YRKESHÖGSKOLAN
NOVIA

Styrning av motorkontroller med Siemens S7-1200 logik via gränssnittet CANopen

Philip Widjeskog

Examensarbete för ingenjörs (YH)-examen

Utbildningsprogrammet för elektroteknik

Vasa 2016

EXAMENSARBETE

Författare:	Philip Widjeskog
Utbildningsprogram och ort:	Elektroteknik, Vasa
Inriktningsalternativ:	Automationsteknik
Handledare:	Erik Englund

Titel: Styrning av motorkontroller med Siemens S7-1200 logik via gränssnittet CANopen

---

Datum: 5.12.2016 Sidantal: 50 Bilagor: 7

Abstrakt

Detta examensarbete behandlar hur man har gått till väga för att styra en motorkontrollerenhet som kommunicerar via ett gränssnittet CANopen. Som styrenhet används en Siemens PLC, med en tilläggsmodul som kommunicerar med CANopen. Man beskriver också hur man går tillväga för att kommunicera med en tredjeparts I/O-modul, samt en Siemens operatörspanel. Som programmeringsverktyg används bl.a. SIMATIC STEP7 och WinCC som är integrerade i verktyget TIA-Portal V13, vilket är de mest kända programmeringsverktygen inom industriell automation. För att få testat systemet gjordes en simuleringsapparatur som omfattade alla behövliga komponenter.

Syftet med detta arbete är att man skall ta fram en användbar lösning för att hantera dataöverföring mellan en motorkontrollerenhet och en PLC. Med hjälp av att använda CANopen-protokollet behöver man inte använda fysiska digital in-och utsignaler, som man använder i dagens läge. Detta leder också till att man sparar materiellt utrymme och får tillgång till ett bredare dataregister och en överskådligare felmeddelandehantering.

Resultatet av hela arbetet blev att man tog fram ett testprogram och skrev ett programblock som sköter parameter, processdata och felmeddelandehantering. Även ett programblock som hanterar felmeddelandevisualisering i operatörspanelen på ett vettigt sätt togs fram. Som programmeringsspråk användes till största del strukturerad kod (SCL) genom hela projektet.

Språk: svenska Nyckelord: Siemens, CANopen, Datakommunikation

---

BACHELOR'S THESIS

Author:	Philip Widjeskog
Degree Programme:	Electrical Engineering
Specialization:	Automation Engineering
Supervisors:	Erik Englund

Title: Control of Motor Controller With Siemens S7-1200 via CANopen Interface

---

Date: 5.2.2016 Number of pages: 50 Appendices: 7

Abstract

This master thesis deals with the control of a motor controller via a CANopen interface. The master control unit is a Siemens PLC with an add-on card for the CANopen interface. The thesis also describes how to communicate with a third-part I/O-module and a Siemens operating panel. Some of the programming tools used are SIMATIC STEP7 and WinCC, both integrated in the TIA-Portal v 13. These tools are among the most used programming tools in industrial automation. The system was tested in a simulator consisting of all the required components.

The aim of this thesis is to develop a sensible solution for data transfer between a motor controller and a PLC. Using the CANopen protocol the physical I/O's used in the current solution can be omitted. This means saving some space in the electric cabinets, more flexible I/O's and a possibility to develop a better error handling.

The result of the thesis is a program, including a functional block that handles parameters, processdata and error handling. The program also has a block for visualization of the error handling in the operating panel. The main programming language used in the project is Structured Control Language (SCL).

Language: Swedish Key words: Siemens, CANopen, Datacommunication

---

Innehållsförteckning

Bilageförteckning

Ordförklaring

PLC	Programmable Logic Controller även kallad programmerbart styrsystem på svenska.
HMI	Human machine interface eller människa-maskin-samspel på svenska. Den mest använda benämningen är ändå operatörspanel.
Användargränssnitt	Länk mellan operatören och programvaran eller själva hårdvaran. Inmatning av data som gör att användaren kan påverka systemet.
Logg	Sparar sampel med jämna mellanrum, för att senare kunna diagnostisera data trafik.
I/O	Fysiska in och utgångar på en PLC.
EDS	Electronic data sheet kan översättas till elektroniskt datablad på svenska. Används för att kunna kommunicera mellan en slav och en master på ett CANopen-nätverk.
OB	Organisations block.
FC	Function (utan minne).
FB	Function block (med minne).
SCL	Structured Control Language. Ett strukturerat kodspråk som används för programmering.
FBD	Function Block Diagram. Ett programmeringsspråk som baserar sig på olika block som uppfyller olika matematiska funktioner.
CAN	Controller Area Network är en buss standard för fordon. Kan kommunicera utan en värddator.
PDO	Processdata Object. Ett CAN-paket som innehåller process-data.
SDO	Service Data Object. Service meddelande. Används för felsökning och konfigurering.
NMT	Network Management. En pakettyp i CANopen som kan definiera hur nätverket startas och stoppas.
Pre-Operational	Konfigurationsläge för CAN-nätverket. Endast SDO paket skickas, PDO är förbjudna.
Operational	Normalläge för CAN-nätverket, alla datapaket är tillåtna.

1 Inledning

Detta examensarbete behandlar hur man har gått tillväga för att förnya kommunikationssätt mellan en motorkontroller och PLC, från I/O till CANopen fältbusskommunikation. Första delen som behandlas är teorin bakom CANopen-standarden. Där ges en överblick i vad som skiljer sig mellan vanlig I/O-kommunikation och fältbuss. Vidare beskrivs tillvägagångssättet, hur man har gjort för att förverkliga detta. Som avslutning till detta presenteras en diskussion och ett resultat av hela arbetet.

1.1 Bakgrund

På Ab Solving Oy har man en längre tid använt sig av analoga och digitala signaler för att styra sina större luftkuddetruckar. I och med att en ny Siemens logik serie har lanserats, finns det nu möjlighet att vidareutveckla styrsystemen i de större luftkudde truckarna. Användning av CANopen gränssnittet som kommunikationsprotokoll istället för I/O-kommunikation, skulle ge mera utrymme i apparatskåpen, innebära mindre kabeldragning samt erbjuda ett bredare dataregister för behandling av data. Övervakning och konfigurering av systemet skulle samtidigt bli betydligt smidigare. Man har redan en längre tid använt sig av CANopen-styrning i Solvings förarlösa truckar. Där kommunicera vagnsdatorn med CANopen flytande.

1.2 Uppdragsgivare

Uppdragsgivaren till detta examensarbete är Ab Solving Oy. Solvning har etablerat sig i Sandsund, Finland år 1977 för att sälja luftkuddebaserade bas system. Produkten har successivt utvecklats för att skapa flera sofistikerade lösningar och för att kunna möta mer krävande applikationer på en mängd olika marknader. Ab Solving Oy har specialiserat sig på att tillverka materialhanteringssystem som är skräddarsytt för kundens ändamål. Företaget är idag ledande inom sin bransch och har utrustning över hela värden. Under 1990-talets början grundades ett dotterbolag i Sverige vid namn Solving Sweden. Därefter har man börjat bygga upp ett ”Team Solving” som idag består av dotterbolag i Sverige, Italien, Tyskland och England. Sedan 2008 äger Solving Fluid-Bag, i Jakobstad till hundraprocent. Företaget har idag en omsättning på ca 27 miljoner euro och omkring totalt 148 anställda.

1.3 Målsättningar

Målsättningar är flera och kan delas upp i följande delmoment:

- Ta fram en lämplig EDS-fil för motorkontrollern.
- Färdigställa ett testsystem bestående av en logik, CANopen modul, Tredjeparts I/O-modul, motorstyrning samt operatörspanel.
- Konfigurera CANopen-modulen med hjälp av tillhörande mjukvara.
- Skriva behövlig programkod för att kommunicera med motorkontrollern.
- Skapa lämpligt användargränssnitt till HMI-panelen, där man kan styra, monitorera processvärden och läsa felmeddelanden.
- Skapa en instruktion som gör att en annan applikator enkelt kan få en ny CANopen-nod att fungera.

Under hela projektets gång ska målet vara att man, skriver en tydlig programkod, dokumenterar och gör en tydlig programstruktur. Allt detta för att möjliggöra att en ny applikator enkelt skall kunna ta i bruk ett nytt system av denna typ.

2 Vad är datakommunikation?

I alla tider har människor kommunicerat med varandra på långa avstånd med hjälp av olika system, till exempel med hjälp av röksignaler, trummor, vandrings-boskaps-stigar, att ropa, vissla eller springa. De flesta kulturer har haft någon typ av system för att kommunicera med på längre distanser. När man pratar om olika kommunikations metoder brukar man använda begreppet telekommunikation i vissa sammanhang. Ordet *tele* kommer ursprungligen från det grekiska språket och betyder ”på långt håll”. Begreppet telekommunikation behandlar all kommunikation på längre distanser mellan både datorer och människor. Ofta använder man begreppet *datorkommunikation* när man pratar om kommunikation mellan datorer och för telefoni och liknande tjänster brukar man använda ordet *telekommunikation*. För att man skall ha ett gemensamt ord som representerar både *datorkommunikation* och *telekommunikation* har man börjat använda begreppet *datakommunikation*, eftersom att det är data som skickas i båda fallen. [1][2]

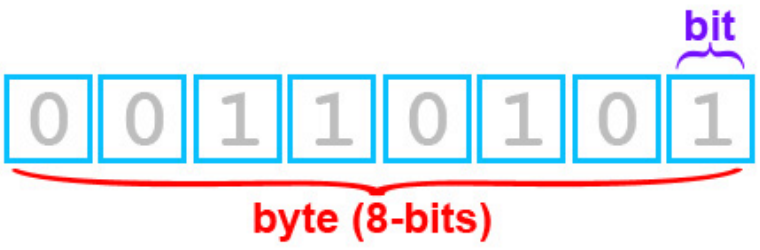
Varje datakommunikations system består huvudsakligen av tre olika komponenter *sändare, medium* och *mottagare*. Sändaren är normalt den källa där data uppstår, medium är det medel som data använder sig av för att ta sig fram till mottagaren. Mottagaren används också för

att markerar slutet på dataöverföringen. Dessa tre komponenter skulle man i dagens läge kunna jämföra med Internet, bredbandsförbindelse och en persondator. Internet servern fungerar som en *sändare* som sänder data till en persondator, som fungerar som en *mottagare*. För att data skall komma fram till persondatorn använder vi en bredbandsförbindelse, som fungerar som ett *medium*. [2]

Figur 1 Huvudsakliga komponenter i datakommunikationssystem.

I dagen läge finns det två olika typer av datakommunikation, *analog* och *digital*. Begreppen analog och digital beskriver hur man för över data från punkt A till punkt B. Nästan all datakommunikation i dagens läge har övergått till digital överföring av data. Därför kommer man i detta lärdomsprov enbart att koncentrera sig på att beskriva vad som menas med digital datakommunikation. Men för att man skall få en liten överblick på var man har använt analog datakommunikation kan man nämna att telefonen och televisionen använde sig av analog datakommunikation för ca 10 år tillbaka, sedan har man övergått till digital datakommunikation. [2]

När man pratar om digital datakommunikation handlar det om att överföra ettor och nollor från punkt A till punkt B. När man behandlar ettor och nollor använder man ofta engelska begreppen *bit* och *byte*, t.ex. överföringshastighet mellan punkt A och punkt B mäts i bitar per sekund (bit/s) och lagringen av en data mängd mäts i uttrycket *byte*. För att förklara var detta används i verkligheten kan man nämna att vid en knapp tryckning på ett tangentbord på en vanlig PC skapas en viss kombination av åtta stycken ettor och nollor, denna data mängd lagras i en byte. En bit kan endera vara en etta eller en nolla och en byte har utrymme för åtta stycken bitar. [2][3]



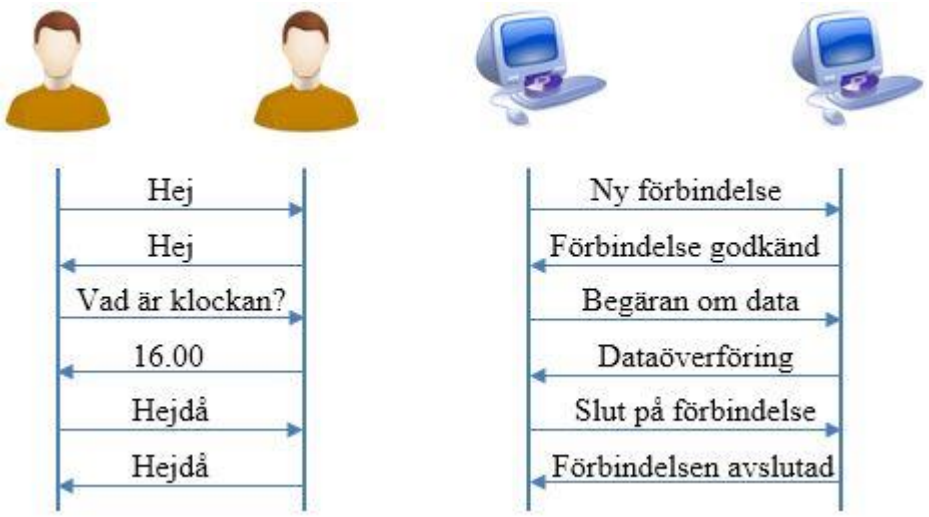
Figur 2 Skillnad mellan bit och byte.

3 Principer för datakommunikation

I detta avsnitt behandlar man dom mest centrala begreppen inom datakommunikation. Begreppen man tar upp här hör till grundprinciperna inom datakommunikation. Genom att förstå dessa begrepp kan man få en bild av hur ett datakommunikationssystem kan var uppbyggt och förstå dess grunder.

3.1 Protokoll

För att man skall kunna kommunicera med två datorer krävs det att dom pratar samma språk. I människans värd använder man begreppet ”språk” för att klargöra hur man vill få fram sin information. I datorvärlden har man valt att använda ordet ”protokoll” för att få fram information (data) mellan två parter. Så man kan säga att språk och protokoll är precis samma sak men används vid olika situationer. I datakommunikationsprotokoll har man implementerat regler som gäller vid dataöverföring. Liknande regler kan man finna i människans vardagliga språk. [1]



Figur 3 Kommunikation mellan datorer och människor.

I ett protokoll finner man olika slags funktioner. Alla funktioner fungerar olika i olika slags protokoll. Har man två utrustningar med olika slags protokoll, kan utrustningarna inte kommunicera med varandra, eftersom dom inte förstår varandra. Funktioner som ett protokoll vanligtvis måste kunna hantera är: [2]

- Dialog, i vilken ordning man sänder data meddelanden.
- Adressering, hur man berättar vem man är och vilka man vill kommunicera med.
- Ramformat, hur ett datameddelande skall se ut.

- Felhantering, hur man skall hantera situationer där data har blivit felaktigt sänt eller mottagit. [2]

3.2 Standard

Inom Europa använder man sig oftast av ISO-standarden, men det finns också andra nationella och internationella standardiseringsorgan. Det finns många fördelar med att använda sig av en standard när man tillverkar en produkt bl.a. produktens säljbarhet samt livslängd ökar och man får en billigare produkt på marknaden pga. ökad konkurrens. Om flera tillverkare använder sig av samma standard kommer också kompletteringsmöjligheterna att öka. Det finns inte bara positiva saker med standardisering, det finns också nackdelar. Vill man vidareutveckla en produkt kan det låsa sig i.o.m. att den har blivit låst mot en standard. Pga. komplicerade standarder kan också prestandan bli lidande. [3]

I dagens läge är nästan allt vi använder oss av i vardagen standardiserat t.ex. toaletter, spisar, tv-apparater, sängar och tvättmaskiner. Inom datakommunikationen krävs det att man använder sig av samma protokoll för att man skall kunna kommunicera mellan två enheter. Därför finns det skilda organisationer och forum ute i värden som kommer överens om olika protokollstandarder som man använder sig av inom datakommunikationen. [1]

3.3 Nätverksdesign

Nätverksdesign handlar om hur man bygger upp ett nätverk, vilka komponenter, operativsystem, kablar och protokoll man använder. Ett nätverk har ett antal faktorer man bör ta i beaktande när man bygger ett nätverk. Faktorerna som påverkar datanätverkets egenskaper är: [2]

Skalbarhet – Är en faktor som betyder att man skall kunna ändra på ett nätverk utan att behöva ändra på dess befintliga system. Detta är en faktor som också kan påverkas av den ekonomiska faktorn. En bra nätdesigner skall kunna kompromissa mellan denna faktor och den ekonomiska.

Anpassningsbarhet – Denna faktor handlar om hur man anpassar anläggningen för framtida ändamål. Om man t.ex. vill byta kommunikationstjänster vill man kunna göra det utan att behöva ändra grunddesignen nämnvärt.

Kostnadskontroll – Denna faktor kan vara den som är den mest problematiska. Man kan säga att det är denna faktor som styr hur systemet kommer att börja se ut. Oftast har man en budget när man bygger ett system. Är budgeten för liten t.ex. på grund av fel kostnadskalkyleringar kommer man att behöva spara på systemets egenskaper, vilket påverkar alla andra faktorer inom nätverksdesignen.

Enkelhet att hantera och felsöka – Denna faktor handlar om att systemets grunddesign skall vara gjord så att det är enkelt att felsöka och allmänt hantera systemet. Stora system kan bli väldigt komplexa och protokoll svåra att hantera ifall man har missat på denna faktor. En bra tumregel kan vara att man försöker hålla ner antalet protokoll som kommer att användas inom samma system.

Säkerhet – Denna faktor hanterar säkerheten i ett system. I nästan alla organisationer i dagens läge fungerar datornätverk som ett stort nervsystem i anläggningen. Redan i ett tidigt läge bör man tänka på säkerheten för att förhindra att obehöriga tar sig in på nätverket. Identitetshandlingar och brandväggar är mycket viktiga för att detta skall hanteras professionellt. [2]

4 ISO/OSI – modellen

För att alla datorer skall kunna kommunicera med varandra oberoende av tillverkare eller operatör, ansåg man i mitten av 1970-talet att man behöver göra en gemensam kommunikationsstandard. *International Organization for Standardization* (ISO) började med att ta fram en gemensam arkitektur, som 1983 kunde presenteras som *Open System Interconnection* (OSI). Referens modellen kunde presenteras som ett resultat av detta arbete. Referens modellen OSI blev ett ramverk som man använder sig av för att definiera standarder mellan olika datakommunikationsnät. [2][1]

OSI-modellen blev sist och slutligen inte betraktad som en standard, utan man använder den som en referensmodell för att förklara hur man skall bygga upp ett kommunikationsnät. OSI-modellen har funktionen som en röd tråd mellan olika tillverkares system. [2]

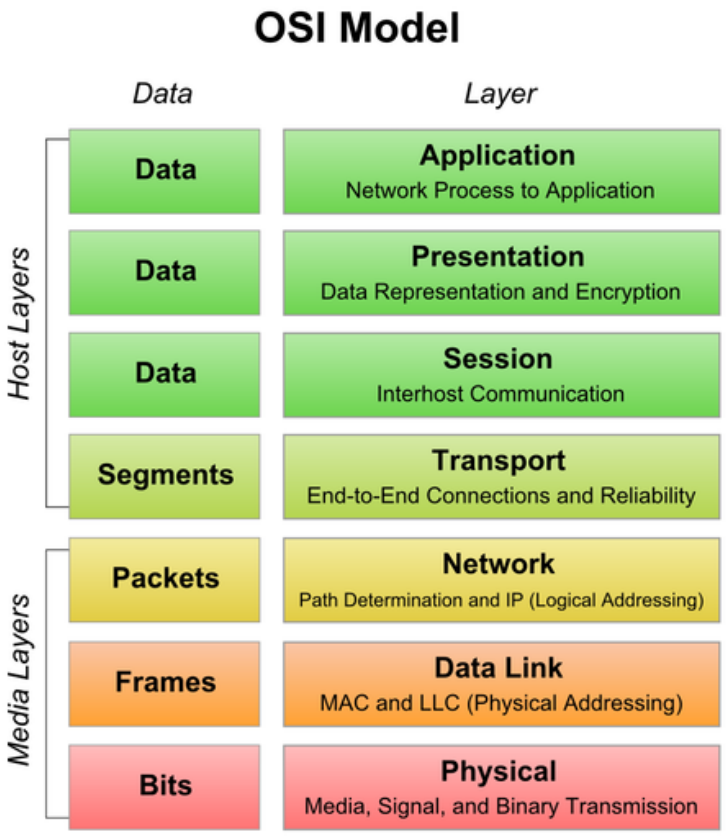


Figure 4 OSI-modellens alla sju skikt.

Modellen består av sju skikt (layer) eller lager där varje skikt har en egen specifik uppgift. I sin tur så kommunicerar varje skikt med ”andra sidans” motsvarande lager. De digitala signalerna man sänder mellan olika datorer måste kunna tolkas till olika tecken när de kommer fram till en enhet, binära ettor och nollor får inte heller ändra tillståndsläge under

färden mellan olika datorer. För att detta skall fungera i verkligheten använder man sig av ett kontrollregelverk som ser till att data kommer fram i exakt samma ordning som den har blivit sänt. Lagren i OSI-modellen kan delas upp i två kategorier, skikt 1–4 representerar nätverksorientering och skikten 5–7 står för användarorienteringen. [2]

- **Fysiska skiktet** är själva överföringsmediet. Detta kan vara en kabel eller radiolänk. Detta skikt ser även till att sändare och mottagare är synkroniserade och att överföringshastigheten blir korrekt.
- **Länkskiktet** innehåller regler hur data skall placeras i ramar så att mottagaren skall kunna tolka vad de mottagna bitarna betyder. Länkskiktet fungerar också som en länk mellan de två intill liggande skikten. Och för att dataöverföringen skall bli tillförlitligt finns det också regler implementerade för felhantering och flödeskontroll i detta lager.
- **Nätverksskiktet** hanterar data transporten mellan ett eller flera nät. Skiktet sköter även om adressering så att datapaketen kommer fram till rätt mottagare, oberoende av vilket nät mottagaren är kopplat till. Det finns även implementerat funktioner som hanterar att data hittar den lämpligaste vägen genom nätet.
- **Transportskiktet** upprätthåller kommunikation mellan olika processer både i inkommande och utgående riktning. Detta lager innehåller även en felhantering och flödeskontrol funktion för att ta hand om det fel som inte de undre lagren 1-3 har klarat av att eliminera.
- **Sessionsskiktet** används när det är frågande om två eller flera datorer som behöver synkronisera med varandra. Skiktet innehåller även regler för hur man sätter upp dialoger, samt hur dom synkroniseras och underhålls, t.ex. vilka riktningar man kan sända data, om två datorer kan sända samtidigt eller om dom måste turas om.
- **Presentationsskiktet** konverterar om data från sändaren till binär data som mottagaren kan ta emot och tolka på ett korrekt sätt. Skall data krypteras eller komprimeras finns det funktioner som presentationsskiktet använder sig av för att göra detta.
- **Applikationsskiktet** är det lager som ser till att användaren kommer åt nätet, vilket kan vara en människa eller programvara. Detta lager är själva användargränssnittet t.ex. HMI eller en webbläsare. [1][2][3]

För att förstå hur OSI-modellen fungerar rent praktiskt har man nedan konverterat om datakommunikation till dialoger mellan människor. Detta för att enkelt förstå hur meddelanden byggs upp i varje skikt.

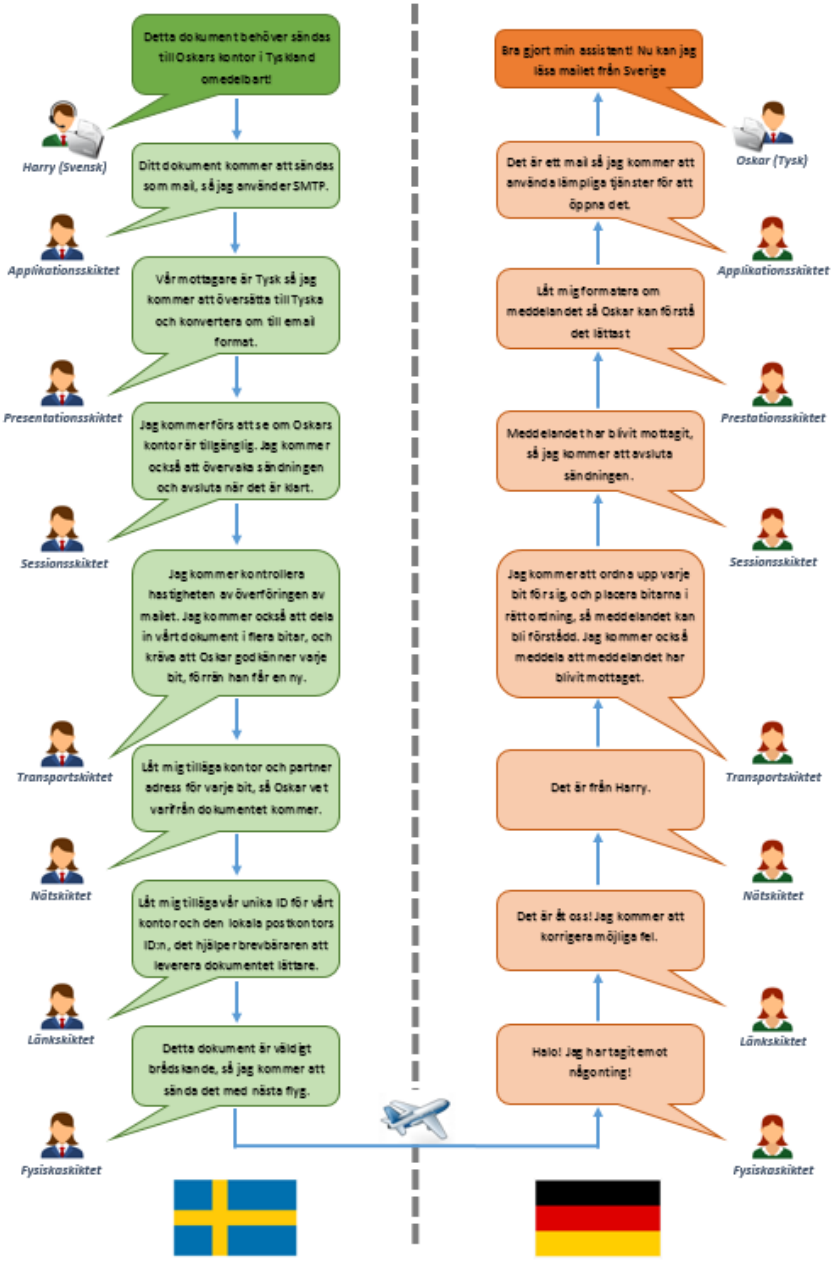


Figure 5 OSI-modellen konverterad om till dialoger mellan människor.

5 Nätverkstopologier

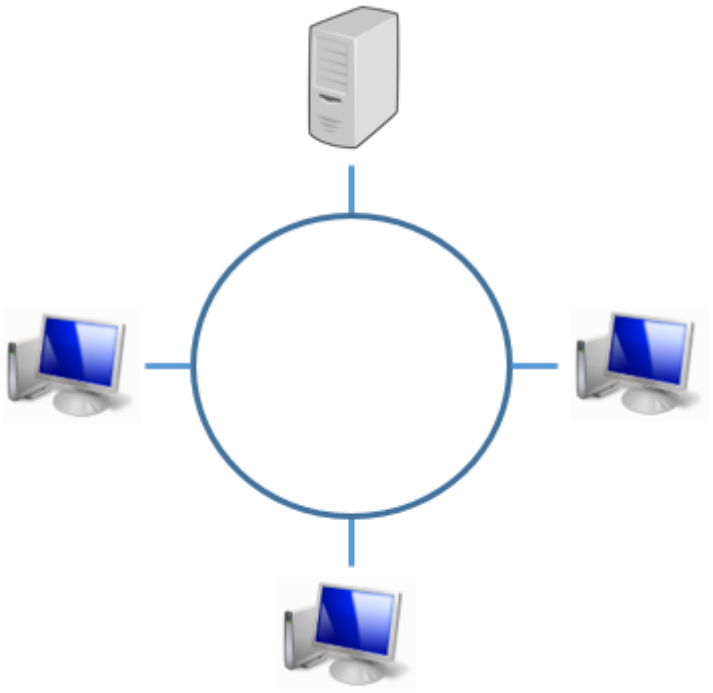
Inom datakommunikation beskriver begreppet topologi hur man datanätverket är sammanbundit rent fysiskt. Den fysiska topologin är det som man ritar upp på ett papper när man beskriver hur utrustning är sammankopplat rent fysiskt med en kabel på ett nätverk. Logiska topologin bestämmer hur kommunikationen flödar mellan olika enheter på nätverket, därför är det inte lika lätt att rita upp på ett papper funktionen hos denna. Den fysiska topologin kan vara sammanlänkad på följande sätt: [2]

- Bussnät
- Ringnät
- Stjärnnät

Bussnätet består av en lång nätverkskabel man hänger på utrustning enligt behov. Denna metod lanserades i början av 1970-talet och användes för den lokala nätverksstandarden *Ethernet.* I bussnätet kan alla enheter ta emot data samtidigt men endast en enhet kan sända i taget. Datalänksprotokollet är det lager i OSI-modellen som bestämmer hur man får sända data på den gemensamma kabeln. I bussnätet är det viktigt att alla enheter har en egen ID så dom kan känna igen när ett meddelande har blivit adresserat till sin egen adress. För att få ett fungerande bussnät behöver bussen vara ansluten med termineringsmotstånd både i början och slutet av bussen. Ett problem med denna buss är att får man ett avbrott på huvudkabeln stoppas all trafik på bussen. CANopen som man behandlar i detta lärdomsprov använder sig av denna nätverkstyp. [2][3]

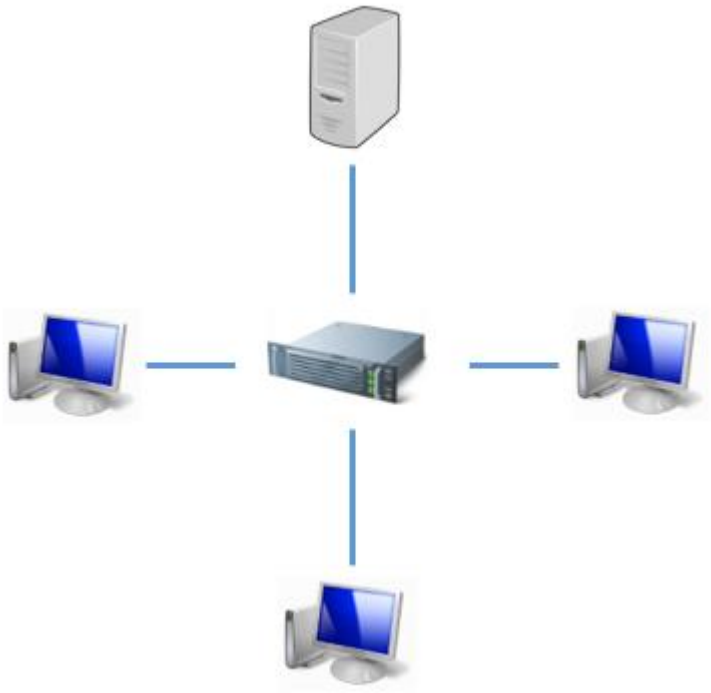
Figur 6 Fysisk sammankoppling av ett bussnät.

I **ringnätet** går den långa nätverkskabeln i en ring som alla kommunicerande enheter är anslutna till. Varje enhet behöver ha två anslutningar, för att kunna ta emot och sända data. Också detta nät är känsligt för avbrott av huvudkabeln. Det är vanligt att leverantörer av ringnät använder sig av dubbla parallella ringar som går bredvid varandra för att undvika avbrott. All data man sänder i detta nätverk går igenom varje enhet och förs sen vidare till bredvidliggande enhet. Nackdelar med detta nätverk är att nätet kan vara svårt att bygga ut och störningskänslig och som man nämner ovan kan hela nätverket slås ut om man får ett avbrott i ringen. [2][3]



Figur 7 Fysisk sammankoppling av ett ringnät.

Stjärnnätet använder sig av en växel (switch) som kopplingspunkt. Enheterna runt växeln är direkt anslutna till switchen, vilket gör att denna nätverks typ inte är direkt känslig för avbrott. Blir det avbrott på nätverkskabeln är det endast den enhet som kabeln är kopplad till som faller bort. Istället har detta nätverk sin svaga punkt i växeln. Går växeln sönder så slås hela nätverket ut. Denna nätverkstyp är vanligt för hemmanätverk och mindre kontorsnätverk. Jämfört med stjärn och bussnätverket går det åt mycket mera kabel för att göra ett stjärnnätverk i.o.m att alla enheter har samma knutpunkt. [2][3]



Figur 8 Fysisk sammankoppling av ett stjärnnät.

5.1 Användarmodeller

Olika nätverk använder sig av olika kommunikationssätt. Med hjälp av de olika användarmodellerna klargör man vad var och en får göra på ett nätverk. T.ex. om man får sända och ta emot data, eller om man enbart får ta emot data. Vissa användarmodeller tillåter t.ex. endast att man får sända data om man har tillåtelse av en master enhet. Exempel på denna användarmodell är *Master/Slave* kommunikationssättet. [4]

Klient/Server – Klient/Server modellen kan man säga att är den vanligaste modellen inom datakommunikation. Denna modell används bl.a. för internetanvändning. När man använder sig av denna modell så är det de kommunicerande ändpunkterna som kallas klient respektive server. Den enhet som kallas klient hämtar data från en server. Servern kan exempelvis vara en webbserver som förser en klient med webbsidor. En session mellan klient/server modellen består utav att klienten sänder en förfrågan till servern, därefter förväntar sig klienten ett svar i form av data. När sessionen är slut och klienten har fått sitt svar avslutas sessionen i normala fall såvida inte klienten har flera förfrågningar som ligger och väntar. [2]

Peer to Peer (P2P) – I denna modell är alla kommunicerande applikationer likvärdiga. Här finns det inga speciella servrar inblandade. Det som skiljer från föregående kommunikationsmodell och peer to peer modellen är att denna modell saknar rollfördelningen mellan klienter och servrar. Ändpunkterna i denna modell fungerar som

både klient och server och är oberoende av utomstående krafter. Inom P2P modellen finns det protokoll och metoder hur den skall arbeta för att hitta det material den behöver. Eftersom materialet den behöver är utspritt på andra klienter i nätverket krävs det sådana metoder och protokoll.[2][3]

Master/Slave – När man pratar om ett Master/Slave nätverk behandlas mastern som ett kontrollvärktyg över slavarna. Det är mastern som ger tillåtelse till slavarna att prata. I ett Master/Slave nätverk kan inte slavarna normalt kommunicera med sig själva eller med varandra. Det är bara mastern som kan inleda en kommunikation och slavarna svarar endast när dom har fått en förfrågan av mastern att börja kommunicera. Eftersom man alltid behöver data i båda riktningarna, en förfrågan från mastern och ett svar från slaven när man använder Master/Slave modellen betyder detta att man kommer ha dubbelt mer data ute på nätverket, än om man skulle använda sig av ett direkt kommunicerande system. Detta gör att det är effektivare att använda sig av t.ex. Peer to Peer modellen. [4]

Producer/Consumer – Producer/Consumer användarmodellen används sällan som huvud kommunikationssätt i ett nätverk, utan ett nätverk som t.ex. använder sig av Klient/Server modellen som huvudkommunikationssätt kan använda sig av Producer/Consumer-modellen för att kommunicera med en viss sort data t.ex. service eller felmeddelanden. Producern överför data till ett nätverk och consumern tar emot data från samma nätverk. För en specifik uppsättning data kan det bara finnas en producer och åtminstone en, men möjligen flera consumers för samma datameddelande. [4]

Som tillägg av de olika användarmodellerna kan man tilläga att det går att använda sig av flera av de ovannämnda modellerna i samma nätverk, man är inte begränsad av att använda sig utav endast en modell. I ett nätverk där man använder olika typer av data meddelanden kan man binda ihop olika grupper av datameddelanden med en specifik användarmodell, som lämpar sig bäst för den gruppen av data. Ett exempel på var man har gjort detta är i nätverksprotokollet CANopen. Där använder man sig av Klient/Server modellen när man tilldelar nätverket åtkomstpunkter, Producer/Consumer-modellen används för att synkronisera nätverket och Master/Slave modellen för servicemeddelanden. [4]

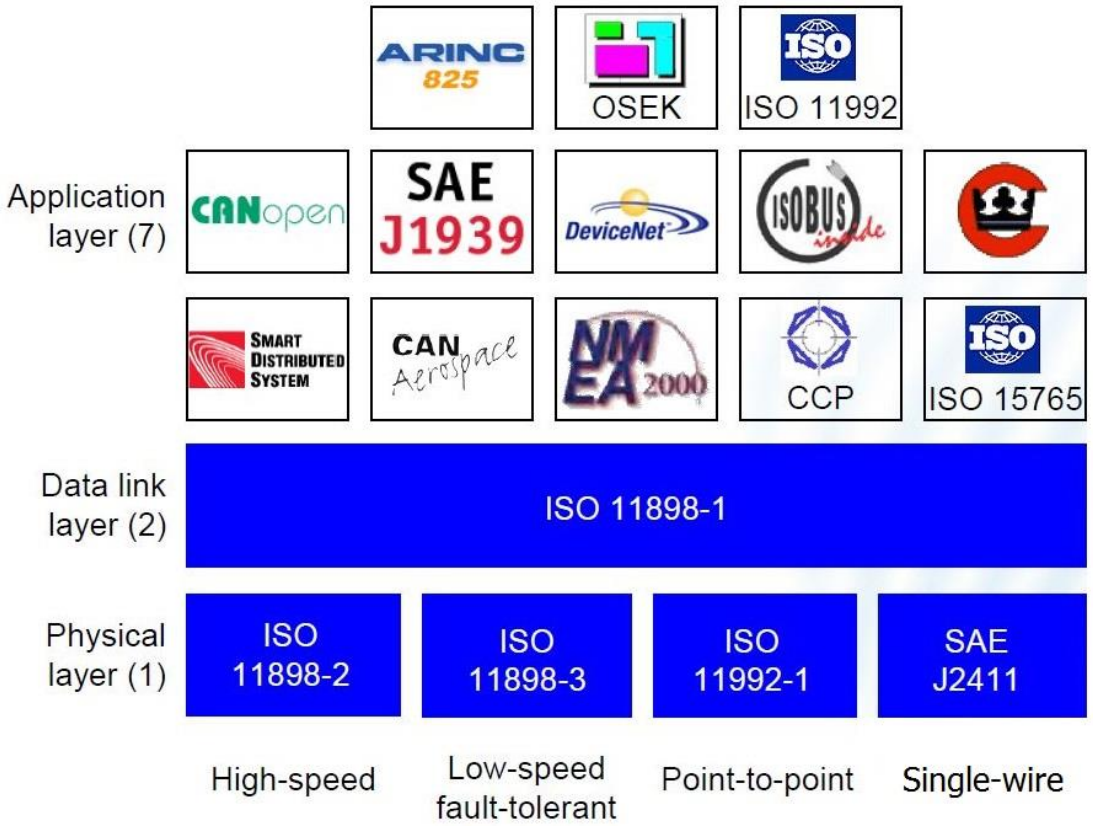
6 CAN och CANopen

I detta kapitel kommer det förklaras vad CAN och CANopen är. På grund av att CANopen är huvudämnet för detta lärdomsprov kommer man att hoppa över alla andra högreskiktsprotokoll som kan användas med CAN-standarden. Vad ett högre skiktsprotokoll är behandlas nedan.

6.1 CAN-standarden

CAN-nätverket är från början en seriell fältbuss som har tagits fram av Robert Bosch år 1986, för att användas inom fordonsbranschen. I dagens läge så innehåller nästan varje personbil som har blivit tillverkad i Europa en CAN-fältbuss. Även i andra typer av fordon används denna standard, såsom i tåg och i fartyg. Inom industrin används denna typ av fältbussar flitigt, oftast i mobila utrustningar, men även i industrifastigheter.[4]

Som man beskriver ovan utformades CAN ursprungligen för att användas inom fordonsindustrin. Under de senaste årtiondet har biltillverkare samt tillverkare utöver bilindustrin börjat med att tillverka sina egna mikrokontrollers baserade på CAN-standarden. Detta skapade ett krav på ett öppet standardiserat högreskiktsprotokoll, som skulle ge ett tillförlitligare datahanteringssystem.[4]



Figur 9 CAN-standarden är placerade i OSI-modellens två första lager, fysiska och länkskiktet.

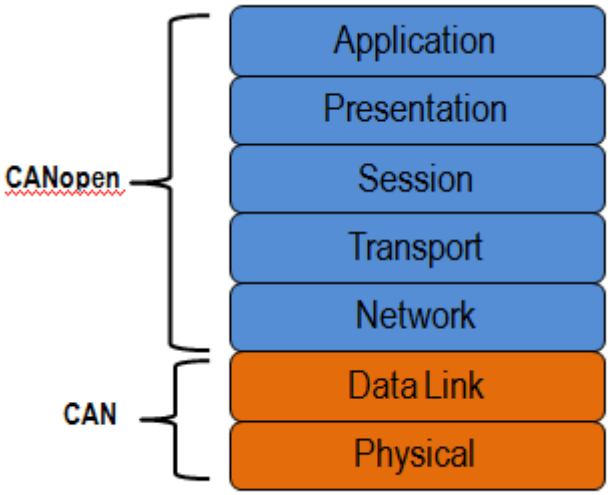
I samband att man tog fram ett högreskiktsprotokoll framkom bl.a. DeviceNet, CANopen, SAE J1939. Varje protokoll fick egna användarområden t.ex. SAE J1939 används inom lastbil och båt branschen, DeviceNet och CANopen inom industriell automation och tunga fordon. Man kan också använda flera CAN protokoll inom samma utrustning, t.ex. på ett fordon kan man använda CANopen till hydraulik och sensorer och SAE J1939 till motor och drivlina, detta pga. av att protokollen är anpassade för att tillämpas inom olika områden. [4][5]

Vilket kan ses i figur 9 använder CAN-standarden sig av endast två lager i OSI-modellen, länkskiktet och fysiskaskiktet. Högreskiktsprotokoll som kommer att användas tillsammans med CAN-standarden är placerad i applikationsskitet. [4]

6.2 CANopen-protokollet

I detta projekt kommer CANopen-protokollet att användas som kommunikations medel mellan en motorkontroller och ett CANopen kommunikations interface, som numera finns tillgänglig för den nya Siemens S7-1200 logikserien.

I fortsättningen av detta lärdomsprov när CANopen-protokollet behandlas kommer begreppet *node* användas. Node beskriver en enhet i ett nätverk som har en unik nätverksadress. En node kan t.ex. vara en PC, skrivare, gateway eller någon annan nätverksenhet som används i ett nätverk. [3]



Figur 10 OSI-modell för CANopen-protokollet.

CANopen är ett CAN-baserat datakommunikationssystem. CANopen-profilen består av ett högreskiktsprotokoll samt profilspecifikationer. I OSI-modellen använder sig CANopen-protokollet, vilket är ett högreskiktsprotokoll av applikationslagret. CANopen har utvecklats som ett standardiserat inbyggd datakommunikationssystem med mycket flexibla konfigurationsmöjligheter. Profilen designades ursprungligen för rörelseorienterade maskinstyrningssystem. Idag används protokollet inom olika tillämpningsområden, såsom medicinsk utrustning, terrängfordon, sjöfart, elektronik, järnvägstillämpningar eller fastighetsautomation. [4]

Inom CANopen så finns det tre olika typer av ”identifierare” Objekt index, Node-ID och COB-ID eller CAN ID. En av utmaningarna med CANopen är att förstå skillnaderna mellan olika identifierare. Eftersom protokollet bygger på att använda identifierare och objekt index för att kommunicera, är det oerhört viktigt att förstå skillnaden mellan dessa.[4]

- Node-ID begreppet används för att identifiera en CANopen node på ett nätverk. Totala användbara noder på ett nätverk kan vara från 1 – 127 stycken.

- Objekt index används för att identifiera specifika variabler inom en Node. Objekt index kan användas tillsammans med PDO-eller SDO-datameddelanden.

- COB-ID vilket på engelska betyder ”Connection Object ID”. I ett CANopen-nätverk är COB-ID unikt och används för att kommunicera med en nod till en annan, eller till flera noder via en specifik kommunikationskanal. Till exempel ett stop meddelande som skall stänga ner bussen kan användas på ett COB-ID. Även styrbitar så som en aktiverings eller in aktiverings bit kan inkluderas i meddelandet. [4]

CANopen använder sig ofta av definitioner som involverar termen ”objekt”. Dessa objekt används delvis för att lagra eller sända data.

- Object Dictionary (OD) (*Bilaga A.2*) används för att spara process och konfigurationsdata till en sorts ”look-up-table”.

- Processdata Object (PDO) (*Bilaga A.3*) är meddelanden som innehåller processdata, t.ex. data från givare.

- Service Data Object (SDO) (*Bilaga A.5*) används för konfiguration och servicedata, t.ex. motorparametrar i en motorstyrning kan behandlas via SDO.[7]

CANopen-protokollet använder sig huvudsakligen av tre olika kommunikationsmodeller, Master/slave, client/server och Producer/consumer. Oftast när man pratar om CANopen så använder man begreppen Master/slave.[4]

Master/Slave: På ett CANopen-nätverk kommer det alltid att finnas en master som har en specifik funktionalitet. Det kan aldrig finnas två noder som fungerar som master. Alla utomstående noder kommer att fungera som slavar. Mastern används för att skicka förfrågningar till slavarna på nätverket, som i sin tur svarar på masterns förfrågningar. En slav kan i normalt fall inte kommunicera med sig själv eller andra slavar.[6]

Klient/Server: Kommunikationsmodellen klient/server förser CANopen-nätverket med service av olika slag. En typisk service är att den tilldelar nätverket åtkomstpunkter (COB-ID). En klient är en nod som tar nytta av dessa tjänster. Oberoende om en nod är master eller slav, så kan den vara en klient eller server, master/slav och klient/server kommunikationsmetoderna är helt oberoende av varandra.[4]

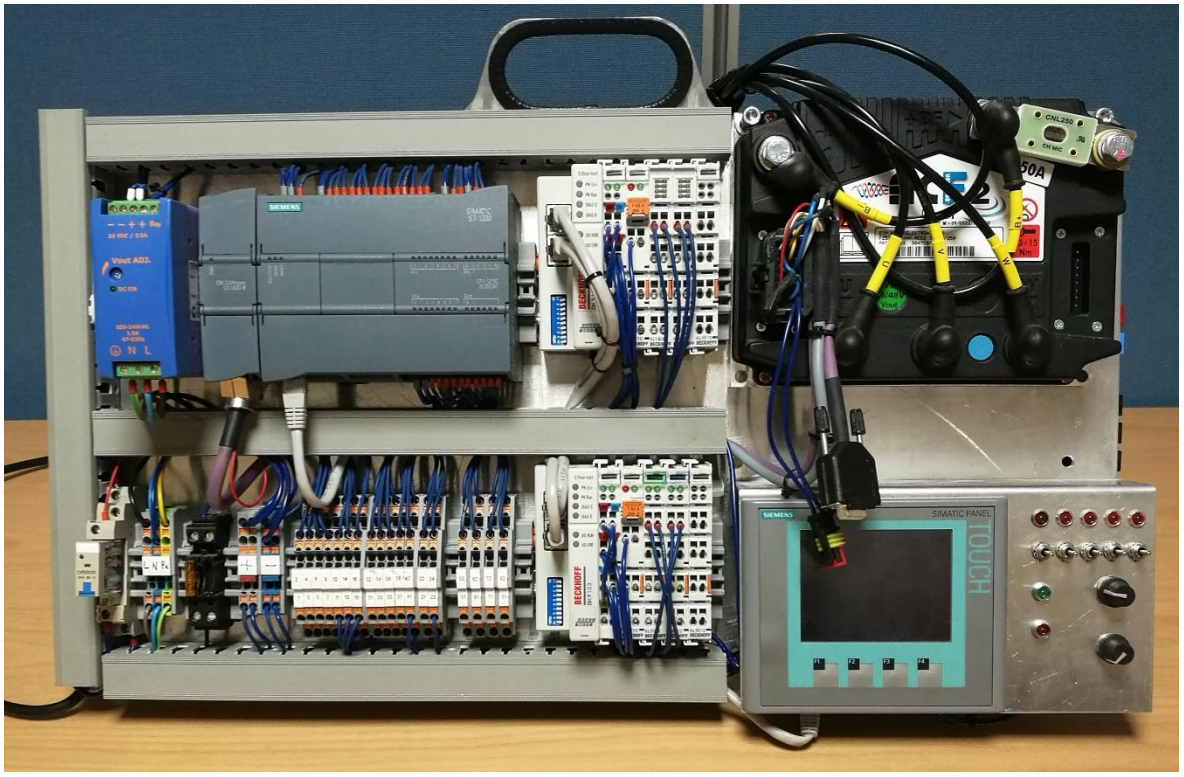
Producer/Consumer: Alla synk meddelanden som finns på CANopen bussen använder sig av Producer/Consumer-modellen. En producer överför data till nätverket medan en consumer tar emot och läser data från nätverket. På ett nätverk finns det endast en producer, medan det kan finnas flera consumers. Producer/consumer kommunikations modellen kan tex. användas för att säkerställa att alla noder i nätverket är synkade med varandra.[4]

7 Principer för applikationen

I detta lärdomsprov har det tagits fram en användbar lösning för att kunna styra en motorkontroller med hjälp av ett CANopen interface som används som tilläggsmodul för en Siemens S7-1200 logik. Det som gör arbetet komplicerat är att dom två huvudkomponenter som används i projektet använder två olika kommunikations protokoll. Siemens logiken använder sig av profinet medan motorkontrollern använder sig av CANopen. För att enheterna skall kunna prata med varandra behövs en modul som översätter dem två olika kommunikationsprotokollen. Detta görs med en CM CANopen modul och en hel del strukturerad programmeringskod. Nedan kommer det att beskrivas vilka komponenter som används i projektet och hur man har gått till väga för att välja rätt komponenter.

7.1 Simuleringsapparatur

I början av projektet gjordes en simuleringsapparatur, som används genom hela projektet för att simulera de funktioner som arbetet kräver. Apparaturen gjordes även portabel så att byte av arbetes punkter kan ske på ett smidigt sätt. Det som även togs i beaktan vid planeringen och tillverkningen av simuleringsapparaturen var att den även skall kunna användas för framtida simuleringar av olika utrustningar.



Figur 11 Bild av den framtagna simuleringsapparaturen.

7.2 Maskinvara

När hårdvara väljs till ett projekt bör man vara noggrann med att välja komponenter som klarar av det funktioner som är specificerat för projektet. Några faktorer som bör tas i beaktan är om utrustningen har några speciella säkerhetskrav eller om utrustningen behöver vara IP-klassificerad. Eftersom detta är en simuleringsapparatur som kommer användas inomhus i kontorsmiljö så klarar man sig med IP20 klassificerad utrustning, utan några extra säkerhets krav. Största kravet som var specificerat för projektet var att det behövdes en PLC som klarar av databearbetning inom rimliga programcykeltider och som är från Siemens S7-1200 logikserien. Detta pga. av att CANopen kommunikationsmodulen endast finns tillgänglig till denna serie i dagens läge. Samt behövdes en motorstyrning med CANopen kommunikationsgränssnitt.

7.2.1 PLC-enhet

I projektets planeringsskede när typen av PLC-enhet valdes, viste man inte vilken prestanda den färdiga programkoden kommer att kräva av PLC:n. Riktlinjen för val av PLC som användes var att en programcykel inte får överstiga 20ms, detta pga. av att motorstyrningen kräver en synk bit som växlar mellan noll och ett, var tjugonde millisekund. På grund av otillräcklig information om vad projektet kommer att kräva för prestanda av PLC:n så valde man den kraftigaste PLC:n i Siemens logik serie S7-1200.

Figur 12 PLC S7-1200 CPU 1215C DC/DC/DC.

PLC:n som valdes blev en Siemens S7-1200 med CPU 1215C DC/DC/DC. Denna logik använder sig av två stycken profinet anslutningar och har möjligheten att sammanfogas med CM CANopen kommunikationsmodulen. Profinet anslutningarna kommer att användas för att koppla in programmeringsmiljön via en PC, samt HMI-panelen och externa I/O kort kommer att kopplas in. Se bilaga F.1 för tekniks specifikation av PLC enheten.

PLC:n har ett antal inbyggda digitala och analoga in och utgångar. I detta projekt kommer endast en analog ingång att användas för att kunna ge kunna ge börvärde till PLC för

motorhastighet. Resten av manövreringen kommer att ske via HMI-interfacet, samt kommunikation mellan motorkontroller och PLC sker via CANopen gränssnittet.

7.2.2 CM CANopen kommunikationsinterface

CM CANopen kommunikationsmodulen är en modul som har tagits fram av tyska företaget IXXAT och gör det möjligt för siemens logikserie S7-1200 att prata CANopen. I detta arbete kommer denna modul att kommunicera med motorkontrollern. I CANopen-nätverket kommer denna modul att användas som master och motorkontrollern kommer användas som en fristående slav.

Modulen har tre stycken statusindikatorer som berättar vad CM CANopen modulen har för status. Med hjälp av statusindikatorerna indikeras vilket NMT state modulen befinner sig i, eller om fältbussen rapporterar några fel meddelanden. Utvändigt på enheten finns två stycken portar, en USB-MINI-A port som används för konfigurering av modulen och en RS232 port som man kopplar in sig på CAN-nätverket med.

Figur 13 CM CANopen modul för Siemens S7-1200 serien.

Denna modul måste konfigureras skilt med ett program som heter CM CANopen Configuration Studio. Detta är ett gratisprogram som kan laddas ner från IXXATs hemsida.

Technical specifications

Power supply	+5 VDC +10 / -10%, supplied from S7-1200 controller
Current consumption	max. 200 mA at 5 V
Dimensions	Width: 30 mm, 1.18" Height: 100 mm, 3.94" Depth: 72 mm, 2.91"
Weight	120 g, 0.26 lb
Temperature	Operating: 0 °C to +60 °C, +32 °F to +140 °F Non-operating: -40 °C to +70 °C, -40 °F to +158 °C
Relative humidity	Operating: 15 to 95% non condensing Non-operating: 5 to 95% non condensing
Housing	Plastic housing, DIN rail (35•7,5/15), protection class IP20
Certification	CE, UL, cUL

Figur 14 Teknisk specifikation för CM CANopen modulen.

7.2.3 Motorstyrning

Motorstyrningen till detta projekt blev vald på basis av vad företaget Ab Solving Oy använder sig av. Motorstyrningen ZAPI ACE2 är en motorstyrning som har blivit använt för att styra AC-motorer t.ex. för pump motorer för hydraulik och drivverk. ZAPI ACE2 styrningen har en CANopen gränssnitt som gör det möjligt att kontrollera den via CANopen.

I denna tillämpning används 48VDC som matningsspänning, vilket ger en 350 A (RMS) max ström för drivdonet och 170A (RMS) kontinuerlig ström under en timme. Mobila utrustningar inom industrin använder sig ofta av 48VDC huvudspänning för drifter, vilket leder till mindre strömmar som i sin tur leder till mindre diameter på kablar. Detta kan man direkt referera till ohms lag (Ström = Spänning/Resistans).

ACE2 motorstyrningen kan användas för drift (traction) och för pump styrning (pump control) och kan användas med AC-motorer upp till 9kW. Motorkontrollern stöder två olika kommunikationsstandarder, CANopen och vanlig I/O-kommunikation. På basis av vilket kommunikationssätt som skall användas behövs en mjukvara för motorstyrningen som är specifikt för den kommunikationsstandard som kommer användas. För att byta mjukvara i en ZAPI-motorstyrning används två olika flash program. I/O mjukvara kräver programmet ZpWinFlasher, vilket är en programvara som kommunicerar mellan PC:n och motorkontrollern med en speciell serie kabel. Vid användning av en CAN mjukvara används ingen serie kabel, utan PC:n kopplas in på fältbussen med ett CANinterface och för att byta mjukvara användas programvaran ZpCanFlasher.

Inverter for AC asynchronous 3-phase motors
Regenerative braking functions
Can-bus interface
Flash memory (128 Kbytes On-Chip Program Memory)
Digital control based upon a microcontroller

Voltage:	24 - 36 - 48 V
Maximum current ACE2 24V/400:	400 A (RMS) for 3'
Maximum current ACE2 36-48V/350:	350 A (RMS) for 3'
Maximum current ACE2 24V/500:	500 A (RMS) for 3'
Maximum current ACE2 36-48V/450:	450 A (RMS) for 3'
1 hour current rating ACE 24V/400:	200 A (RMS)
1 hour current rating ACE 36-48V/350:	170 A (RMS)
1 hour current rating ACE 24V/500:	250 A (RMS)
1 hour current rating ACE 36-48V/450:	225 A (RMS)
Operating frequency:	8 kHz
External temperature range:	-30 °C ÷ 40 °C
Maximum inverter temperature (at full power):	85 °C

Figur 15 Teknisk specifikation för ZAPI ACE2 motorstyrningen.

7.2.4 Motor

Kraven på motorn till detta arbete var att det skulle vara en tre fas 48VAC motor under 9 kW, med en inbyggd enkoder med pulsantal som stöds av motorstyrningens mjukvara.

Efter som att det inte fans specificerat i motorkontrollerns tekniska specifikationer om vad motorkontrollern stöder för enkoder pulser tog man kontakt med en ETP kraftelektronik Ab i Sverige, vilka är en underleverantör åt företaget ZAPI. ETP förklarade att puls antalet på enkodern väljs via mjukvaran som motorkontrollern använder sig av. Under samma diskussion med ETP kraftelektronik meddelades vilken enkoder som kommer att användas i projektet. ETP tog fram en CAN mjukvara som var kompatibel med den specificerade motorn till projektet.

Motorn som blev valde har en effekt på 1kW, max ström på 31A, max 2850 varv/min och en enkoder med 48 pulser per varv.

7.2.5 Extern I/O

I detta projekt valdes att använda två stycken Beckhoff moduler för att få ett större nätverk. Själva Beckhoff modulen består av en Profinet busskopplare och ett antal digitala och analoga in/ut signals kort samt ett par strömmatningskort. Hela modulpaketet har en intern databuss som Beckhoff kallar för K-buss. K-bussen löper fritt genom alla modulkort som är anslutna till busskopplaren, som i denna tillämpning är av typen BK9103. Förutom denna interna databuss så delas 24V matning ut till alla kort som är anslutna till bussmodulen, om inte matningarna har delats upp med skilda strömmatningskort. I slutet av varje modulpaket måste en slutterminal appliceras. För att kommunicera med en Siemens logik och en tredjepartsmoduler behöver man en GSD-fil. GSD-filen fungerar som en drivrutin för PLC:n. Konfigurering av GSD-fil tas upp under rubrik 8.1 Programmering.

BK9103 är en Profinet kommunikationsmodul. Enheten fungerar som busskopplare och kan sammanfogas med 1-64 stycken I/O kort, samt 255 stycken om man använder en K-buss förlängning. Kontakterna för anslutning av nätverks kabel är vanliga RJ45 kontakter. Busskopplaren stöder 10 Mbit/s och 100 Mbit/s Ethernet/Profinet anslutning. Den maximala längden mellan två busskopplare kan vara upp till 100 m. Den maximala längden på en linje är 2km om man använder sig av 20st busskopplare. Pga. av att BK9103 kommunicerar via Profinet så använder den sig av IP-adresser. IP-adressen kan ställas in med hjälp av dip-switchar som finns på modulen. IP-adressen går också att ställas in programmässigt.

Figur 16 BK9103 Busskopplare.

KL1408 är ett exempel på ett modul kort som används i detta projekt. Detta kort har åtta stycken fysiska digitala ingångar som är elektriskt isolerat från verkligheten. Numret i slutet av typ beteckningen på kortet, indikerar hur många ingångar kortet har. Kortet har åtta stycken status indikatorer, som visar status för alla åtta ingångar. 0V från kortets matningskontakter fungerar som referensspänning för ingångarna. För att få en låg signal på en ingång med detta kort behövs en spännings nivå mellan -3...+5 och för hög nivå på en ingång spänningsförs ingången med 11-30V.

Figur 17 KL1408 Ingångs kort.

Modulpaket som används i detta projekt:

BK9103 – Busskopplare 1

- KL9210 – Matningsmodul med två stycken ingångar
- KL1408 – Digital ingångs modul med åtta stycken ingångar
- KL2408 – Digital utgångs modul med åtta stycken utgångar
- KL9010 – End terminal

BK9103 – Busskopplare 2

- KL9210 – Matningsmodul med två stycken ingångar
- KL3062 – Analog ingångs modul med två stycken ingångar

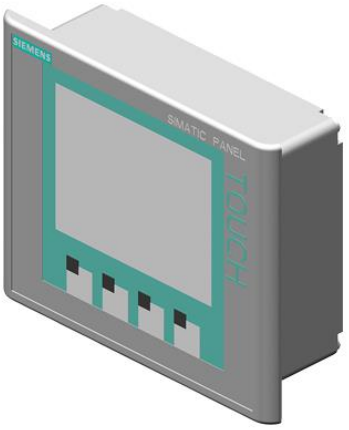
- KL4002 – Analog utgångs modul med två stycken utgångar
- KL9010 – End terminal

Se bilaga F.2 för teknisk specifikation för modul paketen.

4.1.6 HMI-interface

All manövrering som man kommer att behöva göras med simuleringsapparatur kommer att ske via användargränssnittet i en HMI panel. Val av HMI panel skedde på basis av att detta är en test utrustning och en portabel sådan, så behovet av en HMI panel med stor skärm eller bra upplösning hade man inte.

HMI-panelen som blev vald är en Siemens KTP400 basic mono panel med profinet interface. Detta är en panel med en fyra tums touch display och fyra stycken integrerade knappar på panelens framsida, vilka i detta projekt kommer att förbli okonfigurerade. På panelens baksida finns en profinet port och en 24VDC matnings kontakt. Eftersom att panelen endast har en profinet port kan panelen inte användas för att länka vidare profinet kedjan om man har flera noder. Det finns två metoder att undvika detta. Första metoden är att ansluta panelen som sista nod i profinet kedjan, från sista noden på ett profinet nätverk behöver man inte kunna länka vidare. Andra metoden är att ansluta den direkt till PLC:n eller annan enhet som har en port som är ledig. Oftast så använder sig logiker av 2 till 3 profinet portar. Optimalt är att ha en PLC med tre profinet portar, då kan man använda den sista lediga porten för konfigurering. Siemens operatörspaneler konfigureras med programmet SIMATIC WinCC, som är ett inbyggt program i TIA-Portalen.



Figur 18 KTP400 Operatörspanel.

Hardware		
Display	4.3 inch TFT widescreen display, 16 Mio colors	
Resolution	480 x 272 px	
Control elements	Touch screen, 4 function keys with LED	Membrane Keypad (28 keys) 8 function keys with LEDst
User memory	4 MB	
Interfaces	1 x RJ 45 Ethernet for PROFINET 1 x RS 485/422 for PROFIBUS/MPI 1 x USB-host, 1 x USB- device 2 x SD card slot	
Degree of protection	IP 65, NEMA 4x (front if mounted) / IP 20 rear	
Installation cutout	122 x 98 mm (W x H)	135 x 171 mm (W x H)
Front panel	140 x 116 mm (W x H)	152 x 188 mm (W x H)
Device depth	48 mm	48mm

Figur 19 Teknisks specifikation för KTP400 panelen.

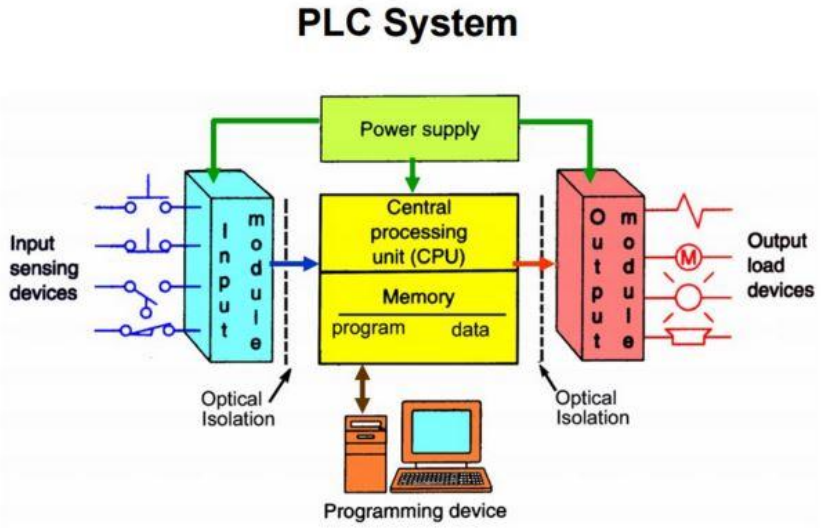
8 Implementeringen av applikationen

I detta avsnitt kommer det att beskrivas vilka metoder som har använts för att genomföra det som arbetsuppgiften är specificerad för. Själva huvuduppgiften för att få fram en fungerande lösning består till största delen av programmering. På grund av detta kommer man inte gå in på hur det praktiska arbetet kring uppkopplingen av utrustningen har gått till, endast dom viktigaste delarna för att få fram en fungerande applikation. All detaljerad information kring fördjupande av vissa delar i arbetet kommer att placeras som bilagor.

8.1 Programmering

Inom industrin är det vanligaste sättet att styra en process, med hjälp av en PLC. PLC:n och en vanlig dator har väldigt många likheter. Skillnaden mellan en dator och en PLC är att dom är anpassade för olika ändamål. PLC:n är anpassad för att styra en process medan en vanlig dator är mera anpassad för att göra visualisering och olika slags beräkningar.

Hjärnan i en PLC är CPU:n. CPU:n kan vara en 16 eller 32 bit mikroprocessor som består av ett inbyggt minne, integrerade kretsar för logiska funktioner samt monitorering och kommunikation. Alla instruktioner som PLC:n utför instrueras av CPU:n. Instruktioner som PLC: utför kan bl.a. vara logiska eller aritmetiska operationer, kommunikation med andra enheter och intern diagnostik. [8]

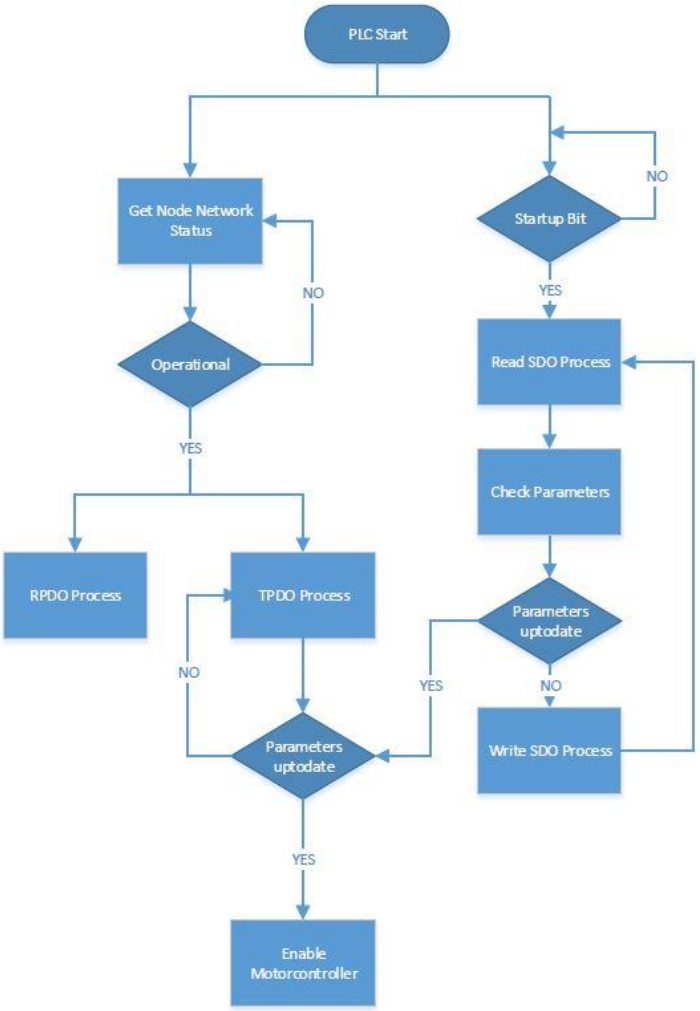


Figur 20 Uppbyggnad av en PLC.

Inom industrin använder man sig av en PLC för att kontinuerligt monitorera status av ingångar. Med hjälp av programmet i PLC:n tar man sedan beslut vad man skall göra med utgångarna. På detta sätt får man en funktion för ett styrsystem. PLC:n används också för att registrera information hos en process. För att registrera information använder man sig av olika typer av givare som är anpassade för den typ av mätning man vill registrera. [9]

PLC introducerades på sena 1960-talet av uppfinnaren Richard Morley. PLC:n hade som uppgift att ersätta relä systemen som fanns på den tiden. Relä systemen tenderade till olika typer av fel och fördröjningar inom logik systemen. Vid uppkomna fel så kunde man felsöka en hel vägg med relän för att hitta ett simpelt fel. [9]

Alltid när man skall börja göra ett PLC program bör man göra ett flödesschema som beskriver hur man har tänkt gå till väga när man gör programmet. Figur 21 beskriver programsekvensen för CANopen kommunikationsprincipen för att styra en motorkontroller. I flödesschemat beskriver man dock inte hur funktionerna fungerar internt och hur felhantering kommer att fungera, endast en helhet av övergången mellan olika funktioner illustreras här.



Figur 21 Uppbyggnad av CANopen kommunikations princip med motorkontroller flödesschema.

Programmet delas in i följande funktioner:

- Read-SDO function
  - Denna funktion använder man för att läsa in parametrar från motorstyrningen till ett datablock. Parametrarna sparas som ÄR-värden.

- Write-SDO function
  - Denna funktion använder man för att skriva parametrar från ett datablock till motorstyrningen. Parametrarna som man skriver över är sparade som BÖR-värden i datablocket.

- Check parameter function
    - Denna funktion jämför ÄR-och BÖR-värden. Denna funktion kommer alltid att köras efter att Read-SDO funktionen meddelar att den är klar. Meddelar denna funktion att ÄR-värden och BÖR-värden inte är lika, kommer Write SDO att köras.

- Get node network status
    - Denna funktion håller bl.a. koll på operationslägen för NMT:n hos slaven. Om inte noden är i operationsläge operational kommer den inte tillåta att några PDO-meddelanden sänds på bussen.

- TPDO processfunktion
    - Denna funktion läser PDO-meddelanden från slaven.

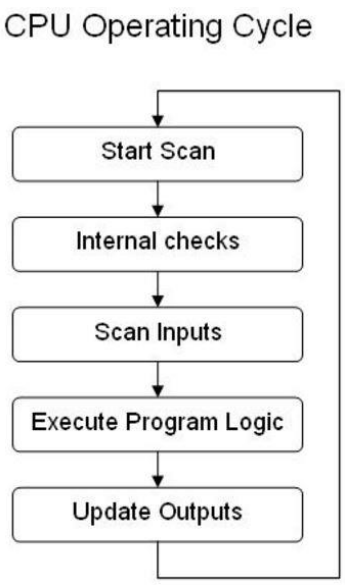
- RPDO processfunktion
    - Denna funktion skriver PDO-meddelanden till slaven.

- Alarmhanteringsfunktion
    - Denna funktion kommer att vara ett separat block gentemot dom andra fem funktionerna man har räknat upp. Detta är en funktion som kommer att hantera visualisering av felmeddelanden i operationspanelen.

Programmet man har tagit fram i detta arbete är uppbyggt av flera olika program funktioner, som alla måste gås i genom i rätt ordning för att upprätthålla en fungerande kommunikation mellan motorkontrollern och master enheten.

Som programmeringsspråk ha man använt sig av SCL vilket är ett strukturerat programspråk som man kan finna sig vissa likheter med C och C++ kod. SCL har man använt sig av genom hela projektet, förutom några enstaka funktion blocks diagram har man använt sig av för några animationer i operatörspanelen och in skalning av börvärde för motorhastighet.

Oberoende vad man använder sig av för programmeringsspråk så beter sig en programexekvering alltid lika. Programexekvering sker alltid cyklist med att utföra en instruktion åt gången. I figur22 kan man se ett exempel på hur en programexekvering ser ut i en PLC. [9]



Figur 22 Operations cykel för en PLC.

8.2 Programvara

För att kunna utföra en CANopen implementering i en Siemens S7-1200 logik och sedan kunna kontrollera en ZAPI ACE2 motorkontroller behöver man ha tillgång till ett antal konfigurations verktyg.

Programmeringsverktyget TIA-Portal V13 är det verktyg som man kommer att göra själva programmeringen i, samt användargränssnittet i operationspanelen. Konfigurationen av kommunikationsmodulen CM CANopen kommer att göras i konfigurationsverktyget CM CANopen Configurations Studio. För att kunna göra en konfiguration för CM CANopen modulen behöver man en EDS-fil för motorstyrningen. EDS-filer skall enligt standard finnas färdigt tillverkade och tillgängliga för en CANopen nod. Men i detta fall så hade tillverkaren ingen EDS-fil för denna enhet, så man var tvungen att framställa en egen. Programmet man använder för att framställa EDS-filer heter Vector CANeds.

Som alla andra tillverkare har också Zapi sina egna konfigureringsverktyg. Som man redan har nämnt i kapitel 7.2.3 Motorstyrning behöver man en CANopen mjukvara till motorstyrningen, för att kunna kommunicera via CANopen gränssnittet. Pga. av att man här har en CAN mjukvara kommer man att använda konfigureringsverktyget Zp CanFlasher för att byta mjukvara i controllern. Zp CanConsole är ett verktyg som man inte kommer att vara i behov av i framtiden när detta projekt är färdigt. Det som man gör i dagens läge med Zp CanConsole eller Zp WinConsole skall kunna göras via TIA-portalen. Men för att

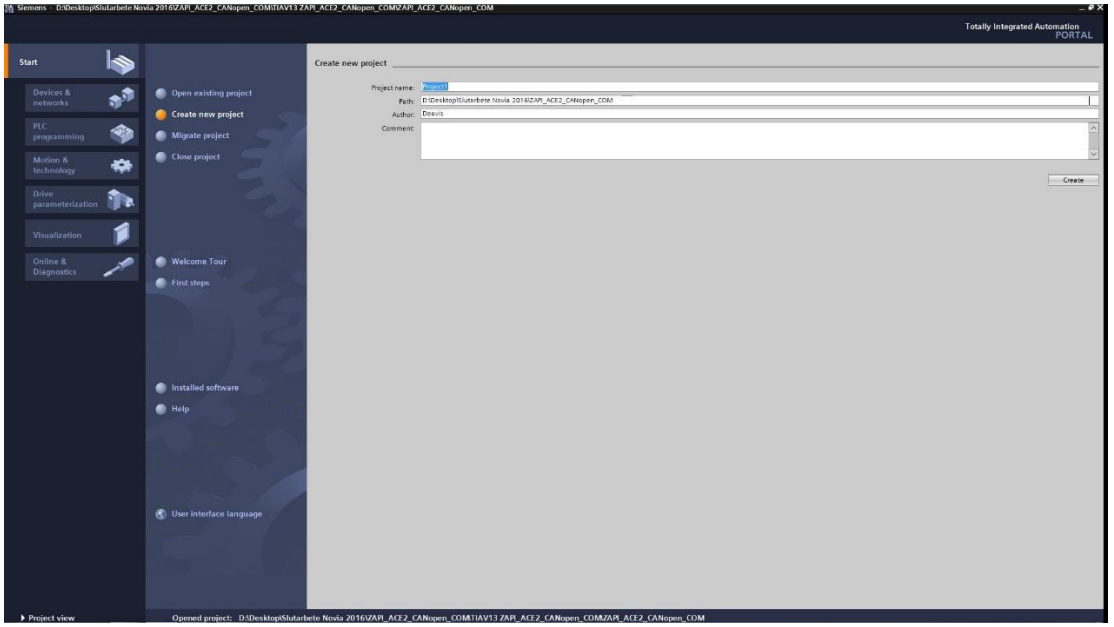
färdigställa projektet behöver man kunna läsa felmeddelanden och konfigurera motorstyrningen via Zp CanConsole.

8.2.1 TIA-Portal V13

TIA-portalen är ett programmeringsverktyg som man programmerar Siemens utrustning med. TIA-portalen eller Siemens Simatic Totally Integrated Automation Portal är uppbyggd av flera programmeringsverktyg, där av namnet TIA-portal. [10]

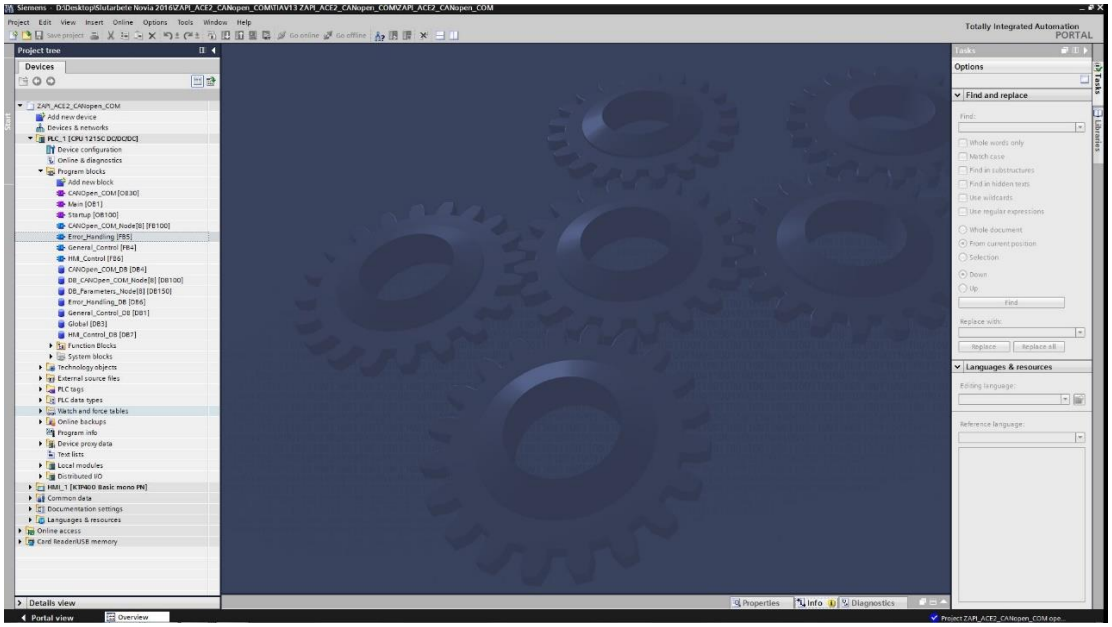
Simatic STEP7 är världens mest kända programmeringsverktyg inom industriell automation för att programmera Simatic logiker. Till Simatic familjen hör även WinCC, Startdrive och Simocode ES. WinCC används för konfigurering av allt som Siemens tillverkar inom HMI, från bas operatörspaneler till SCADA system. Programmet Startdrive och Simocode ES är konfigurations och parametriserings program för Siemens Sinamics inverter familj. Alla dessa program har man direkt åtkomst till via den nya TIA-portalen. Före TIA-portalens tid så var alla dessa program åtskilda program som man använde separat, vilket gjorde att man inte hade lika bra överblick över programmen. [10]

Tillvägagångssättet när man skall börja programmera är att först göra ett nytt projekt som innehåller alla behövliga komponenter. När man startar programmet TIA-portal V13 kan man välja att göra ett nytt projekt. I portal vyn i figur 23 sätter man in namn för vad projektet skall heta, samt vem som gör projektet.



Figur 23 Portal vy över nytt projekt.

Efter att man har gjort ett nytt projekt behöver man göra en hårdvarukonfiguration, detta tas upp i kapitel 8.3 Hårdvarukonfiguration. Från projekt vyn i figur 24 har man tillgång till all funktioner som TIA-portalen erbjuder. Till vänster av vyn kan man se projektets innehåll. Under fliken ”*Program blocks*” har man tillgång till alla programblock som projektet innehåller. Går man längre ner har man en flik som heter ”*PLC tags*”, där finns alla fysiska I/O som man har definierat, samt alla minnes adresser. Går man ännu lägre ner i listan kan man hitta operatörspanelen, i detta fall heter den ”*HMI_1”*. Under HMI_1 fliken har man tillgång till allt som gäller operatörspanelen.



Figur 24 Projekt vy.

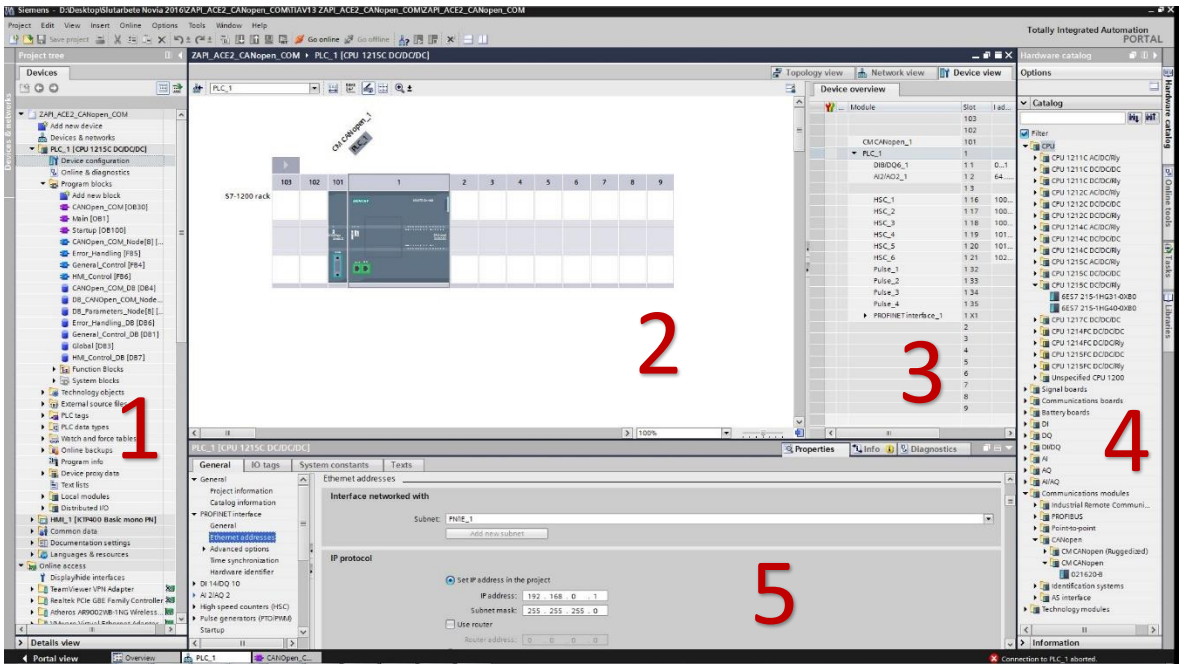
8.3 Hårdvarukonfiguration

I hårdvarukonfigurationen konfigurerar man alla hårdvarumoduler som kommer att användas med Siemens logiken. Eftersom att man huvudsakligen kommunicerar med profinet behöver man ställa in IP-adresser och enhets namn för varje modul. Och eftersom att man använder sig av en CANopen kommunikationsmodul behöver man ställa in vilken node-id modulen har. Benämning och konfigurering av fysiska in och utgångar hör också till hårdvarukonfigurationen. I detta projekt kommer man endast att använda sig av en analog ingång för att reglera motorhastighet, övriga manövreringar kommer man att göra via användargränssnittet i operatörspanelen. Men för framtida bruk kommer man att definiera alla in och utgångar som finns tillgängliga i systemet.

För att kunna använda en enhet av en annan tillverkare än Siemens i hårdvarukonfigurationen för logiken behöver man en GSD eller GSDML fil. En GSD fil innehåller bl.a. information om datastruktur och kommunikations möjligheter. En GSD fil kan till vis del jämföras med CANopen-protokollets EDS-fil. En GSD fil kan man ladda ner från tillverkarens hemsida, eller så skall den finnas tillgänglig med produkten vid leverans. [11]

I Step7 i TIA portalen kan man mycket enkelt ”drag and drop” olika moduler och enheter till konfigurationen. Detta är en av många saker som Siemens har lyckats förbättra från föregångaren Simatic manager. TIA portalens konfigurations vy kan man säga är uppbyggd av fem olika fönster. Alla fönster beskrivs med numreringar i figur 25.

Nummer ett är ett fönster som visar projektets innehåll. I detta fönster kan man bl.a. se vilka moduler man har tillgängliga samt in/utgångar och operationsstatus för varje enskild modul när man är uppkopplad mot en äkta hårdvara. Enheten som för tillfället håller på att bli konfigurerad hittar man alltid i fönster nummer två. Genom att klicka på enheten eller modulen man har i huvud fönstret nummer två får man fram fönster nummer fem och fyra. I dessa fönster konfigurerar man specifika parametrar för den aktiva enheten. Vy nummer fem används för kommunikation och egenskaps konfigurering. I detta fönster definierar man bl.a. IP-adress och enhets namn. Fönster nummer tre som man har i högra sidan av vyn, är ett fönster som man använder för att adressera in och utgångar. Adresserna man anger här kommer man sedan använda sig av när man definierar symbol namn för I/O. Nummer fyra som befinner sig längst till höger i vyn, är en lista på alla enheter och tillbehör som finns tillgängliga till systemet. Genom att importera en GSD-fil kommer enheten för GSD:n att hamna i denna lista. När man har hittat den modul man kommer att använda sig av kan man enkelt ta tag i enheten och dra över den till huvudfönstret nummer två. Är enheten kompatibel med den enhet man har i huvudfönstret kommer enheten att tillägas till den aktiva modulen, vid annat fall så kommer modulen inte att tilläggas.



Figur 25 Vy över hårdvarukonfiguration för PLC.

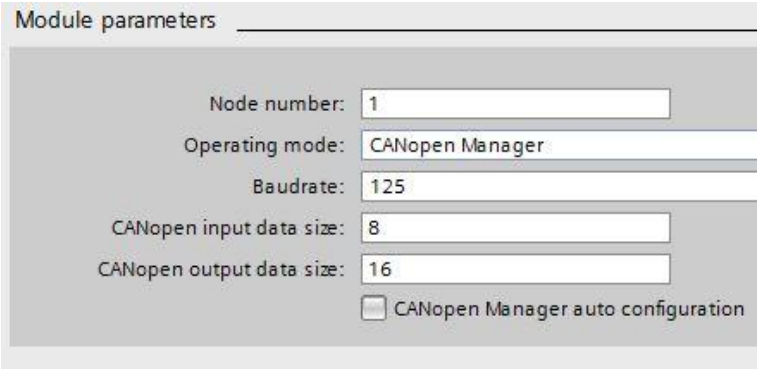
8.3.1 Konfigurering av PLC och kommunikationsmodul

Det första man gör vid konfigurering av hårdvaran är att välja vilken PLC man kommer använda. Genom att använda sig av koden som finns fysiskt definierat på varje enhet, i detta fall 6ES7-215-1HG31-0XB0 kommer man att välja rätt enhet när man söker i listan för enheter. När man har hittat en kod som stämmer överens med den fysiska hårdvaran kan man dra över den till huvudfönstret. Nästa steg i PLC konfigurationen är att välja vilken IP-adress man kommer att använda. En fördefinierad IP-adress för alla logiker kommer att vara 192.168.000.001. IP-adressen kan behöva ändras till någon lämplig adress pga. av vilket nätverk man kommer att vara anslutet till. Men i detta fall kommer man att använda sig av den fördefinierade IP-adressen, eftersom systemet är ett standalone system. IP-adresserna kommer sedan att användas i stigande ordning från nummer ett.

I PLC konfigurationen finns det färdigt definierade funktioner för olika system minnes bitar. Exempel på system bitarna är olika klockpulser och en bit som kommer att vara hög ändats första programcykel. Dessa bitar aktiverar man under fliken ”System and clock memory” i egenskaps konfigurerings fönster nummer fem. Klock bitarna definierar man att börja från minnes adress M200.0 till M200.7 och system bitarna kommer att användas från M1.0 till M1.3.

Digitala ingångarna för PLC:n kommer att ligga på noll till ett samt noll till ett för digitala utgångarna. Digitala in och utgångar definieras i hela byte. Eftersom denna logik stöder 14 ingångar kommer byte 1.6 och 1.7 att vara oanvända samt byte 1.2-7 för utgångarna. Fysiska analoga signaler definieras i word. Denna logik stöder två analog in och två analog ut signaler. Dessa två signaler kommer att bli använd på adresserna 64 till 67. Första in/utgången kommer att befinna sig på word 64-65 och den andra på word 66-67.

Efter att man ha installerat GSD-filen för CM CANopen modulen plockar man in den från modul menyn, till logiken. För att säkerställa att man använder rätt kommunikations modul behöver man säkerställa att hårdvarans artikel nummer stämmer överens med den fysiska modulen. I konfigurationen för kommunikationsmodulen behöver man ändats ange vilken node-id man vill använda för CANopen modulen samt kommunikations hastighet och hur mycket ut/in data man använder. Figur 43 beskriver konfigurations parametrar för kommunikationsmodulen.

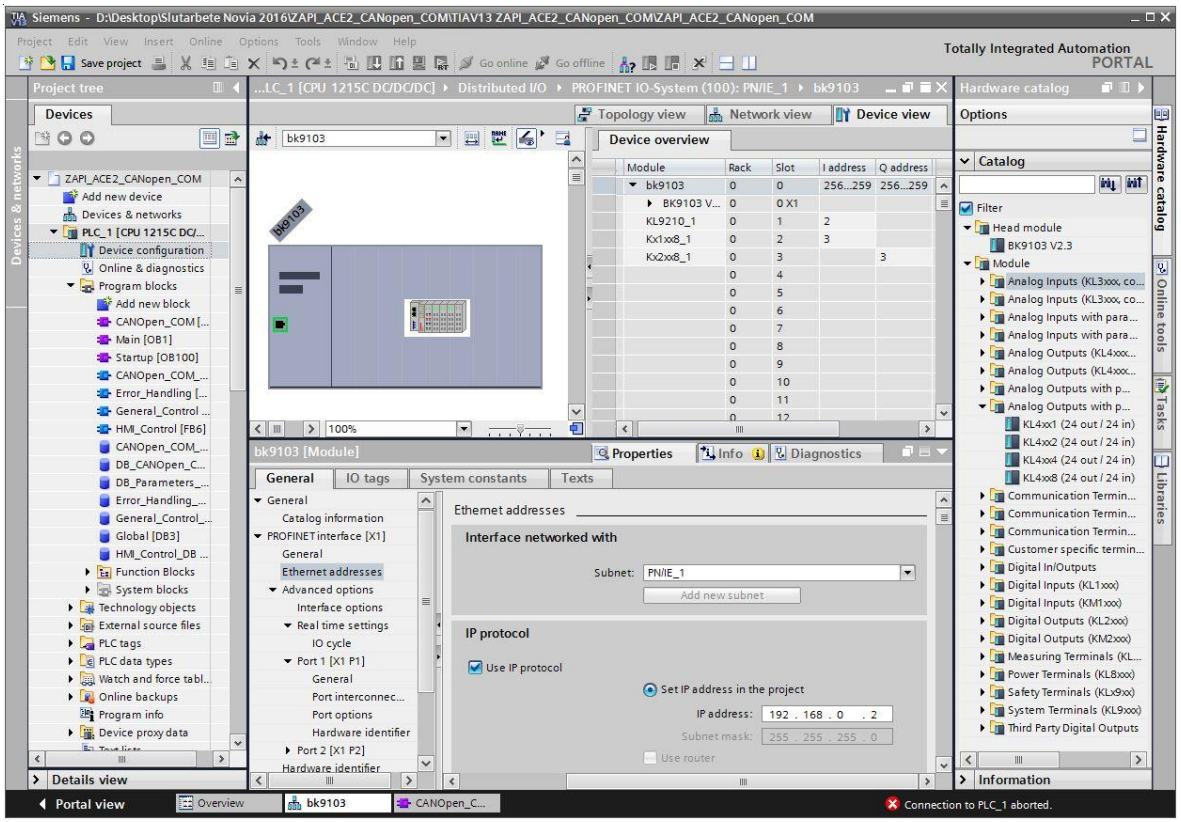


Figur 26 Konfigurations parametrar för CM CANopen.

8.3.2 Konfigurering av externa profinet I/O-moduler

I detta arbete använder man sig också av två stycken externa profinet I/O-moduler. Som man nämner ovan är I/O-modulerna av märket Beckohoff. För att kunna använda modulerna på ett vettigt sätt behövar man också tillägga dessa i hårdvarukonfigurationen för PLC:n. Eftersom Beckhoff modulerna inte tillhör Siemens familjen så behöver man en GSD-fil för dessa. GSD-filen laddas ner från Beckhoffs hemsida. Enlig samma utförande som tidigare gjordes för CM CANopen modulen importerar man GSD-filen för att sedan kunna hitta modulen, i modul listan till höger av vyn i figur 27. Som man läste i avsnitt 7.2.5 så är modulen uppbyggd av modulkort som används som in och utgångar. Första man gör vid en konfigurering av en modul som består av flera delar är att importera bas enheten till

konfigurationen. I detta fall så kommer busskopplaren BK9103 fungera som basplatta. Efter att man har tillagt en basenhet bör man placera in alla tilläggs kort. Hårdvarukonfigurationen kräver att man placerar korten i samma ordning gentemot den fysiska apparaturen. Modulkorten kommer att placeras in i adresserings fönster som också kallas anordnings översikt. Anordnings översikten hittar man till höger om I/O-modulen i figur 27, samt samma fönster som beskrivs med nummer tre i figur 25.



Figur 27 Vy över Beckhoff I/O-modul konfiguration.

Som man nämner i avsnitt 7.2.5 kommer man att splittra på analoga och digitala signaler på två moduler. Modulen som beskrivs i figur 27 kommer att fungera som en digital modul. Första modulkortet på denna busskopplare fungerar som ett strömmatningskort. Förutom förmågan att kunna förse andra kort med spänningsmatning har detta kort två status ingångar. I detta projekt kommer man inte att använda sig av dessa ingångar, men tas ändå i beaktan som ingång två i konfigurationen. Följande kort för busskopplare 1 är ett digital ingångs kort med åtta digitala ingångar, därpå har man ett åtta kanaligt utgångs kort. Dessa två kort kommer att adresseras med ingång nummer tre och utgång nummer tre. Samma som för logik konfigurationen använder man sig av hela byte för att adressera digitala ingångar. Pga. av att logikens fysiska in och utgångar slutar på nummer ett fortsätter man här med nummer två.

Busskopplare nummer två kommer att konfigureras likadant som modul ett förutom att man använder sig av analoga kort istället för digitala. Busskopplare nummer två kommer också att innehålla ett strömmatningskort som första modul, för att förse analogkorten med spänning. Digitala ingångarna för detta modulkort kommer att få sitt nummer efter ordningsföljden för ingångssystemet, vilket är nummer fyra. Analoga in och utgångarna definieras enligt samma nummersystem till 68-71, vilket betyder att korten har två kanaler, eftersom analoga signaler använder sig av word som data typ.

Som IP-adress för modul 1 kommer man att använda sig av IP-adress 192.168.000.002 och för modul 2 kommer man att använda IP-adress 192.168.000.003. Pga. av att man använder 192.168.000.xxx som bas för IP-adressering för logiken, behöver man använda samma bas för alla moduler inom nätverket. Före man definierar adressen för den fysiska modulen bör man ange IP-adressen i hårdvarukonfigurationen för den specifika hårdvaran. För att göra detta går man tillväga på samma sätt som för IP definitionen för PLC:n, se figur 25 fönster nummer fem.

Den fysiska IP-adress definitionen av dessa moduler kan göras med hjälp av två olika metoder. Första metoden är att använda sig av dip-switchar som finns integrerade på bussmodulerna. Dip-switchen består av tio enskilda dipp där varje dipp har en specifik betydelse för kommunikationen. Med hjälp av dipp ett till åtta använder man för att ange IP-adressens sista ordningsnummer 192.168.000.xxx. Ordningstalet man kommer att använda anger man i binärt format. Dipp nummer ett kommer att motsvara decimala talet ett och dipp nummer åtta motsvarar decimalt 128. Med hjälp av en kombination av dip-switchens två sista dippar nio och tio definierar man hur IP-adressen skall användas, se figur 28. IP-adressen för modulerna definierades enligt figur 29 och 30.

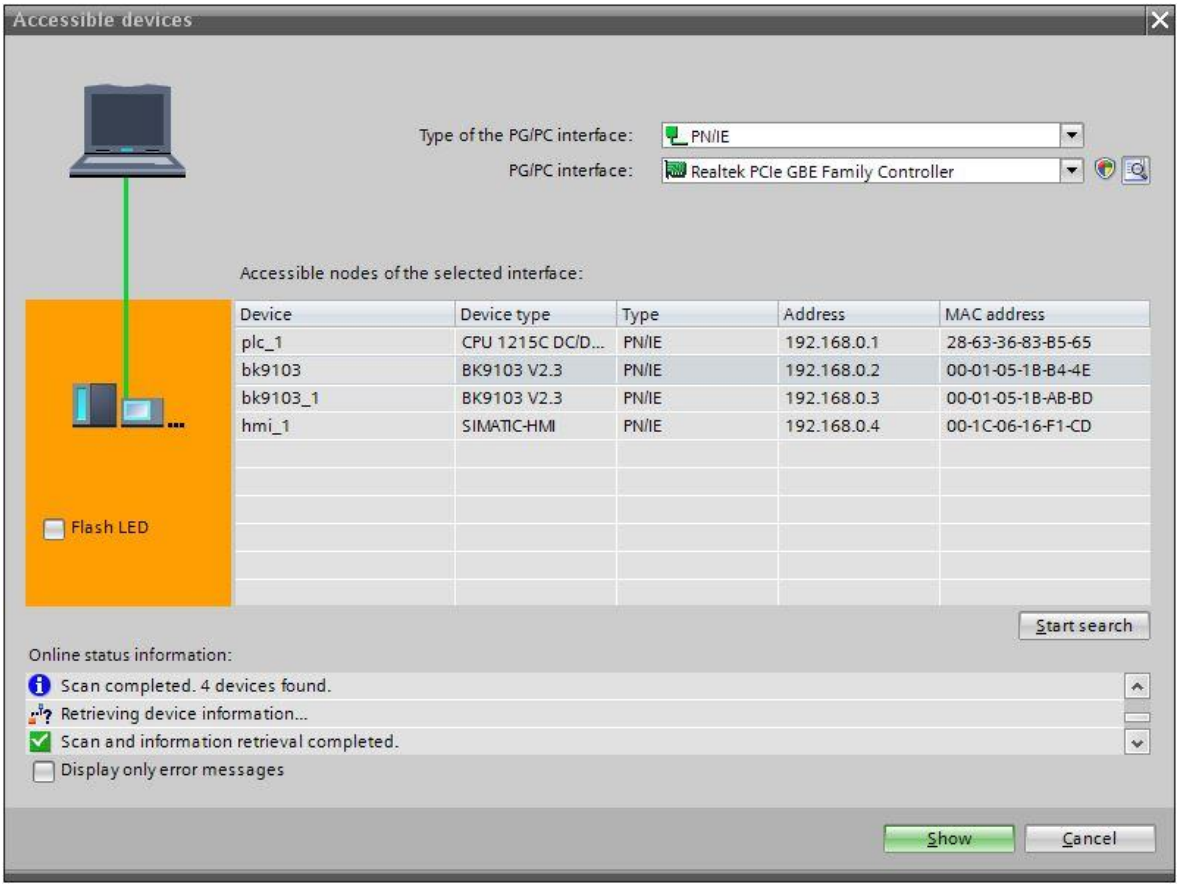
DIP 9	DIP 10	Description	Restart Behavior	Behavior with factory settings
0	0	Last byte of the IP address via DIP switches 1-8	- PN name from memory - IP address via DIP switches 1 - 8	- PN name becomes empty string - IP address via DIP switches 172.16.17.xxx (xxx DIP switch) SNM 255.255.0.0
0	1	DHCP DIP switches 1-8 set to "0"	- PN name from memory - IP address and SNM via DHCP	- PN name becomes empty string - IP address and SNM via DHCP - DNS name "bk9103-xxyyzz" xxyyzz last 3 bytes of MAC ID
		DHCP DIP switches 1-8 not set to "0"	- PN name via DIP switches 1 - 8* - IP address and SNM via DHCP	- PN name via DIP switches 1 - 8 - IP address and SNM via DHCP - DNS name "bk9103-xxyyzz" xxyyzz last 3 bytes of MAC ID
1	0	BootP	- PN name from memory - IP address and SNM via BootP	- PN name becomes empty string - IP address via BootP
1	1	PROFINET-compliant	- PN name from memory - IP address from memory	- PN name becomes empty string - IP address 0.0.0.0
		PROFINET with fixed name	- PN name via DIP switches 1 - 8* - IP address from memory	- PN name via DIP switches 1 - 8 - IP address 0.0.0.0

Figur 28 Kombination av DIP9 och DIP10 för IP-adressering av I/O-modul.

Figur 29 IP-Adressering av modul 1.

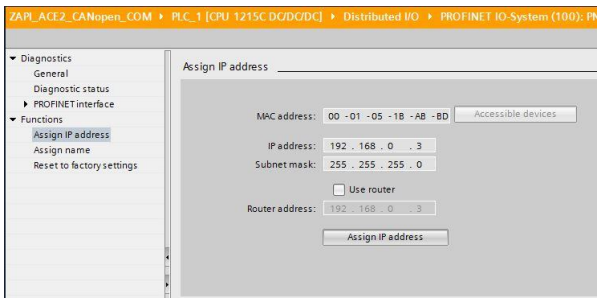
Figur 30 IP-Adressering av modul 2.

Andra metoden för att definera IP-adresser för moduler är att använda sig av TIA-portalens inbyggda funktioner, blinka status led och ange IP-adress. När man använder sig av denna metod anger man IP-adress över nätverket. Detta kräver att alla dip-switchar är ställt i OFF läge och att man har laddat ner konfigurationen till PLC:n och arbetar i online läge. För att kunna ange IP-adress över nätverket behöver man veta var modulerna är placerade fysiskt. Detta är endast problem om man använder sig av multipla enheter som är av samma modell och fabrikat. Genom att använda sig av ”Accessible devices” som är en inbyggd funktion i TIA-portalen som visar alla tillgängliga enheter på nätverket, kan man se MAC-adressen för varje enhet, se figur 31. Denna metod används ofta när man har en enhet som man inte kan välja IP-adress på fysiskt. I detta fall är inte denna metod nödvändig.



Figur 31 Accessible devices program vy i TIA-portalen.

Genom att markera en modul som finns tillgänglig på nätverket och använda sig av funktionen ”Flash LED” som finns till vänster av vyn i figur 31, kommer en status lampa att börja blinka på den modul som har samma MAC-adress som den modul man har valt att diagnostisera. Genom att hålla reda på MAC-adresserna kan man sedan leta upp modulen i projekt vyn (se figur 33) och använda sig av tangent kombinationen CTRL + D. Meny som kommer fram är ett diagnosticeringscenter där man kan använda sig av ett fåtal funktioner. Under denna meny kan man använda sig av funktionen ”Assign IP-adress” se figur 32.



Figur 32 Diagnosticerings center för en specifik enhet.

När man använder sig av denna funktion kommer nätverket att använda sig av MAC-adressen för att ange den valda IP-adressen, till en specifik enhet på nätverket.

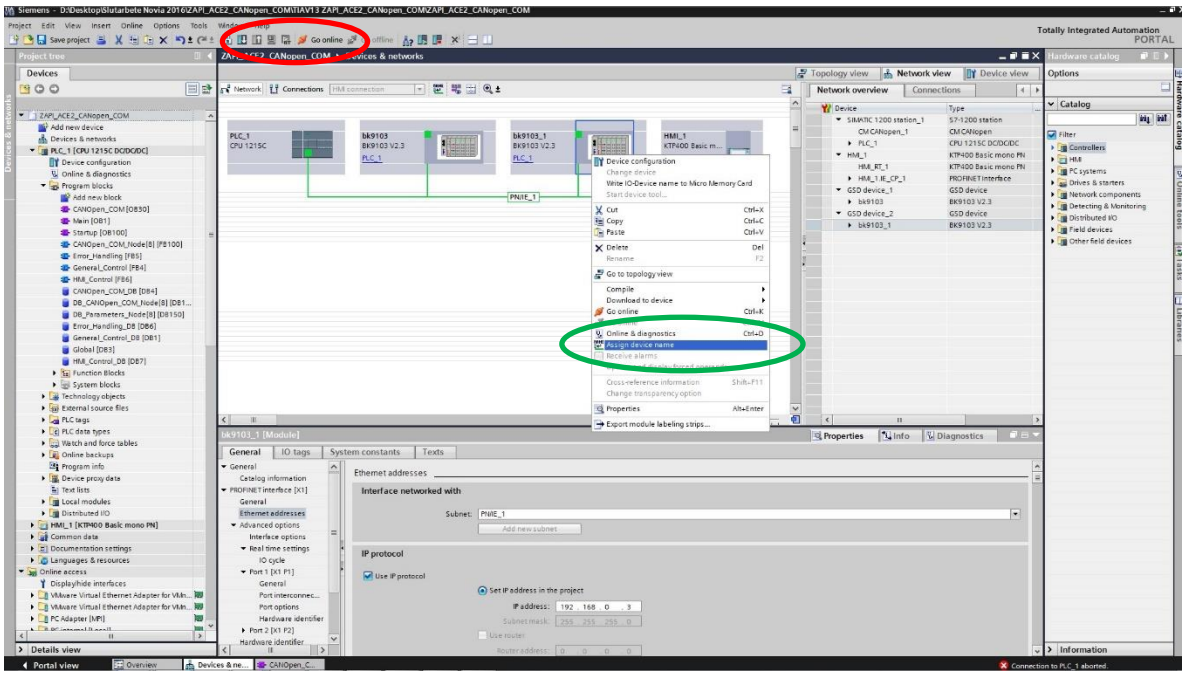
8.3.3 Konfigurering av HMI-interface

Eftersom HMI-interfacet man använder sig av i detta projekt är en produkt ur Siemens familjen behöver man ingen GSD-fil. Denna enhet kan man importera direkt till projektet från modul listan utan några extra omständigheter. Ända man behöver göra för att konfigurera operatörspanelen till nätverket är att ange en lämplig IP-adress. IP-adressen man kommer att använda sig av är nästa lediga adress enligt nummerordningen, vilket kommer att vara 192.168.000.004. Adressen matas in enligt samma procedur som man har gjort för PLC:n och I/O-modulerna, se figur 25 fönster nummer fem. Om man i ett annat utförande skulle använda sig av två HMI-interface av samma märke och modell, skulle man vara i behov av att använda sig av ovannämnda metod två, för IP definiering.

8.3.4 Profinet namnimplementering

Eftersom profinet använder sig av TCP/IP behöver man en IP-adress. Men IP-adressen användes inte för prosess dataöverföring. IP-adressen används endast för konfiguration, parametrisering och diagnostik. Profinet använder sig av anordnings namn som är DNS baserade för realtidskommunikation. Anordnings namnen måste skrivas över till alla enheter via ett profinet verktyg, i detta fall TIA-portalen. Viktigt med anordnings namnen är att dom måste vara exakt lika skrivna i både PLC:n och i den specifika fysiska modulenheten.

Tillvägagångssättet för att skriva anordnings namnet till en specifik modulenhet görs på samma sätt för alla moduler på ett nätverk. Första steget för denna procedur är att ladda ner hårdvarukonfigurationen till PLC:n och arbeta online. Symbolen man använder för att ladda ner ett projekt till logiken hittar man i projekt vyns övre kant (se figur 33). Även i den övre kanten av vyn hittar man en orange symbol med texten go online. Efter att man har laddat ner projektet kan man använda sig av denna symbol för att komma online och kunna skriva över anordnings namnen. Genom att högerklicka med muspekaren på en enhet kan man välja ”assign device namne”. Denna funktion används för att skriva över anordnings namn för samtliga modulenheter.



Figur 33 Projekt vy över moduler i nätverket.

Efter att man är klar med denna procedur är man färdig med hårdvarukonfigurationen och kan börja med att skriva ett PLC program.

8.4 Definiering av funktionsblock

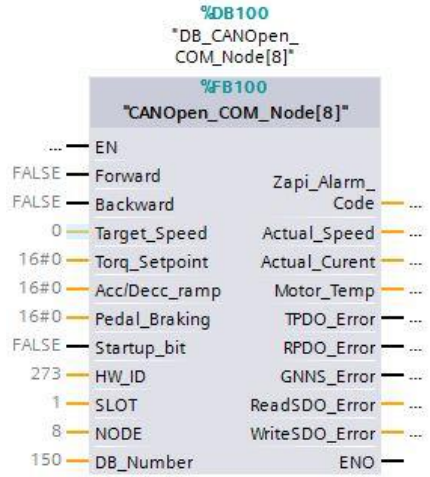
Det som man gjorde före man började skriva programkoden för att styra motorkontrollern via PLC:n, var att ta reda på hur man skall gå till väga för att få en fungerande lösning. Det som man gjorde var att köra motorn manuellt genom att sända CAN-meddelanden via ett program som heter CAN trace. CAN trace är ett diagnostiseringsprogram som man har använt sig för att analysera CAN trafik, samt simulera och sända meddelanden till CANopen-nätverket. Genom att göra en simulering av en master node med programmet CAN trace kunde man få en överblick av hur CAN-meddelanden som man sänder skall vara uppbyggda för att kunna styra motorn. Samt konfigurering av parametrar gjorde man vid denna simulering. Efter en lång diskussion med ETP kraftelektronik i Sverige fick man en överblick över funktionen hos motorkontrollern. Nedan i figur 34 har man tagit ett skärmklipp ur programmet CAN trace när man kör motorn med 50Hz. Alla service data och processdata objekt man kommer att behandla i detta avsnitt finns förklarade i bilaga G.

208	1	8	88 13 4a 00 00 00 00 00	0.000	1879	Tx
80	1	8	00 00 00 00 00 00 00 00	0.000	10054	Tx
188	1	8	00 00 02 c4 00 00 00 00	300.693	6082	Rx
208	1	8	88 13 4a 00 00 00 00 00	0.000	1880	Tx
488	1	8	0d 0b 31 00 26 4b 00 01	300.699	8535	Rx
80	1	8	00 00 00 00 00 00 00 00	0.000	10055	Tx

Figur 34 Kör motorn med hastigheten 50Hz.

När man hade simulerat CAN trafiken med hjälp av CANtrace började man skriva kod för att kunna styra motorkontrollern via logiken. Det som man tog fram var ett funktionsblock som innehåller ett antal funktioner, utseendet på blocket kan man se i figur 35. Funktionerna kommer man att beskriva enskilt mera i detalj med hjälp av rubrikerna nedan.

Tanken med detta funktionsblock är att man enkelt skall kunna importera funktionsblocket och endast av några få variabler kunna styra en motor. Och beroende på hur många noder man kommer att använda plockar man in ett funktionsblock för varje nod. Blocket har ett antal in och utgångar som man använder sig av för manövrering och diagnostisering av CANopen noden. Flödesschema och detaljerad funktionsprincip för denna funktion kan man se som bilaga E.1.



Figur 35 CANopen funktionsblock för styrning av motorkontroller.

8.4.1 Read-SDO funktion

För att läsa parametrar från en CANopen node gjorde man en funktion som stegvis läser ut alla parametrar man har definierat som objekt index i ett datablock. Till samma datablock kommer man sedan att skriva parametrarnas ÄR-värden samt läsa BÖR-värden för Write-SDO funktionen som behandlas i nästa kapitel. Read-SDO funktionen är den första funktionen som kommer att startas i programsekvensen. Flödesschema och detaljerad funktionsprincip för denna funktion kan man se som bilaga E.2.

8.4.2 Write-SDO funktion

För att kunna skriva parametrar till en CANopen node har man gjort en funktion som stegvis skriver in alla parametrar från ett datablock till CANopen-nätverket. Denna Write-SDO funktion har samma uppbyggnad som föregående funktion Read-SDO. Ända skillnad mellan funktionerna är att Write-SDO skriver parametrar till en CANopen node medan Read-SDO

läser parametrar från en node. Write-SDO funktionen har även en separat spar cykel som den kommer att köra som sista program cykel i programmet när alla parametrar har blivit överförda till CANopen enheten. Denna spar cykel gör så att alla parametrar blir sparade i enhetens EEPROM. Flödesschema och detaljerad funktionsprincip för denna funktion kan man se som bilaga E.3

8.4.3 Get node up-to-date status funktion

För att CANopen noden alltid skall hålla sig uppdaterad med korrekta BÖR-värden gjordes en funktion som alltid jämför BÖR och ÄR-värden vid en slutförd Read-SDO process. Får man ett svar från funktionen att BÖR och ÄR-värden inte stämmer överens kommer man att köra en Write-SDO process, vilket leder till att efter slutförd Write-SDO process kör man igen en Read-SDO plus en uppdaterings koll. Genom att använda sig av denna metod kommer man alltid att ha uppdaterade parametrar i CANopen enheten. Denna funktion kan man dra till stor nytta till vid byte av CANopen node, i detta fall motorstyrning. Vid byte av motorstyrningen behöver man endast montera den nya motorstyrningen hårdvarumässigt och slå på strömmen till PLC:n, sedan kommer den nya motorstyrningen att uppdateras med nya parametrar. Flödesschema och detaljerad funktionsprincip för denna funktion kan man se som bilaga E.4.

8.4.4 Get node network status

För att hålla reda på vilket operations status CANopen Mastern har gjorde man en funktion som behandlar NMT state machine status. Denna funktion väljer också om det är tillåtet att sända processdata (PDO) på CANopen nätveket. Finner funktionen att CANopen mastern har något olika operationsstatus än Operational kommer funktionen inte att tillåta processdata trafik på bussen. Flödesschema och detaljerad funktionsprincip för denna funktion kan man se som bilaga E.5.

8.4.5 RPDO processfunktion

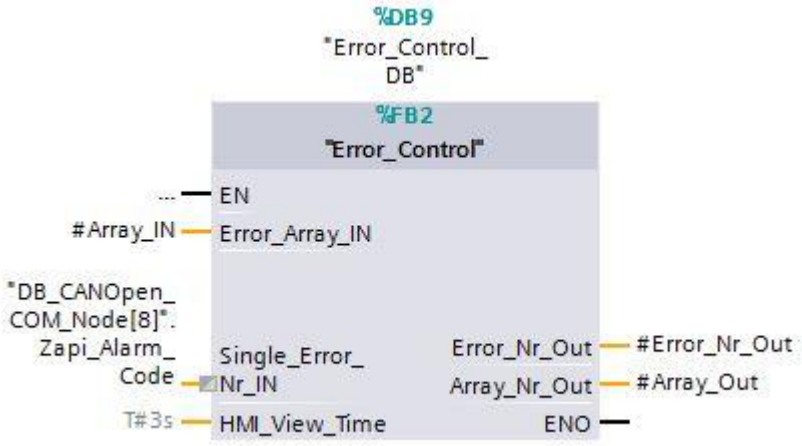
RPDO funktionen kommer att fungera som huvudfunktion för styrning av motorkontrollern. Till skillnad från SDO funktionerna som bara behandlar parametrar, behandlar denna funktion endast styrkomandon till invertern. Pga. av att denna funktion använder sig av PDO-meddelanden kommer funktionen endast att fungera när slaven, i detta fall motorkontrollern är i operationsläge operational. Funktionerna i föregående kapitel som behandlar SDO meddelande kommer även att fungera här, i.o.m att SDO är det ända datapaketet som stöds att användas i oberoende operationsläge. Flödesschema och detaljerad funktionsprincip för denna funktion kan man se som bilaga E.6.

8.4.6 TPDO processfunktion

TPDO funktionen kommer enbart att läsa realtidsinformation från motorkontrollern med hjälp av TPDO1 och TPDO2. T.ex. när man läser ÄR värde på motorhastighet används denna funktion. Dessa PDO paket fungerar som utgångar från invertern. TPDO1 fungerar som en återkoppling av RPDO1 vilket man kan se i kapitel bilaga G.3. Flödesschema och detaljerad funktionsprincip för denna funktion kan man se som bilaga E.7.

8.4.7 Alarmhanteringsfunktion

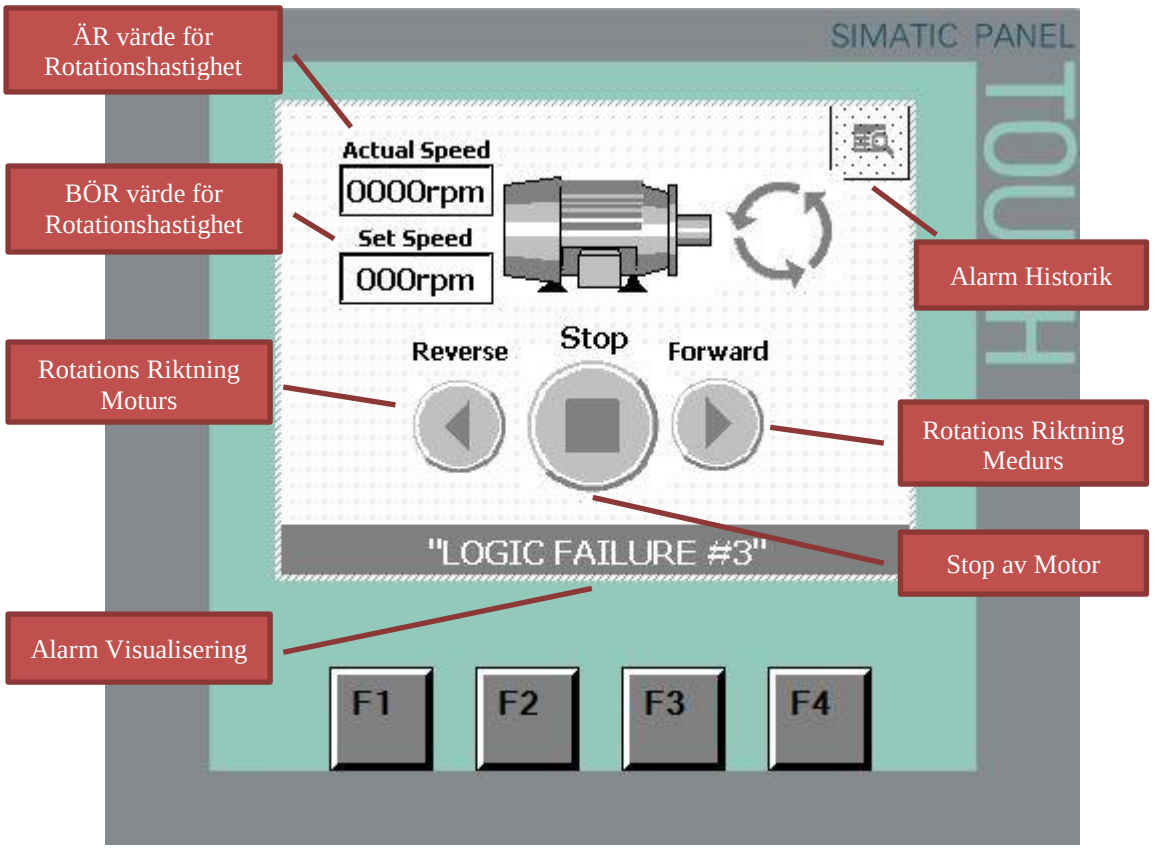
För att kunna hantera alarm meddelanden på ett vettigt sätt gjordes ett separat funktionsblock som konverterar felmeddelanden till data man kan använda i en HMI display för diagnostik. Funktionsblocket använder sig av två olika typer av inputdata, Integer och Array of Bool. Som utgångs data ur funktionen har man också två olika typer av data, integer och Array of Integer. Funktioner är gjord så att den kan behandla felmeddelande historik och felmeddelandevisualisering. Visualiseringen är gjord så att beroende på hur många felmeddelande man har aktiva på samma gång kommer funktionen att visa varje felmeddelande i skärmen x antal sekunder, beroende på vilken tid man har ställt in på ”HMI_View_Time”. Har man inga felmeddelanden aktiva kommer ingenting att visas. Flödesschema och detaljerad funktionsprincip för denna funktion kan man se som bilaga E.8.



Figur 36 Felmeddelandehanterings funktionsblock

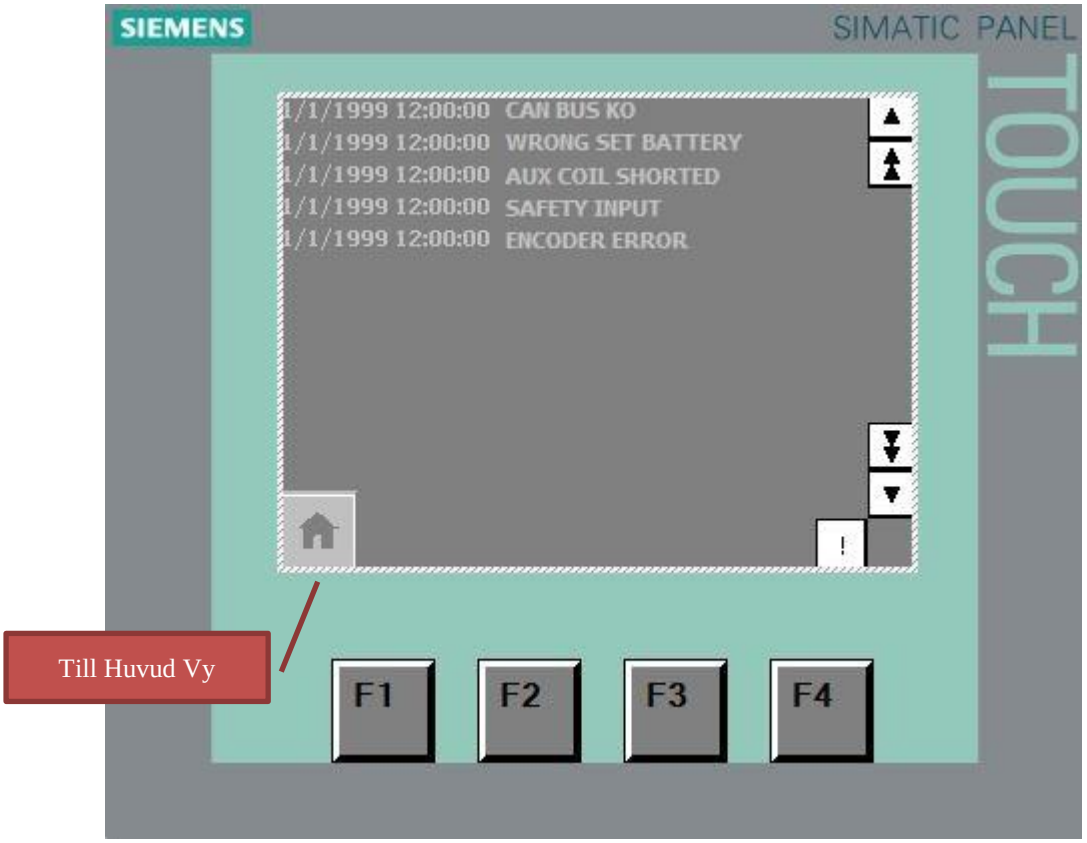
8.5 HMI-visualisering

HMI-visualiseringen gjordes med verktyget WinCC Advanced som är implementerat i TIA portalen. Med hjälp av detta program kan man konfigurera allt som har med en Siemens HMI panel att göra. I denna applikation använder man sig av endast två vyer, en huvud vy varifrån man kontrollerar motorstyrningen och en skärm som används för alarmhistorik. I figur 34 kan man se hur layouten för huvudvyn ser ut. Från denna vy kan man ange rotations riktningar, stoppa, starta motorn se BÖR och ÄR värde för rotationshastighet, samt aktiva felmeddelanden.



Figur 37 Huvud vy.

Alarmhistoriken kommer man åt genom att klicka på ikonen uppe i höger kant i figur 37. Layouten för alarmhistoriken kan man se i figur 38. Denna meny är en meny som Siemens har tillämpat för diagnostisering av olika typer av alarmhistoriker. Genom konfiguration kan man anpassa den enligt egna behov. Alarmhistoriken anger vilket alarm man har haft aktivt, samt datum och klockslag när alarmet har blivit aktiverat. Detaljerad funktionsprincip för alarmhistoriken finns beskrivit som bilaga E.10

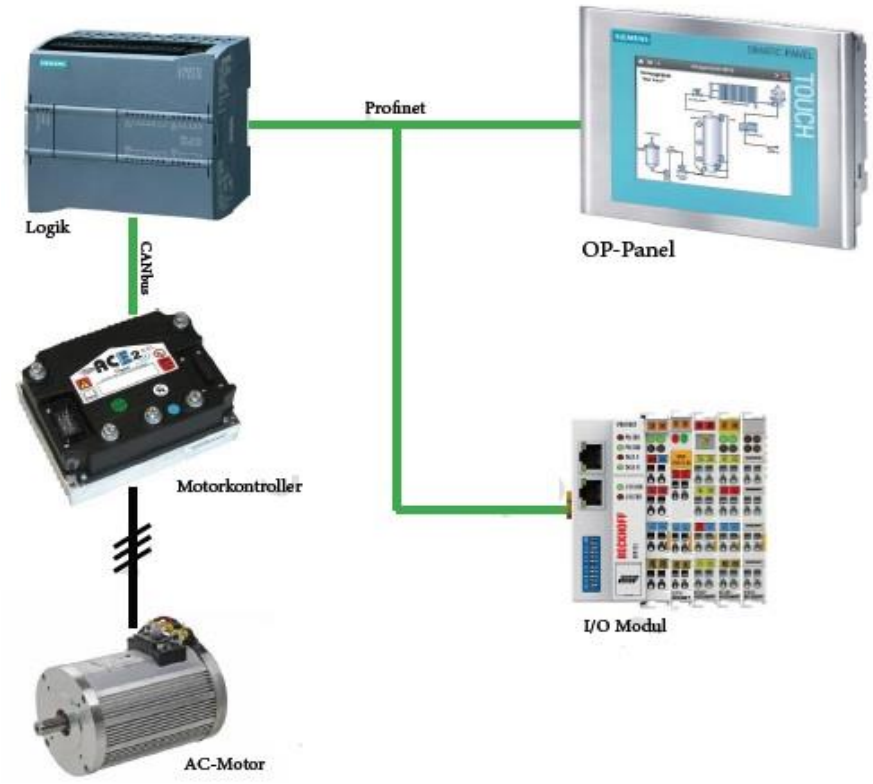


Figur 38 Alarmhistorik vy.

9 Resultat

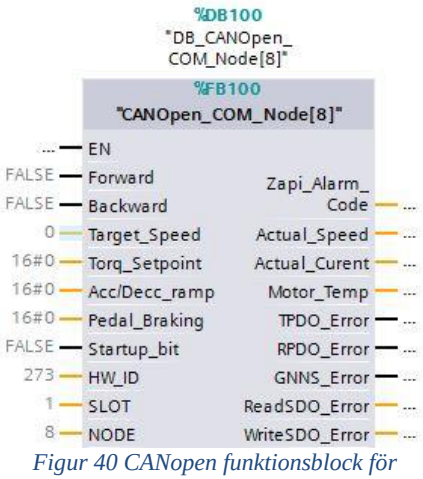
Syftet med detta examensarbete var att ta fram en lösning för att kunna styra en motorkontroller via ett CANopen gränssnitt, samt implementera externa profinet I/O-moduler och en HMI panel i profinet systemet. För att kunna simulera hela processen gjordes en simuleringsapparatur som bestod av alla behövliga delar. I projektets specifikation har man inte inkluderat den parameterhantering som gjordes i detta arbete, utan detta gjordes utav eget intresse, för att få ett så användarvänligt och modernt system som möjligt. Lika så togs ett alarmhanteringsblock fram, detta p.g.a. att man var i behov av ett sådant block till ett annat projekt, som utfördes samtidigt som detta projekt.

Genom hela arbetet har fokus legat på att göra allting så användarvänligt som möjligt för att en annan applikator lätt skall kunna ta igång ett liknande system som detta. För att åstadkomma detta koden kommenterats noggrant och det som utförts har dokumenterats. Arbetet har koncentrerats på att göra en kod som lätt kan tillämpas till vilken CANopen-nod som helst, vilket också leder till att man i framtiden sparar tid om man skulle vara i behov av att kommunicera med en CANopen-nod av annat slag.

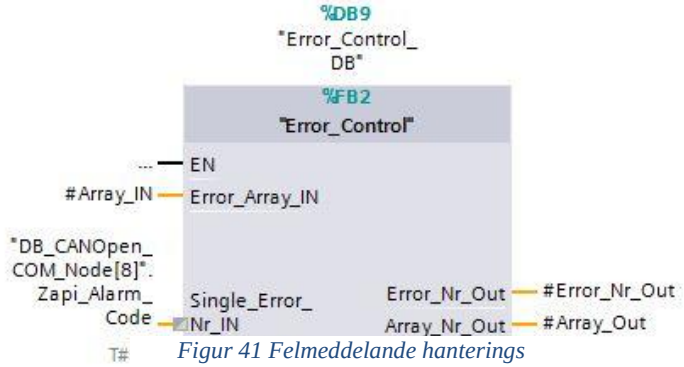


Figur 39 Helhets bild över arbetet.

Resultatet av projektet blev en välfungerade process som hanterar CANopen kommunikation mellan en Siemens S7-1200 logik och en Zapi motorkontroller. Styrning av motorn görs vi en Siemens HMI panel som man har implementerat i systemet. Rotationshastigheten av motorn styrs via en analog ingång på en extern Beckhoff I/O-modul. Siemens logiken har även inbyggda analog ingångar som man kunde ha använt för rotationshastighet, men pga. av att man ville testa en extern profinet I/O-modul gjordes det, på detta sätt. Helhets bild över arbetet kan ses i figur 39.



Figur 40 CANopen funktionsblock för styrning av motorkontroller.



Figur 41 Felmeddelande hanterings funktionsblock.

Man tog fram ett funktionsblock som behandlar CANopen kommunikation samt ett funktionsblock som behandlar alarmhantering. Funktionsblocket för CANopen kommunikationen blev ett block som hanterar både processdata och servicedata, samt felmeddelandehantering inom CANopen-nätverket. I.o.m att man implementerade en funktion för parametrisering av motorkontrollern kommer man i framtiden inte att vara i behov av att parametrisera motorenheten med hjälp av en handkontroller, vilket man gör idag. Funktionsblocket för alarmhanteringen blev ett väldigt användbart funktionsblock, som man redan före projektets slut har börjat använda i några av Solvings utrustningar. Figur 40 och 41 beskriver med bild dom två funktionsblocken man har tagit fram.

10 Diskussion

Det först som jag gjorde i samband med examenarbetets start var att installera alla behövliga program och titta igenom olika funktioner i mjukvarorna. I samband med att jag började göra ett litet test program i TIA portalen konstaterades att jag behövde uppgradera sig till en nyare version av programmet. TIA portalen uppdaterades till version V13 och efter det var det bara att köra på.

Största utmaningen i början var att konstruera en lämplig EDS-fil för motorstyrningen. När jag hade tagit fram en EDS-fil som stämde överens med inverter tillverkarens specifikationer började jag försöka få kommunikation mellan logiken och motorstyrningen. Här började det uppstå problem som gjorde att jag behövde ta kontakt med underleverantören till motorstyrningen samt tillverkaren av CANopen modulen till logiken. I ett senare skede hade jag också kontakt med TK Engineering Oy i Vasa, vilka arbetar med CANopen heltid. Jag konstaterade slutligen att motorstyrningens CANopen protokoll inte följer CiA standarden till hundra procent. Detta ledde till att jag konfigurerade om EDS-filen medhjälp av Ixxat som var tillverkaren av CANopen modulen till logiken. Slutligen fick jag upprätthållit kommunikation mellan motorkontrollern och logikens CANopen modul.

När jag hade en kommunikation som var intakt mellan mastern och slaven började jag simulera CAN-meddelanden för att styra motorkontrollern. Detta gjorde jag för att jag skall få en överblick över vilka meddelanden jag behöver sända för att att kunna styra en motorkontroller med alla behövliga kommandon. Även här stötte jag på problem pga. av att CAN-standarden inte stöddes till hundraprocent. Medhjälp av ETP kraftelektronik i Sverige löste jag problemen och kunde börja kontrollera motorn medhjälp av manuella CAN-meddelanden.

Nästa steg var att göra en kod för att kunna styra motorn via logiken. När jag visste vilka meddelanden jag ville sända för att motorn skall börja röra på sig, gick det relativt smärtfritt. Det som tog mest tid med PLC programmeringen var parameterhanteringen och handskakningen mellan slaven och mastern. Eftersom handskakningen mellan motorkontrollern och logiken kräver att man växlar mellan noll och ett var 20ms, så uppstod det små problem med programcykel tiderna. Vissa programcyklar kunde överstiga 20ms vilket ledde till att handskakningen mellan motorkontrollern och logiken avbröts. Jag konstaterade att programcyklarna oftast var längst vid uppstart av systemet. Problemet åtgärdades med att starta CANopen kommunikationen strax efter uppstart av logiken. Jag funderade också över om man använde sig av en logik som hade för dålig prestanda, men övervägde detta, när jag gjorde om starten av CAN kommunikationen, vilket ledde till ett fungerande system.

Parameterhanteringen skapade mycket problem när jag skulle köra över parametrar till motorstyrningen. Objekt index som var specificerade i tillverkaren av motorstyrningens datablad stämde inte överens med de objekt index som fanns i motorstyrningen. För att klura ut vilka objekt index som fanns i motorstyrningen körde jag igenom ett register med objekt indexen 2000h – 5FFFh och tittade vilka index man fick svar på. Jag behövde också ta reda på vilket objekt index som styrde vilken parameter. Detta gjordes med hjälp av att ändra parametrarna genom att sända CAN-meddelanden manuellt och sedan använda sig av ZpCANconsole för att se vilka parametrar jag har ändrat på. Detta var väldigt tidskrävande, men till slut fick jag allt att passa ihop.

Av detta examensarbete har jag lärt mig väldigt mycket. På grund av att projektet inkluderades av flera olika konfigurations plattformar och kommunikations system har jag arbetat och lärt mig hur saker fungerar över ett bredare område. CANopen kommunikationsstandarden har varit ett väldigt intressant område att fördjupa sig i, detta kommer jag att ha stor nytta av i framtiden. Efter att ha gjort ett program med strukturerad kod kunde jag konstatera att jag hade valt rätt programmeringsspråk för detta ändamål. Att göra detta i ladder eller funktionsblock hade varit rättsakt omöjligt. Av den erfarenhet jag har fått att programmera med detta programmeringsspråk kommer jag i framtiden att försöka använda det så mycket som möjligt.

All utrustning jag har använt under projektets gång har varit modern och användarvänligt, förutom Zapis konfigurerings verktyg som man kunde förnya. TIA-portalen, som likt alla andra program var nya för mig, fungerade bra och man kan konstatera att Siemens har lyckats väldigt bra. För mig som var nybörjare när jag startade med projektet, var det väldigt lätt med att komma igång med programmet. Alla program som har blivit inkluderade i en enda samlings portal fungerade utomordentligt bra. Till skillnad från föregångaren Step7 där alla program var utspridda och väldigt svår att få kontroll över har man lyckats väldigt bra med denna version.

I övrigt tycker jag att projektets praktiska och teoretiska del har varit mycket lärorikt. Jag har haft flera problem som jag har fått ta tag i och lösa med och utan utomstående krafter, men detta har också lett till att jag har lärt mig att samarbeta på olika språk, samt sett hur man löser problem i det verkliga arbetslivet. Skulle jag göra om projektet skulle jag kanske inte ha koncentrerat mig så mycket på parameterhanteringen, eftersom den blev mycket tidskrävande och inte var specificerad för projektet. Men pga. av att jag såg att detta skulle underlätta flera saker för framtiden, samt spara tid i framtiden, tog jag mig an uppgiften att även göra denna funktion.

Litteraturförteckning

[1] Kihl, M. & Andresson, J. A., 2013. *Datakommunikation och nätverk.* 1:1 red. Lund: Studentliteratur AB.

[2] Berti, V., Björkman, M., Lindgren, A. & Norden, L.-Å., 2012. *Data Kommunikation.* 1:1 red. Stockholm: Liber AB

[3] Mayer, K., 2003. *Datakommunikation i praktiken.* 2 red. Sundbyberg: Pagina Förlags AB.

[4] Pfeiffer, O., Ayre, A. & Keydel, C., 2003. *Embedded Networking with CAN and CANopen.* 1:a red. USA: Copperhill Technologies Corporation.

[5] CiA. (2016). *Technical documents.* http://www.can-cia.org/standardization/specifications/ (hämtad 03.12.2016)

[6] CiA 301. (2011). *CANopen application layer and communication profile.*

[7] CiA. (2016). *Technical documents.* http://www.can-cia.org/standardization/specifications/ (hämtad 04.12.2016)

[8] Penton. (2016). *Machine design.* http://machinedesign.com/engineering-essentials/engineering-essentials-what-programmable-logic-controller (hämtad 05.12.2016)

[9] AMCI. (2014). *AMCI Advanced micro controls inc.* http://www.amci.com/tutorials/tutorials-what-is-programmable-logic-controller.asp (hämtad 05.12.2016)

[10] Siemens. (2016). *Totally Integrated Automation Portal - Integrated EngineeringFramework.*http://www.industry.siemens.com/topics/global/en/tia-portal/Pages/default.aspx (hämtad 05.12.2016)

[11] PI. (2016). *Profibus profinet.* http://www.profibus.com/products/gsd-files/ (hämtad 05.12.2016)

[12] Beckhoff. (2011). *BK9103 and EK9300 Profinet Fieldbus Couplers on S7 PLC.*

Bilaga A Detaljerad beskrivning av CANopen-komponenter

A.1 EDS-fil

I en EDS-fil beskriver man hur object dictionary är implementerat för en specifik nod. För att analysera, konfigurera eller enbart använda en nod, behöver man en EDS-fil. Vid inköp av en CANopen enhet skall EDS-filen levereras av försäljaren.

A.2 Object dictionary

Varje CANopen nod måste ha en egen object dictionary. En object dictionary är en grupp av objekt index som finns tillgängliga för en specifik nod. Objekt indexen beskriver sedan var för sig vilken funktionalitet som finns hos noden. Varje object dictionary använder sig av en 16-bits adressering och 8-bitar för sub-index.

Index områden är delade in i flera olika sektioner (*Tabell 1,2,3*). Vissa sektioner är standardiserade såsom 0001h – 0FFFh och 1000h – 1FFFh, här anger man endast data typer och kommunikations parametrar. Ett område som man fritt kan använda till tillverknings specifika parametrar är 2000 - 5FFFh, här kan man t.ex. ange motorhastighet, temperatur osv.

Index Range	Description
0000h	Reserved
0001h – 0FFFh	Data Types
1000h – 1FFFh	Communication Entries
2000h – 5FFFh	Manufacturer Specific
6000h – 9FFFh	Device Profile Parameters
A000h – FFFFh	Reserved

Tabell 1.

Index Range	Description
0001h – 001Fh	Standard Data Types
0020h – 0023h	Pre-defined Complex Data Types
0024h – 003Fh	Reserved
0040h – 005Fh	Manufacturer Complex Data Types
0060h – 007Fh	Device Profile Standard Data Types
0080h – 009Fh	Device Profile Complex Data Types
00A0h – 025Fh	Multiple Device Modules Data Types
0260h – 0FFFh	Reserved

Tabell 2.

Index	Name
1008h	Manufacturer Device Name
1009h	Manufacturer Hardware Version
100Ah	Manufacturer Software Version
100Ch	Guard Time
100Dh	Life Time Factor
1010h	Store Parameters
1011h	Restore Default Parameters
1012h	COB ID Time
1013h	High Resolution Time Stamp
1014h	COB ID EMCY
1015h	Inhibit Time EMCY
1016h	Consumer Heartbeat Time
1017h	Producer Heartbeat Time
1018h	Identity Object
1200h – 127Fh	Server SDO Parameters
1280h – 12FFh	Client SDO Parameters
1400h – 15FFh	RxPDO Communication Parameters
1600h – 17FFh	RxPDO Mapping Parameters
1800h – 19FFh	TxPDO Communication Parameters
1A00h – 1BFFh	TxPDO Mapping Parameters

Tabell 3.

A.3 Processdata object (PDO)

För att överföra data mellan noder i realtid använder man sig av processdata object (PDO), t.ex. för mätdata och I/O diagnostik. En PDO kan överföra 8bytes data åt gången i endera riktningen, server – klient eller klient – server. En PDO är inte i direkt kontakt med objekt index, utan man använder en mappning där ett antal objekt index är mappade till en PDO. Antalet objekt index som rymmer en PDO är begränsade till max 64bitar, t.ex. har man ett objekt index som är 1 bit långt, så rymmer PDO:n 64 objektindex, detta pga. av att

mappningsprocessen arbetar på bit nivå. En PDO mappning innehåller en array av ”single mapping parameters” vilka är begränsade till max 32-bitars data längd. Detta betyder att har man ett objekt index på 64bitar måste man dela upp objektet i två enskilda mappningar.

Subindex	Name	Data type
0	Number of entries	UNSIGNED8
1	1st OD entry mapped	UNSIGNED32
2	2nd OD entry mapped	UNSIGNED32
3	3rd OD entry mapped	UNSIGNED32
4	4th OD entry mapped	UNSIGNED32
--	--	--
64	64th OD entry mapped	UNSIGNED32

Tabell 4.

Varje enskild mappning (single mapping parameters) kan man adressera ett objekt index till, vilket i sin tur består av Index, Subindex och datalängd på bitnivå.

Index	Subindex	Length (bits)
Bits 31 .. 16	Bits 15 .. 8	Bits 7 .. 0

Tabell 5.

Inom realtids dataöverföring finns det två typer av processdata objekts, TPDO och RPDO. TPDO kallas även också för producer och RPDO för consumer. RPDO använder man när man skall sända data till en nod. TPDO används när man vill läsa data från en nod. Denna typ av dataöverföring kan endast användas när enheten är i operational läge, vilket är det ända datapaket inom CANopen som enbart kommunicerar i operational operations läge.

A.4 TPDO och RPDO meddelande uppbyggnad

Genom att en PDO mappning består av flera objektindex, kan man läsa eller skriva data mot alla adresserade objekt index med ett enda meddelande. Vad som skiljer producern och consumern åt, är att dom kommunicerar på olika COB-ID. TPDO kommunicerar på COB-ID Node-ID + 180h och RPDO på COB-ID Node-ID + 200h. Om inget annat anges när man pratar om COB-ID så är det alltid TPDO1 och RPDO1, samt för SDO använder man alltid det första som finns listat.

PDO	COB ID
TPDO1	185h
RPDO1	205h
TPDO2	285h
RPDO2	305h
TPDO3	385h
RPDO3	405h
TPDO4	485h
RPDO4	505h

Tabell 6.

I tabell 7 visar man uppbyggnaden hos ett TPDO datapaket. I detta exempel har man använt sig av objekt index 6000h (digital inputs) och 6401h (analog inputs) vilka är mappade till en TPDO1 som har indexet 1A00h. Här beskrivs det vilken TPDO man använder, så COB-ID för denna TPDO kommer att bli Node-ID + 180h.

Index	SubIdx	Type	Description
1A00h			1st Transmit PDO - Mapping
	0	UNSIGNED8	= 4 (Number of used map entries)
	1	UNSIGNED32	= 6000 01 08h (Idx 6000h, SubIdx 1, 8 bit)
	2	UNSIGNED32	= 6000 02 08h (Idx 6000h, SubIdx 2, 8 bit)
	3	UNSIGNED32	= 6401 01 10h (Idx 6401h, SubIdx 1, 16 bit)
	4	UNSIGNED32	= 6401 02 10h (Idx 6401h, SubIdx 2, 16 bit)
6000h			Process data, digital inputs
	0	UNSIGNED8	= 2 (Number of sub-index entries)
	1	UNSIGNED8	8-bit digital input
	2	UNSIGNED8	8-bit digital input
6401h			Process data, analog inputs
	0	UNSIGNED8	= 2 (Number of sub-index entries)
	1	UNSIGNED16	16-bit analog input
	2	UNSIGNED16	16-bit analog input

TPDO1	D IN 1	D IN 2	A IN 1		A IN 2		Unused	
	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8

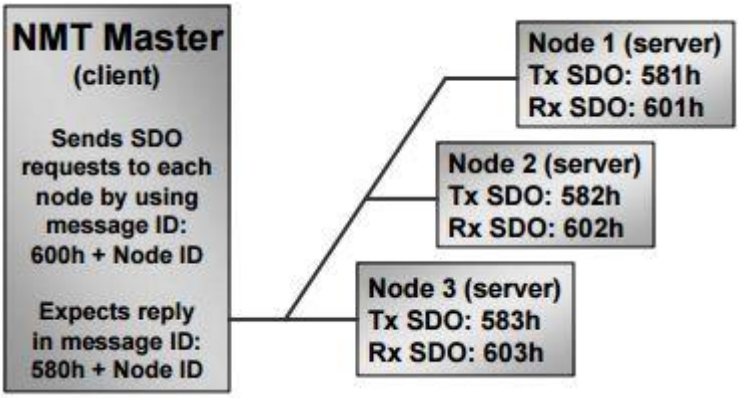
Tabell 7.

A.5 Service data object (SDO)

SDO används för att överföra asynkrona kommunikations meddelande mellan två CANopen enheter. Service data objekts används främst för konfiguration, men vissa leverantörer använder SDO även för dataöverföring. Varje SDO meddelande är 8 bytes, även om man inte använder allt utrymme i meddelandet. Två skillnader mellan PDO och SDO, är att SDO bekräftas alltid av mottagaren, klarar av att överföra mera data och kan även bestå av flera datatyper. När man kommunicerar med service data object är man i direkt kontakt med ett

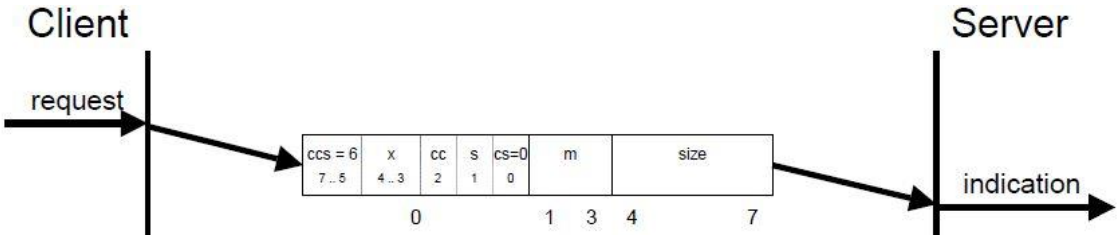
objekt index, vilket man inte är när man använder sig av PDO. SDO är det ända datapaketet som kan användas i både operational och preoperational operations läge.

När man skriver och läser meddelanden med SDO så använder man sig av en identifierare (COB-ID) som är implementerat i NMT mastern. När man sänder i riktningen Client – Server använder man sig av COB-ID 600h + Node-ID. Bekräftelse på meddelanden kommer att ligga på COB-ID 580h + Node-ID.



Tabell 8.

A.6 Klient till server SDO meddelande uppbyggnad



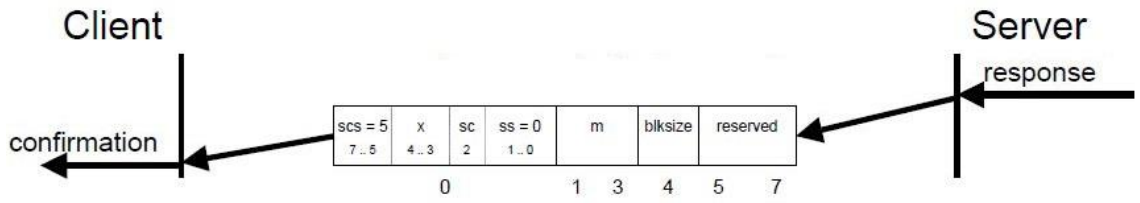
Tabell 9.

En klient till server SDO meddelande är uppbyggd av följande komponenter. Komponenter som inte används skall alltid vara 0:

- COB-ID
  - Client – Server = Node-ID + 600h
- Byte 0, Specifier
  - Bit 5-7, CCS: Client command specifier
    - Segment nerladdnings förfrågan CCS = 0

        - Initierad nerladdnings förfrågan CCS = 1
        - Initierad uppladdnings förfrågan CCS = 2
        - Uppladdnings förfrågan av ett SDO segment CCS = 3
        - Avbryta en SDO överföring CCS = 4
        - SDO block uppladdning CCS = 5
        - SDO block nerladdning CCS = 6
    - Bit 3-4, X: Reserved
        - Reserverad
    - Bit 2, CC: Client CRC support
        - Klienten stöder inte CRC CC = 0
        - Klienten stöder CRC CC = 1
    - Bit 1, S: Size indicator
        - Storlek på data är inte indikerat S = 0
        - Storlek på data är indikerat S = 1
    - Bit 0, CS: Client subcommand
        - Initiera nerladdnings förfrågan CS = 0
- Byte 1-2, M: Multiplexer
    - Object index
        - Representerar objekt index som skall bli överförd av SDO
            - Byte 1 = LSB
            - Byte 2 = MSB
- Byte 3, M: Multiplexer
    - Subindex
        - Representerar SUB-objekt index som skall bli överförd av SDO
- Byte 4-7, Size
    - Data som skall överföras till SDO
        - Kan endast användas om size indicatorn (S) = 1
            - Byte 4 = LSB
            - Byte 7 = MSB

A.7 Server till klient SDO meddelande uppbyggnad



Tabell 10.

Ett serve till client SDO meddelande är uppbyggd av följande komponenter. Komponenter som inte används skall alltid vara 0:

- COB-ID
  - Server – Client = Node-ID + 580h
- Byte 0, Specifier
  - Bit 5-7, SCS: Server command specifier
    - Uppladdnings respons av ett SDO segment SCS = 0
    - Segment nerladdnings respons SCS = 1
    - Initierad uppladdnings respons SCS = 2
    - Initierad nerladdnings respons SCS = 3
    - SDO block uppladdning SCS = 2
    - SDO block nerladdning SCS = 5
    - SDO block uppladdning SCS = 6
  - Bit 3-4, X: Reserved
    - Reserverad alltid X = 0
  - Bit 2, SC: Server CRC support
    - Klienten stöder inte CRC SC = 0
    - Klienten stöder CRC SC = 1
  - Bit 0-1, SS: Server subcommand
    - Initiera nerladdnings respons SS = 0
- Byte 1-2, M: Multiplexer
  - Objekt index
    - Representerar objekt index som skall bli läst av SDO
      - Byte 1 = LSB
      - Byte 2 = MSB
- Byte 3, M: Multiplexer
  - Subindex

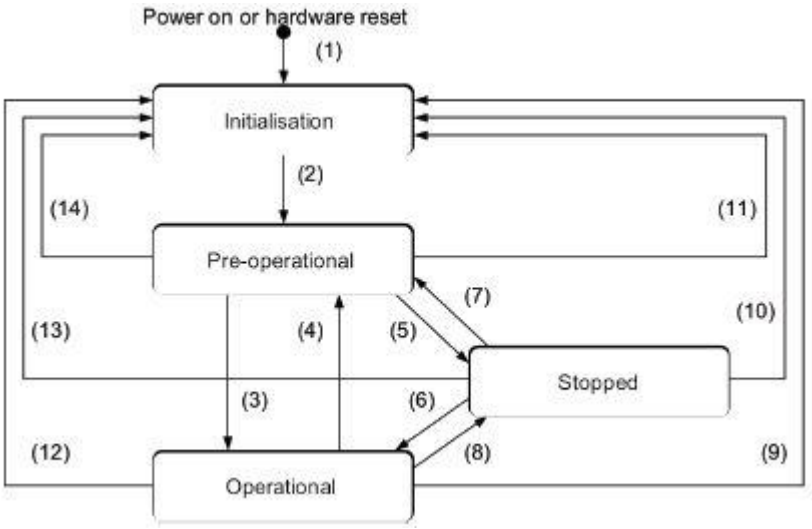
- Representerar SUB index som skall bli läst av SDO

- Byte 4, Blksize
  - Antal segement som skall användas per block av klienten för block nerladdning 0 < blksize < 128
- Byte 5-7,Reserved
  - Reserverad alltid 0

Bilaga B CANopens olika uppstartningsfaser

B.1 Network management (NMT)

Varje CANopen klient måste ha implementerat en NMT state machine. State maskinen tillåter noden att vara i olika operationslägen. Vissa operationsläges övergångar kan ske automatiskt i enheten, medan vissa övergångar endast kan ske när klienten har fått ett NMT master meddelande. Ett NMT master meddelande sänds från server enheten och kan sändas till alla noder på samma gång, eller endast till en nod på nätverket.



Tabell 11.

Beskrivning av olika operations övergångar i tabell 11:

1. Power-On or hardware reset:
   - Vid uppstart av klienten kommer övergången till operationsläge Initialisation ske automatiskt

2. Initialisation:
   - Efter initialisations steget är klart kommer NMT att övergå till Pre-operational automatiskt

3. NMT service startar Node-IDentifikation, eller väntar på NMT master meddelande att den kan gå över i operationsläge Operational

4. Om ett NMT master meddelande har mottagits, Startar NMT service identifikation om klienten kan återgå till Pre-operational operationsläge

5. NMT service begär att gå i stoppläge, från att vara i Pre-operational läge

6. NMT service begär att få gå i Operational läge, direkt efter stopp

7. NMT service startar Node-IDentifikation, eller väntar på NMT master meddelande att den kan gå över i operationsläge Pre-operational

8. NMT service begär att gå i stoppläge, från att vara i Operational läge

9. (10), (11) NMT service kommer att göra en omstart av Node-IDentifikationen

12. (13), (14) NMT service kommer att göra en omstart av kommunikations identifikation

B.2 NMT state initialisation

Operations läge initialisations är det första läget state maskinen kommer att befinna sig i efter uppstart eller omstart av klienten. Initialisations läget kan delas in i tre delar.

- Initialising
- Reset application
- Reset communication

Den första delen ”initialising” kommer NMT att befinna sig i direkt efter omstart, eller uppstart av systemet. Därefter kommer den automatiskt att övergå till ”reset application”. I detta steg kommer alla objekt index som finns inom sektorerna 2000h – 5FFFh och 6000h – 9FFFh att ställas in till definierade uppstarts värden. Efter att objekt indexen har fått sina uppstarts värden kommer övergången till ”reset communication” att ske automatiskt. Reset communication är det sista steget av NMT state initialisation. I detta steg kommer alla objekt index som finns inom sektorn 1000h – 1FFFh att ställas in till uppstarts värden. När alla dessa tre steg är genomförda kommer det att ske en uppstart och en övergång till operationsläge pre-operational.

B.3 NMT state pre-operational

Pre-operational är ett läge som används för service och konfiguration. I detta läge av state maskinen går det inte att kommunicera med processdata objekt. SDO data är tillåtet att användas och kommer att användas i.o.m konfiguration. NMT kommer att befinna sig i pre-operational läge ända tills den får ett NMT master meddelande om att den kan övergå till operational, eller om den är inställd på att göra detta automatiskt.

B.4 NMT state operational

NMT state operational är enda operations läge det är tillåtet att kommunicera med alla kommunikationspaket som CANopen stöder.

B.5 NMT state stopped

Genom att ställa in NMT till att arbeta i state stopped, så kommer kommunikationen till samtliga noder att avbrytas. Ända kommunikation som kommer att finnas kvar är synk meddelanden. Får man ett nödstopps meddelande från bussen kommer också state stopped att triggas.

	Pre-operational	Operational	Stopped
PDO		X	
SDO	X	X	
SYNC	X	X	
TIME	X	X	
EMCY	X	X	
Node control and error control	X	X	X

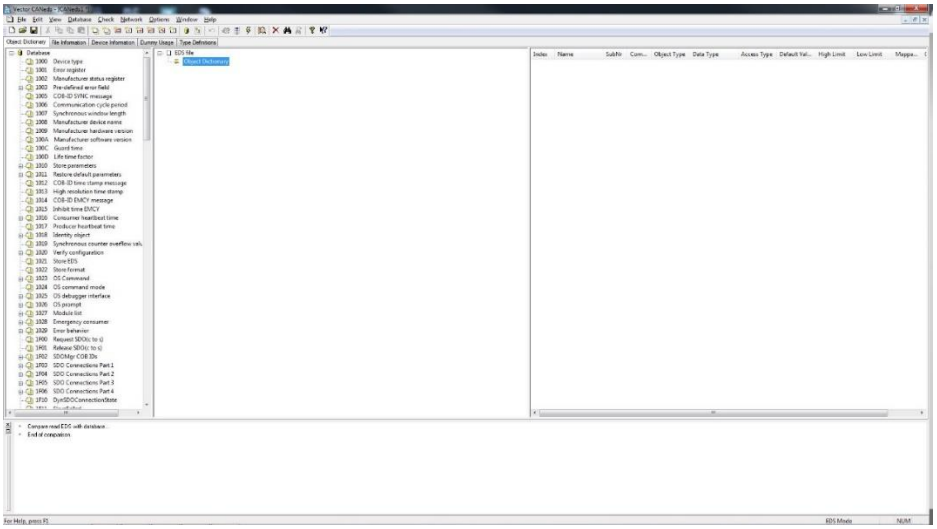
Tabell 12.

Bilaga C Beskrivning av programvara

C.1 Vector CANeds

För att kunna kommunicera med en CANopen node behöver man en EDS-fil för att veta vilka PDOmappar och Objekt index man kan använda. Oftast när man köper en CANopen enhet så får man också en EDS-fil. I detta fall så finns det inga färdiga EDS-filer. Grunden till att ZAPI inte har några EDS-filer till sina motorstyrningar är att ZAPI endast kommunicerar med sina egna enheter, som färdigt har inbyggda CANopen mjukvaror för att vara kompatibla med varandra.

Genom att använda Vector CANeds för att tillverka EDS-filer har man tillgång till alla standard objekt och deras attribut som CiA standarden stöder. I vänstra fliken på tabell 13 finns en hierarkisk trädstruktur som representerar objektattribut som objekt index och datatyper.



Tabell 13.

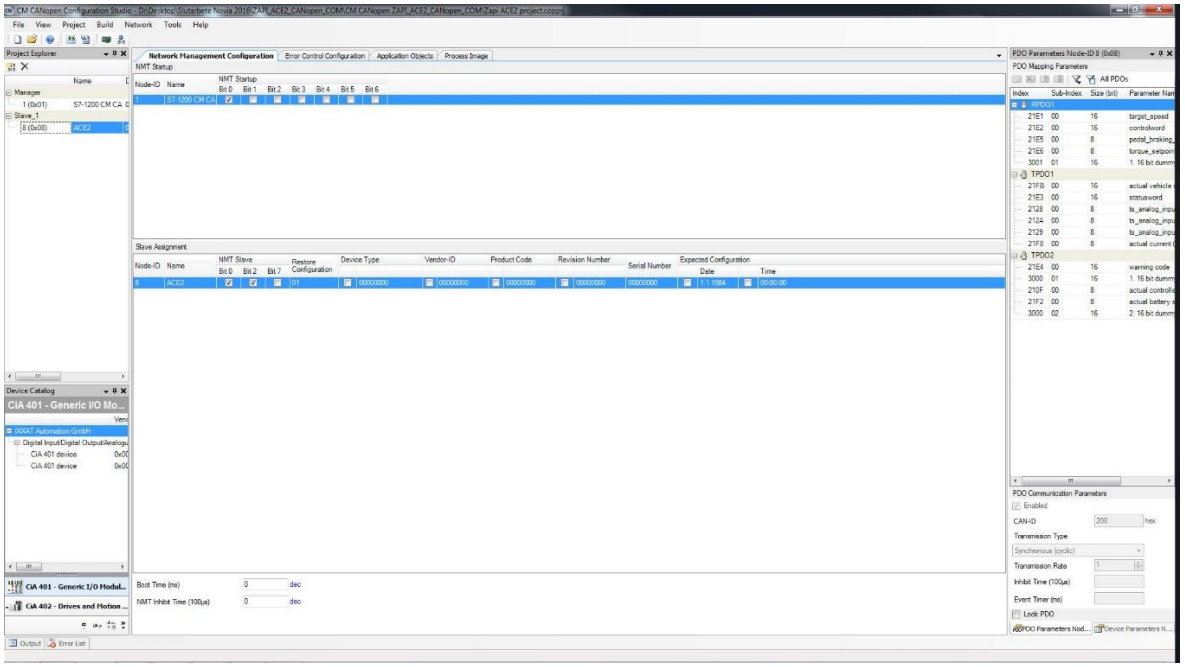
C.2 CM CANopen Configuration Studio

För att man skall få åtkomst till CM CANopen modulen via TIA-portalen så krävs det att man konfigurerar kommunikationsmodulen. Konfigurationen görs med programmet CM CANopen Configuration Studio. CM CANopen Configuration Studio är ett program som har tagits fram av tyska företaget IXXAT.

När man har ett projekt som är färdigt konfigurerat kan man se i vänstra fliken på vyn i tabell 14 vilka noder man har på nätverket. Högra sidan av vyn ser man vilka PDO:n man kommer att kunna kommunicera med. Om man tittar i mitten av projekt vyn så finner man fyra olika

flikar. Under första fliken *Network Management Configuration* ställer man in hur man vill använda NMT state maskinen för både slaven och mastern. Har man en slave som stöder *heartbeat control* eller *node guarding* är det möjligt att konfigurera detta under Error Control Configuration vilket är andra fliken. Under fliken *Application Objects* finner man alla mappnings bara PDO. Genom att höger klicka på fliken och välja *Select Default Mappings of Device* kommer man få samma PDO mappning som man har i EDS-filen. Om man går under fliken *Process Image* kommer man att hitta alla PDO mappar som man valde under *Application Objects* fliken.

Efter att man har konfigurerat CM CANopen modulen är det möjligt att ladda ner konfigurationen till modulen. Och om allting går bra kommer RUN status indikationen att lysa grönt på modulen.



Tabell 14.

C.3 Zp CanFlasher och Zp CanConsole

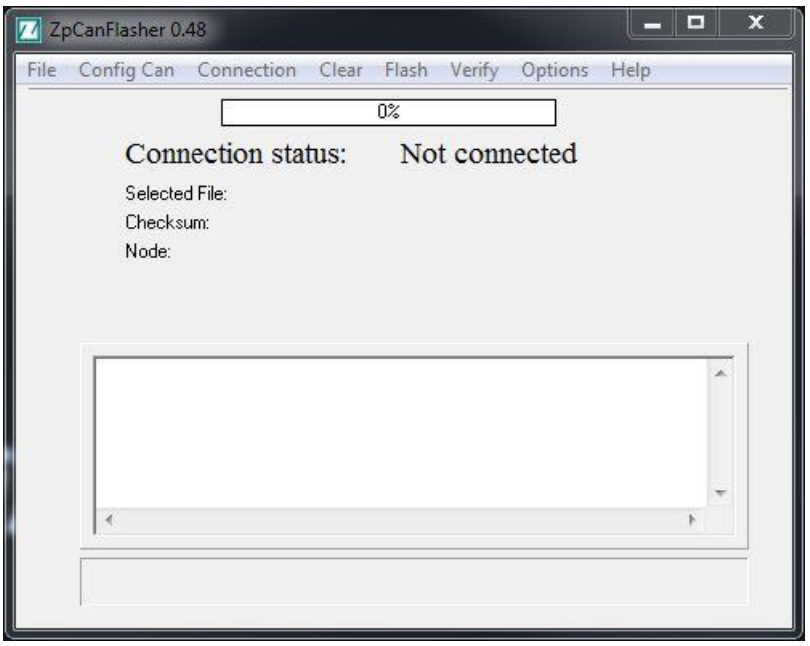
Konfigurationsprogrammen ZpCanFlasher och ZpCanConsole är program som är utvecklade av företaget ZAPI. Med dessa program konfigurerar man ZAPI-motorstyrningar. ZpCanFlasher använder man när man skall byta mjukvara i en kontroller. För att kunna byta mjukvara med ZpCanFlasher behöver man en CANmjukvara.

I mjukvarans fil namn kan man se om det är en CANmjukvara eller en I/O mjukvara man har. Har mjukvaran ett m i filnamnet kan den användas både med ZpCanFlasher och

ZpWinFlasher. Har mjukvaran IO i slutet av filnamnet kan den endast användas med ZpWinFlasher. Ända skillnaden mellan ZpWinFlasher och ZpCanFlasher är att dom använder olika kommunikations metoder för att kommunicera med motorstyrningen. ZpWinFlasher använder sig av vanlig serie kommunikation medan ZpCanFlasher använder sig av CANopen kommunikation.

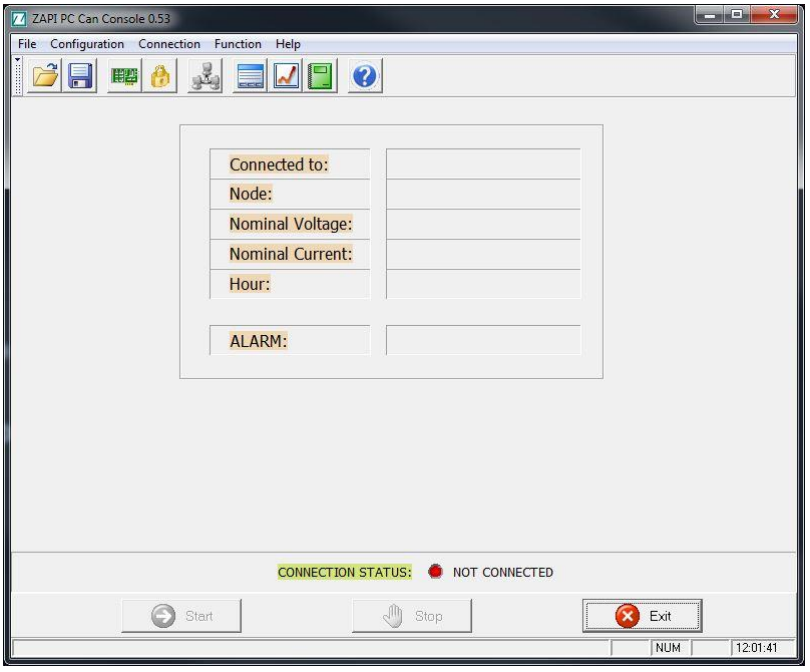
Skall man använda sig av ZpCanFlasher behöver man ett CAN interface för att koppla in sig på motorstyrningen. Detta CAN interface kan vara av vilket fabrikat som helst. Använder man sig av serie kommunikation behöver man en adapterkabel som man kan köpa av ZAPI. Adapter kabeln har en flashnings adapter som man måste ansluta till kabeln om man skall byta mjukvara i en kontroller. Flashnings adaptern består endast av en länk mellan två pinnar i stöpseln.

I tabell 15 kan man se en vy över ZpCanFlasher start sida. På startsidan kan man direkt se om man är uppkopplad till en motorkontroller. Fliken *File* upptill använder man för att välja mjukvara och typ av kontroller. *Connection* fliken används för att upprätthålla kommunikation mellan PC och motorkontroller. När man har vald mjukvara och kontroller typ och har en kommunikation mellan PC:n och kontrollenheten kan man välja att flasha kontrollern. Detta görs under fliken *Flash*. Efter att man har flashat en motorkontroller enhet bör man alltid verifiera mjukvaran, detta görs under fliken *Verify*. Om programmet inte meddelar några fel efter att man har verifierat enheten, så har man lyckats med att byta mjukvaran.



Tabell 15.

Direkt från ZpCanConsoles startsida som visas i tabell 16 kan man se vilken motorkontroller man är sammankopplad med, samt vilken node-id, nominal spänning, nominal ström, drift tiden för motorkontroller enheten och alarm. För att upprätthålla en kommunikation mellan programvaran och kontrollerenheten går man tillväga på samma sätt som för ZpCanFlasher. Konfigurering av parameterinställningar kan man göra under fliken *Functions*. Under denna flik kommer man också åt en tester. Testern är en sorts realtids diagnostik meny, där man kan se ÄR och BÖR-värden för vissa parametrar i motorkontrollern.



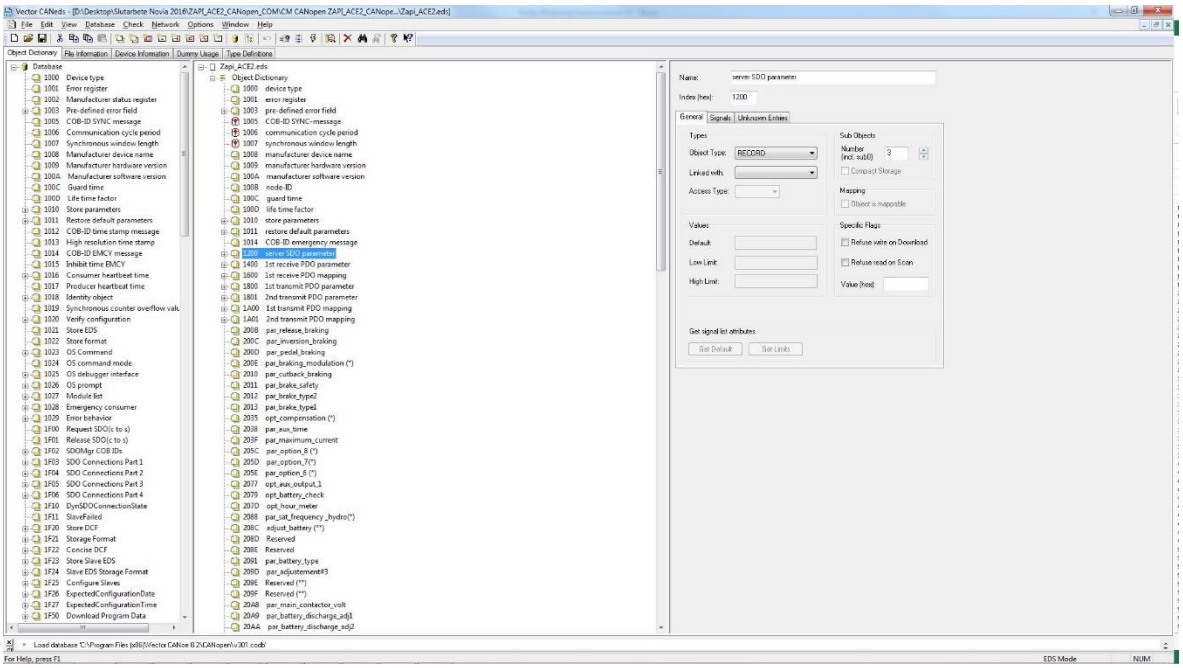
Tabell 16.

Bilaga D Detaljerad konfigureringsmetod

D.1 Framställning av EDS-fil

Tillvägagångssättet när man tillverkar en EDS-fil med hjälp av Vector CANeds, är att man först av allt gör ett nytt projekt. Efter att man har gjort ett nytt projekt kan man börja ”drag and drop” objekt index från den hierarkiska trädstrukturen i vänster sida av vyn i tabell 13. För att veta vilka objekt index man behöver ha i sin EDS-fil är det viktigt att man har ett CAN protokoll eller liknande som beskriver vilka objekt index som finns i den enhet man tillverkar EDS-filen till. CAN protokollet bör också beskriva av vilken åtkomst typ och data typ objekt indexen är. CAN protokollet till detta projekt kan man se som Bilaga G.

När man gjorde EDS-filen till detta projekt började man med att sätta in alla index inom sektionen 1000h till 1FFFh vilka är definierade som Communication entries. Därefter plockade man in alla index inom sektionen Manufacture specific, vilka är alla index från 2000h till 5FFFh. Alla objekt index och subindex har fördefinierade namn. Dessa namn döper man om så indexen kommer att heta samma som i CAN protokollet för motorstyrningen.



Tabell 17.

Efter att man hade plockat in alla index, subindex och ändrat namn på dem började man definiera olika typer av index data för varje enskilt index. Till index data hör:

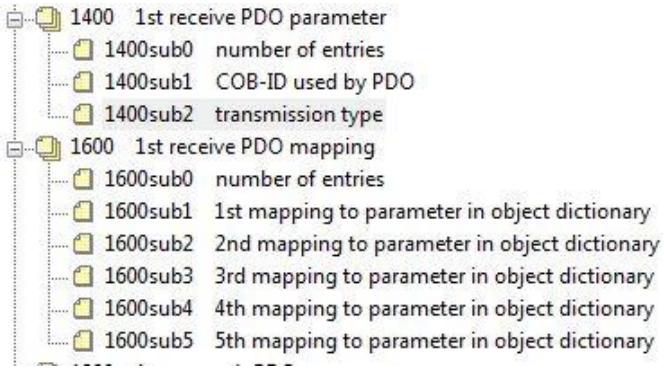
- Objekttyp
- Datatyp,
- Åtkomsttyp
- Standard värden

Genom att klicka på ett objekt index eller subindex kan man definiera all behövlig data. Se till höger av vyn i tabell 17. Varje index måste ha definierade alla dessa data typer för att en CANopen enhet skall kunna veta om man kan skriva till indexet eller bara läsa, vilken typ av data man kan skriva/läsa och inom vilket områden data kan vara som man läser/skriver.

PDO mappning är en väldigt viktig del av EDS-filen. Har man inte gjort någon PDO mappning så kommer man inte att kunna skicka eller ta emot processdata objekt. En PDO mapp måste huvudsakligen bestå av fyra delar för att vara rätt konfigurerad.

- Ett objekt som beskriver vilken COB-ID man kommer att sända PDO:n på
- Ett objekt som beskriver vilka mappar man använder
- En mapp måste i sin tur bestå av ett index
- Indexet måste vara vald att gå att mappa. Detta gör man genom att klicka i rutan *Object is mappable*. Se till höger av vyn i tabell 17

I denna EDS-fil använder man sig av en RPDO och två stycken TPDO. Tabell 18 visar hur RPDO:n ser ut på index nivå. I subindex 1400sub1 anger man vilken COB-ID RPDO:n kommer att användas sig av. Objekt index 1600 är ett index där man samlar alla PDO mappar för RPDO1. Alla subindex i objekt 1600 består av mappar som man har länkat ihop med ett objekt index som man vill ha mappat. T.ex. index för motorhastighet.



Tabell 18.

Alla mappade objekt som man kommer att kunna använda som in och utgångar i PLC är listade nedan:

- RPDO1
  - BÖR värde för motorhasighet (Byte 0-1)
  - Controlword (Byte 2-3)
  - Pedal broms begäran (Byte 4)
  - Vridmoments begäran (Byte 5)
  - Acceleration/Deacceleration ramp (Byte 6)
- TPDO1
  - ÄR värde för motorhastighet (Byte 0-1)
  - Statusword (Byte 2-3)
  - Analog ingång 3 (Byte 4)
  - Analog ingång 1 (Byte 5)
  - Analog ingång 2 (Byte 6)
  - Motor ström (Byte 7)
- TPDO2
  - Varningar och alarm (Byte 0)
  - Temperatur (Byte 4)
  - BDI (%) (Byte 5)
  - Motor temperatur (Byte 6)

Controlword och statusword som man hittar i RPDO1 och TPDO1 är data paket av typen word, vilket betyder två byte eller 16bitar. Dessa datapaket innehåller ett antal bitar som varje specifik bit har en speciell funktion. Controlword är ett datapaket som man styr motorkontrollern med och statusword är ett diagnostiseringspaket som man kan läsa status

av specifika funktioner i motorkontrollern. Nedan har man listat vad alla enskilda bitar betyder i control och status wordet:

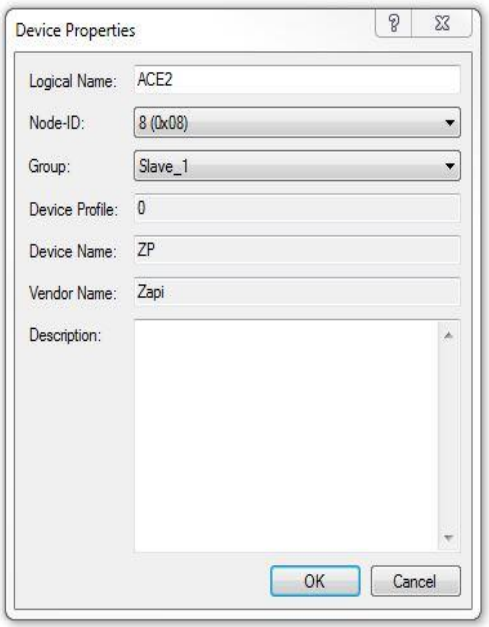
- Controlword
  - Utgång A16 (Byte 2.1)
  - Utgång A18 (Byte 2.2)
  - Aktivering av motorkontroller (Byte 2.3)
  - Felmeddelande kvittering (Byte 2.4)
  - Utgång A18 effekt på (Byte 2.5)
  - Rotations riktning medsols (Byte 2.6)
  - Rotations riktning motsols (Byte 2.7)
  - Hydraulik motor styrning (Byte 3.0)
  - Säkerhetsutgång för tillbaka koppling av säkerhet (Byte 3.1)
  - Oanvänd (Byte 3.2)
  - Bromstyp (Byte 3.3-4)
  - Oanvänd (Byte 3.5)
  - Oanvänd (Byte 3.6)
  - Stuffing (Används för handskakning) (Byte 3.7)

- Statusword
  - Utgång A16 (Byte 2.1)
  - Utgång A18 (Byte 2.2)
  - Aktivering av motorkontroller (Byte 2.3)
  - Utgång A19 (Byte 2.4)
  - Oanvänd (Byte 2.5)
  - Oanvänd (Byte 2.6)
  - Oanvänd (Byte 2.7)
  - CNA13 (ID1) (Byte 3.0)
  - CNA5 (ID0) (Byte 3.1)
  - CNA6 (Byte 3.2)
  - CNA4 (Nödstopp) (Byte 3.3)
  - Oanvänd (Byte 3.4)
  - Oanvänd (Byte 3.5)
  - Tomgång (Byte 3.6)
  - Stuffing (Används för handskakning) (Byte 3.7)

Alla COB-ID:n i en EDS-fil har liknande uppbyggnad. Uppbyggnaden är Node-id + identifierare. Identifieraren kommer vara olika på alla COB-ID. T.ex. för RPDO1 är identifieraren 200. I detta projekt kommer man använda sig av Node-id 8 för motorkontrollern. Detta leder till att COB-ID för RPDO1 kommer att vara 208 och för en client – server SDO med identifierar nummer 600 kommer COB-ID:n att vara 608.

D.2 Konfigurering av kommunikationsmodul

I CM CANopen Configuration Studio konfigurerar man hur man vill använda CM CANopen modulen, om man vill använda den som slave eller master. I detta fall kommer man att konfigurera den att var master. För att konfigurera slaven på CAN-nätverket, vilket i detta fall kommer att vara motorkontrollern, behöver man importera en EDS-fil. När man har importerat EDS-filen för motorkontrollern som man gjorde i Vector CANeds så kan man börja själva konfigureringen. Tillvägagångssättet för att genomföra en konfiguration av kommunikationsmodulen delas in i fem steg.



Tabell 19.

Första steget av konfigureringen är att välja vilket node-id motorstyrningen kommer att ha och om den skall vara master eller slave på nätverket. Konfigurationen gjordes enligt tabell 19. Som node-id valde man att använda nummer 8. Detta pga. av att motorkontrollerenheten som standard är vald att vara node-id 8. Eftersom att denna enhet kommer att användas som en klient på nätverket kommer den att höra till gruppen slave. Device profile, Device Name och Vendor Name är färdigt definierat i EDS-filen. Ända namnet man kan ange kontrollern i detta skede är det logiska namnet.

Andra skedet i konfigureringen är att välja hur man vill att NMT skall användas för både mastern och slaven. Eftersom att motorkontrollern fungerar som slav på nätverket vill man att den skall ha en slave NMT och var tillgänglig på hela nätverket. För att få definierat detta i master mjukvaran använder man sig av bit noll i slavtilldelningen. Bit två i kommer man att använda sig av för att mastern skall kunna kontrollera operationslägen i NMT slaven vid eventuella fel eller vid uppstart. Vill man att slaven skall grund inställas ti

fabriksinställningar genom att använda objekt index 1011h vid start skall man använda sig av bit sju, vilket är ointressant i detta fall.

Efter att man har gjort slavtilldelnings konfiguration bör man också definiera hur man använder NMT i mastern vid uppstart. Ända biten man kommer att använda sig av här är bit noll. Med bit noll definierar man att CM CANopen modulen kommer att ha en NMT master.

Betydelsen av bitarna man inte använder har man kort beskrivit nedan:

- Bit 1 – Vid uppstart av NMT mastern börjar man att starta alla noder, med början från Node-id 0.

- Bit 2 – NMT mastern skall inte byta operationsläge till operational av sig själv.

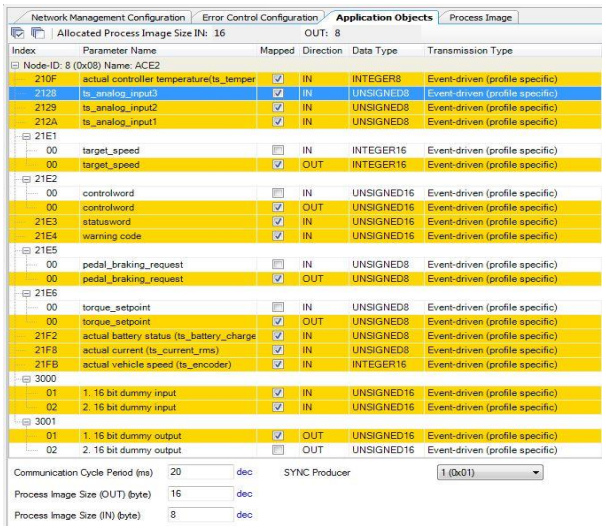
- Bit 3 – NMT skall inte starta NMT i slavarna, utan detta görs via programmet i logiken.

- Bit 4 – Om en slave på nätverket rapporterar ett fel, kommer slavarna att startas om, med början från Node-id 0.

- Bit 5 – Tilldelar master enheten högst prioritet vid en NMT flygande master förhandling. Denna enhet kommer att startas först av alla.

- Bit 6 – Om en slave på nätverket rapporterar ett fel, kommer slavarna att sättas i stoppläge, med början från Node-id 0. I fall denna bit används kan man inte använda bit 4.

Konfigurationens tredje steg är att definiera heartbeat och node guarding för kontroll av fel på CAN nätverkket. I denna tillämpning kommer man inte att använda sig av vilkendera heartbeat kontroll eller node guarding. Detta pga. av att CM CANopen modulen kommer att starta node guardingen när den har fått ett relevant meddelande av slaven att den är i operational opertionsläge. Slaven i detta fall sänder inga status meddelanden för operationsstatusar, vilket leder till att kommunikationsmodulen aldrig kommer att starta node guardingen, vilket resulterar att modulen kommer att meddela ett felmeddelande.



Tabell 20.

I steg fyra så konfigurerar man allt som har med process dataöverföring att göra. Till detta hör bl.a. att definiera PDO som man kommer att använda som in och utgångar i PLC programmet. Alla PDO som man har gjort mappnings bara i EDS-filen kommer man att ha tillgång i denna del av konfigurationen. I nedre delen av vyn i tabell 20 har man allokerat hur mycket data man använder för att ta emot data, samt sända data. För denna typ av slav konfiguration använde man 16 byte som man använder som in data från motorkontrollern, samt åtta byte som man använder för ut data.

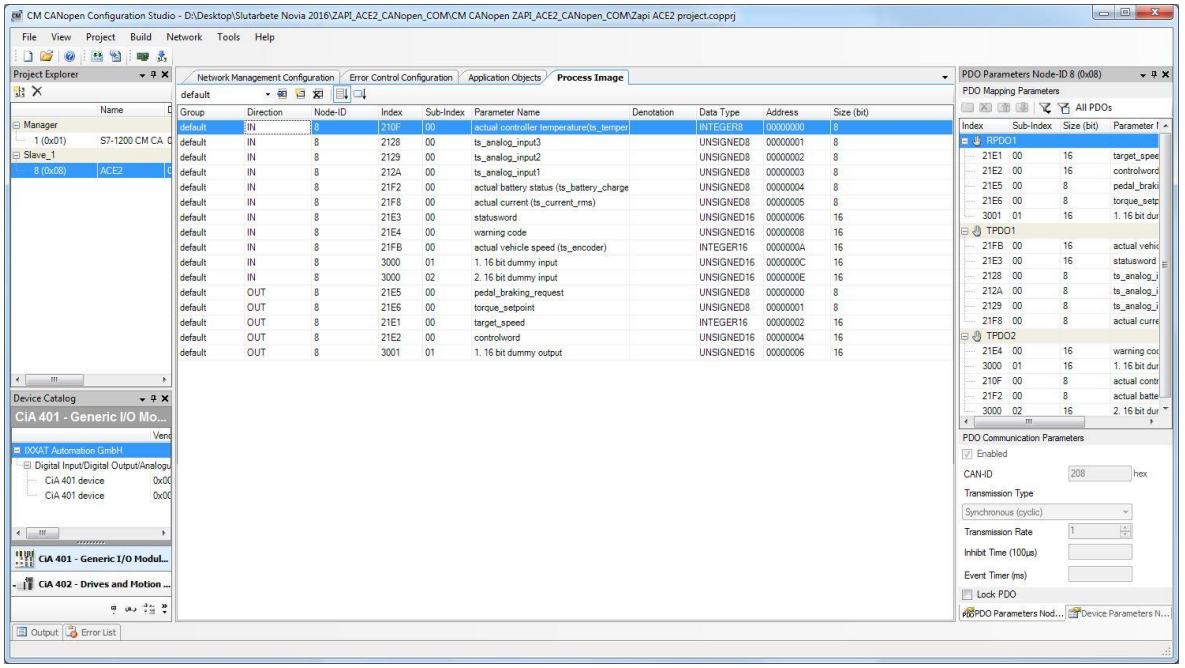
För att slaven skall gå i operational läge så behöver man sända ett synk meddelande var 20ms, se bilaga C.4. För att kunna sända ett synk meddelande måste man definiera vilken enhet som skall sända synken. I detta fall kommer synk produseraren att vara CM CANopen modulen. Detta definierar man också i nedre delen av vyn i tabell 20.

Genom att höger klicka inne i mitten av vyn i tabell 20 kan man välja att använda standard mappning för enheten. Genom att välja detta kommer man att använda alla PDO mappar som är gjorda i EDS-filen. Jämför man det gul markerade områdena i tabell 20 med PDO mapparna i CAN protokollet bilaga G.3 så ser man att PDO mapparna har blivit rätt valda.

Sista steget är att ladda ner konfigureringen till CM CANopen modulen. Men före man kan ladda ner konfigurationen måste man kalkylera och generera sammanställningen. När man kalkylerar gör bakgrundsprogrammet en beräkning på sammanställning, för att kontrollera att allting är intakt. Vid själva genereringen kommer man att skapa en fil som man sedan kan använda för att ladda ner i själva kommunikationsmodulen.

Genom att använda sig av tangenten F5 kommer man att göra en beräkning på konfigurationen. Output fönstret nere på vyn i tabell 21 kommer att meddela att beräkningen lyckades, om man har gjort en fullständig konfigurering. Efter att man har en godkänd version av sammanställningen kan man generera ner konfigurationen till en .cmcdc fil. Detta

gör man genom att använda sig av F6 tangenten eller går under fliken *Build* i övre kanten av tabell 21.



Tabell 21.

Genom att anväda sig av den genererade .cmcdc filen kan man ladda ner konfigurationen till kommunikationsmodulen. Första delen i nerladdnings sekvens är att placera logiken i operationsläge STOP. Detta kräver att man har gjort en hårdvarukonfiguration för PLC:n. Genom att använda sig av fliken *Network* och *download* kommer man att kunna välja vilken konfigurations fil man vill ladda ner. När man har lyckats ladda ner .cmcdc filen är man klar med konfigurationen av CM CANopen modulen och kan börja med att göra själva kommunikations mjukvara i logiken.

D.3 Byte av mjukvara i motorkontroller

Som man nämner ovan behöver man byta mjukvara i kontrollen för att kunna kommunicera via CANopen. Mjukvaran man kommer att använda sig av i detta projekt är en CAN mjukvara för en motor med en encoder på 48pulser per revolution. Eftersom man inte hade en mjukvara som var lämpad för denna typ av kommunikation, tog man kontakt med ETP kraftelektronik AB i sverige. ETP tog fram en mjukvara som skulle vara kompatibel med komponenterna man använder i detta projekt. För att flasha motorkontrollern valde man att använda Zp CanFlasher utöver den seriella versionen Zp WinFlasher.

Första steget när man byter mjukvara i en ZAPI inverter är att ansluta enheten med systemspänning. Detta görs genom att koppla 48VDC till key ingången som finns på stift A1. Key signalen är en signal som man använder för att enheten skall vakna till liv. I en eventuell nödstopps krets kan man bryta denna signal för att säkerställa att man inte har någon aktivitet i motorn.

Andra steget är att ansluta sig till CAN-nätverket med ett CAN interface och starta upp Zp CanFlasher. CAN interfacet man använder i detta projekt är av märke Kvaser. Interfacet består av en CAN – USB konverter, som man kan ansluta till en vanlig USB port på en PC. I detta skede i programmet väljer man vilken mjukvara och vilken motorkontroller enhet man vill använda. Dessa valmöjligheter hittar man under fliken *File* i övre kanten av tabell 13.

Tredje steget är att sammankoppla PC:n med motorkontrollern, radera den gamla mjukvaran i enheten och skriva in en ny. För att upprätthålla kommunikation mellan enheterna använder man sig av fliken *Connetion* och Start. När kommunikationsstatusen meddelar att man har en sammankoppling mellan enheterna kan man använda sig av fliken *Clear* för att radera den gamla mjukvaran i motorkontrollern. Efter att man har raderat den gamla mjukvaran kan man skriva in den nya mjukvaran. Detta gör man enkelt genom att gå under fliken *Flash* för att sedan skriva ner mjukvaran.

Fjärde och sista steget i denna process är att verifiera den ny inlagda mjukvaran och formatera enhetens eeprom. Verifiering av enheten kommer man åt under fliken *Verify*. Efter att invertern är verifierad är man klar för att rensa enhetens eeprom. Detta gör man för att säkerställa att inga gamla parameterinställningar finns kvar i enheten. För att utföra en eeprom formatering behöver man koppla upp sig med programmet Zp CanConsole. Under fliken *Function*, hittar man en eeprom funktion som man kommer åt att formatera enhetens minne. Efter att formateringen är klar gör man en omstart av enheten. Efter omstart av enheten kommer man att ha en motorkontroller som använder CANopen som kommunikations gränssnitt.

Bilaga E Framställning av funktionsblock

E.1 Framställning av funktion block

När man startar upp motorkontrollern kommer den att befinna sig i startuppläge. I detta läge kommer man inte att ha några andra meddelanden på bussen än TPDO2 och TPDO4. TPDO2 vilket är en PDO som använder sig av COB-ID 288h är en transmit PDO. Denna PDO används för att kunna läsa felmeddelanden och temperatur hos motorkontrollern. Oberoende av vilket operations status enheten kommer att vara i måste man ha möjlighet att kunna läsa felmeddelanden. TPDO4 använder sig av COB-ID 488 och är en PDO som man inte har funnit någon information om vad den används till i denna kontrollerenhet. Pga. av att den inte är definierad i någon EDS-fil eller CAN protokoll, konstaterade man att detta är frågande om ett processdata objekt som bakgrundsprogrammet använder sig av.

488	1	8	11 10 47 01 42 00 00 03	115.496	2700	Rx
488	1	8	11 10 47 01 42 00 00 03	115.528	2701	Rx
488	1	8	11 10 47 01 42 00 00 03	115.560	2702	Rx
288	1	8	42 ff 00 00 19 00 00 00	115.568	249	Rx
488	1	8	11 10 47 01 42 00 00 03	115.592	2703	Rx
488	1	8	11 10 47 01 42 00 00 03	115.623	2704	Rx
488	1	8	11 10 47 01 42 00 00 03	115.655	2705	Rx

Tabell 22 Motorkontroller NMT state bootup.

Första CAN meddelandet man bör använda sig av är ett synk meddelande. Genom att sända ett synk meddelande varje 20ms kommer motorkontrollern att börja synkronisera med master noden. Synkroniseringen kommer tas i bruk när man begär att kontrollen skall övergå till operations läge operational. Orsaken till att man börjar sända synk meddelanden (COB-ID 80h) fören kontrollen har övergått i operations läge operational är att mastern och slaven måste vara synkroniserade redan vid första CAN meddelandet som kommer att komma vid uppstart av operational läge.

488	1	8	11 10 47 01 42 00 00 03	163.011	4197	Rx
80	1	8	00 00 00 00 00 00 00 00	0.000	3171	Tx
488	1	8	11 10 47 01 42 00 00 03	163.043	4198	Rx
80	1	8	00 00 00 00 00 00 00 00	0.000	3172	Tx
288	1	8	42 ff 00 00 19 00 00 00	163.051	436	Rx
80	1	8	00 00 00 00 00 00 00 00	0.000	3173	Tx
488	1	8	11 10 47 01 42 00 00 03	163.075	4199	Rx
80	1	8	00 00 00 00 00 00 00 00	0.000	3174	Tx

Tabell 23 Start av synk meddelande 80h.

Efter att man har satt igång synk processen sänder man ett meddelande om begäran att få övergå i operations läge operational (tabell 24). Meddelandet man sänder kommer att ha

följande betydelse: COB-id 0, byte 0 berättar att det är operationsläge operational man vill komma åt och byte 1 är vilken node man vill ända operationsstatus på.

0	1	2	01 08	0.000	4007	Tx	NMT request set node: 0x8 Operational
80	1	8	00 00 00 00 00 00 00 00	0.000	7564	Tx	
188	1	8	00 00 00 c4 00 00 00 00	250.893	3592	Rx	
488	1	8	11 10 47 01 f8 00 00 03	250.899	6966	Rx	
0	1	2	01 08	0.000	4008	Tx	NMT request set node: 0x8 Operational
80	1	8	00 00 00 00 00 00 00 00	0.000	7565	Tx	
188	1	8	00 00 00 44 00 00 00 00	250.913	3593	Rx	

Tabell 24 Begäran om att få övergå till operations läge operational.

Efter att man har gjort en begäran att få övergå till operationsläge operational, kommer byte tre att växla mellan 44h och C4h för PDO1 med COB-ID 188. Konverterar man om detta till binärt format kommer man att se att bit sju växlar mellan noll och ett. Byte tre och bit sju kan man se i bilaga G.3 motsvarar en ”stuffing” bit, vilket är en bit som används för handskakning mellan slav och master. För att få en fullständig handskakning mellan motorkontroller och PLC måste man programmässigt göra en funktion som svarar med motsatta värden på RPDO1 byte tre bit sju. Men för denna simulation har man stängd av stuffing kontrollen i motorcontrollerenheten med hjälp av ZP CanConsole.

188	1	8	00 00 00 44 d7 00 00 00	261.649	4281	Rx
80	1	0		0.000	6429	Tx
188	1	8	00 00 00 c4 d7 00 00 00	261.669	4282	Rx
80	1	0		0.000	6430	Tx
188	1	8	00 00 00 44 d7 00 00 00	261.689	4283	Rx
80	1	0		0.000	6431	Tx
188	1	8	00 00 00 c4 d7 00 00 00	261.708	4284	Rx
80	1	0		0.000	6432	Tx
188	1	8	00 00 00 44 d7 00 00 00	261.728	4285	Rx
80	1	0		0.000	6433	Tx
188	1	8	00 00 00 c4 d7 00 00 00	261.748	4286	Rx

Tabell 25 Motorkontrollern har övergått till Operational läge.

Genom att använda sig av RPDO1 COB-ID 208h har man tillgång till all prosessadata för att kunna styra motorn. För att kunna styra motorn behöver man ange en motorhastighet, rotations riktning och några andra kontroller specifika parametrar. Uppbyggnaden av processdata meddelandet gjordes enligt följande:

Rotations hastighet anges av byte 0-1 i enheten Hz x 100. Önskad hastighet för simulationen kommer att vara 50Hz.

$$50Hz * 100 = 5000mHz \;\; 5000mHz \rightarrow 1388h$$

För att förse motorn med ström använder man en kontaktor som styrs av byte 2 bit 1 och för att motorkontrollern skall vara kontrollerbar behöver man möjliggöra inverterns slutsteg. Detta görs med hjälp av att sätta byte 2 bit 3 till en etta. Sista man behöver göra är att ange rotations riktning vilket anges via byte 2 bit 6. För att se de specifika bitarnas funktion se bilaga G.3.

Byte 2:

Byte 2 bit 0	Byte 2 bit 1	Byte 2 bit 2	Byte 2 bit 3	Byte 2 bit 4	Byte 2 bit 5	Byte 2 bit 6	Byte 2 bit 7	HEX
0	1	0	1	0	0	1	0	4A

Byte 3:

Byte 3 bit 0	Byte 3 bit 1	Byte 3 bit 2	Byte 3 bit 3	Byte 3 bit 4	Byte 3 bit 5	Byte 3 bit 6	Byte 3 bit 7	HEX
0	0	0	0	0	0	0	0	0

PDO CAN meddelande:

COB-ID	Byte0 MSB	Byte1 LSB	Byte2 MSB	Byte3 LSB	Byte 4	Byte5	Byte 6
208	88	13	4A	00	00	00	00

Nedan I tabell 26 använder man sig av det meddelande som man har konstruerat ovan. TPDO1 kommer i detta fall fungera som en återkoppling av RPDO1.

208	1	8	88 13 4a 00 00 00 00 00	0.000	1879	Tx
80	1	8	00 00 00 00 00 00 00 00	0.000	10054	Tx
188	1	8	00 00 02 c4 00 00 00 00	300.693	6082	Rx
208	1	8	88 13 4a 00 00 00 00 00	0.000	1880	Tx
488	1	8	0d 0b 31 00 26 4b 00 01	300.699	8535	Rx
80	1	8	00 00 00 00 00 00 00 00	0.000	10055	Tx

Tabell 26 Kör motorn med hastigheten 50Hz.

En funktion som man också är i behov av i projektet är att kunna ändra motor parametrar. För att veta hur man skall gå till väga gör man först en simulation över hur processen går till för att ändra en parameter med CANtrace. För att läsa service data objekt från motorcontrollern kommer man att använda sig av COB-ID 608h, byte 0 i meddelandet kommer att vara 40h, eftersom att man vill läsa en parameter. Parametern man kommer att läsa heter ”par_main_contactor_volt” och har objekt indexet 20A8h. Uppbyggnaden av CAN meddelandet gjordes enligt följande:

Read SDO CAN meddelande:

COB-ID	Byte0	Byte1 MSB	Byte2 LSB	Byte3	Byte 4 MSB	Byte5	Byte 6	Byte 7 LSB
608	40	A8	20	00	00	00	00	00

Nedan i tabell 27 kan man se när man läser SDO:n och får ett svar på COB-ID 588h. Parametern man läser har värdet 64h vilket motsvarar 100 decimalt. Genom att använda sig av CAN protokollet i bilaga G.2 kan man se att detta värde ges i procent.

208	1	8	00 00 00 80 00 00 00 00	2.228	112	Rx	
188	1	8	00 00 00 c4 00 00 00 00	2.231	112	Rx	
608	1	8	40 a8 20 00 00 00 00 00	2.234	14	Rx	Node: 0x8, SDO upload request [0x20a8,0]
588	1	8	42 a8 20 00 64 00 00 00	2.235	14	Rx	Node: 0x8, SDO upload response [0x20a8,0] Value: 0x64
80	1	0		2.247	113	Rx	
208	1	8	00 00 00 00 00 00 00 00	2.248	113	Rx	
488	1	8	11 10 47 01 42 01 00 03	2.249	71	Rx	

Tabell 27 Läser ett objekt index 20A8h från motorcontrollern.

För att kunna ändra samma parameter i motorkontrollern behöver man bygga om CAN meddelandet. I byte 0 kommer man att skriva 22. Detta byter att man väljer att skriva till slaven istället för att läsa. I byte 4 – 7 skriver man in det värde man vill skriva till nätverket. Eftersom att man i detta fall väljer att skriva 63h vilket motsvarar 99 decimalt behöver man endast använda sig av byte 4. Uppbyggnaden av CAN meddelandet gjordes enligt följande:

Write SDO CAN meddelande:

COB-ID	Byte0	Byte1 MSB	Byte2 LSB	Byte3	Byte 4 MSB	Byte5	Byte 6	Byte 7 LSB
608	22	A8	20	00	63	00	00	00

På samma sätt som när man läser en parameter får man tillbaka ett respons meddelande på COB-ID 588. I detta fall meddelas endast till vilket objekt index och sub index man har skrivit meddelandet (se tabell 28).

608	1	8	22 a8 20 00 63 00 00 00	0.000	3446	Tx	Node: 0x8, SDO download request [0x20a8,0] Value: 0x63
588	1	8	60 a8 20 00 00 00 00 00	1,617.043	3424	Rx	Node: 0x8, SDO download response [0x20a8,0]
288	1	8	42 ff 00 00 17 00 00 00	1,617.045	753	Rx	
488	1	8	11 10 47 01 42 00 00 03	1,617.069	7914	Rx	
608	1	8	22 a8 20 00 63 00 00 00	0.000	3447	Tx	Node: 0x8, SDO download request [0x20a8,0] Value: 0x63
588	1	8	60 a8 20 00 00 00 00 00	1,617.095	3425	Rx	Node: 0x8, SDO download response [0x20a8,0]

Tabell 28 Skriver till objekt index 20A8h i motorcontrollern.

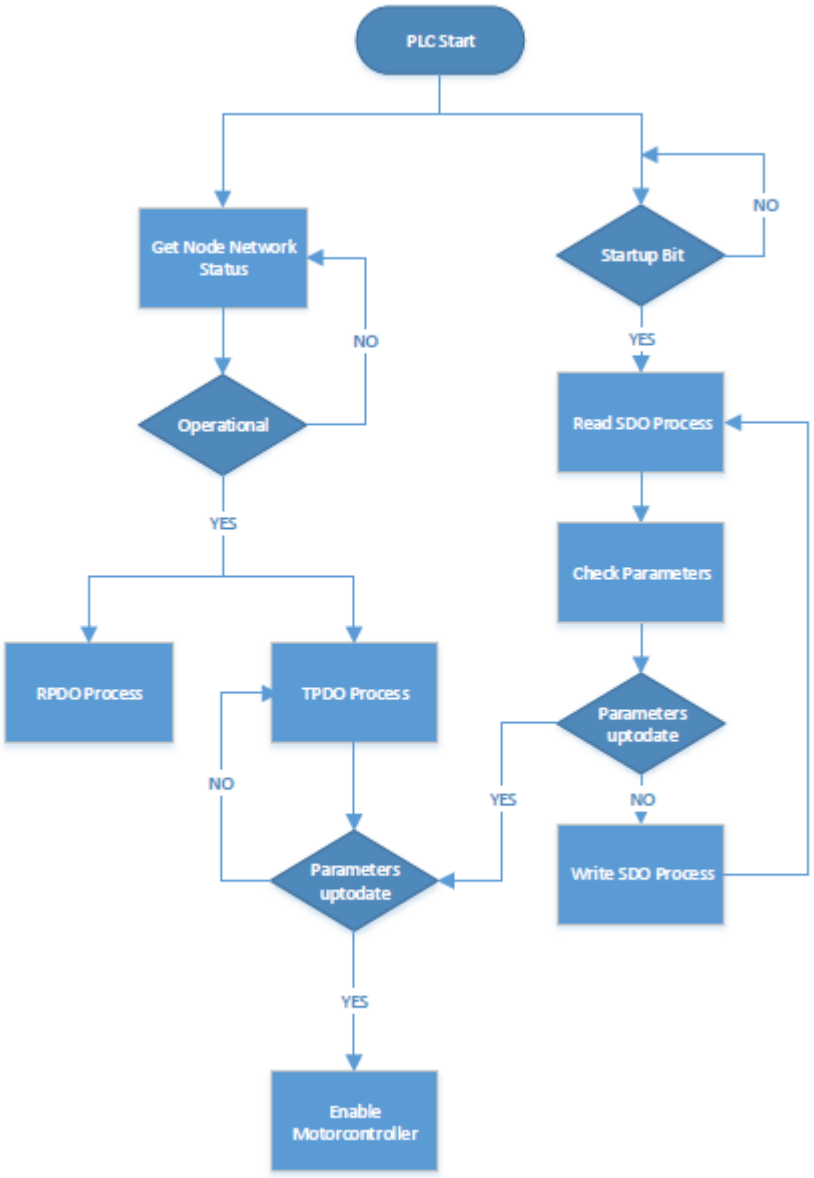
Slutligen för att parametrar som man skriver till motorcontrollern skall sparas permanent bör man skriva till hårdvaran att parametrarna skall sparas i EEPROM. Undviker man att göra detta kommer parametrarna att vara opåverkade när man gör en omstart av slaven. Genom att skriva värdet 65766173h till objekt index 1010h kommer man att spara alla parametrar till EEPROM. Konverterar man om värdet 65766173h till ASCII kommer detta värde att stå för ”SAVE”. Uppbyggnaden av detta meddelande beskrivs nedan:

Write SDO CAN meddelande:

COB-ID	Byte0	Byte1 MSB	Byte2 LSB	Byte3	Byte 4 MSB	Byte5	Byte 6	Byte 7 LSB
608	22	10	10	00	73	61	76	65

När man hade simulerat CAN trafiken med hjälp av CANtrace började man skriva kod för att kunna styra motorkontrollern via logiken. Det som man tog fram var ett funktionsblock som innehåller ett antal funktioner. Funktionerna kommer man att beskriva enskilt mera i detalj med hjälp av rubrikerna nedan.

Nedan i flödesdiagram 1 kan man se hur hela funktionsblocket är uppbyggt.



Flödesdiagram 1 Funktion för funktionsblocket.

E.2 Read-SDO funktion

För att enkelt kunna köra en specifik funktion åt gången använder man sig av en funktions uppbyggnad av olika steg. Varje steg har en specifik uppgift och ändats ett steg kan vara aktiverad åt gången. Medhjälp av att använda denna metod kan man hoppa mellan olika steg och vara säker på att stegen kommer att utföras i rätt ordning, beroende på olika villkor. För att ändats kunna läsa parametrar från en CANopen nod kan man göra denna funktion mycket simpelt. Men pga. av att man vill ha full kontroll över allting som händer på CAN-nätverket behöver man ha vissa kontroller. Denna funktion är huvudsakligen uppbyggd av fyra olika huvudfunktioner som i sin tur består av flera olika steg. Första huvudfunktionen används för att skriva till nätverket att man vill läsa en parameter, samt vilken parameter man vill läsa. Detta görs med en funktion vid namnet WRREC vilket är ett inbyggt funktionsblock i TIA portalen. Parametrarna anger man på objekt index nivå i en Array som funktionen kommer att använda sig av för att bygga upp ett CAN meddelande. Konverteringen mellan hexadecimala objekt index värden från datablocket till datatypen Array kommer att ske sekvent för varje parameter man har definierat i datablocket. För att WRREC funktionen sedan skall veta vart man skall skriva det uppbyggda meddelande använder man sig av hårdvaru-ID och kommunikations kanal index. Varje modul man använder i ett Siemens system har ett eget specifikt hårdvaru-ID och kanal index. ID:t och kommunikationskanalen man använder bör definieras för att användas för CANopen mastern, vilket i denna tillämpning är CANopen kommunikationsmodulen. För att bakgrundsprogrammet skall veta om man vill skriva eller läsa från nätverket bör man definiera detta med hjälp av INDEX variabeln. CAN meddelandet man sänder kommer automatiskt att byggas upp av bakgrundsprogrammet för kommunikationsmodulen, med hjälp av de parametrar man har definierat. Kod 1 beskriver definitionen av WRREC funktionsblocket på variabel nivå. Variablerna definieras i sin tur i tidigare steg före den nämnda funktionen.

```
110:    //Write

    #Read_Write_SDO_WRREC(
                        REQ := #REQWrite_ReadSDO,                   // REQ = 1: Transfer data record
                        ID := #HW_ID,                               // ID number of the hardware component
                        INDEX := #RecordNo,                         // Data record number
                        LEN := #RequestTotalSize,                   // Maximum length of the data record to be transferred in bytes
                        DONE => #ReadWriteWriteSDODone,             // Data record was transferred
                        BUSY => #Write_ReadSDOBusy,                 // BUSY = 1: The writing process is not yet complete
                        ERROR => #Read_WriteWriteSDOError,          // ERROR = 1: An error occurred during the writing process
                        STATUS => #Status,                          // Block status or error information
                        RECORD := #abReadSDORequestArray            // Data record
    );

    #REQWrite_ReadSDO := FALSE;                                     // Reset variable to prevent this SFC from being called twice

    IF NOT #Write_ReadSDOBusy THEN                                  // Wait on SDO_WRREC instruction to be done
        #StateRead := 120;                                          // Read SDO function state 120 (Evaluate writing)
    END_IF;
```

Kod 1 WRREC funktionen för ReadSDO.

Andra huvudfunktionen i programsekvensen har till uppgift att läsa in svaret man får när man skrivit ut en läs begäran. För att läsa in parameter ÄR värdet använder man sig av en liknade funktion som WRREC. Men istället för att skriva en begäran läser man ett svar. Denna funktion heter RDREC och används i samma utförande som WRREC funktionen. Funktionens definitioner kan man se kod 2. Svaret man kommer att få från CANopen-nätverket kommer i förstahand sparas i en Array. I samma utförande som man i föregående funktion har en automatisk konvertering av DB till Array, kommer konverteringen här att utföras i motsvarande riktning, Array till DB. Eftersom CANopen är baserad på att använda hexadecimala tal kommer alla värden som man läser in i datablocket vara av data typen Word.

```
130:  //Read

    #Read_Write_SDO_RDREC(
                          REQ := #REQReadSDOSeq,                    // REQ = 1: Transfer data record
                          ID := #HW_ID,                             // ID number of the hardware component
                          INDEX := #RecordNo,                       // Data record number
                          MLEN := #ResponseTotalSize,               // Maximum length in bytes of the data record information to be read
                          VALID => #ReadReadReadSDOValid,           // New data record was received and is valid
                          BUSY => #Read_ReadSDOBusy,                // BUSY = 1: The reading process is not yet complete
                          ERROR => #Read_ReadReadSDOError,          // ERROR = 1: An error occurred during the reading process
                          STATUS => #Status,                        // Block status or error information
                          LEN => #LEN_SDO,                          // Length of the read data record information
                          RECORD := #abReadSDOResponseArray         // Destination area for the data record read
    );
```

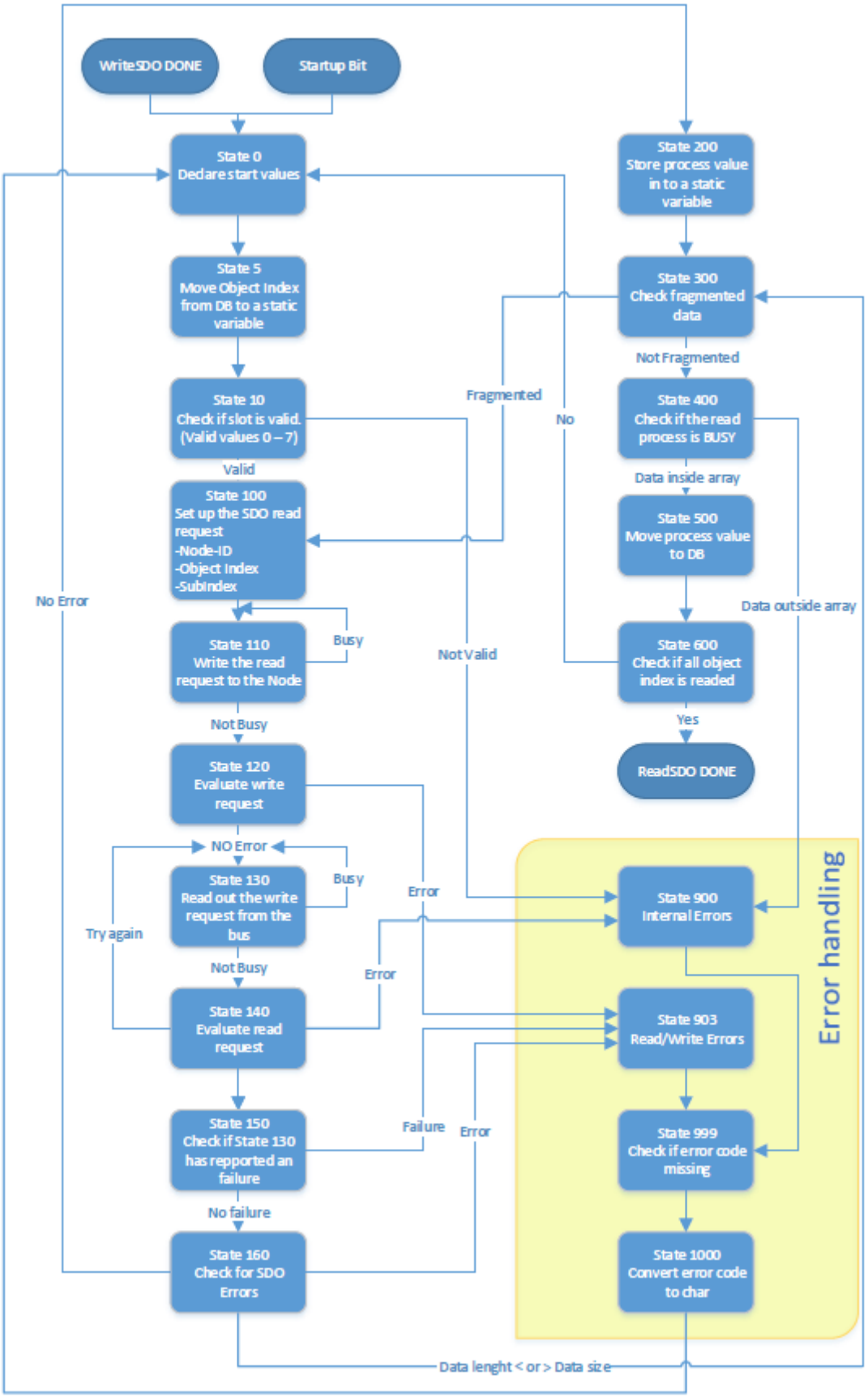
Kod 2 RDREC funktionen för ReadSDO.

ReadSDO funktionens tredje huvudfunktion håller funktionen uppdaterad med hur många parametrar som har blivit behandlade. För att hålla reda på om alla parametrar har blivit överförda till dataregistret, jämför man varje pendlande objekt index med 0000h. När en läsbegäran skrivs med objekt index 0000h kommer ReadSDO funktionen att avslutas. Orsaken till att man jämför med 0000h är att alla odefinierade objekt index i datablocket, som anger parameter typ kommer att ha värdet noll. Uppbyggnaden av denna specifika funktion kan ses i kod 3.

```
600:  // Check if all parameters are readed

    #OffsetRead := #OffsetRead + 2;                                 // Increase offset with 2

    #ObjectIndexTemp := PEEK_WORD(area := 16#84,                    // 16#84 = Read from DB
                                  dbNumber := #DB_Number,           // DBnumber on SDOParameterDB
                                  byteOffset := #OffsetRead);       // Offset adress to read from. byteoffset increases with 2 every cycle


    IF #ObjectIndexTemp = 16#0000 THEN                              // Check if all parameters are writen and readed, if TRUE jump to END
        #OffsetRead := 0;
        #RET_ReadSDOChar := 'No errors';
        #BUSY_Read := FALSE;
        #ReadSDOFuncDone := TRUE;                                   // Read function is done,
        #StateRead := 0;                                            // Jump to begining (Declare variables)
        #StateUptodate := 100;                                      // 000000000000 (Compare process and setvalue SDO)
    ELSE
        #ReadSDOCycleDone := TRUE;                                  // Read function is done, and start a new read cycle
        #StateRead := 0;                                            // Jump to begining (Declare variables)
    END_IF;
```

Kod 3 Avslutnings funktion för ReadSDO.

Mellan huvudfunktionerna ett, två och tre har man implementerat olika typer av kontroller av felmeddelande. Felmeddelande kontrollen håller koll på att all data som används är intakt. Kontrollen man har implementerat före WRREC funktionen håller t.ex. koll på att man skriver till en kommunikationskanal som finns i systemet och att meddelandet befinner sig inom ramen som datamängden tillåter. Efter att man har fått ett svar från CANopen-nätverket kommer också detta meddelande gå igenom en kontroll för att säkerställa att meddelandet innehåller allt som CANopen-protokollet kräver. Ifall ett felmeddelande påträffas kommer program sekvensen att avbrytas och övergå till en felmeddelandehantering. Denna felmeddelandehantering kontrollerar vad som har gått fel i processen. Beroende på vad som har gått fel i ReadSDO cykeln kommer felmeddelandehanterings process överväga om man skall starta läs cykeln från ett specifikt steg i processen, från början eller om man skall avsluta processen och skriva ut ett felmeddelande. Felmeddelanden från felhanteringsprocessen kommer att skrivas ut som en text string i det kompilerade funktionsblockets datablock. Som output från funktionslocket kommer man endast få felmeddelanden i hexadecimal form. Orsaken till att man skriver ut felmeddelanden i textform i datablocket är för att användaren lätt skall kunna se vad som har gått fel i processen utan att behöva använda sig av någon manual för att söka upp olika felkoders betydelse.



Flödesdiagram 2 Beskrivning av flödet genom Read-SDO funktionen.

E.3 Write-SDO funktion

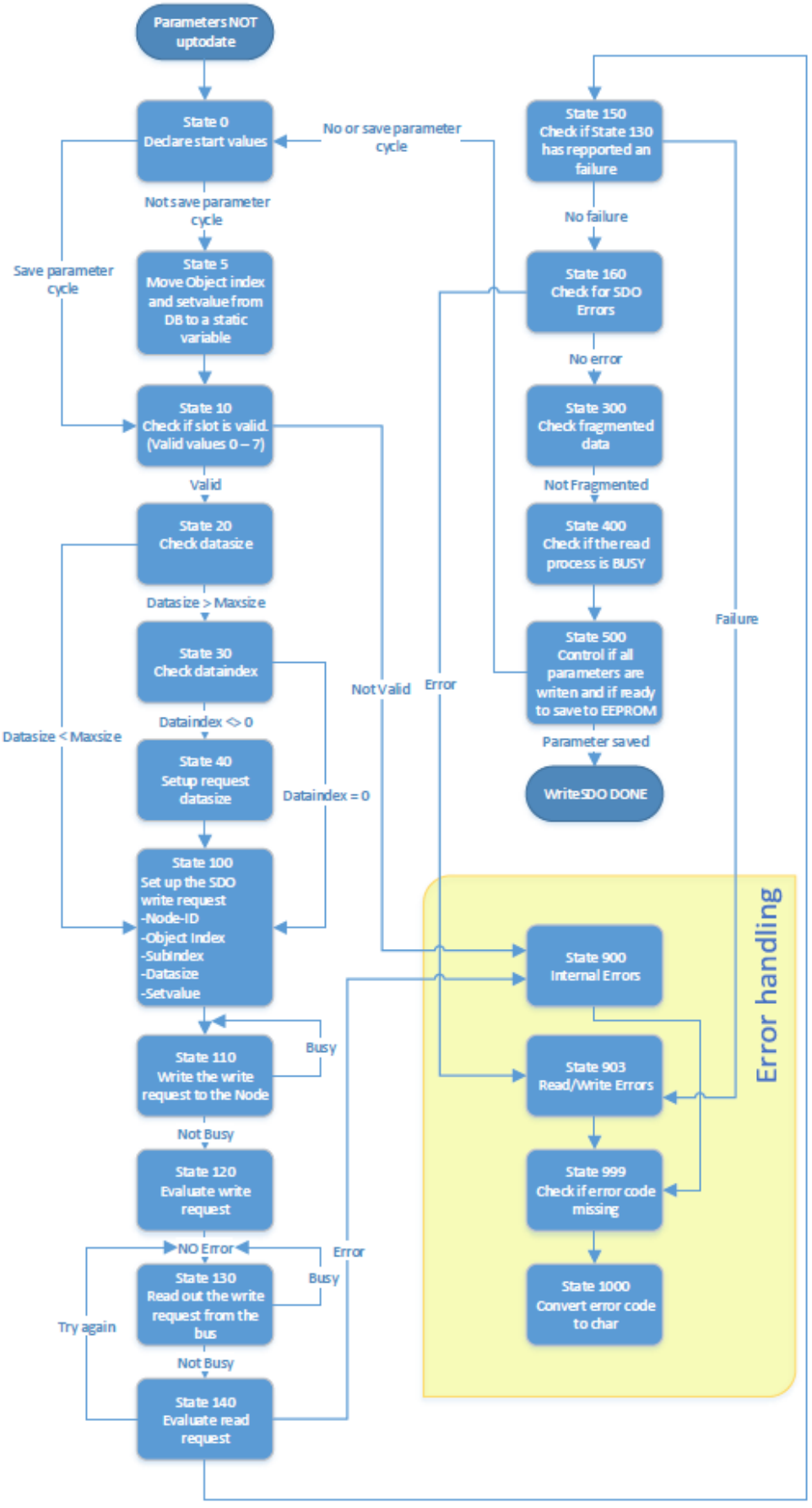
Pga. av att Write-SDO och Read-SDO funktionerna är uppbyggda efter samma princip kommer man inte att beskriva Write-SDO funktionen lika detaljerat som man gjorde med Read-SDO i föregående kapitel. Man kommer ändats att koncentrera sig på det som avviker funktionerna från varandra.

För att CANopen mastern skall veta att man vill skriva till nätverket bör man berätta detta till funktionen genom att använda sig av INDEX variabeln för WRREC och RDREC funktionerna. Vill man använda sig av ett skriv kommando definierar man INDEX variabeln till 256 + Slot-ID och vill man ha läsa från nätverket definieras INDEX variabeln till 272 + Slot-ID. Denna INDEX variabel kommer att ha direkt koppling till byte 0 (*Specifier)* i data ramen för CANopen meddelandet. Som man beskriver i föregående bilaga har man definierat alla BÖR-värden i samma datablock som man har definierat Read-SDO funktionens ÄR och input värden för Objekt index. Detta datablock är en gemensam knutpunkt för hela CANopen kommunikation funktionen.

Funktion sekvensen för överförsel av alla BÖR-värden kommer att ske enligt samma sekvens som för ÄR-värden hos Read-SDO funktionens fyra huvudsteg. Pga. av att Write-SDO funktionen behöver kunna spara parametrar till enhetens EEPROM kommer denna funktion att ha ett femte steg. Detta steg kommer alltid att köras när man har skrivit ut alla BÖR-värden till nätverket och läser in ett objekt index med värdet 0000h. Objekt index värdet 0000h betyder i Read-SDO funktionens tillämpning att funktionen är färdig och kan hoppa över till nästa steg i processen, men i detta fall betyder 0000h att man skall köra en spar cykel. Spar cykeln kommer att bestå av samma meddelande uppbyggnad som man gjorde när man sände manuella CAN-meddelanden medhjälp av CANtrace. Uppbyggnaden av detta CAN meddelande har man beskrivit i början av bilaga E. Som man kan läsa ur den nämnda bilagan behöver man skriva värdet 65766173h till objekt index 1010h för att alla parametrar skall sparas fullständigt i enhetens EEPROM. Sekvensen för denna cykel har man beskrivit i kod 4 med hjälp av några rader strukturerad kod. Med hjälp av koden nedan kan man se att när man får ett svar att objekt index har värdet 0000h kommer man att skriva 1010h till nästa cykels objekt index samt 65766173h för bör värde. Efter att man har kört spar cykeln kommer man att avsluta Write-SDO funktionen genom att hoppa till steg noll i sekvensen och starta Read-SDO funktionen för att säkerställa att alla BÖR och ÄR-värden stämmer överens.

```
500:  // Save and control parameters

    #OffsetWrite := #OffsetWrite + 2;                                   // Increase offset with 2

    #ObjectIndexTemp := PEEK_WORD(area := 16#84,                        // 16#84 = Read from DB
                                  dbNumber := #DB_Number,               // DBnumber on SDOParameterDB
                                  byteOffset := #OffsetWrite);          // Offset adress to read from. byteoffset increases with 2 every cycle

    IF #ObjectIndexTemp = 16#0000 AND NOT #SaveParameterCycle THEN      // Check if parameters are ready to be saved
        #ObjectIndex := 16#1010;                                        // Object index for SAVE comand (depend on slave hardware)
        #SetValue := 16#65766173;                                       // Parameter to save (depend on slave hardware)
        #OffsetWrite := #OffsetWrite + 2;                               // Increase di_iOffsetWrite with 2, this is the second last cycle in the process
        #WriteSDOCycleDone := FALSE;                                    // Write cycle is not yet done
        #SaveParameterCycle := TRUE;                                    // Run the save parameter cycle
        #StateWrite := 0;                                               // Jump to begining (Declare variables)
    ELSIF #SaveParameterCycle THEN                                      // Check if save parameter cycle is done
        #SaveParameterCycle := FALSE;                                   // Save parameter cycle is done
        #BUSY_Write := FALSE;                                           // Not busy
        #OffsetWrite := 0;                                              // Reset offset to 0
        #RET_WriteSDOChar := 'No errors';                               // No errors
        #ControlCycle := TRUE;                                          // Run a control cycle to ensure every thing is ok
        #WriteSDOFuncDone := TRUE;                                      // Write function is done
        #StateWrite := 0;                                               // Jump to begining (Declare variables)
    ELSE
        #WriteSDOCycleDone := TRUE;                                     // Write cycle is done
        #StateWrite := 0;                                               // Jump to begining (Declare variables)
    END_IF;
```

Kod 4 Write-SDO spara till EEPROM function.



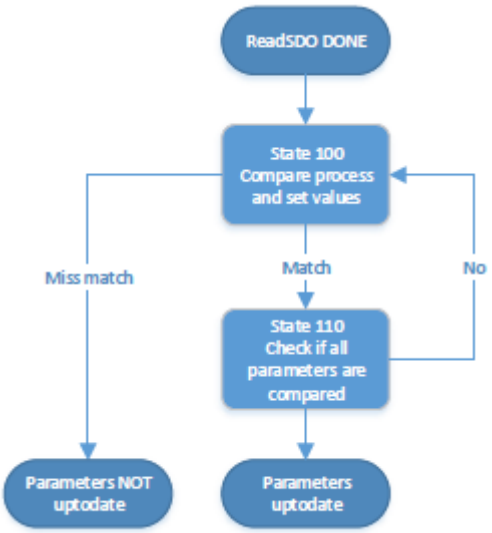
Flödesdiagram 3 Beskrivning av flödet genom Write-SDO funktionen.

E.4 Get node up-to-date status funktion

För att kunna jämför parametrar läser man in ÄR-och BÖR-värden från datablocket som alla parametrar finns lokaliserade i med hjälp av ett PEEK_WORD kommando. Detta kommando är en inbyggd funktion i TIA portalen som gör det möjligt att kunna läsa från datablock. Enligt exemplet nedan har man byggt upp en funktion som jämför BÖR och ÄR värdet för varje parameter definierat i datablocket.

Jämförelse Exempel:

```
IF Börvärde <> Ärvärde THEN
        InteUpdaterad := SANT;
        Updataerad := FALSK;
ELSE
        Updataerad := SANT;
        InteUpdaterad := FALSK;
End_IF;
```

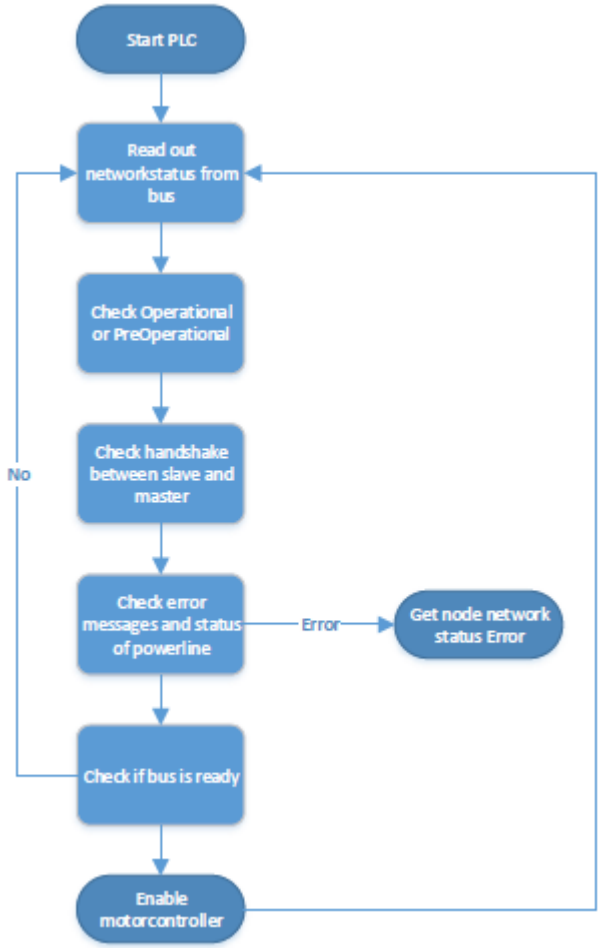


Flödesdiagram 4 Beskrivning av flödet genom Get node up-to-date funktionen.

E.5 Get node network status

Denna funktion använder sig av den inbyggda funktionen RDREC, vilket alla andra funktioner som läser CAN trafik använder sig av. Variabeln INDEX för denna tillämpning kommer att vara 153, INDEX för denna tillämpning är definierad för att läsa CAN status. Förutom att denna funktion hanterar operations statusar loggar den också felmeddelanden på bussen. Felhanteringen som man använder sig av hos Write och Read-SDO hanterar enbart

felmeddelanden hos dataöverföring. Felmeddelandehanteringen i denna process hanterar enbart kommunikation.



Flödesdiagram 5 Beskrivning av flödet genom Get node network status funktionen.

E.6 RPDO processfunktion

RPDO funktionen består av två stycken funktioner som är inbakade i en funktion. Första delen av funktionen är en handskaknings funktion som håller kommunikationen uppe mellan mastern och slaven. Bryts denna handskakning kommer alla aktiva processer i motorstyrningen att avbrytas och felmeddelande rapporteras till PLC:n. Denna funktion är uppbyggd av en skild stuffingbit som fungerar som ett sync meddelande. Detta sync meddelande måste sändas varje 20ms växlande mellan noll och ett. För att veta när man kan starta syncningen använder man sig av en sync som sänds av motorkontrollern som ett TPDO meddelande. När man har mottagit första TPDO sync meddelandet kan man börja handskakningen med hjälp av RPDO stuffingbiten. Genom att använda sig av denna metod kommer RPDO och TPDO-meddelanden alltid att vara synkroniserade. Nedan i kod 5 har man beskrivit hur man genererar en bit som växlar mellan noll och ett beroende på när funktionen blir triggad.

```
// Stuffing bit switch between 1 and 0
IF #Stuffing_RX_R_TRIG.Q THEN
    IF #Stuffing_RX THEN
        #Stuffing_RX := False;
    ELSIF NOT #Stuffing_RX THEN
        #Stuffing_RX := TRUE;
    END_IF;
END_IF;
```

Kod 5 Handskaknings funktion.

Andra delen av funktionen består av själva styrkommandot för hanteringen av invertern. Med hjälp av denna funktion kan man styra motorn att rotera medurs och moturs med olika hastigheter samt accelerationer. För att styra motorn använder man sig av RPDO1. Denna PDO är uppbyggd av olika komponenter som man kan använda sig av när man vill kontrollera motorn. För att se vad denna PDO innehåller se bilaga G.3.

För att kunna skriva ut data till invertern använder man sig av samma inbyggda funktion WRREC som man använde för att skriva SDO meddelanden till motorkontrollern. I denna PDO tillämpning använder man sig av en liknande array som man använde sig av i SDO tillämpningen för att sända data ut på nätverket, med hjälp av WRREC funktionens RECORD variabel, kod 6.

```
// Send a array of input data to the Slave
#RPDO(REQ := NOT #BUSY_RPDO,
      ID := #HW_ID,
      INDEX := #INDEX_RPDO,
      LEN := #MLEN_IN_OUT_PDO,
      DONE => #DONE_RPDO,
      BUSY => #BUSY_RPDO,
      ERROR => #RPDO_Error,
      STATUS => #STATUS_RPDO,
      RECORD := #abData_Out);
```

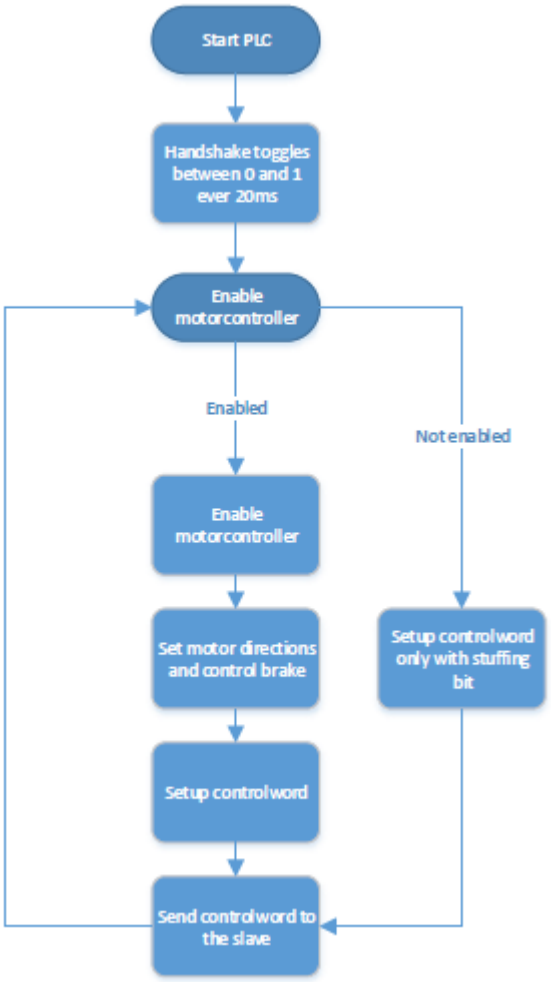
Kod 6 WRREC för RPDO.

Denna Array kommer att innehålla all data man behöver för att kunna styra motorn. Till skillnad från SDO data överförsen är denna PDO data behandling mer komplicerad. När man sänder SDO data läster man endast in data från ett hårdkodat datablock vilket man sedan sänder rakt ut på nätverket. I detta fall går man inte direkt på hårdkodad data, utan man behöver behandla data i några steg före man sänder ut den på nätverket. RPDO1 byte 0,1,4 och 5 är data som man använder som input till funktionen, som man rent hårdkodat kan läsa

av, beroende på vilken applikation systemet används i. Börvärdet för motorhastigheten vilket är byte 0 och 1 har man multiplicerat med 100 för att kunna ange inputvärdet i Hz, samt konverterat om från integer till byte för att kunna ge ett börvärde i decimalform. Samma sak gäller för byte sex och sju men här har man multiplicerat input värdet med tio för att få en acceleration samt deacceleration i Hz. För att kunna ge rotationsriktning samt kunna kontrollera diverse kontaktor utgångar använder man sig av RPDO:ns skilda controlword. Detta controlword består av 16 bitar vars varje bit har en specifik funktion. För att kunna styra varje bit var för sig har man maskat ut varje bit i det 16bitars långa controlwordet. Som namnet säger är det frågan om data typen word, vilket har storleken av två byte. Nedan i kod 7 beskriver man hur man har byggd upp RPDO1 meddelandet, samt hur man maskar ut varje enskild bit ur controlmeddelandet.

```
// Define a array of input data to be sent to the Slave
#abData_Out[0] := #Pedal_Braking;
#abData_Out[1] := #Torq_Setpoint;
#abData_Out[2] := INT_TO_BYTE(#Target_Speed * 100);
#abData_Out[3] := INT_TO_BYTE(SHR(IN := #Target_Speed * 100, N := 8));

#Masked_Controlword := BOOL_TO_WORD((#Controlword))
                       OR SHL(IN := BOOL_TO_WORD(#Close_out_A16), N := 1)
                       OR SHL(IN := BOOL_TO_WORD(#Close_out_A18), N := 2)
                       OR SHL(IN := BOOL_TO_WORD(#Power_line_present), N := 3)
                       OR SHL(IN := BOOL_TO_WORD(#Fault_reset), N := 4)
                       OR SHL(IN := BOOL_TO_WORD(#Out_A18_power_on), N := 5)
                       OR SHL(IN := BOOL_TO_WORD(#Forward_Flag), N := 6)
                       OR SHL(IN := BOOL_TO_WORD(#Reverse_Flag), N := 7)
                       OR SHL(IN := BOOL_TO_WORD(#Hydro_Flag), N := 8)
                       OR SHL(IN := BOOL_TO_WORD(#Safety_present), N := 9)
                       OR SHL(IN := BOOL_TO_WORD(#Free), N := 10)
                       OR SHL(IN := BOOL_TO_WORD(#Braking_type_bit1), N := 11)
                       OR SHL(IN := BOOL_TO_WORD(#Braking_type_bit2), N := 12)
                       OR SHL(IN := BOOL_TO_WORD(#Free_1), N := 13)
                       OR SHL(IN := BOOL_TO_WORD(#Free_2), N := 14)
                       OR SHL(IN := BOOL_TO_WORD(#Stuffing_RX), N := 15);

#abData_Out[4] := WORD_TO_BYTE(#Masked_Controlword);
#abData_Out[5] := WORD_TO_BYTE(SHR(IN := #Masked_Controlword, N := 8));
#abData_Out[6] := INT_TO_BYTE(#"Acc/Decc_ramp" * 10);
#abData_Out[7] := INT_TO_BYTE(SHR(IN := #"Acc/Decc_ramp" * 10, N := 8));
```

Kod 7 Uppbyggnad av RPDO1-kontrollmeddelande.



Flödesdiagram 6 Beskrivning av flödet genom RPDO processfunktionen.

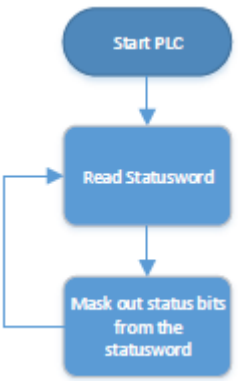
E.7 TPDO processfunktion

Denna motorkontroller använder sig även av TPDO1 och TPDO2 för att leverera realtidsinformation. Dessa PDO paket fungerar som utgångar från invertern. TPDO1 fungerar som en återkoppling av RPDO1 vilket man kan se i bilaga G.3. RPDO1 använder sig av ett kontrollmeddelande controlword medan TPDO1 använder sig av ett status meddelande statusword. T.ex. ger man ett kontrollmeddelande att man vill sluta kontaktor utgång A16 kommer man att se i statusmeddelandet att utgång A16 har öppnats. Om man fortfarande ger kommandot att sluta utgång A16 men utgången inte öppnas kommer man att få ett felmeddelande till logiken. TPDO2 används för att meddela eventuella alarm och temperaturer i invertern och motorn. I denna konfiguration använder man inte TPDO2 som en PDO utan istället läser man den via SDO. Detta pga. av att man alltid vill kunna läsa felmeddelanden och temperaturer oberoende vilket operationsläge invertern kommer befinna sig i. T.ex. om invertern meddelar ett fel och övergår i preoperational läge kommer man aldrig kunna läsa ut felkoder med hjälp av PDO, men med SDO klarar man av att göra det.

För att kunna läsa TPDO från nätverket använder man sig av den inbyggda RDREC funktionen som man också gjorde för att läsa SDO. Uppbyggnaden av funktionen är uppbyggd på samma sätt som för RPDO. Men denna funktion kommer alltid att loopa på i bakgrunden direkt man har tillgång för PDO kommunikation. Uppbyggnaden av statusmeddelandet har man beskrivit i kod 8.

```
// Define output data from readed array from Slave
#Controller_temperature := BYTE_TO_INT(#abData_In[0]);
#Analog3 := #abData_In[1];
#Analog2 := #abData_In[2];
#Analog1 := #abData_In[3];
#Battery_Status := #abData_In[4];
#Actual_Curent := #abData_In[5] / 5;
#Masked_Statusword := BYTE_TO_WORD(#abData_In[6])
                OR SHL(IN := BYTE_TO_WORD(#abData_In[7]), N := 8);

#Statusword := WORD_TO_BOOL(#Masked_Statusword);
                #A16_Status := WORD_TO_BOOL(SHR(IN := #Masked_Statusword, N := 1));
                #A18_Status := WORD_TO_BOOL(SHR(IN := #Masked_Statusword, N := 2));
                #Controller_EN := WORD_TO_BOOL(SHR(IN := #Masked_Statusword, N := 3));
                #A19_Status := WORD_TO_BOOL(SHR(IN := #Masked_Statusword, N := 4));
                #Free_3 := WORD_TO_BOOL(SHR(IN := #Masked_Statusword, N := 5));
                #Free_4 := WORD_TO_BOOL(SHR(IN := #Masked_Statusword, N := 6));
                #Free_5 := WORD_TO_BOOL(SHR(IN := #Masked_Statusword, N := 7));
                #"CNA13(ID1)" := WORD_TO_BOOL(SHR(IN := #Masked_Statusword, N := 8));
                #"CNA5(ID0)" := WORD_TO_BOOL(SHR(IN := #Masked_Statusword, N := 9));
                #CNA6 := WORD_TO_BOOL(SHR(IN := #Masked_Statusword, N := 10));
                #"CNA4(emergency)" := WORD_TO_BOOL(SHR(IN := #Masked_Statusword, N := 11));
                #Free_6 := WORD_TO_BOOL(SHR(IN := #Masked_Statusword, N := 12));
                #Free_7 := WORD_TO_BOOL(SHR(IN := #Masked_Statusword, N := 13));
                #idle_mode := WORD_TO_BOOL(SHR(IN := #Masked_Statusword, N := 14));
                #Stuffing_TX := WORD_TO_BOOL(SHR(IN := #Masked_Statusword, N := 15));
```

Kod 8 Uppbyggnad av TPDO1 status meddelande.

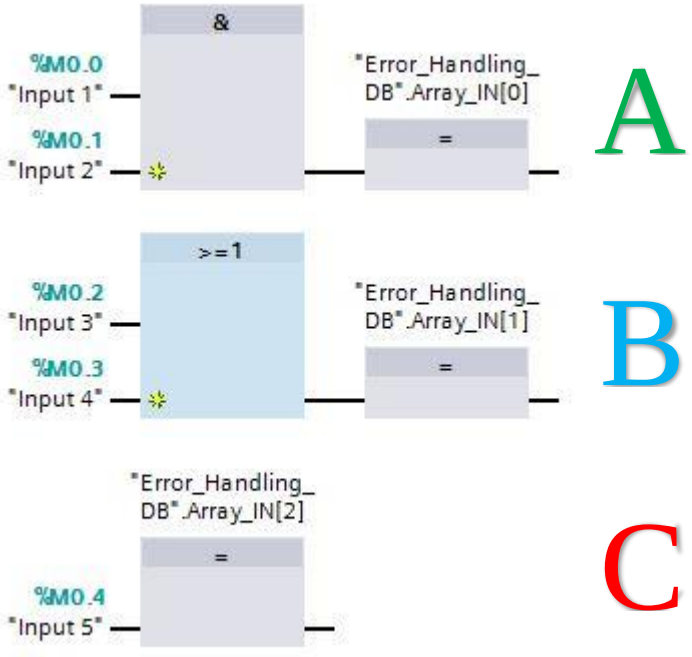


Flödesdiagram 7 Beskrivning av flödet genom TPDO processfunktionen.

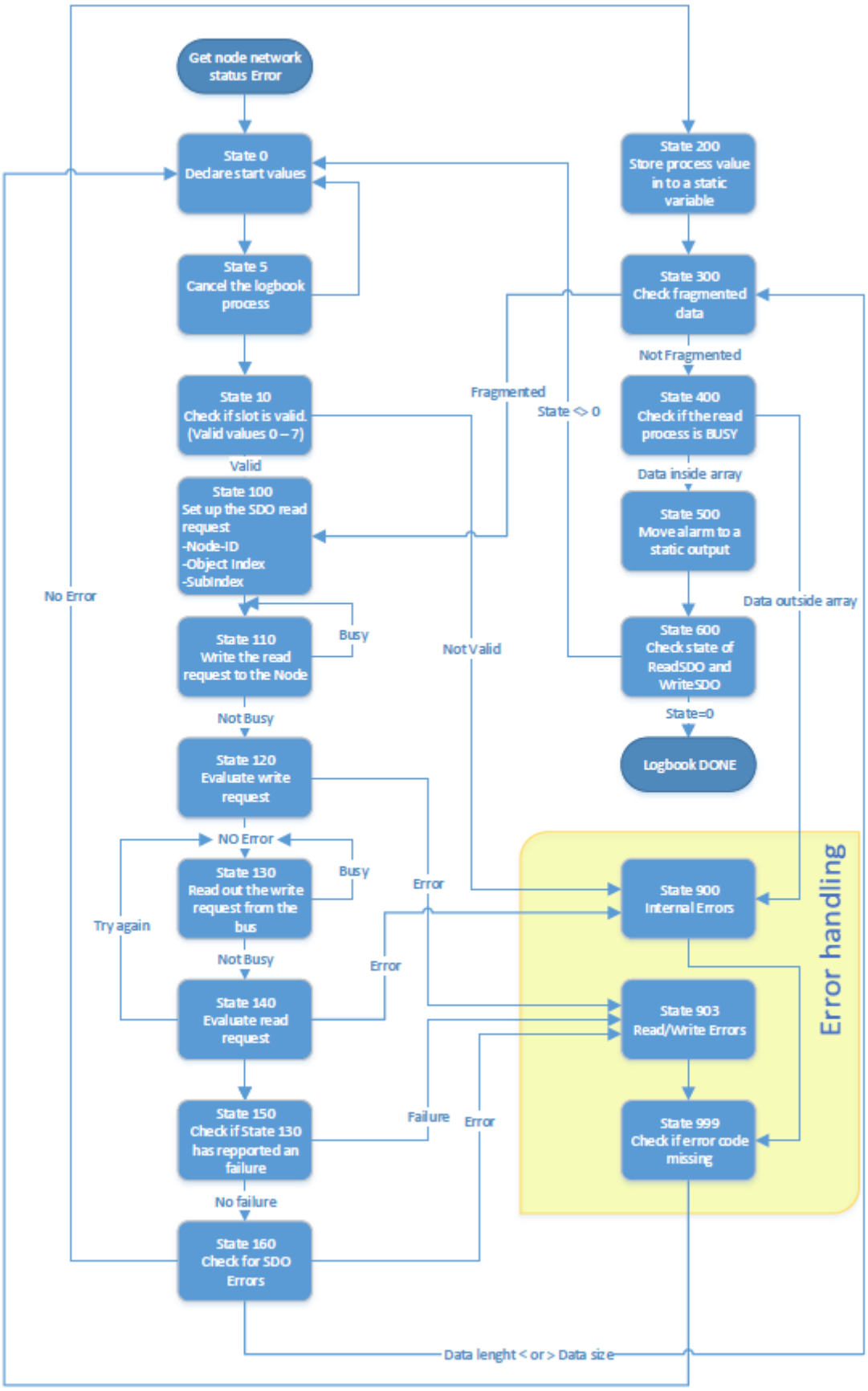
E.8 Alarmhanteringsfunktion

Inputen som behandlar Integer data använder man för funktionsblock eller teknologiobjekt som använder sig av en output som anger ett decimalt nummer som felmeddelande. T.ex. i denna CANopen-tillämpning använder sig CANopen kommunikationsblocket av en utgång som anger ett felmeddelande i decimal form. Varje specifikt decimaltal som kommunikationsblocket genererar som felmeddelande kan man i en konverteringstabell som inverter tillverkaren har specificerat, konvertera om till text och sedan förstå vad felmeddelandet betyder. T.ex. genererar kommunikationsblocket decimala talet 248 som felmeddelandet betyder det genom att titta i konverteringstabbelen ”CAN BUS KO” se bilaga G.5.

Den andra omtalade inputen behandlar Array of Bool data. Denna input kan användas för vanliga PLC program där man inte genererar decimala felmeddelande nummer. Denna typ av felmeddelanden kan användas direkt på funktionsblock som behandlas t.ex. av direkt fysiska in eller utgångar. I tabell 22 har man försökt beskriva hur man kan använda Array of Bool data ingången med hjälp av logiska grindar. I A exemplet kommer felmeddelande noll att bli aktivt när Input 1 och Input 2 är aktiva. När input 1 och 2 är aktiva kan man läsa från utgången ”Error_Nr_Out” decimala talet noll. Exempel B kommer man att få ett felmeddelande när input 3 eller 4 är aktiv, OR grinden kommer att generera felmeddelandet ett. I sista exemplet C kommer man att få felmeddelande nr två när input fem är aktivt. Har man alla ingångar 1-5 aktiva kommer utgången ”Error_Nr_Out” generera noll ett och två med tre sekunders mellanrum. Tre sekunders mellanrum beror på att man har definierat ”HMI_View_Time” till tre sekunder. För att sedan konvertera om bitar till visualiserings text använder man sig av en text lista i HMI konfigurationen. Utgången som använder sig av data typen Array of Integer används för alarmhistorik i HMI-panelen. Använder man sig av exemplet nedan kommer Array noll, ett och två att bestå av decimala talet ett.



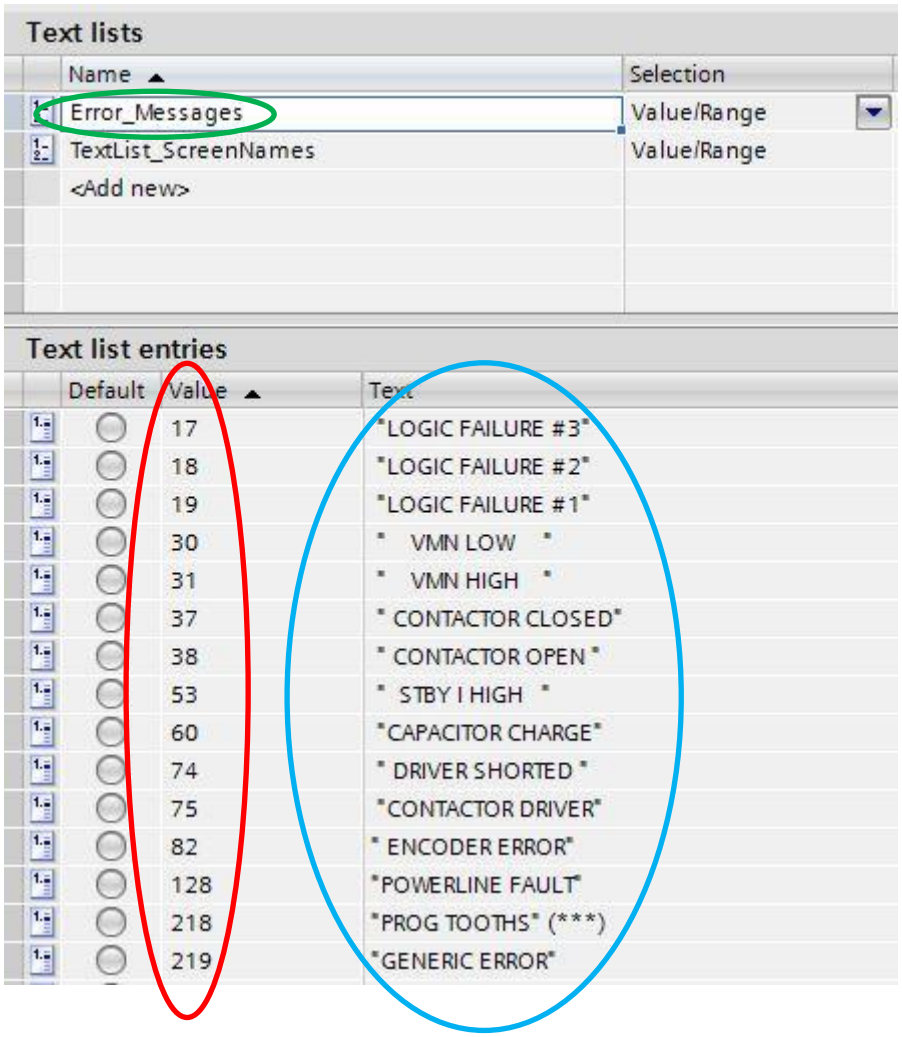
Tabell 22 Array of Bool felmeddelandehantering.



Flödesdiagram 8 Beskrivning av flödet genom alarmhanterings processen i CANopen funktionsblock för styrning av motorkontroller.

E.9 Konfigurering av alarmvisualisering

För att man skall få realtidsalarm att visas i huvudvyn behöver man konfigurera en textlista med alla alarm namn och start värden. Alla alarm namn och start värden kan man se i tabell 23. För att sedan använda denna textlista som en output i HMI-panelens huvud vy behöver man ange ett namn för textlistan som en valbar variabel. Denna textlista heter ”Error_Messages” vilket man kan se i det grön markerade området uppe till vänster i tabell 23. I fönstret ”Text list entries” konfigurerar man sedan sina alarm meddelanden. Varje felmeddelande måste ha ett eget specifikt startnummer, dessa nummer är av decimala tal. För att ett alarm meddelande skall visas i huvud vyn behöver man trigga det nummer som hör i hop med den text man vill visa. T.ex. vill man att ”LOGIC FAILURE #3” skall visas i skärmen behöver man trigga värde nummer 17.



Tabell 23 Konfiguering av textlista för felmeddelanden.

E.10 Konfigurering av alarmhistorik

På liknande sätt som man gjorde en textlista för realtidsalarm bör man göra en alarmlista med diskreta alarm för alarmhistorik. Här använder man sig av Siemens inbyggda funktion för alarm diagnostisering. Till skillnad från realtidsalarmen som triggades medhjälp av ett decimalt nummer, triggas alarmhistoriken med hjälp av en bit i en byte. Det som man behövde ta i beaktan var att alarmen startas vid stigande samt sjunkande flank. För att undvika att samma alarm startades två gånger när en bit blev hög och sedan återvände till sitt normala läge lågt, var att man gjorde en funktion i alarmhanteringsfunktionen som hanterade detta. Alarm som blev startade ändrade läge beroende på vilket läge den hade haft föregående gång när samma alarm varit aktivt, lågt eller högt.

I tabell 24 kan man se hur man har konfigurerat alarmhistoriken. Vill man att felmeddelandet ”LOGIC FAILURE #3” skall visas i alarmhistoriken behöver man sätta ”Error_Handling_DB_Array_Out[17]” bit noll till en etta. Nästa gång man vill att alarmet skall startas kommer man att behöva sätt bit noll till en nolla. Vilken bit man använder för att visualisera ett alarm kan man se under tabellen ”Trigger bit”.

Discrete alarms

ID	Alarm text	Alarm class	Trigger tag	Trigger bit
17	"LOGIC FAILURE #3"	Warnings	"Error_Handling_DB_Array_Out[17]"	0
18	"LOGIC FAILURE #2"	Warnings	"Error_Handling_DB_Array_Out[18]"	0
19	"LOGIC FAILURE #1"	Warnings	"Error_Handling_DB_Array_Out[19]"	0
30	" VMN LOW "	Warnings	"Error_Handling_DB_Array_Out[30]"	0
31	" VMN HIGH "	Warnings	"Error_Handling_DB_Array_Out[31]"	0
37	" CONTACTOR CLOSED"	Warnings	"Error_Handling_DB_Array_Out[37]"	0
38	" CONTACTOR OPEN "	Warnings	"Error_Handling_DB_Array_Out[38]"	0
53	" STBY I HIGH "	Warnings	"Error_Handling_DB_Array_Out[53]"	0
60	"CAPACITOR CHARGE"	Warnings	"Error_Handling_DB_Array_Out[60]"	0
74	" DRIVER SHORTED "	Warnings	"Error_Handling_DB_Array_Out[74]"	0
75	"CONTACTOR DRIVER"	Warnings	"Error_Handling_DB_Array_Out[75]"	0
82	" ENCODER ERROR"	Warnings	"Error_Handling_DB_Array_Out[82]"	0
128	"POWERLINE FAULT"	Warnings	"Error_Handling_DB_Array_Out[128]"	0
218	"PROG TOOTHS" (***)	Warnings	"Error_Handling_DB_Array_Out[218]"	0
219	"GENERIC ERROR"	Warnings	"Error_Handling_DB_Array_Out[219]"	0
223	" WATCHDOG #1 "	Warnings	"Error_Handling_DB_Array_Out[223]"	0
224	" AUX COIL SHORTED"	Warnings	"Error_Handling_DB_Array_Out[224]"	0
227	" WATCHDOG #2 "	Warnings	"Error_Handling_DB_Array_Out[227]"	0
228	"SAFETY INPUT"	Warnings	"Error_Handling_DB_Array_Out[228]"	0

Tabell 24 Konfigurering av alarmhistorik.

Bilaga F Teknisk specifikation

Nedan har man bifogat bilagor för tekniska specifikation av den hårdvara man använder sig av i projektet. Alla bilagor är hämtade direkt av tillverkaren hos komponenterna.

F.1 Siemens S7-1200 Logik

SIEMENS

Datasheet **6ES7215-1AG31-0XB0**

SIMATIC S7-1200, CPU 1215C, COMPACT CPU, DC/DC/DC, 2 PROFINET PORT, ONBOARD I/O: 14 DI 24V DC; 10 DO 24V DC 0.5A 2 AI 0-10V DC, 2 AO 0-20MA DC, POWER SUPPLY: DC 20.4 - 28.8 V DC, PROGRAM/DATA MEMORY: 100 KB

General information	
Engineering with	
• Programming package	STEP 7 V11 SP2 or higher
Display	
with display	No
Supply voltage	
24 V DC	Yes
permissible range, lower limit (DC)	20.4 V
permissible range, upper limit (DC)	28.8 V
Load voltage L+	
• Rated value (DC)	24 V
• permissible range, lower limit (DC)	5 V
• permissible range, upper limit (DC)	250 V
Input current	
Current consumption (rated value)	500 mA; Typical
Current consumption, max.	1 500 A; 24 V DC
Inrush current, max.	12 A; at 28.8 V DC
Output current	

Current output to backplane bus (DC 5 V), max.	1 600 mA; Max. 5 V DC for SM and CM
Power losses	
Power loss, typ.	12 W
Memory	
Type of memory	EEPROM
Usable memory for user data	100 kbyte
Work memory	
• Integrated	100 kbyte
• expandable	No
Load memory	
• Integrated	4 Mbyte
Backup	
• present	Yes; maintenance-free
• without battery	Yes
CPU processing times	
for bit operations, typ.	0.085 µs; / instruction
for word operations, typ.	1.7 µs; / instruction
for floating point arithmetic, typ.	2.5 µs; / instruction
CPU-blocks	
Number of blocks (total)	DBs, FCs, FBs, counters and timers. The maximum number of addressable blocks ranges from 1 to 65535. There is no restriction, the entire working memory can be used
OB	
• Number, max.	Limited only by RAM for code
Data areas and their retentivity	
retentive data area in total (incl. times, counters, flags), max.	10 kbyte
Flag	
• Number, max.	8 kbyte; Size of bit memory address area
Address area	
I/O address area	
• Inputs	1 024 byte
• Outputs	1 024 byte
Process image	
• Inputs, adjustable	1 kbyte
• Outputs, adjustable	1 kbyte
Hardware configuration	
Number of modules per system, max.	3 comm. modules, 1 signal board, 8 signal modules
Time of day	
Clock	
• Hardware clock (real-time clock)	Yes

• Deviation per day, max.	+/- 60 s/month at 25 °C
• Backup time	480 h; Typical

Digital inputs	
Number of digital inputs	14; Integrated
• of which, inputs usable for technological functions	6; HSC (High Speed Counting)
integrated channels (DI)	14
m/p-reading	Yes
Number of simultaneously controllable inputs	
all mounting positions	
— up to 40 °C, max.	14
Input voltage	
• Rated value (DC)	24 V
• for signal "0"	5 V DC at 1 mA
• for signal "1"	15 VDC at 2.5 mA
Input current	
• for signal "1", typ.	1 mA
Input delay (for rated value of input voltage)	
for standard inputs	
— Parameterizable	Yes; 0.2 ms, 0.4 ms, 0.8 ms, 1.6 ms, 3.2 ms, 6.4 ms and 12.8 ms, selectable in groups of four
— at "0" to "1", min.	0.2 ms
— at "0" to "1", max.	12.8 ms
for interrupt inputs	
— Parameterizable	Yes
for counter/technological functions	
— Parameterizable	Yes; Single phase : 3 at 100 kHz & 3 at 30 kHz, differential: 3 at 80 kHz & 3 at 30 kHz
Cable length	
• Cable length, shielded, max.	500 m; 50 m for technological functions
• Cable length unshielded, max.	300 m; For technological functions: No

Digital outputs	
Number of digital outputs	10
• of which high-speed outputs	4; 100 kHz Pulse Train Output
integrated channels (DO)	10
short-circuit protection	No; to be provided externally
Limitation of inductive shutdown voltage to	L+ (-48 V)
Switching capacity of the outputs	
• with resistive load, max.	0.5 A
• on lamp load, max.	5 W
Output voltage	
• for signal "0", max.	0.1 V; with 10 kOhm load
• for signal "1", min.	20 V
Output current	

• for signal "1" rated value	0.5 A
• for signal "0" residual current, max.	0.1 mA
Output delay with resistive load	
• "0" to "1", max.	1 µs
• "1" to "0", max.	5 µs
Switching frequency	
• of the pulse outputs, with resistive load, max.	100 kHz
Relay outputs	
• Max. number of relay outputs, integrated	0
Cable length	
• Cable length, shielded, max.	500 m
• Cable length unshielded, max.	150 m

Analog inputs	
Number of analog inputs	2
Integrated channels (AI)	2; 0 to 10 V
Input ranges	
• Voltage	Yes
Input ranges (rated values), voltages	
• 0 to +10 V	Yes
• Input resistance (0 to 10 V)	≥100k ohms
Cable length	
• Cable length, shielded, max.	100 m; twisted and shielded

Analog outputs	
Number of analog outputs	2
Integrated channels (AO)	2
Cable length	
• Cable length, shielded, max.	100 m; Shielded, twisted wire pair

Analog value creation	
Integration and conversion time/resolution per channel	
• Resolution with overrange (bit including sign), max.	10 bit
• Integration time, parameterizable	Yes
• Conversion time (per channel)	625 µs

Encoder	
Connectable encoders	
• 2-wire sensor	Yes

1st interface	
Interface type	PROFINET
Physics	Ethernet
Isolated	Yes
Automatic detection of transmission speed	Yes
Autonegotiation	Yes

Autocrossing	Yes
Functionality	
• PROFINET IO Controller	Yes
2nd interface	
Interface type	PROFINET
Physics	Ethernet
Communication functions	
S7 communication	
• supported	Yes
• as server	Yes
• As client	Yes
Open IE communication	
• TCP/IP	Yes
• ISO-on-TCP (RFC1006)	Yes
• UDP	Yes
Web server	
• supported	Yes
• User-defined websites	Yes
Number of connections	
• overall	16; dynamically
Test commissioning functions	
Status/control	
• Status/control variable	Yes
• Variables	Inputs/outputs, memory bits, DBs, distributed I/Os, timers, counters
Forcing	
• Forcing	Yes
Diagnostic buffer	
• present	Yes
Integrated Functions	
Number of counters	6
Counter frequency (counter) max.	100 kHz
Frequency meter	Yes
controlled positioning	Yes
PID controller	Yes
Number of alarm inputs	4
Number of pulse outputs	4
Galvanic isolation	
Galvanic isolation digital inputs	
• Galvanic isolation digital inputs	500V AC for 1 minute
• between the channels, in groups of	1
Galvanic isolation digital outputs	

• Galvanic isolation digital outputs	Yes
• between the channels	No
• between the channels, in groups of	1
Permissible potential difference	
between different circuits	500 V DC between 24 V DC and 5 V DC
EMC	
Interference immunity against discharge of static electricity	
• Interference immunity against discharge of static electricity acc. to IEC 61000-4-2	Yes
— Test voltage at air discharge	8 kV
— Test voltage at contact discharge	6 kV
Interference immunity to cable-borne interference	
• Interference immunity on supply lines acc. to IEC 61000-4-4	Yes
• Interference immunity on signal lines acc. to IEC 61000-4-4	Yes
Surge immunity	
• on the supply lines acc. to IEC 61000-4-5	Yes
Immunity against conducted interference induced by high-frequency fields	
• Interference immunity against high-frequency radiation acc. to IEC 61000-4-6	Yes
Emission of radio interference acc. to EN 55 011	
• Limit class A, for use in industrial areas	Yes; Group 1
• Limit class B, for use in residential areas	Yes; When appropriate measures are used to ensure compliance with the limits for Class B according to EN 55011
Degree and class of protection	
Degree of protection to EN 60529	
• IP20	Yes
Standards, approvals, certificates	
CE mark	Yes
CSA approval	Yes
UL approval	Yes
cULus	Yes
RCM (formerly C-TICK)	Yes
FM approval	Yes
Marine approval	
• Marine approval	Yes
Ambient conditions	
Operating temperature	
• Min.	-20 °C
• max.	60 °C
• horizontal installation, min.	-20 °C
• horizontal installation, max.	60 °C

• vertical installation, min.	-20 °C
• vertical installation, max.	50 °C
Storage/transport temperature	
• Min.	-40 °C
• max.	70 °C
Air pressure	
• Operation, min.	795 hPa
• Operation, max.	1 080 hPa
• Storage/transport, min.	660 hPa
• Storage/transport, max.	1 080 hPa
Relative humidity	
• Operation, max.	95 %; no condensation
Vibrations	
• Vibrations	2G wall mounting, 1G DIN rail
• Operation, checked according to IEC 60068-2-6	Yes
Shock test	
• checked according to IEC 60068-2-27	Yes; IEC 68, Part 2-27 half-sine: strength of the shock 15 g (peak value), duration 11 ms

Climatic and mechanical conditions for storage and transport	
Climatic conditions for storage and transport	
Free fall	
— Drop height, max. (in packaging)	0.3 m; five times, in dispatch package
Temperature	
— Permissible temperature range	-40 °C to +70 °C
Relative humidity	
— Permissible range (without condensation) at 25 °C	95 %

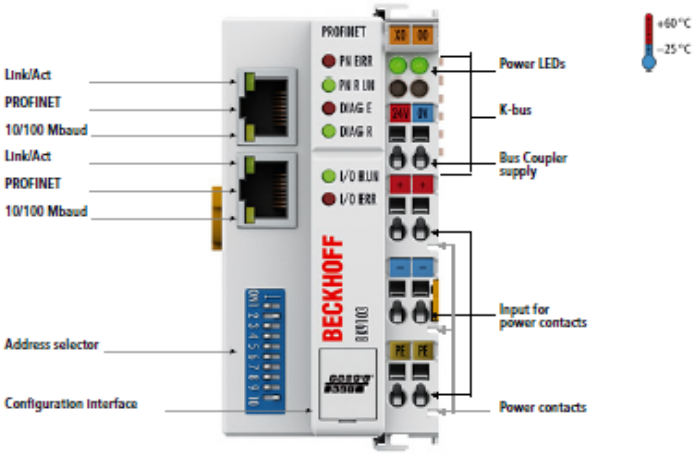
Mechanical and climatic conditions during operation	
Climatic conditions in operation	
Temperature	
— Min.	-20 °C
— max.	60 °C
Air pressure acc. to IEC 60068-2-13	
— Permissible air pressure	1080 to 795 hPa
— Permissible operating height	-1000 to 2000 m
Pollutant concentrations	
— SO2 at RH < 60% without condensation	S02: < 0.5 ppm; H2S: < 0.1 ppm; RH < 60% condensation-free

Configuration	
programming	
Programming language	
— LAD	Yes
— FBD	Yes

— SCL	Yes
Cycle time monitoring	
• can be set	Yes
Dimensions	
Width	130 mm
Height	100 mm
Depth	75 mm
Weights	
Weight, approx.	520 g
last modified:	14.10.2014

F.2 Beckoff I/O-modulpaket

PROFINET **BK9103**



BK9103 | PROFINET Bus Coupler

The BK9103 Bus Coupler connects PROFINET RT with the modular, extendable electronic terminal blocks. One unit consists of one Bus Coupler, any number from 1 to 64 terminals (255 with K-bus extension) and one end terminal.

The Bus Couplers recognise the terminals to which they are connected, and perform the assignment of the inputs and outputs to the words of the process image automatically. The BK9103 Bus Coupler supports 10 Mbit/s and 100 Mbit/s Ethernet. Connection is through normal RJ45 connectors. The IP address is set on the DIP switch (offset to a freely selectable start address). In networks with DHCP (a service for the allocation of the logical IP address to the physical node address [MAC-ID]) the Bus Coupler obtains its IP address from the DHCP server.

The BK9103 contains a 3-port switch. Two ports operate external on RJ45 connectors and can be utilised. The I/O stations can thus be configured with a line topology instead of the classic star topology. In many applications this significantly reduces the wiring effort and the cabling costs. The maximum distance between two couplers is 100 m. Up to 20 BK9103 Bus Couplers are cascadable, so that a maximum line length of 2 km can be achieved.

PROFINET is the open Industrial Ethernet standard of the PNO (PROFIBUS User Organisation). Internationally established IT standards such as TCP/IP are used for communication. PROFINET RT describes the data exchange between controllers and field devices and can be used in standard Ethernet networks. Commercial switches are used for networking purposes.

Complex signal processing for analog I/Os, position measurement, …

The BK9103 Bus Coupler supports the operation of all Bus Terminal types.

The analog and multi-functional Bus Terminals can be adapted to each specific application using the KS2000 configuration set. Depending on the type, the analog Bus Terminals' registers contain temperature ranges, gain values and linearisation characteristics. With the KS2000, the required parameters can be set on a PC. The Bus Terminals store settings permanently and in a fail-safe manner.

Optionally, the Bus Terminals can also be controlled by the control system. Via function blocks (FBs), the programmable logic controller (PLC) or the Industrial PC (IPC) handle configuration of the complete periphery during the start-up phase. If required, the controller can upload the decentrally created configuration data in order to centrally manage and store this data. Therefore, new adjustments are not necessary in the event of replacement of a Bus Terminal. The controller carries out the desired setting automatically after switching on.

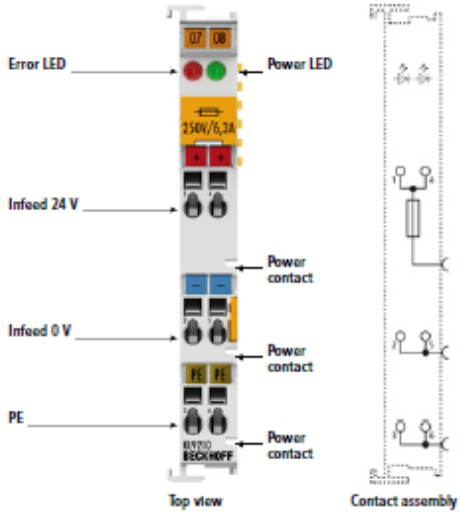
System data	PROFINET | BK9103
Number of I/O stations	only limited by IP addresses
Number of I/O points	depending on controller
Data transfer medium	4 x 2 twisted pair copper cable; category 3 (10 Mbaud), category 5 (100 Mbaud)
Distance between stations	100 m between hub/switch and Bus Coupler or between Bus Coupler and Bus Coupler
Data transfer rates	10/100 Mbaud
Topology	line or star wiring
Cascading	up to 20 BK9103 or max. line length 2 km

Technical data	BK9103
Number of Bus Terminals	64 (255 with K-bus extension)
Max. number of bytes fieldbus	512 byte input and 512 byte output
Digital peripheral signals	512 inputs/outputs
Analog peripheral signals	256 inputs/outputs
Protocol	PROFINET RT (Class B)
Configuration possibility	via KS2000
Data transfer rates	10/100 Mbaud, automatic recognition of the transmission rate
Bus interface	2 x RJ45 (2-channel switch)
Power supply	24 V DC (-15 %/+20 %)
Input current	70 mA + (total K-bus current)/4, 500 mA max.
Starting current	2.5 x continuous current
Recommended fuse	≤ 10 A
Current supply K-bus	1,750 mA
Power contacts	24 V DC max./10 A max.
Electrical isolation	500 V (power contact/supply voltage/fieldbus)
Weight	approx. 170 g
Operating/storage temperature	-25…+60 °C/-40…+85 °C
Relative humidity	95 %, no condensation
Vibration/shock resistance	conforms to EN 60068-2-6/EN 60068-2-27
EMC immunity/emission	conforms to EN 61000-6-2/EN 61000-6-4
Protect. class/installation pos.	IP 20/variable
Approvals	CE, UL, Ex, GL

Accessories	
TS6271	license for using the TwinCAT PROFINET RT Controller
KS2000	configuration software for extended parameterisation
Cordsets	cordsets and connectors
FC90xx-00xx	PC Fieldbus Cards with PCI interface

Ordering information	Description
BK9103	PROFINET Bus Coupler for up to 64 Bus Terminals (with integrated 2-channel switch)
BK9053	PROFINET "Compact" Bus Coupler for up to 64 Bus Terminals (255 with K-bus extension)
CX8093	PROFINET Embedded PC

System	
PROFINET	For further PROFINET products please see the system overview

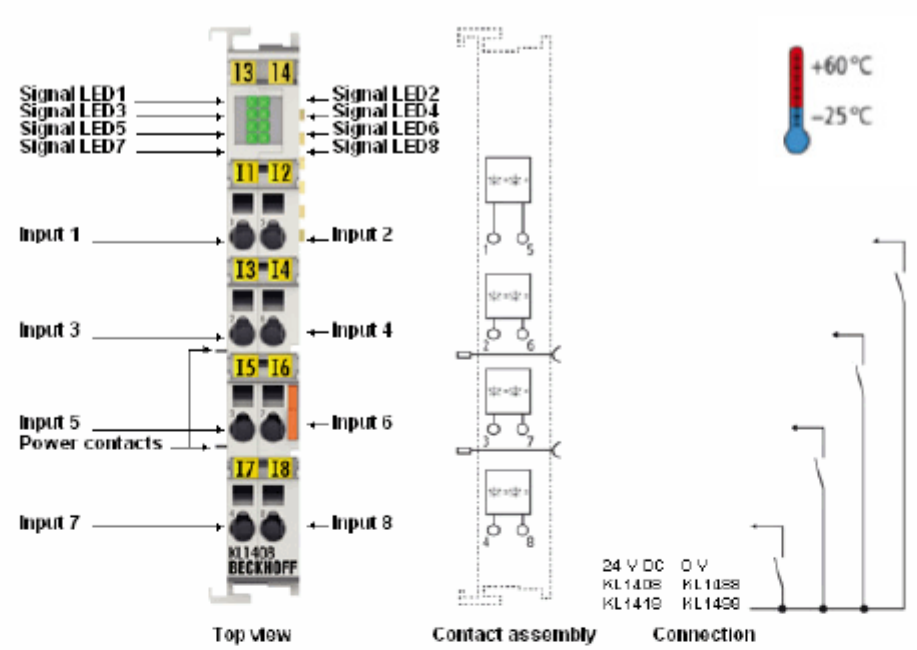


KL9210 | Potential supply terminal, 24 V DC, with diagnostics and fuse

The KL9210 power feed terminal makes it possible to set up various potential groups with the standard voltage of 24 V DC. In order to monitor the supply voltage, the terminal with diagnostics reports the status of the power feed terminal to the Bus Coupler through two input bits. It is thus possible for the controller to check the distributed peripheral voltage over the fieldbus.

Technical data	KL9210
Nominal voltage	24 V DC
Current load	≤ 10 A
Integrated fine-wire fuse	...6.3 A
Diagnostics	yes
Power LED	green
Defect LED	red
Reported to K-bus	yes
PE contact	yes
Shield connection	–
Connection facility to additional power contact	1
K-bus, looped through	yes
Bit width in the process image	2
Electrical connection to DIN rail	–
Current consumpt. K-bus	typ. 10 mA
Electrical isolation	yes
Housing width in mm	12
Special features	integrated fuse
Weight	approx. 55 g
Side by side mounting on Bus Terminals with power contact	yes
Side by side mounting on Bus Terminals without power contact	yes
Operating/storage temperature	-25...+60 °C/-40...+85 °C
Approvals	CE, UL, Ex, GL

Special terminals	
KL9210-0020	with 2 A fuse (slow-blow) and modified label

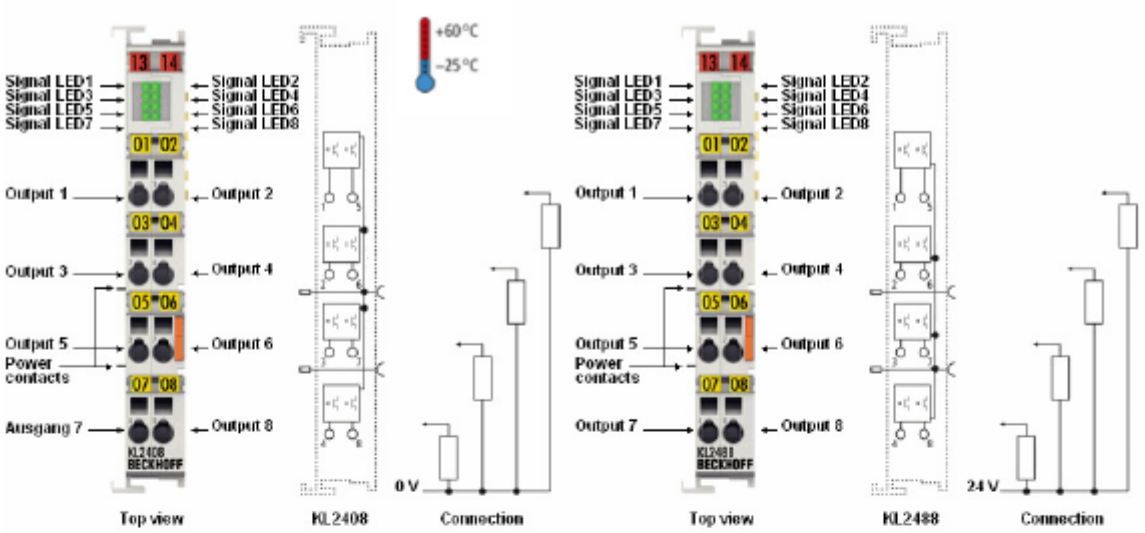


Eight-channel, digital input terminals, 24 V_{DC}

The digital input terminals KL1408 and KL1418 (positive switching) and KL1488 and KL1498 (negative switching) acquire the binary control signals from the process level and transmit them, in an electrically isolated form, to the higher-level automation unit. The Bus Terminals each contain eight channels, whose signal states are displayed by LEDs. They are particularly suitable for space-saving use in control cabinets. By using the single-conductor connection technique a multi-channel sensor can be connected in the smallest space with a minimum amount of wiring. The power contacts are looped through. For the KL1408 and KL1418 Bus Terminals, the reference ground for all inputs is the 0 V power contact. For the KL1488 and KL1498 Bus Terminals, the reference point for all inputs is the 24 V power contact. These versions have input filters with different speeds.

Technical data	KL1408/KS1408	KL1418/KS1418	KL1488/KS1488	KL1498/KS1498
Number of inputs / rated voltage	8 / 24 V_{DC} (-15% / +20%)			
Signal voltage “0”	-3 V ... +5 V (IEC 61131-2, type 1/3)		18 V ... 30 V	
Signal voltage “1”	15 V ... 30 V (IEC 61131-2, type 3)		0 V ... 7 V	
Signal current “0”	0 ... 1.5 mA	0 ... 1.5 mA	-	-
Signal current “1”	2.0 ... 2.5 mA	2.0 ... 2.5 mA	typ. 3 mA	typ. 3 mA
Input filter	3 ms	0.2 ms	3 ms	0.2 ms
Current consumption from K-bus	typ. 5 mA			
Electrical isolation	500 V (K-Bus/field potential)			
Bit width in process image	8 input bits			
Configuration	no address-or configuration settings required			
Dimensions (W x H x D)	15mm x 100mm x 70mm (connected width: 12mm)			
Weight	approx. 55 g			
Permissible ambient temperature range	-25 °C ... +60 °C in operation 0 °C ... +55 °C (according to cULus for Canada and USA) 0 °C ... +55 °C (according to ATEX, see special conditions)		-0 °C... +55 °C in operation	
	-40 °C... +85 °C during storage		-25 °C... +85 °C during storage	
Relative humidity	5% ... 95%, no condensation			
Vibration / shock resistance	conforms to EN 60068-2-6 / EN 60068-2-27			

Beckhoff Automation GmbH, Eiserstr. 5, 33415 Verl, Germany, Tel.: +49 (0) 5246 963 0, Fax.: +49 (0) 5246 963 149, http://www.beckhoff.com

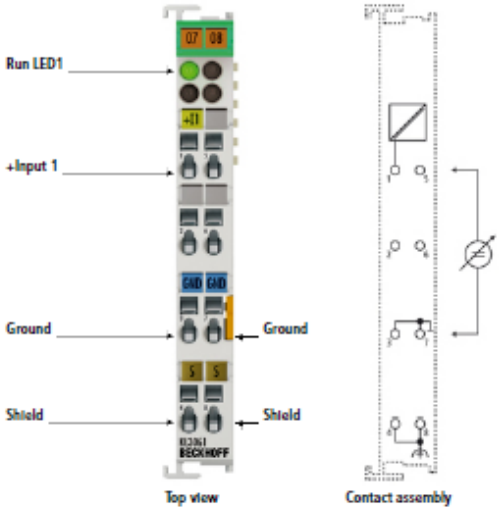


Eight-channel, digital output terminal, 24 V_{DC}

The KL2408 (positive switching) and KL2488 (negative switching) digital output terminals connect the binary control signals from the automation unit on to the actuators at the process level with electrical isolation. The KL2408/KL2488 variants are protected against reverse polarity connection. They handle load currents with outputs that are protected against overload and short circuit. The Bus Terminals contain eight channels which indicate their signal state by means of light emitting diodes. They are particularly suitable for space-saving use in control cabinets. The connection technology is particularly suitable for single-ended inputs. All components have to use the same reference point as the KL2408 or KL2488. The power contacts are looped through. In the KL2408 terminal, the outputs are supplied by the 24 V power contact. In the KL2488 terminal, they are supplied via the 0 V power contact.

Technical data	KL2408 / KS2408	KL2488 / KS2488
Connection technology	1 wire	
Number of outputs	8 (positive switching)	8 (negative switching)
Rated voltage	24 V_{DC} (-15% / +20%)	
Load type	ohmic, inductive, lamp load	
Output current max. (per channel)	0.5 A (short-circuit-proof)	
Short circuit current	< 2 A	< 7 A
Breaking energy	< 150 mJ/channel	< 100 mJ/channel
Reverse voltage protection	yes	
Electrical isolation	500 V (K-Bus/field potential)	
Current consumption K-bus	typ. 18 mA	
Current consumption power contacts	typ. 60 mA +load	
Bit width in process image	8 output bits	
Configuration	no address-or configuration settings required	
Dimensions (W x H x D)	15mm x 100mm x 70mm (connected width: 12mm)	
Weight	approx. 70 g	
Permissible ambient temperature range	-25°C ... +60°C in operation 0°C ... +55°C (according to cULus for Canada and USA) 0°C ... +55°C (according to ATEX, see special conditions)	-0°C... +55 °C in operation
	-40°C... +85 °C during storage	-25°C... +85 °C during storage
Relative humidity	5% ... 95%, no condensation	
Vibration / shock resistance	conforms to EN 60068-2-6 / EN 60068-2-27	

Analog input KL3061, KL3062



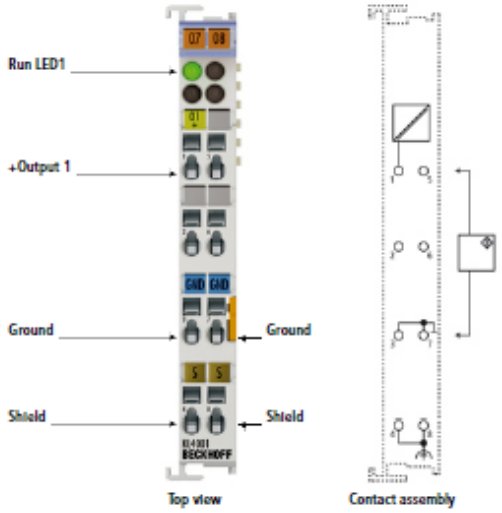
25 g

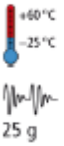
KL3061, KL3062 | 1-, 2-channel analog input terminals 0…10 V

The KL3061 and KL3062 analog input terminals handle signals in the range from 0 to 10 V. The voltage is digitised to a resolution of 12 bits and is transmitted, electrically isolated, to the higher-level automation device. The input channels of a Bus Terminal have a common ground potential, the reference ground. The KL3061 is the single-channel variant and is characterised by its fine granularity and electrical isolation. The KL3062 version combines two channels in one housing. The run LEDs give an indication of the data exchange with the Bus Coupler.

Technical data	KL3061 | KS3061	KL3062 | KS3062
Number of inputs	1	2
Power supply	via the K-bus	
Signal voltage	0…10 V	
Internal resistance	> 130 kΩ	
Common-mode voltage U_{CM}	–	
Resolution	12 bit	
Conversion time	~ 1 ms	~ 2 ms
Measuring error	< ±0.3 % (relative to full scale value)	
Electrical isolation	500 V (K-bus/signal voltage)	
Current consumption power contacts	– (no power contacts)	
Current consumpt. K-bus	typ. 60 mA	
Bit width in the process image	input: 1 x 16 bit data (1 x 8 bit control/status optional)	input: 2 x 16 bit data (2 x 8 bit control/status optional)
Configuration	no address or configuration setting	
Weight	approx. 60 g	
Operating/storage temperature	0…+55 °C/-25…+85 °C	
Relative humidity	95 %, no condensation	
Vibration/shock resistance	conforms to EN 60068-2-6/EN 60068-2-27	
EMC immunity/emission	conforms to EN 61000-6-2/EN 61000-6-4	
Protect. class/installation pos.	IP 20/variable	
Pluggable wiring	for all KSxxxx Bus Terminals	
Approvals	CE, UL, Ex	

Special terminals	
KL3062-0010	Siemens S5 format
KL3062-0011	voltage level 0…20 V
KL3062-0012	fast µP, scan time approx. 0.5 ms
KL3062-0013	voltage level 0…30 V
KL3062-0050	Siemens S7 format





KL4001, KL4002 | 1-, 2-channel analog output terminals 0…10 V

The KL4001 and KL4002 analog output terminals generate signals in the range from 0 to 10 V. The voltage is supplied to the process level with a resolution of 12 bits and is electrically isolated. The output channels of a Bus Terminal have a common ground potential. The KL4001 is the single-channel variant and is particularly suitable for signals with electrically isolated ground potentials. The KL4002 version combines two channels in one housing. The run LEDs give an indication of the data exchange with the Bus Coupler.

Technical data	KL4001 | KS4001	KL4002 | KS4002
Technology	–	single-ended
Number of outputs	1	2
Power supply	via the K-bus	
Signal voltage	0…10 V	
Load	> 5 kΩ (short-circuit-proof)	
Output error	< ±0.1 % (relative to end value)	
Resolution	12 bit	
Conversion time	~ 1.5 ms	
Electrical isolation	500 V (K-bus/signal voltage)	
Current consumption power contacts	– (no power contacts)	
Current consumpt. K-bus	typ. 75 mA	
Bit width in the process image	output: 1 x 16 bit data (1 x 8 bit control/status optional)	output: 2 x 16 bit data (2 x 8 bit control/status optional)
Configuration	no address or configuration setting	
Special features	potential-free output	–
Weight	approx. 85 g	
Operating/storage temperature	-25…+60 °C/-40…+85 °C	
Relative humidity	95 %, no condensation	
Vibration/shock resistance	conforms to EN 60068-2-6/EN 60068-2-27	
EMC immunity/emission	conforms to EN 61000-6-2/EN 61000-6-4	
Protect. class/installation pos.	IP 20/variable	
Pluggable wiring	for all KSxxxx Bus Terminals	
Approvals	CE, UL, Ex	

Special terminals	
KL4002-0010	Siemens S5 format
KL4002-0011	fast µP, scan time approx. 0.15 ms
KL4002-0050	Siemens S7 format

K-Bus end terminal

The KL9010 K-Bus end terminal is necessary for data exchange between the Bus Coupler and the Bus Terminals. Each Bus Terminal block has to be terminated at the right hand end with a K-Bus end terminal.
The K-Bus end terminal KL9010 does not have any other function or connection facility.

Technical Data	KL9010
Current consumption from the K-Bus	-
Electrical isolation	500 V (K-Bus/signal voltage)
Bit width in the process image	-
Configuration	no address or configuration settings
Weight	ca. 50 g
Permissible ambient temperature range	-25 °C ... +60 °C in operation 0 °C ... +55 °C (according to cULus for Canada and USA) 0 °C ... +55 °C (according to ATEX, see special conditions)
	-40 °C ... +85 °C (during storage)
Permissible relative humidity	5% ... 95%, no condensation
EMC resistance burst / ESD	conforms to EN 61000-6-2 / EN 61000-6-4
Vibration / shock resistance	conforms to EN 60068-2-6 / EN 60068-2-2
Protection class	IP20
Installation pos.	Variable
Approval	CE, cULus, ATEX, GL

Bilaga G CANopen-profil för ZAPI-motorstyrning

Nedan kan man se CANopen-profilen för motorstyrningen man har använt i projektet. Detta är en korrigerad version, eftersom den version man använde sig av i början inte var fullständig.

G.1 Communication entries Objekt index

Index (hex)	Subindex (hex)	Bit	Name	Type(index)	Acc	Def	Description
1000	0		device type	Unsigned32(7)	ro	0h	**non standard value**
1001	0		error register	Unsigned8(5)	ro	0h	
		0	generic error				if set to 1 <=> error
		1..7	not used				not used
1003			pre-defined error field				
	0		number of errors	Unsigned8(5)	rw		
	1		actual error code	Unsigned32(7)	ro		
1005	0		COB-ID SYNC-message	Unsigned32(7)	ro	80h	SYNC: Master ->All
1006	0		communication cycle period	Unsigned32(7)	ro	32000d	32ms
1007	0		synchronous window length	Unsigned32(7)	ro	16000d	16ms
1008	0		manufacturer device name	visible-string(9)	ro	ZP	**non standard (4 char only)**
1009	0		manufacturer hardware version	visible-string(9)	ro	AET2	**non standard (4 char only)**
100A	0		manufacturer software version	visible-string(9)	ro	118	**non standard (4 char only)**
100B	0		node-ID	Unsigned32(7)	ro	8+ID	ID: A5 and A13 input
100C	0		guard time	Unsigned16(6)	ro	0	
100D	0		life time factor	Unsigned8(5)	ro	0	
1010			store parameters				
	0		largets supported sub-index	Unsigned8(5)	ro	1h	
	1		save all parameters	Unsigned32(7)	rw		signature 73H 61H 76H 65H <=> "save"
1011			restore default parameters	Unsigned32(7)			
	0		largets supported sub-index	Unsigned8(5)	ro	1h	
	1		restore all default parameters (clear eeprom)	Unsigned32(7)	rw		signature 6CH 6FH 61H 64H <=> "load"
1014	0		COB-ID emergency message	Unsigned32(7)	ro	80h+ID	EMCY: TMC->ATC
1200			server SDO parameter				
	0		number of entries	Unsigned8(5)	ro	2h	
	1		COB-ID client->server (rx)	Unsigned32(7)	ro	600h+ID	SDO1: Master -> AC
	2		COB-ID server->client (tx)	Unsigned32(7)	ro	580h+ID	SDO1: AC1 -> Master
1400			1st receive PDO parameter				PDO1: Master -> AC
	0		number of entries	Unsigned8(5)	ro	2h	
	1		COB-ID used by PDO	Unsigned32(7)	ro	200h+ID	
	2		transmission type	Unsigned8(5)	ro	1d	synchronous cyclic PDO (1 SYNC/PDO)
1600			1st receive PDO mapping				PDO1: Master -> AC
	0		number of entries	Unsigned8(5)	ro	5h	
	1		1st mapping to parameter in object dictionary	Unsigned32(7)	ro	21E10010h	target speed
	2		2nd mapping to parameter in object dictionary	Unsigned32(7)	ro	21E20010h	controlword
	3		3rd mapping to parameter in object dictionary	Unsigned32(7)	ro	21E50008h	pedal braking request
	4		4th mapping to parameter in object dictionary	Unsigned32(7)	ro	21E60008h	perc max slip
	5		5th mapping to parameter in object dictionary	Unsigned32(7)	ro	30010110	free
1800			1st transmit PDO parameter				PDO1: AC -> Master
	0		number of entries	Unsigned8(5)	ro	2h	
	1		COB-ID used by PDO	Unsigned32(7)	ro	180h+ID	
	2		transmission type	Unsigned8(5)	ro	1d	synchronous cyclic PDO (1 SYNC/PDO)
1801			2nd transmit PDO parameter				PDO2: AC->Master
	0		number of entries	Unsigned8(5)	ro	2h	

	1		COB-ID used by PDO	Unsigned32(7)	ro	280h+ID	
	2		transmission type	Unsigned8(5)	ro	40d	synchronous cyclic PDO (40 SYNC/PDO)
1A00			1st transmit PDO mapping				PDO1: AC->Master
	0		number of entries	Unsigned8(5)	ro	6h	
	1		1st mapping to parameter in object dictionary	Unsigned32(7)	ro	21FB0010h	actual vehicle speed
	2		2nd mapping to parameter in object dictionary	Unsigned32(7)	ro	21E30010h	statusword
	3		3rd mapping to parameter in object dictionary	Unsigned32(7)	ro	21280008h	Aanalog input#3 CNA23
	4		4th mapping to parameter in object dictionary	Unsigned32(7)	ro	212A0008h	Analog input #1 CNA3
	5		5th mapping to parameter in object dictionary	Unsigned32(7)	ro	21290008h	Analog input #2 CNA10
	6		6rd mapping to parameter in object dictionary	Unsigned32(7)	ro	21F80008h	actual current
1A01			2nd transmit PDO mapping				PDO2: AC->Master
	0		number of entries	Unsigned8(5)	ro	5h	
	1		1st mapping to parameter in object dictionary	Unsigned32(7)	ro	21E40010h	warning code
	2		2nd mapping to parameter in object dictionary	Unsigned32(7)	ro	30000110	free
	3		3nd mapping to parameter in object dictionary	Unsigned32(7)	ro	210F0008h	actual controller temperature
	4		4rd mapping to parameter in object dictionary	Unsigned32(7)	ro	21F20008h	actual battery status
	5		5rd mapping to parameter in object dictionary	Unsigned32(7)	ro	30000210	free

G.2 Manufacture Specific Objekt index

				Manufacturer specific profile area							
Index(hex)	PDO map.	Subindex	bit	Name	Type	Acc	default	Description	Min	Max	Unit
200B	No			par_release_braking	Unsigned8(5)	rw	30		5	200	s/10
200C	No			par_inversion_braking	Unsigned8(5)	rw	15		5	200	s/10
200D	No			par_pedal_braking	Unsigned8(5)	rw	55		5	200	s/10
200E	No			par_braking_modulation (*)	Unsigned8(5)	rw	100		5	200	Hz
2010	No			par_cutback_braking	Unsigned8(5)	rw	55		5	200	s/10
2011	No			par_brake_safety	Unsigned8(5)	rw	5	target_time_100->0Hz	5	200	s/10
2012	No			par_brake_type2	Unsigned8(5)	rw	15	target_time_100->0Hz	5	200	s/10
2013	No			par_brake_type1	Unsigned8(5)	rw	15	target_time_100->0Hz	5	200	s/10
2035	No			opt_compensation (*)	Unsigned8(5)	rw	1	0:off;1:on	0	1	-
2038	No			par_aux_time	Unsigned8(5)	rw	15	elec_brake time	0	255	s/10
203F	No			par_maximum_current	Unsigned8(5)	rw	100	current_percentage	0	100	%
205C	No			par_option_8 (*)	Unsigned8(5)	rw	7		0	9	-
205D	No			par_option_7(*)	Unsigned8(5)	rw	1		0	9	-
205E	No			par_option_6 (*)	Unsigned8(5)	rw	6		0	9	-
2077	No			opt_aux_output_1	Unsigned8(5)	rw	1		0	1	-
2079	No			opt_battery_check	Unsigned8(5)	rw	0		0	3	-
207D	No			opt_hour_meter	Unsigned8(5)	rw	0	0:running;1:key_on	0	1	-
2088	No			par_sat_frequency _hydro(*)	Unsigned8(5)	rw	94		10	200	Hz
208C	No			adjust_battery (**)	Unsigned16(6)	rw	240		0	480	V/10
208D	No			Reserved							
208E	No			Reserved							
2091	No			par_battery_type	Unsigned8(5)	rw	2	0:24V; 1:36V; 2:48V; 3:72V; 4:80V; 5:96V			
209D	No			par_adjustement#3	Unsigned8(5)	rw	80		10	90	%
209E	No			Reserved (**)							
209F	No			Reserved (**)							
20A8	No			par_main_contactor_volt	Unsigned8(5)	rw	202		0	255	100/255%
20A9	No			par_battery_discharge_adj1	Unsigned8(5)	rw	3		0	9	-
20AA	No			par_battery_discharge_adj2	Unsigned8(5)	rw	3		0	9	-
20AB	No			opt_dc_link_compensation (*)	Unsigned8(5)	rw	1	0:off;1:on	0	1	-
20AC	No			par_sat_frequency (*)	Unsigned8(5)	rw	94		10	200	Hz
20AD	No			par_boost_at_hi_freq (*)	Unsigned8(5)	rw	39	voltage percentage	0	200	100/255%
20AE	No			par_boost_corner_freq (*)	Unsigned8(5)	rw	80		0	200	Hz
20AF	No			par_braking_booster (*)	Unsigned8(5)	rw	141	voltage percentage	0	255	100/255%
20B0	No			par_slip_coefficient (*)	Unsigned8(5)	rw	0		0	9	-
20B2	No			Reserved							
20B6	No			Reserved							
20B7	No			Reserved							
20B8	No			par_safety_in_config	Unsigned8(5)	rw	3		0	9	-
20BB	No			par_main_cont_v_rid	Unsigned8(5)	rw	202		0	255	100/255%
20BC	No			par_aux_output_v_rid	Unsigned8(5)	rw	202		0	255	100/255%
20BF	No			par_maxslip0 (*)	Unsigned8(5)	rw	TBD	slip	0	200	Hz/10
20C0	No			par_maxslip1 (*)	Unsigned8(5)	rw	TBD	slip	0	200	Hz/10
20C1	No			par_maxslip2 (*)	Unsigned8(5)	rw	TBD	slip	0	200	Hz/10
20C2	No			par_maxslip3 (*)	Unsigned8(5)	rw	TBD	slip	0	200	Hz/10
20C3	No			par_maxslip4(*)	Unsigned8(5)	rw	TBD	slip	0	200	Hz/10
20C4	No			par_freq_slip1(*)	Unsigned8(5)	rw	TBD		0	200	Hz
20C5	No			par_freq_slip2(*)	Unsigned8(5)	rw	TBD		0	200	Hz
20C6	No			par_freq_slip3(*)	Unsigned8(5)	rw	TBD		0	200	Hz
20C7	No			par_freq_slip4(*)	Unsigned8(5)	rw	TBD		0	200	Hz
20C8	No			par_maxslip0_brk (*)	Unsigned8(5)	rw	TBD	slip	0	200	Hz/10
20C9	No			par_maxslip1_brk(*)	Unsigned8(5)	rw	TBD	slip	0	200	Hz/10
20CA	No			par_maxslip2_brk(*)	Unsigned8(5)	rw	TBD	slip	0	200	Hz/10
20CB	No			par_maxslip3_brk(*)	Unsigned8(5)	rw	TBD	slip	0	200	Hz/10
20CC	No			par_maxslip4_brk(*)	Unsigned8(5)	rw	TBD	slip	0	200	Hz/10
20CD	No			par_freq_slip1_brk(*)	Unsigned8(5)	rw	TBD		0	100	Hz
20CE	No			par_freq_slip2_brk(*)	Unsigned8(5)	rw	TBD		0	100	Hz
20CF	No			par_freq_slip3_brk(*)	Unsigned8(5)	rw	TBD		0	100	Hz
20D0	No			par_freq_slip4_brk(*)	Unsigned8(5)	rw	TBD		0	100	Hz
20D1	No			par_acceleration0_inversion_delay	Unsigned8(5)	rw	x		3	100	s/10
20D2	No			par_acceleration3_delay	Unsigned8(5)	rw	x		3	100	s/10
20D3	No			par_acc_prof_freq3	Unsigned8(5)	rw	100		5	200	Hz
20D4	No			opt_debug_type	Unsigned8(5)	rw	1	0:option #1;1:option #2	0	1	-
20D5	No			par_brake_smooth	Unsigned8(5)	rw	-		0	50	s/10
20D6	No			Reserved							
20D7	No			par_stop_brk_smooth	Unsigned8(5)	rw	TBD		0	100	Hz
20D8	No			opt_auto_park_brake	Unsigned8(5)	rw	1	0:off -> ebrake is drived by bit2 in controlword ;1:on -> ebrake is drived directly by zapi motor control	0	1	-
20D9	No			par_frequency_creep	Unsigned8(5)	rw	3	frequency	3	200	Hz/10
20DA	No			par_adj_slip_mot_temperature	Unsigned8(5)	rw	3		0	150	%
20DB	No			opt_emergency_switch	Unsigned8(5)	rw	1	0:off;1:on	0	1	-
20DC	No			par_tooths (***)	Unsigned8(5)	rw	3	0:177; 1:48; 2:64; 3:80;			
20DE	No			par_boost_at_lo_freq (*)	Unsigned8(5)	rw	39	voltage percentage	0	200	100/255%
20DF	No			par_maxslip_reset (*)	Unsigned8(5)	rw	6	slip	0	200	Hz/10
20E2	No			par_minimum_voltage (*)	Unsigned8(5)	rw	10	voltage percentage	0	50	100/255%
20E3	No			opt_slip_control (*)	Unsigned8(5)	rw	1	0:off;1:on Option for select if A18 positive is always present or is cut by an external device ABSENT: positive is always present PRESENT: controller need positive status in controlword BIT 5	0	1	-
20E4	No			A18 POWER CUTOUT	Unsigned8(5)	rw	1		0	1	-
20E6	No			par_acceleration0_delay	Unsigned8(5)	rw	x		3	100	s/10
20E7	No			par_acceleration1_delay	Unsigned8(5)	rw	x		3	100	s/10
20E8	No			par_acceleration2_delay	Unsigned8(5)	rw	x		3	100	s/10

20E9	No			opt_stop_on_ramp	Unsigned8(5)	rw	0	0:off;1:on	0	1	-
20EB	No			opt_hl_dynamic	Unsigned8(5)	rw	1	0:off;1:on	0	1	-
20EC	No			opt_torque	Unsigned8(5)	rw	1	0:off;1:on	0	1	-
20EE	No			par_motor_resistance (*)	Unsigned8(5)	rw	0		0	9	-
20ED	No			par_deceleration_braking	Unsigned8(5)	rw	89	target_time_100->0Hz	5	200	s/10
20F1	No			opt_th_mot_type	Unsigned8(5)	rw	0		0	2	-
								OFF = A16 is not used OPTION #1 = A16 is used for drive power line OPTION #2 = A16 used like a generic output. If parameter set to OFF or OPTION #2 presence of			
20F2	No			A16 function	Unsigned8(5)	rw	1	power on DC link has to be send in controlword BIT	0	2	-
20F3	No			par_tiller_release_braking	Unsigned8(5)	rw	15		5	200	s/10
20F5	No			par_acc_prof_freq1	Unsigned8(5)	rw	100		5	200	Hz
20F6	No			par_acc_prof_freq2	Unsigned8(5)	rw	100		5	200	Hz
20F7	No			opt_overmodulation	Unsigned8(5)	rw	0	0:off;1:on	0	1	-
20F8	No			opt_fault_reset_bit	Unsigned8(5)	rw	1	0:off;1:on	0	1	-
20F9	No			opt_aux_function	Unsigned8(5)	rw	1	0:off;1:on	0	2	-
20FA	No			par_aux_output_volt	Unsigned8(5)	rw	202		0	255	100/255%
20FB	No			opt_guarding	Unsigned8(5)	rw	1	0:off;1:on	0	1	-
20FC	No			par_overmodulation_delta_fc	Unsigned8(5)	rw	0	voltage percentage	10	100	100/255%
20FD	No			opt_debug_message	Unsigned8(5)	rw	1	0:running;1:keyon	0	1	-
210F	default			actual controller temperature(ts_temperature)	Signed8(2)	ro	20	mapped on PDO_TX2	-128	127	°C
2110	default			actual motor temperature	Signed8(2)	ro	20		0	200	°C
2114	No			ts_battery_voltage	Unsigned16(8)	ro	240		0	480	V/10
2115	No			ts_motor_voltage	Unsigned8(5)	ro	20		0	255	100/255%
2128	default			ts_analog_input3	Unsigned8(5)	ro	0	E3 analog input	0	255	100/255%
2129	default			ts_analog_input2	Unsigned8(5)	ro	0	F8 analog input	0	255	100/255%
212A	default			ts_analog_input1	Unsigned8(5)	ro	0	E1 analog input	0	255	100/255%
2151	No			ts_frequency	Integer16(3)	ro	0		-20000	20000	Hz/100
21E0	No			ts_node_id	Unsigned8(5)	ro	0		0	3	-
21E1	default			target_speed	Integer16(3)	rw	0	Mapped on PDO_RX1	0	20000	Hz/100
21E2	default			controlword	Unsigned16(8)	rw	0	Mapped on PDO_RX1			
			0		Boolean	rw	False				
			1	close_out_A16	Boolean	rw	False	F8			
			2	close_out_A16	Boolean	rw	False	F9			
			3	power_line_present	Boolean	rw	False	Future use			
			4	fault_reset	Boolean	rw	False	Future use			
			5	out_A16_power_on	Boolean	rw	False	gives full speed in fork direction during X seconds			
			6	forward_req	Boolean	rw	False				
			7	reverse_req	Boolean	rw	False				
			8	hydro_req	Boolean	rw	False				
			9	safety_present	Boolean	rw	False				
			10	Free	Boolean	rw	False				
			11-12	braking_type	Boolean	rw	False	0:standard; 1:brake type 1; 2 brake type2; 3 safety brake			
			13	Free	Boolean	rw	False				
			14	Free	Boolean	rw	False				
			15	stuffing	Boolean	rw	False				
21E3	default			statusword	Unsigned16(8)	ro	-	Mapped on PDO_TX1			
			0		Boolean	ro	False				
			1	out_A16_status							
			2	out_A18_status							
			3	require_power_line	Boolean	ro	False				
			4	out_A19_status	Boolean	ro	False				
			5	Free	Boolean	ro	False				
			6	Free	Boolean	ro	False				
			7	Free	Boolean	ro	False				
			8	CNA13 (ID1)	Boolean	ro	False				
			9	CNA5(ID0)	Boolean	ro	False				
			10	CNA8	Boolean	ro	False				
			11	CNA4 (emergency)	Boolean	ro	False				
			12	Free	Boolean	ro	False				
			13	Free	Boolean	ro	False				
			14	idle_mode	Boolean	ro	False				
			15	stuffing	Boolean	ro	False				
21E4	default			warning code	Unsigned16(8)	ro	0	Mapped on PDO_TX2			
21E5	default			pedal_braking_request	Unsigned8(5)	rw	0	Mapped on PDO_RX1	0	255	100/255%
21E6	default			torque_setpoint	Unsigned8(5)	rw	0	Mapped on PDO_RX1	0	255	100/255%
21F1	default			motor_power	Unsigned8(5)	rw	0		-32767	32768	
21F2	default			actual battery status (ts_battery_charge)	Unsigned8(5)	ro	100	mapped on PDO_TX2	0	100	%
21F3	No			ts_voltage_booster	Unsigned8(5)	ro	0		0	255	100/255%
21F8	default			actual current (ts_current_rms)	Unsigned8(5)	ro	0	mapped on PDO_TX1	0	2000	A*5
21FA	No			ts_slip	Integer16(3)	ro	0		-20000	20000	Hz/100
21FB	default			actual vehicle speed (ts_encoder)	Integer16(3)	ro	0	mapped on PDO_TX1	-20000	20000	Hz/100
2200	No			Alarm logbook							
		0		number of entry	6						
		1		Logbook entry	Unsigned8(5)	rw	0	writing 0 clear logbook	0	5	-
		2		Alarm Zapi code	Unsigned8(5)	ro	0		0	0xFF	-
		3		occurences	Unsigned8(5)	ro	0		0	99	-
		4		Temperature	Integer8(2)	ro	0		-128	127	°C
		5		hour_meter	Unsigned16(8)	ro	0		0	65536	h

(*) Motor control parameter
(**) adjusted in Zapi
(***) only for Atech

G.3 PDO map

PDO MAP							
Object Name	**Id(Hex)**	**Sender**	**Delay Time**	**Dimension**	**Position(byte.bit)**	**Means**	
PDO1_RX	200h+ID	Master	16ms				
Target_speed (21E1)				Unsigned16	0.0	Target speed in Hz/100	
Control word (21E2)				Unsigned16	2.0		
					2.1	close_out_A16	command for A16 output (see parameter A16 function)
					2.2	close_out_A18	command for A18 output (see parameter AUTO PARK BRAKE and A18 FUNC
					2.3	power_line_present	Power presence on + Capacitor (see parameter A16 function)
					2.4	fault_reset	
					2.5	out_A18_power_on	presence positive on A18 output (see A18 power cutout)
					2.6	forward_req	
					2.7	reverse_req	
					3.0	hydro_req	
					3.1	safety_present	A11 input status for cros check
					3.2	Free	
					3.3-3.4	braking_type	
					3.5	Free	
					3.6	Free	
					3.7	stuffing	
pedal braking request(21E5)				Unsigned8	4.0	0-255	Change braking intenity. Do not use if you are using torque request.
torque request (21E6)				Unsigned8	5.0	0-255	0 = coasting mode, 255 = max torque
accel./decel. ramp				Unsigned8	6.0	0-255	[sec/10] for 0-100 Hz. If value = 0 -> controller ramps are used
Object Name	**Id(Hex)**	**Sender**	**Delay Time**	**Dimension**	**Position(byte.bit)**	**Means**	
PDO1_TX	180+ID	AC	1 syncro				
actual vehicle speed(21FB)				Signed16	0.0	speed in Hz	
Status Word(21E3)				Unsigned16	2.0		
					2.1	out_A16_status	
					2.2	out_A18_status	
					2.3	require_power_line	
					2.4	out_A19_status	
					2.5	Free	
					2.6	Free	
					2.7	Free	
					3.0	CNA13 (ID1)	
					3.1	CNA5(ID0)	
					3.2	CNA6	
					3.3	CNA4 (emergency)	
					3.4	Free	
					3.5	Free	
					3.6	Idle_mode	
					3.7	stuffing	
Analog input#3 (2128)					4.0	CNA22 [0,,,255]	
Analog input#1 (212A)					5.0	CNA3 [0 ,,,, 255]	
Analog input #2 (2129)					6.0	CNA10 [0 ,,, 255]	
actual current(2128)					7.0	[A/5]	
Object Name	**Id(Hex)**	**Sender**	**Delay Time**	**Dimension**	**Position(byte.bit)**	**Means**	
PDO2_TX	280+ID	AC	256ms				
warning/alarm				Unsigned16	0.0		
temperature				Signed8	4.0		
BDI percentage				Unsigned8	5.0		
motor temperature				Signed16	6.0		

G.4 PDO Exempel

check in message 0x88 if is present alarm.
send every 20ms syncro message 0x80.
send every 20ms pdo1_rx 0x208
0x208.3.7 has to change from 0 to 1 and viceversa every messages

ID	data0	data1	data2	data3	data4	data5	data6
0x88	0	0	0	0	0	0	0
0x80							
0x208	0	0	3	0	0	0	0
0x80							
0x208	0	0	3	0x80	0	0	0

	0	1	0				
0x80							
0x208	0	0	3	0x80	0	0	0
0x188	0	0		3 0	0	0	0
0x80							
0x208	0	0	3	0x00	0	0	0
0x188	0	0		3 0	0	0	0

0x80							
0x208	0xF4	1	0x43	0x80	0	0	0
0x188	5	0		3 0	0	0	0

G.5 Alarm koder

ZAPI code (dec)	ZAPI string	Emergency Code
0	" NONE "	0000h
17	"LOGIC FAILURE #3"	FF11h
18	"LOGIC FAILURE #2"	FF12h
19	"LOGIC FAILURE #1"	FF13h
30	" VMN LOW "	3120h
31	" VMN HIGH "	3110h
37	" CONTACTOR CLOSED"	FF25h
38	" CONTACTOR OPEN "	5441h
53	" STBY I HIGH "	2311h
60	"CAPACITOR CHARGE"	3130h
74	" DRIVER SHORTED "	3211h
75	"CONTACTOR DRIVER"	3221h
82	" ENCODER ERROR"	7310h
218	"PROG TOOTHS" (***)	FFFDh
219	"GENERIC ERROR"	1000h
223	" WATCHDOG #1 "	6010h
224	" AUX COIL SHORTED"	3222h
227	" WATCHDOG #2 "	6011h
229	"SAFETY INPUT"	FFE5h
230	" MC COIL SHORTED"	2252h
231	" COIL SHORTED HW KO"	2250h
232	" KEY OFF SHORT "	5114h
233	" POWER MOS SHORTED "	FFE9h
235	"EMERGENCY"	FFh
237	"ANALOG INPUT"	FFEDh
238	"WRONG 0 VOLTAGE"	FFEEh
239	"SAFETY OUTPUT"	FFEFh
240	"HARDWARE FAULT"	FFF0h
241	"FLASH CHECKSUM"	FFF1h
242	"ENCODER LOCKED"	7311h
244	"SOFTWARE ERROR"	FFF4h
245	"WRONG RAM ADDRESS"	FFF5h
248	" CAN BUS KO "	8130h
251	"WRONG SET BATTERY"	FFFBh
254	"AUX OUTPUT KO"	FFFEh

ZAPI code (dec)	ZAPI string	Warning Code
0	" NONE "	0000h
13	" EEPROM KO "	5530h
62	"TH. PROTECTION"	FF3Eh
65	"MOTOR TEMPERATURE"	4310h
80	"FORWARD+BACKWARD"	FF50H
236	"CURRENT GAIN"	FFECh
243	"SENS MOT TEMP KO "	FFF3h
250	"THERMIC SENS. KO "	4211h
253	"SLIP PROFILE"	FFDA

(***) only for Atech