CHALMERS

Styrsystem för testriggar till linjära aktuatorer

Utveckling och implementering av ett PLC-baserat styrsystem

Examensarbete inom högskoleingenjörsprogrammet Mekatronik

LUKAS WIKANDER

Institutionen för Signaler och System
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige, 2013
Examensarbete 2013

Control System for Testing of Linear Actuators
Development and implementation of a PLC based control system
F. LUKAS WIKANDER

Examinator: Bertil Thomas
Department of Signals and Systems
Chalmers University of Technology
412 96 Göteborg
Sweden

+46 31 772 10 00

The cover depicts the Siemens LOGO! PLC which is used by the developed control system, along with its modules.

Göteborg, Sweden 2013

FÖRORD

Denna rapport behandlar ett examensarbete på cirka tio veckor inom högskoleingenjörsprogrammet mekatronik (180 hp). Författaren går sitt tredje år på Chalmers Tekniska Högskola.

Examensarbetet har utförts hos SKF Actuation Systems.

Författaren skulle vilja framföra sitt tack till Johanna Pauli för att hon gav möjligheten till att få göra examensarbetet hos SKF Actuation Systems. Ett tack framförs även till Sven Svensson för att han varit ett utomordentligt stöd som handledare vid företaget och för att han fått igång arbetet igen då författaren själv kört fast. Slutligen vill författaren tacka Pär Högberg, som trots att han inte var handledare till detta examensarbete varit väldigt hjälpsam och bistått med information och råd, men framför allt för att han lagt ned mycket av sin tid på och varit intresserad av båda av de examensarbeten som samtidigt utfördes på SKF Actuation Systems.

SAMMANFATTNING

Författare: Lukas Wikander
Institutionen för Signaler och System
Chalmers Tekniska Högskola

Denna rapport behandlar framtagandet av ett nytt PLC-baserat styrsystem för testlaboratoriet för linjära ställdon hos SKF Actuation Systems. Den existerande testmiljön har sedan länge haft en föråldrad styrutrustning med låg robusthet och tillförlitlighet, med utnyttjande av manuella kontroller utöver den manuella mätningen av testdata. Av den anledningen ansågs behov finnas för ett nytt, automatiserat, styrsystem.

Metoderna som används för framtagandet av det nya testsystemet är intervjuer av berörda personer för att ta reda på vari problemen med det gamla testsystemet ligger, samt hur ett nytt system bör se ut. Efter detta jämförs olika lösningar – huvudsakligen LabVIEW samt två typer av PLC:er – för att få reda på vilken som bäst uppfyller de krav som tagits fram. Slutligen realiseras lösningen i form av en testmodell.

Den resulterande modellen består av en Siemens LOGO! PLC inhyst i ett kopplingsskåp tillsammans med ett antal andra moduler och används för test av två olika ställdon; det ena med en växelströmsmotor och det andra med en borstlös likströmsmotor. Den manuella mätningen av testdata har ersatts med automatisk mätning och loggning vilken sköts genom PLC:n. Mekaniska givare har ersatts av beröringsfria, vilket förbättrar tillförlitlighet och kvalitet hos mätresultaten.

Nyckelord: utvärdering, PLC, Siemens LOGO!, LabVIEW, Finite State Machine, Function Block Diagram, testlaboratorium, linjärt ställdon.

ABSTRACT

Author: Lukas Wikander
Department of Signals and Systems
Chalmers University of Technology
The full report is written in Swedish.

This report covers the development of a new PLC based control system for a linear actuator testing laboratory at SKF Actuation Systems. The previous testing environment has for a long time had outdated control equipment with low ruggedness and reliability, requiring considerable amounts of manual checking in addition to the measurement and collection of test data which has needed to be done by hand. Therefore, SKF was interested in a new system, but were not sure which solution would best meet their needs.

The methods used for development of the new control system are conduction of interviews with relevant persons to determine what the actual problems were and what is required of a new system, followed by comparison of various solution concepts to find which best meets SKF needs. The investigated concepts were LabVIEW and two different types of PLCs. Finally, this work also covers implementation of the chosen solution in the form of a model.

The resulting model consists of a Siemens LOGO! PLC encased in a junction cabinet along with a few other modules and is used for the testing of two different actuators; one with an AC motor and the other with a brushless DC motor. Manual measurement of test data has been replaced with automatic measurement and logging done by the PLC. In addition, mechanical position sensors have been exchanged for inductive positioning sensors, prolonging sensor life and improving test data quality.

Keywords: evaluation, PLC, Siemens LOGO!, LabVIEW, Finite State Machine, Function Block Diagram, testing laboratory, linear actuator.

INNEHÅLL

FÖRKORTNINGAR

PLC – *Programmable Logic Controller*.

MCU – *Microcontroller Unit*.

FBD – *Function Block Diagram*.

SFC – *Sequential Function Chart*.

FSM – *Finite State Machine.*

NC – *Normally Closed.*

NO – *Normally Open.*

HMI – *Human-Machine Interface*.

BLDC (-motor) – *Brushless Direct Current* (-motor).

UDF – *User Defined Function*.

SHF – *State Handle Function*.

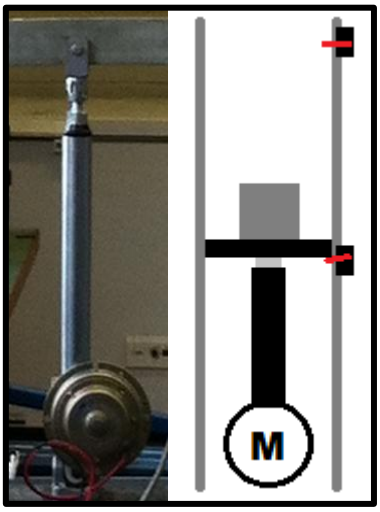
SDF – *State Determine Function*.

1 INLEDNING

Detta kapitel innehåller en kortare bakgrund till projektet, syftet och målet med projektet, en problemuppställning samt några avgränsningar vad gäller arbetet. Vidare innehåller kapitlet några skissartade bilder av den vanligaste typen av testuppställning som används idag.

1.1 Bakgrund

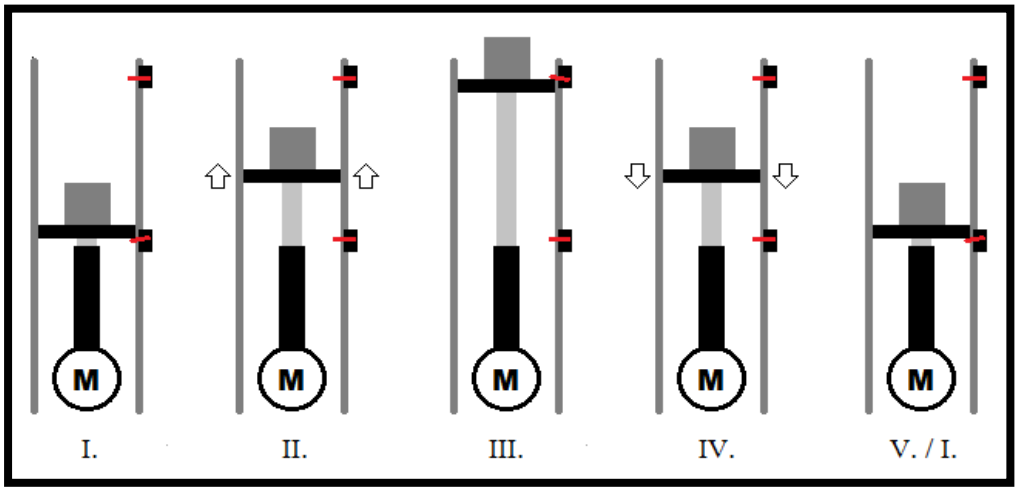
I testlabbet hos SKF Actuation System utförs i dagsläget olika typer av tester av linjära ställdon, som var och en i dagsläget har en del problem. Ett sådant test görs med hjälp av en testrigg (se Figur 1.1, till höger) i vilket ställdonet körs mellan de två ändlägena (mekaniska ändlägesgivare, markerade med röd färg). Exempel på vanligare tester som utförs är;



Figur 1.1. Principskiss av den vanligaste testriggstypen, och en bild på en verklig testrigg att jämföra med.

- Livslängdstester, där det testas ifall en produkt klarar ett krav (ställt av t.ex. en kund) på ett visst antal körcykler. Det kan också vara så att donet helt enkelt körs fram och tillbaka tills det havererar, och att det därefter noteras hur många cykler donet klarade av. Livslängdstest är den absolut vanligaste typen av test som utförs i testlabbet.
- Prestandatester – produkten testas vad gäller motortemperatur och strömförbrukning vid vissa förhållanden. Prestandatesterna är även till för att få reda på ifall (eller visa att) produkten beter sig annorlunda vid sådana förhållanden.
- Intermittenstester, som testar hur ofta ett ställdon kan köras utan att temperaturen överstiger ett bestämt gränsvärde.
- Förstörande tester, det vill säga att donet körs i ett extremfall (exempelvis med hög last eller i någon extrem miljö) för att se om det håller eller inte.
- Klimattester, vilket innebär att ställdonet körs en längre tid i en påfrestande miljö såsom till exempel extrem kyla eller hög luftfuktighet.

Livslängdstester utförs idag oftast med en enkel fram- och återgående cykel (se Figur 1.2).



Figur 1.2. Enkel testcykel med fram- och återgående rörelse.

Tyvärr är det så att närhelst ett test som innebär något mer avancerat än en ren fram- och återgående rörelse (exempelvis varierande hastighet eller att något skall vara annorlunda var femte körcykel) skall ställas upp krävs en skräddarsydd lösning, något som gör ett sådant test svåruppreprepat. Ett test enligt ovan behöver egentligen ständig mätning och kontroll för att det skall vara så exakt som möjligt, men då detta inte är realistiskt att göra manuellt kontrolleras testriggarna idag cirka en gång per dag.

1.2 Syfte och problemformulering

Syftet med detta arbete är att kartlägga de testförfaranden som finns idag, önskemål om ytterligare testförfaranden, samt att kartlägga de praktiska problem som finns vad gäller test. Vidare är syftet även att ta fram och ställa upp en exempelmodell för en lösning som löser existerande problem.

De huvudsakliga problemen som finns idag är följande (övergripande beskrivning):

- Svårt att skapa kundanpassade test: det är komplicerat att skapa en testcykel som är mer avancerad än ”standardtestet”; den enkla fram- och återgående cykeln. Till exempel körs ställdonen idag med konstant last i testen, något som inte motsvarar den verklighet som de utsätts för. För ett test som är mer verklighetsnära (och därför mer invecklat) krävs en unik lösning som inte alltid kan återskapas vid andra, liknande, test.
- Låg upplösning och -spårbarhet av mätdata: testdata måste antecknas för hand vilket gör datan svårhanterlig – det är omständigt att göra om sådan data till exempelvis ett diagram. Dessutom har datan inte så hög upplösning som kanske skulle kunna vara intressant för analys av produkten, då det vore kostsamt att ha någon som står och mäter hela tiden.
- Degenerering hos givare: ändlägesgivarna går ofta sönder eller flyttas vilket utgör ett ständigt irritationsmoment för dem som ställer upp testen. Ett test görs för det mesta inte om då en givare flyttats men givetvis är datan som fås ur ett sådant test inte lika tillförlitlig som den kunnat vara.

1.3 Mål

Målet för detta projekt är att en testmiljö skall tas fram som är bättre än den som finns idag, anpassat till dagens kunder och framtida produktutveckling. Den slutligt framtagna testmiljön skall ha vägts mot andra alternativ, så att det är säkerställt att lösningen är den bästa.

1.4 Avgränsningar

Trots att det är ett stort problem i testverksamheten att de mekaniska reläerna för motorstyrning slits, behandlas inte det problemet i detta arbete. Däremot utförs ett annat projekt parallellt med detta som rör halvledarreläer som ersättning för dagens motorstyrning.

Förutom den ovanstående avgränsningen skall klargöras att detta projekt endast behandlar det berörda testlabbet hos SKF Actuation Systems – det är möjligt att lösningen skulle gå att applicera på andra tillämpningar men detta är inget som tas i beaktande vid analyser.

1.5 Rapportens disposition

Då själva projektet är indelat i tre tydliga delar (kartläggning av nuvarande situation, framtagning av tekniska lösningsförslag, samt skapande av testmodell) disponeras de flesta kapitlen i rapporten på samma sätt. Rapporten har skrivits allteftersom de tre delarna utförts, vilket till exempel kan medföra att kapitlet ”Genomförande – framtagning av förslag på ny testmiljö” har delar som bygger på kartläggningsdelen av resultatkapitlet (detta kapitel kommer efter kapitlet Genomförande).

2 TEORETISK REFERENSRAM

Detta kapitel beskriver de begrepp som används i rapporten vilka kanske inte är kända för läsaren, alternativt är det sådant som behöver preciseras exakt vad som menas med begreppet då det annars skulle kunna misstolkas.

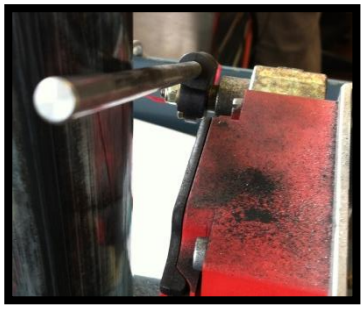
2.1 Aktuator

En aktuator (eng. *actuator*), eller ställdon, är en mekanisk anordning som används för att åstadkomma rörelse[1]. Det kan till exempel vara hydraulcylindrar eller pneumatiska cylindrar men den typ som huvudsakligen kommer att behandlas i detta dokument är aktuatorer som använder motorer för att utföra rörelsen. Några exempel på områden där aktuatorer används är inom pappersindustrin, sjukvården och i terrängfordon[2]. I detta projekt behandlas endast ställdon som utför linjära rörelser, då det är just sådana som testas i det berörda testlabbet.

Precis som för hydraulcylindrar och pneumatiska cylindrar benämns det yttre läget ”plusläge” och det inre läget ”minusläge”. Dessa ändlägen detekteras oftast med hjälp av ändlägesgivare inmonterade i ställdonet (en för varje läge). För det mesta är det inte önskvärt att donet når det så kallade mekaniska ändläget, det vill säga då donet bromsas mekaniskt på grund av att det inte kan köras längre (detta medför stort slitage på donet). Istället bromsas ofta motorn eller så tillsätts en yttre broms innan det mekaniska ändläget nåtts.

2.2 Mekanisk positionsgivare

Mekaniska positionsgivare är vad som används i testlabbet hos SKF Actuation Systems idag – ställdonen har egna ändlägesgivare monterade internt, men dessa används inte vid testning då de mekaniska ändlägesgivarna är enklare att flytta och byta ut då de slitits sönder. Figur 2.1 föreställer en sådan givare, här ej monterad för användning.



Figur 2.1. Bild på en av testlabbets mekaniska ändlägesgivare.

Ändlägesdetekteringen innefattar en mekanisk överföring av information i form av fysisk kontakt mellan oket och staven (synlig i bild). Staven är kopplad till ett tungelement (*Reed sensor*) som översätter informationen till en digital elektrisk signal.

2.3 Induktiv positionsgivare

Induktiva givare (*Figur 2.2*) används då det är önskvärt att detektera metalliska föremål utan fysisk kontakt. Med hjälp av en ström som går genom sensorn, en spole och en oscillator genereras ett magnetiskt fält vars oscillationsamplitud dämpas då ett yttre metalliskt föremål förs in i det. Amplitudändringen hos det elektromagnetiska fältet detekteras med hjälp av en

Figur 2.2. En induktiv positionsgivare.

[1] en.wikipedia.org/wiki/Actuator, acc. 2013-03-28.
[2] www.skf.com/group/products/actuation-systems/linear-actuators/cat-series/index.html, acc. 2013-03-28.

krets som ett- eller nollställer utgången (beroende på om sensorn är NC eller NO) vid ett visst tröskelvärde hos amplituden[3].

Då induktiva givare är beröringsfria har de en mycket högre förväntad livslängd än de mekaniska givarna som beskrivits ovan, då det inte finns någon del som utsätts för mekaniskt slitage och således inte nöts ned. Detta kan ofta vara fallet för stavarna på de mekaniska ändlägesgivarna (Figur 2.1) som i högsta grad utsätts för ett sådant slitage[4]. När en givare slits på ett sådant sätt försämras dess precision, och ger ett ändläge som förändras med slitaget (utöver att hela givaren kan flyttas längs med axeln den är monterad på).

2.4 Tillståndsövervakning

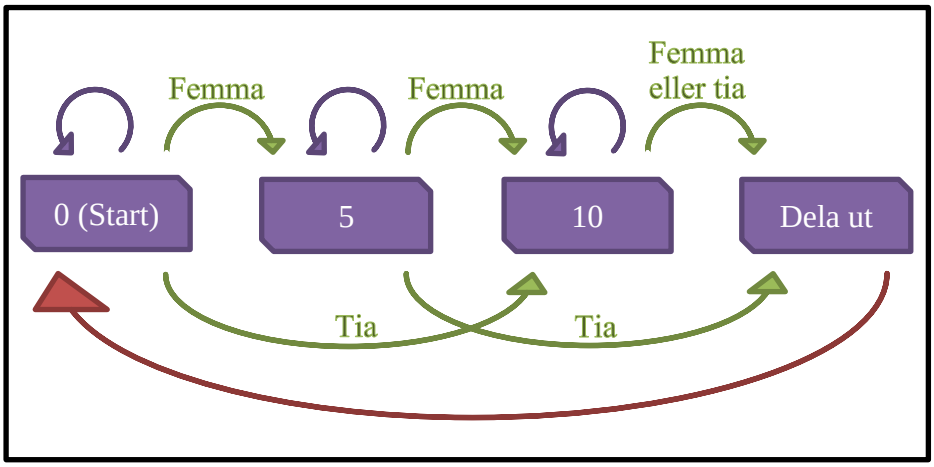
Tillståndsövervakning (eller *condition monitoring*) innebär att en produkt övervakas under drift för att användaren skall få information om nära förestående haveri, innan det faktiskt sker. Denna information kan användaren dra nytta av genom att underhåll kan planeras och schemaläggas i förväg, delar kan beställas, och andra underhållsarbeten kan utföras samtidigt – kostnaden för ett sådant stopp blir således mycket mindre. Förutom den rent ekonomiska vinsten medför övervakningen också en ökad säkerhet i och med att en maskin kan stängas av då någon del är på väg att gå sönder, istället för att maskinen körs tills delen i fråga faktiskt går sönder och vilket eventuellt skulle kunna medföra personskada eller skada på andra delar av maskinen[5].

Exempel på tillståndsövervakning skulle kunna vara att temperaturen på en motor mäts övervakas; det måste då tidigare ha skett tester av motorn så att information finns kring temperaturens beteende vid normal drift, samt hur temperaturen (eventuellt) ändras då motorn är på väg att haverera. Aktuella värden jämförs vid drift med testvärden och motorn kan stoppas ifall exempelvis en stegring av temperaturen skulle observeras (eller något annat fenomen hos temperaturen som observerats vid test precis innan haveri).

[3] www.fargocontrols.com/sensors/inductive_op.html, acc. 2013-05-16.
[4] Intervju av Kjell Hermansson, 2013-03-18.
[5] www.skf.com/group/products/condition-monitoring/index.html?alias=www.skf.comfcm, acc. 2013-04-16.

2.5 Tillståndsbaserad programmering

Tillståndsbaserad programmering är ett sätt att bygga upp program på, och ett sådant program benämns på engelska *Finite State Machine* (*FSM*). En FSM består av ett finit antal tillstånd, var och en med olika villkor för att gå över till något av de andra tillstånden. Bilden nedan (Figur 2.3) beskriver hur ett sådant program skulle kunna byggas upp. Programmet som beskrivs av bilden nedan är bara ett exempel, och är tänkt att styra en läskautomat som bara säljer läsk för femton kronor per behållare. Automaten tar inte emot enkronor.



Figur 2.3. Flödesdiagram för en FSM.

De lila rutorna representerar de fyra tillstånden som automaten kan befinna sig i; antingen har den inte fått några pengar alls, eller så har den fått fem kronor, tio kronor (antingen i form av en tia eller två femmor), eller femton alternativt tjugo kronor (och i så fall ger den ut en behållare). Programmet ligger kvar i samma tillstånd (lila pilar) tills att något händer (gröna pilar), med undantag för den sista rutan som direkt går över till den första (röd pil) efter att den gett ut en läskbehållare.

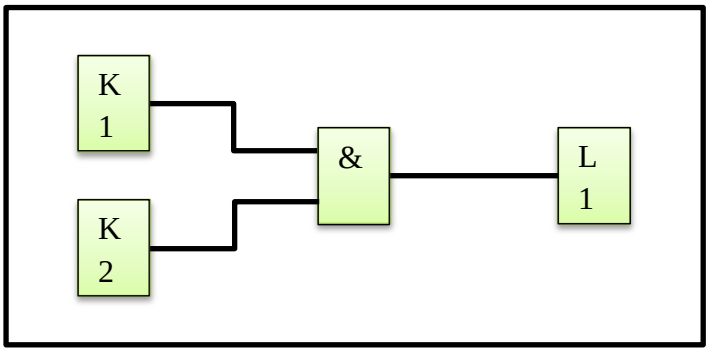
Eftersom bilden endast gjorts i demonstrationssyfte ignorerades det faktum att det kanske borde finnas ett tillstånd för att ge ut en läskbehållare och ge tillbaka fem kronor i växel utifall att två tior tas emot.

2.6 Programmeringsspråk

I underkapitlet nedan beskrivs de programmeringsspråk som använts i projektet. De tre programmeringsspråk som beskrivs är SFC och FBD – dessa är vanliga PLC-programmeringsspråk – samt LabVIEW:s programmeringsspråk.

2.6.1 FBD – Function Block Diagram

FBD är ett programmeringsspråk som används i PLC-system. Ett FBD-diagram består som namnet antyder av sammanlänkade funktionsblock. Funktionsblocken kan till exempel vara logiska grindar, timers, in- eller utgångsblock eller räknare. Ett enkelt exempel på ett FBD-program kan ses nedan (Figur 2.4). För att exemplet skulle bli så lättöverskådligt som möjligt har det inte gjorts i en riktig FBD-programmeringsmiljö.

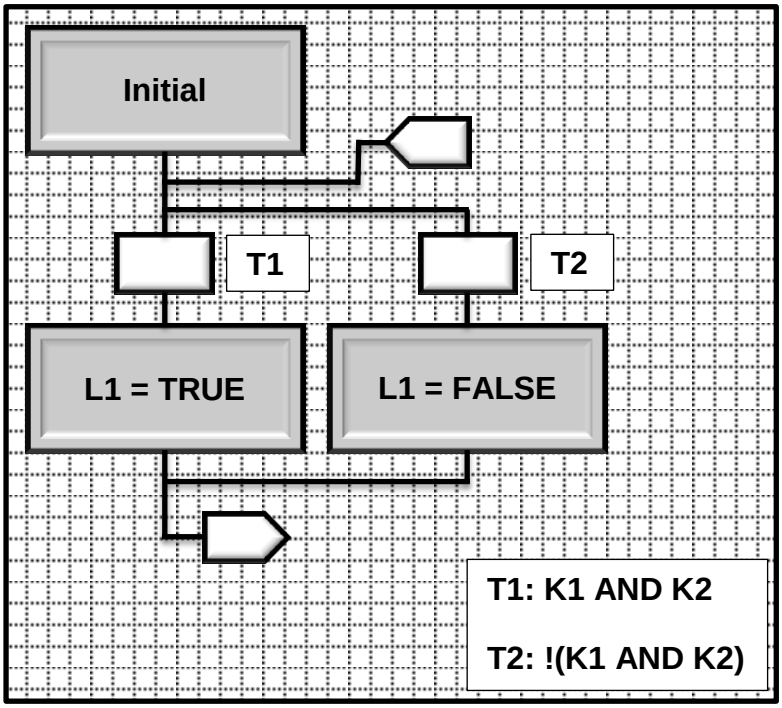


Figur 2.4. Enkelt exempel på ett FBD-program.

Blocken K1 och K2 respektive L1 motsvarar faktiska ingångar och utgångar på PLC:n. Ansluts K1 och K2 till tryckknappar och L1 till en lysdiod kommer lysdioden att lysa endast då båda knapparna är intryckta samtidigt. Ingen användarspecificerad start- eller stoppunkt finns i programmet – alla block gås systematiskt igenom i tur och ordning medan PLC:n är påslagen.

2.6.2 SFC – Sequential Function Chart

Precis som FBD är SFC ett programmeringsspråk som används i PLC-system. Ett SFC-program liknar i många hänseenden ett flödesschema, som ju också beskriver ett skeende enligt en sekventiell struktur. Precis som ett flödesschema har ett SFC-program rutor för handlingar som skall utföras och övergångsvillkor som ser till att handlingsrutor inte utförs vid fel tillfälle. Dessutom har ett SFC-program också en startpunkt som visar var exekveringen av programmet börjar – en stoppunkt är möjlig men inte nödvändig. Nedan återfinnes ett (mycket enkelt) exempel på ett SFC-program.



Figur 2.5. Enkelt exempel på ett SFC-program.

Programmet fyller exakt samma funktion som FBD-programmet i föregående underkapitel. Det må se lite krångligare ut än FBD, men det är endast på grund av att exemplet är så banalt; vid större och mer invecklade program är SFC-program mer lätthanterliga.

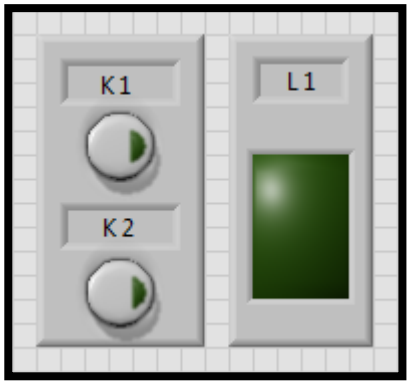
T1 respektive T2 är övergångsvillkor, som styr huruvida programmet tillåts exekvera den underliggande rutan. De pilformade flaggorna representerar hopp i programmet.

Utropstecknet innan villkoret T2 representerar en invertering (1→0 alternativt 0→1).

Det kan noteras att det program som beskrivs av Figur 2.4 och Figur 2.5 enkelt skulle kunna byggas upp i exempelvis programmeringsspråket C genom att lägga en ’if/else’-sats inuti en oupphörlig loop.

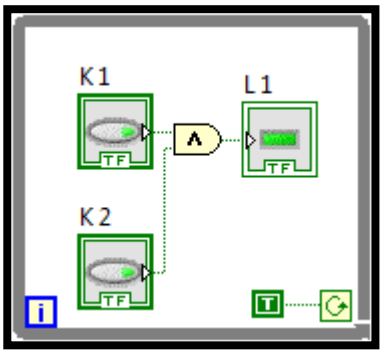
2.6.3 LabVIEW-program

Ett LabVIEW-program består av två huvudsakliga delar som visas i varsitt fönster; den ena delen uppfyller funktioner mest liknande de hos en kontrollpanel – alla knappar, reglage, displayer, realtidsgrafer etcetera visas i det fönstret (av LabVIEW kallat ”Front Panel”). Exempel på hur en sådan kontrollpanel skulle kunna se ut kan ses i ***Figur 2.6***, till höger. Knapparna K1 och K2 respektive LED-lampan L1 skulle lika gärna kunnat kopplas till yttre enheter istället för de simuleringsenheter som LabVIEW-miljön tillhandahåller (som kan ses i bilden).



Figur 2.6. Exempel på LabVIEW:s front panel-ruta.

Den andra delen är den som innehåller själva programmet, bestående av funktionsblock, logiska grindar, in- och utgångar till ’kontrollpanelen’ eller till faktiska utgångar för processtyrning, med mera. Vid programmering sker det huvudsakliga arbetet i detta fönster. Ett program gjort i detta fönster kan ses i bilden till höger (***Figur 2.7***). Den grå pilen som omsluter programmet representerar en while-loop. Den lilla gröna rutan i hörnet är villkoret för att få fortsätta loopen (här är loopen evig). AND-grinden kan ses i mitten i beige.



Figur 2.7. Exempel på LabVIEW:s programmeringsruta.

Så som programmet ser ut nu kontrolleras ingångarna K1 och K2 och utgången L1 korrigeras en gång varje loopiteration; hade dessa istället placerats utanför loopen hade de kontrollerats vid start av loopen och sedan hade loopen återanvänt samma värden som fanns då den startade hela vägen genom sin körning. Dessutom hade inte utgången korrigerats förrän loopen körts klart.

Ett block i ett LabVIEW-program exekveras då alla värden vid dess ingångar finns tillgängliga, och om två block får sina ingångsvärden tillgängliga samtidigt exekveras blocken i den ordning de skapats. Givetvis finns ett sätt att komma runt denna inbyggda exekveringsordning med exempelvis sequence-strukturer.

3 METOD

Detta kapitel beskriver tillvägagångssättet som används för att lösa problemet. Sammanfattningsvis inleds arbetet med en kartläggning för att en stabil grund skall fås, vilket sedan följs upp med studier av olika lösningar till problemet som kartlagts. Avslutningsvis realiseras den lösning som är best anpassad till problemet i form av en testmodell.

3.1 Kartläggning av nuvarande testförfaranden och framtida behov

Den huvudsakliga metoden som används för att måla upp en bild av nuläget är att intervjua olika personer som har eller har haft något att göra med testlabbet. Det resonerades att denna information torde vara den mest tillförlitliga och användbara då det är personer som ställt upp och utfört tester som har mest erfarenhet av hur man ställer upp test, vilka problem som vanligen uppstår och vilka testförfaranden det finns behov av som inte kan göras idag.

Intervjuerna ställs i en första omgång upp för att ge en överblicksvy av testsituationen och de problem som finns samt för att se önskemål rörande ett nytt testsystem. I en andra omgång utförs intervjuer med mer detaljerade frågor. Dessa intervjuer ställs upp med detaljfrågor baserade på testdokumentation och är färre till antalet.

I motsats till intervjuer ger läsning av testdokumentation en mer detaljerad och förhållandevis objektiv vy av testförfaranden och testparametrar, men ger såklart ingen information överhuvudtaget vad gäller behov av test som inte kan göras idag. För att mer information om test som inte kan göras eller som helt enkelt inte görs idag skall fås ur sådan testdokumentation måste beställaren av testet intervjuas mer ingående om testet i fråga. Det är ju den personen som har mest kunskap om testet och som vet vad han/hon egentligen skulle velat göra om han/hon kunde. Dessutom vet den personen vilka problem som uppstått under testets gång.

Utifrån den information som fås ur intervjuer och läsning av dokumentation ställs en kravspecifikation upp där önskvärda funktioner och egenskaper hos ett nytt styrsystem för test listas. Denna kravspecifikation överlåtes sedan till några av de intervjuade personerna så att de kan kontrollera ifall kraven/önskemålen i kravspecifikationen är tillräckliga (eller överdrivna), samt för att kraven och önskemålen skall kunna poängsättas efter hur viktiga de är. Det är av största vikt att denna bedömning görs av personer som har faktisk erfarenhet med testlabbet då ingen förkunskap eller erfarenhet finns hos författaren – en bedömning gjord av en sådan utomstående skulle ha gett en skev bild av vad som skall prioriteras.

3.2 Framtagande av förslag till ny testmiljö

Till en början hålls en del diskussioner för att en del grundläggande information kring olika lösningar skall fås fram. En del av de olika lösningsförslag som fås fram ur denna process testas sedan genom att ett styrprogram för en enkel testcykel tas fram, varefter de olika lösningarna (programmen) kan ställas mot varandra för att eventuella svagheter och styrkor hos dem skall visa sig. Vidare erhålls ur denna provning en del erfarenhet inom de olika miljöerna, vilket medför att en bedömning av miljöerna blir väl underbyggd.

Efter denna initiala provning av programmeringsmiljöer utvärderas deras förmåga att hantera mer invecklade test genom att en befintlig (och invecklad) teststyrning återskapas i de olika programmeringsmiljöerna. Om en programmeringsmiljö anses för komplicerad att använda vid ett enklare test utvärderas den inte heller för mer invecklade tester.

Då båda dessa utvärderingar av miljöer gjorts ställs en Pugh-matris (se bilaga A för beskrivning av en Pugh-matris) upp för att konkret information skall komma ur jämförelsen. De förslag som jämförs i denna matris samt även poängsättningen som kommit ur matrisen presenteras för berörda personer för att de skall ge ett slutligt godkännande av den lösning som väljs med Pugh-matrisen som grund.

Inga vetenskapliga artiklar eller dylikt undersökes då detta område inte direkt kan anses vara ”rocket science”, utan mer en applicering av existerande teknologi.

3.3 Skapande av testmodell

Utöver själva kopplingsarbetet som behövs för att koppla in den valda PLC:n skapas ett program för styrning av test. Dessa test är nu tydligt definierade i form av vilka in- och utgångar som behövs på PLC:n och programmet behöver inte längre ha den väldigt generella form som de ovan nämnda jämförelseprogrammen.

Behov finns av att någon form av strömmätning tas fram eftersom det är ett välfungerande sätt att uppmäta hur väl motorn fungerar – går strömförbrukningen exempelvis upp utan någon yttre påverkan innebär det antagligen att motorn eller växeln slitits. Av den anledningen studeras olika färdiga lösningar för strömmätning i form av integrerade kretsar eller moduler för montering på DIN-skena.

Programmet som skapats och hela uppkopplingen behöver kontrolleras efter att det hela är klart, varför ytterligare arbete läggs på att se till att allting är rätt inkopplat och att programmet fungerar som tänkt. Eftersom en regulator tas fram för en av de två testen måste det även här kontrolleras att den fungerar enligt beräkningar.

4 GENOMFÖRANDE

Detta kapitel beskriver genomförandet av projektet. Precis som det nämnts i kapitel 1.5 är kapitlet nedan indelat i tre delar, nämligen kartläggning, framtagande av förslag samt skapande av testmodell. Kapitlet är förhållandevis kort eftersom resultaten presenteras i kapitel 5.

4.1 Kartläggning av nuvarande testförfaranden och framtida behov

Detta kapitel beskriver arbetet med att kartlägga hur det ser ut idag. Efter att allt som behandlas i detta kapitel avslutats ställdes en kravspecifikation upp och lämnades över till Kjell Hermansson (testingenjör vid SKF Actuation Systems) och Pär Högberg (maskinkonstruktionsingenjör vid SKF Actuation Systems) för att de olika kraven skulle utvärderas.

4.1.1 Intervjuer

Flera intervjuer utfördes till en början för att en överblick över området skulle fås. Fler intervjuer utfördes efter hand då mer kunskap inom området inhämtats. Till att börja med ställdes några enklare frågor upp enligt följande;

- Vad för typer av test gör ni idag?
- Varför utför ni testen?
- Finns det några typer av test som ni inte gör idag men som ni kommer eller vill kunna göra?
- Vad mäter ni under ett test?
- Vilka problem brukar ni stöta på under testning?
- Vad använder ni idag för att styra testen?

Dessutom ställdes några frågor upp med syfte att planeringen kring lösningen skulle kunna komma igång och att ideallösningen därmed kunde finnas med i bakgrunden vid kartläggning och framtagning av lösningsförslag.

- Finns det några funktioner som ni vill ha in i testlabbet som ni inte har idag?
- Hur användarvänligt vill ni att ett nytt testsystem skall vara (helt förprogrammerat, moduluppbyggnad, vill ni kunna programmera allt själva, etc.)?
- Om testdata skall loggas – vad skall i så fall loggas, och i vilket format vill ni ha ut det?

Den förste som intervjuades var Pär Högberg. Många av testen ställs upp av honom och därför sågs han som en viktig person att intervjua. Han intervjuades också om ett test han ställt upp för att pröva en växels verkningsgrad då den utrustas med en nyare motor.

Därefter intervjuades Kjell Hermansson, som är den som utför de flesta av testen i själva labbet. Eftersom han varit med i så många av testen hade han mycket kunskap om vanliga fel som uppstår och vad som är viktigt hos ett nytt testsystem.

Den tredje personen som intervjuades var Jan Lorentzon, områdeschef på SKF Actuation System. Vid det är laget noterades att många intervjusvar var likadana på vissa frågor (till exempel vad som mäts) och därför skalades intervjuerna ned något.

Peter Hansson, som också arbetar som maskinkonstruktionsingenjör, intervjuades efter det. Han hade väldigt detaljerad kunskap om det test han själv ställt upp som pågick samtidigt som detta projekt utfördes – ett test av tätningar med koldamm som efterliknade en kunds applikation. Lyckligtvis ’snöade hela intervjun in’ på ett specifikt test då det redan efter några få inledande intervjuer fanns tillräcklig överblick av dagens läge.

4.1.2 Läsning av testdokumentation

Inledningsvis lästes åtta testprotokoll igenom för att få utförlig information kring testförfaranden. Resultatprotokollen lästes inte igenom då testprotokollen ansågs ge tillräcklig information. Precis som det antagits tidigare visade det sig att Per Högberg beställt många av testen. Efter läsning av testprotokoll noterades ett antal möjliga lösningar på problem som ansågs finnas i respektive test; till exempel hur det skulle vara enklare att avgöra om en temperatur stabiliserat sig kring ett visst värde eller fortfarande var transient.

När läsningen av ett testdokument var klar intervjuades den som beställt testet för att mer ingående information skulle fås kring testet och för att utröna ifall det finns metoder som egentligen borde ha använts och som är bättre än de som beskrivits i det berörda testdokumentet. Problem med dagens testförfaranden framkom således tydligt.

4.2 Framtagande av förslag till ny testmiljö

Då det i tre av intervjuerna direkt kom fram att de nuvarande mekaniska ändlägesgivarna var ett problem (antingen flyttas de undan för undan under ett test och måste således flyttas tillbaka eller så överhettas de och måste då bytas) var det en självklarhet att dessa behövde bytas ut mot någon form av beröringsfria givare. De alternativa givartyper som studerades var huvudsakligen induktiva lägesgivare.

Vidare fanns från företagets sida några föreslagna styrningslösningar; PLC och LabVIEW. Utöver dessa fanns erfarenhet inom MCU (*Microcontroller Unit*)-programmering varför MCU studerades, och av samma anledning studerades även MATLAB något. Dessa tre lösningsförslag jämfördes med varandra genom att ett enkelt program för teststyrning skapades, så att en uppfattning kunde fås kring hur komplicerad programmeringsmiljön var att använda samt vilken funktionalitet som fanns. För vidare beskrivning av detta, se kapitel 4.2.1, ”

Referenstest – grund för jämförelse av lösningar”. Efter dessa enklare program skapats gjordes två mer invecklade program för styrning (eller egentligen bara ett; det ena fanns redan som styrning av ett annat test, det andra gjordes i en annan miljö men för att uppfylla samma funktion).

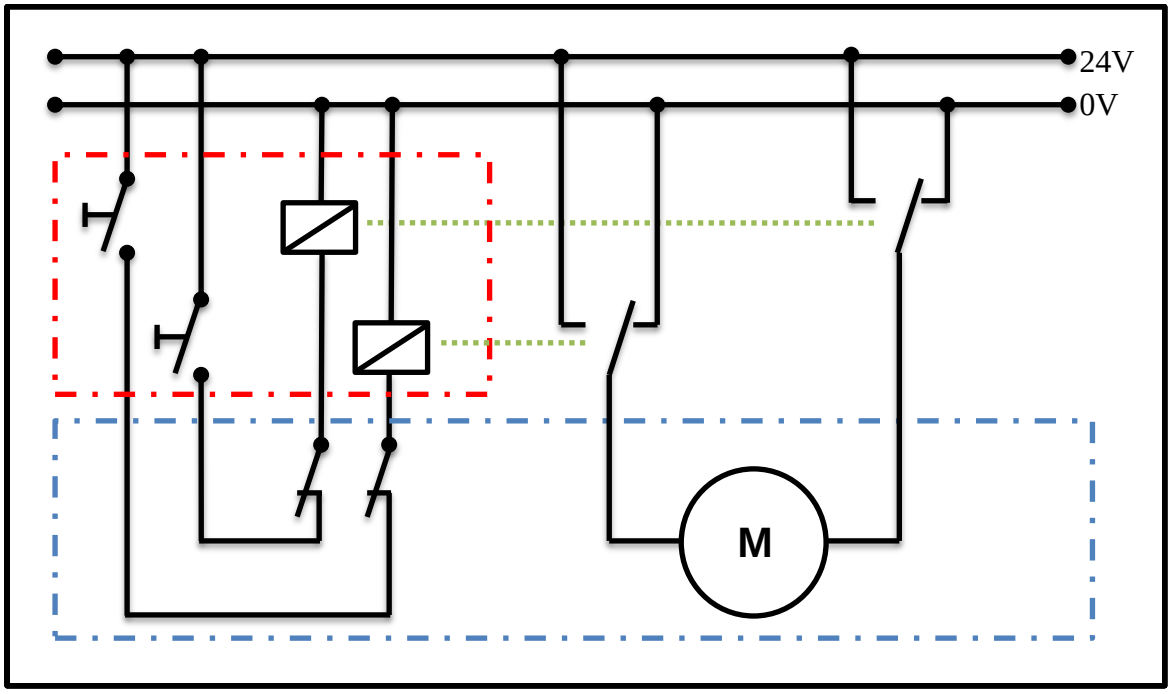
Två typer av PLC-miljöer studerades; en med språket begränsat till funktionsblock (Siemens LOGO! / Soft Comfort) och den andra med SFC-programmering tillgänglig (Siemens SIMATIC S7-1200).

Ett studiebesök gjordes till SKF:s testlabb, Test Centre Göteborg, för att information skulle fås kring hur tester utförs vid en facilitet som har som sin huvudsakliga uppgift att utföra tester. Vid studiebesöket ställdes till att börja med några allmäna frågor (till Mikael Holgerson, chef vid SKF Test Centre Göteborg) kring verksamheten varefter en rundtur i labbyggnaden gjordes.

De två styrningsmetoderna som rekommenderades av Mikael Holgerson var LabVIEW och PLC varför det efter detta besök beslutades att just PLC och LabVIEW skulle studeras mer ingående än de andra.

4.2.1 Referenstest – grund för jämförelse av lösningar

Uppkopplingen som vanligen används för ställdonet vid standardtestet (fram- och återgående rörelse) beskrivs av följande schema[6] (figur 4.1);



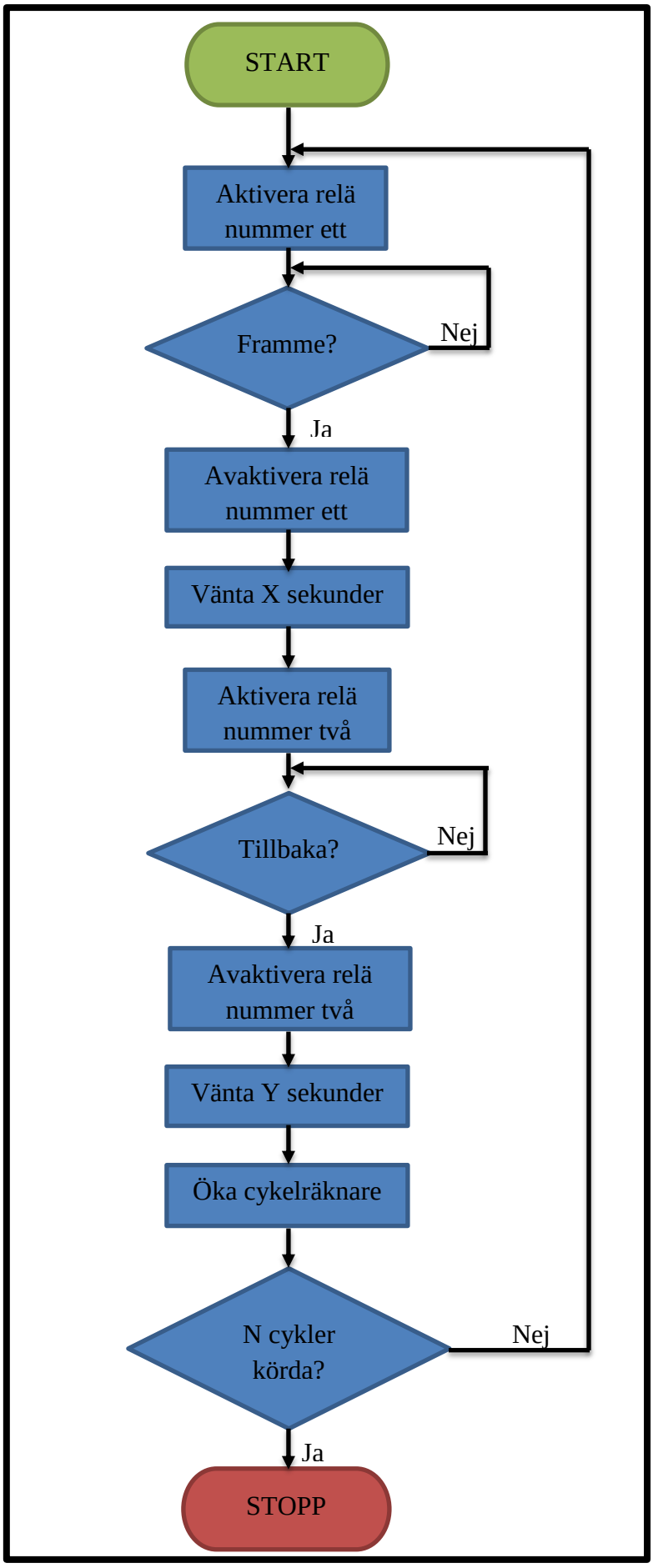
Figur 4.1. Uppkopplingsschema av ställdon. Det rödmarkerade är sådant som styrs direkt av ett tänkt styrsystem, det blåmarkerade är inmonterat i ställdonet.

Vid normal drift körs motorn då någon av knapparna (till vänster) aktiveras så länge som motsvarande ändlägesbrytare inte är aktiverad. Dessa ändlägesbrytare används inte vid test utan ändlägesgivare fästes på testriggen istället (det är kanske inte alltid önskvärt att ha en direkt brytande funktion vid ändlägena, exempelvis vill man ibland köra till det mekaniska ändläget).

Vid autonom testdrift genereras signalerna till motorstyrningsreläerna i ett program; programmet ettställer eller nollställer utgångar beroende på vilken typ av rörelse som skall utföras. Programmen, som togs fram i jämförelsesyfte i respektive programmeringsmiljö, är baserade på kopplingsschemat ovan vad gäller konfiguration av ut- och ingångar.

[6] Datablad, www.skf.com/medialibrary/asset/0901d1968009e1db, acc. 2013-04-16.

Testprogrammet som användes som referens vid jämförelse mellan de olika styrningssätten finns beskrivet i flödesschemat på nästa sida (*Figur 4.2*). Testförfarandet är den standardiserade ”fram-och-tillbaka”-cykeln som görs i de flesta av testen.



Figur 4.2. Flödesschema som beskriver hur programmen i de olika miljöerna är uppbyggda. Den huvudsakliga funktion som programmen uppfyller kan också utläsas ur detta schema.

Till att börja med skapades det tidigare beskrivna programmet i de olika miljöerna. I LabVIEW skapades dessutom ett till program med samma funktion och en annan uppbyggnad då LabVIEW var en av de två lösningar som rekommenderats av Mikael Holgerson och det redan fanns fler programexempel ur PLC-miljön från den Siemens LOGO! som sedan tidigare fanns på företaget.

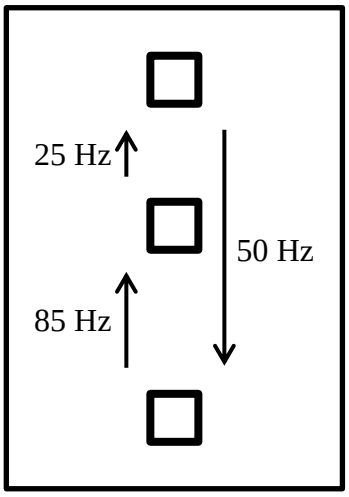
Författaren ansåg att det var alldeles för svårt att konfigurera in- och utgångar samt att simulera knapptryckningar i Simulink (MATLAB), och att det var av mindre värde för projektet att denna kunskap inhämtades. Därför gjordes inte programmet fullständigt klart i den miljön. Det resonerades dessutom att LabVIEW egentligen gjorde samma sak fast på ett bättre sätt än vad MATLAB gjorde; MATLAB kanske var mer användbart då avancerade beräkningar skall göras, men betydligt mindre användbart vid styrning och mätning.

Ett enklare program skapades i C som ett försök att efterlikna hur ett program hade sett ut i en MCU, men författaren ansåg att en MCU inte skulle ha den modifierbarhet och användarvänlighet som efterfrågades varför MCU inte studerades vidare.

Det mer invecklade programmet skapades därefter endast i LabVIEW för att jämföras med det program som fanns sedan tidigare för att styra det test som finns beskrivet i kapitel 5.1.3. LabVIEW-programmet finns också beskrivet i resultatkapitlet. Det som skilde detta program från det enklare var att en utgång skulle aktiveras och sedan avaktiveras var tredje cykel, samt att programmet även innefattade sektioner som hanterade manuell styrning av donet som testades (se paragraf 5.2.2.2 i kapitel 5.2.2).

4.3 Skapande av testmodell

De två testförfaranden som systemet byggdes för var väldigt specifika. Det ena testförfarandet var styrning av en växelströmsmotor kopplad till ett ställdon (*Figur 4.4*, nästa sida), där tre lägesgivare kopplats på (*Figur 4.3*, till höger). Mellan den innersta (nedersta) givaren och den i mitten körs motorn vid maximal frekvens (85 Hz), mellan den i mitten och den yttersta givaren körs motorn vid en låg frekvens (25 Hz), och på vägen tillbaka körs motorn med frekvensen 50 Hz. All denna körning görs per automatik där frekvensen bestäms av programmet i PLC:n. Vid manuell drift körs motorn vid frekvensen 50 Hz. Minimal last lades på oket.



Figur 4.3. Beskrivning av lägesgivarnas positioner samt vilken frekvens AC-motorn körs vid mellan positionerna.

Det andra testet som kopplingar gjordes för var det som beskrevs i kapitel 5.1.4 – ett test av effektförluster där lasten egentligen skulle ha ökats undan för undan (*Figur 4.5*, nästa sida). Istället löstes detta direkt i PLC-programmet med en regulator för reglering av temperaturen genom styrning av intermittensen.

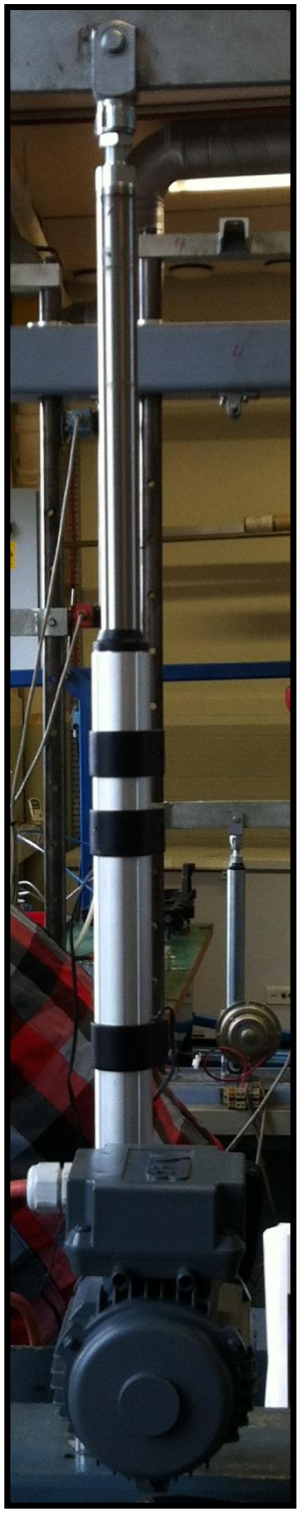
Figur 4.5. Ställdonet med BLDC-motor och två ändlägen (givare under kåpa).

Innan något programmeringsarbete kunde påbörjas var det dock först nödvändigt att välja vilken modell av Siemens LOGO! som skulle användas. Siemens broschyr för LOGO![7] studerades ingående för att rätt modell skulle kunna väljas baserat på testbehov. Parametrar som togs i beaktande var exempelvis matningsspänning, antal ut- och ingångar, maximalt antal funktionsblock per program, huruvida utgångarna var relä- eller transistorutgångar, huruvida modulen hade möjlighet till anslutning via ethernet, med mera.

Den modell av Siemens LOGO! som till slut valdes var 12/24 RCE, med påkopplade moduler AM2 AQ, AM2 RTD samt DM16 24 och en tillhörande display LOGO! TD för att användare skall slippa öppna kopplingsskåpet för att se värden på den skärm som finns på Siemens LOGO! 12/24 RCE.

För strömmätning valdes strömtransformatorer (nominell uppmätt ström 5 A) eftersom de var galvaniskt isolerade mellan den uppmätta strömmen och den utskickade signalen. Den uppmätta strömmen var i storleksordningen 7 A vilket vore osäkert att leda genom en mätingång i LOGO!:n. Dessutom var strömtransformatorerna små och billiga vilket var det säkrare valet med den kunskapsnivå som förelåg. Strömtransformatorerna med tillhörande komponenter monterades på ett experimentkort för montering på DIN-skena.

Det vore möjligt att seriekoppla ett shunt-motstånd med motorn och att på så sätt mäta strömmen men eftersom strömförbrukningen i motorerna är så hög vid den låga spänning som föreligger skulle ett shunt-motstånd medföra ett alldeles för stort spänningsfall innan motorn. Dessutom skulle spänningen behöva förstärkas eftersom LOGO! tar in en signal mellan 0 och 10 V vilket



Figur 4.4. Ställdonet med AC-motor där frekvensen varieras beroende på de tre positionsgivarna (svarta).

[7] Försäljningsbroschyr LOGO!, www1.elfa.se/data1/wwwroot/assets/datasheets/LOGO-V7_eng_bro.pdf, acc. 2013-05-22.

vore ett oacceptabelt spänningsfall att ha innan en 24 V-matad motor. Av den anledningen studerades inte lösningen med shunt-motstånd vidare.

På grund av ett problem med strömtransformatorernas utsignal vid mätning av växelström lades ytterligare en koppling till för att signalen skulle kunna läsas av PLC:ns mätingång.

PLC:n kopplades upp tillsammans med resten av skåpet, vilket gjordes i samarbete med det projekt som löpt parallellt med detta som behandlar motorstyrningsreläer. Skåpet kom slutligen att innehålla en frekvensomriktare, ett motorkort för BLDC-styrning, ett säkerhetsrelä, några säkringar, ett nätaggregat samt ovan nämnda experimentkort, utöver PLC:n. I skåpets dörr monterades ett antal knappar för manuell styrning, ett nödstopp kopplat till säkerhetsreläet, samt LOGO!-displayen.

BLDC-testet (verkningsgradstestet, med två ändlägesgivare) kördes med 400 kg lastat på oket. Till en början användes ändlägesgivare som var standard för ställdonet, men eftersom PLC:ns programcykel var så lång (runt 0,2 s) missades ibland signalen vid det nedre ändläget. Den nedre ändlägesgivaren byttes därför mot en induktiv givare, och därmed löstes problemet.

5 RESULTAT OCH ANALYS

Detta kapitel innehåller precis som föregående kapitel delar som rör kartläggningen av dagens testförfaranden och behov, jämförelse mellan lösningar, samt det slutliga utseendet för testmodellen.

Att kapitlet är så långt motiveras av att större delen av detta projekt bestått av studier av olika lösningsförslag (programmeringsmiljöer), samt att programmen som tagits fram för att dessa skall kunna jämföras finns beskrivna i både text och bild.

5.1 Kartläggning av nuvarande testförfaranden och framtida behov

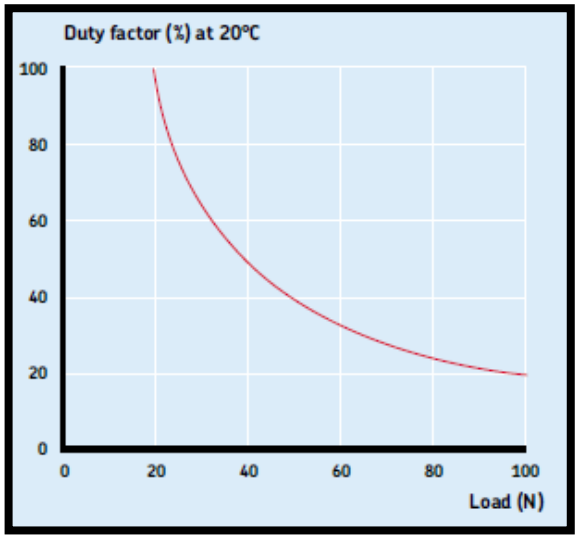
Följande underkapitel beskriver vad som framkommit ur den inledande kartläggning som gjorts. Texten beskriver resultatet av intervjuerna, läsningen av testdokumentation, samt ger två mer detaljerade exempel på test som utförs. Slutligen innehåller underkapitlet en kravspecifikation vilken ställts upp som en sammanfattning av kartläggningen samt för att användas vid senare jämförelser.

5.1.1 Intervjuer

Ur de två första intervjuerna, med Pär Högberg och Kjell Hermansson, framkom redan tidigt tillräcklig information för att informationssökandet kring lösningsförslag skulle kunna börja. Då det var dessa två personer som använder testsystemet mest ställdes färre frågor till de resterande intervjuade om hur ett framtida testystem borde se ut.

5.1.1.1 Livslängds- och intermittenstest

Det framkom ur dessa två intervjuer att det är vanligast att livslängd testas (hos till exempel tätningar eller donet självt), och då körs donet vid maximal prestanda fram och åter till dess att donet antingen går sönder eller att man uppnått ett krav på ett visst antal cykler. Ett annat vanligt test är intermittenstest, som visar hur ofta ett don kan köras utan att det blir för varmt (i produktkatalogen återfinns last-intermittensdiagram som visar hur mycket last som kan läggas på vid en viss intermittens utan att donet överhettas, se ***Figur 5.1***[8]). Är intermittensen exempelvis 25 % skall donet köras 25 % av tiden och stå i vila 75 % av tiden.



Figur 5.1. Exempel på ett last-intermittensdiagram. Källa: www.skf.com

I vilken del av cykeln vilotiden placeras bestäms av huruvida skruven i donet är självhämmande eller inte; eftersom donet körs vertikalt med ett ok längst upp och en last ligger på oket behöver motorn fortfarande arbeta i plusläget då skruven inte är

[8] Datablad, www.skf.com/medialibrary/asset/0901d1968009e1db, acc. 2013-05-14.

självhämmande. Vilotiden placeras i det fallet helt i minusläget. Är skruven självhämmande kan vilotiden placeras arbiträrt[9].

När ett ”standardtest” utförs görs det antingen för att parametrarna ovan skall tas fram, eller för att kontrollera ifall produkten faktiskt uppfyller parametrarna som angivits i katalogen. Detta görs huvudsakligen genom att translationshastigheten uppmäts med hjälp av stoppur samt att temperatur, last och intermittens antecknas och sedan jämförs med katalogvärdet för hastighet respektive den kurva som beskrivs av ***Figur 5.1***.

5.1.1.2 Klimattest

Ur en annan intervju framkom att det utöver dessa två vanligare testerna även görs klimattester, det vill säga att ett ställdon får arbeta i någon extrem miljö. Bland dessa är de fyra vanligaste typerna följande;

1 Hög omgivningstemperatur.
2 Låg omgivningstemperatur.
3 Vibrationsmiljö.
4 Hög luftfuktighet.

5.1.1.3 Prestandatest

Det görs även prestandatest av donen, det vill säga att ställdonet körs vid en viss last och med en viss intermittens och att ström- och temperaturvärden noteras då dessa stabiliserat sig vid aktuella driftsbetingelser. Värdena kan sedan studeras för att se om de ändras mer vid vissa betingelser som kanske kan vara svåra för donet att hantera. Datan från dessa test visas i diagram i produktdatablad och -kataloger.

5.1.1.4 Förstörande test

Till sist görs ibland även förstörande tester, vilka syftar till att kartlägga ramarna för produktens prestanda. Uppkommer då en förfrågning från kundsidan om huruvida donet kan lastas med exempelvis ett ton tryckande last kan företaget svara med att visa upp ett test där det visat sig att 800 kg tryckande last gjorde så att donet havererade.

Några av de ovanstående testerna kräver ibland att donet körs till det så kallade mekaniska ändläget, det vill säga då donet inte *kan* gå längre – detta innebär att motorströmmen skjuter i höjden vilket är att tänka på vid styrning av ett sådant test. Om en strömgräns ställs in med tanke på normal drift kommer denna att överskridas då donet når ett av de mekaniska ändlägena.

Ett test pågår vanligtvis i en vecka upp till ett par månader, men i ett fåtal fall pågår det till och med i några år. Under ett test mäts framför allt två parametrar: cykelantal och strömförbrukning, och ibland mäts även temperatur (vilket vore önskvärt att göra med ditsatta Pt100-givare[10]). Det som också mäts är translationshastighet hos ställdonet – detta görs idag

[9] Intervju av Kjell Hermansson, 2013-03-22.
[10] Intervju av Pär Högberg, 2013-03-18.

genom att slaglängden är känd och att tiden för ett slag mäts manuellt med stoppur. All data kontrolleras och antecknas för hand ungefär en gång om dagen – även datan från testen som drivs med hjälp av en PLC – och denna kontroll och nedskrivning av data behövs göras även på helgdagar. Den begränsade övervakningen gör att det inte är så tydligt vilka förhållanden som rådde då ett eventuellt haveri skedde, och idealt vore egentligen om det gick att stoppa testet två sekunder innan haveri för att se exakt vid vilka betingelser som haveriet sker[11].

Ett problem som ofta stöts på är att det uppkommer begränsningar i vad som kan göras; det är vanligt att ett test ställs upp i förväg men att det sedan ändras när det skall sättas ihop, bara för att det visade sig praktiskt svårt att göra[12]. Ett annat vanligt problem är att ändlägesgivare flyttas, överhettas (detta sker för det mesta mellan 60000 och 70000 körcykler) eller går sönder vilket leder till att start- och ändpositionen (och därmed slaglängden) för donet ändras med tiden[13]. Då detta händer görs inte testet om utan givaren flyttas bara tillbaka varpå testet återupptas, men det säger sig självt att testdatan blir något mindre tillförlitlig då detta händer.

I dagsläget används reläer med paustid för att styra testerna – tiden ställs in med binär kod, som måste ställas in med skruvmejsel och lathund för översättning av decimala till binära tal. Dessutom används en PLC som nyligen införskaffats, vilken används för de mer invecklade testerna. I de fall då PLC anses något överflödig eller då den är upptagen med andra test görs en handgjord koppling med seriekopplade tidsreläer som inte är arbetsplatssäkerhetstekniskt tillåten. Detta behövs i ungefär vart femte test[13].

Ett av testen som styrs med PLC (Siemens LOGO!) har induktiva givare istället för mekaniska ändlägesgivare och det har gått felfritt och utan stopp ”hur länge som helst”[13], vilket visar hur nyttigt det hade varit att helt enkelt byta de mekaniska ändlägesgivarna mot deras induktiva motsvarigheter. Ur de mekaniska givarnas otillförlitlighet uppkommer ett problem i att testet kanske avbryts precis då alla gått hem för dagen – testet står då stilla i sexton timmar vilket är tillräckligt för att temperaturen i donet skall sjunka på ett högst märkbart sätt. Det är trots allt störst sannolikhet att testet skall stoppas på någon av de två tredjedelar av dygnet som ingen är där, och än mer sannolikt blir det eftersom det uppkommer spänningsvariationer i strömförsörjningen då resten av produktionsfaciliteten stängs ned för dagen[11].

Ett annat stort problem är att testerna inte egentligen motsvarar den verklighet som donen utsätts för. Det är ytterst sällan som ett don körs fram och tillbaka mellan två fixa ändlägen med exakt samma last och utan någon radiell påverkan på axeln. Egentligen finns kanske en friktionslast som varierar med translationshastigheten eller en last som ändras beroende på hur långt ut donet har körts etcetera[11].

Ytterligare ett problem (som ju inte kommer att lösas i detta projekt) är att reläerna för motorstyrning överhettas efter endast 10000 cykler och det är vanligt att kunder kör många fler cykler än så. Av den anledningen skall halvledarreläer tas fram i ett annat projekt, parallellt med detta.

[11] Intervju av Jan Lorentzon, 2013-03-26.
[12] Intervju av Pär Högberg, 2013-03-18.
[13] Intervju av Kjell Hermansson, 2013-03-18.

Det huvudsakliga önskemålet som fanns var att ett nytt system skulle vara så självgående som möjligt, ha en hög tillförlitlighet, och vara lättanvänt – som det är idag måste någon gå in och kontrollera så att ingenting gått sönder på helgerna och det vore önskvärt om detta behov försvann[14]. Dessutom fanns önskemål om att det skulle vara enkelt att översätta något som ska testas till ett färdigt program[15], samt ett önskemål om att det skulle vara lätt att se nuvarande ström- temperatur- och cykelvärden. Vidare fanns önskemål om att kunna ställa in en maximal ström och temperatur vid vilka testet stoppas automatiskt[14], och att last skulle kunna mätas automatiskt med någon form av givare. Det vore också bra om det gick att testa en del åt gången (växel, motor, skruv etc.) för att se vari den största förlusten ligger och således var det finns störst förbättringspotential. Till sist vore det önskvärt om det gick att ha varierande last, då konstant last för det mesta inte motsvarar den last som vanligtvis läggs på donen. [16]

Resultatet av två av intervjuerna beskrivs i kapitel 5.1.3 respektive 5.1.4 då formatet skiljer sig något från de tidigare beskrivna intervjusvaren.

5.1.2 Läsning av testdokumentation

Det tydligaste som framkom ur läsningen var hur sällan data loggas – ofta läses data endast av vid X antal cykler (X = 100, 1000, 10000, 50000, 100000, 150000, 200000, …), om testet pågår så många cykler[17]. För det mesta läses data inte heller av vid just den cykeln som angetts då det är svårt för den som utför testet att ständigt ha kontroll på cykelräknare[14,15].

Resultatet av läsningen är egentligen bara en stor mängd exempel på testuppställningar, varav två finns beskrivna i detta kapitel (5.1.3 och 5.1.4).

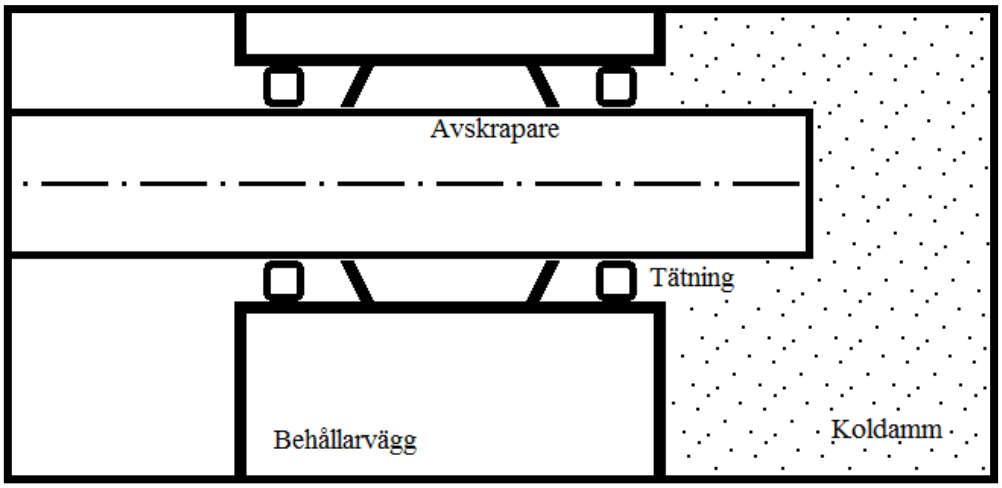
5.1.3 Exempel på komplicerat test som liknar applikation hos kund

Det är ur en företagssynpunkt eftersträvansvärt att i ett test försöka efterlikna kundens applikation så mycket som möjligt eftersom testet då ger så relevant data som möjligt. Det är där behovet av mer komplicerade test kommer in – kunden kör för det mesta inte donet fram och åter med samma last hela tiden och i exakt den miljö som testlabbet har.

5.1.3.1 Beskrivning av test

Ett exempel på ett sådant komplicerat test som utförs är ett test av tätningar och material/beläggningar som idag utförs med hjälp av en PLC, då körcykeln som föreligger skulle varit onödigt omständlig att göra med seriekopplade reläer. Testet går till på så sätt att ett ställdon körs ut ur och in i en tät behållare, och vid kontaktpunkten mellan behållaren och ställdonets kolv sitter tätningar – det är dessa som testas. I behållaren finns koldamm, något som gör att insidan av behållaren liknar just den miljö som kunden kör produkten i. Koldammet svävar inte runt i behållaren av sig själv – yttre stimulans krävs nämligen i form av att dammet blåses runt var tredje körcykel[18]. Nedan följer en enkel bild av hur kontaktpunkten mellan behållaren och ställdonets axel ser ut (***Figur 5.2***).

[14] Intervju av Kjell Hermansson, 2013-03-18.
[15] Intervju av Pär Högberg, 2013-03-18.
[16] Intervju av Jan Lorentzon, 2013-03-26.
[17] Testdokumentation, test order no 2013LT0001.
[18] Intervju av Peter Hansson, 2013-03-27.



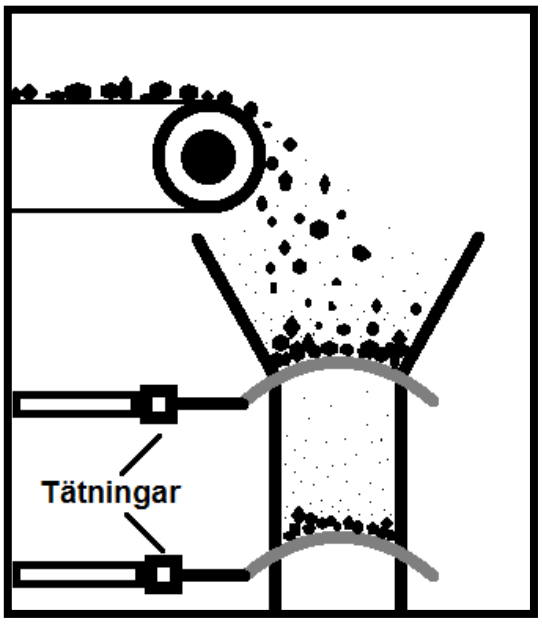
Figur 5.2. Schematisk bild av testuppställning.

Även om temperaturen mäts till en början är den huvudsakliga delen av testet slitagetestning vilket innebär att väldigt många cykler måste köras för att någon vettig information rörande slitage skall komma ur testet. Just temperaturen är den enda storheten som mäts i någon enhet som skulle kunna mätas med givare – slitaget på tätningar och beläggning antecknas för hand genom att det noteras efter hur lång tid en tätning behövt bytas. Temperaturen mäts till en början då den kan ge en indikation på friktion (cykeltiden för detta test är så lång att nästan ingen temperaturökning kommer ur intermittens) och denna mäts idag för hand[19].

Ett försök att ta bort smuts gjordes i detta test genom att luft blåstes in med högt tryck i kammaren mellan avskraparna, för att partiklar skulle blåsas bort från utsidan. Luften blåstes in var fjärde cykel, och eftersom testet redan styrdes med PLC var det enklast att sköta styrningen av denna operation genom den också. Tyvärr visade det sig att det inte hjälpte att blåsa in luft på det viset – all tryckluft tog sig ut genom samma punkt på avskraparen och lämnade resten av avskraparen lika täckt av koldamm som innan[19].

5.1.3.2 Beskrivning av egentlig användning

Som en jämförelse med hur testet utförs kan själva användningen som testet försöker simulera beskrivas. Testet är som sagt ett test av tätningar, och tätningarna är tänkta att användas till ställdon som styr en sluss enligt *Figur 5.3*. Locken (grå i bild) öppnas växelvis för att rätt mängd kol skall komma vidare. Då slussen är igång frigörs koldamm från kolet som skall slussas vidare, vilket är varför tätningarna behövs – kommer koldamm in i ställdonet medför detta såklart märkbart slitage på komponenterna[19].



Figur 5.3. Enkel schematisk bild av vad testet försöker efterlikna.

[19] Intervju av Peter Hansson, 2013-03-27.

5.1.4 Exempel på test av effektförluster

Ytterligare ett exempel på ett test följer nedan (testet har i skrivande stund ännu inte utförts, men är i planeringsstadiet). Testet i fråga syftar till att undersöka effektförlusten hos en växel som används i ett av ställdonen då den drivs med en annan motor. Anledningen till att detta test utförs är att det anses vara intressant att se hur en av företagets standardprodukter (CAT32B-vxl2 mässing-300) interagerar med en kundspecial-motor 0451960 ”Buster” som just varit med i ett stort projekt på företaget[20,21]. Det är även tänkbart att ställdon med lägre effektförluster kan komma att ingå i SKF:s så kallade Beyond Zero-familj, där produkter erbjuds som hjälper användare att minska påverkan på miljön[22].

Testet görs genom att donet körs ett antal slag (cykler) med en trycklast på 400 kg och en ”50/50-intermittens” till dess att donets temperatur stabiliserat sig (50/50 innebär att motorn körs hälften av cykeltiden och får kylas den andra hälften). Donet körs med en slaglängd på 250 mm där varje slag skall ta cirka 12 s vilket innebär en ungefärlig hastighet på 21 mm/s. Motorhastigheten regleras med hjälp av Hallsensorer på ett motorkort[20,23].

Stabiliserar sig temperaturen under 65 °C ökas lasten i steg om tjugo kg till dess att lasten är 500 kg eller att temperaturen stabiliserat sig vid 65 °C (vilket som än händer först). Stabiliseras temperaturen under 65 °C vid lasten 500 kg ökas istället intermittensen i steg (60/40, 70/30, osv.). När temperaturen stabiliserat sig vid respektive last- eller intermittenssteg noteras aktuell last och intermittens samt motortemperatur i ett protokoll[20].

Idealt vore egentligen om det gick att ha reglering av motortemperaturen genom att öka eller sänka intermittensen för att motorn skall stabilisera sig kring 65 °C istället för att lasten behöver ökas manuellt. Dock är detta svårt att göra då processen har ett så långt insvängningsförlopp med en högst noterbar tidskonstant[21].

5.1.5 Kravspecifikation

Som en sammanställning av allt det som kartlagts sattes en kravspecifikation ihop, och efter att den lämnats över till Kjell Hermansson och Pär Högberg såg den ut enligt *Tabell 5.1*.

Kravspecifikation: ny styrutrustning för testriggar		
	Krav / önskemål	**Viktning**
Typ av test		
Livslängdstest	K	4,5
Prestandatest	K	4
Intermittenstest	K	4,5
Förstörande test	K	4
Klimattest	Ö	0,5
Mätning av storheter		

[20] Testdokumentation, test order no 2013LT0004.
[21] Intervju av Pär Högberg, 2013-03-28.
[22] www.beyondzero.com, acc. 2013-06-03.
[23] Datablad, www.electromen.com/pdf/EN_em-206-48.pdf, acc. 2013-03-28.

Temperatur – motor	Ö	3,5
Temperatur – frys	Ö	1
Strömstyrka	K	5
Antal cykler	K	5
Translationshastighet	Ö	3,5
Last	K	4,5
Övriga funktioner		
Ställbar maxström	K	4
Ställbar maxtemperatur	Ö	3,5
Ställbar frystemperatur	Ö	3
Stopp vid andra händelser	Ö	4
Manuellt stopp	Ö	3,5
Ställbar väntetid vid ändlägen	K	4,5
Möjlighet att köra enkla testcykler	K	3
Möjlighet att köra invecklade testcykler	Ö	4
Funktioner utanför programmeringsmiljö		
Ställa in maxström	Ö	3,5
Ställa in maxtemperatur	Ö	3,5
Display av nuvarande mätvärden	Ö	3,5
Starta ett test med knapptryck	Ö	4
Stoppa ett test med knapptryck	Ö	4
Ställa in väntetider vid ändlägen	K	4,5
Modifierbarhet – användarvänlighet		
Enkelt att ställa upp ett enkelt test	Ö	4,5
Enkelt att ställa upp ett mer avancerat test	Ö	4
Enkelt att anpassa till nya produkter	K	3,5
Enkelt att starta upp ett test	K	4
Dataloggning		
Automatisk dataloggning	Ö	4,5
Hög samplingsfrekvens	Ö	1,5
Fler än ett sampel per cykel	Ö	3,5
Lättanvänt dataformat	K	3,5
Dynamisk samplingsfrekvens	Ö	3
Tillförlitlighet		
Hög livslängd hos ändlägesgivare	K	4,5
Exakthet hos mätgivare	K	4
Data sparas trots avbrott	K	4,5
Test går att återuppta efter avbrott	K	4,5
Självgående	K	5
Övrigt		
Kostnad	-	5

Tabell 5.1. Kravspecifikation för en ny styrutrustning för testriggarna i testlabbet.

Viktningen är ett medelvärde av vad som angivits av Kjell Hermansson och Pär Högberg, och kolumnen med krav alternativt önskemål (K/Ö) har satts till ’krav’ ifall någon av dem angivit att det var ett krav, och ’önskemål’ endast om båda angivit att det var ett önskemål.

Det kan ses hur viktigt det är att testningen är självgående, något som förmodligen kommer att lösas oavsett hur problemet löses eftersom förslagen huvudsakligen kommer att innefatta någon slags automatisering. Vidare kan ses att det är mycket viktigt att kunna mäta strömstyrka och att räkna antalet cykler – båda dessa torde vara tämligen vanliga problem och alla lösningsförslag bör alltså med enkelhet kunna hantera dessa.

Kravspecifikationen användes sedan inte bara som stöd vid analys av olika lösningsförslag (kapitel 5.2) men även som bas för Pugh-matrisen (*Tabell 5.2* på sidan 45) vilken gjordes som en del av analysen av lösningar.

5.2 Framtagande av förslag till ny testmiljö

Detta delkapitel beskriver till största del programmen som tagits fram för jämförelser sinsemellan, med undantag för MATLAB- och MCU-programmen som ansågs för ofullständiga för att jämförelser med dem skulle ge rättvisande information. Kapitlet jämför också de olika programmen och i den sista delen presenteras en Pugh-matris där de olika lösningsförslagen ställs mot varandra. Till sist beskrivs även studiebesöket som gjordes till SKF Test Centre Göteborg.

Kapitlet har mycket ingående beskrivningar av program och bilder på program som tagits fram, vilket kanske inte är av intresse för alla läsare. I sådana fall rekommenderas läsaren att hoppa över läsning av kapitel 5.2.2, 5.2.3 samt 5.2.4 och gå direkt till kapitlet som beskriver jämförelsen av de olika lösningarna.

5.2.1 Studiebesök Test Centre Göteborg

Nedan följer en kort beskrivning av testverksamheten vid SKF Test Centre. Vid labbet görs vibrations- och temperaturmätningar på rullningslager utsatta för en variabel last; denna kan ändras manuellt genom påläggning av vikter, vilket ökar trycket i ett hydraulsystem som i sin tur belastar testlagren. Dynamisk testning görs med en vibrerande testrigg – körs med 75 Hz frekvens och används mest för att testa hållare. Det görs även lagertest som utförs vid höga omgivningstemperaturer, och friktionstester där lagrets ytterring bärs upp av ett tunt lager oljefilm i ett hydrodynamiskt glidlager (kan anses friktionsfritt) vilket gör att friktionen i lagret vid ett känt drivmoment kan mätas[24].

Condition monitoring (tillståndsövervakning) används också i testverksamheten, möjliggjord av ytterst noggranna tidigare mätningar av hur produkten beter sig under normal drift samt då haveri är nära förestående. De hade gjort egna moduler för vibrationsmätning som sedan interfaceade med LabVIEW:s I/O-moduler för att vibrationerna skulle kunna användas för den ovan nämnda tillståndsövervakningen[24].

[24] Intervju av Mikael Holgerson, 2013-04-09.

All mätning i testlabben sker med LabVIEW, samt även en del styrning – ett test med varierande last styrs till exempel med LabVIEW. Ett test görs även med LabVIEW som ensam styrenhet, delvis för att se hur bra LabVIEW faktiskt är på styrning (går detta test bra skall all styrning bytas från dagens PLC-styrning till helt LabVIEW-styrt)[25].

5.2.2 PLC – Siemens LOGO!

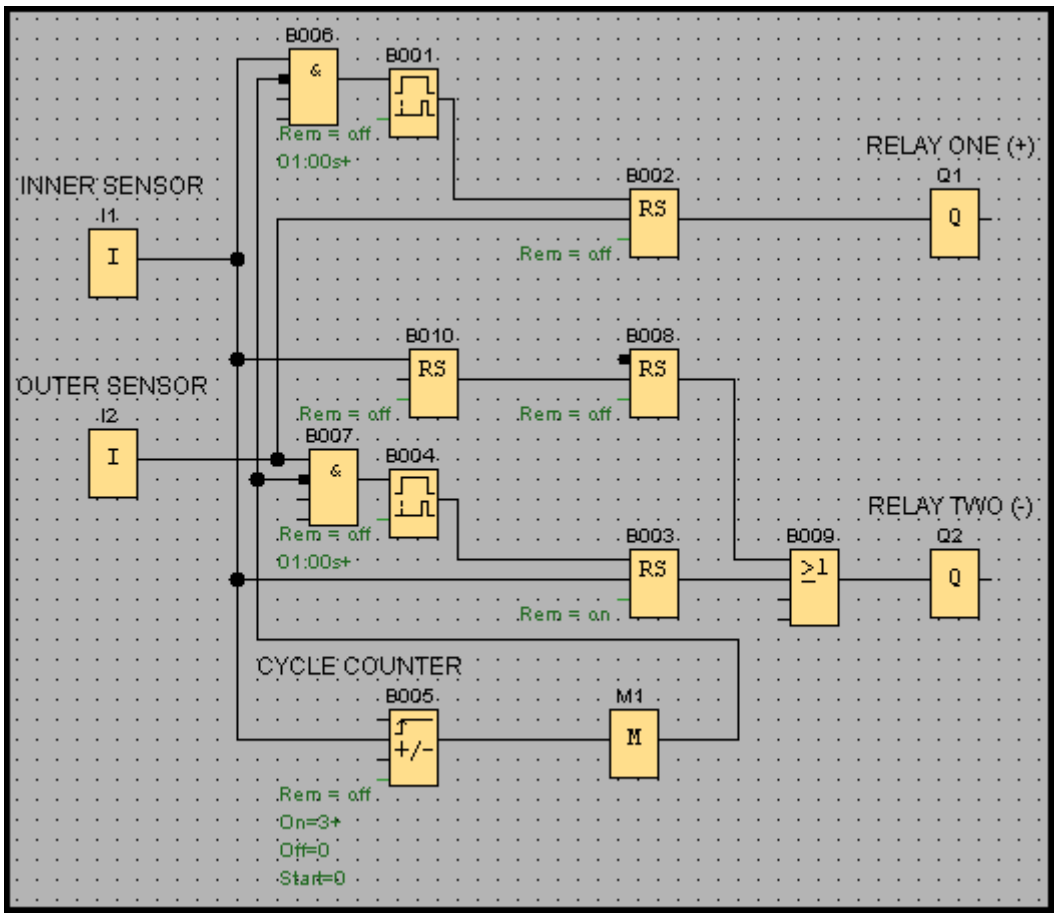
Då denna PLC redan fanns i testlabbet hos SKF Actuation Systems och några test redan styrts med den var det en självklarhet att den skulle jämföras med de andra. Utvärderingen gjordes med demonstrationsprogramvara till LOGO!Soft Comfort (programmeringsprogramvara till Siemens LOGO!) som hämtades från Siemens hemsida – den enda inskränkning i funktion som uppstod ur detta val av programvara jämfört med den fulla versionen var att färdiga (eller, för den delen, ofärdiga) program inte kunde föras över till PLC:n. Alltså gjorde det ingen skillnad vid jämförelse mellan program.

Siemens LOGO! stödjer ISO-programmeringsspråken FBD och LAD, samt ett programmeringsspråk de själv lagt in för att användaren skall kunna göra återkommande sektioner lätthanterliga i form av mindre funktionsblock – UDF (*User Defined Function*).

[25] Intervju av Mikael Holgerson, 2013-04-09.

5.2.2.1 FBD – enkelt program

Ett enklare program enligt flödesschemat i *Figur 4.2* (sid. 16) återskapades i ett FBD (LAD ansågs vara överflödigt) enligt nedanstående figur (*Figur 5.4*).



Figur 5.4. FBD-program som skapades för styrning av en enkel fram- och återgående cykel.

Vid analys av bilden är det enklast att börja vid utgångarna (Q1 respektive Q2). Relä två (drift mot inre läge) aktiveras i två olika situationer; den ena (de två seriekopplade RS-vipporna i mitten av bilden) sker bara vid start, och vipporna ser till så att donet går till minusläge då det startas – utan vipporna skulle inte programmet kunna avgöra åt vilket håll ställdonet ska vid start. Den andra situationen är då den yttre lägesgivaren varit aktiv i 1,00 sekund, det vill säga att donet är i den andra delen av sin körcykel. Relä ett aktiveras då den inre lägesgivaren varit aktiv i 1,00 sekund. Reläerna avaktiveras då respektive ändläge har nåtts.

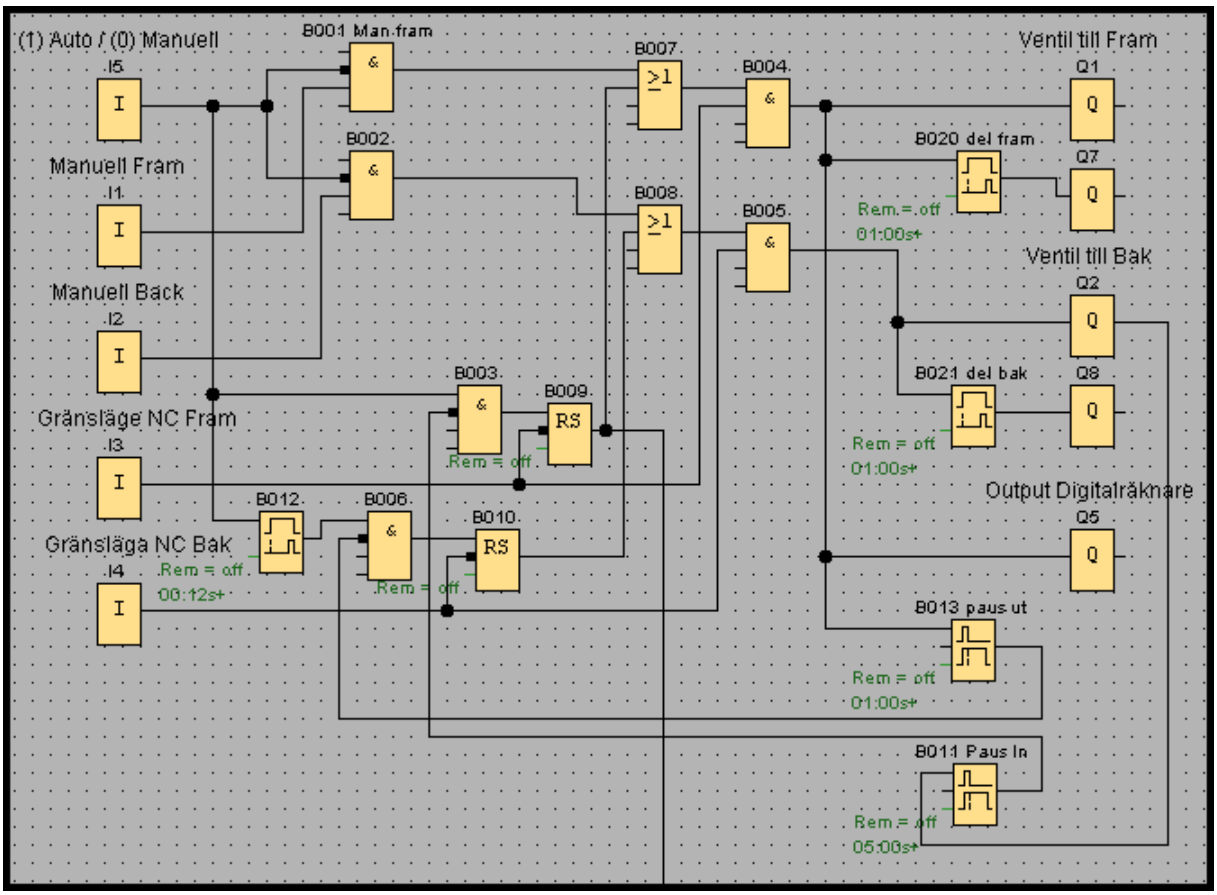
Vid positiv flank hos den inre lägesgivaren räknas cykelräknaren upp, och då tre cykler körts hindrar räknaren ytterligare cykler från att köras. Naturligtvis körs i en verklig situation fler än tre cykler.

Då detta program inte blev särskilt svåröverblickat gjordes ingen del av det i UDF. Däremot gjordes en utvärdering av UDF; det är till synes en användbar editor då mer invecklade program skall skrivas. Ett UDF-program har precis som ett vanligt FBD-program ingångar och utgångar, fast en UDF:s utgångar ansluts alla i ett FBD till andra block eller riktiga

utgångar. Tyvärr har UDF en begränsning på fyra utgångar och åtta ingångar, och kan dessutom inte använda vissa av funktionsblocken som finns tillgängliga i FBD-editorn. Detta omöjliggör fullständig modularisering av FBD-programmering då det är en stor begränsning i programmeringsmiljön att inte kunna använda alla funktionsblock och att behöva hushålla med så få ingångar. Finns ett av funktionsblocken inte tillgängligt i UDF-editorn måste hela UDF-programmet läggas över i huvudprogrammet vilket gör huvudprogrammet mycket svåröverskådligt vid invecklade program.

5.2.2.2 FBD – invecklat program

Det mer invecklade jämförelseprogrammet togs ur en redan existerande PLC-styrning, nämligen styrningen för testet som beskrivs i kapitel 5.1.3. Programmet kan ses i bilden nedan (*Figur 5.5*).

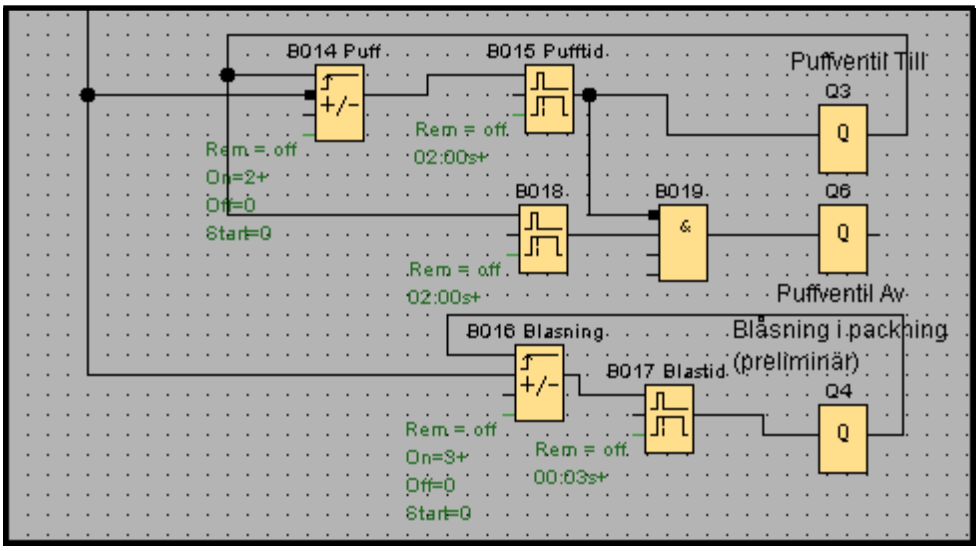


Figur 5.5. Ett mer invecklat FBD-program.

Den huvudsakliga anledningen till att detta program är mer invecklat är att manuell styrning integrerats i programmet. Återigen analyseras programmet från utgång till ingång; Q1 (drift framåt) får endast aktiveras då det främre gränsläget inte nåtts. Har detta gränsläge inte nåtts kan drift framåt ske på två olika sätt – antingen genom att ingången ”Manuell Fram” aktiveras samtidigt som manuell drift är aktiverad, eller genom att programmet märker att det i automatisk drift har väntat tillräckligt länge vid det inre gränsläget (5,00 sekunder). Motsvarande gäller för utgången Q2 (drift bakåt) med tillägget att drift bakåt även aktiveras

då brytaren för automatisk drift slås till. Utgångarna Q7 respektive Q8 sänder ut samma signal som för Q1 och Q2, fast med en fördröjd aktivering på 1,00 sekunder.

Längst ned i diagrammet går en ledning ut till nästa del av programmet (nedan, *Figur 5.6*).



Figur 5.6. Andra delen av det mer invecklade FBD-programmet.

Denna del av programmet hanterar saker som skall ske var tredje respektive var fjärde cykel. Här är det enklare att börja vid ingången; ingången är aktiv då donet är i det yttre läget – inverteringen som sker innan signalen går in i räknaren B014 finns på grund av att lägesgivarna har en brytande funktion. Aktiveras ändlägesgivaren stegas värdet i räknaren B014 upp ett steg, och när tre steg har räknats aktiveras dess utgång vilket leder till att utgången Q3 aktiveras. När Q3 avaktiveras efter 2,00 sekunder (se block B015) aktiveras Q6 i 2,00 sekunder för att ventilen till omblandar-utblåset åter skall stängas.

Det som behandlar utgången Q4 är troligtvis inte helt färdigställt (vilket är rimligt då det inte hjälpte att blåsa ur packningen). Då den yttre lägesgivaren aktiverats stegas räknaren B016 upp och aktiveras ifall fyra steg räknats, vilket gör att utgången Q4 aktiveras i 0,03 sekunder.

Funktionerna som finns beskrivna i *Figur 5.6* är ett bra exempel på var UDF skulle kunna användas. Eftersom UDF kan ha olika in- och utparametrar skulle en funktion som utför något var X:te cykel kunna skapas och användas två gånger i bilden ovan, istället för de funktionblock som används i nuläget. Det skall noteras att programmet ovan skapats med åtanke att det skall vara kompatibelt med en äldre version av LOGO!, som inte stödjer UDF-program.

Slutligen skall även nämnas att den nyare versionen av LOGO! har stöd för loggning av mätvärden eller inre värden. Detta görs genom att ett loggningsblock läggs till; loggningsblocket har en ingång och positiv flank på den ingången triggar en (1) mätning av de parametrar användaren själv valt i loggningsblocket. Värden kan sedan fås ut i en fil med .csv-format som stöds av exempelvis Excel. Detta möjliggör enkel lagring av mätvärden då ett test utförts och resultatet skall sparas undan, och dessutom kan data sparas i ett lättanvänt

format. Vidare möjliggör överföringen av mätdata till en Excel-fil även enkel visualisering av testresultat i form av grafer eller dylikt.

Loggningen är dock något begränsad i och med det faktum att loggen måste föras över från LOGO!:ns internminne eller från det SD-kort som LOGO!:n har plats för, till en PC för slutlig lagring. Trots att automatisk loggning helt eliminerar behovet av att manuellt anteckna testvärden måste loggarna ändå överföras manuellt från PLC:n till någonstans där de skall sparas, så ett visst mått av handpåläggning kommer fortfarande att finnas kvar ifall detta lösningskoncept väljs.

Väljs en LOGO! med ethernetanslutning kan loggarna i och för sig kommas åt via fjärranslutning, vilket skulle göra det något enklare att spara dem (ingen behöver ju gå till labbet för att göra det).

5.2.3 LabVIEW

LabVIEW:s programmeringsmiljö skiljer sig något från de vanligare, textbaserade programmeringsmiljöerna där ett program kan läsas uppifrån och ned. LabVIEW är mer av ett visuellt programmeringsspråk, vilket kommer att tydliggöras ytterligare av bilder i sektionen som följer.

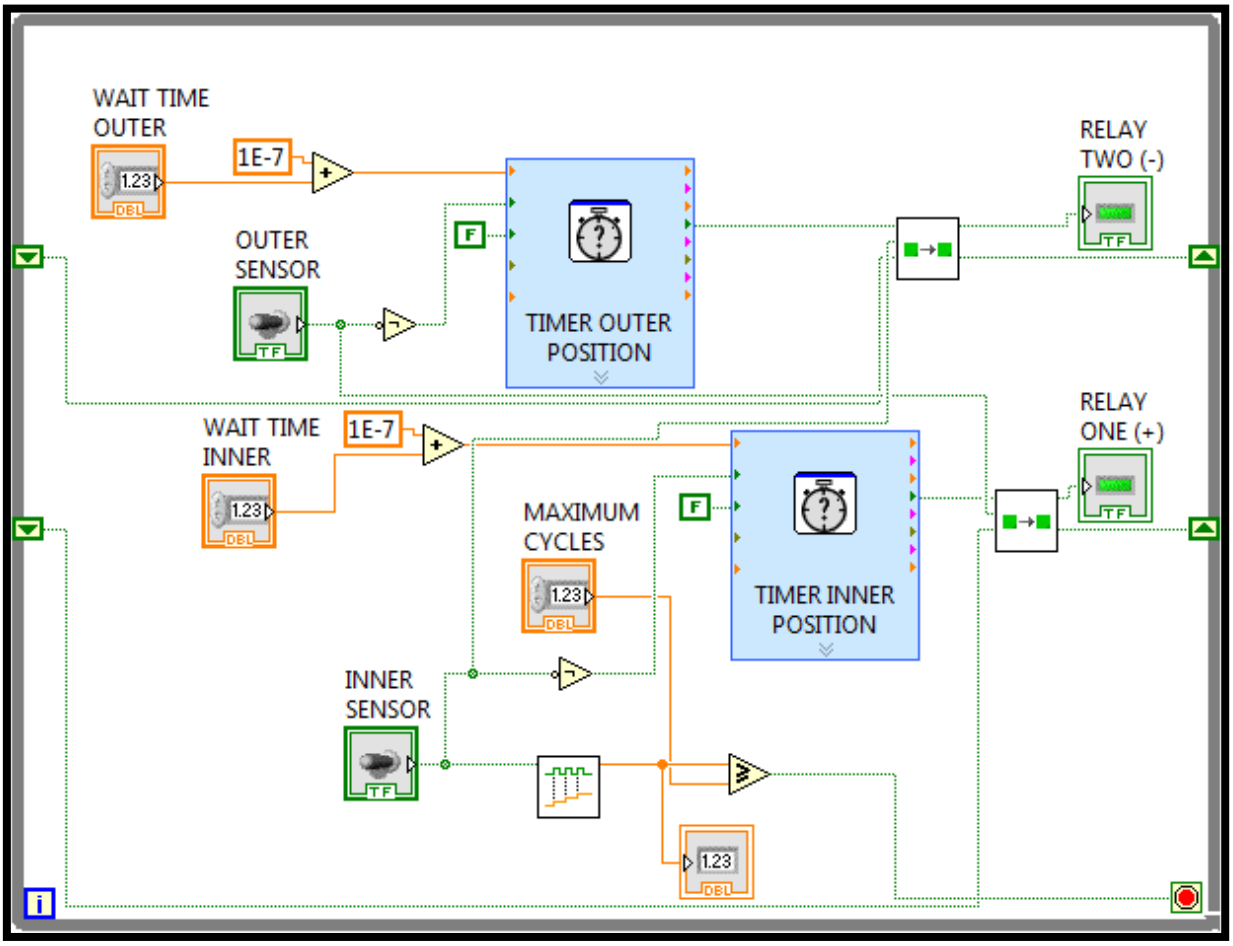
Trots dess olikhet jämfört med textbaserade programmeringsspråk är LabVIEW närmare ett sådant språk än vad PLC-programmeringsspråken FBD och SFC är. Efter att några program skrivits i LabVIEW inses lätt att ett C-program kan översättas direkt till LabVIEW:s språk, trots att utseendet skiljer sig markant.

För att LabVIEW skall kunna användas behövs utöver mjukvaran moduler för I/O som National Instruments själva tillverkar, eller som deras konkurrenter tillverkar (dessa har inte studerats eftersom information om deras existens tillkom sent i projektet). De finns för en stor mängd tillämpningar utöver grundläggande digitala och analoga I/O-moduler – exempelvis finns en modul gjord helt för styrning av BLDC-motorer[26].

[26] sine.ni.com/nips/cds/view/p/lang/sv/nid/210005, acc. 2013-05-20.

5.2.3.1 LabVIEW – enkelt exempel utan tillståndsbaserad logik

Att utseendet skiljer sig markant blir väldigt tydligt då bilden nedan (***Figur 5.7***) studeras.



Figur 5.7. Ett LabVIEW-program utan tillståndsbaserad logik.

Bilden föreställer programfönstret till det program som utför den enklare fram- och återgående körcykeln. Ytterligare ett program för detta skapades i en annan form, och finns beskrivet i kapitel 5.2.3.2.

Färgerna på sladdarna mellan blocken visar vilka typer av värden de överför. En grön sladd överför booleska variabler medan en orange överför variabler av datatypen double float (decimaltal).

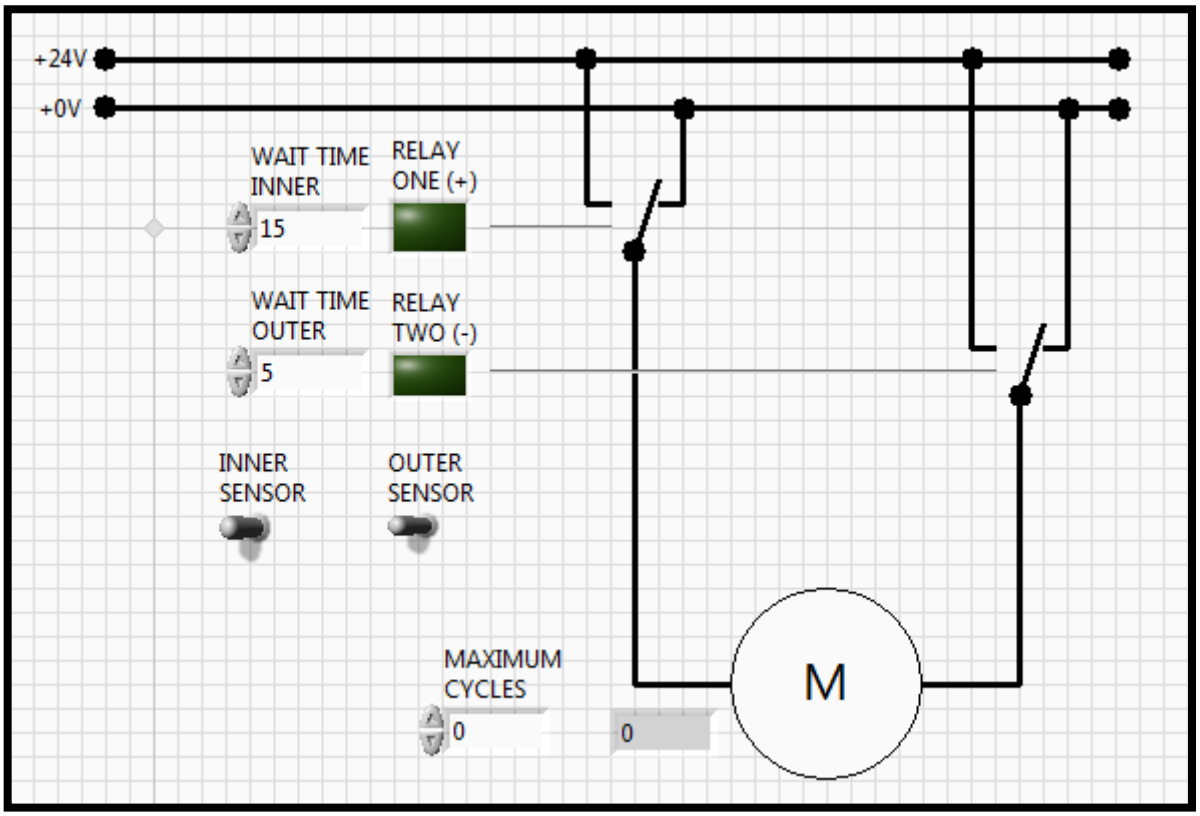
De två blocken långt till höger i figuren är funktioner som skapats för att en boolesk variabel skall sparas en cykel i while-loopen. Den översta ingången är Set-ingången, under den är Reset-ingången och längst ned återfinns en anslutning för förra loopiterationens värde (funktionen har nämligen inget eget minne). Funktionerna är ett försök att återskapa en RS-vippa i LabVIEW-miljön. Som hjälp för att utföra detta används skiftregister (gröna inramade pilar längst till höger och vänster) – de överför ett värde från en loopiteration till nästa.

Blocket i mitten längst ned (detaljer i både orange och grönt) är en räknare, som i det här fallet tar in värdet från den tänkta inre ändlägesgivaren och räknar upp på positiv flank hos det värdet. Den beige komparatorn till höger om funktionen jämför det aktuella räknarvärdet med

det användarinställda (”maximum cycles”) och stoppar programmets while-loop då det maximala cykelantalet uppnåtts.

De två timerblocken tar in ett användarinställt värde (”wait time outer” respektive ”wait time inner”) för hur länge de skall vänta mellan att de får in en signal på de näst översta ingångarna och att de skickar ut en signal på utgångarna till höger på blocken (när detta sker återställer timerblocket sig självt tack vare False-konstanten på ingångssidan). En etta som utsignal från dessa block ettställer respektive RS-vippa och vipporna håller signalen – motor in alternativt motor ut – till dess att motsvarande ändläge nåtts.

Frontpanelen som hör till programmet kan ses nedan (*Figur 5.8*). Samma frontpanel används till programexemplet i kapitel 5.2.3.2.

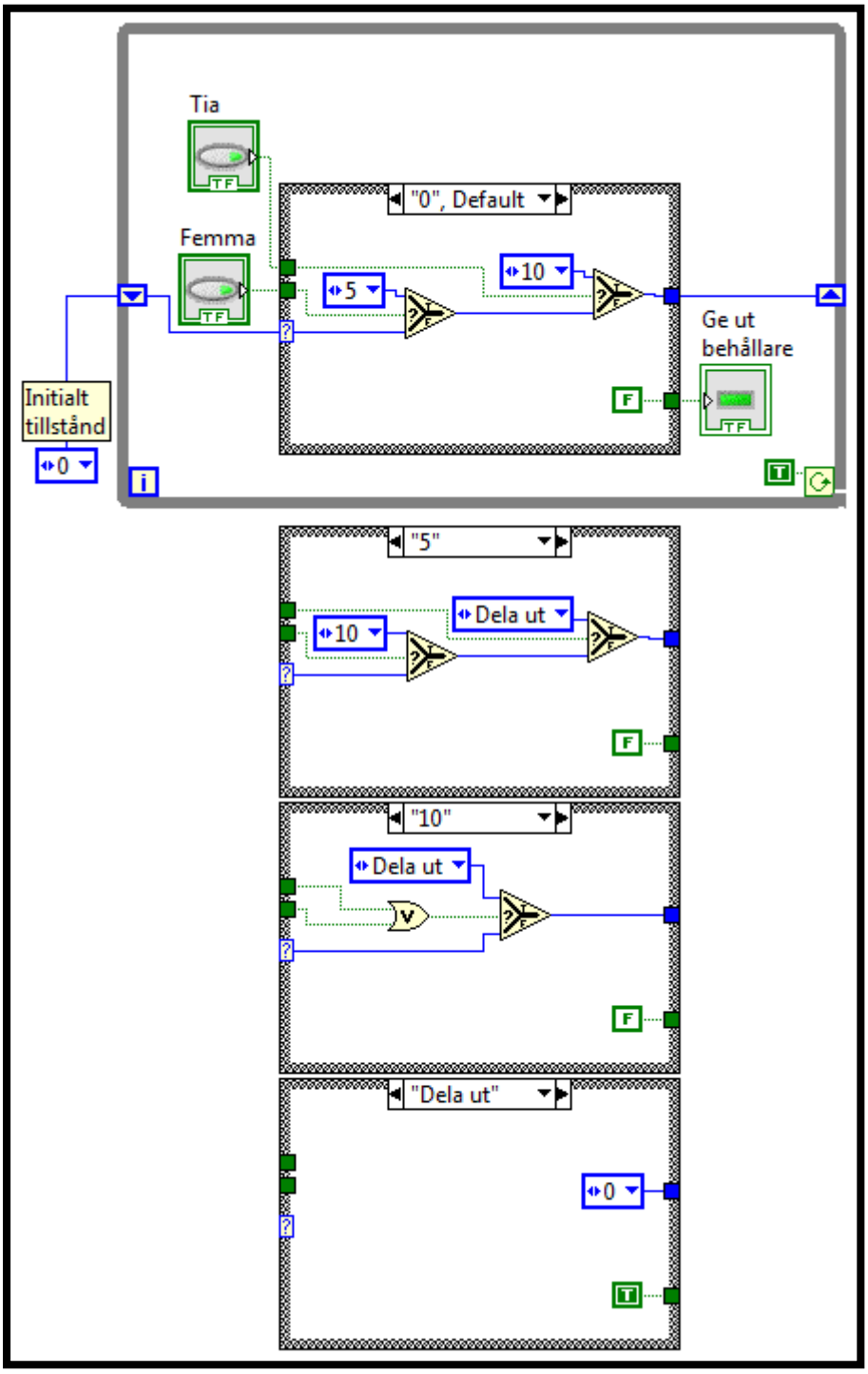


Figur 5.8. Front panel-fönstret till det enklare LabVIEW-programmet.

5.2.3.2 Tillståndsbaserad logik i LabVIEW

Då LabVIEW-miljön hade en mall för FSM-program gjordes ett program med sådan logik också, med samma funktion som programmet i föregående delkapitel. Detta delkapitel beskriver ett grundläggande exempel som gjordes som förberedelse inför skapandet av ett motsvarande program för teststyrning.

Bilden nedan föreställer ett LabVIEW-program på FSM-format som följer programflödet beskrivet av ***Figur 2.3*** på sidan 7 (läskautomat), och har lagts in för att läsaren skall få en grundläggande förståelse för hur bilderna som följer skall tolkas.



Figur 5.9. LabVIEW-program för att styra en läskautomat som endast tar emot femmor och tior, samt inte ger tillbaka någon växel. Läskbehållarna kostar femton kronor.

Den inre rektangeln är en case-sats och alla fall har lagts under huvud-while-

Precis såsom bildtexten beskriver är det så att de tre undre rutorna är tre fall (cases), en för varje tillstånd – initialtillståndet ”0” kan ses inuti while-loopen. Egentligen är de tre undre rutorna också inuti while-loopen och skulle också visats om användaren klickat på höger- respektive vänsterpilarna bredvid etiketten ”Default” högst upp i case-rutan.

Om programmet studeras från vänster till höger; den blå rutan under ”Initialt tillstånd” är en konstant av typen *enum*, som i det här fallet egentligen bara är siffrorna 0, 1, 2 och 3 fast namngivna till ”0”, ”5”, ”10” respektive ”Dela ut” (LabVIEW benämner alltså tillstånden internt som 0-3 i detta fall). Programmet (eller while-loopen) börjar alltså i tillståndet 0, varefter ingångarna kontinuerligt kontrolleras i varje iteration av loopen. Den översta case-rutan exekveras i varje iteration fram till att någon av mynten insätts, eftersom tillståndet ej ändras i case-rutan ifall inget mynt insätts.

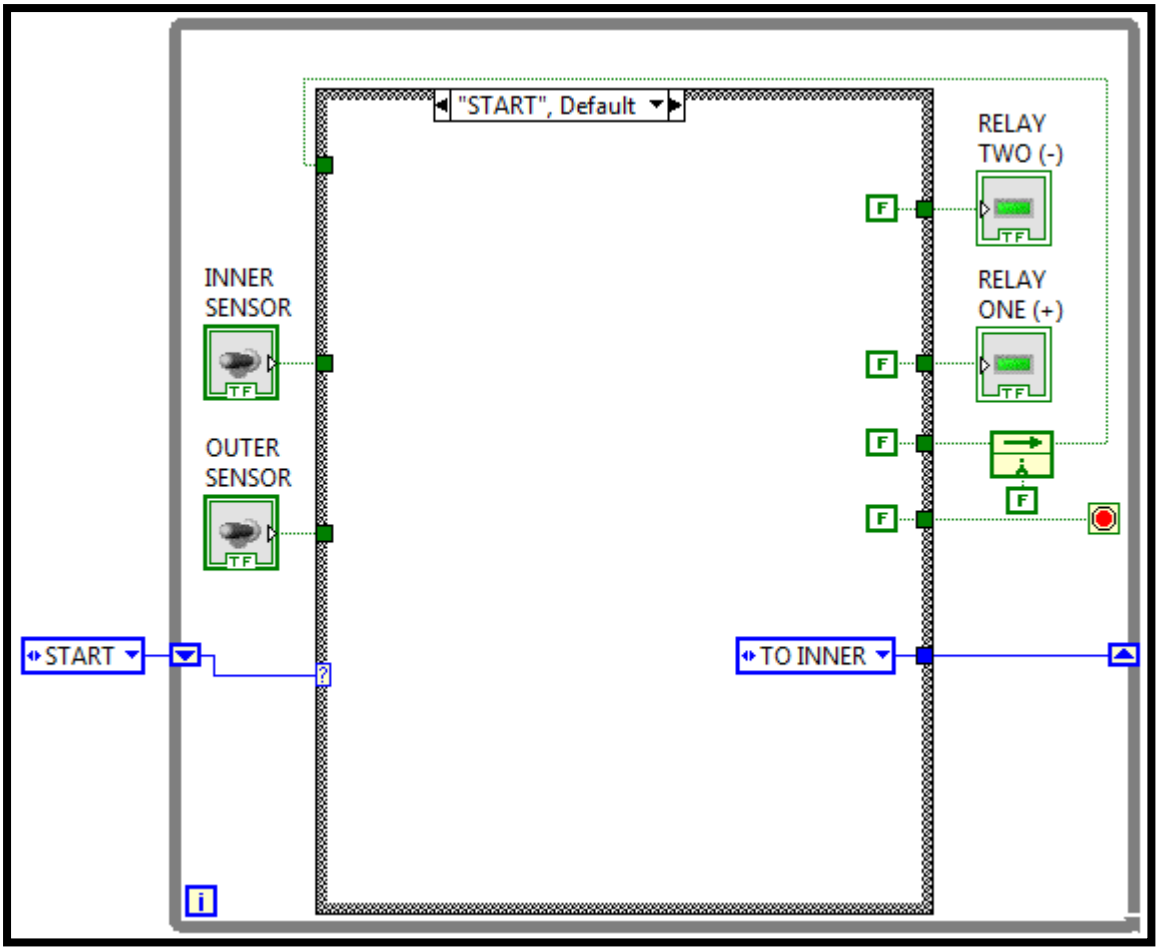
De triangulära blocken inuti case-rutorna är så kallade selector-block, som ger ut en av de två värdena vid ingångssidan beroende på om ingången i mitten är en logisk etta eller en logisk nolla. Tillståndet ändras som sagt ej om inga mynt insätts, detta kan ses på att det gamla tillståndet (blå kabel) leds vidare genom alla false-fall i selector-blocken och tillståndet ändras således ej utan att någon av ingångarna ettställs. Beroende på vilket mynt som insätts i respektive state väljer blocken vad nästa state skall vara (blå rutor inuti case-rutorna – *enum*-konstanter).

Den enda tillståndsrutan som sänder ut en etta på ”Ge ut behållare”-utgången är just tillståndet ”Dela ut”. Detta tillstånd har inga kontroller av ingångar utan går direkt tillbaka till tillståndet ”0” – programmet startar om från början.

5.2.3.3 Tillståndsbaserad logik – enkel testcykel

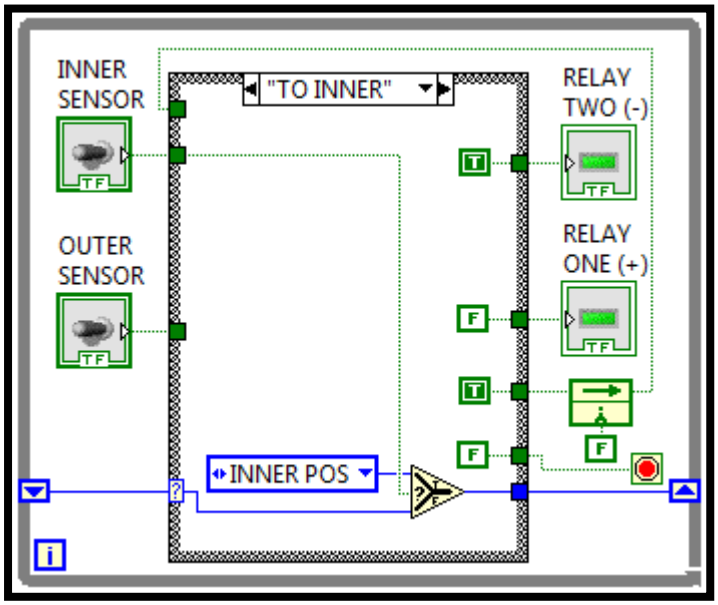
Programmet som beskrivs i detta kapitel skapades för att två olika programmeringssätt skulle kunna jämföras. Samma front panel som i kapitel 5.2.3.1 användes till detta exempel varför bilden som visar den inte återfinnes här. Programmet följer precis som tidigare exempel på enklare testcykler flödesschemat på sidan 16.

Figur 5.11 (nästkommande sida) beskriver en while-loop som omsluter en inre case-sats för hantering av tillstånd. Det initiala tillståndet är ”START”, vars enda funktion är att sätta nästa tillstånd till ”TO INNER” och att nollställa alla reläutgångar och all information om föregående cykel. Blocket under ”relay one”-blocket är ett återkopplingsblock som för vidare information från en loopiteration till nästa, och false-konstanten under är vad blocket skickar ut i första loopiterationen då ingen information finns från den föregående. I detta program används återkopplingsblocket för att skapa en flankdetektering för att kunna räkna upp cykelantalet på ett bra sätt.



Figur 5.11. Tillståndsbaserat LabVIEW-program, endast START-tillståndet är med i denna bild.

Som sagt blir nästa tillstånd efter detta tillstånd ”TO INNER” – följs de blå kablarna från *enum*-konstanten inses lätt att nästa while-loopiteration kommer att ha ”TO INNER” som insignal till case-satsen vilket leder till att rutan nedan (***Figur 5.10***) exekveras;



Figur 5.10. Andra tillståndet i det tillståndsbaserade LabVIEW-programmet.

Programmet har redigerats för att bilden inte skall ta alldeles för stor plats varför utseendet ändrats något från det i Figur 5.11.

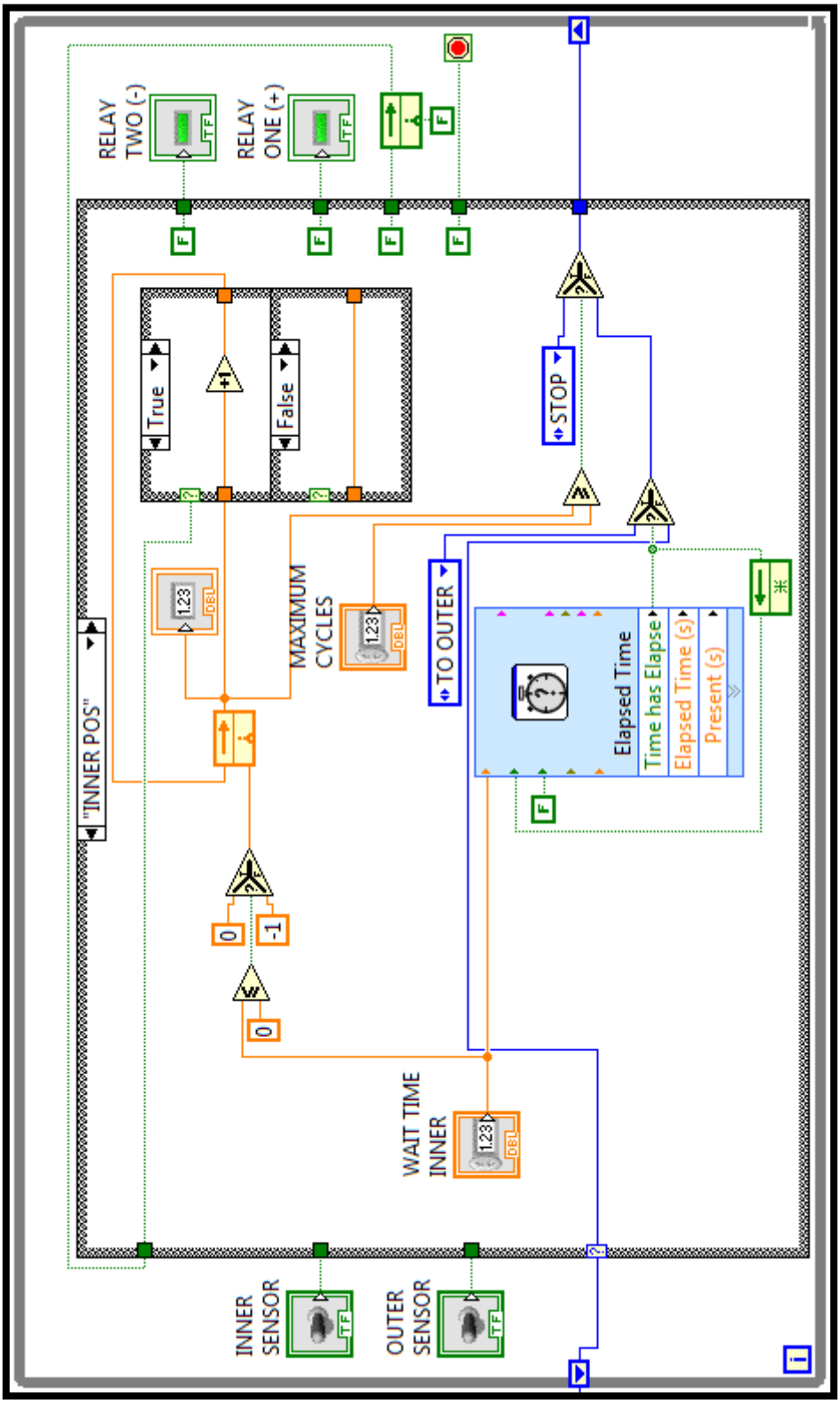
Det som bilden tydliggör ytterligare är att relä två aktiveras (drift mot minusläge) samt att återkopplingsblocket får en logisk etta på ingången – denna logiska etta används som sagt för flankdetektering; mer precist talar den om för programmet att tillståndet i den senaste loopiterationen var just ”TO INNER”. Se *Figur 5.12* för en mer exakt beskrivning av hur denna information används.

Bilden på nästa sida beskriver tillståndet som kommer efter det i ***Figur 5.10***, nämligen ”INNER POS”. Detta tillstånd är något mer invecklat. Ungefär samma operationer som innan utförs på *enum*-värdena; drift mot plusläget aktiveras (”TO OUTER”) då en viss tid gått sedan det aktuella tillståndet startats, alternativt så blir nästa tillstånd ”STOP” ifall det maximala antalet cykler uppnåtts.

Less than or equal-blocket med -1 och 0 vid selector-blocket efter var ett (ganska dåligt) sätt att hantera felräkningar som uppkommer i vissa situationer.

Den lilla case-satsen (som egentligen är en if-sats i detta fall) illustrerar hur omständigt det är att göra något så enkelt som flankdetektering i LabVIEW, även om det vore möjligt att skapa en komplicerat som hanterar det hela och sedan aldrig mer behöva göra om det. Jämförs detta med exempelvis Siemens LOGO! blir det tydligt att Siemens lagt mer krut på sådana typer av funktioner.

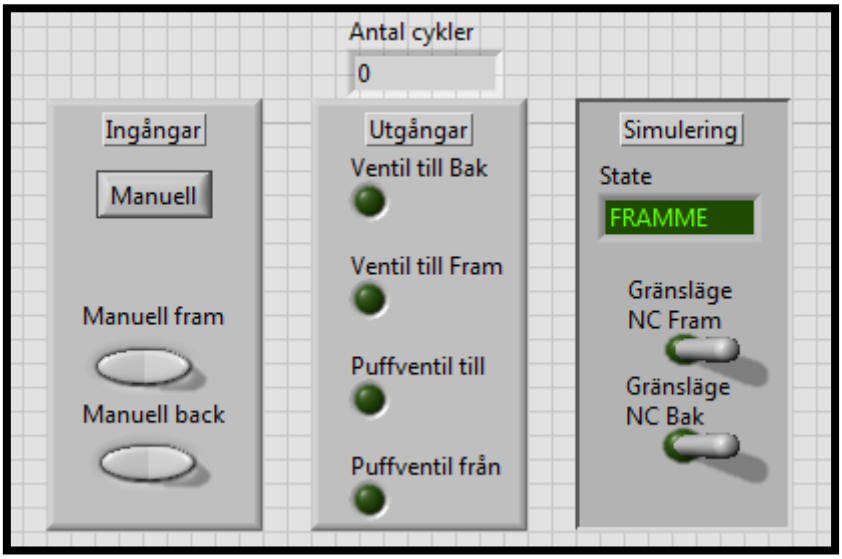
De återstående tillstånden återfinnes i bilaga C då de egentligen är likadana som de hittills beskrivna tillstånden, fast med ändrade värden. Ett tillstånd skiljer sig något – ”STOP” – men det enda det tillståndet gör är att ettställa utgången för att stoppa while-loopen.



Figur 5.12. Tillståndet ”INNER POS” för det tillståndsbaserade LabVIEW-programmet.

5.2.3.4 LabVIEW – invecklat program

Detta program skapades för att fylla samma funktion som det i kapitel 5.2.2.2, och en något mer strömlinjeformad frontpanel skapades för att också ge exempel på vad LabVIEW-miljön kunde åstadkomma grafiskt. Givetvis är denna frontpanel inte särskilt välutvecklad – skulle en riktig styrning skapas kan allt fås som en egen .exe-fil och ej behöva köras i LabVIEW-fönstret. Exempelvis skulle även pop-up-fönster kunna läggas till då fel uppstår eller då det maximala cykelantalet nåtts, eller så skulle någon annan grafisk representation av data kunna läggas till. Möjligheterna är många. Front panel-rutan kan ses nedan.



Figur 5.13. Front panel-rutan till det mer invecklade LabVIEW-programmet.

Bilden är egentligen självförklarande, men för säkerhets skull kan sägas att det finns fem knappar på panelen varav tre skulle funnits i verkligheten – de under ”ingångar”. LED-lamporna under ”utgångar” används för att representera vilken information som sänds ut till den tänkta processen.

Det blir tydligt redan vid panelen att detta program gjorts med tillståndsbaserad logik eftersom det finns ett fält som beskriver det nuvarande tillståndet.

Programmet är ganska stort och invecklat; den intresserade hittar det i bilaga D. Det som kan sägas om programmet är att det var relativt enkelt att göra, då tillståndsbaserad programmering är väldigt enkelt i LabVIEW – tillstånden beskrivs ju av en lätt identifierbar variabel i blått, och vad som skall göras i tillstånden kan programmeras in tillstånd för tillstånd i den inre case-rutan. Att tillståndsbaserad programmering är så intuitivt är något som ytterligare talar för LabVIEW-programmeringsmiljön; det är som sagt väldigt enkelt att göra FSM-program i LabVIEW.

Det finns dessutom en färdig mall för en FSM.

5.2.3.5 LabVIEW, helhetsbild

LabVIEW tycks vara en lösning som skulle kunna ha fler användningsområden än de två PLC:er som studerats. Mätning är ett sådant exempel; det går att få realtidsgrafer av temperaturförlopp och strömförbrukningar vilket gör att själva övervakningen blir lätt att hantera. Det går till exempel att se om någonting är fel redan tidigt i testet, istället för att testet behöver göras klart innan sådana fenomen kan observeras. Samtidigt kan dessa värden sparas i en logg på en vald adress då det finns block som låter användaren specificera *file path* när data skall sparas. Loggarna skulle till och med kunna sparas automatiskt någonstans där de kan kommas åt från företagets datorer - datorer som inte nödvändigtvis behöver vara i samma rum som testet. LabVIEW kan spara loggade mätvärden i .csv-format (mycket vanligt format, kan till exempel läsas av Excel), eller i andra format om funktioner för det läggs till.

Dessutom går det med LabVIEW att få så detaljerade mätningar att det skulle vara möjligt att använda dem för tillståndsövervakning. Det skulle sedan gå att skriva program i LabVIEW och programmera in dem i någon National Instruments-MCU som sedan skulle kunna sköta övervakningen baserat på de mätningar som gjorts i testlabbet.

Eftersom LabVIEW verkar vara en programmeringsmiljö som är tänkt för lite större program kan programmen modulariseras i form av funktionsblock som sedan kan återanvändas vid skapandet av liknande program. Till skillnad från LOGO!Soft Comfort finns det en mycket högre gräns på antalet in- och utgångar till blocken – så mycket högre att antalet troligtvis aldrig kommer att uppnås och programmen kan därför modulariseras helt efter vad som behövs.

Det negativa som kan ses med LabVIEW är att det är ett lite annorlunda programmeringsspråk, vilket gör att den som skall göra något med det kommer att behöva fräscha upp sina LabVIEW-kunskaper ifall det går för långt mellan programmeringstillfällena (det vill säga att LabVIEW har en förhållandevis hög inlärningströskel). Skall en ny person göra något med programmen måste troligtvis några dagar läggas på att förstå ett tidigare gjort program innan några modifieringar kan göras. Det är också ofta något mer omständigt att göra modifikationer som i ett vanligt programmeringsspråk bara hade kunnat göras genom att ytterligare en rad kod förs in någonstans.

Slutligen måste också nämnas att LabVIEW är betydligt dyrare än PLC-lösningarna; en modul med 4 analoga ingångar (CompactRIO) kostar 9325 kronor[27] medan Siemens LOGO! 12/24 RCE 0BA7 base-moduler (hanteringskapacitet 4 analoga ingångar, 4 digitala, samt 4 digitala utgångar, kan dessutom utföra styrning själv samt har ethernet-anslutning) kostar 2412 kronor[28]. Mjukvaran LabVIEW kostar dessutom 8995 alternativt 25030 kronor[29] beroende på hur mycket beräkningar och dylikt det är önskvärt att göra, vilket är att jämföra med LOGO! Soft Comfort-mjukvaran som kostar 496 kronor[30].

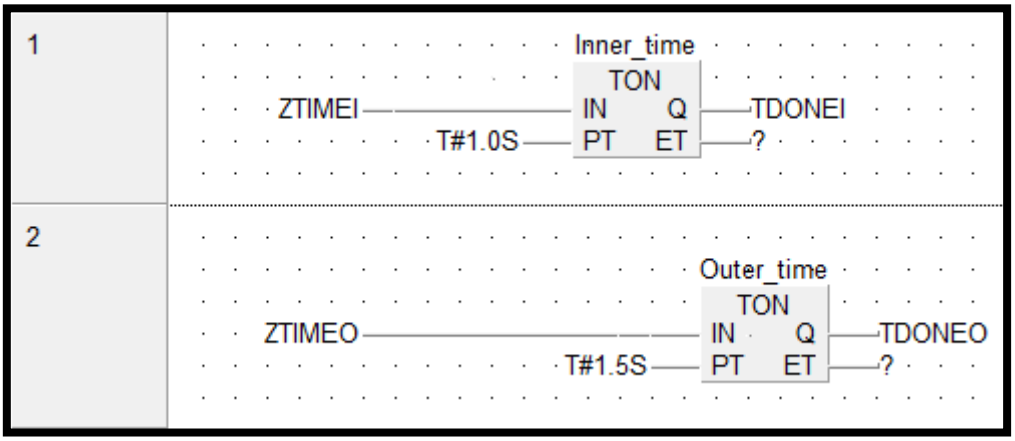
[27] sine.ni.com/np/app/main/p/ap/global/lang/sv/pg/1/ps/30/sn/n25:device,n24:cRIO/, acc. 2013-05-20.
[28] www.elfa.se/elfa3~se_sv/elfa/init.do?item=25-708-78&toc=0&q=siemens+LOGO%21, acc. 2013-05-20.
[29] www.ni.com/labview/buy/, acc. 2013-05-20.
[30] www.elfa.se/elfa3~se_sv/elfa/init.do?item=25-706-81&toc=0&q=logo+soft+comfort, acc. 2013-05-20.

5.2.4 PLC – Siemens S7-1200

Då Siemens ej hade någon tillgänglig demonstrationsprogramvara till sin SIMATIC S7-serie användes istället Mitsubishis GX IEC Developer som också har stöd för den typ av tillståndsbaserad programmering som skulle jämföras – SFC. Siemens PLC-programvara har istället för SFC stöd för ett språk de kallar STL[31], som också är ett tillståndsbaserat programmeringsspråk[32] och de två språken torde således ha liknande struktur. Programmeringsmiljön till Siemens S7-1200 har dessutom stöd för de båda språk som Siemens LOGO! kan hantera; FBD och LAD[31].

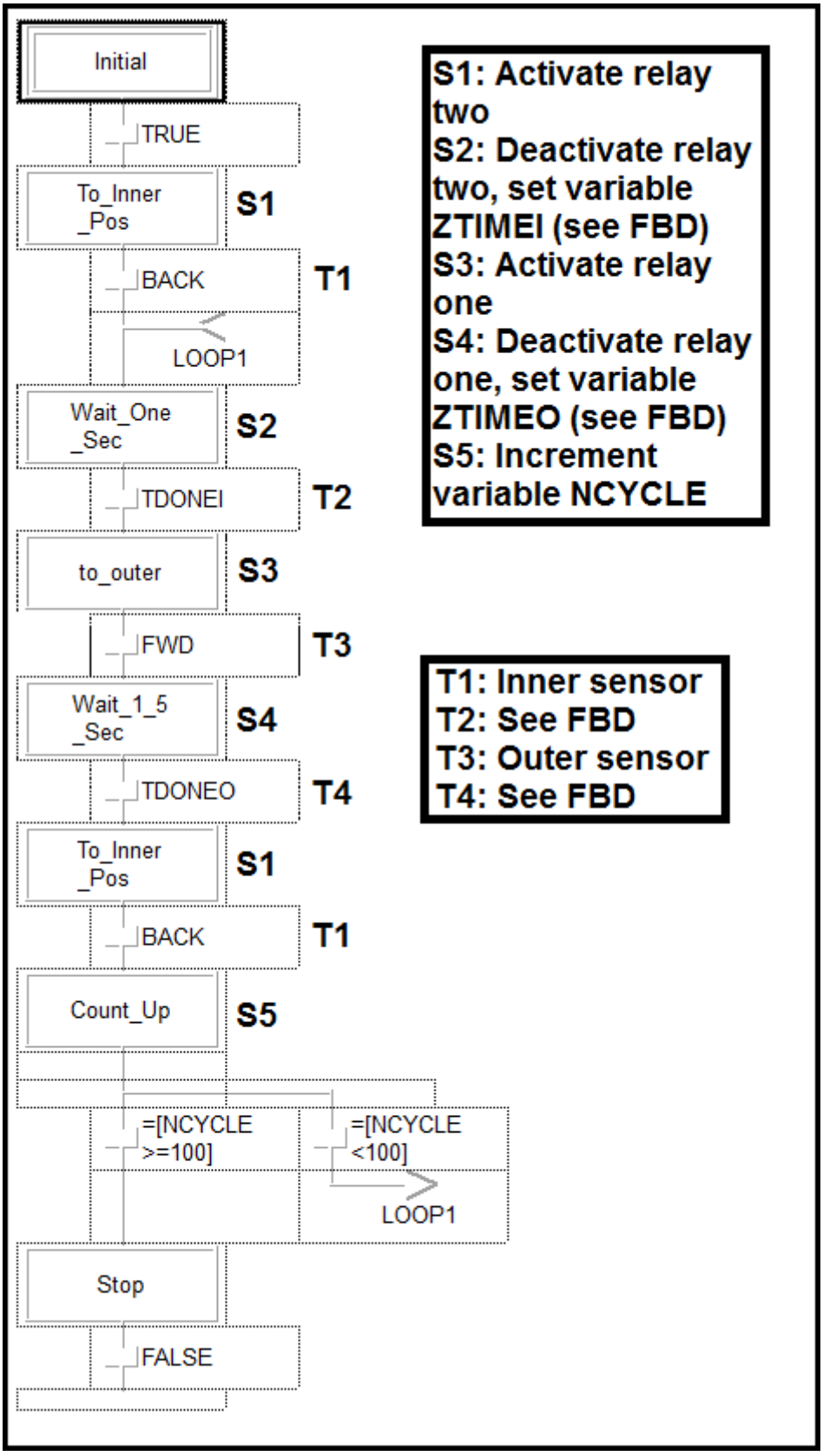
SFC-programmet som skapades kan ses nedan (*Figur 5.14*) samt på nästa sida (*Figur 5.15*), där bilden nedan beskriver FBD-delen som krävs för att hantera vissa förlopp trots att SFC används. I det här programexemplet behövde endast två timers göras i FBD men vanligtvis skulle även exempelvis hantering av stopp- eller startknappar, nödstopp, och knappar för manuell styrning behöva hanteras genom FBD.

Bilden på nästa sida beskriver hur programmets SFC-del ser ut. Precis som i ett flödesschema börjar programmet högst upp och fortsätter nedåt genom olika övergångsvillkor och programhopp. Programmet stoppas då 100 cykler körts – en evig loop har gjorts längst ned då programstopp-blocket ej gick att återfå då det tagits bort (tom ruta längst ned).



Figur 5.14. FBD-delen av SFC-programmet, gjord i GX IEC Developer.

[31] www.automation.siemens.com/mcms/simatic-controller-software/en/programming-options/Pages/Default.aspx, acc. 2013-05-02.
[32] Försäljningsbroschyr SIMATIC, www.automation.siemens.com/salesmaterial-as/brochure/en/brochure_simatic-industrial-software_en.pdf, acc. 2013-06-04.



Figur 5.15. SFC-program gjort i GX IEC Developer.

Vid programmering med SFC eller liknande programmeringsspråk med tillståndsbaserat programflöde är det enklare att hantera (och framför allt att visualisera) den typ av program som uppkommer vid repetitiva cykler än det är med FBD-program. SFC är dock såklart mest

intressant vid mer invecklade program – program som kan vara svåra att överföra från planerat test till färdig programkod; vid enklare programflöden är det något överdrivet att använda SFC istället för FBD (se sid. 8 för exempel på sådant överdrivet användande).

På grund av att Siemens S7-1200 har samma funktionalitet som LOGO! och mer därtill, samtidigt som testerna som görs har en relativt liten omfattning (i automationssammanhang), är S7-1200 lite överdriven i för den aktuella situationen. Den myriad av menyer, funktioner, funktionsblock, I/O-möjligheter och så vidare som S7-1200 erbjuder bidrar bara till att göra miljön mer svårhanterlig ifall arbete inte sker kontinuerligt med PLC:n. Vidare kommer svårhanterligheten också att visa sig då en oinsatt eller ny person skall arbeta med PLC:n – viss risk finns till och med för att problemet dyker upp även med insatta personer ifall testprogram ej behöver ändras så ofta, just på grund av att programmeringsmiljön är så omfattande.

5.2.5 Jämförelse

Med kravspecifikationen på sidan 26 som grund jämfördes de tre lösningsförslagen Siemens LOGO!, LabVIEW samt Siemens SIMATIC S7-1200 med hjälp av en Pugh-matris (*Tabell 5.2*).

Trots att Siemens SIMATIC S7-1200 hade ett högre sammanvägt värde än referenslösningen LOGO! valdes Siemens LOGO! som den lösning vilken skulle realiseras i form av en modell. Detta gjordes eftersom det fanns en del saker som inte tagits med i den ursprungliga kravspecifikationen, men som upptäcktes under arbetets gång. Exempelvis krävs mer programmeringskunskaper för att hantera S7-1200 än för att hantera LOGO!, och dessutom fanns sedan tidigare en LOGO! på plats vilket innebar att en större vana fanns hos alla inblandade. Förutom det är som sagt S7-1200 en mer omfattande lösning – bara programmeringsmiljön kan te sig lite skrämmande för den som inte använt den innan och då finns risk för att det bara fortsätts göras nya seriekopplingar av reläer, då det går snabbare och den kunskapen redan finns.

Vidare är LOGO!:n bättre anpassad för just den här situationen; enkla automatiska förlopp med enklare mätningar och få beräkningar. Att använda S7-1200 vore kanske intressant om det vore önskvärt att exempelvis köra tio test på en och samma PLC, var och en med invecklad styrning (varierande last, tre eller fler lägesgivare, PID-reglering, med mera) och en HMI-skärm för övervakning av testerna.

Pugh-matris	**Viktn.**	**LOGO! (ref.)**	**LabVIEW**	Viktat ref.-värde	**SIMATIC S7-1200**	Viktat ref.-värde
Typ av test						
Livslängdstest	4,5	0	-1	-4,5	0	0
Prestandatest	4	0	0	0	0	0
Intermittenstest	4,5	0	0	0	0	0
Förstörande test	4	0	0	0	0	0
Klimattest	0,5	0	0	0	0	0
Mätning av storheter						
Temperatur – motor	3,5	0	0	0	0	0

Temperatur – frys	1	0	0	0	0	0
Strömstyrka	5	0	0	0	0	0
Antal cykler	5	0	0	0	0	0
Translationshastighet	3,5	0	-1	-3,5	0	0
Last	4,5	0	0	0	0	0
Övriga funktioner						
Ställbar maxström	4	0	1	4	0	0
Ställbar maxtemperatur	3,5	0	1	3,5	0	0
Ställbar frystemperatur	3	0	0	0	0	0
Stopp vid andra händelser	4	0	2	8	0	0
Manuellt stopp	3,5	0	-1	-3,5	0	0
Ställbar väntetid vid ändlägen	4,5	0	0	0	0	0
Möjlighet att köra enkla testcykler	3	0	-2	-6	0	0
Möjlighet att köra invecklade testcykler	4	0	1	4	1	4
Funktioner utanför programmeringsmiljö						
Ställa in maxström	3,5	0	-1	-3,5	1	3,5
Ställa in maxtemperatur	3,5	0	-1	-3,5	1	3,5
Display av nuvarande mätvärden	3,5	0	2	7	1	3,5
Starta ett test med knapptryck	4	0	0	0	0	0
Stoppa ett test med knapptryck	4	0	0	0	0	0
Ställa in väntetider vid ändlägen	4,5	0	0	0	1	4,5
Modifierbarhet – användarvänlighet						
Enkelt att ställa upp ett enkelt test	4,5	0	-1	-4,5	0	0
Enkelt att ställa upp ett mer avancerat test	4	0	2	8	1	4
Enkelt att anpassa till nya produkter	3,5	0	1	3,5	0	0
Enkelt att starta upp ett test	4	0	-1	-4	-1	-4
Dataloggning						
Automatisk dataloggning	4,5	0	0	0	1	4,5
Hög samplingsfrekvens	1,5	0	0	0	0	0
Fler än ett sampel per cykel	3,5	0	0	0	0	0
Lättanvänt dataformat	3,5	0	1	3,5	0	0
Dynamisk samplingsfrekvens	3	0	1	3	0	0
Tillförlitlighet						
Hög livslängd hos ändlägesgivare	4,5	0	0	0	0	0
Exakthet hos mätgivare	4	0	2	8	0	0
Data sparas trots avbrott	4,5	0	0	0	0	0
Test går att återuppta efter avbrott	4,5	0	-1	-4,5	0	0
Självgående	5	0	-1	-5	0	0
Övrigt						
Kostnad	5	0	-3	-15	-2	-10
Sammanvägning		0		-12		13,5

Tabell 5.2. Pugh-matris – stöd för jämförelse mellan lösningar.

5.2.5.1 Pugh-matris, motiveringar för LabVIEW

Om *Tabell 5.2* studeras från toppen motiveras till att börja med LabVIEW:s ”-1” vid livslängdstester av att LabVIEW körs från en PC, som ju inte är gjord för att köras flera månader efter varandra utan att stängas av i lika hög grad som en PLC är.

Motiveringen till att LabVIEW var sämre på att mäta translationshastighet var att LabVIEW som sagt körs från en PC, och med andra ord är det inte fullständigt säkert att programmet körs i realtid – tidmätningar och därmed hastighetsmätningar blir med andra ord osäkra. Däremot finns mer beräkningsmöjligheter i LabVIEW med deriveringsfunktioner och dylikt, vilket inte LOGO! har, vilket är varför LabVIEW inte får ”-2” här.

Eftersom LabVIEW är bättre lämpat för mätning än LOGO! kommer en inställd maximal ström och -temperatur att fungera på ett bättre sätt ifall det görs i LabVIEW än i en LOGO!. Störningar som kanske hade fåtts i LOGO!:ns mätmoduler skulle kunnat filtreras bort i LabVIEW med någon inbyggd filterfunktion. I kontrast till detta hade inställning av sådana värden fungerat sämre i LabVIEW än i LOGO! eftersom LOGO! har en display för detta där valda parametrar kan visas och ställas in, medan LabVIEW visar allting i samma fönster.

Av den anledningen är det enklare att se nuvarande värden i LabVIEW än i LOGO!-miljön eftersom LOGO!:ns display är så liten och endast kan visa enklare saker. LabVIEW kan visa allt med grafer, samtidigt som aktuella numeriska värden visas.

Vidare är LabVIEW mer lik en traditionell programmeringsmiljö och ”stopp vid andra händelser” skulle därför kunna programmeras in mycket enklare, oavsett hur specifik händelsen i fråga är.

Ett manuellt stopp är enklare att hantera genom att en PLC finns på plats, eftersom nödstoppknappen helt enkelt kan bryta strömförsörjningen till allting, inklusive PLC:n, och att PLC:n efter det kan startas upp och fortsätta köras utan problem. Stoppas en PC med LabVIEW på samma sätt tar det längre tid, och inloggning och start av .exe-filen måste ske manuellt.

Möjligheten att köra enklare alternativt mer invecklade testcykler har poängsatts enligt ovan eftersom LabVIEW-programmet för jämförelse blev ganska stort redan vid det enkla programmet, men storleken ökade inte särskilt mycket då det avancerade programmet gjordes. Det gjorde däremot LOGO!-programmets storlek.

LabVIEW är enklare att anpassa till nya produkter eftersom så många moduler finns för en stor mängd applikationer; det som hade lösts av några digitala utgångar och en analog utgångsmodul på en LOGO!, med ett motorkort för BLDC-styrning påkopplat, löses enkelt av en enda CompactRIO-modul. Nackdelen, en ganska ordentlig sådan, är att det är dyrare.

LabVIEW kommer med ett antal funktioner för att ge ut ett valt dataformat, medan LOGO! endast kan spara data i .csv-format, vilket är varför LabVIEW fått sämre poäng på den raden.

Anledningen till att dynamisk samplingsfrekvens fungerar bättre med LabVIEW är att det finns större möjligheter att göra detaljerade förlopp och tidsstyrning i LabVIEW-miljön än i LOGO!Soft Comfort.

Eftersom LabVIEW är gjort mest för mätning och har egna moduler och givare för sådant var det en självklarhet att LabVIEW skulle få högre poäng i ”exakthet hos mätgivare” än de andra två.

Slutligen skall nämnas att det är den sista raden som sänker LabVIEW. Hade inte kostnaden varit ett så stort problem skulle LabVIEW fått en sammanlagd poäng av 3 och därför blivit vald över Siemens LOGO!. Men det är en så markant investeringsskillnad i LabVIEW jämfört med PLC:er att fördelarna inte väger upp den kostnaden. Dessutom finns idag inte behov av så noggranna mätningar, men LabVIEW är definitivt värt att titta närmare på ifall det är önskvärt att studera produkter väldigt noggrant – troligtvis har Test Centre valt LabVIEW av en god anledning.

5.2.5.2 Pugh-matris, motiveringar för S7-1200

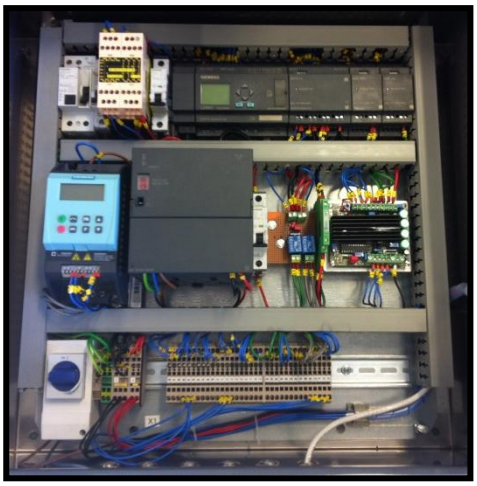
Eftersom S7-1200 likt LOGO! är en Siemens-PLC har den bara fått samma eller högre poäng än LOGO!:n eftersom den helt enkelt är en mer avancerad PLC och inget tagits med i kravspecifikationen som talar emot att ha en mer avancerad lösning. Undantaget är kostnaden, där S7-1200 överträffas av LOGO!, och ”enkelt att starta upp ett test” då in- och utgångar är lättare att mappa i Siemens LOGO! och programmeringsmiljön till S7-1200 rent allmänt är mer otymplig.

Dessvärre för Siemens S7-1200 valdes Siemens LOGO! istället för den på grund av de anledningar som nämndes i början av kapitel 5.2.5, trots att S7-1200 egentligen utgör en bättre lösning.

5.3 Skapande av testmodell

Detta underkapitel beskriver den modell som slutligen realiserades i form av ett kopplingsskåp (*Figur 5.16*) och två ställdon av olika typ – ett ställdon med BLDC-motor och ett med en AC-motor. Underkapitlet beskriver även PLC-programmets slutliga form. Styrsystemet fungerar som tänkt, med undantag för strömmätningen till ett av testen samt att det inte har den generella form som vore önskvärd att ha.

Utöver allt nedanstående angående testmodellen skall läggas till att Pt100-givare lagts in i testuppbyggnaden (för mätning av temperatur), något som bör nämnas men inte förtjänar ett eget delkapitel.



Figur 5.16. Kopplingsskåpet som styrs av vald LOGO! PLC.

5.3.1 Val av LOGO!-typ

Siemens LOGO! finns i flera olika modeller, och i olika versioner – LOGO!:n på plats har version 0BA6 medan den nyaste versionen är 0BA7 som har fler implementerade funktionsblock, större programminne och plats för ett SD-kort. LOGO!:s med version 0BA7 finns dessutom med stöd för ethernet-anslutning och loggning, något som de tidigare versionerna saknar[33]. SD-kortet kan användas för lagring av program och/eller förstoring av minnet för loggar. Vidare möjliggör SD-kortet att loggar kan överföras genom att SD-kortet plockas ut och läses av utifall att ethernet-anslutningen inte skulle fungera eller Soft Comfort inte skulle finnas på den dator som skall ta emot loggarna.

LOGO!-modellerna som har stöd för ethernet-anlutning finns endast tillgängliga med reläutgångar[33] (i motsats till transistorutgångar) – modellerna som har ”E” (*Ethernet*) i namnet har alltid ett ”R” (*Relay*). Detta är något olyckligt då reläutgångarna har en begränsad livslängd – 0,66 A switchad ström sägs ge en livslängd på minst en miljon switchningar[34] vilket egentligen är för lite.

Ethernet-anslutningen är nödvändig för att loggning skall kunna ske via PC, vilket är intressant för att slippa så mycket handpåläggning vid mätningar. Det skall noteras att LOGO! endast kan överföra mätloggar då enheten är avstängd, alltså kan inga realtidsgrafer fås utan att ytterligare (icke-Siemens) programvara läggs till. Ett undantag är regulatorblocket som finns i programmeringsmiljön; det blocket kan visa nuvarande och tidigare värden bör- och ärvärden samt styrsignalens värde i en graf om ”online test” körs.

Allt detta mynnar ut i att den LOGO! som väljs måste vara av typen 12/24 RCE eller 230 RCE, och den som till slut valdes var 12/24 då en 24 V-matning sedan tidigare fanns tillgänglig för logik och styrning.

Moduler som valdes som påbyggnad till huvudmodulen 12/24 RCE var;

- AM2 AQ; för att kunna ge en analog utsignal till frekvensomriktaren. Används också för att ge en konstant 5 V-signal till BLDC-motorkortet.
- AM2 RTD; för direkt inkopping av Pt100-givarna utan mätbrygga och förstärkare.
- DM16 24; digital in- och utgångsmodul med transistorutgångar, förutsätts hålla mycket längre än reläutgångarna på huvudmodulen.

samt en display – Siemens LOGO! TD.

5.3.2 Val av strömtransformatorer

De strömtransformatorer som till slut kopplades in i testskåpet var två likadana; LEM HX 05-P/SP2, med nominell mätt ström 5 A. Utsignalen från dem beskrivs av formeln nedan[35].

$$U_{UT} = 2{,}5\ V + 0{,}125 * I_M$$

[33] Försäljningsbroschyr LOGO!, www1.elfa.se/data1/wwwroot/assets/datasheets/LOGO-V7_eng_bro.pdf, acc. 2013-05-22.
[34] Användarmanual LOGO!, www1.elfa.se/data1/wwwroot/assets/datasheets/LOGOv7_eng_man.pdf, sid. 337, acc. 2013-05-22.
[35] Datablad, www1.elfa.se/data1/wwwroot/assets/datasheets/HX0350PSP2_eng_datasheet.pdf, acc. 2013-05-22.

där I_M är den mätta strömmen i A, och U_{UT} är signalen som transformatorerna skickar ut, i V.

Tyvärr gavs en växelspänning ut som utsignal från transformatorerna när en växelström mättes, vilket inte kunde hanteras av PLC:n. Av den anledningen skapades en likriktare i form av en diod sammankopplad med en kondensator vilken kopplades på utsignalen för att den skulle bli kunna tas emot av PLC:n.

Återigen stöttes problem på eftersom strömmen var så liten (0,36 A) att PLC:ns upplösning (0-10V motsvarar 0-1000 – heltal) förhindrade överföringen av information. Inga ytterligare försök gjordes för att åtgärda detta problem (se kapitlet ”Rekommendationer”).

Mätningen av strömmen till BLDC-motorn fungerade dock utmärkt.

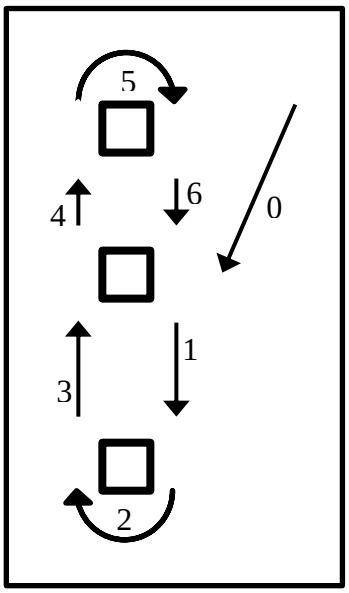
Ett alternativ till de strömtransformatorer som valdes var en något dyrare variant fanns i form av färdigbyggda moduler (fabrikat Weidmüller) gjorda för mätning av både växel- och likström, vilka kunde monteras direkt på de DIN-skenor som fanns i skåpet. Dessutom förmodas dessa fungera bättre än det hembyggda experimentkortet. De valdes inte eftersom de troddes vara för dyra samt att inget svar kom från Weidmüller angående hur precisionen var vid mätning av växelström.

5.3.3 PLC-programmet

Detta är vari huvudsakliga arbetet låg, och nedanstående underkapitel är återigen en beskrivning av ett program. Den ointresserade rekommenderas hoppa över detta kapitel.

Eftersom tillståndsbaserad programmering gjort alla program hittills så mycket mer lätthanterliga, och inget sådant program gjorts till LOGO!, gjordes programmet för styrning av test på det formatet. Det visade sig att programmet faktiskt blev mer lättvisualiserat, precis som för de försök som gjorts i LabVIEW och med SFC-program.

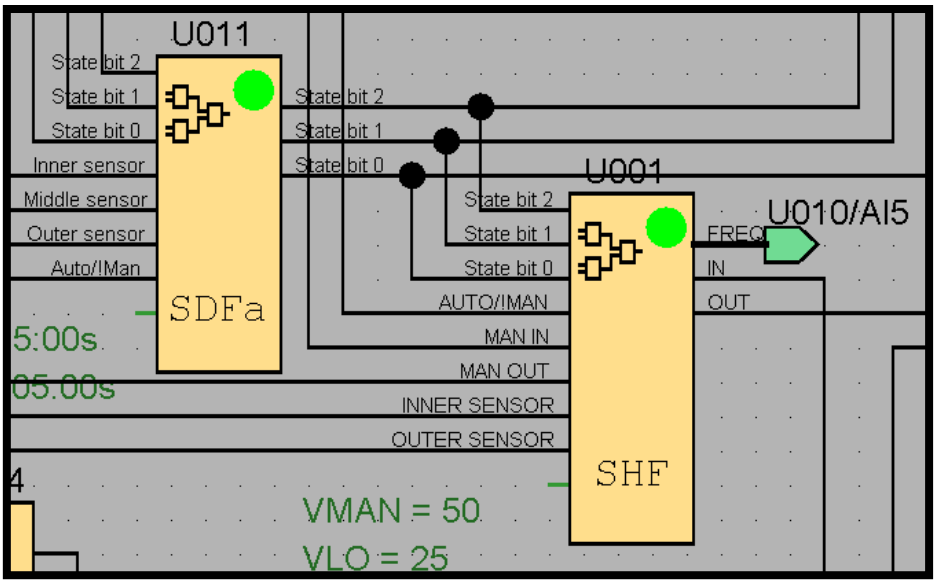
Sättet som tillstånd hanterades på i LOGO! blev något annorlunda; om testet med AC-motorn (3 lägesgivare) tas som exempel, numrerades tillstånden enligt *Figur 5.17* till höger. Tillståndet 0 användes eftersom programmet vid start eller byte från manuell drift till automatisk inte kunde fastställa i vilket läge donet befann sig.



Figur 5.17. Schematisk bild av tillståndens numrering. Tillståndet 0 är det som programmet utgår ifrån.

I LOGO!:s programmeringmiljö behövde tillstånden skrivas som binära tal och digitala ”flag”-block behövde sedan användas för att skifta tillståndet till nästa programcykel. Dessa tillståndsbitar användes sedan av två UDF-block som annars hade tagit upp en enorm plats ifall UDF inte använts.

Ett UDF-block (State Determine Function – *SDF*) skapades för att avgöra vad nästa tillstånd skall vara baserat på det nuvarande tillståndet samt olika insignaler. Ett andra UDF-block (State Handle Function – *SHF*)

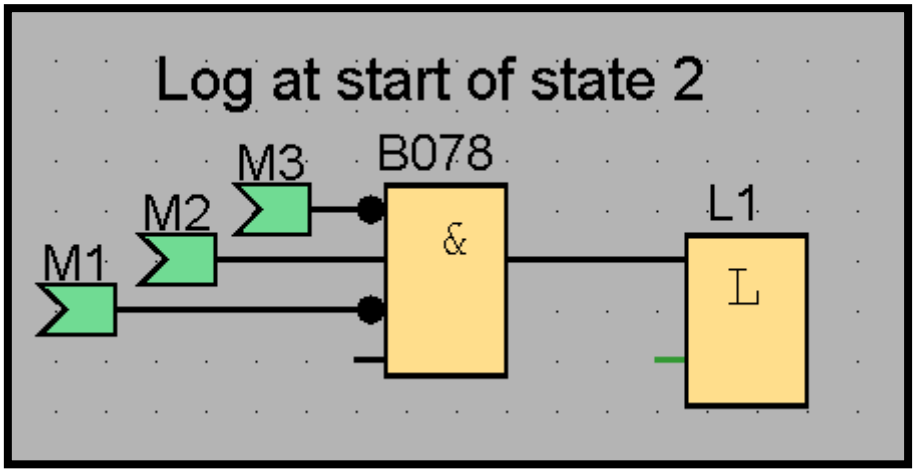


Figur 5.19. De två huvudblocken i det slutliga PLC-programmet.

använder det nuvarande tillståndet för att bestämma vad som skall skickas ut på PLC:ns utgångar. Dessa två huvudblock utgör den största delen av programmet, om än inte den visuellt största. Bilden på nästa sida (*Figur 5.19*) visar de två blocken.

Två liknande, men mindre, UDF-block finns för styrning av BLDC-testet. Resten av programmet tar mycket plats i blockdiagrammet men gör inte lika mycket. Block som återfinns i diagrammet är exempelvis block som omvandlar informationen som fås från RTD-modulen till ^{o}C, block som omvandlar insignalen från strömtransformatorerna tillbaka till ett strömvärde, ett regulatorblock för BLDC-testet, block som sköter visning av information på textdisplayen, med mera. På grund av att programmet tar så mycket plats återfinns det inte i huvudtexten – se bilaga B för fullständigt program.

Mätdata kunde loggas på ett fungerande sätt. Bilden nedan (*Figur 5.18*) visar loggningsblocket och dess inkoppling; loggning sker vid varje positiv flank på ingången.



Figur 5.18. Loggningsblocket och dess inkoppling. M1, M2 och M3 är de tidigare nämnda tillståndsbitarna.

På nästa sida kan ses resultatet av loggningen (*Tabell 5.3*); en tabell över registrerade värden – användaren får egentligen själv hålla koll på vilket blocknummer som representerar vilket värde men det har här markerats för tydlighets skull.

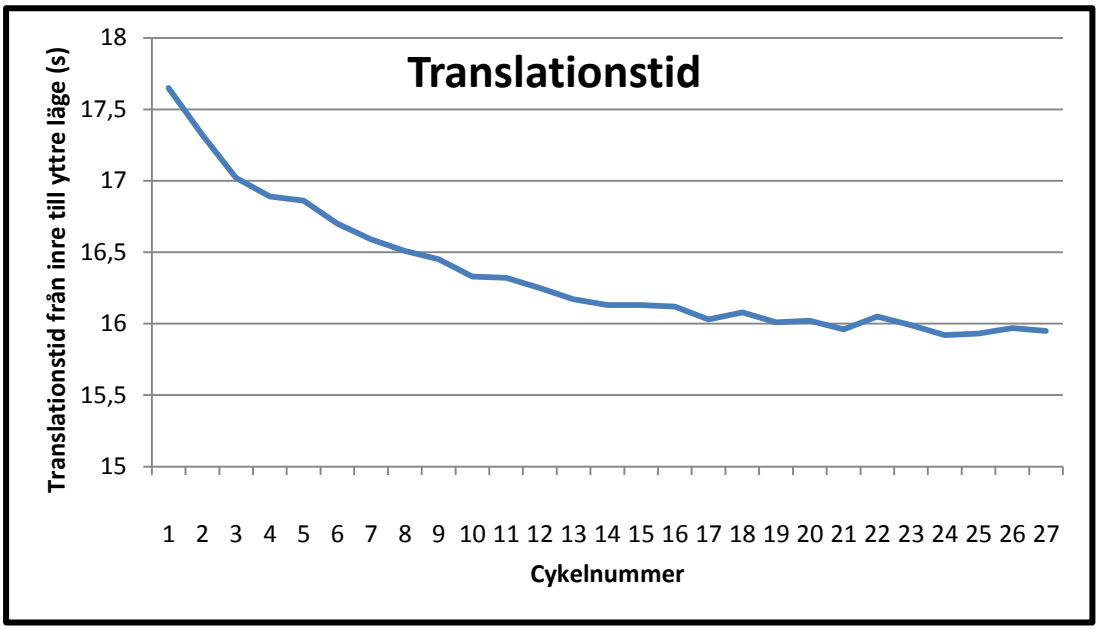
På grund av att LOGO! maximalt kan lagra värden av typen *signed short integer* (-32768 – 32767) mäts tiden i den något ovanliga enheten *centisekunder*.

	BLDC-test				AC-test			
Tid	**Cykel nr.**	**Tid inre-yttre (cs)**	**Temperatur (°C)**	**Max ström (mA)**	**Cykel nr.**	**Tid inre-yttre (cs)**	**Temperatur (°C)**	**Max ström (mA)**
Time	U007.Cnt	U007.Aq	B145.PV	B146.Max	U002.Cnt	U002.Aq	B071.Ax	B077.Max
2013-05-13 13:35:49	13	1459	55	9977	9	1645	33,0	-1390
2013-05-13 13:36:42	15	0	55	1782	10	1633	33,5	-977
2013-05-13 13:37:33	16	0	56	7360	11	1632	34,3	-71
2013-05-13 13:38:25	17	1459	56	7600	12	1625	35,0	-1307
2013-05-13 13:39:16	19	0	57	6908	13	1617	35,3	-1884
2013-05-13 13:40:09	20	1462	57	7064	14	1613	36,0	-813
2013-05-13 13:41:00	21	1465	57	7566	15	1613	36,3	-813

Tabell 5.3. Mätvärden som LOGO! loggat under fem minuters testning. Delar av loggen har klippts bort för att spara plats.

Som nämnts tidigare fungerar inte strömmätningen till växelströmsmotorn på grund av kombinationen av mätingångarnas upplösning och storleken på strömtransformatorns utsignal, vilket är varför negativa värden återfinns i den kolumnen. Dessutom loggas mätvärden vid start av tillstånd två för AC-testet, något som medför att vissa värden för BLDC-testet loggas vid olämpliga tidpunkter. Därav överhoppandet av cykel nummer 14 och 18 samt värdena 0 för vissa av translationstiderna.

Som exempel på hur loggningsfunktionen kan komma till användning har en graf skapats från mätvärden ur den fullständiga versionen av loggen i *Tabell 5.3*, se *Figur 5.20* nedan.



Figur 5.20. Graf av translationstiden mellan det inre och det yttre läget för donet med BLDC-motor, uppmätt över 27 cykler.

På grund av att testen är så specifika går inte programmet att återanvända i sin helhet till annat än just det BLDC-motorkort och den frekvensomriktare som programmet gjorts för. Däremot kan några av UDF-blocken återanvändas, till exempel ett som mäter tiden det tar för donet att nå det yttre ändläget, samt de två huvudblocken för BLDC-styrning – de för AC-styrningen med tre positionsgivare lär inte komma att behövas igen.

5.3.4 Begränsningar hos LOGO!

Hanteringskapaciteten för en LOGO! 12/24 RCE är följande[36];

- 400 funktionsblock sammanlagt
- 24 digitala ingångar
- 16 digitala utgångar
- 8 analoga ingångar
- 2 analoga utgångar
- 64 UDF-block
- 16 typer av UDF-block
- 100 funktionsblock per UDF
- 8 ingångar per UDF
- 4 utgångar per UDF

Vilket räcker till två test – begränsat av de analoga utgångarna – alternativt fyra test om endast två av dem behöver ha en analog styrsignal, här begränsat av analoga ingångar för

[36] Angivet i LOGO! Soft Comfort-programmeringsmiljö.

mätning (två per test). Fler test per LOGO! skulle kunna köras om mätningar nedprioriterades. Ännu en begränsning skulle kunna finnas i antalet funktionsblock, men eftersom de två test som projektet byggts runt är ganska krävande och programmet då endast blev 192 block stort kommer denna gräns förmodligen ej att uppnås.

En ’mjuk’ begränsning finns i att varje block tar upp till 0,1 ms att utföra[36] – det vill säga att en programcykel kan ta 40 ms – vilket skulle kunna vålla problem vid tidskritiska test.

6 SLUTSATSER OCH REKOMMENDATIONER

Detta kapitel beskriver som rubriken antyder slutsatserna som kan dras av arbetet, en återkoppling till målet som tidigare ställts upp och rekommendationer för framtida arbete med detta system.

6.1 Slutsatser

Jämförs detta projekts resultat med målet som ställts upp (kapitel 1.3) kan ses att den första delen – att ta fram en testmiljö som är bättre än den som finns idag – har uppnåtts. Den andra delen är inte lika uppfylld; resultatlösningen har vägts mot andra lösningar men valdes trots att den var sämre i många hänseenden. Emellertid anser författaren att projektets mål ändå uppnåtts med tanke på den budgetram som fanns för detta projekt samt med hänsyn tagen till programmeringsfärdigheterna hos de som huvudsakligen kommer att använda testlaboratoriet.

De huvudsakliga problemen som fanns med det tidigare testsystemet (kapitel 1.2) har lösts i och med implementeringen av det framtagna PLC-baserade systemet. Automatisk loggning tar bort behovet av att någon behöver skriva ned och mäta testdata manuellt, induktiva ändlägesgivare utan någon mekanisk anslutning kommer inte att överhettas eller slitas sönder mekaniskt, och det är nu enklare att bara skapa ett program som styr ett test istället för att seriekoppla reläer.

Det enda som är något tveksamt är huruvida lösningen går att återanvända. Visst går den att återanvända för annan BLDC-styrning och AC-styrning med tre ändlägen (med små modifikationer går den till och med att använda för tvåläges AC-styrning), men om kopplingsskåpet skall användas på någon annan typ av test måste till exempel kablar dras om. Lösningen är alltså inte tillräckligt flexibel. Detta var dock svårt att se då arbetet pågick.

6.2 Rekommendationer

Om noggrannare mätning (alltså väldigt noggrann) önskas göras rekommenderas LabVIEW starkt. Trots att det är mycket dyrare har LabVIEW mycket mer välanpassade moduler för mätning än de som LOGO! klarar av att hantera. Det var ett önskemål att se detaljförlopp av hastighet och position hos donet, något som kan göras med LOGO! men inte alls på den detaljnivå som var önskvärd – LOGO!:s analoga värden från A/D-omvandlarna har upplösningen 0-1000. LabVIEW möjliggör dessutom condition monitoring på ett sätt som LOGO! inte kan klara av – den är alldeles för simpel för något mer än temperatur- och strömövervakning. Detta kanske kanske går med S7-1200 också då den är lite mer välutvecklad, men om man ändå ska lära sig ett nytt system är faktiskt LabVIEW enklare.

Ifall LabVIEW någon gång skall implementeras rekommenderas samarbete och informationsutbyte med Test Centre Göteborg då de har gedigen erfarenhet av LabVIEW.

Utöver det bör systemet byggas om något för att det enkelt skall kunna gå att anpassa till andra typer av test. Det skåp som byggts i detta projekt är anpassat för just de två test som det byggdes för. Exempel på hur systemet kan byggas om är att ett skåp kan innehålla allt som har med PLC:er att göra (antagligen flera stycken ifall fler test önskas köras) medan ett annat har plats för moduler som kan ”knäppas” dit och som är anpassade för specifika test – för

BLDC-testet skulle till exempel en sådan modul ha motorkortet EM-206, bromsstyrningsreläet, ett kort för strömmätning, anslutning från Pt100-givare samt en säkring påmonterade.

Då Siemens LOGO! 12/24 RCE endast har reläutgångar som slits efter ett visst (okänt) antal cykler rekommenderas att PLC:n byggs på med modul LOGO! DM16 24 (inte DM16 24R) då denna har transistorutgångar som bör klara ett mycket högre cykelantal. Det vill säga att när huvudmodulens reläutgångar har slitits bör PLC:n utökas med ytterligare en modul DM 16 24 istället för att den byts mot en ny likadan basmodul.

Styrningen kan utökas så att mer av den kan ske via LOGO! TD. Vid tillräckligt många test blir det till slut för mycket kopplingsarbete med knappar, samt att knapparna blir väldigt många. I det hänseendet är LOGO! TD en bättre lösning. Problemet med att utöka styrningen med TD är att ett program måste skrivas för det, och textmeddelanden kan inte användas inuti UDF-block. Detta medför att hela programmet som hanterar TD kommer att synas i huvudporgrammet som därför blir svåröverskådligt.

Eftersom en av utgångarna på den analoga modulen AM2 AQ i nuläget används för att skicka ut en konstant 5 V-signal skulle en spänningsregulator kunna läggas till och utgången således användas till något som skall styras med en varierande spänning.

Dessutom rekommenderas att en av strömtransformatorerna byts ut mot en som är gjord för 1 A då den nuvarande 5 A-versionen ger för liten spänningsvariation vid de strömmar som föreligger. PLC:n har ju en upplösning på 10 mV.

Kapacitiva givare kan användas istället för de induktiva ifall delar av plast behöver detekteras. I annat fall tycks de induktiva fungera alldeles utmärkt.

REFERENSER

1. Wikipedia, en.wikipedia.org/wiki/Actuator, acc. 2013-03-28.

2. Produktinformationssida SKF Actuation Systems, www.skf.com/group/products/actuation-systems/linear-actuators/cat-series/index.html, acc. 2013-03-28.

3. Fargo Controls, www.fargocontrols.com/sensors/inductive_op.html, acc. 2013-05-16.

5. Produktinformationssida SKF Condition Monitoring, www.skf.com/group/products/condition-monitoring/index.html?alias=www.skf.comfcm, acc. 2013-04-16.

6. Produktdatablad SKF, www.skf.com/medialibrary/asset/0901d1968009e1db, acc. 2013-04-16.

7. Försäljningsbroschyr Siemens LOGO!, www1.elfa.se/data1/wwwroot/assets/datasheets/LOGO-V7_eng_bro.pdf, acc. 2013-05-22.

17. Testdokumentation, test order no 2013LT0001.

20. Testdokumentation, test order no 2013LT0004.

22. Produktinformationssida SKF Beyond Zero, www.beyondzero.com, acc. 2013-06-03.

23. Produktdatablad Electromen, www.electromen.com/pdf/EN_em-206-48.pdf, acc. 2013-03-28.

26. National Instruments, sine.ni.com/nips/cds/view/p/lang/sv/nid/210005, acc. 2013-03-28.

27. National Instruments, sine.ni.com/np/app/main/p/ap/global/lang/sv/pg/1/ps/30/sn/n25:device,n24:cRIO/, acc. 2013-05-20.

28. Prisinformation ELFA, www.elfa.se/elfa3~se_sv/elfa/init.do?item=25-708-78&toc=0&q=siemens+LOGO%21, acc. 2013-05-20.

29. www.ni.com/labview/buy/, acc. 2013-05-20.

30. Prisinformation ELFA, www.elfa.se/elfa3~se_sv/elfa/init.do?item=25-706-81&toc=0&q=logo+soft+comfort, acc. 2013-05-20.

31. Produktinformation SIMATIC, www.automation.siemens.com/mcms/simatic-controller-software/en/programming-options/Pages/Default.aspx, acc. 2013-05-02.

32. Försäljningsbroschyr SIMATIC, www.automation.siemens.com/salesmaterial-as/brochure/en/brochure_simatic-industrial-software_en.pdf, acc. 2013-06-04.

34. Användarmanual LOGO!, www1.elfa.se/data1/wwwroot/assets/datasheets/LOGOv7_eng_man.pdf, sid. 337, acc. 2013-05-22.

35. Produktdatablad LEM, www1.elfa.se/data1/wwwroot/assets/datasheets/HX0350PSP2_eng_datasheet.pdf, acc. 2013-05-22.

Personreferenser

4. Kjell Hermansson, testingenjör vid SKF Actuators AB

10. Pär Högberg, maskinkonstruktionsingenjör vid SKF Actuators AB

11. Jan Lorentzon, områdeschef vid SKF Actuators AB

18. Peter Hansson, maskinkonstruktionsingenjör vid SKF Actuators AB

24. Mikael Holgerson, chef vid SKF Test Centre Göteborg

BILAGOR

Bilaga A – Produktutvecklingsverktyg

Nedan beskrivs de verktyg som använts för att ta fram och jämföra olika lösningsförslag.

A.1 Kravspecifikation

En kravspecifikation är en tabell som listar egenskaper och funktioner som en produkt skall ha för att lösa ett bestämt problem. Egenskaperna/funktionerna listas i den första kolumnen av tabellen. I den andra kolumnen anges om funktionen eller egenskapen är ett krav eller ett önskemål (K/Ö) – en produkt som uppfyller kravspecifikationen skall också uppfylla alla krav som finns i den, men behöver inte uppfylla alla önskemål. I den tredje kolumnen anges viktningen av funktionen/egenskapen; viktningen används i ett senare skede då olika lösningsförslag skall vägas mot varandra. Nedan följer ett mycket enkelt exempel på en kravspecifikation (Tabell A). Märk väl att detta exempel inte är uppställt på ett generellt sätt vilket en kravspecifikation bör vara för att inte låsa läsaren vid en specifik lösning (exempelvis hade en bättre rubrik kanske varit ”Kravspecifikation: hjälpmedel för icke-verbal kommunikation”).

Kravspecifikation: kulspetspenna	**Krav/Önskemål**	**Viktning**
Skall gå att skriva med	Krav	5
Skall tåla att tappas från 75 centimeters höjd	Önskemål	3
Skall tåla att tappas från 180 centimeters höjd	Önskemål	2
Skall ha bläckfärg som är lätt att läsa	Önskemål	4

Tabell A. Exempel på en kravpsecifikation.

Givetvis är detta bara ett exempel på hur en kravspecifikation kan ställas upp. Fler kolumner skulle kunna läggas till som exempelvis beskriver hur man skall mäta ifall funktionen eller egenskapen ifråga uppnåtts eller kolumner som skall fyllas i efter att produkten tagits fram för att någon form av återkoppling skall fås.

A.2 Pugh-matris

En Pugh-matris är likaså en tabell, men som baserar sig på kravspecifikationen. När några lösningskoncept tagits fram används Pugh-matrisen som ett verktyg för att se hur väl de uppfyller kraven i kravspecifikationen jämfört med varandra. Kraven ur kravspecifikationen återfinnes i första kolumnen, i andra kolumnen finns en av lösningarna som är vald som referens – den har således 0 i poängsättning för alla kraven. I tredje kolumnen återfinnes det andra lösningsförslaget, i fjärde återfinnes det tredje förslaget och så vidare. Under respektive lösningsförslag ifylles antingen ett positivt värde eller ett negativt beroende på hur lösningen står sig jämfört med referenslösningen i första kolumnen.

Nedan kan ses ett exempel på en Pugh-matris som baserar sig på kravspecifikationen i tabell A;

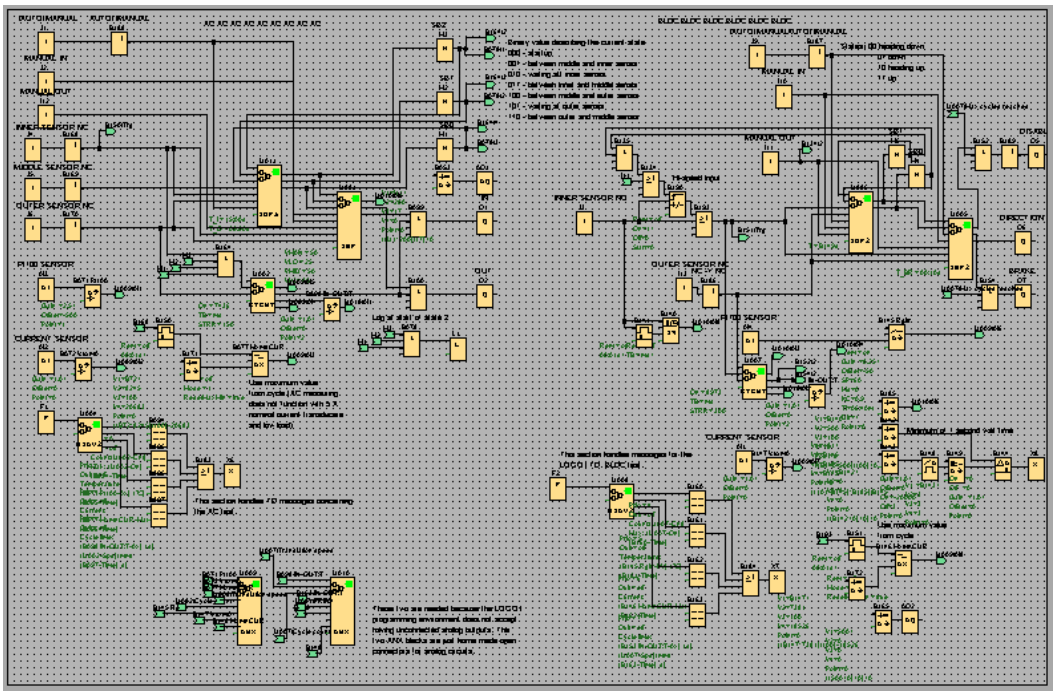
Pugh-matris: kulspetspenna	**Krav / Önskemål**	**Viktning**	**Penna av plast, rött bläck** (ref.)	**Penna av glas, rött bläck**	**Penna av plast, blått bläck**
Skall gå att skriva med	K	5	0	0	0
Skall tåla att tappas från 75 centimeters höjd	Ö	3	0	-1	0
Skall tåla att tappas från 180 centimeters höjd	Ö	2	0	-2	0
Skall ha bläckfärg som är lätt att läsa	Ö	4	0	0	+1
Sammanvägning	-	-	0	-7	+4

Tabell B. Exempel på en Pugh-matris.

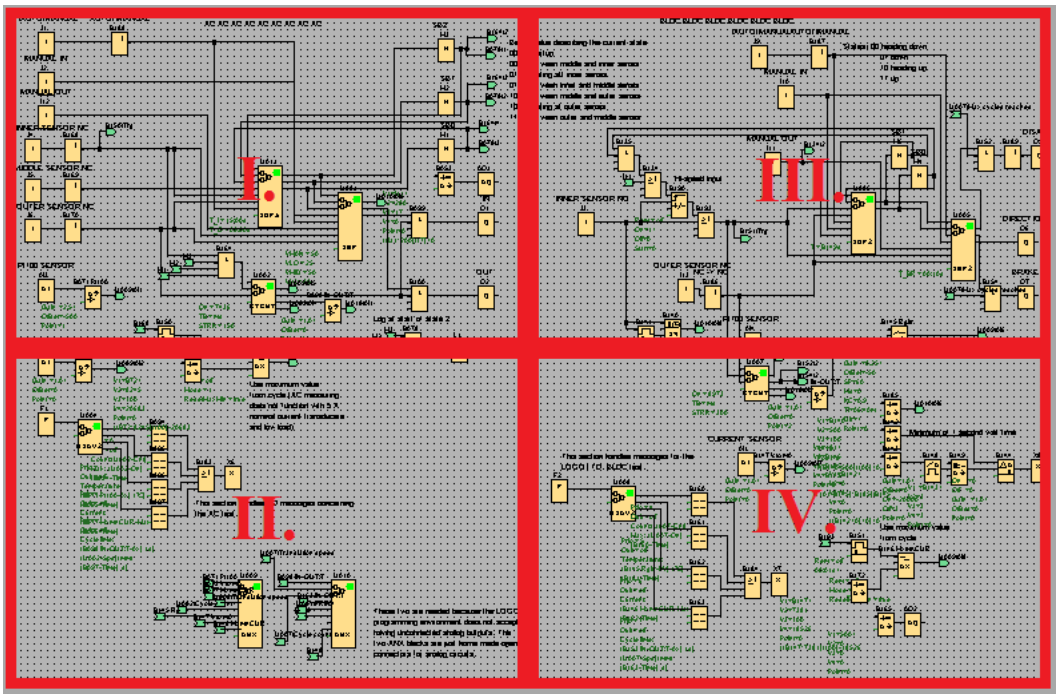
Raden i botten fås fram genom multiplikation av viktningen med den individuella lösningens poängsättning för respektive krav, varefter poängen summeras. Som kan ses ovan uppfylls kraven bäst av det tredje lösningsförslaget – en penna av plast med blått bläck.

Bilaga B – Slutligt PLC-program

Bilden nedan (FigurA) är en utzoomad bild på hela det slutliga PLC-programmet.

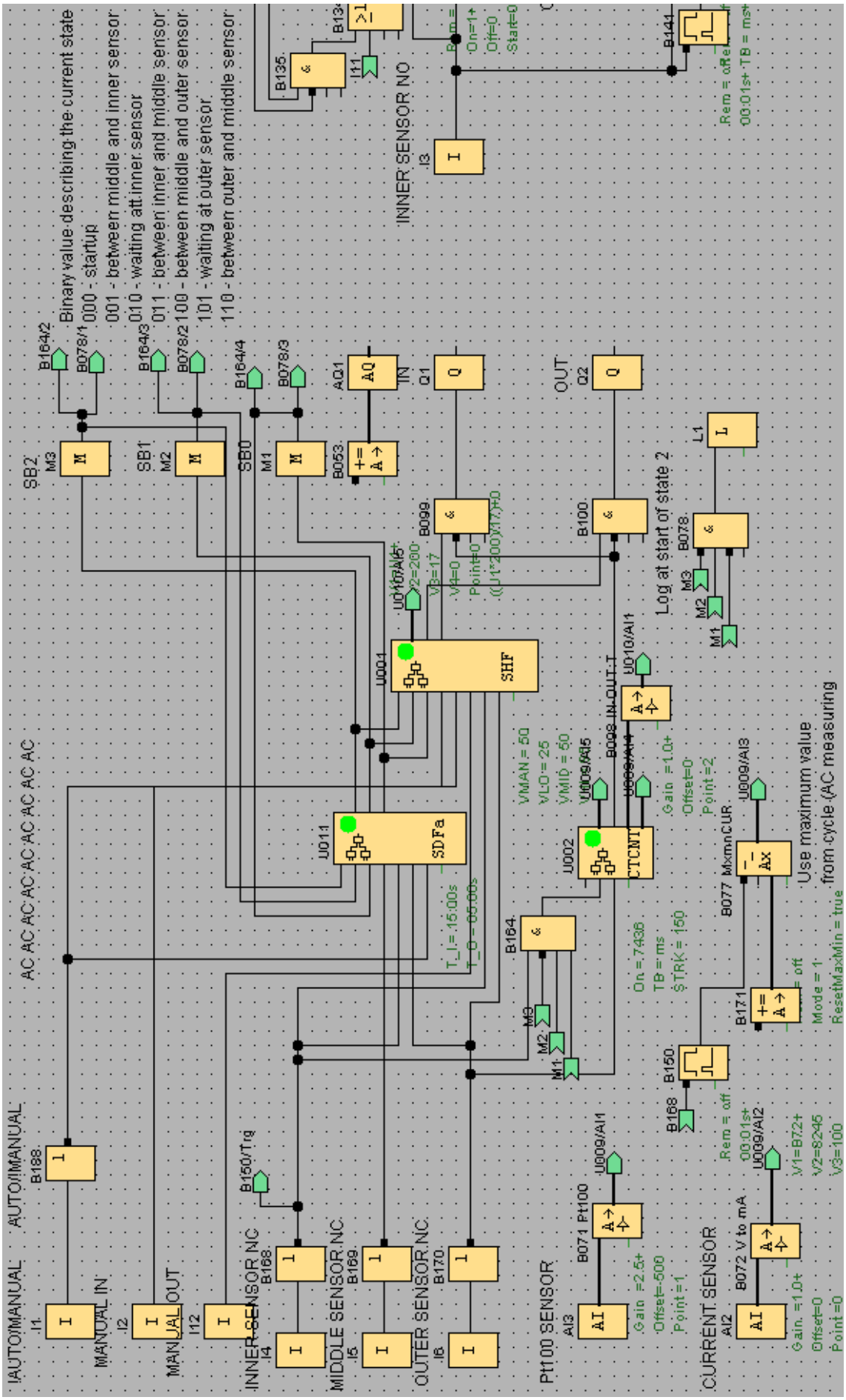


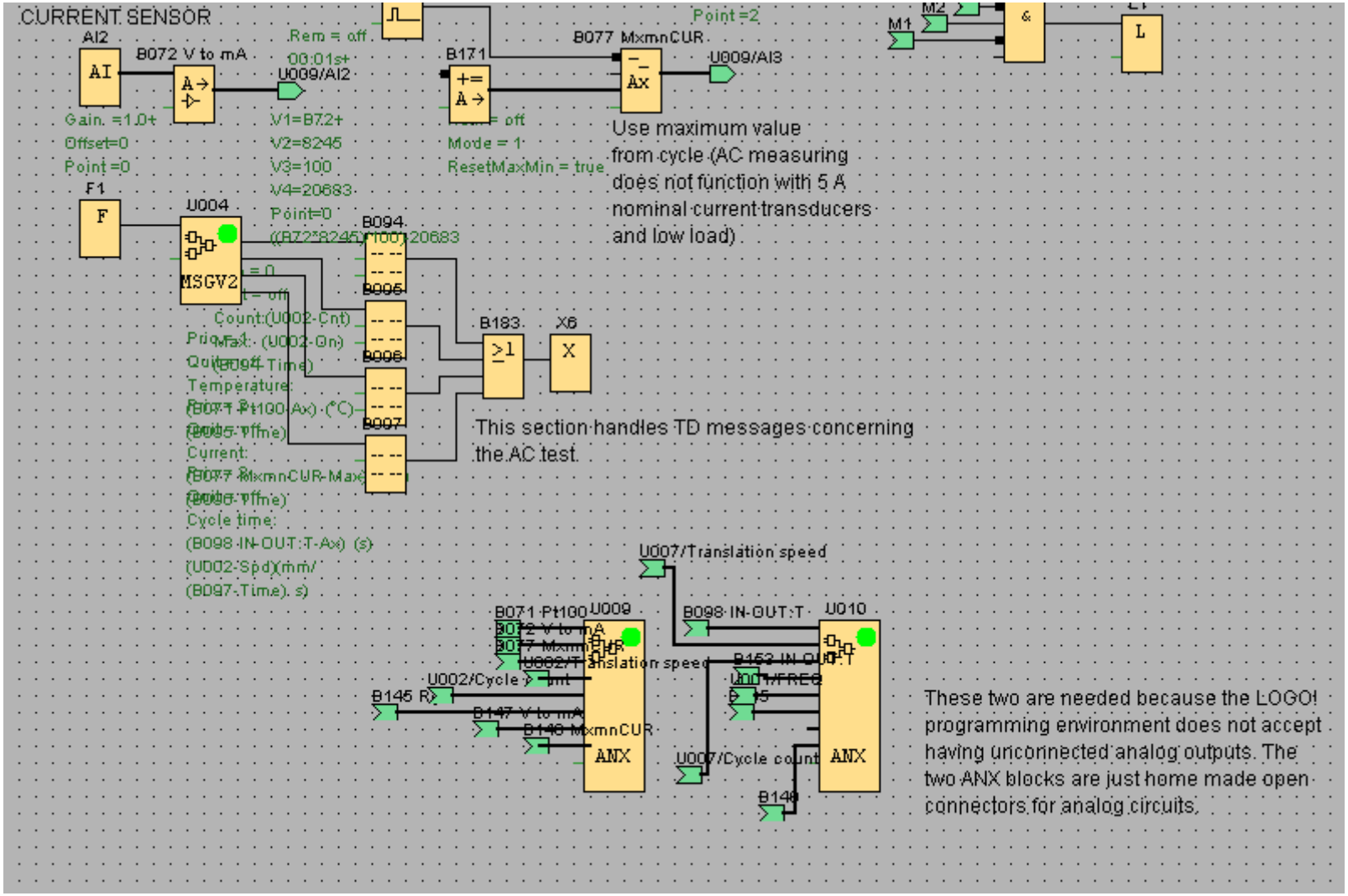
Figur A. Det slutliga PLC-programmet.

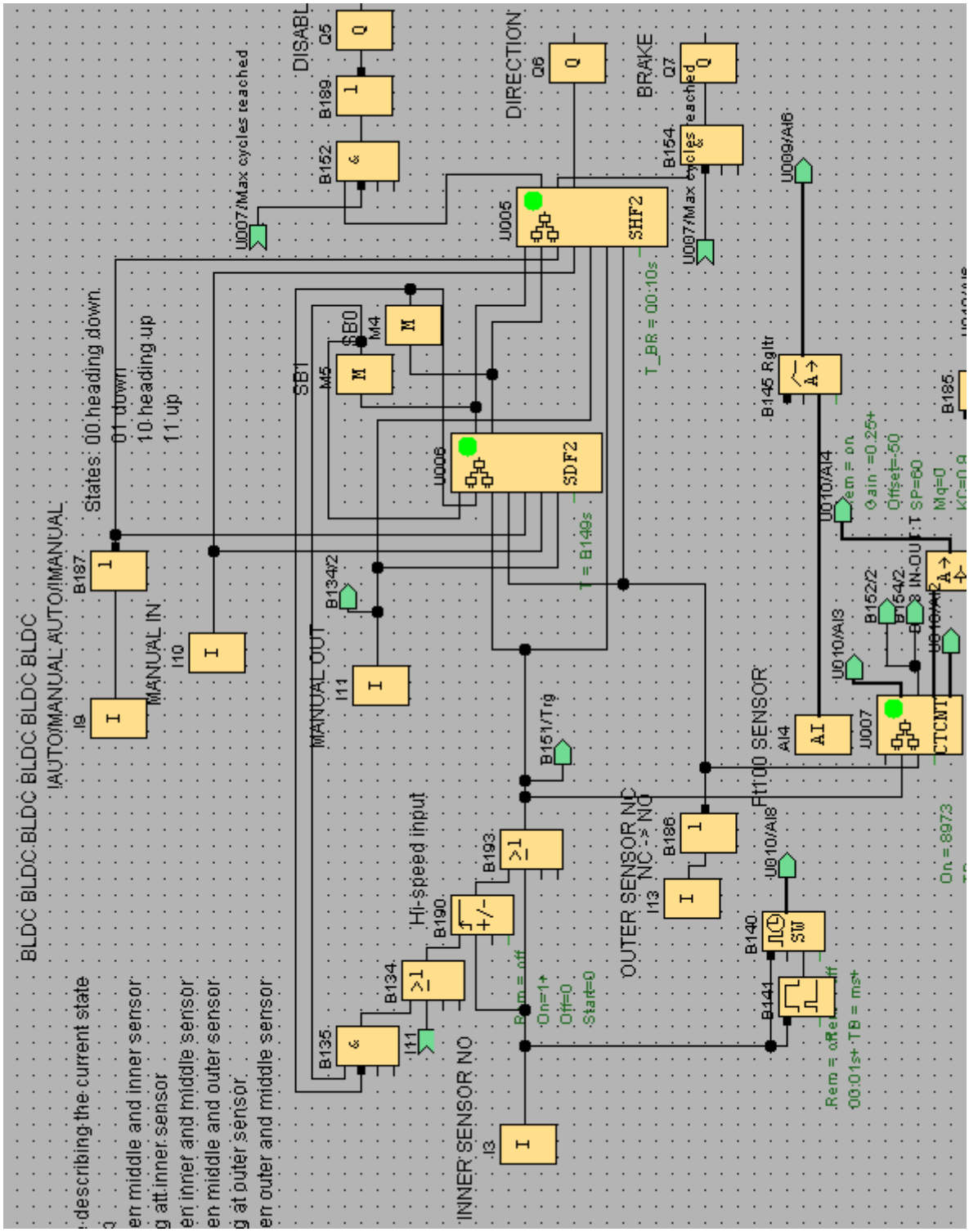


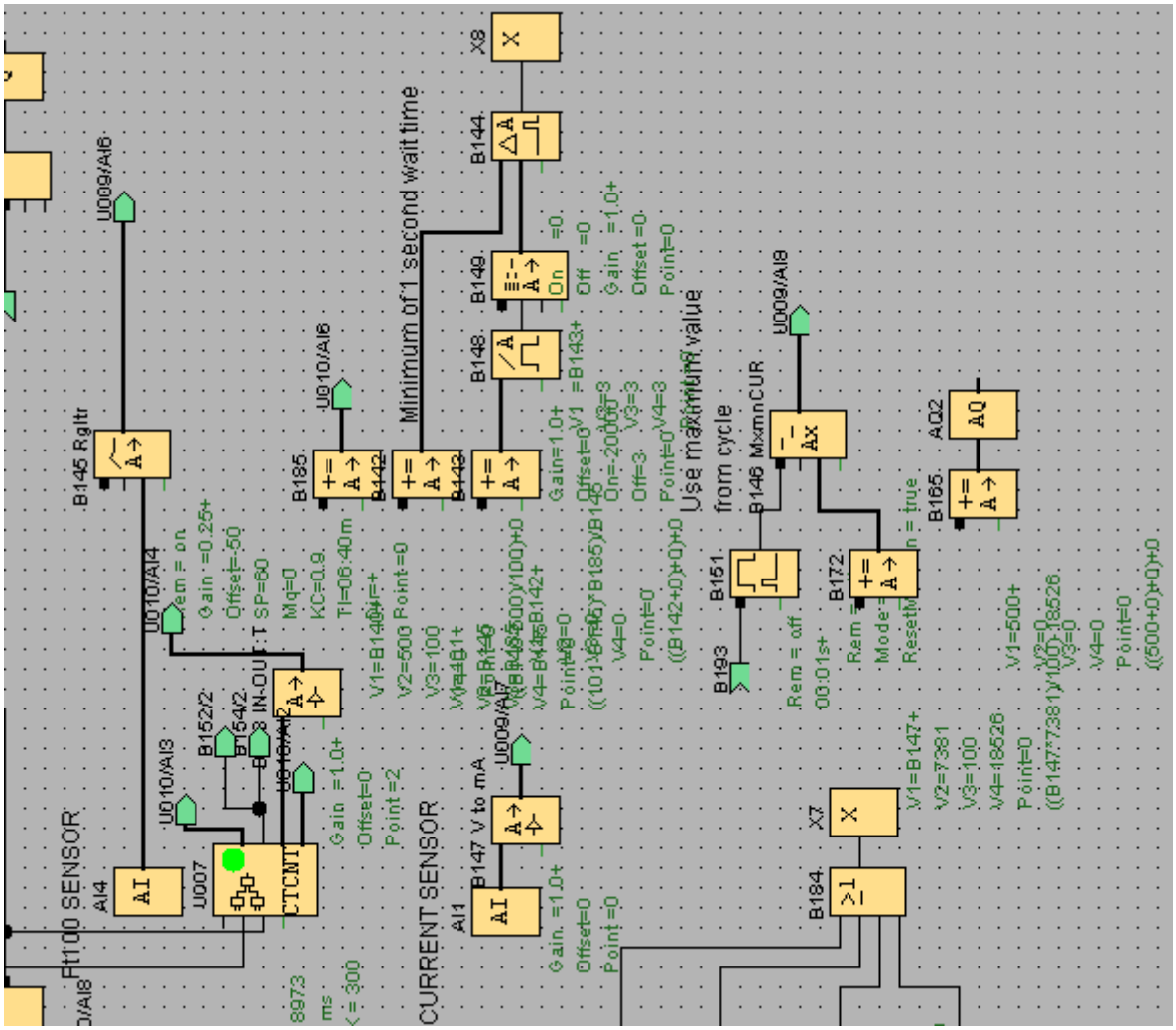
Figur B. Beskrivning av hur det slutliga PLC-programmet delats upp i denna bilaga. Bilderna återfinns i mer detalj på sidorna som följer.

Sektion I. och lite av sektion II. i programmet sköter styrningen av BLDC-donet. De resterande två sektionerna sköter styrningen av AC-donet.


AUTO/MANUAL
MANUAL IN
MANUAL OUT
INNER SENSOR NC
MIDDLE SENSOR NC
OUTER SENSOR NC
Pt100 SENSOR
CURRENT SENSOR
INNER SENSOR NO
Binary value describing the current state
000 - startup
001 - between middle and inner sensor
010 - waiting at inner sensor
011 - between inner and middle sensor
100 - between middle and outer sensor
101 - waiting at outer sensor
110 - between outer and middle sensor
AC AC AC AC AC AC AC AC AC AC
SDFa
SHF
CTCNT
Log at start of state 2
Use maximum value
from cycle (AC measuring

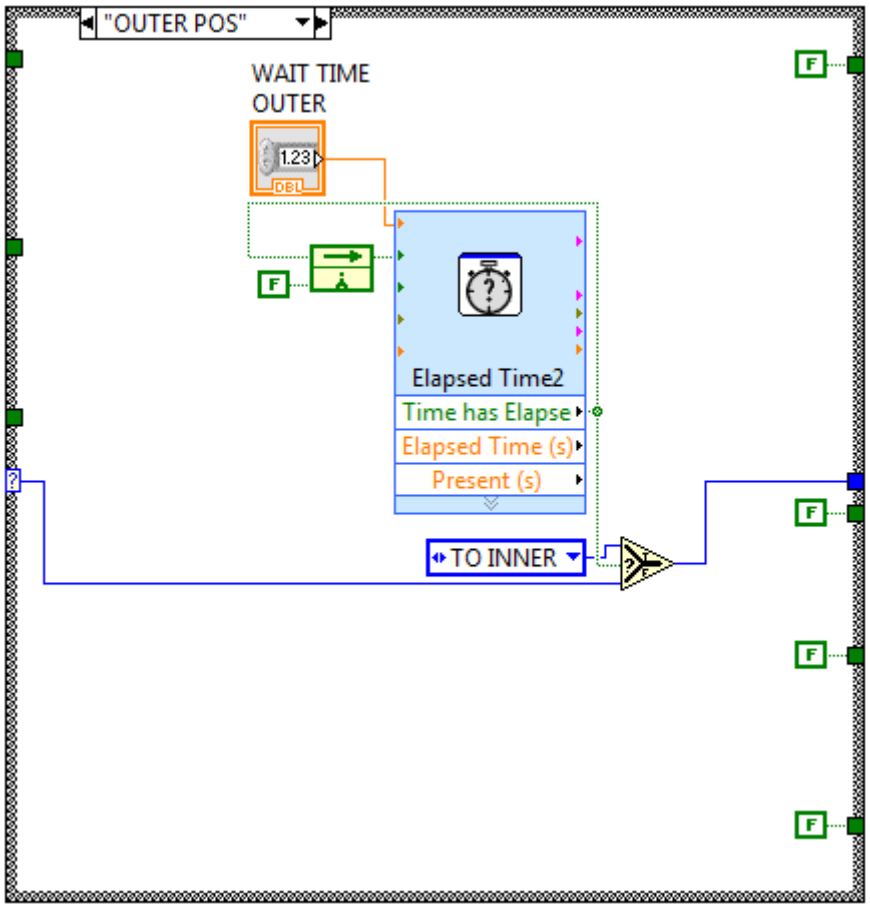

CURRENT SENSOR
AI2
AI
B072 V to mA
A→
Gain. =1.0+
Offset=0
Point =0
Rem = off
00:01s+
U009/AI2
V1=B72+
V2=8245
V3=100
V4=20683
Point=0
B171
+=
A→
= off
Mode = 1
ResetMaxMin = true
B077 MxmnCUR
Ax
Point =2
U009/AI3
Use maximum value from cycle (AC measuring does not function with 5 A nominal current transducers and low load)
M1
M2
&
L
F1
F
U004
MSGV2
B094
B095
B096
B097
Count (U002-Cnt)
Prio (U002-On)
Quit (B094-Time)
Temperature (B071-Pt100-Ax) (°C)
Current (B077-MxmnCUR-Max)
Cycle time (B098-IN-OUT:T-Ax) (s)
(U002-Spd)(mm/(B097-Time). s)
B183
≥1
X6
X
This section handles TD messages concerning the AC test.
U007/Translation speed
B071 Pt100
B072 V to mA
B077 MxmnCUR
U002/Translation speed
U009
U002/Cycle
B145 R
B147 V to mA
B148 MxmnCUR
ANX
B098 IN-OUT:T
U010
B153 IN-OUT:T
U007/Cycle count
B148
These two are needed because the LOGO! programming environment does not accept having unconnected analog outputs. The two ANX blocks are just home made open connectors for analog circuits.


BLDC BLDC BLDC BLDC BLDC BLDC BLDC BLDC
!AUTO!MANUAL AUTO!MANUAL
States: 00.heading down
01.down
10.heading up
11.up
MANUAL IN
I10
MANUAL_OUT
I11
SB1
M5
SB0
M4
U007/Max cycles reached
DISABL
DIRECTION
BRAKE
U005
SHF2
U006
SDF2
T_BR = 00:10s
T = B149s
B187
B189.
B152.
B154.
Q5
Q6
Q7
B145 Rgltr
U009/AI6
U010/AI4
Gain =0.25+
Offset=-50
T SP=60
Mq=0
On=8973
B185.
U010/AI3
B152/2
B154/2
U007
CTCNT
AI
AI4
U010/AI8
Ft100 SENSOR
OUTER SENSOR NC
NC -> NO
I13
B186.
B151/Tig
B193.
B190.
Hi-speed input
B134.
B135.
I11
INNER SENSOR NO
I3
B140.
B141.
SW
Rem = off
On=1+
Off=0
Start=0
00:01s+TB = ms+
describing the current state
en middle and inner sensor
g at inner sensor
en inner and middle sensor
en middle and outer sensor
g at outer sensor
en outer and middle sensor


Pt100 SENSOR
AI4
U007
U010/AI8
U010/AI4
Rem = on
Gain =0.25
Offset=-50
B152/2
B154/2
CTCMT
Gain =1.0
Offset=0
Point=2
8973
ms
<= 300
B145 RgItr
U009/AI6
B185
U010/AI6
Mq=0
KC=0.9
TI=06:40m
V1=B142
V2=500 Point=0
Minimum of 1 second wait time
CURRENT SENSOR
AI1
B147 V to mA
U009/AI7
Gain =1.0
Offset=0
Point=0
V3=100
V4=B142
B142
B143
B148
B149
B144
X8
Gain =1.0
Offset=0
Point=0
V1 =B143
Off =0
On =0.
Gain =1.0
Offset=0
Point=0
Use maximum value
from cycle
B193
B151
B146 MxmnCUR
U009/AI8
B184
X7
B172
B165
AQ2
V1=B147
V2=7381
V3=100
V4=18526
Point=0
V1=500
V3=0
V4=0
Point=0

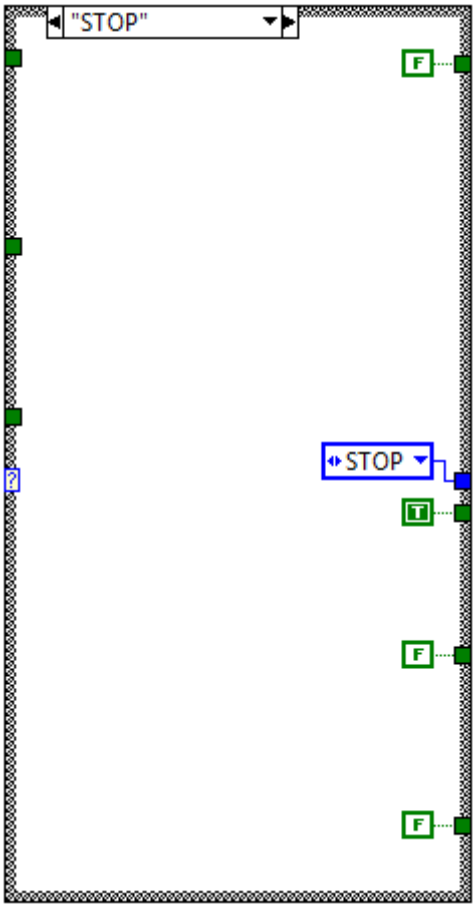
Bilaga C – enkelt LabVIEW-program, återstående tillstånd

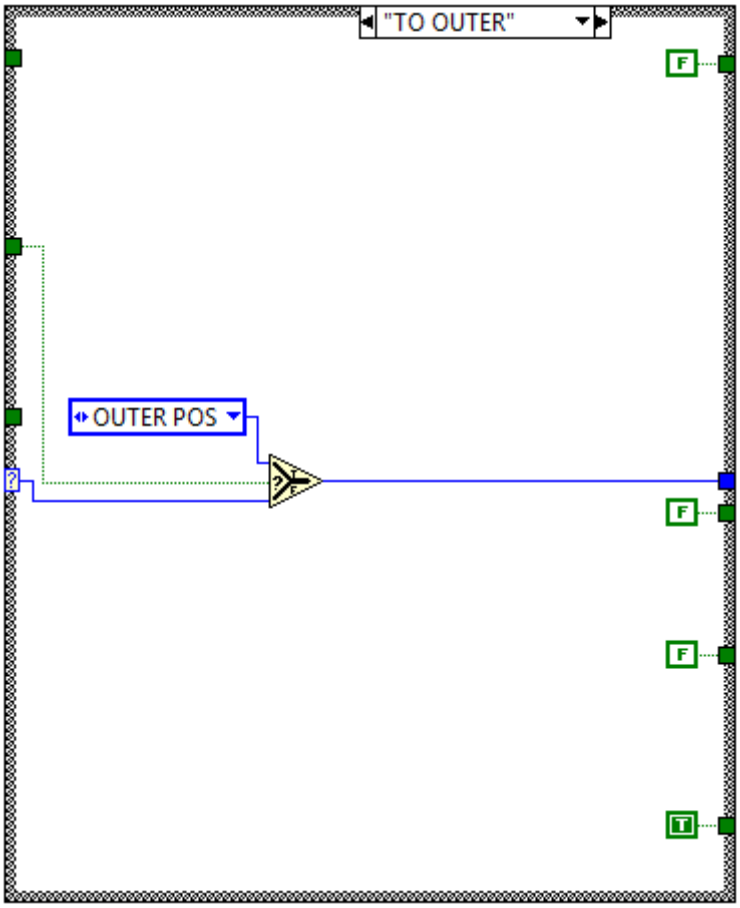
Denna bilaga innehåller de återstående tillstånden ur det tillståndsbaserade LabVIEW-programmet, utöver de som redan tagits med i huvudtexten.



Figur A. Tillståndet "OUTER POS" i det tillståndsbaserade LabVIEW-programmet.

Figur B. Tillståndet "STOP" i det tillståndsbaserade LabVIEW-programmet.

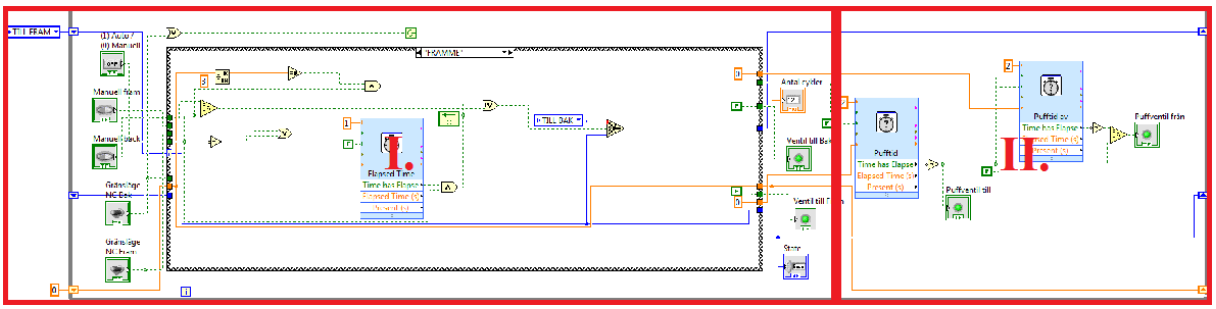




Figur C. Tillståndet "TO OUTER" i det tillståndsbaserade LabVIEW-programmet.

Bilaga D – invecklat LabVIEW-program, fullständigt

Denna bilaga beskriver det program som togs fram i LabVIEW som en kopia av det LOGO! PLC-program som används för att styra testet av tätningar med koldamm, det vill säga att funtionalitet för växling mellan automatisk och manuell styrning har lagts in samt att koldammet blåses runt var fjärde körcykel. Till att börja med delas programmet upp enligt följande bild.

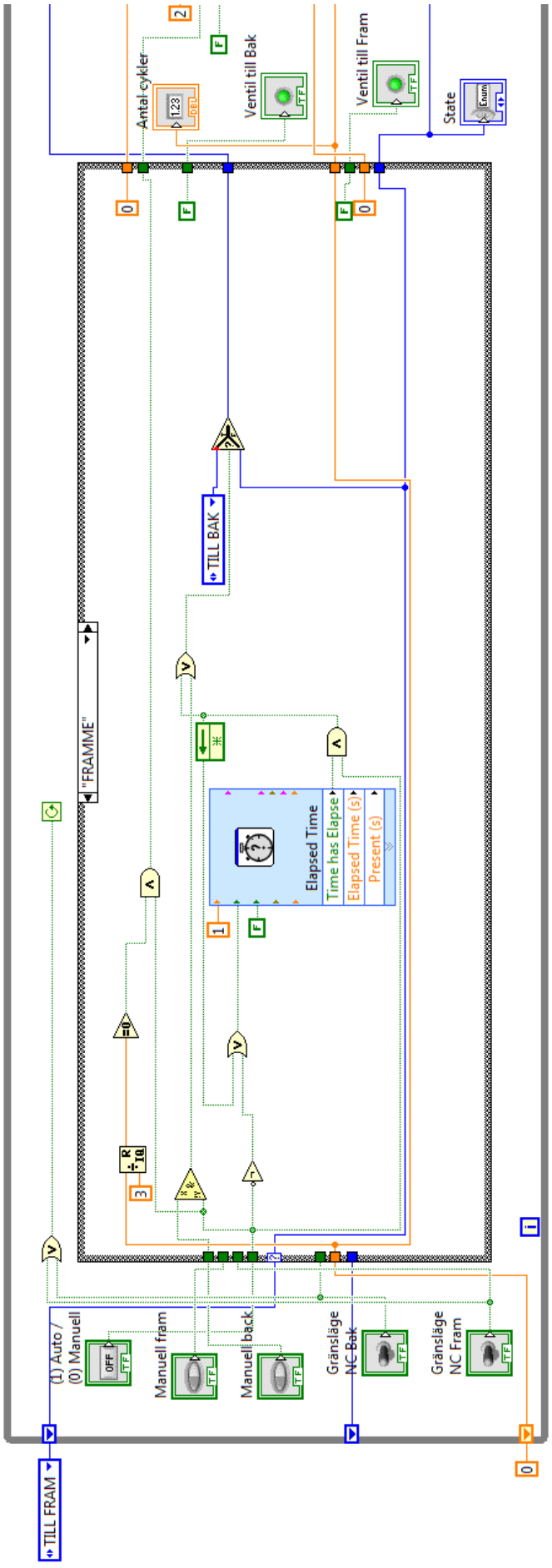


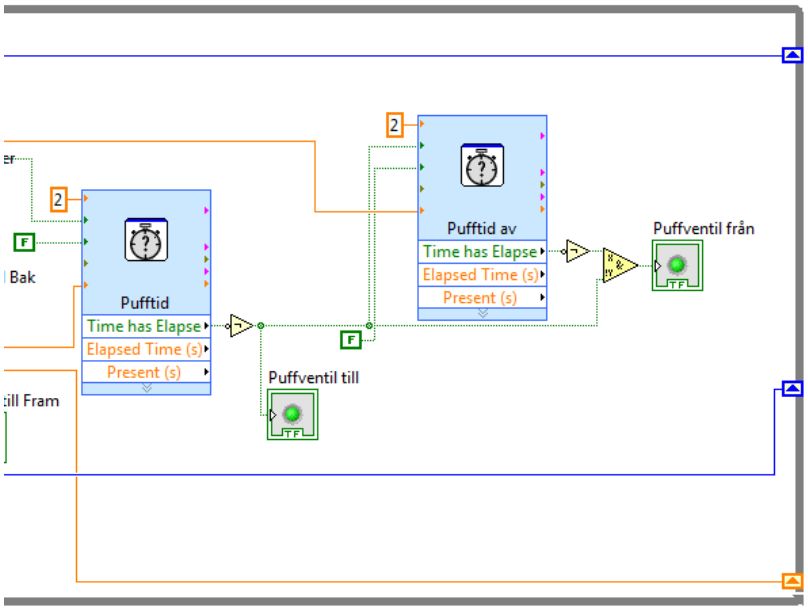
Figur A. Förteckning av hur LabVIEW-programmet delats upp i denna bilaga.

Vidare finns bilder på de olika tillstånden i case-satsen efter de två bilder (B och C) som beskriver while-loopen. Se figur D till G. Eftersom tillståndet ”FRAMME” finns med i figur B visas det inte i en separat bild.

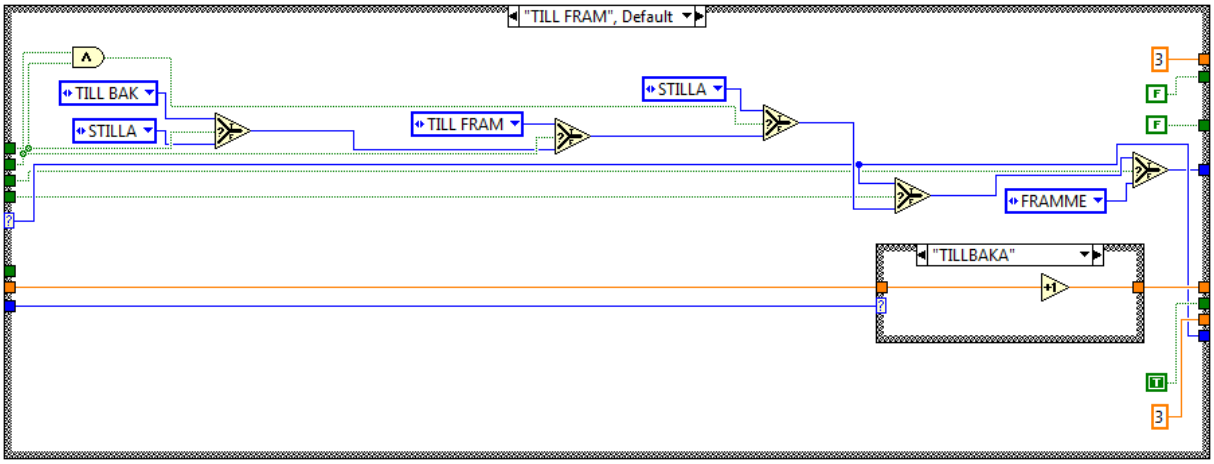
Den första av de två bilderna enligt förteckningen ovan återfinns på nästa sida.

Figur B. Del I av det invecklade LabVIEW-programmet.

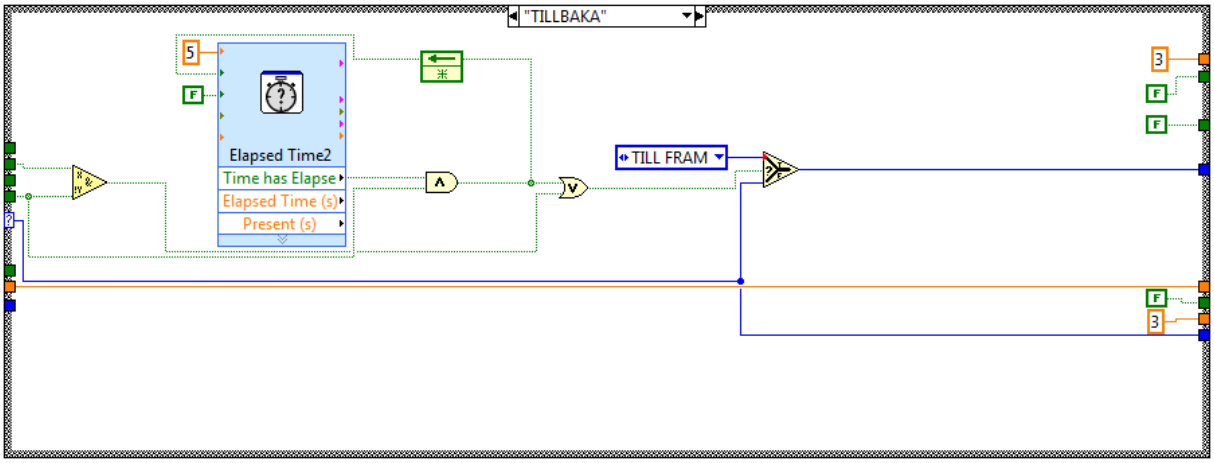




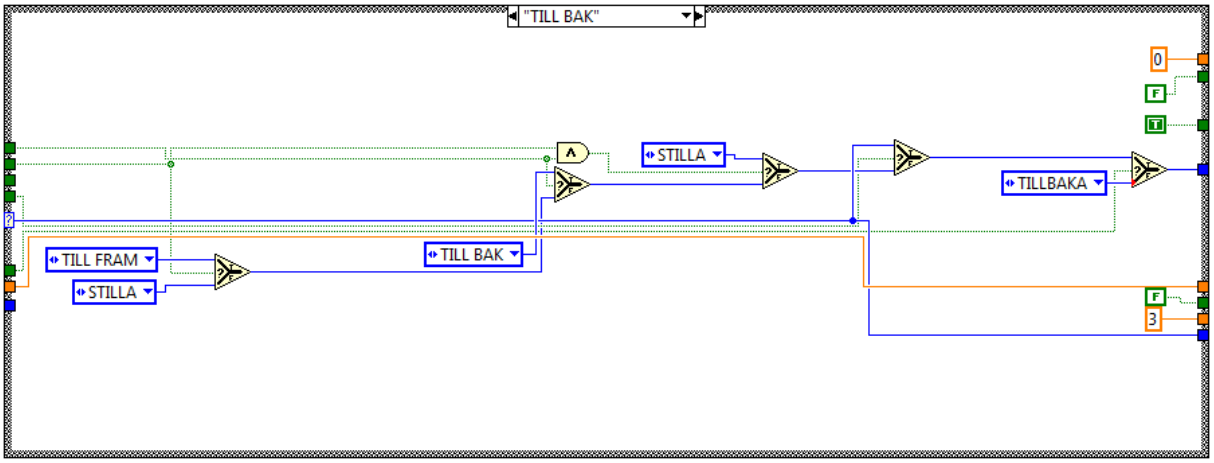
Figur C. Del II av det invecklade LabVIEW-programmet.



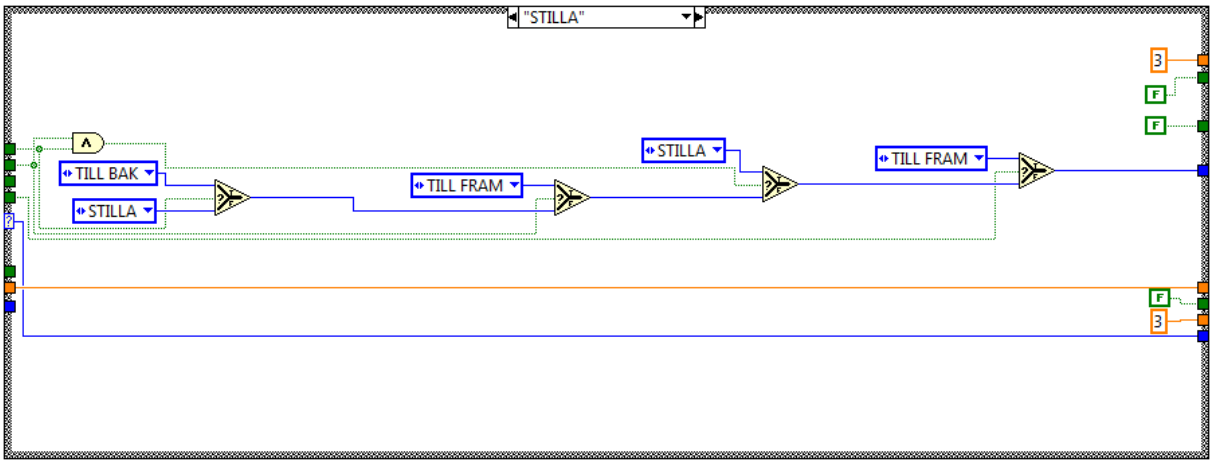
Figur E. Tillståndet ”TILL FRAM” i det invecklade LabVIEW-programmet.



Figur D. Tillståndet ”TILLBAKA” i det invecklade LabVIEW-programmet.



Figur F. Tillståndet ”TILL BAK” i det invecklade LabVIEW-programmet.



Figur G. Tillståndet ”STILLA” i det invecklade LabVIEW-programmet.