CODEN:LUTEDX/(TEIE-3049)/1-88/(2015)

Industrial Electrical Engineering and Automation

Automatiserat transportsystem

-på Software Solutions AB

Jonathan Almström
Richard Andersen

Division of Industrial Electrical Engineering and Automation
Faculty of Engineering, Lund University

Automatiserat transportsystem

- på Software Solutions AB

LUNDS UNIVERSITET
Lunds Tekniska Högskola

LTH Ingenjörshögskolan vid Campus Helsingborg
Elektroteknik med automationsteknik
Avdelningen för Industriell elektroteknik och automation

Examensarbete:
Jonathan Almström
Richard Andersen

LTH Ingenjörshögskolan vid Campus Helsingborg
Lunds universitet
Box 882
251 08 Helsingborg

LTH School of Engineering
Lund University
Box 882
SE-251 08 Helsingborg
Sweden

Tryckt i Sverige
Avdelningen för Industriell Elektroteknik och Automation
Lunds universitet
Lund 2015

Sammanfattning

Examensarbetet grundar sig på uppdragsgivarens problemformulering som handlar om att realisera ett automatiserat transportsystem med hjälp av ett SCADA-system och transportskenor.

Rapporten består av en teoretisk del som förklarar SCADA och de olika delar som bygger upp systemet och en praktisk del som innehåller tillvägagångsättet för uppkoppling av SCADA, kommunikationen mellan enheterna och programmering av PLC, HMI samt transportbandens drivsteg.

Avslutningsvis i rapporten redogörs närmare för programmen WinCC och STEP-7 och tillvägagångsättet för kommunikationen och integrationen med varandra.

Nyckelord: PLC, SCADA, HMI, WinCC, STEP-7

Abstract

The thesis is based on a problem formulated by the company to implement and realize an automated transportation system utilizing a SCADA system and transportation rails.

The report contains a theoretical part that explains the SCADA system and it's components, and a practical part that explains how the SCADA system is connected, the communication between different units and the programming of the PLC, HMI and the transportation rails drives.

Finally the report takes a closer look at the softwares WinCC and STEP-7 and the way they communicate and integrate with each other.

Keywords: PLC, SCADA, HMI, WinCC, STEP-7

Förord

Examensarbetet planerades höstterminen 2014 på Campus Helsingborg och utfördes sedan vårterminen 2015 på Software Solutions AB i Lomma som avslutande del av utbildningen, högskoleingenjör inom elektroteknik med automation.

Ett stort tack till Conrad Benatti och Carlos Olave på Software Solutions AB för att de ville ta emot oss och för all hjälp vi fått på vägen. Vi vill även tacka Johan Björnstedt och Mats Lilja som tog sig an rollen som handledare respektive examinator till vårt arbete. Vi vill även rikta ett tack till Fredrik Stråhle på Bosch Rexroth AB i Stockholm för den välbehövliga hjälp vi fick när drivstegen inte ville samma sak som vi ville.

Terminologi

PLC	Programmable logic controller, programmerbart styrsystem.
HMI	Human machine interface, interaktivt användargränssnitt mellan maskin och människa.
SCADA	Supervisory control and data acquisition, övergripande system som kontrollerar och styr processer.
PROFIBUS	Kommunikationsstandard för fältbussar inom automation.
FB	Funktionsblock, block med in- och utgångar inom PLC-programmering, parametrar till datablock.
FC	Funktion, block utan styrda ingångar, enbart aktivering och avaktivering, utan datablock.
DB	Datablock, lagring av information mellan noder.
OB	Organisation block, cykliskt block där funktioner lagras för att köras i sekvens.
I/O	Input/output, beskriver enhet som kan hantera in- och utsignaler.
(D)INT	(Double) Integer, (32) 16-bitars heltal, representation i decimalform.
(D)WORD	(Double) Word, (32) 16-bitars sträng, representation i decimal-, hexadecimal- och binärform.
BOOL	Boolean, 1-bit, 0 eller 1, TRUE eller FALSE
IP-ADRESS	Internet protocol address, ett nummer som används som adress för kommunikation mellan enheter.
ETHERNET	Ethernet är ett protokoll i länklagret på OSI-modellen för kommunikation mellan datorer.
W(L)AN	Wireless (local) area network är trådlös kommunikation i ett lokalt nätverk.
LADDER-KOD	En programmeringsform inom PLC med grafisk feedback.
ST	Strukturerad text, en textbaserad programmeringsform inom PLC
RELÄ	En elektrisk till- och frånslagsenhet för styrning av andra elektriska kretsar.
OSI	Open system interconnection, en modell som representerar 7 lager med protokoll för hela systemet i ett nätverk.
IMPEDANS	Motståndet som bildas i en växelströmskrets, summan av resistans och reaktans.

EMC	Elektromagnetisk kompabilitet, förmågan för ett system att fungera i en elektromagnetisk omgivning utan att själv ge störningar till andra system.
GSD-FIL	General station description, fil med drivrutiner för PROFIBUS till hårdvarukonfiguration i PLC-miljö.
BAUD	Måttenhet för hastighet i bitar/sekund som används vid seriell överföring.

Innehållsförteckning

1 Inledning

Här tas initieringen och planeringen för examensarbetet upp, om företaget, vilka system som användes under arbetets gång och vad syftet med projektet var.

1.1 Om företaget

Software Solutions AB startades 1998 med Conrad Benatti som huvudman. Software Solutions köper in nya och begagnade robotar av alla olika märken för att sedan skräddarsy en lösning för kunds räkning, allt ifrån en robot till hela lösningar med styrskåp, redo att användas för ett specifikt ändamål. Då det köps in begagnade robotar och styrsystem som fyller samma funktion som nyinköpta produkter kan kostnadseffektiviteten bli väldigt hög samtidigt som det inte tummas på kvalitén.

Carlos Olave som är en av Conrads anställda jobbar som konsult när företag får problem med robotar eller andra delar av sina process-system men bygger även upp helhetslösningarna i deras lokal i Lomma. Carlos har en väldigt hög kompetens inom allt som rör automation, el och säkerhet och är inte inriktad på ett system utan jobbar med alla stora system som finns på marknaden.

1.2 Bakgrund

I dagens samhälle blir automatisering en allt större del av vardagen. Det kan röra sig om allt ifrån övervakning av temperaturer i fastigheter till att plocka varor från band, vakuumförpacka mat o.s.v.

Det senaste decennierna har industriell automatisering formligen exploderat och nu börjar allt fler mindre företag också intressera sig för att göra sina processer mer effektiva, eftersom det har börjat bli mer ekonomiskt försvarbart och för att kunna hålla den höga standard som ett automatiserat system måste uppfylla.

Det som lockade med detta projekt var styrning av motorer med diverse givare för att kunna ha kontroll över hela processen. Allt detta ska självklart kunna göras i realtid och med en bra översikt på hela systemet.

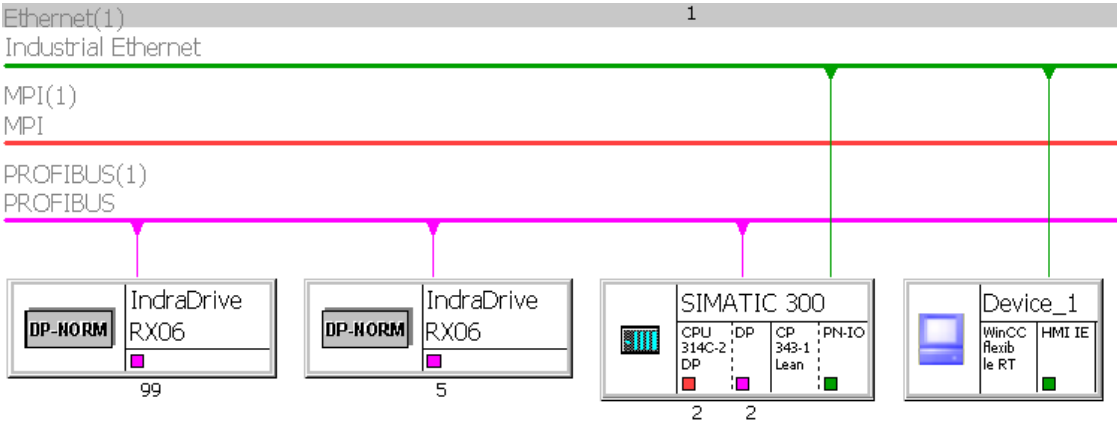
För att kunna göra det möjligt att övervaka systemen på ett smidigt sätt fanns HMI till hjälp, HMI står för ”Human-machine interface” och är ett grafiskt gränssnitt på en touchpanel som är kontakten mellan alla delar i processen och

användaren. Touchpanelen konfigureras med parametrar som sedan kopplas ihop med en styrkrets som jobbar med in- och utgångar, en PLC.

PLC står för ”Programmable logic controller” och används för att styra processen. Den kan liknas vid en hjärna som skickar signaler till de olika delarna i systemet.

PLC:n kopplas ihop med HMI:t och interagerar med varandra för att sedan kunna kommunicera under arbetets gång. Vid styrningen av elmotorerna används drivsteg som är speciellt framtagna för att driva servomotorerna, dessa interagerar också med PLC:n och HMI:t i Siemens STEP-7 mjukvara.

I Fig.1 visas hur dessa är sammanlänkade samt vilken typ av fältbuss som användes (PROFIBUS). Systemet kan sägas utgöra ett mindre SCADA-system.



Figur 1 - Hårdvarukonfiguration

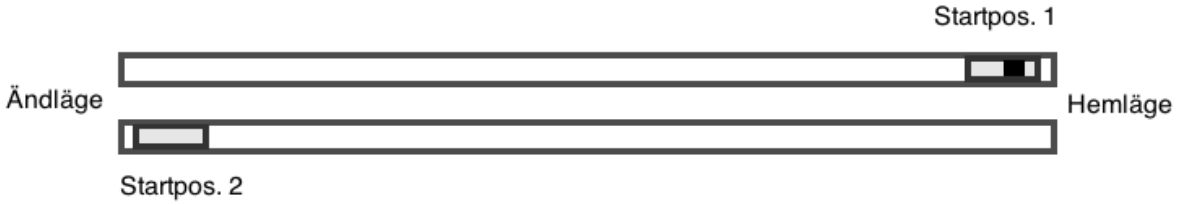
1.3 Syfte

Syftet med examensarbetet har varit att få en praktisk uppfattning av hur man bygger upp ett helt system med alla dess delar som PLC, HMI, drivsteg, elmotorer samt uppkoppling av dessa system.
Vi lärde oss även hur man programmerar dessa och vad som krävdes i de olika momenten, vad som var viktigt att tänka på och hur man rent praktiskt utförde de olika momenten.

1.4 Problemformulering

Examensarbetet som var förlagt på Software Solutions AB har inneburit att ta fram en helhetslösning rörande två stycken transportband som skulle röra sig parallellt med varandra med hjälp av ett HMI. Idén var att (se Fig. 2):

- Transportskena 1 och 2 placerades i startposition och transportskena 1 skulle förflytta sig från hem- till ändläge.
- Transportskena 2 väntade i ändläget och skulle börja röra sig till hemläge när transportskena 1 var framme i ändläget. Här skulle en robot, i mån av tid, sättas in för att förflytta en enhet som låg på transportskena 1 till transportskena 2.
- Samtidigt som transportskena 2 åkte till hemläge skulle även transportband 1 åka till hemläge och då tillbaka till sin startposition.
- När transportskena 2 nått hemläge och den förflyttade enheten var i slutposition, skulle transportskena 2 åka tillbaka till ändläget och då tillbaka till sin startposition.
- Sedan skulle proceduren upprepas tills avstängning av programmet.



Figur 2 - Transportbandens uppställning

För att få det här systemet att fungera så krävdes kunskap inom PLC-programmering, HMI-system, drivstegshantering samt grundläggande kunskap inom ellära.

1.5 Avgränsningar

Det planerades att göra allting som rörde systemet, uppkoppling, konfigurering samt hanteringen av de problem som uppkom. När arbetet kring transportskenorna var färdigt skulle även en robot programmeras till att vara en del i processen och förflytta ett föremål mellan de olika transportskenorna, denna var dock inte en direkt del av examensarbetet och är därför inte beskriven så detaljerat som det övriga arbetet.

1.6 Källkritik

Många av källorna kom direkt ifrån Bosch Rexroth och Siemens, antingen från deras hemsida eller från deras programs hjälp-länkar och var därför väldigt trovärdiga källor då de kom direkt ifrån tillverkaren. Resterande källor var samkörda med egna erfarenheter och kunskaper och bedöms vara trovärdiga.

Då projektet konstruerades med äldre produkter blev en hel del tid lagd på att hitta rätt dokument. Även viss mailkorrespondans direkt med tillverkarna var behövlig.

2 Teknisk bakgrund

2.1 Inledning

Här beskrivs en teknisk bakgrund till de olika systemen och begreppen som ingick i arbetet.

2.2 SCADA

Ett SCADA-system används för att styra, övervaka och hantera information från automatiserade system. Ett typiskt SCADA-system är uppbyggt av flera olika komponenter, vanligtvis består det av ett HMI (interface för övervakning och styrning), en PLC (informationshanterare), trådlösa överföringspunkter mellan sensorer och informationshanterare samt även centrala punkter för att sammanlänka alla de olika delarna så att en kommunikation blir möjlig som t.ex. lokala nätverk inom en industri eller WAN om det är större system. [1]

SCADA-systemet jobbar med kodade signaler över kommunikationskanaler och överför snabbt information mellan de olika komponenterna i systemet. Det vanligaste sättet att transportera signalerna i dagsläget är via ethernet (nätverk), dels för att det är en säker överföring men även för att det blir möjligt att transportera signalerna väldigt långa sträckor. [2]

En redan befintlig process kan göras effektivare och säkrare genom att integrera ett SCADA-system som då gör att man får en bra överblick över hela systemet och kan övervaka och styra alla delar utan att behöva befinna sig på de olika ställena. Systemet kan varna vid olika faror eller fel och kan även programmeras att agera utifrån vad för fel som uppkommer.

Ett SCADA-system kan hantera säkerheten kring en produktionslinje där direkt fara uppkommer om någon skulle ta sig in till säkerhetszonen kring processen under tiden den är igång. Om en grind då öppnas drar systemet slutsatsen att någon är på väg in på området och kan då göra ett tvärstopp i hela systemet vilket förhindrar personskador.

I och med att hela processen är sammankopplad kan man lätt återgå till föregående tillstånd när man har klivit ut från området och sagt till systemet att det nu är säkert att köra. Förslagsvis kan processen bara startas igen via HMI:t och med hjälp av en kontrollkod som bara en ansvarig har och kan matas in direkt på manöverpanelen.

SCADA-systemen utmärker sig från andra liknande system genom dess förmåga att hantera stora och geografiskt utspridda system.
T.ex. återfinns det i stora delar av vår infrastruktur, i tåg och trafikstyrning med trafikljus m.m.

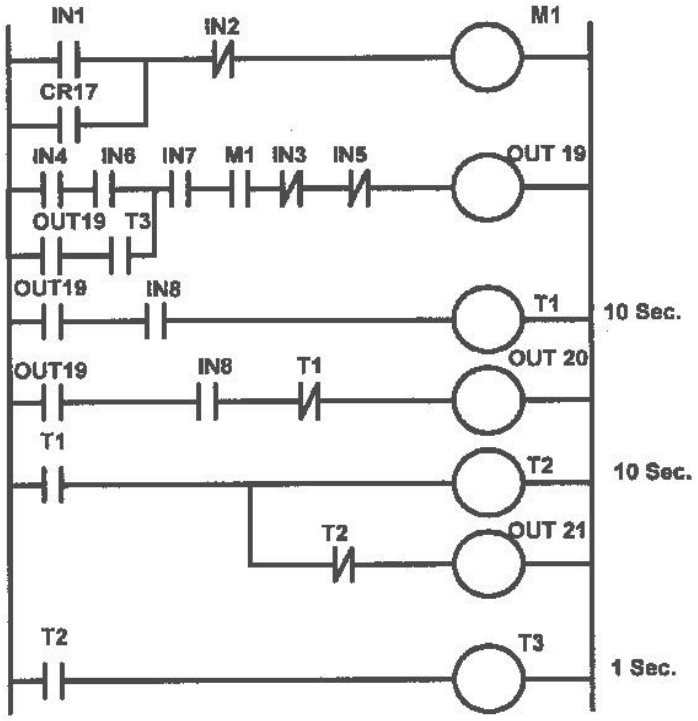
2.3 PLC

[3] PLC är ett programmerbart styrsystem som främst används inom automation och kan via I/O-moduler ta in både digitala och analoga signaler som den sedan kan behandla. PLC:ns är bl.a. uppbyggd av en mikrokontroller och ett programmerbart minne. Detta minne används för att lagra instruktioner om hur den ska hantera de olika signalerna när de inkommer och vad de ska resultera i för operation.

PLC:n jobbar med en logisk struktur och samma program kan skrivas i tre olika språk, varav det ena språket kallas ”Ladder” (se Fig. 3). PLC:ns logiska struktur bygger på att man anger en ingång som den ska övervaka, när denna ingång får en signal in ska en utgång aktiveras. Detta kan göras väldigt komplext i form av att flera ingångar måste vara aktiverade för att en utgång ska aktiveras.

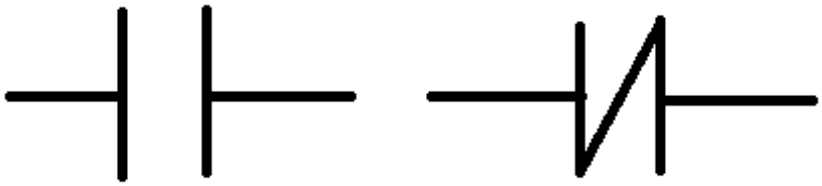
Som exempel kan man titta på en hiss och hur den reagerar utifrån på knapptryckning.
Vid begäran av hiss från första våningen till valfri våning, då hissen befinner sig på 5:e våningen och är på väg ner för hämtning, är att om ytterligare en begäran att åka ner från våning 3 finns ska hissen stanna på den tredje våningen först innan den åker vidare till första våningen.
Om begäran var att åka upp från tredje våningen stannar inte hissen på den våningen utan åker direkt till första våningen. Detta måste systemet veta om och implementeras med kod där det anges att om hissen är på väg ner kan inte hissen stanna om inte någon vill åka i hissens färdriktning.

Ett enkelt ladderdiagram visas i Fig. 3:



Figur 3 - Ladderschema

Den översta raden i schemat visar att antingen ska ”IN1”eller ”CR17” vara aktiv för att signalen ska kunna gå vidare, därefter får inte ”IN2” vara aktiv då det är en så kallad inversgrind, som hela tiden ger ett resultat som är motsatsen till sitt egentliga värde (se Fig. 4). Om dessa villkor uppfylls kommer utgången ”M1” att bli aktiv.



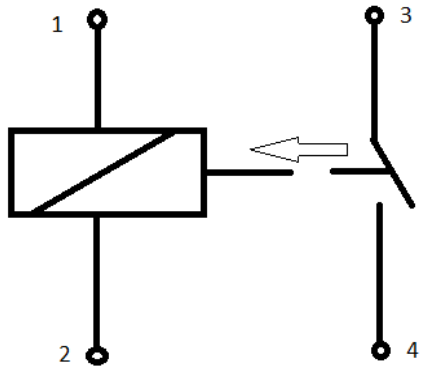
Figur 4 - Logiska grindar, t.v. normalt öppen, t.h. normalt stängd

Ladderschemat i Fig. 3 avläses från vänster där samtliga insignaler finns, till höger där cirkeln markerar utsignalen. Utsignaler används även väldigt ofta som insignal, d.v.s. när en utgång blir aktiv som resultat av händelserna till vänster kan denna vara starten till en ny reaktion.

PLC:n kan bl.a. hantera temperatur ”T” av en motor som körs, blir den för varm kan en fläkt startas som ger extra kylning, eller kan motorn stängas av ifall det är en överbelastning som den inte kan hantera. Det är även vanligt att hastighet, position och liknande värden avläses.

När man först började ta fram en PLC var målet att ersätta tekniken som redan fanns, denna var baserad på reläer som var utformade att när en ingång fick signal skulle en annan också aktiveras.

Om ingång 1 i Fig. 5 blir aktiv sluts kretsen med nod 2. Då dras reläet (i pilens riktning) och ingång 3 sluts och får kontakt med utgång 4 som också blir en sluten krets. Det var den här tekniken man ville göra bättre, det gjordes med en PLC som har samma funktion.



Figur 5 – SPST- (single pole single throw) relä

Fördelen med en PLC jämfört med reläer är att man kan själv välja hur dessa kopplingar ska göras då allting styrs med en dator, till skillnad från ett relä som alltid kommer ha samma funktion då det är fördragna ledningar som är svårare att efterkonfigurera.

2.4 HMI

Ett HMI gör det möjligt att i realtid övervaka och förändra värden på de olika variablerna i processen.
HMI:t ger en grafisk överblick på en skärm med touchfunktion som gör att man kan peka på skärmen vid val av de olika inställningarna och för att navigera sig genom menyerna. [4]

Vid programmering av Siemens HMI används Simatic WinCC flexible där man direkt ser hur menyerna ser ut och de ändringar man gör.
Man kan sedan koppla variabler till de olika knapparna för att kunna förändra deras värden. Variablerna kan sedan kopplas till t.ex. en PLC. En integreringfunktion finns inbyggd i programmet för integration till PLC:ns kod. Touchdisplayerna varierar i storlek från ca 4” hela vägen upp till över 20”.

2.5 I/O-enheter

I/O är förkortning på Input/Output och anger om en enhet fungerar som mottagare (INPUT) av signaler och/eller sändare (OUTPUT) av signaler.

Signalerna motsvaras av en elektrisk ström där man kommunicerar med 0 och 1 (av och på). Vanligt strömintervall ligger mellan 0-20 mA. [3]
Projektets I/O-enheter innefattades av HMI, PLC och drivsteg till servomotorerna som kopplades samman med Ethernet- och PROFIBUS-kablar.

2.6 Kommunikation

I det här kapitlet behandlas de kommunikationsprotokoll som använts.

2.6.1 Industrial ethernet

Till projektet användes industrial ethernet (IE) som blir mer och mer vanligt, främst för dess förmåga att skicka kodade signaler över nätverk vilket gör det möjligt att styra processer över långa sträckor. Det gör även att man kan ha en central som hanterar informationen mellan hårdvara av olika slag.
I ett mindre kommunikationssystem där transport över långa sträckor inte är nödvändig används vanliga nätverkskablar som är sammanlänkade med en switch/router, precis som i ett hushåll.

En annan fördel med IE är att den är väldigt tålig mot yttre åverkan.
I en industri måste allting vara dimensionerat för att kunna hantera bl.a. stora krafter, yttre störningar och fukt. Detta utan att information går förlorad eller att det blir avbrott i kommunikationen.
Vid användning av fiber-ethernet isolerar man sig även ifrån elektriska störningar då fiber-ethernet använder sig av fiberoptik som är ljusstrålar istället för elektriska signaler.
Ethernet användes mellan touchdisplayen och PLC:n då utrustningen hade stöd för detta, det gjorde även att integreringen mellan dessa var väldigt enkel i och med att programmen hade funktionerna inbyggda för denna typ av användning.

2.6.2 PROFIBUS

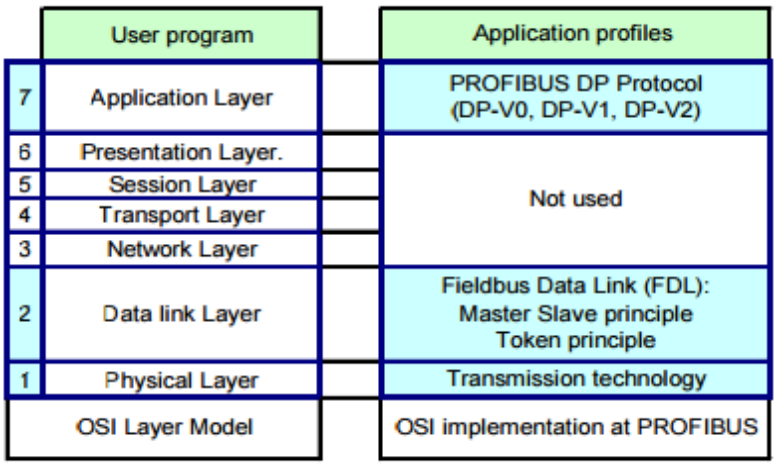
PROFIBUS som är nästa typ av kommunikation som kom att användas i arbetet är en standard som fortfarande är vanlig, men som håller på att fasas ut till fördel för ethernet.

PROFIBUS är uppbyggd på OSI:s lagermodell (open systems interconnection reference model). OSI är en standard för kommunikation mellan två noder och går över sju olika lager, från lager 1 som är det fysiska lagret till lager 7 som är applikationslagret.
PROFIBUS använder dock bara lager 1,2 och 7 och dessa lager definieras som:

Lager 1: Fysiska lagret, som står för överföringen från nod till nod. Där kan det användas kopparkabel, optisk kabel eller trådlös överföring.

Lager 2: Beskriver bus-accessen och inkluderar datasäkerhet. Lager 2 definierar också att det är en master-slave metod som används, d.v.s. att mastern skickar styrsignal som slave-noden tar emot.

Lager 7: Är interfacet till applikationen och representerar länken mellan applikationen och kommunikationsnoden.



Figur 6 - Profibuslagerna

Hårdvara som använder PROFIBUS för kommunikation använder sig av PROFIBUS DP (Decentralized Periphery) protokollet, vilket är samma för alla applikationer som tillåter cyklisk eller icke-cyklisk kommunikation.

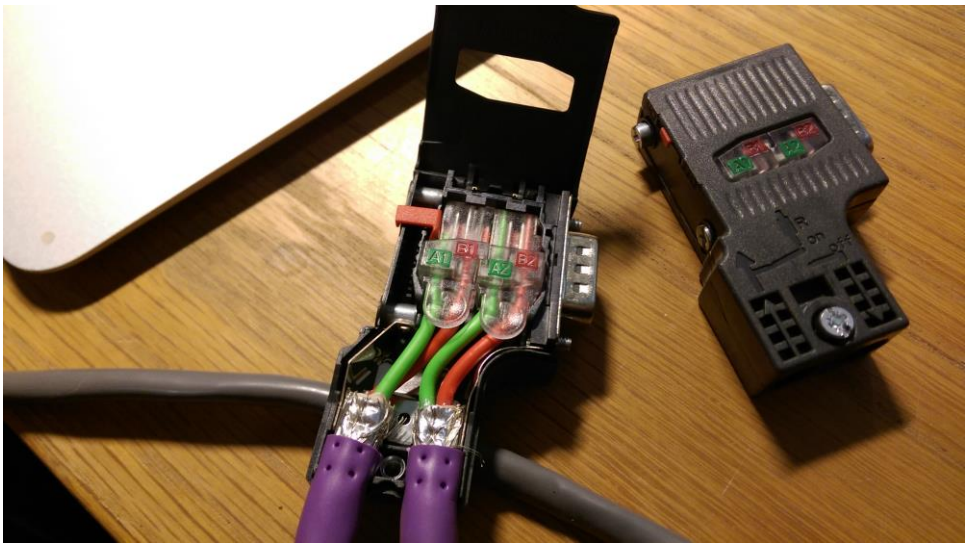
Grunden för kommunikationen bygger på master-slave metoden där mastern, som är en aktiv nod, påtvingar den påkopplade slaven, som är en passiv nod, att utbyta data. Mastern skickar ut en begäran om vad den vill få reda på (t.ex. hastigheten på en motor) och slaven svarar sedan på denna begäran, den senaste informationen från t.ex. en sensor. En cykel är avslutad först när mastern har fått svar från slaven.

PROFIBUS kan även användas för att konfigurera hårdvara, då med icke-cyklisk kommunikation. Det kan även vara mer än en master i ett system, d.v.s. det kan vara flera noder som begär ut information från slavarna. [5] Man kan använda olika sätt att transportera PROFIBUS-signalerna. Den metoden som användes i projektet var en vanlig dubbeltrådig elektrisk överföring.

I Fig. 7 syns två stycken dubbeltrådiga kopparkablar (det lila ytterhöljet innehåller två kablar, en grön och en röd). Det ena trådparet är för inkommande signaler och det andra är ifall man kopplar på mer hårdvara kan

den vara en del på samma ”lina”, det vill säga man slipper dra en separat ledning mellan centralen och varje enskild komponent.

Dessa kablar har en resistans på ca 150 ohm och jobbar i hastigheter mellan 9kbit/s upp till 12 Mbit/s vilket är fullt tillräckligt för överföringen som ska skickas.



Figur 7 – PROFIBUS-kabel

Nackdelen med denna teknik är att signalen måste förstärkas om den transporteras längre sträckor. Redan efter 1200-1900 meter behöver det vanligen sitta en förstärkare [5], och sedan kontinuerligt för att bibehålla signalens kvalitet.

Fiber förekommer också och kan skicka en signal 10 gånger så långt som en vanlig kopparkabel utan en förstärkare, men är i sin tur känsligare mot yttre åverkan.

2.6.3 Seriellt interface

En seriell port existerar i flera olika former, från datorns sida brukar den benämnas ”COM-port” (se Fig. 8). Andra standarder som också använder sig av en seriell ström när de skickar data är USB, firewire och ethernet, dock brukar man, när man talar om seriella portar tala om COM-porten, eller portar som är mer eller mindre kompatibla med RS-232 och RS-485 standarden.

Figur 8 - COM-port

En seriell anslutning skickar eller tar emot en bit i taget till skillnad från en parallell port som kan ha strömmar både in och ut samtidigt.
PIN-konfigurationen i en seriell port ska följa en standard. Denna standard är för högst 25 pinnar, dock används sällan så många. [6]
Vid uppstart skapas en asynkron kommunikation där inställningar görs, bl.a. för hastighet och antalet bitar som kommer att skickas per gång.

Fördelen med seriell kommunikation är att den kräver ytterst lite från mjukvaran hos värden, vilket gör att den är lätt att implementera då ingen hänsyn behöver tas från gästen.

De kontakter som användes till projektet som utnyttjar seriell kommunikation var COM-porten (sladd med USB och COM-port) samt en Mini DIN-8 kontakt. Mini DIN-kontakten finns med olika antal pinnar. För att inte en kontakt med fler eller färre till antalet pinnar av misstag ska användas är pinnarna utplacerade på ett sätt som gör att detta inte är möjligt.

2.7 Drivsteg

Drivsteg används i många olika applikationer inom industrier för att driva elmotorer.
Drivstegen konfigureras utefter vad för motor som avses användas samt hur den ska användas. Efter att konfigureringen är gjord integreras denna med PLC:n för att kunna ta emot direktiv därifrån.

Vid en konfigurering ställer man in vilken motor som används, detta för att drivsteget ska veta hur mycket spänning motorn kräver och vad motorn har för övriga specifikationer. Man ställer även in hur snabbt motorn ska accelerera samt hur hårt motorn ska bromsa in. Detta kan vara viktigt av flera olika anledningar, dels för att lasten motorn ska dra kanske inte klarar av en för snabb acceleration/retardation eller för att motorn inte klara av den höga belastning som skapas.

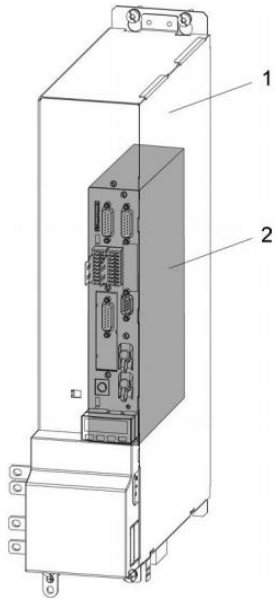
Vid konfigurering ställer man även in om motorn har en slutväxel, d.v.s. en växellåda. Det är av största vikt att drivsteget vet om detta för att kunna

reglera motorns hastighet då man vill ställa in utgående hastighet och inte motorns hastighet i många fall, även för att positioneringen ska bli rätt då den räknar ut position utefter antalet varv motorn har roterat.

I slutet av konfigurationen av drivsteget ställer man in hur den kommer att kommunicera med PLC:n och vilka variabler den kommer att använda vid denna kommunikation. Drivsteget kopplas sedan ihop med elmotorn via en trefasledning.

2.7.1 IndraDrive C

Drivsteget består utav två delar, en strömförsörjningsdel och en kontrolldel (se Fig. 9) [7].

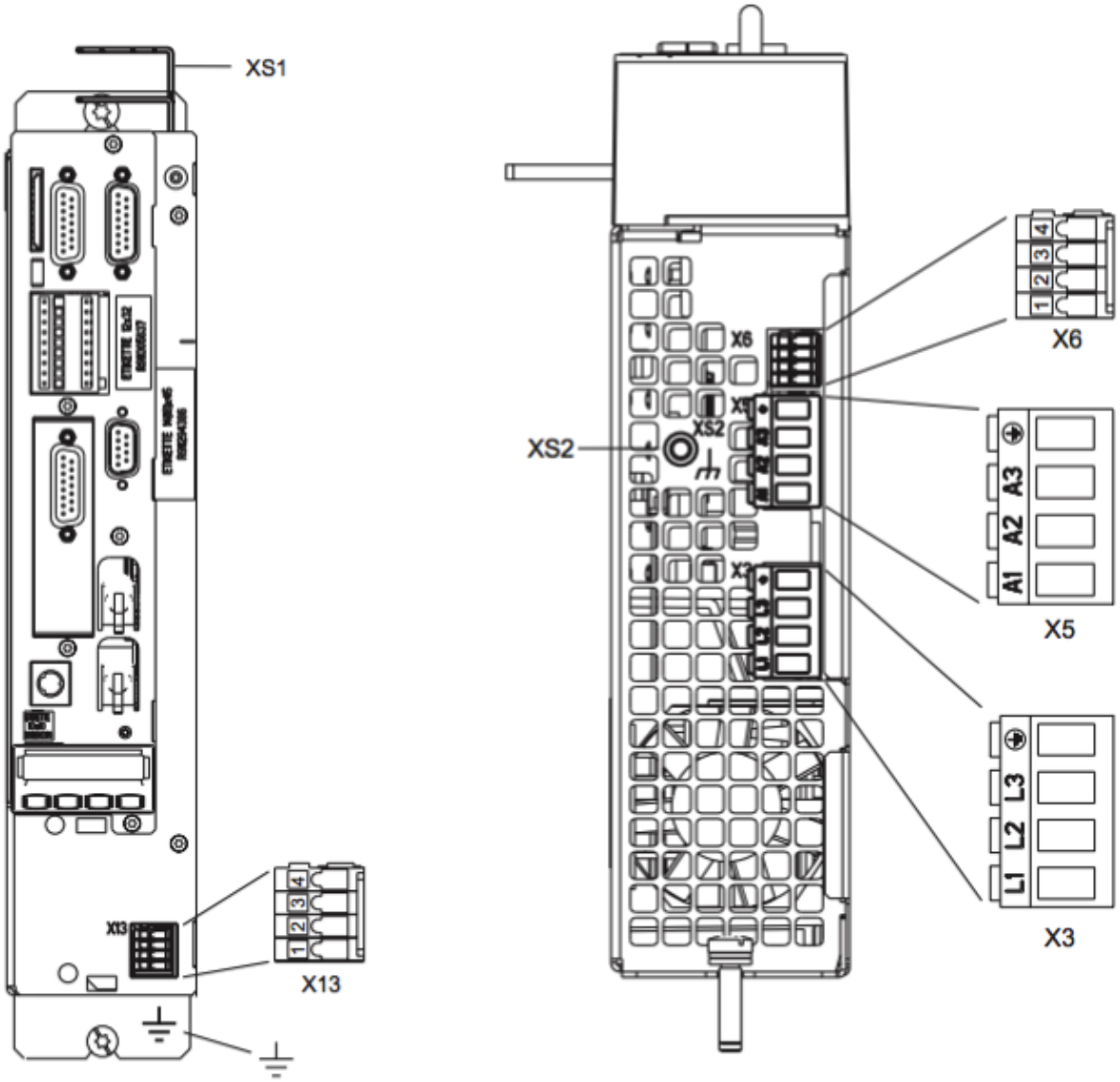


Figur 9 - Uppbyggnaden av drivsteget. Strömförsörjningsdel (1) och kontrolldel (2).

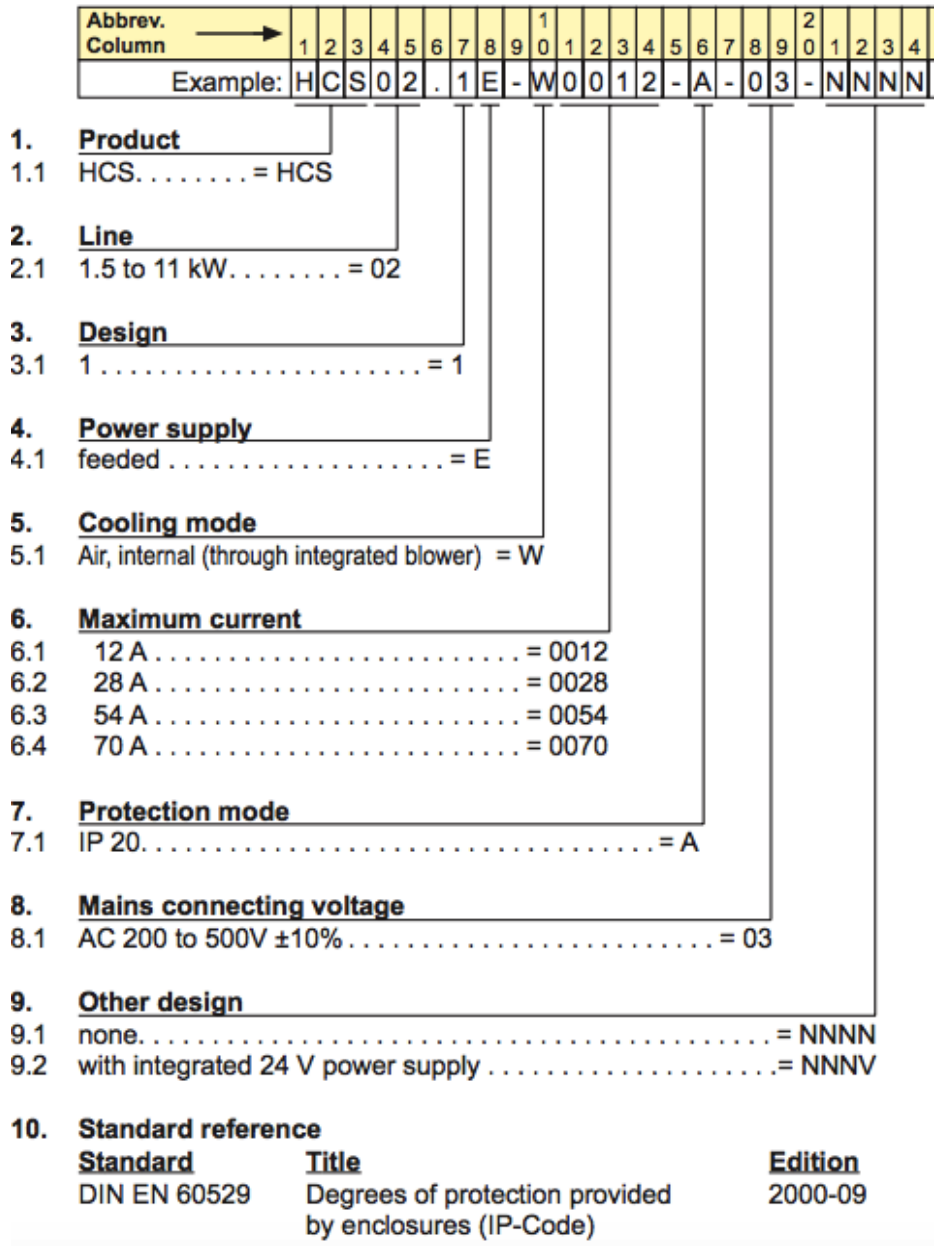
Strömförsörjningsdel – HCS02.1E-W0012-A-03-NNNN

Strömförsörjningsdelen matas med 24V via en transformator till port X13. Trefasspänning matas in i drivsteget på port X3 och ut till motorn från port X5. X6 har hand om övervakning av motortemperatur samt motorbroms (se Figur 10) [8].

Från typskylten i Figur 11 kan man utläsa att denna modell har en maximal ström ut på 12 A och är inte utrustad med integrerad spänningskälla.



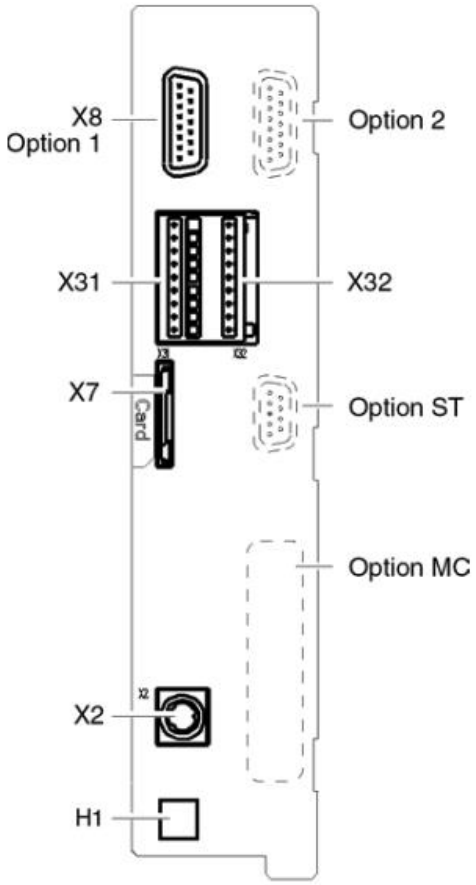
Figur 10 – In- och utgångarna X13, X6, X5 och X3



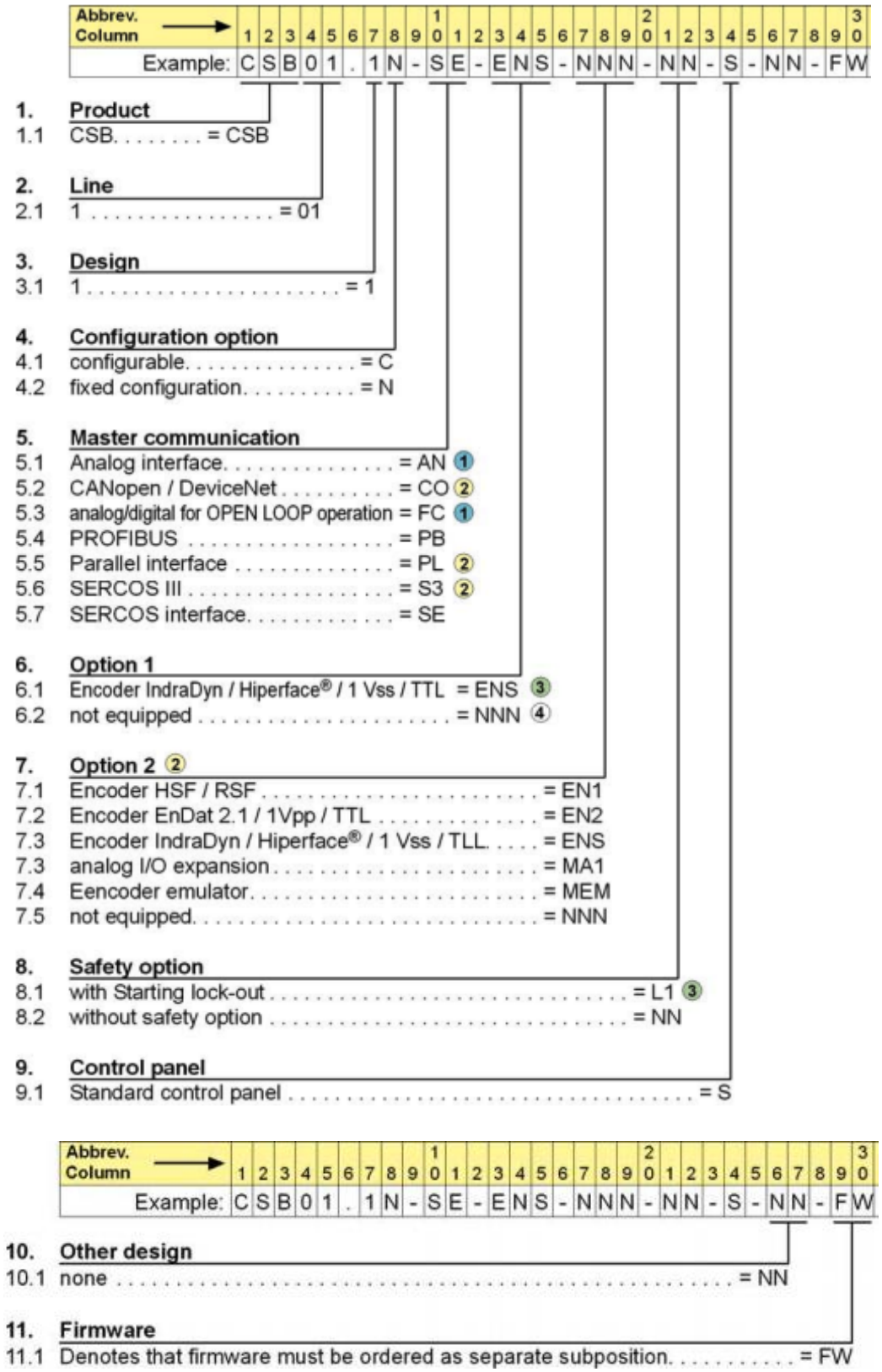
Figur 11 - Typskylt för strömförsörjningsdelen.

Kontrolldel – CBS01.1C-PB-ENS-NNNN-NN-S-NN-FW

Från typskylten i Figur 13 kan man utläsa att modellen använder PROFIBUS för kommunikation mellan drivsteg och PLC-enhet (Option MC i Fig. 12). Option 1-porten (X8) är för kommunikation till synkronmotorerna som driver transportskenorna. X7 är en MMC-kortplats för att spara undan parametrar och konfigurationer så att man lätt kan flytta över samma information till ett annat drivsteg. Platserna X31 och X32 är isolerade digitala och analoga in- och utgångar för koppling med sladd. X2 är Mini DIN-ingången för konfiguration av drivsteget via extern enhet.



Figur 12 – Portbenämningarna på drivstegets kontrollenhet



Figur 13 - Typskylt för kontrolldelen.

.

2.7.2 Drivstegsfilter – NFD03.1-480-016

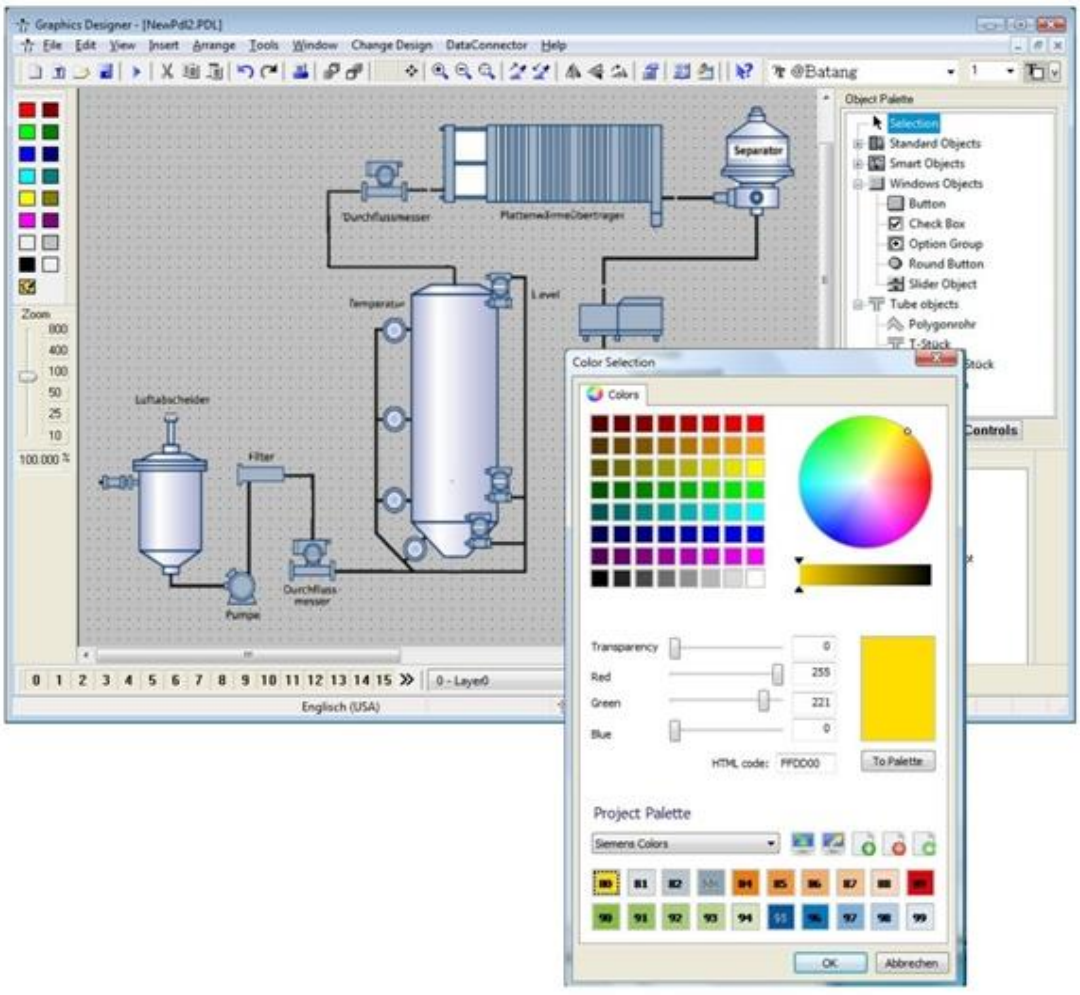
Drivstegsfiltret skyddar mot att störningar skickas ut på nätet enligt EMC standard [9].

2.8 IndraDyn S motorer – MSK040C-0600-NN-M1-UG0-NNNN

Motorerna som används är trefas permanentmagnetiserade synkronmotorer med maximalt varvtal på 7500 min^{-1} och maximalt vridmoment på 8.1 Nm [10].

2.9 WinCC

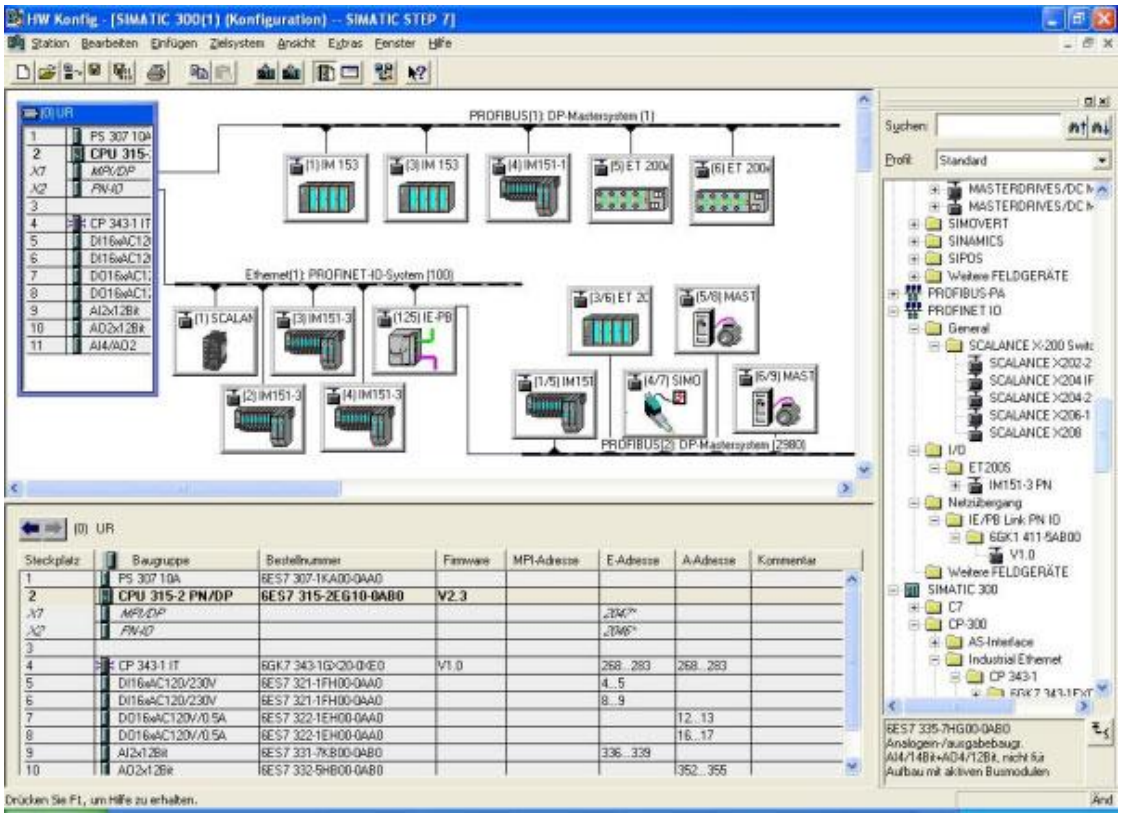
Simatic WinCC är ett program för SCADA- och HMI-system skapat av Siemens för Windows OS. Programmet använder sig av en WYSISYG (What You See Is What You Get) interface med tillhörande simulering så att man ser direkt resultaten på datorskärmen.



Figur 14 – HMI-interface

2.10 STEP-7

STEP-7 är ett program för programmering och hårdvarukonfigurering av SIMATIC S7, SIMATIC C7 och SIMATIC WinAC som är namnen på olika PLC-enheter man använder i ett SCADA-system (se Fig 15)[14]. Programmeringsspråken som används är bl.a. ladderdiagram och strukturerad text.



Figur 15 – Hårdvarukonfiguration I STEP-7

3 Metod

Arbetsmetoden utgick till stor del från programmens egna självinlärningsguider för uppsättning och konfiguration av SCADA-enheterna. Vid uppstarten av projektet hjälpte Carlos Olave till med hårdvaruinställningar i PLC:n samt med kopplingen mellan PLC och HMI. Vidare information inhämtades från forumsidor, pdf-filer, tillverkarnas egna sidor samt videolektioner på Youtube.com.

Till PLC-systemet användes programmet STEP-7 och för HMI-systemet WinCC. Ladderkod användes för att programmera PLC-systemet då det är lättare vid mindre projekt för att få bättre helhetsbild av programmet.

För att programmen skulle fungera dels på en MacBook samt på en PC med Windows 8 eller senare, användes en virtuell kopia av Windows XP som kördes i VMware som är ett virtualiseringsprogram, för att få programmen att exekvera korrekt.

Projektet baserades till största delen på att testa sig fram ”learn by doing” m.h.a. den information som internet hade att tillgå med, dels för att det inte fanns internutbildning på plats och dels för att få lärdom i hur man hittar information på egen hand. Det ansågs fullt tillräckligt för ändåmålet och inget ytterligare tillvägagångsätt för information användes med untantag för mailkorrespondansen med Fredrik Stråhle från Bosch Rexroths supportsida angående drivstegskonfigurationen.

Projektets gång kan beskrivas kort som:

- Starta upp HMI samt skapa interface, leta information i elektroniska handboken.
- Koppla samman HMI och PLC samt upprätta en kommunikation mellan dessa.
- Skapa de första taggarna och testa kommunikation mellan HMI och PLC.
- Starta upp transportskenorna och lösa problemen med drivstegen.
- Skriva PLC-koden.
- Lägga in funktioner i HMI för att styra transportskenorna samt testsekvensen.
- Integrera roboten i sekvensen.

4 Utförande

Det första som skulle göras var att montera PLC:n i ett rack och se över HMI:t så att det var komplett. Sedan skulle kablar dras, ström till vägguttag och nätverkskablar till en switch för kommunikation mellan enheterna. Datorerna kopplades in till switchen och det tilldelades ip-adresser till alla enheter. Systemet startades sedan igång och konfigureringen fortsatte.

Det började med att koppla upp ett HMI och därefter skapades ett interface som utgångspunkt, den första versionen av HMI:t skapades för användning till att styra PLC:n och för att se att kopplingen mellan dessa system fungerade. Därefter gjordes en mer avancerad version av HMI:t för styrning av PLC:n fullt ut som i sin tur kontrollerade drivstegen och elmotorerna.

Drivstegen till servomotorerna konfigurerades, dessa tog bara emot en signal från PLC:n, för att utför simpla åtgärder som att starta motorerna eller att bromsa dem. Konfigurationen bestod av vad för motor som avsågs användas, acceleration samt retardation.

4.1 HMI

Under projektets gång användes en virtuell version av en Windows XP-kopia med WinCC och STEP-7 förinstallerade. För att få en bra start användes programmets ”Project Wizard” där det bland annat ställdes in korrekt touchskärm. Sedan sattes knappar ut och kopplades till att utföra bestämda kommandon.

4.1.1 Konfigurera HMI

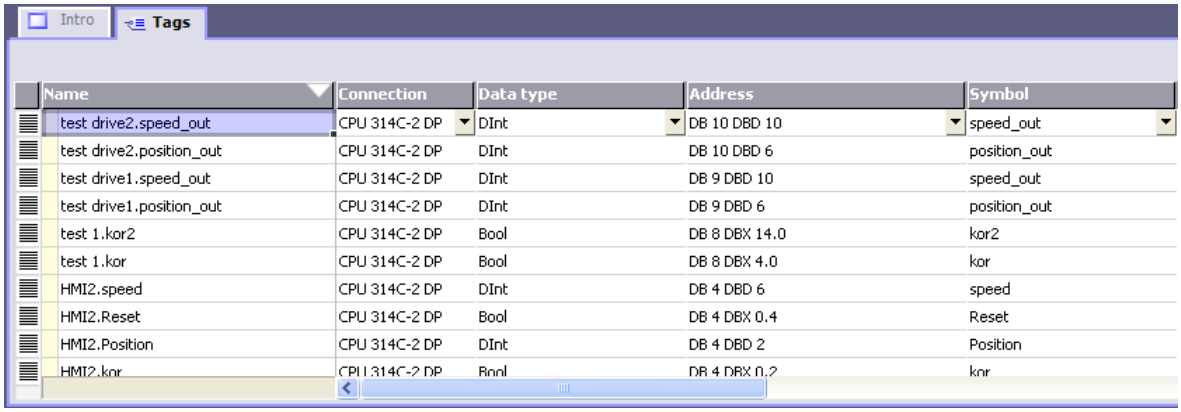
Steg ett var att välja vilken typ av touchdisplay som skulle användas, detta för att WinCC skulle veta upplösning, begränsningar och funktioner som displayen stödjer. I detta fall blev det en KTP1000 basic PN som kommunicerade med ethernet till både datorn för att ladda upp ny programvara och till PLC:n för att skicka styrsignaler.

4.1.2 Komma igång med interfacet

När WinCC hade startats upp med rätt inställd display så kopplades HMI:t in så att det fick ström och kontakt med nätverket, detta skedde med en vanlig cat-5 nätverkskabel.
Efter att det kontrollerats att kommunikationen fungerade skapades olika menyer. En startskärm som kördes igång när man startade HMI:t, en meny där man skulle kunna köra transportskenorna manuellt, en för att kunna köra den automatiserade sekvensen där skenorna kör automatiskt samt en meny för meddelande om någonting skulle gå fel med kommunikationen.

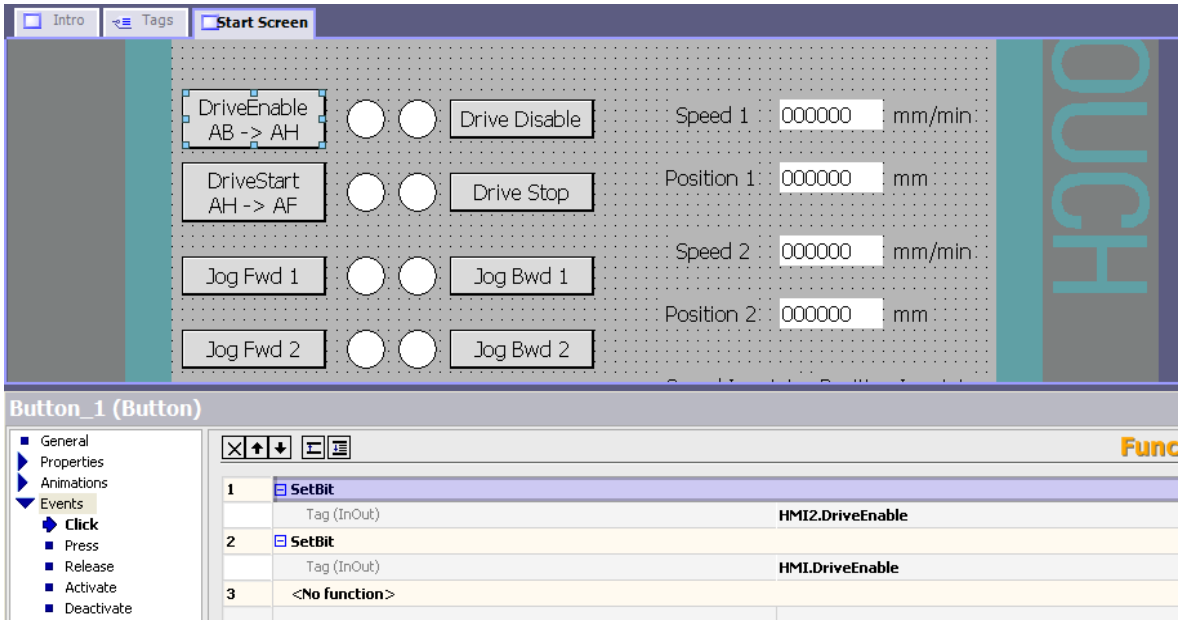
Det fanns en meny som kallades för template. Allting som läggs in i denna meny finns med i alla andra menyer, oberoende av vad som redan finns på dessa. Knapparna för att navigera mellan de olika menyerna placerades därför här, detta för att de ska vara lättåtkomliga oavsett var man befinner sig i HMI:t.

De funktioner som främst användes var ”taggar” och knapparna.
Taggarna är en av de viktigaste funktionerna i hela WinCC då dessa är kopplade till alla olika kommandon och därför lästes det noga på om dessa.



Figur 16 - Taggar i WinCC

I Fig. 16 syns menyn för alla taggar som är kopplade till funktioner i WinCC-projektet. Det är inte bara knappar som har taggar kopplade utan även vid avläsning av hastighet, position o.s.v. behövs det taggar.
All form att kommunikation mellan PLC och HMI sker via taggar.



Figur 17 – Knappen Drive Enable i WinCC

I Fig. 17 syns knappen ”DriveEnable AB -> AH” och vid aktivering sätts bitarna ”HMI2.DriveEnable” samt ”HMI.DriveEnable” höga.

Inga av taggarna fanns från början utan är skapade och konfigurerade vartefter en ny funktion behövdes.

4.1.3 HMI med PLC

När några av taggarna var skapade testades kommunikationen mellan PLC och HMI med hjälp av dessa.
För att kommunikationen ska fungera måste HMI:t veta IP-adressen till PLC:n och vice versa. WinCC har en kolumn där adressen fylls i och även plats i racken för PLC:ns processor. Problem uppstod med kommunikationen mellan HMI:t och PLC:n. Platsen för PLC:ns processor i racken var inte angiven och därför kunde inte HMI:t skicka signalerna rätt.

Synkronisering med STEP-7

Synkronisering av STEP-7 och WinCC gjordes för att variablerna i programmen skulle fungera sinsemellan.
Eftersom taggarna används både i HMI:t och i PLC:n måste dessa finnas med på båda ställena.
Siemens, som har skapat båda programmen har lagt in en funktion som integrerar projekten i WinCC med STEP-7 och efter att de är synkroniserade håller de sig uppdaterade med varandra.

Till en början användes taggarna som skapats i STEP-7, de dedikerades till in- och utgångar från I0.0 → I0.3 och Q0.0 → Q0.3 där I är en ingång och Q är en utgång.
Dessa in- och utgångar användes för att med ladder-kod aktivera en av utgångarna när en av ingångarna blev aktiva, och visualiserades genom att lägga in en grafisk lampa i HMI:t som kopplades till en tag. När en knapp aktiverades skickades en signal till PLC:n (I0.0) som aktiverade en utgång (Q0.0). Den grafiska lampan var inställd på att börja lysa när (Q0.0) blev aktiv, vilket också fungerade.

Att använda taggarna som det först var tänkt, med de fysiska portarnas adresser kunde innebära problem och taggarnas system gjordes om.
Detta då de fysiska portarna i PLC:n är dedikerade till I- och Q-adresser vilket gjorde dessa opraktiska att använda när signalbehandling skedde via ethernet och PROFIBUS, d.v.s. inga av dessa portar användes.
Användning av datablock blev istället aktuellt.

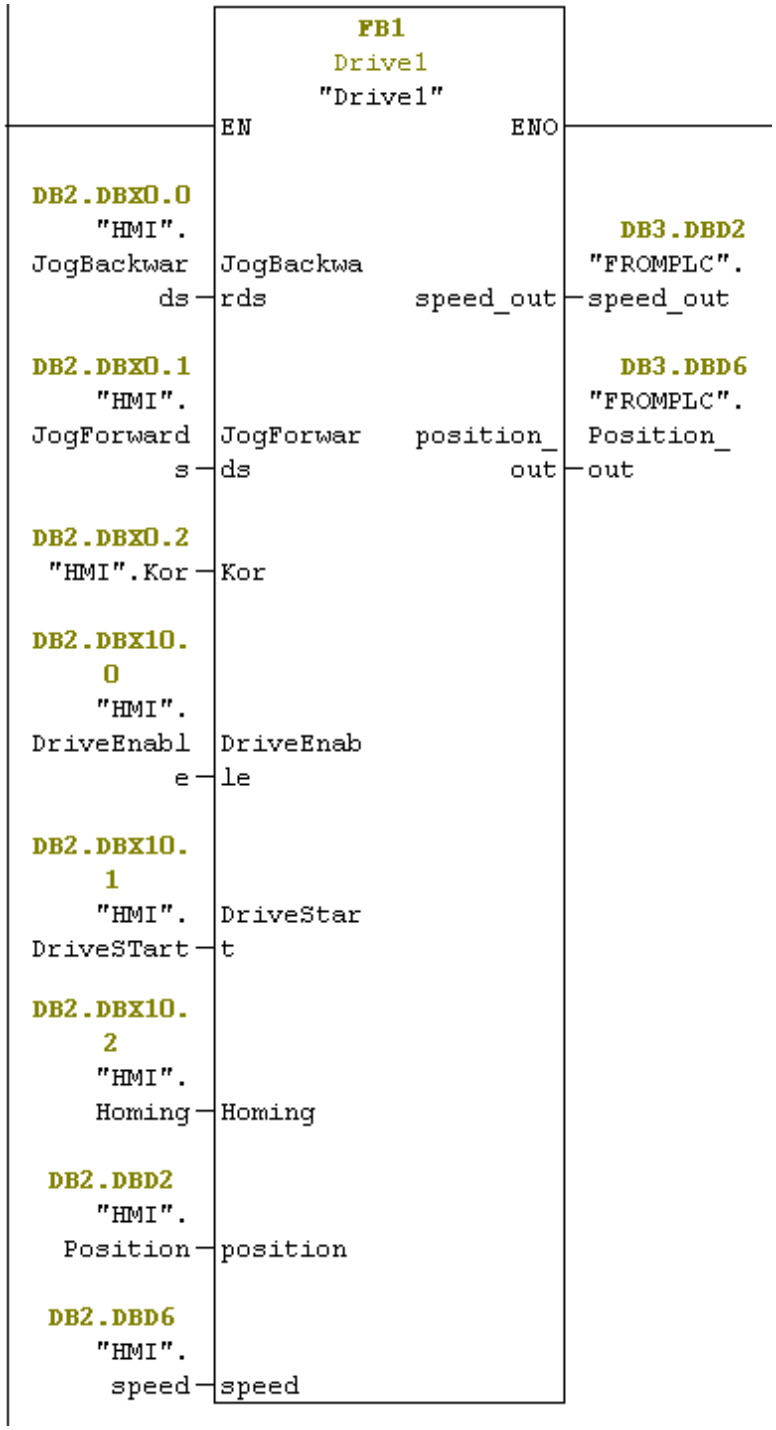
4.2 PLC

PLC-enheterna konfigurerades genom att leta upp rätt enhetsnamn från den fysiska racken och sedan hitta motsvarande namn under ”Hardware” i STEP-7, därefter lägga till dem till programmets virtuella racket så att man får samma motsvarighet mellan den fysiska och den virtuella verkligheten (se Fig. 15).

4.2.1 Skapa funktionsblock

När funktionen för drivstegen kontrollerades användes enkla in- och utgångar som aktiverades för att få motorerna att röra sig. Det insågs rätt snabbt att det skulle ta alldeles för mycket tid och bli alldeles för komplext att kunna arbeta med senare.
För att förenkla infördes ett funktionsblock, vilket försågs med de funktioner som var nödvändiga för att kunna köra motorn.
Det lades till fler funktioner än vad som skulle komma att användas men då funktionerna fanns i drivstegen togs dessa med för att inte behöva göra om arbetet senare om ändringar skulle göras i skenornas uppförande.

Det första funktionsblocket såg ut som följande (se Fig. 18):



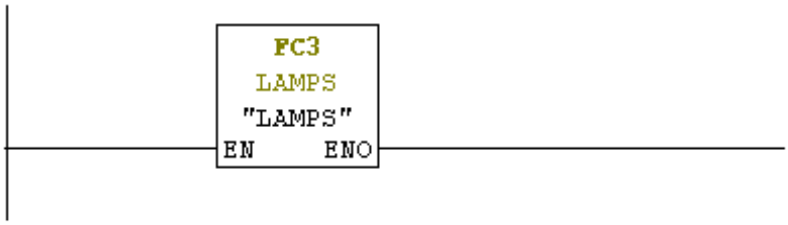
Figur 18 - Funktionsblock

Funktionsblocket fungerade att till höger är utgångar för position och hastighet som avlästes i realtid via drivstegen till transportskenorna och till vänster samtliga ingångar.
Funktionsblocket innehåller ladder-kod där variabler dedikerades till att ligga internt i funktionsblocket och sedan kopplades dessa till en ingång. När man sedan jobbade med funktionsblocket aktiverades en ingång och sedan processades det internt i blocket med ladder-kod.

Funktionsblock användes för all kod som kördes i PLC:n och låg i en fil som hette OB1 (Organization Block). Hade all kod lagts efter varandra hade det blivit onödigt tidsödande att redigera och felsöka denna i efterhand.
För att se hur funktionsblocket ser ut invändigt, se kapitel 7 - Bilagor.

4.2.2 Skapa funktioner

Funktioner skapades, dessa liknade funktionsblocken men med skillnaden att funktioner inte har in- och utgångar.
Funktioner körs utan ingående variabler, d.v.s. körs bara precis som de är och kan inte ha inparametrar som ändrar funktionens sekvens.
En typisk funktion ser ut som följande (se Figur 19):



Figur 19 - Funktion

Skillnaden, som synes är att detta block körs utan påverkan från andra variabler som i funktionsblocket.

4.2.3 Skapa datablock

En stor del i programmeringen av PLC:n rörde datablocken, vars funktion till största del är att hantera mellanlagring av information mellan PLC och HMI.
I funktionskoden användes variabler som kopplades till datablock, detta för att lätt kunna komma åt dem i de olika funktionerna och funktionsblocken.
Två datablock skapades som skulle hantera informationen mellan HMI:t och de två drivstegen och döptes till ”FROMPLC” samt ”FROMPLC2” som då var för drivsteg 1 resp. drivsteg 2.
I dessa datablock låg alla knappar som användes i HMI:t.
När datablocken hade skapats var de olika variablerna från HMI:t tillgängliga och dessa kopplades till de olika knapparna, WinCC lade sedan till dessa taggar automatiskt vartefter de kopplades.

Datablocken behövde inte vara kopplade till knappar utan kunde även, som i det här fallet, lagra hastighet och varvtal som plockades ut från drivstegen. Informationen lästes sedan av i HMI:t för att få en realtidsuppdatering.
I fallen om utläsning av hastighet och varvtal behövdes en omskalning av siffrorna då de kom med för många nollor. Det löstes genom att använda datablocken och med funktioner dividera bort nollorna och sedan sparades det i nya datablock.

Nya datablock skapades efterhand för de nya funktionsblocken och även för ny information som behövde lagras separat. Att lagra allting i samma datablock är väldigt opraktiskt då det lätt blir ostrukturerat och då också svårhanterat.

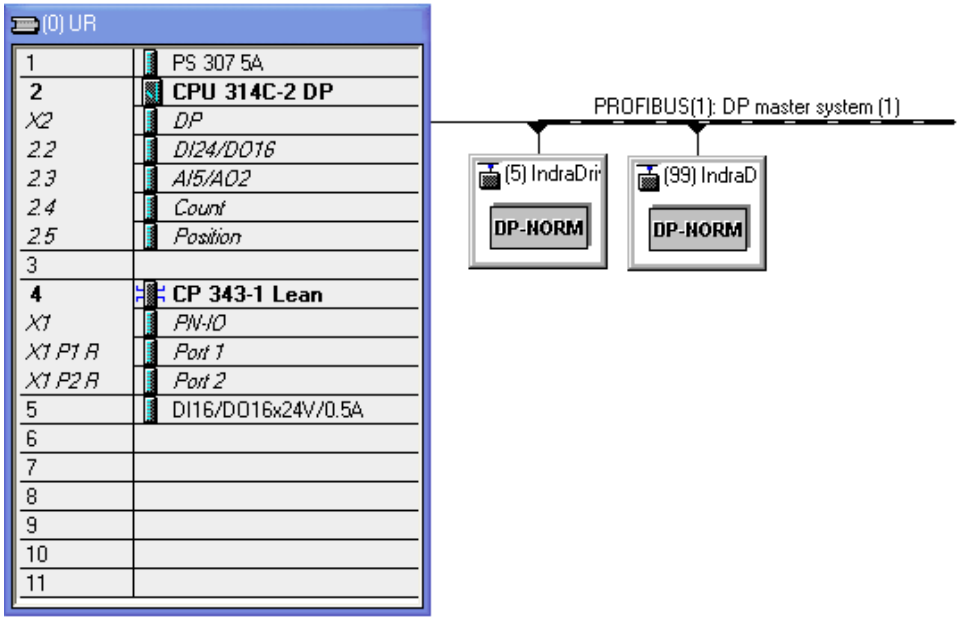
4.2.4 Hårdvara

När en ny PLC sattes upp behövde det göras en hårdvarukonfigurering. Den laddades sedan in i PLC:n för att denne skulle veta vad för hårdvara som användes, dels PLC:ns uppbyggnad då den bestod av en processor och en eller flera I/O-kort och även ethernet-kort.
Det behövdes också anges vad för extern hårdvara som användes, som t.ex. drivstegen och HMI:t, där också adresserna som användes fanns med. Drivstegen, som kommunicerade med PROFIBUS fick adresserna 5, resp. 99.

HMI:t och PLC:ns olika delar fanns redan med i STEP-7 eftersom dessa är gjorda av Siemens båda två, det gjorde dessvärre inte drivstegens drivrutiner. STEP-7 behövde en GSD-fil för drivstegen. Den filen innehåller information, likt drivrutiner för mus och tangentbord som används till datorn och görs av tillverkaren, som är Bosch Rexroth.

STEP-7 tog informationen från filen och lade till drivsteget i menyn som innehöll all hårdvara som programmet kunde hantera.
Drivsteget kunde sedan läggas till rätt kommunikationskanal.

I Fig. 20 syns PLC:n till vänster där CPU 314C-2 DP kör den skrivna koden och CP 343-1 Lean är ethernet-delen.
Till höger syns drivstegen med adresserna inom parenteser och även att dessa var anslutna med PROFIBUS.



Figur 20 - Hårdvarukonfigurering STEP-7

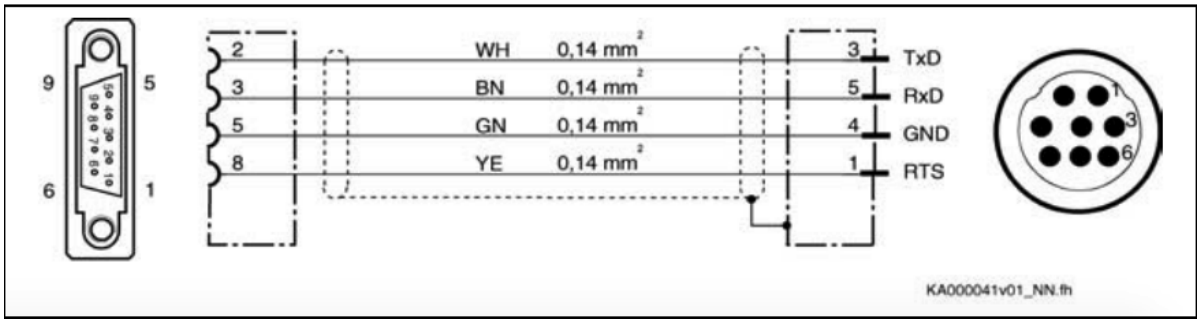
4.3 Drivsteg Bosch Rexroth IndraDrive C

Transportskenorna som användes krävde drivsteg för spänning och styrning och Bosch Rexroth IndraDrive C modeller användes.
Drivstegen kopplades från varsin spänningskälla via ett drivstegsfilter, mot elektriska störningar, in i drivstegen och ut till transportskenorna från utgång X6, X5 och X8 (se Fig. 10).
Drivstegen använde PROFIBUS som I/O-kommunikation med PLC-enheten och en Mini-DIN till D-Sub adapter (se kapitel 4.3.1) för kommunikation och konfiguration med datorn.
Mjukvaran för att konfigurera drivstegen var IndraWorks som går att hitta och ladda ner som en 30-dagars-version på Bosch Rexroth hemsida [12].

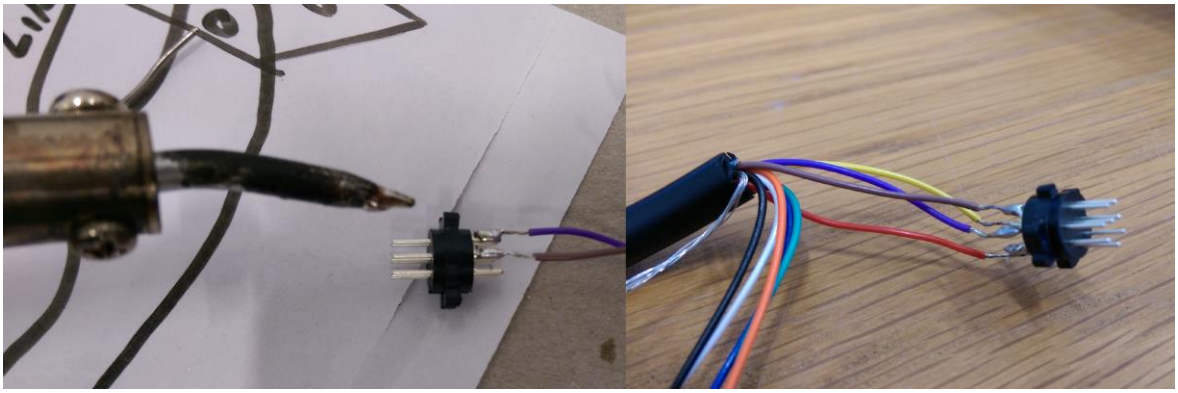
4.3.1 8-Pin Mini Din hane till 9-Pin D-Sub hona

Vid konfigurering av drivstegen saknades en kabel för uppkoppling till datorn. Kabeln, som kostade ca 1500kr tillverkades på plats med enbart ett inköp av en Mini Din-adapter för ca 50kr från Elfa [13].
Företaget hade tillgång till D-Subadaptern (com-kabeln). I manualen för drivstegen hittades pinkonfigurationen. Kabeln skars upp och löddes ihop enligt Fig. 21.
Pin 2 och 6 uteslöts, Clear to send och supply voltage, då de inte ingick i ritningen.

Pin	Signal	Function
1	RTS	Request to send
2	CTS	Clear to send
3	TxD	Transmit Data
4	GND	reference potential
5	RxD	Receive Data
6	V_{cc}	supply voltage
7	n.c.	n.c.
8	n.c.	n.c.



Figur 21 - Pinkonfiguration mellan en D-Sub och en Mini Din adapter



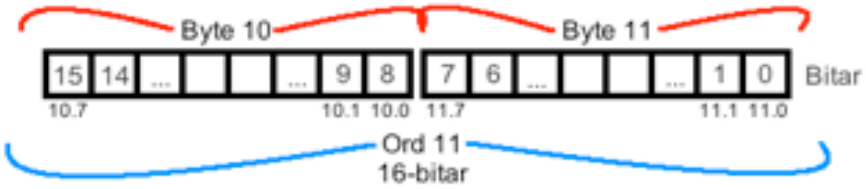
Figur 22 – Lödning av Mini DIN

Figur 23 – Det färdiga resultatet av Mini DIN -> D-sub

4.3.2 Ställa in drivstegen

Vid uppstart av IndraWorks valdes ”Nytt Projekt” och stegen följdes för att komma igång. Det skannades efter drivsteget och rätt baudvärde ställdes in (bitar per sekund på en serieport).
När enheten hittades så användes snabbguiderna från Bosch Rexroths hemsida för att konfigurera drivsteget. Guiden ”Inställningar för linjärpositionerande axel med begränsad rörelse” [11] följdes. Längden på skenorna, matningskonstanten för hur många millimeter axeln rör sig om utgående axel på växellådan vrids ett varv samt utväxling n1 och n2 ställdes in. Alla värden fanns på typskyltarna för servomotorerna, växellåda och skenorna.

De fundamentala inställningarna var vilka parametrar i *Real Time Input (AT)* och *Real Time Output (MDT)* projektet behövde. Obligatoriska parametrar var *Field bus: status word (P-0-4078)* och *Field bus: control word (P-0-4077)* (se Fig. 24). Parametrarna bestod av ”ord” och ett ord har storleken 2 bytes eller 16 bitar och var något som man skulle ta hänsyn till när man lade till parametrar i IndraWorks. Storleken av orden skulle stämma överens med storleken på orden man konfigurerade i STEP-7 under hårdvaruinställningarna under PROFIBUS-input/Output word. Det skulle dedikeras rätt adresser till rätt parametrar så att t.ex. bit 10.0 motsvarade 8:e biten på ord 11 (se Fig. 24). Parametrar med storlek 14 bytes i både Real Time input/output listan, och således 7 ord i STEP-7 på PROFIBUS:ens input/output-lista lades till.



Figur 24 – uppbyggnad av ord i indraworks

4.3.3 Hantera words, in- och utgångarna till och från drivsteget

”Fieldbus control word”-parametern hade hand om output-värdena till drivsteget medans ”Fieldbus status word”-parametern hade hand om input-värdena.
P-0-4078 läste av värden som drivsteget hade skicka vidare informationen till HMI:t. P-0-4077 användes för att initiera rörelse av transportskenorna genom att flanka bitar som ”Drive Enable” och ”Drive Start” som sedan kopplades till HMI:t för taktil styrning.

För att starta igång rörelse i transportbanden utfördes följande initiering:

1. Starta drivstegen.
2. Finns det kraft in samt inga alarm visas koden ”Ab” i displayen. Saknas kraft står det ”Bb”.
3. Flankning av bit 13 i P-0-4077, som motsvarar ”Drive Enable”, aktiverar halt-mode och displayen går över till ”AH”. Sedan flankas ”Drive Start” som är bit 15 i samma ord. ”AF”, reglering, syns i displayen.
4. Inställning av hastigheten som skenorna ska röra sig med (mm/min) och position om de ska köra till en predestinerad längd (mm). Testsekvensen använder sig av 70000 mm/min som hastighet.
5. Flankning av bit 0 i P-0-4077 får skenorna att röra sig.

P-0-4077 Fieldbus control word

Bit	Designation/function
0	**command value acceptance** upon a change (S-0-0346, bit 0) - a positioning block is activated or - the command position is accepted
1	**operating mode setting** **0->1:** change to operating mode **1->0:** change to parameter mode
2	**going to zero (S-0-0148)** **0->1:** start homing command "C6" **1->0:** complete homing command "C6"
3	**absolute / relative (S-0-0346, bit 3)** (only effective when using **S-0-0282, Positioning command value**) **0:** positioning command value (S-0-0282) is processed as absolute target position in the drive **1:** positioning command value (S-0-0282) is processed as relative travel distance in the drive
4	**immediate block change (S-0-0346, bit5)** (only effective when using **S-0-0282, Positioning command value**)

P-0-4078 Fieldbus status word

Bit	Designation/function
1/0	**operating mode acknowledgment** **10:** operating mode **01:** no longer relevant as of MP*04VRS **00:** parameter mode
2	**in reference (status of reference encoder)** (S-0-0403, bit 0) actual position value (encoder 1 or 2) is **0:** relative **1:** homed
3	**in standstill (S-0-0331, bit 0)** **1:** actual velocity | < standstill window |S-0-0040| < S-0-0124
4	**command value reached for ...** **... velocity control:** **1:** command speed reached (S-0-0330, bit 0) **... cyclic position control:** **1:** in position (S-0-0336, bit 0) **... drive-internal interpolation:** **1:** |(S-0-0258) - (S-0-0051/53)| S-0-0057 (S-0-0437, bit 1) **... drive-controlled positioning:**

Figur 25 - Attributen till bitvärdena hos in- och utparametrarna.

4.3.4 Oscilloskopet i indraworks engineering

Det inbyggda oscilloscopet i IndraWorks användes för att felsöka om rätt bit/ord fått rätt värde, som t.ex. hastighet (S-0-0259 Positioning Velocity) och position (S-0-0282 Positioning command value). Genom att välja en parameter fick man direkt information i talsystemen decimalt, hexadecimalt och binärt. Decimalt var användbart när man ville se ett int- eller dint-värde och binärt när man ville se om en bit flankades eller ej.

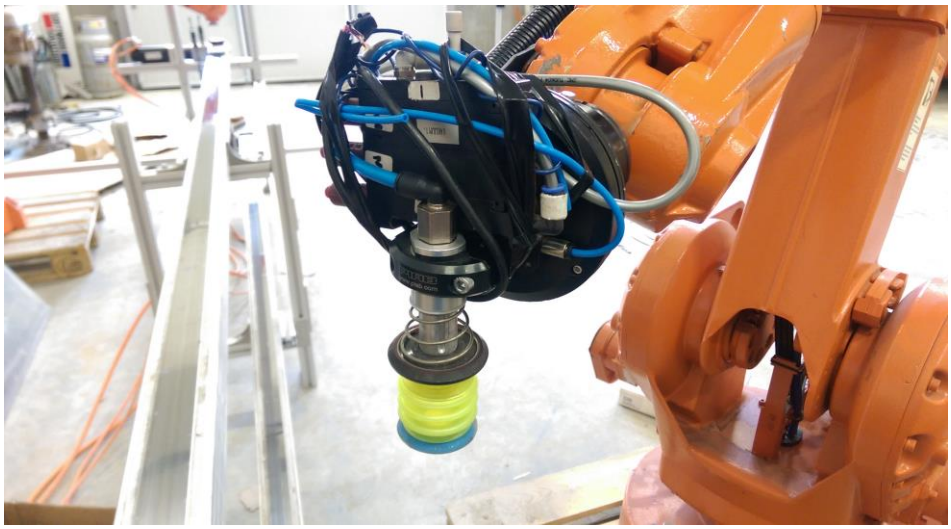
4.4 Roboten

För att illustrera transportbandens tilltänkta funktion integrerades det en robot i processen.
Då den enbart skulle vara med som en visuell del så beskrivs den inte i detalj. Det som kommer att beskrivas är hur tillvägagångssättet gick till rent praktiskt för att få den att fungera i sekvensen. Handledaren på företaget bistod med hjälp, detta för att det hela skulle gå snabbare på grund av tidsbrist.

Figur 26 - Transportskenorna med robotar av modell IRB2400 M97A

4.4.1 Utförande

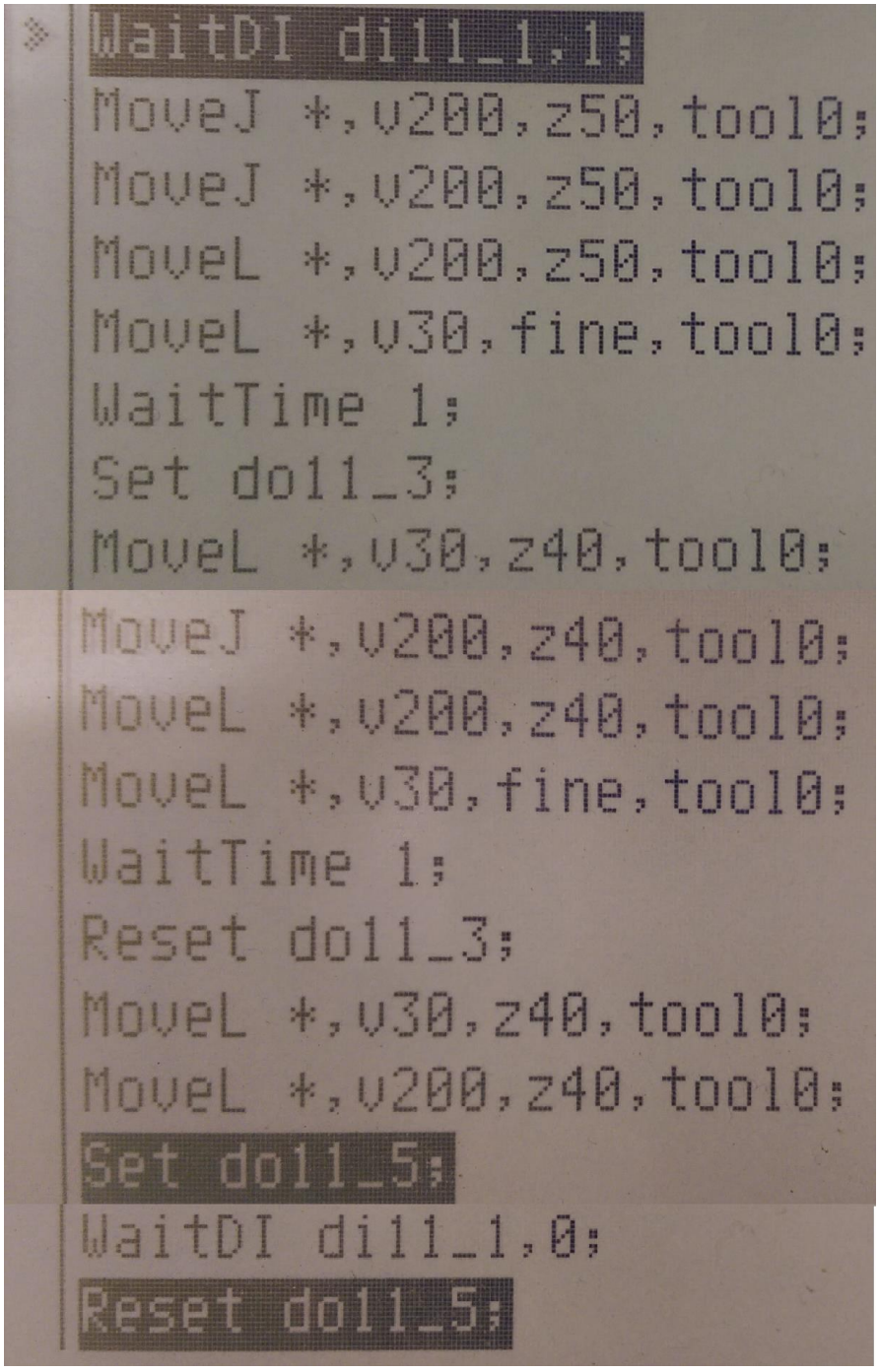
Valet av verktyg för roboten blev en vakuum-klocka som det kopplades komprimerad luft till via en kompressor för att vakuumet skulle fungera.
För att signalbehandlingen mellan robot och PLC skulle fungera användes ett relä, detta för att robotens signal inte skulle gå direkt till PLC:n utan i stället via reläet mata PLC:ns I/O med +24V via extern strömkälla.
PLC:ns I/O-kort behövde även matning med +/-24V för att fungera (detta utan att beröra styrsignalerna) och för att utesluta störningar användes samma spänningskälla som till reläet.



Figur 27 - Robotens vakuum-sug

Roboten, som var från ABB var färdigprogrammerad till att följa instruktioner som man matade in i handenheten, denna användes också för att manuellt styra roboten.
Positionerna för upplockning respektive avlämning ställdes in genom att manuellt flytta robotens arm med handenheten och sedan spara undan koordinaterna.
För att undvika kollision behövde man ta hänsyn till att roboten ville gå kortaste vägen mellan positionerna och då skenorna låg ovanför varandra behövdes en förflyttning ut från transportbanden först göras efter att upplockning hade skett.

Efter att alla kommandon var klara ändrades PLC-koden om för transportskenorna så att roboten började köra när det övre transportbandet var framme i position för avlastning. När avlastningen var klar och produkten var lämnad på det nedre bandet skickade roboten en signal tillbaka till PLC:n och skenorna körde vidare.
Då det tidigare hade använts en timer som triggades när den övre skenan kom fram till slutposition för att sedan två sekunder senare skicka en signal att den undre skenan kunde åka tillbaka var det lätt att ersätta denna kod.
Timern kunde bytas ut mot signalen som skickades till roboten och när timern skulle löst ut ersattes av signalen som roboten skickade tillbaka när den var klar med förflyttningen.



Figur 28 - Robotens sekvenskod

5 Diskussion och slutsats

Vårt huvudsakliga mål om att sätta upp och konfigurera ett fungerande transportsystem fullföljdes enligt planeringen. Kommunikationen mellan enheterna fungerar som det var tänkt i stora drag. Mindre buggar i programmeringskoden orsakar ibland konflikter i systemet, där bland annat byte mellan skärmar i HMI:t orsakar felaktigt utförande av ”Test-funktionen” som för transportskenorna enligt upprepande mönster. En nollställning (Reset) av koden rättar till problematiken.
En orsak kan vara uppbyggnaden av OB1-blocket där vi tillkallar och uppdaterar parametrar flera gånger under en cykel (OB1 exekverar kod enligt sekvenser och efter sista sekvensen så börjar den om). Istället borde vi, när vi började programmeringen, initierat alla ingångar och parametrar och utfört bytena i sista sekvens så att det inte blir krockar mitt i OB1-blocket.

Roboten blev som tänkt, en förflyttning av en produkt mellan övre och undre skenan och blev även integrerad i sekvensen som kör transportbanden. Tanken med roboten var att visuellt kunna demonstrera vår produkts tilltänkta ändamål vilket blev väldigt lyckat.
Då roboten redan kom förprogrammerad så jobbade vi bara med inmatning i handenheten vilket inte kändes tillräckligt eftersom vi gärna hade velat lära oss mer om hur den fungerar och hur den är uppbyggd (framförallt skåpet som styr denna).

Ser man på arbetet i sin helhet har vi skapat oss en väldigt bra uppfattning om hur en automationsingenjör jobbar, vad för olika grenar som finns, olika program och hur man ska tänka i olika situationer.
Vi har lärt oss grunderna för hur Siemens PLC och HMI fungerar, vad som är viktigt att tänka på i de olika stegen samt strukturering och felsökning under arbetets gång.
Vi har fått använda oss mycket av manualer och hjälpsektionerna i programmen och har på grund av detta lärt oss väldigt mycket mer än om vi bara hade blivit instruerade i hur vi skulle göra.

6 Referenslista

6.1 Information

[1] - http://www.engineersgarage.com/articles/scada-systems# (SCADA) [2015-05-25]

[2] - http://www.cisco.com/web/strategy/docs/manufacturing/industrial_ethernet.pdf (industrial ethernet) [2015-05-25]

[3] - http://www.plcdev.com/how_plcs_work (PLC) [2015-05-25]

[4] - http://www.motioncontroltips.com/2011/10/06/hmi-software/ (HMI) [2015-05-25]

[5] - http://www.profibus.com/nc/download/technical-descriptions-books/downloads/profibus-technology-and-application-system-description/download/14844/ (profibus) [2015-05-25]

[6] – http://www.lammertbies.nl/comm/cable/RS-232.html (RS232) [2015-05-25]

[7] Rexroth IndraDrive Drive Controllers Control Section, Edition 06, 2007 https://www.boschrexroth.com/country_units/america/united_states/sub_websites/brus_dcc/documentation_downloads/ProductDocumentation/CurrentProducts/Drives/IndraDrive/29501206.pdf [2015-05-25]

[8] Rexroth IndraDrive C Drive Controllers HCS02.1, HCS03.1, Edition 01, 2006 http://www.convertingsystems.com/uploads/2/6/8/5/26859557/rexroth_indradrivecdrivecontrollershcs021hcs031.pdf [2015-05-25]

[9] Rexroth Bosch Group, Electric Drives and Controls – Product Catalog http://www.boschrexroth.com/country_units/america/united_states/sub_websites/brus_dcc/ProductCatalog_2003_copy/Product_Catalog/Drives/ECODRIVE/Auxiliary_Components/NFD/index.jsp [2015-05-25]

[10] IndraDyn S, MSK – meets all requirements, 2012 http://www.valinonline.com/images/support_docs/Bosch-Rexroth-IndraDyn-S-MSK-Servo-Motors-2014.pdf [2015-05-25]

[11] http://dc-emea.resource.bosch.com/media/se/files_3/edc_snabbguider/SnabbGuide_-_IndraDrive_Grund_ver_4.pdf [2015-05-25]

[12] http://www.boschrexroth.com/sv/se/hem/index [2015-05-25]

[13] www.elfa.se [2015-05-25]

[14] http://w3.siemens.com/mcms/simatic-controller-software/en/step7/step7-professional/pages/default.aspx [2015-05-25]

6.2 Bilder

Figur 3 – Basic Guide to PLCs
http://www.plcmanual.com/examples-vi [2015-05-25]

Figur 6 – PROFIBUS System, Description Technology and Application
http://www.profibus.com/nc/download/technical-descriptions-books/downloads/profibus-technology-and-application-system-description/download/14844/ [2015-05-25]

Figur 8 – http://en.wikipedia.org/wiki/File:Serial_port.jpg [2015-05-25]

Figur 9 - Rexroth IndraDrive Drive Controllers Control Section, Edition 06, 2007
https://www.boschrexroth.com/country_units/america/united_states/sub_websites/brus_dcc/documentation_downloads/ProductDocumentation/CurrentProducts/Drives/IndraDrive/29501206.pdf [2015-05-25]

Figur 10,21 - Rexroth IndraDrive C Drive Controllers HCS02.1, HCS03.1, Edition 01, 2006
http://www.convertingsystems.com/uploads/2/6/8/5/26859557/rexroth_indradrivecdrivecontrollershcs021hcs031.pdf [2015-05-25]

Figur 11 - Rexroth IndraDrive C, Drive Controllers, Power Swections HCS02.1, Edition 03, 2006
http://www.convertingsystems.com/uploads/2/6/8/5/26859557/rexroth_indradrivecontrollerspowersectionhcs021.pdf [2015-05-25]

Figur 12,13 - Rexroth IndraDrive, Drive Controllers, Control Sections, Edition 06, 2007
https://www.boschrexroth.com/country_units/america/united_states/sub_websites/brus_dcc/documentation_downloads/ProductDocumentation/CurrentProducts/Drives/IndraDrive/29501206.pdf [2015-05-25]

Figur 14 – Siemens SIMATIC WinCC
http://w3.siemens.com/mcms/human-machine-interface/en/visualization-software/scada/simatic-wincc/Pages/default.aspx#w2gImgRC-/mcms/human-machine-interface/en/visualization-software/scada/simatic-wincc/tabcards/PublishingImages/graphics-designer-600.jpg [2015-05-25]

Figur 15 – SIMATIC STEP-7, Basic Software
http://w3.siemens.com/mcms/simatic-controller-software/en/step7/step7-professional/PublishingImages/SIMATIC_STEP7_Basic_software.jpg [2015-05-25]

Figur 25 - IndraDrive – Profibus inställningar, Grundläggande inställningar I Profibus, version 3
http://dc-emea.resource.bosch.com/media/se/files_3/edc_snabbguider/Indradrive_-_Profibus_instaellningar_31.pdf [2015-05-25]

7 Bilagor

7.1 PLC

7.1.1 Ladder-kod

Härefter presenteras all ladder-kod med kommentarer om de olika nätverken. Varje rad i koden är ett nytt nätverk där varje nätverk har en eller flera ingångar för att starta och sedan en eller flera utgångar för att avsluta.

OB1

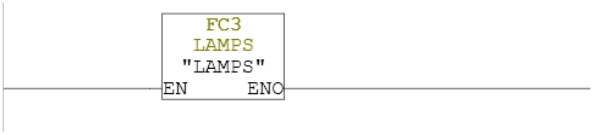
Name	Data Type	Address	Comment
TEMP		0.0	
OB1_EV_CLASS	Byte	0.0	Bits 0-3 = 1 (Coming event), Bits 4-7 = 1 (Event class 1)
OB1_SCAN_1	Byte	1.0	1 (Cold restart scan 1 of OB 1), 3 (Scan 2-n of OB 1)
OB1_PRIORITY	Byte	2.0	Priority of OB Execution
OB1_OB_NUMBR	Byte	3.0	1 (Organization block 1, OB1)
OB1_RESERVED_1	Byte	4.0	Reserved for system
OB1_RESERVED_2	Byte	5.0	Reserved for system
OB1_PREV_CYCLE	Int	6.0	Cycle time of previous OB1 scan (milliseconds)
OB1_MIN_CYCLE	Int	8.0	Minimum cycle time of OB1 (milliseconds)
OB1_MAX_CYCLE	Int	10.0	Maximum cycle time of OB1 (milliseconds)
OB1_DATE_TIME	Date_And_Time	12.0	Date and time OB1 started
bryt	Bool	20.0	

Block: OB1 "Main Program Sweep (Cycle)"

Huvudsekvens

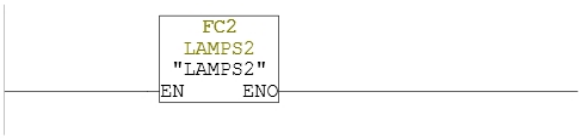
Network: 1

Lampor i HMI för feedback från drivsteg 1



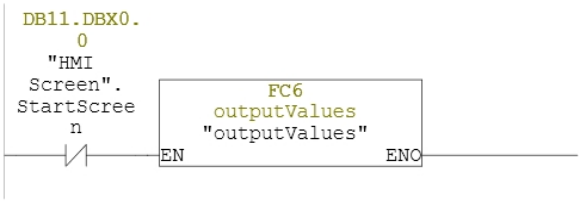
Network: 2

Lampor i HMI för feedback från drivsteg 2



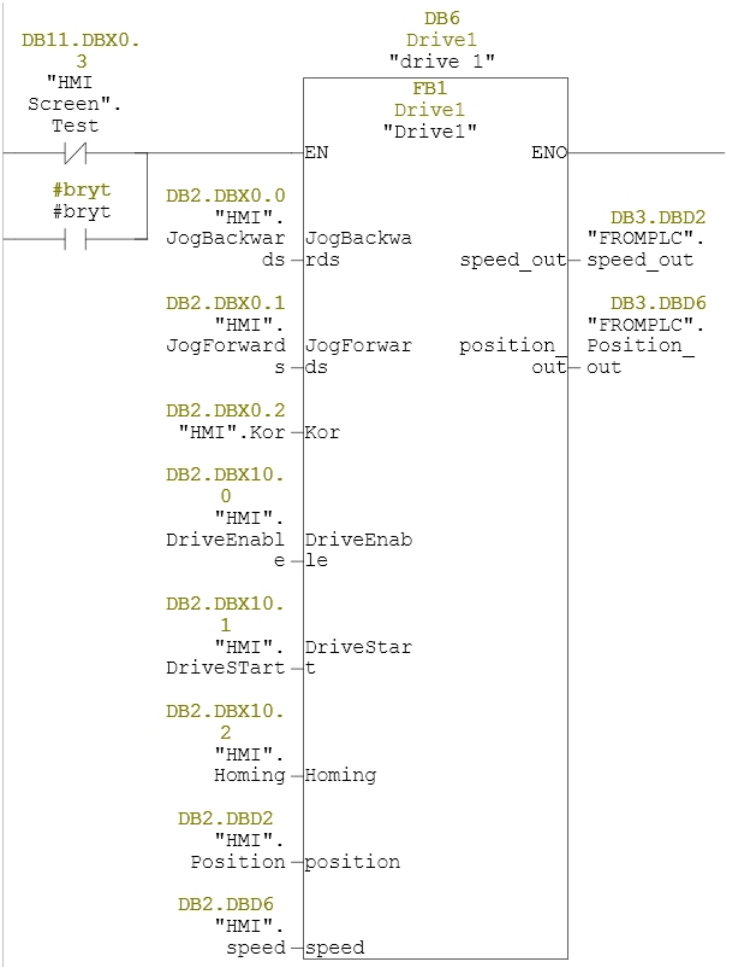
Network: 3

Hastighet och position för avläsning i HMI

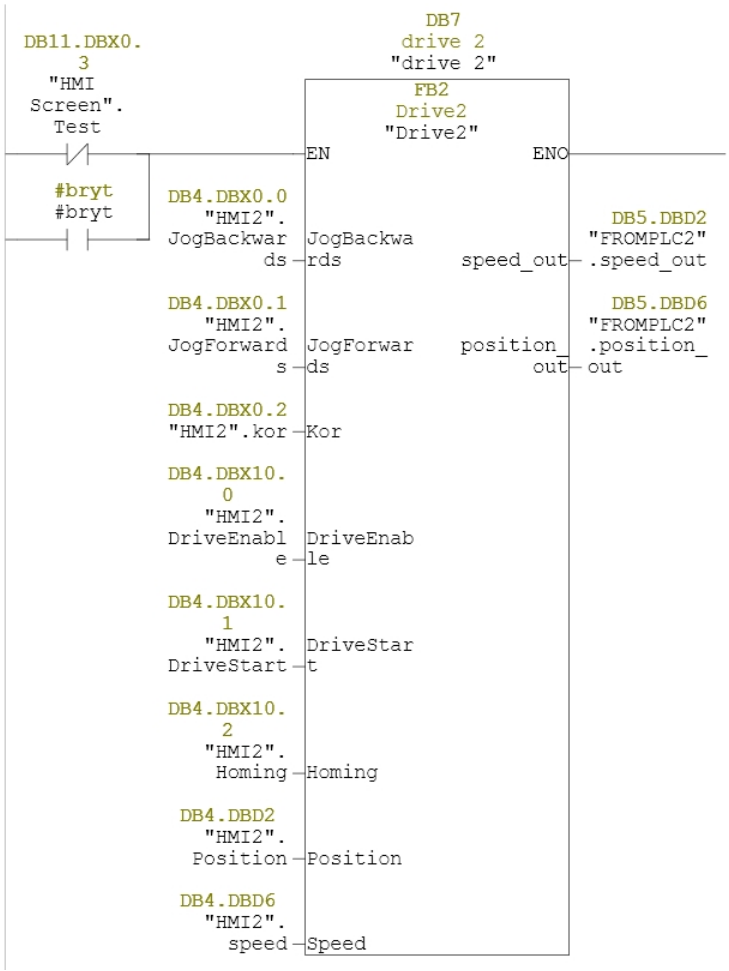


Network: 4

Funktionsblock för drivsteg 1

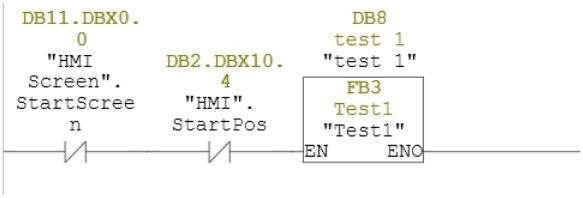


Network: 5
Funktionsblock för drivsteg 2



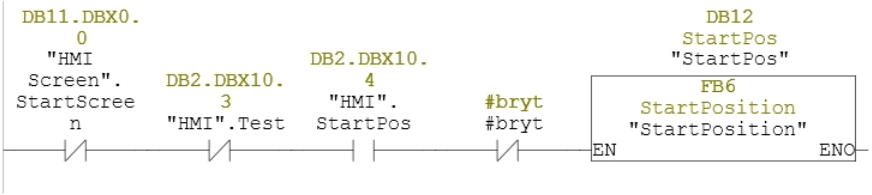
Network: 6

Testsekvensen för automatiserad rörelse



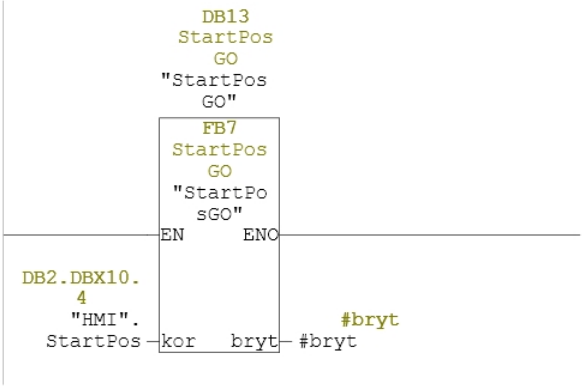
Network: 7

Sekvens för startpositionering inför testsekvens



Network: 8

brytfunktion för startpositionering



FB1

Name	Data Type	Address	Initial Value	Comment
IN		0.0		
JogBackwards	Bool	0.0	FALSE	
JogForwards	Bool	0.1	FALSE	
Kor	Bool	0.2	FALSE	
DriveEnable	Bool	0.3	FALSE	
DriveStart	Bool	0.4	FALSE	
Homing	Bool	0.5	FALSE	
position	DInt	2.0	L#0	
speed	DInt	6.0	L#0	
OUT		0.0		
speed_out	DInt	10.0	L#0	
position_out	DInt	14.0	L#0	
IN_OUT		0.0		
STAT		0.0		
TEMP		0.0		

Block: FB1

Funktionsblock för drivsteg 1

Network: 1

för att jogga bakåt

Network: 2

för att jogga framåt

Network: 3

kör sekvens till inställd position och med inställd hastighet

Network: 4

DriveEnable - sätter drivstegen i ett startklart läge

Network: 5

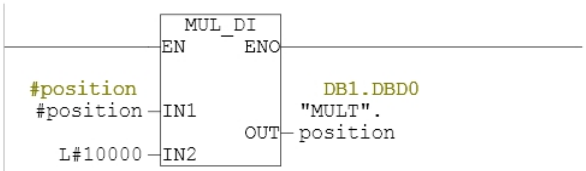
DriveStart - sätter drivstegen i reglerläge

Network: 6

Homing - går till inställd hemposition

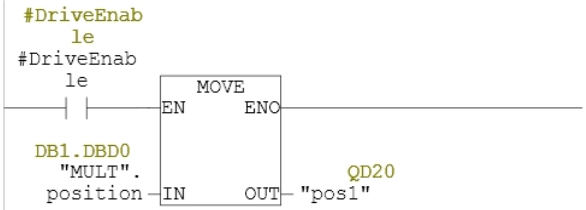
Network: 7

multiplicerar position med 10000 för att få rätt utvärde



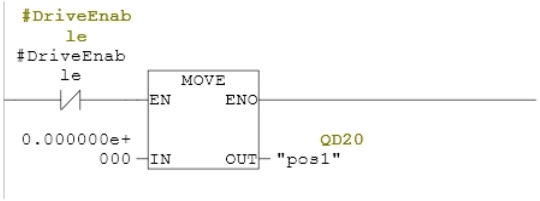
Network: 8

Flyttar positionen, efter multiplikation till korrekt utgång (till drivsteget)



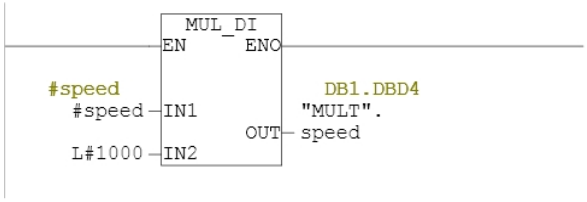
Network: 9

Nollställer position när drive enable slås av, för att undvika konflikter



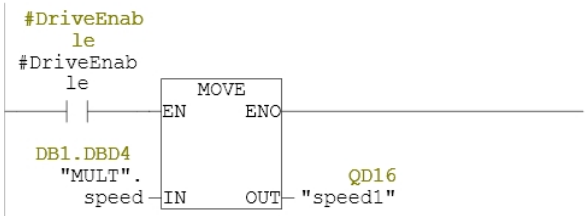
Network: 10

multiplicerar hastighet med 1000 för att få rätt utvärde



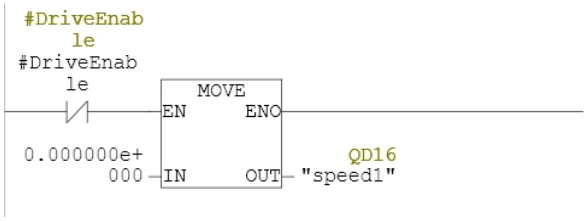
Network: 11

Flyttar hastighet, efter multiplikation till korrekt utgång (till drivsteget)



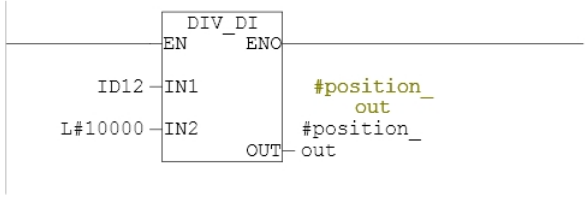
Network: 12

Nollställer hastighet när drive enable slås av, för att undvika konflikter



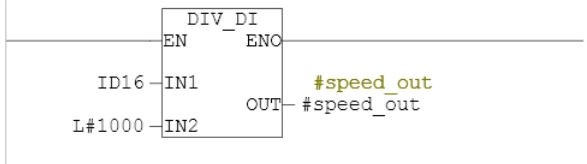
Network: 13

Dividerar inläst värde från drivstegen och dividerar sedan detta med 10000 för att få ett korrekt värde



Network: 14

Dividerar inläst värde från drivstegen och dividerar sedan detta med 1000 för att få ett korrekt värde



FB2

Name	Data Type	Address	Initial Value	Comment
IN		0.0		
JogBackwards	Bool	0.0	FALSE	
JogForwards	Bool	0.1	FALSE	
Kor	Bool	0.2	FALSE	
DriveEnable	Bool	0.3	FALSE	
DriveStart	Bool	0.4	FALSE	
Homing	Bool	0.5	FALSE	
Position	DInt	2.0	L#0	
Speed	DInt	6.0	L#0	
OUT		0.0		
speed_out	DInt	10.0	L#0	
position_out	DInt	14.0	L#0	
IN_OUT		0.0		
STAT		0.0		
TEMP		0.0		

Block: FB2

Funktionsblock för drivsteg 2

Network: 1

För att jogga bakåt

Network: 2

För att jogga framåt

Network: 3

kör sekvens till inställd position och med inställd hastighet

Network: 4

DriveEnable - sätter drivstegen i ett startklart läge

Network: 5

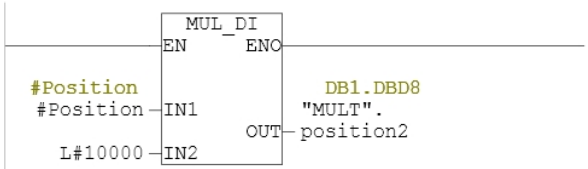
DriveStart - sätter drivstegen i reglerläge

Network: 6

Homing - går till inställd hemposition

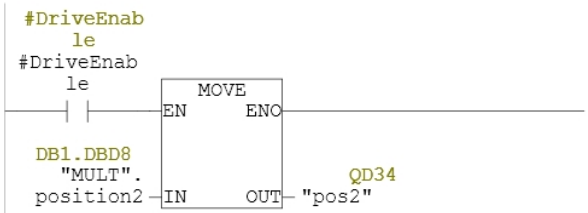
Network: 7

multiplicerar position med 10000 för att få rätt utvärde



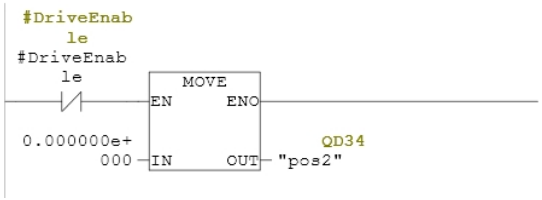
Network: 8

Flyttar positionen, efter multiplikation till korrekt utgång (till drivsteget)



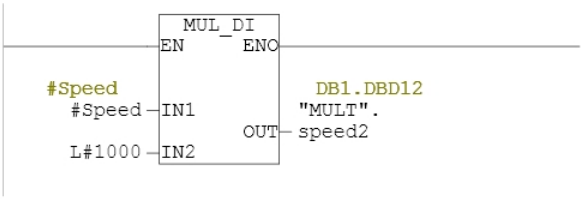
Network: 9

Nollställer position när drive enable slås av, för att undvika konflikter



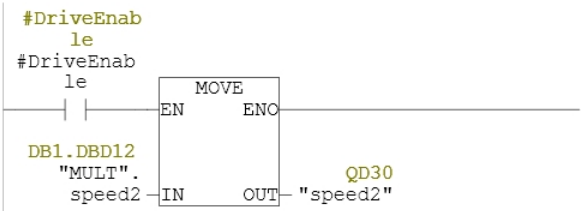
Network: 10

multiplicerar hastighet med 1000 för att få rätt utvärde



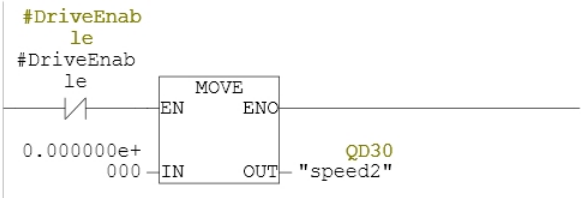
Network: 11

Flyttar hastighet, efter multiplikation till korrekt utgång (till drivsteget)



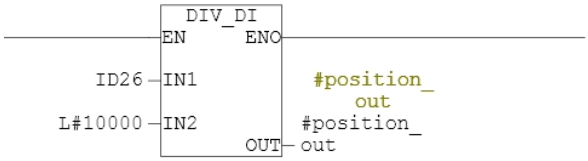
Network: 12

Nollställer hastighet när drive enable slås av, för att undvika konflikter



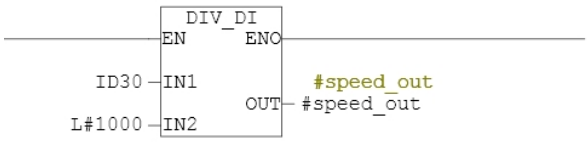
Network: 13

Dividerar inläst position från drivstegen och dividerar sedan detta med 10000
för
att få ett korrekt värde



Network: 14

Dividerar inläst hastighet från drivstegen och dividerar sedan detta med 1000
för
att få ett korrekt värde



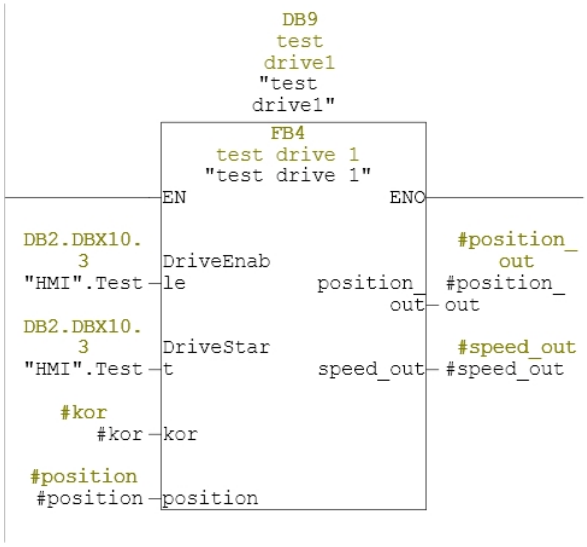
FB3

Name	Data Type	Address	Initial Value	Comment
IN		0.0		
OUT		0.0		
IN_OUT		0.0		
STAT		0.0		
position_out	DInt	0.0	L#0	
kor	Bool	4.0	FALSE	
speed_out	DInt	6.0	L#0	
position	DInt	10.0	L#0	
kor2	Bool	14.0	FALSE	
position2_out	DInt	16.0	L#0	
speed2_out	DInt	20.0	L#0	
position2	DInt	24.0	L#0	
TEMP		0.0		

Block: FB3
Testsekvensens funktionsblock

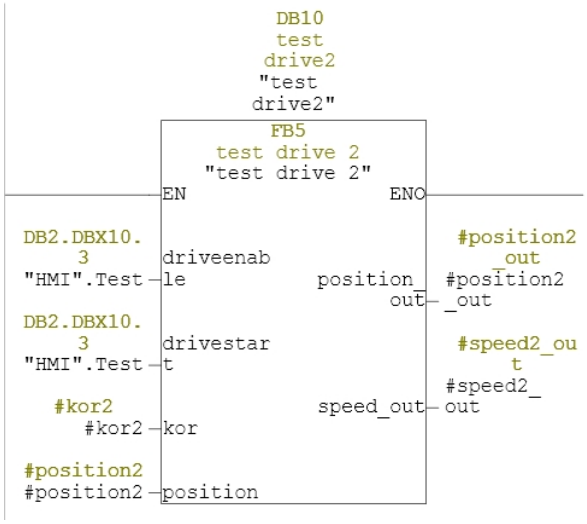
Network: 1

Förenklat funktionsblock för drivsteg 1



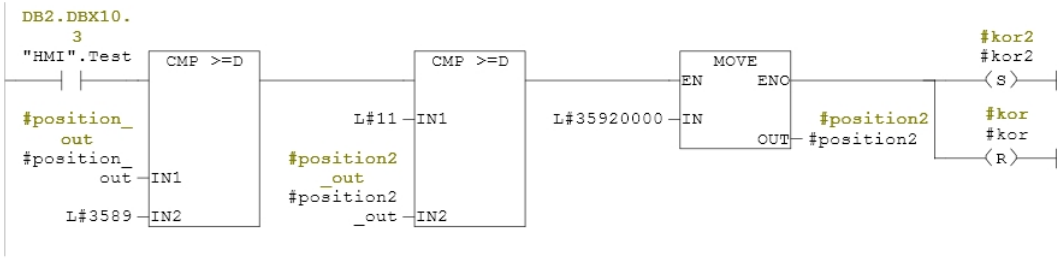
Network: 2

Förenklat funktionsblock för drivsteg 2



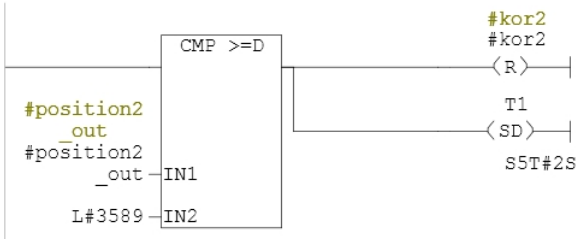
Network: 3

om "test-knappen" är aktiv jämför den om skena 1 och skena 2 är i sina
respektive startpositioner.
Därefter flyttas position för skena 2 till variabel "position2" som är en
ingång
i det förenklade funktionsblocket för drivsteg 2.
Variabel "kor2" sätts för att tillåta rörelse samt variabel "kor" nollas, detta
ifall man tidigare brutit sekvensen utan att variablen hunnit nollas i koden
som den ska

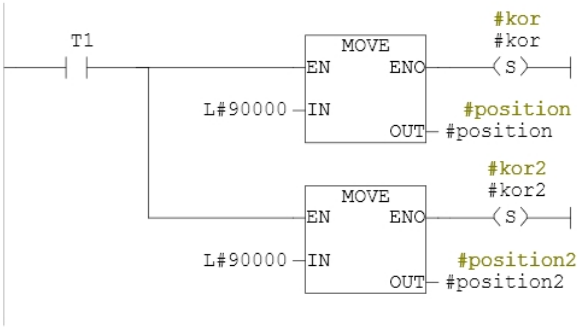


Network: 4

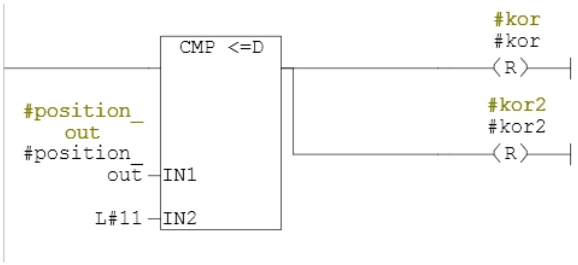
jämför position för skena 2, när den är i position nollas "kor2" och timer
"T1" på 2 sekunder startas



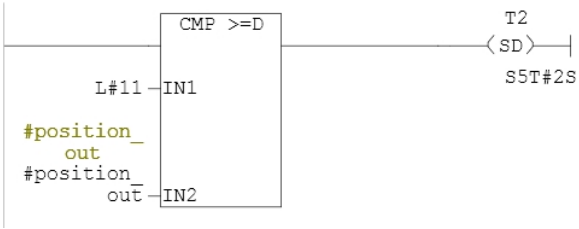
Network: 5 test drive1
timer "T1" triggar förflyttning av position för både skena 1 och 2 och sedan sätts kör-variablerna för de båda så att förflyttning blir tillåten



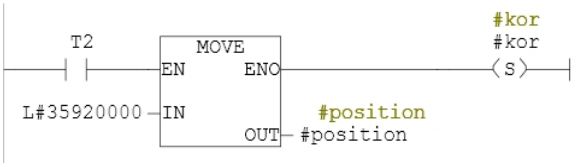
Network: 6 HMI
jämför när skena 1 är i endläge för att sedan nolla kör-variablerna för både skenorna



Network: 7
när skena 1 är i endläge startas en timer "T2" på 2 sekunder.



Network: 8 HMI
Timer "T2" triggar en förflyttning av positionsparametrar för skena 1 till det förenklade drivstegsblocket och sedan sätts "kor" för att tillåta skena 1 att flytta sig till sin startposition.



FB4

Name	Data Type	Address	Initial Value	Comment
IN		0.0		
DriveEnable	Bool	0.0	FALSE	
DriveStart	Bool	0.1	FALSE	
kor	Bool	0.2	FALSE	
position	DInt	2.0	L#0	
OUT		0.0		
position_out	DInt	6.0	L#0	
speed_out	DInt	10.0	L#0	
IN_OUT		0.0		
STAT		0.0		
speed	DInt	14.0	L#70000000	
TEMP		0.0		

Block: FB4

Funktionsblock för testsekvens - drivsteg 1

Network: 1

kör-variabel som tillåter rörelse till inställd poisition och med inställd hastighet

Network: 2

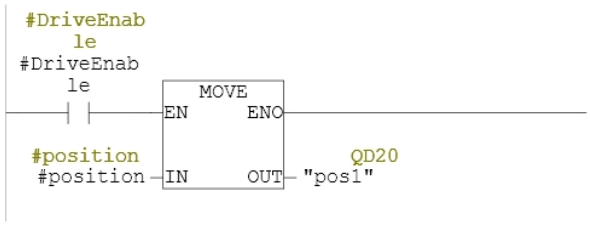
DriveEnable - sätter drivstegen i ett startklart läge

Network: 3

DriveStart - sätter drivstegen i reglerläge

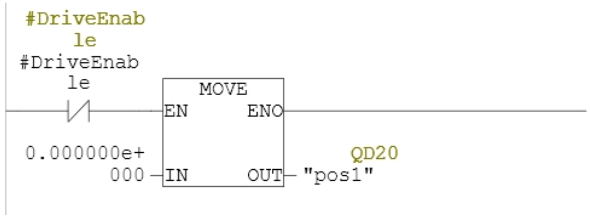
Network: 4

Flyttar positionen till rätt utgång för drivsteg 1



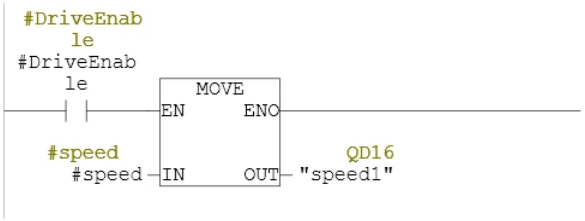
Network: 5

Nollställer positionen när drive enable slås av, för att undvika konflikt



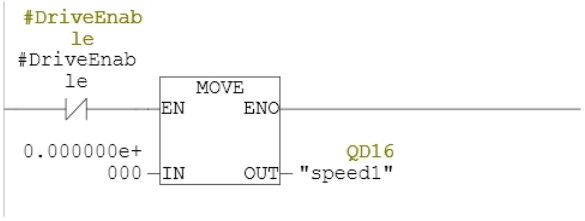
Network: 6

Flyttar hastigheten till rätt utgång för drivsteg 1



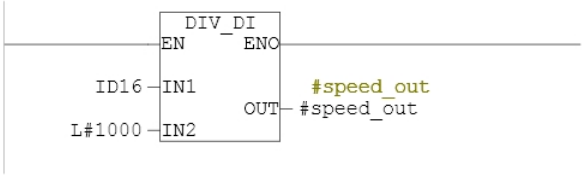
Network: 7

Nollställer hastigheten när drive enable slås av, för att undvika konflikt



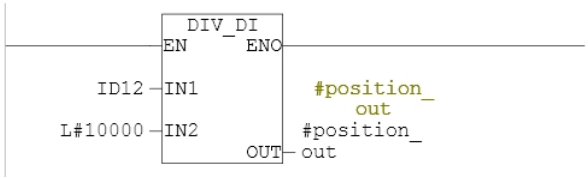
Network: 8

dividerar inläst hastighetsvärde med 1000 för att få en korrekt utläsning på HMI



Network: 9

dividerar inläst positionsvärde med 10000 för att få en korrekt utläsning på HMI



FB5

Name	Data Type	Address	Initial Value	Comment
IN		0.0		
driveenable	Bool	0.0	FALSE	
drivestart	Bool	0.1	FALSE	
kor	Bool	0.2	FALSE	
position	DInt	2.0	L#0	
OUT		0.0		
position_out	DInt	6.0	L#0	
speed_out	DInt	10.0	L#0	
IN_OUT		0.0		
STAT		0.0		
speed	DInt	14.0	L#70000000	
TEMP		0.0		

Block: FB5

Funktionsblock för testsekvens - drivsteg 2

Network: 1

kör-variabel som tillåter rörelse till inställd poisition och med inställd hastighet

Network: 2

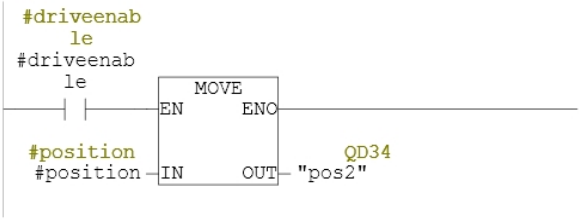
DriveEnable - sätter drivstegen i ett startklart läge

Network: 3

DriveStart - sätter drivstegen i reglerläge

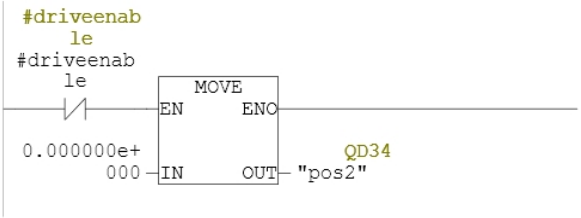
Network: 4

Flyttar positionen till rätt utgång för drivsteg 2



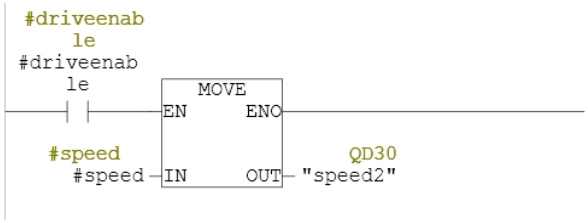
Network: 5

Nollställer positionen när drive enable slås av, för att undvika konflikt



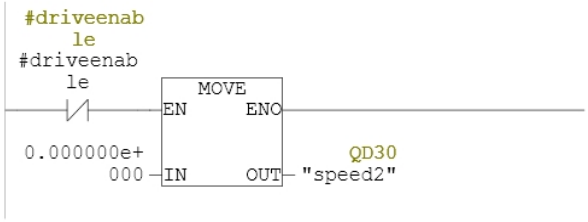
Network: 6

Flyttar hastigheten till rätt utgång för drivsteg 2



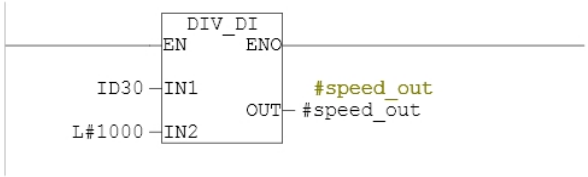
Network: 7

Nollställer hastigheten när drive enable slås av, för att undvika konflikt



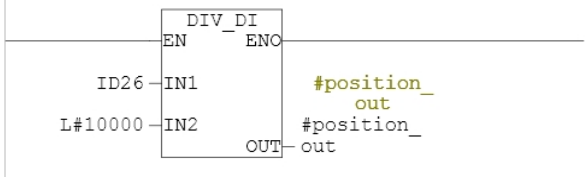
Network: 8

dividerar inläst hastighetsvärde med 1000 för att få en korrekt utläsning på HMI



Network: 9

dividerar inläst positionsvärde med 10000 för att få en korrekt utläsning på HMI

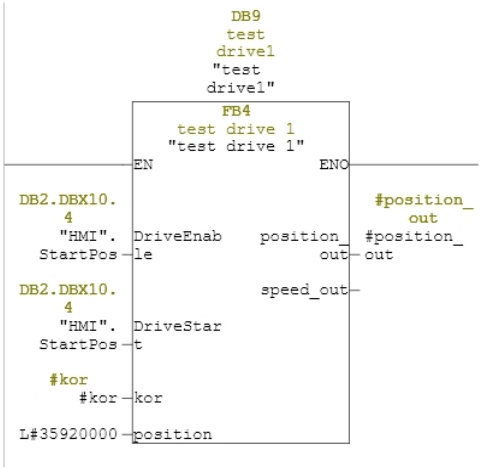


FB6

Name	Data Type	Address	Initial Value	Comment
IN		0.0		
OUT		0.0		
IN_OUT		0.0		
STAT		0.0		
position_out	DInt	0.0	L#0	
kor	Bool	4.0	FALSE	
position2_out	DInt	6.0	L#0	
kor2	Bool	10.0	FALSE	
TEMP		0.0		

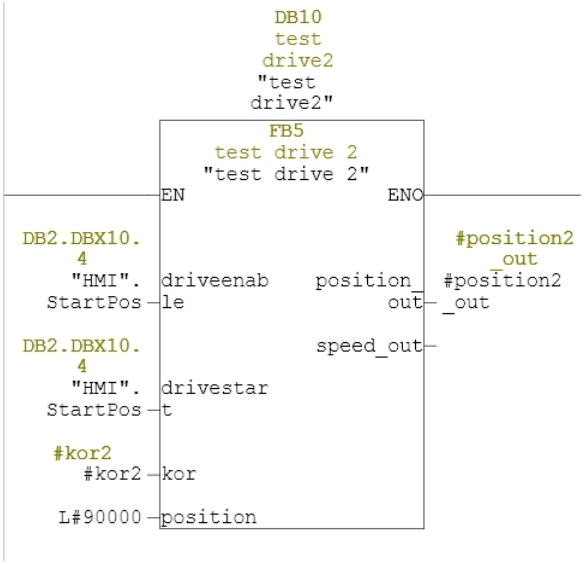
Block: FB6
Funktionsblock för startpositionering

Network: 1
Förenklat funktionsblock för skena 1



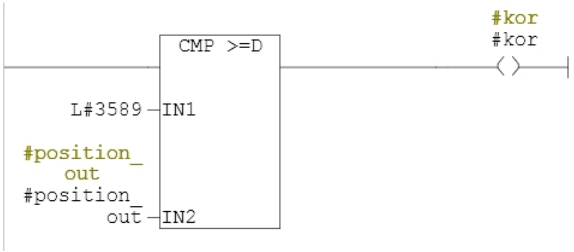
Network: 2

Förenklat funktionsblock för skena 2



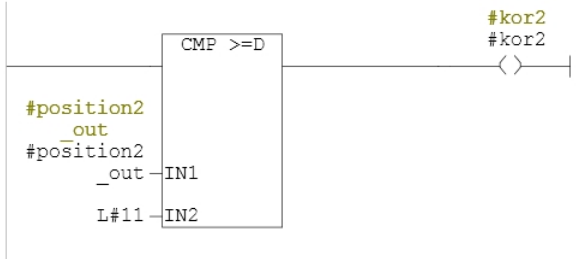
Network: 3 test drive2

jämför om skena 1 är i startposition, om inte sätts kör-variabeln och skena 1
kör till förinställt värde (3592mm)



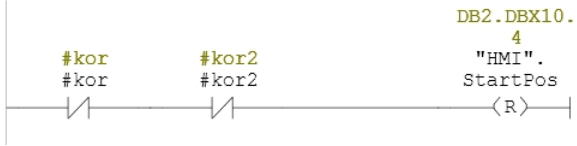
Network: 4

jämför om skena 2 är i startposition, om inte sätts kör-variabeln och skena 2
kör till förinställt värde (9mm)



Network: 5

när skena 1 och 2 är i startposition nollställer sig variabeln som kör igång
sekvensen i OB1



FB7

Name	Data Type	Address	Initial Value	Comment
IN		0.0		
kor	Bool	0.0	FALSE	
OUT		0.0		
bryt	Bool	2.0	FALSE	
IN_OUT		0.0		
STAT		0.0		
TEMP		0.0		

Block: FB7

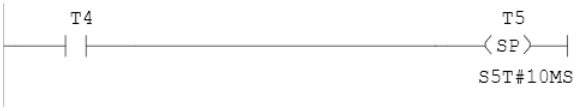
Brytsekvensen som ibland beöhvs för att startpositionering ska fungera

Network: 1

in-variabel som triggas i OB1 och startar timer "T4" på 10 ms

Network: 2

timer "T4" triggar en puls som håller i sig i 10 ms, därefter bryter den



Network: 3

timer "T5" håller signalen i 10 ms som då aktiverar utången "bryt" som syns i OB1

FC1

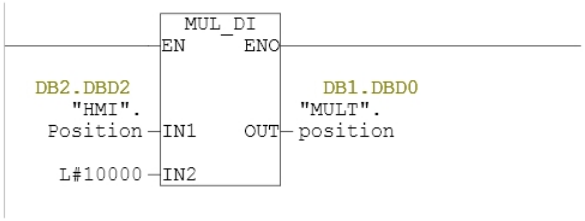
Name	Data Type	Address	Comment
IN		0.0	
OUT		0.0	
IN_OUT		0.0	
TEMP		0.0	
RETURN		0.0	
RET_VAL		0.0	

Block: FC1

Funktion för multiplikation av hastighet och position

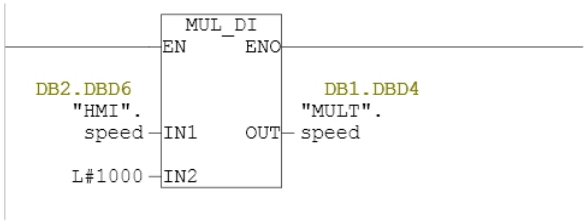
Network: 1

Multiplicerar inmatat positionsvärde i HMI för skena 1 och sparar sedan detta i DB1



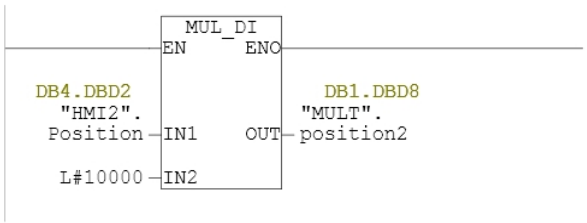
Network: 2

Multiplicerar inmatat hastighetsvärde i HMI för skena 1 och sparar sedan detta
i
DB1



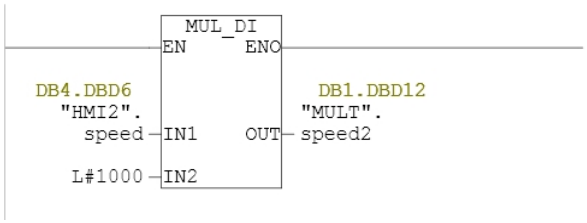
Network: 3

Multiplicerar inmatat positionsvärde i HMI för skena 2 och sparar sedan detta i
DB1



Network: 4

Multiplicerar inmatat hastighetsvärde i HMI för skena 2 och sparar sedan detta
i
DB1



FC2

Name	Data Type	Address	Comment
IN		0.0	
OUT		0.0	
IN_OUT		0.0	
TEMP		0.0	
RETURN		0.0	
RET_VAL		0.0	

Block: FC2

Grafiska lamporna i HMI för feedback från skena 2.

När man trycker på knappen i HMI't så aktiveras motsvarande variabel som skickas
ut från PLC'n.
Denna variabeln läses sedan av HMI't

Network: 1 Temporary placeholder variable

Drive Enable

Network: 2

Kor

Network: 3

Drive Start

Network: 4

Homing

Network: 5

Reset

Network: 6

jog forward

Network: 7

jog backwards

FC3

Name	Data Type	Address	Comment
IN		0.0	
OUT		0.0	
IN_OUT		0.0	
TEMP		0.0	
RETURN		0.0	
RET_VAL		0.0	

Block: FC3

Grafiska lamporna i HMI för feedback från skena 1.

När man trycker på knappen i HMI't så aktiveras motsvarande variabel som skickas
ut från PLC'n.
Denna variabeln läses sedan av HMI't

Network: 1 Temporary placeholder variable

drive enable

Network: 2

kor

Network: 3

drive start

Network: 4

homing

Network: 5

reset

Network: 6

jog forward

Network: 7

jog backwards

FC6

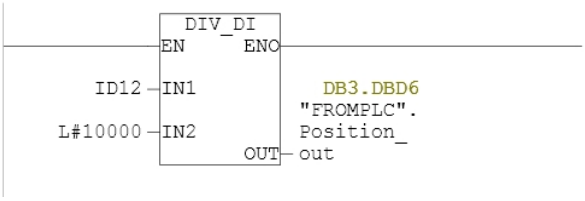
Name	Data Type	Address	Comment
IN		0.0	
OUT		0.0	
IN_OUT		0.0	
TEMP		0.0	
RETURN		0.0	
RET_VAL		0.0	

Block: FC6

Läser av hastighet och position från drivstegen och lagrar resultatet i DB3

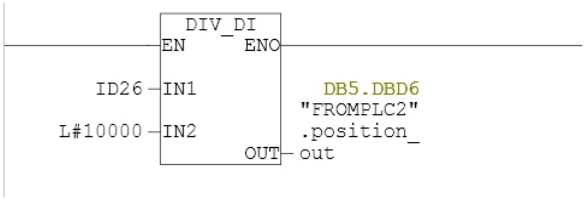
Network: 1

Drive 1 position_out - läser ut position från drivsteg 1 och dividerar det med 10000 för att få korrekt värde



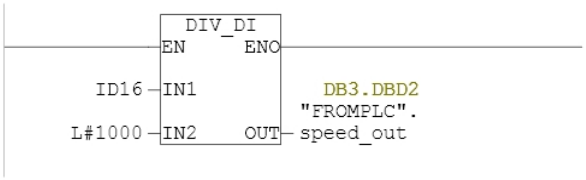
```
Network: 2

Drive 2 position_out - läser ut position från drivsteg 2 och dividerar det med
10000 för att få korrekt värde
```



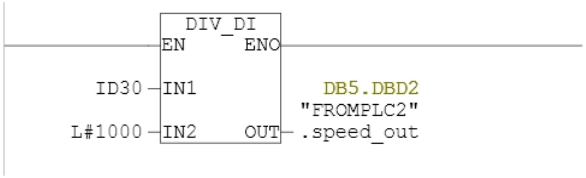

```
Network: 3

Drive 1 speed_out - läser ut hastighet från drivsteg 1 och dividerar det med
1000 för att få korrekt värde
```




```
Network: 4

Drive 2 speed_out - läser ut hastighet från drivsteg 2 och dividerar det med
1000 för att få korrekt värde
```



7.1.2 Datablocken

DB1

Block: DB1

Address	Name	Type	Initial value	Comment
0.0		STRUCT		
+0.0	position	DINT	L#0	
+4.0	speed	DINT	L#0	
+8.0	position2	DINT	L#0	
+12.0	speed2	DINT	L#0	
=16.0		END_STRUCT		

DB2

Block: DB2

Address	Name	Type	Initial value	Comment
0.0		STRUCT		
+0.0	JogBackwards	BOOL	FALSE	
+0.1	JogForwards	BOOL	FALSE	
+0.2	Kor	BOOL	FALSE	
+0.3	OnlinePLC	BOOL	FALSE	
+0.4	PM	BOOL	FALSE	
+0.5	Reset	BOOL	FALSE	
+0.6	StartToPLC	BOOL	FALSE	
+2.0	Position	DINT	L#0	
+6.0	speed	DINT	L#0	
+10.0	DriveEnable	BOOL	FALSE	
+10.1	DriveSTart	BOOL	FALSE	
+10.2	Homing	BOOL	FALSE	
+10.3	Test	BOOL	FALSE	
+10.4	StartPos	BOOL	FALSE	
+10.5	activate_startPos	BOOL	FALSE	
=12.0		END_STRUCT		

DB3

Block: DB3

Address	Name	Type	Initial value	Comment
0.0		STRUCT		
+0.0	DriveEnable	BOOL	FALSE	
+0.1	DriveStart	BOOL	FALSE	
+0.2	Homing	BOOL	FALSE	
+0.3	JogBackwards	BOOL	FALSE	
+0.4	JogForwards	BOOL	FALSE	
+0.5	MotorRun	BOOL	FALSE	
+0.6	OnlinePLC	BOOL	FALSE	
+0.7	Reset	BOOL	FALSE	
+1.0	Start	BOOL	FALSE	
+1.1	kor	BOOL	FALSE	
+2.0	speed_out	DINT	L#0	
+6.0	Position_out	DINT	L#0	
=10.0		END_STRUCT		

DB4

Block: DB4

Address	Name	Type	Initial value	Comment
0.0		STRUCT		
+0.0	JogBackwards	BOOL	FALSE	
+0.1	JogForwards	BOOL	FALSE	
+0.2	kor	BOOL	FALSE	
+0.3	PM	BOOL	FALSE	
+0.4	Reset	BOOL	FALSE	
+0.5	StartToPLC	BOOL	FALSE	
+2.0	Position	DINT	L#0	
+6.0	speed	DINT	L#0	
+10.0	DriveEnable	BOOL	FALSE	
+10.1	DriveStart	BOOL	FALSE	
+10.2	Homing	BOOL	FALSE	
=12.0		END_STRUCT		

DB5

Block: DB5

Address	Name	Type	Initial value	Comment
0.0		STRUCT		
+0.0	DriveEnable	BOOL	FALSE	
+0.1	DriveStart	BOOL	FALSE	
+0.2	Homing	BOOL	FALSE	
+0.3	JogBackwards	BOOL	FALSE	
+0.4	JogForwards	BOOL	FALSE	
+0.5	MotorRun	BOOL	FALSE	
+0.6	Reset	BOOL	FALSE	
+0.7	Start	BOOL	FALSE	
+1.0	kor	BOOL	FALSE	
+2.0	speed_out	DINT	L#0	
+6.0	position_out	DINT	L#0	
=10.0		END_STRUCT		

DB6

Address	Declaration	Name	Type	Initial value	Actual value	Comment
0.0	in	JogBackwards	BOOL	FALSE	FALSE	
0.1	in	JogForwards	BOOL	FALSE	FALSE	
0.2	in	Kor	BOOL	FALSE	FALSE	
0.3	in	DriveEnable	BOOL	FALSE	FALSE	
0.4	in	DriveStart	BOOL	FALSE	FALSE	
0.5	in	Homing	BOOL	FALSE	FALSE	
2.0	in	position	DINT	L#0	L#0	
6.0	in	speed	DINT	L#0	L#0	
10.0	out	speed_out	DINT	L#0	L#0	
14.0	out	position_out	DINT	L#0	L#0	

DB7

Address	Declaration	Name	Type	Initial value	Actual value	Comment
0.0	in	JogBackwards	BOOL	FALSE	FALSE	
0.1	in	JogForwards	BOOL	FALSE	FALSE	
0.2	in	Kor	BOOL	FALSE	FALSE	
0.3	in	DriveEnable	BOOL	FALSE	FALSE	
0.4	in	DriveStart	BOOL	FALSE	FALSE	
0.5	in	Homing	BOOL	FALSE	FALSE	
2.0	in	Position	DINT	L#0	L#0	
6.0	in	Speed	DINT	L#0	L#0	
10.0	out	speed_out	DINT	L#0	L#0	
14.0	out	position_out	DINT	L#0	L#0	

DB8

Address	Declaration	Name	Type	Initial value	Actual value	Comment
0.0	stat	position_out	DINT	L#0	L#0	
4.0	stat	kor	BOOL	FALSE	FALSE	
6.0	stat	speed_out	DINT	L#0	L#0	
10.0	stat	position	DINT	L#0	L#0	
14.0	stat	kor2	BOOL	FALSE	FALSE	
16.0	stat	position2_out	DINT	L#0	L#0	
20.0	stat	speed2_out	DINT	L#0	L#0	
24.0	stat	position2	DINT	L#0	L#0	

DB9

Address	Declaration	Name	Type	Initial value	Actual value	Comment
0.0	in	DriveEnable	BOOL	FALSE	FALSE	
0.1	in	DriveStart	BOOL	FALSE	FALSE	
0.2	in	kor	BOOL	FALSE	FALSE	
2.0	in	position	DINT	L#0	L#0	
6.0	out	position_out	DINT	L#0	L#0	
10.0	out	speed_out	DINT	L#0	L#0	
14.0	stat	speed	DINT	L#70000000	L#70000000	

DB10

Address	Declaration	Name	Type	Initial value	Actual value	Comment
0.0	in	driveenable	BOOL	FALSE	FALSE	
0.1	in	drivestart	BOOL	FALSE	FALSE	
0.2	in	kor	BOOL	FALSE	FALSE	
2.0	in	position	DINT	L#0	L#0	
6.0	out	position_out	DINT	L#0	L#0	
10.0	out	speed_out	DINT	L#0	L#0	
14.0	stat	speed	DINT	L#70000000	L#70000000	

DB11

Address	Name	Type	Initial value	Comment
0.0		STRUCT		
+0.0	StartScreen	BOOL	FALSE	
+0.1	Drive1	BOOL	FALSE	
+0.2	Drive2	BOOL	FALSE	
+0.3	Test	BOOL	FALSE	
+0.4	Messages	BOOL	FALSE	
=2.0		END_STRUCT		

DB12

Address	Declaration	Name	Type	Initial value	Actual value	Comment
0.0	stat	position_out	DINT	L#0	L#0	
4.0	stat	kor	BOOL	FALSE	FALSE	
6.0	stat	position2_out	DINT	L#0	L#0	
10.0	stat	kor2	BOOL	FALSE	FALSE	

DB13

Address	Declaration	Name	Type	Initial value	Actual value	Comment
0.0	in	kor	BOOL	FALSE	FALSE	
2.0	out	bryt	BOOL	FALSE	FALSE	

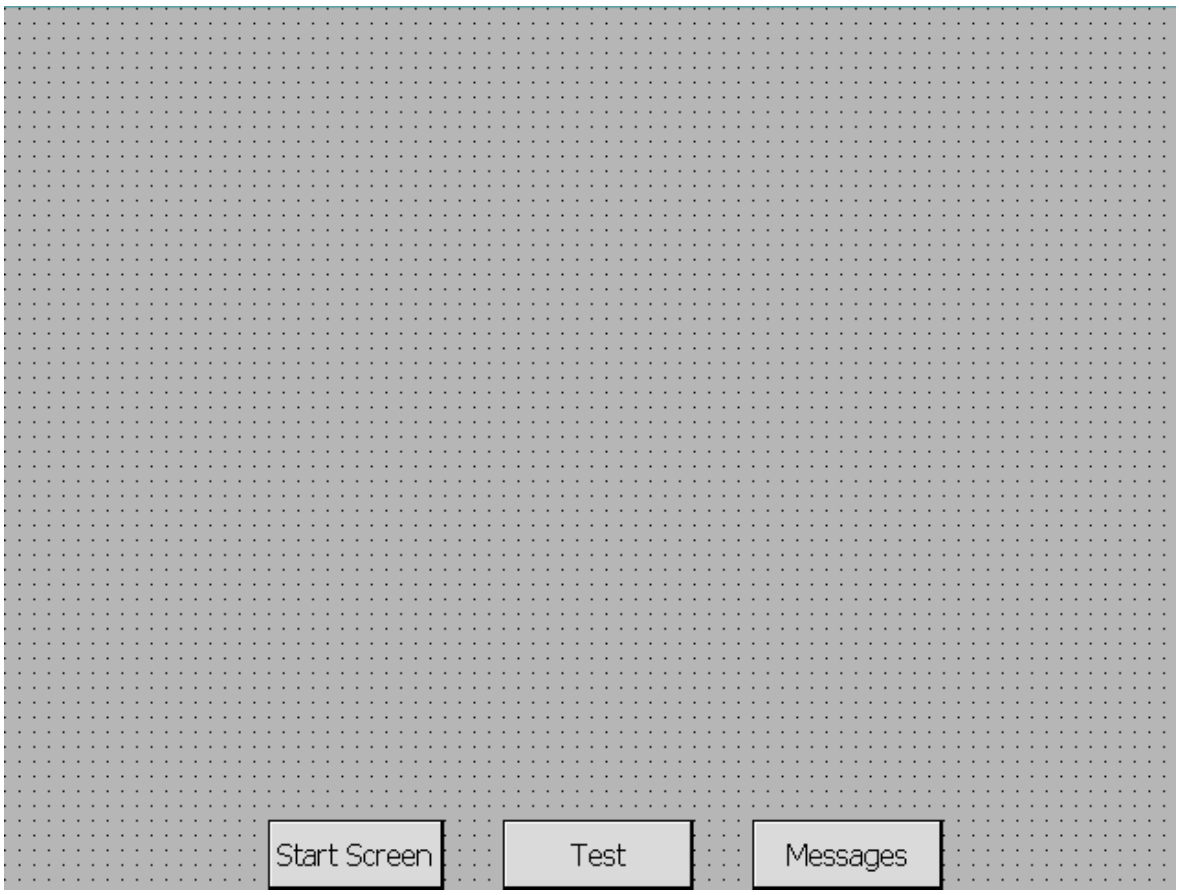
7.2 HMI

7.2.1 Taggar

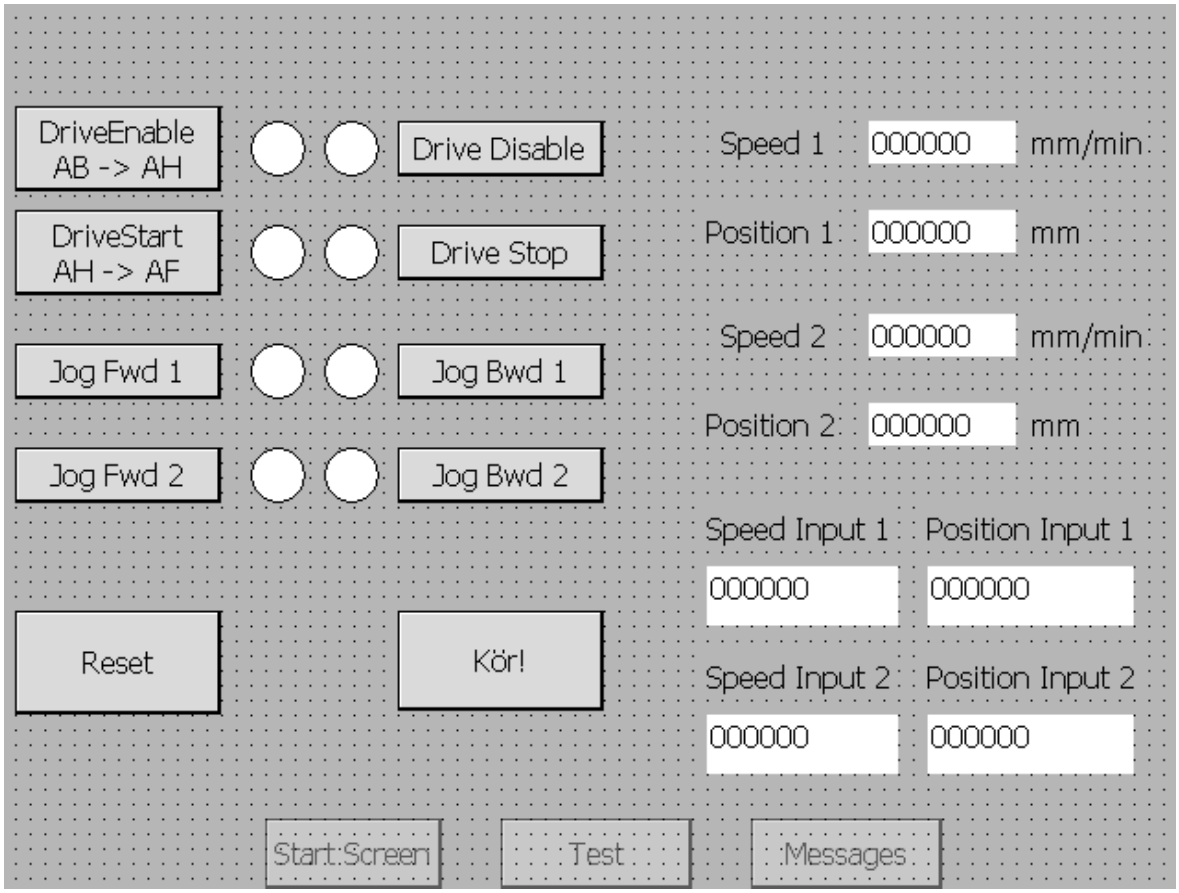
Name	Connection	Data...	Address	Symbol
FROMPLC.DriveEnable	CPU 314C-2 DP	Bool	DB 3 DBX 0.0	DriveEnable
FROMPLC.DriveStart	CPU 314C-2 DP	Bool	DB 3 DBX 0.1	DriveStart
FROMPLC.Homing	CPU 314C-2 DP	Bool	DB 3 DBX 0.2	Homing
FROMPLC.JogBackwards	CPU 314C-2 DP	Bool	DB 3 DBX 0.3	JogBackwards
FROMPLC.JogForwards	CPU 314C-2 DP	Bool	DB 3 DBX 0.4	JogForwards
FROMPLC.kor	CPU 314C-2 DP	Bool	DB 3 DBX 1.1	kor
FROMPLC.OnlinePLC	CPU 314C-2 DP	Bool	DB 3 DBX 0.6	OnlinePLC
FROMPLC.Position_out	CPU 314C-2 DP	DInt	DB 3 DBD 6	Position_out
FROMPLC.Reset	CPU 314C-2 DP	Bool	DB 3 DBX 0.7	Reset
FROMPLC.speed_out	CPU 314C-2 DP	DInt	DB 3 DBD 2	speed_out
FROMPLC2.DriveEnable	CPU 314C-2 DP	Bool	DB 5 DBX 0.0	DriveEnable
FROMPLC2.DriveStart	CPU 314C-2 DP	Bool	DB 5 DBX 0.1	DriveStart
FROMPLC2.Homing	CPU 314C-2 DP	Bool	DB 5 DBX 0.2	Homing
FROMPLC2.JogBackwards	CPU 314C-2 DP	Bool	DB 5 DBX 0.3	JogBackwards
FROMPLC2.JogForwards	CPU 314C-2 DP	Bool	DB 5 DBX 0.4	JogForwards
FROMPLC2.kor	CPU 314C-2 DP	Bool	DB 5 DBX 1.0	kor
FROMPLC2.position_out	CPU 314C-2 DP	DInt	DB 5 DBD 6	position_out
FROMPLC2.Reset	CPU 314C-2 DP	Bool	DB 5 DBX 0.6	Reset
FROMPLC2.speed_out	CPU 314C-2 DP	DInt	DB 5 DBD 2	speed_out
HMI Screen.StartScreen	CPU 314C-2 DP	Bool	DB 11 DBX 0.0	StartScreen
HMI Screen.Test	CPU 314C-2 DP	Bool	DB 11 DBX 0.3	Test
HMI.activate_startPos	CPU 314C-2 DP	Bool	DB 2 DBX 10.5	activate_startPos
HMI.DriveEnable	CPU 314C-2 DP	Bool	DB 2 DBX 10.0	DriveEnable
HMI.DriveSTart	CPU 314C-2 DP	Bool	DB 2 DBX 10.1	DriveSTart
HMI.Homing	CPU 314C-2 DP	Bool	DB 2 DBX 10.2	Homing
HMI.JogBackwards	CPU 314C-2 DP	Bool	DB 2 DBX 0.0	JogBackwards
HMI.JogForwards	CPU 314C-2 DP	Bool	DB 2 DBX 0.1	JogForwards
HMI.Kor	CPU 314C-2 DP	Bool	DB 2 DBX 0.2	Kor
HMI.PM	CPU 314C-2 DP	Bool	DB 2 DBX 0.4	PM
HMI.Position	CPU 314C-2 DP	DInt	DB 2 DBD 2	Position
HMI.Reset	CPU 314C-2 DP	Bool	DB 2 DBX 0.5	Reset
HMI.speed	CPU 314C-2 DP	DInt	DB 2 DBD 6	speed
HMI.StartPos	CPU 314C-2 DP	Bool	DB 2 DBX 10.4	StartPos
HMI.Test	CPU 314C-2 DP	Bool	DB 2 DBX 10.3	Test
HMI2.DriveEnable	CPU 314C-2 DP	Bool	DB 4 DBX 10.0	DriveEnable
HMI2.DriveStart	CPU 314C-2 DP	Bool	DB 4 DBX 10.1	DriveStart
HMI2.Homing	CPU 314C-2 DP	Bool	DB 4 DBX 10.2	Homing
HMI2.JogBackwards	CPU 314C-2 DP	Bool	DB 4 DBX 0.0	JogBackwards
HMI2.JogForwards	CPU 314C-2 DP	Bool	DB 4 DBX 0.1	JogForwards
HMI2.kor	CPU 314C-2 DP	Bool	DB 4 DBX 0.2	kor
HMI2.Position	CPU 314C-2 DP	DInt	DB 4 DBD 2	Position
HMI2.Reset	CPU 314C-2 DP	Bool	DB 4 DBX 0.4	Reset
HMI2.speed	CPU 314C-2 DP	DInt	DB 4 DBD 6	speed
test 1.kor	CPU 314C-2 DP	Bool	DB 8 DBX 4.0	kor
test 1.kor2	CPU 314C-2 DP	Bool	DB 8 DBX 14.0	kor2
test drive1.position_out	CPU 314C-2 DP	DInt	DB 9 DBD 6	position_out
test drive1.speed_out	CPU 314C-2 DP	DInt	DB 9 DBD 10	speed_out
test drive2.position_out	CPU 314C-2 DP	DInt	DB 10 DBD 6	position_out
test drive2.speed_out	CPU 314C-2 DP	DInt	DB 10 DBD 10	speed_out

7.2.2 Menyerna

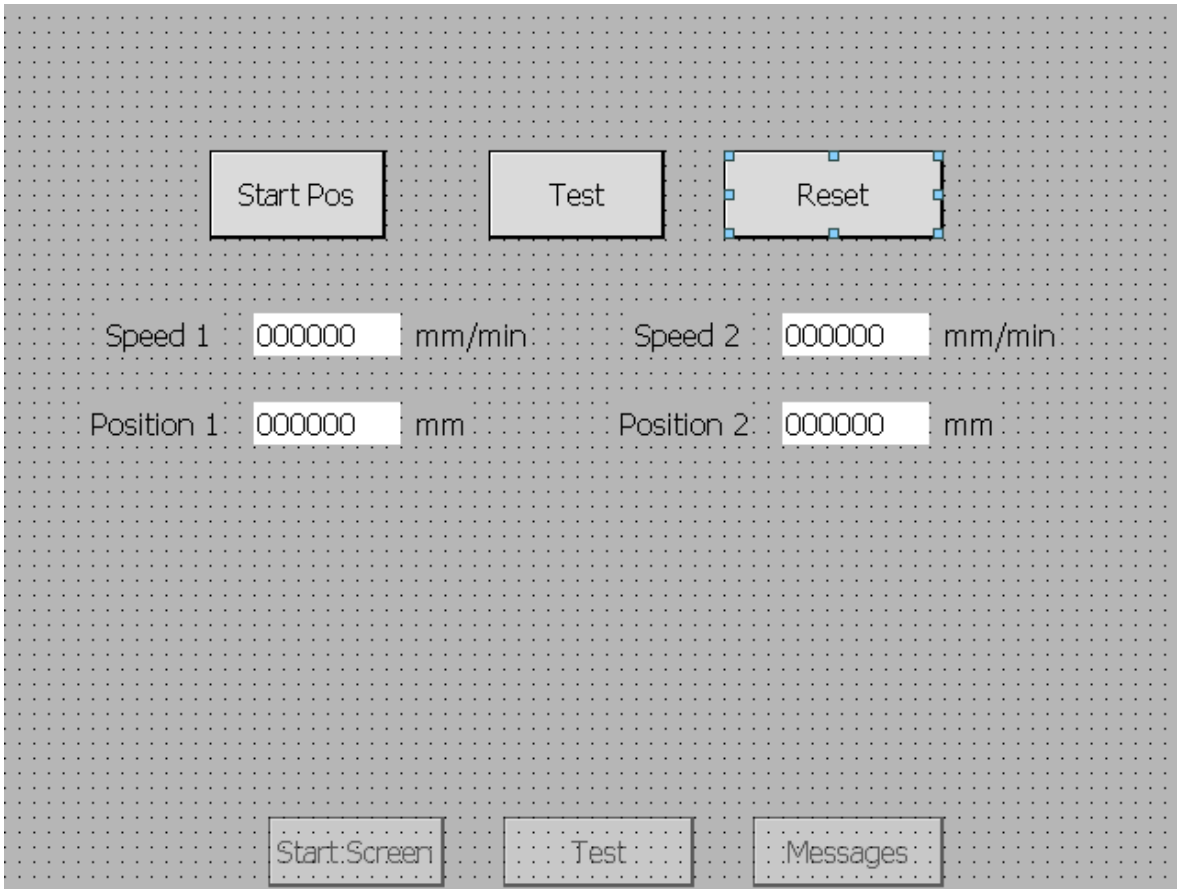
Intro - Skärmen som visas vid uppstart av HMI.



Template – Denna skärmen ligger som underlag, allting som läggs till här är åtkomligt från samtliga skärmar.



Start Screen – För individuell reglering av skenorna.



Test - Härifrån körs testsekvensen som kör skenorna automatiskt.

Messages – För att kontrollera eventuella felmeddelanden