Realisering av SCADA-system

LUNDS UNIVERSITET
Lunds Tekniska Högskola

LTH Ingenjörshögskolan vid Campus Helsingborg
Elektroteknik med automationsteknik

Examensarbete:
Robin Levenhammar
Markus Björk

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
Media-Tryck
Biblioteksdirektionen
Lunds universitet
Lund 2014

Sammanfattning

Det här examensarbetet beskriver processen att realisera ett SCADA-system med hjälp av Codesys, iX developer samt ett givet driftkort. Det följs upp av en mindre utredning om Modbusprotokollet, vilket tillämpas för att göra kommunikationen mellan de olika delarna i systemet möjlig.

Rapporten består av en grundläggande teoridel som behandlar begrepp som SCADA, PLC, I/O-enheter, kommunikation och andra termer som rör arbetet. Tanken med teoridelen är att man genom att ta hjälp av den ska få en bättre bild över de områden som vårt arbete behandlar. Teoridelen avslutas med en mer omfattande beskrivning av kommunikationsprotokollet Modbus.

SCADA-systemet delades upp i två delar. En där PLC-programmering ingick och en där design av användargränssnitt, HMI, var det centrala. Därefter utnyttjades en Modbus TCP/IP-simulator för att kontrollera huruvida gränssnittet fungerade eller ej. PLC-programmet blev i sin tur kontrollerat av en virtuell PLC-simulator som fanns inbyggd i Codesys.

Nyckelord: PLC, Codesys, HMI, iX developer, Modbus, SCADA.

Abstract

This is a thesis which describes a process for realizing a SCADA system using Codesys and iX developer, after a given function description. This is followed by an investigation about the Modbus protocol. That is applied to do the communication possible between the different parts in the system.

The report includes a simple part of theory. This treats concepts like SCADA, PLC, I/O-devices, communication and other terms that our study has an influence on. The whole idea about this section is to help the reader to obtain an understanding of the area that our study is concerned with. The section will end with an investigation about the communication protocol named Modbus.

The SCADA system was split into two parts. The first part includes the PLC programing and the other part concerns the design of a human machine interface, HMI. To see how the interface worked, a Modbus TCP/IP simulator was used. Our PLC program was checked by a virtual PLC simulator that was built into the environment of Codesys.

Keywords: PLC, Codesys, HMI, iX developer, Modbus, SCADA.

Förord

Följande examensarbete motsvarar 22,5 högskolepoäng och är den avslutande delen i vår utbildning, högskoleingenjör inom elektroteknik med automationsteknik. Utbildningen tillhör Lunds Tekniska Högskola/Campus Helsingborg. Arbetet är utfört under våren 2014 på Goodtech AB i Malmö.

Vi vill rikta ett stort tack till Lars Kjällström och Jamal Ibrahim på Goodtech. Dels för möjligheten att utföra vårt examensarbete hos er, men också för all hjälp vi har fått under arbetets gång. Ett tack riktas även till Mats Lilja som, utöver att ställa upp som vår examinator, också har kommit med värdefulla tips och råd.

Terminologi

ACK	Acknowledgement, används inom TCP som bekräftelse.
ADU	Application Data Unit, datapaket för applikationsnivån.
ADC	Analog to Digital Conversion, en funktion som omvandlar en analog signal till en diskret digital motsvarighet.
ARP	Address Resolution Protocol, kommunikationsprotokoll som kopplar samman IP-adresser med MAC-adresser.
BOOL	Boolean, en datatyp som enbart kan anta sant eller falskt värde. Detta representeras som 1 respektive 0 binärt.
CRC	Cyclical Redundany Check, en metod för att räkna ut en kontrollsumma av den data som ska skickas eller tas emot.
CRLF	Carriage Return and Line Feed, används för att markera radbrytning.
CSMA/CD	Carrier Sense Multiple Access/Collision Detection, ett protokoll som Ethernet använder sig av för att kunna skicka data och upptäcka eventuella kollisioner.
DAC	Digital to Analog Conversion, en funktion som omvandlar digital data till analog.
FIN	Finish, används inom TCP när en förbindelse skall stängas.
HMI	Human Machine Interface, ett användargränssnitt som möjliggör interaktion mellan människa och maskin.
INT	Integer, en datatyp som representerar ett heltal.
IP	Internet Protocol, kommunikationsprotokoll för överföring av information över internet.
LRC	Longitudinal Redundancy Check, metod för uträkning av kontrollsumma av data som skickas eller tas emot.
MAC-adress	Media Access Control-adress , fungerar som en unik identifierare för nätverkskort.
MBAP	Modbus Application Protocol, en header som läggs på applikationen för en PDU, tillhörande Modbus TCP/IP.
OSI-modell	Open Systems Interconnect-modell, en standardmodell för datorkommunikation uppdelat i 7 lager.
RARP	Reverse Address Resolution Protocol, protokoll på nätverksnivå som tillämpas av en dator som vill ha reda på sin IPv4-adress.
REAL	Datatyp som motsvarar ett reellt tal.

RTU	Remote Terminal Unit, mikroprocessorstyrd enhet som skickar mätdata till ett överordnat system samt styrsignaler till kringenheter i systemet.
PLC	Programmable Logic Controller, är ett programmerbart styrsystem.
POU	Program Organisation Unit, används för att strukturera upp programmet i Codesys.
SCADA	Supervisory Control and Data Acquisition, ett system som övervakar och styr processer.
SYN	Synchronize, synkronisera.
PDU	Protocol Data Unit, innefattar en samling av data.
P2P	Peer to Peer-nätverk, ett nätverk som inte använder master/slav-tekniken. Inga roller används utan alla noder har samma rättigheter.
TCP	Transmission Control Protocol, ett protokoll som beskriver hur data skickas i nätverk.
Telemetri	Används för att överföra mätdata trådlöst

Innehållsförteckning

1 Inledning 2
1.1 Bakgrund 2
1.2 Syfte 2
1.3 Problemformulering 3
1.4 Begränsningar 3
1.5 Källkritik 4

2 Teori 5
2.1 Inledning 5
2.2 SCADA 5
2.3 PLC 6
2.4 Codesys 7
2.5 HMI 8
2.6 I/O-enheter 8
2.7 Kommunikation 9
2.8 Modbus 11
2.8.1 Modbus ASCII/RTU 14
2.8.1.1 Meddelande 14
2.8.1.2 Adressfält 14
2.8.1.3 Funktionsfält 15
2.8.1.4 Datafält 15
2.8.1.5 Felsökningsmetoder 16
2.8.1.6 Parity checking 16
2.8.1.7 Modbus ASCII 17
2.8.1.8 RTU 18
2.8.2 Modbus TCP/IP 20
2.8.2.1 Inledning 20
2.8.2.2 Hur förhåller sig Modbus TCP/IP till OSI-modellen? 20
2.8.2.3 Applikationsnivå 21
2.8.2.4 Transportnivå 23
2.8.2.5 Nätverksnivå 24
2.8.2.6 Länknivå 25

3 Metod 27

4 Utförande 28
4.1 Codesys 28
4.1.1 Att skapa ett funktionsblock 28
4.1.2 De Olika funktionsblocken 29
4.1.2.1 Spjäll 29
4.1.2.2 Reglerbart spjäll 30
4.1.2.3 Start/Stopp-motor 31
4.1.2.4 Frekvensmotor 32

4.1.2.5 Skalningsblock *34*
4.1.2.6 Temperaturblock *35*
4.1.2.7 Tryckblock *36*
4.1.2.8 Elluftvärmare *36*
4.1.3 Uppbyggnad av PLC-program 37
4.1.4 Simulering av PLC-program 39
4.1.5 I/O-lista 40
4.2 Framtagning av HMI 40
4.2.1 Översikt 40
4.2.2 Aggregat 40
4.2.3 Kommunikation och taggar i HMI 41
4.2.4 Larm 42
4.2.5 Trender 42
4.2.6 Simulering av HMI 42
5 Diskussion och slutsats 46
5.1 Vidarestudie 47
6 Bilagor 48
6.1 Funktionsblocken 48
6.2 Driftkortsbeskrivning 55
6.3 I/O-lista 60
6.4 Nätverk för AL001 61
6.5 Taggar i HMI 67
6.6 Sekvensdiagram för motion 73
7 Referenslista 74

Om företaget

Goodtech grundades år 1913 som en kabelfabrik, då med namnet Norsk elektrisk kabelfabrik. Man var en kabelfabrik fram till slutet av 80-talet då produktionen avvecklades, mycket på grund av den stora bank- och börskrisen. Istället började man titta framåt på vad som skulle bli den nya marknaden. Kabeltekniken byttes ut mot miljöteknologi och företaget tog sig det nya namnet, Goodtech. Med det nya tänket växte Goodtech i Norge och Sverige. Idag är Goodtech en av Nordens ledande leverantörer inom automation, el, industri- och miljöteknik.

Man är delaktig i stora projekt så som att ta fram citytunneln i Malmö, MAX-lab IV i Lund, Tele2 arena och nu även Mall of Scandinavia i Stockholm.

Framförallt inom fastighetsautomationen har man blivit riktigt stora. Där sysslar man bl.a. med att ta fram SCADA-system vilket vårt examensarbete också kommer att handla mycket om.

1 Inledning

1.1 Bakgrund

Allt större krav ställs på hur reglering och stryning ska ske för att produktionen ska öka (Goodtech, u.å.). När det gäller styrning och övervakning av system inom industrin finns det ett antal olika lösningar. En variant av överordnat system är DCS, vilket är en användningsmetod för att kontrollera enheter som finns i ett system. Den andra varianten heter SCADA-system och det används väldigt mycket inom industri och infrastruktur. Numera används SCADA mycket även inom fastighetsautomation. Själva idén bygger på övervakning och styrning av automatiserade processer i realtid. På så sätt kan man upptäcka eventuella driftstörningar och andra fel som kan uppstå i ett tidigare skede. Genom att ha kontroll på systemets händelser i realtid blir det enklare att se vilka åtgärder som ska göras för att effektivisera systemet.

Övervakningen av systemet sker oftast via ett HMI vilket gör det enkelt för operatören att se hur den automatiserade processen rullar (ibid). Det ger en grafisk bild för att enkelt kunna följa olika trender i processen.

I SCADA-system interagerar en PLC-del med en HMI-del. Den tillhörande standarden 61131-3 används för att programmera PLC-mjukvaran och på så sätt sätta ihop det väl fungerande programmet.

I de traditionella systemen utbyts information mellan styrenhet och instrumentella enheter parallellt (Zurawski 2009, s 2,3,4 kap20). Men med tanke på att systemet fysiskt inte blir särskilt komprimerat har man tagit tekniken vidare med hjälp av fältbussar. Dessa knyter samman styrenheten med de olika instrumentella enheterna på ett seriellt vis och kablage som tidigare tog upp stor plats förhindras. Denna teknik bärs vidare för att binda samman automationstekniken med informationstekniken som oftast är Ethernet baserad.
Beroende på vilken miljö och överföringshastighet man vill ha finns det en mängd olika fältbussar som följs under standarden 61158.

1.2 Syfte

Det finns ett par olika syften med examensarbetet. Först och främst att ta fram ett renodlat SCADA-system med ett PLC-system som styrning. Vidare att utreda hur kommunikationen mellan systemets olika enheter ska fungera tillsammans. Detta omfattar HMI, PLC-mjukvara, I/O-enheter och styrenheter.

1.3 Problemformulering

Examensarbetet, som vi skall utföra på Goodtech, går ut på att ta fram ett renodlat SCADA-system, med ett PLC-system för styrning. Systemet ska användas för styrning/övervakning av ett ventilationssystem. Styrdelen innefattar ett PLC-system och övervakningsdelen ett HMI-system.

Till en början behöver vi implementera funktionsblock för de komponenter som bygger upp systemet. Vi kommer att skapa dem i programmet Codesys som kommer att vara vår arbetsmiljö gällande PLC-delen.

När ett välfungerande PLC-system är färdigt ska vi skapa ett HMI. Dessa system ska interagera med varandra för att SCADA-systemet ska vara fullständigt.

För att SCADA-systemet ska fungera behöver vi välja vilken kommunikationsstandard som ska tillämpas. Vi kommer att utreda en standard som lämpar sig bäst för vår uppgift.

För att nå slutmålet har vi tagit fram ett gantt-schema:

Realisering av SCADA-system

DM#	Delmoment	Krav	Vecka
			1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
1	Rapport		
2	Inlärning av Codesys		
3	Skapa funktionsblock	2	
4	Tolka driftkort		
5	Uppbyggnad av PLC-system	3,4	
6	Skapa en I/O-lista	5	
7	Test av PLC-program	6	
8	Inlärning av HMI-miljö		
9	Skapa en visulation i HMI-program	8	
10	Utreda kommunikationsstandarder		
11	Test av HMI	9	
12	Finslipning		
13	Slutprodukt		

1.4 Begränsningar

Arbetet kommer att rikta sig mot att ta fram PLC-mjukvara och ett användargränssnitt, HMI. Undersökningen kommer att rikta sig mot hur kommunikationen fungerar i automationspyramiden. I den mängd av de kommunikationsstandarder som används kommer vi att rikta blickarna mot att undersöka en specifik standard, som riktar sig mot industriell automation och industriellt Ethernet.

1.5 Källkritik

Ett flertal av källorna som vi har använt oss av publicerades för många år sedan. Tekniken inom elektroteknik och datakommunikation utvecklas i en enorm takt. Detta gör att en del parametrar och information inte överrenstämmer med den tid som vi lever i nu. Exempelvis är transmissionstid en parameter som har ökat mycket.

Vi har många gånger använt dokument som publicerats av människorna bakom tekniken. Modicon, som var med och utvecklade Modbusprotokollet, har skrivit en guide som finns att hämta som PDF-fil. Med tanke på att det är deras produkt har vi förtroende för den källan. Likadant vad gäller fakta om programmet Codesys och iX developer som vi hämtade från deras egen hemsidan.

Vi tror att man inte har valt att skriva om en ny omfattande beskrivning för t.ex. Modbus eftersom grunden inte förändrats nämnvärt med tiden.

Förutom dokument skrivna av utvecklarna har vi studerat annat material som i sin tur har utvecklarnas dokument som referens. PDF-filen som Pefhany har författat är ett sådant exempel. Inledningsvis i Pefhanys dokument nämner han Modicons hemsida som referens (Pefhany. S, 2000).

2 Teori

2.1 Inledning

För att skapa en bättre förståelse för arbetet kommer vi att ta upp och grundligt förklara termer som innefattas i vårt arbete. Tanken är att det ska underlätta för läsaren så att det blir enklare att följa med.

2.2 SCADA

Lika länge som styrsystem av olika slag har existerat, har även SCADA funnits (Medida 2008, s. 127). SCADA är en kombination av telemetri och insamling av data.

SCADA är inte ett renodlat styrsystem utan fokus ligger till största del på övervakning (Daneels och Salter 1999, s. 339-340). Två områden där man väldigt ofta utnyttjar SCADA-system är inom industri och infrastruktur. Det finns både små och stora system, allt från versioner som behandlar 1000 till 10 000 olika I/O-kanaler. Samtidigt utvecklas SCADA i en mycket hög takt och det är idag möjligt att hitta system bestående av omkring 100 000 olika I/O-kanaler.

Att utvecklingen sker i en hög hastighet är inte enbart positivt, det kan också medföra nackdelar (ibid). SCADA drar mycket nytta av informationsteknologi vad gäller mjukvara. Framtagen mjukvara kan därför bli gammalmodig och obrukbar i takt med att utvecklingen rusar fram.

Ett SCADA-system har till uppgift att samla in information från en eller flera processer och skicka den vidare till en central (Medida 2008, s. 127). Med hjälp av insamlad data genomförs lämpliga analyser. Analyser och tillhörande resultat visas på förutbestämda operatörspaneler. Därifrån skickas nödvändiga styrsignaler tillbaka till de processer som ingår i systemet.

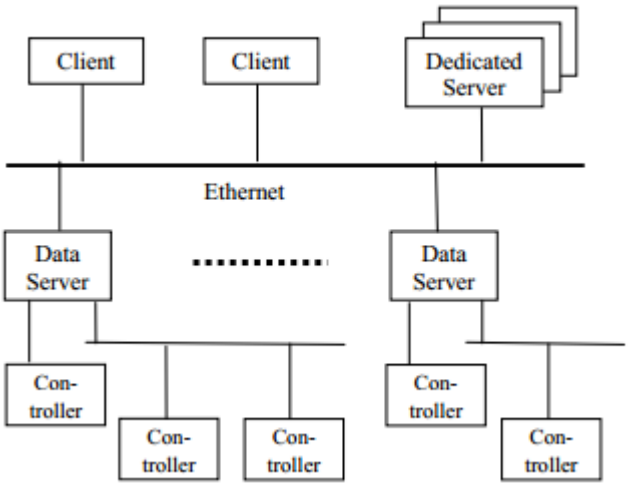
Genom att välja SCADA får operatörer en lättöverskådlig bild över hela sitt system och att skicka styrsignaler i realtid blir en möjlighet (Stouffer, Falco och Kent 2006, kap. 2, s. 6). Med andra ord går det bra att övervaka och styra hela system i realtid ifrån en central. För varje individuellt system ges möjligheten att bestämma huruvida styrning av specifika system, operationer eller uppgifter ska ske automatiskt eller manuellt via kommando från operatör.

Ett SCADA-system består av både mjuk- och hårdvara (Daneels och Salter 1999, s. 339-340). Enkelt sagt finns det två delar som tillsammans bygger upp arkitekturen för hårdvaran. Den ena delen behandlar främst kommunikation

mellan klient och data (klientlagret), medan den andra har till uppgift att sköta styrning eller andra aktiviteter som rör processer (datalagret).

För att möjliggöra kommunikation mellan dataservrar och komponenter tar man hjälp av någon typ av "process controller" (ibid). En PLC är en vanligt förekommande sådan. PLCn i sin tur ansluts antingen direkt eller med hjälp av nätverk/fältbussar. När information kommit fram till PLCn används den sedan för att skicka styrsignaler vidare till ställdon och/eller för att bevaka sensorer (Stouffer, Falco och Kent 2006, kap. 2, s. 6). Vid sammankoppling av de två delarna, samtliga data- och klientservrar, utnyttjar man kommunikationssystem (Daneels och Salter 1999, s. 339-340). Ethernet är mycket vanligt förekommande.

Lagring av mätdata, loggning, larmcheckar etc. sker i realtidsdatabaser som placeras i en eller flera servrar i systemet (ibid).



Överblick på hårdvaruarkitekturen hos ett SCADA-system.
(Daneels och Salter 1999, s. 339).

2.3 PLC

En PLC är ett mikroprocessorbaserat styrsystem där programmerbart minne utnyttjas för att dels lagra instruktioner och dels för att införa funktioner (Bolton 2009, kap. 1, s. 3). Vid programmering avses i första hand implementation av logiska operationer. Exempelvis om temperaturen T1 överstiger en förutbestämd nivå slås fläkt för kalluft igång. För att, som i föregående exempel, ha kontroll över när en temperatur blivit för hög/låg tar

man hjälp av sensorer. En sensor är en typisk manick vars signal skickas till en bestämd ingång i PLCn. Utsignaler från PLCn skickas ofta till motorer, ventiler etc.
Inledningsvis när man utvecklade PLC hade man som mål att ersätta befintliga inbyggda elektriska kretsar baserade på relälogik (Galloway, Hancke u.å., s 4). Man ville ta fram en apparat som var enkel att arbeta med. Kravbilden var följande:

- Enkelhet både gällande programmering och gällande omprogrammering.
- Simpel underhållning och reparation.
- Lägre pris och mer kompakt än de befintliga systemen.
- Kapabel att arbeta inuti en fabrik och samtidigt kommunicera med ett centralt datainsamlingssystem.

Dagens PLC-maskiner har förmågan att hantera både digitala och analoga in- och utgångar (ibid). Det går också bra att implementera regulatorer av olika slag. Exempelvis går det utmärkt att realisera en PID-regulator.

Komponenter som tillsammans utgör grunden hos en PLC är någon form av strömförsörjning, processor, in- och utgångsmodul och avslutningsvis en kommunikationsmodul (ibid). När det gäller större och kraftfullare PLCs sitter delarna ofta separat vilket också gör dem utbytbara. Tack vare detta utformningssätt, med separata delar, förenklas underhåll och flexibilitet när det kommer till installation.

2.4 Codesys

I den industriella världen är Codesys ett av de program som lämpar sig till att programmera olika mjukvaru-PLCer (Codesys, u.å.). Programmet förhåller sig till standarden IEC 61131-3, som är en av de 5 olika sektionerna tillhörande IEC 61131 (Zurawski 2014, s. 2). Samtliga delar beskrivs enligt:

- “Generell information”.
- “Utrustnings- och provningskrav”.
- “Programmeringsspråk”.
- “Riktlinjer för användaren”.
- “Kommunikation”.

IEC 61131-3 är den del som riktar sig mot programmeringsdelen (ibid). Den består av 5 stycken olika språk som användaren kan välja att programera i.

Dessa språk är ladderdiagram, strukturerad text, funktionsblocksdiagram, instruktionslista och sekvensdiagram (Galloway, Hancke u.å., s 4). Med hjälp av standardbiblioteket för funktionsblock, IEC 61499, är det möjligt för användaren att skapa sitt PLC-program.

I Codesys strukturerar man sitt program i olika "POUs". En POU kan antingen vara en funktion, ett funktionsblock eller ett program (Heinz John and Tiegelkamp 2010, s. 21-22). Funktionsblocket skiljer sig från en funktion genom att den har ett minne och på så sätt returneras inte samma resultat varje gång för en specifik inparameter. På höjden av PLC-hierarkin för användarprogrammet läggs programdelen. Den har till uppgift att ansluta I/Os ifrån PLCn med de olika POUs:en.

2.5 HMI

Ett HMI gör det möjligt att förändra värden på olika variabler i ett PLC-system. (Encyclopedia, u.å.). HMI:et ger en grafisk överblick för hela systemet som visas upp på en pekskärm vars storlek oftast varierar mellan 4,3 och 21 tum (Beijer Electronics, u.å.). Konfigureringen sker mot ett PLC-system som tillsammans skapar en kompakt maskinvara (Beijer Electronics, u.å.).

2.6 I/O-enheter

För att det ska vara möjligt att styra de olika komponenterna i diverse aggregat använder man sig av I/O-enheter (Bolton 2009, s. 21-57). En enhet kan antingen vara en ingång eller en utgång.

Ingångsenheter tar emot fysiska signaler och konverterar dem till elektriska, resultatet skickas därefter vidare till exempelvis en PLC. Den elektriska signalen kan antingen vara digital eller analog. Ingångsenheter innefattar olika givare, exempelvis temperaturgivare eller tryckgivare.

Utgångsenheterna fungerar omvänt jämfört med ingångsenheter. Dessa tar emot en elektrisk signal och omvandlar den till en fysisk. Dessa innefattar bland annat reläer och kontaktorer. Ett relä i sig kan användas för att styra en switch.

PLC-systemet kommunicerar med dessa in- och utgångsenheter med hjälp av antingen fältbussar eller nätverk. Signalen kan som tidigare nämnt antingen vara digital eller analog.

Den analoga signalen är direkt proportionell mot det uppmätta fysiska värdet (Bolton 2009, s. 75-109). Signalen skickas oftast som en 0-10 V/4-20 mA ström. Oftast används 4-20 mA för att kunna använda 0 A till feldetektering. Strömsignalen måste bli omvandlad från analog till digital för att kunna tas emot som insignal till PLC-programmet. Detta görs med hjälp av en A/D-omvandlare. När man vill lägga en analog signal på utgången hos en PLC, sker en motsvarande omvanling, i detta fall med hjälp av en D/A-omvandlare. Konverteringens upplösning blir bättre parallellt med antalet bitar hos omvandlaren. Antalet möjliga utfall på den digitala signalen blir enligt formeln:

$$2^{N}-1$$, där N är antalet bitar.

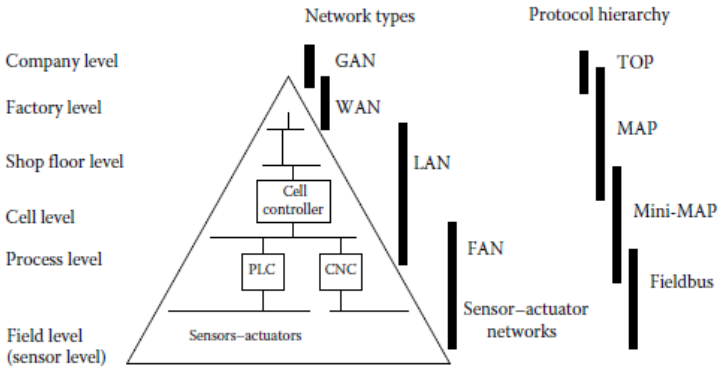
För exempelvis en 15-bitars omvandlare kan man skapa 32768 olika digitala värde, 0-32767. Om man vill jämföra detta med den analoga signalen, låt oss säga 0-10 V, kommer den maximala upplösningen mellan varje digital bit att motsvara 10/32767 V, vilket ungefär är 0,0003 V.

För fallet där en ingångsenhet genererar en digital utsignal, kan PLCn direkt avläsa den. Då till/från, 1/0, enbart är representativa.

2.7 Kommunikation

För att kunna sammanknyta automationssystemet finns det ett par olika metoder (Zurawski 2009, s 3 kap20). Som tidigare nämnt i vår inledning är fältbussen ett sätt som möjliggör en sammanknytning. Denna variant är en seriell kommunikation som bygger på en gemensam länk i form av två stycken ledningar mellan sändare och mottagare. Detta ersatte den dåvarande parallella kommunikationsmetoden som innebar ett enormt kablage.

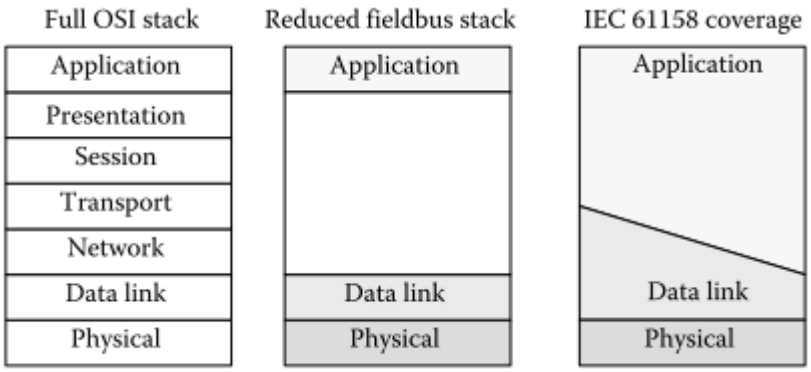
Den viktiga biten för fältbussutvecklingen har riktat sig mot automationspyramiden (Zurawski 2009, s 4 kap20). Detta är en filosofi, indelat i olika nivåer, som fokuserar på hur information inom processautomation flödar. Fältbusskommunikationen har framförallt innefattat sammankoppling mellan I/O-enheter med PLCn.



Hierarkin med nätverk för automation (Zurawski. R 2009, s.5).

Kommunikationen i ett system innehållande fältbussar byggs upp med en master och ett flertal slavar (PROFIBUS Nutzerorganisation e.V. PNO (2010), s.9). Ett annat namn för detta är centraliserat automationssystem (Zurawski 2009, s 28 kap20). Mastern beskrivs som den aktiva centralenheten som anropar sina slavar, som är de passiva enheterna (Modicon 1996). Kommunikationen sker via datautbyte mellan en master och diverse slavar. Detta genomförs genom att master sänder ut en signal, så kallad förfrågan eller ”Query”, som slaven/slavarna bekräftar med ett svar/respons (Modicon 1996, s4). Allt datautbyte utgår från mastern (Zurawski 2009, s 28 kap20). Den enda gången slaven har tillåtelse att agera “aktiv” är då denne fått en tillåtelse av mastern. Antingen kan det vara ett svar innehållande det data som mastern har efterfrågat, eller så kan det vara uppgifter som slaven blivit ombedd att göra.

Grunden till fältbussystemet kommer utifrån “OSI reference model” som är en struktur av diverse protokoll (Zurawski 2009, s 23 kap20)(Kurose, Ross (2010), s. 77-78). Det skiljer sig en viss del då bl.a. transport- och nätverkslagret inte finns med. Dessa lager är helt enkelt inbakade i applikationslagret för fältbussprotokollet.



OSI-modellen (Zurawski 2009, s. 23 kap 20).

Den andra kommunikationsmetoden är via nät. Denna skiljer sig en hel del ifrån fältbusskommunikation då denna inte är deterministisk (Acromag 2005, s. 6). Detta innebär att man inte kan förutse när paketen kommer fram eftersom bl.a. kollisioner och förseningar kan uppstå. Inom determinism är det förutsägbara datamängder som både skickas och tas emot vid specifika tidsintervall. Överföringshastigheten är en annan sak som skiljer dem åt. Den nya tekniken har gjort att överföringshastigheten inom nätverk har nått höga höjder. Då man inom den seriella kommunikationen överför med maximalt 115 kbps (Introduction to Modbus Serial and Modbus TCP, 2008). Den hastigheten är inte jämförbar med nät som istället kan nå upp i hastigheter omkring 1 Gbps. Inom många kommunikationsprotokoll har detta ingen betydelse då överföringshastigheten redan har nått sitt absoluta maximum.

Likaså bygger nätkommunikationen på OSI referensmodell. Man kan betrakta nätverksmodellen som 5 lager ifrån OSI för att sessions- och presentationslagret, lager 5 respektive 6, inkluderas i det översta lagret som heter applikationslagret.

2.8 Modbus

1979 utvecklades Modbus av Modicon i syfte att jobba ihop med industriella automationssystem och styrenheter tillhörande Modicon (Acromag 2005, s. 3). I dagsläget är Modbus en industriell standardmetod både för att transportera information om digitala/analoga in- och utgångar, och för att registrera data som skickats mellan exempelvis ett styrdon och en operatörspanel.

Modbusapparater använder “master/slav”-tekniken för att kommunicera med varandra (ibid). Endast den apparat som valts till master kan påbörja transaktioner, en slav väntar på förfrågningar från mastern. När en förfrågan anländer till en slav finns det två möjliga utfall. Vid det första fallet skickar mastern en förfrågan efter information om en specifik enhet, exempelvis kan mastern vara nyfiken på huruvida ventil V1 befinner sig i öppet eller stängt läge. Efterfrågad information skickas tillbaka till mastern.

Vid det andra fallet vill mastern att en eller flera slavar skall utföra något (ibid). Tänkbart är att mastern vill stänga ett spjäll.

En slav är kringutrustning av olika slag som behandlar information och skickar sina utsignaler till mastern med hjälp av Modbus (ibid). Ventiler och in- och utgångsomvandlare är exempel på slavenheter. En värddator är ett vanligt val som masterenhet.

Med hjälp av fyra stycken tabeller lagrar slavenheten information (Simply Modbus, u.å.). Två av tabellerna behandlar diskreta värden (coils), medan de andra två sköter analoga värden (registers). Det finns också två varianter på tabeller, två olika typer. Den ena är "Read-Write" och den andra är "Read-Only". De två varianterna finns både för coils och för registers. Var och en av tabellerna har 9999 värden. Kontaktorer och spolar representeras med en bit och blir tilldelade en adress mellan 0000-270E uttryckt i hexadecimal talform (0-9998 decimalt). Varje register tilldelas på samma sätt en adress inom samma intervall, 0000-270E.

Coil/Register Numbers	Data Addresses	Type	Table Name
1-9999	0000 to 270E	Read-Write	Discrete Output Coils
10001-19999	0000 to 270E	Read-Only	Discrete Input Contacts
30001-39999	0000 to 270E	Read-Only	Analog Input Registers
40001-49999	0000 to 270E	Read-Write	Analog Output Holding Registers

Figur med info om de fyra olika tabellerna (Simply Modbus, u.å.).

En förfrågan från mastern innehåller antingen en adress som tillhör en specifik slav, men det är också möjligt att skicka en broadcast (utsändning) till samtliga slavar i systemet (Acromag 2005, s. 3). Gensvar skickas alltid från slav till master med undantag av då det sistnämnda alternativet använts.

Masterns förfrågan upprättas, vid utnyttjandet av protokollet Modbus, genom att inkludera en rad olika saker (Pefhany 2000, s. 5). Adress till mottagarenheten (Slav ID) till att börja med, detta gäller inte vid en så kallad utsändning (broadcast). En funktionskod som specificerar vilken handling mottagaren skall utföra. Nödvändig data som skall skickas iväg och till sist ett fält som används till felhantering (error-checking field).

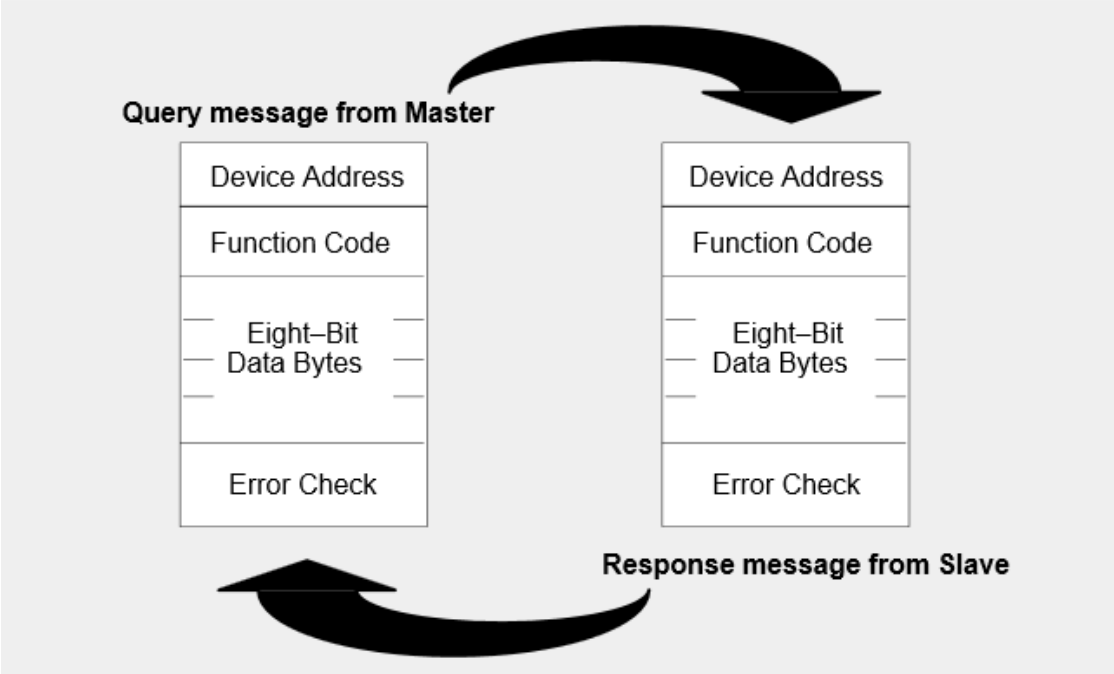
Adressen som varje slav blir tilldelad är unik och ligger inom intervallet 1-247 (Simply Modbus, u.å.). När ett meddelande skickas från master läggs adressen i början så att en slav snabbt kan välja att ignorera det om adressen i meddelandet inte stämmer överrens med den egna.

Utöver de standardmöjligheter som Modbus erbjuder finns det styrenheter från Modicon som, genom att använda inbyggda portar eller nätverksadaptrar, kan kommunicera via Modbus Plus (Modicon 1996, s. 13). Det är också möjligt att kommunicera via MAP om nätverksadaptrar väljs.

Modbus Plus och MAP skiljer sig från Modbus standard vad gäller kommunikationsteknik (ibid). Till skillnad från standarden, som använder

master/slav-tekniken, används istället “P2P”-tekniken. Detta innebär att samtliga enheter har samma rättigheter och kan uppträda både som master och som slav.

Trots att det är “P2P”-tekniken som används för nätverkskommunikation appliceras master/slav-principen nere på meddelandenivå (ibid). Om en enhet agerar som en master och skickar en förfrågan förväntas ett svar från den enhet som i detta fall jobbar som slav. Det fungerar likadant om en enhet tar emot en förfrågan, då tar den rollen som slav.



Cykel för Master/Slave Fråga/Svar (Modicon 1996, s. 14).

Funktionskoden i förfrågan underrättar den slav som är mottagare om vilken handling som skall utföras (Modicon 1996, s. 14). Giltiga funktionskoder är mellan 1-255 där koderna 128-255 är reserverade och används till felmeddelanden (Modbus-IDA 2006, s. 4). Den del som innehåller bytes med data ger ytterligare info som är nödvändig för att uträtta handlingarna i fråga (Modicon 1996, s. 14). Det går bra att lämna datafältet tomt. I så fall behöver inte slaven någon ytterligare information för att utföra handlingarna som efterfrågats (Modbus-IDA 2006, s. 4).
Exempelvis betyder funktionskoden 03 att slaven skall läsa och skicka tillbaka innehållet i ett register som svar (Modicon 1996, s. 14). För att slaven ska veta vilket register den ska börja läsa från, och hur många som ska läsas totalt, måste fältet innehållande data upplysa slaven om det. Fältet “Error Check”, se figur ovan, gör det möjligt för slaven att bekräfta meddelandets integritet. Att meddelandet inte har modifierats av någon obehörig etc.

Ett svar från en slav ser olika ut beroende på om något fel har inträffat eller inte (ibid). Ifall inget fel har skett kommer funktionskoden i svaren se ut som ett echo, oförändrat jämfört med det som fanns med i förfrågan. Datan ska då innehålla information, så som aktuell status eller värde, som slaven samlat in från utvalda register. Skulle det ske något fel ändrar slaven i funktionskoden för att upplysa mastern om att någonting gått snett. Beskrivning om vad det är för fel som inträffat läggs i fältet med data. Precis som en slav använder fältet "Error Check" för att verifiera att inget hänt med meddelandet längs med vägen, använder mastern fältet på samma vis.

2.8.1 Modbus ASCII/RTU

När en styrenhet installeras för användning i ett standard-Modbusnätverk finns det två överföringsmetoder att välja mellan vid kommunicering (Modicon 1996, s. 15-17). Antingen väljs ASCII eller RTU. Personen som konfigurerar styrenheten väljer en metod för kommunicering och andra parametrar, såsom överföringshastighet etc. För att ett Modbusnätverk ska fungera krävs att samtliga enheter är konfigurerade på samma vis. Det går t.ex. inte bra att blanda ASCII och RTU eller olika överföringshastigheter.

Valet mellan ASCII och RTU sker enbart vid användandet av standard-Modbusnätverk (ibid). Bitinnehållet i meddelandefälten som seriellt skickas definieras genom valet, likaså hur information ska packas in i meddelandefält samt avkodas.

2.8.1.1 Meddelande

Oavsett vilket läge, ASCII eller RTU, som valts placerar sändaren meddelande i ramar som har kända ändpunkter (ibid). Tack vare detta kan mottagaren utläsa var början av meddelandet är, titta på adressdelen för att snabbt ta reda på vilken enhet meddelandet är adresserad till och slutligen veta när meddelandet är slut.

2.8.1.2 Adressfält

I meddelandets adressfält återfinns två tecken om det är inställt på ASCII och 8 bitar om det istället är RTU (Modicon 1996, s. 19-23). Adresser som är giltiga för slavenheter ligger mellan 0-247 uttryckt på decimalform. Den första adressen, nollan, tilldelas aldrig till en ensam enhet utan används vid utsändningar (broadcast). Därför är möjliga adresser till slavenheter istället 1-247. Om mastern vill adressera meddelandet som ska skickas görs det enkelt genom att placera adressen till mottagarenheten i adressfältet. För att mastern ska veta att ett inkommande svar kommer från rätt slav placerar slaven i fråga sin egen adress i adressfältet när svaret ska skickas tillbaka.

2.8.1.3 Funktionsfält

I meddelandets fält för funktionskoden återfinns två tecken om det är inställt på ASCII och 8 bitar om det istället är RTU (ibid). De koder som finns är inom intervallet 1-255 uttryckt på decimalform. Endast somliga av koderna är applicerbara på samtliga styrenheter från Modicon. Vissa koder fungerar endast ihop med speciella modeller samtidigt som några kan vara reserverade för framtida behov.

Funktionsfältet i meddelandet som mastern skickar beskriver för slaven vad den ska utföra (ibid). En uppgift kan vara att läsa av till/från-läget hos en grupp reläer eller ingångar. En annan kan vara att läsa av data i ett eller flera register. Mastern kan också säga till en slav att skriva ett värde till antingen ett relä eller ett register.

Funktionskoden i funktionsfältet används för att indikera om ett meddelande har kommit fram felfritt eller inte (ibid). Om meddelandet är felfritt skickas samma funktionskod tillbaka som ett echo. I de fall där meddelandet på något sätt råkat ut för något skickas funktionskoden tillbaka något modifierad. Den mest signifikanta biten i koden sätts till en logisk 1, därefter placeras den modifierade koden i funktionsfältet.

För att ytterligare informera mastern vid fel läggs en unik kod in i datafältet (ibid). Koden beskriver vad det är för slags fel som har uppstått eller orsaken till felet.

2.8.1.4 Datafält

Datafältet konstrueras genom att använda två hexadecimala siffror inom intervallet 00-FF (ibid). Siffrorna kan byggas upp av två stycken ASCII-tecken eller ett RTU-tecken beroende på vilken metod som valts för överföring.

Innehållet i datafältet används av mastern för att förse slaven med information som krävs för att verkställa de efterfrågade uppgifterna (ibid). Information som kan tänkas finnas i datafältet är: adress till det register slaven ska börja läsa från, antalet register som ska läsas eller aktuellt antal databytes i fältet.

Ett litet exempel: Mastern vill att en slav ska läsa en grupp av register(ibid). Till att börja med läggs rätt funktionskod in i rätt fält, i detta fall är koden 03 (hexadecimalt). Datafältet ska specificera vilket register som ska läsas först och hur många som ska läsas totalt. Om mastern istället väljer funktionskoden 10 (hexadecimalt), skriva till en grupp register, ska datafältet informera om startregister, antal register, antal bytes med data i fältet samt den data som ska skrivas till registren.

Om inget fel inräffar läggs den efterfrågade data in i fältet och skickas till mastern (ibid). Vid fel läggs istället en unik kod in. Med hjälp av koden kan mastern bestämma vilka åtgärder som ska göras. Det finns tillfällen då datafältet utan problem kan lämnas tomt. Exempelvis om mastern frågar efter en logg med kommunikationshändelser (funktionskod 0B). Är så fallet behövs inte ytterligare information utöver funktionskoden.

2.8.1.5 Felsökningsmetoder

Det finns två olika metoder för att felsöka (ibid). Ett alternativ är att valfritt felsöka specifika tecken. Det andra alternativet innebär att hela meddelandet felsöks, ramvis. För att felsöka utvalda tecken tillämpas så kallad parity checking. Ramvis felsökning görs genom att använda antingen CRC (RTU) eller LRC (ASCII). Båda felsökningar genereras av mastern innan meddelandet skickas iväg, när meddelandet kommit fram kontrollerar mottagaren varje tecken och hela meddelandet.

Operatören eller användaren konfigurerar mastern till att vänta en viss tid innan förbindelsen avbryts (ibid). Tiden väljs så att en slav utan problem hinner svara på ett normalt sätt. I de fall då slaven upptäcker ett överföringsfel tas ingen åtgärd i bruk, inte heller skapas något svar till mastern. Eftersom inget svar skickas löper tiden ut, varpå mastern blir informerad om det och kan hantera problemet.

2.8.1.6 Parity checking

Vid konfigurering av en Modbusapparat bestäms huruvida parity checking ska användas eller ej (Automation.com, u.å.). Om det ska användas finns det två varianter, jämn respektive ojämn. Ifall parity checking väljs räknas antalet bitar som är 1 i varje ram. Beroende på vilken variant som valts väljs paritetsbiten till 1 eller 0 för att resultatet ska bli rätt.
Låt oss säga att en teckenram, i RTU-läge, innehåller följande 8 bitar: 1100 0011. Antal 1-bitar är 4, ett jämnt tal. Om ojämn parity checking är valt kommer paritetsbiten vara 1 för att få resultatet udda, annars sätts den till 0.

Paritetsbiten räknas ut och läggs in i varje dataram som tillhör meddelandet innan det skickas (ibid). I ASCII-läge är antalet bitar 7 stycken. Antalet är 8 i RTU-läget. Mottagaren räknar i sin tur ut antalet 1-bitar och sätter en flagga för fel om resultatet inte stämmer överrens med det som ska finnas i enheten. En svaghet med parity checking är att fel endast upptäcks om det är ett udda antal bitar som är fel. Om exempelvis 5 stycken 1-bitar överförs, ojämn parity checking, kommer mottagaren att reagera om den får antalet till 4 eftersom det

är jämnt. Däremot om två bitar försvinner kommer resultatet vara ojämnt, precis som det ska, och mottagaren noterar inget fel.

2.8.1.7 Modbus ASCII

Om enheter konfigureras i ASCII-läge innebär det att varje 8-bitars byte i meddelanden skickas som två ASCII-tecken (ibid). En av de främsta fördelarna med att jobba i detta läge är att tidsintervall på upp till 1 sekund tillåts att inträffa mellan tecken, utan att något fel uppstår.

ASCII Mode Byte Format	
Coding System	Hexadecimal of ASCII characters 0-9, and A-F. One hexadecimal character contained in each ASCII character (7 bits) of the message.
Bits Per Byte	1 start bit + 7 data bits, lsb sent first + 1bit for even/odd parity or no bit for no parity + 1 stop bit if parity is used or 2 stop bits with no parity.
Error Check Field	Longitudinal Redundancy Check (LRC)

Beskrivning för formatet för varje byte för ASCII(Automation.com, u.å.).

Meddelanden börjar med ett kolon och avslutas med ett par CRLF-tecken (Sena 2002, s. 4-5). I resterande fält tillåts enbart hexadecimala tecken 0-9 & A-F. I ASCII-läge övervakas nätverket konstant av enheterna för att upptäcka starttecknet (:). När det har mottagits avkodas det efterföljande fältet, adressfältet, för att konstatera vilken enhet som står som mottagaradress.

START	ADDRESS	FUNCTION	DATA	LRC CHECK	END
1 CHAR :	2 CHARS	2 CHARS	*n* CHARS	2 CHARS	2 CHARS CRLF

Meddelandets uppbyggnad (ASCII) - (Modicon 1996, s. 17).

I ASCII-läge innehåller fältet för felkontroll två stycken ASCII-tecken (ibid). De två tecknen väljs med hänsyn till resultatet av en uträkning som görs med hjälp av LRC-metoden. Den inledande delen, kolonet, och de avslutande CRLF-tecknen i meddelandet utelämnas vid uträkningen. Det är enbart de andra delarna som används. Fältet för felkontroll är placerat näst sist i meddelandet, precis innan CRLF-tecknen.

Mottagaren räknar på samma vis ut ett LRC-värde baserat på det mottagna meddelandet. Det mottagna LRC-värdet jämförs därefter med det uträknade, skiljer dem sig från varandra genereras ett fel.

När ett LRC-värde räknas ut görs det genom att successivt addera 8-bitars byte från meddelandet, minnessiffror (carry bits) kastas medan kolontecknet i början samt de avslutande CRLF-tecknen ignoreras (ibid). Resultatet tvåkompleteras därefter. Det 8-bitars värde som tagits fram överförs avslutningsvis till meddelandet. Biten av högst ordning läggs in först följt av den bit med lägst ordning. Exempelvis om ett 8-bitars LRC-värde är B6 (1011 0110) är B det tecken som har högst ordning och 6 har lägst.

2.8.1.8 RTU

Om enheter konfigureras i RTU-läge innebär det att varje 8-bitars byte i meddelanden byggs upp av två 4-bitars hexadecimala tecken (ibid). Den främsta fördelen med RTU är att teckendensiteten är högre jämfört med den i ASCII. Det innebär att man får en datagenomströmning som är bättre, trots att överföringshastigheten är densamma. Meddelanden som skickas måste överföras via en kontinuerlig ström.

RTU Mode Byte Format	
Coding System	8-bit binary, hexadecimal 0-9, A-F, two hexadecimal characters in each 8-bit field of the message.
Bits Per Byte	1 start bit + 8 data bits, lsb sent first + 1bit for even/odd parity or no bit for no parity + 1 stop bit if parity is used or 2 stop bits with no parity.
Error Check Field	Cyclical Redundancy Check (CRC)

Beskrivning för formatet för varje byte för RTU(Automation.com, u.å.).

Meddelanden börjar med ett tyst intervall som minst motsvarar sändningstiden för 3,5 tecken (Sena 2002, s. 4-5). Intervallet varierar beroende på vilken överföringshastighet som används. Nästkommande fält är adressfältet.

I de resterande fälten är det hexadecimala tecken, 0-9 och A-F, som är tillåtna (ibid). Varje enhet övervakar kontinuerligt nätverket, även vid de tysta intervallen. Så fort ett fält, efter intervallet, tagits emot avkodas det för att identifiera vilken enhet som står som adressat. Efter det sista tecknet i meddelandet återkommer ett likadant tyst intervall som fanns i början. Återigen motsvarar intervallet minst sändningstiden för 3,5 tecken. Först när intervallet är slut kan ett nytt meddelande börja.

Hela meddelandet måste överföras kontinuerligt, utan avbrott (ibid). Ifall det sker ett tyst intervall som minst motsvarar tiden det tar att sända 1,5 tecken slopar mottagaren det ofullständiga meddelandet och förväntar sig att nästkommande byte är adressfältet tillhörande ett nytt meddelande.

Skulle ett nytt meddelande börja innan tidsintervallet löpt ut antar mottagaren att det är en fortsättning från föregående meddelande (ibid). Sker detta kommer det resultera i ett fel. Felet uppstår i slutet, närmare bestämt i CRC-fältet.

START	ADDRESS	FUNCTION	DATA	CRC CHECK	END
T1–T2–T3–T4	8 BITS	8 BITS	*n* x 8 BITS	16 BITS	T1–T2–T3–T4

Meddelandets uppbyggnad (RTU) - (Modicon 1996, s. 18).

I RTU-läge innehåller fältet för felkontroll ett 16-bitars värde som delas upp i två 8-bitars byte (ibid). En uträkning på meddelandets innehåll görs genom att utnyttja CRC-metoden. Resultatet från uträkningen läggs sedan in i fältet, som i sin tur läggs in allra sist i meddelandet.

Avsändarenheten räknar ut och applicerar CRC-värdet sist i meddelandet. Det byte med lägst ordning placeras först följt av den med högst ordning. Mottagaren räknar ut ett CRC-värde på meddelandet som tagits emot och jämför resultatet med det medskickade CRC-värdet. Skiljer dem sig från varandra genereras ett fel.

Uträkningen inleds med att ett 16-bitars register skapas och fylls med ettor (ibid). Sedan adderas 8-bitars bytes från meddelandet successivt till det nuvarande innehållet i registret. Start-, stopp- samt paritetsbit ignoreras vid denna process. Varje 8-bitars tecken blir i sin tur utsatt för exklusiv eller med registret. Resultatet skiftas i riktning mot den bit som är minst signifikant och platsen för den bit som är mest signifikant får en nolla som värde. Slutligen extraheras och räknas LSB ut. Om LSB är lika med 1 utsätts den för exklusiv eller med ett förbestämt värde.

Processen ovan upprepas fram tills 8 skiftningar genomförts (ibid). Efter det åttonde skiftet utsätts nästkommande 8-bitars byte för exklusiv eller med det nuvarande innehållet i registret. Detta görs för resterande 8 skiftningar och det innehåll som till slut finns i registret är det slutgiltiga CRC-värdet.

2.8.2 Modbus TCP/IP

2.8.2.1 Inledning

Modbus TCP/IP är ett kommunikationssätt som använder sig av nätkommunikation, i detta fall via Ethernet (Acromag 2005, s. 4). TCP står för Transmission Control Protocol och IP för Internet Protocol. De två protokollen har olika uppgifter. TCP ser till att paket blir levererade på ett korrekt sätt medan IP tar fram en färdväg för paketen och adresserar ut dem. De bägge lagren tillsammans, med hjälp av Ethernet, transporterar Modbusapplikationen till de olika enheterna.

2.8.2.2 Hur förhåller sig Modbus TCP/IP till OSI-modellen?

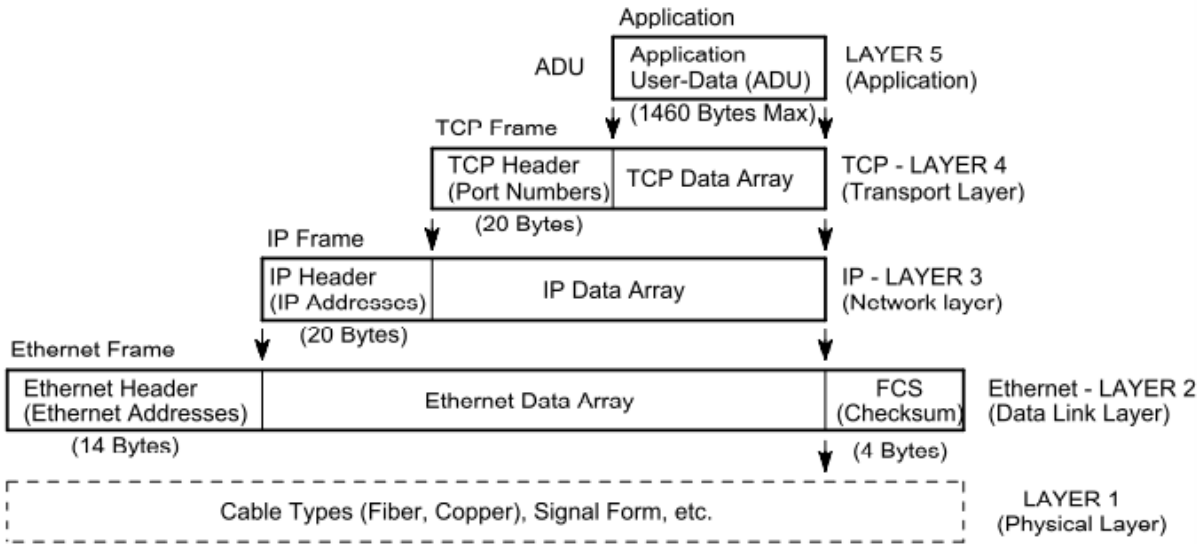
Tidigare i denna rapport har den så kallade OSI nätverksmodell tagits upp. Detta är grunden till hur nätverkskommunikation går till. Varje lager i modellen ifrån sändarsidan har till uppgift att erbjuda sina tjänster och kommunicera med mottagarsidan. Just vad gäller kommunikation med Modbus TCP/IP ser stackmodellen ut enligt nedan (Acromag 2005, s.10).

MODBUS TCP/IP COMMUNICATION STACK			
#	**MODEL**	**IMPORTANT PROTOCOLS**	**Reference**
7	Application	Modbus	
6	Presentation		
5	Session		
4	Transport	TCP	
3	Network	IP, ARP, RARP	
2	Data Link	Ethernet, CSMA/CD, MAC	IEEE 802.3 Ethernet
1	Physical	Ethernet Physical Layer	

Bilden beskriver vilka protokoll som används på respektive nivå. I detta avsnitt kommer vi att gå in grundligt och förklara de olika protokollen. (Acromag 2005, s.10).

Lager n erbjuder sina tjänster till lagret ovanför, n+1, och använder sig av tjänster som kommer underifrån, lager n-1. Detta görs genom att packa in ovanstående lager i en vektor och addera en header längst fram på paketet. Denna header innefattar de tjänster som just det särskilda lagret erbjuder (Acromag 2005, s.11).

När paketet anländer till mottagaren, kommer samma procedur att ske, men i detta fall spegelvänt. Varje lager i respektive ände sammarbetar med varandra. Detta betyder att mottagaren kontrollerar så att uppgifterna tillhörande varje lager överensstämmer.

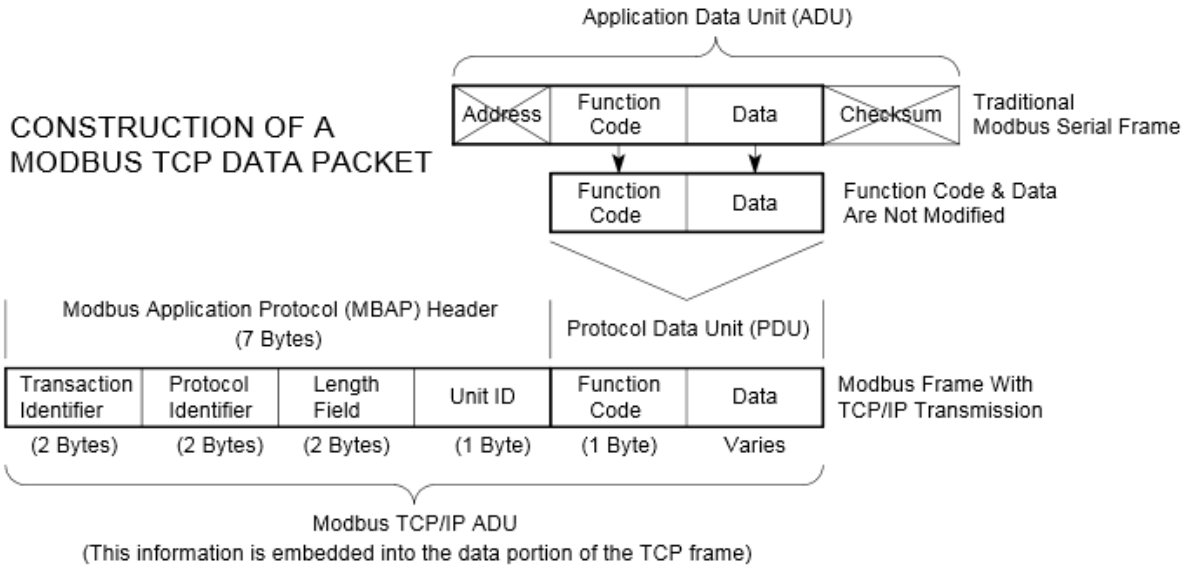


Beskrivning på hur olika lager appliceras på varandra (Acromag 2005, s.10).

2.8.2.3 Applikationsnivå

Den generella bilden för en Modbus TCP/IP ADU skiljer sig ifrån standarden för en Modbus ADU (Acromag 2005, s.4). Strukturen byggs istället upp med en MBAP header som läggs längst fram på den ordinära PDUn. Dessa tillsammans bildar Modbus TCP/IP ADUn. De bortplockade fälten, checksumma och adress, har ingen funktion i detta fall då dessa tjänster redan finns med i TCP och IP.

Bilden nedan ger en lättsam överblick för hur det nya applikationspaketet ser ut. Som ni ser innefattar MBAP headern 7 bytes. Dessa 7 bytes är indelade i 4 stycken olika fält: “Transaction Identifier, Protocol Identifier, Length Field och Unit ID”(ibid).



(Acromag 2005, s.4)

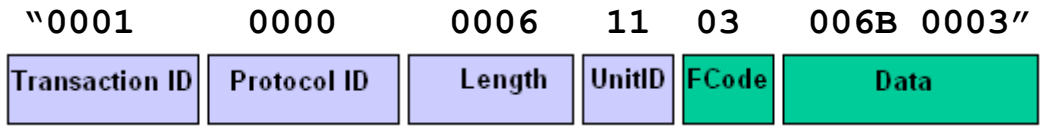
Transaction identifier innefattar 2 bytes av de 7 möjliga i MBAP headern (Acromag 2005, s. 5). Eftersom det är möjligt att skicka ett flertal paket utan att få ett svar inom nätverk, används denna för att identifiera vilket paket som är vilket under en och samma TCP-upprättelse.

Protocol Identifier använder sig också av 2 bytes (ibid). Den talar om vilket protokoll som applikationslagret använder sig av. För att tala om att applikationen Modbus används är detta fältet satt till en 0a.

Length är 2 bytes data som får sin bytesumma av (ibid):

Length = Unit ID + Function Code + Data

Unit Identifier innefattar den sista byten i MBAP (ibid). Den används enbart om det är ett flertal servrar inblandade. I annat fall sätts detta byte till det minimala värdet eller det maximala, vilket skrivs som 00 respektive FF uttryckt på hexadecimal form.
Ett meddelande kan exempelvis se ut så här (Simply modbus, u.å.):



Observera att meddelandet står skrivet i hexadecimal form för att underlätta läsandet.

Här följer en kortfattad beskrivning av ett meddelande.

- 0001: Första paketet som skickas under TCP förbindelsen.
- 0000: Applikationsprotokollet som används är modbus.
- 0006: längden av skickad data är 6 bitar.
- 11: Identifierar en specifik slav i anslutning till en seriell server. Slavens ID är, uttryckt på decimalform, i detta fall 16.
- 03: Registret som ska användas är "Read Analog Output Holding Register"
- 006B: Det första registret som slaven ska läsa, i detta fall register 107.
- 0003: Antal register som slaven ska läsa ifrån. Det första registret är 107, vilket följs av 108 och 109. Eftersom det första registret startar på 40001 kommer efterfrågade register att bli 40108 till 40110.

Applikationslagret använder sig av transportlagrets tjänst för att kontrollera vilken port paketet ska till. Specifikt för Modbus används portnummer 502.

2.8.2.4 Transportnivå

Protokollet har till uppgift att kapsla in ett applikationspaket och transportera detta till rätt portnummer (Acromag 2005, s. 29). TCP erbjuder tjänster som gör detta möjligt. Det är tjänster som innefattar flödeskontroll, sekvensnummer, bekräftelser och timers (Ross and Kurose (2010), s. 228).

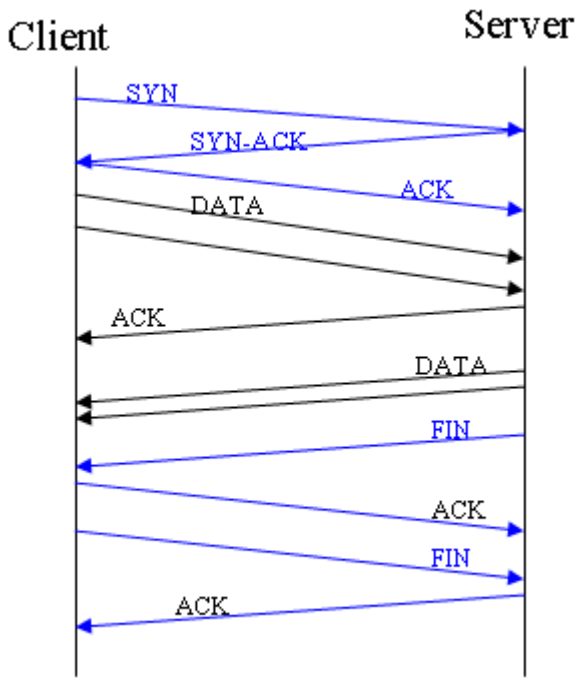
För att det ska gå bra att överföra information via TCP måste först en förbindelse upprättas. Så är fallet på grund av att protokollet kräver en anslutning mellan sändare och mottagare, till skillnad ifrån UDP där en upprättad förbindelse inte är något krav (Ross and Kurose (2010), s. 236). En TCP-förbindelse består av tre stycken olika avsnitt: anslutning, dataöverföring och nedkoppling.

Anslutningsfasen, även kallad three way handshake, sker genom 3 stycken olika steg (Ross and Kurose (2010), s. 290-291). Under det första steget skickar klienten/mastern ett SYN-segment till servern/slaven. Detta paket innehåller ingen information ifrån applikationslagret utan behandlar enbart en "förbindelseförfrågan". Servern/slaven svarar med ett ACKSYN-segment, som är ett svar på att den vill starta en förbindelse. Anslutningsfasen avslutas med ett ACK-segment som skickas från klienten/mastern till servern/slaven. Segmentet kan ses som en bekräftelse på att förbindelsen är upprättad.

När förbindelsen är etablerad kan datautbyte mellan klient/master och server/slav ske (Ross and Kurose (2010), s. 273-276, 282). För att förhindra att data går förlorad används sekvensnumrering, bekräftelse och tidtagning.

- Sekvensnumret kontrollerar ordningen på paketen.
- Bekräftelsen sker i form av ett ACK-meddelande ifrån servern/slaven som talar om för klienten/mastern att meddelandet har kommit fram och vilket meddelande som den nu förväntar sig.
- Tidtagningen sker med hjälp av en timer som sätts igång när ett meddelande skickas. Har tiden runnit ut och ACK'et inte har nått fram, skickar man om meddelandet igen.

Efter att data har utbytts finns det möjlighet till att stänga förbindelsen igen (Ross and Kurose (2010), s.293). Detta görs genom att den ena parten skickar ett så kallat FIN. Detta ACKas och därefter utför motparten en liknande procedur.



TCP-förbindelse (TCP Traffic Analyzer, u.å.).

2.8.2.5 Nätverksnivå

I nätverkslagret finner vi internetprotokollet, IP. Det visar sig vara ett bra komplement till TCP, som oftast nämns gemensamt. Då TCP står för pålitlig överföring mellan sändare och mottagare, svarar IP med rätt adressering och leverering av datapaket.

Varje IP-adress är bunden till en specifik enhet i nätverket, som samtidigt gör enheten unik. Detta skiljer sig från MAC-adressen som istället är bunden till hårdvaran för varje enhet (Acromag 2005, s. 35). För att binda samman MAC-adressen med IP-adressen, använder sig nätverkslagret ett protokoll som nämns under ARP och RARP. ARP används för att översätta en IP-adress till MAC-adress och RARP för att gå andra vägen, så kallad reverse.

2.8.2.6 Länknivå

Inom realtidskommunikation vill man ha en garanti på att paketen kommer fram och att de skickas under ett förutsägbart tidsintervall, ifall kommunikationen uppfyller dessa krav sägs det vara deterministisk (Acromag 2005, s6). Ethernet är i grunden inte ett bra val sett till det. Där tillåter man alla enheter att sända på en gemensamt inkopplad kanal så fort den är ledig. Det kan leda till kollisioner och därmed förseningar.

För att skicka paket och upptäcka eventuella kollisioner använder sig Ethernet av protokollet CSMA/CD (Acromag 2005, s6, 40-42) (Ross and Kurose 2010, s. 489-492). Protokollet utgår från att varje enhet lyssnar på kanalen för att direkt upptäcka om den är ledig. Om den är det skickar enheten meddelandet. Ibland kan det hända att en annan enhet vid exakt samma tillfälle väljer att skicka ett meddelande. Denna kommer precis som den andra enheten uppfatta att kanalen är ledig och därför också skicka iväg sitt meddelande. Detta kommer i sin tur leda till en kollision. De båda enheterna kommer då att slumpa fram en tid och vänta tills den löpt ut innan ett nytt försök att skicka meddelandet görs.

I de flesta systemen vill man ha en transmissionstid som är mindre än 100ms, detta gör att CSMA/CD inte löser problemet med eventuella kollisioner som ibland uppstår vid användning av standardvarianten för Ethernet (Ross and Kurose 2010, s. 512-519) (Acromag 2005, s.7, 40-42). Lösningen för att göra Ethernet mer deterministiskt är att använda “fast Ethernet”. En variant där switchar ansluts direkt till specifika enheter. Varje enhet kommer därmed att ha sin egen port till en switch. För att veta hur switchen ska skicka vidare data, har varje enhet tilldelats en specifik MAC-adress. Varje enhets MAC-adress är lagrad i den switch den är ansluten till. På så sätt vet switchen vilken utgångsport som den ska skicka ut data på när den väl får det.

Tack vare att varje enhet har en individuell förbindelse till en switch, så kallad stjärntopologi, tar man bort risken för att en kollision ska inträffa. Detta gör att man får en ökad överföringshastighet, en så kallad full-duplex, vilket innebär att man skulle kunna skicka respektive ta emot data med hastigheter på hela 10-100 Mbps.

Tyvärr är inte detta hela sanningen. Switcharna har inte möjlighet att filtrera inkommande trafik ifrån andra nätverk. Man löser detta med separation av diverse nätverk för att genom detta öka determinismen. Bland annat kan man separera med hjälp av bryggor som filtrerar “broadcast”-information som anses vara onödig för det andra nätverket.

3 Metod

För att få kunskap inom områden genomförde vi litteraturstudier. Vi studerade främst PDF-dokument och e-böcker. Studien hjälpte oss med att komma in i programmiljöer och gav oss djupare förståelse för hur kommunikationen fungerar. Det resulterade i att vi kunde skriva en teoretisk del som behandlade olika begrepp som på något sätt rörde vårt arbete.

PLC-systemet byggdes främst upp genom programmering med ladderdiagram. Under programmeringens gång gjordes observationer på ett givet driftkort. Observationer i driftkortet påverkade även uppbyggnaden av vårt HMI, med anledning av att PLC-programmet och HMI:et är tänkt att arbeta tillsammans.

Simulering användes för att konstatera huruvida programmen fungera användar- och mjukvarumässigt. Användaren vill kunna navigera runt i gränssnittet på ett så enkelt sätt som möjligt. Det är också viktigt med en väldisponerad bild över systemet.

Utifrån mjukvaran verifierades att:

- Kommunikationen var felfri så att signaler mellan HMI och simulator utbyttes korrekt.
- Programmen var funktionsdugliga
- Eventuella fel uppstått.

4 Utförande

4.1 Codesys

Det första vi gjorde var att ladda hem programmet Codesys till våra datorer, därefter gick vi in i en inlärningsperiod under 10-14 dagar. Under den tiden lärde vi känna miljön i programmet för att på så sätt underlätta arbetet längre fram. Detta gjordes främst genom att titta på och själva göra grundliga exempel på projekt.

4.1.1 Att skapa ett funktionsblock

Efter en tids inlärning kunde vi se fram emot nästa uppgift vilken innebar att, utefter ett givet driftkort, implementera en rad olika funktionsblock. Dessa block kom till användning när vi i nästa steg programmerade PLC-systemet för hela anläggningen.
Som vi skrev i teoridelen använder Codesys sig av PLC-standarden IEC 61131-3. Det är tack vare den som det är möjligt att skapa egna funktionsblock (Codesys u.å.). Det första som man gör är att skapa ett nytt POU av typen FB (funktionsblock). Ett funktionsblock är ett objekt som representerar en viss typ, t.ex. ett spjäll eller en motor. När man skapar egna funktionsblock får man möjligheten att bygga upp dem på precis det sätt man vill. In- och utgångar bestämmer man själv, likaså implementeras olika händelseförlopp.

Varje funktionsblock har ingångar, utgångar och privata variabler som deklareras internt inne i den POU som blocket tillhör. När man deklarerar variabler bestäms samtidigt vilken typ de ska ha. Återigen görs det utefter IEC-standarden 61131-3. Utöver interna variabler finns det också en global variabellista. Variabler som deklareras i den kan nås från vilken del av programmet som helst.

Ingångar gör det möjligt att påverka och bestämma hur utgångar ska bete sig. Beroende på vilket värde som en ingång ges sker förutbestämda saker inuti blocket varefter en eller flera utgångar påverkas. Det är helt enkelt de variabler som man kan ändra ifrån utsidan av blocket. De privata/interna variablerna är "privata" för funktionsblocket i fråga och går därför inte att nås utifrån. Dessa sätts av ingångar och fungerar ofta som hjälpvariabler för att man sedan ska kunna ge utgångar olika värde.

I vårt PLC-program implementerade vi sammanlagt 8 stycken funktionsblock för generella komponenter. Det var följande block: öppna/stäng-spjäll,

reglerbart spjäll, frekvensmotor, start/stopp-motor, skalningsblock, temperaturblock, tryckblock och block för elluftvärmare. Nedan följer en kortare beskrivning av varje funktionsblock och deras in- respektive utgångar.

4.1.2 De Olika funktionsblocken

4.1.2.1 Spjäll

Detta block representerar ett spjälls funktioner. För att en utgång ska sättas krävs det att ett visst antal ingångar går höga. Spjällets ändläge indikeras med hjälp av givare. Den visar om spjället antingen är stängt eller öppet.

Ingångar:

- Enable (nödstopp): Sätter spjällets funktion till ett dynamiskt läge, för automatisk drift. Fungerar som strömbrytare för ett bestämt aggregat. (typ: BOOL)
- Manual: Används för att ställa in spjället i manuellt läge. (typ: BOOL)
- Auto: Används för att ställa in spjället i automatiskt läge. (typ: BOOL)
- ManOpen: Används för att öppna spjället manuellt. Kräver att spjället är satt i manuellt läge först. (typ: BOOL)
- ManClosing: Används för att stänga spjället manuellt. Kräver att spjället är satt i manuellt läge först. (typ: BOOL)
- Opened: Ändlägesindikering för spjällets öppna läge. Om spjället har lämnat sitt ändläge blir ingången satt till false. (typ: BOOL)
- Closed: Ändlägesindikering för spjällets stängda läge. Om spjället har lämnat sitt ändläge blir ingången satt till false. (typ: BOOL)
- Fire: Förhindrar användning av aggregatet. Går hög ifall rök eller brand detekterats.
  (typ: BOOL)
- Condition: Villkor som sätts utifrån. Bestämmer när spjället skall stängas respektive öppnas. Fungerar enbart när spjället arbetar i automatiskt läge. (typ: BOOL)
- Timer: Sätter den maximala öppningstiden för spjället till det värde som ingången sätts till. Överstigs denna tid kommer utgången “Error” att gå hög. (typ: INT)
- Reset error closing: Återställer flaggan för fel vid stängning om ingången är hög. (typ: BOOL)
- Reset error opening: Återställer flaggan för fel vid öppning om ingången är hög. (typ: BOOL)

Utgångar:

- Status: En integersignal som talar om för gränssnittet vilket tillstånd som spjället befinner sig i. De olika tillstånden är: manuellt läge, automatiskt läge, brand samt fel. (typ: INT)
- Till: Om signalen är satt till true öppnas spjället. Är signalen istället satt till false stänger spjället. (typ: BOOL)
- Error_opening: Går hög om det tagit för lång tid att öppna spjället. (typ: BOOL)
- Error_closing: Går hög om det tagit för lång tid att stänga spjället. (typ: BOOL)

4.1.2.2 Reglerbart spjäll

Detta block representerar inte ett ensamt spjäll utan två. Dessa spjäll jobbar tillsammans. Exempelvis om det ena spjället är 70 % öppet kommer det andra spjället istället att vara 70 % stängt osv. Detta fungerar genom att man kastar om faserna tillhörande det ena spjället. Resultatet kan jämföras med att köra en motor i motsatt riktning. Det ena spjället representerar insläppet för uteluft och det andra insläppet av återluft.

Utöver de nämnda egenskaperna arbetar även spjällen utefter positionerade lägen, d.v.s. spjällen reglerar sitt genomflöde med en viss position. Denna position är inte låst till öppet/stängt läge. Positionen som beskriver läget anges i procent. Regleringen av positionen sker med hjälp av en PID-regulator.

Ingångar:

- Enable (nödstopp): Sätter spjällets funktion till ett dynamiskt läge, för automatisk funktion. Fungerar som strömbrytare för ett visst aggregat. (typ: BOOL)
- Manual: Används för att ställa in spjället i manuellt läge. (typ: BOOL)
- Auto: Används för att ställa in spjället i automatiskt läge. (typ: BOOL)
- ManOpen: Används för att öppna spjället manuellt. Kräver att spjället är satt i manuellt läge först. (typ: BOOL)
- ManClose: Används för att stänga spjället manuellt. Kräver att spjället är satt i manuellt läge först. (typ: BOOL)
- Fire: Förhindrar användning av aggregatet. Går hög ifall rök eller brand detekterats.
  (typ: BOOL)

- Pos: Bestämmer vilket läge spjällen ska inta. Anges i procent och regleringen sköts med hjälp av en PID-regulator. (typ: INT)
- Condition open: Villkor som bestämmer när spjället skall öppnas. Fungerar i automatiskt läge. (typ: BOOL)
- Condition close: Villkor som bestämmer när spjället skall stängas. Fungerar i automatiskt läge. (typ: BOOL)

Utgångar:

- Status: En integersignal som talar om för gränssnittet vilket tillstånd som spjället befinner sig i. De olika tillstånden är: manuellt läge, automatiskt läge, brand samt fel. (typ: INT)
- SetPos: Ärvärdet för spjällets aktuella position. (typ: INT)
- Active: Indikerar om spjället är i aktivt läge. Motsvarar "Till"-utgången för öppna/stäng-spjället. (typ: BOOL)

4.1.2.3 Start/Stopp-motor

Motorns uppgift är att driva en fläkt som forcerar luften framåt genom systemet. Principen för en start- och stoppmotor är densamma som för ett spjäll. Antingen körs motorn på fullt varvtal eller så körs den inte alls. Den är med andra ord ej inställbar för andra lägen än av/på.

Ingångar:

- Enable (nödstopp): Sätter motorns funktion till ett dynamiskt läge, för automatisk funktion. Fungerar som strömbrytare för ett visst aggregat. (typ: BOOL)
- Disconnect: Säkerhetsbrytare. (typ: BOOL)
- Manual: Används för att ställa in motorn i manuellt läge. (typ: BOOL)
- Auto: Används för att ställa in motorn i automatiskt läge. (typ: BOOL)
- ManStart: Används för att starta motorn manuellt. Kräver att motorn är satt i manuellt läge först. (typ: BOOL)
- ManStop: Används för att stoppa motorn manuellt. Kräver att motorn är satt i manuellt läge först. (typ: BOOL)
- Fire: Förhindrar användning av aggregatet. Går hög ifall rök eller brand detekterats.
  (typ: BOOL)
- Condition start: Villkor som bestämmer när motorn skall startas. Fungerar i automatiskt läge. (typ: BOOL)

- Condition stop: Villkor som bestämmer när motorn skall stoppas. Fungerar i automatiskt läge. (typ: BOOL)
- Driftindikering: När insignalen är hög betyder det att motorn är i drift. (typ: BOOL)
- Timer: Sätter den maximala start/stopptiden för motorn till det värde som ingången sätts till. Tiden som det får ta att starta/stoppa motorn. Timern går igång när man har valt att starta eller stoppa motorn. Om driftindikeringsreläet inte är draget när tiden löpt ut kommer ett fel att sättas. Vid stopp ska driftindikeringen inte vara draget när tiden löpt ut. (typ: INT)
- Reset error starting: Återställer flaggan för fel vid uppstart om ingången är hög. (typ: BOOL)
- Reset error stopping: Återställer flaggan för fel vid stängning om ingången är hög. (typ: BOOL)

Utgångar:

- Motor: Lägesindikering för om motorn är till eller från. (typ BOOL)
- Status: En integersignal som talar om för gränssnittet vilket tillstånd som motorn befinner sig i. De olika tillstånden är: manuellt läge, automatiskt läge, brand samt fel. (typ: INT)
- Error_starting: Går hög om det tagit för lång tid att starta motorn. (typ: BOOL)
- Error_stopping: Går hög om det tagit för lång tid att stoppa motorn. (typ: BOOL)

4.1.2.4 Frekvensmotor

Frekvensmotorn skiljer sig ifrån start/stoppmotorn på så vis att den har ett reglerbart varvtal. Regleringen sker med hjälp av en PID-regulator. Den talar om för motorn hur många procent som motorn skall köra på. Procenten i sig motsvarar en frekvens som maximalt ligger på 50 Hz, vilket motsvarar 100 %. Frekvensmotorn har till uppgift att driva en fläkt.

Ingångar:

- Enable (nödstopp): Sätter motorns funktion till ett dynamiskt läge, för automatisk funktion. Fungerar som strömbrytare för ett visst aggregat. (typ: BOOL)
- Manual: Används för att ställa in motorn i manuellt läge. (typ: BOOL)

- Auto: Används för att ställa in motorn i automatiskt läge. (typ: BOOL)
- ManStart: Startar motorn på en bestämd effekt, som man via gränssnittet själv får välja. Ingen reglering av varvtalet sker. Kräver att motorn är satt i manuellt läge först. (typ: BOOL)
- ManStop: Stoppar motorn. Kräver att motorn är satt i manuellt läge först.
  (typ: BOOL)
- Fire: Förhindrar användning av aggregatet. Går hög ifall rök eller brand detekterats. (typ: BOOL)
- Disconnect: Säkerhetsbrytare. (typ: BOOL)
- Driftindikering: När insignalen är hög betyder det att motorn är i drift. (typ: BOOL)
- Condition start: Villkor som bestämmer när motorn skall starta. Fungerar i automatiskt läge. (typ: BOOL)
- Condition stop: Villkor som bestämmer när motorn skall stoppa. Fungerar i automatiskt läge. (typ: BOOL)
- Speed: Motorns effekt sätts till det värde som når ingången. I automatisk drift beror regleringen av värdet på trycket i ventilationstrumman. Regleringen sker med hjälp av en PID-regulator. (typ: INT)
- Reset error starting: Återställer flaggan för fel vid uppstart om ingången är hög. (typ: BOOL)
- Reset error stopping: Återställer flaggan för fel vid stängning om ingången är hög. (typ: BOOL)

Utgångar:

- Motor: Informerar om motorns aktuella hastighet. (typ: INT)
- Status: En integersignal som talar om för gränssnittet vilket tillstånd som motorn befinner sig i. De olika tillstånden är: manuellt läge, automatiskt läge, brand samt fel. (typ: INT)
- Active: Indikerar om motorn är i aktivt läge. Motsvarar “Till”-indikering för start/stopp motorn. (typ BOOL)
- Error_starting: Går hög om det tagit för lång tid att starta motorn. (typ: BOOL)
- Error_stopping: Går hög om det tagit för lång tid att stoppa motorn. (typ: BOOL)

4.1.2.5 Skalningsblock

Har till uppgift att skala om den analoga insignalen till en 0-100 skala. Anledningen till att vi vill använda den skalan är att frekvens och vinkel redan anges i procent. Den analoga signalen som kommer från I/O-enheten motsvarar 16 bitar. En av bitarna är reserverad till felhantering. De 15 tillgängliga bitarna utgör tillsammans 32767 olika kombinationer, skrivs som 7FFF på hexadecimal form. Mer information om detta finns i det teoretiska kapitlet, under rubriken I/O-enheter.

Skalningen sker linjärt med nedanstående formel (Matteguiden (u.å.)):

Gain = (Scaledmax - Scaledmin) / (Inputmax - Inputmin)
Offset = Scaledmin - (Gain * Inputmin)
ScaledValue = Gain * ActualInputValue + offset

Formeln har sitt ursprung i formeln för linjens ekvation (ibid):

y = kx + m

Ingångar:

- ScaledMax: Det maximala värdet som resultatet ska kunna anta. I vårt fall 100, (100 %). (typ: REAL)
- ScaledMin : Det minsta värdet som resultatet ska kunna anta. I vårt fall 0, (0 %). (typ: REAL)
- InputMax: Det värde som ska motsvara maximal insignal, i vårt fall 2^15 - 1 (32767). (typ: REAL)
- InputMin: Det värde som ska motsvara minsta möjliga insignal, med andra ord 0. (typ: REAL)
- ActualInputValue: Den analoga signal som kommer från exempelvis en givare. Den skalas vidare om i blocket. (typ: REAL)

Utgångar:

- ScaledValue: Det nya, omskalade värdet. Ligger mellan 0-100. (typ: REAL)

De olika givarna skickar analoga signaler, 4-20 mA/0-10V, använder på 4-20 mA signalen för att vi kan genom detta upptäcka eventuella fel som inträffat.

Denna analoga signal skalas om till 0-100 för att underlätta beräkningar i funktionsblock.

4.1.2.6 Temperaturblock

Jämför temperaturen i ett visst medium med hänsyn till en förutbestämd gräns. Som utsignal skickas ett värde som berättar om den aktuella temperaturen är för hög, låg etc.

Ingångar:

- ScaledValue: Utgången från skalningsblocket kopplas till denna ingång. (typ: REAL)
- MaxValue: Den maximala temperatur som får lov att uppnås i ett rum eller en specifik miljö. (typ: REAL)
- MaxMaxValue: Den kritiskt maximala temperatur som får lov att uppnås i ett rum eller en specifik miljö. (typ: REAL)
- MinValue: Den minimala temperatur som får lov att uppnås i ett rum eller en specifik miljö. (typ: REAL)
- MinMinValue: Den kritiskt minimala temperatur som får lov att uppnås i ett rum eller en specifik miljö. (typ: REAL)

Utgångar:

- Status: En integersignal som talar om för gränssnittet vilket tillstånd som temperaturen befinner sig i. Exempelvis: hög, kritiskt hög, låg, kritiskt låg.
  (typ: INT)
- TempHigh: Indikerar att temperaturen är hög, sett till det inställda värdet, genom att gå hög. (typ: BOOL)
- TempHigHigh: Indikerar att temperaturen är kritiskt hög, sett till det inställda värdet, genom att gå hög. (typ: BOOL)
- TempLow: Indikerar att temperaturen är låg, sett till det inställda värdet, genom att gå hög. (typ: BOOL)
- TempLowLow: Indikerar att temperaturen är kritiskt låg, sett till det inställda värdet, genom att gå hög. (typ: BOOL)

4.1.2.7 Tryckblock

Jämför trycket i ett visst medium med hänsyn till en förutbestämd gräns. Som utsignal skickas ett värde som berättar om det aktuella trycket är för hög, låg etc. Fungerar annars på samma vis som vårt temperatursblock.

Ingångar:

- ScaledValue: Utgången från skalningsblocket kopplas till denna ingång. (typ: REAL)
- MaxValue: Det maximala tryck som får lov att uppnås i ett rum eller en specifik miljö. (typ: REAL)
- MaxMaxValue: Det kritiskt maximala tryck som får lov att uppnås i ett rum eller en specifik miljö. (typ: REAL)
- MinValue: Det minimala tryck som får lov att uppnås i ett rum eller en specifik miljö. (typ: REAL)
- MinMinValue: Det kritiskt minimala tryck som får lov att uppnås i ett rum eller en specifik miljö. (typ: REAL)

Utgångar:

- Status: En integersignal som talar om för gränssnittet vilket tillstånd som trycket befinner sig i. Exempelvis: hög, kritiskt hög, låg, kritiskt låg.
  (typ: INT)
- PressureHigh: Indikerar att trycket är högt, sett till det inställda värdet, genom att gå hög. (typ: BOOL)
- PressureHigHigh: Indikerar att trycket är kritiskt högt, sett till det inställda värdet, genom att gå hög. (typ: BOOL)
- PressureLow: Indikerar att trycket är lågt, sett till det inställda värdet, genom att gå hög. (typ: BOOL)
- PressureLowLow: Indikerar att trycket är kritiskt lågt, sett till det inställda värdet, genom att gå hög. (typ: BOOL)

4.1.2.8 Elluftvärmare

Finns i två olika varianter. Den ena varianten arbetar i tre olika lägen. De tre lägena motsvaras av två effektnivåer som värmaren kan jobba efter samt ett läge då värmaren är från. När värmaren väl är påslagen kan man välja mellan att antingen köra den på 100 % av full effekt (Steg 2), eller 50 % (Steg 1).

Den andra varianten arbetar enbart i två lägen, till/från. Är värmaren till jobbar den på full effekt. Blocken för de två varianterna är uppbyggda på samma sätt. För till/från-värmaren används som tidigare nämnt enbart det ena läget. Den andra utgången väljs helt enkelt bort när man programmerar. Samtidigt försvinner även ManStart2, TempLowLow samt Steg2.

Ingångar:

- Enable (nödstopp): Sätter elvärmarens funktion till ett dynamiskt läge, för automatisk funktion. (typ: BOOL)
- Auto: Används för att ställa in värmaren i automatiskt läge. (typ: BOOL)
- Manual: Används för att ställa in värmaren i automatiskt läge. (typ: BOOL)
- ManStop: Stoppa motorn manuellt, fungerar endast om den manuella indikeringen är satt. (typ BOOL)
- ManStart1: Sätter värmaren till att köra i steg 1. (typ BOOL)
- ManStart2: Sätter värmaren till att köra i steg 2. (typ BOOL)
- TempLow: Är denna ingång hög startar värmaren på 50 %, steg 1 med andra ord. (typ: BOOL)
- TempLowLow: Är denna ingång hög startar värmaren på 100 %, steg 2 med andra ord. Denna ingång utnyttjas enbart när värmaren med 2 steg används.
  (typ: BOOL)
- Fire: Förhindrar användning av aggregatet. Går hög ifall rök eller brand detekterats. (typ: BOOL)
- Disconnect: Säkerhetsbrytare. (typ: BOOL)

Utgångar:

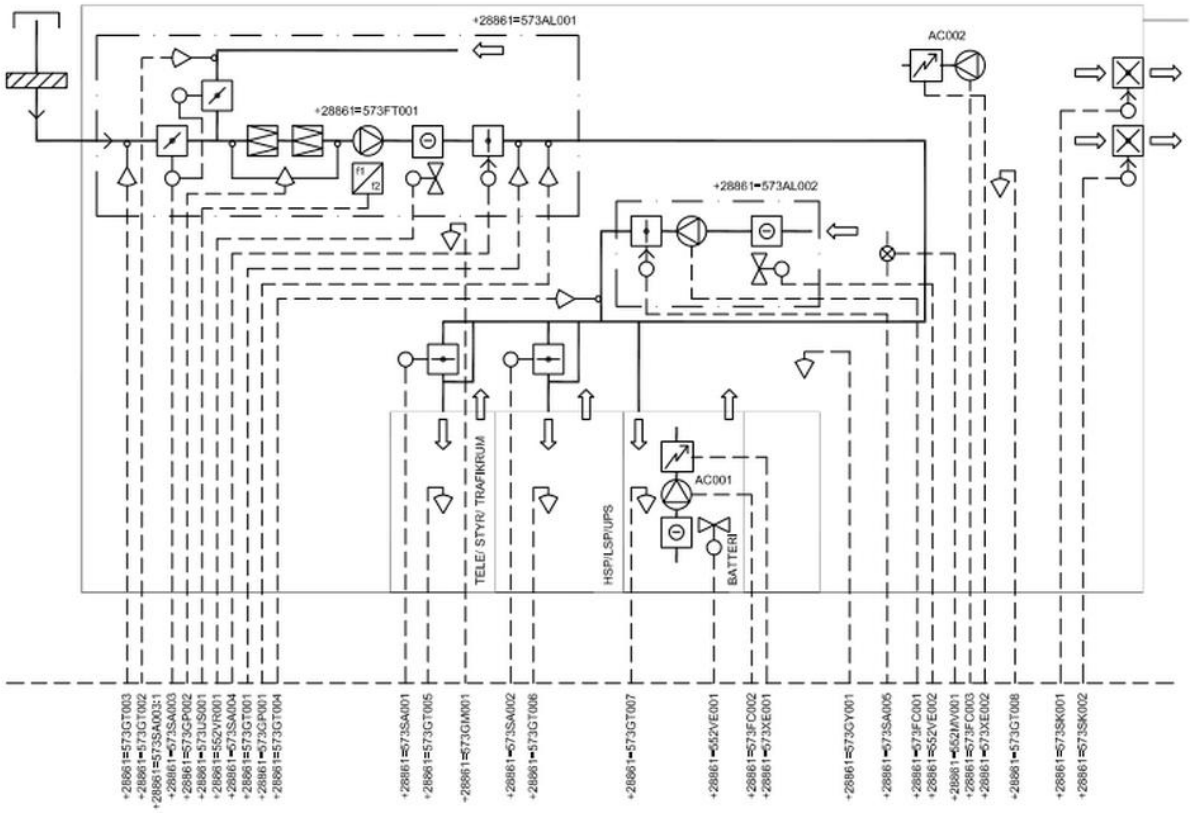
- Steg1: Går hög när värmaren ska köras på 50 % (typ: BOOL)
- Steg2: Går hög när värmaren ska köras på 100 % (typ: BOOL)
- Status: En integersignal som talar om för gränssnittet vilket tillstånd som värmaren befinner sig i. De olika tillstånden är: steg 1, steg 2, från eller brand. (typ: INT)

4.1.3 Uppbyggnad av PLC-program

Med blocken färdigimplementerade kunde vi sätta igång med nästa fas, den fas som skulle resultera i ett PLC-system för anläggningen. Driftkortet vi

tillhandahöll låg till grund för hur skapandet av systemet gick till. Ett driftkort är mer eller mindre en beskrivning på hur samtliga komponenter i systemet skall fungera och bete sig. Funktionsbeskrivning är ett annat ord för driftkort, det senare är vanligt förekommande inom fastighetsautomation.

Saker som hur en specifik fläkt ska bete sig vid automatisk drift, vilka givare den är kopplad till etc. finner man i driftkortet. I bilaga 2 finns en beskrivning i textformat som tillhör det driftkort som vi blev tilldelade av vår handledare.



Figur - Överblick av systemet, del av driftkortet.

I vårt fall beskrivs ett ventilationssystem där tryck, temperatur och genomströmning av luft ska regleras baserat på inställda värden. Anläggningen drivs med hjälp av två aggregat, AL001 och AL002. Det sistnämnda fungerar som redundans och aktiveras ifall huvudaggregatet, AL001, skulle sluta fungera eller om det inte klarar av att upprätthålla en bestämd temperatur.

Utöver de komponenter vi skapat funktionsblock för omfattas anläggningen av ett x-antal givare som tillsammans kontrollerar temperatur, tryck samt brand.

Genom att utnyttja de funktionsblock som vi tidigare skapat samt med hänsyn till de riktlinjer som driftkortet innehöll byggde vi upp ett PLC-system. Detta

gjordes främst med hjälp av det grafiska programmeringsspråket ladderdiagram men vi kom även i kontakt med funktionsblocks- och sekvensdiagram. Samtliga programspråk härstammar från IEC-standarden 61131-3. Systemet/Programmet delades upp för att det skulle bli enklare att forma från grunden. Här tog vi hjälp av de olika POU:s som Codesys erbjuder. Vi använde enbart typerna program och funktionsblock.

Ett POU, av typen program, gjordes för varje aggregat/delprogram. Totalt blev det 7 delprogram och dem listas nedan:

- AC001
- AC002
- AL001
- AL002
- Brandspjäll
- Tele/styr/trafikrum
- HSP/LSP/UPS

När man skapar ett program görs det med hjälp av ett flertal nätverk. Funktionsblocken som vi implementerade tidigare lades in i olika nätverk. I bilaga 4 finner ni skärmdumpar på hela AL001, nätverk för nätverk.

Utöver programmen ovan skapade vi också två POU:s för motionering av delar i systemet. Den ena motioneringen är för brandspjällen och den andra är för aggregatet AL002. Brandspjällen motioneras var 48e timme och AL002 en gång i månaden. Till skillnad från när vi skapade programmen valdes här sekvensdiagram. POU:s för respektive är listade nedan:

- Motion_brand
- Motion_AL002

I bilaga 6 finns skärmdumpar på hur sekvensdiagrammet ser ut för Motion_AL002.

4.1.4 Simulering av PLC-program

För att kontrollera att vårt program fungerade kompilerade vi och körde det mot Codesys virtuella PLC. Under simuleringen var det möjligt att “tvinga” globala variabler till ett värde för att på så sätt upptäcka eventuella fel i programmet.

4.1.5 I/O-lista

Efter att systemet byggts upp och kompilering inte resulterade i några fel övergick vi till att tolka en given I/O-lista som beskrivs i bilaga 3. Listan gav en beskrivning för hur vi skulle knyta ihop PLC-systemets fysiska in- och utgångar mot I/O-enheterna. För att kunna testa funktionen krävs en uppkoppling mellan PLC och I/O. Eftersom vi inte hade någon PLC till förfogan valde vi att gå vidare.

4.2 Framtagning av HMI

4.2.1 Översikt

Det finns en hel del program man kan ta hjälp av för att forma ett HMI. Vi gjorde det med hjälp av iX Developer som är framtaget av Beijer Electronics. Precis som när vi skulle börja med PLC-delen krävdes det att vi lärde oss den programmiljö vi skulle ta hjälp av, i detta fall iX. Genom att kolla på en introduktionsguide och göra exempel lärde vi oss de grundliga delarna av programmet.

När man formar ett HMI, ett gränssnitt, åt ett SCADA-system vill man att det ska vara lättöverskådligt. Man ska få en bra överblick av sitt system och att navigera runt ska vara så simpelt som möjligt. Det hade vi i åtanke när vi bestämde hur utformningen av gränssnittet skulle se ut.

4.2.2 Aggregat

Vi skapade en skärm där de olika aggregaten/rummen skulle visas var för sig. För att komma till den skärmen klickar man antingen på knappen “aggregat”, men det går också bra att klicka på den stora bilden över hela systemet. Önskar man att enbart kolla på AL001 klickar man bara någonstans i området som hör till aggregatet. När man befinner sig i fönstret “aggregat” visas olika delar beroende på vilken knapp man trycker på. Till skillnad från bilden av hela systemet ser man här huruvida motorer, spjäll etc. är till/från samt vad givarna visar för tryck/temperatur. Här ges även möjlighet att klicka på exempelvis ett spjäll varpå ett pop up-fönster visas. I fönstret ser man om spjället är till(öppet) eller från(stängt) och aktuell status (manuellt/automatiskt läge, brand eller fel har uppstått). Det är också möjligt att ändra läget från manuellt till automatiskt och vice versa.

Skärmar för alarm och trender går också att nå via menyn. På alarmsidan syns en lista som uppdateras med larm i takt med att dem uppstår. Genom att markera ett larm kan man genom knapptryck på “info” se mer information om larmet. I trender visas historik för temperatur- och tryckgivare.

4.2.3 Kommunikation och taggar i HMI

För att gränssnittet ska kunna arbeta tillsammans med PLC-systemet krävs en sammanknytning. Vi använde Modbus TCP/IP som kommunikation däremellan. Valet vi gjorde berodde på att överföringshastigheten för Modbus TCP/IP är betydligt högre än vad den är för Modbus ASCII och Modbus RTU.

För att knyta samman vårt HMI och PLC-system använde vi oss av så kallade taggar. En tag kan antingen vara intern eller extern. Man kan välja att knyta ett objekt till en tag. Beroende på hur taggen är adresserad hämtar respektive skriver objektet till taggen. Med andra ord refererar antingen objektet till taggen eller tvärt om.

De interna taggarnas adressfält låter man förbli tomt. Exempelvis knöt vi processbilden över systemet till en tag utan att gå vidare och ange adress. Vi utnyttjade taggen till att bestämma när bilden skulle vara synlig eller ej. I fall som detta behövs ingen knytning med adress utan man får ut det man vill enbart genom att koppla bilden till taggen.

För externa taggar bestämmer man karakteristiken genom att ange om den ska vara analog eller diskret samt om det ska vara "Read-Only" eller "Read-Write". Här bestäms med andra ord om taggen ska vara en in- eller utgång till gränssnittet. Mer information angående teorin bakom detta ges under rubriken Modbus i teoriavsnittet. Ingångar har förmågan att kunna läsa medan utgångar både kan skriva och läsa. För att göra det enkelt för dig som läsare, väljer vi att referera till följande bild på nytt:

Coil/Register Numbers	Data Addresses	Type	Table Name
1-9999	0000 to 270E	Read-Write	Discrete Output Coils
10001-19999	0000 to 270E	Read-Only	Discrete Input Contacts
30001-39999	0000 to 270E	Read-Only	Analog Input Registers
40001-49999	0000 to 270E	Read-Write	Analog Output Holding Registers

Figur med info om de fyra olika tabellerna (Simply Modbus, u.å.).

Observera att adresserna ovan är uttryckta utifrån HMI:ets perspektiv. Det innebär att exempelvis en "Discrete Output Coil" kopplas till en utgång från HMI:et. Det första intervallet, 1-9999, används till att ställa in ett spjäll etc. i automatiskt läge via operatörspanelen.

I det andra intervallet, 10001–19999, knöt vi de diskreta taggarna som skulle vara ingångar till HMI:et. Sensorer som tittar på om spjäll är i öppet eller stängt läge är exempel på ingångar.

För de analoga in- och utgångarna gjordes knytningen på liknande vis som beskrevs ovan. Den stora skillnaden här är att signalerna inte enbart kan anta högt(true) eller lågt(false) värde utan de varierar mellan 4-20 mA som därefter skalas om.

Hela samlingen med taggar finns att hitta i bilaga 5.

4.2.4 Larm

I iX developer är det möjligt att skapa en "larm viewer". Den visar en tabell för de olika larmen. Larm dyker upp i tabellen då de inträffar, aktivt larm indikeras med röd färg. Genom att bekräfta att man sett larmet ändras färgen till grönt. När larmet är åtgärdat blir det inaktivt och färgen ändras till neutralt gul.

Larmen delades upp i olika grupper för att det skulle bli enklare att jobba med. Larm som alla rörde spjäll samlades för sig, ventiler för sig osv.

4.2.5 Trender

När vi skapade sidan för trender tog användning av interna taggar. Skärmen innehåller två olika diagram. Beroende på om man klickar på temperatur eller tryck sätts värdet hos den tag som är kopplad till knappen till true. Tack vare att de olika diagrammen refererar till respektive tag, som sätts av knapparna, kommer man att genom ett knapptryck bestämma vilket diagram som ska visas på skärmen.

När man bestämt sig för vilket diagram som ska visas finns det två ytterligare knappar, "history" och "legend". Klickar man på "history" visas historik för temperaturen/trycket. Klickar man istället på "legend" ges möjligheten att bestämma vilka kurvor som ska synas i diagrammet. Det finns t.ex. 8 stycken temperaturgivare, skulle man vara intresserad av att se endast en av dem klickar man på "legend" och väljer vilken.

4.2.6 Simulering av HMI

För att kontrollera hur kommunikationen mellan HMI:et och PLC:n fungerade användes en simulator som förhöll sig till Modbus-protokollet.

I samband med knytningen av taggar i iX developer valdes ett kommunikationssätt. Som tidigare nämnt föll valet på Modbus TCP/IP. Samtidigt som detta valdes skrevs en IP-adress in. Denna IP-adress representerar PLCn. Då en PLC inte fanns att tillhandahålla användes som tidigare nämnt en simulator istället (PLC-simulator, 2011). Även denna hade

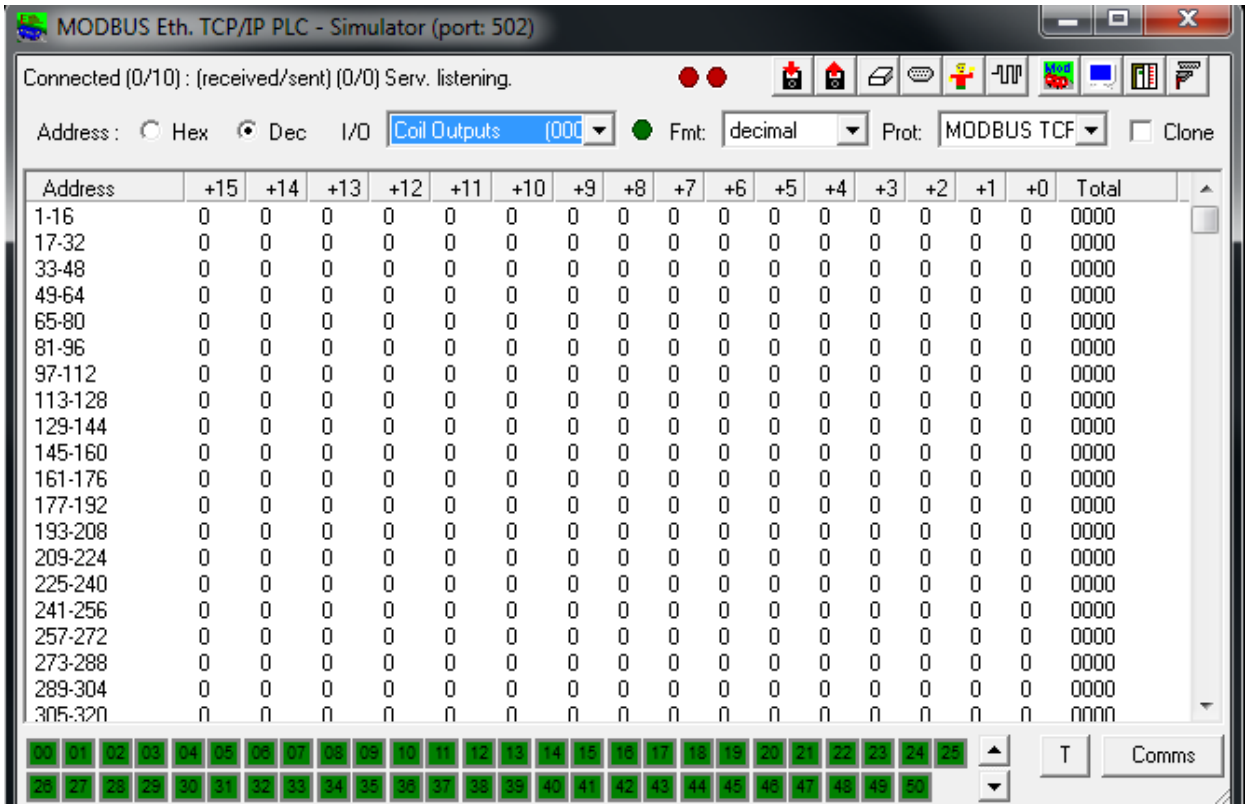
en IP-adress man kunde ange. Med hjälp av simulatorn var det möjligt att skicka och ta emot signaler. I simulatorn ställdes kommunikationssätt in så att det motsvarade valet vi gjorde i iX. Portnumret 502 var en förinställning allt eftersom simulatorn är avsedd för modbus-standarden.

I I/O-fönstret, se bild nedan, bestämmer man vilka signaler man vill kolla på genom att ändra i fönstret där det står “Coil Outputs”. Man kan välja mellan digitala in- eller utsignaler samt analoga dito. För att kontrollera de utgående diskreta signalerna från HMI:et till PLCn valdes “Coil Outputs” i I/O-fönstret. Genom att titta här kunde vi kontrollera att signalerna från HMI:et skickades iväg och togs emot korrekt hos PLCn (simulatorn i vårt fall). En logisk 1:a i HMI skulle innebära motsvarande i PLC.

För adresser i området 10001–19999, vilka beskriver diskreta insignaler till HMI:et (utsignaler från PLCn), sattes en logisk 1:a/0:a genom att dubbelklicka på valfri adress. När ett värde uppdaterats i adressfältet skickades signalen vidare till HMI:et. I vår HMI-visualisering kunde vi därefter kontrollera om kommunikationen mellan länkarna fungerade på ett korrekt vis.

När det gäller de analoga signalerna skickas ett analogt värde istället för ett diskret. Detta värde omvandlas med hjälp av en A/D-omvandlare med 16 bitar, numrerade som 0-15. Den nollte biten är reserverad till felhantering och används således ej. Den funktionella kontrollen gjordes på liknande sätt som för den diskreta, fast istället för en logisk 1:a/0:a visades ett analogt värde. 30001–39999 representerar ingångar till HMI:et och 40001–49999 som utgångar från HMI:et.

I samband med att vi kontrollerade kommunikationen med hjälp av simulatorn fick vi även möjlighet att testa om våra alarm fungerade som det var tänkt. Utlöste larmen när de skulle etc.



Med hjälp av simuleringen kunde vi följa hur kommunikationen utfärdades mellan PLC och HMI. Allting i HMI fungerade efter de kriterier som vi ville att det skulle göra, vilket gjorde att vi var nöjda med resultatet.

Nedan visas en bild för de möjliga inportarna till panelen. De möjliga inportarna för seriell, RS-485 och RS-232 som är de yttre ingångarna. Den mellersta som är en inport för Ethernet.

5 Diskussion och slutsats

Uppgifterna har fullföljts i stora drag sett till de syften och mål som sattes upp när vi befann oss i startgroparna av arbetet. PLC-programmet skapades, HMI-visualiseringen fick sin form och kommunikationen för det sammanslutna systemet utreddes. HMI:et som testkördes mot en Modbussimulator, istället för en riktig PLC, genererade i att vi fick en klar bild över hur kommunikationen mellan de olika dataadresserna (taggarna) fungerade. PLC-programmet innefattade 0 kompileringsfel och testkördes mot en virtuell PLC som fanns inbyggd i Codesys.

Ett problem som uppstod var att vi inte hade en riktig PLC att tillgå. Codesys är en så kallad “open source”-version. Detta innebär att hårdvaran (en PLC) krävs för att man ska kunna köra programmet tillsammans med I/O-enheter och HMI. I vårt fall hade vi inte tillgång till en PLC vilket gjorde att vi inte kunde testköra det PLC-program som vi hade skapat fullständigt. Med hjälp av den virtuella-PLCn som vi hade tillhands kunde vi konstatera att programmet i Codesys fungerade.

Enligt GANTT-schemats beskrivning hade vi i åtanke att undersöka ett flertal olika kommunikationsstandarder. Allt eftersom arbetet fortlöpte insåg vi konsekvenserna av detta val. Kvaliteten av arbetet hade sjunkit till en lägre grad och vår kunskap över hur kommunikationen fungerade hade med största sannolikhet inte blivit lika stor. Vi valde därför att fokusera på ett specifikt protokoll. Valet föll på Modbus, vilket innefattade överföring både seriellt och via nät.

Till en början när Ethernet skulle lämpas till industriell automation ansågs det vara ett omöjligt uppdrag. Mycket på grund av att den industriella automationen kräver kortare och precisa svarstider i större utsträckning till skillnad från vad som krävs i kontorsmiljö. Att behöva vänta lite extra på att en sida ska ladda är inte lika allvarligt som om en robot inte skulle få sin styrsignal i precis rätt tidpunkt. I vårt fall kan det handla om att t.ex. ett visst spjäll skall stänga vid en viss tidpunkt.

Ethernet i grunden är inte lämpat att arbeta i realtid på samma sätt som fältbussar. Anledningen till det är att det inte är determinsitiskt. Vilket kortsagt betyder att man inte kan förvänta sig när någonting ska hända. I detta fallet hur mycket data som skickas och när. Bakgrunden till det är att Ethernet är ett öppet nätverk och varje nod kan skicka data så fort kanalen är ledig.

Ethernet gör det möjligt att välja en topologi som nätverket fysiskt byggs upp efter. För att göra systemet bättre, sett till determinismen, ansluts varje enhet

till en switch. Det förhindrar i sin tur krockar från att inträffa i det interna nätverket, vilket också förbättrar svarstider inom realtidskommunikationen.

Det har utvecklats en rad olika Ethernetlösningar för att kunna tillämpa det till industriell automation. En av dessa är Modbus TCP/IP. En egenskap är att Modbus TCP/IP bl.a. möjliggör att skicka flera paket åt gången, utan att en bekräftelse på att skickad data har tagits emot. Man kan även skicka stora datamängder över Ethernet, d.v.s. bandbredden är väldigt hög. En av de främsta anledningarna till att man väljer att använda sig av Ethernet är just att man får en högre bandbredd. Framförallt för att man vill nå längre med utvecklingen. En annan fördel med ethernet är att det är lättåtkommligt eftersom det sker online.

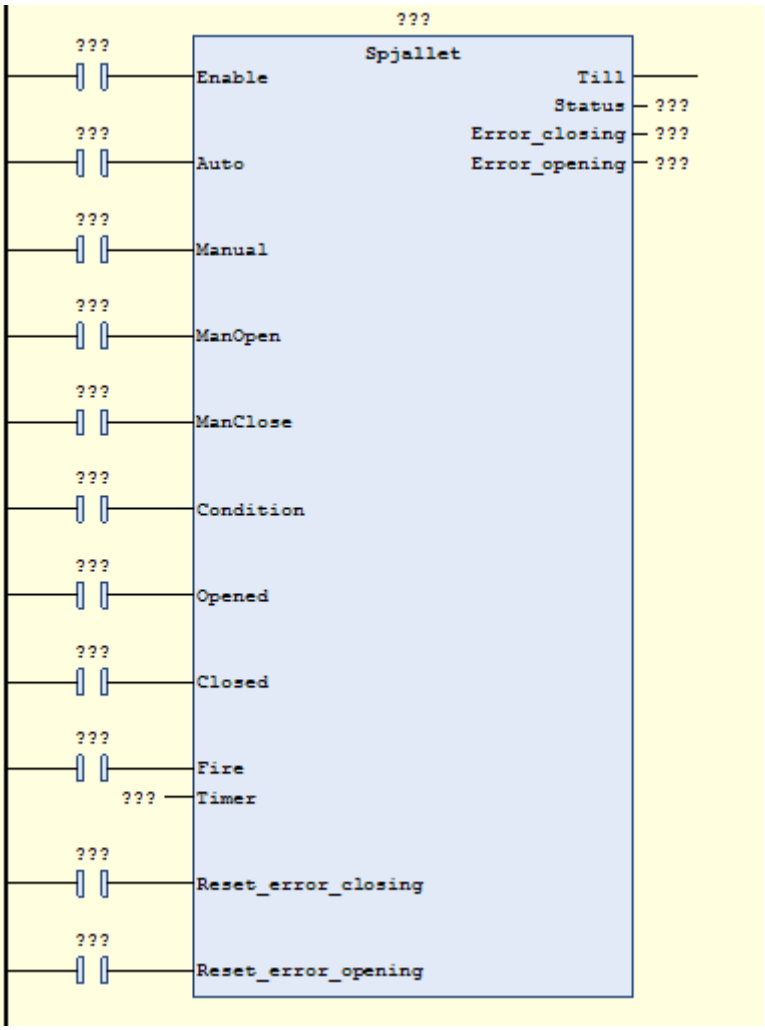
Hade vi använt oss av en seriell överföring, via antingen Modbus RTU eller ASCII, hade vi fått en deterministisk överföring med specificerade datamängder som både hade skickats och tagits emot under förutsägbara tidsintervall. Eftersom den seriella överföringen är deterministisk innebär det att svarstiden är mycket kortare. En seriell överföring hade dock inneburit en långsammare överföring med lägre bandbredd. Dessutom tillhör detta det billigare alternativet.

5.1 Vidarestudie

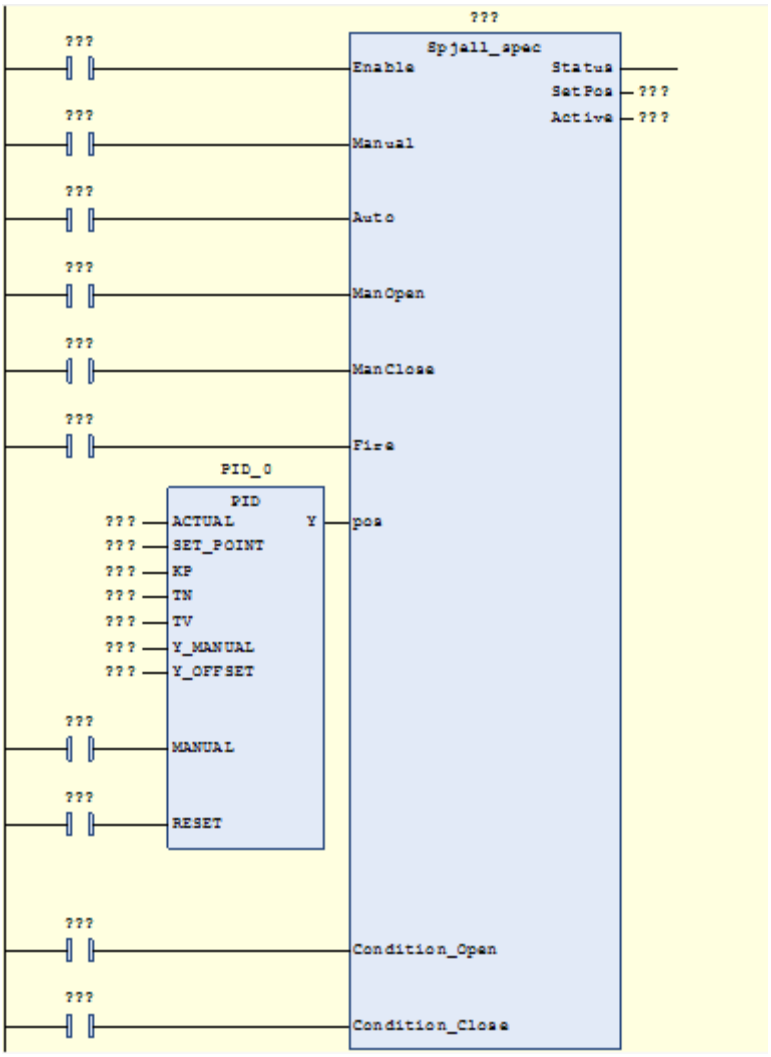
Om en vidarestudie skulle gjorts av arbetet, skulle det vara för att testa om hela systemet interagerade korrekt. I detta fall skulle det krävas en riktig PLC. Vidare hade vi behövt ersätta den nuvarande datorskärmen med en riktig HMI-panel.

6 Bilagor

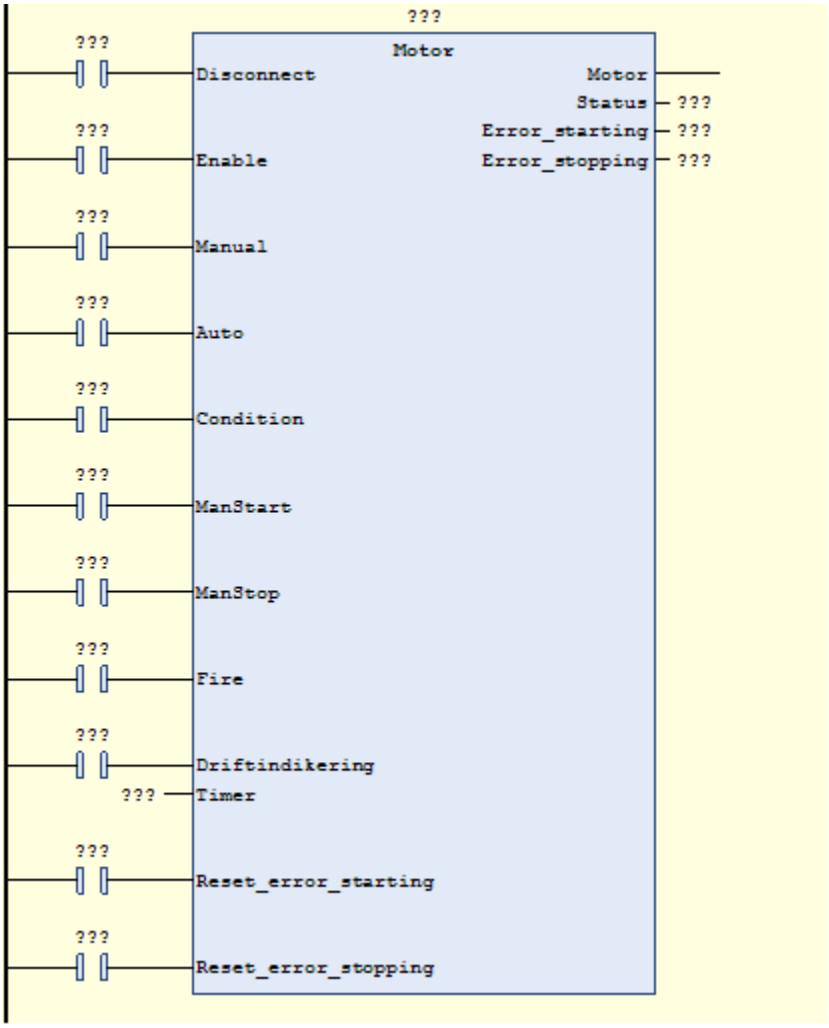
6.1 Funktionsblocken



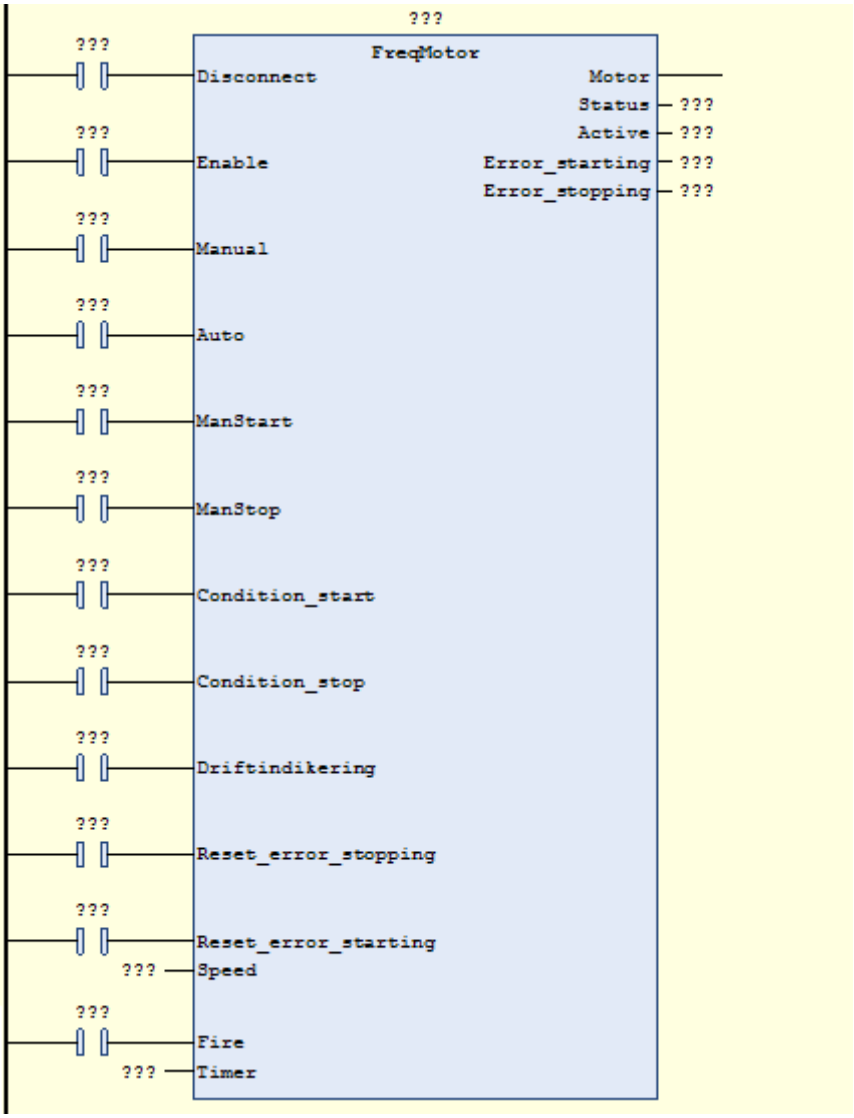
Spjällblock



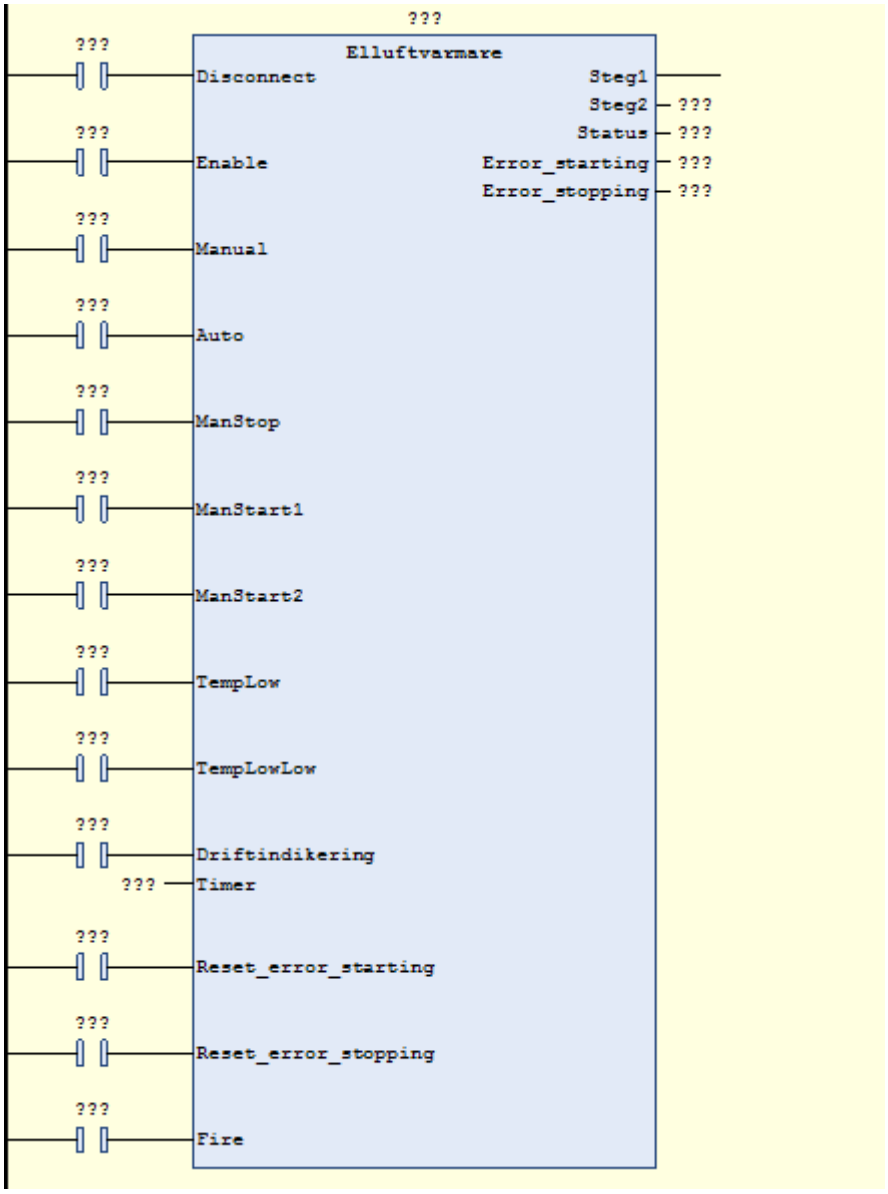
Reglerbart spjällblock med PID-regulator



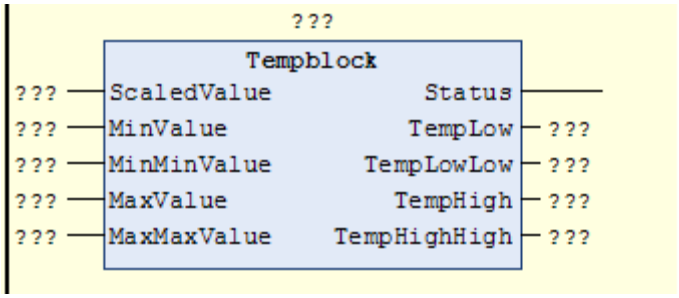
Start/stopp motorblock



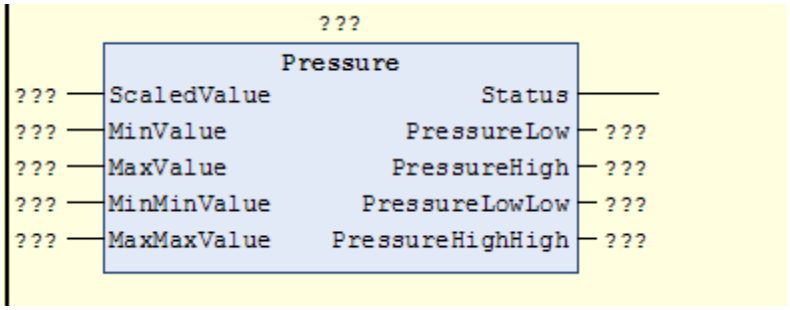
Frekvensmotorblock



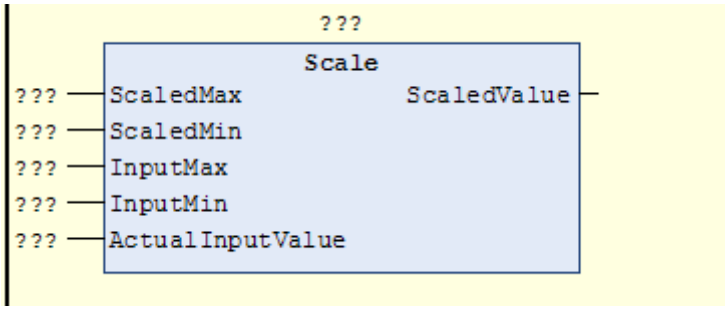
Elluftvärmarblock



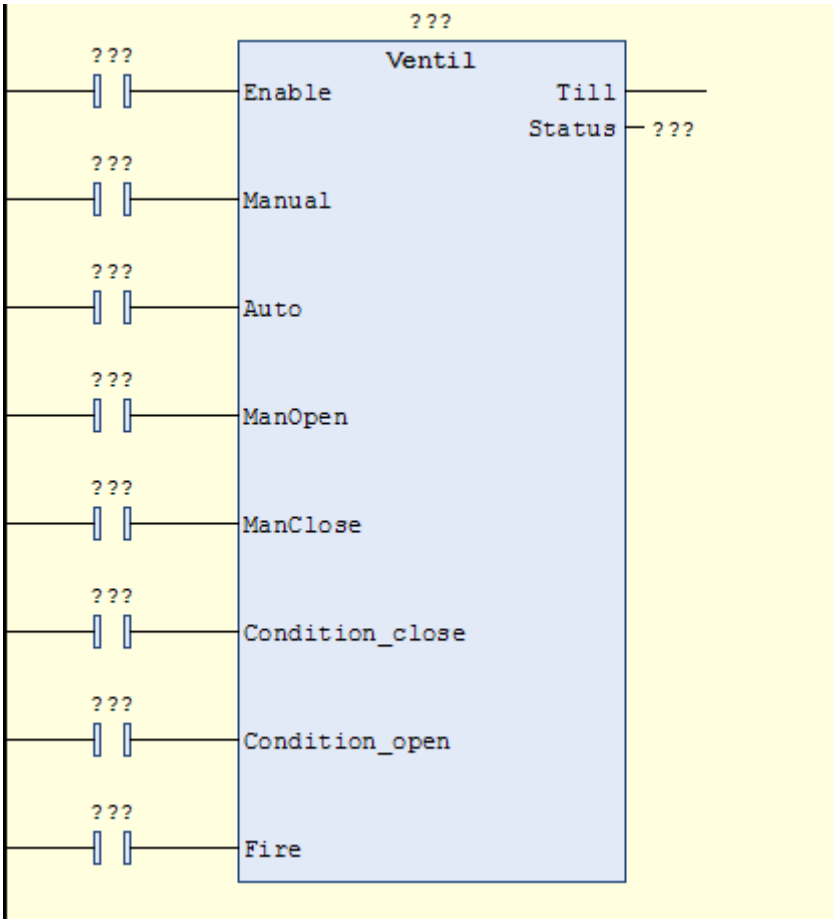
Temperaturblock



Tryckblock



Skalningsblock



Ventilblock

6.2 Driftkortsbeskrivning

ÖVERSIKT

Aggregat AL001 betjänar eldriftutrymme 22861 med förbehandlad uteluft.
Indikeringar för drift visas på operatörspanel samt överförs vidare till överordnat system,
PCMS MMI.
Larmen indikeras på operatörspanel samt överförs vidare till överordnat system PCMS MMI.

1. START/STOPP

För luftbehandlingssystem i anläggningen är tillufttemperaturen GT001 inställbar via överordnat system (PCMS MMI).
AL001 är i drift kontinuerligt. Vid utebliven styrsignal från PCMS ICS, på grund av fel i PCMS ICS stoppar aggregat AL001 varvid uteluftsspjäll stänger och aggregat AL002 startar med fullt öppen styrventil VE002 för kyla. AL002 startar även om temperaturen vid GT006 i rum Tele/Styr överskrider Hög/Hög temp +35°C.
Återkommer styrsignalen från PCMS ICS intar aggregat AL001 och AL002 läge för normaldrift.
Vid fel på aggregat AL001 stänger spjäll och aggregat AL002 startar för nödkyla.
Enskilda fläktar, spjäll och liknande i luftbehandlingssystem i anläggningen manövreras manuellt via operatörspanel.

5. BRAND/RÖK FUNKTIONER

Vid indikering rök på rökdetektor GY001 i driftutrymme stoppar aggregatet AL001 och AL002, tilluftspjäll SA004 och frånluftspjäll SK001-004 stänger.
Larm eventuella fellägen av brand/brandgasspjäll avges.
Vid mjukvaro larm (CBL) via TCP/IP, ska aggregat AL001 stoppa och AL002 starta.

6. SPJÄLL

Spjällställdon SA003 öppnar för uteluft vid start av tilluftsfläkt FT001, stänger vid stopp.
Spjällställdonen SA004-005 resp. SK001-002 stänger vid spänningsbortfall.
Ändlägen indikeras individuellt för SA004, SA005 och brand/brandgasspjäll med ställdon.
Brand/brandgasspjäll SK001-002 motioneras automatiskt var 48 timme i sekvens, först SK001 och sedan SK002 så att fläktar inte behöver stoppas under motioneringen. Vid motionering av spjäll i ELDU alterneras spjällen så att alltid ett av brand/brandgasspjällen är öppna.
Brand/brandgasspjäll SK001-002 avger larm om stängningstiden överstiger 10 sek.

9. FILTERVAKT

Filtervakt GP002 avger larm när tryckfallet över filtret i tilluften överstiger inställt värde.

10. REGLERING

Temperatur

Systemet AL001, reglerar ute- och återluftsspjällen så att tilluftstemperatur +16°C vid GT001 erhålls.

Om återluftstemperaturen GT002 överstiger +25°C öppnar kylventilen VR001 för köldbärare.

När återluftstemperaturen GT002 understiger +23°C stänger kylventilen VR001.

Om återluftstemperaturen GT002 är lägre än utomhustemperaturen GT003 stänger uteluftsspjället SA003 till min-läge och öppnar återluftsspjället SA003:1.

Cirkulationsluftaggregatet AC002 reglerar återluftstemperaturen genom i och urkoppling av eleffekten för elluftvärmaren XE002 i två steg.

Cirkulationsluftaggregatet AC002 startar då rumstemperaturen GT008 understiger +16ºC, sjunker temperaturen ytterligare kopplas elvärmarens 2:a steg in. Cirkulationsluftaggregatet AC002 stoppar då temperaturen vid GT008 överstiger +18ºC.

Temperaturstyrning batterirum sker via reglering av värme XE001/kyla VE001 i sekvens i aggregat AC001 så att inställd rumstemperatur vid GT006 erhålls.

Reglering av temperaturen för rum HSP/LSP./USP samt TELE/STYR/TRAFIKRUM sker genom att öppna och stänga reglerspjällen SA001, SA002 för ökat resp. minskat luftflöde.

Tryckregelering

Tryckgivare GP001 i tilluftskanal reglerar via PLC, frekvensomriktare US001 och tilluftsfläkt FT001 så att inställt börvärde erhålls.

11. LARMER

Temperaturlarmer för driftutrymmen inställs till följande värden enligt tabell

Rumstyp	Larm låg Temperatur °C	Larm hög Temperatur °C	Larm hög –hög Temperatur °C
Batterirum		+22	+25
Styr/Tele		+30	+35
HSP/LSP/ UPS		+30	+35

Larmgränser korrigeras efter drifttagandet för att kunna optimeras via lokal operatörspanel.

Fuktlarm GM001 installeras på fläktrumsgolv i ELDU.

12. LOGGNING

Värden för rumstemperatur GT005-008, tilluftstemperatur GT001 och utetemperatur GT003 samlas in (“loggas”) för lagring lokalt, Värden skall presenteras i OP panel och överförs till PCMS AÖ för presentation i PCMS MMI.

Gångreserv på realtidsklocka träder i funktion vid strömavbrott.

13. MOTIONERING

Inställda tider för motionskörningar presenteras och är inställbara från PCMS MMI.
Reservaggregat AL002 med nödkylfunktion i ELDU motioneras 1 gång per månad.
Styrventiler för kylvatten i ELDU motioneras 1 gång per månad.
Fellarm vid motioneringsfel visas på operatörspanel i apparatskåp samt överförs vidare till PCMS MMI.

14. LARMER / INDIKERINGAR

Beteckning	Benämning	Klass	Lokal operatörs panel	PCMS MMI
GP002	Filtervakt, högt tryckfall	B3	x	x
GP001	Tryckgivare lågt tryck	B2	x	x
US001	Summalarm frekvensomriktare.	B2	x	x
FC001	Konfliktlarm	B2	x	x
FC002	Konfliktlarm	B2	x	x
GTxx	Larm Hög temperatur	B2	x	x
GTxx	Larm Hög-Hög temperatur	B2	x	x
PCMS AÖ	Larm kommunikationsfel	B2	x	x
Apparatskåp	Summalarm utlöst motorskydd	B2	x	x
Elskåp	Larm utebliven matning 400VAC	B1	x	x
Elskåp	Larm utlöst säkring 400VAC	B2	x	x
Elskåp	Larm utlöst säkring 230VAC	B2	x	x
Elskåp	Larm utlöst säkring 24VDC	B2	x	x
GY001	Larm rökdetektor	B1	x	x
Centralt brandlarm	Larm	B1	x	x
Säkerhets brytare xx	Larm frånslagen	B2	x	x
SAxx, SKxx	Brand/brandgasspjäll motioneringsfel	B2	x	x
SAxx, SKxx	Brand/brandgasspjäll lång stängningstid / fel spjälläge	B2	x	x
GM001	Larm fukt golv i fläktrum	B2	x	x
MV001	Larm gränsvärde uppnått	B3	x	x
Analog ingång	Larm signalfel	B2	Per givare	Summa för PCMS ICS
	Indikering Manöver, FRÅN-TILL		x	x
PCMS ICS	Detaljerade parameterinställningar ex. -Regulator -Larmfördrjning -Larmgränser		x	x

Beteckning	Benämning	Klass	Lokal operatörs panel	PCMS MMI
GTxx	Indikering, Ärvärde		x	x
GTxx	Indikering, Börvärde		x	x
GTxx	Indikering, Gränsvärde		x	x
MV001	Indikering Mätvärde		x	x

15. MATERIALLISTA

Beteckning	Benämning	Artikel	Anmärkning
GP002	Filtervakt	LGW3A2	
GP001	Tryckgivare	QBM65-5	
GT001	Temperaturgivare	QAM2112.040	
GT002	Temperaturgivare	QAM2112.040	
GT003	Temperaturgivare	QAM2112.040	
GT004	Temperaturgivare	QAM2112.040	
GT005	Temperaturgivare	QAA2012	
GT006	Temperaturgivare	QAA2012	
GT007	Temperaturgivare	QAA2012	
GT008	Temperaturgivare	QAA2012	
VR001	Ventilställdon	SAX61.03	
VE002	Magnetventil		
VE001	Magnetventil		
US001	Frekvensomriktare	SED2-3/35B	
SK001	Brand/brandgasspjäll		
SK002	Brand/brandgasspjäll		
SA001	Spjällställdon	GEB131.1E	
SA002	Spjällställdon	GEB131.1E	
SA003	Spjällställdon	GBB161.1E	
SA003:1	Spjällställdon	GBB161.1E	
SA004	Spjällställdon	GCA126.1E	
SA005	Spjällställdon	GCA126.1E	
GM001	Fuktband	NVPF	
MV001	Kallvattenmätare		
GY001	Rökdetektor Sockel	EVC-PY-DA UB-6	

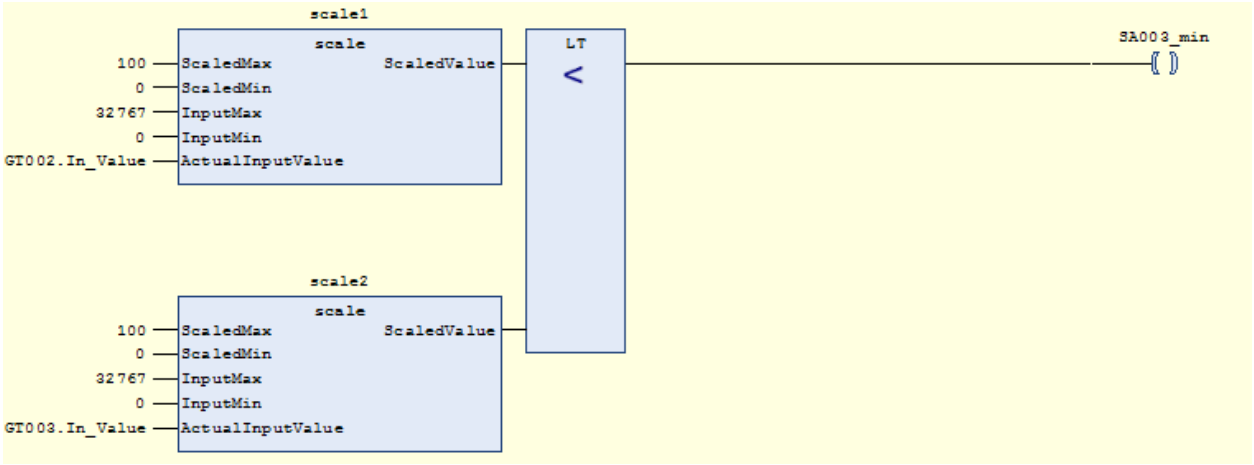
6.3 I/O-lista

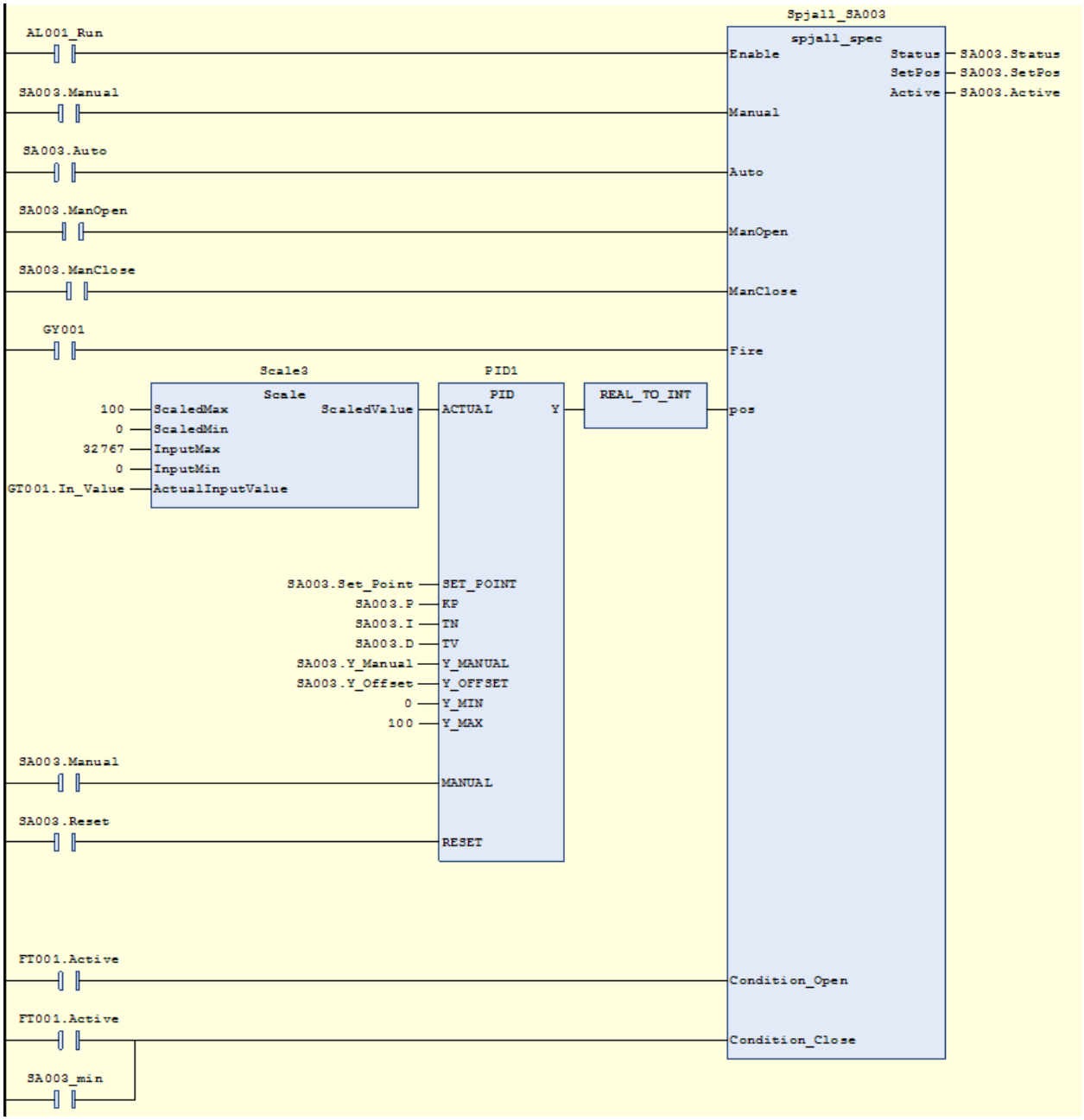
Apparatskåp +28861=573WA001						NI001 (DI)	NI001 (DO)	NI001 (AI)	NI001 (AO)	NI002 (AI)	NI003 (AI)	NI004 (AO)	NI005 (DI)	NI006 (DO)			
Beteckning	**Objekttyp**	**Benämning**				Plint	Plint	Plint	Plint	Plint	Plint	Plint	Plint	Plint	Mätområde	Enhet	Larm
	28861_AC002																
573FC003	Fläkt Cirkulation	Cirkulation	Start/Stopp				27										B2
	Fläkt Cirkulation		Säkerhetsbrytare			9											
573GT008	Temperaturgivare	Rum								35-38					0-50	°C	B2
573XE002	Elvärme	Värme	Överhettning			14											B2
			Steg 1											3			
			Steg 2											7			
552MV01	Vattenmätare Puls	Kyla	Driftindikering			15											
573GM001	Fuktband	Vattenläckage golv	Larmindikering			16											B2
	28861_HSP/LSP/UPS																
573SA002	Spjällställdon	Forcering	Till/Från				32										
573GT006	Temperaturgivare	Rum								27-30					0-50	°C	B2
	28861_TELE/STYR/TRAFIKRUM																
573SA001	Spjällställdon	Forcering	Till/Från				28										
573GT005	Temperaturgivare	Rum								23-26					0-50	°C	B2
	28861_AC001/BATTERI																
573FC002	Fläkt Cirkulation	Cirkulation	Start/Stopp				33										B2
	Fläkt Cirkulation		Driftindikering			17											
	Fläkt Cirkulation		Säkerhetsbrytare			18											
573GT007	Temperaturgivare	Rum								35-38					0-50	°C	B2
573XE001	Elvärme	Värme	Driftindikering			19											B2
			Start/Stopp											3			
			Säkerhetsbrytare										12				
552VE001	Magnetventil	Kyla					34										

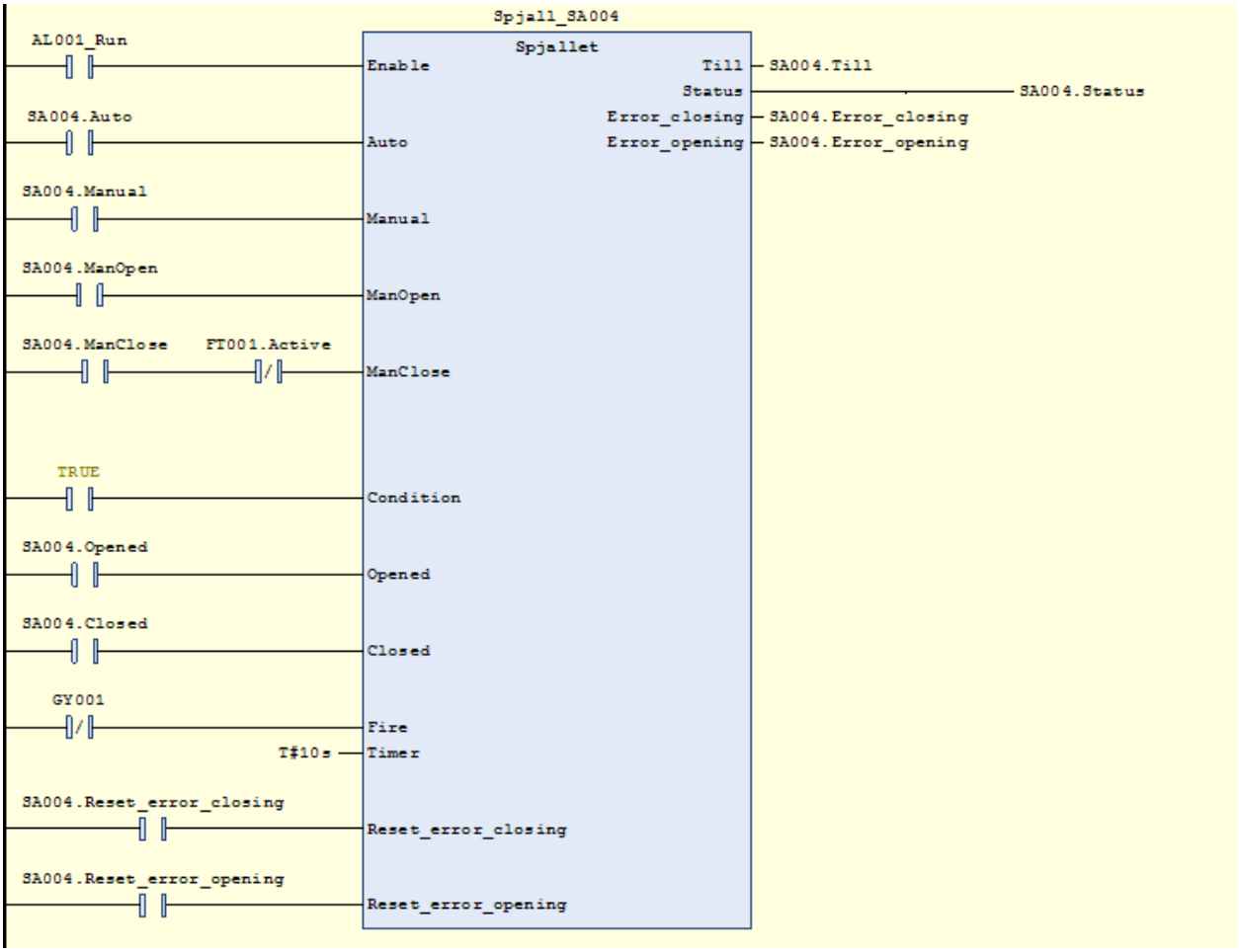
Apparatskåp +28861=573WA001						NI001 (DI)	NI001 (DO)	NI001 (AI)	NI001 (AO)	NI002 (AI)	NI003 (AI)	NI004 (AO)	NI005 (DI)	NI006 (DO)			
Beteckning	**Objekttyp**	**Benämning**				Plint	Plint	Plint	Plint	Plint	Plint	Plint	Plint	Plint	Mätområde	Enhet	Larm
	28861_573AL001																
573US001	Frekvensomriktare	Tilluft	Start/Stopp	FT001			22										B2
	Frekvensomriktare		Driftindikering	FT001		2											
	Frekvensomriktare		Larmindikering	FT001		3											
	Frekvensomriktare		Säkerhetsbrytare	FT001		4											
	Frekvensomriktare		Analogt börvärde	FT001								3				0-10V	
573GT003	Temperaturgivare	Uteluft								3-6					-35-+35	°C	B2
573GT002	Temperaturgivare	Återluft								7-10					0-50	°C	B2
573SA003	Spjällställdon	Ute/återluft	Till/Från						18							0-10V	
573GP002	Filtervakt	Tilluft				5											B3
552VR001	Ventilställdon								16							0-10V	
573SA004	Spjällställdon	Tilluft	Till/Från				23										B2
	Spjäll		Stängd			12											
	Spjäll		Öppet			13											
573GT001	Temperaturgivare	Tilluft								11-14					0-50	°C	B2
573GP001	Tryckgivare	Tilluft						2								Pa	B2
	28861_573AL002																
573FC001	Fläkt Cirkulation	Tilluft	Start/Stopp				24										B2
	Fläkt Cirkulation		Driftindikering			6											
	Fläkt Cirkulation		Säkerhetsbrytare			7											
573GT004	Temperaturgivare	Tilluft								15-18					0-50	°C	B2
573SA005	Spjällställdon	Tilluft	Till/Från				25										B2
	Spjäll		Stängd										7				
	Spjäll		Öppet										8				
552VE002	Magnetventil	Kyla					26										

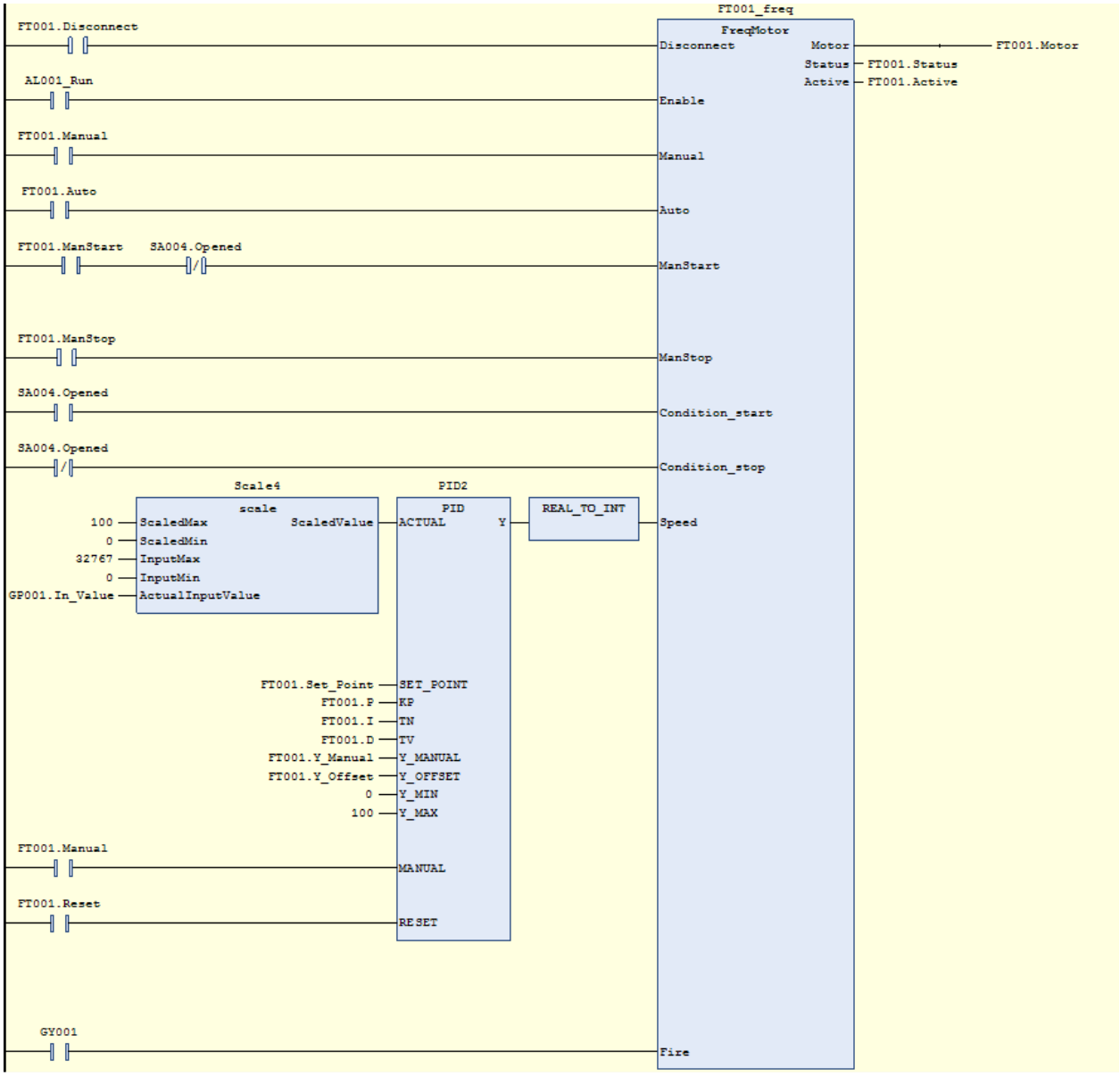
Apparatskåp +28861=573WA001						NI001 (DI)	NI001 (DO)	NI001 (AI)	NI001 (AO)	NI002 (AI)	NI003 (AI)	NI004 (AO)	NI005 (DI)	NI006 (DO)			
Beteckning	**Objekttyp**	**Benämning**				Plint	Plint	Plint	Plint	Plint	Plint	Plint	Plint	Plint	Mätområde	Enhet	Larm
	28861_BRANDSPJÄLL																
573SK001	Brand/brandgasspjäll	Frånluft trafikutrymme	Till/Från				35										B2
	Brand/brandgasspjäll		Stängd			22											
	Brand/brandgasspjäll		Öppet			23											
573SK002	Brand/brandgasspjäll	Frånluft trafikutrymme	Till/Från				36										B2
	Brand/brandgasspjäll		Stängd			25											
	Brand/brandgasspjäll		Öppet			26											
	INTERNT I APPARATSKÅP																
	Utebliven matning 400AC												4				B1
	Utlöst manöversäkring 230VAC												5				B2
	Utlöst manöversäkring 24VDC												6				B2
	Utlöst motorskydd												13				B2
	Centralt brandlarn BLC												9				B1

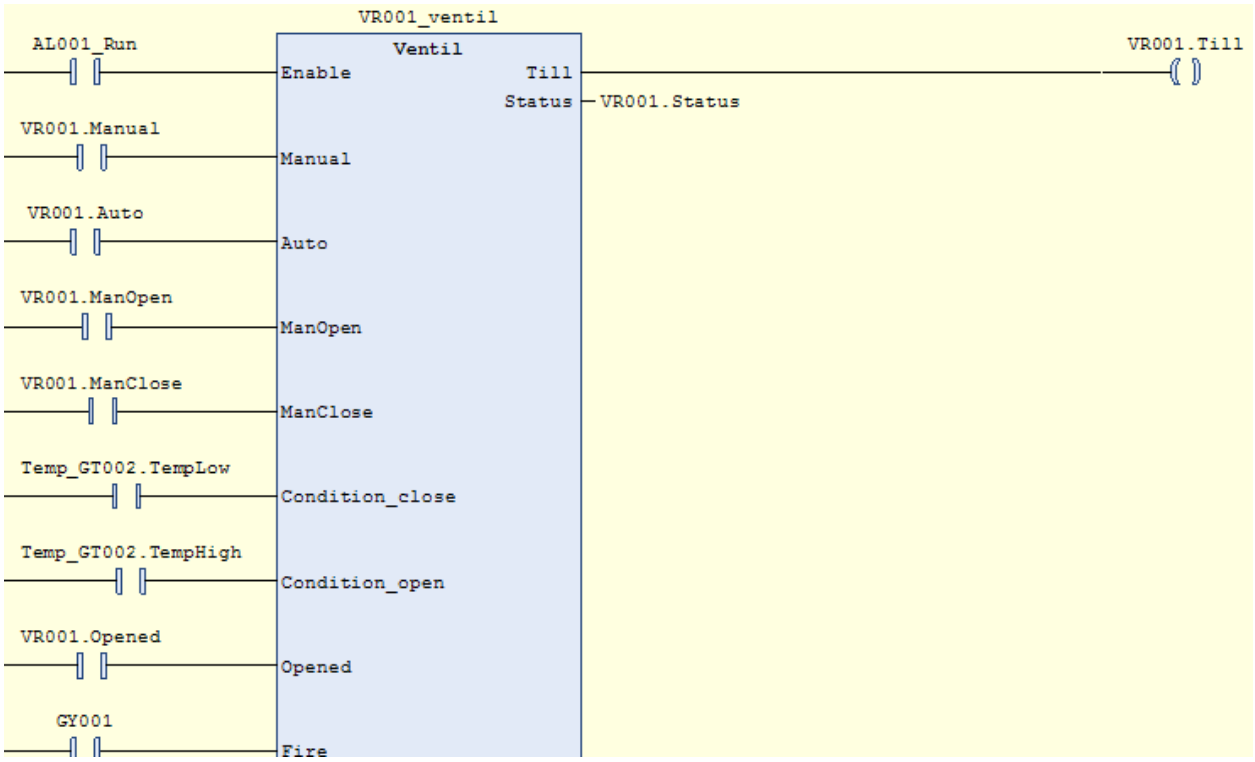
6.4 Nätverk för AL001

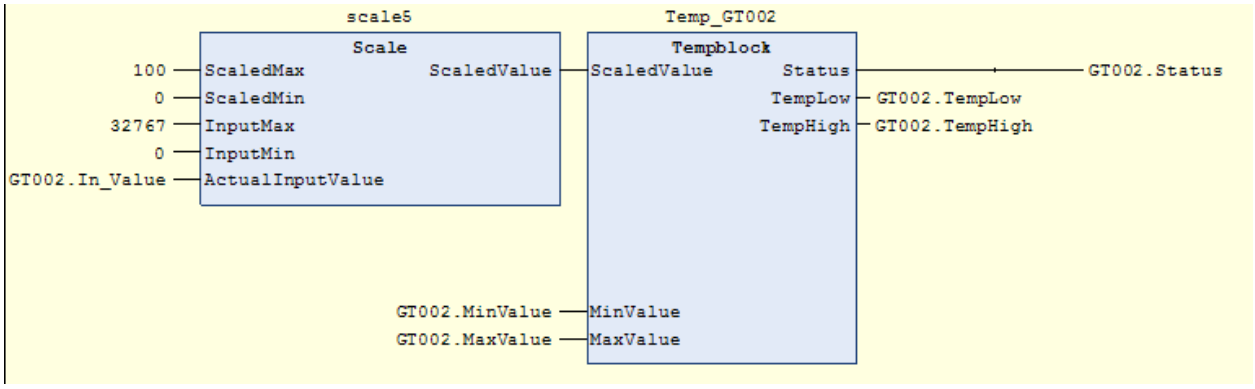


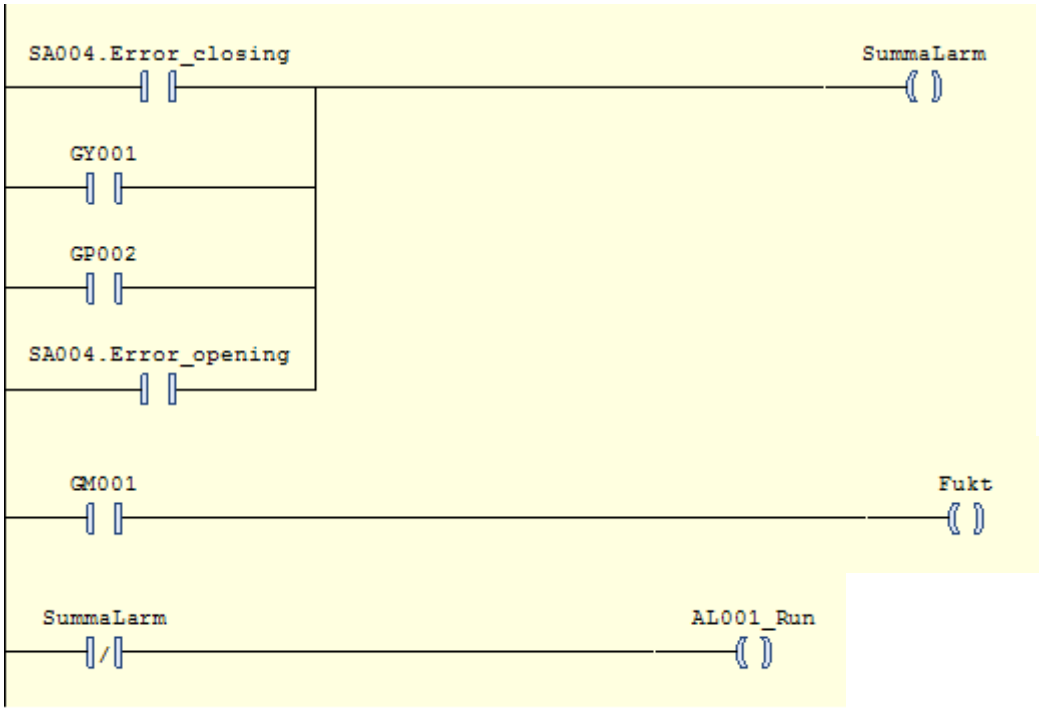

Spjall_SA003
spjall_spec
AL001_Run
Enable
Status
SA003.Status
SetPos
SA003.SetPos
Active
SA003.Active
SA003.Manual
Manual
SA003.Auto
Auto
SA003.ManOpen
ManOpen
SA003.ManClose
ManClose
GY001
Fire
Scale3
Scale
PID1
PID
REAL_TO_INT
100
ScaledMax
ScaledValue
ACTUAL
Y
pos
0
ScaledMin
32767
InputMax
0
InputMin
GT001.In_Value
ActualInputValue
SA003.Set_Point
SET_POINT
SA003.P
KP
SA003.I
TN
SA003.D
TV
SA003.Y_Manual
Y_MANUAL
SA003.Y_Offset
Y_OFFSET
0
Y_MIN
100
Y_MAX
SA003.Manual
MANUAL
SA003.Reset
RESET
FT001.Active
Condition_Open
FT001.Active
Condition_Close
SA003_min


Spjall_SA004
Spjallet
AL001_Run
Enable
SA004.Auto
Auto
SA004.Manual
Manual
SA004.ManOpen
ManOpen
SA004.ManClose
FT001.Active
ManClose
TRUE
Condition
SA004.Opened
Opened
SA004.Closed
Closed
GY001
Fire
T#10s
Timer
SA004.Reset_error_closing
Reset_error_closing
SA004.Reset_error_opening
Reset_error_opening
Till
SA004.Till
Status
SA004.Status
Error_closing
SA004.Error_closing
Error_opening
SA004.Error_opening


FT001_freq
FreqMotor
FT001.Disconnect
Disconnect
Motor
FT001.Motor
Status
FT001.Status
Active
FT001.Active
AL001_Run
Enable
FT001.Manual
Manual
FT001.Auto
Auto
FT001.ManStart
SA004.Opened
ManStart
FT001.ManStop
ManStop
SA004.Opened
Condition_start
SA004.Opened
Condition_stop
Scale4
scale
100
ScaledMax
ScaledValue
0
ScaledMin
32767
InputMax
0
InputMin
GP001.In_Value
ActualInputValue
PID2
PID
ACTUAL
Y
REAL_TO_INT
Speed
FT001.Set_Point
SET_POINT
FT001.P
KP
FT001.I
TN
FT001.D
TV
FT001.Y_Manual
Y_MANUAL
FT001.Y_Offset
Y_OFFSET
0
Y_MIN
100
Y_MAX
FT001.Manual
MANUAL
FT001.Reset
RESET
GY001
Fire


VR001_ventil
Ventil
AL001_Run
Enable
Till
VR001.Till
Status
VR001.Status
VR001.Manual
Manual
VR001.Auto
Auto
VR001.ManOpen
ManOpen
VR001.ManClose
ManClose
Temp_GT002.TempLow
Condition_close
Temp_GT002.TempHigh
Condition_open
VR001.Opened
Opened
GY001
Fire


scale5
Scale
100
ScaledMax
ScaledValue
0
ScaledMin
32767
InputMax
0
InputMin
GT002.In_Value
ActualInputValue
Temp_GT002
Tempblock
ScaledValue
Status
GT002.Status
TempLow
GT002.TempLow
TempHigh
GT002.TempHigh
GT002.MinValue
MinValue
GT002.MaxValue
MaxValue


SA004.Error_closing
SummaLarm
GY001
GP002
SA004.Error_opening
GM001
Fukt
SummaLarm
AL001_Run

6.5 Taggar i HMI

Name	Data type	Access Right	Data type	Modbus
SA001				
SA001_öppna	DEFAULT	ReadWrite	BIT	00001
SA001_Till	DEFAULT	ReadWrite	BIT	10005
SA001_stäng	DEFAULT	ReadWrite	BIT	00016
SA001_Status	DEFAULT	ReadWrite	INT16	30010
SA001_Reset_ErrorStop	DEFAULT	ReadWrite	BIT	00075
SA001_Reset_ErrorStart	DEFAULT	ReadWrite	BIT	00063
SA001_man	DEFAULT	ReadWrite	BIT	00047
SA001_errorstopp	DEFAULT	ReadWrite	BIT	10033
SA001_errorstart	DEFAULT	ReadWrite	BIT	10021
SA001_auto	DEFAULT	ReadWrite	BIT	00031
SA002				
SA002_man	DEFAULT	ReadWrite	BIT	00048
SA002_auto	DEFAULT	ReadWrite	BIT	00032
SA002_öppna	DEFAULT	ReadWrite	BIT	00002
SA002_stäng	DEFAULT	ReadWrite	BIT	00017
SA002_Status	DEFAULT	ReadWrite	INT16	30011
SA002_Till	DEFAULT	ReadWrite	BIT	10006
SA002_errorstopp	DEFAULT	ReadWrite	BIT	10034
SA002_errorstart	DEFAULT	ReadWrite	BIT	10022
SA002_Reset_ErrorStart	DEFAULT	ReadWrite	BIT	00064
SA002_Reset_ErrorStop	DEFAULT	ReadWrite	BIT	00076
SA003				
SA003_auto	DEFAULT	ReadWrite	BIT	00033
SA003_man	DEFAULT	ReadWrite	BIT	00049
SA003_vinkel	DEFAULT	ReadWrite	INT16	30026
SA003_Status	DEFAULT	ReadWrite	INT16	30012
SA003_Till	DEFAULT	ReadWrite	BIT	10007
SA003_Vinkel_Man	DEFAULT	ReadWrite	INT16	40006
PID_SA003_Man	DEFAULT	ReadWrite	DEFAULT	
SA003_K	DEFAULT	ReadWrite	INT16	40011
SA003_I	DEFAULT	ReadWrite	INT16	40012
SA003_D	DEFAULT	ReadWrite	INT16	40013
SA004				
SA004_öppna	DEFAULT	ReadWrite	BIT	0003
SA004_stäng	DEFAULT	ReadWrite	BIT	00018
SA004_auto	DEFAULT	ReadWrite	BIT	00034
SA004_man	DEFAULT	ReadWrite	BIT	00050
SA004_Status	DEFAULT	ReadWrite	INT16	30013
SA004_Till	DEFAULT	ReadWrite	BIT	10008

SA004_errorstart	DEFAULT	ReadWrite	BIT	10023
SA004_errorstopp	DEFAULT	ReadWrite	BIT	10035
SA004_Reset_ErrorStart	DEFAULT	ReadWrite	BIT	00065
SA004_Reset_ErrorStop	DEFAULT	ReadWrite	BIT	00077

SA005

SA005_auto	DEFAULT	ReadWrite	BIT	00035
SA005_man	DEFAULT	ReadWrite	BIT	00051
SA005_stäng	DEFAULT	ReadWrite	BIT	00019
SA005_öppna	DEFAULT	ReadWrite	BIT	00004
SA005_Status	DEFAULT	ReadWrite	INT16	30014
SA005_Till	DEFAULT	ReadWrite	BIT	10009
SA005_errorstopp	DEFAULT	ReadWrite	BIT	10036
SA005_errorstart	DEFAULT	ReadWrite	BIT	10024
SA005_Reset_ErrorStart	DEFAULT	ReadWrite	BIT	00066
SA005_Reset_ErrorStop	DEFAULT	ReadWrite	BIT	00078

SK001

SK001_auto	DEFAULT	ReadWrite	BIT	00036
SK001_man	DEFAULT	ReadWrite	BIT	00052
SK001_öppna	DEFAULT	ReadWrite	BIT	0005
SK001_stäng	DEFAULT	ReadWrite	BIT	00020
SK001_Status	DEFAULT	ReadWrite	INT16	30015
SK001_Till	DEFAULT	ReadWrite	BIT	10010
SK001_errorstopp	DEFAULT	ReadWrite	BIT	10037
SK001_errorstart	DEFAULT	ReadWrite	BIT	10025
SK001_Reset_ErrorStart	DEFAULT	ReadWrite	BIT	00067
SK001_Reset_ErrorStop	DEFAULT	ReadWrite	BIT	00079

SK002

SK002_auto	DEFAULT	ReadWrite	BIT	00037
SK002_errorstart	DEFAULT	ReadWrite	BIT	10026
SK002_errorstopp	DEFAULT	ReadWrite	BIT	10038
SK002_man	DEFAULT	ReadWrite	BIT	00053
SK002_Reset_ErrorStart	DEFAULT	ReadWrite	BIT	00068
SK002_Reset_ErrorStop	DEFAULT	ReadWrite	BIT	00080
SK002_Status	DEFAULT	ReadWrite	INT16	30016
SK002_stäng	DEFAULT	ReadWrite	BIT	00021
SK002_Till	DEFAULT	ReadWrite	BIT	10011
SK002_öppna	DEFAULT	ReadWrite	BIT	00006

FC001

FC001_auto	DEFAULT	ReadWrite	BIT	00041
FC001_Brytare	DEFAULT	ReadWrite	BIT	00088
FC001_errorstart	DEFAULT	ReadWrite	BIT	10027
FC001_errorstopp	DEFAULT	ReadWrite	BIT	10039

FC001_man	DEFAULT	ReadWrite	BIT	00057
FC001_Reset_ErrorStart	DEFAULT	ReadWrite	BIT	00069
FC001_Reset_ErrorStop	DEFAULT	ReadWrite	BIT	00081
FC001_start	DEFAULT	ReadWrite	BIT	00010
FC001_Status	DEFAULT	ReadWrite	INT16	30020
FC001_stopp	DEFAULT	ReadWrite	BIT	00025
FC001_Till	DEFAULT	ReadWrite	BIT	10012

FC002

FC002_auto	DEFAULT	ReadWrite	BIT	00042
FC002_Brytare	DEFAULT	ReadWrite	BIT	00089
FC002_errorstart	DEFAULT	ReadWrite	BIT	10028
FC002_errorstopp	DEFAULT	ReadWrite	BIT	10040
FC002_man	DEFAULT	ReadWrite	BIT	00058
FC002_Reset_ErrorStart	DEFAULT	ReadWrite	BIT	00070
FC002_Reset_ErrorStop	DEFAULT	ReadWrite	BIT	00082
FC002_start	DEFAULT	ReadWrite	BIT	00011
FC002_Status	DEFAULT	ReadWrite	INT16	30021
FC002_stopp	DEFAULT	ReadWrite	BIT	00026
FC002_Till	DEFAULT	ReadWrite	BIT	10013

FC003

FC003_auto	DEFAULT	ReadWrite	BIT	00043
FC003_Brytare	DEFAULT	ReadWrite	BIT	00090
FC003_errorstart	DEFAULT	ReadWrite	BIT	10029
FC003_errorstopp	DEFAULT	ReadWrite	BIT	10042
FC003_man	DEFAULT	ReadWrite	BIT	00059
FC003_Reset_ErrorStart	DEFAULT	ReadWrite	BIT	00071
FC003_Reset_ErrorStop	DEFAULT	ReadWrite	BIT	00083
FC003_start	DEFAULT	ReadWrite	BIT	00012
FC003_Status	DEFAULT	ReadWrite	INT16	30022
FC003_stopp	DEFAULT	ReadWrite	BIT	00027
FC003_Till	DEFAULT	ReadWrite	BIT	10014

FT001

FT001_auto	DEFAULT	ReadWrite	BIT	00044
FT001_Brytare	DEFAULT	ReadWrite	BIT	00087
FT001_D	DEFAULT	ReadWrite	INT16	40010
FT001_errorstart	DEFAULT	ReadWrite	BIT	10030
FT001_errorstopping	DEFAULT	ReadWrite	BIT	10043
FT001_I	DEFAULT	ReadWrite	INT16	40009
FT001_K	DEFAULT	ReadWrite	INT16	40008
FT001_man	DEFAULT	ReadWrite	BIT	00060
FT001_Reset_ErrorStart	DEFAULT	ReadWrite	BIT	00072
FT001_Reset_ErrorStop	DEFAULT	ReadWrite	BIT	00084
FT001_setPoint	DEFAULT	ReadWrite	INT16	40007

FT001_speed	DEFAULT	ReadWrite	INT16	40005
FT001_Status	DEFAULT	Read	INT16	30023
FT001_Till	DEFAULT	Read	BIT	10045
GIVARE				
GM001	DEFAULT	ReadWrite	BIT	10003
GP001	DEFAULT	ReadWrite	INT16	30009
GP002	DEFAULT	ReadWrite	BIT	10001
GT001	DEFAULT	ReadWrite	INT16	30001
GT002	DEFAULT	ReadWrite	INT16	30002
GT003	DEFAULT	ReadWrite	INT16	30003
GT004	DEFAULT	ReadWrite	INT16	30004
GT005	DEFAULT	ReadWrite	INT16	30005
GT006	DEFAULT	ReadWrite	INT16	30006
GT007	DEFAULT	ReadWrite	INT16	30007
GT008	DEFAULT	ReadWrite	INT16	30008
GY001	DEFAULT	ReadWrite	BIT	10002
MV001	DEFAULT	ReadWrite	BIT	10004
Centralt_brandlarm	DEFAULT	ReadWrite	BIT	10050
Interna knappar				
PID_FT001_Man	DEFAULT	ReadWrite	DEFAULT	
PID_SA003_Man	DEFAULT	ReadWrite	DEFAULT	
Processbild_AC002	DEFAULT	ReadWrite	DEFAULT	
Processbild_AL001	DEFAULT	ReadWrite	DEFAULT	
Processbild_AL002	DEFAULT	ReadWrite	DEFAULT	
Processbild_Batterirum	DEFAULT	ReadWrite	DEFAULT	
Processbild_Brand	DEFAULT	ReadWrite	DEFAULT	
Processbild_HSP	DEFAULT	ReadWrite	DEFAULT	
Processbild_Trafikrum	DEFAULT	ReadWrite	DEFAULT	
Temperatur	DEFAULT	ReadWrite	DEFAULT	
Tryck	DEFAULT	ReadWrite	DEFAULT	
Säkringar				
UteblivenMatning	DEFAULT	ReadWrite	BIT	10049
säkring_230VAC	DEFAULT	ReadWrite	BIT	10047
säkring_24VDC	DEFAULT	ReadWrite	BIT	10046
säkring_400VAC	DEFAULT	ReadWrite	BIT	10048
Motorskydd	DEFAULT	ReadWrite	BIT	10051
Timers				
timer_brandspjäll	DEFAULT	ReadWrite	INT16	40001
timer_elluftvärmare	DEFAULT	ReadWrite	INT16	40002
timer_motor	DEFAULT	ReadWrite	INT16	40003
timer_spjäll	DEFAULT	ReadWrite	INT16	40004

VE001

VE001_auto	DEFAULT	ReadWrite	BIT	00038
VE001_man	DEFAULT	ReadWrite	BIT	00054
VE001_Status	DEFAULT	ReadWrite	INT16	30017
VE001_stäng	DEFAULT	ReadWrite	BIT	00022
VE001_Till	DEFAULT	ReadWrite	BIT	10018
VE001_öppna	DEFAULT	ReadWrite	BIT	00007

VE002

VE002_auto	DEFAULT	ReadWrite	BIT	00039
VE002_Man	DEFAULT	ReadWrite	BIT	00055
VE002_Status	DEFAULT	ReadWrite	INT16	30018
VE002_stäng	DEFAULT	ReadWrite	BIT	00023
VE002_Till	DEFAULT	ReadWrite	BIT	10020
VE002_öppna	DEFAULT	ReadWrite	BIT	00008

VR001

VR001_auto	DEFAULT	ReadWrite	BIT	00040
VR001_man	DEFAULT	ReadWrite	BIT	00056
VR001_Status	DEFAULT	ReadWrite	INT16	30019
VR001_stäng	DEFAULT	ReadWrite	BIT	00024
VR001_Till	DEFAULT	ReadWrite	BIT	10019
VR001_öppna	DEFAULT	ReadWrite	BIT	00009

XE001

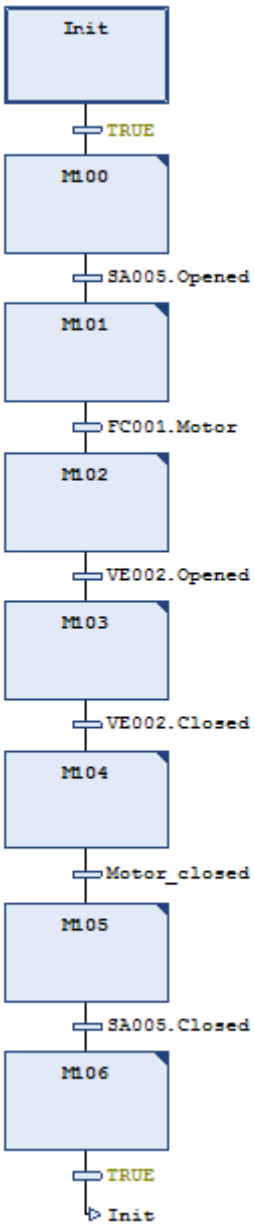
XE001_auto	DEFAULT	ReadWrite	BIT	00045
XE001_Brytare	DEFAULT	ReadWrite	BIT	00091
XE001_errorstart	DEFAULT	ReadWrite	BIT	10031
XE001_errorstopp	DEFAULT	ReadWrite	BIT	10041
XE001_man	DEFAULT	ReadWrite	BIT	00061
XE001_Reset_ErrorStart	DEFAULT	ReadWrite	BIT	00073
XE001_Reset_ErrorStop	DEFAULT	ReadWrite	BIT	00085
XE001_start	DEFAULT	ReadWrite	BIT	00013
XE001_Status	DEFAULT	ReadWrite	INT16	30024
XE001_stopp	DEFAULT	ReadWrite	BIT	00028
XE001_till	DEFAULT	ReadWrite	BIT	10015

XE002

XE002_auto	DEFAULT	ReadWrite	BIT	00046
XE002_Brytare	DEFAULT	ReadWrite	BIT	00092
XE002_errorstart	DEFAULT	ReadWrite	BIT	10032
XE002_errorstopp	DEFAULT	ReadWrite	BIT	10044
XE002_man	DEFAULT	ReadWrite	BIT	00062
XE002_manstart1	DEFAULT	ReadWrite	BIT	00014
XE002_manstart2	DEFAULT	ReadWrite	BIT	00015
XE002_Reset_ErrorStart	DEFAULT	ReadWrite	BIT	00074

XE002_Reset_ErrorStop	DEFAULT	ReadWrite	BIT	00086
XE002_Status	DEFAULT	ReadWrite	INT16	30025
XE002_steg1	DEFAULT	ReadWrite	BIT	10016
XE002_steg2	DEFAULT	ReadWrite	BIT	10017
XE002_stopp	DEFAULT	ReadWrite	BIT	00029

6.6 Sekvensdiagram för motion



7 Referenslista

Goodtech (u.å.). **http://goodtech.se/sv/tjanster/automation/operatorssystem?openDocument** [2014-04-16]

Zurawski.R (2009). Embedded Networks for Automation. DOI: 10.1201/9781439807620.ch20

Codesys (u.å.) **http://www.codesys.com/branches/factory-automation.html** [2014-04-14]

Zurawski,R (2004). The Industrial Information Technology Handbook. DOI: 10.1201/9781420036336.ch65

Galloway. B, Hancke. P. G (u.å.). Introduction to Industrial Control Networks . http://www.rfidblog.org.uk/Preprint-GallowayHancke-IndustrialControlSurvey.pdf [2014-04-15]

Heinz John, A, M Tiegelkamp (2010). IEC 61131-3: Programming Industrial Automation Systems. DOI: 10.1007/978-3-642-12015-2

Bolton. W (2009) Programmable Logic Controllers (Fifth Edition). Finns som e-bok här http://www.sciencedirect.com/science/book/9781856177511 [2014-04-16] ISBN: 978-1-85617-751-1

Medida. S (2008). IDC Engineering Pocket Guide, Volume 6, Industrial Automation: http://student.ch.lu.se/lth/mats/kurser/automation/pdf/Pocket_Guides/PG6_Automation_r2.2.pdf

Daneels. A och Salter. W (1999). International Conference on Accelerator and Large Experimental Physics Control Systems - What is SCADA?: http://accelconf.web.cern.ch/accelconf/ica99/papers/mc1i01.pdf

Stouffer. K, Falco. J och Kent. K (2006). Guide to Supervisory Control and Data Acquisition (SCADA) and Industrial Control Systems Security: http://www.dhs.gov/sites/default/files/publications/csd-nist-guidetosupervisoryanddataccquisition-scadaandindustrialcontrolsystemssecurity-2007.pdf

Facstaff (u.å.)
http://www.facstaff.bucknell.edu/mastascu/elessonshtml/Interfaces/ConvAD.html

Kurose. J.F och Ross.K.W (2010). Computer and networking, femte upplagan.

TCP traffic analyser, u.å. http://yconalyzer.sourceforge.net/ [2014-05-02]

Introduction to Modbus Serial and Modbus TCP, 2008
http://www.ccontrols.com/pdf/Extv9n5.pdf [2014-05-03]

Acromag (2005). Introduction to Modbus TCP/IP.
http://www.prosoft-technology.com/kb/assets/intro_modbustcp.pdf

Simply Modbus (u.å.) http://www.simplymodbus.ca/FAQ.htm [2014-05-04]

Pefhany. S (2000). Modbus Protocol
http://irtfweb.ifa.hawaii.edu/~smokey2/software/about/sixnet/modbus/modbus_protocol.pdf

Modicon (1996). Modicon, Modbus Protocol Reference Guide
http://web.eecs.umich.edu/~modbus/documents/PI_MBUS_300.pdf

Modbus-IDA (2006). Modbus Application Protocol Specification.
http://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b.pdf

Sena (2002). Introduction to Modbus - Technical tutorial
http://www.sena.com/download/tutorial/tech_Modbus_v1r0c0.pdf

Automation.com - Introduction to Modbus (u.å.)
http://www.automation.com/library/articles-white-papers/fieldbus-serial-bus-io-networks/introduction-to-modbus [2014-05-05]

encyclopedia, u.å. http://www.pcmag.com/encyclopedia/term/44300/hmi
[2014-05-05]

Beijer electronics , u.å.
http://www.beijer.se/web/web_se_be_se.nsf/alldocuments/534B417A18A42EE1C12579BF004522C0 [2014-05-05]

Beijer electronics , u.å.
http://www.beijer.se/web/web_se_be_se.nsf/AllDocuments/0ADB4B4126703B91C1257AFA00291ADC [2014-05-05]

PLC-simulator, versionen som släpptes 10/1 2011.
http://www.plcsimulator.org/ [2014-05-09]

Matteguiden (u.å.)
http://www.matteguiden.se/matte-b/linjara-samband/en-linjes-ekvation/ [2014-05-02]