ayre & Keydel, 2003) (CAN in Automation. *CANopen*)

Receive PDO Communication Parameter (20H)					
1400	RECORD	1st receive PDO Parameter	PDOCommPar	rw	M/O*
1401	RECORD	2nd receive PDO Parameter	PDOCommPar	rw	M/O*
:::::	:::::	:::::	:::::	:::::	:::::
15FF	RECORD	512th receive PDO Parameter	PDOCommPar	rw	M/O*
Receive PDO Mapping Parameter (21H)					
1600	ARRAY	1st receive PDO mapping	PDOMapping	rw	M/O*
1601	ARRAY	2nd receive PDO mapping	PDOMapping	rw	M/O*
:::::	:::::	:::::	:::::	:::::	:::::
17FF	ARRAY	512th receive PDO mapping	PDOMapping	rw	M/O*
Transmit PDO Communication Parameter (20H)					
1800	RECORD	1st transmit PDO Parameter	PDOCommPar	rw	M/O*
1801	RECORD	2nd transmit PDO Parameter	PDOCommPar	rw	M/O*
:::::	:::::	:::::	:::::	:::::	::::
19FF	RECORD	512th transmit PDO Parameter	PDOCommPar	rw	M/O*
Transmit PDO Mapping Parameter (21H)					
1A00	ARRAY	1st transmit PDO mapping	PDOMapping	rw	M/O*
1A01	ARRAY	2nd transmit PDO mapping	PDOMapping	rw	M/O*
:::::	:::::	:::::	:::::	:::::	:::::
1BFF	ARRAY	512th transmit PDO mapping	PDOMapping	rw	M/O*

Figur 31 *Objektlistans struktur med avseende på PDO (CAN in Automation CANopen)*

Kommunikationen för en PDO har tre olika utlösningsmetoder, se Figur 32. Den första utlösningsmetoden är att en applikationsspecifik händelse inträffar, som är specificerad i enhetsprofilen eller att en timer styr utlösningen. Den andra utlösningsmetoden är att en begäran av en PDO skickas från en annan enhet. Den tredje utlösningsmetoden är att överföringen utlöses av en mottagen SYNC-signal. Man skiljer på asynkron och synkron överföring, där synkron överföring innebär att en PDO enbart kan bli triggad av SYNC-signalen. En synkron överföring har också högre prioritet än en asynkron överföring. De två första alternativen är asynkrona, men de kan även kombineras med SYNC-signal, så att överföringen sker synkront. (Pfeiffer, Ayre & Keydel, 2003) (CAN in Automation *CANopen*)



Figur 32 *Olika överföringsmetoder (CAN in Automation CANopen)*

En synkron överföring kan i sin tur delas in i periodisk och operiodisk synkron överföring. En periodisk synkron överföring innebär att en PDO utlöses regelbundet varje gång ett bestämt antal, mellan 1 upp till 240, SYNC-signaler mottagits. En operiodisk synkron överföring innebär att en PDO utlöses av att en SYNC-signal mottagits ihop med antingen en begäran från en annan enhet eller att en applikationsspecifik händelse inträffar. (Pfeiffer, Ayre & Keydel, 2003) (CAN in Automation *CANopen*)

Överföringstypen av en PDO definieras med en kommunikationsparameter och genom att ange olika värden till parametern kan man utnyttja olika utlösningsmetoder, se Figur 33. (Pfeiffer, Ayre & Keydel, 2003) (CAN in Automation *CANopen*)

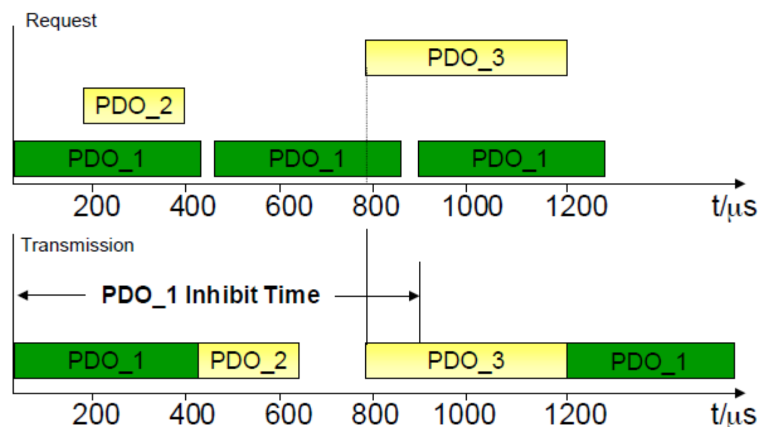
Överföringstyp	Synkron	Begäran	Händelse	Beskrivning
0	X		X	Synkron, Operiodisk
1-240	O			Synkron, Periodisk
241-251				Reserverade
252	X	X		Synkron, efter begäran
253		O		Asynkron, efter begäran
254		O	O	Asynkron, specifik manufaktur händelse
255		O	O	Asynkron, specifk enhetsprofil händelse

Figur 33 *Hur olika överföringstyper sätts*

X = Båda måste inträffa

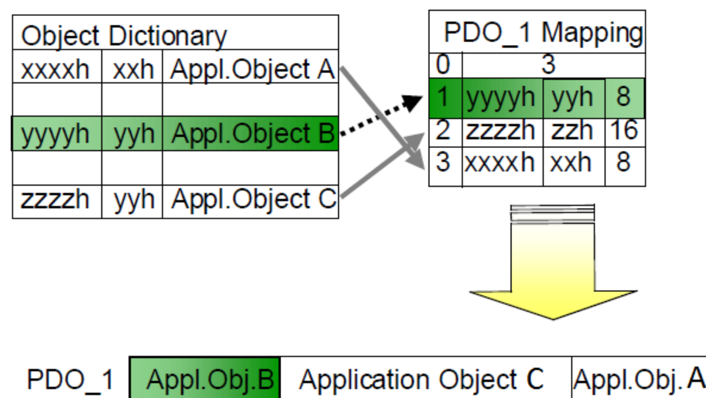
O = En eller båda måste inträffa

För att förhindra att en PDO med högre prioritet blockar PDOs med lägre prioritet, genom att ockupera bussen hela tiden, kan en PDO tilldelas en fördröjningstid. Fördröjningstiden är den minsta tid som måste passera mellan två överföringar av en och samma PDO. Som man kan se i Figur 34 har PDO_2 och PDO_3 en lägre prioritet än PDO_1, men fördröjningstiden för PDO_1 gör att PDO_2 och PDO_3 kan få tillträde till bussen och sända. (Pfeiffer, Ayre & Keydel, 2003) (CAN in Automation *CANopen*)



Figur 34 *Överföring av PDO med fördröjningstid (CAN in Automation CANopen)*

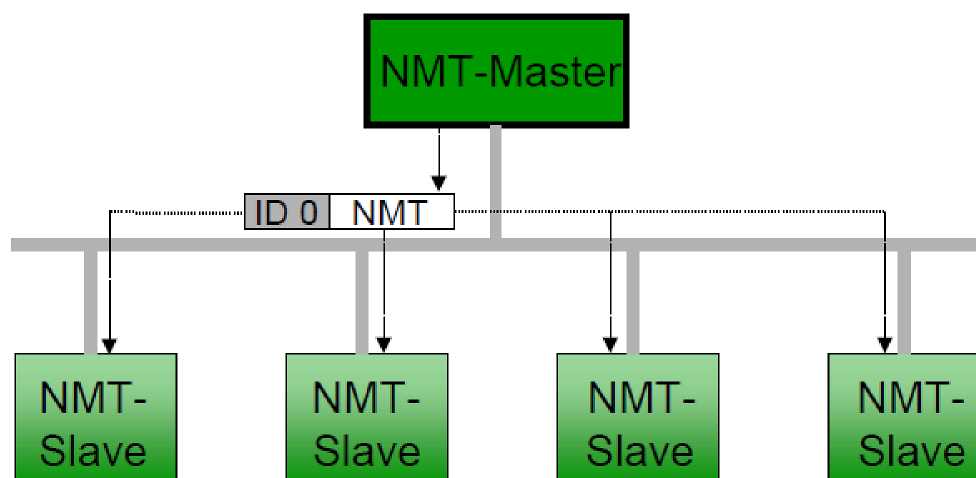
Mappningen för en PDO görs som sagt med mappningsparametrar som definierar vad en specifik PDO ska innehålla för applikationsobjekt. I Figur 35 ses ett exempel på hur en PDO mappning kan se ut. Applikationsobjekten är placerade vid olika poster i objektlistan och dessa poster mappas till PDO_1 via mappningsparametrar, så att PDO_1 vet var dessa applikationsobjekt finns. (Pfeiffer, Ayre & Keydel, 2003) (CAN in Automation *CANopen*)



Figur 35 *Mappningsexempel för PDO (CAN in Automation CANopen)*

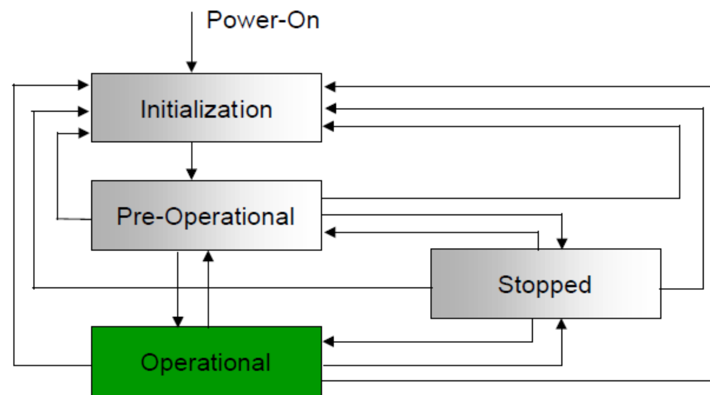
3.3.7 Network Management Protocol, NMT

NMT-protokollet fungerar enligt Master/Slave modellen, där en enhet i nätverket fungerar som en NMT-master medan de andra enheterna är NMT-slavar, enligt Figur 36. NMT-protokollet bidrar med flera olika funktioner, däribland initiering och styrning av NMT-slavar och nodövervakning. NMT-meddelanden är de meddelanden som har allra högst prioritet, då identifieraren är 0. (Pfeiffer, Ayre & Keydel, 2003) (CAN in Automation *CANopen*)



Figur 36 *NMT-struktur (CAN in Automation CANopen)*

Med NMT använder man sig av en tillståndsmaskin, se Figur 37 och det är masterenheten som styr slavenheterna mellan de olika tillstånden. Tillståndsmaskinen består av fyra tillstånd; ”initiering”, ”före driftläge”, ”driftläge” och ”stoppad”, där kommunikationen sker med fem olika NMT-kommandon för att styra en enhet till olika tillstånd. Vid strömpåslag initieras NMT-slavenheten automatiskt och går vidare till tillståndet ”före driftläge”. I det här tillståndet kan enheten konfigureras med hjälp av SDO, svara på nodövervakning och hantera funktionen för sina interna applikationssignaler. En NMT-master kan sätta enheten till ”driftläge” och i det tillståndet är enheten också tillåten att utföra ytterligare funktioner, som PDO-överföring och sända akuta felmeddelanden. Om en enhet sätts till tillståndet ”stoppad”, så stoppas all kommunikation med SDO, PDO och andra funktioner, däremot hanteras fortfarande de interna applikationssignalerna. (Pfeiffer, Ayre & Keydel, 2003) (CAN in Automation *CANopen*)

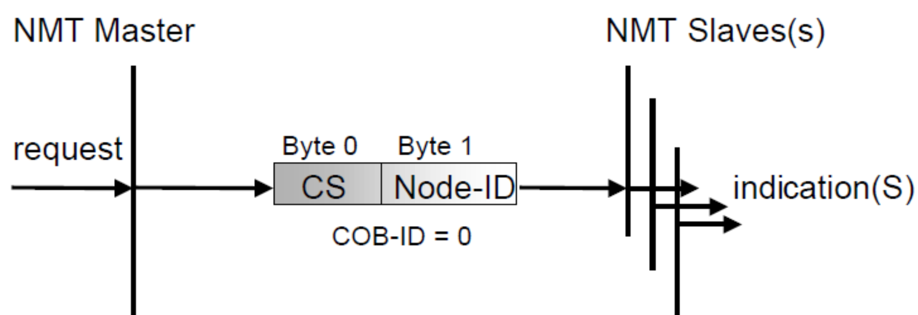


Figur 37 *CANopen tillståndsmaskin (CAN in Automation CANopen*

Kommunikationen mellan de olika tillstånden sker som sagt med fem olika NMT-kommandon. Ett NMT-meddelande, se Figur 38, är uppdelat i två delar, där den första delen är på en byte och det är här NMT-kommandot placeras genom att olika värden anges för olika kommandon enligt:

- Starta enhet/Sätt till driftläge = 1
- Stoppa enhet/Sätt till stoppad = 2
- Sätt till före driftläge = 128
- Återställ enhet/Sätt till initiering = 129
- Återställ kommunikation/Sätt till initiering = 130

Den andra delen är också på en byte och här definieras nod-ID för meddelandets destination. Man kan antingen ange nod-ID 1-127 för att välja en specifik nod eller 0 för att välja alla noder. (Pfeiffer, Ayre & Keydel, 2003) (CAN in Automation *CANopen*)



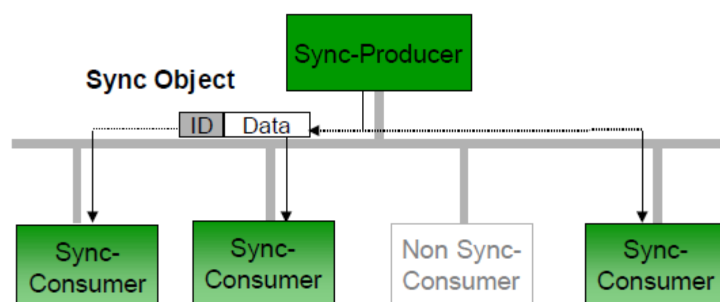
Figur 38 *NMT meddelandestruktur (CAN in Automation CANopen)*

För att hålla koll på status för alla enheter finns det två protokoll, Node Guarding-protokollet och Heartbeat-protokollet. Node Guarding-protokollet fungerar som sådant att NMT-mastern regelbundet hämtar vilket tillstånd alla NMT-slavarna befinner sig i genom att skicka en begäran som svaras av slavarna. Dessa tillstånd jämförs sedan med tillstånd som registrerats i en nätverksdatabas för att säkerställa att alla enheter fungerar korrekt. Hearbeat-protokollet fungerar genom att NMT-slavarna periodiskt skickar ett hjärtslagsmeddelande med vilket tillstånd de befinner sig i till NMT-mastern. Periodtiden mellan två, på varandra följande, hjärtslag kan definieras och om inte NMT-mastern erhåller ett hjärtslags-meddelande inom denna tid signaleras ett fel. Man använder sig bara av en av dessa metoder och det är Heartbeat-protokollet som är rekommenderat. (Pfeiffer, Ayre & Keydel, 2003) (CAN in Automation *CANopen*)

3.3.8 Special Object Protocol

Det finns flera mindre protokoll, som tillhör Special Object protokollet och dessa är Synchronization Object (SYNC) protokollet, Time Stamp Object (TIME) protokollet och Emergency Object (EMCY) protokollet. (Pfeiffer, Ayre & Keydel, 2003) (CAN in Automation *CANopen*)

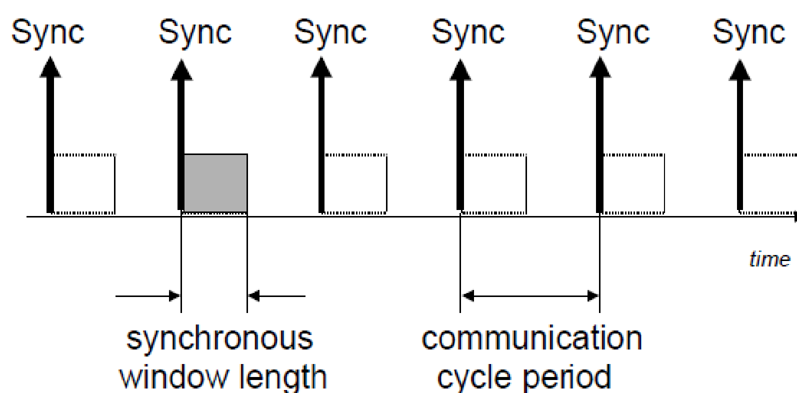
SYNC-protokollet fungerar enligt Producer/Consumer modellen, där en enhet agerar producent och skickar en SYNC-signal till en eller flera andra enheter som agerar konsumenter, enligt Figur 39. Så fort SYNC-signalen tagits emot av en konsument börjar den utföra alla dess synkrona arbetsuppgifter. (Pfeiffer, Ayre & Keydel, 2003) (CAN in Automation *CANopen*)



Figur 39 *SYNC-signalens gång i nätverket (CAN in Automation CANopen)*

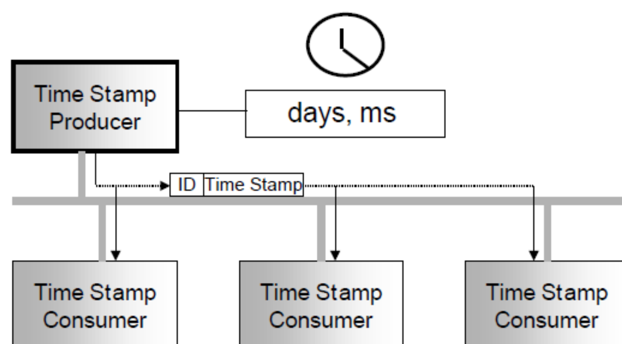
SYNC-signalen ska ha snabb tillgång till bussen, så därför har den en mycket hög prioritet i form av en låg identifierare. I de allra flesta fall med CANopen har identifieraren värdet 128 eller 80h. Ingen form av data transporteras med SYNC signalen. SYNC signalens identifierare definieras vid index 1005h. (Pfeiffer, Ayre & Keydel, 2003) (CAN in Automation *CANopen*)

En synkron överföring av en PDO innebär att överföringen är anpassad i förhållande till överföringen av SYNC-signalen. SYNC-signalen har ett givet tidsfönster, se Figur 40, som den synkrona PDO-överföringen sänds inom. Vid index 1006h definieras SYNC-signalens tidsfönster och vid index 1007h definieras tidsperioden mellan två följande SYNC-signaler. (Pfeiffer, Ayre & Keydel, 2003) (CAN in Automation *CANopen*)



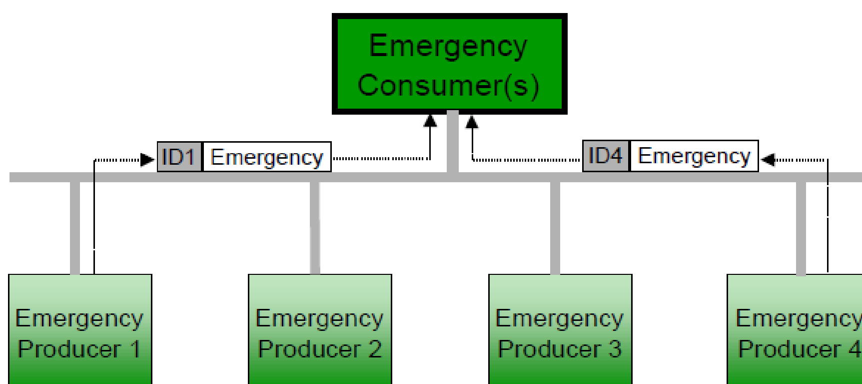
Figur 40 *SYNC-signalens tidsfönster och period (CAN in Automation CANopen)*

TIME-protokollet fungerar också enligt Producer/Consumer-modellen med en sändande enhet som producent och minst en mottagande enhet som konsument, enligt Figur 41. Identifieraren för denna typ av meddelande är 256 eller 100h och är definierad vid index 1012h. Ofta representerar ett sådant meddelande en tid i millisekunder efter midnatt och antalet dagar sedan 1 januari 1984 och detta ger en datalängd på 6 bytes. En del nätverk med tidskritiska applikationer kräver mycket exakt synkronisering. Att synkronisera de lokala klockorna med en noggrannhet på mikrosekunden kan då vara nödvändigt för att justera den oundvikliga avdriften av de lokala klockorna. Det finns ett högupplöst synkroniseringsprotokoll som man kan tillämpa för att åstadkomma detta. Vid index 1013h definieras en 32-bitars variabel, som har en upplösning på 1 mikrosekund, som används som tidräknare. (Pfeiffer, Ayre & Keydel, 2003) (CAN in Automation *CANopen*)



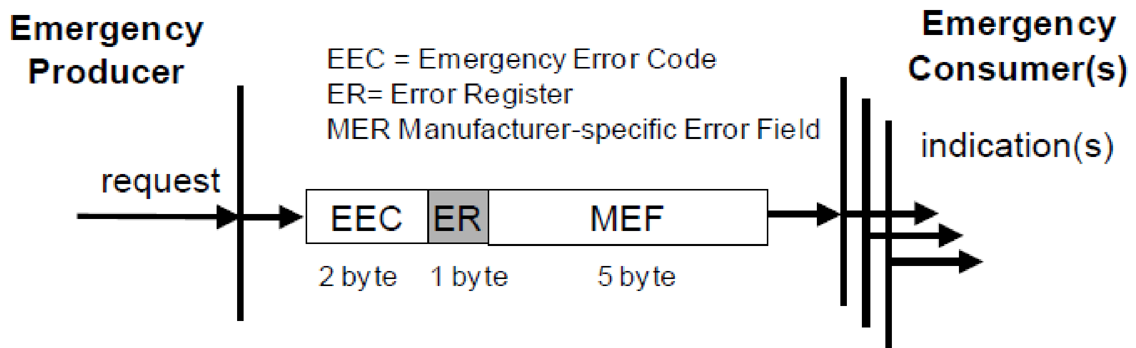
Figur 41 *Time Stamp-struktur (CAN in Automation CANopen)*

Även EMCY-protokollet fungerar enligt Producer/Consumer-modellen och utlöser ett felmeddelande om en enhet upptäcker ett allvarligt fel internt, se Figur 42. Detta felmeddelande skickas med hög prioritet till andra enheter på nätverket och det skickas bara en gång per händelse. (Pfeiffer, Ayre & Keydel, 2003) (CAN in Automation *CANopen*)



Figur 42 *Felmeddelandens gång i ett nätverk (CAN in Automation CANopen)*

Ett felmeddelande har en identifierare mellan 129 till 255 beroende på vad den sändande enheten har för nod-ID. Ett felmeddelande innehåller 8 bytes, som är uppdelade i tre delar, enligt Figur 43. Den första delen är på 2 bytes och beskriver detaljerat vad det är för fel som inträffat genom att det finns fördefinierade koder för olika typ av fel, enligt Figur 44. Den andra delen är på 1 byte och är ett register där felen dokumenteras och lagras tills de är lösta, så denna delen beskriver inte bara det nyss inträffade felet utan även om det är nått annat typ av fel kvar inom enheten. Detta görs också genom att det finns fördefinierade koder för olika typ av fel, dock inte lika detaljerat. (Pfeiffer, Ayre & Keydel, 2003) (CAN in Automation *CANopen*)



Figur 43 *Felmeddelandets struktur (CAN in Automation CANopen)*

Code	Description	
00xx	Error Reset or No Error	
10xx	Generic Error	
20xx	Current	
21xx		device input side
22xx		inside of device
23xx		device output side
30xx	Voltage	
31xx		main
32xx		inside of device
33xx		output
40xx	Temperature	
41xx		ambient
42xx		device
50xx	Device Hardware	

Code	Description	
60xx	Device Software	
61xx		internal
62xx		user
63xx		data set
70xx	Additional Modules	
80xx	Monitoring	
81xx		communication
8110		CAN overrun
8120		Error Passive (EP)
8130		Life Guard Error
8140		recovered from Bus-off
82xx	Protocol Error	
8210		PDO not processed
8220		length exceeded
90xx	External Error	
F0xx	Additional Functions	
FFxx	Device Specific	

Figur 44 *Fördefinierade akuta felkoder (CAN in Automation CANopen)*

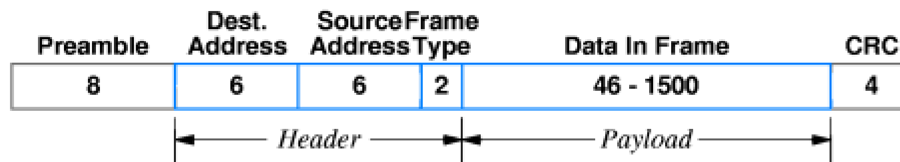
3.4 Ethernet

Ethernet är en teknologi för lokala nätverk, Local Area Network, LAN och introducerades av DIX-trion (Digital Equipment Corporation, Intel och Xerox) år 1980. Termen Ethernet hänvisar idag till en familj av nära relaterade protokoll som karakteriseras av det medium som används och den datahastighet som uppnås. De tidiga versionerna, version 1 & 2 Ethernet används fortfarande men många nya system nyttjar IEEE 802.3-standarden från 1984. (Spencer, 2012)

Ethernet tillhandahåller tjänster som täcks av OSI-modellens två lägsta lager, det fysiska lagret och datalänklagret. Signaler skickas med Manchesterkodning vilken bland annat har fördelen att signalen innehåller klockningsdata vilket medför att mottagaren kan synkronisera sin klocka och avgöra var en bit börjar och en annan slutar. Ett Ethernet nätverk har oftast en buss- eller stjärntopologi. Den förstnämnda utnyttjar samma princip som CAN, med ett gemensamt medium som alla enheter kopplas in på. Med en stjärntopologi är alla enheter inkopplade via en individuell kabel till en central enhet såsom en switch eller brygga. (Wildpackets Inc.)

Enheter på ett Ethernet nätverk med busstopologi, som arbetar i halv-duplex läge, är liksom de på CAN sammankopplade via ett gemensamt medium vilket gör att någon form av Media Access Control, MAC, måste implementeras. Här används åtkomstmetoden *Carrier-Sense Multiple Access with Collision Detection*, CSMA/CD vilken beskrivits tidigare, se kapitel 3.2.5. (Wildpackets Inc.)

Alla enheter på nätverket har en unik fysisk identitet för kommunikation, en så kallad MAC-adress. Den är 48 bitar lång och används både för att specificera avsändare och mottagare av ett meddelande. Liksom CAN, har alla enheter tillgång till den data som skickas på nätverket men istället för att använda identifierare används direkt adressering. Det första en enhet gör när ett meddelande läggs ut på mediumet är att analysera adressfältet, stämmer destinationsadressen ej överens med enhetens egen adress så avslutas läsningen. Ett undantag är om destinations- och avsändaradressen är lika, då är meddelandet riktat till alla enheter på nätverket. (Wildpackets Inc.)



Figur 45 *Ethernet Meddelanderam (Singh, 2010)*

Ethernets versionsmångfald har lett till att flera olika meddelandeformat uppstått, men ett generellt meddelande består av delarna illustrerat i Figur 45.

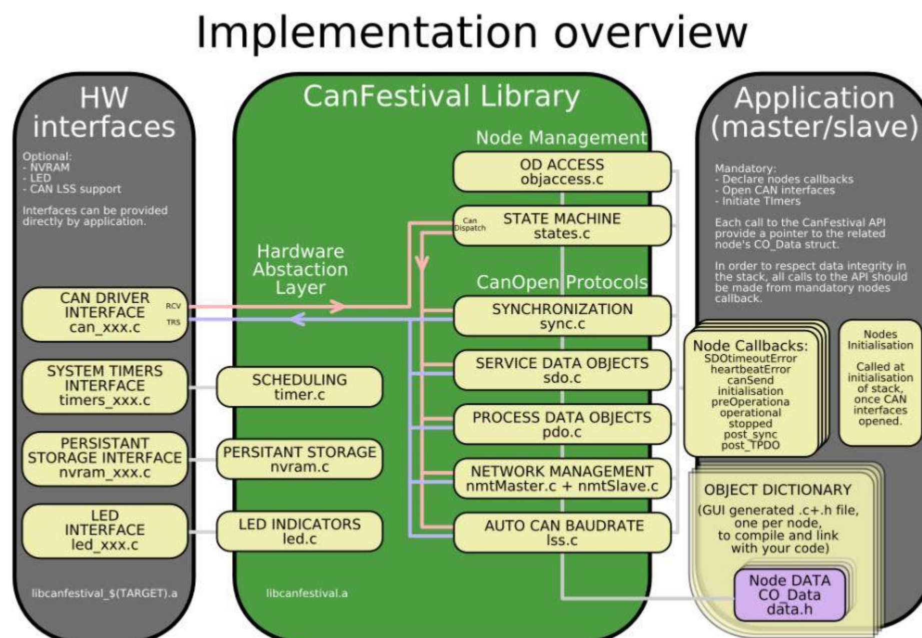
Alla meddelanden startar med en ingress på 64 bitar som består både av skiftande 1:or och 0:or som mottagaren använder för att synkronisera med den inkommande signalen och ett 8-bitars SOF fält. Därefter följer en 48-bitars destinationsadress, en 48-bitars avsändaradress och en 16-bitars typindikerare för identifiering av vilket protokoll som används. Datafältet innehåller meddelandet och är i storleksordningen 46 till 1500 bytes. Sist är ett 32-bitars CRC fält. (Wildpackets Inc.)

All kommunikation sänds på nätverket i form av sådana meddelanderamar eller paket. Dessa innehåller alltså den data som ska skickas, tillsammans med information om bland annat mottagaren och avsändaren. Paket konstrueras genom att den applikation som genererar data skickar denna till en protokollstack som sedan delar upp datan i mindre bitar och lägger till ett omslag (eng. *Wrapper*) till varje databit. Omslaget gör att databitarna kan assembleras i korrekt följd av mottagande applikation. Protokollstacken skickar därefter vidare paketen, som nu har omslag, till hårdvaruenheten som i sin tur lägger till ytterliggare omslag för att leda meddelandet till dess rätta destination på nätverket. I mottagaränden görs samma steg fast i reverserad ordning. (Wildpackets Inc.)

3.5 CANfestival

CANfestival är ett gratis CANopen-ramverk och tillhandahåller en plattformsoberoende CANopen-stack för implementation av master- eller slavnoder, se Figur 46 för en detaljerad översikt. Det är ett pågående projekt som startades 2001 av Edouard Tisserant men har sedan dess växt och drivs nu även av medlemarna Francis Dupin och Laurent Bessard. (CANfestival, 2001)

CANfestival överensstämmer med CANopen DS301. V.4.02 och innehåller förutom funktioner såsom SDO, PDO och NMT även ett verktyg för att generera objektlistor för noder. Det är rekommenderat att köra CANfestival under en Linux-distribution men det går även att köra under Windows med Cygwin installerat. Beroende på vilket operativsystem biblioteket installeras på så varierar stödet för olika CAN-gränssnitt. För tillverkare såsom IXXAT eller Kvazer erbjuds endast stöd för Windows, medan PEAK Systems produkter stöds av de vanligaste Linux/Windows distributionerna. Med CANfestival går det även att skapa virtuella gränssnitt vilket kan vara tacksamt vid testning. (CANfestival, 2001)



Figur 46 *CANfestival-implementation(CANfestival, 2001)*

CANfestival innehåller flera exempelprogram vilka kan vara mycket användbara ur ett inlärningsperspektiv. Biblioteket innehåller bland annat program för uppstart och test av master-slave konfiguration och virtuella gränssnitt. (CANfestival, 2001)

4 Genomförande

Projektet kan delas in i ett antal olika moment med avseende på utförandet. Det första momentet bestod, förutom att läsa på om CAN och CANopen, av att undersöka utbudet på marknaden efter lämplig hård- och mjukvara. Gemensamt för dessa var att alla skulle ha stöd eller vara uppbyggda för CANopen, då Saabs system ska använda sig av denna standard och att de i största möjliga mån skulle uppnå de tidigare specificerade kraven och önskemålen. Det fastslogs att det skulle vara både enklare och bättre att utgå ifrån mer färdigutvecklade produkter och program istället för att utveckla helt nya från grunden. En hårdvaruenhet att upprätta som master i en CANopen miljö behövdes. För att uppfylla kraven på hårdvaran, bland annat krav på uppstartstider, pekades det ganska snart på en ARM processor med MMU-stöd som levereras med någon form av Linux OS-distribution, med passande GNU toolchain. Det behövdes även en CAN-till-USB-adapter för att möjliggöra kommunikation mellan mastern och en simulerad CANopen miljö bestående av en eller flera slavar på en dator. Krav för denna adapter var att den skulle vara kompatibel med valet av mjukvara. Utöver dessa hårdvaruenheter behövdes en passande mjukvara att använda för utveckling av hela CANopen miljön, det vill säga både master- och slavsidan. Krav för denna mjukvara var att den antingen skulle möjliggöra enkel utveckling av egna master- och slavapplikationer eller fungera som en färdig simulerad CANopen miljö, framförallt med avseende på slavsidan.

Nästa moment bestod i att utveckla CANopen miljön. Implementationsmässigt delas detta upp i två olika delar, en masterdel och en slavdel. Mastern på hårdvaruenheten skulle kunna styra och hantera övervakning av slaven, men även kunna begära och tolka slavens status.

De sista momenten bestod av integrering och programkörning av systemet, det vill säga CANopen-miljön med en master på hårdvaruenheten och en slav på en bärbar dator samt med GPIO signaler. Integrering var något som gjordes för att kunna upprätta önskad kommunikation i systemet medan programkörning av systemet gjordes för att kunna korrigera eventuella fel i master- och slavdelen.

4.1 Undersökning och val av hårdvara och mjukvara

Vid sökandet efter kvalificerad hård- och mjukvara fanns det som sagt många krav och önskemål att ta hänsyn till. I följande kapitel beskrivs hård- och mjukvarulösningarna samt varför de valdes.

4.1.1 Hårdvara

Efter en tids efterforskning hittades till slut några tänkbara alternativ av typerna utvecklingskort och framförallt industriella boxdatorer. De utvecklingskort som hittades hade i sitt basutförande endast ett fåtal av de önskade gränssnitten och det krävdes således ytterligare tillbehör för att uppfylla kravet på interoperabilitet. Därför prioriterades andra alternativ, såsom industriella boxdatorer. Dessa boxdatorer var i regel bättre i avseende på uppfyllnad av kraven, men många visade sig sakna stöd för fler än ett CAN- och Ethernetgränssnitt.

Det alternativet som anammades var en Artila Matrix 522, se Figur 47, som är en industriell Linux-box-dator med ARM-processor. Anledningen till detta var att den uppfyllde många av de krav och önskemål vi hade för en sådan enhet, tack vare följande egenskaper och funktioner:

- ARM9 Processor, 400 MHz med MMU
- Två CAN 2.0A/2.0B
- Två 10/100 Mbps Ethernet
- 9 – 40 VDC spänningsmatning
- Färdiginstallerat Linux 2.6.29 OS
- GNU toolchain tillgänglig
- Support SocketCAN and CANopen Library

Stödet av CANopen var nödvändigt, inte minst för att projektet baseras på CANopen, men även det extra stödet för SocketCAN var välkommet. SocketCAN är en uppsättning drivrutiner och en protokollstack som möjliggör simultan tillgång till en CAN-enhet från flera applikationer. Vidare tillåts en applikation prata med flera CAN-nätverk samtidigt.

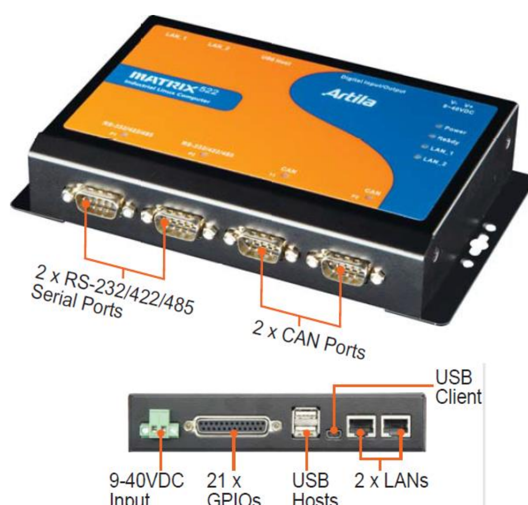
De två CAN-gränssnitten gjorde att kravet för interoperabilitet uppfylldes, genom att det ena CAN-gränssnittet kan användas för styrning och övervakning av systemet och det andra CAN-gränssnittet kan användas för systemets transportplattform, till exempel en lastbil.

De två Ethernet-gränssnitten gjorde det möjligt för kommunikation med UCP och styrning av datorer/datorkort samtidigt, så det kravet uppfylldes också.

Det fanns krav på att stöd skulle finnas för 28 V spänningsmatning och då den har stöd för 9 – 40 VDC spänningsmatning uppfylldes även detta krav.
I övrigt är den lätthanterad och enkel att komma igång med, då den har ett färdiginstallerat OS och en tillgänglig passande toolchain.

Tolerans mot yttre faktorer, såsom temperatur och vibrationer, och de övriga kraven, såsom kort startup-tid och hantering av diskreta signaler, var några vi inte tog full hänsyn till, men det är fullt möjligt att dessa krav kan uppfyllas.

Den hade ytterligare användbara egenskaper och funktioner, såsom två seriella gränssnitt, två USB-gränssnitt, 21 programmerbara digitala in- och utsignaler samt låg strömförbrukning. En annan fördel med Artila Matrix 522 gentemot många andra alternativ, såsom utvecklingskort, var det faktum att den är inkapslad. För mer information om Artila Matrix 522, se kapitel 8.1.



Figur 47 ***Artila Matrix 522 (Artila 2013)***

Vid sökandet av en USB-till-CAN-adapter ställdes kravet att den skulle ha bästa möjliga kompabilitet med CANfestival, då vi valde att använda denna mjukvara, se kapitel 4.1.2. Det fanns flera adaptrar som hade stöd för CANfestival, men många av dessa var plattformsberoende, därför valde vi att köpa in och använda oss av en från PEAK Systems, PEAK PCAN-USB, se Figur 48, då denna hade stöd för de vanligaste Linux/Windows-distributionerna. För mer information om PEAK PCAN-USB, se kapitel 8.2.

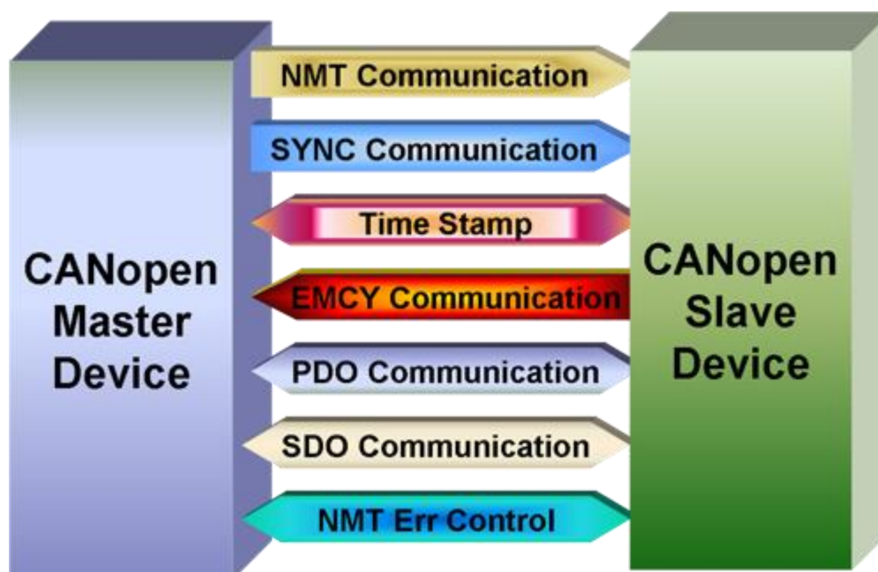
Figur 48 *PEAK PCAN-USB (PEAK SYSTEMS 2013)*

4.1.2 Mjukvara

Vid sökningen efter en lämplig mjukvara var vi först ute efter en som tillhandahöll en färdig simulerad CANopen miljö med avseende på slavsidan och att därefter själva enbart skapa masterdelen till denna simulerade CANopen miljö. En sådan mjukvara visade sig vara svår att få tag på till ett rimligt pris. Vi övergav därför denna idé och började istället undersöka om en färdig CANopen-protokollstack kunde vara ett alternativ, där man med dess hjälp kunde skapa egna master- och slavapplikationer. Det visade sig att det absolut var en möjlighet och då vi sökte efter ett alternativ av typen öppen källkod upptäckte vi CANfestival. Vi beslutade att vi skulle använda oss av denna protokollstack på båda enheterna, det vill säga både för master på den inköpta hårdvaruenheten och för slav på en bärbar dator. Det ingick flera olika testprogram i CANfestival och de visade sig vara mycket användbara som grund för skapandet av master- och slavapplikationer.

4.2 Programdesign

När det kom till utvecklingen av master- och slavapplikationer skulle mastern på hårdvaruenheten kunna styra och hantera övervakning av slaven, men även begära och tolka slavens status. För att lyckas med det har CANopen olika kommunikationsprotokoll som används för olika ändamål och kommunikationsvägen för dessa ges av Figur 49 nedan. All kommunikation mellan master och slav initieras av mastern med undantag för felmeddelanden.



Figur 49 *Master/Slave-kommunikation (ICPDAS 2013)*

Det är masterns uppgift att konfigurera slaven, framförallt med avseende på de olika kommunikationsprotokollens funktion. Det görs med SDO-kommunikation genom att mastern skriver in information på korrekt plats i slavens objektlista.

Med PDO-kommunikation kan mastern begära och få information om slavens status. Detta genom att skicka en PDO-begäran till slaven om att få ett eller flera aktuella statusvärden, som lagras i slavens objektlista. Slaven tillhandahåller sedan mastern dessa statusvärden genom att skicka dem till mastern, som i sin tur lagrar dem i sin objektlista.

Mastern skall, med ett visst tidsintervall, kontinuerligt skicka synkroniseringssignaler med SYNC-kommunikation, detta kan bland annat användas vid PDO-kommunikation för att synkronisera kommunikationsöverföringen.

Mastern skall konstant övervaka slavenheten, så att mastern vet att den fungerar korrekt, detta görs med NMT Err Control och vanligast är att använda sig av en hjärtslagsmetod, där slaven skickar vilket tillstånd den befinner sig i till mastern. Detta är något som görs periodvis, och periodtiden för detta konfigureras av mastern.

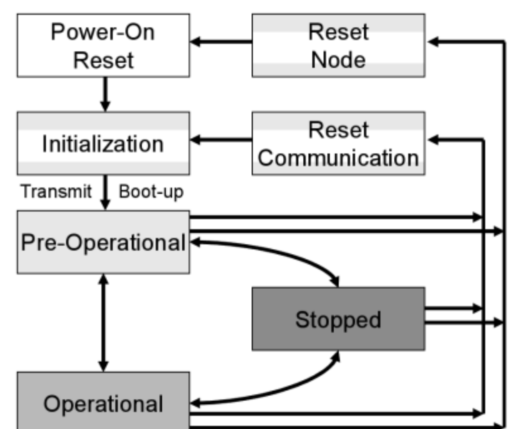
EMCY-kommunikation används när ett fel har inträffat på enheten och det är bara slaven som skall sända sådana typer av felmeddeladen, där mastern skall stå som mottagare. Givetvis kan fel inträffa även på masterenheten, men då behövs och skall inte det rapporteras vidare till andra enheter.

Time Stamp-kommunikation har vår master inget behov av, så det behövs inte implementeras.

Mastern skall kunna styra slaven mellan olika tillstånd i CANopens tillståndsmaskin, se Figur 51, med NMT-kommunikation, där varje tillstånd har specifika arbetsuppgifter och varierande begränsningar vad gäller användandet av olika kommunikationsprotokoll enligt Figur 50. Mastern skall också använda sig av CANopens tillståndsmaskin, men till skillnad från slaven skall mastern hantera övergångar på egen hand.

	Initializing	Pre-operational	Operational	Stopped
Boot-Up	◆			
SDO		◆	◆	
Emergency		◆	◆	
SYNC/TIME		◆	◆	
Heartbeat/ Nodeguard		◆	◆	◆
PDO			◆	

Figur 50 *Kommunikationsbegränsningar*
(Pfeiffer, Ayre & Keydel, 2003)



Figur 51 *Tillståndsmaskin (Pfeiffer, Ayre & Keydel, 2003)*

Till CANfestival ingick som sagt flera olika testprogram, varav ett innehöll både en master och en slav. Vi valde att nyttja detta testprogram för att skapa både vår master och vår slav genom att vi modifierade programmet. Detta program gick att köra med både master och slav aktiva eller med enbart en av dem. Det var mycket användbart för oss, då vi kunde modifiera två testprogram, där masterdelen modifierades på den ena och slavdelen på den andra. Således kördes programmen, den ena på hårdvaruenheten och den andra på den bärbara datorn, mot varandra.

Det som utgör själva kärnan i en nod är objektlistan. Det är i objektlistan som all information om noden finns, till exempel hur kommunikation med de olika kommunikationsprofilerna ska fungera och lagring av processignaler. Med CANfestival ingick ett program för att kunna skapa nya objektlistor och ändra tidigare gjorda objektlistor. Med detta program modifierade vi vissa delar av testprogrammens objektlistor till att passa våra önskemål för både master- och slavnoden.

4.2.1 Programdesign, Master

Objektlistan för mastern modifierades genom att alla RPDO, som skulle användas för läsning av slavens statusvärden, mappades till korrekta platser i masterns objektlista, så att statusvärdena kunde lagras där. Dessa platser och variabler för masterprogrammets processignaler definierades på ett sätt som gav önskvärd funktion.

Vid start av själva masterprogrammet initieras mastern genom att bland annat sätta dess nod-ID till 1. Nu initierar mastern också hårdvaruenhetens General Purpose Input/Output, GPIO, som ska användas för att begära de olika statusvärdena samt tända en grön LED som indikerar att slavenheten fungerar som den ska. Det görs genom att alla 21 pinnar definieras som insignaler förutom pin 12, som definieras som utsignal. Mastern konfigurerar sina fyra RPDO för att matcha slavens fyra TPDO, så att kommunikationen ska fungera mellan dessa, genom att skriva i sin egen objektlista. Efter detta går mastern vidare till tillståndet ”före driftläge”.

När slaven har initierats korrekt och befinner sig i tillståndet ”före driftläge” precis som mastern, konfigurerar mastern slavens överföringsteknik, när det gäller alla fyra TPDO, till att enbart sändas vid mottagande av PDO-begäran från mastern. Denna konfigurering görs genom SDO och på så sätt ändras informationen i nodens objektlista. Mastern konfigurerar också periodtiden, för sändning av slavens hjärtslag, till 1000 ms. Därefter går mastern vidare till ”driftläge” och begär att slaven ska göra detsamma genom att skicka ett NMT-meddelande.

I detta läge har programmet nått sin fulla potential och det är nu mastern kan börja begära statusvärden från slaven. Mastern skickar kontinuerligt synkroniseringssignaler med ett tidsintervall av en halv sekund mellan två på varandra följande synkroniseringssignaler. Dessa kan användas för att trigga och synkronisera slavens PDO-överföring, en funktion vi inte använder då vi istället utnyttjar begäran för att trigga slavens PDO överföring. Vid varje skickad synkroniseringssignal granskar mastern de fyra första GPIO-signalerna. Dessa pinnar används för att begära olika statusvärden från slaven genom att varje pinne är matchad med en RPDO. Så om någon av dessa GPIO-pinnar är höga ska mastern skicka en PDO-begäran för den eller de RPDO som motsvaras av den eller de höga GPIO-pinnarna.

Mastern kan även ta emot felmeddelanden från slaven. Tas ett sådant emot så sätts GPIO pin 13 till hög, vilket släcker den gröna LED-lampan och ett fel har indikerats. Åtgärdas felet så meddelas mastern om det och lampan tänds igen.

4.2.2 Programdesign, Slav

Objektlistan för slaven modifierades på liknande sätt som för mastern genom att alla TPDO, som skulle användas för överföring av slavens statusvärden, mappades till korrekt platser i slavens objektlista, så att värden hämtas därifrån. Dessa platser och variabler för slavprogrammets processignaler definierades på ett önskvärt sätt.

Vid start av slavprogrammet hämtas en sökväg till en fil som slaven ska hämta sina statusvärden ifrån. Slaven initieras genom att bland annat sätta sitt nod-ID till 2. Efter det går den vidare till tillståndet ”före driftläge”.

När slaven har initierats korrekt och befinner sig i tillståndet ”före driftläge” konfigureras den av mastern. Slaven går sedan vidare till ”driftläge” när ett NMT-meddelande med en sådan begäran mottagits från mastern.

I ”driftläge” kan slaven ta emot begäran av statusvärden, i form av PDO, från mastern. Mastern svarar med en TPDO, som motsvarar den begäran som mottagits, innehållande det aktuella statusvärdet. Slaven mottar också kontinuerligt synkroniseringssignaler från mastern. Vid varje mottagen signal uppdaterar slaven sina statusvärden genom att öppna och läsa av den fil som den tidigare hämtade sökvägen pekar på. Värdena lagras i slavens objektlista. Skulle de överträda gränserna som satts för de individuella värdena skickas ett felmeddelande med tillhörande felkod till mastern. Av felkoden framgår det vad för typ av fel som inträffat. När slaven är i driftsläge hanterar den även två räknare, en som räknar upp och en som räknar ner.

4.3 Integration av systemet

Integration av systemet var något som påbörjades direkt vid leverans av de beställda produkterna och pågick sedan parallellt med utvecklingen av CANopen-miljön.

Testprogrammet som ingick med CANfestival kunde man ju som sagt köra med både master och slav aktiva eller enbart en av dem. Vi började därför testningen med att köra programmet på en bärbar Linux-dator med både master och slav aktiva. Detta gjordes genom att vi använde oss av ett bibliotek, tillhörande CANfestival, som gjorde det möjligt att skapa virtuella CAN-gränssnitt och köra programmet via dessa. En likadan testning utfördes på hårdvaruenheten, Artila Matrix 522, för att säkerställa att programmet fungerade även där. Efter att dessa test lyckats var nästa steg att köra programmet via de verkliga CAN-gränssnitten.

Vid användning av CAN-gränssnitt är det viktigt att tänka på att de ska vara terminerade på ett korrekt sätt och att baudrate för de använda CAN-gränssnitten matchas. På Artila Matrix 522 var emellertid termineringen inbyggd, men på PEAK PCAN-USB saknades den, så vi terminerade denna ände med 120 Ω. Att inställningen av baudrate stämmer överens är en nödvändighet för att kommunikationen ska fungera och ställdes in till 250 Kbit/s för de båda gränssnitten.

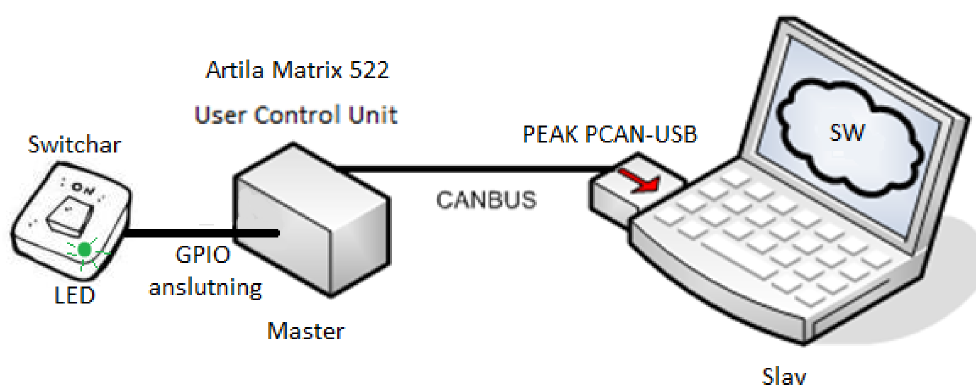
Då Artila Matrix 522 har två CAN-gränssnitt valde vi först att testköra programmet mellan dessa två gränssnitt, istället för de två virtuella CAN-gränssnitten. Detta genomfördes med hjälp av ett annat bibliotek, tillhörande CANfestival, som gjorde användningen av SocketCAN möjlig. Enhetens CAN-gränssnitt måste ha stöd för detta, genom att vara korrigerat och installerat som nätverksgränssnitt, något som Artila Matrix var redan från början.

Därefter var det dags att upprätta kommunikationen mellan master- och slavenheten via PEAK PCAN-USB genom att köra ett program på varje enhet, med en master på Artila Matrix 522 och en slav på den bärbara Linux-datorn. På Artila Matrix 522 körde vi programmet på samma sätt som vid föregående testning med undantag för att vi inaktiverade slaven. På den bärbara Linux-datorn valde vi också att använda oss av

det bibliotek, tillhörande CANfestival, som gjorde användningen av SocketCAN möjlig. Vi var även tvungna att korrigera och installera PEAK PCAN-USB som ett nätverksgränssnitt för att SocketCAN skulle stödjas. På den bärbara Linux-datorn körde vi sedan programmet med enbart en slav genom att inaktivera mastern. Det visade sig slutligen att kommunikation mellan de två enheterna fungerade på önskat sätt, vilket gjorde att all fokus nu kunde läggas på att modifiera och bygga vidare på testprogrammen. Naturligtvis fortsatte testningen, men nu med syftet att upptäcka och korrigera eventuella fel som uppstått efter implementering av nya funktioner.

4.4 Programkörning av systemet

Efter mycket modifiering av programmen så fungerade till slut CANopen-miljön på ett önskvärt sätt. Systemet bestod nu av en master på Artila Matrix 522 och en slav på Linux-datorn och kommunikationen gick via PEAK PCAN-USB enligt Figur 52. Dessutom är en GPIO-anslutning, med switchar och en LED, kopplad till Artila Matrix 522. Switcharna används för att begära statusvärden av slaven, medan LED-lampan indikerar om dessa statusvärden är acceptabla eller inte.



Figur 52 *Översikt av systemet*

Vid en programkörning av systemet ska flera parametrar anges, dels för masterprogrammet och dels för slavprogrammet, se Figur 53. De är likadana för båda programmen med undantag för att sökvägen till filen som innehåller statusvärdena även ska anges i slavprogrammet. De resterande alternativen som ska anges för båda programmen är sökväg till biblioteket, vilka/vilket gränssnitt slaven och/eller mastern ska använda och baudrate för slaven och/eller mastern.

```
Usage:
./TestMasterSlave   [OPTIONS]

OPTIONS:
-l : Can library [\"/usr/local/lib/libcanfestival_can_socket.so\"]

-f : File  dir [\"/vart/filen/ar/placerad/status.txt\"]

Slave:
-s : bus name [\"0\"]

-S : 1M,500K,250K,125K,100K,50K,20K,10K,none(disable)

Master:
-m : bus name [\"1\"]

-M : 1M,500K,250K,125K,100K,50K,20K,10K,none(disable)
```

Figur 53 *Programval vid körning av slavprogram*

5 Resultat

Målsättningen med examensarbetet var att ta fram en prototyp för att kunna genomföra en demonstration och ge förslag till hur kommunikationsgränssnittet kan användas i systemet. Användbar inköpt hård- och mjukvara har identifierats och använts. Den hårdvara som använts köptes in, istället för att en egen konstruktion togs fram och mjukvaran bestod av protokollstackar av typen öppen källkod. Majoriteten av de krav som ställts på enheten är uppfyllda:

Kravet på interoperabilitet uppfylls då enheten har mer än två CAN- och Ethernetgränssnitt.

Toleransen mot yttre faktorer togs i beaktande vid inköpen och kraven uppfylls till stora delar, dock råder det osäkerhet kring prototypens mottståndskraft mot vibrationer. De övriga kraven rörande spänningsmatning uppfylls och uppstartstid kan troligen uppfyllas.

En prototyp av ett CANopen-nätverk med master-slave konfiguration har tagits fram. En lyckad kommunikation mellan olika noder har åstadkommits och prototypens funktionalitet kan demonstreras med de testprogram som utvecklats.

Mastern kan via en switchbräda styras till att begära information om status från slavnoden. Slavnodens statusvärden uppdateras från en fil, så det är enkelt att ändra dessa efter behov. De programkörningar som gjorts har visat att systemet fungerar som tänkt och slaven svarar på de anrop/förfrågningar som görs från mastern. Nodövervakning sker och felmeddelanden både skickas och hanteras på korrekt sätt. Sammantaget fungerar prototypen som tänkt och tillhandahåller Saab med en grundläggande plattform för simulering och utveckling.

6 Slutsats

Inom ramen för detta projekt har en prototyp, som visar funktionaliteten hos ett CANopen-nätverk med master-slave konfiguration, tagits fram. Denna prototyp är till hjälp för Saab, som kan använda den till att utvärdera möjligheten att implementera tekniken i markbaserade radarsystem. Enheten tillhandahåller goda förutsättningar till fortsatt utveckling då den grundläggande kommunikationen med CANopen redan är implementerad.

En stor del av arbetet bestod av inlärning och att studera CAN och det högre applikationsprotokollet CANopen. Avgränsningar gjordes tidigt i arbetet, bland annat togs beslutet att inte implementera styrningen av datorer/datorkort med IPMI. Även funktionen att kunna styra masterenheten och övervaka resten av CANopen-nätverket från en kontrollpanel via Ethernet valdes att inte implementeras. En alternativ lösning gjordes dock för styrning av masterns statusbegäran till slaven, i form av digitala insignaler som kunde ställas via en switchbräda. En LED som visade slavens status implementerades även. Det visade sig vara ett bra beslut att avvakta med Ethernet, då inköp av komponenter och utveckling av ett CANopen-nätverk tog längre tid än vad som först avsatts till de ändamålen. Mjukvaran som användes bestod av CANfestivals protokollstack, vilken implementerar ett CANopen-ramverk och är av typen öppen källkod. Den hårdvara som representerade CAN-nätverkets master och således systemets UCU var en Artila Matrix 522 industriell Linux boxdator. En CANopen-slavnod skapades på en bärbar dator med Linux OS. Kommunikationen dem emellan gick via en PEAK PCAN-USB-adapter. Den färdiga prototypen består alltså av en CANopen-master som med adaptern kan kommunicera med en, på en bärbar dator implementerad, slavnod.

Tack vare det faktum att den UCU som valts har mer än ett CAN- och Ethernetgränssnitt bereds goda möjligheter till utveckling av prototypen. Styrning av UCU och datorer/datorkort via Ethernet samt integrering av ytterligare en CAN-standard, såsom transportfordonets egen, är ett nästa steg i denna utveckling.

6.1 Kritisk diskussion

Den prototyp som tagits fram beskriver funktionen hos ett CANopen-nätverk med en masternod och en slavnod. Trots att avgränsningar gjordes i arbetets början är resultatet tillfredsställande och förutsättningarna för fortsatt utveckling goda. Det finns dock utrymme för förbättring av de implementerade funktionerna hos prototypen.

Det faktum att nätverket endast är uppbyggt av två noder gör att några av fördelarna med CAN och CANopen, såsom arbitrering, inte åskådliggörs. Skälet till detta var tidsbrist, men prototypen hade bättre påvisat dessa fördelar och varit närmare de verkliga systemen om fler implementerats.

Då den ursprungliga tanken om att implementera Ethernet lades åt sidan till förmån för utveckling av CAN, saknades ett mer avancerat och tillfredsställande sätt att styra vår master. Den går nu efter en förutbestämd programslinga där den enda riktiga styrning som görs är vid ställandet av de digitala insignalerna via switchbrädan. Vi utnyttjar således inte masterns fulla potential, då vi inte har möjlighet att styra när och hur dess olika kommunikationsprotokoll ska användas.

Prototypen uppfyllde med säkerhet de flesta krav som ställts på den. Det krävs dock ytterligare utredning för att helt säkerställa att alla krav uppfyllts.

Trots dessa brister har en god grund för fortsatt utveckling skapats och vi är nöjda med resultatet.

6.2 *Fortsatt utveckling*

De områden som står näst på tur att implementeras är många av de avgränsningar som gjorts tidigare och även vidareutveckling av det befintliga CANopen-nätverket.

6.2.1 CANopen över EtherCAT?

Då steget att bygga prototypen till att inkludera möjligheten att hantera och styra CANopen-nätverket och datorer/datorkort via Ethernet ansågs för stort och tidskrävande lämnades det till möjlig fortsatt utveckling. Ethernet for Control Automation Technology, EtherCAT är en nätverksteknologi som gör att stora delar av CANopens protokollstackar kan återanvändas. Den stödjer alla CANopens enhetsprofiler och utnyttjar PDO, SDO, NMT och objektlista. En möjlig vidareutveckling skulle därför kunna vara att implementera stöd för EtherCAT i den prototyp som utvecklats. Ett annat alternativ skulle vara att helt konfigurera om nätverket till EtherCAT. Visserligen skulle nätverket inte längre vara av typen CANopen men samma funktioner skulle finnas tillgängliga och master/slave-implementation möjlig. (Beckhoff, 2005)

6.2.2 Intelligent Platform Management Interface, IPMI

Själva styrningen av datorer/datorkort skulle kunna implementeras med hjälp av IPMI om det uppfyller krav på IA-säkerhet. IPMI kan ses som ett subsystem vilket fungerar helt oberoende av operativsystemet och ger operatörer tillgång till datorer/datorkort även om operativsystemet ej körs. Det kan alltså användas till att få tillgång till datorer/datorkort innan systemets primära mjukvara hunnit starta eller om det helt saknas. Datorn eller kortet behöver inte ens vara påslaget för att IPMI-tjänsten ska fungera, det räcker att enheten har ström och är inkopplad på bussen.

6.2.3 SAE J1939

Stöd för lastbilsstandarden SAE J1939 kan implementeras på den UCU som tagits fram. Ett oanvänt CAN-gränssnitt finns på enheten men en utredning av möjlig mjukvara skulle behöva göras, då den protokollstack som hittills använts har saknat stöd för denna standard.

6.2.4 Befintligt CANopen-nätverk

Ett område att utveckla, som endast skulle kräva mindre modifiering av aktuella program för att realisera, är att lägga till ytterligare slavnoder i nätverket. Det skulle bland annat möjliggöra bättre evaluering av systemet som helhet.

7 Referenser

Artila (2013). Embedded Networking and Computing
<http://www.artila.com/p_matrix.html>
(2013-06-08)

Beckhoff, Martin (2005). *CANopen over EtherCAT*
<http://www.can-cia.org/fileadmin/cia/files/icc/10/cia_paper_rostan.pdf>
(2013-05-31)

CANfestival (2001)
<http://www.canfestival.org/>
(2013-05-31)

CAN in Automation. *CAN history*
<http://www.can-cia.org/index.php?id=systemdesign-can-history#c137>
(2013-05-31)

CAN in Automation. *CANopen*
<http://www.kuebler.com/PDFs/Feldbus_Multiturn/CANopen/busprofil_CAN.pdf>
(2013-05-31)

CAN in Automation (2002). *CANopen Application Layer and Communication Profile*
<http://overpof.free.fr/schneider/CAN%20&%20CANopen/CANopen/%A9CiA%20
CANCANopen%20CD%20V5.1/standard/ds301.pdf>
(2013-05-31)

CAN in Automation. *CAN physical layer*
<http://www.can-cia.org/index.php?id=systemdesign-can-physicallayer>
(2013-05-31)

CAN in Automation. *CAN protocol*
<http://www.can-cia.org/index.php?id=systemdesign-can-protocol>
(2013-05-31)

CAN in Automation. *Controller Area Network*
<http://www.can-cia.org/index.php?id=systemdesign-can>
(2013-05-31)

Corrigan, Steve. (2008). *Introduction to the Controller Area Network (CAN)*
<http://www.ti.com/lit/an/sloa101a/sloa101a.pdf>
(2013-05-31)

ICPDAS (2013) *CANopen Master Library*
<http://www.icpdas.com/products/Remote_IO/can_bus/cpm_lib.htm>
(2013-06-08)

Mannisto, Daniel & Dawson, Mark. (2003). *An Overview of Controller Area Network*
<http://www.parallax.com/dl/docs/prod/comm/cantechovew.pdf>
(2013-05-31)

Microsoft Corp (2002). *The OSI Model's Seven Layers*
<http://support.microsoft.com/kb/103884>
(2013-05-31)

PEAK SYSTEMS (2013). *PCAN-USB*
<http://www.peak-system.com/Produktdetails.49+M5c6d753fe9e.0.html?&L=1&tx_commerce_pi1%5BcatUid%5D=6&tx_commerce_pi1%5BshowUid%5D=16>
(2013-06-08)

Pfeiffer, Olaf, Ayre, Andrew & Keydel, Christian (2003).
Embedded Networking with CAN and CANopen, San Clemente, CA : RTC Books

RT-Labs AB (2009). *CANopen*.
<http://www.rt-labs.com/embedded_fieldbus_canopen.shtml>
(2013-05-31)

Saab AB (2013). *Arthur WLR*
<http://www.saabgroup.com/en/Land/Force_Protection/Sense_and_Detect/ARTHUR_WLR_Weapon_Locating_Radar_System_Saab/Technical_specifications/>
(2013-05-31)

Saab AB (2013). *Giraffe AMB Surveillance System*
<http://www.saabgroup.com/en/Land/Ground_Based_Air_Defence/Ground-Based-Surveillance/Giraffe-AMB/Technical-specifications/>
(2013-05-31)

Singh, Deep, Akash (2010). *What is Ethernet – CSMA/CD*
<http://www.routemybrain.com/what-is-ethernet-csma-cd-introduction-to-osi-layer-model-the-internetworking-part3/>
(2013-05-31)

Sony Corp. *The Technology behind High-Speed Processing*
<http://www.sony.net/SonyInfo/technology/technology/theme/felica_01.html>
(2013-05-31)

Spencer, Will (2012). *Ethernet*
<http://www.tech-faq.com/ethernet.html>
(2013-05-31)

Wells, J, Christopher (2001). *Industrial Networks – Controller Area Network.*
<*http://www.technologyuk.net/telecommunications/industrial_networks/can.shtml*>
(2013-05-31)

Wikipedia (2013). *CANopen*
<http://en.wikipedia.org/wiki/CANopen>
(2013-05-31)

Wildpackets Inc. *Ethernet Packets and Protocols*
<http://www.wildpackets.com/resources/compendium/ethernet/ethernet_packets>
(2013-05-31)

8 Bilagor

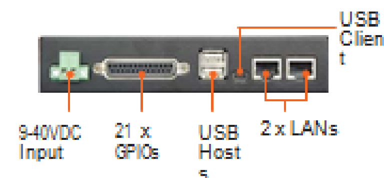
8.1 Appendix A – Artila Matrix 522

Matrix-522

Linux-ready, 400MHz ARM9-based Box Computer with 2 x CAN interface, 2 x Ethernet and 2 x RS-232/422/485 Ports

- ATMEL 400MHz ARM9 processor (AT91SAM9G20)
- Linux kernel 2.6.29 with file system
- 64MB SDRAM, 128MB NAND Flash, 2MB Data Flash
- MicroSD socket inside the box
- 2 x CAN bus ports support SocketCAN/CANOpen
- 2 x 10/100M Ethernet ports
- 2 x RS-232/422/485 serial ports
- 2 x USB hosts and 21 x GPIOs
- Less than 3W power consumption

Back View



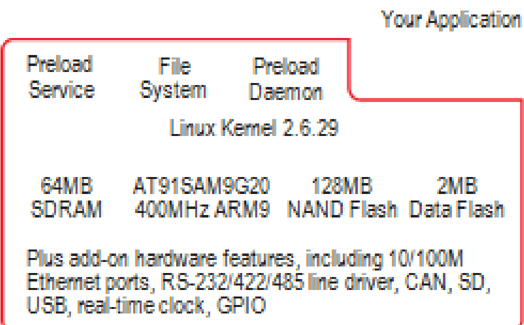
Overview

Artila's Matrix-522 industrial box computer is a small form-factor, low power consumption, and Linux-ready computing platform. With a 400MHz ARM9 CPU, 64MB SDRAM, 128MB NAND Flash, and 2MB Data Flash inside, Matrix-522 ensures system high performance. Matrix-522 is equipped with multiple I/O interfaces, including 2 LANs, 2 x RS-232/422/485 serial ports, 2 x CAN bus ports, 2 x USB hosts, and 21 x GPIOs. It is easy to develop pure C/C++ programs or Web-based applications (ex: SQLite+ PHP) to run on top of the Matrix-522.

The CAN bus (Control Area Network) is one of the dominating communication interface for embedded control systems both in automobile industry and factory automation industry. The Matrix-522 comes with two isolated CAN bus 2.0 compliant ports, supporting CANSocket and CANOpen APIs.

Matrix-522 also provides a useful fail-proof design for system backup and recovery. Normally, Matrix-522 boots up from its NAND Flash. However, the Data Flash in Matrix-522 includes a backup Linux file system as well, which will automatically boot up Matrix-522 in case the primary NAND Flash fails. If the Matrix-522 boots up from the Data Flash, a menu-driven program will appear to help users restore NAND Flash images from a USB pen drive or an SD card. Also, this program can help users make an custom image of the current file system to a USB pen drive or an SD card, so that users can easily recover the file system, or duplicate it to other Matrix-522s. Please refer to the user guide for more detailed information.

System Architecture



With pre-installed Linux OS, users can easily operate Matrix-522 via Telnet, or transfer files using FTP.

Matrix-522 comes with a free GNU tool chain for users to develop C/C++ programs. Comprehensive examples are also available in Matrix-522 product CD to demonstrate how to access I/O interface and hardware features.

Matrix-522's Linux environment is highly scalable and configurable. A variety of driver modules and software utilities are included in the product CD. Advanced customization service is also available upon request.

Hardware Specifications

CPU/Memory
CPU: ATMEL 400MHz AT91SAM9G20 w/MMU
Memory: 64MB SDRAM, 128MB NAND Flash
DataFlash®: 2MB, for system backup

Network Interface
Type: 2 x 10/100BaseT, RJ-45 connector
Protection: 1.5KV magnetic isolation

TTY(Serial) Ports
2 x RS-232/422/485
Connector: DB9 male connector

TTY(Serial) Port Parameters
Baud Rate: up to 921.6Kbps
Parity: None, Even, Odd, Mark, Space
Data Bits: 5, 6, 7, 8
Stop Bit: 1, 1.5, 2
Flow Control: RTS/CTS, XON/XOFF, None
RS-485 Direction Control: auto, by hardware

USB Ports
Host Ports: Two
Client Port: One, reserved
Speed: USB 2.0 compliant, supports low-speed (1.5Mbps) and full-speed (12Mbps) data rate

Digital I/Os (GPIO)
No. of Pins: 21 Pins
Signal Level: CMOS/TTL compatible
Each pin can be programmed as input or output.

CAN Bus Ports
Type: 2 x CAN bus 2.0A/B compliant ports
Speed: up to 1Mbps
Isolation: 2500Vrms
Connector: DB9 male connector

General
WatchDog Timer: Yes
Real Time Clock: Yes
Buzzer: Yes
Power Input: 9 to 40VDC
Power Consumption: 190mA@12VDC
Dimensions: 160 x 104 x 32mm

Ordering Information

Matrix-522	Industrial Linux-ready ARM9 CAN Bus Box Computer
CB-RJ45F9-150	RJ45 to DB9 Female Connection Cable, 150cm
CBL-F10M9-20	Serial Console Cable, 20cm
DK-35A	DIN RAIL Mounting Kit

Software Specifications

General
OS: Linux, Kernel 2.6.29
Boot Loader: U-Boot 1.1.2
File Systems: UBI, JFFS2, ETX2/ETX3, VFAT/FAT, NFS

Pre-installed Utilities
bash, busybox, sysvinit, wget, ipkg, procps (for webmin), psmics, lighttpd, vsftpd, iptable, ppp, ssh, wireless_tools, util-linux-mount/umount, usbutils, Artila utility, and more

Daemons Started by Default
ssh (secured shell) with sftp
syslog/klogd (system and kernel log)
telnet server (disable root with/etc/securetty)
ftp server (vsftp)
Web server (lighttpd)
amgrd (Artila broadcast search daemon)

Package Management & System Administration
Supports ipkg to manage the package installation, upgrade and removal.
Supports webmin (use ipkg install webmin to install) for web-based system administration.

Tool Chain for Linux
GCC: C/C++ PC cross compiler
GLIBC: POSIX Library

USB Host Drivers
Generic Flash drive
RS-232 adaptors (for Prolific PL-2303 compatibles)
WiFi dongles (rt2500/rtl8187/rt73/zd1211 chips)

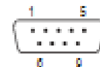
Pin Assignments

DB9 Male RS-232/422/485

PIN	RS-232	RS-422	RS-485
1	DCD	Rx-	—
2	Rx	Rx+	—
3	Tx	Tx-	Data-
4	DTR	—	—
5	GND	GND	GND
6	DSR	—	—
7	RTS	Tx+	Data+
8	CTS	—	—
9	—	—	—

DB9 Male CAN Bus

PIN	CAN
1	—
2	CAN_Lo
3	—
4	—
5	—
6	GND-ISO
7	CAN-HI
8	—
9	—



8.2 Appendix B – PEAK PCAN-USB

The PCAN-USB adapter enables simple connection to CAN networks. Its compact plastic casing makes it suitable for mobile applications.

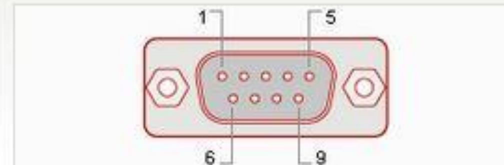
The opto-decoupled version guarantees galvanic isolation of up to 500 Volts between the PC and the CAN side.

The package is also supplied with the CAN monitor PCAN-View for Windows® and the programming interface PCAN-Basic.

Specifications

- Adapter for USB connection (USB 1.1, compatible with USB 2.0)
- USB voltage supply
- Bit rates up to 1 Mbit/s
- Time stamp resolution approx. 42 µs
- Compliant with CAN specifications 2.0A (11-bit ID) and 2.0B (29-bit ID)
- CAN bus connection via D-Sub, 9-pin (in accordance with CiA® 102)
- NXP SJA1000 CAN controller, 16 MHz clock frequency
- NXP PCA82C251 CAN transceiver
- 5-Volts supply to the CAN connection can be connected through a solder jumper, e.g. for external bus converter
- Extended operating temperature range from -40 to 85 °C (-40 to 185 °F)

Pin assignment D-Sub



Pin	Pin assignment
1	Not connected / optional +5V
2	CAN-L
3	GND
4	Not connected
5	Not connected
6	GND
7	CAN-H
8	Not connected
9	Not connected / optional +5V

8.3 Appendix C – Kommunikationsgränssnittmanual

Inledning

Denna manual beskriver vårt examensarbetes prototyp av ett CANopen-baserat nätverk med master-slave konfiguration. Prototypen består av två noder, en master och en slave, som kommunicerar med varandra enligt CANopen-protokollet CiA DS301.
Mastern ligger på en Artila Matrix 522 Linux-boxdator och kommunicerar med slaven, som ligger på en bärbar Linux-dator, via en Peak PCAN-USB-adapter. Linux-datorn vi använt oss av har en installation av GRAAL version LZYM2200015 R3B01. Insignaler kan till mastern sättas via GPIO-interface med hjälp av en switchbräda.

Denna manual gäller endast för dessa komponenter/system. Vi rekommenderar även att den ses som ett komplement till de befintliga manualer som finns för respektive komponent/mjukvara och bör alltså inte ersätta dessa.

Hårdvara

- Artila Matrix 522 Linux Box Computer
- Peak PCAN-USB adapter

Mjukvara

- CANfestival
- Python
- Drivrutiner
- NetBeans

Tillbehör

- Strömkontakt till Matrix 522
- CAN-CAN-sladd
- Switchbräda med GPIO-anslutning
- Ethernet-sladd
- USB-Serial-anslutning mellan Linux-datorn och Artila Matrix 522

Installationsanvisningar

I stort sett allt som behövs för att bygga upp utvecklingsmiljö och konfigurera vår master-slave prototyp finns på USB-minnet. Resterande delar kan införskaffas med ett verktyg som heter ”Zypper”.

CANfestival

Ett OpenSource CANopen-ramverk med fokus på att tillhandahålla en plattformsoberoende CANopen-stack som kan implementeras som master- eller slavnoder. En komplett installation av CANfestival på den datorn som agerar slav är ett krav.

Lägg över CANfestival till den datorn som ska agera slav.
Linux-kompilering och installation görs sedan med följande kommandon ståendes i CANfestival directory:

(./configure –help för att se hjälp-meny.)

./configure --timers=unix --can=socket
make
make install

Python

CANfestival kommer med ett verktyg kallat Object Dictionary Editor GUI. Det är ett WxPython Model-View-Controler baserat GUI som visserligen ej är nödvändigt för körning av denna prototyp men kan vara användbart om man vill generera/ändra Object Dictionary-källkod för noder. Om så är fallet krävs att Python med WxPython-moduler version 2.6.3 är installerade. För att göra detta kan man använda sig av verktyget ”Zypper”.

För att lägga till ett repository och installera python anges följande

zypper addrepo "http://ftp.gwdg.de/pub/linux/packman/suse/11.2" packman
zypper install python
zypper install python-wxGTK

För ytterligare anvisningar i användandet av detta verktyg, se den manual som kom med installationen av CANfestival.

NetBeans

På USB-minnet finns även en mapp vilken innehåller NetBeans för Linux. Vi använde oss av detta när vi programmerade och modifierade våra program, då vi tyckte det var mycket bättre och enklare än att använda oss av en vanlig text-editor. Framförallt felsökning och dylikt blev mer effektivt.

Peak PCAN-USB-adapter

Som namnet antyder är detta en adapter mellan CAN och USB. Det är genom denna som all kommunikation, slaven och mastern emellan, sker. I mappen PEAK, som ligger på USB-minnet finns förutom drivrutiner, även dokumentation och diverse verktyg. Drivrutinen som behövs hittas under

/PEAK/Develop/PC interfaces/Linux

Linux-kompilering och installation:

Packa upp filen *peak-linux-driver.x.y.tar.gz* i din hemkatalog enligt:

tar –xzf peak-linux-driver.x.y.tar.gz

Ett bibliotek saknas, därför behöver man installera rpm-paketet

popt-devel-1.7-37.46.15.i586.rpm

som ligger på USB-minnet. Detta görs genom att skriva

rpm –i popt-devel-1.7-37.46.15.i586.rpm

Drivrutinen kompileras för *netdev*-användning som standard(kernel 2.6.25 och upp) så det behöver inte sättas manuellt.

cd peak-linux-driver-x.y
make clean
make
make install

Därefter räcker det att skriva in

/sbin/modprobe pcan

för att installera drivrutinen och eventuella moduler.

Med nedanstående kommando kan vi ta upp vår nätverksenhet

ifconfig can0 up

För att se att drivrutinen har installerats korrekt, skriv in

cat /proc/pcan

du bör då se en utskrift som visar den installerade nätverksenheten.

Bithastighet kan sättas med kommandot

echo ”i 0x011C e” > /dev/pcanusb0 // 0x011C representerar här 250 Kb/s

Nästa steg är att installera socket-CAN paketet. Det är en implementation av CAN-protokoll för Linux. Just denna använder sig av Berkeley socket API, Linux network stack och implementerar CAN-drivrutiner som nätverksinterface(netdev). SocketCAN API:n har designats för att likna TCP/IP-protokollen.

SocketCAN-källkod kan nås genom

svn co http://svn.berlios.de/svnroot/repos/socketcan/trunk /var/du/vill/lägga/det

Om ovanstående kommando inte skulle fungera kan det bero på att det saknas en subversion av opensuse/SLE11. Om så är fallet följ anvisningarna nedan

zypper ar "http://download.opensuse.org/distribution/11.2/repo/oss" opensuse

detta kommer att lägga till ett repository så att subversionen kan installeras med kommandot

zypper in subversion

Du bör nu inte ha några problem att nå källkoden med hjälp av

svn co http://svn.berlios.de/svnroot/repos/socketcan/trunk /var/du/vill/lägga/det

make clean
make all

Se den tillhörande README-filen för instruktioner om kompilering och laddning av socketCAN.

Artila Matrix 522

Detta är en ARM9-baserad Linux-boxdator och de främsta fördelarna med denna är att den har två Ethernet I/O och två CAN I/O, har färdiginstallerat Linux OS och en lämplig GNU toolchain tillgänglig. Den har även två seriella portar, två USB-portar och en 25-pins GPIO-port.

På USB-minnet ligger en mapp, Matrix522, som innehåller allt från dokumentation till exempelprogram och toolchain. Kopiera över denna mapp till en Linux-dator. Installera sedan GNU toolchain genom att logga in som root-användare och kopiera

/Matrix522/toolchain/arm-linux-4.3.3.tar.bz2

till root directory. Under root directory anges följande kommando för att installera toolchainen

tar -xvf arm-linux-4.3.3.tar.bz2

Du kan nu bygga de olika exempelprogram som finns i Matrix522 för att sedan skicka över dessa till Artila Matrix 522. För att logga in på Artila Matrix 522 kan man tillämpa SSH-kommando.

Ethernet sladd mellan Linuxdatorn och Artila Matrix 522

För att kunna logga in på Artila Matrix 522, så att man kan styra den.

GPIO-anslutning för switchar och LED

Sex insignaler i form av switchar, hög, 3,3 V, vid "status"-begäran från mastern annars ska den vara jordad.
En utsignal i form av en grön LED, som är släckt vid error, annars tänd.

CAN-CAN-anslutning mellan PEAK PCAN-USB och Artila Matrix 522

Terminerad med hjälp av ett 120 Ω motstånd på PCAN-USB-sidan, färdigt terminerad på Artila Matrix 522.

USB-Serial-anslutning mellan Linuxdatorn och Artila Matrix 522

Om man vill komma åt serial-konsolen från Linuxdatorn.

Programkörning

Master

Vi har som utgångspunkt använt oss av exempelprogrammet

/Matrix522/Example/Canbus/canfestival/TestMasterSlave

och modifierat detta till

/Matrix522/Example/Canbus/canfestival/Exjobb

Då Artila Matrix 522 ska agera master används endast masterdelen av programmet och det är denna som vi har modifierat. Kortfattat så fungerar det som sådant att mastern sätter igång sig själv och därefter sätter den igång och styr slaven. Mastern skickar SYNC-signaler i ett visst tidsintervall till slaven, och vid varje SYNC-signal kollar mastern GPIO-insignalerna och om någon av dessa är höga tyder det på att en "status" begäran gjorts, så en PDO request skickas till slaven för att få "status"-värdet av slaven. Detta fungerar genom att masterns och slavens RPDO och TPDO matchar varandra. Om en error-signal tas emot av mastern ska den gröna LED som är kopplad på GPIO-utsignalen släckas, och tvärtom om error-signalen nollställs.

Följande kommandon anges för att logga in via Lan_1 till Artila Matrix 522:

User: root/guest
Password: root/guest
ifconfig eth0 192.168.2.1 netmask 255.255.255.0
ssh root@192.168.2.127

Under Exjobb directory anges följande kommandon för att bygga och föra över vårt program till Artila Matrix 522:

make clean
make all
scp TestMasterSlave 192.168.2.127:/home/guest/

Inloggad på Artila Matrix 522 anges följande kommando för att sätta upp och ställa in bithastighet på CAN0 så att den matchar Linux-datorns bithastighet:

ip link set can0 down
ip link set can0 type can bitrate 250000
ip link set can0 up
ip -details link show can0

Inloggad på Artila Matrix 522 under guest directory anges följande kommando för att köra programmet med enbart en master på CAN0:

./TestMasterSlave -l /usr/lib/libcanfestival_can_socket.so -s 0 -S none -m 0 -M 250

Inloggad på Artila Matrix 522 kan följande kommandon vara bra att använda till CAN portarna, vi använde oss framförallt av candump och cansend:

canconfig
candump
canecho
cansend
cansequence

Slave

Vi har som utgångspunkt använt oss av exempelprogrammet

/CanFestival/examples/TestMasterSlave

och modifierat detta till

/CanFestival/examples/Exjobb

Då Linuxdatorn ska agera slave används endast slavedelen av programmet och det är denna som vi har modifierat. Kortfattat så fungerar det som sådant att slaven sätts igång och styrs sedan av mastern. Vid varje SYNC-signal mastern skickar uppdaterar slaven sina "status"-värden genom att läsa av en fil, direkt därefter kollar slaven om värdena är okej, annars skickas ett error-meddelande till mastern. Mastern kan begära information om ett visst "status"-värde genom att skicka en PDO request och då svarar slaven med att skicka korrekt "status" värde tillbaka till mastern genom PDO.

Följande kommandon anges för att sätta upp och ställa in bithastighet på CAN0 så att den matchar Artila Matrix 522 bithastighet:

ifconfig can0 down
echo "i 0x011c e" > /dev/pcanusb0
ifconfig can0 up
cat /proc/pcan

Under Exjobb directory anges följande kommandon för att bygga och köra programmet med enbart en slave på CAN0:

make clean
make all
./TestMasterSlave -f /vart/filen/ar/placerad/testing.txt -l /usr/local/lib/libcanfestival_can_socket.so -s 0 -S 250 -m 1 -M none

testing.txt är en textfil med tre olika värden, ange rätt sökväg till denna.