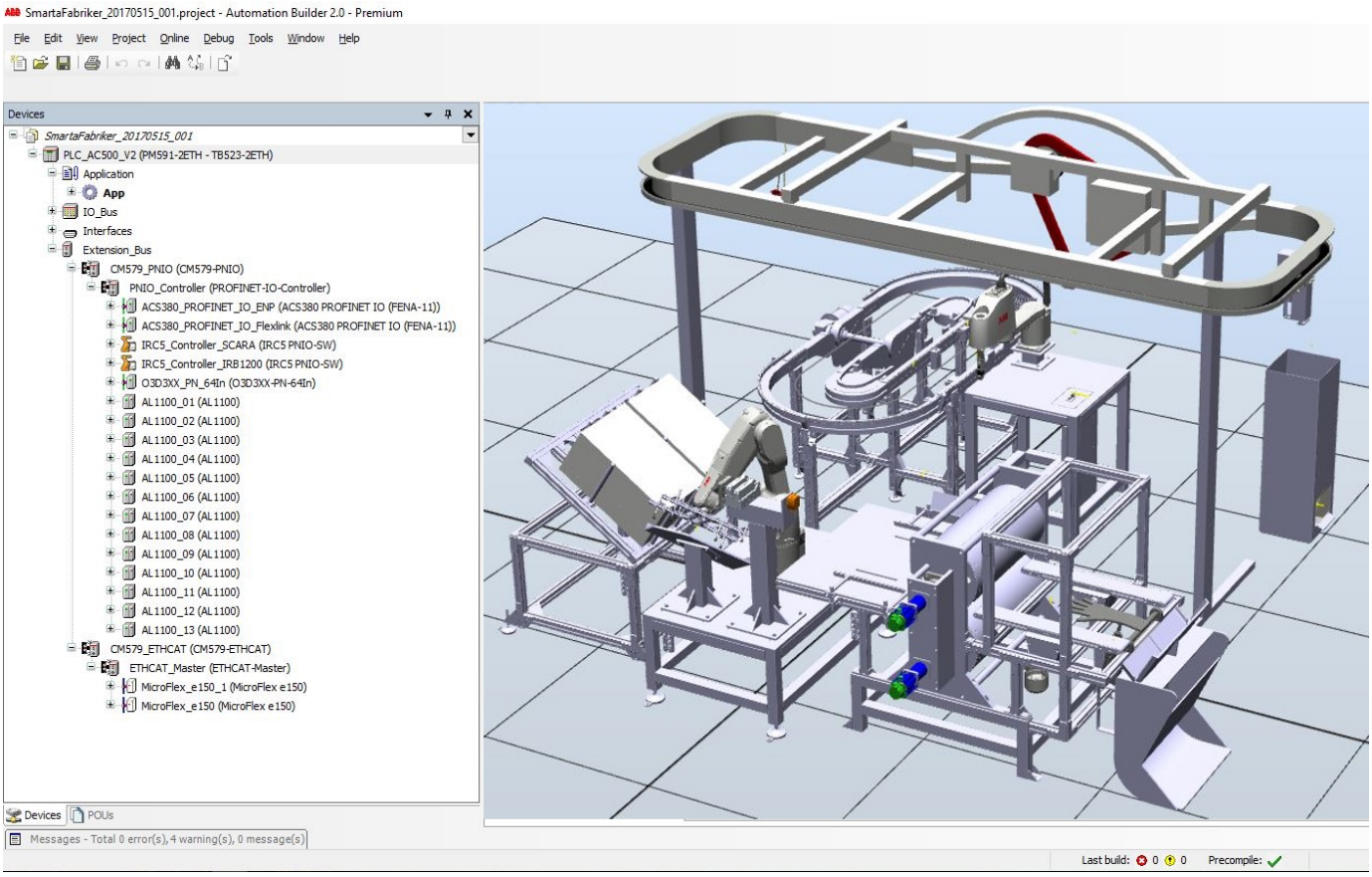
CHALMERS



PLC-programmering av automatiserad process för Smarta Fabriker

PLC-programming of an automated process for Smarta Fabriker

Examensarbete inom högskoleingenjörsprogrammet Mekatronik

ERIKA LINDQVIST
JULIA LINDHOLM

Institutionen för Signaler och system
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2017

EXAMENSARBETE INOM HÖGSKOLEINGENJÖRSPROGRAMMET
MEKATRONIK

PLC-Programmering av automatiserad process för Smarta Fabriker

ERIKA LINDQVIST
JULIA LINDHOLM

CHALMERS

Institutionen för signaler och system
Chalmers tekniska högskola
Göteborg, Sverige 2017

PLC-Programmering av automatiserad process för Smarta Fabriker
Examensarbete inom högskoleingenjörsprogrammet Mekatronik
ERIKA LINDQVIST
JULIA LINDHOLM

Handledare/Examinator: Veronica Olesen, Signaler och System

Examensarbete 2017 Institutionen för signaler och system

Chalmers tekniska högskola
SE-412 96 Göteborg
Telefonnummer +46 31 772 1000

Förstasida:
Projektet i Automation Builder och den virtuella fabriken.

[Egen bild]
Göteborg, Sweden 2017

Förord

Den 5e oktober 2016 besökte vi, två Mekatronikstudenter på Chalmers Tekniska Högskola, Scanautomatic mässan i Göteborg. Där stod Göteborgs Tekniska College och visade sitt projekt Smarta Fabriker. Vi fick tala med Johan Bengtsson, övergripande projektledare, och han berättade om projektets syfte och mål, att de skulle bygga upp en fabrik tillsammans med studenter. Vidare kontaktades Richard Hedman, projektledare fabrik och exjobb, vilket senare ledde till ett examensarbete inom PLC-programmering. Projektet har samarbetat med 6 huvudaktörer och 50 partners varav vi har samarbetat tätt ihop med ÅF och ABB. Vi vill säga speciellt tack till Johan och Richard som gett oss chansen att vara delaktiga i detta projekt, våra tre handledare på ÅF, Andreas Buhlin, Niels Glamheden och Per Ström. De som hjälp oss med programvaran och programmeringen på ABB, Mats Bengtsson, Björn Magnusson och Jonathan Schwartz.

Erika Lindqvist och Julia Lindholm, Göteborg, maj 2017

Sammanfattning

För att öka intresset för tekniska studier och dra åt sig ungas intresse att välja arbete inom industrin skapar Smarta Fabriker en plattform för att sprida kunskap inom industriell digitalisering. Denna plattform består av en minifabrik.

I detta arbete genomförs programmeringen av processen i samarbete med ÅF och ABB. Programmeringen är genomförd i styrsystemet PLC och innefattar flera olika lösningar så som transportlösningar, robotkommunikation, kommunikation mot överordnat system, IO-Link och frekvensomriktare. Slutprodukten till beställaren är ett par så kallade cardboard, ett par VR-glasögon (Virtual Reality) gjorda av kartong. Genom en virtuell PLC kopplas programmet upp och testas mot en dynamisk modell, en virtuell fabrik, uppbyggt av andra studenter. I den virtuella fabriken görs en virtuell driftsättning där alla fabrikens funktioner testas. Efter godkänd virtuell driftsättning är programmet provkört och testat och kan smidigt implementeras i den verkliga fabriken.

Abstract
In order to increase the interest for technical studies and to encourage the interest of young people to work in industries, Smarta Fabriker wanted to create a platform to spread knowledge about industrial digitalization. This platform contains a minifactory.

In this project the task is to perform the programming of the factory in cooperation with ÅF and ABB. The programming is executed in PLC and contains several solutions like transportation, robot communication, communication a superior system, IO-Link and drives. The product that will be delivered to costumer is a cardboard, a pair of VR-glasses. Through a virtual PLC the program is connected and tested to a dynamic model, a virtual factory, built by other students. Then a virtual factory acceptance test is done where all the functions in the program are tested. This implicates that the program is ready to be implemented in the real factory.

Innehåll

Förkortningar

DPS	Diab Produktionssystem
FBD	Function Block Diagram
SFC	Sequence Function Chart
PLC	Programmable Logic Controller
QR	Quick Response
ST	Structured Text
VFAT	Virtual Factory Acceptance test
VirtCom	Virtual Commissioning
VR	Virtual Reality

1. Inledning

Regeringen presenterade i januari 2016 en nyindustrialiseringsstrategi [1]. Denna strategi talar om hur industrin står inför ett paradigmskifte som drivs av globalisering, digitalisering och omställningen mot en grön resurseffektiv ekonomi, vilket ger stora möjligheter att utveckla en smartare och mer hållbar industri och smarta arbetsplatser där automation tillsammans med människan skapar en hög konkurrenskraft. Ett huvudfokusområde för strategin är kunskapslyft inom industrin, som innefattar såväl kunskap om modern produktion som industriell digitalisering.

1.1 Bakgrund

Smarta Fabrikers syfte och mål med projektet är att skapa en plattform för att sprida kunskap om industriell digitalisering för att öka intresset för tekniska studier och intresset för ungdomar att välja arbeten inom industrin.

Smarta Fabriker drivs av Göteborgs Tekniska College som fungerar som en länk mellan projektets olika aktörer. Dessa aktörer utgörs av företag, akademi, organisationer och skolor. Idén med projektet är att bygga en demonstrator (minifabrik) med tillhörande utställning tillsammans med Universeum i Göteborg. Demonstratorn ska användas i aktiviteter kopplade till kompetensutveckling och kompetensförsörjning inom industriell digitalisering. Arbetsgången delas in i tre faser, rigga, bygga och sprida.

En av grundtankarna med projektet är att utveckla och driva byggfasen genom studentarbeten. Detta för att skapa förebilder och inspirera unga till tekniska studier. Tio examensarbeten på högskolenivå tillsammans med deltagande företag är kopplade till projektet. Den mekaniska konstruktionen av fabriken görs av en klass blivande CAD-konstruktörer från YRGO. De tio examensarbetena representerar flera funktioner. Några av dessa är simulering, elkonstruktion, programmering, smart underhåll och hållbar produktion. Examensarbetarna är utspridda på olika företag men synkar varje vecka på planerings- och uppföljningsmöten på Visual Arena - Lindholmen Science Park. Detta examensarbete handleds av ÅF. Under maj kommer minifabriken att byggas. Totalt innehåller Projektet Smarta Fabriker cirka 21 000 timmar studentarbete.

Tanken med minifabriken är att beställaren ska få ett par så kallade cardboard, ett par VR-glasögon gjorda av kartong. I minifabriken ska arket först plockas av en robot, därefter transportera arket förbi en skrivare och en läsare som ger arket identifiering och läser om skrivningen blev godkänd. Roboten lämnar sedan arket på en inmatning. Inmatningen puttar arket in till två valsar som ska skära ut mönstret för cardboarden. Efter valsarna finns ett transportband som transporterar arket till slutet av den första anläggningen. I slutet av anläggning 1 finns ett ställdon som vänder ned arket i en brevlåda där beställaren kan hämta sitt ark och vika ihop det efter beskrivningar vid arbetsstationer. En annan anläggning, anläggning 2, ska parallellt med anläggning 1 hanterar linserna som ska sitta i cardboarden. Beställaren ska föra in sina händer i ett utkast och då ska en lins släppas ned. Anläggning 2

innefattar bl.a. två transportlösningar från Flexlink och Eton, en robot från ABB och ett ställdon från SKF.

1.2 Syfte och mål

Projektets syfte är att skapa en plattform för att sprida kunskap om industriell digitalisering för att öka intresset för tekniska studier och dra åt sig ungdomars intresse till att välja arbeten inom industrin.

Målen med detta delprojekt är ett PLC-programmerat styrsystem av fabriken samt tillsammans med examensarbetet inom Virtual Commissioning visa fabrikens arbetsgång i en dynamisk modell genom en virtuell PLC.

1.3 Avgränsningar

Examensarbetet behandlar enbart PLC-programmering, alltså behandlas inte virtuell testning, robotprogrammering, programmering av ställdon och servo, underhåll och installation av fabriken. Tidsramen för projektet är halvtid i 16 veckor.

1.4 Precisering av arbetet

PLC-programmeringen ska genomföras i ABB:s programvara Automation Builder, där programspråket kan variera. Flödet preciseras i ett flödesschema och uppdelas i fyra moduler för att underlätta arbetsgången. Digital planering och synkronisering med andra arbeten i projektet sker under arbetets gång.

2. Teoretisk bakgrund

De tekniska komponenterna beskrivs, uppdelat i två delkapitel, hårdvara och mjukvara.

2.1 Hårdvara

Detta delkapitel innefattar teoretisk bakgrund kring hårdvaran som används i projektet.

2.1.1 Frekvensomriktare

En frekvensomriktare används för att styra en elektrisk motors varvtal genom att variera frekvens och spänningstillförseln till denna. Detta ger minskad energiförbrukning eftersom frekvensomriktare anpassar hastigheten genom att variera frekvensen. Utan frekvensomriktare körs elektriska motorer i full fart och bromsas ned till rätt hastighet vilket sliter mycket på utrustning och ger en högre energiförbrukning [2].

2.1.1.1 ABB MicroFlex e150

MicroFlex e150 är en högpresterande servofrekvensomriktare. Den har ett integrerat EtherCAT-gränssnitt. Den har ett standard Ethernetprotokoll för att ge fullständig flexibilitet med PLC-systemet och andra PC-lösningar som programmerbara Motion Control-system. Den har 2 digitala utgångar och fyra digitala ingångar, där två av dessa programmeras att detektera axelposition inom 1/µs, [3].

2.1.1.2 ABB ASC380

ASC380 är en skalbar frekvensomriktare, speciellt optimerad för maskinapplikationer och maskinbyggare. Denna programmeras i ABB:s PLC-system genom förkonfigurerade maskindrivpaket [4].

2.1.3 I/O-link

IO-Link är den första standardiserade IO teknologin (IEC 61131-9) för kommunikation med sensorer och ställdon. Den används för att lättare kunna utnyttja all information som sensorerna kan erbjuda. Istället för att bara få ut booleska värden på om givaren är aktiverad eller inte kan man exempelvis få reda på hur sensorn mår och hur smutsig den är. Information som detta kan vara väldigt viktig ur ett underhållsperspektiv [5].

2.2 Mjukvara

Programvaror som används, kommunikationsbussar och överordnat system förklaras i detta delkapitel.

2.2.1 Automation Builder

Automation Builder är ett programpaket från ABB som underlättar automatisering av maskiner och system på ett effektivt sätt. Automation Builder tillhandahåller verktyg som behövs för att programmera, konfigurera, felsöka och hantera projekt genom ett gemensamt

gränssnitt. För programmering av PLC använder sig Automation Builder av Codesys som är ett komplett IEC 61131-3 programsystem, [6].

2.2.2 Robot Studio

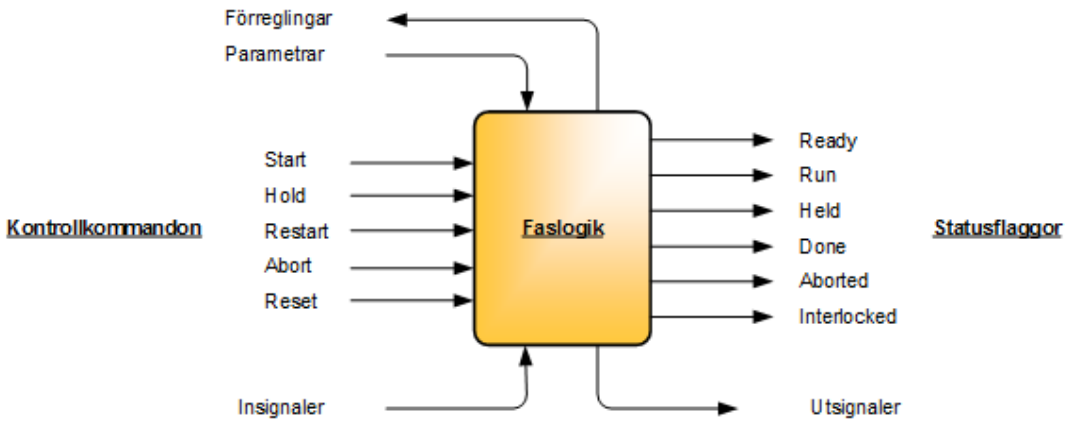
Robot Studio är en programvara från ABB som används för simulering och offline-programmering. Programmering av robotar sker i detta program. Programvaran innehåller en exakt kopia av den programvara som finns i den verkliga roboten ute i produktion. Realistiska simuleringar görs med riktiga robotprogram och konfigurationsfiler som är identiska med dem som används på verkstadsgolvet, [7].

2.2.3 Kommunikationsbussar

Kommunikationsbussar som används här är EtherCAT, Profinet och TCP/IP. Fältbussarna används för sammankoppling av automationsutrustning, PLC, I/O-Link, robotar, frekvensomvandlare m.fl. EtherCAT och Profinet är Ethernet baserade, innebär att datakommunikationen sker på samma sätt. EtherCAT används för hård- och mjukvaror med realtidskrav [8]. Profinet gör bäst nytta då det finns en tidsbegränsning. En tidsbegränsning som handlar om 1ms eller mindre. TCP/IP hanterar datakommunikation över nätverk.

2.2.4 Överordnat system eLIPS

Diab Produktiontionssystem sköter överordnad styrning, hanterar och presenterar information från flera olika system. Exempelvis kan ett överordnat system hantera ordrar som styr produktionen. Underliggande utrustning återrapporterar och (vid behov) larmar. Utrustningen (PLC) och DPS (PC) kommunicerar via nätverk, Ethernet TCP/IP. DPS ansluter till utrustning via en OPC-server. OPC-server används för kommunikation mellan olika datorer och i detta fall används OPC-DA som står för Data Access. En OPC-server samlar data och skickar vidare denna till en OPC-klient. Klienten kan antingen fråga efter data eller prenumerera på data [9]. Om DPS ska skicka specifika kommandon, som begäran att starta operationer m.m. används fasfunktioner. Fasfunktionerna hanterar start, stopp, halt, avbryt och återstart på ett standardiserat sätt. En fasfunktion har alltid en specifik uppgift. Figur 1 visar en bild på ett blockdiagram över ett faslogikblock.



Figur 1 Blockdiagram över ett faslogikblock.

DPS kan använda fem kontrollkommandon för att styra en fas och det finns sex statusflaggor som fasen kan sätta som svar på kommandona. Endast en statusflagga får vara satt åt gången och endast ett kontrollkommando åt gången får skickas [10].

3. Metod

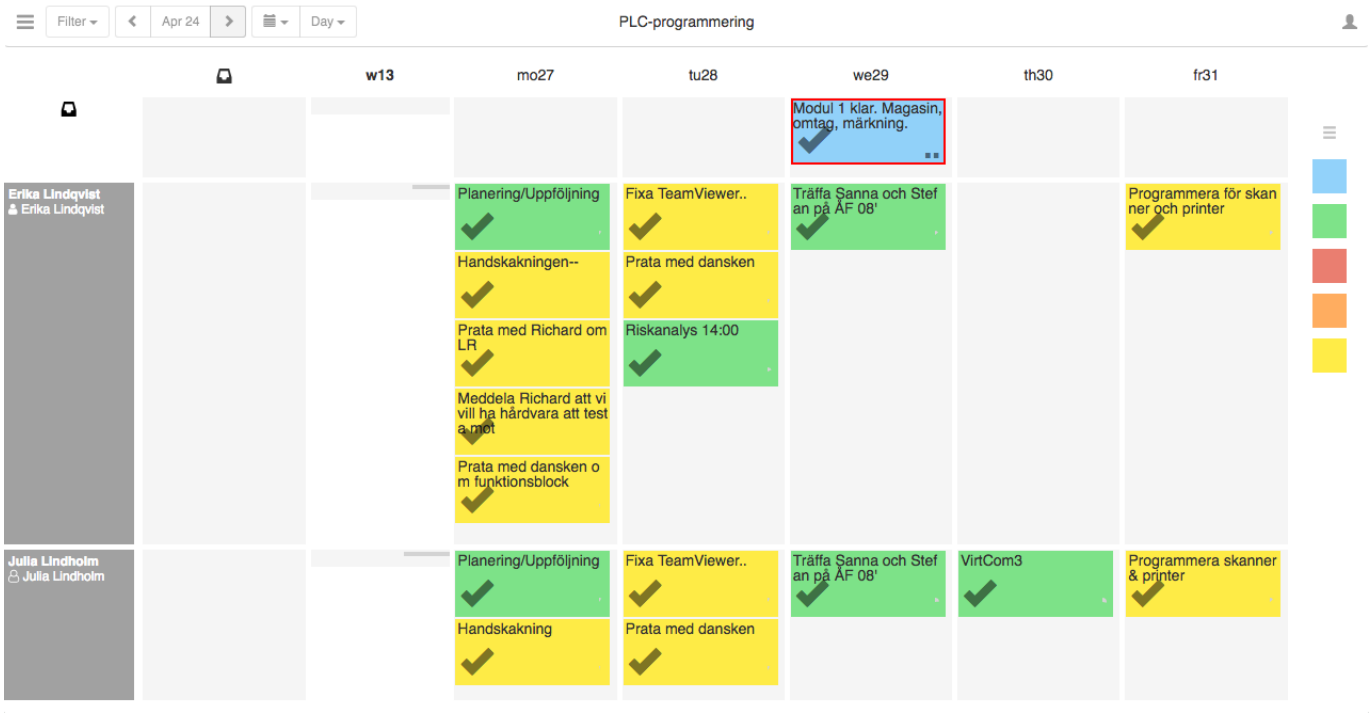
Metodkapitlet beskriver utförligt hur arbetet är upplagt.

3.1 Systemeringsarbetet

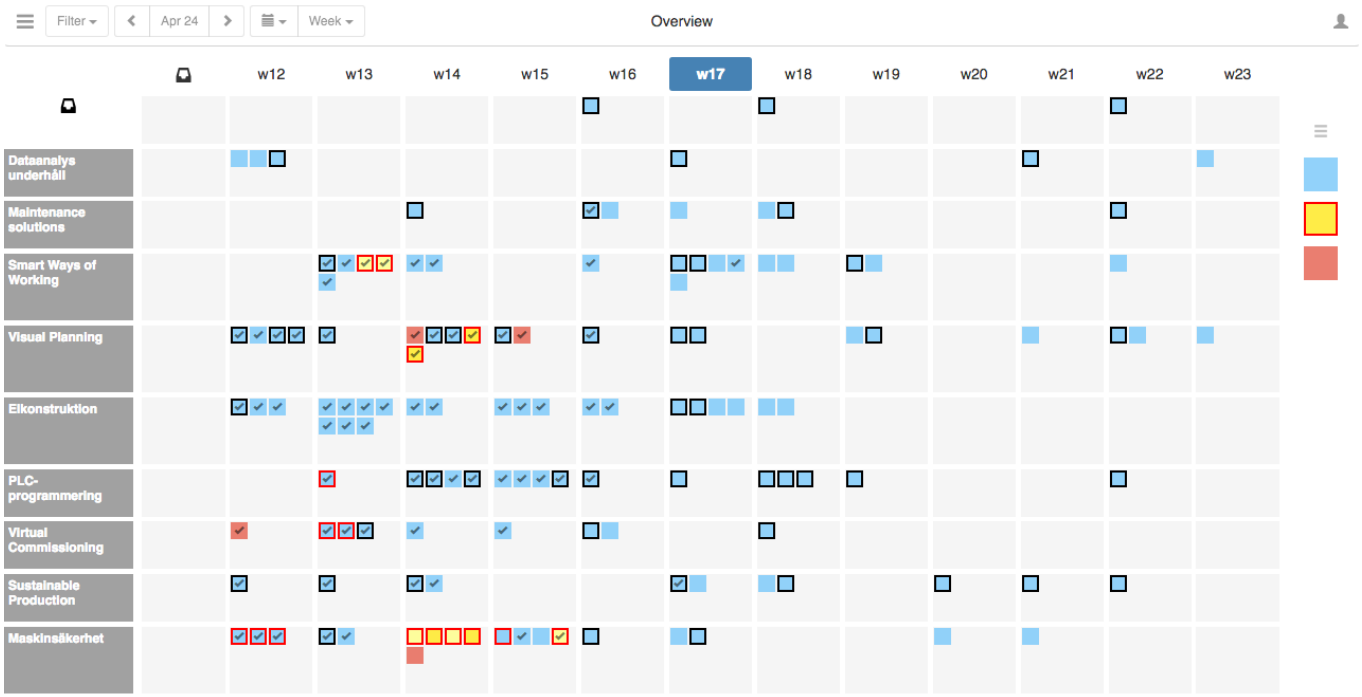
Tidigt i projektet påbörjades systemeringsmöten där förslag på lösningar för funktioner togs upp och återkommande genomgångar av flödet. Det har varit tre stycken så kallade VirtCom möten, på dessa har det tagits upp problem som uppstått i programmeringen eller från hårdvaror. Varje möte har fokuserat på en specifik del av flödet, se flödesschemat i bilaga 1. De som närvarat på mötena är alla studenter som gör examensarbeten, några partners, handlare från företagen och ansvariga för projektet. Att så många närvarat har gjort att problem lysts upp i tidigt skede och rätt person har tagit sig ann att hjälpa till för att lösa det.

3.2 Digital planering

Planeringen av arbetsgången i projektet har skett digitalt genom ett verktyg som kallas Yolean. Verktyget motsvarar en whiteboard-tavla som är uppdelad i veckor och dagar. Milstolpar, mål och anteckningar markeras med post it lappar. Olika färger på lapparna identifierar vad de står för. Alla grupper har varsin rad där de lägger egna lappar som hör till deras arbetsuppgift. Att detta görs digitalt innebär att redigering kan ske vart du än är, det blir enklare att se/göra ändringar. Då flera examensarbeten är beroende av varandra är det viktigt att snabbt se ifall ett problem uppstår och hur det påverkar resterande parter. I figur 2 och 3 nedan ses ett exempel på hur en vy med en vecka inom en grupp kan se ut och en övergripande vy över alla grupper i projektet.



Figur 2, En veckas vy för en grupp i Yolean



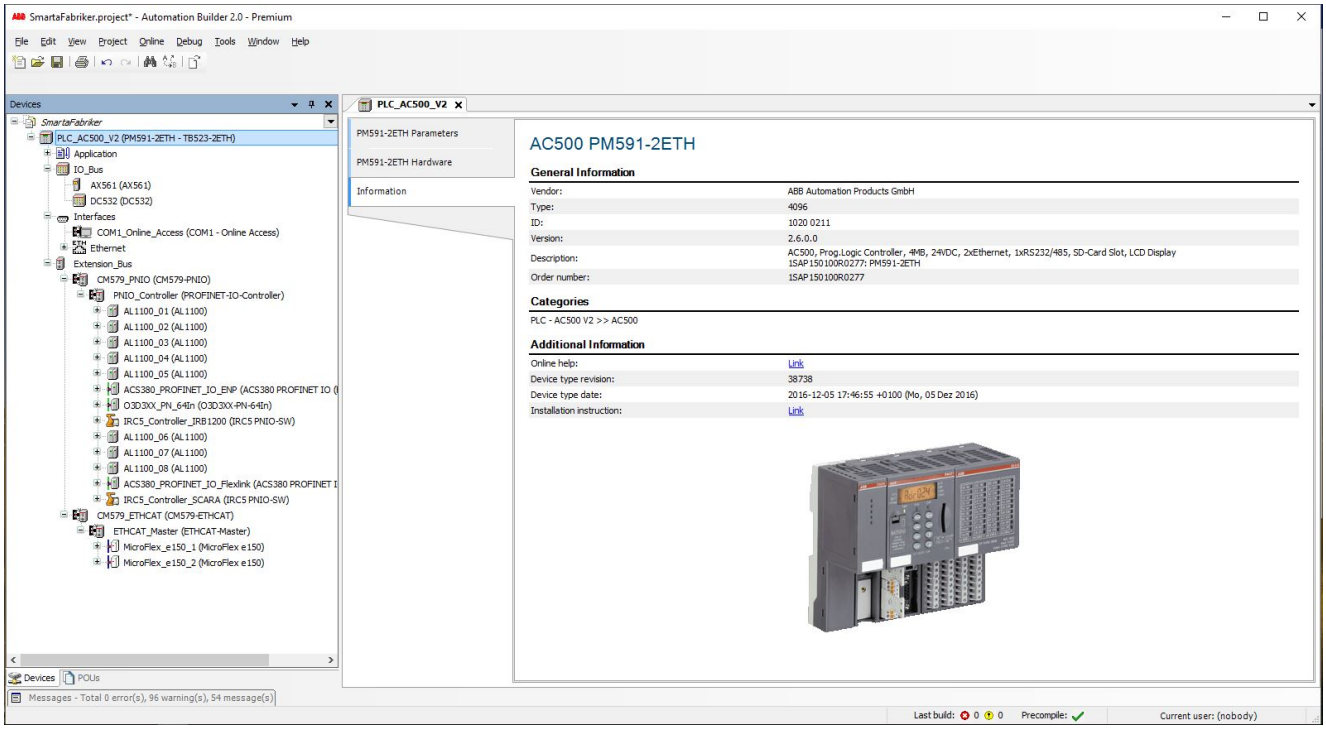
Figur 3, Överblick över alla gruppers milstolpar och mål över flera veckor i Yolean

3.3 Hårdvara

Eftersom projektet har många företag som stöttar finns det en mängd olika fabrikat på hårdvaran i fabriken. All hårdvara har valts ut med hänseende till funktion och användningsområde. De flesta givare som används är från IFM och då används främst fotoceller och ultraljudsgivare. Det finns även tre induktiva givare på flexlinkbanan som är från SICK. För märkning av kartongark behövdes en skrivare och en QR-kodsläsare och då valdes en skrivare från Matthews och en multikodsläsare från IFM. För att kunna förflytta kartongarket från ett magasin, förbi skrivaren och läsaren och placera på en inmatning användes en IRB 1200-robot [11] från ABB. Denna robot har en räckvidd på 0.9 m och en lyftförmåga på 5 kg. Till förflyttning av linserna från en Flexlink transportbana till en fixtur användes istället en SCARA-robot [12], även denna från ABB. SCARA-roboten är konstruerad för att klara av mer allmänna tillämpningar som punkt till punkt rörelser. Den har lastkapacitet på 6 kg och räckvidd på 650 mm. De båda robotarna har en inbyggd robot controller, IRC5 PNIO-SW, som kommunicerar mot Profinet, dessa controllers implementeras hos PLC:n för kommunikation mot denna. Ställdonen CASM som används i projektet är elektromekaniska och kommer från SKF.
För att rullstansen ska kunna röra på sig används en frekvensomriktare för servomotorer (MicroFlex e150) från ABB. För att kunna förflytta ark och linser används frekvensomriktare (ASC380) från ABB för att styra utmatningsbandet och flexlinkbanan.
I/O-link används till alla givare och ställdon för att underlätta underhåll och övrig informationshämtning.

3.4 Starta ett projekt i Automation Builder

Efter systemeringsarbetet, när beslut om hårdvara och processens flöde är taget, så skapas ett projekt i Automation Builder. Detta är mjukvaran som används för all programmering. Automation Builder fungerar så att först läggs all hårdvara in, PLC, Robot Controller, IO-Link, frekvensomvandlare, servo osv. Sedan genereras alla variabler till respektive in-/utgång på inlagda komponenter. Detta görs för att senare kunna programmera mot dessa i CodeSys. CodeSys hanterar all programmering och är direkt uppkopplat mot Automation Builder. Programfilerna delas upp efter funktioner och syfte. Figur 4 visar hur ett uppstartat projekt i Automation Builder ser ut.



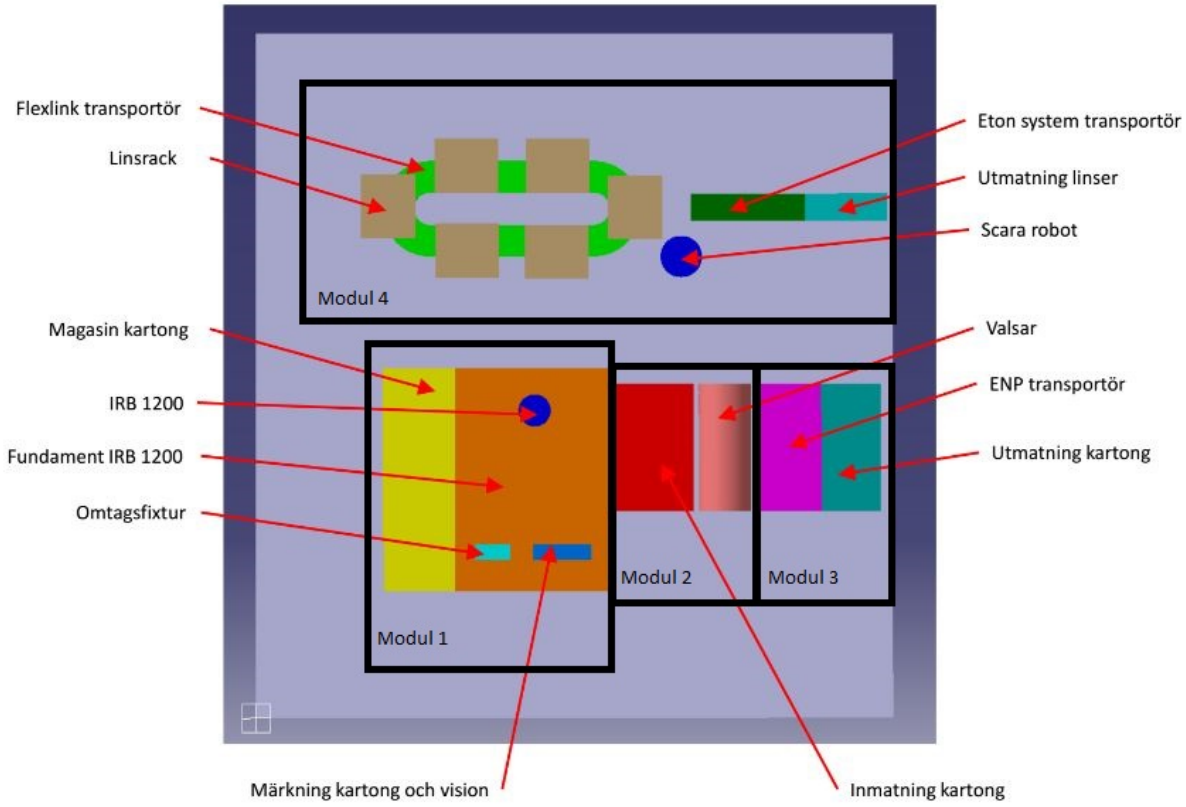
Figur 4, Startat projekt i Automation Builder med inlagda komponenter

3.5 Test mot Virtuella Fabriken

Den virtuella fabriken finns uppbyggd och programmerad i programmet Robot Studio, detta är gjort av ett annat examensarbete. Programmet för styrsystemet som är skrivet i Automation Builder synkas med denna virtuella fabrik. Signaler testas och en tydlig bild på flödesgången ges. Detta förarbete med test mot en virtuell fabrik leder till att problem hittas tidigt och blir lättare (och billigare) att lösa i detta stadie än om det skulle testas mot en redan byggd fabrik.

4. Genomförande

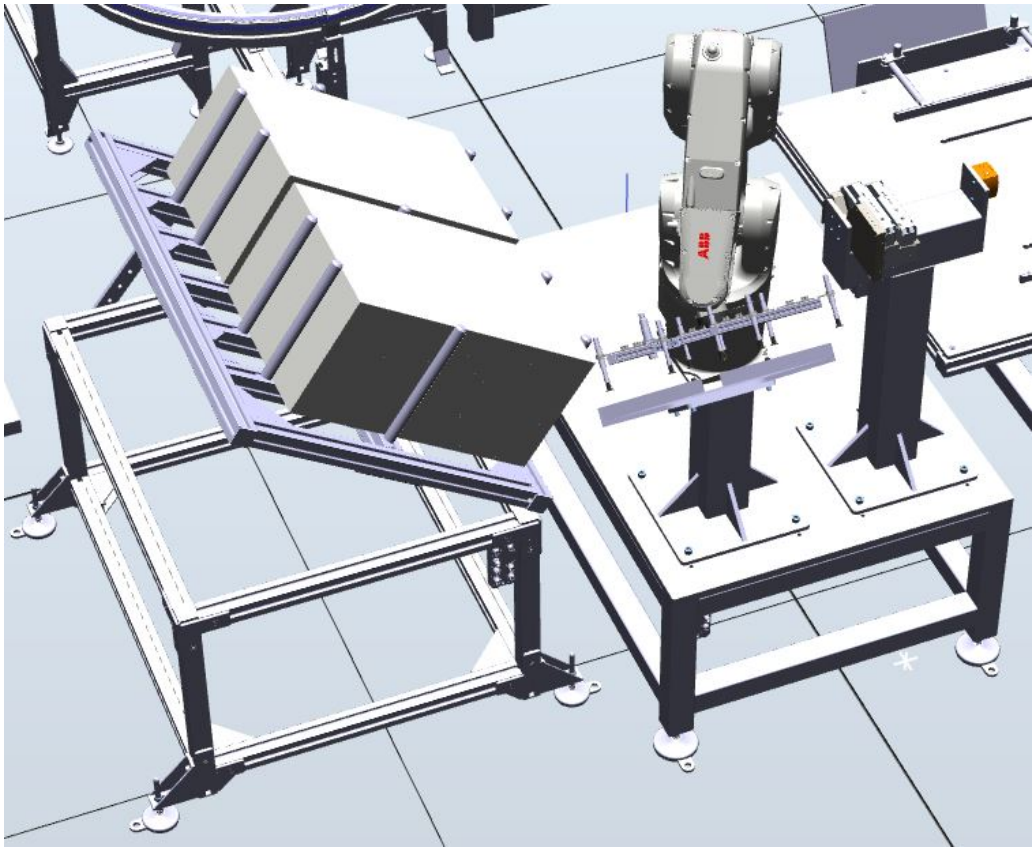
Kapitel fyra beskriver hur programmeringen har genomförts uppdelat modulvis i totalt fyra moduler. Modul 1-3 innefattas i anläggning 1 för kartongarket, modul 4 innefattas i den andra anläggningen för transport av lins. i figur 5 nedan ses uppdelningen av anläggningarna i moduler. Hela programmet återfinns i bilaga 2.



Figur 5, Uppdelning av fabriken i moduler

4.1 Modul 1

Modul 1 innefattar ett magasin där arken ligger i två fack, en IRB1200 robot från ABB, en fixtur och en skrivare och QR-kodsläsare, se figur 6 på modul 1 som är tagen från den virtuella fabriken uppbyggd av andra studentarbeten. När det överordnade systemet (eLIPS) har fått en beställning ska roboten plocka ett ark från något fack i magasinet. Sedan läggs arket i en omtagsfixtur för att säkerställa att det har tagits upp på korrekt sätt. Därefter för roboten arket framför en skrivare som skriver ut en QR-kod för produktmärkning. Denna QR-kod ska sedan kontrolleras och godkännas av en QR-kodsläsare från IFM. Om koden inte är godkänd så läggs arket i en ”scrap” och kasseras. Om QR-koden är godkänd tar roboten tillbaka arket till skrivaren för att printa beställarens namn. Namnet behöver inte godkännas av läsaren. Efter namnet skrivs ytterligare en QR-kod på arket. Denna kod används för att återvinna de överblivna kartongbitarna som blir kvar efter montering. Nu har arket två QR-koder och ett namntryck och roboten ska då, om båda QR-koderna är godkända, lämna arket på inmatningen i modul 2.



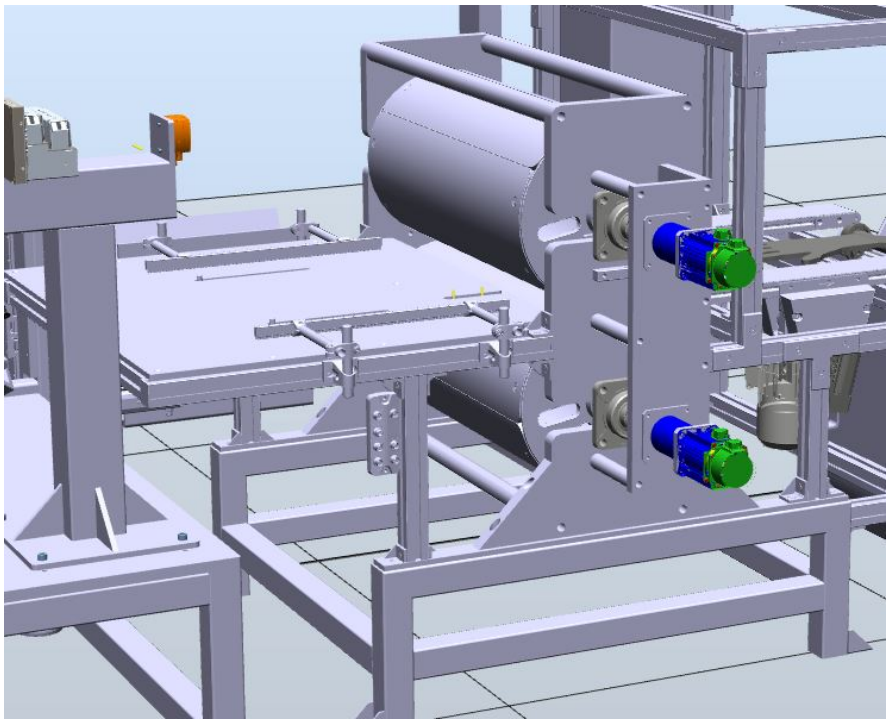
Figur 6, Modul 1 i den virtuella fabriken

För att processen ska starta måste PLC:n skicka till det överordnade systemet att den är redo och är i autoläge. När det överordnade systemet får in en beställning så skickas den till PLC:n som säger åt roboten att börja plocka. För att roboten ska veta i vilket fack den ska plocka måste PLC:n säga åt den var den ska plocka. Detta löstes med hjälp av två fotoceller under respektive fack som då blir ”FALSE” när facket har tomt på kartongark. För att det skulle bli lätt för operatören att fylla på magasinet plockar roboten ur vartannat fack så att båda facken blir tomma samtidigt. För detta användes en räknare som håller koll på i vilket fack roboten ska plocka och hur många ark som har plockats. För att roboten skulle veta var den skulle plocka användes en group-output, en utgång som kan anta flera olika värden.

Roboten kör sedan självständigt, genom ett fristående program utanför PLC:n, till omtagningsfixturen och går till en bestämd position framför skrivaren. Då säger PLC:n till roboten att med konstant hastighet föra arket framför skrivaren samtidigt som det överordnade systemet ger order om att börja skriva första QR-koden som är kartongarkets ID-märkning. När detta är klart ställer roboten arket framför QR-kodsläsaren och PLC:n ber om ett resultat från det överordnade systemet om QR-koden är godkänd eller ej. Om koden är godkänd kommer PLC:n be roboten att fortsätta med att skriva namnet på arket och sedan skriva en QR-kod till som ska läsas på samma sätt som första koden. Om någon av QR-koderna inte skulle vara godkända kommer roboten att lägga arket i en papperskorg för att kasseras. Om båda QR-koderna är godkända lägger roboten kartongarket på inmatningen (Modul 2) och går sedan tillbaka för att börja plocka ett nytt ark igen.

4.2 Modul 2

Modul 2 innefattar en inmatning i form av ställdon från SKF och två valsar (övre och undre), se figur 7 över modul 2.
Inmatningen används för att transportera kartongarket fram till rullstansen som ska simulera hur VR-glasögonen stansas.

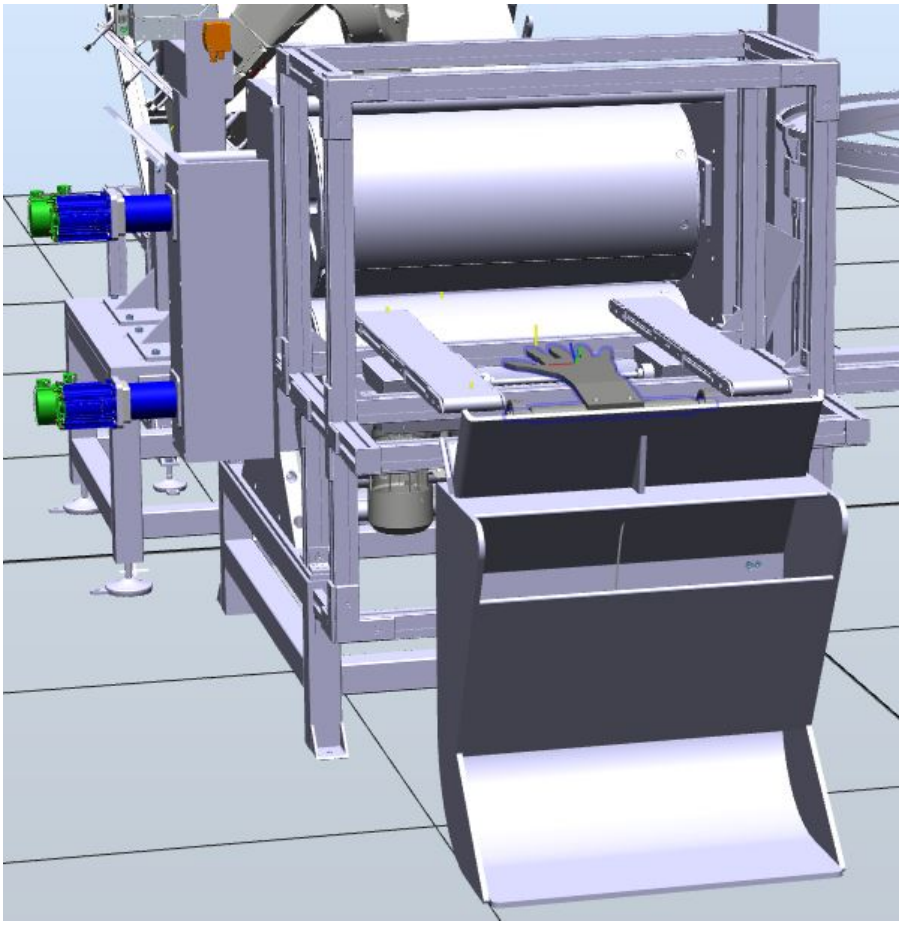


Figur 7, Modul 2 i den virtuella fabriken

Via en fotocell som är placerad på inmatningen som aktiveras till ”TRUE” när kartongarket placeras på inmatningen förs informationen överförs till PLC:n. När roboten har gått till en säker position rör sig ställdonet utåt och för arket fram mot stansen. För att kunna veta när rullstansen ska starta så används en fotocell till för att kunna starta stansen så att den börjar mata igenom arket. Stansen matar igenom arket fram till modul 3.

4.3 Modul 3

Modul 3 innefattar ett transportband från ENP, en vippa som ska slå över arket i en brevlåda, med hjälp av ett ställdon från SKF, se figur 8 på modul 3.
För att kunna leverera det stansade arket till kund behöver det föras ut genom rullstansen och levereras i en ”brevlåda” där kunden kan plocka sin produkt.



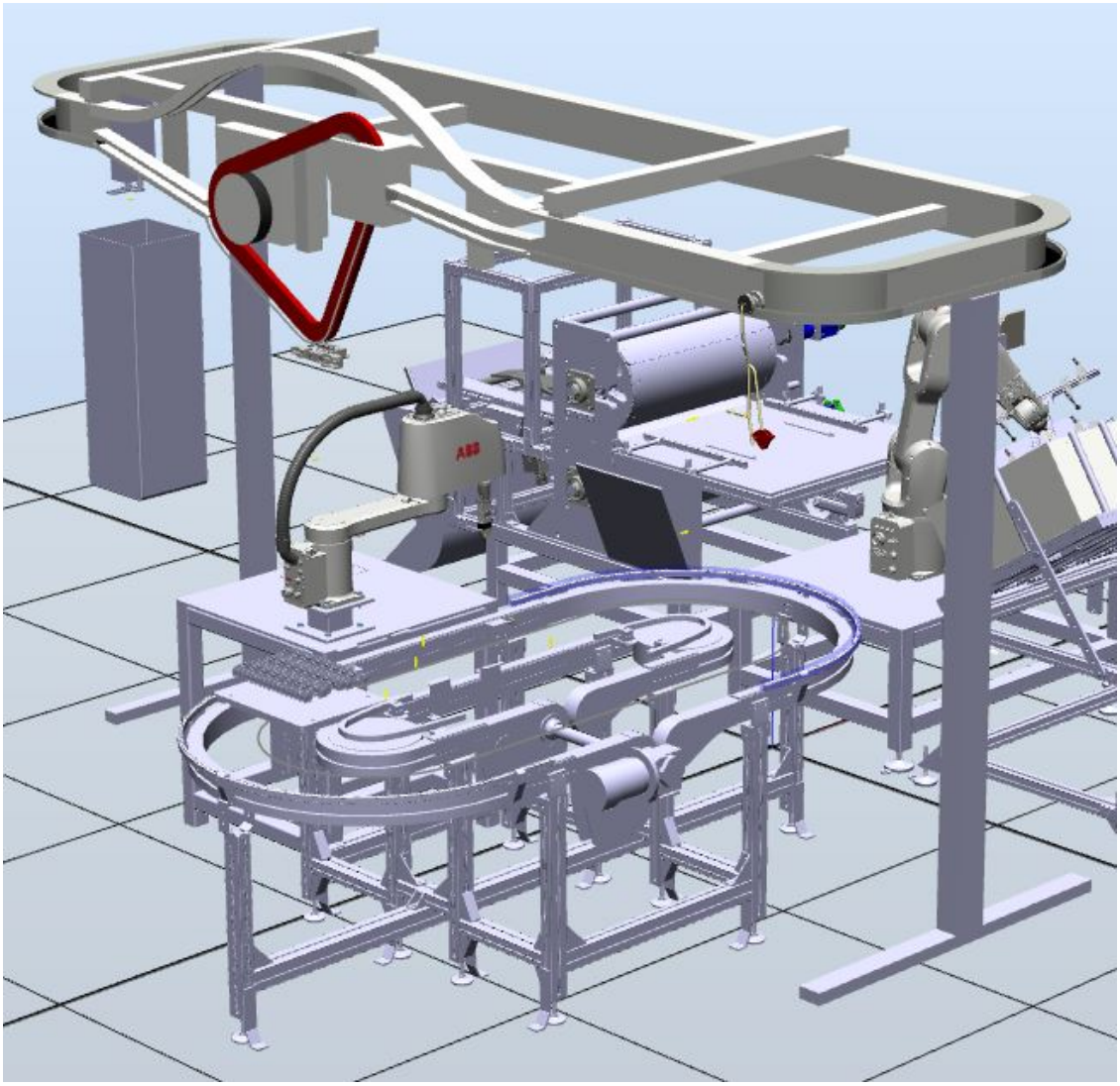
Figur 8, Modul 3 i den virtuella fabriken

När arket kommer igenom stansen så triggas en givare vid transportbandet som ger signal att det ska starta för att ta emot arket. Hastigheten ställs in genom att ange den i procent av den totala kapaciteten. Detta görs sedan om till rpm i PLC-programmet som skickar värdet till frekvensomriktaren. Bandet transporterar fram arket tills det når en givare på ”vippan”, då stannar transportbandet och vippans ställdon aktiveras och vänder ner kartongarket i brevlådan. Då meddelas det överordnade systemet att produkten är utlevererad till beställaren.

4.4 Modul 4

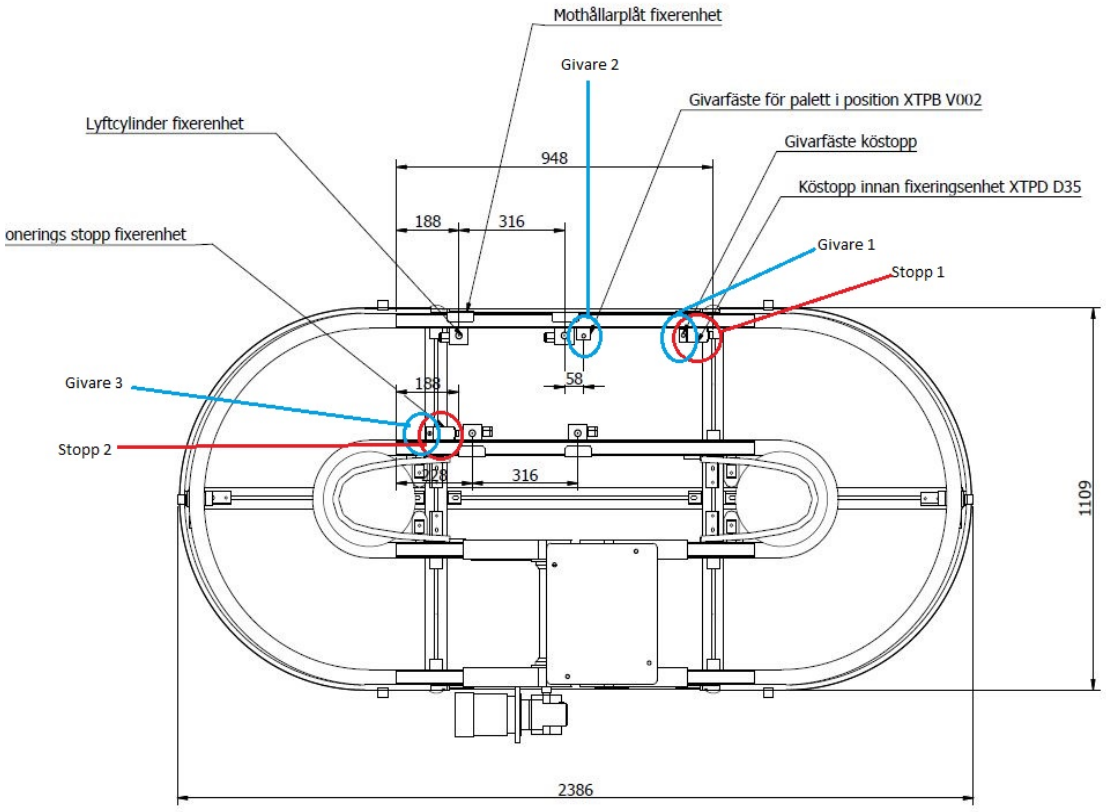
Modul 4 omfattar hela anläggning 2, ett Flexlink och ett Eton system, se figur 9 där hela anläggning 2 visas. Denna anläggning ska leverera linsen som ska placeras i VR-glasögonen ut till beställaren. Anläggning 2 körs parallellt med anläggning 1. Ett transportsystem från Flexlink hanterar fem lådor, placerade på paletter, fyllda med 50 linser vardera som ligger i racks. Då Flexlinkbanan bör vara i rörelse så mycket som möjligt ska en lins plockas sedan ska lådan transporteras ett varv på banan. Efter varvet plockas nästa lins ur samma låda, detta år upprepande tills lådan är tom, då plockas det ur lådan bakom. När alla lådor är tömda på linser fylls det på med fem nya lådor fyllda med linser av personal. En SCARA-Robot från ABB hanterar plockningen av linsen. Sedan släpper roboten linsen i en fixtur på ett ställdon från SKF. När roboten är i en säker position och linsen är i fixturen ska ställdonet åka upp

40 cm för att hamna i en position där linsen kan plockas av Etonbanans produktbärare. Etonbanan har 20 produktbärare som ska plocka linser, transportera dessa på en bana och till sist släppa linsen till beställaren när dennes händer är i position i utkastet.



Figur 9, Modul 4/Anläggning 2 i den virtuella fabriken

För att starta processen för anläggning 2 krävs att det finns beställningar gjorda i det överordnade systemet, eLIPS. PLC:n tar emot signalerna och vid körning så startas Flexlinkmotorn genom en frekvensomvandlare, ASC380. Flexlink har tre lägesgivare och två stopp, se markerat på figur 10.

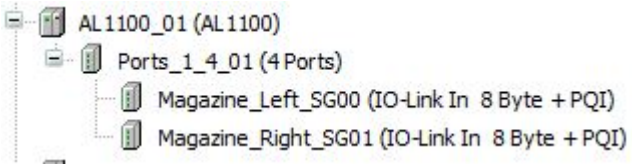


Figur 10, Ritning för Flexlink med markeringar för givare och stopp.

Givare 1 går aktiv när en palett passerar, dvs. när plattan som lådan är placerad på passerar. Då ska stopp 2 aktiveras så att roboten kan plocka ur den lådan. Samtidigt aktiveras stopp 1 kopplat till givare 2 så att lådan bakom stannas så de inte kolliderar. Roboten får signal från PLC:n vilken lins den ska plocka. Parallellt räknar PLC:n hur många som plockats för att ge roboten rätt position och för att indikera när linserna är slut och den ska plocka från en ny kartong. PLC:n håller ständig koll på vilken låda som roboten plockar ur, eftersom paletten ska transporteras ett varv efter att en lins plockats. Detta innebär att PLC:n måste räkna förbipasserade paletter, med hjälp av givare 3, för att stanna samma palett när den kommer tillbaka. När roboten plockats genomförs en handskakning mellan PLC och robot. Denna handskakning avslutas när roboten släppt linsen i fixturen på samma bord som roboten är placerad. PLC:n håller koll att det inte är någonting i vägen för roboten när den ska röra sig. Roboten rör sig till en säker position och PLC:n meddelar Eton Systems genom I/O:s kopplade från deras system till PLC:n att en produktbärare kan åka ner i position för att plocka en lins. Ställdonet som fixturen är placerad på höjs då en givare indikerar att linsen sitter i fixturen, produktbäraren är på plats och roboten är i säker position. Produktbäraren nyper tag i linsen och ställdonet åker ner, sedan åker den iväg för att senare hamna i position för att släppa. Väl i position skickar Eton Systems signal till PLC:n att den är redo att släppa. I utkastet ska beställaren placera sina händer. Där sitter en ultraljudsgivare som känner av händerna. PLC:n hanterar informationen från denna och signalerar att det är okej att släppa linsen. Ett ställdon finns placerat där för att trycka upp mot produktbäraren så att linsen släpper. Linsen är levererad och överordnade systemet eLIPS meddelas.

4.5 Hantering av IO-Link

IO-Link mjukvaran var inte direkt kompatibel med Automation Builder, därför fick lösningarna för hantering av all IO som skulle gå genom IO-Link se lite annorlunda ut än vad den vanligtvis hade gjort. IO-link modulen som används är IFM:s AL1100 modell. En beskrivningsfil för PLC från IFM:s hemsida installerades och implementerades som en hårdvara i Automation Builder. Därifrån lades portarna för AL1100 till och storlek på vardera port valdes, se figur 11. Informationen som skulle hanteras på varje port behövde egna funktionsblock för att skicka ut rätt information. Allt som skulle sitta på portarna listades därför upp och skickades till ABB Support i Danmark där funktionsblocken för varje port skapades. Efter implementering i programmet så kunde rätt utgång från funktionsblocket väljas beroende på vilken information signalen skulle innehålla.



Figur 11, AL1100 med valda portar

4.6 Handskakning mellan PLC och Robot

För att kommunikationen mellan robot och PLC ska fungera korrekt skapades en handskakning för att hela tiden säkerhetsställa att alla signaler har uppfattats av båda parter. Handskakningen åstadkoms genom att använda en strobe, en variabel som validerar signalerna mellan PLC och robot, som sätts hög av PLC:n när roboten ska läsa av com-bitarna (som står för vad roboten skall utföra). När roboten har läst com-bitarna och satt sin egen strobe hög och sina com-bitar på samma sätt som PLC:n så nollställer PLC:n sin strobe bit och väntar på att roboten ska svara med att nollställa sin strobe. När denna sekvens är över vet PLC:n att roboten har förstått och utfört uppgiften. Denna handskakning används alltid när PLC:n ber roboten att utföra en uppgift, se exempelkod nedan.

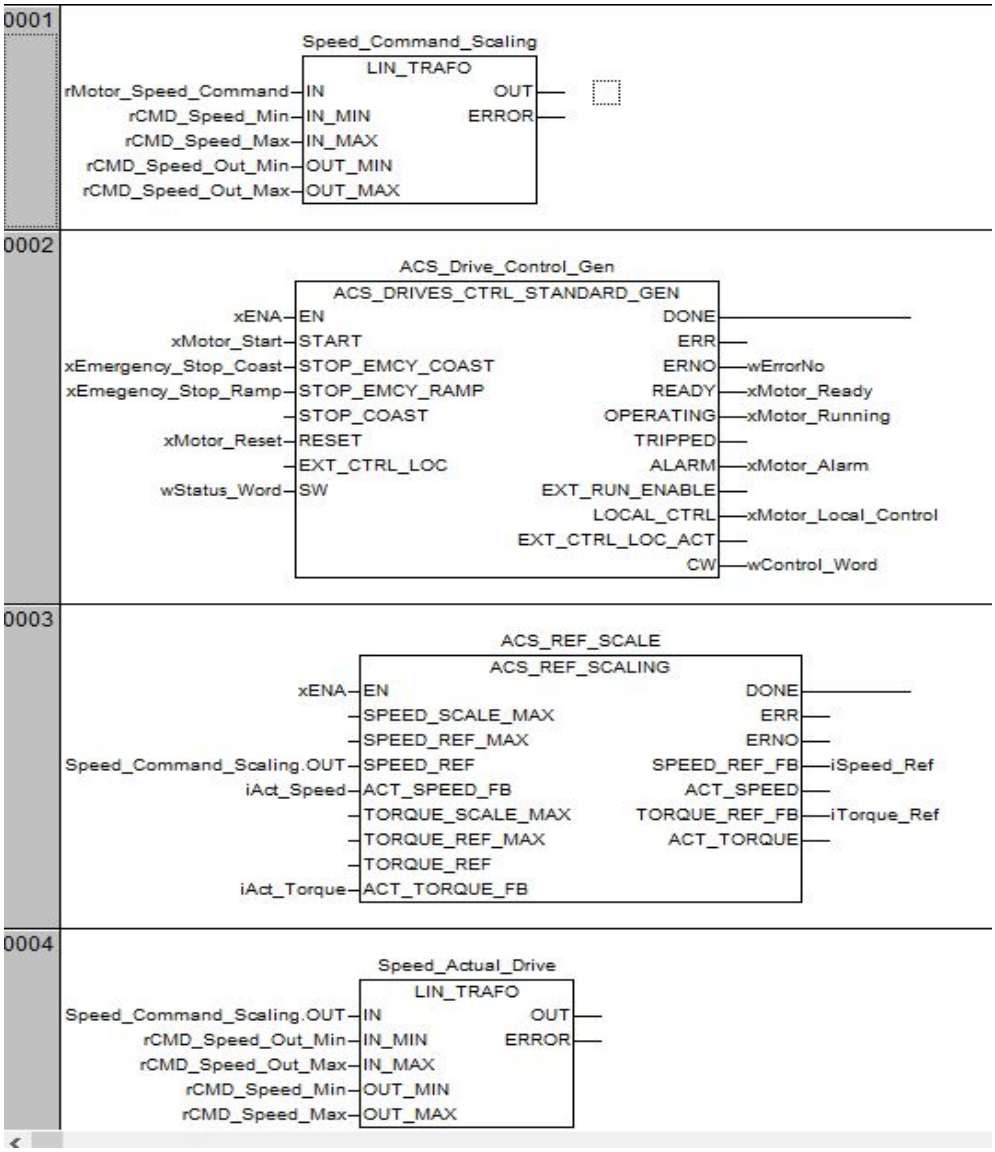
```
(*_________________Robot Handshake_________________*)
CASE Y OF
        15:     doPLCStrobe := 1;               (* Set strobe so that robot starts reading bits *)
                goPLCComBits= 1                 (*Set goBit to value that says what should be done *)
                Y := 16;
        16: IF diIRBStrobe=1 AND giPLCCombits=goPLCComBits THEN
                                (* Check that robot communication bits are correct and that strobe is high*)
                doPLCStrobe:=0;                 (* Set strobe low to end handshake*)
                Y:=17;
        END_IF
        17: IF diIRBStrobe=0 THEN               (*Handshake done when robot strobe is low*)
                (* Next task *)
        END_IF
```

4.7 Programmering av Servo (MicroFlex e150)

Information från motorerna fås av MicroFlex e150. Där hanteras informationen och i PLC-koden så programmeras det som ska genomföras av servomotorerna. Det finns två MicroFlex e150 för två axlar, övre och undre valsen i modul 2. Dessa programmeras att röra sig i positioner och även att kunna joggas om manuellt läge i fabriken är valt. Jog-funktionen krävs då något gått fel, som att ett ark har fastnat i valsen och manuellt måste matas ut. Programmeringen av informationshanteringen för MicroFlex 150 sker separat i en programvara som kallas MINT Workbench, vilket inte hanteras i detta projekt.

4.8 Programmering av frekvensomriktare ASC380

Frekvensomriktarna ASC380 används i modul 3 för transportbandet vid utmatningen och i modul 4 för att driva Flexlink-banan. Ett funktionsblock skapades som används för bägge frekvensomriktarna. Ingångarna som ska variera för de två olika fallen sätts i separata programdelar. I funktionsblocket hanteras flera färdiga funktionsblock för omriktarna. Först sker en linjär transformation i funktionsblocket LIN_TRANFO, detta funktionsblock är implementerat i Automation Builder och är generella för alla typer av drives. Ingångarna sätter verkliga max/min-värde för driven och det max/min-värde som önskar för driven. Utgångarna skickar ut vilket värde som faktiskt skickas ut till driven och meddelar även om det blir något fel under transformationen. Sedan används ett block som heter ACS_DRIVES_CTRL_STANDARD_GEN, hanterar statusord från driven och skickar ut ett kontrollord tillbaka. Efter detta är det tid för blocket ACS_REF_SCALE som får in den verkliga hastigheten från fältbussen och referenshastigheten till fältbussen för driven. Vid stopp så skalas hastigheten ned i funktionsblocket LIN_TRANFO. Se hur funktionsblocket är programmerat i figur 12.



Figur 12, Funktionsblock för frekvensomriktare

4.9 Programmering av ställdon

För att kunna styra de elektriska ställdonen behövs en home-position som används efter strömavbrott eller omprogrammering och två olika positioner där ändläget motsvarar ställdonets slaglängd. Se hur de två olika positionerna sätts i koden nedan.

```
(*Position 1 *)
IN0_in    := 1;
IN1_in    := 0;
IN2_in    := 1;
IN3_in    := 0;
IN4_in    := 1;
```

```
(* Position 2 *)
IN0_in    := 1;
IN1_in    := 0;
IN2_in    := 0;
IN3_in    := 1;
IN4_in    := 1;
```

4.10 Val av programspråk

För att programmeringen skulle vara effektiv användes flera typer av språk.
FBD har valts när koden ska exekveras parallellt och för att till exempel styra frekvensomriktarna för servon till båda valsarna. En av de största fördelarna med FBD är att det är väldigt lätt att återanvända samma funktionsblock och det är lätt att felsöka och följa programmet.

ST har använts mest då det är ett högnivåspråk vilket gör att det är väldigt fritt och lätt att skriva särskilt när man är van vid andra högnivåspråk som till exempel C. I detta projekt har det använts mycket till funktioner som använder räknare av olika slag och där det finns olika fall som kan hända. Som i alla andra högnivåspråk så används funktioner och nyckelord.

SFC har valts då programmet måste agera i sekvenser. SFC är uppbyggt av step, transition och action. Steps associeras med en action, en action är ett program som kan skrivas i vilket språk som helst (ST, FBD m.m.), programmet i exekveras när programmet är i den sekvensen. När ett action har exekverats så finns ett övergångsvillkor som måste uppfyllas för att programmet ska kunna gå vidare. Stegen och övergångsvillkoren gör så att programmet körs sekvensvis och programmeringen inuti action är det som ger programmet information om vad som ska göras.

4.11 Felsökning

Under de sista veckorna av projektet implementerades PLC-programmet modulvis i Robot Studio, i den virtuella modellen genom en virtuell PLC. På så sätt kunde fel upptäckas och korrigeras i programmet/modellen. Flera fall som inte ännu varit implementerade i programmet kunde upptäckas och hanteras. Fall som exempelvis vad som kommer hända när magasinen är tomma på ark, när alla linser är plockade eller när kartongarket tappas bort på vägen. Även saker som programmet hanterade men som inte modellen kunde utföra upptäcktes. Detta var en mycket viktig process då fel är svåra att upptäcka genom att bara läsa i koden.

4.12 Virtuell driftsättning (VFAT)

Innan VFAT (Virtuell driftsättning) skapades en lista där funktionerna listades upp utefter fabrikens flöde, ett VFAT-protokoll, se bilaga 3. Protokollet innefattar funktioner som ska testas. Under VFAT bockas sedan i om det är OK/NOT OK, d.v.s. om funktionen fungerar eller inte i den virtuella miljön. Utifrån detta hanteras funktionerna som inte är godkända för att fabriken ska fungera för olika fall.

5. Resultat

Målet med projektet var att genomföra programmeringen av fabriken och med hjälp av Virtual Commissioning visa fabrikens arbetsgång i en dynamisk modell genom en virtuell PLC. Då programmet ansågs vara färdigt genomfördes en VFAT. Programmet kopplades upp mot en virtuell PLC och sedan följdes VFAT-protokollet, se bilaga 3 parallellt med flödet i den virtuella fabriken och flera fall testades. Resultatet blev att flödet fungerar, fabriken svarar mot programmet. Flera fall testades, de flesta blev godkända och några blev inte godkända. Det innebär att om dessa händelser inträffar så vet inte fabriken vad den ska göra just då. Några exempel på fall som hanteras i programmet är att kommunikationen mellan eLIPS och PLC fungerar, att programmet hanterar tomt/inte tomt magasin, att handskakningen mellan PLC och robot fungerar, att felaktig QR-kod hanteras och att flexlink givarna är i korrekt position. Några fall som inte hanteras i programmet är om skannern slutar fungera (hanteras av eLIPS), om linsen är i produktbäraren för Eton Systems och även statuskommunikation mellan robot och PLC.

6. Diskussion

Flertal saker i detta projekt hade kunnat hanteras annorlunda än de gjorde. Mycket som borde tagits upp under systemeringen i tidigt skede kom fram senare under arbetets gång vilket ledde till att processen haltade under perioder. Vi upptäckte efter några veckor att IO-Link inte var direkt kompatibelt med Automation Builder, varför det krävdes extra arbete för att kunna implementera detta i programmet. Då tiden inte fanns att lägga på detta fick det hanteras vid sidan av under arbetets gång av supporten på ABB. När väl den delen av programmet var färdigt var det bara att lägga in den i det färdiga programmet och hantera alla IO till rätt port på IO-Link.

Att arbeta med digital planering är något som vi anser underlättat arbetsgången. Verktyget Yolean har använts för att planera veckorna i detalj, hantera större mål och milstolpar och koppla dessa till aktiviteterna varje vecka. Tiden på varje aktivitet har lagts in, detta gjorde att det enkelt kunde ses hur lång tid som lagts under veckorna. Förseningar upptäcktes i tidigt skede och det var tydligt hur de påverkade andra delar i projektet. Varje vecka har alla delaktiga i projektet träffats och gått igenom veckan som gått och veckan som kommer. Då har frågor och problem lyfts. Dessa möten har gjort att frågor fått snabba svar och problem hanterats och lösningar för dessa har lyfts.

Val av programspråket är något som ständigt lyfts fram då de flesta har olika åsikter om hur det borde se ut eller så som de själva anser vara det enklaste sättet att arbeta. Vi bestämde oss därför i tidigt skede att flera programspråk skulle testas. I slutändan blev det mest strukturerad text då det passade oss bäst och vi ansåg att det var enklast att programmera i. Programmet innehåller även SFC och FBD.

Vi har upptäckt under arbetets gång är att det inte är enkelt att arbeta i stora projekt med flera olika parter inblandade. Resultatet blir oftast till det bättre då många fler bra idéer och åsikter lyfts fram. Det svåra är att alla ska synka med varandra och i slutändan komma överens om slutprodukten. Personerna som har varit relaterade till projektet är flera vilket innebär att det alltid finns någon att fråga och bolla idéer med.

Nu med en färdig virtuell fabrik, och snart även den verkliga fabriken redo för att driftsättas, har det skapats en plattform där kunskap om industriell digitalisering kan öka intresset för tekniska studier och dra åt sig ungas intresse till att välja arbete inom industrin.

Referenser

[1] Regeringens nyindustraliseringsstrategi för Sverige
http://www.regeringen.se/48f359/contentassets/869c75f458fc4585ab4ec8c13b250a07/informationsmaterial-smart-industri---en-nyindustrialiseringsstrategi-for-sverige
[Använd 06 21 2017]

[2] Frekvensomriktare (2017)
http://www.vfds.com/blog/what-is-a-vfd
[Använd 05 22 2017]

[3] ABB Drivsystem - MicroFlex e150 (2017)
http://www.abb.com/product/seitp322/0bc104744abed5d9c1257a6300478b87.aspx
[Använd 05 04 2017]

[4] ABB Drives, ACS380 - Frekvensomriktare för maskinbyggare (2017)
http://new.abb.com/drives/sv/frekvensomriktare-for-lagspanning/frekvensomriktare-maskinapplikationer/acs380
[Använd 12 04 2017]

[5] IO-Link Overview (2017)
http://www.io-link.com/en/Technology/what_is_IO-Link.php?thisID=76
[Använd 19 04 2017]

[6] ABB Automation Builder (2017)
http://new.abb.com/plc/sv/automation-builder
[Använd 27 04 2017]

[7] ABB Robot Studio (2017)
http://new.abb.com/products/robotics/sv/robotstudio
[Använd 27 04 2017]

[8] EtherCAT (2017-05-06)
https://en.wikipedia.org/wiki/EtherCAT#Functional_Principle
[Använd 05 22 2017]

[9] OPC (2016-03-30)
https://sv.wikipedia.org/wiki/OPC
[Använd 05 22 2017]

[10] Gränssnittsbeskrivning PLC, Diab Produktionssystem, DPS. DIAB AB. 2007-05-10

[11] ABB IRB1200 - List Of Industrial Robots (2017)
http://new.abb.com/products/robotics/industrial-robots/irb-1200

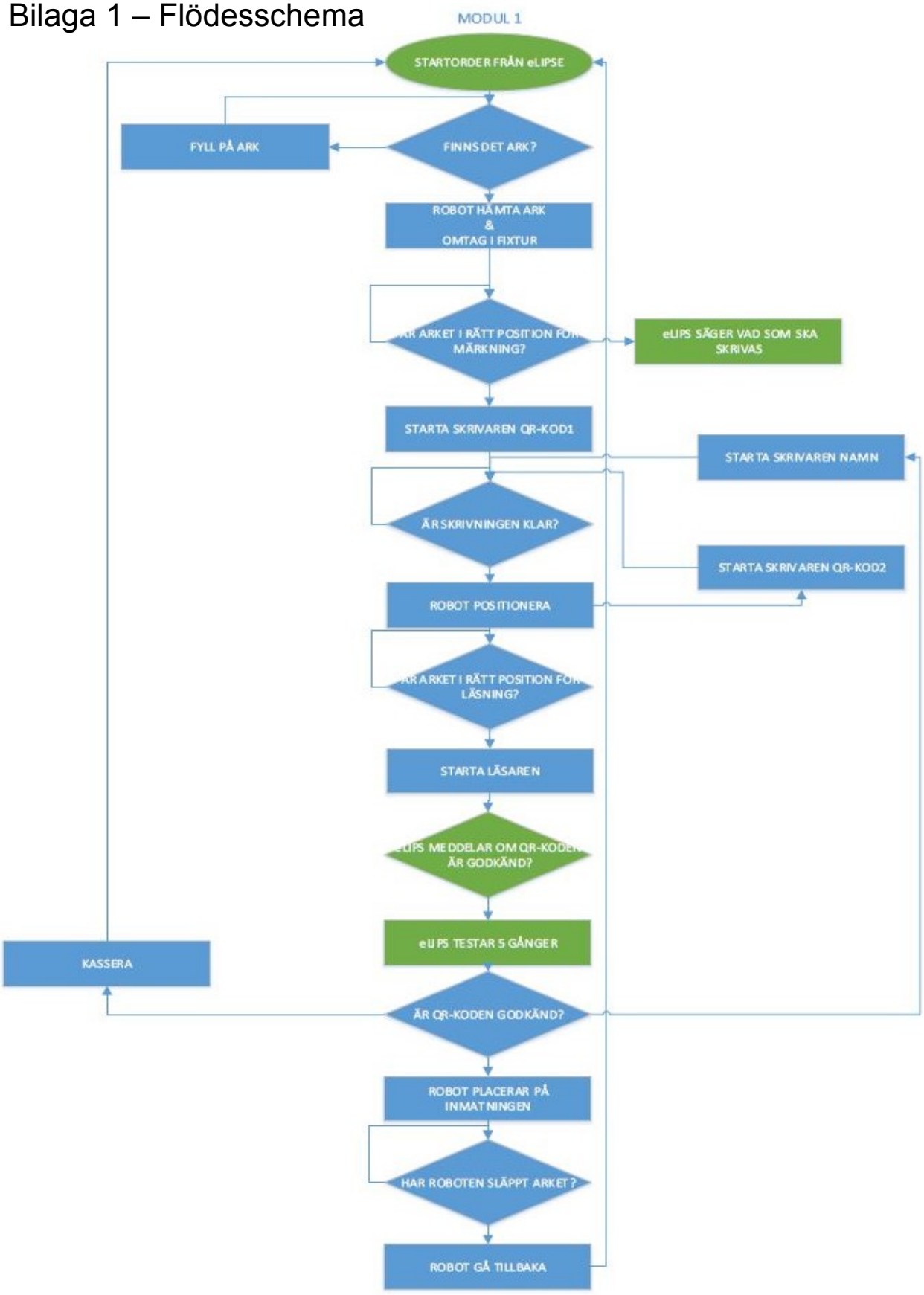
[Använd 10 04 2017]

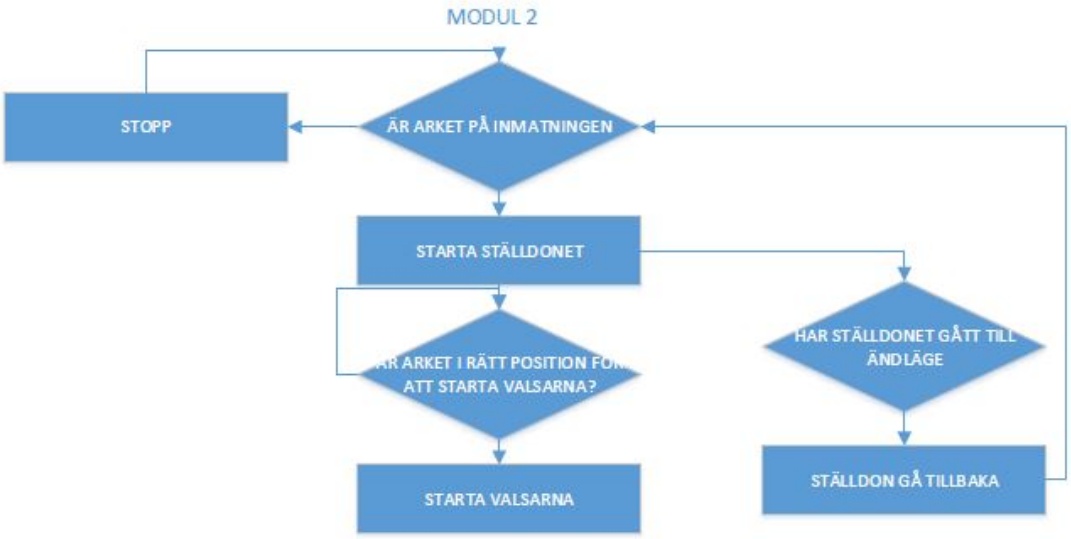
[12] ABB IRB 910DC/ABB SCARA-Robot - List Of Industrial Robots (2017)
http://new.abb.com/products/robotics/industrial-robots/irb-910sc
[Använd 10 04 2017]

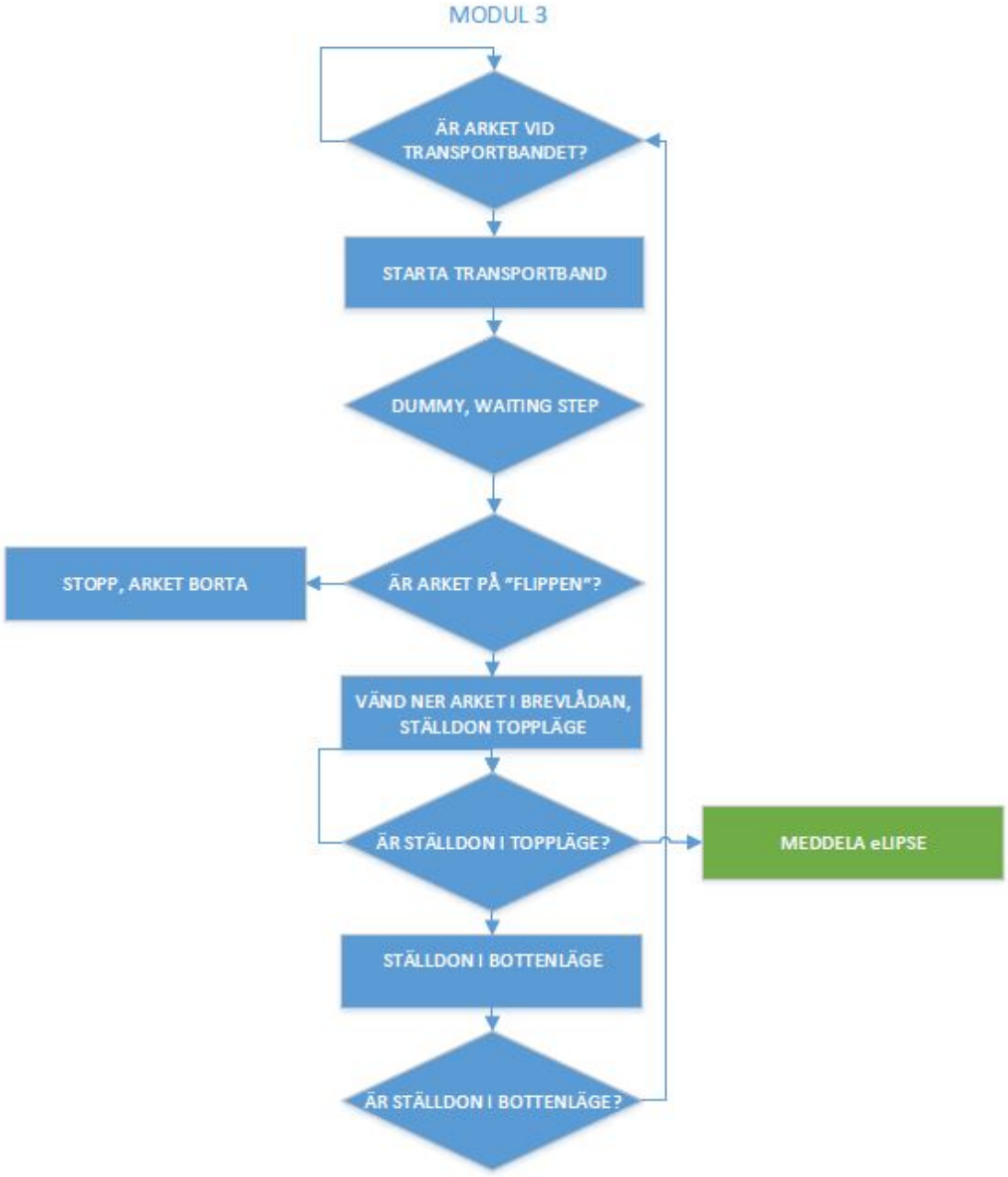
Bilagor

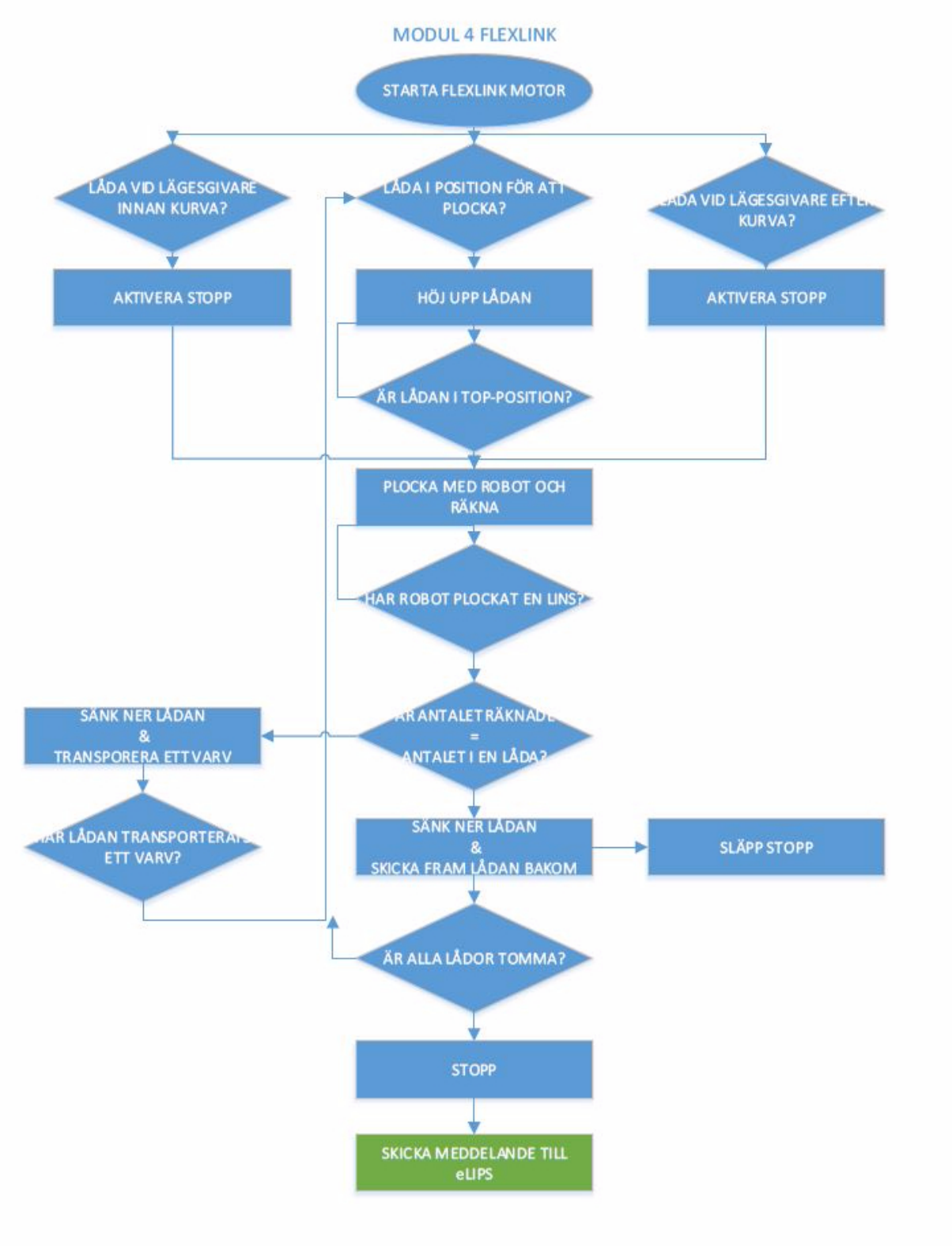
Innefattar flödesschema, programkod och VFAT-protokoll.

Bilaga 1 – Flödesschema

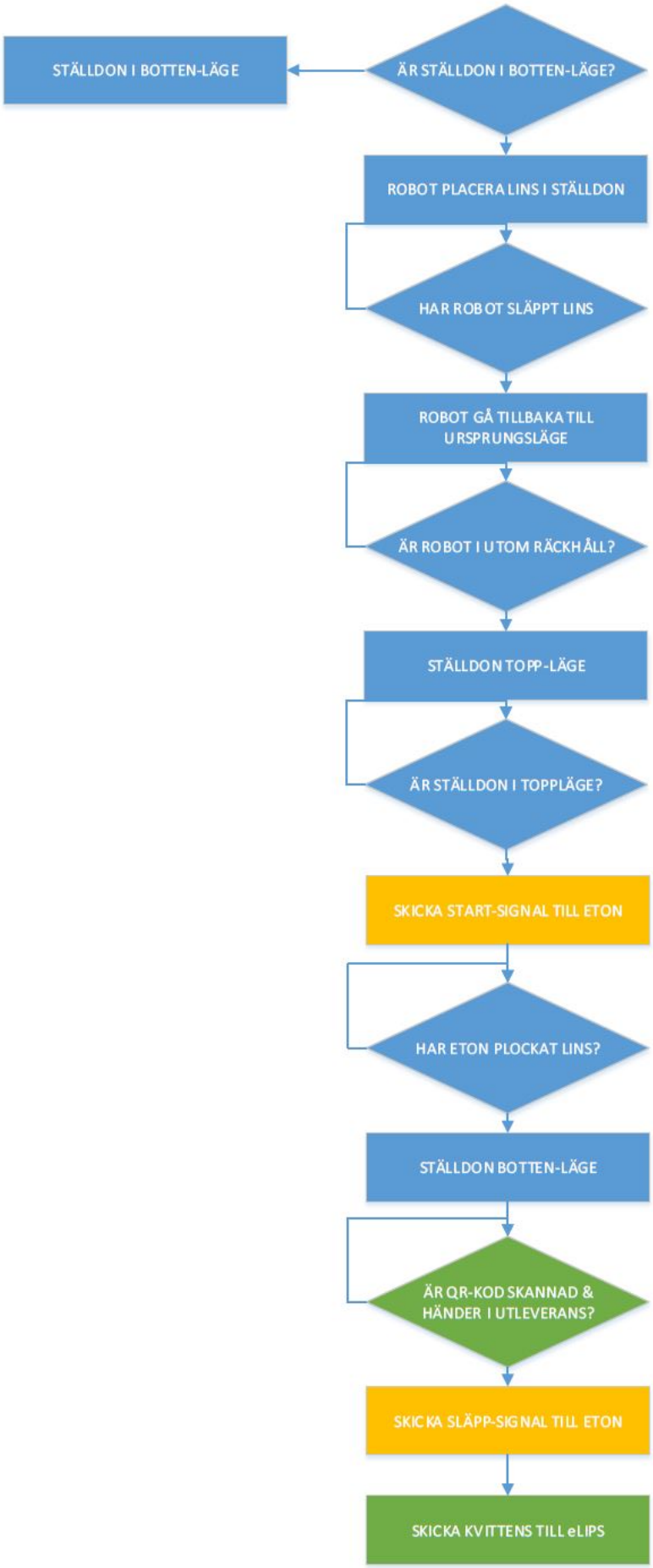



MODUL 2
STOPP
ÄR ARKET PÅ INMATNINGEN
STARTA STÄLLDONET
ÄR ARKET I RÄTT POSITION FÖR
ATT STARTA VALSARNA?
STARTA VALSARNA
HAR STÄLLDONET GÅTT TILL
ÄNDLÄGE
STÄLLDON GÅ TILLBAKA


MODUL 3
ÄR ARKET VID
TRANSPORTBANDET?
STARTA TRANSPORTBAND
DUMMY, WAITING STEP
STOPP, ARKET BORTA
ÄR ARKET PÅ "FLIPPEN"?
VÄND NER ARKET I BREVLÅDAN,
STÄLLDON TOPPLÄGE
ÄR STÄLLDON I TOPPLÄGE?
MEDDELA eLIPSE
STÄLLDON I BOTTENLÄGE
ÄR STÄLLDON I BOTTENLÄGE?

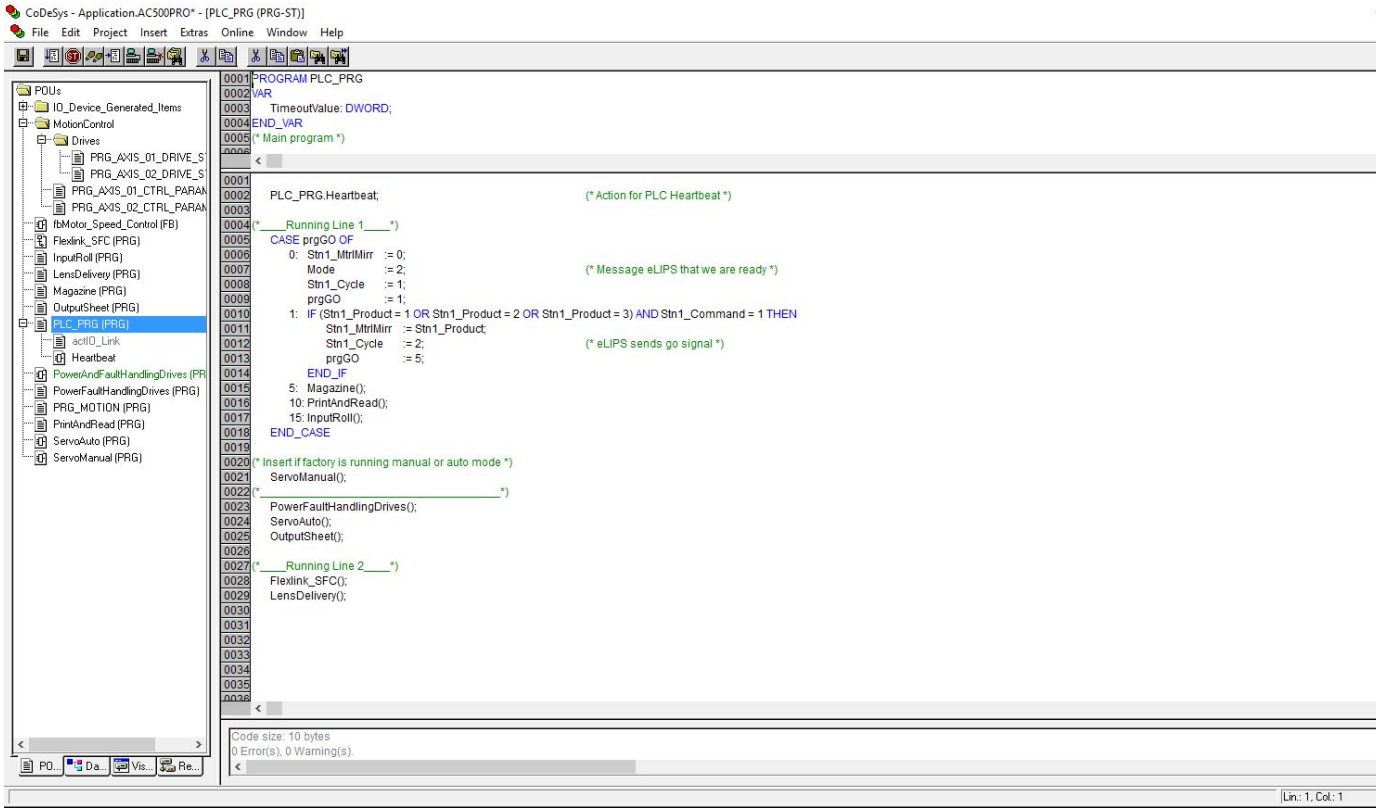

MODUL 4 FLEXLINK
STARTA FLEXLINK MOTOR
LÅDA VID LÄGESGIVARE INNAN KURVA?
LÅDA I POSITION FÖR ATT PLOCKA?
KURVA?
AKTIVERA STOPP
HÖJ UPP LÅDAN
AKTIVERA STOPP
ÄR LÅDAN I TOP-POSITION?
PLOCKA MED ROBOT OCH RÄKNA
HAR ROBOT PLOCKAT EN LINS?
SÄNK NER LÅDAN & TRANSPORERA ETT VARV
ANTALET I EN LÅDA?
SÄNK NER LÅDAN & SKICKA FRAM LÅDAN BAKOM
SLÄPP STOPP
ETT VARV?
ÄR ALLA LÅDOR TOMMA?
STOPP
SKICKA MEDDELANDE TILL eLIPS

MODUL 4 ROBOT+STÄLLDON+ETON



Bilaga 2 - Programkod

Main Program



Modul 1

PROGRAM Magazine

```
VAR
        Counter_Left:           INT;
        Counter_Right:   INT;
        Y:          INT     :=5;
END_VAR
(* Check magazine status and if status is OK pick carton from every other compartment,
includes three handshakes with robot, two for each compartment and one for when robot is back in pick pos *)

        CASE  Y OF      (* Y = Which container the robot should pick from the magazine*)
                5:      IF diMagazineLeftEmpty = 1 AND diMagazineRightEmpty= 1 THEN
(*Both magazines have sheets*)
                                Y := 10;
                                RefillMagazine := 0;
                        END_IF;
                        IF diMagazineLeftEmpty = 1  AND diMagazineRightEmpty = 0 THEN(* Left
magazines have sheets*)
                                Y := 6;
                                RefillMagazine := 0;
                        ELSIF diMagazineLeftEmpty = 0 AND diMagazineRightEmpty = 1 THEN      (*
Right magazines have sheets*)
                                Y := 7;
                                RefillMagazine := 0;
                        END_IF;
```

```
                IF diMagazineLeftEmpty = 0 AND diMagazineRightEmpty = 0 THEN  (* No
magazines have sheets*)
                        Y := 8;
                END_IF;
        6:      goPLCComBits := 0;
        (* Robot pick left *)
                Y := 15;
        7:      goPLCComBits := 1;
        (* Robot pick right*)
                Y := 15;
        8:      goPLCComBits := 2;
        (* Robot reset pick pos.*)
                Y := 15;
        10:     IF Counter_Left >= Counter_Right THEN
                        goPLCComBits := 1;
        (* Robot pick right*)
                        Y := 15;
                ELSE
                        goPLCComBits := 0;
        (* Robot pick left *)
                        Y := 15;
                END_IF;
(*_____________Robot Handshake_____________*)
        15:     doPLCStrobe := 1;
        (* Set strobe so that robot starts reading bits *)
                Y := 16;
        16: IF diIRBStrobe=1 AND giPLCCombits=goPLCComBits THEN (* Check that robot
communication bits are correct and that strobe is high*)
                        doPLCStrobe:=0;
                (* Set strobe low to end handshake*)
                        Y:=17;
                END_IF
        17: IF diIRBStrobe=0 THEN
    (* Handshake done when robot strobe is low*)
                        IF goPLCComBits= 1 THEN
    (* Increase counter depending on where robot picked*)
                                goCheckPLCComBits := 1;
    (*Signal for checking what robot has done, up on panel *)
                                Counter_Right := Counter_Right +1;
                        ELSIF goPLCComBits = 0 THEN
                                goCheckPLCComBits := 0;
    (*Signal for checking what robot has done, up on panel *)
                                Counter_Left := Counter_Left +1;
                        ELSIF goPLCComBits = 2 THEN                                    (*
Robot in pick pos *)
                                goCheckPLCComBits := 2;
    (*Signal for checking what robot has done, up on panel *)
                                RefillMagazine := 1;
            (* Message staff that there are no cartons left*)
                                Counter_Left := 0;
                                Counter_Right := 0;
                        END_IF
```

```
                    Y:=18;
                END_IF
        18:     IF goPLCComBits = 0 OR goPLCComBits = 1 THEN
                    prgGO  := 10;
                (* Move on to next procedure*)
                END_IF
                Y := 5;
                    (* init. value 5*)
    END_CASE
```

PROGRAM PrintAndRead

```
VAR
    A:      INT             :=0;            (* Variable that count what's printing, name or QR-code*)
    B:      INT     :=0;            (* Variable that controlls if qr-code is ok/not ok*)
    Z:      INT     := 4;
    Printer_temp: BOOL;
END_VAR
(*Print/Read Case including eLIPS communication*)
(*_____*)
(* Robot passes printer then stops in front of scanner,
eLIPS decides if QR-code is OK/NOT OK,
if OK, then continue process, if NOT OK scrap sheet and start over *)

    CASE Z OF
        4:      A       := 0;
                B       := 0;
                Printer_temp    := 0;
                Z       := 5;
        5:      goPLCComBits    := 3;
                doPLCStrobe             := 1;
                A       := A+1;
                Z       := 10;
        10:             (* If robot is in position then move forward to print*)
                IF diIRBStrobe=1 AND giPLCCombits=goPLCComBits THEN
     (* Check that robot communication bits are correct and that strobe is high*)
                    doPLCStrobe := 0;
                                            (* Set strobe low to end handshake*)
                    Z       := 15;
                END_IF
        15:     IF diIRBStrobe=0 THEN
                                    (* Handshake done when robot strobe is low*)
                    Printer_temp                    := 1;
                                            (* Temporary variable, print *)
                    goCheckPLCComBits       := 3;
                            (*Signal for checking what robot has done, up on panel *)
                    IF A  = 2 THEN
                        Z       := 5;
                    ELSE
                        Z       := 20;
                    END_IF
                END_IF
        20:     goPLCComBits            := 4;
```

```
                doPLCStrobe                     := 1;
                Z       := 25;
        25:     IF diIRBStrobe=1 AND giPLCCombits=goPLCComBits THEN
(* Check that robot communication bits are correct and that strobe is high*)
                        doPLCStrobe := 0;
                                                (* Set strobe low to end handshake*)
                        Z       := 30;
                END_IF
        30:     IF diIRBStrobe=0 THEN
                                        (* Handshake done when robot strobe is low*)
                        Stn1_ResultReq          := 1;
                        Stn1_Cycle                      := 3;
                        goCheckPLCComBits := 4;
                                        (*Signal for checking what robot has done, up on panel *)
                        Z       := 35;
                END_IF
        35:     IF Stn1_ResultCMD = 2 THEN
                                (*QR-code OK*)
                        Stn1_Cycle              := 2;
                        goPLCComBits    := 5;
                        Z       := 99;
                ELSIF Stn1_ResultCMD = 3 THEN
                                (* QR-code not ok *)
                        goPLCComBits := 6;
                        Z       := 98;
                END_IF
        99:             doPLCStrobe := 1;
                                                (* Activate handshake *)
                        Z       := 36;
        98:             doPLCStrobe := 1;
                        Z       := 37;
        36:     IF diIRBStrobe=1 AND giPLCCombits=goPLCComBits THEN
(* Check that robot communication bits are correct and that strobe is high*)
                        doPLCStrobe:=0;
                                                        (* Set strobe low to end handshake*)
                        B       := 1;
                        Z       := 40;
                END_IF
        37:     IF diIRBStrobe=1 AND giPLCCombits=goPLCComBits THEN
(* Check that robot communication bits are correct and that strobe is high*)
                        doPLCStrobe:=0;
                                                        (* Set strobe low to end handshake*)
                        B       := 2;
                        Z       := 40;
                END_IF
        40:     IF diIRBStrobe = 0 THEN
                                        (* Print again or if done, end or if scraped scheet goto
magazine*)
                        IF A = 3 AND B = 1 THEN
                                                (*Done whith printing (QR-code, name, QR-code
*)
```

```
                                    goCheckPLCComBits := 5;
                                        (*Signal for checking what robot has done, up on panel *)
                                    prgGO  := 15;
                                    Z          := 4;
                                                                    (* Move on to next
procedure*)
                            ELSIF( A = 1 OR A  = 3) AND B = 2 THEN
                                    goCheckPLCComBits := 6;
                                        (*Signal for checking what robot has done, up on panel *)
                                    Stn1_Cycle := 4;
                                            (* Message eLIPS that we are DONE *)
                                    prgGO := 5;
                                    Z          := 4;
                            ELSE
                                    Z          := 5;
                            END_IF
                    END_IF
        END_CASE
```

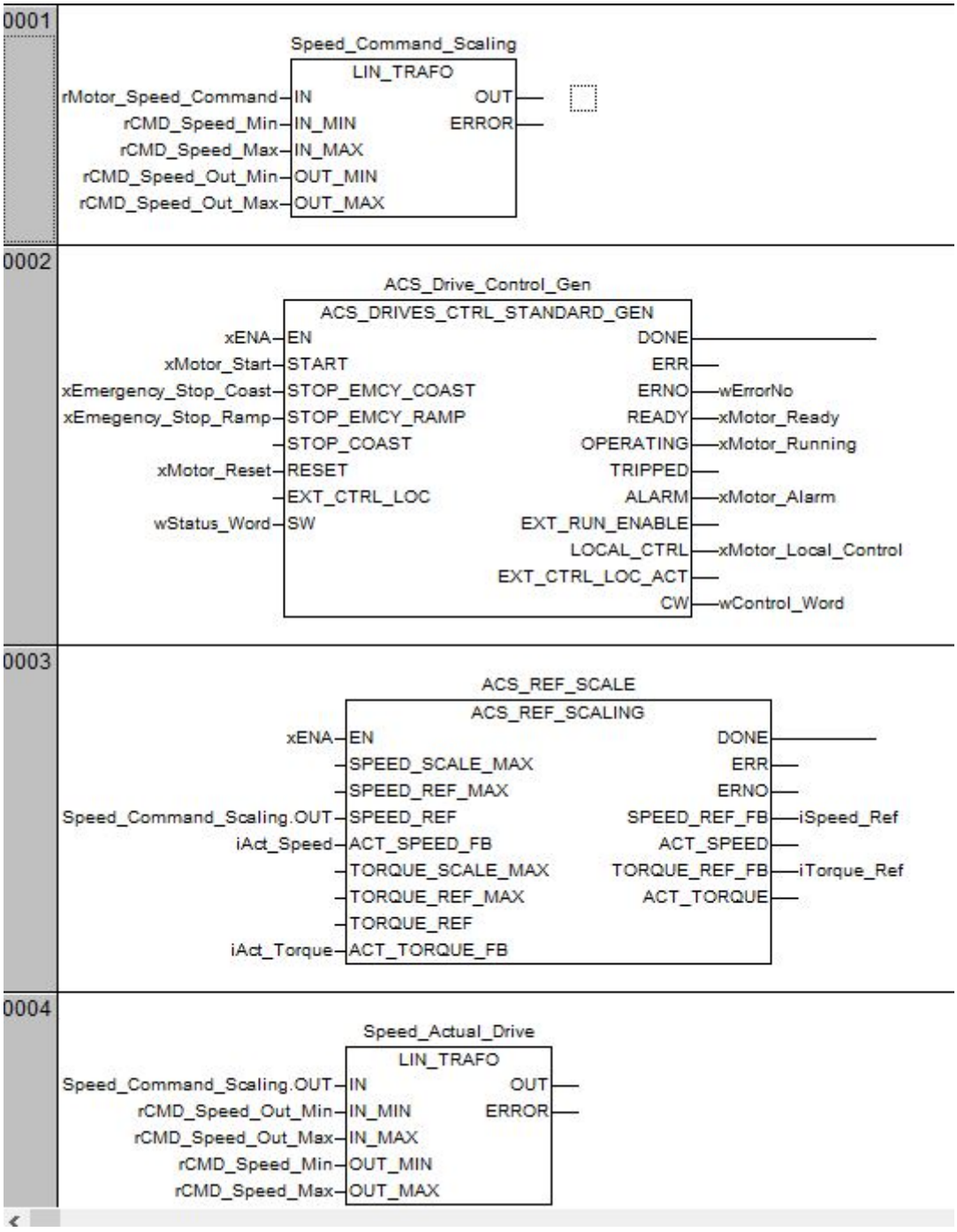
Modul 2

PROGRAM InputRoll

```
VAR
        Q: INT  :=      5;
END_VAR
(* Input to rolls with acuators from SKF *)
(*_____________________________________________________*)
(* When sheet is on input, actuators will start, when sheet is in right position, rolls starts *)

CASE Q OF
        5:      goPLCComBits   := 7;
                Q                                       := 6;
        6:      doPLCStrobe             := 1;
                Q := 7;
        7:       IF diIRBStrobe=1 AND giPLCCombits=goPLCComBits THEN (* Check that robot
communication bits are correct and that strobe is high*)
                        doPLCStrobe     := 0;
                        Q                               := 8;
                END_IF
        8:      IF diIRBStrobe=0 AND diSheetAtInput = 1 THEN                    (* go to pos 2 on
acurator, out*)
                        IN0_in  := 1;
                        IN1_in  := 0;
                        IN2_in  := 0;
                        IN3_in  := 1;
                        IN4_in  := 1;
                        doInputMoveOut          := 1;
        (* Signal to robot studio*)
                        doInputMoveIn                   := 0;
                        goCheckPLCComBits := 7;
        (*Signal for checking what robot has done, up on panel *)
                        Q       := 9;
                END_IF
```

```
        9:  IF OUT1_in = 0 AND OUT2_in =1 THEN                                  (* Controls
that acurator is in motion *)
                Q       := 10;
                END_IF
        10: IF OUT1_in = 1 AND OUT2_in = 0 THEN                                 (* When
actuators is in end positon, goto start pos (pos 1) *)
                        IN0_in    := 1;
                        IN1_in    := 0;
                        IN2_in    := 1;
                        IN3_in    := 0;
                        IN4_in    := 1;
                        doInputMoveOut  := 0;
                        doInputMoveIn           := 1;
                        (* Signal to robot studio *)
                        Q       := 11;
                END_IF
        11:     IF OUT1_in = 0  AND OUT2_in = 1 THEN                    (* Controls that
acurator is in motion *)
                        Q       :=12;
                END_IF
        12: IF OUT1_in = 1 AND OUT2_in = 0 THEN                                 (* When
actuators is in end positon, continue prg *)
                        prgGO                   := 0;
                        doInputMoveIn :=0;
                        Q       := 5;
                END_IF
END_CASE


FUNCTION_BLOCK fbMotor_Speed_Control
VAR_INPUT
        xENA: BOOL;
        xMotor_Start: BOOL;
        xEmergency_Stop_Coast: BOOL;
        xEmegency_Stop_Ramp: BOOL;
        xMotor_Reset: BOOL;
        rMotor_Speed_Command: REAL;
        rCMD_Speed_Min: REAL;
        rCMD_Speed_Max: REAL;
        iAct_Speed: INT;
        iAct_Torque: INT;
        wStatus_Word: WORD;

END_VAR
VAR_OUTPUT
        iSpeed_Ref: INT;
        iTorque_Ref: INT;
        wControl_Word: WORD;
        xMotor_Ready: BOOL;
        xMotor_Running: BOOL;
        xMotor_Alarm: BOOL;
        xMotor_Local_Control: BOOL;
```

```
        wErrorNo: WORD;
END_VAR
VAR
        Speed_Command_Scaling: LIN_TRAFO;
        Speed_Actual_Drive: LIN_TRAFO;
        ACS_Drive_Control_Gen: ACS_DRIVES_CTRL_STANDARD_GEN;
        ACS_REF_SCALE: ACS_REF_SCALING;
        rCMD_Speed_Out_Min: REAL;
        rCMD_Speed_Out_Max: REAL;
END_VAR
(* Functionblock that controls speed on both drives *)
```

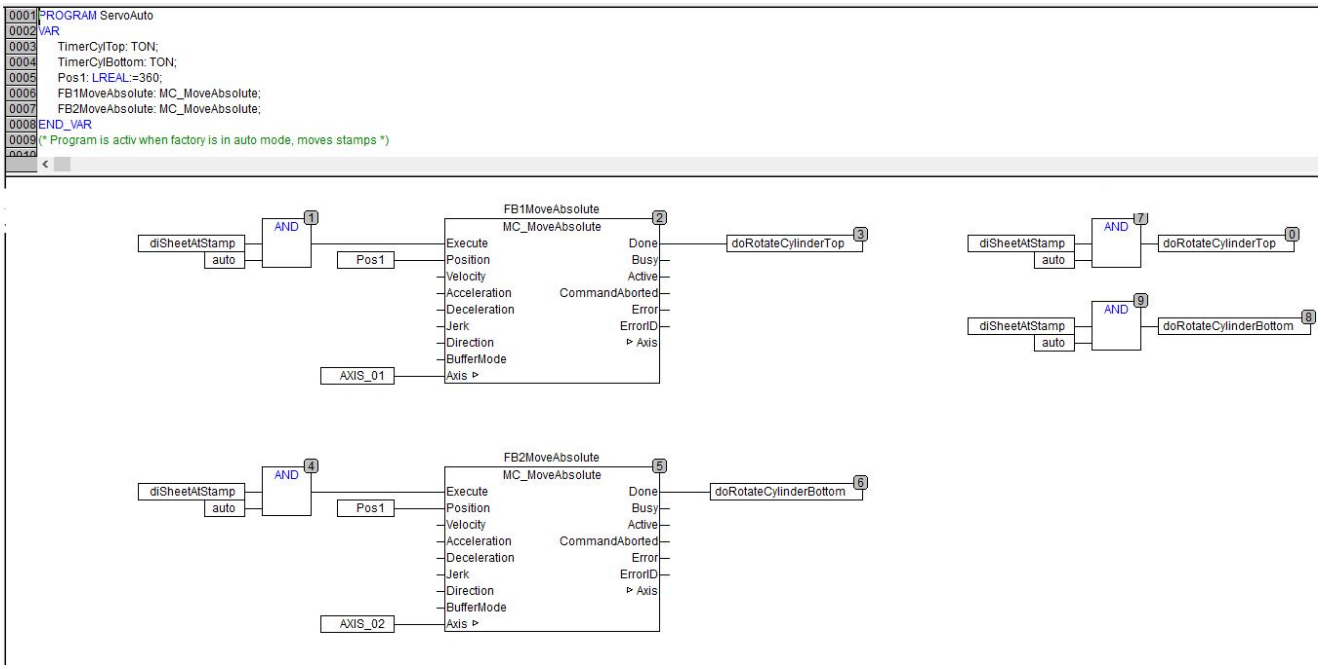


PROGRAM PowerFaultHandlingDrives

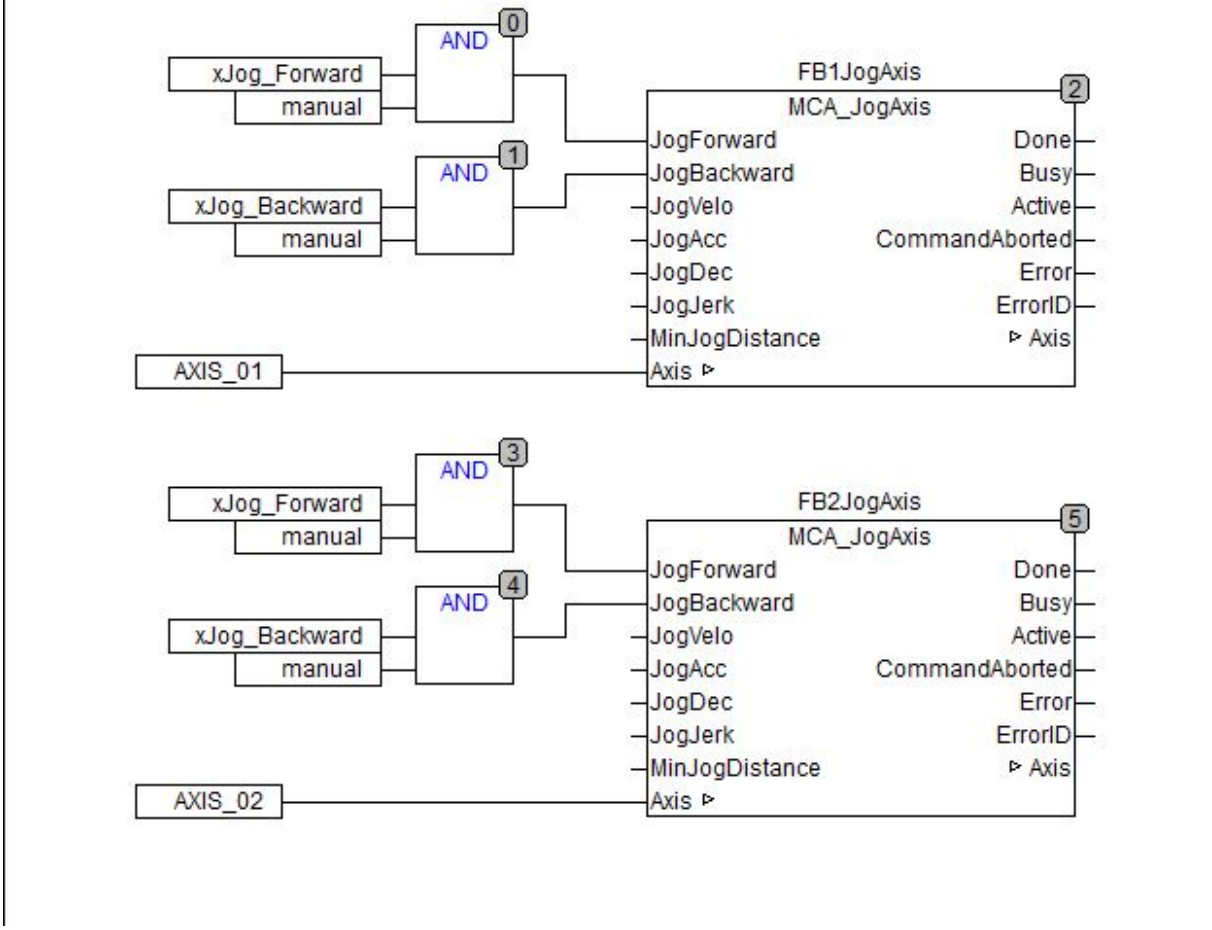
```
VAR
        FBPowerDrive1: MC_Power;
    FBPowerDrive2: MC_Power;
    PowerOn: BOOL;
    FBRstDrive1:MC_Reset;
    FBRstDrive2:MC_Reset;
        DistAxis1: LREAL;
    DistAxis2: LREAL;
        Acc: LREAL;
        Dec: LREAL;
        Vel: LREAL;
        FBMoveRelAxis2:MC_MoveRelative;
        MC_HomeAxis1: MC_Home;
        MC_HomeAxis2: MC_Home;
        HomingDone: BOOL;
        Homing1: BOOL;
        HomingPos: LREAL := 0;
        Estop: BOOL;
        FBReadAxis1Error: MC_ReadAxisError;
        FBReadAxis2Error: MC_ReadAxisError;
        SRHoming1:SR;
        SRHoming2:SR;
END_VAR
(* Program handles drives, power enable, reset, read axis error and homing position *)

(*Enable Drive 1*)

FBPowerDrive1(
        Enable:=PowerOn , (* inga nödstod, ok att köra *)
        Axis:=AXIS_01 ,
        Status=> ,
        Error=> ,
        ErrorID=> );

(*Reset Drive 1*)

FBRstDrive1(
        Execute:=  FBPowerDrive1.Error ,
        Axis:=AXIS_01 ,
        Done=> ,
        Busy=> ,
        Error=> ,
        ErrorID=> );

(*Enable Drive 2*)

FBPowerDrive2(
        Enable:=PowerOn ,
        Axis:=AXIS_02 ,
        Status=> ,
        Error=> ,
        ErrorID=> );
```

```
(*Reset Drive 2*)

FBRstDrive2(
        Execute:= FBPowerDrive2.Error ,
        Axis:=AXIS_02 ,
        Done=> ,
        Busy=> ,
        Error=> ,
        ErrorID=> );

(*Read Axis Error Drive 1 *)

FBReadAxis1Error(
        Enable:= FBPowerDrive1.Status,
        Axis:=AXIS_01 ,
        Valid=> ,
        Error=> ,
        ErrorID=> ,
        AxisErrorID=> );


(*Read Axis Error Drive 2 *)

FBReadAxis2Error(
        Enable:= FBPowerDrive2.Status,
        Axis:= AXIS_02,
        Valid=> ,
        Error=> ,
        ErrorID=> ,
        AxisErrorID=> );

(* Homing pos Drive 1 *)

IF HomingDone = 0 AND FBPowerDrive1.Status = 1 THEN
        Homing1 := 1;
END_IF

IF Estop = 1 OR FBReadAxis1Error.Error = 1 THEN
        HomingDone := 0;                        (* HomingDone reset*)
END_IF

MC_HomeAxis1(
        Execute:= Homing1,
        Position:= HomingPos,
        BufferMode:= ,
        Axis:= AXIS_01,
        Done=> ,
        Busy=> ,
        Active=> ,
        CommandAborted=> ,
        Error=> ,
```

```
        ErrorID=> );

(*SR flipflop for SET HomingDone when axis flags done*)
SRHoming1(SET1:= MC_HomeAxis1.Done , RESET:=0 );
HomingDone := SRHoming1.Q1 ;

(* Homing pos Drive 2 *)

IF HomingDone = 0 AND FBPowerDrive2.Status = 1 THEN
        Homing1 := 1;
END_IF

IF Estop = 1 OR FBReadAxis2Error.Error = 1 THEN
        HomingDone := 0; (* HomingDone reset*)
END_IF

MC_HomeAxis2(
        Execute:= Homing1,
        Position:= HomingPos,
        BufferMode:= ,
        Axis:= AXIS_02,
        Done=>,
        Busy=> ,
        Active=> ,
        CommandAborted=> ,
        Error=> ,
        ErrorID=> );

(*SR flipflop for SET HomingDone when axis flags done*)
SRHoming2(SET1:= MC_HomeAxis2.Done , RESET:=0 );
HomingDone := SRHoming2.Q1 ;
```



```
0001 PROGRAM ServoManual
0002 VAR
0003     FB1JogAxis: MCA_JogAxis;
0004     FB2JogAxis: MCA_JogAxis;
0005     xJog_Forward: BOOL;
0006     xJog_Backward: BOOL;
0007 END_VAR
0008 (* Program is activ when factory is in manual mode, joggs stamps *)
```



Modul 3

PROGRAM OutputSheet

```
VAR
        fbMotor_Speed_Control_1: fbMotor_Speed_Control;
        xENPMotorSpeed:                 BOOL;
        xEmrMotorSpeed_Coast:  BOOL;
        xEmrMotorSpeed_Ramp:  BOOL;
        xENPMotorReset:                 BOOL;
        state:                                  INT:=5;
END_VAR

(* Output to customer with acuators from SKF *)
(*_____*)
(* When sheet is on output, actuators will start, when sheet is in right position, "flip" sheet to "mailbox" *)
```

```
(*ACS380 ENP Motor Drive Setup*)
(*_____________________________________________________________________*)

        fbMotor_Speed_Control_1.xMotor_Start := 1;              (* Motor always enable for ENP *)

        IF diSheetAtOutput = 1 THEN
                doMotorActive := 1;
                rENPMotor_Percent  := 70;                                   (* Signal to
robotstudio when motor is active *)
                fbMotor_Speed_Control_1(
                xENA                                            :=          TRUE,
                xMotor_Start                                    :=          xENPMotorSpeed,
                xEmergency_Stop_Coast           :=              xEmrMotorSpeed_Coast,
                xEmegency_Stop_Ramp             :=              xEmrMotorSpeed_Ramp,
                xMotor_Reset                                    :=          xENPMotorReset,
                rMotor_Speed_Command :=         rENPMotor_Percent,
                rCMD_Speed_Min                  :=              0,
                rCMD_Speed_Max                  :=              100,
                iAct_Speed                                      :=          goMotorSpeedENP,
                iAct_Torque                                     :=          ,
                wStatus_Word                    :=              StatusENP,
                iSpeed_Ref                                      =>          MotorSpeedRefENP,
                iTorque_Ref                                     =>          ,
                wControl_Word                   =>              CmdENP,
                xMotor_Ready                                    =>          ,
                xMotor_Running                  =>              ,
                xMotor_Alarm                                    =>          ,
                xMotor_Local_Control            =>              ,
                wErrorNo                                        =>          );
        ELSIF diSheetAtOutput = 0 AND diSheetAtFlip = 1 THEN
                fbMotor_Speed_Control_1.rMotor_Speed_Command    :=          0;
                rENPMotor_Percent  := 0;
        END_IF

(*SKF_Actuators*)
(*_____________________________________________________________________*)
        CASE state OF
                5:      IF diSheetAtFlip = 1 THEN
                                IN0_out  := 0;
                                IN1_out  := 1;
                                IN2_out  := 0;
                                IN3_out  := 0;
                                IN4_out  := 1;
                                doFlipMoveUp            := 1;
        (* Signal to robot studio *)
                                doFlipMoveDown   := 0;
                                Stn1_Cycle              := 4;
        (* Notify eLIPS that sheet is delivered when flipped *)
                                state      := 6;
                        END_IF
                6:      IF OUT1_out = 0  AND OUT2_out = 1 THEN          (* Flip moving up*)
                                state      := 7;
```

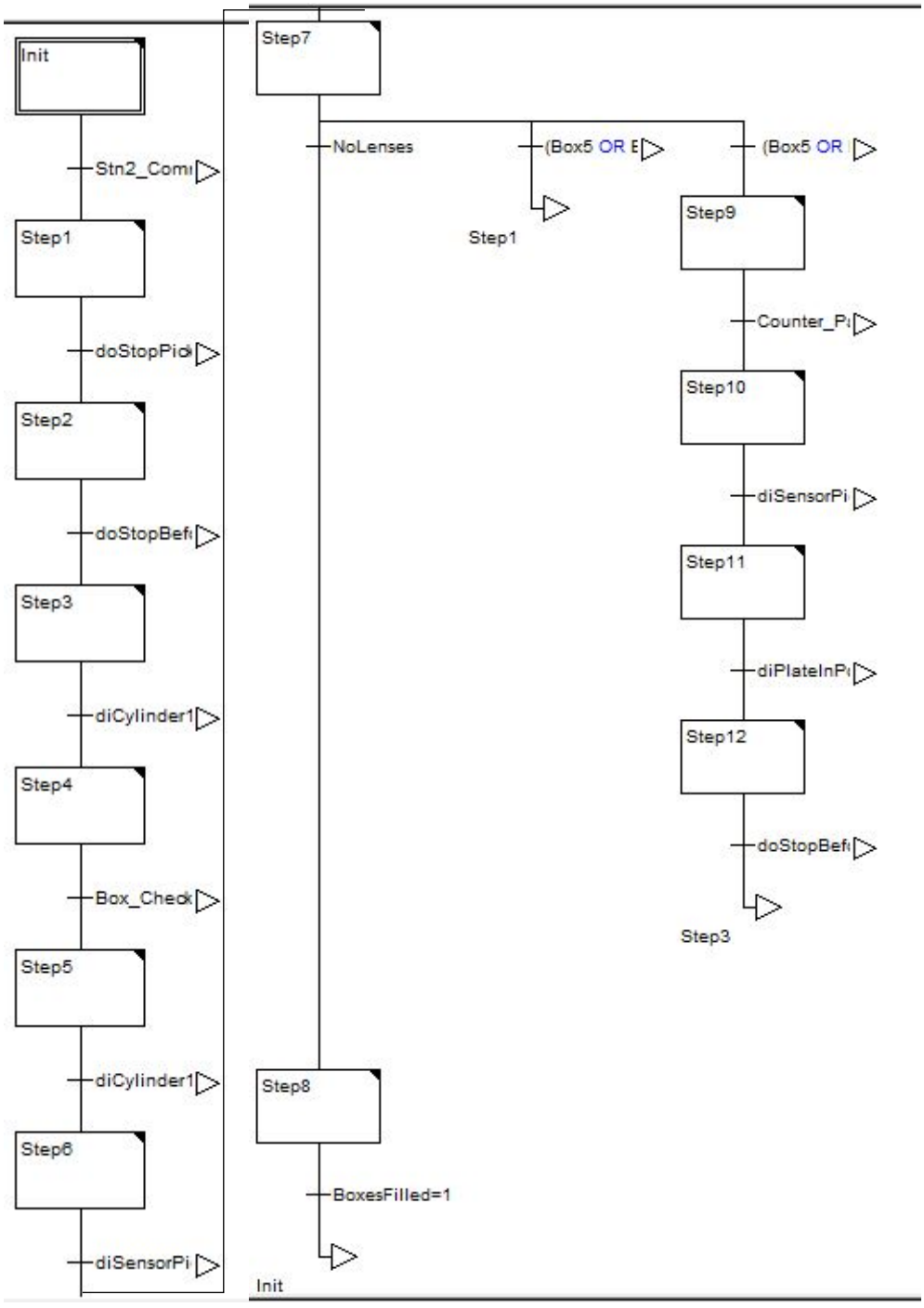
```
                END_IF
        7:      IF OUT1_out = 1  AND OUT2_out = 0 THEN          (*When top position,
begin homing *)
                        IN0_out  := 0;
                        IN1_out  := 1;
                        IN2_out  := 0;
                        IN3_out  := 0;
                        IN4_out  := 1;
                        doFlipMoveDown          := 1;
    (* Signal to robot studio *)
                        doFlipMoveUp            := 0;
                        state    := 8;
                END_IF
        8:      IF OUT1_out = 0  AND OUT2_out = 1 THEN          (* Flip moving
down*)
                        state    := 9;
                END_IF
        9:      IF OUT1_out = 1  AND OUT2_out = 0 THEN          (* Flip is down.
return to start state*)
                        state    := 5;
                END_IF
    END_CASE
```

Modul 4 Flexlink

PROGRAM Flexlink_SFC

```
VAR
    fbMotor_Speed_Control_2: fbMotor_Speed_Control;
    xEmrMotorSpeed_Coast:   BOOL;
    xEmrMotorSpeed_Ramp:   BOOL;
    xFlexlinkMotorReset:        BOOL;
    xFlexlinkMotorSpeed:        BOOL;
    LensNr:                     USINT:=1;
    L:                                  INT:=5;
    BoxNr:                      INT := 4;
    Counter_Passed:     INT:=0;
    Counter_Passed2:            INT:=0;
    Box_Check:      BOOL;
    NoLenses:       BOOL;
    BoxesFilled:    BOOL;                   (* Manually sets when 5 new boxes are placed*)
    RTRIGCounter_Passed:        R_TRIG;
    RTRIGCounter_Passed2:  R_TRIG;
    Counter_Passed_Trigged:  BOOL:=0;
    Counter_Passed_Trigged2:    BOOL:=0;
    Box5: BOOL;
    Box4: BOOL;
    Box3: BOOL;
    Box2: BOOL;
    Box1: BOOL;
END_VAR
(* Flexlink Conveyor unit will convey boxes with lenses, SCARA robot will pick a lens from the box
A box will be conveyed to the picking spot, when lens is picked, the same box will conveyed one lap then
another lens will be picked from the same box until that box is full
```

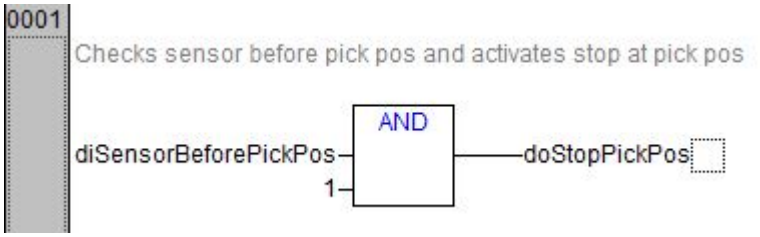
A Counter will be running parallell to the process when lens is picked so it can be noticed when box is emty, then take the next box,
if all boxes are emty, message eLIPS that new lenses needs to be delivered *)



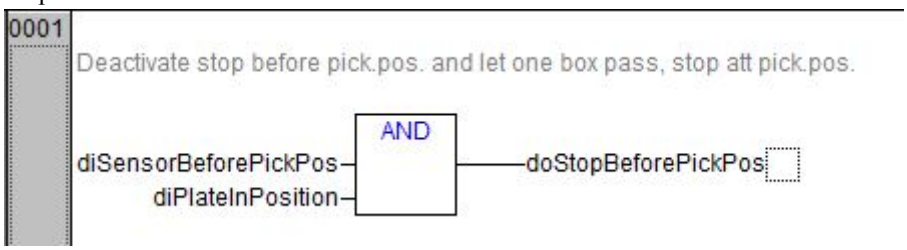
Init:
(*Step activates flexlink conveyor*)
(*_____*)
(* Message eLIPS that we are ready *)
Mode := 2;
Stn2_Cycle := 1;
(*ACS380 ENP Motor Drive Setup*)
(*_____*)

```
        fbMotor_Speed_Control_2.xMotor_Start := 1;              (* Motor always enable for ENP *)
(*!!!!!!!!!!!!!!!  Check this before moving on to next step (intead of Stn2_Command = 1) in action..we had to
remove to make communication with robot studio work!!!!!!!!*)


        IF Stn2_Command = 1 THEN                                                        (*If
eLIPS sends start signal *)
                rFlexlinkMotor_Percent := 70;
                Stn2_Cycle              := 2;                                           (* Sends
signal to eLIPS, running *)
                doLine2Running  := 1;                                   (* Sends signal to eton, running
*)
                NoLenses        :=      0;
                BoxNr           :=      4;

                (*Function block for starting flexlink *)
                fbMotor_Speed_Control_2(
                xENA                                            :=              TRUE,
                xMotor_Start                                    :=              xFlexlinkMotorSpeed,
                xEmergency_Stop_Coast           :=              xEmrMotorSpeed_Coast,
                xEmegency_Stop_Ramp             :=              xEmrMotorSpeed_Ramp,
                xMotor_Reset                                    :=              xFlexlinkMotorReset,
                rMotor_Speed_Command :=         rFlexlinkMotor_Percent,
                rCMD_Speed_Min                  :=              0,
                rCMD_Speed_Max                  :=              100,
                iAct_Speed                                      :=
        goMotorSpeedFlexlink,
                iAct_Torque                                     :=              ,
                wStatus_Word                    :=              StatusFlexlink,
                iSpeed_Ref                                      =>
        MotorSpeedRefFlexlink,
                iTorque_Ref                                     =>              ,
                wControl_Word                   =>              CmdFlexlink,
                xMotor_Ready                                    =>              ,
                xMotor_Running                  =>              ,
                xMotor_Alarm                                    =>              ,
                xMotor_Local_Control            =>              ,
                wErrorNo                                                =>              );
        ELSE
                                        (*Stand still if eLIPS tells not to run*)
                fbMotor_Speed_Control_2.rMotor_Speed_Command            :=              0;
                rFlexlinkMotor_Percent
                :=              0;
        END_IF
```

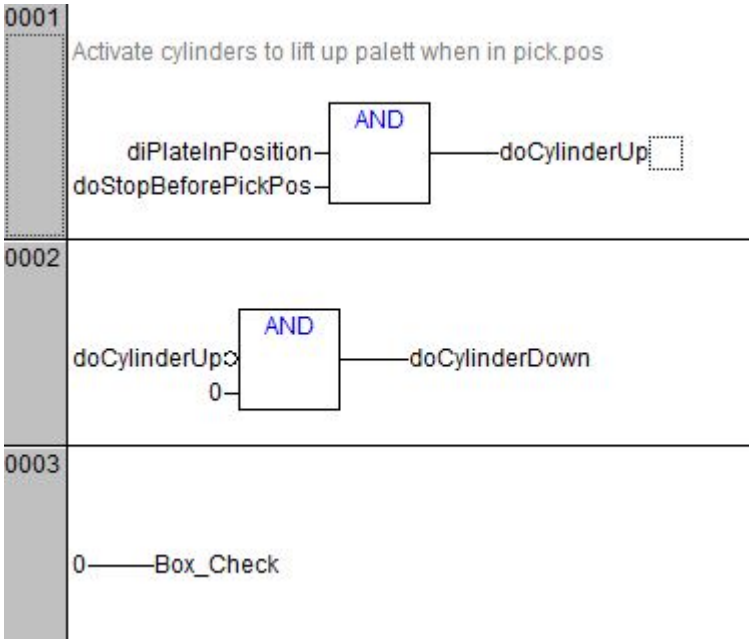
Step 1:



Step 2:



Step 3:

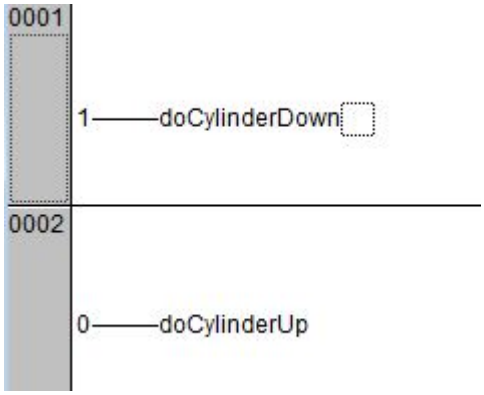


Step 4:

```
(* Step handles handshake between robot and PLC, handshake includes which lens that shall be picked *)
(*_____________________________________________________________________________*)
        IF diLensInFixture = 0 AND ((OUT1_lens = 1 AND OUT2_lens =0) OR (OUT1_lens = 0 AND
OUT2_lens =0)) THEN
                CASE L OF
                        5:      goPLCComBits2 := LensNr;
                                                (*Robot pick lens nr x *)
                                L := 6;
                        6:      doPLCStrobe2 := 1;
                                                (* Activates handshake between robot
*)
                                L := 10;
                        10:     IF diSCARAStrobe=1 AND giPLCCombits2=goPLCComBits2 THEN (*
Check that robot communication bits are correct and that strobe is high*)
                                        doPLCStrobe2 := 0;
                                                (* Set strobe low to end handshake*)
                                        L := 15;
                                END_IF
```

```
                15:     IF diSCARAStrobe = 0 THEN
                                        (* Handshake done when robot strobe is low*)
                            goCheckPLCComBits2 := LensNr;
                            (*Signal for checking what robot has done, up on panel *)
                            HandshakeSCARADone := 1;
                            IF LensNr = 50 THEN
                                    LensNr := 1;
                                                (* When all lenses in one box
is gone, clear variable and go to next box *)
                                    Box_Check := 1;
                                    BoxNr := BoxNr - 1;
                            ELSE
                                    LensNr := LensNr+1;
                                    Box_Check := 1;
                            END_IF
                            L := 5;
                        END_IF
            END_CASE
        END_IF
```

Step 5:



Step 6:

```
        doStopPickPos           := 0;
        doStopBeforePickPos := 0;
        Box5                            := 0;
        Box4                            := 0;
        Box3                            := 0;
        Box2                            := 0;
        Box1                            := 0;
        NoLenses                    := 0;
        Counter_Passed          := 0;
        Counter_Passed2 := 0;
        RTRIGCounter_Passed(CLK:= diSensorPickPos);
```

Step 7:

```
(* Counting Passed Boxes, stop when new full box is in position *)
(*_____________________________________________________________*)
(*R_Trig when one box passes,The function block R_TRIG detects a rising edge.  *)
(*_____________________________________________________________*)
        RTRIGCounter_Passed(CLK:= diSensorPickPos);
        RTRIGCounter_Passed2(CLK:= diSensorBeforePickPos);
```

```
	Counter_Passed_Trigged2 := RTRIGCounter_Passed2.Q;			(*Trigger on sensor before pick pos*)
	Counter_Passed_Trigged := RTRIGCounter_Passed.Q;			(*Trigger on sensor on pick pos*)

(*_____*)
	(* Counting passed boxes *)
	IF	Counter_Passed_Trigged = 1 THEN
		Counter_Passed := Counter_Passed + 1;
	END_IF

	IF Counter_Passed =0 THEN
		Counter_Passed:=1;
	END_IF

	IF Counter_Passed2 =0 THEN
		Counter_Passed2:=1;
	END_IF

	IF	Counter_Passed_Trigged2 = 1 THEN
		Counter_Passed2 := Counter_Passed2 + 1;
	END_IF

(*Checks boxes passed and controls which box shall be picked from and that the last lens not is picked from the box*)
	IF Counter_Passed = 5 AND BoxNr = 4 AND  LensNr<>1 THEN
		Box5 := 1;
	ELSIF Counter_Passed = 5 AND BoxNr = 3 AND LensNr <> 1 THEN
		Box4 := 1;
		doStopPickPos:=1;
	ELSIF Counter_Passed = 5 AND BoxNr = 2 AND  LensNr<>1 THEN
		Box3 := 1;
		doStopPickPos:=1;
	ELSIF Counter_Passed = 5 AND BoxNr = 1 AND LensNr<>1 THEN
		Box2 := 1;
		doStopPickPos:=1;
	ELSIF Counter_Passed = 5 AND BoxNr = 0  AND   LensNr<>1THEN
		Box1:=1;
		doStopPickPos:=1;
	END_IF

(*Checks boxes passed and controls which box shall be picked from and that the last lens is picked from the box*)
	IF Counter_Passed = 5 AND BoxNr = 3 AND LensNr=1 THEN
		Box4 := 1;
		Counter_Passed2:=0;
	ELSIF Counter_Passed = 5 AND BoxNr = 2 AND LensNr=1 THEN
		Counter_Passed2:=0;
		Box3 := 1;
	ELSIF Counter_Passed = 5 AND BoxNr = 1 AND LensNr=1 THEN
		Counter_Passed2:=0;
		Box2 := 1;
```

```
        ELSIF Counter_Passed = 5 AND BoxNr = 0  AND LensNr=1  THEN
                Counter_Passed2:=0;
                Box1 := 1;
        ELSIF Counter_Passed = 5 AND BoxNr = -1  AND LensNr=1  THEN
                NoLenses := 1;              (*Message panel*)
END_IF

Step 8:
(* If there are no lenses, stop flexlink conveyor *)
rFlexlinkMotor_Percent := 0;
(* When boxes are filled go to init *)

Step 9:
(* Step checks when box passed sensor before pick pos *)
        RTRIGCounter_Passed2            (CLK:= diSensorBeforePickPos);
        Counter_Passed_Trigged2  :=      RTRIGCounter_Passed2.Q;

        IF      Counter_Passed_Trigged2 = 1 THEN
                Counter_Passed2 := Counter_Passed2 + 1;
        END_IF

Step 10:
(* Step activates stop before picking position *)
        IF diSensorBeforePickPos=0 THEN
                doStopBeforePickPos:=1;
        END_IF
Step 11:
(* Step checks when box passed sensor before pick pos and activetes stop at pick pos *)
        RTRIGCounter_Passed2(CLK:= diSensorBeforePickPos);
        Counter_Passed_Trigged2 := RTRIGCounter_Passed2.Q;

        IF      Counter_Passed_Trigged2 = 1 THEN
                Counter_Passed2 := Counter_Passed2 + 1;
        END_IF

        doStopBeforePickPos:=0;
        doStopPickPos:=1;

Step 12:
        doStopBeforePickPos:=1;
```

Modul 4 Eton och leverans

PROGRAM LensDelivery

```
VAR
        G:                      INT             :=1;
        delivery:INT            :=5;
END_VAR
(*_____________________________________________________________*)
(* Program handles the part when lens is delivered to customer,
when handshake between robot and PLC is done, lens is dropped in fixture on acturator,
eton systems handles transportation of lens and then droppes lens*)
```

```
CASE G OF
(* Robot put lens in fixture handshake *)
        1:      IF HandshakeSCARADone = 1 THEN                  (*When robot picked up a lens
*)
                        doSCARASafePos := 0;
                        G       := 2;
                END_IF
        2:      IF diEtonClear = 1 THEN
                                (* Checks that there are no carriers in the way *)
                        G       := 3;
                END_IF
        3:      goPLCComBits2 := 51;
                G       := 4;
        4:      doPLCStrobe2 := 1;
                G       := 5;
        5:      IF diSCARAStrobe=1 AND giPLCCombits2=goPLCComBits2 THEN (* Check that
robot communication bits are correct and that strobe is high*)
                        doPLCStrobe2 := 0;
                                (* Set strobe low to end handshake*)
                        G       :=9;
                END_IF
        9:      IF diSCARAStrobe=0 THEN
                        doSCARASafePos := 1;
                        G       := 10;
                END_IF
        10:     IF diCarrierAtPickingPos= 1 AND diLensInFixture = 1 THEN        (* Handshake
done when robot strobe is low, robot moved to safe home position*)
                        IN0_lens        := 1;
                                (* Activate acturator, going up, top position (pos
2) *)
                        IN1_lens        := 0;
                        IN2_lens        := 0;
                        IN3_lens        := 1;
                        IN4_lens        := 1;
                        doLensFixtureDown       := 0;
                                (* "Fake" Signal to robot studio *)
                        doLensFixtureUp         := 1;
                                (* "Fake" Signal to robot studio *)

                        goCheckPLCComBits2      := 51;
                        (*Signal for checking what robot has done, up on panel *)
                        HandshakeSCARADone      := 0;
                        G       := 15;
                END_IF
        15:  IF OUT1_lens = 0 AND OUT2_lens =1 THEN
(* Checks if acturator is moving*)
                        G       := 20;
                END_IF
        20: IF OUT1_lens = 1 AND OUT2_lens = 0 AND diEtonRunning = 1 THEN       (* When
actuators is in top positon and carrier has lens, goto pos 1 (home) *)
                        IN0_in  := 1;
                        IN1_in  := 0;
```

```
                    IN2_in    := 1;
                    IN3_in    := 0;
                    IN4_in    := 1;
                    doLensFixtureDown          := 1;
                        (* "Fake" Signal to robot studio *)
                    doLensFixtureUp            := 0;
                        (* "Fake" Signal to robot studio *)
                    doDeliveryDone             := 0;
                        (*Resets delivery done so new lens can be delivered *)
                    G          := 25;
                END_IF
            25: IF OUT1_lens = 0  AND OUT2_lens = 1 THEN
        (* Checks if acturator is moving*)
                G          := 30;
                END_IF
            30: IF OUT1_lens = 1 AND OUT2_lens = 0 THEN
        (* When actuators is in bottom positon (pos 1)*)
                    G          := 31;
                END_IF
            31: IF diLensInFixture = 0 THEN
                    doDeliveryDone := 1;
                            (* Sucessfully delivered lens to eton carrier *)
                    G          := 1;
                END_IF
        END_CASE

        CASE delivery OF
            5:      IF diCarrierAtDelivery = 1 AND diHandAtDelivery = 1 THEN
                        doReadyToDrop := 1;
                        delivery              := 10;
                    END_IF
            10:     IF diLensInCarrier = 0  THEN
                        (* Look up if this works, it always sets cycle= done, not only when
the lens is delivered*)
                        doLensInCarrier  := 0;
                            (*Photocell for carrier empty after drop, not sure what it does.. *)
                        doCarrierEmpty   := 1;
                            (* Sucessfully delivered lens from eton, carrier is empty *)
                        Stn2_Cycle                := 4;
                                (* Message eLIPS that lens is delivered *)
                        delivery                  := 5;
                    END_IF
        END_CASE
```

Bilaga 3 - VFAT-lista

Function	What should be checked?	Test	OK	Not OK	Comment	Date
Stn1.Elipse	Does startorder from Elipse work?	Check that PLC receives startorder and starts process	x			2017-05-11
Stn1.PLC	Programstart PLC	Check that PLC starts the process	x			2017-05-11
Stn1.Sensors magazine	Does PLC handle empty/not empty magazine?	Check that correct process is running depending on sensor signals	x			2017-05-11
Stn1.Tool IRB1200	Does the tool work?	Can robot pick and release sheet?	x			2017-05-11
Stn1.Tool IRB1200	Does robot handle if vacuum guard goes inactive	Routine that checks vacuumguard and sends error signal		x	Function is not implemented	2017-05-11
Stn1.Printer	Does printer work?	Scan print	x		Elipse handles printer	2017-05-11
Stn1.Scanner	Does scanner work?	Check status signal from elipse		x	No signal received from Elipse	2017-05-11
Stn1.Sensor input	Is start signal set depending on sensor signal and robot status signal?	Check sensor signal, robot signal and PLC signal	x			2017-05-11
Stn1.Actuator input	Does communication between PLC and actuator work?	Check PLC signal, movement of actuator and and input signals to PLC	x			2017-05-11
Stn1.Stamp	Does communication between PLC and servo work?	Check signals and movement of stamp cylinders	x			2017-05-11
Stn1.Sensor output	Is start signal set depending on sensor signal?	Check sensor signal and PLC signal	x			2017-05-11
Stn1.Actuator output	Does communication between PLC and conveyor work?	Check PLC signals and movements of conveyors	x			2017-05-11
Stn1.Sensor flip	Is stop signal and actuator start signal set depending on sensor signals?	Check sensor signal and PLC signal	x			2017-05-11
Stn1.Actuator flip	Does communication between PLC and actuator work?	Check PLC signal, movement of actuator and and input signals to PLC	x			2017-05-11
Stn1.Elipse	Is sheet delivered to customer?	Check signal sent to elipse when flip is up	x			2017-05-11
Stn1.Robot handshake	Does handshake between robot and PLC work?	Check flowchart and signal exchange	x			2017-05-11
Stn1.Scrapping	Does PLC and robot handle faulty QR-codes?	Check PLC signal to robot and robot movements	x			2017-05-11
Stn2.Elipse	Does start order work?	Check that PLC receives startorder and starts process	x			2017-05-11
Stn2.Flexlink conveyor	Does start of conveyor work?	Check PLC signal and conveyor movements	x			2017-05-11
Stn2.Flexlink stops	Does communication between PLC and stops work?	Check signals and movement of stops	x			2017-05-11
Stn2.Flexlink stops	Are the stops in correct position?	Check if plates are stopped?	x			2017-05-11
Stn2.Flexlink inductive s	Are sensors in correct position?	Checkpoint(sensor before pick pos., plate in pos. and at pick pos.)	x			2017-05-11

Stn2.Flexlink cylinders	Does communication between PLC and cylinders work?	Check PLC signal, cylinder movements and input signals to PLC	x			2017-05-11
Stn2.Robot	Does the tool work?	Can robot pick and drop lens?	x			2017-05-11
Stn2.Robot	Does the robot pick correct lens?	Check communication between robot and PLC and robot movements	x			2017-05-11
Stn2.PLC	Does the program handle if a box is empty?	Check if lens position is reset and another box selected	x			2017-05-11
Stn2.PLC	Does the program handle if all boxes are empty?	Check PLC signal and conveyor movements	x			2017-05-11
Stn2.Sensor lens fixture	Is sensor in correct position?	Check sensor signal when lens is in fixture	x			2017-05-11
Stn2.Actuator lens fixtur	Does communication between PLC and actuator work?	Check PLC signal, movement of actuator and and input signals to PLC	x			2017-05-11
Stn2.Eton	Does eton start signal work?	Check eton signal and movements	x			2017-05-11
Stn2.Eton	Does the "Clear" signal work?	Check signal and position of carriers	x			2017-05-11
Stn2.Eton	Does the "Drop" signal work?	Check signal and if lens is dropped	x			2017-05-11
Stn2.Eton	Does the "Carrier at delivery" signal work?	Check signal and position of carriers	x			2017-05-11
Stn2.Eton	Is the lens in carrier?	Check signal from sensors on eton		x	Function is not impl	2017-05-11
Stn2.Eton	Does the "Carrier at picking position" signal work?	Check signal and position of carriers	x			2017-05-11
Stn2.Elipse	Is "delivered"-signal set when lens is dropped?	Check PLC signals and if lens is dropped	x			2017-05-11
Stn2.Delivery	Is sensor in correct position?	Checkpoint	x			2017-05-11
Stn2.Handshake	Does handshake between robot and PLC work?	Check flowchart and signal exchange	x			2017-05-11
Stn2.Robot	Does the program handle if the robot fails to deliver lens to fixture?	Check sensor after robot movement is done. Send error signal		x	Function is not implemented	2017-05-11
General	Does robot send/PLC receive status signals	Checkpoint		x	Function is not implemented	2017-05-11
General	Does the robots have safety zones?	Check robot safety zones		x	Function is not implemented	2017-05-11
General	Does factory handle emergency stop?	Check if factory shut down		x	Function is not implemented	2017-05-11
General	Does factory handle controlled stop?	Check if factory does a controlled shut down		x	Function is not implemented	2017-05-11