CODEN:LUTEDX/(TEIE-3047)/1-137/(2015)

Industrial Electrical Engineering and Automation

Realisering av PLC-system och simulering med mikrokontroller

Marcus Andersson
Marcus Andersson

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

Realisering av PLC-system och simulering med mikrokontroller

LUNDS UNIVERSITET
Campus Helsingborg

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

Examensarbete:
Marcus Andersson
Marcus Andersson

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

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

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

Sammanfattning

Examensarbetet syftade till att bygga om en industriell press för centrumtegel på Höganäs Bjuf AB, vilket skulle leda till att öka säkerheten för drift- och underhållspersonal. För att öka säkerheten togs ett nytt styrsystem fram bestående av en PLC-baserad styrlogik tillsammans med ett HMI. Det nya styrsystemet kommer att höja arbetssäkerheten för underhållspersonalen då spänningen ute på pressen kommer sänkas till 24VDC. Samtidigt medför det nya styrsystemet att nya säkerhetsåtgärder så som ett tvåhandsgrepp kommer att införas.

För att testa funktionaliteten hos styrsystemet implementerades en simuleringsmiljö i form av en mikrokontroller av modellen Arduino Mega 2560. Denna programmerades så att den återspeglade tillverkningsprocessen på ett realistiskt sätt.

I rapporten beskrivs de mjukvaror vi har jobbat med för att ta fram PLC programmet, HMI och simuleringsmiljön. Mjukvaror som använts är GxWorks2, iX Developer och Atmel studio.

Nyckelord: PLC, HMI, GxWorks2, iX Developer, Arduino, Atmel Studio.

Abstract

The purpose of this thesis with the conversion of the industrial press for centrumbrick on Höganäs Bjuf AB was to improve the security for the work- and maintenance personnel. To improve the security a new control system containing a PLC based control logic and HMI was developed. The new control system will improve the work security for the maintenance personal because the operating voltage for sensors on the machine will be lowered to 24VDC. At the same time the new control system allows new safety precautions as a two hand control to be implemented.
To test the functionality of the new control system a simulation environment was developed. The simulation environment was implemented on a microcontroller of the model Arduino Mega 2560. This microcontroller was programed to reflect the manufacturing process in a realistic way.
All the software that has been used to develop the PLC program, HMI and the simulation environment is described in the report. Software that has been used is GxWorks2, iX Developer and Atmel studio.

Keywords: PLC, HMI, GxWorks2, iX Developer, Arduino, Atmel studio.

Förord

Vi vill tacka Höganäs Bjuf AB för möjligheten att få göra vårt examensarbete hos dem. Examensarbetet har gett oss en stor insikt i hur automationsarbete i dagens industri kan se ut.
Ett extra tack riktas till Per Andersson, Bernard Andersson och underhållspersonalen på företaget som hjälpt oss att bolla idéer och stöttat oss då vi stött på problem under arbetets gång.

Vi tackar även vår examinator Mats Lilja och vår handledare Johan Björnstedt som har gett oss goda råd och hänvisningar under projektets gång.

Terminologi

COM-port	Communication port, serieport för överföring av seriell kommunikation
CPU	Central Processing Unit, centralprocessorn som utför beräkningar(i PLC:n)
CTC	Clear Timer on Compare, funktion för generering av avbrott
HMI	Human Machine Interface, maskin-människa gränssnitt för styrning och övervakning
IDE	Integrate Development Environment, utvecklingsmiljö för programmering
IEC	International Electrotechnical Comission, kommission med största syfte att ta fram internationella standarder
I/O	Input/Output, Ingång/Utgång
OP	Operationsförstärkare
PLC	Programmable Logic Controller, programmerbar logisk styrenhet
POU	Program Organisation Unit, del i programmet för PLC
PWN	Pulse Width Modulation, pulsbreddsmodulering
VAC	Volt Alterning Current, växelspänning
VDC	Volt Direct Current, likspänning

Innehållsförteckning

1 Inledning

1.1 Bakgrund

Höganäs Bjuf AB har anor ända från 1797 då Höganäs stenkolsverk bildades. 1797 var man verksam inom gruvindustrin och bröt kol i nordvästra Skåne. År 1825 startade koncernen sin första produktion av eldfast tegel. Det dröjde sedan ända fram till 1876 innan den första anläggningen för produktion av eldfast tegel startade i Bjuv.
1988 bildas Höganäs Bjuf AB som ett fristående bolag då man tidigare bestått av sammanslagningar av mindre bolag och ingått i Höganäs koncernen. 10 år senare köptes man upp av den norska koncernen Borgestad Group som har flera produktionsanläggningar i Europa. 2001 omstrukturerades verksamhet för eldfast tegel inom koncernen och produktionen i Bjuv var den fabrik som behölls för tillverkning, alla andra lades ner.

För produktionen av eldfast tegel på Höganäs Bjuf AB används industriella pressar. Maskinen som kommer behandlas i detta examensarbete är en hydraulisk press som utsätter lera för högt tryck. Sedan förflyttas den pressade leran för förvaring. Slutprodukten är ett centrumtegel som används för tillverkning av stål.

Inom dagens industri ställs det allt större krav på säkerhet. Man vill minska kostnaderna av sjukskrivningar för personal med skador relaterade till arbetet. Som pressen av centrumtegel ser ut idag så finns det brister i säkerheten. För underhållspersonal då maskinen drivs med hög växelspänning ända ut på givarnivå, men även för driftpersonalen då det saknas många säkerhetsåtgärder för t.ex. klämrisk.

Man vill också öka effektiviteten för produktionen och byta ut gamla och förslitna komponenter. Detta uppnås genom att byta ut det befintliga styrsystem som finns idag mot ett styrsystem som är uppbyggt av en mikroprocessorbaserad styrenhet, en så kallad Programmable Logic Controller(PLC). Till detta ska en operatörspanel i form av ett Human-Machine-Interface (HMI) tas fram. HMI:et har ett grafiskt gränssnitt som gör det möjligt för operatören att styra och övervaka pressprocessen.

Det finns i detta projekt inte anledning att jobba med kommunikationsbussar då det är fysiskt möjligt att dra in alla ingångs- och utgångssignaler direkt till styrenheten.
Med hänsyn till dessa aspekter har Höganäs Bjuf AB beslutat att bygga om maskinen och byta ut styrsystemet. Efter ombyggnationen kommer spänningen på givarnivå vara 24 VDC. Höganäs Bjuf AB kommer bygga till säkerhetsanordningar för att öka säkerheten för driftpersonalen. PLC-styrningen kommer leda till mer kontinuerlig och repetitiv produktion med ökad effektivitet.

1.2 Syfte

Syftet med examensarbetet var att ta fram ett fungerande PLC-styrsystem med ett tillhörande HMI för styrning av maskinen som producerar centrumtegel. Styrsystemet skulle sedan köras mot en simuleringsmiljö. Simuleringen implementerades i form av en mikrokontroller som tog emot, behandlade och skickade tillbaka signaler som speglade processen.

1.3 Problemformulering

Examensarbetet går ut på att automatisera en industriell press och samtidigt höja säkerheten för operatören. Automatiseringen ska leda till att pressen blir mer tidseffektiv och ska vara lätt för operatören att köra. För att uppnå detta behövs följande göras:

- Analysera pressen i detalj för att skapa en förståelse för pressprocessen.
- Skapa flödesscheman som representerar hela processen.
- Programmera en PLC som styr processen efter det framtagna flödesschemat.
- Ta fram ett användarvänligt HMI som ska användas för att styra och övervaka pressen.
- Skapa en simuleringsmiljö som återspeglar den verkliga pressen och dess mekaniska begränsningar.
- Implementera nya säkerhetsanordningar på pressen för att höja säkerheten på utsatta delar enligt en riskbedömning som gjorts av Höganäs Bjuf.

1.4 Begränsningar

Examensarbetet motsvarar 22,5 högskolepoäng, vilket motsvarar ungefär 15 arbetsveckor i tid. Med hänsyn till detta begränsades arbetet till att ta fram PLC-styrsystemet med fungerande HMI och testning mot en simuleringsmiljö. Då inte alla delar till ombyggnationen fanns och att under arbetets gång lades projektet på is, kunde bara delar av PLC-styrningen testas. Av ekonomiska skäl begränsades simuleringsmiljön till delar av processen då denna finansierades av oss själva.

1.5 Källkritik

Den litteratur vi har använt oss av publicerades för många år sedan och tekniken inom automation och elektro har utvecklats mycket de senaste åren. Detta gör att en del av informationen inte stämmer överens med dagens teknik.

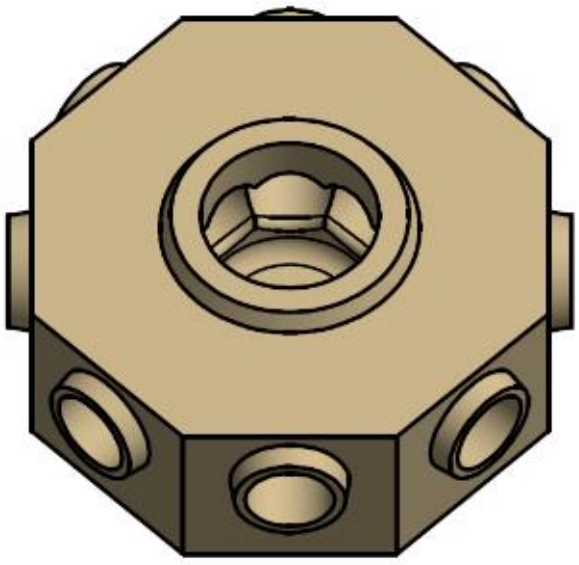
De guider och datablad som använts är skrivna av företagen bakom programmet eller hårdvaran. Vi anser att det är av största intresse för tillverkaren att den information som skrivs är trovärdig.

2 Beskrivning av pressen

I detta kapitel görs en djupare beskrivning av produkten som tillverkas i den maskin som examensarbetet behandlar och tillverkningsprocessen av densamma.

2.1 Produkten

Produkten är en centrumtegelsten som tillsammans med fler tegelstenar används som gjutform vid tillverkning av stålstänger. Produkten tillverkas med en metod som kallas för våtpressning. Det betyder att lera som innehåller 12-14% fuktighet utsätts för högt tryck i en form. Därefter förtorkas den pressade leran för att sedan brännas i en ugn. Leran benämns som ämne innan de pressas och som sten efter pressning. Först efter att stenen bränts i ugnen kan den kallas för tegelsten. Den färdiga tegelstenen har en oktogons form med ett hål på varje sida och ett hål på ovansidan.



Figur 1: Färdigpressad sten

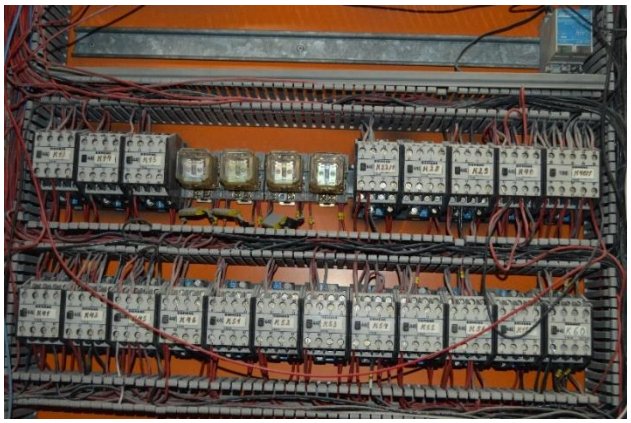
2.2 Tillverkningsprocessen idag

Maskinen som tillverkar centrumteglet är en hydraulisk press som byggdes år 1969.
För att tillverka teglet placeras ämnet manuellt vid en införararm. Denna arm drivs av pneumatiska kolvar som för ämnet in under pressen. När ämnet har nått positionen under pressoket sänks det ned i en form. I formen förs det sedan in inlägg från sidorna för att säkerställa den slutgiltiga formen av produkten. Med inläggen på plats förs pressoket ner och gör hålet på ovansidan av stenen. Med pressoket nere förs knivarna, vilka är ihåliga rör som gör hålen på sidorna av stenen in. När knivarna är i stenen pressar man ner pressoket med ett högre tryck för att få den slutgiltiga formen och hållfastheten på teglet. Alla pressrörelser sker med hydrauliska kolvar. Efter en viss tid av högt tryck återgår pressoket till sitt toppläge och knivar och inlägg går ur formen. Stenen skjuts upp ur formen för att sedan föras fram när nästa ämne placeras under pressoket. Den färdiga stenen förflyttas från pressen till brätten, som är en metallskiva som stenarna förvaras på efter pressning. Förflyttningen sker med en linjärenhet där en vakuumsug är monterad. På varje brätte får det plats två stycken stenar. När ett brätte blivit fullt trycks detta fram av tomma brätten som matas ut från ett magasin. De fyllda brättena åker sedan in i en elevator som rymmer två brätten i bredd. I elevatorn hissas sedan brättena med stenar upp för att förtorkas och sedan brännas.

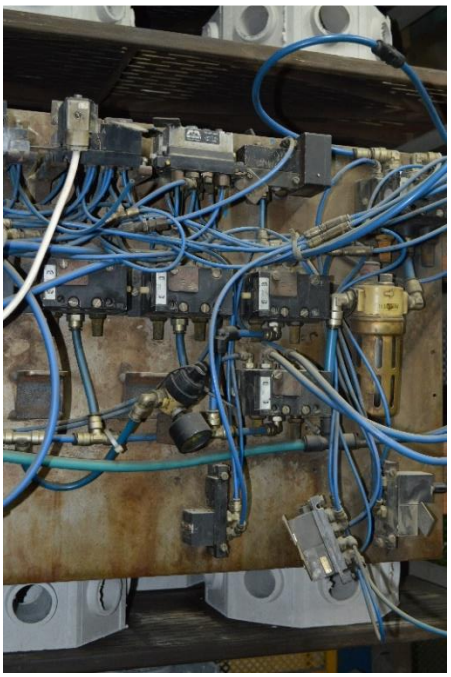
Figur 2: Hela maskinen

2.3 Nuvarande styrsystem

För att styra rörelser under processen används idag kontaktorer och tidreläer. Kontaktorerna får sin signal från givare ute på maskinen som drivs med 230 VAC eller av andra kontaktorer. Tidreläerna används för att säkerställa att man uppnår rätt tryck och för säkerställning av att bl.a. hydrauliska delar har kommit i rätt läge. Detta elektriska styrsystem är kompletterat med pneumatisk logik för att styra alla pneumatiskt styrda cylindrar.



Figur 3: Kontaktorer i styrskåp



Figur 4: Pneumatiska ventiler och logik

2.4 Kommande styrsystem

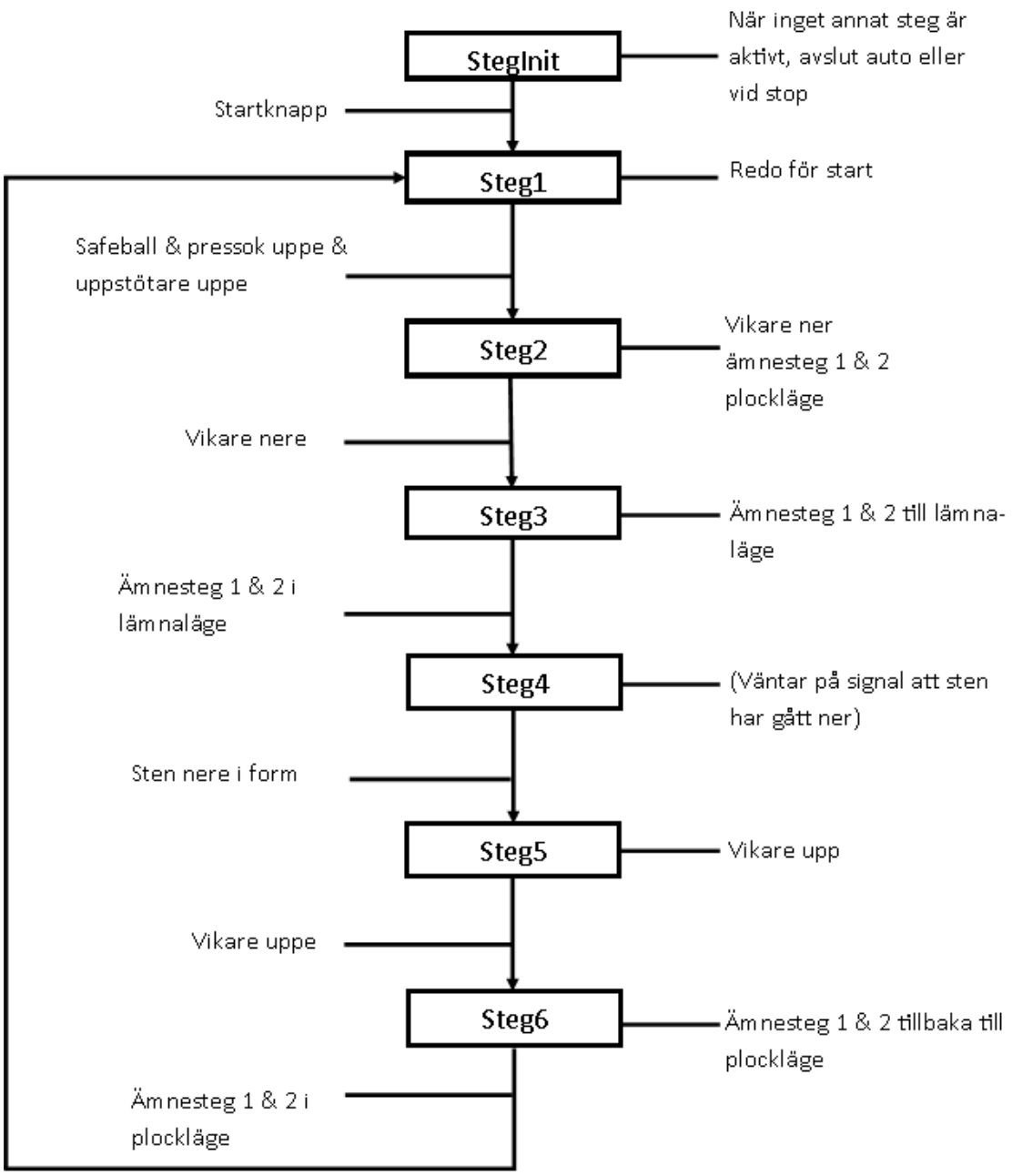
Det nya styrsystemet ska bestå av två huvuddelar, en PLC och ett HMI. Logiken i systemet kommer programmeras i en PLC. PLC:n kommer ta emot, behandla och skicka utsignaler till maskinen. Dessa signaler kommer att vara 24 VDC och kommer skickas till och från olika in- och utgångs(I/O) enheter. Till PLC:n kommer ett HMI och ett antal tryckknappar kopplas för att kunna styra maskinen. Man kommer även kunna ställa in olika inställningar(t.ex. trycknivåer) för maskinen på HMI:et.
Man vill från företagets sida att styrningen för pressning ska styras med tryckgivare istället för dagens tidsstyrning. Man vill även öka säkerheten kring maskinen och därför kommer ett tvåhandsmanöverdon installeras. Detta för att minska risken för operatören att fastna i rörliga delar under produktion.

3 Analys av CW-pressen

För att kunna programmera PLC:n med korrekt logik krävdes en noggrann analys av hur tillverkningsprocessen såg ut. Det anordnades en testkörning av processen från underhållspersonalen. Under testkörningen dokumenterades maskinens olika sekvenser. De gamla el-scheman från 1969 som fanns att tillgå, studerades men var inte helt uppdaterade då en del ombyggnationer har gjorts.
Processen bröts ner i sex olika delprocesser. För att möjliggöra programmeringen till PLC-logiken togs det fram flödesschema för varje delprocess.

3.1 Införaren

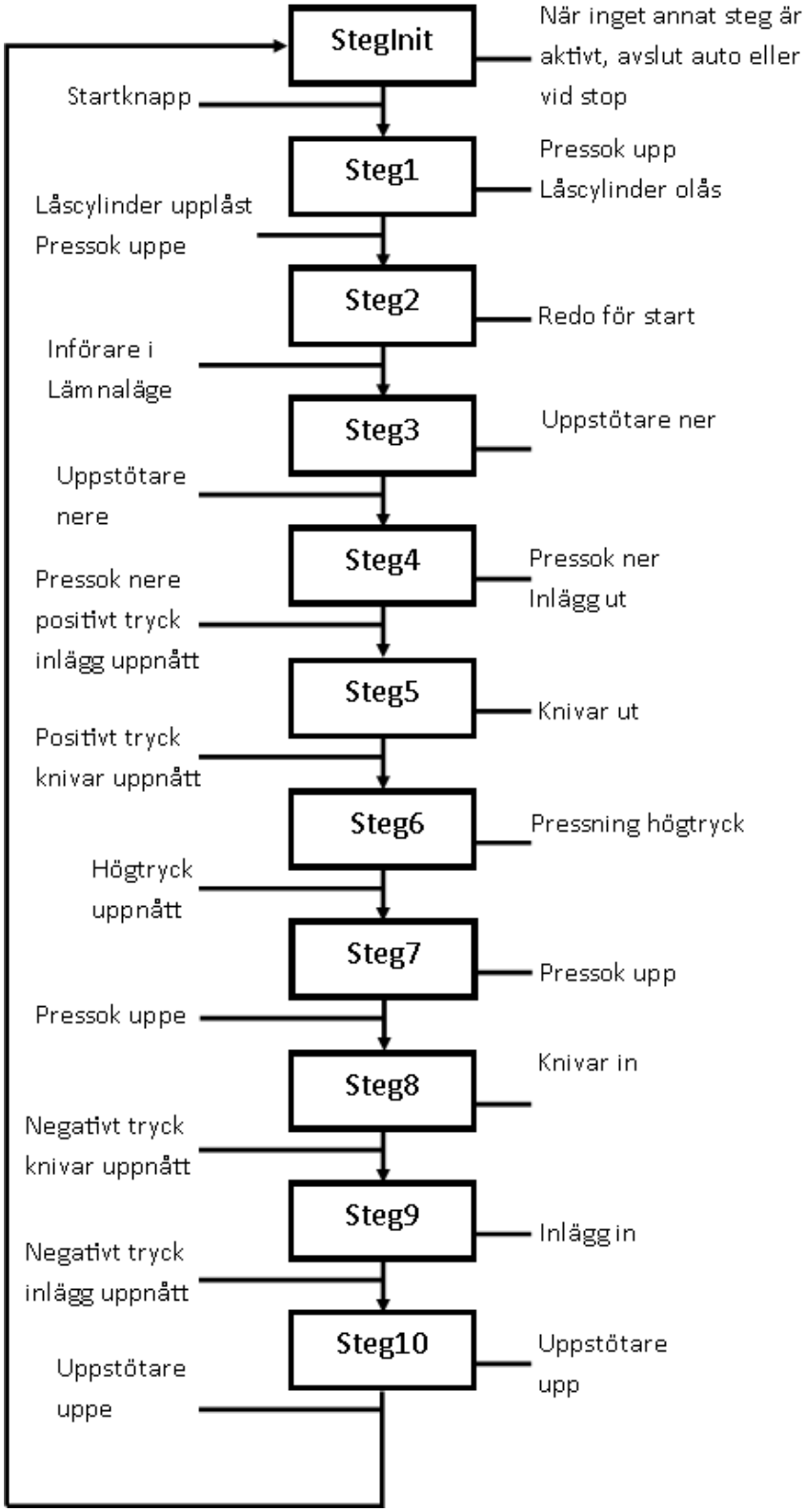
Införaren är den del på maskinen som för in ämnet under pressoket. Den är uppbyggd av en arm som drivs av två pneumatiska cylindrar. På armen sitter det ytterligare en pneumatisk cylinder, denna kommer benämnas vikaren. På vikaren sitter en arm som fälls upp och ner för att föra stenarna framåt. Ämnet placeras bakom vikaren av en operatör som sedan trycker på en startknapp. I dagsläget räcker det med en knapptryckning på startknappen så sker hela införingsprocessen. Detta skall bytas ut mot ett tvåhandsdon som måste vara påverkat under hela införingen och släppas för att starta en ny cykel. Det har diskuterats om en lucka skulle byggas till för att innesluta alla rörliga delar. Då skulle luckan manövreras av tvåhandsdonet. När tvåhandsdonet blivit intryckt fälls vikaren ner för att skjuta fram föregående sten till positionen under pressoket. När ämnet under pressoket har åkt ner i formen fälls vikaren upp för att inte trycka tillbaka nästkommande ämne som matats fram ett steg av införaren. När införaren har återgått till sitt ursprungsläge startar pressningen och operatören måste upprepa proceduren för inmatning. Proceduren för inmatningen kan inte upprepas förrän pressekvensen slutförts.



Figur 5: Flödesschema för införaren

3.2 Pressning

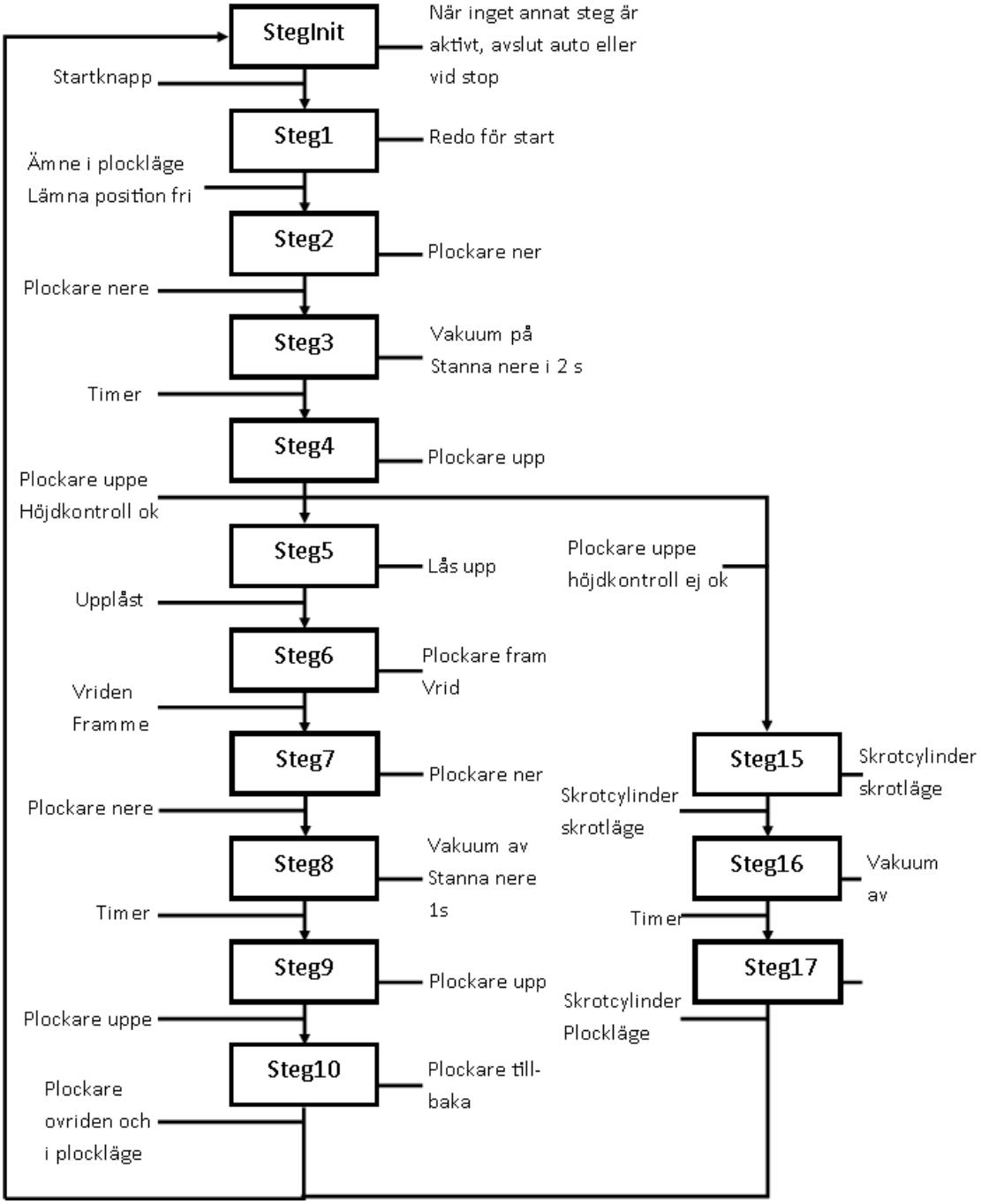
När införaren är i läget där ämnet ligger under pressoket, sänks ämnet ned i formen med hjälp av en pneumatisk cylinder kallad uppstötaren. När uppstötaren är i sitt bottenläge åker införaren tillbaka för att en pressning ska kunna ske. För att erhålla den önskade formen på ämnet skjuts det in inlägg mot ämnets sidor. Samtidigt går pressoket ner på ovansidan av ämnet. När pressoket kommit ned till sitt nedre läge skjuts ihåliga runda knivar in på alla åtta sidor av stenen. Med knivarna i sitt ändläge pressar man ned pressoket med ett högre tryck för att leran ska packas tillräckligt. Efter en inställd tid åker pressoket upp, inlägg och knivar ut. Ämnet som nu blivit en sten förs upp igen av uppstötaren för att slutföra pressekvensen. Maskinen väntar nu på ny inmatning av ett ämne. Vid en ny inmatning skjuts den färdigpressade stenen vidare fram mot linjärenheten.



Figur 6: Flödesschema för pressning

3.3 Linjärenhet

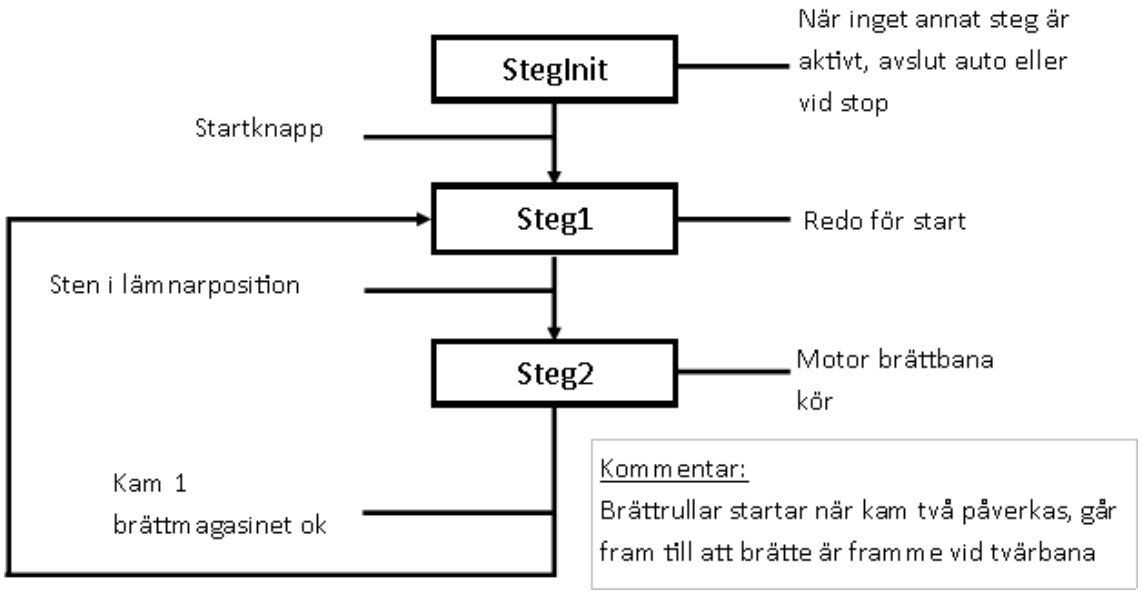
Linjärenheten är den del som förflyttar de färdiga stenarna till brättena. När en färdig sten flyttas fram av införaren till steget efter pressen förs en vakuumsug ner av en pneumatisk cylinder som benämns plockare. Plockaren stannar i sitt nedersta läge en bestämd tid för att säkerställa att vakuumsugen fått grepp om stenen, varpå plockaren återgår till toppläget. För att stenarna ska få plats på brättena måste stenen vridas en aning. Detta görs också med en liten pneumatisk cylinder. Vridningen sker när plockaren kommer till sitt toppläge och börjar förflytta sig mot avlämningspositionen. Framme vid avlämningspositionen förs stenen ned och plockaren stannar i sitt nedre läge en inställd tid för att säkerställa att stenen släppt från vakuumsugen. När plockaren återgått till sitt toppläge och börjar köra tillbaka till ursprungsläget så återställs vridningen. Väl tillbaka väntar linjärenheten på nästa sten.



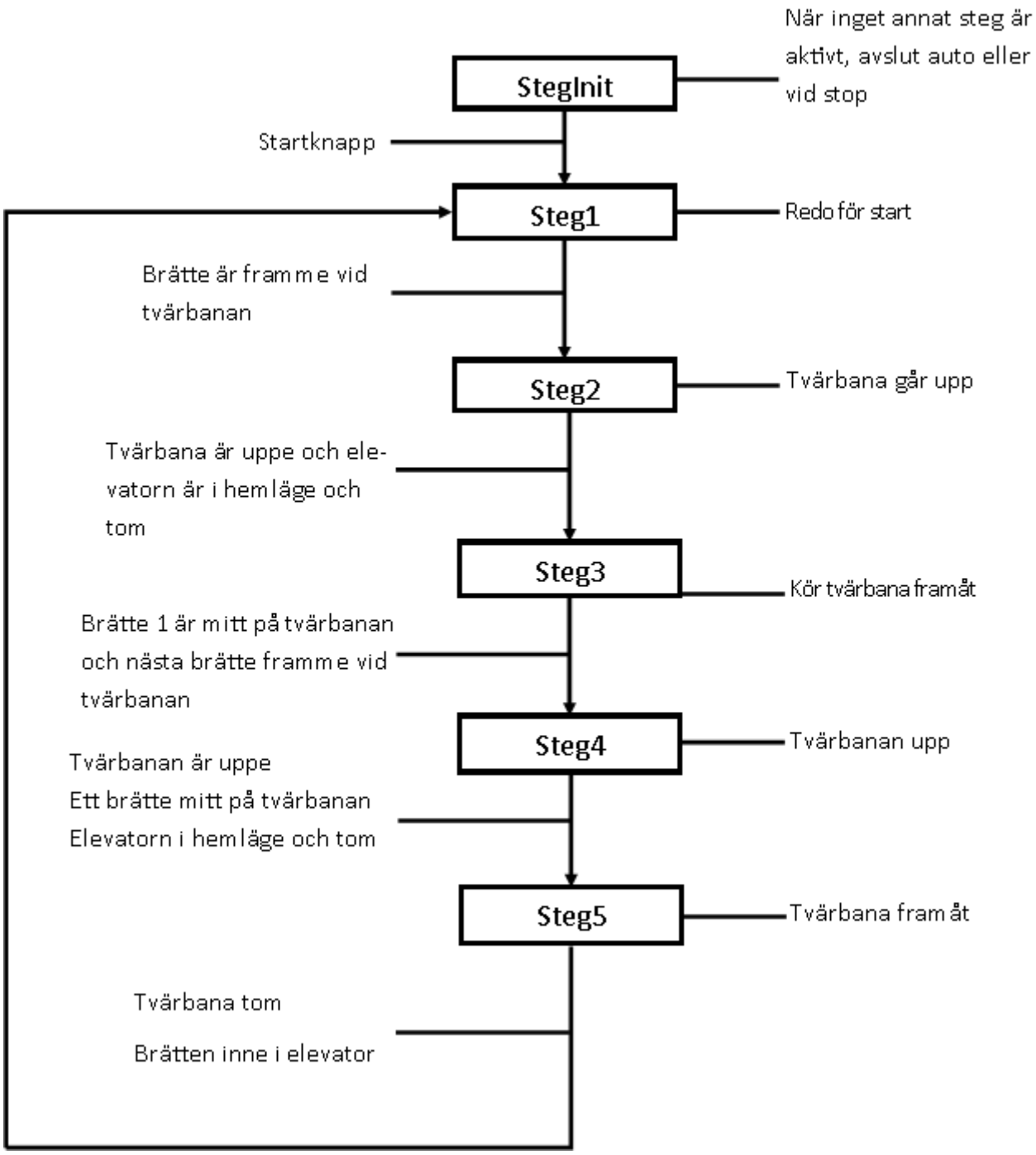
Figur 7: Flödesschema för linjärenheten

3.4 Brättmatning

När en sten sätts ner av linjärenheten bryts en ljusstråle från en lasergivare som detekterar att stenen landat på brättet. Då matas ett nytt brätte fram från brättmagasinet. Frammatningen av brätten körs olika länge beroende på att stenarna ska placeras förskjutet på vartannat brätte, detta för att senare få plats i elevatorn. För att åstadkomma den varierade frammatningen av brättena sitter det tre olika kammar på en axel som driver frammatningen. Dessa påverkar tre olika induktiva givare som skapar mönstret för hur länge respektive omgång av frammatning ska köras.
När ett brätte har fått två stenar på sig förs det fram mot en tvärbana. Tvärbanan består av en kedjebana som lyfts av en pneumatisk cylinder. Först körs ett brätte ut till hälften av denna bana eftersom de i elevatorn ska lagras två och två i bred. När nästa brätte kommer till tvärbanan förs båda två in i elevatorn bredvid varandra till dess att induktiva givare indikerar att brättena är på elevatorn.



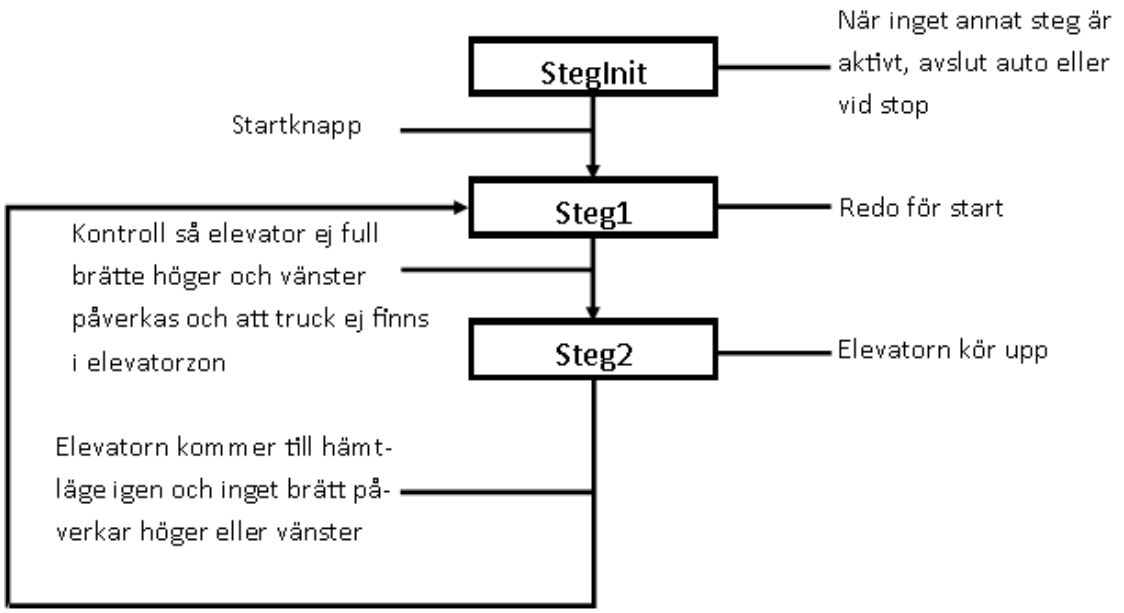
Figur 8: Flödesschema för brättbanan



Figur 9: Flödesschema för tvärbanan

3.5 Elevator

Elevatorn förflyttar de fulla brättena till övreplan av fabriken där de ska förtorkas för att sedan brännas. När elevatorn detekterar att brätten har kommit in kör elevatorn upp ett steg för att göra plats till nästkommande två brätten. I toppen av elevatorn finns det två säkerhetsanordningar. Den ena är en fotocell som detekterar när elevatorn är full och behöver tömmas av personal med truck. Den andra är en mekanisk arm som detekterar om det skulle komma upp brätten i toppläget. Man har även en fotocell framför elevatorn på övreplan för att säkerställa att det inte finns någon truck i elevatorzonen när den kör upp.



Figur 10: Flödesschema för elevatorn

4 Utförande

4.1 Programmering av simulatorn

4.1.1 Teori

En Arduino är en liten enkel dator som bygger på öppen hård- och mjukvara. Arduinon är ett mikrokontrollerkort som programmeras via den egna programmeringsmiljön. Till skillnad från en vanlig PC är Arduinon ämnad för att i huvudsak hantera signaler till/från externa komponenter. Komponenterna ansluts med elektriska ledare till Arduinons kopplingsplintar som sitter på kretskortet och är anslutna till mikroprocessorns olika pinnar[2]. Kopplingsplintarna sitter lättillgängligt och standardiserat mellan de olika modellerna av Arduino. Detta möjliggör att det finns en rad olika moduler att bygga ut sin Arduino med. Modulerna kommer från tredjepartstillverkare och har olika funktioner som hjälper utvecklare att skapa kreativa projekt.
Programmeringsmiljön som används kallas Arduino IDE. Programkod skrivs i ett eget språk som bygger på Wiring[7] och är ämnat för personer som inte har programmerat tidigare eller är nya inom programmering. För att även locka till sig mer erfarna och avancerade programmerare så kommer Arduino IDE med C/C++ inbyggt. Därför kan man kombinera sin programkod mellan det egna språket och C/C++. Då kan man få ut alla de funktioner som kommer med den mikroprocessor som sitter på Arduinon men som inte stöds av Arduinos egna programspråk. De officiella Arduinokorten kommer med en mikroprocessor från Atmels ATmega-serie. Därför kan man även välja att programmera i Atmels egen programmeringsmiljö Atmel Studio.

4.1.2 Simulatorns funktion

Simulatorns huvuduppgift var att simulera de mekaniska rörelser som maskinen hade. De mekaniska rörelserna kan ses som tidsfördröjningar mellan styrsignaler och lägessignaler till och från en PLC. Som programmerare måste man ta hänsyn till i vilken ordning givare påverkas och man får inte skicka styrsignaler till rörliga delar så att dessa kolliderar och går sönder. För att uppnå en god verklighetsrelaterad simulering delades pressens rörliga delar upp var för sig. Det simulatorn skulle göra när en styrsignal till t.ex. en pneumatisk cylinder kom från PLC:n, startades en timer. När timern kom fram till en given tid så skickade simulatorn en lägessignal tillbaka till PLC:n att den pneumatiska cylindern har nått sitt ändläge.

4.1.3 Timerfunktion och programmeringsmiljö

PLC:n styr pressen så att rörelser sker parallellt. Detta ställde krav på Arduinon att den var tvungen läsa av sina ingångar som är kopplade till utgångarna på PLC:n, hela tiden. Vidare måste den vid flera insignaler kunna starta separata timers som kan räkna olika länge. Under tiden den räknade upp måste den kunna läsa av nya styrsignaler och skicka ut lägessignaler tillbaka till PLC:n. En mikrokontroller kan inte i sig själv jobba med parallella händelser. Utan den har sitt huvudprogram som den kör uppifrån och ner, om och om igen. För att komma runt detta så användes mjukvarustyrda avbrott(eng. Interrupts). Mjukvarustyrda avbrott stöds inte i Arduinos programspråk utan detta kommer man åt genom att använda de bibliotek som följer med i Arduino IDE. Eftersom avbrotten då måste programmeras i C valdes att programmera allt i C. Därför kändes det lämpligt att använda Atmel Studio för att skriva all programkod. En fördel med Atmel Studio gentemot Arduino IDE var att man kunde dela upp koden i olika filer för att få koden mer överskådlig.

Arduinon som användes till simuleringen hade modellbeteckningen Mega 2560[1]. Mikrokontrollern hade fem stycken timers som kunde användas för att generera avbrott eller leverera en spänning via pulsbreddsmodulering. Varje timer hade en egen intern räknare som räknar upp varje gång den fick en signal från mikroprocessorns klockfrekvens. Megans mikroprocessor är av modellen ATmega2560[4] och hade en klockfrekvens på 16 MHz. Timern fick därför $16x10^6$ signaler per sekund att räkna upp räknaren. Timerns periodtid mellan signalerna ges av

$$T = \frac{1}{frekvens} = \frac{1}{16 \times 10^6} = 0{,}0625\mu s$$

Timer1 som har användes till avbrotten hade en 16-bitars räknare, vilket motsvarar ett decimalt tal på 65535 (2^{16}-1). Varje gång räknaren slog om från 65535 till 0 och började om, genererades ett avbrott. Avbrottets periodtid blev då

$$65535 \times 0{,}0000000625s \approx 4ms$$

För simuleringen behövdes ett mer kontrollerat avbrott som sker med ett jämt tidsintervall. Intervallet fick heller inte komma oftare än att processorn hinner köra all den kod som skulle köras under varje avbrott. Ett lämpligt tidsintervall för simuleringen var en tiondels sekund. Då erhålls tillräckligt hög upplösning för att läsa av styrsignaler in till Arduinon och samtidigt räkna upp heltalsvariabler. Heltalsvariablerna representerade då tiden och 1 motsvarade 0,1s. För att förlänga periodtiden och få avbrotten på en exakt tid kunde man använda sig av prescaler och en inställning som hette Clear Timer on Compare(CTC). En prescaler används för att dela ner klockfrekvensen med givna exponenter av talet två för att få timern att räkna upp färre antal gånger per sekund. På så vis genererades avbrotten mer sällan men fortfarande på en ojämn tid. CTC användes för att generera avbrott då räknaren kom fram till ett visst värde som motsvarade den tid man önskade istället för att avbrottet kom när räknaren slog om.
Till simuleringen valdes en prescaler med värdet 256. Timerns periodtid mellan signalerna blir då

$$T = \frac{1}{\left(\frac{frekvens}{prescaler}\right)} = \frac{1}{\left(\frac{16 \times 10^6}{256}\right)} = \frac{1}{62500} = 0{,}016ms$$

Avbrottets nya periodtid blir då

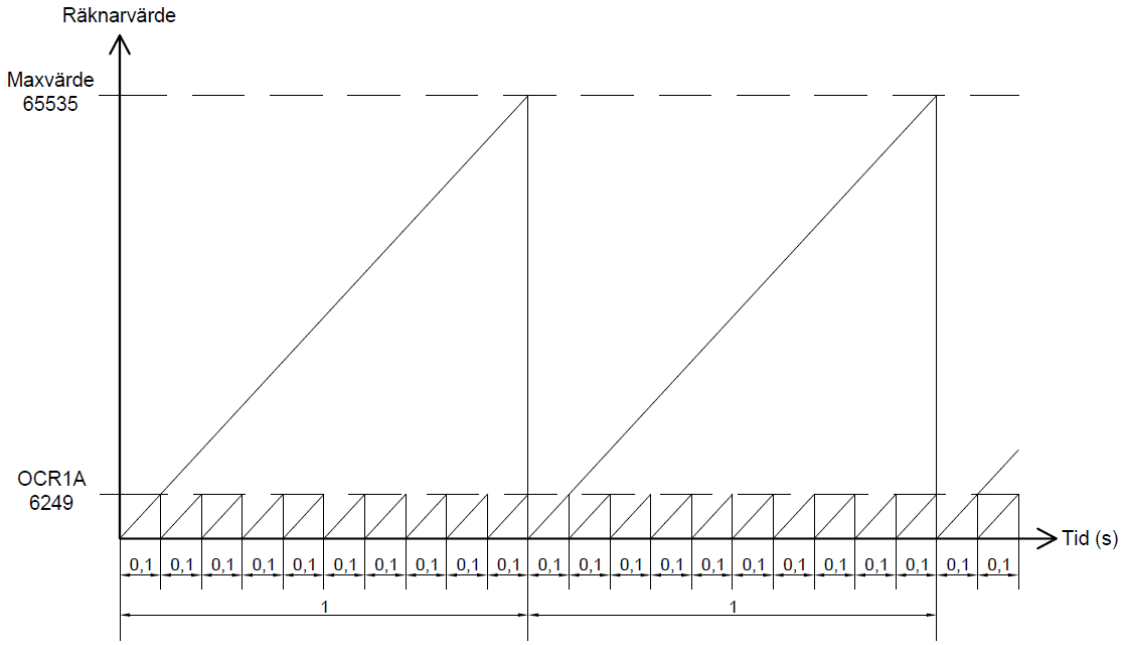
$$65535 \times 0{,}016ms = 1{,}04856s$$

För att beräkna värdet som räknaren skulle räkna till i CTC-mode användes ekvationen

$$\text{önskad tid} = \text{timer periodtid} \times (\text{räknarvärde} + 1)$$

Eftersom det är räknarvärdet som är den enda obekanta i ekvationen och det som skulle ställas in bröts det ut ur ekvationen och gav värdet

$$\text{räknarvärde} = \frac{0{,}1s}{0{,}016ms} - 1 = 6250 - 1 = 6249$$



Figur 11: Diagram för CTC-mode

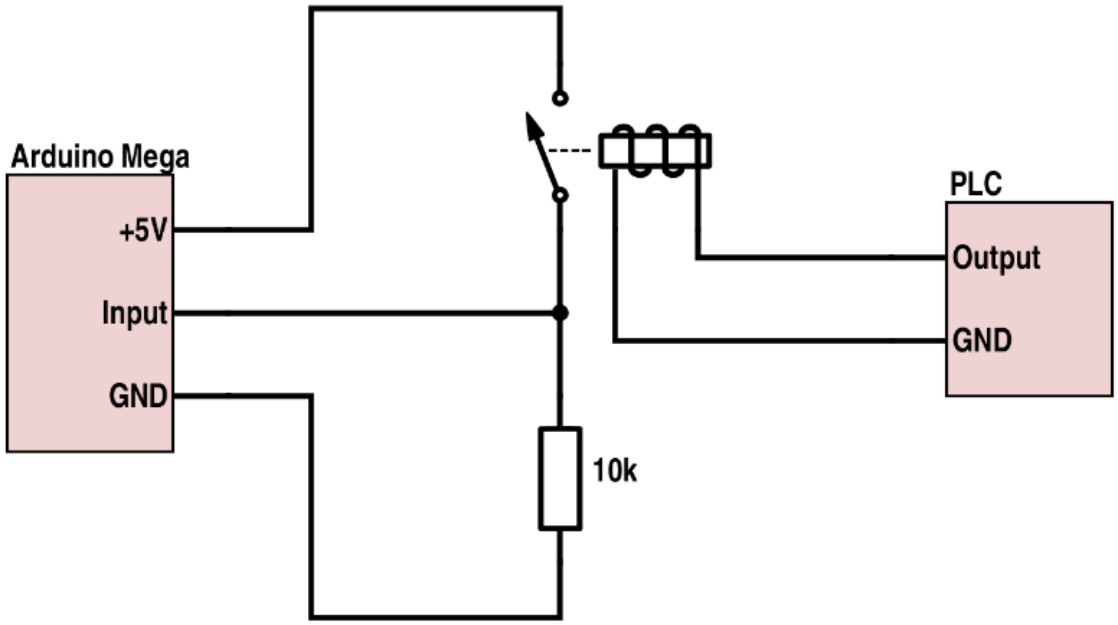
Alla inställningar som kunde göras på ATmega2560 görs i olika 8-bitars register. Varje bit i respektive register representerade en inställning eller i vissa fall där kombinationer av bitar representerade olika varianter av en inställning, t.ex. när man skulle välja prescaler för avbrottet. Valen som kunde göras var 1,8,64,256 samt 1024. Eftersom en bit endast kan anta värdena 0 och 1 räcker det inte för att definiera prescalern. Man behövde därför tre bitar för att kunna ställa in vilken prescaler som skulle användas. För att initiera avbrotten skrevs en funktion som gjorde inställningarna vid uppstart. Nedan följer ett utdrag med några av de registren som användes för att ställa in avbrottet.

```
//sätter compare match registret till valt räknarvärde
OCR1A = 6249;

//startar CTC mode
TCCR1B |= (1 << WGM12);

//Sätter CS12 biten till 256 prescaler
TCCR1B |= (1 << CS12);

//Startar timer compare interrupt
TIMSK1 |= (1 << OCIE1A);
```

I Atmel Studio finns fördefinierat vad de olika registren heter och vad dess bitar hade för benämning. Detta underlättade programmeringen då man inte behövde hålla koll på vilken av de åtta bitarna som skulle ändras utan enbart dess namn. Detta illustreras i programkoden ovan för registren TCCR1B och TIMSK1. Register OCR1A representerade det värde som räknaren skulle ge ett avbrott vid. OCR1A bestod av två stycken register på 8-bitar. OCR1AH representerade de åtta högsta bitarna och OCR1AL de åtta lägsta bitarna. Antingen kunde man tilldela registren sina värden var för sig med binära eller hexadecimala tal, eller så kunde man tilldela ett decimalt tal till OCR1A så konverterade Atmel Studio själv det decimala talet till de båda registren.

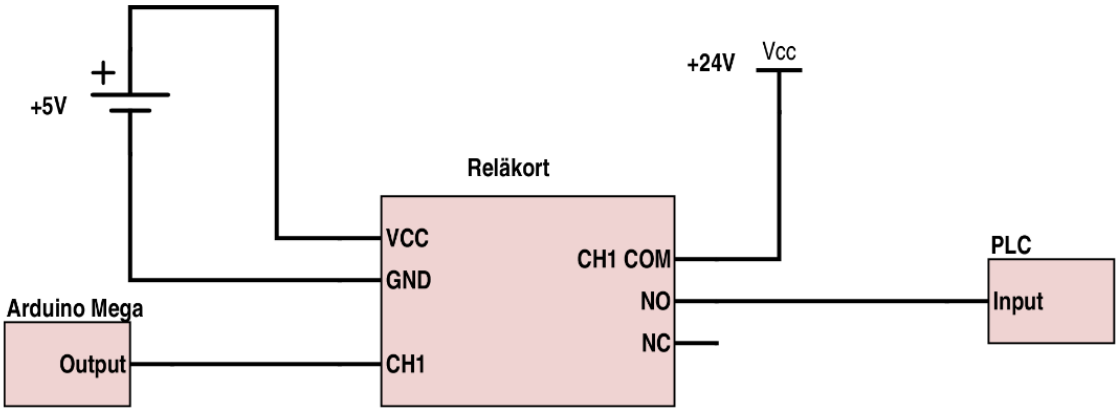
4.1.4 Digitala in- och utgångar

På Arduino Mega fanns det 54 stycken digitala in- och utgångar som var utdragna till kopplingsplintarna på kretskortet. Detta betydde att man har 54 stycken kopplingspunkter som kunde programmeras antingen som ingång eller utgång. Med digital menas att en utgång, då detta användes, hade två stycken lägen, noll eller ett. Med en digital nolla menas att utgången inte ger ut någon spänning eller ström. En digital etta menas med att utgången ger ut +5VDC och en maximal ström på 40mA. Då ingång istället användes är det de motsatta nämligen att +5VDC och maximalt 40mA in gav en digital etta och benämns att ingången går hög. Om ingen spänning och ström kunde mätas på ingången så blev det en digital nolla och benämns då att gå låg.
Till ingångarna kopplades styrsignaler från PLC:ns utgångar. Då PLC:ns utgångar lämnade ut 24VDC så användes ett relä mellan de båda systemen för att skilja de olika potentialerna åt. Beroende på om PLC:n levererade spänning eller inte så växlade reläet. En spänning på 5VDC kopplades genom reläets potentialfria sida till en ingång på Arduinon för att indikera styrsignalerna från PLC:n.



Figur 12: Kopplingsschema för ingångar

Samma upplägg som för ingångarna användes på utgångarna fast tvärt om. Här var det Arduinons utgångar som styrde reläer. En 24VDC spänning var kopplad genom reläerna till ingångarna för att indikera de rörliga delarnas ändlägen.



Figur 13: Kopplingsschema för utgångar

Likt för avbrotten skulle inställningar göras för att ställa in om ingång eller utgång skulle användas. Detta gjordes i register som hette DDRx, där x ska vara en bokstav mellan A och L. Alla in- och utgångar var knutna till ett register och vilket register, fås av kopplingsschemat för Arduino Mega-kortet[2]. För att göra inställningarna mer lättlästa så definierades variabler för att ersätta de faktiska inställningsvärdena. Till exempel ställdes register DDRG in såhär

```
DDRG = (output<<X4) | (input<<X39) | (input<<X40) |
                    (input<<X41);
```

Output var definierat som en 1:a och input var definierat som en 0:a. I utdraget ovan var X4 definierat som PG5. PG5 berättade att det var den femte biten i register G. X4 valdes för att det var in- eller utgång fyra som var ansluten till PG5 enligt kopplingsschemat[2].
För de kopplingspunkter som var programmerade som utgångar användes ett register som hette PORTx, där x var samma bokstav som användes för att ställa in en ingång eller utgång. Varje utgång hade då en bit i registret som sattes till ett om utgången skulle leverera +5VDC eller till noll om den skulle vara avstängd. Det var samma bit som skulle ändras som för inställningen ovan, t.ex. bit nummer fem för utgång nummer fyra. För att inte ändra de andra bitarna så maskerades de bort när man skulle ändra värden. Vid aktivering av en utgång användes

```
PORTG = PORTG | 0x20
```

Då gjordes en logisk *ELLER* mellan registrets aktuella värden och ett 8-bitars tal där endast den bit man ville sätta till ett var en etta. I exemplet ovan var det bit nummer fem som var en etta och 8-bitartalet är skrivet med hexadecimala tecken. 0x berättade för programmet att det var ett hexadecimalt tal. Maskningen hade även kunnat skrivas som ett binärt tal och hade då blivit *0b00100000*.
När sedan utgången skulle avaktiveras användes samma princip fast med en logisk *OCH* istället

```
PORTG = PORTG & 0xDF
```

I denna maskning sattes enbart biten som skulle ändras till noll och övriga bitar till ett. Då behöll övriga bitar sina värden.

För att läsa av en utgång användes en logisk *OCH* där man enbart satte biten som skulle kontrolleras till ett. Resultatet jämfördes sedan med ett 8-bitarstal där biten som man vill kontrollera sattes till noll eller ett. Ett användes för att kontrollera om utgången var aktiverad och noll om den var avaktiverad. I simuleringen kontrollerades om utgångarna var aktiverade, t.ex. utgång nummer 4

```
(PORTG & 0x20) == 0x20
```

Till avläsningen av ingångarna var det endast av intresse om en ingång gick hög. Maskeringen blev då identisk med den för att läsa av en utgång så när som att man använde sig av ett annat register. För ingångar användes registret PINx, där x skulle vara samma bokstav som använts tidigare. Maskeringen för ingång nummer 40 såg då ut såhär

```
(PING & 0x02) == 0x02
```

Likt för inställningarna så definierades all maskering för ingångar och utgångar med variabler för att förenkla själva programkoden. Alla dessa definitioner sparades i en headerfil med namnet define. De definierade variablerna kunde sedan anropas i programkoden för att läsa av in- och utgångar eller nollställa eller ettställa utgångarna. Exempel på utgångar för cylindern på pressen som kallas vikare

```
#define vikare_nere     ((PORTH & 0x01) == 0x01)
#define vikare_nere_ON  (PORTH = PORTH | 0x01)
#define vikare_nere_OFF (PORTH = PORTH & 0xFE)
#define vikare_uppe     ((PORTH & 0x02) == 0x02)
#define vikare_uppe_ON  (PORTH = PORTH | 0x02)
#define vikare_uppe_OFF (PORTH = PORTH & 0xFD)
```

Och två stycken definierade ingångar

```
#define vikare_upp  ((PINL & 0x10) == 0x10)
#define vikare_ner  ((PINL & 0x20) == 0x20)
```

4.1.5 Fördröjningar med digitala lägen

Rörliga delar på pressen hade digitala givare som detekterade olika lägen och som oftast var ändlägen, därför har alla programmerats med liknande fördröjningsfunktioner. Det som skiljde de olika delarna åt i programkoden var variablerna. Varje fördröjning hade ett antal variabler som höll koll på funktionernas status. Först två heltalsvariabler av typen integer, en innehöll

rörelsens tid i antal tiondelar och en som räknade upp varje gång fördröjningen anropades, vilket skede i varje avbrott var tiondels sekund. Sedan användes två stycken sanningsvariabler som skulle definiera åt vilket håll rörelsen gick på, fram eller tillbaka. I fördröjningsfunktion nummer 2 som hade namnet *Fordrojning2* benämndes variablerna med *tid_for2*, *counter_for2*, *Fordrojning2out* samt *Fordrojning2in*. Sanningsvariablerna sattes till *FALSE* när programmet initierades. Denna fördröjning simulerade tiden för vikaren. Följande exempel visar koden för vikarens uppåtrörelse

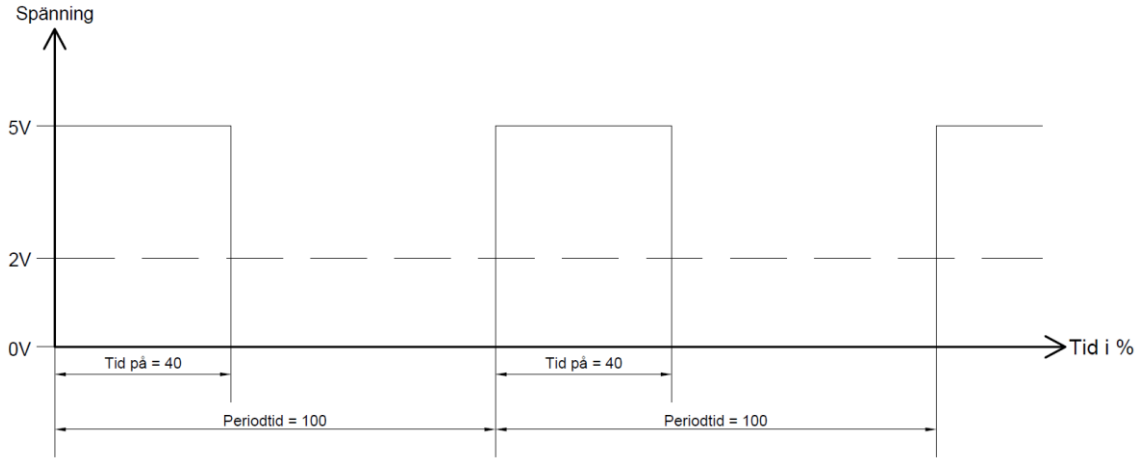
```
counter_for2++;

if(vikare_upp & vikare_nere){
    counter_for2 = 0;
    Fordrojning2out = true;
    vikare_nere_OFF;}

if((counter_for2==tid_for2) & Fordrojning2out){
    vikare_uppe_ON;
    Fordrojning2out = false;}
```

Ingångsvillkoret i den första if-satsen uppfylldes när det kommer en styrsignal från PLC:n att vikaren skulle gå upp och vikaren samtidigt var i positionen nere. I if-satsen nollställdes räknaren för fördröjningstiden, villkoret att cylindern var i rörelse uppåt sattes till *TRUE* och slutligen avaktiverades utgången för att vikaren var nere eftersom cylindern hade börjat att röra sig. *Tid_for2* är satt till 5, så efter fem avbrott var villkoret för den andra if-satsen uppfyllt och cylindern har då nått läget uppe. I ingångsvillkoret för den andra if-satsen är det samma variabler i parentesen för cylinderns båda rörelser (finns i bilaga 7.1.4). Sanningsvariabeln avgjorde därför vilken av if-satserna som skulle köras.
Koden inuti den andra if-satsen aktiverade utgången för att vikaren var uppe och återställde sanningsvillkoret att cylindern var på väg upp.
För vikarens nedåtrörelse var kodens upplägg densamma men med motsatta styr- och lägessignaler och fanns i samma fördröjningsfunktion. I simuleringen togs inte hänsyn för speciella driftfall. Därför har det antagits att rörelser alltid fullföljs och en motsatt rörelse inte kunde påbörjas förrän en cylinder var i ett ändläge.

4.1.6 Fördröjningar med analoga lägen och pulsbreddsmodulering

Pressens tre hydrauliskt styrda rörelser, inlägg, knivar och pressok. Saknade alla digitala lägesgivare kopplade till PLC:n för att detektera

ändlägen. Istället hade de tryckgivare för rörelsernas vardera riktningar. Trycket som rörde kolven ut ur godset benämndes som positivt tryck och motsatt riktning benämndes som negativt tryck.
Tryckgivarna som användes till pressen skickade en ström på 4-20mA till det analoga ingångskortet på PLC:n. Den analoga strömmen behövde därför skapas från utspänningen som Arduinon levererade. Spänningen var tvungen gå att variera för att strömmen in till PLC:n skulle variera och simulera de olika trycken hos de hydrauliska cylindrarna. En variabel spänning kunde skapas genom någon av utgångarna som stödjer pulsbreddsmodulering(PWM). PWM används ofta för att styra elektrisk utrustning som t.ex. elmotorer. Vid PWM slår man på och av spänningen snabbt för att få en önskad spänning. Förhållandet mellan på-tiden och periodtiden kallas duty cycle och avgör hur stor spänningen blir. Det är viktigt att spänningen slås på och av så pass snabbt att ansluten utrustning inte känner av ändringen. Spänningen får formen av en fyrkantspuls och ser ut såhär



Figur 14: Diagram för duty cycle

PWM utgångarna styrdes och programmerades med samma timerfunktioner som för avbrotten. Till simuleringen behövdes fem olika analoga signaler. Varje timer kunde hantera tre stycken PWM-utgångar, så Timer3 och Timer4 valdes. För att generera PWM-signalen användes ett läge som hette Fast-PWM, 8-bitar. Likt avbrotten hade Fast-PWM en räknare som räknade upp till ett 8-bitars tal för varje period. Vid varje ny period aktiverades utgången och när inställt räknarvärde nåddes avaktiveras utgången för resten av perioden. Varje timer hade en räknare med tre olika räknarvärden som kunde ställas in. Räknarvärdena var i sig kopplade till

var sin utgång där PWM-signalen plockades ut. Eftersom en kort periodtid var att föredra för att inte påverka ansluten utrustning, sattes timerns prescaler till ett. Fast-PWM hade en switchfrekvens och periodtid som ges av

$$f_{PWM} = \frac{f_{clk}}{prescaler \times (1 + 0xFF)} = \frac{16MHz}{256} = 62{,}5kHz$$

$$T_{PWM} = \frac{1}{f_{PWM}} = \frac{1}{62{,}5kHz} = 16\mu s$$

Inställningarna för PWM gjordes i samma register som avbrotten fast för Timer3 och Timer4. Inställningar för Timer4 på utgångarna sex och sju gjordes enligt följande

```
//Clear OC4A on Compare Match, set OC4A at BOTTOM(non-
inverting mode)
TCCR4A |= (1 << COM4A1) | (0 << COM4A0);

//Clear OC4B on Compare Match, set OC4B at BOTTOM(non-
inverting mode)
TCCR4A |= (1 << COM4B1) | (0 << COM4B0);

//sätter "Waveform Generation Mode" till Fast-PWM 8-bit
TCCR4A |= (0 << WGM41) | (1 << WGM40);

//sätter "Waveform Generation Mode" till Fast-PWM 8-bit
TCCR4B |= (0 << WGM43) | (1 << WGM42);

//Startar klockan och sätter prescaler till 1;
TCCR4B |= (1 << CS40);
```

De olika räknarvärdena kunde ändras när som helst och det värde registret för räknarvärdet hade när en ny period började, gällde för den perioden. Räknarvärdet sattes i fördröjningsfunktionerna och beskrivs längre fram i rapporten.

För att skapa en jämn spänning att omvandla användes ett lågpassfilter som kopplades på PWM-utgången. Lågpassfiltret släpper genom de låga frekvenserna och dämpar de höga frekvenserna. Detta ledde till att spänningens amplitud sänktes till samma procentuella nivå från 5VDC som

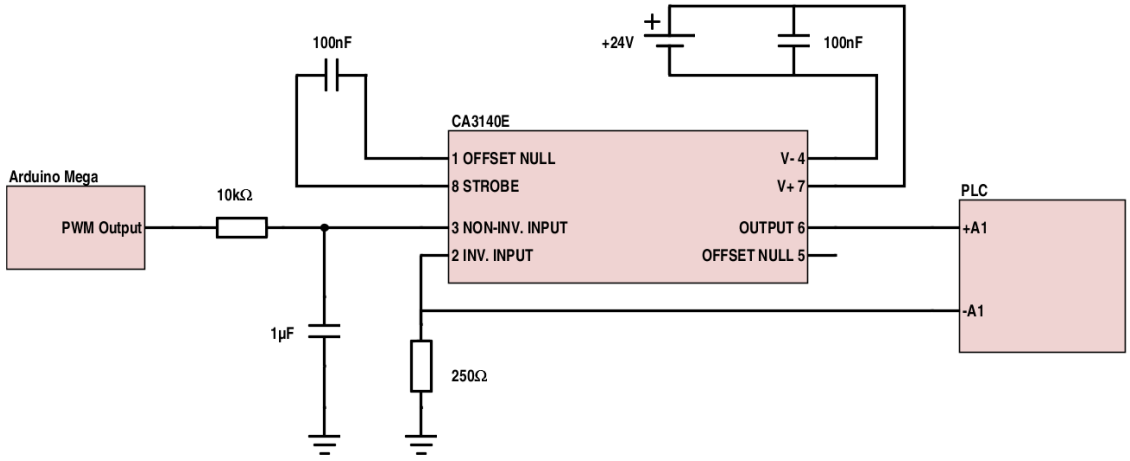
dutycycle värdet. De höga frekvenserna filtreras bort och kapacitansen hjälpte till att jämna ut spänningen. Kvar blev endast lite rippel på spänningen. Brytfrekvensen för vilken filtret börjar dämpa, bestäms av filtrets tidskonstant och valdes så att den minst var tio gånger så lång som PWM-signalernas periodtid. Tidskonstanten beräknades med

$$\tau = R \times C = 10k\Omega \times 1\mu F = 0{,}01s$$

$$\frac{\tau}{T_{PWM}} = \frac{0{,}01s}{16\mu s} = 625\ ggr$$

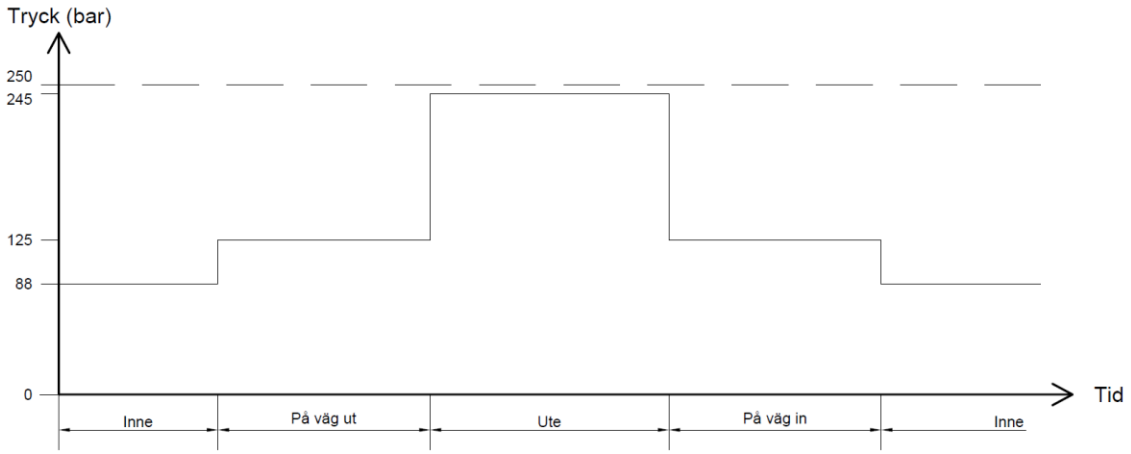
$$brytfrekvens = f_c = \frac{1}{2\pi\tau} = 15{,}915\ Hz$$

Den filtrerade spänningen skulle nu omvandlas till en ström. Till detta användes en operationsförstärkare(OP) av modellen CA3140E[8] och en 250Ω resistor. OP:ns egenskaper med väldigt hög inimpedans och låg utimpedans gjorde att den lämpade sig för omvandlingen. Den höga inimpedansen gjorde att ingen ström kunde antas gå in i OP:n. Därför blev där inget spänningsfall över den. Med negativ återkoppling på den inverterande ingången, hamnade hela spänningen över resistorn. En ström drevs då ut ur OP:n genom PLC:n och resistorn. Med ohms lag kunde strömmen beräknas. Strömmens storlek ändras proportionellt mot spänningen.
För att driva OP:n används vanligen dubbel matningsspänning, det vill säga ±24VDC. Då en sådan spänning inte fanns att tillgå användes istället en OP som kunde drivas med enkel matning på +24VDC. För att stabilisera matningen till OP:n och kompensera eventuell Miller-effekt användes två stycken kondensatorer. Den första kopplades parallellt över matningen till OP:n. Den andra kopplades mellan *Strobe* och *Offset Null* utgångarna.



Figur 15: Kopplingsschema för trycksimulering

Fördröjningsfunktionerna för dessa tre rörelser var uppbyggda på samma sätt som med digitala givare. Skillnaden var att sanningsvariabler användes istället för att utgångar drar. Trycknivåerna på respektive sida har generaliserats grovt och storleken på trycket var inte i nivå med verkligheten. I PLC-programmeringen kontrollerades endast om två olika trycknivåer har uppstått. Att trycket ökade och nådde ett maxvärde eller att trycket minskade och nådde ett minimivärde. När kolven rörde sig ut ur godset gick det positiva trycket mot sitt maxvärde och det negativa samtidigt mot sitt minimivärde. När kolven rörde sig in i godset är värdena det inverterade. Max- och minvärdena valdes för att detektera ändlägen för cylindrarna. Vilka nivåer trycket har mellan dessa två ändlägen var ointressant ur programmeringens synvinkel. Tryckkurvorna som skapades med Arduinon ser ut så här för en rörelse ut ur godset



Figur 16: Diagram över tryckkurva

Programkod för knivarnas rörelse utåt

```
counterKnivar++;

     if(knivar_fram & knivar_inne){
    knivar_inne = false;
    knivar_aktivt_ut = true;
    counterKnivar = 0;
    knivar_negativt_i_rorelse;
    knivar_positivt_i_rorelse;
     }

     if(knivar_aktivt_ut & (counterKnivar == tid_knivar)){
    knivar_aktivt_ut = false;
    knivar_ute = true;
    knivar_positivt_ute;
    knivar_negativt_ute;
     }
```

Likt fördröjningarna för digitala givare har antagits att rörelser alltid fullföljs. Inuti if-satserna sätts räknarvärdena via de definierade variablerna knivar_negativt_ute med flera. Definitionerna ser ut såhär

```
#define knivar_positivt_i_rorelse   (OCR4A = 127)
#define knivar_positivt_ute         (OCR4A = 250)
#define knivar_positivt_inne        (OCR4A = 90)
#define knivar_negativt_i_rorelse   (OCR4B = 127)
#define knivar_negativt_ute         (OCR4B = 80)
#define knivar_negativt_inne        (OCR4B = 250)
```

Värdena som valts kan max anta värdet 255 då räknaren var ett 8-bitars. Utspänningarna blir därför

$$Duty\ cycle = D_{låg} = \frac{90}{255} \approx 35\% \ \rightarrow \ U_{låg} = 0{,}35 \times 5 = 1{,}7\ VDC$$

$$D_{medel} = \frac{127}{255} \approx 50\% \ \rightarrow \ U_{medel} = 0{,}5 \times 5 = 2{,}5\ VDC$$

$$D_{hög} = \frac{250}{255} \approx 98\% \ \rightarrow \ U_{hög} = 0{,}98 \times 5 = 4{,}9\ VDC$$

Spänningarna gav strömmar till PLC:n på

$$I_{låg} = \frac{U_{låg}}{R} = \frac{1{,}7\ V}{250\ \Omega} = 6{,}8\ mA$$

$$I_{medel} = \frac{U_{medel}}{R} = \frac{2{,}5\ V}{250\ \Omega} = 10\ mA$$

$$I_{hög} = \frac{U_{hög}}{R} = \frac{4{,}9\ V}{250\ \Omega} = 19{,}6\ mA$$

Trycknivåerna är orealistiskt höga för knivar och inlägg men till simuleringen behövde inte hänsyn tas för trycket. Endast inställda värden i PLC:n skulle uppnås.

4.1.7 Specialfall i fördröjningsfunktionerna

Ett antal specialfall programmerades i fördröjningsfunktionerna för att kunna köra simuleringen.
När en autokörning startades av operatören laddade han eller hon pressen med ett ämne. Det krävdes sedan att införaren körde fram två gånger innan ämnet kom fram till pressoket och uppstötaren. Pressen skulle ha en lasergivare som skulle detektera att ämnet var framme vid uppstötaren och även detektera att ämnet följde med ner i formen när uppstötaren gick ner. För att simulera att denna givare blev påverkad placerades en räknare i fördröjningen för cylindern som heter *ämneSteg2*. När räknaren hade räknat till två eller mer skickade Arduinon en signal till PLC:n som simulerade att givaren blev påverkad varje gång som *ämneSteg2* nådde sitt lämnaläge. Givarens signal återställdes sedan av att uppstötaren nådde sitt nedre ändläge.
När samma räknare nådde värdet tre eller mer skickades på samma sätt en signal till PLC:n för en annan givare som skulle detektera att en sten var i position så att linjärenheten kunde starta sin sekvens. Denna signal återställdes när fördröjningen för linjärenhetens framåtrörelse startades.

4.1.8 Huvudprogrammets uppbyggnad

Programkoden för simuleringsmiljön bestod av fem stycken filer. Definitioner för fördröjningarna var samlade i en headerfil som hette *define*. Alla fördröjningsfunktioner fanns i en c-fil som hette *Actions*. I en tillhörande headerfil fanns alla funktioner och variabler deklarerade för att

kunna anropas från programmets huvudfil. Huvudfilen hette Arduino_CW och är programmet som styrde simuleringen. I huvudprogrammet fanns definitioner som användes för att förenkla inställningen av Arduinons in- och utgångar.
När Arduinon spänningssattes eller återställningsknappen trycktes in började koden som fanns i main-programmet att köras. Här anropades först funktionen *Initialize* som ställde in in- och utgångar. Därefter kördes en funktion *StartingPositions* för att ställa simulatorn i ett startläge. Detta innebar att alla rörelser ställdes i något av sina ändlägen för att simuleringen från PLC:n skulle kunna startas. Med ändlägen menas att utgångar på Arduinon aktiverats för att lägessignaler skulle skickas till PLC:n. Även simuleringen av de analoga signalerna fick sina startvärden. Startläget för Arduinon är inte detsamma som startläget som pressen behövde för att kunna starta sin autodrift. Detta för att kunna simulera att pressen själv ställde sig rätt när uppstartsekvensen av automatiken startades.
Vidare i main-programmet anropades funktionerna InitOC1, InitOC3 och InitOC4 som ställde in timerfunktionaliteten som skapade avbrotten och PWM-signalerna. Efter detta ställde sig main-programmet i en oändlig while-loop tills dess att spänningen till kortet bröts.
Alla fördröjningsfunktioner anropades inuti avbrotts-funktionen som var inställd på att köras varje tiondels sekund. För att kunna simulera ett nödstopp på ett så verklighetstroget sätt som möjligt lades detta i avbrottsfunktionen. Nödstoppet var tvunget att vara OK för att simuleringen skulle kunna köras och fördröjningsfunktionerna anropas. När ett nödstopp simulerades stannade alla rörelser i sitt aktuella läge. När nödstoppet återställdes igen anropades en funktion som ställde pressen i startläge för automatiskt drift. I verkligheten görs denna återställning via manuell körning via operatörspanelen eller av pressen själv under uppstartssekvensen.
En säkerhetsinställning som gjordes för införaren placerades också i avbrottsfunktionen. Införaren fick endast köras automatiskt om ett tvåhandsgrepp var påverkat av operatörens båda händer. Detta för att risken för klämrisk skulle försvinna. Därför hade de fördröjningsfunktioner som gäller införaren begränsats så att de endast anropades om tvåhandsgreppet var påverkat. En signal från PLC:n skickades därför till Arduinon för att få denna funktion. Om tvåhandsgreppen av någon anledning släpps under en pågående rörelse stannar uppräkningen av tidsvariabeln och införaren kan anses stå still. Övriga rörelser påverkas inte av tvåhandsgreppet.

4.2 Programmering av PLC

4.2.1 Teori

En PLC är en mikroprocessorbaserad styrenhet. En typisk PLC har fem beståndsdelar *processor, minne, strömförsörjningsenhet, ingångs- och utgångsmoduler och programmeringsenhet*. Minnet i PLC:n används för att lagra den logik och resultat av beräkningar som sker i mikroprocessorn[10, sida 3]. Det är i in- och utgångsmodulerna som PLC:n tar emot signaler från olika givare och skickar ut styrsignaler till t.ex. motorer och ventiler. Dessa moduler brukar förkortas I/O-moduler.
Centralenheten läser av signaler från givare genom ingångsmodulerna och instruktionerna från programminnet. Beroende av tillståndet på givarna och instruktionerna i programminnet beräknas tillståndet för utgångarna. Då hela programmet avverkats ändras utgångarnas tillstånd beroende på beräknade resultat samtidigt som en ny avläsning av givarna påbörjas. Detta cykliska arbetssätt är styrsystemets mest karakteristiska drag. Tiden mellan avläsningarna bestäms bl.a. av centralenhetens prestanda och längden på programmet[9].
Framtagningen av PLC:er kom i början av 1970-talet då man ville ersätta de gamla relästyrda systemen. Fördelen med ett PLC-systemen är att om man behövde ändra något i systemet ändrar man bara instruktionerna i PLC-programmeringen. Detta är mycket enklare och mer tidseffektivt än de gamla relästyrda systemen som krävde att man kopplade om och drog nya kablar till de olika reläerna om man behövde ändra funktionerna. Från 70-talet och fram till idag har PLC:erna utvecklats från att vara en liten kompakt PLC med ett lägre antal inbyggda I/O:s till modulära PLC:er som kan modifieras med fler I/O-enheter, analoga enheter och kommunikationsenheter för att användas i större och mer komplexa system[10, sida 3].

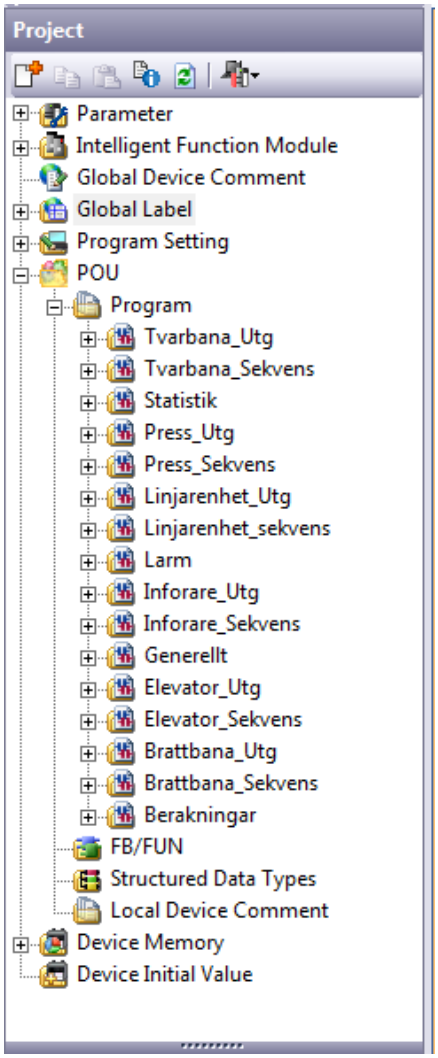
GxWorks2 är ett verktyg för programmering av MELSEC system. MELSEC är en produktserie av PLC:er från tillverkaren Mitsubishi Electric. GxWorks2 ersätter de gamla programmeringsmiljöerna GX Developer och GX IEC Developer.
GxWorks2 har stöd för fem IEC 61131-3 standardiserade programspråk och MELSEC-reläschema.

De programspråk som stöds i GxWorks2 är

- IL (instruktionslista)
- LD (ladderdiagram)
- FBD (funktionsblocksdiagram)
- ST (struktureradtext)
- SFC (Sequential function chart)

4.2.2 Inledning

Programmeringskod till en maskin som benämns 5905, som styrs av en PLC fanns att tillgå. Denna kod studerades för att få en uppfattning av hur PLC programmeringen var uppbyggd på företagets övriga utrustning. Uppbyggnaden av programmet efterliknades denna för att hålla samma standard som man använt sig av tidigare. Programmet är uppbyggt av olika Program Organisation Units (POU). Det finns tre typer av POU:s i IEC standarden 61131-3, dessa är *funktioner, funktionsblock och program*. Program representerar toppen av ett PLC-program och har tillgång till PLC:ns I/O-enheter. Programmet gör I/O-enheterna tillgängliga till de andra POU:erna[11, sidan 22].
PLC-programmet som byggdes upp grundades på ett generellt POU och hade koppling till delar som rörde hela maskinen. För att styra varje delprocess som maskinen delades upp i, skapades två POU:s. Ett POU gjordes för styrning av sekvenser som delprocessen kunde befinna sig i. I programmet stegas rader igenom med olika övergångsvillkor för att bestämma i vilket steg maskinen är i. Med information från sekvens POU:t styrdes det andra POU:t som var till utgångarna. Utgångarnas tillstånd ställdes beroende på information från sekvens POU:t. Alla POU:s för delprocesserna baserades på flödesschemat för respektive delprocess.
Det blev totalt 16 stycken POU:s för hela PLC-programmet.



Figur 17: Lista över PLC-program

Programmet gjordes i programmeringsspråket strukturerad ladder som är en kombination där man kan använda ladderschema kombinerat med funktioner och funktionsblock.
Nedan följer beskrivningar av de olika POU:erna. Sekvens- och utgångs POU:erna för de olika delprocesserna var uppbyggda på samma sätt, därför beskrivs bara en av dessa.
Fullständig programmeringskod kan läsas i Bilaga 7.2.2.

4.2.3 Generellt

I programdelen generellt gjordes delar som berör hela maskinen så som att hydraulmotorn skulle vara i drift så länge det inte fanns risk för person eller maskinskada.

I den här programdelen körs även de olika tillstånden för automatisk drift. *Uppstart_Auto* där maskinen kontrollerades om den befann sig i startläge. Befann maskinen sig i ett felaktigt startläge t.ex. efter ett nödstopp eller efter manuell körning körs maskinens delar till startläget automatiskt och övergår till *Auto* läge. *Auto* läget är aktivt under körning då inga larm är utlösta. *Avsluta_Auto* som var ett läge som blev aktivt då summalarmet blivit aktivt eller stoppknappen blivit intryckt. Detta läge försöker ställa maskinen i initieringsläget för att normal uppstart ska vara möjlig, att utföras senare. Under avslutet finns en timeout som aktiveras om någon av sekvenserna inte kommit till sitt initieringsläge inom en vis tid. Om timeouten blir aktiv sätts de stegen som inte nått sitt initieringsläge till initieringsläge och *Avsluta_Auto* blir inaktivt.

För att få funktionen med tvåhandsgreppet att fungera som det var tänkt skapades en minnescell som kontrollerade om tvåhandsgreppet hade släppts innan en ny införarcykel kunde göras.

4.2.4 Beräkningar

De analoga signalerna som mottogs från tryckgivarna var rådata och var tvungna att skalas om.
Mätområdet som givarna hade, var mellan 0-250 bar som omvandlas i givarna till en ström på 4-20 mA. Denna ström kopplas till analogmodulen Q68ADI från Mitsubishi. Upplösningen valdes till 4000 vilket är normalupplösning[ref]. Detta gör att strömmen representeras i ett digitalt heltal mellan 0-3999 i PLC:n. Heltalet måste sedan multipliceras med 250 och divideras med 4000 för att kunna presentera mätvärdet i storheten Bar på HMI:et.

4.2.5 Införarsekvens

Alla sekvensprogram börjar med ett initieringssteg som användes då maskinen inte befinner sig i något sekvenssteg eller då maskinen ska avsluta automatisk drift.
För att införaren ska komma till steg ett där maskinen är redo för körning måste *Auto* eller *Uppstart_Auto* vara aktivt samtidigt som införaren befinner sig i initieringssteget eller steg sex som är sista steget i införarens sekvens.

För att sekvensen ska kunna gå vidare måste föregående steg vara aktivt samtidigt som övergångsvillkoren är aktiva. Då går maskinen vidare och aktiverar kommande steg samtidigt som föregående steg inaktiveras.

4.2.6 Införare utgångar

Utgångarna har två lägen som de kan aktiveras i, automatik eller manuellt. Under automatik, beroende på i vilket sekvenssteg införaren befinner sig i aktiveras olika utgångar. Här kontrolleras hela tiden så att grindskydd och nödstopp är OK. Införarens utgångar är lite speciella då kontrollen av tvåhandsgreppet är ett villkor som kontrolleras för att en utgång ska kunna bli aktiv under automatik. Vid manuellkörning kontrolleras villkor så att maskinen befinner sig i manuellt läge och att en knapp från HMI:et är aktiv.

4.2.7 Larm

Då en säkerhets-PLC var tänkt att implementeras tas bara signaler in för om grindskydd och nödstopp är OK. Om grindskydden öppnas eller ett nödstopp trycks in stannar alla rörelser och eventuell autodrift avslutas direkt.
I PLC programmet fanns ett summalarm som blir aktivt då något annat larm blir aktivt och sätter maskinen i stoppläge.
Det fanns i huvudsak tre larm som behandlades under summalarmen *timeoutlarm, motorskyddslarm och larm rörande hydrauloljan*.

Ett larm gick högt då en ingång på ingångsmodulen går hög. För att ett larm skulle kunna återställas var ingången tvungen att vara låg och en återställningsknapp på HMI:et vara hög.

4.2.8 Statistik

För att statistik skulle kunna presenteras på HMI:et gjordes en programdel för beräkningar av statistik. En variabel för att räkna upp tiden skapades och kallades *Puls_1sek*, denna kopplades till standardminnet SM412 i PLC:n som skickar en puls varje sekund.
Beräkningen av producerade och kasserade stenar baserades på att pressen befann sig i bestämda sekvenssteg och då räknades variabler upp som representerade producerat och kasserat stenar.

4.2.9 Speciallösningar för simulering

Simuleringen begränsades till delar av maskinen. Därför gjordes en del speciallösningar för att det skulle gå att simulera. Då brättmatningen, tvärbana och elevatorn uteslöts ur simuleringen var det tvunget att bygla ingången *X_lamnaPosfri* för att linjärenheten skulle köra normalt. För att linjärenheten ska köra måste även skrotcylindern befinna sig i ett av lägena skrotläge eller plockläge. Då det inte kommer att ske några skrotningar i simuleringen byglades *X_SkrotcylinderPlocklage*.
För att simuleringen skulle bli realistisk var det tvunget att skicka en extra signal från PLC till Arduinon att tvåhandsgreppet blev påverkat.

4.3 Framtagning av HMI

4.3.1 Teori

Ett HMI (Human Machine Interface) kan vara allt från en enkel operatörspanel med tryckknappar, till en kombinerad hård- och mjukvara där ett grafiskt gränssnitt gör det möjligt för operatören att följa och styra processen. Placering av ett HMI kan också variera från att vara en lite touchpanel ut i processen till en datorskärm i ett större kontrollrum.
HMI är den del av styrsystemet som sköter interaktionen mellan människan och maskinen. Det är från HMI som människan kan påverka styrningen och inställningar av processen.

4.3.2 Inledning

Dessa punkter ville vi uppfylla med vårt HMI.

- Lättnavigerat och användarvänligt
- Informerande
- Larmhantering ska finnas
- Manuell körning ska kunna göras från HMI
- Stilrent och enkelt

Skärmen som fanns att tillgå var av modellen iXT10A från Beijer Electronics. Till denna skärm användes iX Developer.
iX Developer är en mjukvara framtagen av Beijer Electronics. Den används för att skapa det grafiska gränssnittet för panelerna i Beijers iX-serie.

4.3.3 Kommunikation och taggar

iXT10A är utrustad med två olika kommunikationsformer. Den kan kommunicera över Ethernet eller seriellkommunikation. Önskemålet från företaget var att vi skulle användas oss av Ethernet vilket krävde en TCP/IP modul till PLC:n. Då projektet fick köpstopp kunde inte TCP/IP modulen handlas in och seriell kommunikation fick väljas då det alternativet är en inbyggd standard i den CPU som fanns att tillgå. Den standard av seriellkommunikation som var inbygg i CPU:n var RS232.
För att kommunikationen mellan PLC och HMI skulle fungera konfigurerades inställningar för vilken kommunikationsport på iXT10A:n som skulle användas. COM-port 1 valdes då det är denna som kunde ta emot seriellkommunikation av typen RS232. Transmissionshastigheten (baudrate) valdes till 9600 då det anses vara fullt tillräckligt.

I uppbyggnaden av ett HMI i iX Developer använder man sig av taggar. Taggar är en adress till en variabel. Taggar kan vara interna och bara användas inom HMI:et eller kopplas mot externa adresser i PLC:n. Taggar kunde väljas så att de antingen bara gick att läsa ifrån (Read) eller läsa och skriva (Read/Write). Karakteristiken på de externa taggarna kunde väljas till diskreta värden eller analoga. De externa taggarna tilldelades i adressfältet samma adress som de skulle kopplas mot i PLC:n. Hade inte karakteristik valts gjordes detta automatiskt av programmet.
Samtliga taggar kan ses i bilaga ???
Taggarna kopplades till de grafiska objekten i användargränssnittet och olika funktioner valdes.

4.3.4 Hemskärm

På hemskärmen presenteras statistik för dagsproduktionen. Detta görs för att förenkla för operatören så att han/hon vet när dagsproduktionen är uppnådd. Statistiken kan nollställas innan en ny dags produktion påbörjas. Det finns indikeringar för i vilket läge maskinen körs, automatik eller manuell drift. Vid uppstarten av automatisk drift dyker ett varningsmeddelande upp. Det informerar operatören om att man måste använda tvåhandsgreppet för att införaren ska gå till sitt startläge om det inte redan är i startläget. Detta varningsmeddelande försvinner när uppstarten är avslutad och maskinen är i full automatik.

Figur 18: HMI:ets hemskärm

4.3.5 Manuell

Under denna flik kan man köra maskinens delar manuellt. För att aktivera manuellt läge måste automatiken vara frånslagen. Så länge automatiken är till, syns ett varningsmeddelande och manuell läget går inte att starta. När manuell läget är startat dyker en undermeny upp för 6 olika sektioner av maskinen. Väljer man en av sektionerna presenteras en skärm med de olika rörelser man kan göra manuellt. Det är begränsat så att man bara kan göra en rörelse åt gången i HMI programmet. Detta medför att kollisioner och dubbelsignaler inte kan ske. För att avsluta det manuella läget återgår man till manuell startmeny eller så väljer man att slå till automatiken.

4.3.6 Inställningar

Operatören ska ha möjlighet att ändra parametrar vid körning. Dessa parametrar ändras under inställningar. De inställningar som kan göras är att ändra till vilka tryckgränser maskinen ska pressa. Detta vill man kunna ändra för att ämnets recept kan variera och kräver olika tryck.

4.3.7 Statistik
Här presenteras den totala produktionen och drifttiden. Detta sparas för att man ska kunna gå tillbaka och se hur mycket som producerats sedan man byggt om maskinen. Man vill även kunna se hur länge maskinen varit i drift sedan ombyggnation. Denna statistik kan endast nollställas genom programmeringen av PLC:n.

4.3.8 Larm
En larmlista som presenterar de olika larm som är eller har varit aktiva. Aktiva larm presenteras med rödfärg. När man bekräftat larmet ändras färgen till grön. Så länge larmet fortfarande är aktivt men bekräftat förblir det grönt. När larmet åtgärdats blir det vitt. Om ett larm blir åtgärdat innan det har bekräftats i larmtablån blir det gult.
För att larmen ska presenteras i tablån skapades en larmserver. I larmservern skapas de olika larmen som kopplas till larm i PLC:n. I larmservern görs inställningar för hur kvittering av larm ska hanteras.

4.3.9 Sekvensinfoformation
Varje sektion styrs i sekvenser. Sekvensernas steg presenteras för varje sektion. Man kan få mer detaljerad information om varje steg för respektive sektion. Detta gjordes för att underlätta felsökning och för att man ska kunna följa sekvenserna genom processen. Sekvensinformationen kan bara utläsas under automatisk drift.

4.3.10 Serviceläge
För att underlätta för underhållspersonal gjordes en flik för service. På denna presenteras alla givare ute på maskinen i en lista. Man kan se om givare är påverkade eller inte. Genom inloggning ges man rättigheter till att aktivera ett trelägesdon för bortkoppling av grindskydd. Man ges även rättigheter till mer avancerade inställningar till maskinen som inte operatören behöver ha tillgång till. Det är inställningar för larmgränser för hydrauloljan och timeouttider för maskinen.

5 Diskussion och analys

De uppgifter som sattes upp i inledningen av projektet har i stort sett uppfyllts. Ett PLC-program för hela maskinen har tagits fram, likaså ett väl fungerande HMI. Simuleringen begränsades till att bara innefatta maskinens huvuddelar nämligen *införare, pressning och linjärenhet*. Därför har inte hela maskinen kunnat simuleras.
Säkerheten höjs på maskinen i och med att spänningen kommer att sänkas vid kommande ombyggnation. Det föreslagna tvåhandsgreppet har implementerats i PLC-programmeringen och i simuleringen. En rad olika larmtyper har skapats för maskinen som inte har kunnat detekteras tidigare och som kommer förhindra mekaniska fel i ett tidigare skede. Maskinen kommer även att sänka sin strömförbrukning då rörelser kommer styras på tryck istället för tid. Vid tidsstyrning måste tiderna ställas med en säkerhetsmarginal så att resultat hinner uppnås och t.ex. hydraulmotorn kör längre än vad som egentligen behövs. Efter ombyggnationen kommer inte motorer behöva köras mer än precis vad som är nödvändigt.

Från början var tanken att vi i projektet skulle ta fram underlag för en ombyggnation och sedan vara med under densamma. Men i ett tidigt skede innan vi hade kommit så långt med elkonstruktionen för maskinen belades ett köpstopp på projektet från företagets sida. Detta för att maskinens framtid var oviss. Vi fick snabbt ändra projektets inriktning mot programmeringsbiten och istället införa en simulering för att kunna testa programmeringen som gjordes. Nu i slutet av vårt projekt har en ombyggnation kommit upp till diskussion igen och kommer troligen bli av under hösten 2015.

Under inledningen av projektet fick vi ett litet problem med vår simuleringsmiljö. Tanken från början var att använda oss av Arduinons egna programmeringsspråk. Vi hade gjort upp en grov plan för hur den skulle se ut, men planen var inte förenad med verkligheten och föll då mjukvaruavbrott inte stöds av det språket. Mjukvaruavbrott var tvunget att användas för att klara av att hantera parallella räknare för fördröjningar. Lösningen att programmera mikroprocessorn under Arduinons skal visade sig fungera lika bra och simuleringen blev lyckad.
Under programmeringen av simulatorn skaffade vi oss väldigt mycket kunskap om mekanik. Vi fick verkligen sätta oss in i mekaniken för maskinen för att kunna skapa en så realistisk simulering som möjligt.

Själva simuleringen kände vi var väldigt nyttig för oss som programmerade en maskin för första gången. Det var skönt att kunna testa våra olika lösningar utan risk att mekaniska delar kolliderar och går sönder. GxWorks2 har en inbyggd simulator men där styr man varje signal var för sig så det är svårt att testa längre sekvenser.

5.1 Framtida utvecklingsmöjligheter

Nästa steg för PLC-programmeringen hade varit att använda programmeringen på den faktiska maskinen och se om det fungerar lika bra i verkligheten som i simuleringen.

Vår simuleringsmiljö hade kunnat utvecklas på en rad punkter. Bland annat skulle man i detta projekt kunna ha med alla in och utsignaler för att kunna simulera hela maskinen. Då hade fler reläer behövts och ytterligare en Arduino. Vidare hade man kunna göra simulatorn standardiserad för att kunna användas till vilken simulering som helst för system med digitala signaler. Ett program hade kunnat skapas för att ställa in fördröjningstider utan att behöva röra själva Arduinons programmeringskod.

6 Referenser

1. http://www.arduino.cc/en/Main/ArduinoBoardMega2560 (Maj 2015)
   Arduino Mega 2560 produktsida

2. http://www.arduino.cc/en/uploads/Main/arduino-mega2560_R3-sch.pdf (Maj 2015)
   Kopplingsschema Arduino Mega 2560

3. http://www.arduino.cc/en/Hacking/PinMapping2560 (Maj 2015)
   Pinmapping Arduino Mega 2560

4. http://www.atmel.com/Images/Atmel-2549-8-bit-AVR-Microcontroller-ATmega640-1280-1281-2560-2561_datasheet.pdf (Maj 2015)
   Datablad, Atmel ATmega 2560 mikroprocessor

5. http://www.arduino.cc (Maj 2015)
   Arduinos hemsida

6. https://asensar.com/blog/2013/how-to-integrate-avrdude-with-atmel-studio/
   Inställningar för programmera Arduino med Atmel Studio

7. http://wiring.org.co/ (Maj 2015)
   Programspråket Wiring

8. http://akizukidenshi.com/download/CA3140E.pdf (Maj 2015)
   Operationsförstärkare CA3140E

9. http://www.ne.se.ludwig.lub.lu.se/uppslagsverk/encyklopedi/l%C3%A5ng/programmerbart-styrsystem (Maj 2015)
   Nationalencyklopedin

10. Bolton,W. 2000. *Programmable Logic Controllers 2nd edition.*
    Newnes

11. John, Karl-Heinz m.fl. 2010. *IEC 61131-3 Programming Industrial Automation Systems 2nd edition.* Springer

7 Bilagor

7.1 Programmeringskod Arduino

7.1.1 Arduino_CW.h

```
/*
 * CW_arduino.h
 */

#ifndef PWM_test_H_
#define PWM_test_H_

//Minne för att reseta programmet när nödstopp är
återställt
extern bool utlostNodstopp;

//Deklaration för att underlätta inställning av PINs
#define input 0
#define output 1

//Omskrivning av PORTnr till PINnr.
#define X0  PE0
#define X1  PE1
#define X2  PE4
#define X3  PE5
#define X4  PG5
#define X5  PE3
#define X6  PH3
#define X7  PH4
#define X8  PH5
#define X9  PH6
#define X10 PB4
#define X11 PB5
#define X12 PB6
#define X13 PB7
#define X14 PJ1
#define X15 PJ0
#define X16 PH1
#define X17 PH0
#define X18 PD3
#define X19 PD2
#define X20 PD1
#define X21 PD0
#define X22 PA0
#define X23 PA1
#define X24 PA2
#define X25 PA3
```

```
#define X26 PA4
#define X27 PA5
#define X28 PA6
#define X29 PA7
#define X30 PC7
#define X31 PC6
#define X32 PC5
#define X33 PC4
#define X34 PC3
#define X35 PC2
#define X36 PC1
#define X37 PC0
#define X38 PD7
#define X39 PG2
#define X40 PG1
#define X41 PG0
#define X42 PL7
#define X43 PL6
#define X44 PL5
#define X45 PL4
#define X46 PL3
#define X47 PL2
#define X48 PL1
#define X49 PL0
#define X50 PB3
#define X51 PB2
#define X52 PB1
#define X53 PB0


#endif /* CW_arduino_H_ */
```

7.1.2 Arduino_CW.c

```
/*
 * CW_arduino.c
 */

#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdbool.h>
#include <util/delay.h>
#include "CW_arduino.h"
#include "Actions.h"
#include "Define.h"

//Minne för att återställa till startläge efter nödstopp
bool utlostNodstopp = false;

//Sätter in- och utgångar
void Initialize(){

    DDRA=( input << X29) | ( input << X28) | ( input <<
    X27) | ( output << X26) | ( output << X25) | ( output
    << X24) | ( output << X23) | ( output << X22);

    DDRB=( output << X13) | ( output << X12) | ( output <<
    X11) | ( output << X10) | ( input << X50) | ( input <<
    X51) | ( input << X52) | ( input << X53);

    DDRC=( input << X30) | ( input << X31) | ( input <<
    X32) | ( input << X33) | ( input << X34) | ( input <<
    X35) | ( input << X36) | ( input << X37);

    DDRD=( input << X38) | ( output << X18) | ( output <<
    X19) | ( output << X20) | ( output << X21);

    DDRE=( output << X3) | ( output << X2) | ( output << X5
        ) | ( input << X1) | ( input << X0);

    DDRG=( output << X4) | ( input << X39) | ( input <<
    X40) | ( input << X41);

    DDRH=( output << X9) | ( output << X8) | ( output <<
    X7) | ( output << X6) | ( output << X16) | ( output <<
    X17);

    DDRJ=( output << X14) | ( output << X15);
```

```
    DDRL=( input << X42) | ( input << X43) | ( input <<
    X44) | ( input << X45) | ( input << X46) | ( input <<
    X47) | ( input << X48) | ( input << X49);
}

//Timer1 för avbrott till fördröjningar
void initOC1(){
    TCCR1A = 0;     // sätter TCCR1A registret till 0
    TCCR1B = 0;     // samma för TCCR1B
    OCR1A = 6249;           // sätter compare match
                            registret till valt räknarvärde
    TCCR1B |= (1 << WGM12); // startar CTC mode
    TCCR1B |= (1 << CS12);  // Sätter CS12 biten till 256
                             prescaler
    TIMSK1 |= (1 << OCIE1A);    // Startar timer compare
                                   interrupt
}



//Timer3 för PWM-signaler på PIN 2,3,5
void initOC3(){
    TCCR3A = 0; // sätter TCCR3A registret till 0
    TCCR3B = 0; // samma för TCCR3B
    TIMSK3 = 0; //Sätter TIMSK-registret till 0

    //Clear OC3A on Compare Match, set OC3A at BOTTOM(non-
inverting mode)
    TCCR3A |= (1 << COM3A1) | (0 << COM3A0);

    //Clear OC3B on Compare Match, set OC3B at BOTTOM(non-
inverting mode)
    TCCR3A |= (1 << COM3B1) | (0 << COM3B0);


    //Clear OC3C on Compare Match, set OC3C at BOTTOM(non-
inverting mode)
    TCCR3A |= (1 << COM3C1) | (0 << COM3C0);

    //sätter "Waveform Generation Mode" till Fast-PWM 8-bit
    TCCR3A |= (0 << WGM31) | (1 << WGM30);

    //sätter "Waveform Generation Mode" till Fast-PWM 8-bit

    TCCR3B |= (0 << WGM33) | (1 << WGM32);
```

```
    TCCR3B |= (1 << CS30);  //Startar klockan och sätter
prescaler till 1;
}

//Timer4 för PWM-signaler på PIN 6,7
void initOC4(){
    TCCR4A = 0; // sätter TCCR4A registret till 0
    TCCR4B = 0; // samma för TCCR4B
    TIMSK4 = 0; //Sätter TIMSK-registret till 0

    //Clear OC4A on Compare Match, set OC4A at BOTTOM(non-
    inverting mode)
    TCCR4A |= (1 << COM4A1) | (0 << COM4A0);


    //Clear OC4B on Compare Match, set OC4B at BOTTOM(non-
    inverting mode)
    TCCR4A |= (1 << COM4B1) | (0 << COM4B0);

    //sätter "Waveform Generation Mode" till Fast-PWM 8-bit
    TCCR4A |= (0 << WGM41) | (1 << WGM40);

    //sätter "Waveform Generation Mode" till Fast-PWM 8-bit

    TCCR4B |= (0 << WGM43) | (1 << WGM42);

    TCCR4B |= (1 << CS40);  //Startar klockan och sätter
prescaler till 1;
}

void startingPositions()
{
    //Digitala utgångar
    plockare_i_lamna_ON;
    vikare_nere_ON;
    amne_steg1_lamna_ON;
    amne_steg2_lamna_ON;
    uppstotare_uppe_ON;
    plockare_uppe_ON;
    plockare_vriden_ON;
    pressok_last_ON;
    pressok_uppe_ON;


    //PWM-signaler
    inlagg_positivt_inne;
    inlagg_negativt_inne;
```

```
    inlagg_inne = true;
    inlagg_ute = false;
    knivar_positivt_inne;
    knivar_negativt_inne;
    knivar_inne = true;
    knivar_ute = false;
    presstryck_low;

}

ISR(TIMER1_COMPA_vect)
{
    if(nodstoppOK & !utlostNodstopp){
        Fordrojning1();

        //Införaren får bara köra(räkna upp) om safeballen
    är intryckt, annars ska den stå still!
        if(safeball_intryckt){
            Fordrojning2();
            Fordrojning3();
            Fordrojning4();
        }

        Fordrojning6();
        Fordrojning7();
        Fordrojning8();
        Fordrojning9();
        Fordrojning10();
        inlagg_simulering();
        knivar_simulering();
    }

    if(nodstoppNotOK){
        utlostNodstopp = true;
    }

    //Sätter tillbaka till startvärden när nödstoppet är
    återställt.
    if(nodstoppOK & utlostNodstopp){
        utlostNodstopp = false;
        resetEfterNodstopp();
    }
}

int main(void)
{
```

```
    Initialize();   //Ställer in vilka PINs som ska vara
                    input/output
    startingPositions();    //Ställer alla cylindrar i
                            startläge och sätter analoga
                            startvärden.

    cli();          //pausar globala avbrott
    initOC1();      //Ställer in Timer1
    initOC3();      //Ställer in Timer3
    initOC4();      //Ställer in Timer4
    sei();          //startar globala avbrott



    while(1)
    {

    }
}
```

7.1.3 Actions.h

```
/*
 * Actions.h
 */


#ifndef ACTIONS_H_
#define ACTIONS_H_

#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdbool.h>


//Fördröjningar för luftcylindrar
extern void Fordrojning1(void);
extern void Fordrojning2(void);
extern void Fordrojning3(void);
extern void Fordrojning4(void);
extern void Fordrojning6(void);
extern void Fordrojning7(void);
extern void Fordrojning8(void);
extern void Fordrojning9(void);
```

```
//Fördröjningar av hydrauliska cylindrar
extern void Fordrojning10(void);
extern void inlagg_simulering(void);
extern void knivar_simulering(void);

//Funktion som återställer till startvärden efter nödstopp
extern void resetEfterNodstopp(void);

//Ändlägesvariabler för hydrauliska cylindrar
extern bool inlagg_inne;
extern bool inlagg_ute;
extern bool knivar_inne;
extern bool knivar_ute;


#endif /* ACTIONS_H_ */
```

7.1.4 Actions.c

```
/*
 * Actions.c
 */

#include "Actions.h"
#include "Define.h"

//Deklaration av variabler till fördröjningar
bool Fordrojning1out = false; bool Fordrojning1in = false;
bool Fordrojning2out = false; bool Fordrojning2in = false;
bool Fordrojning3out = false; bool Fordrojning3in = false;
bool Fordrojning4out = false; bool Fordrojning4in = false;
bool Fordrojning5out = false; bool Fordrojning5in = false;
bool Fordrojning6out = false; bool Fordrojning6in = false;
bool Fordrojning7out = false; bool Fordrojning7in = false;
bool Fordrojning8out = false; bool Fordrojning8in = false;
bool Fordrojning9out = false; bool Fordrojning9in = false;
bool Fordrojning10upp = false;
bool Fordrojning10in = false;
volatile int counter_for1 = 0;
volatile int counter_for2 = 0;
volatile int counter_for3 = 0;
volatile int counter_for4 = 0;
volatile int counter_for5 = 0;
volatile int counter_for6 = 0;
volatile int counter_for7 = 0;
volatile int counter_for8 = 0;
```

```
volatile int counter_for9 = 0;
volatile int counter_for10 = 0;


bool inlagg_inne = false; bool inlagg_ute = false;
bool inlagg_aktivt_ut = false;
bool inlagg_aktivt_in = false;
volatile int counterInlagg = 0;

bool knivar_inne = false; bool knivar_ute = false;
bool knivar_aktivt_ut = false;
bool knivar_aktivt_in = false;
volatile int counterKnivar = 0;

bool pressok_pavag_ner_lag = false;
bool pressok_pavag_ner_hog = false;
bool pressok_pavag_upp_lang = false;
bool pressok_pavag_upp_kort = false;
bool pressningOK = false; bool uppstartssekvens = true;
volatile int countInforarLamning = 0;


//Återställer arduinon till startläge efter ett utlöst
nödstopp.
void resetEfterNodstopp(){

    //Digitala utgångar
    Fordrojning1out = false; Fordrojning1in = false;
    Fordrojning2out = false;
    Fordrojning2in = false; Fordrojning3out = false;
    Fordrojning3in = false;
    Fordrojning4out = false; Fordrojning4in = false;
    Fordrojning5out = false;
    Fordrojning5in = false; Fordrojning6out = false;
    Fordrojning6in = false;
    Fordrojning7out = false; Fordrojning7in = false;
    Fordrojning8out = false;
    Fordrojning8in = false; Fordrojning9out = false;
    Fordrojning9in = false;
    Fordrojning10upp = false; Fordrojning10in = false;

    counter_for1 = 0; counter_for2 = 0; counter_for3 = 0;
    counter_for4 = 0;
    counter_for5 = 0; counter_for6 = 0; counter_for7 = 0;
    counter_for8 = 0;
    counter_for9 = 0; counter_for10 = 0;
```

```
    plockare_i_plock_ON;
    vikare_uppe_ON;
    amne_steg1_plock_ON;
    amne_steg2_plock_ON;
    uppstotare_uppe_ON;
    plockare_uppe_ON;
    plockare_ovriden_ON;
    pressok_last_ON;
    pressok_uppe_ON;


    //PWM-signaler
    inlagg_inne = true; inlagg_ute = false;
    inlagg_aktivt_ut = false; inlagg_aktivt_in = false;
    counterInlagg = 0;
    inlagg_positivt_inne;inlagg_negativt_inne;

    knivar_inne = true; knivar_ute = false;
    knivar_aktivt_ut = false; knivar_aktivt_in = false;
    counterKnivar = 0;
    knivar_positivt_inne;knivar_negativt_inne;

    presstryck_low;
    pressok_pavag_ner_lag = false;
    pressok_pavag_ner_hog = false;
    pressok_pavag_upp_lang = false;
    pressok_pavag_upp_kort = false;
    pressningOK = false;
}

//Plockare fram och tillbaka
void Fordrojning1(){
         counter_for1++;

    if(motor_plockare_fram & plockare_i_plock)
    {
        counter_for1 = 0;
        Fordrojning1out = true;
        plockare_i_plock_OFF;
    }

    if((counter_for1==tid_for1) & Fordrojning1out)
    {
        plockare_i_lamna_ON;
        Fordrojning1out = false;
    }
```

```
    if((motor_plockare_back) & plockare_i_lamna)
    {
        counter_for1 = 0;
        Fordrojning1in = true;
        plockare_i_lamna_OFF;
    }

    if((counter_for1 == tid_for1) & Fordrojning1in)
    {
        plockare_i_plock_ON;
        Fordrojning1in = false;
    }

}

//Vikare upp och ner
void Fordrojning2(){

    counter_for2++;

    if(vikare_upp & vikare_nere)
    {
        counter_for2 = 0;
        Fordrojning2out = true;
        vikare_nere_OFF;
    }

    if((counter_for2==tid_for2) & Fordrojning2out)
    {
        vikare_uppe_ON;
        Fordrojning2out = false;
    }

    if((vikare_ner) & vikare_uppe)
    {
        counter_for2 = 0;
        Fordrojning2in = true;
        vikare_uppe_OFF;
    }

    if((counter_for2 == tid_for2) & Fordrojning2in)
    {
        vikare_nere_ON;
        Fordrojning2in = false;
    }
}
```

```
//Ämne steg1
void Fordrojning3(){
    counter_for3++;

    if(amne_steg1_plocklage & amne_steg1_lamna )
    {
        counter_for3 = 0;
        Fordrojning3out = true;
        amne_steg1_lamna_OFF;
    }

    if((counter_for3==tid_for3) & Fordrojning3out)
    {
        amne_steg1_plock_ON;
        Fordrojning3out = false;
    }

    if(amne_steg1_lamnalage & amne_steg1_plock)
    {
        counter_for3 = 0;
        Fordrojning3in = true;
        amne_steg1_plock_OFF;
    }

    if((counter_for3 == tid_for3) & Fordrojning3in)
    {
        amne_steg1_lamna_ON;
        Fordrojning3in = false;
    }

}

//Ämne steg2 - Räknar även upp när ämne i form ska börja
påverkas.
void Fordrojning4(){
    counter_for4++;

    if(amne_steg2_plocklage & amne_steg2_lamna )
    {
        counter_for4 = 0;
        Fordrojning4out = true;
        amne_steg2_lamna_OFF;
    }

    if((counter_for4==tid_for4) & Fordrojning4out)
    {
        amne_steg2_plock_ON;
```

```
        Fordrojning4out = false;
    }

    if(amne_steg2_lamnalage & amne_steg2_plock)
    {
        counter_for4 = 0;
        Fordrojning4in = true;
        amne_steg2_plock_OFF;
        if(!uppstartssekvens){
            amne_i_form_OFF;
        }
    }

    if((counter_for4 == tid_for4) & Fordrojning4in)
    {
        amne_steg2_lamna_ON;
        Fordrojning4in = false;
        countInforarLamning++;
        if(countInforarLamning > 1){
            uppstartssekvens = false;
            amne_i_form_ON;
        }
        if(countInforarLamning > 2){
            amne_i_plocklage_ON;
        }

    }
}

//Uppstötare upp och ner
void Fordrojning6(){
    counter_for6++;

    if(uppstotare_upp & uppstotare_nere)
    {
        counter_for6 = 0;
        Fordrojning6out = true;
        uppstotare_nere_OFF;
        if(!uppstartssekvens){
            amne_i_form_ON;
        }
    }

    if((counter_for6==tid_for6) & Fordrojning6out)
    {
        uppstotare_uppe_ON;
        Fordrojning6out = false;
```

```
    }

    if(uppstotare_ner & uppstotare_uppe)
    {
        counter_for6 = 0;
        Fordrojning6in = true;
        uppstotare_uppe_OFF;
    }

    if((counter_for6 == tid_for6) & Fordrojning6in)
    {
        uppstotare_nere_ON;
        Fordrojning6in = false;
        if(!uppstartssekvens){
            amne_i_form_OFF;
        }
    }

}

//Plockare upp och ner
void Fordrojning7(){
    counter_for7++;

    if(plockare_upp & plockare_nere)
    {
        counter_for7 = 0;
        Fordrojning7out = true;
        plockare_nere_OFF;
    }

    if((counter_for7==tid_for7) & Fordrojning7out)
    {
        plockare_uppe_ON;
        Fordrojning7out = false;
        amne_i_plocklage_OFF;
    }

    if(plockare_ner & plockare_uppe)
    {
        counter_for7 = 0;
        Fordrojning7in = true;
        plockare_uppe_OFF;
    }

    if((counter_for7 == tid_for7) & Fordrojning7in)
```

```
    {
        plockare_nere_ON;
        Fordrojning7in = false;
    }

}

//Vridning av plockare
void Fordrojning8(){
    counter_for8++;

    if(plockare_vrid & plockare_ovriden)
    {
        counter_for8 = 0;
        Fordrojning8out = true;
        plockare_ovriden_OFF;
    }

    if((counter_for8==tid_for8) & Fordrojning8out)
    {
        plockare_vriden_ON;
        Fordrojning8out = false;
    }

    if(plockare_ovrid & plockare_vriden)
    {
        counter_for8 = 0;
        Fordrojning8in = true;
        plockare_vriden_OFF;
    }

    if((counter_for8 == tid_for8) & Fordrojning8in)
    {
        plockare_ovriden_ON;
        Fordrojning8in = false;
    }

}

//Låsning av pressok
void Fordrojning9(){
    counter_for9++;

    if(pressok_las & pressok_upplast)
    {
        counter_for9 = 0;
        Fordrojning9out = true;
```

```
        pressok_upplast_OFF;
    }

    if((counter_for9==tid_for9) & Fordrojning9out)
    {
        pressok_last_ON;
        Fordrojning9out = false;
    }

    if(pressok_lasupp & pressok_last)
    {
        counter_for9 = 0;
        Fordrojning9in = true;
        pressok_last_OFF;
    }

    if((counter_for9 == tid_for9) & Fordrojning9in)
    {
        pressok_upplast_ON;
        Fordrojning9in = false;
    }

}

//Pressok upp & ner, inkl tryck via PWM
void Fordrojning10(){
    counter_for10++;

    if(pressok_uppe & pressok_ner &
icke_avlastning_lagtryck){
        counter_for10 = 0;
        pressok_pavag_ner_lag = true;
        pressok_uppe_OFF;
        presstryck_low;
        pressningOK = false;
    }

    if((counter_for10==tid_for10_lag) &
pressok_pavag_ner_lag & icke_avlastning_lagtryck){
        pressok_nere_ON;
        amne_i_form_ON;
        presstryck_midd;
        pressok_pavag_ner_lag = false;
    }

    if(pressok_nere & icke_avlastning_hogtryck &
!pressningOK & !pressok_pavag_ner_hog){
```

```
        counter_for10 = 0;
        pressok_pavag_ner_hog = true;
        presstryck_midd;
    }

    if((counter_for10 == tid_for10_hog) &
pressok_pavag_ner_hog & icke_avlastning_hogtryck){
        presstryck_high;
        pressok_pavag_ner_hog = false;
        pressningOK = true;
    }

    if(pressok_upp & pressok_nere){
        counter_for10 = 0;
        presstryck_midd;
        pressok_pavag_upp_lang = true;
        pressok_nere_OFF;
        amne_i_form_OFF;
    }

    if((counter_for10==tid_for10_upp_lang) &
pressok_pavag_upp_lang){
        presstryck_low;
        pressok_uppe_ON;
        pressok_pavag_upp_lang = false;
    }

    if(pressok_upp & pressok_pavag_ner_lag){
        counter_for10 = 0;
        presstryck_low;
        pressok_pavag_upp_kort = true;
    }

    if((counter_for10==tid_for10_upp_kort) &
pressok_pavag_upp_kort){
        pressok_pavag_upp_kort = false;
        pressok_uppe_ON;
        presstryck_low;
    }


}

//Simulering av tryck för inlägg
void inlagg_simulering(void)
{
    counterInlagg++;
```

```
    if(inlagg_fram & inlagg_inne){
        inlagg_inne = false;
        inlagg_aktivt_ut = true;
        counterInlagg = 0;
        inlagg_negativt_i_rorelse;
        inlagg_positivt_i_rorelse;
    }

    if(inlagg_aktivt_ut & (counterInlagg == tid_inlagg)){
        inlagg_aktivt_ut = false;
        inlagg_ute = true;
        inlagg_positivt_ute;
        inlagg_negativt_ute;
    }

    if(inlagg_tillbaka & inlagg_ute){
        inlagg_ute = false;
        inlagg_aktivt_in = true;
        counterInlagg = 0;
        inlagg_negativt_i_rorelse;
        inlagg_positivt_i_rorelse;
    }
    if(inlagg_aktivt_in & (counterInlagg == tid_inlagg)){
        inlagg_aktivt_in = false;
        inlagg_inne = true;
        inlagg_positivt_inne;
        inlagg_negativt_inne;
    }
}

//Simulering av tryck för knivar
void knivar_simulering(void)
{
    counterKnivar++;

    if(knivar_fram & knivar_inne){
        knivar_inne = false;
        knivar_aktivt_ut = true;
        counterKnivar = 0;
        knivar_negativt_i_rorelse;
        knivar_positivt_i_rorelse;
    }

    if(knivar_aktivt_ut & (counterKnivar == tid_knivar)){
        knivar_aktivt_ut = false;
        knivar_ute = true;
```

```
        knivar_positivt_ute;
        knivar_negativt_ute;
    }

    if(knivar_tillbaka & knivar_ute){
        knivar_ute = false;
        knivar_aktivt_in = true;
        counterKnivar = 0;
        knivar_negativt_i_rorelse;
        knivar_positivt_i_rorelse;
    }
    if(knivar_aktivt_in & (counterKnivar == tid_knivar)){
        knivar_aktivt_in = false;
        knivar_inne = true;
        knivar_positivt_inne;
        knivar_negativt_inne;
    }
}
```

7.1.5 Define.h

```
/*
 * Define.h
 */


#ifndef DEFINE_H_
#define DEFINE_H_

/*
***********FÖRDRÖJNINGSTIDER***********
Tider i millisekunder.
*/
#define tid_for1 43     //Plockare fram å tillbaka
#define tid_for2 5      //Vikare
#define tid_for3 20     //Ämne steg1
#define tid_for4 22     //Ämne steg2
#define tid_for6 20     //Uppstötare
#define tid_for7 20     //Plockare upp å ner
#define tid_for8 5      //Vridning
#define tid_for9 5      //Låsning av pressok
#define tid_for10_lag 20        //Pressok ner till ämne
#define tid_for10_hog 10        //Slutpressning
#define tid_for10_upp_kort 4    //Pressok upp innan ämne
#define tid_for10_upp_lang 10   //Pressok upp efter
pressning
```

```
#define tid_inlagg 10       //Inlägg
#define tid_knivar 28       //Knivar


/*
***********UTGÅNGAR***********
ON = "PORT? = PORT? | 0x00", där den givna biten på
position X sätts till 1.
OFF =  "PORT? = PORT? & 0xFF", där den givna biten på
position X sätts till 0.
*/
#define plockare_i_lamna        ((PORTB & 0x20) == 0x20)
#define plockare_i_lamna_ON (PORTB = PORTB | 0x20)
#define plockare_i_lamna_OFF    (PORTB = PORTB & 0xDF)
#define plockare_i_plock        ((PORTB & 0x10) == 0x10)
#define plockare_i_plock_ON (PORTB = PORTB | 0x10)
#define plockare_i_plock_OFF    (PORTB = PORTB & 0xEF)
#define plockare_nere       ((PORTH & 0x40) == 0x40)
#define plockare_nere_ON        (PORTH = PORTH | 0x40)
#define plockare_nere_OFF   (PORTH = PORTH & 0xBF)
#define plockare_uppe       ((PORTG & 0x20) == 0x20)
#define plockare_uppe_ON        (PORTG = PORTG | 0x20)
#define plockare_uppe_OFF   (PORTG = PORTG & 0xDF)
#define pressok_last        ((PORTA & 0x08) == 0x08)
#define pressok_last_ON     (PORTA = PORTA | 0x08)
#define pressok_last_OFF        (PORTA = PORTA & 0xF7)
#define pressok_upplast     ((PORTA & 0x10) == 0x10)
#define pressok_upplast_ON  (PORTA = PORTA | 0x10)
#define pressok_upplast_OFF (PORTA = PORTA & 0xEF)
#define pressok_uppe        ((PORTJ & 0x02) == 0x02)
#define pressok_uppe_ON     (PORTJ = PORTJ | 0x02)
#define pressok_uppe_OFF        (PORTJ = PORTJ & 0xFD)
#define pressok_nere        ((PORTJ & 0x01) == 0x01)
#define pressok_nere_ON     (PORTJ = PORTJ | 0x01)
#define pressok_nere_OFF        (PORTJ = PORTJ & 0xFE)
#define uppstotare_nere     ((PORTA & 0x02) == 0x02)
#define uppstotare_nere_ON  (PORTA = PORTA | 0x02)
#define uppstotare_nere_OFF (PORTA = PORTA & 0xFD)
#define uppstotare_uppe     ((PORTA & 0x01) == 0x01)
#define uppstotare_uppe_ON  (PORTA = PORTA | 0x01)
#define uppstotare_uppe_OFF (PORTA = PORTA & 0xFE)
#define vikare_nere     ((PORTH & 0x01) == 0x01)
#define vikare_nere_ON      (PORTH = PORTH | 0x01)
#define vikare_nere_OFF     (PORTH = PORTH & 0xFE)
#define vikare_uppe     ((PORTH & 0x02) == 0x02)
#define vikare_uppe_ON      (PORTH = PORTH | 0x02)
#define vikare_uppe_OFF     (PORTH = PORTH & 0xFD)
```

```
#define plockare_ovriden        ((PORTB & 0x80) == 0x80)
#define plockare_ovriden_ON (PORTB = PORTB | 0x80)
#define plockare_ovriden_OFF    (PORTB = PORTB & 0x7F)
#define plockare_vriden     ((PORTB & 0x40) == 0x40)
#define plockare_vriden_ON  (PORTB = PORTB | 0x40)
#define plockare_vriden_OFF (PORTB = PORTB & 0xBF)
#define amne_i_plocklage        ((PORTH & 0x20) == 0x20)
#define amne_i_plocklage_ON (PORTH = PORTH | 0x20)
#define amne_i_plocklage_OFF    (PORTH = PORTH & 0xDF)
#define amne_steg1_lamna        ((PORTD & 0x08) == 0x08)
#define amne_steg1_lamna_ON (PORTD = PORTD | 0x08)
#define amne_steg1_lamna_OFF    (PORTD = PORTD & 0xF7)
#define amne_steg1_plock        ((PORTD & 0x04) == 0x04)
#define amne_steg1_plock_ON (PORTD = PORTD | 0x04)
#define amne_steg1_plock_OFF    (PORTD = PORTD & 0xFB)
#define amne_steg2_lamna        ((PORTD & 0x02) == 0x02)
#define amne_steg2_lamna_ON (PORTD = PORTD | 0x02)
#define amne_steg2_lamna_OFF    (PORTD = PORTD & 0xFD)
#define amne_steg2_plock        ((PORTD & 0x01) == 0x01)
#define amne_steg2_plock_ON (PORTD = PORTD | 0x01)
#define amne_steg2_plock_OFF    (PORTD = PORTD & 0xFE)
#define amne_i_form     ((PORTA & 0x04) == 0x04)
#define amne_i_form_ON      (PORTA = PORTA | 0x04)
#define amne_i_form_OFF     (PORTA = PORTA & 0xFB)


/*
***********INGÅNGAR***********
Kollar om ingången är hög
*/
#define motor_plockare_fram     ((PINB & 0x01) == 0x01)
#define motor_plockare_back     ((PINB & 0x02) == 0x02)
#define vikare_upp          ((PINL & 0x10) == 0x10)
#define vikare_ner          ((PINL & 0x20) == 0x20)
#define amne_steg1_plocklage        ((PINL & 0x80) == 0x80)
#define amne_steg1_lamnalage        ((PINL & 0x40) == 0x40)
#define amne_steg2_plocklage        ((PING & 0x02) == 0x02)
#define amne_steg2_lamnalage        ((PING & 0x01) == 0x01)
#define uppstotare_upp          ((PING & 0x04) == 0x04)
#define uppstotare_ner          ((PIND & 0x80) == 0x80)
#define plockare_upp            ((PINL & 0x01) == 0x01)
#define plockare_ner            ((PINL & 0x02) == 0x02)
#define plockare_vrid           ((PINB & 0x04) == 0x04)
#define plockare_ovrid          ((PINB & 0x08) == 0x08)
#define pressok_upp         ((PINL & 0x04) == 0x04)
#define pressok_ner         ((PINL & 0x08) == 0x08)
#define pressok_las         ((PINC & 0x80) == 0x80)
```

```
#define pressok_lasupp          ((PINA & 0x80) == 0x80)
#define nodstoppOK          ((PINA & 0x40) == 0x00)
#define nodstoppNotOK           ((PINA & 0x40) == 0x40)
#define avlastning_lagtryck     ((PINC & 0x20) == 0x20)
#define icke_avlastning_lagtryck        ((PINC & 0x20) ==
0x00)
#define avlastning_hogtryck     ((PINC & 0x40) == 0x40)
#define icke_avlastning_hogtryck        ((PINC & 0x40) ==
0x00)
#define inlagg_fram         ((PINC & 0x01) == 0x01)
#define inlagg_tillbaka         ((PINC & 0x02) == 0x02)
#define knivar_fram         ((PINC & 0x04) == 0x04)
#define knivar_tillbaka         ((PINC & 0x08) == 0x08)
#define safeball_intryckt       ((PINC & 0x10) == 0x10)


/*
***********PWM-nivåer***********
Olika nivåer på PWM för att simulera tryckvärden till PLCn.
Håller konstanta värden
*/
#define inlagg_positivt_i_rorelse   (OCR3B = 127)
#define inlagg_positivt_ute     (OCR3B = 250)
#define inlagg_positivt_inne        (OCR3B = 90)
#define inlagg_negativt_i_rorelse   (OCR3C = 127)
#define inlagg_negativt_ute     (OCR3C = 90)
#define inlagg_negativt_inne        (OCR3C = 250)
#define knivar_positivt_i_rorelse   (OCR4A = 127)
#define knivar_positivt_ute     (OCR4A = 250)
#define knivar_positivt_inne        (OCR4A = 90)
#define knivar_negativt_i_rorelse   (OCR4B = 127)
#define knivar_negativt_ute     (OCR4B = 90)
#define knivar_negativt_inne        (OCR4B = 250)
#define presstryck_low          (OCR3A = 90)
#define presstryck_midd         (OCR3A = 127)
#define presstryck_high         (OCR3A = 250)



#endif /* DEFINE_H_ */
```

7.2 PLC programmering

7.2.1 Globala variabler

Benämning	Typ	Adress
Hojdkontroll_OK	BOOL	TRUE
AktTotTidMaskinSek	DINT	D100
AktPerTidMaskinSek	DINT	D102
TotTidMaskinSek	DINT	D104
TotTidMaskinMin	DINT	D106
TotTidMaskinTim	DINT	D108
PerTidMaskinSek	DINT	D110
PerTidMaskinMin	DINT	D112
PerTidMaskinTim	DINT	D114
AktTotAntalTegel	DINT	D116
AktPerAntalTegel	DINT	D118
AktTotAntalKass	DINT	D120
AktPerAntalKass	DINT	D122
TotGkdTegel	DINT	D124
PerGkdTegel	DINT	D126
AkttryckPressOK	INT	D200
AkttryckInlaggPos	INT	D201
AkttryckInlaggNeg	INT	D202
AkttryckKnivarPos	INT	D203
AkttryckKnivarNeg	INT	D204
AktTempHydraultank	INT	D205
RawDataCH1	INT	D701
RawDataCH2	INT	D702
RawDataCH3	INT	D703
RawDataCH4	INT	D704
RawDataCH5	INT	D705
RawDataCH6	INT	D706
Larm_MSK_UlostHydralik	BOOL	M100
Larm_MSK_UlostPlockare	BOOL	M101
Larm_MSK_UlostReturband	BOOL	M102
Larm_MSK_UlostBrattband	BOOL	M103
Larm_MSK_UlostBrattbandRullar	BOOL	M104
Larm_MSK_UlostTvarbana	BOOL	M105
Larm_MSK_UlostElevator	BOOL	M106
Larm_HydralOljaHogTemp	BOOL	M107
Larm_HydralOljaLagNiva	BOOL	M108
Larm_HydralOljaLagTemp	BOOL	M109

Larm_GrindSkydd	BOOL	M110
Larm_NodStopp	BOOL	M111
Larm_TimeOutInlaggIn	BOOL	M112
Larm_TimeOutInlaggUt	BOOL	M113
Larm_TimeOutKnivarIn	BOOL	M114
Larm_TimeOutKnivarUt	BOOL	M115
Larm_TimeOutPressOkUpp	BOOL	M116
Larm_TimeOutPressOkNer	BOOL	M117
Larm_TimeOutPressOkNerHog	BOOL	M118
Larm_TimeOutUppstotareUpp	BOOL	M119
Larm_TimeOutUppstotareNer	BOOL	M120
Larm_TimeOutPressOkLasning	BOOL	M121
Larm_TimeOutPressOkUpplasning	BOOL	M122
Larm_VakuumLag	BOOL	M123
Larm_ElevatorMaxUppe	BOOL	M125
Larm_TruckIElevatorZon	BOOL	M126
Larm_LuftIn	BOOL	M127
Larm_TimeOutBrattbanaRullar	BOOL	M128
Larm_TimeOutVikareUpp	BOOL	M129
Larm_TimeOutVikareNer	BOOL	M130
Larm_TimeOutAmneSteg1Plock	BOOL	M131
Larm_TimeOutAmneSteg1Lamna	BOOL	M132
Larm_TimeOutAmneSteg2Plock	BOOL	M133
Larm_TimeOutAmneSteg2Lamna	BOOL	M134
Larm_TimeOutPlockareVrid	BOOL	M135
Larm_TimeOutPlockareOvrid	BOOL	M136
Larm_TimeOutPlockareUpp	BOOL	M137
Larm_TimeOutPlockareNer	BOOL	M138
Larm_TimeOutPlockareFram	BOOL	M139
Larm_TimeOutPlockareBack	BOOL	M140
Larm_TimeOutMotorBrattbana	BOOL	M141
Larm_TimeOutTvarbanaUpp	BOOL	M142
Larm_TimeOutMotorTvarbanaKort	BOOL	M143
Larm_TimeOutMotorTvarbanaLang	BOOL	M144
Larm_TimeOutElevatorUpp	BOOL	M145
LarmKV_MSK_UlostHydralik	BOOL	M200
LarmKV_MSK_UlostPlockare	BOOL	M201
LarmKV_MSK_UlostReturband	BOOL	M202
LarmKV_MSK_UlostBrattband	BOOL	M203
LarmKV_MSK_UlostBrattbandrullar	BOOL	M204
LarmKV_MSK_UlostTvarbana	BOOL	M205

LarmKV_MSK_UlostElevator	BOOL	M206
LarmKV_HydralOljaHogTemp	BOOL	M207
LarmKV_HydralOljaLagTemp	BOOL	M208
LarmKV_HydralOljaLagNiva	BOOL	M209
LarmKV_TimeOutInlaggIn	BOOL	M212
LarmKV_TimeOutInlaggUt	BOOL	M213
LarmKV_TimeOutKnivarIn	BOOL	M214
LarmKV_TimeOutKnivarUt	BOOL	M215
LarmKV_TimeOutPressOkUpp	BOOL	M216
LarmKV_TimeOutPressOkNer	BOOL	M217
LarmKV_TimeOutPressOkNerHog	BOOL	M218
LarmKV_TimeOutUppstotareUpp	BOOL	M219
LarmKV_TimeOutUppstotareNer	BOOL	M220
LarmKV_TimeOutPressOkLasning	BOOL	M221
LarmKV_TimeOutPressOkUpplasning	BOOL	M222
LarmKV_VakuumLag	BOOL	M223
LarmKV_ElevatorMaxUppe	BOOL	M225
LarmKV_TruckIElevatorZon	BOOL	M226
LarmKV_LuftIn	BOOL	M227
LarmKV_TimeOutBrattbanaRulllar	BOOL	M228
LarmKV_TimeOutVikareUpp	BOOL	M229
LarmKV_TimeOutVikareNer	BOOL	M230
LarmKV_TimeOutAmneSteg1Plock	BOOL	M231
LarmKV_TimeOutAmneSteg1Lamna	BOOL	M232
LarmKV_TimeOutAmneSteg2Plock	BOOL	M233
LarmKV_TimeOutAmneSteg2Lamna	BOOL	M234
LarmKV_TimeOutPlockareVrid	BOOL	M235
LarmKV_TimeOutPlockareOvrid	BOOL	M236
LarmKV_TimeOutPlockareUpp	BOOL	M237
LarmKV_TimeOutPlockareNer	BOOL	M238
LarmKV_TimeOutPlockareFram	BOOL	M239
LarmKV_TimeOutPlockareBack	BOOL	M240
LarmKV_TimeOutMotorBrattbana	BOOL	M241
LarmKV_TimeOutTvarbanaUpp	BOOL	M242
LarmKV_TimeOutMotorTvarbanaKort	BOOL	M243
LarmKV_TimeOutMotorTvarbanaLang	BOOL	M244
LarmKV_TimeOutElevatorUpp	BOOL	M245
SummaLarmStopp	BOOL	M300
Auto	BOOL	M350
Uppstart_Auto	BOOL	M351
Avslutar_Auto	BOOL	M352

ServiceLage	BOOL	M353
ManuelltLage	BOOL	M354
ElevatorFull	BOOL	M400
Safeball_slappt	BOOL	M401
OP_TomningElevator	BOOL	M500
OP_ResetCounter	BOOL	M501
OP_ManuelltLageON	BOOL	M502
OP_ManuelltLageOFF	BOOL	M503
OP_HandPressOkLasOppna	BOOL	M600
OP_HandPressOkLasStang	BOOL	M601
OP_HandPressOkNer	BOOL	M602
OP_HandPressOkUpp	BOOL	M603
OP_HandUppstotareUpp	BOOL	M604
OP_HandUppstotareNer	BOOL	M605
OP_HandInlaggIn	BOOL	M606
OP_HandInlaggUt	BOOL	M607
OP_HandKnivarIn	BOOL	M608
OP_HandKnivarUt	BOOL	M609
OP_HandPlockareNer	BOOL	M610
OP_HandPlockareUpp	BOOL	M611
OP_HandPlockareVrid	BOOL	M612
OP_HandPlockareOvrid	BOOL	M613
OP_HandPlockareFram	BOOL	M614
OP_HandPlockareBack	BOOL	M615
OP_HandLasPlockareOppna	BOOL	M616
OP_HandSkrotcylinderSkrot	BOOL	M617
OP_HandSkrotcylinerPlock	BOOL	M618
OP_HandVikareNer	BOOL	M619
OP_HandVikareUpp	BOOL	M620
OP_HandAmneSteg1Lamna	BOOL	M621
OP_HandAmneSteg1Plocka	BOOL	M622
OP_HandAmneSteg2Lamna	BOOL	M623
OP_HandAmneSteg2Plocka	BOOL	M624
OP_HandElevatorUpp	BOOL	M625
OP_HandElevatorNer	BOOL	M626
OP_HandMotorBrattbanaFram	BOOL	M627
OP_HandMotorBrattbanaRullarFram	BOOL	M628
OP_HandTvarbanaUpp	BOOL	M629
OP_HandMotorTvarbanaFram	BOOL	M630
OP_HandSkrapranna	BOOL	M631
OP_HandMotorSkrotband	BOOL	M632

InforareStegInit	BOOL	M1000
InforareSteg1	BOOL	M1001
InforareSteg2	BOOL	M1002
InforareSteg3	BOOL	M1003
InforareSteg4	BOOL	M1004
InforareSteg5	BOOL	M1005
InforareSteg6	BOOL	M1006
PressStegInit	BOOL	M1050
PressSteg1	BOOL	M1051
PressSteg2	BOOL	M1052
PressSteg3	BOOL	M1053
PressSteg4	BOOL	M1054
PressSteg5	BOOL	M1055
PressSteg6	BOOL	M1056
PressSteg7	BOOL	M1057
PressSteg8	BOOL	M1058
PressSteg9	BOOL	M1059
PressSteg10	BOOL	M1060
LinjarenhetStegInit	BOOL	M1100
LinjarenhetSteg1	BOOL	M1101
LinjarenhetSteg2	BOOL	M1102
LinjarenhetSteg3	BOOL	M1103
LinjarenhetSteg4	BOOL	M1104
LinjarenhetSteg5	BOOL	M1105
LinjarenhetSteg6	BOOL	M1106
LinjarenhetSteg7	BOOL	M1107
LinjarenhetSteg8	BOOL	M1108
LinjarenhetSteg9	BOOL	M1109
LinjarenhetSteg10	BOOL	M1110
LinjarenhetSteg15	BOOL	M1115
LinjarenhetSteg16	BOOL	M1116
LinjarenhetSteg17	BOOL	M1117
BrattbanaStegInit	BOOL	M1150
BrattbanaSteg1	BOOL	M1151
BrattbanaSteg2	BOOL	M1152
ElevatorStegInit	BOOL	M1200
ElevatorSteg1	BOOL	M1201
ElevatorSteg2	BOOL	M1202
TvarbanaStegInit	BOOL	M1250
TvarbanaSteg1	BOOL	M1251
TvarbanaSteg2	BOOL	M1252

TvarbanaSteg3	BOOL	M1253
TvarbanaSteg4	BOOL	M1254
TvarbanaSteg5	BOOL	M1255
Puls_1sek	BOOL	SM412
OP_TryckInlagg	INT	W100
OP_TryckKnivar	INT	W101
OP_TryckLagPressOk	INT	W102
OP_TryckHogPressOk	INT	W103
OP_ProduktKontroll	INT	W104
OP_TimeOutSekvenser	INT	W110
OP_LarmgTempOljaHog	INT	W112
OP_LarmgTempOljaLag	INT	W113
TidSekvenserMilisekunder	TIME	W114
StegInfoInforare	INT	W300
StegInfoPress	INT	W301
StegInfoLinjarenhet	INT	W302
StegInfoBrattbana	INT	W303
StegInfoElevator	INT	W304
StegInfoTvarbana	INT	W305
X_plockarePlocklage	BOOL	X0
X_plockareLamnalage	BOOL	X1
X_plockareUppe	BOOL	X2
X_plockareNere	BOOL	X3
X_plockareVriden	BOOL	X4
X_plockareOvriden	BOOL	X5
X_stenIplocklAge	BOOL	X6
X_lamnaPosfri	BOOL	X7
X_pressOkUppe	BOOL	X8
X_pressOKnere	BOOL	X9
X_Safeball	BOOL	X0A
X_vikareUppe	BOOL	X0B
X_vikareNere	BOOL	X0C
X_givareAmneSteg1Lamna	BOOL	X0D
X_givareAmneSteg1Plocka	BOOL	X0E
X_givareAmneSteg2Lamna	BOOL	X0F
X_givareAmneSteg2Plocka	BOOL	X10
X_uppstotareUppe	BOOL	X11
X_uppstotareNere	BOOL	X12
X_SkrotcylinderPlocklage	BOOL	X13
X_fotocellAmneIForm	BOOL	X14
X_pressOKLast	BOOL	X15

X_pressOKOlast	BOOL	X16
X_SkrotcylinderSkrotlage	BOOL	X17
X_NodStoppOK	BOOL	X18
X_GrindSkyddOk	BOOL	X19
X_starManSkap	BOOL	X1A
X_stopManSkap	BOOL	X1B
X_KvittElevatorHamtning	BOOL	X1C
X_TreLagesDon	BOOL	X1D
X_BrattmagasinOK	BOOL	X1E
X_BrattmatningKam1	BOOL	X1F
X_BrattmatningKam2	BOOL	X20
X_BrattmatningKam3	BOOL	X21
X_BrattmatningFrammeTvarbana	BOOL	X22
X_TvarbanaUppe	BOOL	X23
X_Bratt1Tvarbana	BOOL	X24
X_BrattElevatorHoger	BOOL	X25
X_BrattElevatorVanster	BOOL	X26
X_ElevatorHamtlage	BOOL	X27
X_ElevatorMaxUppe	BOOL	X28
X_TruckIElevatorZon1	BOOL	X29
X_TruckIElevatorZon2	BOOL	X2A
X_FotocellElevatorMagFullt	BOOL	X2B
X_TryckVaktInLuft	BOOL	X2C
X_VakuumVaktPlockare	BOOL	X2D
X_NivaVaktHydralTank	BOOL	X2E
X_HydralMotorMSKUtlost	BOOL	X2F
X_PlockarMotorMSKUtlost	BOOL	X30
X_ReturBandMotorMSKUtlost	BOOL	X31
X_BrattBandMotorMSKUtlost	BOOL	X32
X_BrattBandRullarMotorMSKUtlost	BOOL	X33
X_TvarbanaMotorMSKUtlost	BOOL	X34
X_ElevatorMotorMSKUtlost	BOOL	X35
X36_Reserv	BOOL	X36
X37_Reserv	BOOL	X37
X38_Reserv	BOOL	X38
X39_Reserv	BOOL	X39
X3A_Reserv	BOOL	X3A
X3B_Reserv	BOOL	X3B
X3C_Reserv	BOOL	X3C
X3D_Reserv	BOOL	X3D
X3E_Reserv	BOOL	X3E

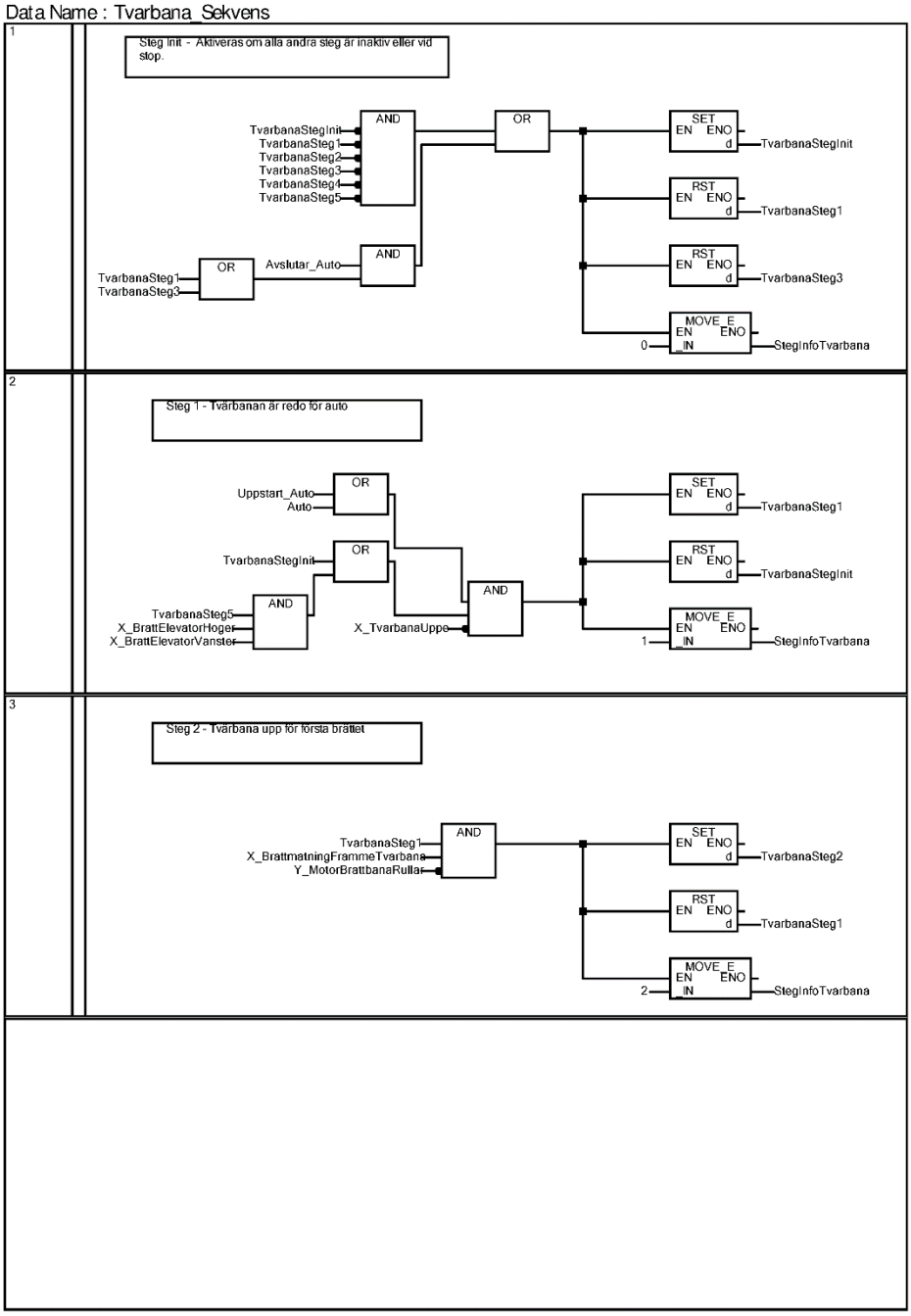
X3F_Resev	BOOL	X3F
Y_plockareFram	BOOL	Y40
Y_plockareBack	BOOL	Y41
Y_ÄmneIPlocklägeÅter	BOOL	Y42
Y_plockareVrid	BOOL	Y43
Y_plockareOvrid	BOOL	Y44
Y_plockareUpp	BOOL	Y45
Y_plockareNer	BOOL	Y46
Y_vakuumPa	BOOL	Y47
Y_LasIRattLage	BOOL	Y48
Y_pressokUPP	BOOL	Y49
Y_pressokNER	BOOL	Y4A
Y_vikareUpp	BOOL	Y4B
Y_vikareNer	BOOL	Y4C
Y_ventilAmneSteg1Lamna	BOOL	Y4D
Y_ventilAmneSteg1Plocka	BOOL	Y4E
Y_ventilAmneSteg2Lamna	BOOL	Y4F
Y_ventilAmneSteg2Plocka	BOOL	Y50
Y_HydraulMotor	BOOL	Y51
Y_uppstotareUpp	BOOL	Y52
Y_uppstotareNer	BOOL	Y53
Y_inlaggIN	BOOL	Y54
Y_inlaggUT	BOOL	Y55
Y_knivarIN	BOOL	Y56
Y_knivarUT	BOOL	Y57
Y_Safeball_paverkad	BOOL	Y58
Y_AvlastningLagtryck	BOOL	Y59
Y_AvlastningHogtryck	BOOL	Y5A
Y_pressokLAS	BOOL	Y5B
Y_pressokOLAS	BOOL	Y5C
Y_MotorBrattbana	BOOL	Y5D
Y_MotorBrattbanaRullar	BOOL	Y5E
Y_MotorTvarbana	BOOL	Y5F
Y_TvarbanaUpp	BOOL	Y60
Y_TvarbanaNer	BOOL	Y61
Y_ElevatorUpp	BOOL	Y62
Y_ElevatorNer	BOOL	Y63
Y_ElevatorFull	BOOL	Y64
Y_MotorReturband	BOOL	Y65
Y_Skrapranna	BOOL	Y66
Y_LuftPa	BOOL	Y67

Y_UtlostNodStopp	BOOL	Y68
Y_SkrotcylinderSkrotlage	BOOL	Y69
Y6A_Reserv	BOOL	Y6A
Y6B_Reserv	BOOL	Y6B
Y6C_Reserv	BOOL	Y6C
Y6D_Reserv	BOOL	Y6D
Y6E_Reserv	BOOL	Y6E
Y_SkrotcylinderPlocklage	BOOL	Y6F

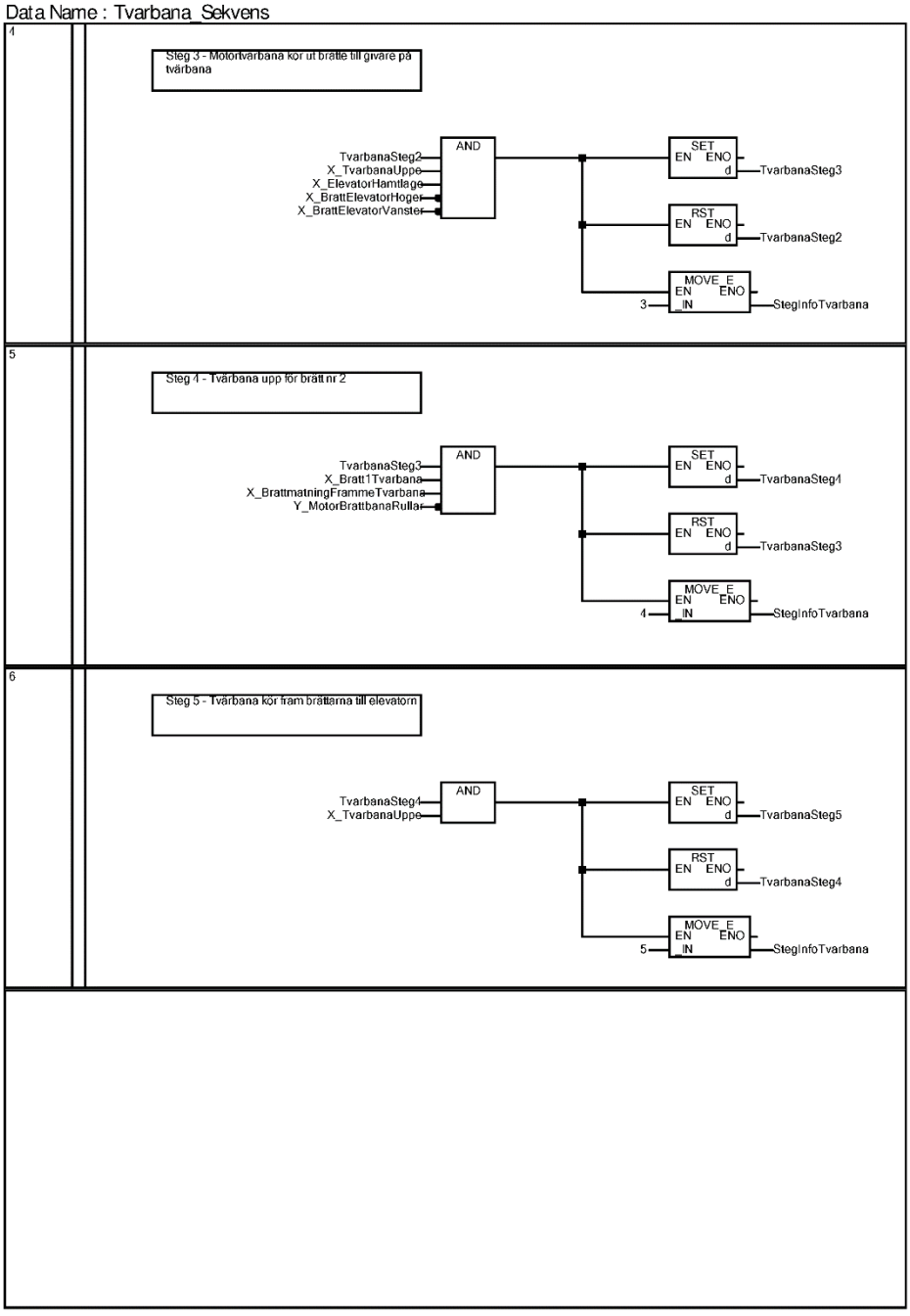
7.2.2 Strukturerad Ladder

På följande sidor finns PLC-programmen presenterade

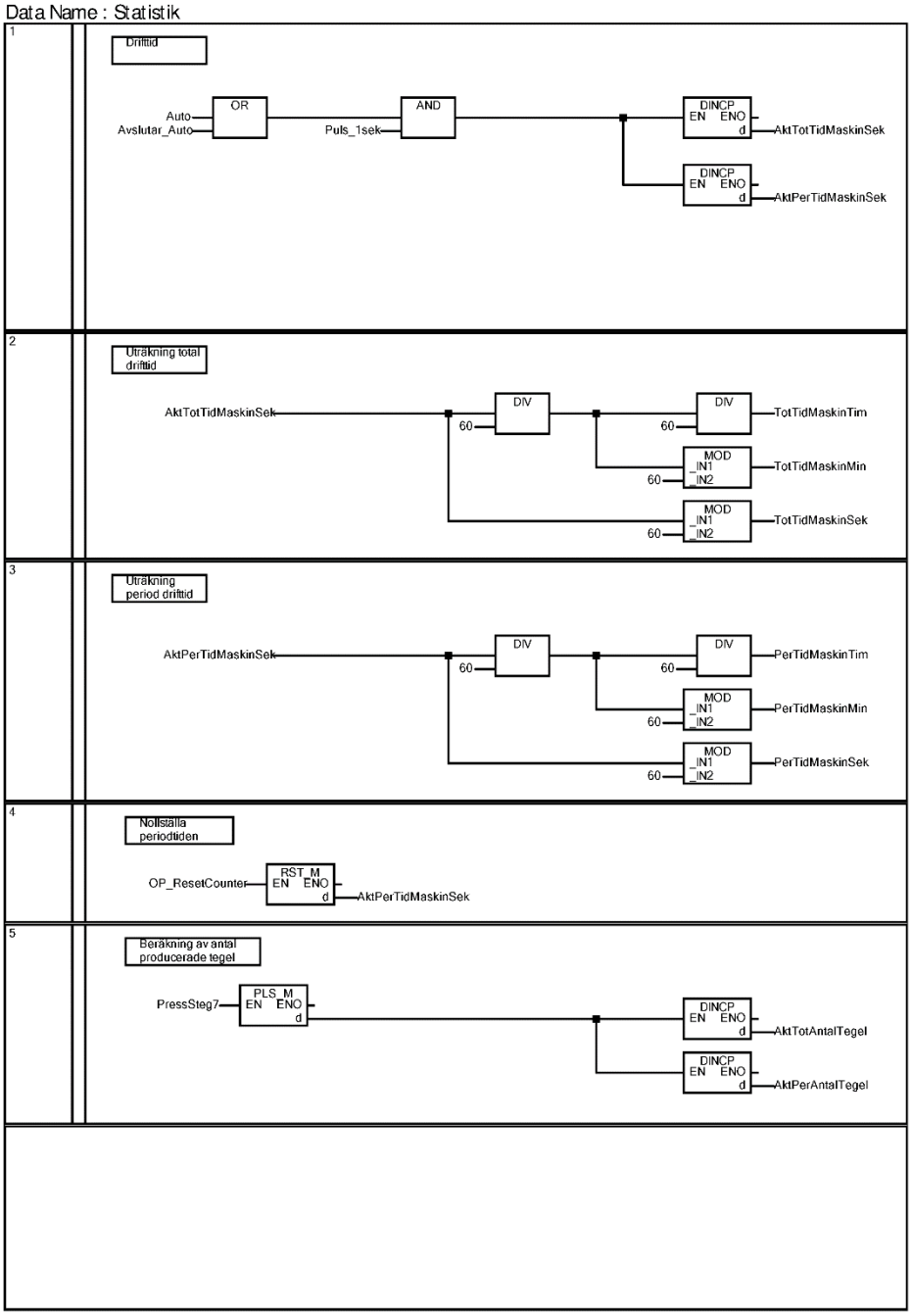
Data Name : Tvarbana_Sekvens



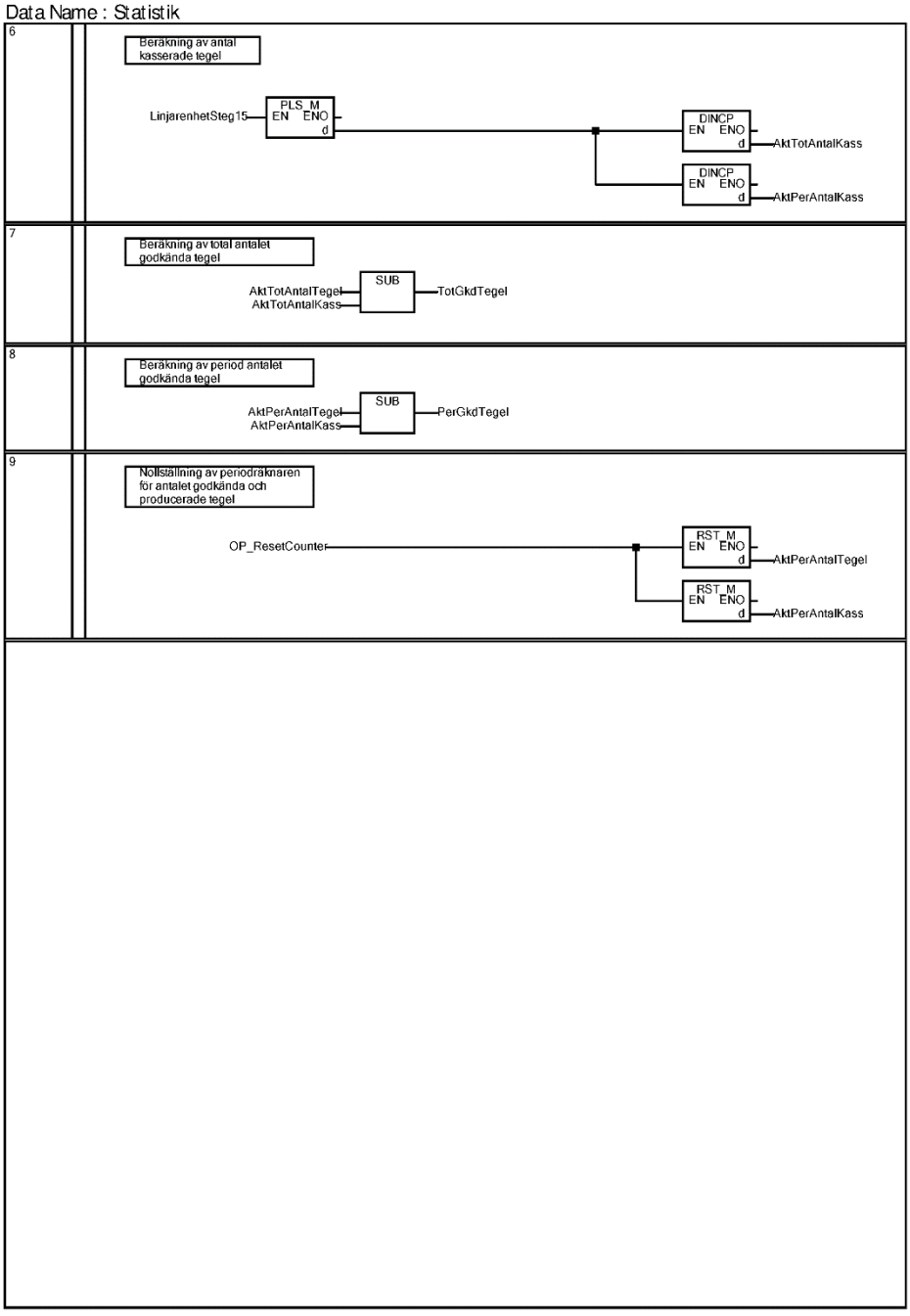
Data Name : Tvarbana_Sekvens



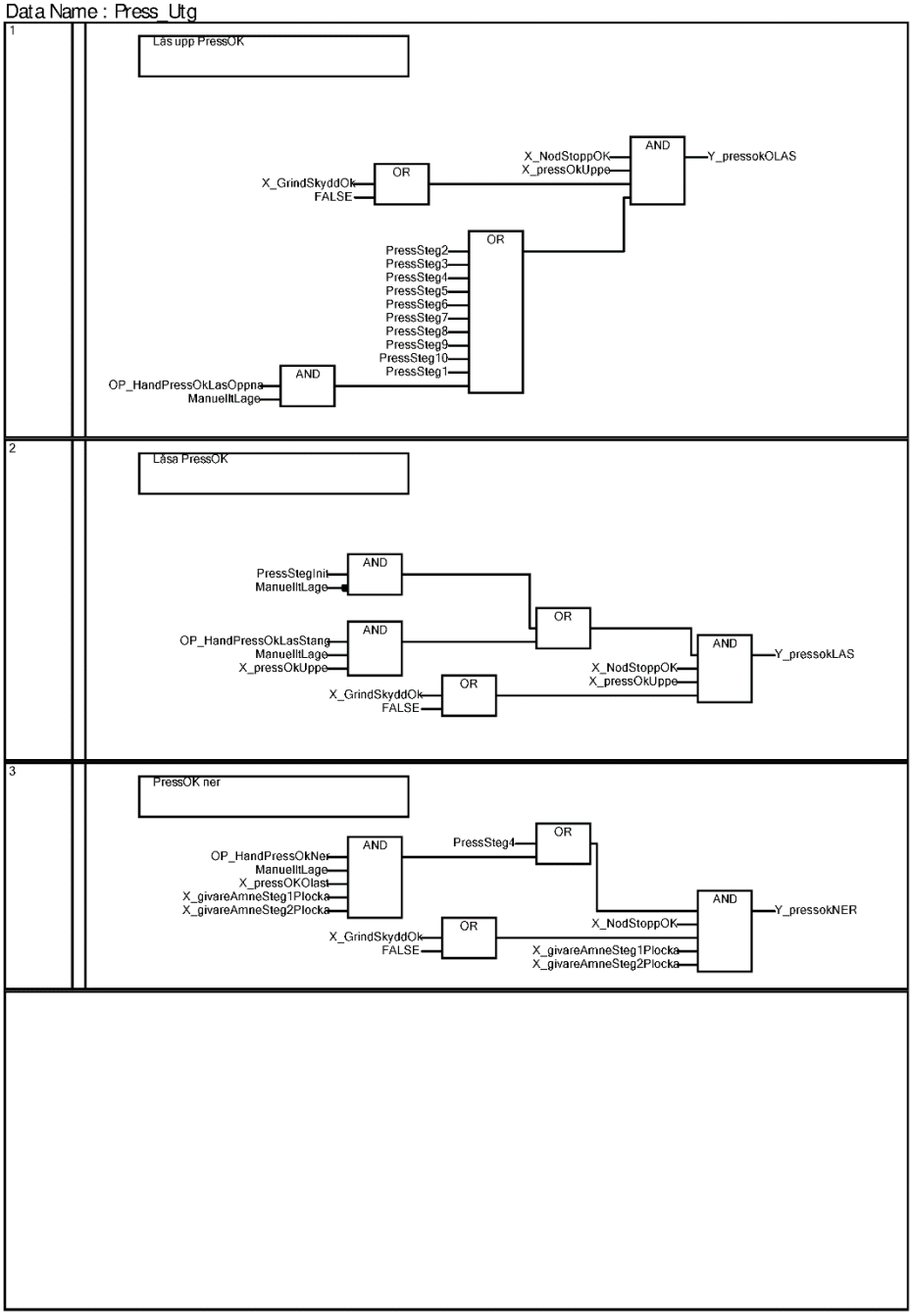
Data Name : Statistik



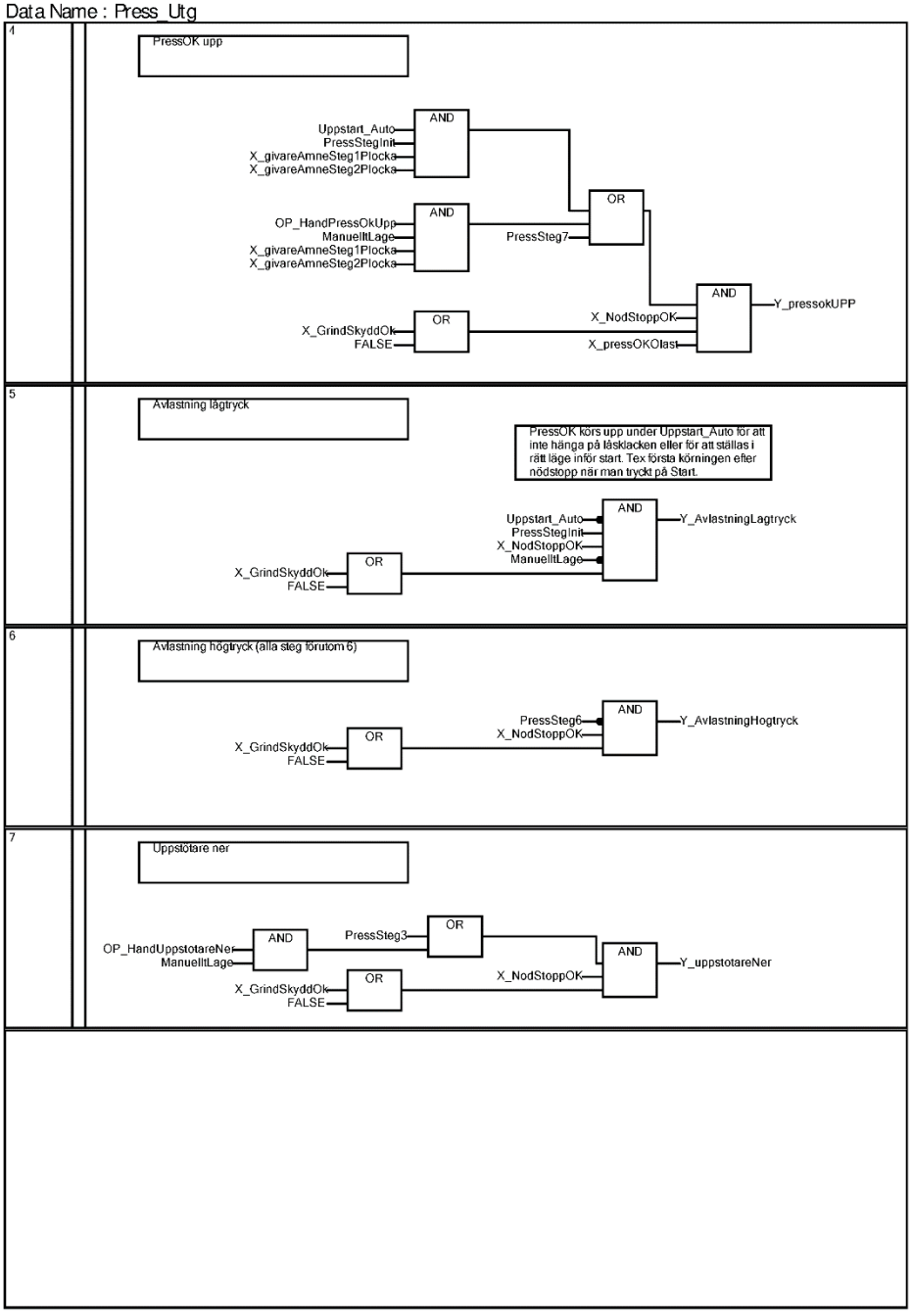
Data Name : Statistik



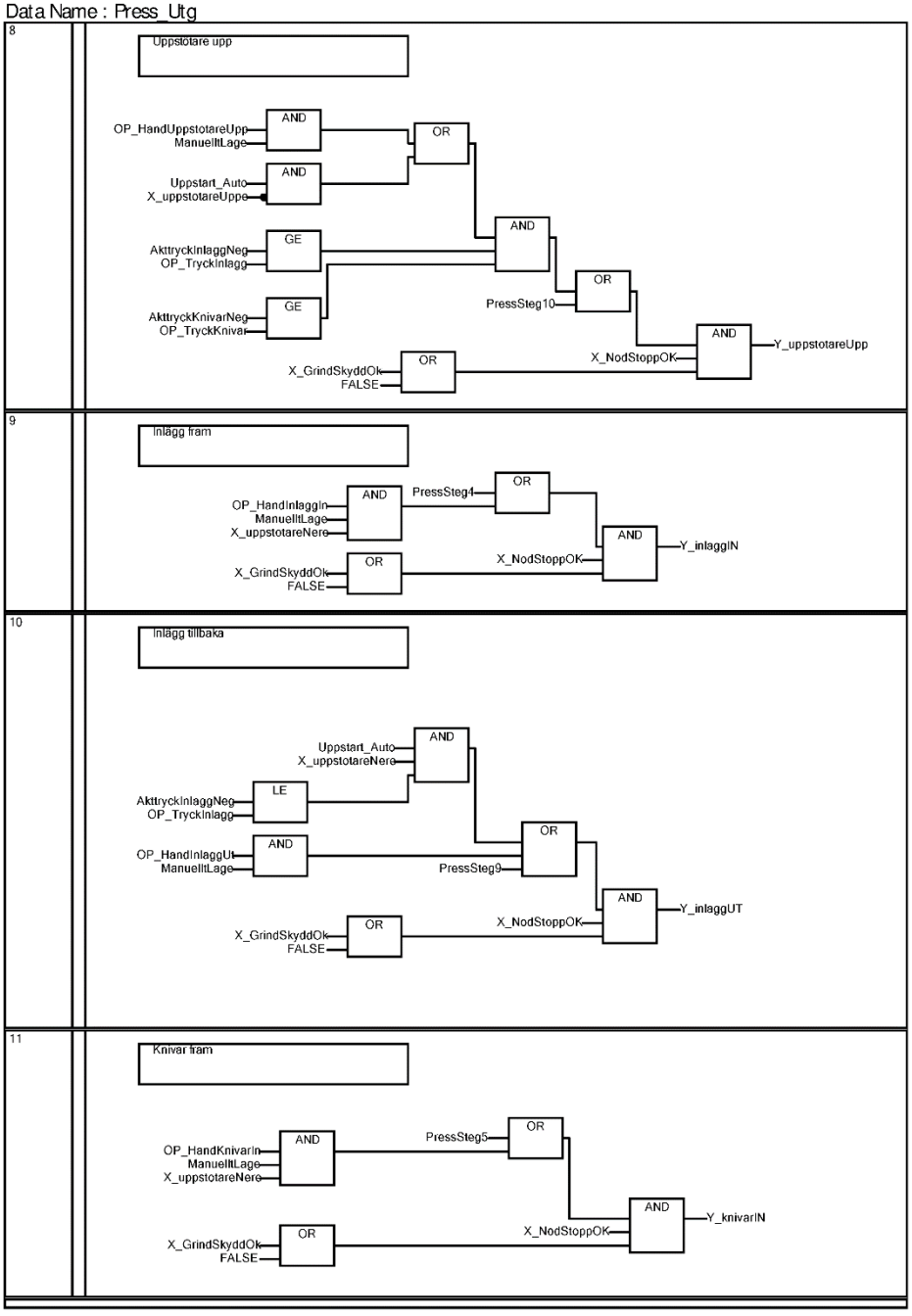
Data Name : Press_Utg



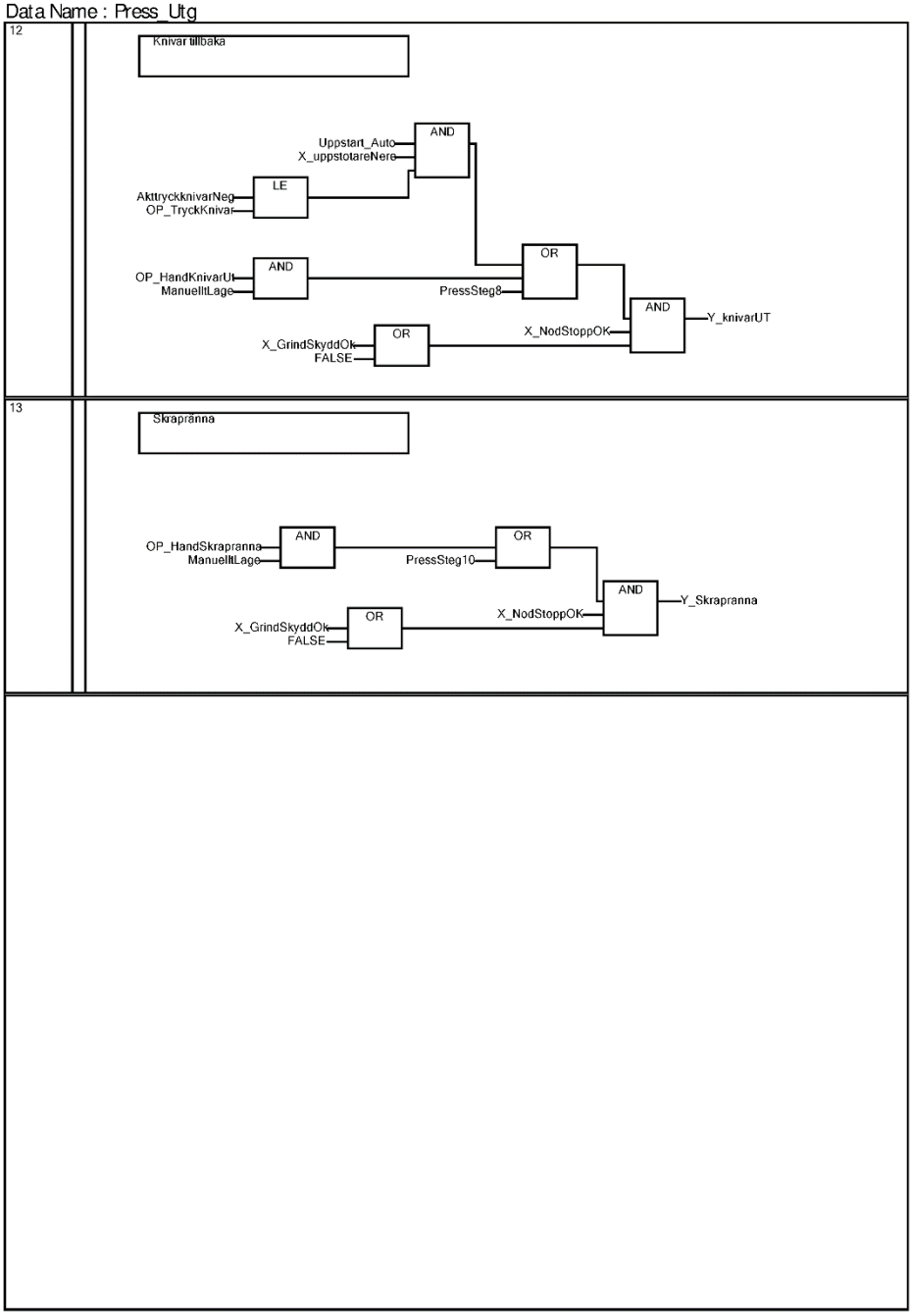
Data Name : Press_Utg



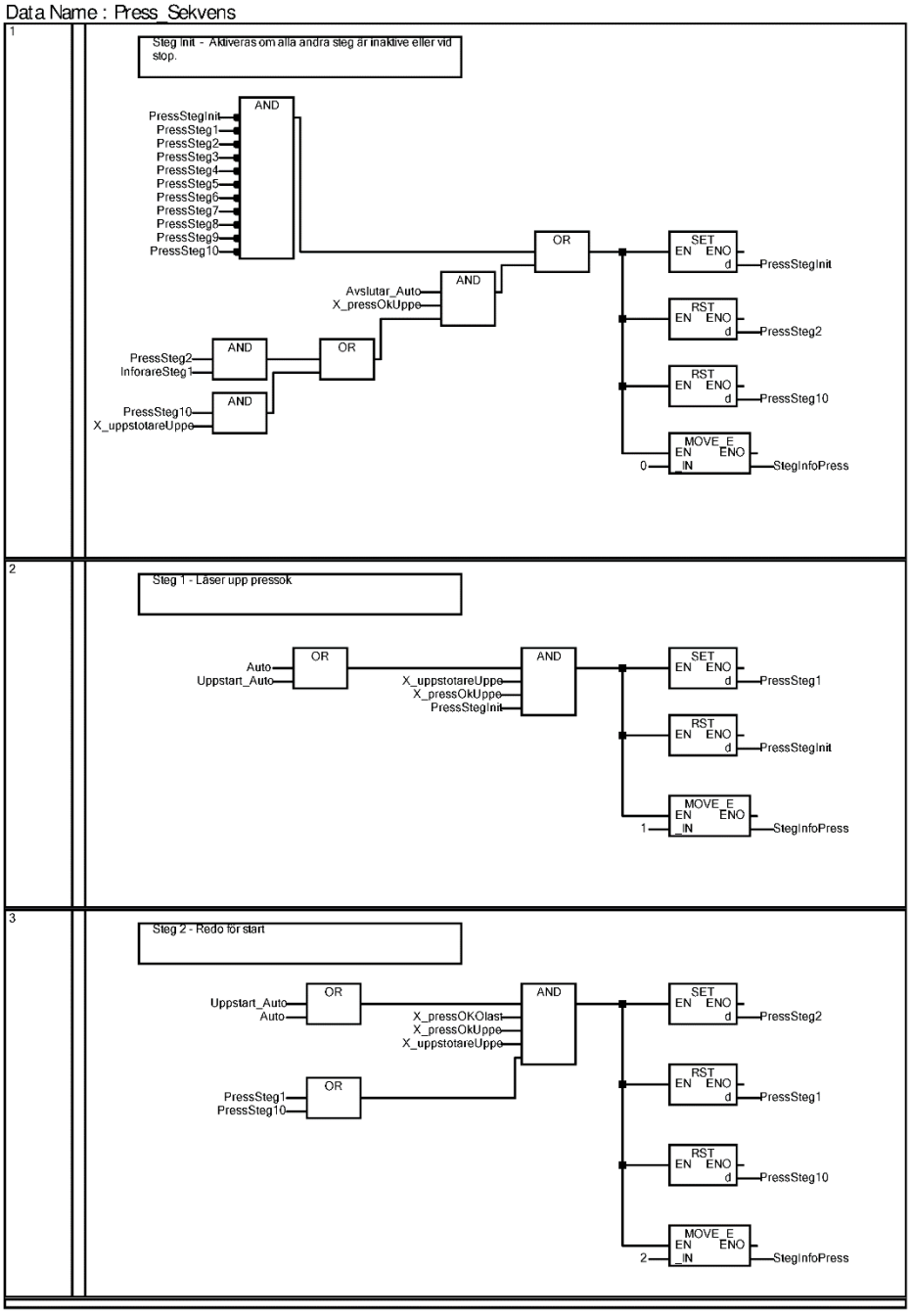
Data Name : Press_Utg



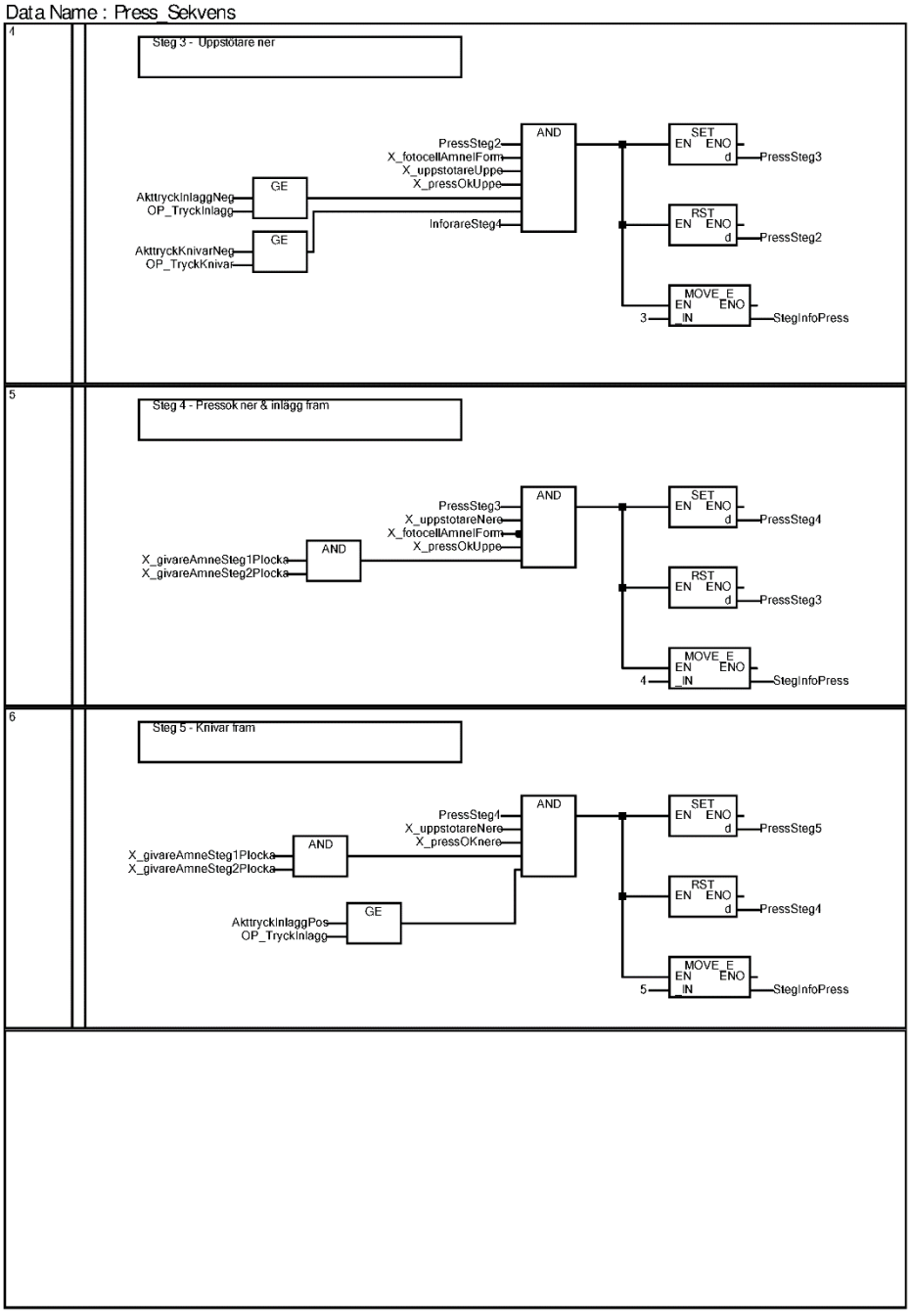
Data Name : Press_Utg



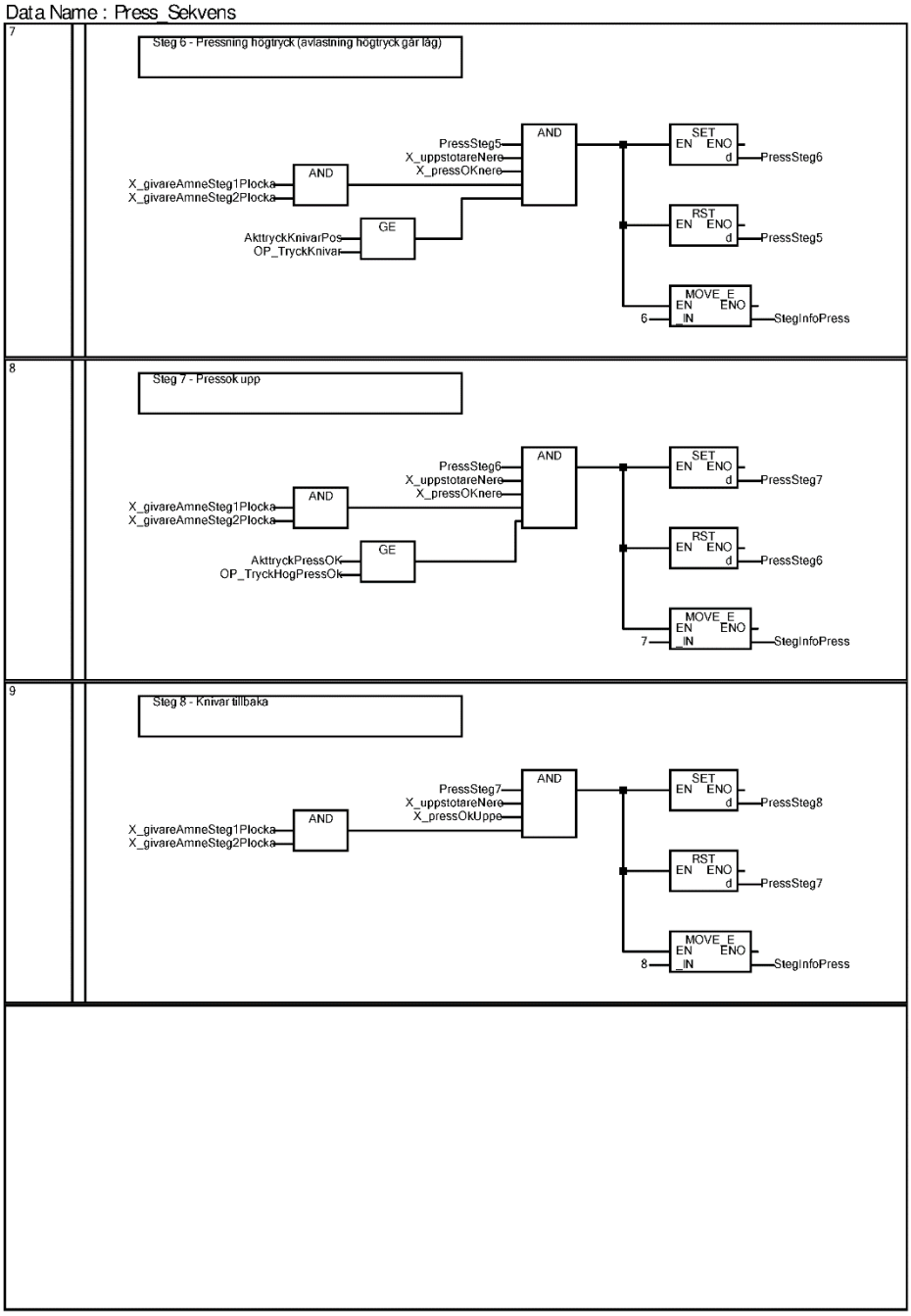
Data Name : Press_Sekvens



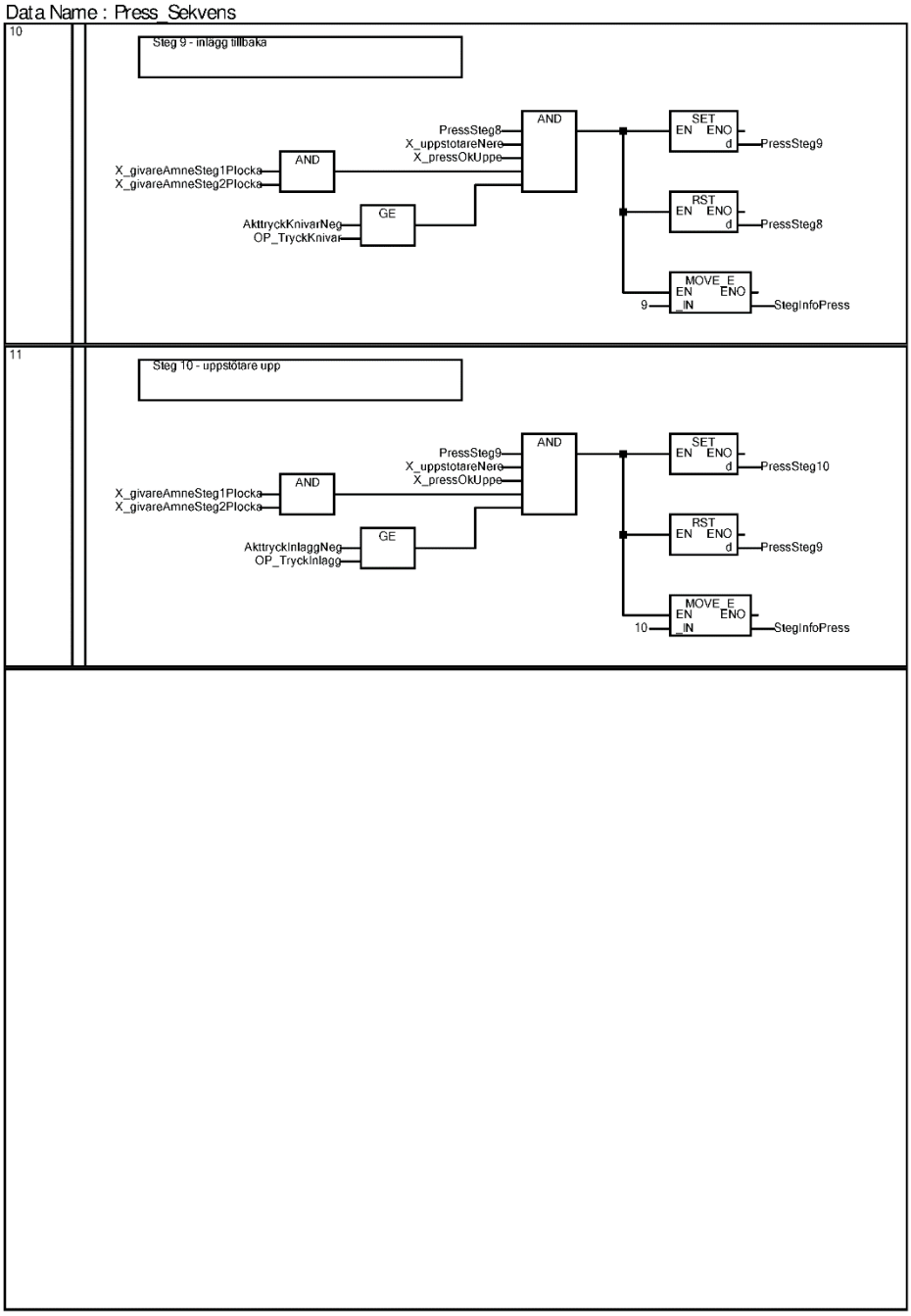
Data Name : Press_Sekvens



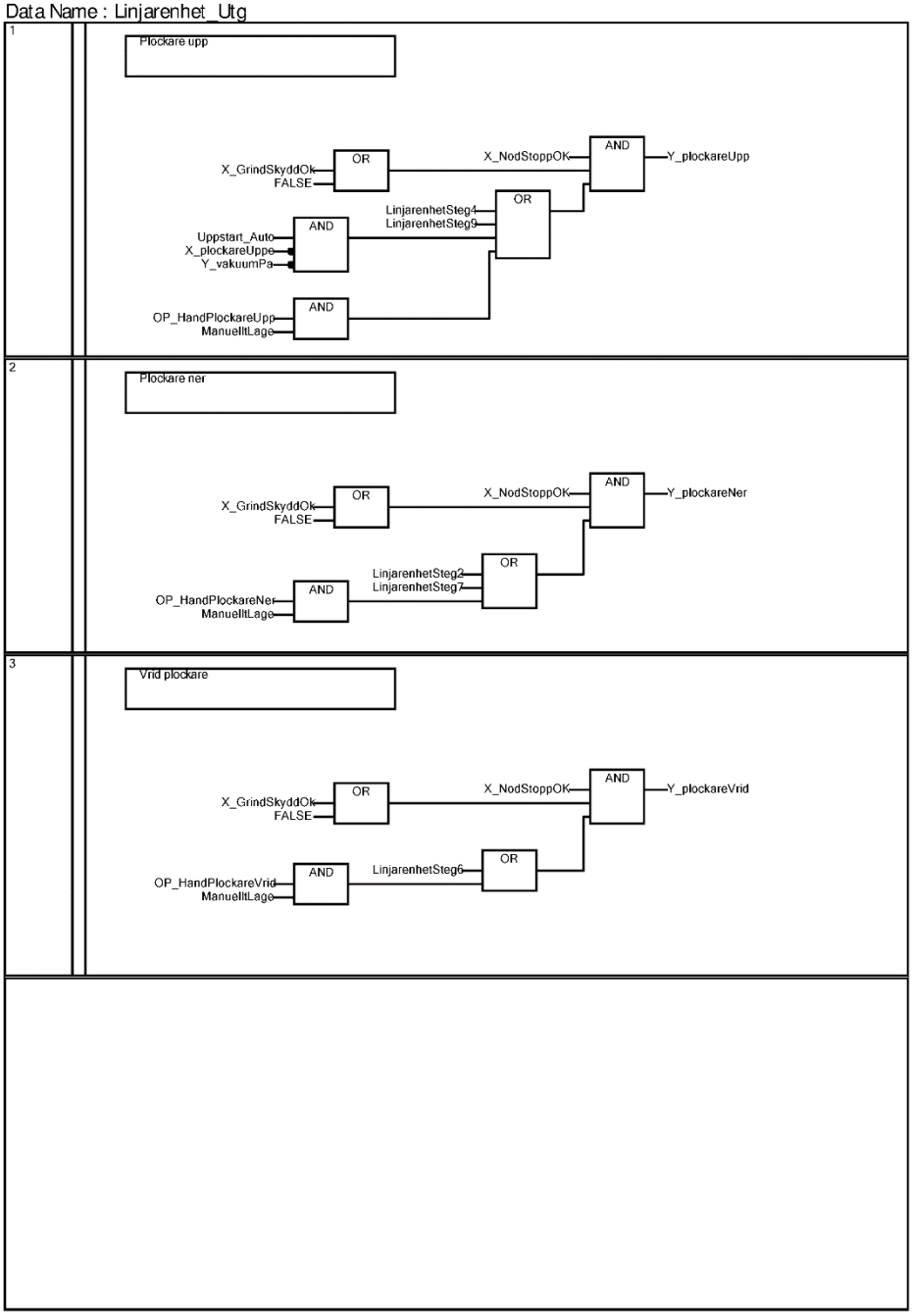
Data Name : Press_Sekvens



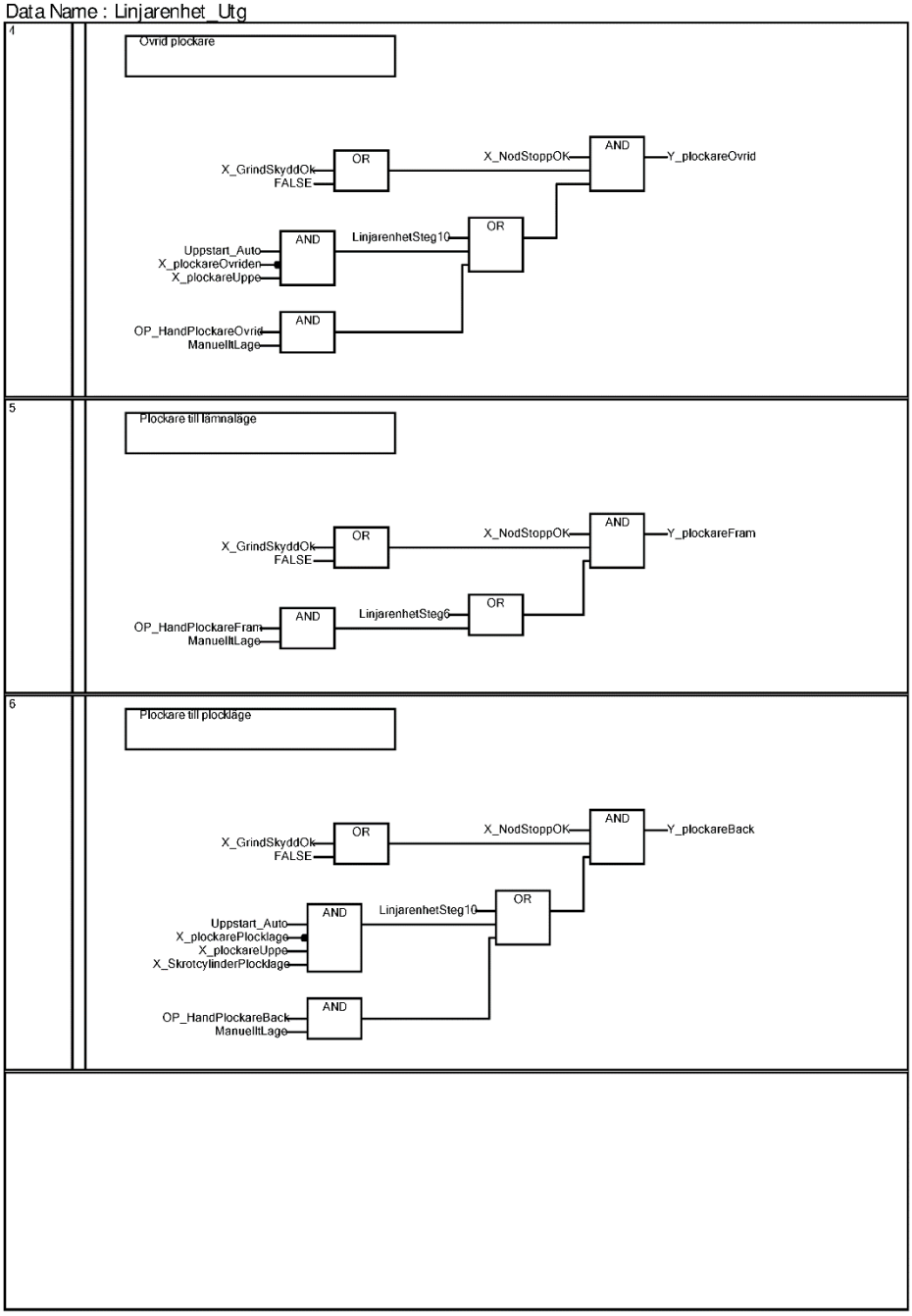
Data Name : Press_Sekvens



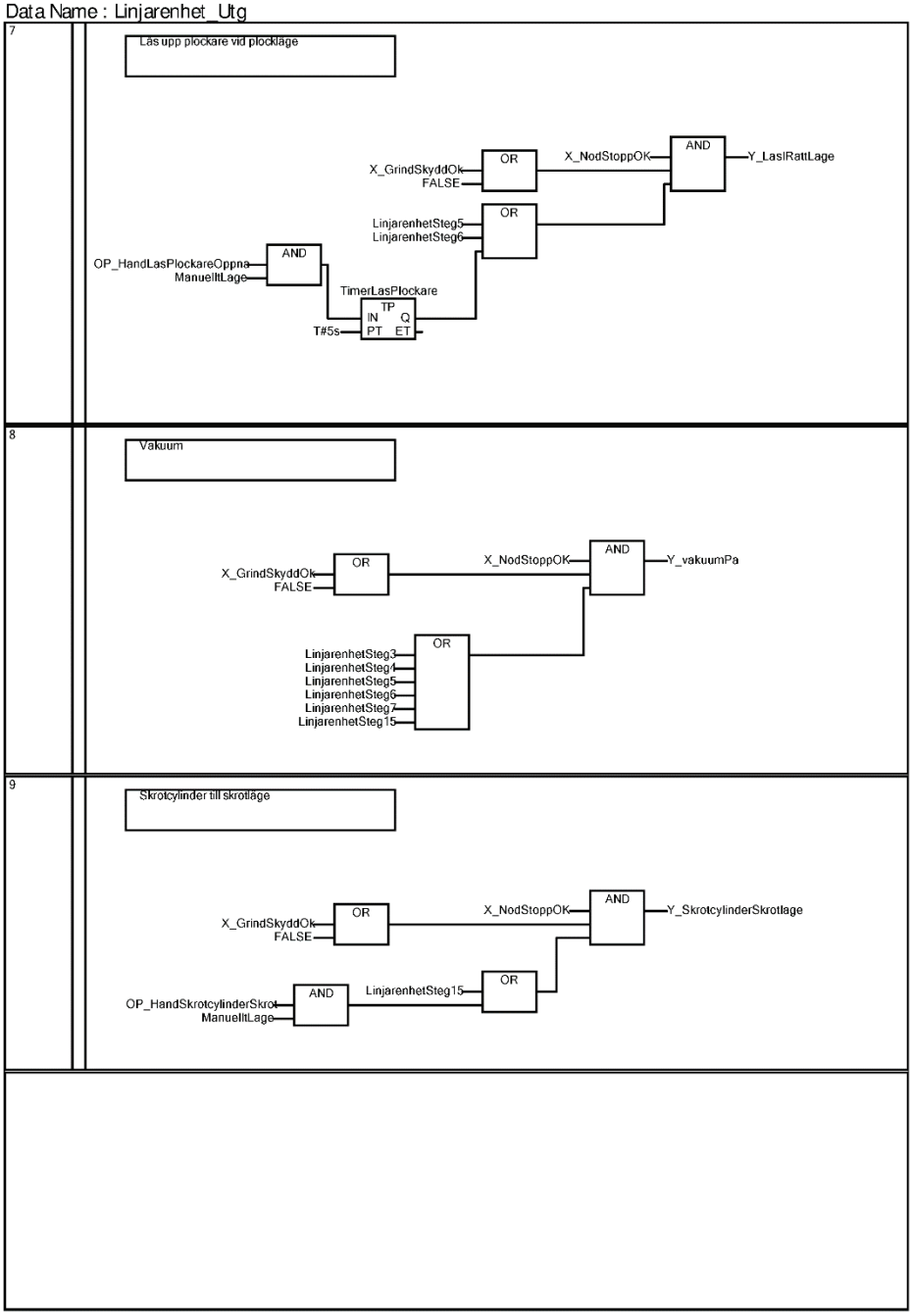
Data Name : Linjarenhet_Utg



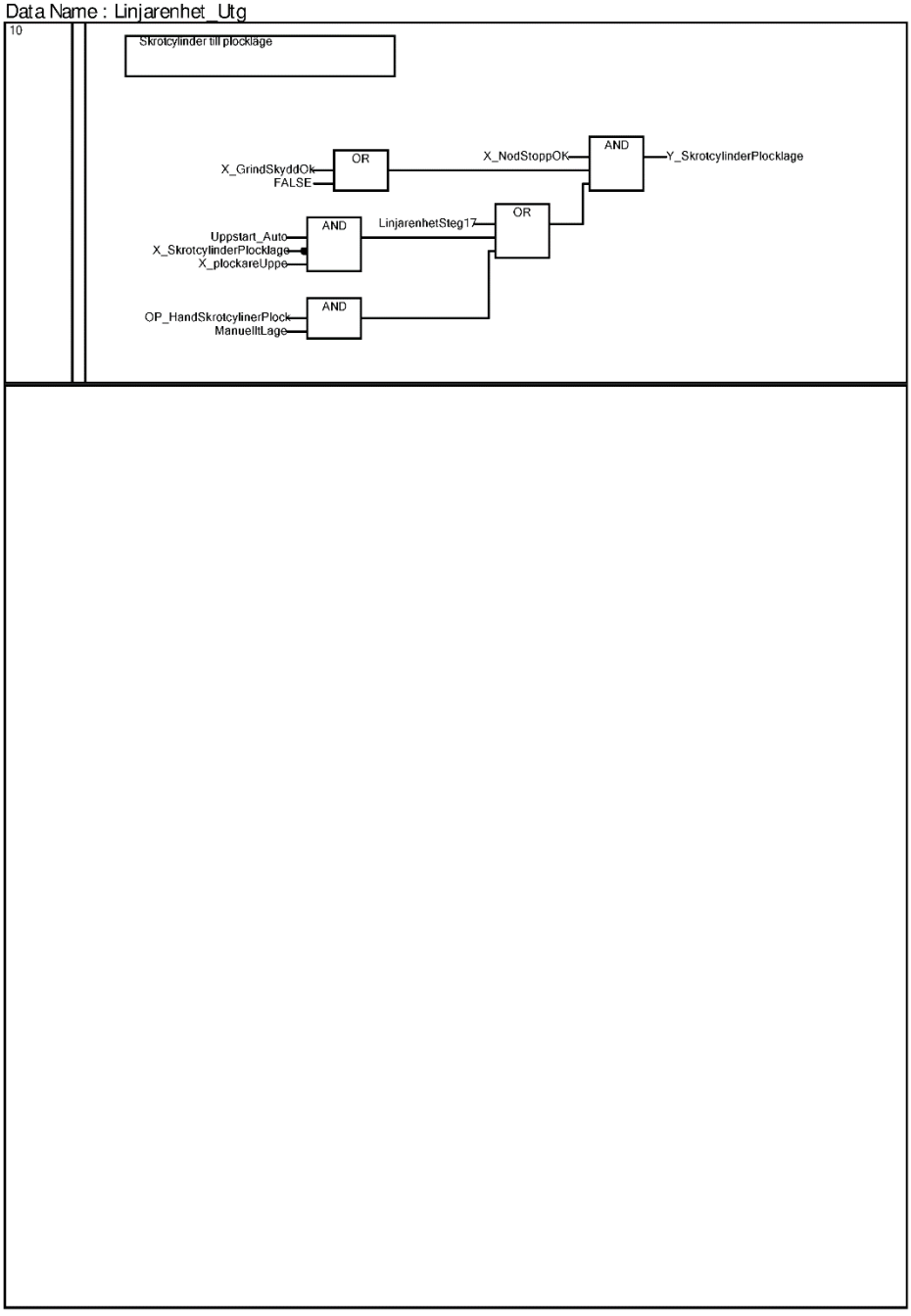
Data Name : Linjarenhet_Utg



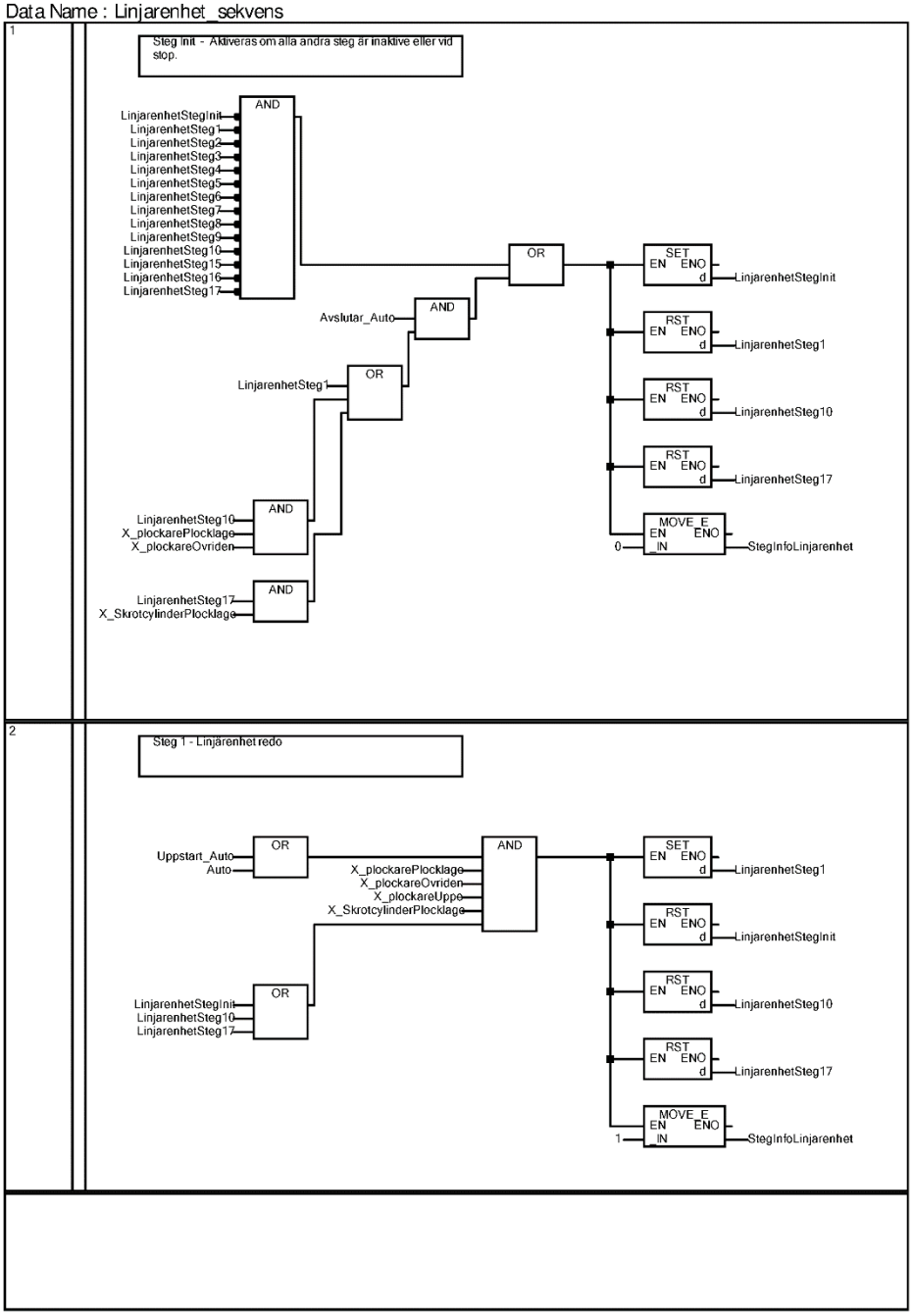
Data Name : Linjarenhet_Utg



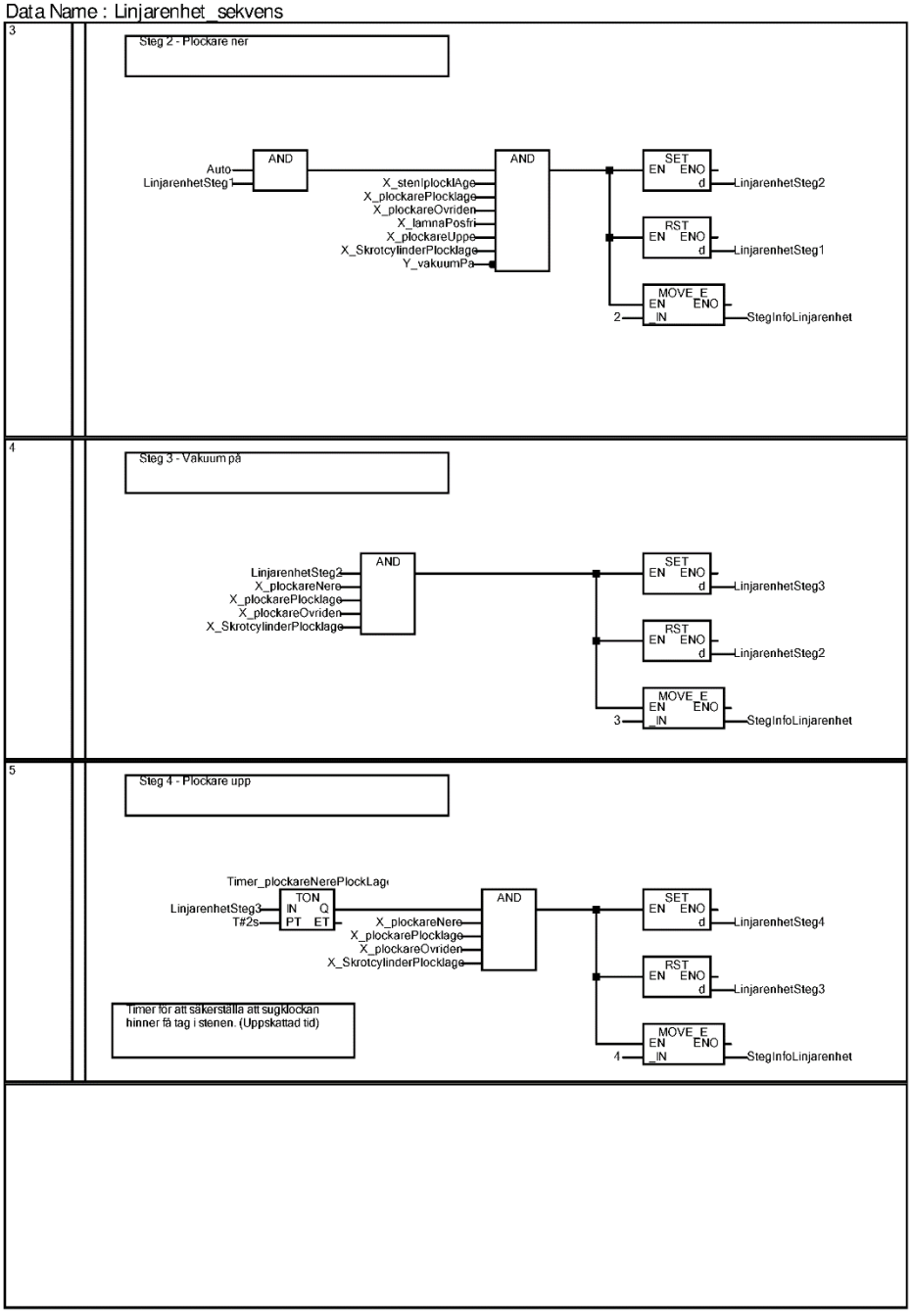
Data Name : Linjarenhet_Utg



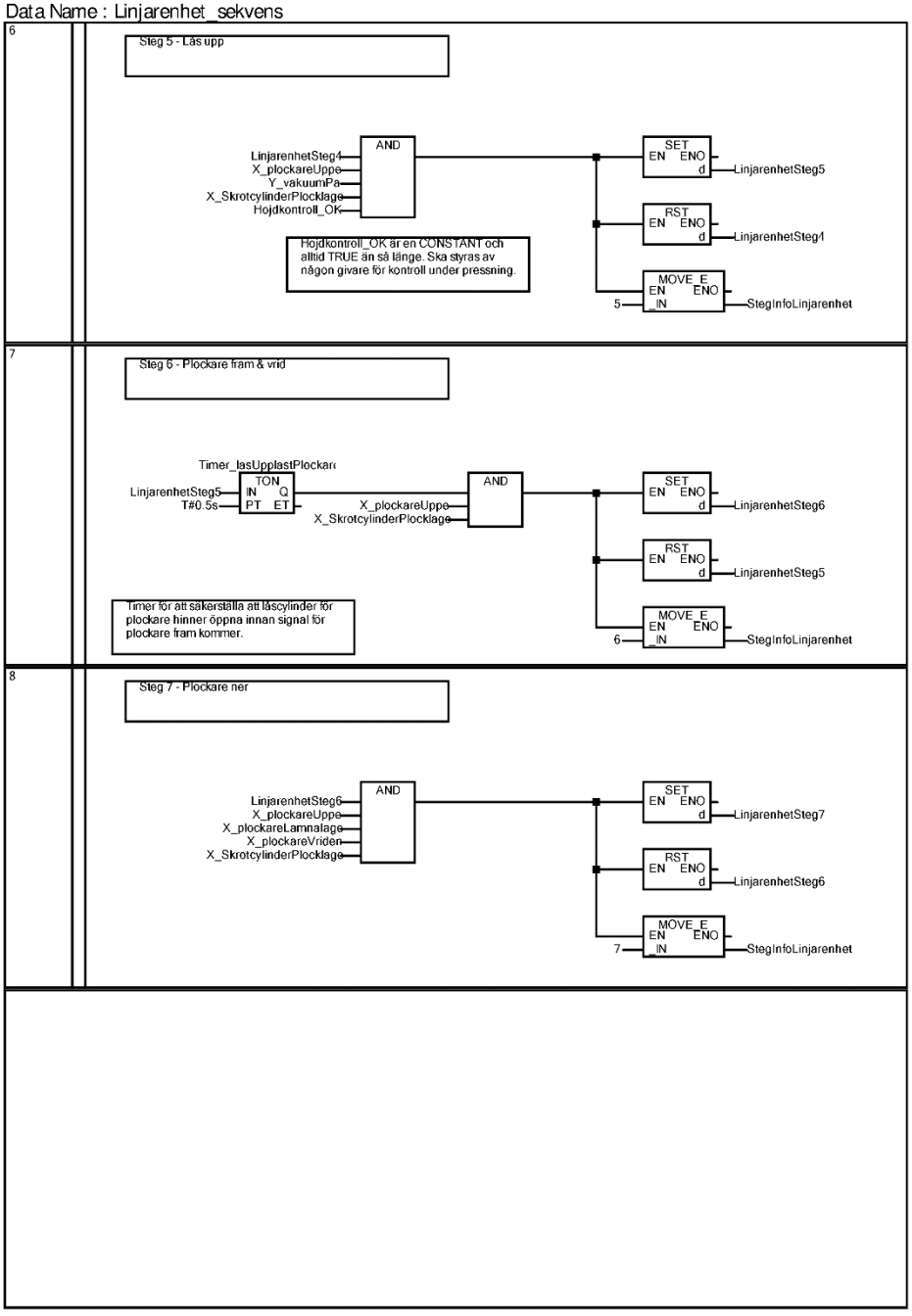
Data Name : Linjarenhet_sekvens



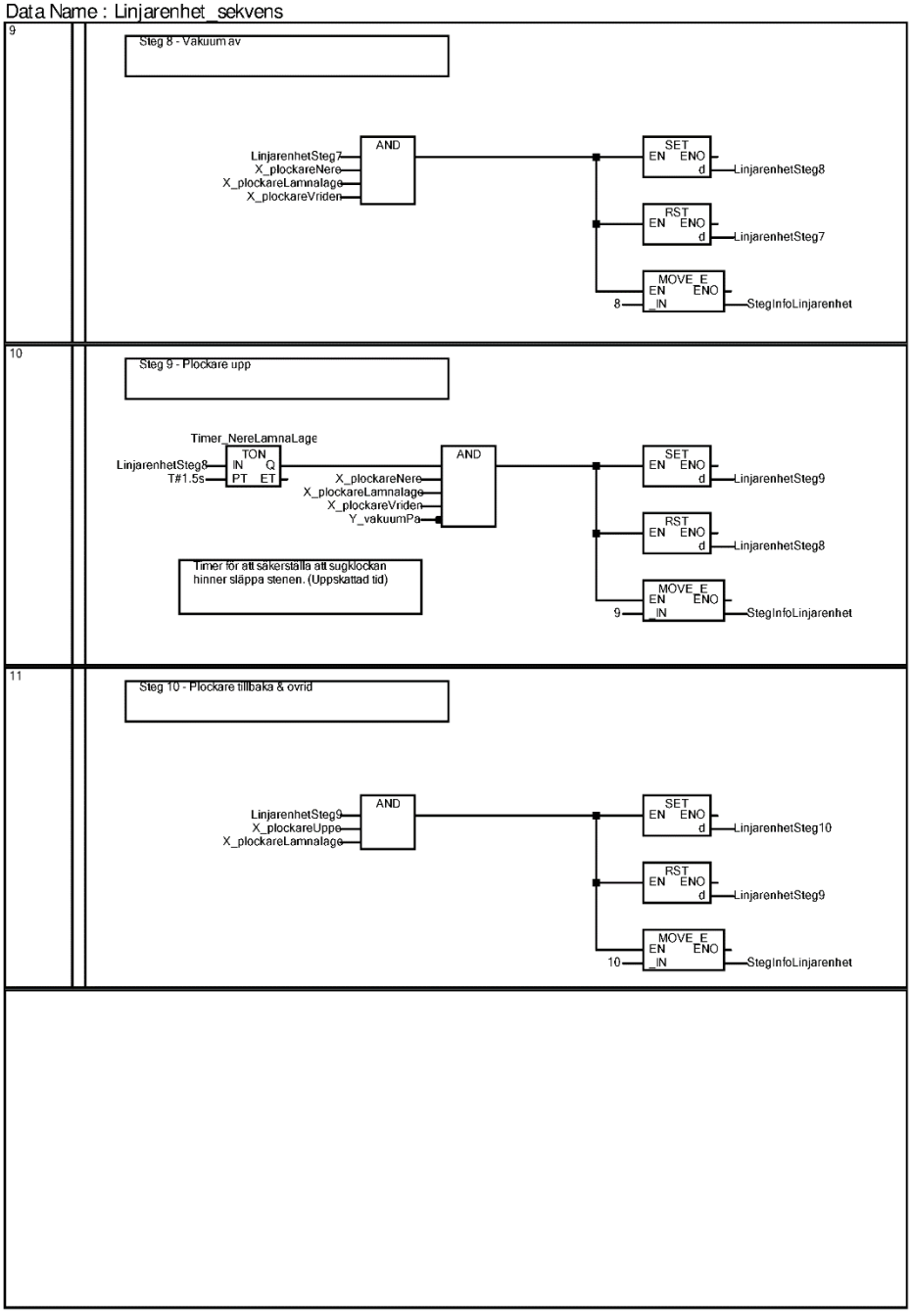
Data Name : Linjarenhet_sekvens



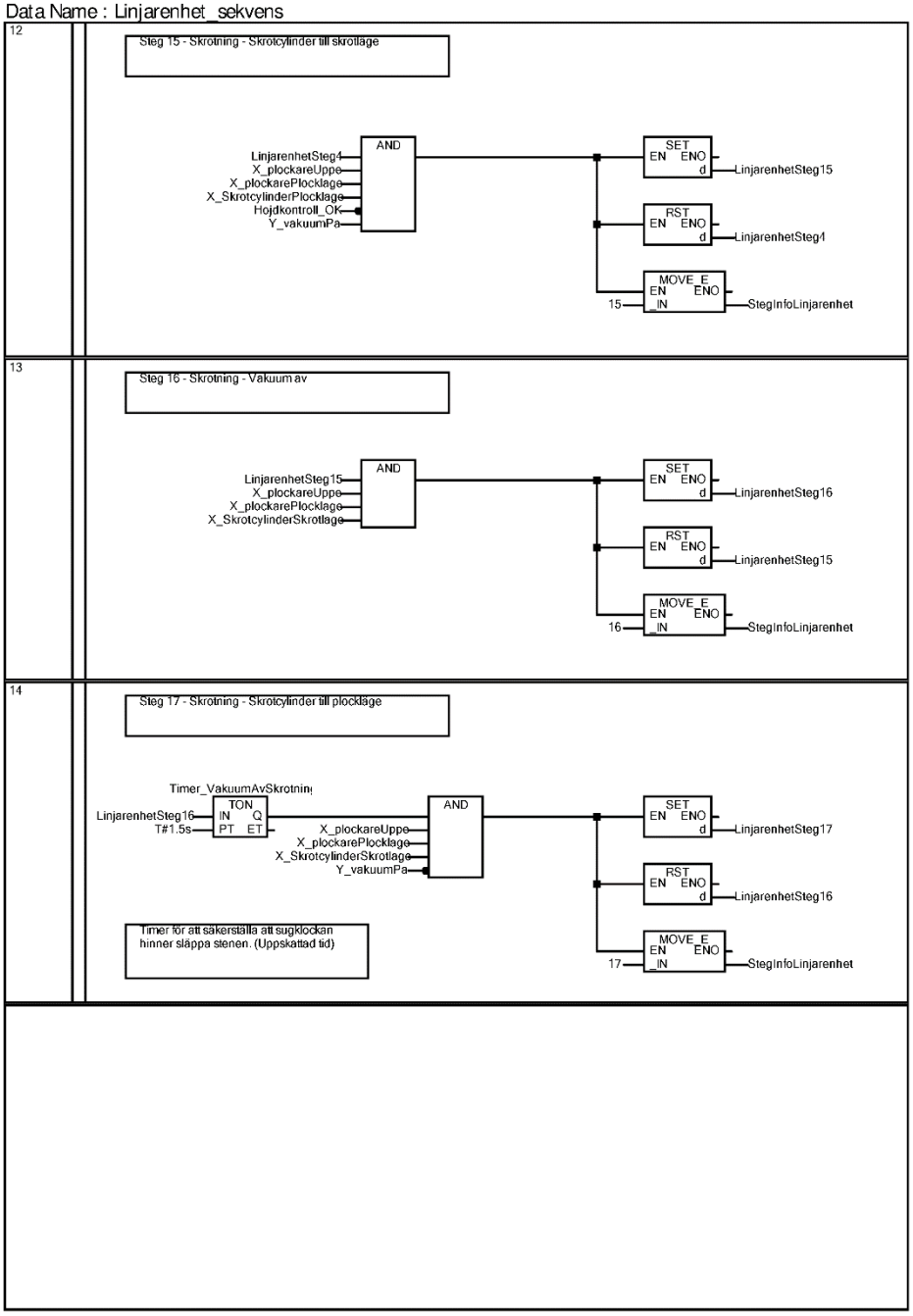
Data Name : Linjarenhet_sekvens



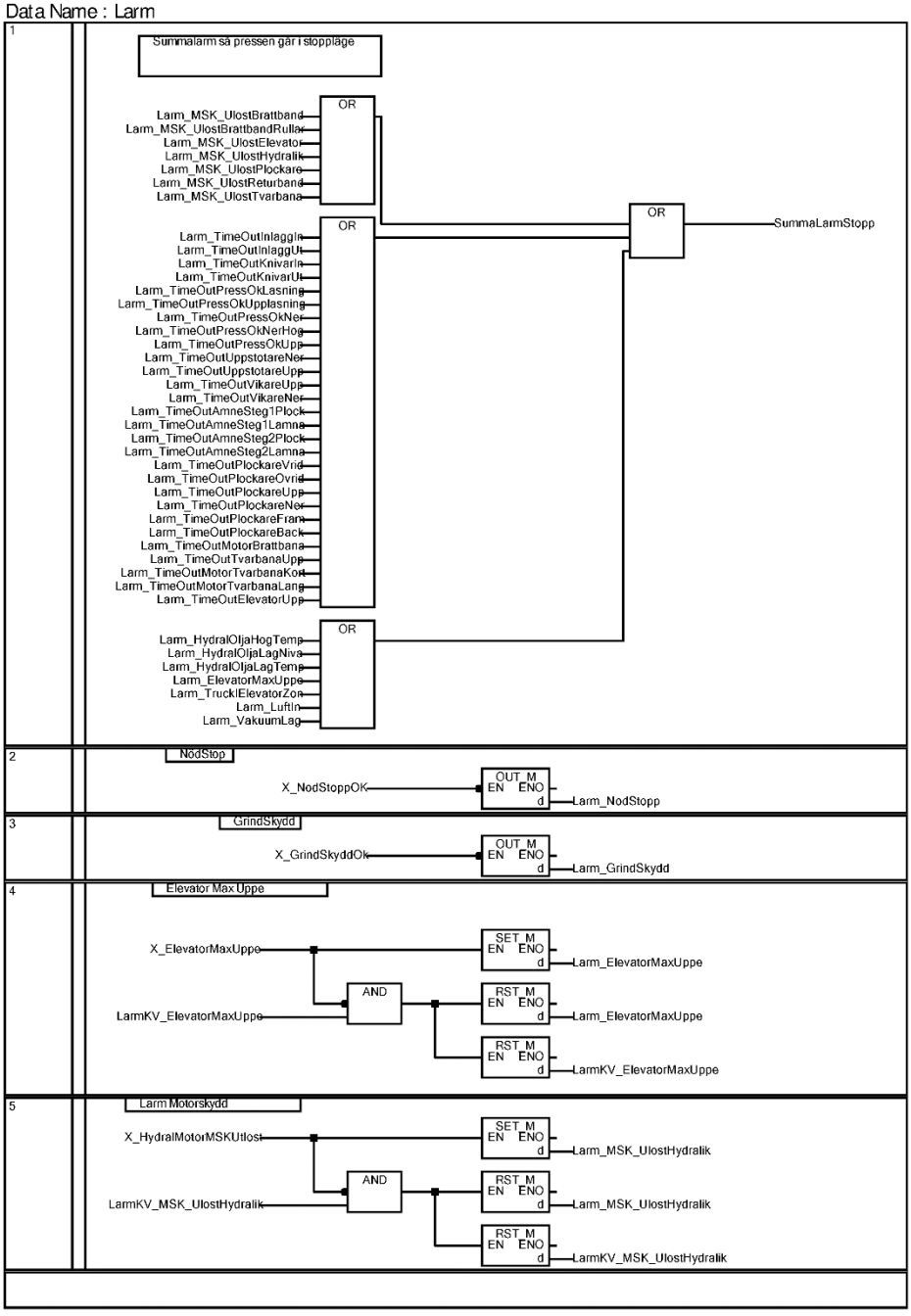
Data Name : Linjarenhet_sekvens



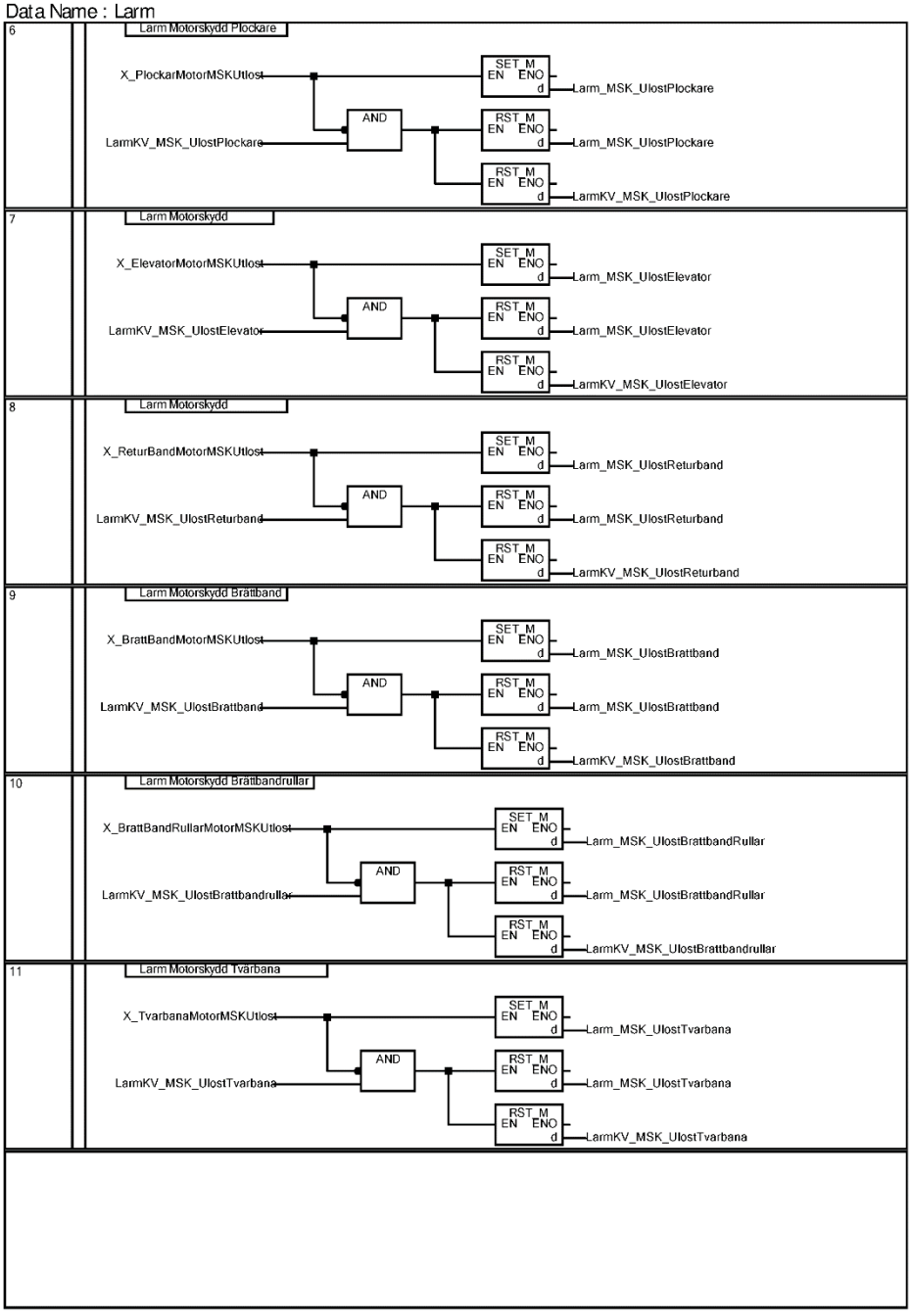
Data Name : Linjarenhet_sekvens



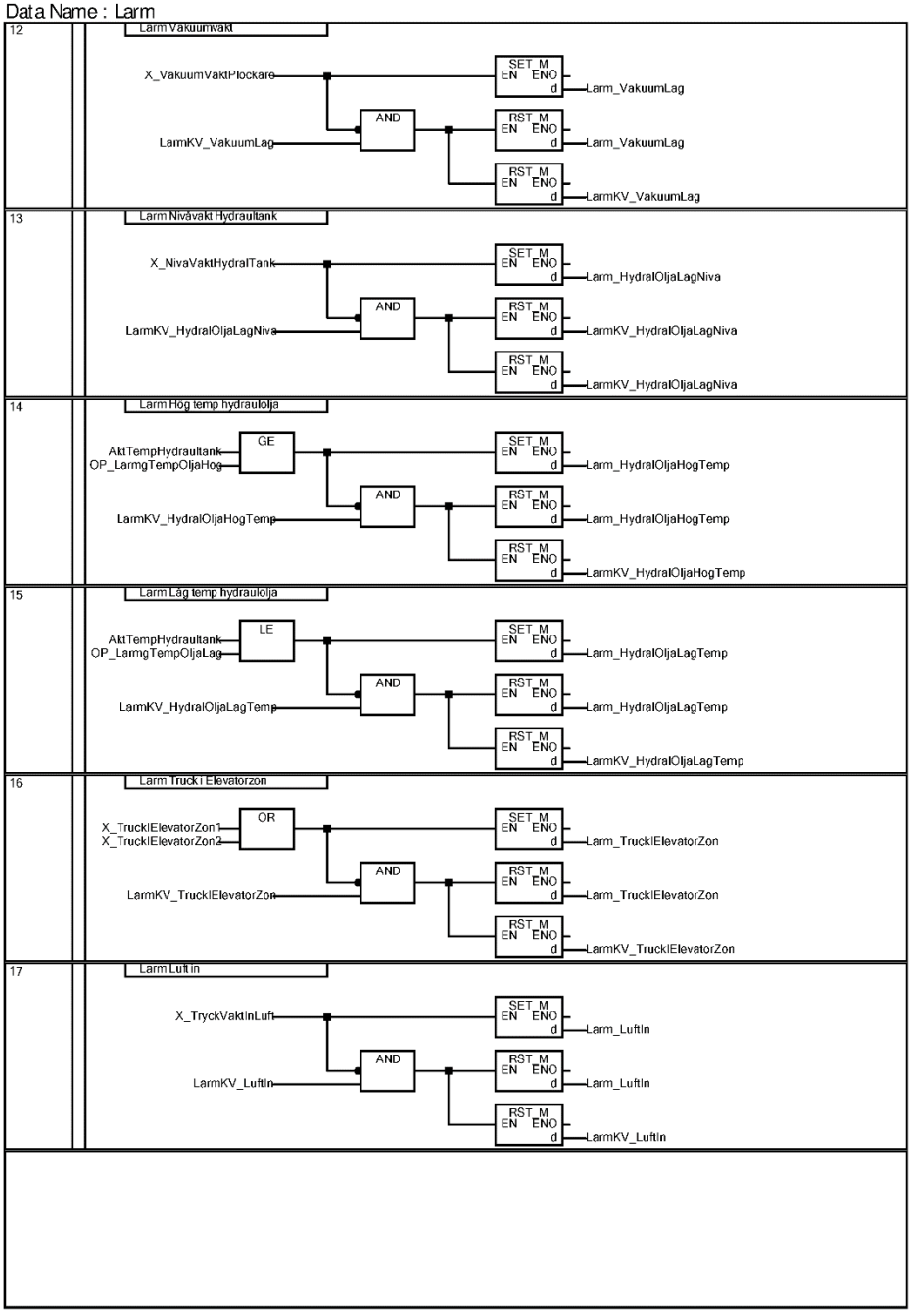
Data Name : Larm



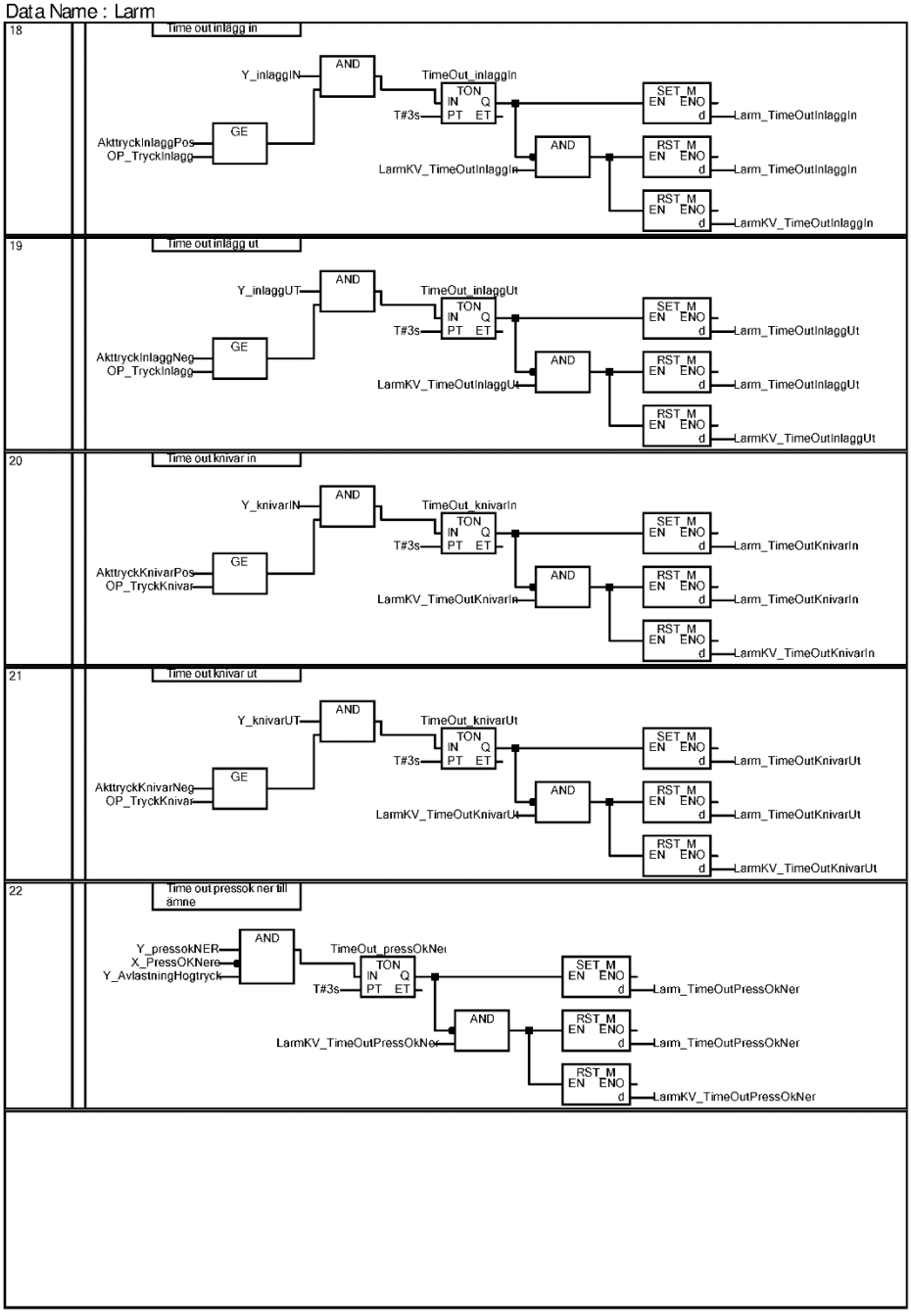
Data Name : Larm



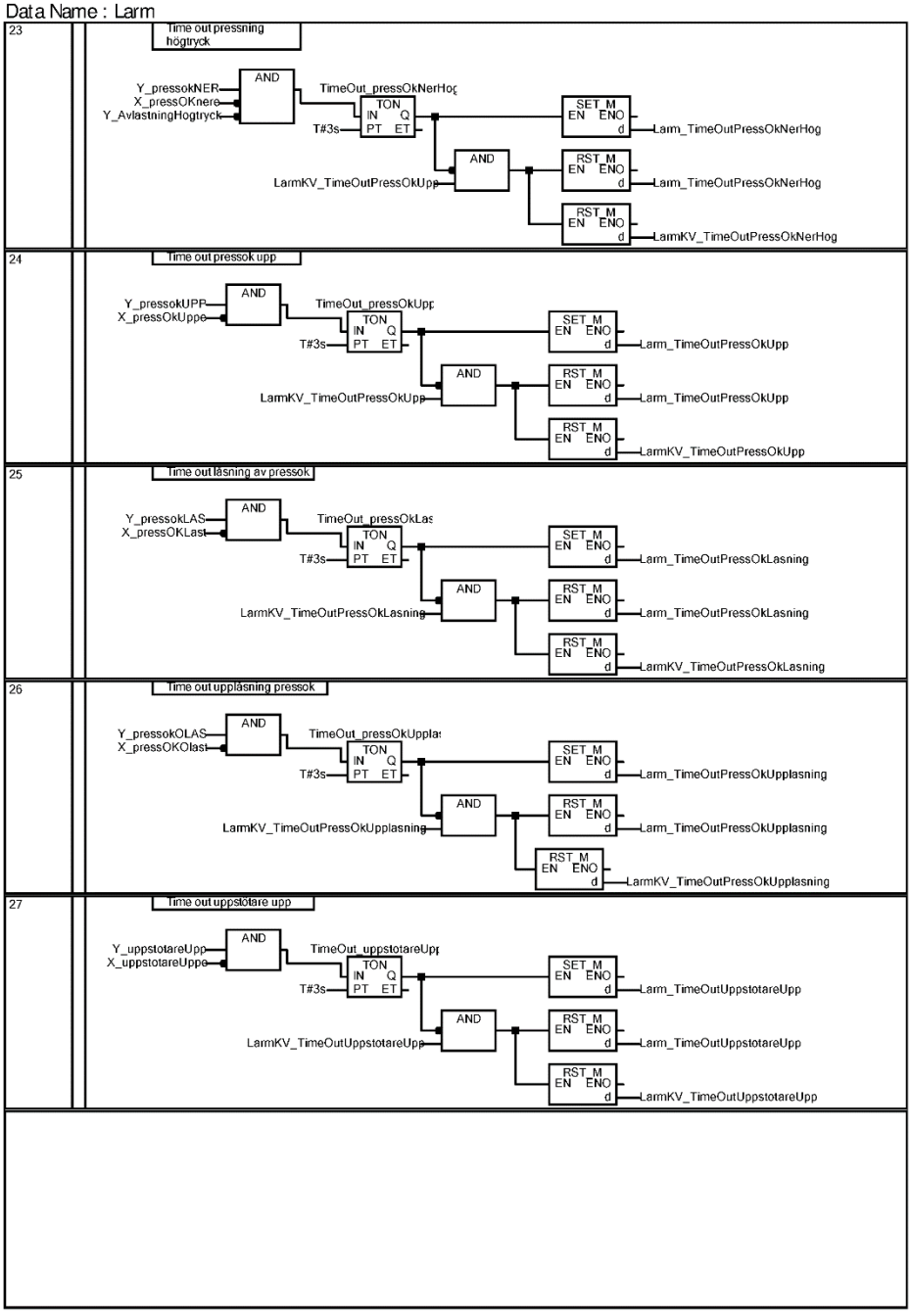
Data Name : Larm



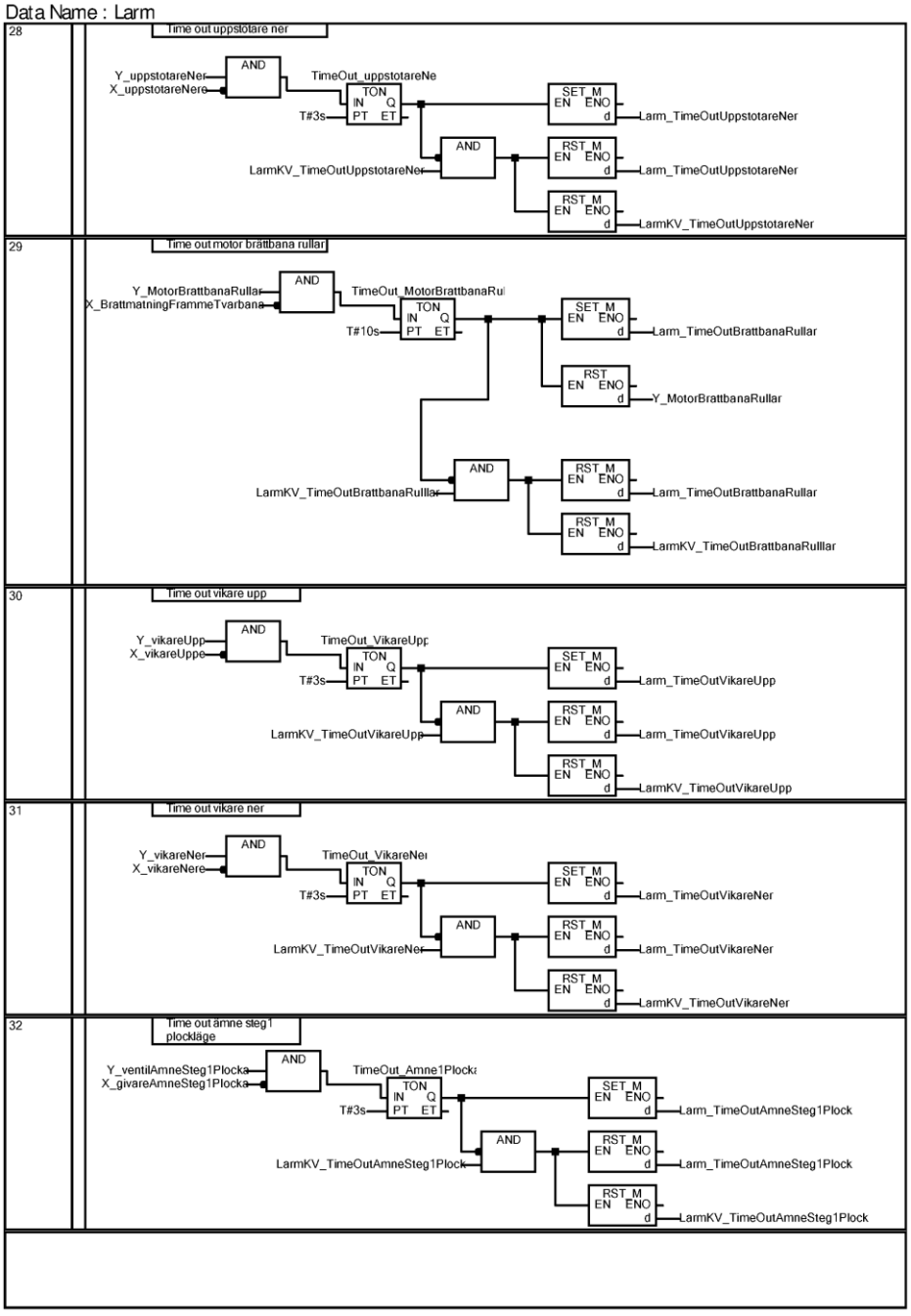
Data Name : Larm



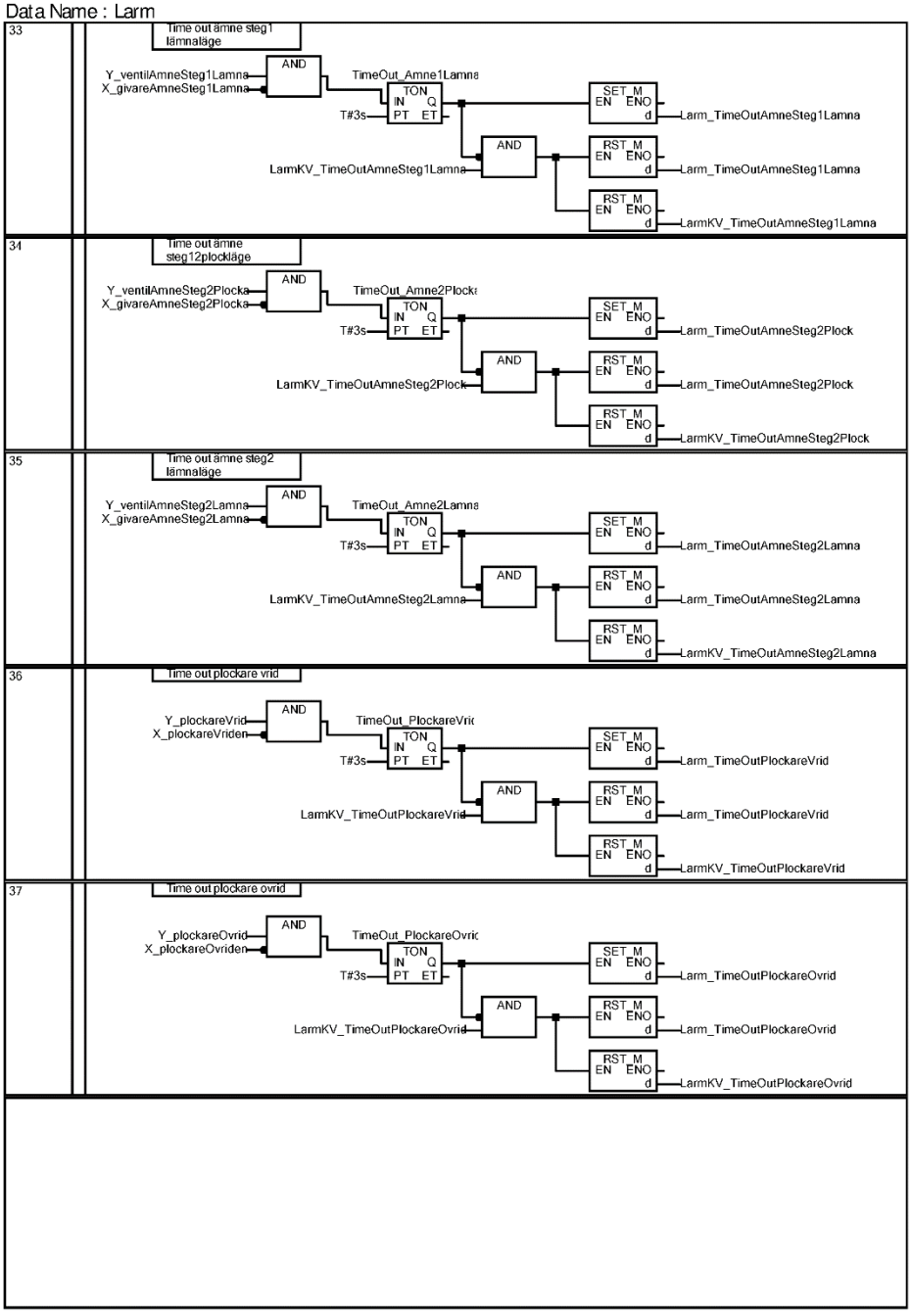
Data Name : Larm



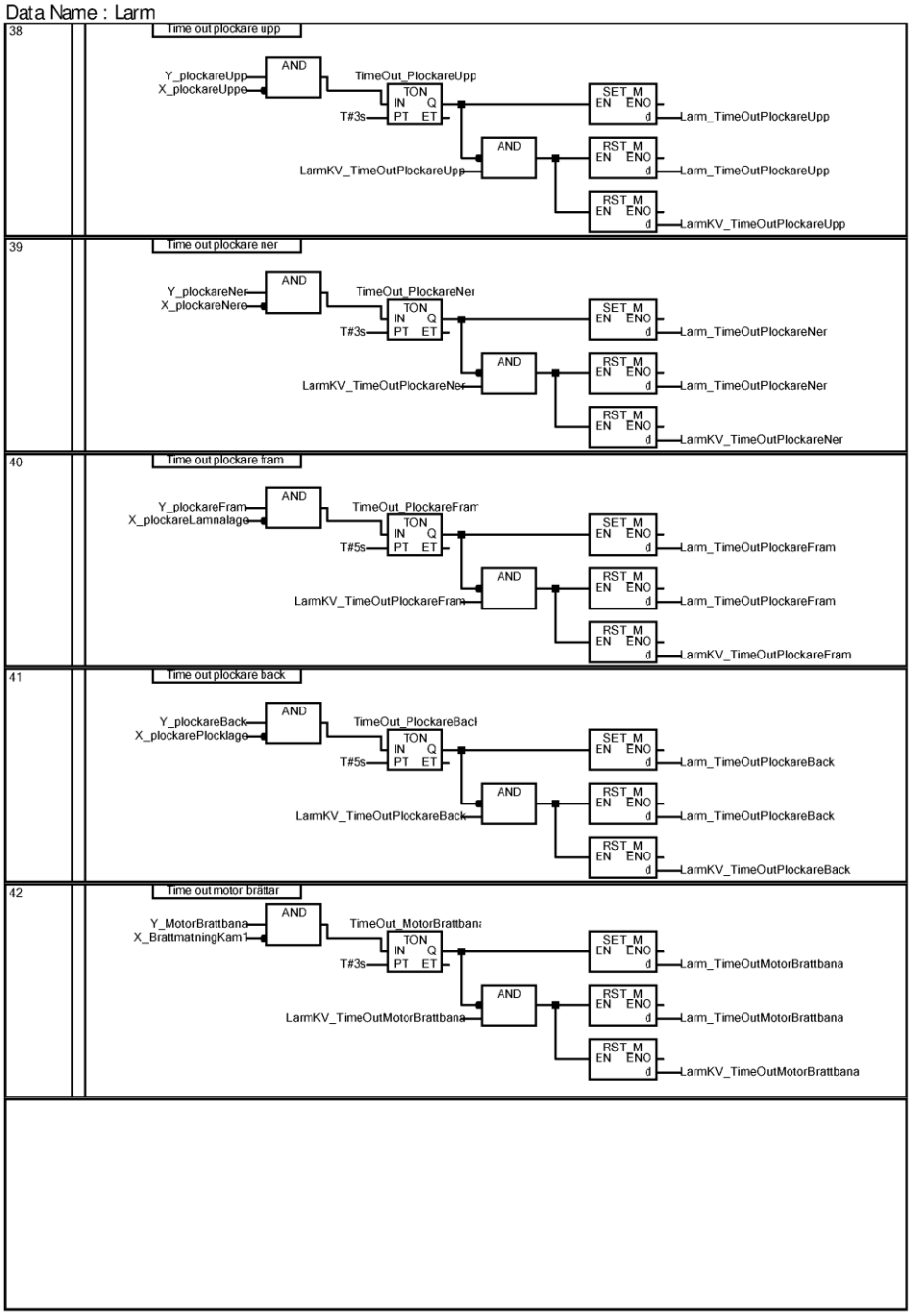
Data Name : Larm



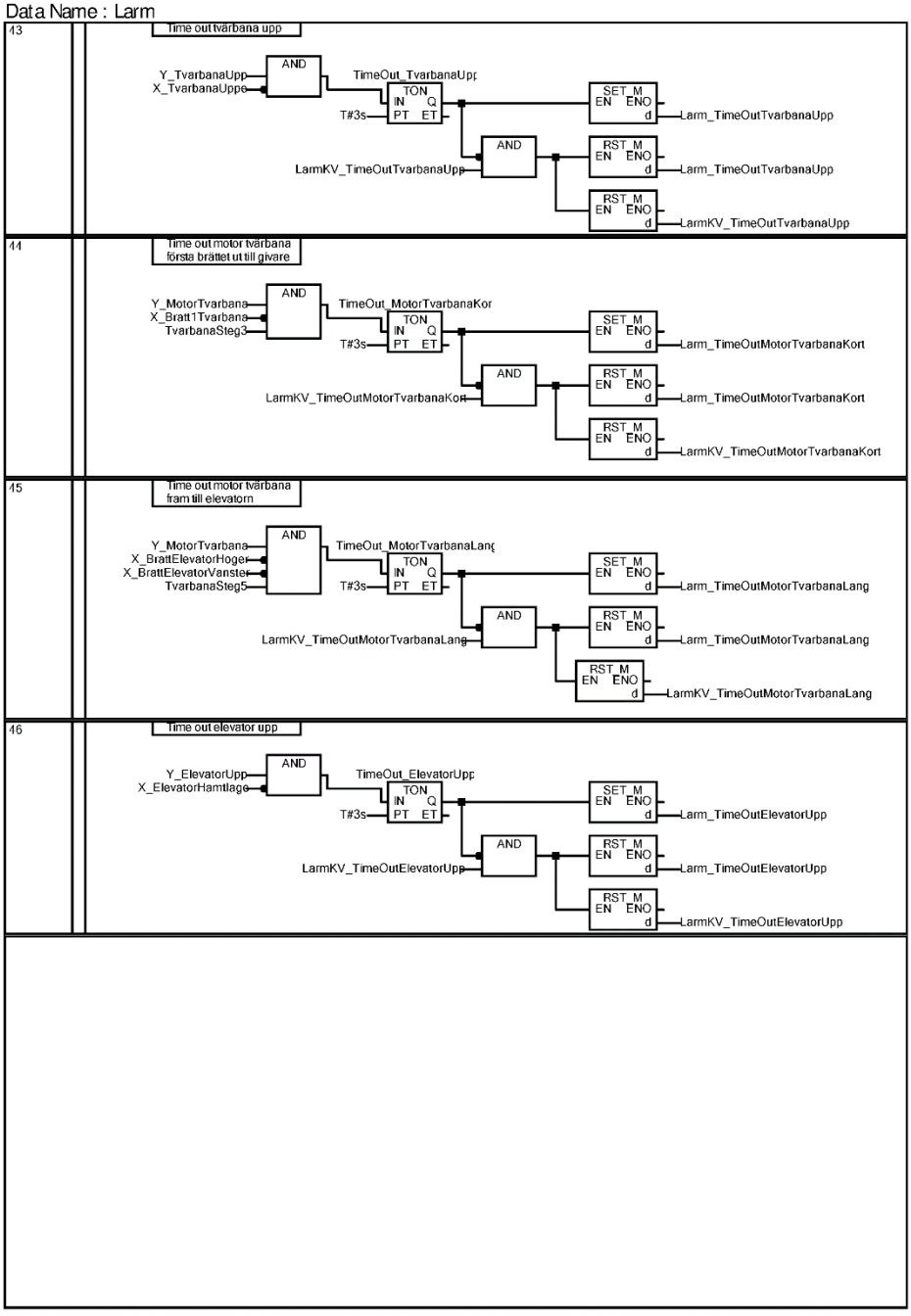
Data Name : Larm



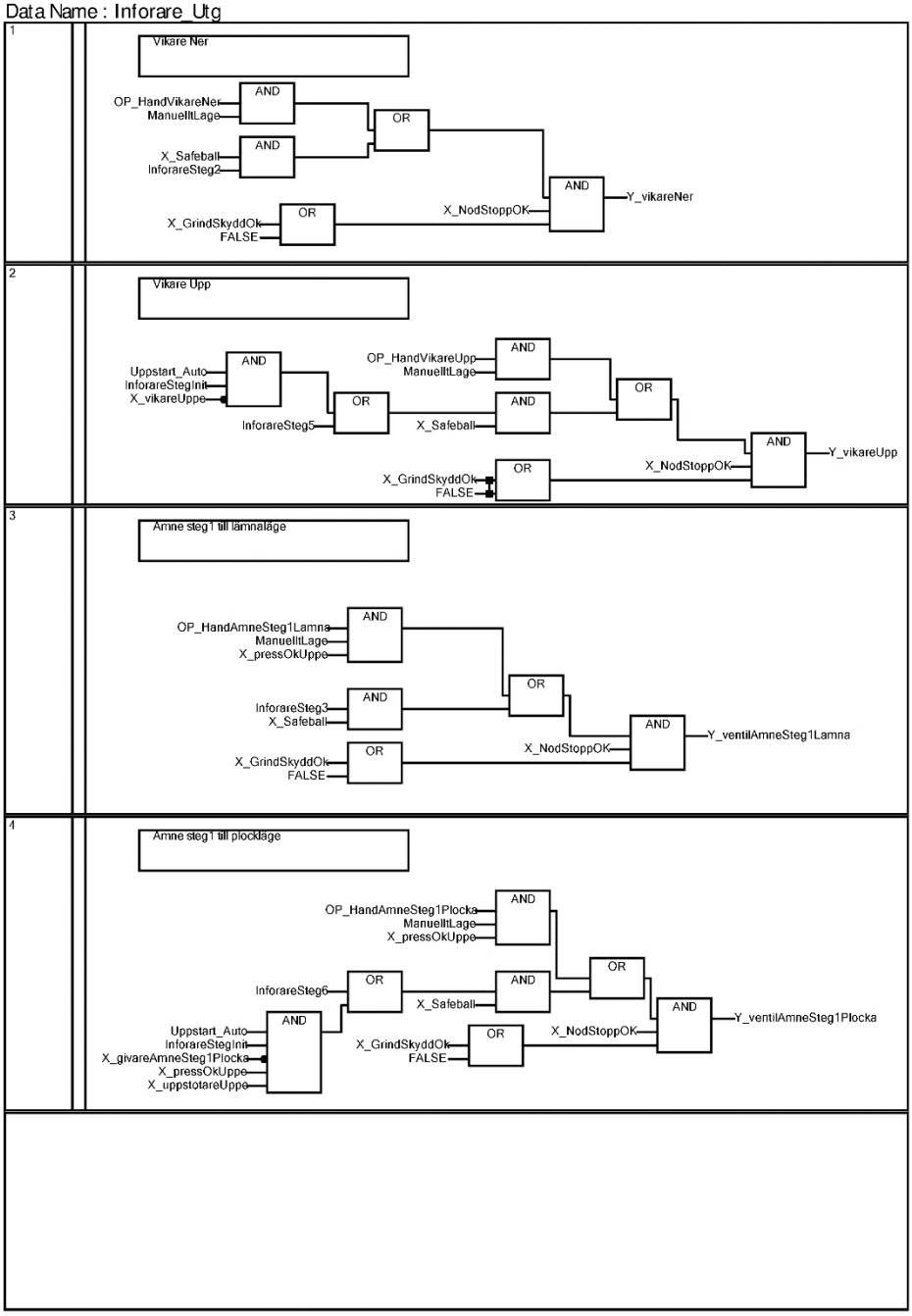
Data Name : Larm



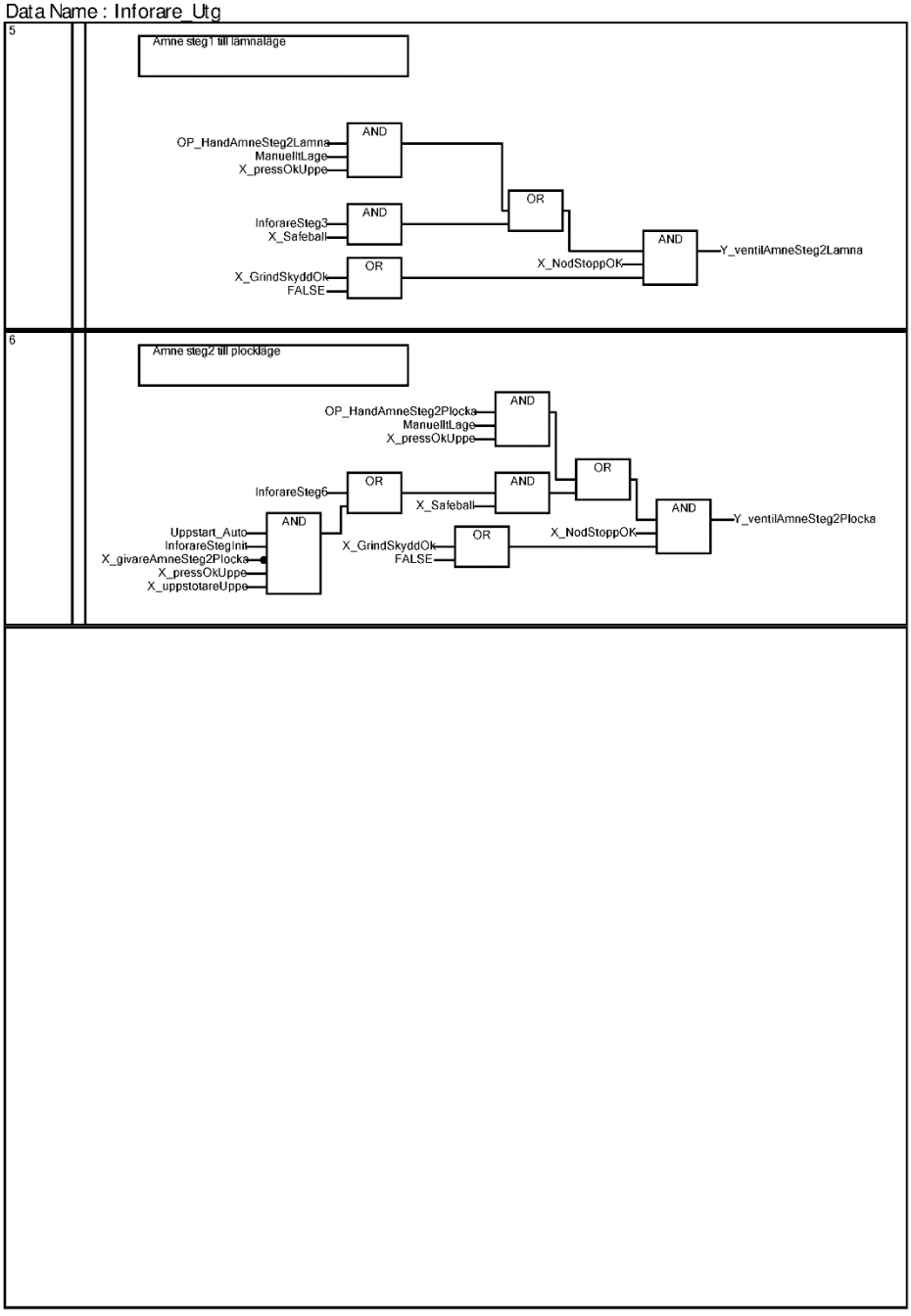
Data Name : Larm



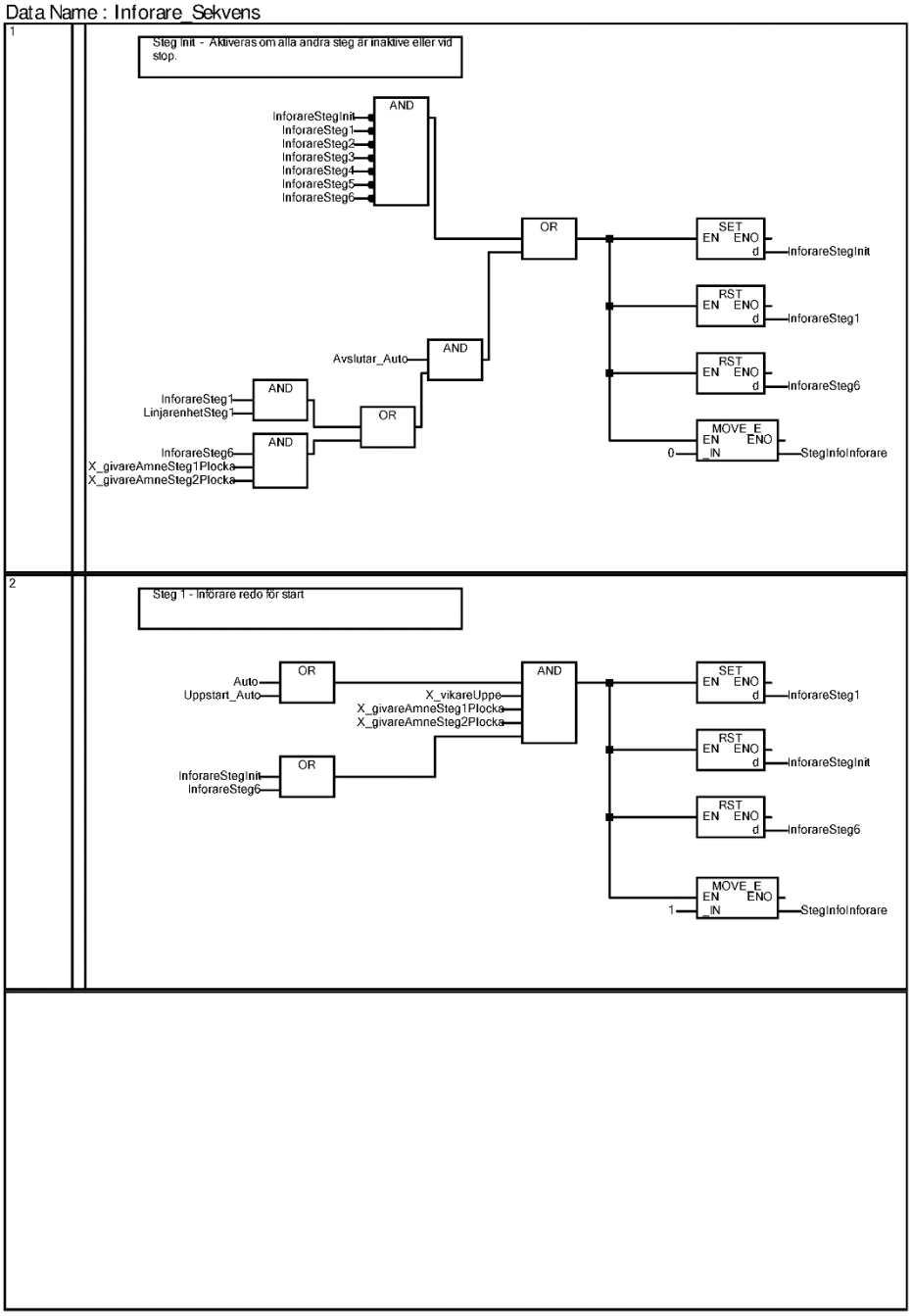
Data Name : Inforare_Utg



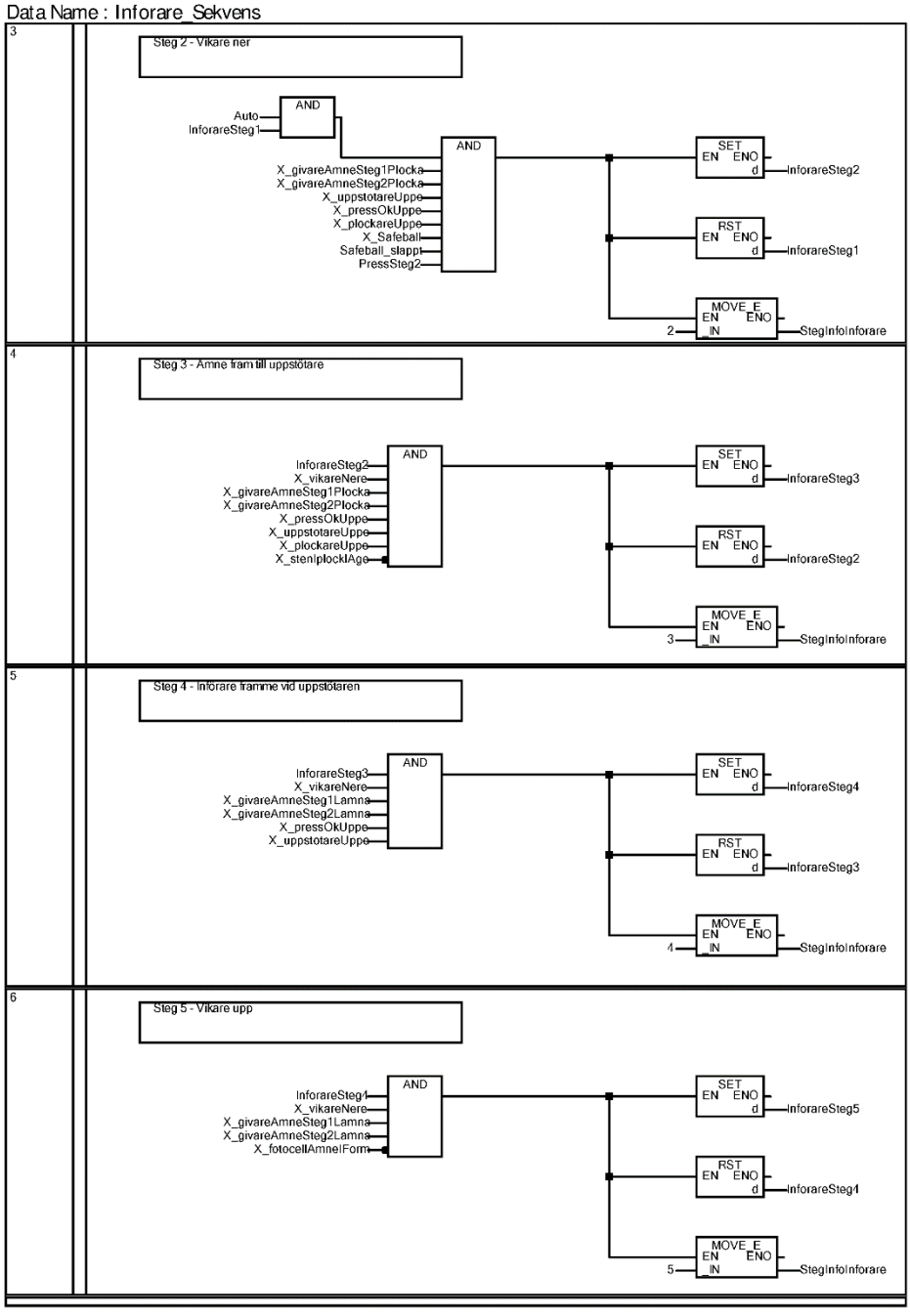
Data Name : Inforare_Utg



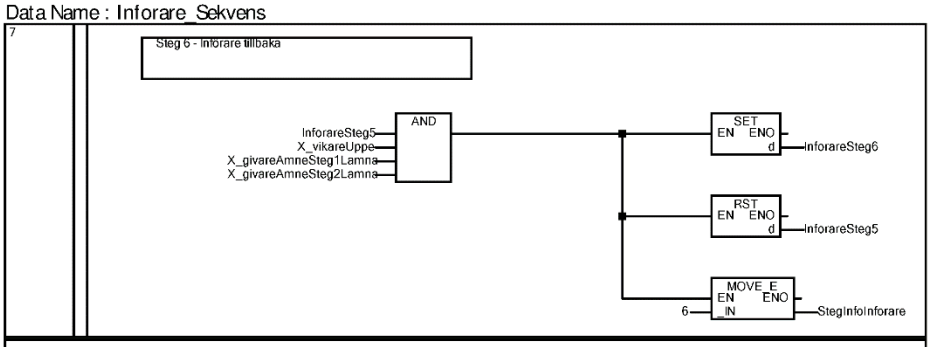
Data Name : Inforare_Sekvens



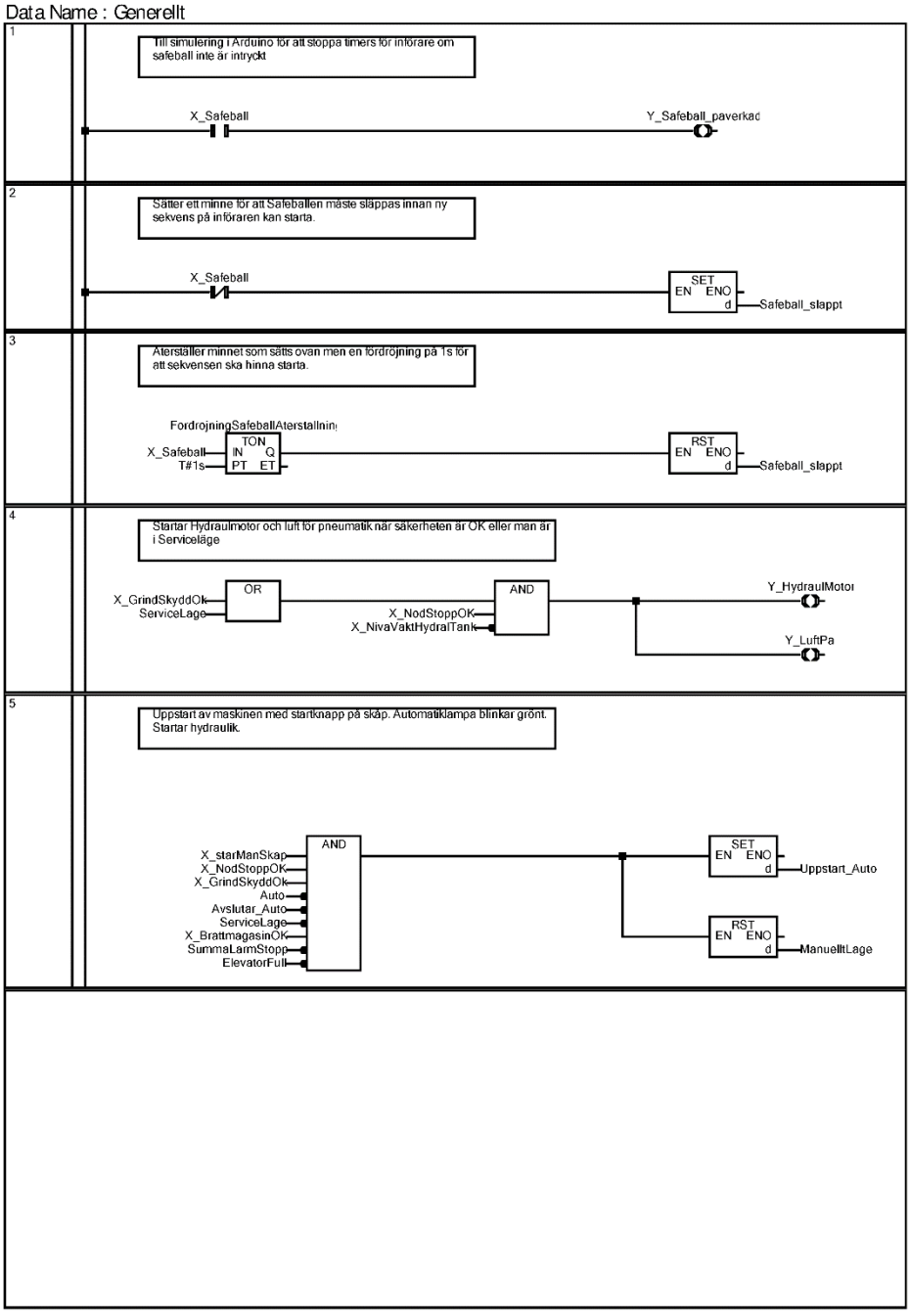
Data Name : Inforare_Sekvens



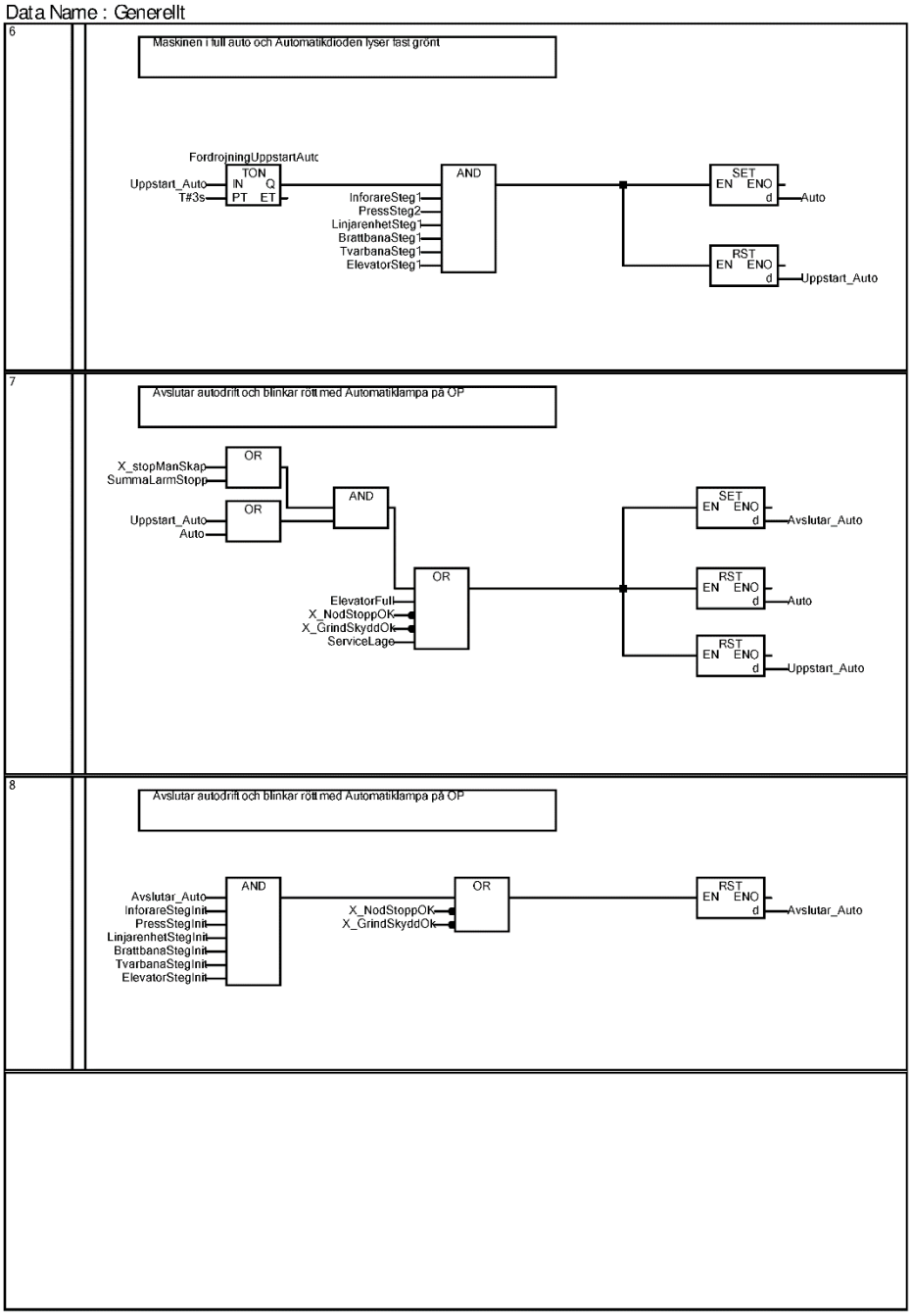
Data Name : Inforare_Sekvens



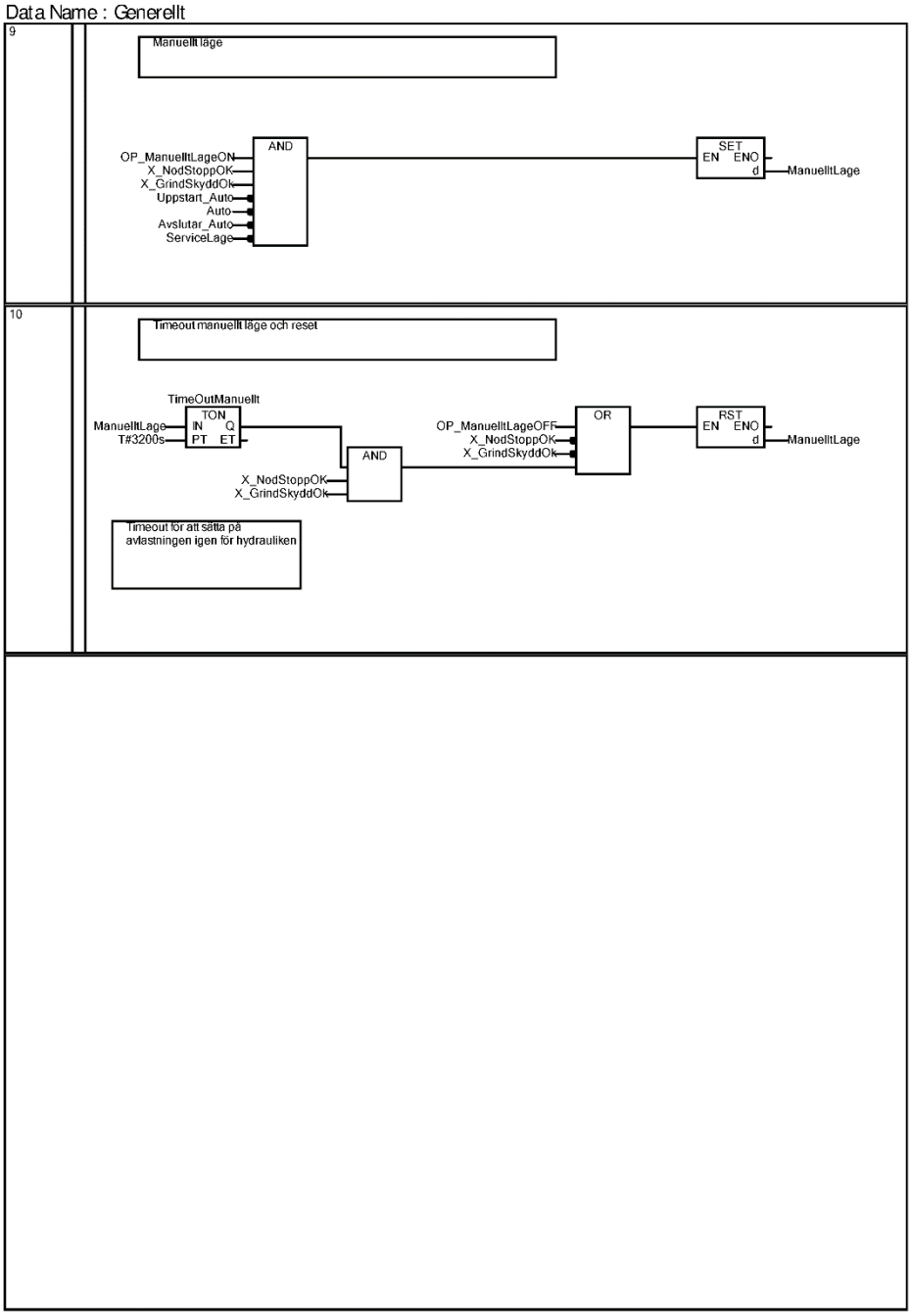
Data Name : GenerellT



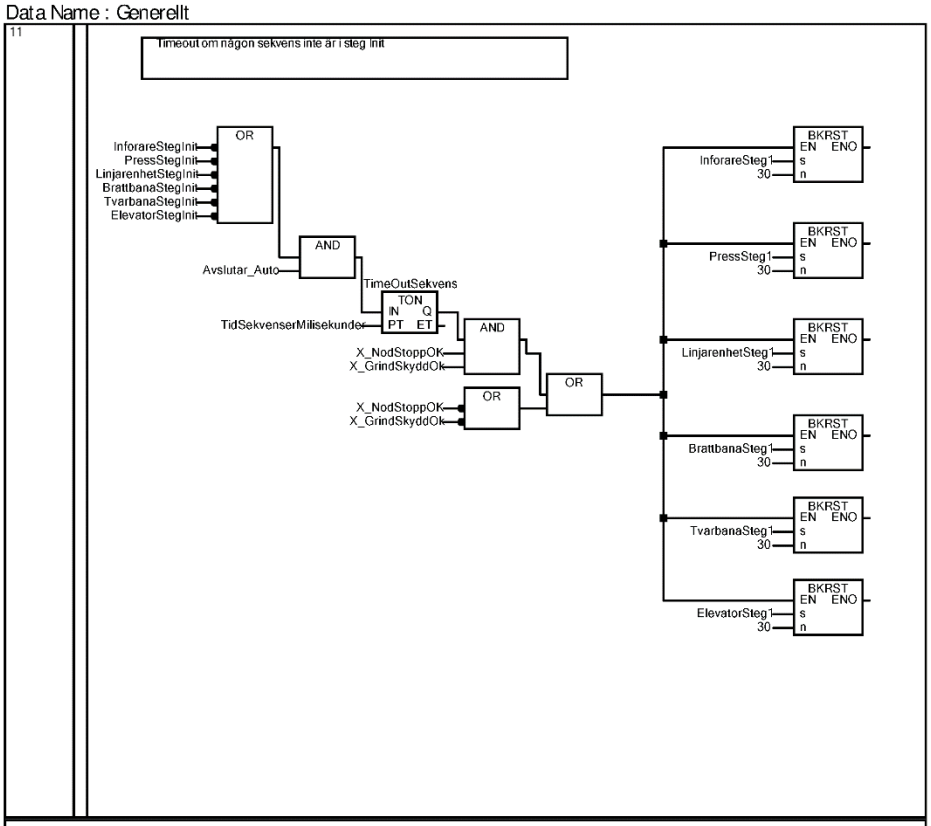
Data Name : GenerellT



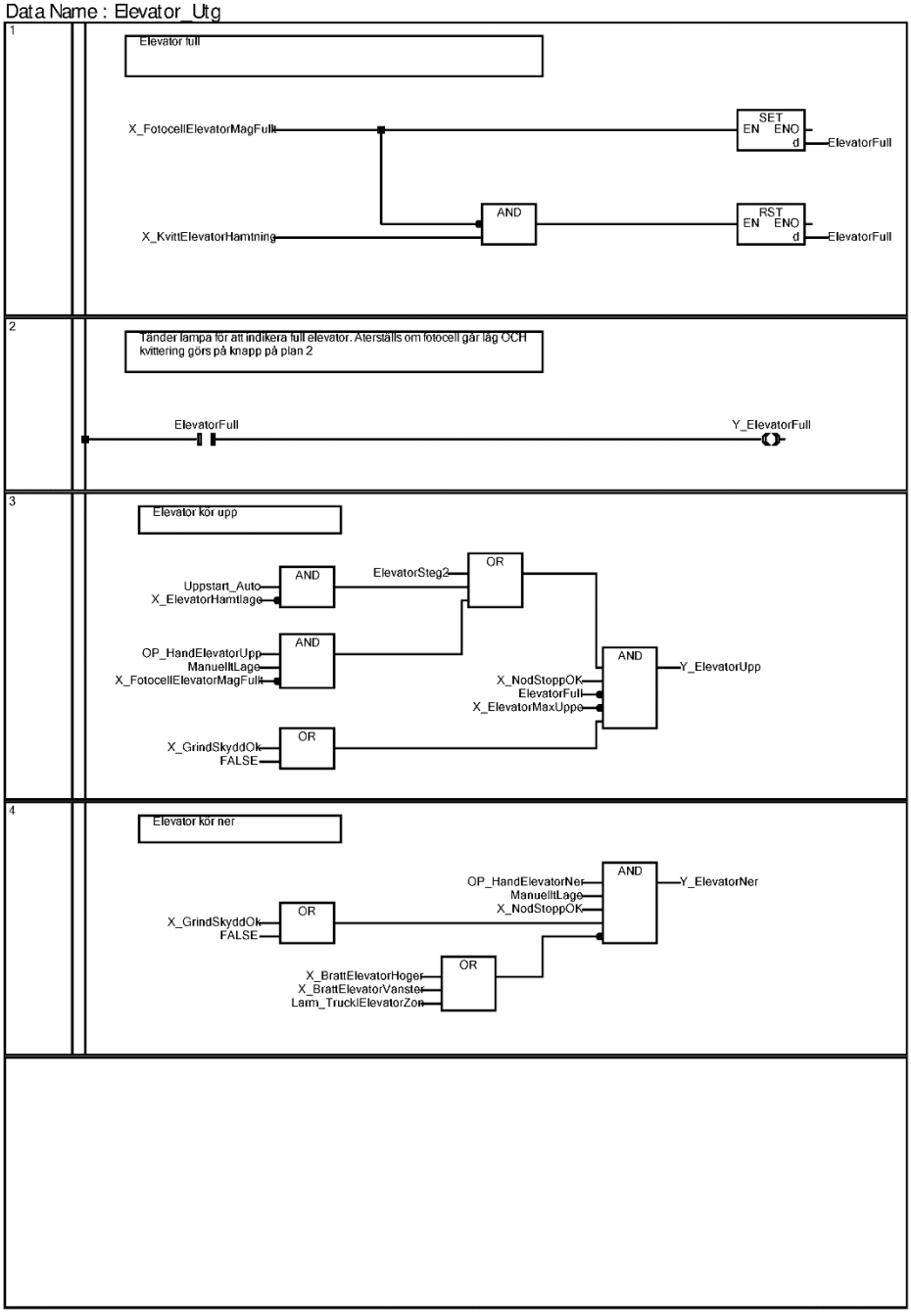
Data Name : GenerellT



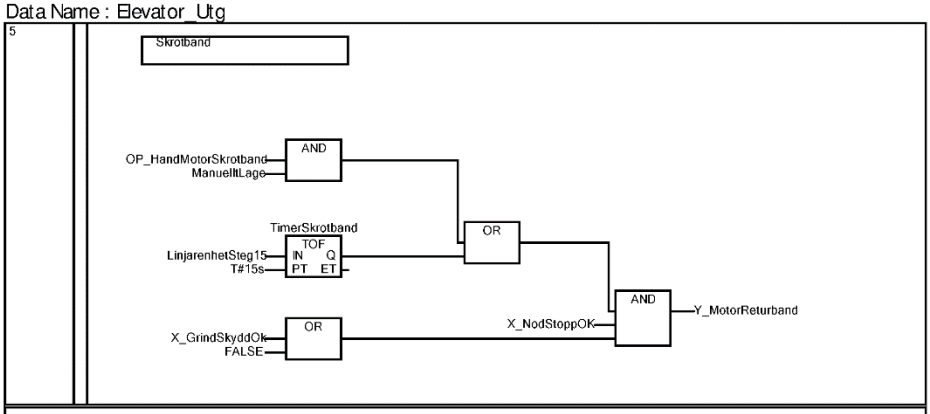
Data Name : GenerellI



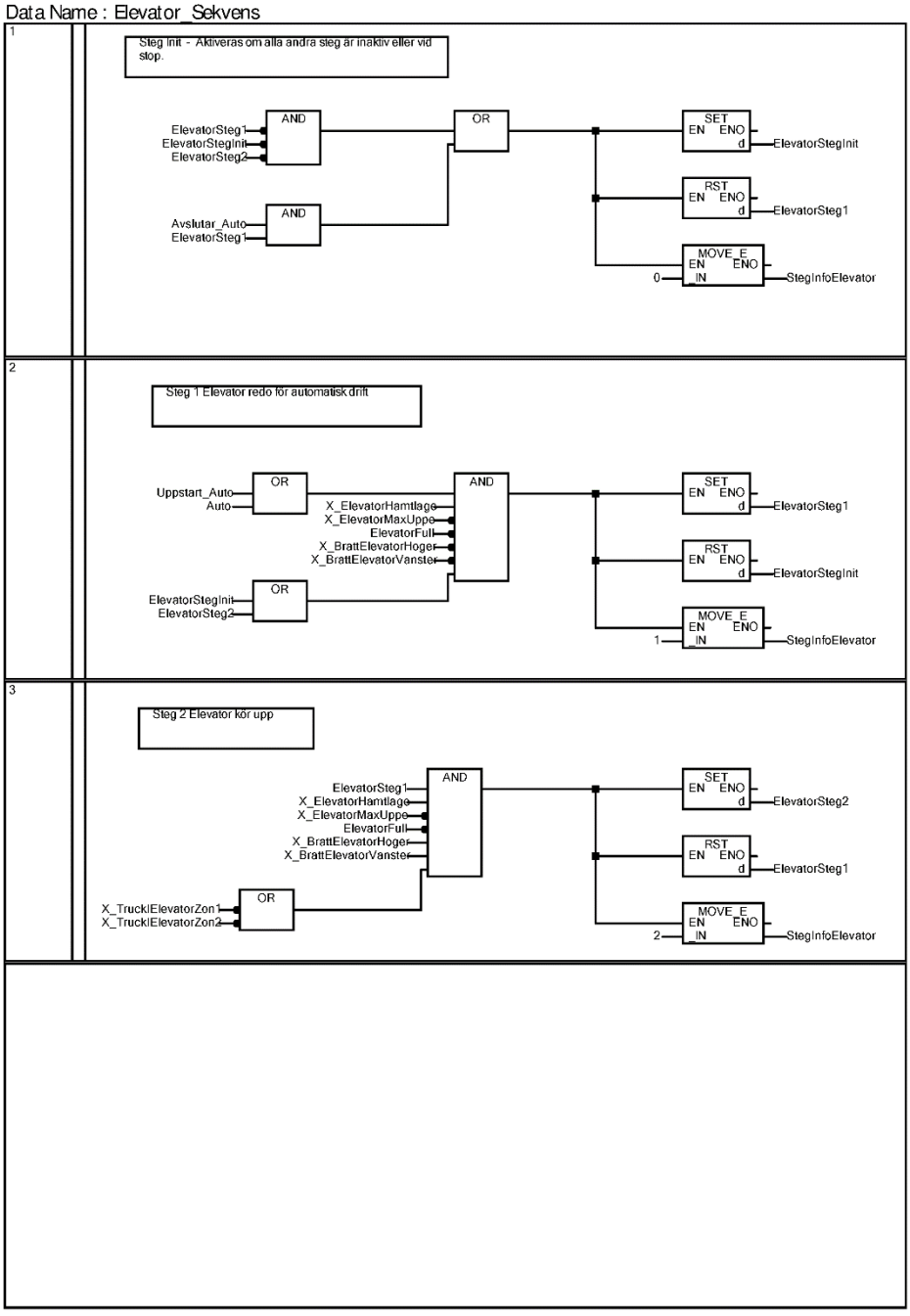
Data Name : Elevator_Utg



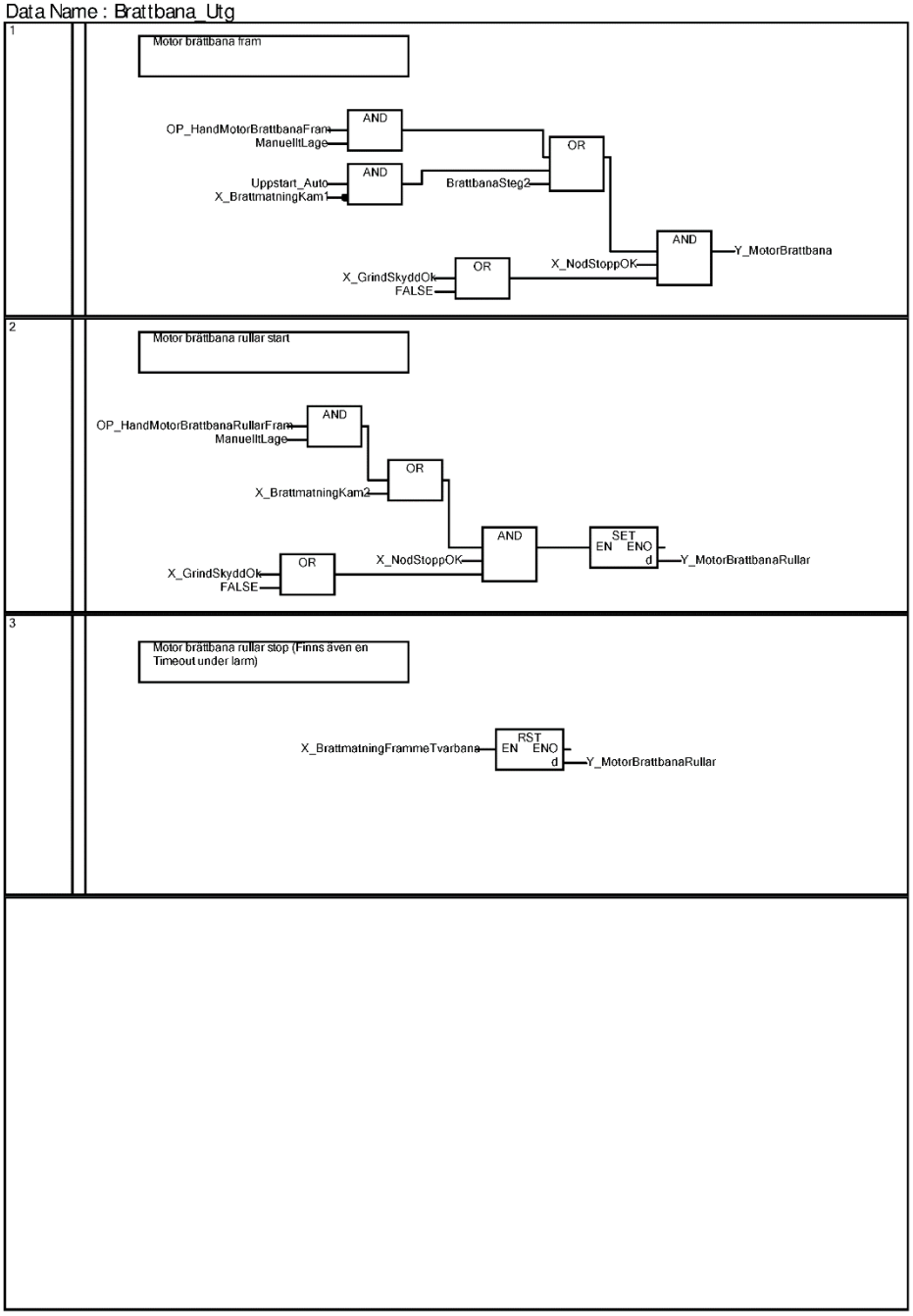
Data Name : Elevator_Utg



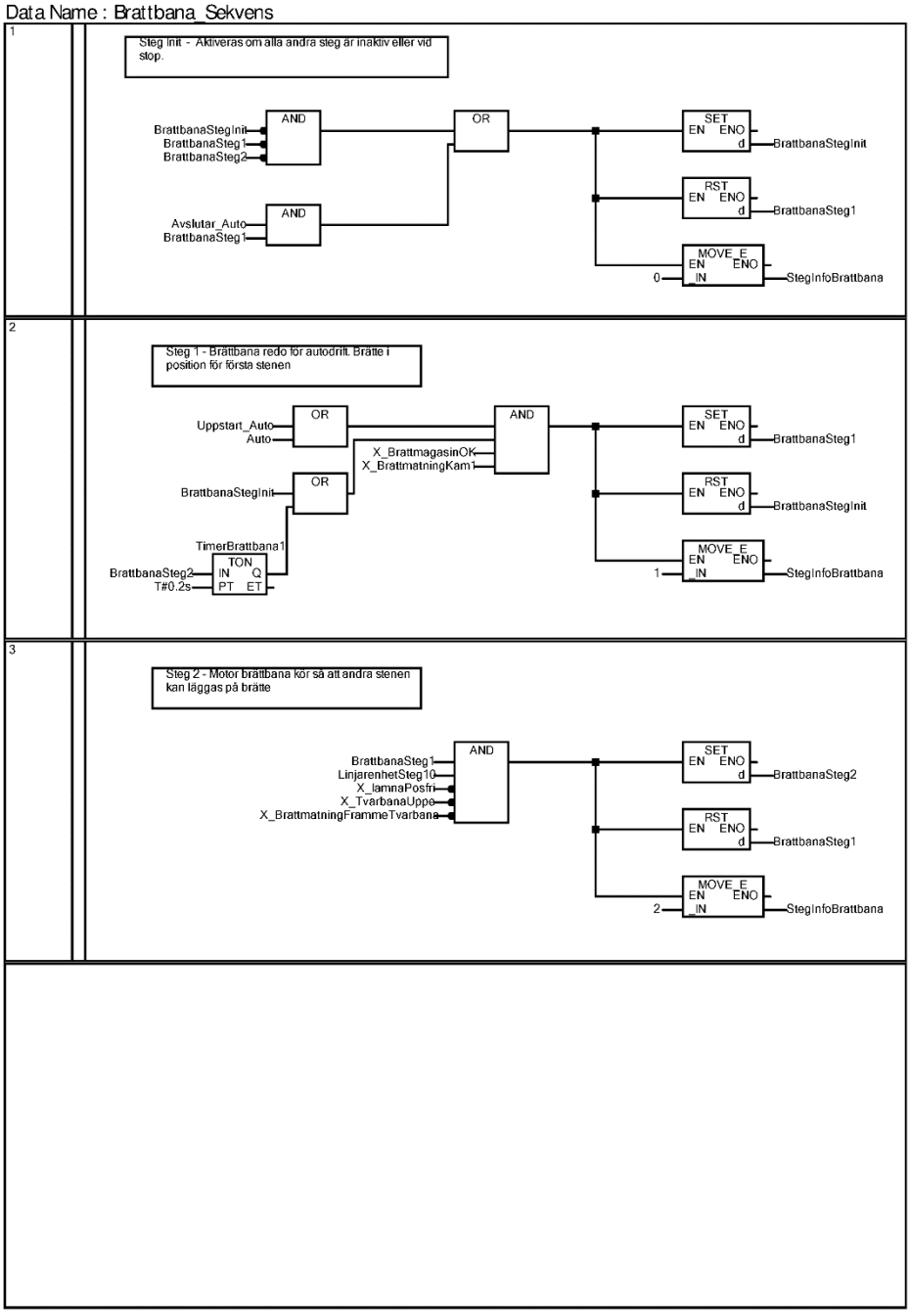
Data Name : Elevator_Sekvens



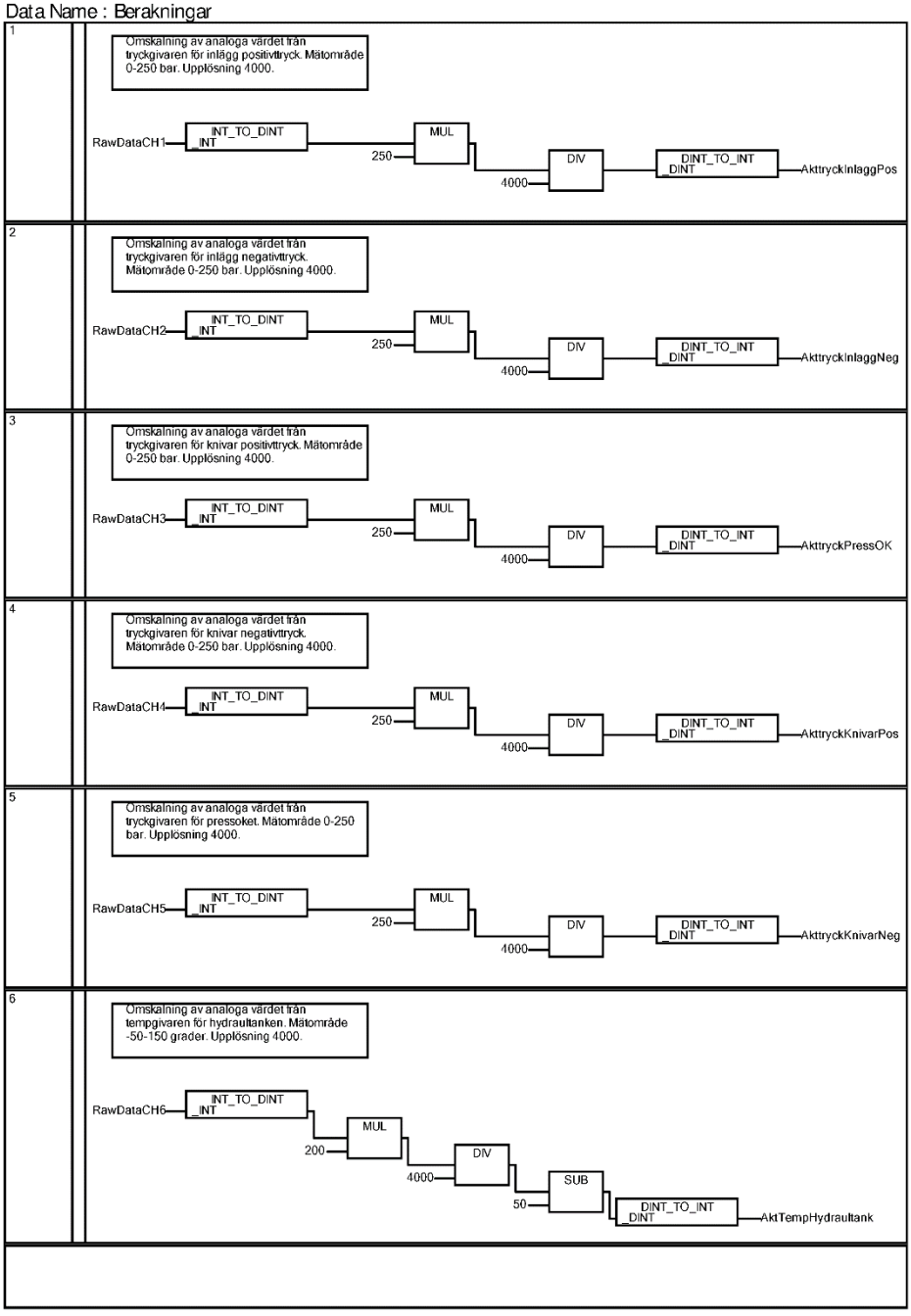
Data Name : Brattbana_Utg



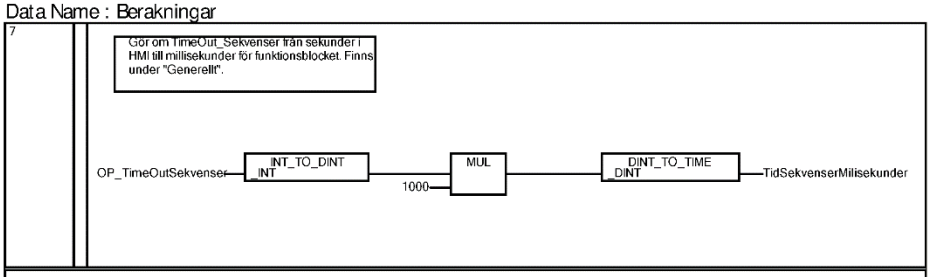
Data Name : Brattbana_Sekvens



Data Name : Berakningar



Data Name : Berakningar



7.3 HMI

7.3.1 Tagglista HMI

Benämning	Typ	Adress
Auto_indikator	BIT	M350
Manuell_drift	BIT	M354
Tryck_inlagg_pos	INT16	D201
Tryck_knivar_pos	INT16	D203
Tryck_pressok	INT16	D200
Tryck_inlagg_neg	INT16	D202
Tryck_knivar_neg	INT16	D204
Y_pressok_upp	BIT	Y49
Y_pressok_ner	BIT	Y4A
Y_knivar_in	BIT	Y56
Y_knivar_ut	BIT	Y57
Y_inlagg_in	BIT	Y54
Y_inlagg_ut	BIT	Y55
Y_uppstotare_upp	BIT	Y52
Y_uppstotare_ner	BIT	Y53
Y_inforaresteg1_fram	BIT	Y4D
Y_inforaresteg1_bak	BIT	Y4E
Y_inforaresteg2_fram	BIT	Y4F
Y_inforaresteg2_bak	BIT	Y50
Y_vikare_upp	BIT	Y4B
Y_vikare_ner	BIT	Y4C
OP_pressok_upp	BIT	M603
OP_pressok_ner	BIT	M602
OP_inlagg_in	BIT	M606
OP_inlagg_ut	BIT	M607
OP_knivar_in	BIT	M608
OP_knivar_ut	BIT	M609
OP_inforaresteg1_fram	BIT	M621
OP_inforaresteg1_bak	BIT	M622
OP_inforaresteg2_fram	BIT	M623
OP_inforaresteg2_bak	BIT	M624
OP_vikare_upp	BIT	M620
OP_vikare_ner	BIT	M619
OP_uppstotare_upp	BIT	M604
OP_uppstotare_ner	BIT	M605
Reset_raknare_producerat	BIT	M501

OP_tryck_inlagg	INT16	W100
OP_tryck_knivar	INT16	W101
OP_tryck_pressokLag	INT16	W102
OP_tryck_pressokHog	INT16	W103
SekvensInfo_inforare	INT16	W300
SekvensInfo_press	INT16	W301
SekvensInfo_linjarenhet	INT16	W302
SekvensInfo_brattbana	INT16	W303
SekvensInfo_elevator	INT16	W304
PeriodTimmar	INT16	D114
PeriodMin	INT16	D112
PeriodSek	INT16	D110
TotTimmar	INT16	D108
TotMin	INT16	D106
TotSek	INT16	D104
TotGkdTegel	INT16	D124
TotKasTegel	INT16	D120
PerKasTegel	INT16	D122
PerGkdTegel	INT16	D126
PerTotTegel	INT16	D118
TotAntalTegel	INT16	D116
OP_TimeOutSekvens	INT16	W110
Uppstart_auto	BIT	M351
Y_LasOppnaPlockare	BIT	Y48
Y_PlockareVrid	BIT	Y43
Y_PlockareOvrid	BIT	Y44
Y_PlockareUpp	BIT	Y45
Y_PlockareNer	BIT	Y46
Y_PlockareFram	BIT	Y40
Y_PlockareBack	BIT	Y41
Y_SkrotcylinderSkrot	BIT	Y69
Y_SkrotcylinderPlock	BIT	Y6F
OP_LasOppnaPlockare	BIT	M616
OP_PlockareVrid	BIT	M612
OP_PlockareOvrid	BIT	M613
OP_PlockareUpp	BIT	M611
OP_PlockareNer	BIT	M610
OP_PlockareFram	BIT	M614
OP_PlockareBack	BIT	M615
OP_SkrotcylinderSkrot	BIT	M617
OP_SkrotcylinderPlock	BIT	M618

Y_ElevatorNer	BIT	Y63
Y_ElevatorUpp	BIT	Y62
OP_ElevatorNer	BIT	M626
OP_ElevatorUpp	BIT	M625
OP_ManuellLageON	BIT	M502
Y_PressOkLasOppna	BIT	Y5C
Y_PressOkLasStang	BIT	Y5B
OP_PressOkLasOppna	BIT	M600
OP_PressOkLasStang	BIT	M601
OP_ManuellLageOFF	BIT	M503
Avslutar_auto	BIT	M352
Larm_MSK_UlostHydraulik	BIT	M100
Larm_MSK_UlostPlockare	BIT	M101
Larm_MSK_UlostReturband	BIT	M102
Larm_MSK_UlostBrattband	BIT	M103
Larm_MSK_UlostBrattbandRullar	BIT	M104
Larm_MSK_UlostTvarbana	BIT	M105
Larm_MSK_UlostElevator	BIT	M106
Larm_HydraulOljaHogTemp	BIT	M107
Larm_HydraulOljaLagNiva	BIT	M108
Larm_HydraulOljaLagTemp	BIT	M109
Larm_GrindSkydd	BIT	M110
Larm_NodStopp	BIT	M111
Larm_TimeOutInlaggIn	BIT	M112
Larm_TimeOutInlaggUt	BIT	M113
Larm_TimeOutKnivarIn	BIT	M114
Larm_TimeOutKnivarUt	BIT	M115
Larm_TimeOutPressOkUpp	BIT	M116
Larm_TimeOutPressOkNer	BIT	M117
Larm_TimeOutPressOkNerHog	BIT	M118
Larm_TimeOutUppstotareUpp	BIT	M119
Larm_TimeOutUppstotareNer	BIT	M120
Larm_TimeOutPressOkLasning	BIT	M121
Larm_TimeOutPressOkUpplasning	BIT	M122
Larm_VakuumLag	BIT	M123
Larm_TruckIElevatorZon	BIT	M126
Larm_LuftIn	BIT	M127
LarmKV_MSK_UlostHydraulik	BIT	M200
LarmKV_MSK_UlostPlockare	BIT	M201
LarmKV_MSK_UlostReturband	BIT	M202
LarmKV_MSK_UlostBrattband	BIT	M203

LarmKV_MSK_UlostBrattbandrullar	BIT	M204
LarmKV_MSK_UlostTvarbana	BIT	M205
LarmKV_MSK_UlostElevator	BIT	M206
LarmKV_HydraulOljaHogTemp	BIT	M207
LarmKV_HydraulOljaLagTemp	BIT	M208
LarmKV_HydraulOljaLagNiva	BIT	M209
LarmKV_GrindSkydd	BIT	M210
LarmKV_NodStopp	BIT	M211
LarmKV_TimeOutInlaggIn	BIT	M212
LarmKV_TimeOutInlaggUt	BIT	M213
LarmKV_TimeOutKnivarIn	BIT	M214
LarmKV_TimeOutKnivarUt	BIT	M215
LarmKV_TimeOutPressOkUpp	BIT	M216
LarmKV_TimeOutPressOkNer	BIT	M217
LarmKV_TimeOutPressOkNerHog	BIT	M218
LarmKV_TimeOutUppstotareUpp	BIT	M219
LarmKV_TimeOutUppstotareNer	BIT	M220
LarmKV_TimeOutPressOkLasning	BIT	M221
LarmKV_TimeOutPressOkUpplasning	BIT	M222
LarmKV_VakuumLag	BIT	M223
LarmKV_ElevatorMaxUppe	BIT	M225
LarmKV_TruckIElevatorZon	BIT	M226
Larm_ElevatorMaxUppe	BIT	M125
LarmKV_LuftIn	BIT	M227
Y_returband	BIT	Y65
OP_returband	BIT	M632
OP_skrapranna	BIT	M631
Y_skrapranna	BIT	Y66
SekvensInfo_tvarbana	INT16	W305
Y_brattmatning	BIT	Y5D
Y_brattRullar	BIT	Y5E
OP_brattmatning	BIT	M627
OP_brattRullar	BIT	M628
Y_tvarbanaFram	BIT	Y5F
Y_tvarbanaUpp	BIT	Y60
OP_tvarbanaFram	BIT	M630
OP_tvarbanaUpp	BIT	M629
X_plockarePlocklage	BIT	X0
X_plockareLamnalage	BIT	X1
X_plockareUppe	BIT	X2
X_plockareNere	BIT	X3

X_plockareVriden	BIT	X4
X_plockareOvriden	BIT	X5
X_stenIplocklAge	BIT	X6
X_lamnaPosfri	BIT	X7
X_pressOkUppe	BIT	X8
X_pressOKnere	BIT	X9
X_Safeball	BIT	XA
X_vikareUppe	BIT	XB
X_vikareNere	BIT	XC
X_givareAmneSteg1Lamna	BIT	XD
X_givareAmneSteg1Plocka	BIT	XE
X_givareAmneSteg2Lamna	BIT	XF
X_givareAmneSteg2Plocka	BIT	X10
X_uppstotareUppe	BIT	X11
X_uppstotareNere	BIT	X12
X_SkrotcylinderPlocklage	BIT	X13
X_fotocellAmneIForm	BIT	X14
X_pressOKLast	BIT	X15
X_pressOKOlast	BIT	X16
X_SkrotcylinderSkrotlage	BIT	X17
X_NodStoppOK	BIT	X18
X_GrindSkyddOk	BIT	X19
X_starManSkap	BIT	X1A
X_stopManSkap	BIT	X1B
X_KvittElevatorHamtning	BIT	X1C
X_TreLagesDon	BIT	X1D
X_BrattmagasinOK	BIT	X1E
X_BrattmatningKam1	BIT	X1F
X_BrattmatningKam2	BIT	X20
X_BrattmatningKam3	BIT	X21
X_BrattmatningFrammeTvarbana	BIT	X22
X_TvarbanaUppe	BIT	X23
X_Bratt1Tvarbana	BIT	X24
X_BrattElevatorHoger	BIT	X25
X_BrattElevatorVanster	BIT	X26
X_ElevatorHamtlage	BIT	X27
X_ElevatorMaxUppe	BIT	X28
X_TruckIElevatorZon1	BIT	X29
X_TruckIElevatorZon2	BIT	X2A
X_FotocellElevatorMagFullt	BIT	X2B
X_TryckVaktInLuft	BIT	X2C

X_VakuumVaktPlockare	BIT	X2D
X_NivaVaktHydraulTank	BIT	X2E
X_HydraulMotorMSKUtlost	BIT	X2F
X_PlockarMotorMSKUtlost	BIT	X30
X_ReturBandMotorMSKUtlost	BIT	X31
X_BrattBandMotorMSKUtlost	BIT	X32
X_BrattBandRullarMotorMSKUtlost	BIT	X33
X_TvarbanaMotorMSKUtlost	BIT	X34
X_ElevatorMotorMSKUtlost	BIT	X35
Trelagesdon	DEFAULT	
Larm_TimeOutBrattbanaRullar	BIT	M128
Larm_TimeOutVikareUpp	BIT	M129
Larm_TimeOutVikareNer	BIT	M130
Larm_TimeOutAmneSteg1Plock	BIT	M131
Larm_TimeOutAmneSteg1Lamna	BIT	M132
Larm_TimeOutAmneSteg2Plock	BIT	M133
Larm_TimeOutAmneSteg2Lamna	BIT	M134
Larm_TimeOutPlockareVrid	BIT	M135
Larm_TimeOutPlockareOvrid	BIT	M136
Larm_TimeOutPlockareUpp	BIT	M137
Larm_TimeOutPlockareNer	BIT	M138
Larm_TimeOutPlockareFram	BIT	M139
Larm_TimeOutPlockareBack	BIT	M140
Larm_TimeOutMotorBrattbana	BIT	M141
Larm_TimeOutTvarbanaUpp	BIT	M142
Larm_TimeOutMotorTvarbanaKort	BIT	M143
Larm_TimeOutMotorTvarbanaLang	BIT	M144
Larm_TimeOutElevatorUpp	BIT	M145
LarmKV_TimeOutBrattbanaRulllar	BIT	M228
LarmKV_TimeOutVikareUpp	BIT	M229
LarmKV_TimeOutVikareNer	BIT	M230
LarmKV_TimeOutAmneSteg1Plock	BIT	M231
LarmKV_TimeOutAmneSteg1Lamna	BIT	M232
LarmKV_TimeOutAmneSteg2Plock	BIT	M233
LarmKV_TimeOutAmneSteg2Lamna	BIT	M234
LarmKV_TimeOutPlockareVrid	BIT	M235
LarmKV_TimeOutPlockareOvrid	BIT	M236
LarmKV_TimeOutPlockareUpp	BIT	M237
LarmKV_TimeOutPlockareNer	BIT	M238
LarmKV_TimeOutPlockareFram	BIT	M239
LarmKV_TimeOutPlockareBack	BIT	M240

LarmKV_TimeOutMotorBrattbana	BIT	M241
LarmKV_TimeOutTvarbanaUpp	BIT	M242
LarmKV_TimeOutMotorTvarbanaKort	BIT	M243
LarmKV_TimeOutMotorTvarbanaLang	BIT	M244
LarmKV_TimeOutElevatorUpp	BIT	M245
OP_HydraulOljaHogTemp	INT16	W112
OP_HydraulOljaLagTemp	INT16	W113