

Programmering av operatörspanel med SoftPLC

Med komma-igång-manualer för iX Developer och CODESYS

Examensarbete inom högskoleingenjörsprogrammet Mekanik

LINDA ALEXANDERSSON
OLA VINGREN

Sammanfattning

Chalmers Lindholmen har köpt in ny utrustning i form av operatörspaneler med integrerat styrsystem för att använda i kurser inom ingenjörsutbildning vid Signaler och system. Då utrustningen och programvarorna inte tidigare använts saknas lättlästa manualer på svenska både för installation och användning. Appendix till denna rapport utgörs av de framtagna komma-igång-manualerna för utvecklingsmiljöerna CODESYS och iX Developer som används för programmering av styrsystemet respektive konstruktion av processbild. För att skaffa material till manualerna och lära känna programvarorna har ett styrprojekt utförts med seriell dataöverföring mellan det integrerade styrsystemet och processen via fältbussen EtherCAT. Processen styrs och övervakas från operatörspanelen. Syftet med manualerna är att framtida studenter ska kunna komma igång och programmera betydligt snabbare än om de inte hade skrivits då installation och konfiguration av utvecklingsmiljöerna varit komplicerad och svåröverskådlig. Huvudrapporten innehåller framförallt beskrivningar av de använda utvecklingsmiljöerna och av industriell kommunikation. Huvudrapporten innehåller dessutom information om den process som använts för att samla material till manualerna.

Summary

Chalmers University has invested in new equipment, HMI-panels with an integrated controller, to be used for educational purposes in courses by the department of System and signals. Since this equipment and the software have not been used previously there is a lack of easy-to-read manuals for installation and general use written in Swedish. This report and the enclosed appendixes contain easy-to-read manuals for the development environments CODESYS and iX Developer used for programming of the controller and construction of the display design. In order to get familiarized with the software and gather material to the manuals a control and programming project was carried out that included serial communication via a fieldbus. The controlled process could be monitored and controlled from the human-machine interface. The purpose of these manuals is to enable future students to faster and more easily get into the programming and construction of both the programmable controller and the display, particularly since the installation and configuration of these development environments was complicated and difficult to follow. The main content of the project report is narratives regarding industrial communication and the software used in the project. Moreover it withholds information about the process used to create program examples for the manuals.

Förkortningar

PLC – Programmable Logic Controller, programmerbart styrsystem

HMI – Human-Machine Interface, gränssnitt mellan människa och maskin.

CODESYS – Controller Development System, utvecklingsmiljö för PLC-program.

EtherCAT – Ethernet for Control Automation Technology, fältbussystem för automation.

CPU – Central Processing Unit, en enhet som exekverar program i en dator.

Innehåll

1 Inledning.....	1
1.1 Bakgrund	1
1.2 Syfte	1
1.3 Mål.....	1
1.4 Avgränsningar.....	2
2 Metod	3
3 Teknisk referensram.....	4
4 Genomförande	5
4.1 Utrustning och programvaror.....	5
4.1.1 PLC och Soft PLC	5
4.1.2 CODESYS.....	6
4.1.3 Fältbuss.....	7
4.1.4 Ethernet.....	8
4.1.5 EtherCAT	9
4.1.6 Decentraliserade input/output-enheter Crevis.....	11
4.1.7 Industriellt HMI	12
4.1.8 iX Developer 2.0.....	12
4.1.9 Processen	13
4.2 Nedladdning och installation av programvaror	14
4.2.1 Installation av CODESYS	14
4.2.2 Installation av iX Developer	14
4.4 Programmering	15
4.5 Uppbyggnad av processbild i iX Developer	15
4.6 Manualskrivande.....	17
4.6.1 Komma i gång med CODESYS	17
4.6.2 Komma i gång med iX Developer	17
4.7 Utredning av funktionsblock på internet	17
5 Resultat	18
6 Slutsatser och diskussion	19
Referenser	21
APPENDIX 1 PLC-programmering i CODESYS, Komma-igång-manual	
APPENDIX 2 Programmering i iX Developer, Komma-igång-manual	

1 Inledning

Här nedan beskrivs projektets bakgrund, syfte, mål och avgränsningar.

1.1 Bakgrund

Då behovet av att automatisera processer har ökat ända sedan 1700-talet samt att fler områden automatiseras är ämnet högintressant för framtidens studenter. Den tekniska utvecklingen har gått från att använda hålkort som ett sätt att styra en automatisk vävstol på 1700-talet via användning av reläer och timers då fabriker började använda el runt 1900 fram till PLC:ns (Programmable Logic Controller, programmerbart styrsystem) inträde på marknaden i slutet av 1960-talet. Dagens teknik ger möjligheter att använda en PC utrustad med programvara så att denna kan agera PLC eller en PLC integrerad i ett HMI, t.ex. en pekskärm. Då fler och fler uppgifter i samhället av kostnads-, ergonomi- och/eller miljöskäl automatiseras är detta ett expanderande område. Energibesparingar vid fastighetsautomation kan nämnas som ett område där miljöpåverkan kan minskas då temperatur och koldioxidhalt i ett utrymme övervakas och värmetillförsel och ventilation sedan styrs. [1], [2].

1.2 Syfte

Arbetets syfte är att ta i drift en HMI-panel med en inbyggd PLC-funktion och ta fram komma-igång-manualer att använda för utbildningsändamål i ingenjörsutbildningen.

1.3 Mål

Följande mål har ställts upp:

- Undersöka och presentera programvaran CODESYS och kommunikationslänken EtherCAT, deras historik, användning, utveckling och egenskaper.
- Idrifttagning av SoftControl (styrsystemet som är integrerat i panelen) samt genomförande av ett styrprojekt där en bearbetningsprocess styrs från panelen.
- Konfiguration av EtherCAT-kommunikation i nät med två noder och enkel processbild.
- Kartlägga tillgång till andrapartsutvecklade funktionsblock.
- Konfiguration av operatörspanel mot utvecklingsmiljö för styrsystemet.
- Konfiguration av operatörspanelen mot Mitsubishis Q02-system.
- Larmhantering i iX-miljö.
- Konstruera processbild för övervakning av processen i iX Developer.
- Skriva komma-igång-manual till framtida studenter för användning av CODESYS i iX-panelen, där konfigurationen av EtherCAT-kommunikationen förklaras och visas. Manualen ska innehålla exempel i språken ST, FBD, LD, SFC och CFC samt instruktion om skapande av egna funktionsblock.
- Skriva en komma-igång-manual för iX Developer för framtida studenter.

1.4 Avgränsningar

Komma-igång-manualerna är avsedda för nya användare av systemen CODESYS och iX Developer, dessa användare är teknologer men har inte använt programvarorna tidigare. Inga ekonomiska hänsynstaganden har gjorts. Tilldelad processutrustning för programmering av en styrprocess har använts. PLC-språket IL nämns kortfattat, men studeras ej närmare. Språket CFC ges också en ganska kort genomgång, eftersom det ligger utanför normal PLC-standard (IEC 61131-3).

2 Metod

Arbetet inleds med litteratursökning och inläsning på de områden som ingår i projektet (EtherCAT, CODESYS, iX-panel, decentraliserade I/O:n, databussar, HMI-paneler och PLC-system, iX Developer). De programvaror inskaffas som behövs för att lösa uppgifterna och dessa installeras. Nästa steg är konfigurering samt idrifttagning av utrustningen (i labbet uppbyggd borr- och stansprocess, iX-panel med tillhörande PLC) i syfte att lära känna utvecklingsmiljön CODESYS och SoftPLC. Till sist att genomföra ett styrprojekt på en borr- och stansprocess i syfte att lära känna CODESYS och dokumentation av arbetet och de kunskaper som förvärvas under dess gång samt att sammanställa dokumentationen till manualer.

3 Teknisk referensram

Detta avsnitt bygger delvis på Osbeck, Styr- och övervakningssystem [2].

Chalmers har köpt in pekskärmar, iX T12B SoftControl från Beijer Electronics, med avsikt att dessa ska användas för processövervakning i ett styrprojekt i utbildningssyfte. Dessa operatörspaneler kan användas tillsammans med en extern PLC eller med det inbyggda styrsystemet som sitter i panelen, kallat SoftControl. Kombinationen av PLC som styr t.ex. en monteringslina eller en klimatanläggning i en fastighet tillsammans med någon typ av operatörsgränssnitt, från enkla knappar till en panel med pekskärm för visning och inmatning är vanligt förekommande inom fastighets- och industriell automation. Som inledningsvis beskrevs är detta ett växande område, inte bara inom tillverkningsindustrin utan även inom telefonin i de switchar som används i nätverken, för att styra värmepannor vid uppvärmning av vatten till bostäder, för temperatur och ventilationsstyrning i bostäder, vid stabilisering och styrning av fartyg, bilar och flygplan samt inom jordbruk. Av detta skäl är det viktigt att framtidens studenter som vill jobba inom automation har fått konfigurera och programmera flera typer av PLC:er i olika utvecklingsmiljöer för att få ett smakprov på verklighetens PLC-programmering.

4 Genomförande

Arbetet inleddes med litteraturstudier, nedladdning av installations- och användarmanualer samt installation av programvarorna iX Developer och CODESYS.

4.1 Utrustning och programvaror

Nedan presenteras de verktyg som använts under projektet.

4.1.1 PLC och Soft PLC

Texten bygger på Wiklund m.fl., Styrprojekt vid elektro- och mekatronikingenjörsprogrammet [1] och Styr- och övervakningssystem [2].

PLC-system har ersatt reläer och timers inom automation sedan 1970-talet. Den första PLC:n utvecklades av Bedford Associates och tanken var att använda den inom amerikansk bilindustri där målet var att minska de installations- och hårdvarukostnader som var förknippat med dåtidens relästyrningar. Sedan den första PLC:n, Modicon 084 (förkortning av MODular DIGital CONTroller), konstruerades 1968 har PLC:ns betydelse inom industrin ökat för att idag även användas inom många andra områden som t.ex. fastighetsautomation, trafikstyrning m.m. Bilindustrin är fortfarande en av världens största användare av PLC.

Ett fundamentalt krav hos PLC-system är att de behöver vara snabba på att läsa och behandla data som krävs för att styra en eller flera processer. En PLC består av ett antal in- och utgångar, ett program och ett minne. In- och utmatningen till portarna är en av de mest tidskrävande processerna i systemet. En PLC har ett cykliskt arbetssätt och en cykel är den tid från start av inläsning av ingångarna till att utsignalerna skickats till utportarna och den beror på klockfrekvens och arbetssätt hos processorn samt av styrprogrammets längd, vanligt är en cykeltid på ett par till ett tiotal millisekunder. Ingångarna läses i början av varje cykel och deras värden bearbetas i det programmet som körs. I slutet av cykeln sätts sedan utgångarna så att deras status stämmer med de i programmet. Cykeln upprepas sedan om och om igen.

PLC:er kan vara modulära eller kompakta, där en kompakt PLC vanligen består av en enda komponent med strömförsörjning, in- och utportar och minne, knappsats och ev. display. Dessa är ofta billigare än modulära men inte lika flexibla ifråga om att utöka in- och utgångar vid behov. De är lätta att byta ut och ersätts därför ofta med nya, i stället för att repareras. En modulär PLC är mer flexibel ifråga om att bygga ut med fler in- och utgångar som läggs till som moduler vid behov, ofta monterade på en skena.

Ett SoftPLC (mjukvaru-PLC på svenska) är en HMI med PLC-funktion integrerad eller en persondator utrustad med programvara som gör att denna kan agera PLC. PC-marknadens kraftiga expansion och snabba utvecklingstakt har medfört att bättre och billigare datorer finns på marknaden. PLC:ns utvecklingstakt har inte varit lika snabb som PC:n och dess prestanda kan anses vara ett antal år bakom PC:ns. Därmed har idén uppstått, att kombinera PC:ns högre prestanda och utrusta denna med mjukvara så att den kan agera PLC tillsammans med decentraliserade I/O:n som samlar ihop data och skickar denna seriellt via en databuss till och från processen.

För- och nackdelar med PLC/Soft PLC kan diskuteras. Den traditionella PLC:n anses av en del vara mer tillförlitlig, speciellt i svåra miljöer inom industrin där den behöver tåla bl.a. vibrationer, stötar, gaser, fukt och magnetiska fält. En PC är mer kraftfull men är avsedd för en mer lätthanterlig miljö och kanske inte klarar de svårare förhållanden som kan råda inom vissa verksamheter. Det finns dock speciella integrerade PLC på marknaden för användning i svårare miljöer.

4.1.2 CODESYS

Avsnittet bygger på CoDeSys Control V3 Manual [3], CODESYS på Wikipedia [4] samt Wikipedia IEC 61131-3 [5].

CODESYS (även stavat CoDeSys, förkortning av **Controller Development System**) är en öppen mjukvaruplattform för programmering av PLC och SoftPLC utvecklad för att uppfylla kraven hos moderna automationsprojekt i industrin. 3S-Smart Software Solutions GmbH i Kempten i Tyskland är tillverkare av CODESYS.

Stora ansträngningar gjordes för att standardisera PLC-programmering och under slutet av 1993 utvecklades en standard för PLC-programmering, IEC 61131-3. Detta resulterade i en programmodell m.a.p. strukturen med ett par medföljande fördelar:

- Exekveringen baseras på tasks, olika tasks kan ha olika villkor för exekvering
- Mjukvarustruktur baserad på POU (Program Organization Units)

Utöver detta definierades fem (nedan givna) standardspråk vilka möjliggör för programmerare att jobba effektivare med PLC från olika tillverkare. Även om ovan givna åtgärder förbättrat läget finns det tyvärr fortfarande skillnader mellan de olika PLC-fabrikaten, vilket medför att programmerare upplever sig låsta till en tillverkare. Bland dessa skillnader kan nämnas att det fortfarande finns leverantörsspecifika tillägg till standardspråken och att skillnader i layout och utseende i utvecklingsmiljöerna kan göra det svårt att programmera PLC:er av olika fabrikat. Skillnaderna kan medföra kostnader p.g.a. försenade projekt och för utbildning av personal.

Med IEC 61131-3 och de återstående utmaningarna som bakgrund släpptes 1994 version 1.0 av CODESYS, där utvecklingsmiljön skulle vara enklare att använda, flexibel m.a.p. leverantörer av PLC så att en programmerare kan använda samma utvecklingsmiljö till PLC:er av olika fabrikat. Detta möjliggör för företag att välja de PLC:er som bäst passar deras behov och kan leda till minskade kostnader.

CODESYS följer internationella industristandarden IEC 61131-3 för programmering av PLC och stödjer de fem standardspråken, plus ett CODESYS-unikt:

1. IL (Instruction List) består av en serie instruktioner som matas in i följd och består av en operand (vilken signal som ska inkluderas i operationen) och en operationsdel (vad som ska göras, kan vara en logisk operation t.ex.). IL liknar språket assembler och användes mycket förr samt även idag då mindre system programmeras direkt via en inkopplad dosa.
2. LD (Ladder Diagram) är ett grafiskt språk som påminner om relälogiken. Slutande eller brytande kontakter och resulterande utsignaler används för att bygga upp logiken.

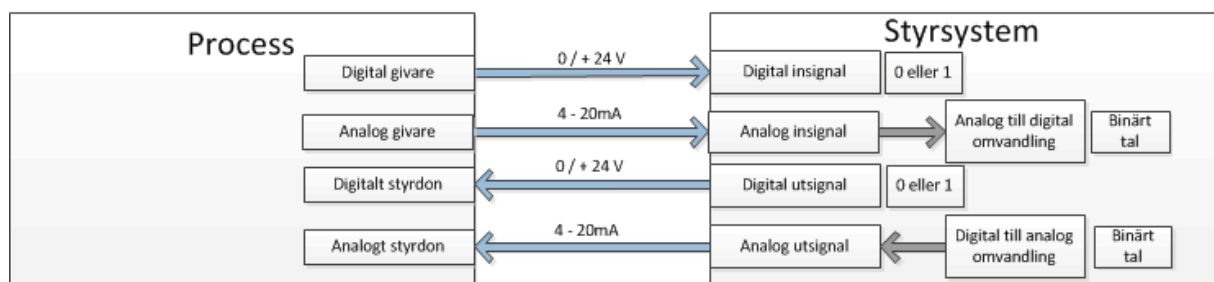
3. FBD (Function Block Diagram) ett grafiskt språk för att beskriva funktionen mellan in- och utsignaler.
4. SFC (Sequential Function Chart) används för att programmera då processen kan delas upp i olika steg med övergångsvillkor mellan stegen. Till varje steg finns actions, d.v.s. uppgifter som ska utföras innan övergångsvillkor och nästa steg.
5. ST (Structured Text) Ett högnivåspråk likt C och Pascal.
6. CFC (Continuous Function Chart) är ytterligare ett språk som finns i CODESYS, detta liknar FBD men med skillnaden att programmeraren i CFC själv behöver koppla ihop operatörer, input och output. Blocken kan placeras efter eget tycke vilket ger större frihet samt möjlighet att bygga återkopplade loopar utan mellanlagring av variabler.

I detta projekt har språken ST, FBD, SFC och LD använts för att styra processen. Vid sidan om detta har FBD använts för att konstruera ett tidtagarur i form av ett funktionsblock.

4.1.3 Fältbuss

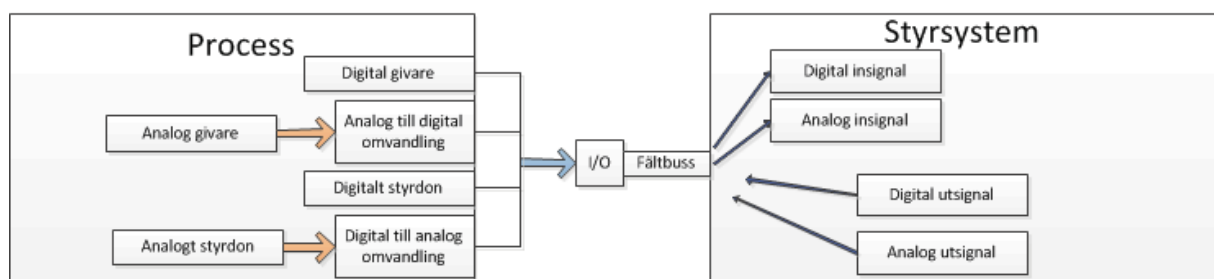
Avsnittet bygger på Styr- och övervakningssystem [2].

Då en process ska styras behöver signaler överföras mellan det industriella styrsystemet och processen. En signalledare krävs för varje givare och varje aktuator (styrdon) för överföring av antingen digitala eller analoga signaler. Vid traditionell parallell signalöverföring mellan process och styrsystem krävdes alltså en ledare för varje ut- och insignal.



Figur 4.1 Parallell traditionell signalöverföring mellan process och styrsystem, en ledare för varje sensor och styrdon. Industristandard vid analog signalering är 4-20 mA och vid digital signalering 0V / 24 V.

Vid användning av en fältbuss och därmed seriell överföring av informationen mellan processen och styrsystemet samlas givar- och aktuatorernas signaler ihop ute i processen. Denna samling sker i en decentraliserad I/O-enhet (I/O är förkortning av Input/Output) och den samlade informationen skickas seriellt via fältbussen mellan I/O:na och styrsystemet. Med seriell överföring avses att informationen översätts till och skickas digitalt, d.v.s som en serie med ettor och nollor.



Figur 4.2 Seriell överföring där signalerna från processen samlas i hop i I/O:t och skickas seriellt via fältbussen.

Att via en fältbuss ansluta enheter ger en del fördelar jämfört med traditionell parallell signalöverföring:

- Minskad kostnad framförallt p.g.a mindre dragning av kabel för signalöverföring vilket ger minskat kablage, mindre kostnad för installations och dokumentation.
- Förenklad anslutning av ny utrustning
- Möjligheter att skriva och läsa parametrar
- Möjligt att göra diagnostik över nätverket

Ett par nackdelar:

- Fältbussens nackdel jämfört med den traditionella uppkopplingen är att den är långsammare, detta p.g.a. den tid som åtgår för att översätta signalerna till digital information istället för att direkt skicka in dem till styrsystemet.
- Flera eller t.o.m. alla funktioner kan slås ut vid fel.
- Ett komplexare och mer svåröverskådligt system kräver ny kunskap vilket kan generera behov av utbildning.

4.1.4 Ethernet

Avsnittet bygger på Osbeck, Styr- och övervakningssystem [2].

Ethernet utvecklades på 1970-talet och är en standard för hur seriell datorkommunikation via kabel ska ske i ett lokalt nätverk. Idag är Ethernet klart dominerande och används primärt av företag som vill ha ett gemensamt nätverk mellan många datorer och är mycket vanligt förekommande inom industriell automation. Accessmetod (åtkomstmetoden) hos Ethernet är ofta **CSMA/CD** (*Carrier Sense Multiple Access with Collision Detection*) vilket innebär att alla de anslutna enheterna delar på kabeln de är anslutna till. Enbart en dator i taget kan sända information via kabeln. För att fastställa om datorn kan sända information behöver den lyssna på kabeln och höra om det är ledigt. Är kabeln upptagen får datorn vänta. Skulle en kollision inträffa d.v.s. att två eller fler datorer samtidigt skickar information så avbryts sändningen och den som skulle ha skickat "datapaketet" måste efter en viss väntetid sända om det igen. Den slumptalsgenererade väntetiden dubblas efter att en kollision har skett.

4.1.5 EtherCAT

Avsnittet bygger på Ten years of EtherCAT. [6] och EtherCAT av ETG [7].

Ethernet for Control Automation Technology (EtherCAT) är ett öppet högprestanda fältbuss-system baserat på Ethernet.

Fältbussar är idag en väletablerad del i industriell automation. Storskalig användning av PC-baserade styrsystem (d.v.s. en PC som agerar PLC) möjliggjordes av fältbussen och dess teknologi. Medan prestandan hos CPU (Central Processing Unit, en enhet som exekverar program i en dator) i styrsystem ökar i snabb takt har konventionella fältbussar tenderat att utgöra "flaskhalsar" som begränsar kapaciteten hos styrsystem.

Då en karaktäristisk egenskap hos nätverk inom automation är kort datalängd per nod (omkopplingspunkt för datatrafiken) blir den användbara andelen data väldigt låg om enskilda Ethernetramar används för varje cykel och inkopplad nod.

Olika angreppssätt har använts för att försöka höja realtidskapaciteten hos Ethernet, exempelvis har Accessmetoden CSMA/CD ersatts med polling (metoder för att kontrollera status hos en periferienhet, t.ex. ett I/O), andra förslag har varit att använda speciella switchar som distribuerar Ethernet paketen vid precisa tillfällen. Medan dessa olika lösningsalternativ kan transportera datapaketerna till de anslutna Ethernet-noderna mer eller mindre snabbt och korrekt, beror ändå tidsåtgången för omdirigering av data till och läsning från I/O starkt på tillämpningen.

De begränsningar hos ovan beskrivna Ethernetlösningar har övertrumpats av EtherCAT då varje paket inte längre mottages, behandlas och kopieras i varje nod. Detta kallas "processing on-the-fly". Istället läser slavmodulen den data som adresseras till den medan ramen passerar igenom noden. På liknande vis hanteras indata, d.v.s. datan läses in medan ramen passerar. Ramen försenas bara ett par nanosekunder och många noder kan adresseras med en enda ram. Detta kan jämföras med att åka tåg, men med skillnaden att passagerare hoppar av och på de hållplatser (noder) som är adresserade medan tåget saktar ner tillfälligt. Jämför belastningen på järnvägsnätet om istället ett enda tåg skulle åka till en enda hållplats och släppa av och på passagerare. Datatrafiken i EtherCAT är dessutom av Duplextyp, vilket innebär att ramar skickas i båda riktningarna på samma gång.

Protokollet som EtherCAT använder är optimerat för processdata och transporteras direkt inuti Ethernetramen m.h.a. en speciell Ethertype (ett fält inuti en ram som informerar om vilket protokoll som ligger inkapslat i den användbara datan), detta kan innehålla flera EtherCAT telegram där varje telegram är avsett för en viss minnesarea.

Adressering kan ske i vilken ordning som helst då serien med data är oberoende av den fysiska ordningsföljden hos de i nätverket inkopplade Ethernetterminalerna. Kommunikation mellan slavar är möjligt samt broadcast (en slav skickar data till en annan) och multicast (en slav distribuerar data till många). I fall då maximal prestanda krävs och EtherCAT modulerna används på samma nivå i nätet som styrsystemet används direktöverföring av Ethernetramar.

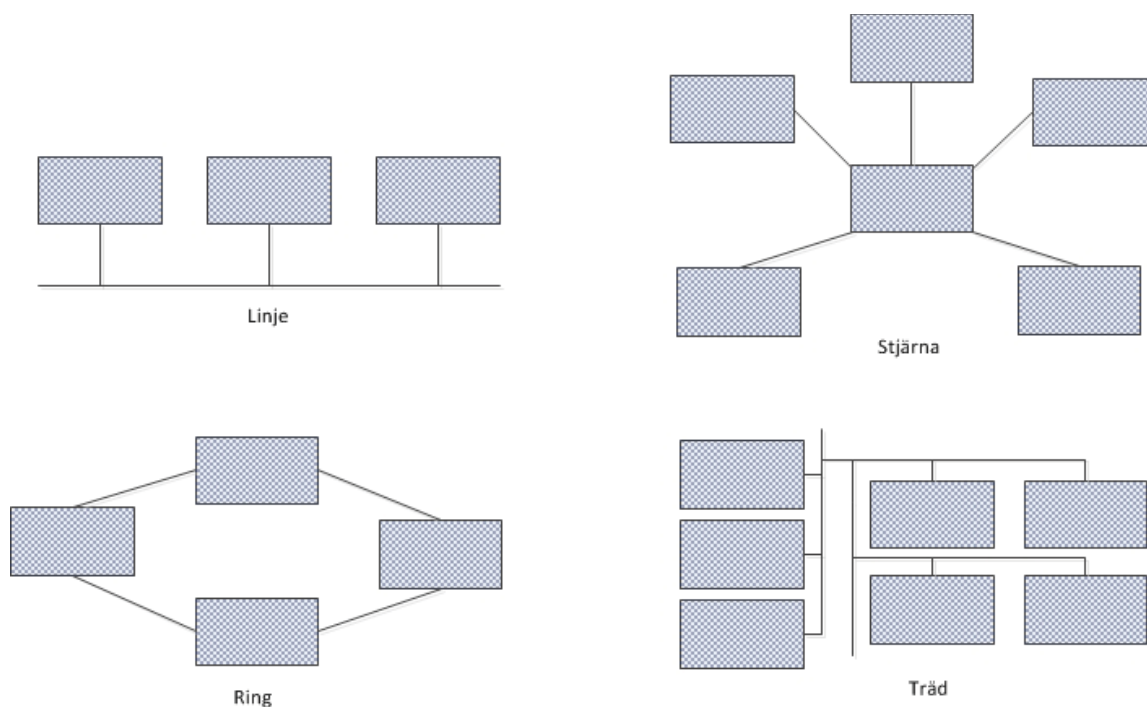
Tillämpningarna på EtherCAT är inte begränsade till ett underliggande nätverk, EtherCAT UDP förpackar EtherCAT-protokollet till UDP/IP (förbindelselöst protokoll i transportsiktet för att skicka datagram över ett IP-nätverk) datagram vilket öppnar för vilket styrsystem som

helst med en Ethernet protokollstack att adressera EtherCAT-system. Även kommunikation igenom routrar in till andra subnät är möjlig samt mellan styrsystem.

Gränssnittet mellan styrsystem och slav samt mellan slav och slav är identiskt. När det gäller kommunikationen mellan slavar finns två möjligheter; den första är när uppströms enheter kan kommunicera ytterst snabbt till nedströms inom en cykel. Då denna metod är beroende av topologin passar den särskilt bra vid sekventiella förlopp. För öppet konfigurierbar slav till slav-kommunikation finns en annan metod; att mastern vidarebefodrar datan. För denna metod krävs två cykler.

Standardramar (enligt IEEE 802.3: Carrier Sense Multiple Access with Collision Detection CSMA/CD) används av EtherCAT, dessa är ej avkortade. Följaktligen kan EtherCAT-ramar sändas från vilken Ethernet MAC (Media Access Control, en för nätverkskort unik identifierare) som helst, t.ex. en operatörspanel.

Ett nätverks topografi är hur enheterna i nätverket inbördes är kopplade till varandra. Stjärn-, träd-, linje-, eller ringtopologi supporteras av EtherCAT. Linjestrukturen (kallas ibland bussstruktur) som är associerad med fältbussarna är därmed åtkomlig för Ethernet. Kombinationen av linje och trädtopografin är särskilt användbar då gränssnittet för denna redan finns i många I/O-enheter vilket gör att inga extra switchar behövs. Den traditionella switchbase-erade stjärntopografin kan också användas.



Figur 4.3 Nätverkstopografi, olika sätt att bygga upp nätverk.

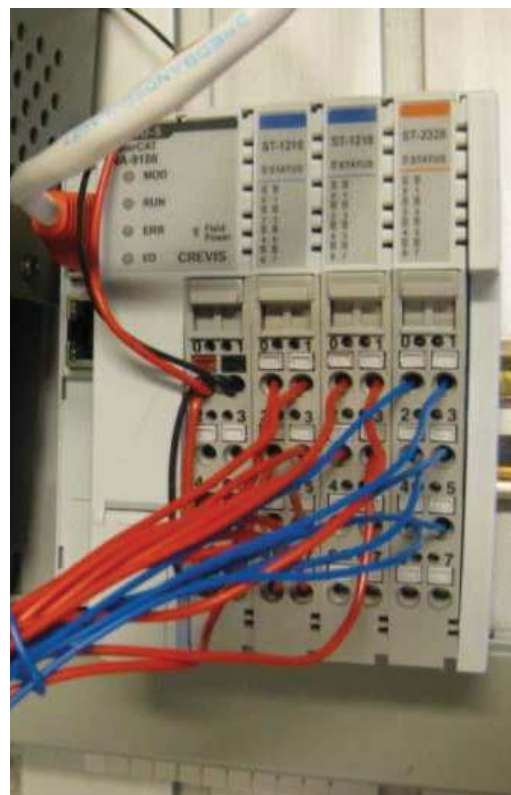
Flexibilitet vid elinstallation är underlättad genom valfrihet mellan olika kabeltyper, standard Ethernetkablar eller olika typer av optiska fiber eller kopparkablar kan användas tillsammans med switchar eller mediumkonverterare (en anordning som möjliggör för två olika typer av medium i kablar att kopplas ihop).

Målet med utvecklingen av EtherCAT var att applicera Ethernet inom industriell automation där kravet är korta cykeltider med mindre jitter (av synkroniseringskäl) vid kommunikationen samt låga hårdvaru- och installationskostnader. Jitter är inom elektroniken en eller flera icke önskvärda variationer av en signals karaktär eller en störning i signalen som kan påverka signalens kvalitet, exempelvis fas, amplitud eller pulsintervall. EtherCAT har gemensam utvecklingshistoria med Lightbus då EtherCAT i princip är Lightbusteknologi anpassad till Ethernet. Företaget Beckhoff lanserade 1989 sin första fältbuss, avsedd för maskinella styrsystem med krav på hög hastighet och till denna utnyttjades optiska fiberkablar. Bussen kallades för Lightbus och dess prestanda var redan då ett resultat av konceptet "processing on-the-fly" som även senare kom att utnyttjas av EtherCAT. Projektet började som "Fast Lightbus" men ändrades senare till EtherCAT och Beckhoff meddelade redan från start att intentionen var att utveckla teknologin till en öppen standard. På Automation- och industrimässan i Hannover 2003 presenterades EtherCAT för första gången och mässan blev en stor framgång för Beckhoff. EtherCAT har sedan dess blivit en etablerad standard i många industrier världen över. I slutet av 2003 grundades EtherCAT Technology Group (ETG) som är en internationell organisation av användare och säljare. ETG:s mål är att främja och erbjuda support till slutanvändare från olika branscher, tillverkare av automationsutrustning och leverantörer som använder EtherCAT och dess tillämpningar. Dessutom ger ETG ut information om EtherCAT, har teknik- och marknadskommittéer och anordnar utbildningar.

4.1.6 Decentraliserade input/output-enheter från Crevis

Följande avsnitt bygger på Input/Output [8] samt Styr- och övervakningssystem [2]. Inom datakunskap avses med I/O (skrivs ibland io eller IO) kommunikationen mellan ett informationsbehandlingsystem (en PLC eller PC exempelvis) och omgivningen, t.ex. en annan dator eller PLC eller en människa som matar in data via en knappsats. Men insignal (input) menas de signaler som går in i systemet och utsignal är de signaler som systemet skickar ut. Med decentraliserat I/O menas en enhet som sitter processnära och samlar in insignal respektive skickar ut utsignal till processen och via en fältbuss skickas data till/från styrsystemet som sitter längre från processen, ofta i ett styrskåp.

I projektet användes Crevis EtherCAT-I/O:n som består av en mastermodul och tre stycken slavmoduler fästa på en DIN-skena. Två av slavarna användes för input och en till output. Slavmodulerna samlar ihop signalerna och mastermodulen "förpackar" informationen enligt EtherCAT (beskrivet i avsnitt 4.1.6). Systemet är expanderbart, d.v.s. det går att lägga till fler slavmoduler samt att utöka med A/D och D/A-omvandlingsenheter.



Figur 4.4 Master- och slavmoduler från Crevis. De med blå markering högst upp är ingångsmoduler och den rödmarkerade är en utgångsmodul.

4.1.7 Industriellt HMI

Texten i detta avsnitt bygger på Osbeck, Styr- och övervakningssystem [2].

HMI är en förkortning av engelskans *Human-Machine Interface* och är ett operatörgränssnitt där interaktionen mellan människa och maskin sker, ofta en operatörspanel med tillhörande knappsats. Syftet med användningen av HMI inom industrin är att operatörer effektivt ska kunna övervaka och ibland styra processer, som exempelvis larmhantering, antal producerade enheter, nivå-, tryck- och temperaturövervakning. En bra utformning av layouten i ett HMI kan innebära ergonomiska och arbetsmiljömässiga fördelar.



Figur 4.5 Operatörspanel från Beijer Electronics.

4.1.8 iX Developer 2.0

Avsnittet är baserat på iX Developer User's Guide [9].

iX Developer är en utvecklingsmiljö för SCADA-system baserad på Windows CE utvecklad av Beijer Electronics. SCADA står för *Supervisory Control And Data Acquisition* och är en typ av programvara som används inom industri- och fastighetsautomation för övervakning och styrning. Mjukvaran iX Developer används för att konfigurera PC-baserade styrda tillämpningar och operatörspaneler, d.v.s. att programmera grafiska bilder kopplade till en process. Det kan handla om en detaljerad bild av tryck, temperatur och flöden, hantering av larm kopplade till en process eller en mer övergripande bild av en fabriks status m.a.p. byggda enheter och råvarulager. En operatörspanel och en SCADA-programvara används tillsammans med ett styrsystem t.ex. en PLC, där PLC:ns roll är att styra processen och operatörspanelen att visa status.

Alla grundläggande funktioner som behövs för att konstruera en processbild finns i iX Developer. Programmeringsspråket är grafiskt och av "drag och släpp"-typ. Styrningen av vad som ska kopplas till objekten på skärmen görs via s.k. tags och dessa knyts i sin tur till processen genom att importera variabellistan från styrsystemet. För varje dynamisk visning ställs intervall då ett objekt ska visas eller röra sig på skärmen. Ett antal fördefinierade

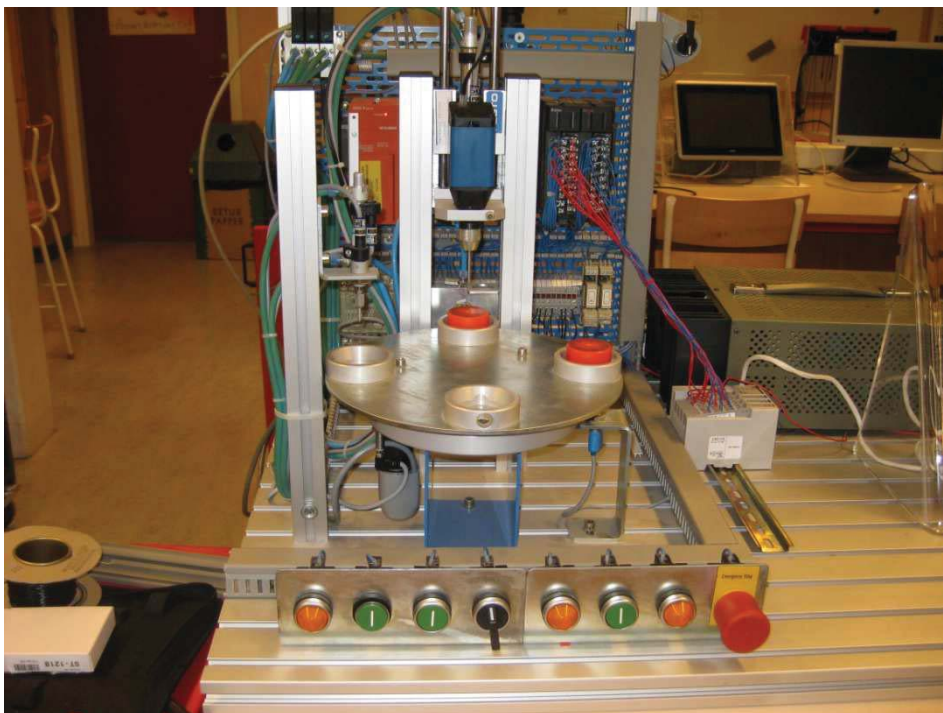
funktioner för larmhantering, reglage och grafiska visare finns tillgängliga men det går även att bygga egna funktioner.

4.1.9 Processen

I syfte att testa det nya styrsystemet och utvecklingsmiljön användes en enkel process (se figur 4.7). Processen som ska styras består av en elmotordriven roterande skiva som har fyra positioner ovanpå där arbetsstycken (kallas här puckar) kan placeras för borring och stansning. Stansens funktion är att kontrollera att ett hål har blivit borrarat. Fyra magneter sitter på skivans undersida och en givare reagerar på dessa när skivan är i rätt position. En pinne används för att kontrollera om positionen under borren innehåller någon puck genom att pinnen går ut och en givare känner av om pinnen nått sitt yttersta läge eller inte. När pinnen är i sitt yttersta läge så att givaren påverkas innebär det att ingen puck finns i B-positionen. Borret, stansen och pinnens linjärrörelser åstadkoms med pneumatiska cylindrar. Borren och stansen har givare för nedre och övre läget medan pinnen endast har givare för det yttre läget. Processen startar när ett vred ställs om från av- till påläge och startknappen på operatörspanelen tryckts in. Dessa funktionskrav sammanställdes i en kravspecifikation som sedan blev underlag till styrprogrammet.



Figur 4.6 Knappar för start och stopp av process.



Figur 4.7 Borrprocess.

Nr	Funktion	Krav
1	Vred av/på	När vredet slås (se figur 4.7) på och start tryckts ner på operatörspanelen (se grön knapp figur 4.6) ska processen starta och när vredet dras av eller stop trycks in på panelen ska processen stanna omedelbart.
2	Lokalisering av puck i position	M.h.a. givare på pinnen ska information erhållas, om det finns en puck eller inte.
3	Borrning och stansning	Borr och stans ska köras enbart när det finns puck i position under dem. Borr kör då skivan har stannat och det finns en puck i B-positionen och stans kör när skivan har stannat det finns puck i C-positionen.
4	Lägespositionering skiva	Skivan ska stanna då magnetgivaren är påverkad av någon av de fyra magneterna och förbli stillastående om position A och B är tomma. Skivan ska rotera då magnetgivaren är opåverkad, eller om position A eller B innehåller en puck.

Figur 4.8 Kravspecifikation för styrning av Borr- och stansprocess.

4.2 Nedladdning och installation av programvaror

Nedan följer de nedladdningar och installationer som gjorts under projektet.

4.2.1 Installation av CODESYS

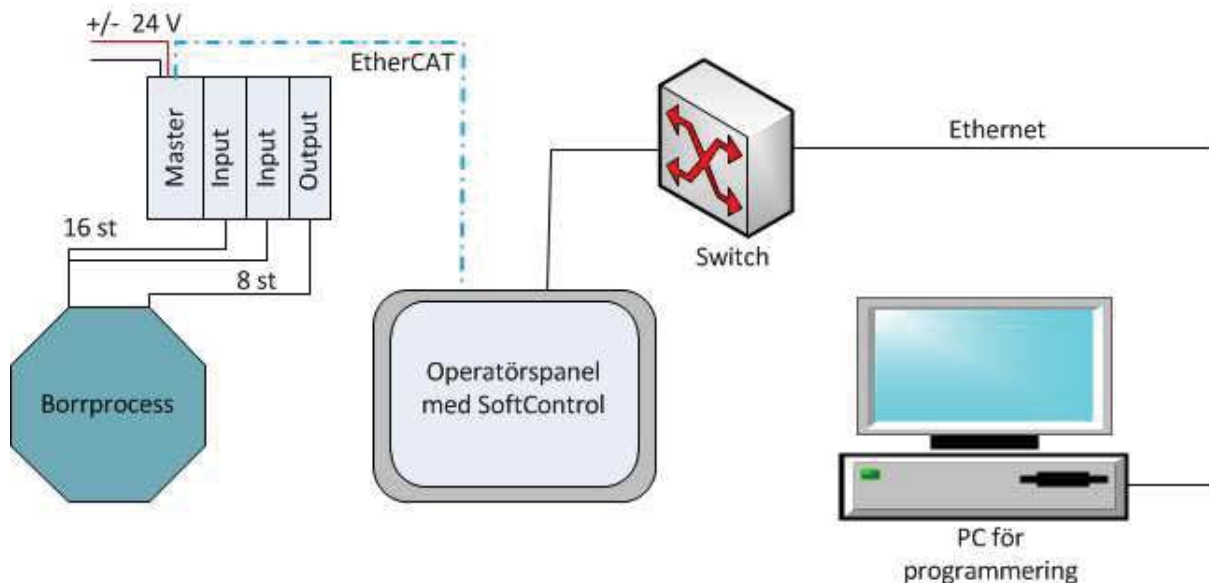
CODESYS och installationsmanual laddades ner och programvaran installerades. Ytterligare en installation av SoftControl d.v.s. drivrutiner för det integrerade styrsystemet i panelen gjordes samt inställningar och konfigurationer gjordes för att upprätta kontakt mellan SoftControl och CODESYS. För att lägga till fältbussen EtherCAT, en mastermodul samt slavarna öppnades ett programexempel och ett par ytterligare inställningar gjordes innan kommunikationen mellan processen och SoftControl fungerade. Detta finns beskrivet i kapitel 2 i Komma-igång-manual för CODESYS.

4.2.2 Installation av iX Developer

Programvaran erhöles på en USB-sticka och installationen startades automatiskt. Detta finns beskrivet i kapitel 2 i Komma-igång-manual för iX Developer.

4.3 Kommunikation med processen

I/O:ts mastermodul matades med 24 volt och ledare drogs mellan I/O:ts ut- och ingångar till processens plint, 16 st. ingångar och 8 st. utgångar (en ingång är här en insignal till PLC och en utsignal en styrsignal till processen). En GVL (global variabellista i CODESYS) skapades för samtliga in- och utsignaler som skulle användas. Genom att prova med knapptryckningar och försök med att skicka ut signaler till processen skapades en global variabel för varje in- och utsignal. Enkla program skapades och laddades ner för att lära känna process och program.



Figur 4.9 Schematisk bild av uppkoppling.

4.4 Programmering

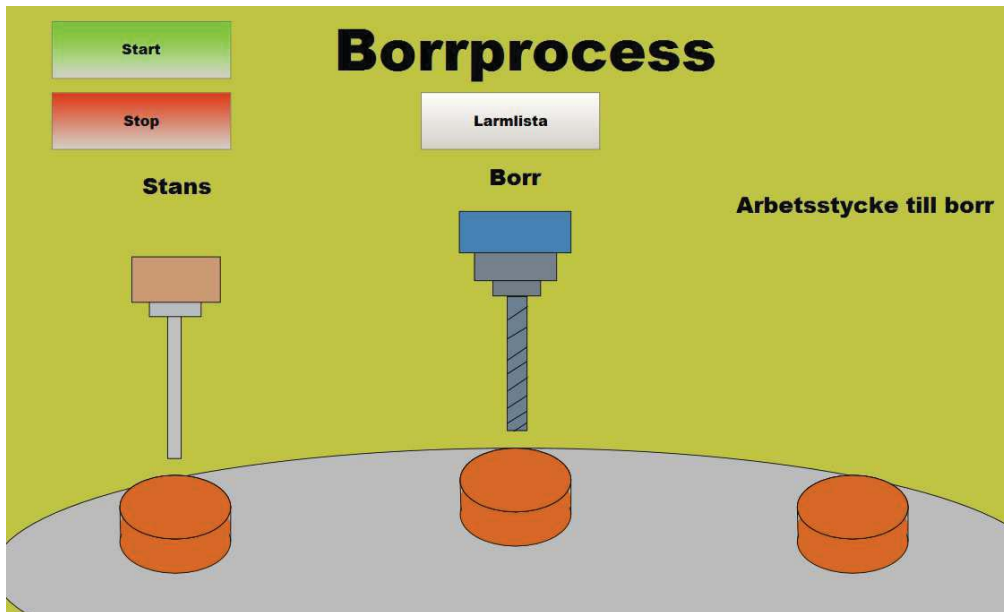
De fyra språken LD, ST, FBD och SFC skulle enligt frågeställningen användas för att styra processen och inslag skulle finnas av andrapartsutvecklade funktionsblock. Planen för arbetet var att skriva ett program i ett av språken, testköra och ev. förbättra detta och sedan att översätta detta till de andra språken och då få programmen så lika som möjligt. Syftet med att göra programmen lika var främst pedagogisk då de olika programvarianterna skulle visas i komma-igång-manualen för CODESYS som exempel och att det skulle var lättare att hänga med då koden kunde jämföras stycke för stycke av läsaren. Tyvärr visades det sig olämpligt och i vissa fall omöjligt att översätta "rakt av" och då har programmet skrivits på det sätt som var mest lämpligt. Eget funktionsblock skapades i form av ett program för tidtagning. Utförliga programexempel i språken LD, ST, FBD, SFC och CFC finns i Appendix 1.

4.5 Uppbyggnad av processbild i iX Developer

Två bilder byggdes i iX Developer för att styra och övervaka processen. En bakgrundsbild och de fyra positionerna för puckar skapades.

En processbild byggdes upp m.h.a. grundläggande figurer, borren och stansen byggdes i hop av rektanglar av olika storlekar som sedan sattes samman till ett objekt och taggades (d.v.s. kopplades till variabelistan i CODESYS).

Puckarna visas på skärmen när givarna har hittat en puck. De rör sig från höger till vänster i bilden och stannar under borr- och stanspositionerna, där borrning och stansning sker. Processen startas vid tryck på knappen "Start" och stannar vid "Stopp" och dessa två knappar används tillsammans med ett vred då programmeringsdatorn inte längre används för att starta/stoppa processen (kallas "stand alone").



Figur 4.10 Processbild uppbyggd i iX Developer.

Ytterligare en bakgrundsbild skapades och på denna en larmlista för visning och återställning av driftlarm. Driftlarmet styrs av antal borrhöjningar och då ett bestämt antal höjningar överskridits visas larmet på skärmen. Processen fortsätter att gå även om larmet är aktivt. Larmvariabeln kan återställas genom knappen "Larmåterställning", detta gör att larmet på skärmen blir inaktivt men ligger kvar i larmlistan. Larmraderna på listan tas sedan bort genom att de bekräftas ("Ack All") och sedan raderas ("Clear").

Knappar för byte av bild lades till så att det blev möjligt för användaren att förflytta sig mellan process- och larmbild när programmet laddats ner till panelen.



Figur 4.11 Larmbild uppbyggd i iX Developer.

Varabellistan från styrprogrammet i språket ST importerades och process- och larmbilderna provkördes. Utförliga programexempel finns i Appendix 2.

4.6 Manualskrivande

Nedan förklaras hur skapandet av manualerna gick till.

4.6.1 Komma i gång med CODESYS

Då styrprogrammet till borrarprocessen skrevs gjordes många erfarenheter som dokumenterades i syfte att använda dessa vid manualskrivandet. Arbetet började med att de viktigaste momenten under installationen dokumenterades samt att de viktigaste menyerna i programmen gick igenom och förklarades med bild och text. Vidare hur ett projekt skapas och den kunskap som behövs för att komma igång. Delar av programmet med förklarande text från styrningen av processen användes som exempel på hur de olika språken används. Även de tips som erhållits från Beijer inkluderades i CODESYS-manualen. Se Appendix 1 för installation och programexempel.

4.6.2 Komma i gång med iX Developer

Arbetet började med att de viktigaste menyerna i programmen gick igenom och förklarades med bild och text, samt hur ett projekt skapas och den kunskap som behövs för att komma i gång. Enkla programexempel skapades t.ex. att ändra bakgrundsfärg, lägga till text och att få ett enkelt objekt att ändra färg då en variabel ändrar värde från noll till ett. Vidare gavs exempel på hur en larmlista läggs till, vilka inställningar som kan göras till den för samt en knapp för nollställning av larmvariabeln (d.v.s. inläsning från skärmen). Se Appendix 2 för installation och programexempel.

4.7 Utredning av tillgång till funktionsblock på internet

Sökningar på internet gjordes och det fanns företag som hade lagt ut funktionsblock till kunder som kunde laddas hem, företaget Festo hade t.ex. på Support Portal [10] funktionsblock gjorda i CODESYS till styrning av deras automationsutrustning. Dessa gick bara att komma åt med lösenord, referens Festo Support Portal [8].

5 Resultat

Resultatet av arbetet presenteras nedan m.a.p. de mål som ställdes upp i avsnitt 1.3.

Mål	Resultat
Undersöka och presentera programvaran CODESYS och kommunikationslänken EtherCAT, deras historik, användning, utveckling och egenskaper.	Uppfyllt, beskrivs i 4.1.2 och 4.1.6.
Idrifttagning av SoftControl (styrsystemet som är integrerat i panelen) samt genomförande av ett styrprojekt där en bearbetningsprocess styrs från panelen.	Uppfyllt, beskrivs i 4.1.9, 4.2 till 4.4 och Appendix 1 kapitel 2.2 till 2.3 och 5.
Konfiguration av EtherCAT-kommunikation i nät med två noder och enkel processbild.	Uppfyllt, EtherCAT-konfigurationen beskrivs i Appendix 1 kapitel 2.4 och framtagning av processbild i huvudrapporten kapitel 4.5.
Kartlägga tillgång till andrapartsutvecklade funktionsblock.	Ej uppfyllt, beskrivs i kapitel 4.7.
Konfiguration av operatörspanel mot utvecklingsmiljö för styrsystemet.	Uppfyllt, beskrivs i 2.5.
Konfiguration av operatörspanelen mot Mitsubishi Q02-system.	Uppfyllt, finns beskrivet i kapitel 4 i Appendix 2.
Larmhantering i iX-miljö.	Uppfyllt, finns beskrivet i kapitel 4.5. och Appendix 2 kapitel 5.3.
Konstruera processbild för övervakning av processen i iX Developer	Uppfyllt, finns beskrivet i kapitel 4.5. och Appendix 2 kapitel 5.
Skriva komma-i-gång-manualer till framtida studenter för användning av CODESYS i iX-panelen, där konfigurationen av EtherCAT - kommunikationen förklaras och visas. Manualen ska innehålla exempel i språken LD,FBD,SFC och ST samt instruktion om skapande av egna funktionsblock.	Uppfyllt, se kapitel 4.6.1 och Appendix 1.
Skriva en komma-igång-manual för iX Developer för framtida studenter	Uppfyllt, se kapitel 4.6.2 och Appendix 2.

6 Slutsatser och diskussion

Arbetet påbörjades med en planeringsrapport med en grov tidsplan som också lades in i och jämfördes med projektets loggbok. Eftersom kunskap och tidigare erfarenheter m.a.p flera av de uppgifter som ingick i projektet saknades var det omöjligt att i förväg förutsäga tidsåtgången. I synnerhet tog installation, konfiguration och felsökning mycket mer tid i anspråk än vad som var förväntat. Tidplanen kom därför att nästan helt sakna betydelse. Information om var filer har hämtats fanns oftast på mail p.g.a. att svar gällande installationer och konfigurationer behövdes från tillverkaren. Detta ledde till att loggboken ansågs överflödig. En insikt av vad som kunde ha gjorts bättre är dokumentera speciellt installationen, de steg som ingår där och vilka filer som krävdes i varje steg redan från början på ett ställe, gärna i det som skulle utvecklas till Appendix eller huvudrapport.

Vårt generella intryck av CODESYS är bra, jämfört med den utvecklingsmiljö för PLC som vi tidigare haft erfarenhet av. Vi vet inte om vår kombination av hårdvara var extra krånglig, eller om andra kombinationer av PLC, HMI och I/O från olika fabrikat skulle innebära samma problem. Om vi skulle ge synpunkter och förslag på förbättringar till 3S-Smart Software Solutions och Beijer Electronics skulle de inkludera bl.a. följande:

- Företagen skulle kunna samarbeta bättre. Det skulle inte kosta mycket att göra ordet "CODESYS" sökbar på Beijers hemsida, eller tvärtom. Kompetensen när det gäller installation av olika hårdvarukombinationer finns hos Beijer (och förmodligen även hos 3S) och företagen skulle spara pengar på att göra sina sidor mer sökvänliga, eftersom supportavdelningen skulle belastas mindre. Även när man vet exakt vilken fil man söker efter på Beijers hemsida, kan man först behöva söka upp rätt sökfältet, eftersom det första sökfältet inte letar bland filnamn.
- När vi följde alla steg i de hjälpdokument vi fick fungerade installationen, men datorn själv borde kunna utföra samma steg. En "guide" skulle kunna fråga vilken hårdvara man vill använda i sitt system, och bara två minuter senare skulle användaren kunna lämna datorn, som sköter resten av installationen. En sådan guide kostar givetvis att utveckla, men många användare skulle spara mycket tid. Dessutom behöver inte alla tänkbara kombinationer av hårdvara ingå, men båda företagen kunde vara tydligare med hur deras produkter fungerar i kombination med produkter av andra fabrikat.
- Hjälpsystemet i CODESYS kan förbättras. Det kan ta lång tid öppna denna hjälp, men tiden skulle minska kraftigt om man kommer till en söksida, så endast de hjälpavsnitt man sedan efterfrågar laddas in i minnet. Många hjälpavsnitt kunde ha mer bilder – sidan med "Introduction and Basic Concepts" har t.ex. inga. Programmet har en startsida, som visas när programmet öppnas, men varför inte utnyttja lite av denna yta till hjälplänkar? Att det finns pedagogiska och användbara hjälpfilmer på Youtube med tusentals "views" visar att många användare föredrar detta.
- Kodytan i programmet borde kunna ses i helskärm, om man tillfälligt vill ha överblick över en större del av koden. Speciellt gäller detta FBD, där ett network kan bli ganska stort på skärmen även om det inte uträttat speciellt mycket. Att zooma ut till 50% är enkelt, men då blir variabeltexten nästan oläslig.
- Vissa menyfunktioner fungerar dåligt, eller mycket opedagogiskt, t.ex. automatisk konvertering mellan programspråk, användning av *breakpoints* m.m.
- Vid byggfel i CODESYS var felmeddelandena svårlästa, långa meddelanden där bara hälften av texten var synlig.

- I iX Developer då "hemmabyggda" symboler som t.ex. vår borrh skulle kopieras och läggas till i ett nytt projekt blev borren låst i en viss position på arbetsytan i iX och gick inte att flytta. Vi blev tvungna att plocka isär borren igenom "ungroup" och sedan gruppera i hop den igen. Detta var mycket osmidigt, att flytta symboler mellan projekt borde vara mycket lättare.
- I övrigt upplevde vi att iX Developer var ganska lätt att lära sig och att använda, när man förstått det grundläggande om hur tags ska användas så går det snabbt att bygga upp lite mer avancerade processbilder.

Referenser

- [1] Ingemar Wiklund, Göran Hult och Osbeck, Morgan. Styrprojekt vid elektro- och mekatronikingenjörsprogrammet. Kompendium i kurs LEU 075, Chalmers tekniska Högskola, 2012.
- [2] Osbeck, Morgan. Styr- och övervakningssystem. Kompendium i kurs LEU 340, Chalmers tekniska Högskola, 2014.
- [3] Beijer Electronics. CoDeSys Control V3 Manual.
Websida (http://www.beijer.se/web/web_se_be_se.nsf/alldocuments/7249AD761B68C652C125798E004AB5FC 2014-05-20)
- [4] Wikipedia CODESYS (<http://en.wikipedia.org/wiki/CODESYS> 2014-04-16)
- [5] Wikipedia. IEC 61131-3. Websida (http://en.wikipedia.org/wiki/IEC_61131-3 2014-05-14)
- [6] Gomez, Kevin. Ten years of EtherCAT. Reed Business Information Pty Ltd, a division of Reed Elsevier Inc, Apr 2013.
- [7] ETG (EtherCAT Technology Group).
Websida (<http://www.ethercat.org/en/technology.html> 2014-05-02)
- [8] Wikipedia. Input/Output. Websida (<http://en.wikipedia.org/wiki/Input/output> 2014-05-04)
- [9] iX Developer User's Guide
(http://beijerinc.com/pdf/iX_Developer_2_1_User_Guide_MAEN832F.pdf 2014-05-06)
- [10] Festo Support Portal. Websida (http://www.festo.com/net/en-gb_gb/SupportPortal/default.aspx?q=%22codesys%22+OR+%22PositioningDrives%22+OR+%22FCT%22+OR+%22CPX-FMT%22&tab=5 2014-05-18)

Appendix 1

PLC-programmering i CODESYS

Komma-igång-manual

Innehåll

1 Inledning.....	I
2 Installation och konfiguration	II
2.1 Systemkrav och mjukvara	II
2.2 Installation av CODESYS.....	III
2.3 Konfiguration av SoftControl	III
2.4 Konfiguration av Crevis EtherCAT-enheter	VI
2.5 Konfiguration av iX Developer och CODESYS.....	X
2.6 Skapa tillgång till variabellista i iX Developer	XI
3 Översikt av CODESYS-miljön.....	XIII
3.1 Skapa ett projekt.....	XIII
3.2 Kör ett program.....	XIV
4 Menyerna och funktionsblock	XV
4.1 Menyerna i CODESYS.....	XV
4.2 Funktionsblock.....	XXI
5 Exempel på programmering i CODESYS	XXVIII
5.1 Deklaration av variabler	XXVIII
5.2 Programkod för borrarprocess.....	XXX
5.3 Programkod i ST, FBD och LD.....	XXXI
5.3.1 Sekvens Reset	XXXI
5.3.2 Sekvens STP (Skiva till position)	XXXII
5.3.3 Sekvens PU (Pinne ut)	XXXIV
5.3.4 Sekvens PI (Pinne in)	XXXVI
5.3.5 Sekvens BSN (Borr och stans ner)	XXXVIII
5.3.6 Sekvens BSU (Borr och stans upp)	XL
5.3.7 Sekvens SUP (Stans ur position).....	XLI
5.4 Olika sätt att programmera.....	XLIII
5.5 Programkod i SFC	XLVI
5.6 Att göra egna funktionsblock.....	XLVIII
5.7 Programkod i CFC	LI
Referenser	LIII

1 Inledning

Denna manual vänder sig till nybörjare och kräver ingen förkunskap mer än allmän datorkunskap och kunskap i grundläggande digitalteknik och programmering.

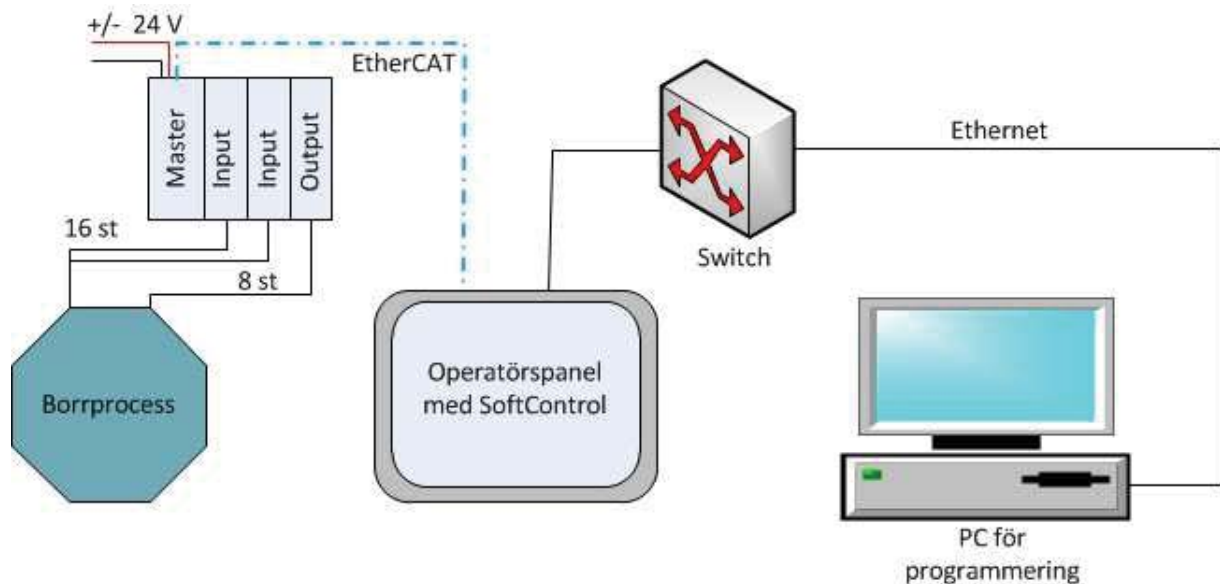
CODESYS är en öppen mjukvaruplattform för PLC-programmering. Denna manual bygger på CODESYS v 3.5 SP4 Patch1, operatörspanelen T12B, Crevis EtherCAT NA-9186 I/O:n. Både installation och användning utgår från denna utrustning. Andra versioner av mjukvara eller annan hårdvara kan avvika på olika punkter från nedanstående beskrivning.

När ett program är gjort i CODESYS överförs detta till det inbyggda styrsystemet i operatörspanelen. Styrsystemet fungerar fristående och när användaren är nöjd med programmet kan datorn med CODESYS kopplas bort. Även operatörspanelen är fristående från programexekveringen, men i praktiken vill man oftast ha möjlighet att styra och/eller övervaka processen och därmed är det bekvämt att ha styrsystem och operatörspanel i samma fysiska enhet.

Manualen börjar med en installationsanvisning och behandlar därefter användning av CODESYS-miljön, att skapa och köra program och att använda menyer och funktionsblock. Slutligen görs en genomgång av språken ST, FBD, LD, SFC och CODESYS:s egna språk CFC. Det assemblerliknande språket IL behandlas inte.

2 Installation och konfiguration

Hårdvaran som nämns i kapitel 1 kopplas enligt figur 2.1. Borrprocessen till vänster är en testprocess för att provköra CODESYS och iX Developer.



Figur 2.1. Uppkoppling av hårdvara. Operatörspanelen har två Ethernetportar, varav LAN A (höger) används för att ladda ner program och operatörsbilder från datorn, och LAN B (vänster) kopplas till EtherCAT-systemet.

2.1 Systemkrav och mjukvara

Denna manual handlar om installation och programmering i CODESYS och CODESYS:s interaktion med iX Developer. iX Developer är en utvecklingsmiljö för processbilder i operatörspaneler som också behandlas i en separat komma-igång-manual (Appendix 2). På operatörspanelen kan vissa inställningar göras direkt på skärmen, t.ex. att radera nerladdade projekt, men detta behandlas inte i denna manual.

Följande krävs av datorn som CODESYS ska installeras på:

- Windows 2000 eller senare.
- 512 MB RAM-minne, 1 GB rekommenderas.
- 200 MB ledigt hårddiskutrymme, 1 GB rekommenderas.
- Pentium V / Centrino på minst 1,8 GHz eller Pentium M på minst 1 GHz.

Följande krävs av datorn som iX Developer ska installeras på:

- Microsoft Windows 7, Microsoft Windows Vista eller Microsoft Windows XP SP3.
- Minst 2 GB RAM-minne.
- Processor på minst 2 GHz.
- Grafikkort: Tier 2, DirectX version: 9.0 eller senare, Video RAM: 120 MB eller större,
- Pixel shader: version level 2.0 eller högre, Vertex shader: version level 2.0 eller senare och Multitexture units: 4 eller mer.

Under installationsprocessen används dessutom följande filer, som finns tillgängliga på USB-minne.

- "Setup_CODESYSV35SP4Patch1.exe" [1]
- "KI00328B.pdf" [2]
- "iX_TxB_SoftControl.devdesc.xml" [3]
- "ProgExample_TxB_SC_EtherCAT_KI00329.zip" [4]
- "TxB_SoftControl_Setup_iX_2.0.msi" [5]
- "CoDeSys_SoftControl_Direct_Access_Pre2.mpd" [6]
- "KI00329.pdf" packas upp ur [4]
- Installationsfil för iX Developer 2.0 SP1 [7].

2.2 Installation av CODESYS

CODESYS installeras i datorn genom att filen "Setup_CODESYSV35SP4Patch1.exe" [1] körs. Programmet guidar användaren genom installationen. Behåll rekommenderade inställningar och lämna förbockade rutor oförändrade.

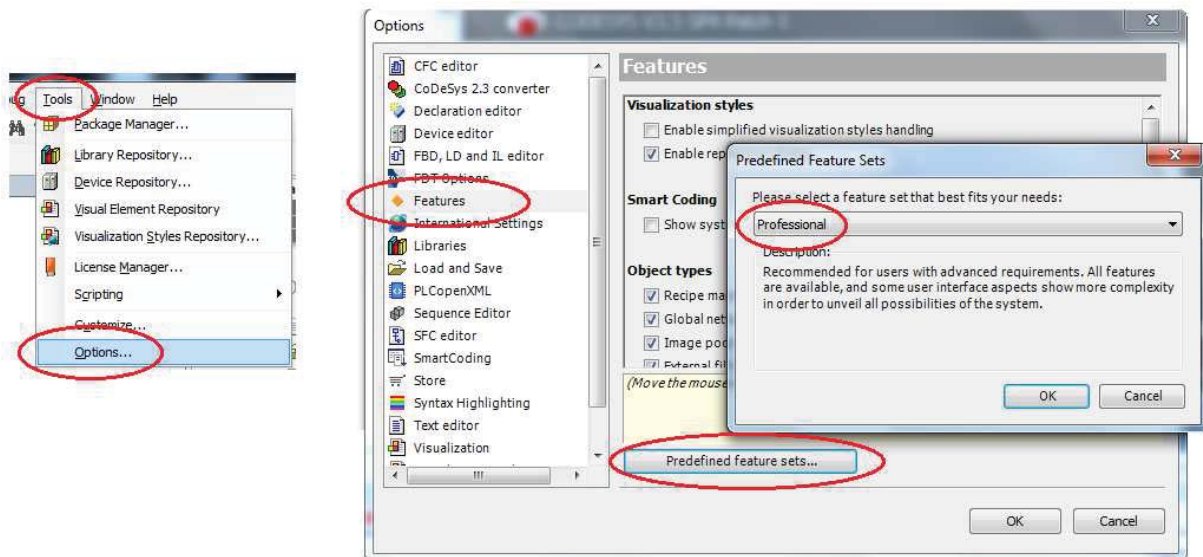
Efter installation kan programmet startas från en ikon på skrivbordet eller från startmenyn (sök på "CODESYS"). Första gången programmet körs får användaren välja "**environment settings**". Välj **Standard** och tryck på **Start**. Denna inställning kan ändras senare i Project Options.

2.3 Konfiguration av SoftControl

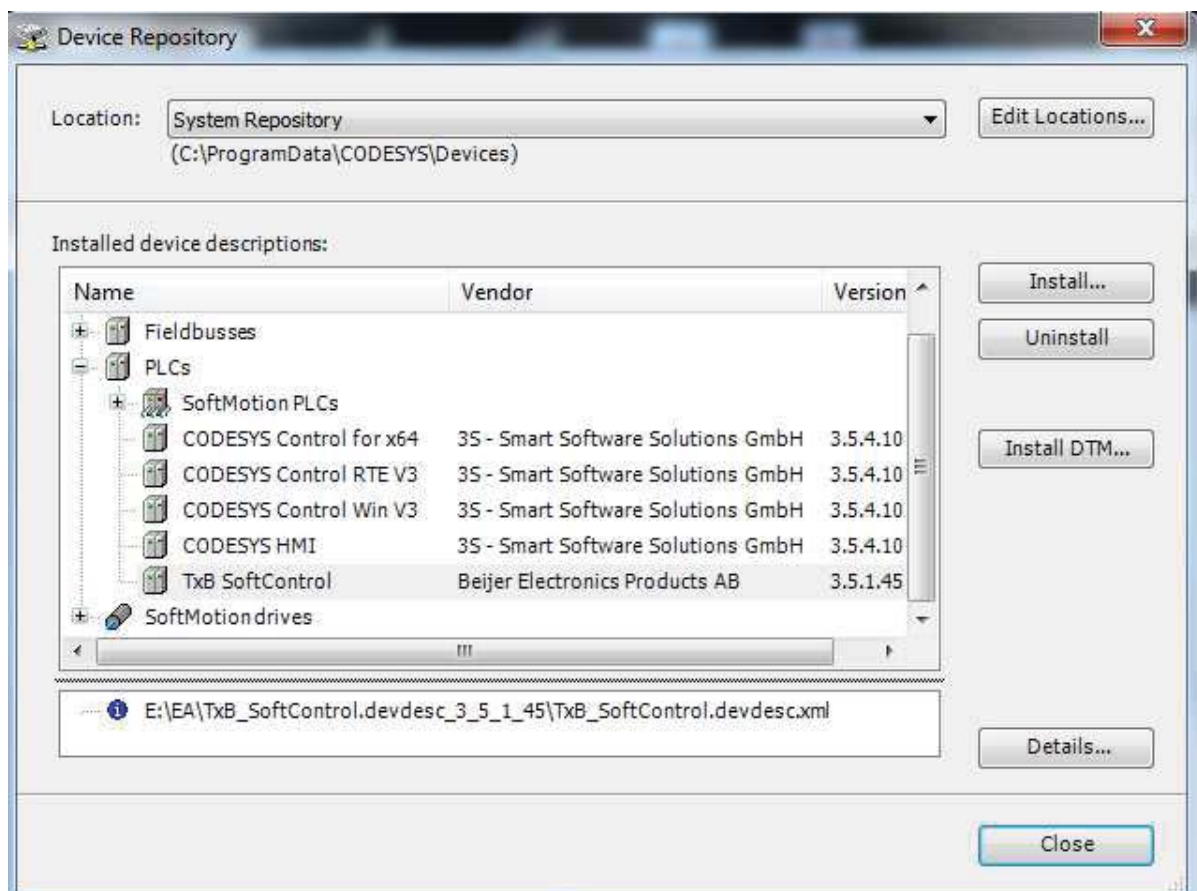
Denna instruktion är baserad på filen "KI00328B.pdf" [2] och användning av mjukvaror enligt avsnitt 2.1:

Starta CODESYS från ikon på skrivbordet, startmenyn etc. och följ följande steg:

- Välj **Tools / Options / Features** och klicka på **Predefined feature sets...**
- Ändra från "**Standard**" till "**Professional**", se bild.



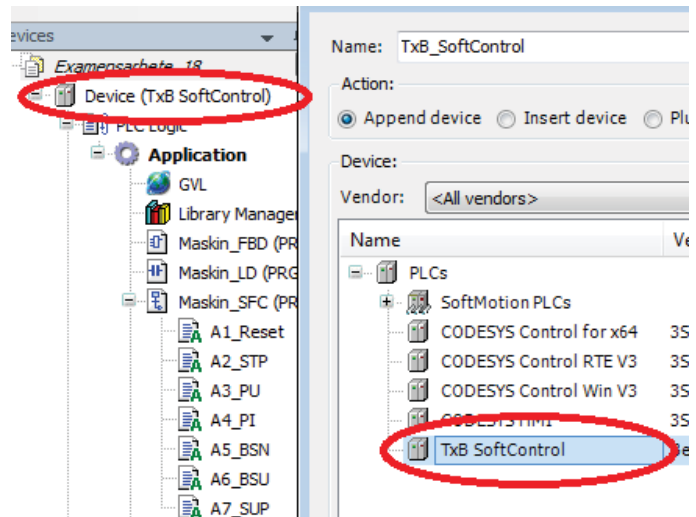
- Välj **Tools / Device Repository / Install...**
- Välj filen "iX_TxB_SoftControl.devdesc.xml" [3]. Denna fil innehåller information som möjliggör kommunikation mellan CODESYS i utvecklings-PC:n och SoftControl i iX-panelen genom att en "device", d.v.s. en enhet (SoftControl i detta fall) kan läggas till i projektet. Resultatet ska vara enligt bilden.



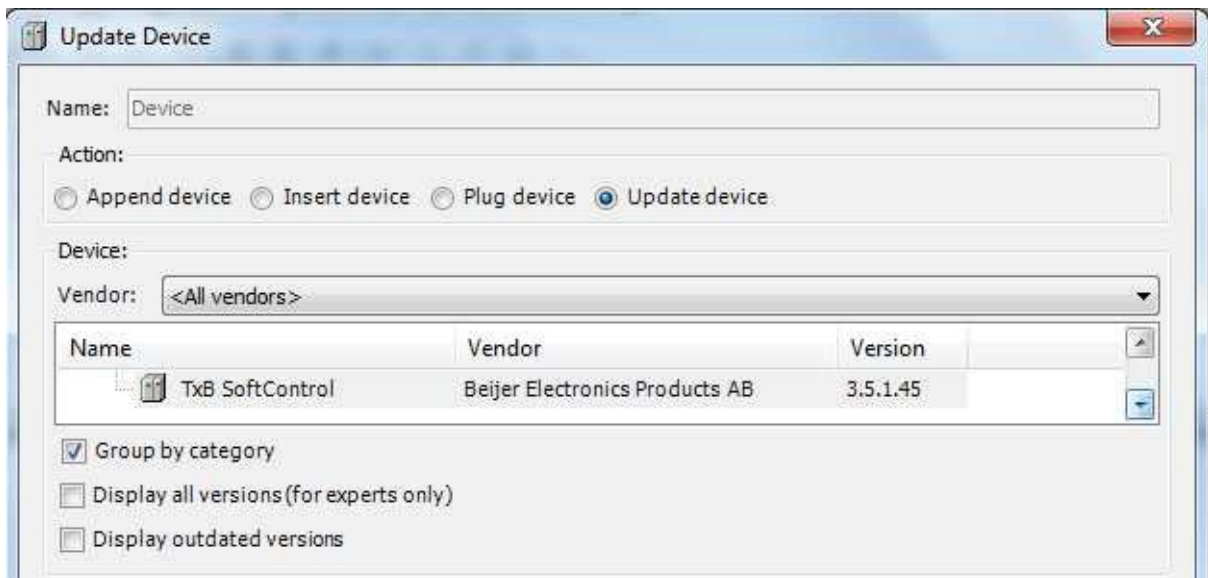
Högerklicka på Device näst högst upp trädstrukturen och välj add device. I rullgardinen Vendor leta upp Beijer Electronics Products AB och klicka i rutan display all versions (for

experts only) under det vita fönstret. Leta upp TxB SoftControl 3.5.1.45 och markera genom att klicka. Tryck sedan på **Add Device** längst ner.

Kontrollera enligt bilden till höger att TxB SoftControl ligger under PLCs i dialogrutan och att andra nivån i trädstrukturen är **Device (TxB SoftControl)**. Annars gör följande (se nästa bild):



- Högerklicka på **Device** i trädstrukturen.
- Välj **Update Device**.
- Leta upp TxB SoftControl under PLCs och klicka på **Update Device**.

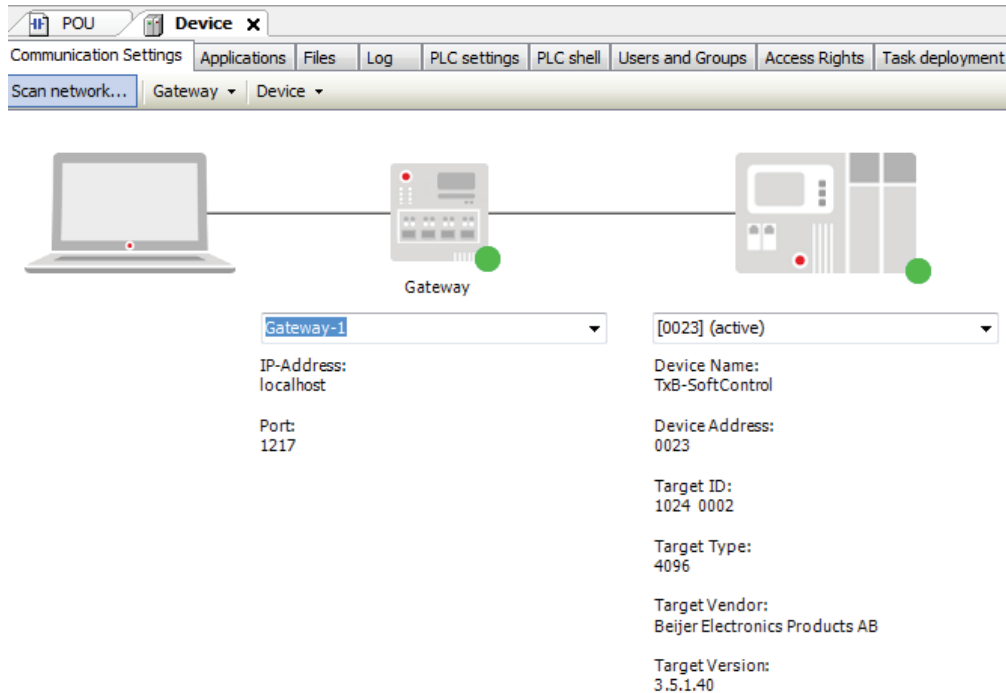


Kontrollera att datorn har kontakt med SoftControl:

- Ta reda på ip-adressen för SoftControl. Den visas på operatörspanelen när denna startas (eller startas om genom att tillfälligt koppla från spänningsmatningen).
- Kör "cmd" i PC:n t.ex. från startmenyn och skriv "ping <ip-adress för SoftControl>" (Programmet ping.exe ligger ofta i "c:\Windows\System32".)
- Om kontakt saknas, testa att ändra nätverksinställningarna i PC:n. I Windows 7 och 8 hittar man dem på "Kontrollpanelen \ Nätverk och Internet \ Nätverks- och delningscenter". Klicka på aktivt nätverk och välj **Egenskaper**. Välj "Internet Protocol Version 4". Klicka på **Egenskaper** och mata in ip-adressen för SoftControl. Testa därefter "ping" igen.

När kommunikationen fungerar:

- Dubbelklicka på **Device** i trädstrukturen.
- Om någon av cirkarna vid Gateway respektive SoftControl (se bild nedan) inte är fyllda med grön färg, testa att klicka på denna cirkel eller välj **Scan network...** Båda cirkarna måste vara gröna för att kommunikationen ska fungera.



2.4 Konfiguration av Crevis EtherCAT-enheter

- Starta CODESYS och skapa ett standardprojekt.
- Extrahera följande arkiv till valfri plats på hårddisken: "Project_TxB_EtherCAT_NA9186 \ CoDeSys \ T7B_SC_NA9186_Ex1.projectarchive" från "ProgExample_TxB_SC_EtherCAT_KI00329.zip" [4]. Alla rutor under "Items" ska vara förbockade.
- Stäng projektet.
- Välj **File / New project / Standard project**.
- Välj namn och plats för projektfilen och klicka på **OK**.
- Välj att starta programmet med "Ladder Logic Diagram (LD)".
- Högerklicka på **Application** i trädstrukturen till vänster på skärmen.
- Välj **Add Object / POU**.
- Välj typen "Program" och språket "LD". Klicka på **Add**.

Nu ska EtherCat Master installeras.

- Högerklicka på **Device** i trädstrukturen och välj **Add Device**.
- Välj i EtherCat Master version 3.5.1.0. (Om inte versionen

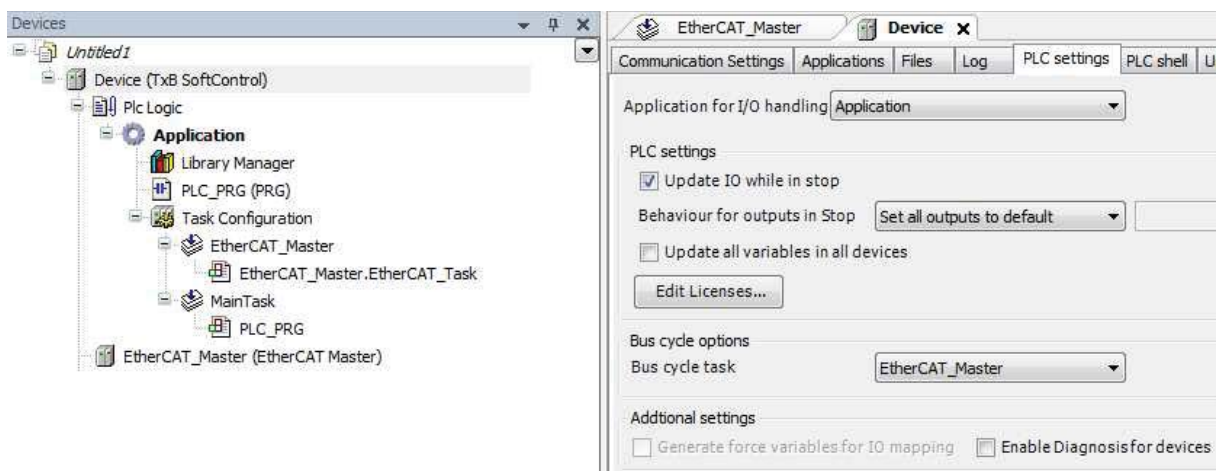
*Observera att installationen av EtherCat är kopplat till projektet och försvinner om man öppnar ett nytt projekt med **File / New Project...** Det enklaste sättet att få med dessa inställningar är att spara projektet med installationen och använda detta som mall. När ett nytt projekt ska skapas används då (**File / Open Project...**).*

syns kan man välja "Display all versions (for experts only)".

- Klicka på **Add Device** och **Close**.

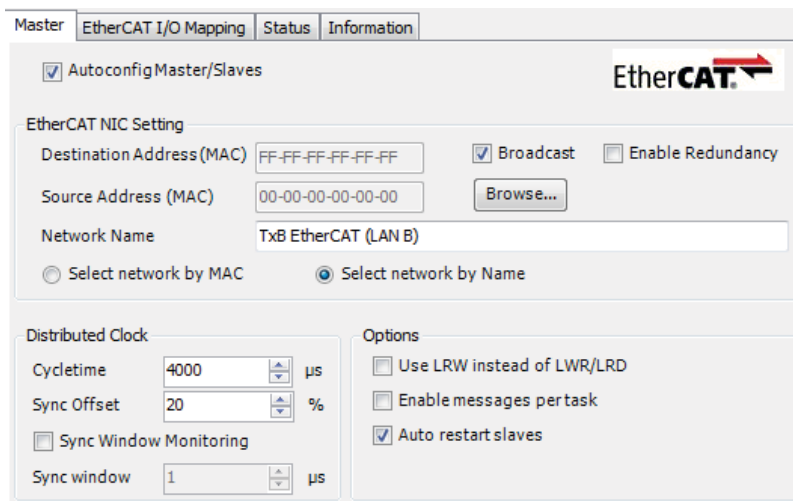
EtherCat Master ska nu ligga i trädstrukturen som en Device som symboliserar hårdvaran, enligt bilden nedan. Dessutom skapas en Task med samma namn under Task Configuration. Under denna task läggs de programdelar (POU) som ska vara med när man bygger och laddar ner projektet. MainTask kan tas bort. Program som ligger under MainTask kommer inte att laddas ner till SoftControl.

- Dubbelklicka på **EtherCat Master** i **Task Configuration**.
- Välj typen **Cyclic** och intervallet 4000 µs.
- Dubbelklicka på **Device** i trädstrukturen och välj fliken **PLC settings**.
- Gör inställningar enligt bilden.



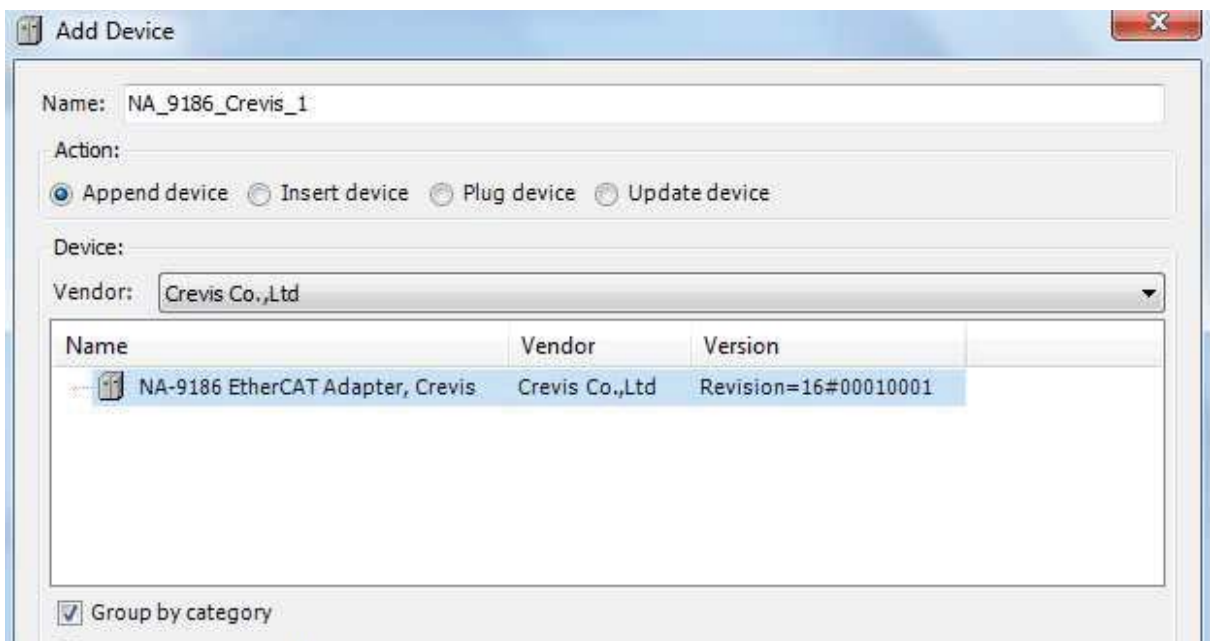
Konfigurering av EtherCAT-kopplingen i CODESYS:

- Välj **Tools / Device Repository / Install...**
- Installera filen "Crevis_EtherCAT_V1.000.xml" från "ProgExample_TxB_SC_EtherCAT_KI00329.zip" [4]
- Dubbelklicka på **EtherCAT_Master** under **Device** (längst ner i trädstrukturen) och ställ in enligt bilden nedan. Funktionen "Auto restart slaves" innebär att noderna startas om efter ett kommunikationsfel.
- Om "Source Address (MAC)" i samma fönster visas som 00-00-00-00-00-00, välj **Browse...** för att erhålla en MAC-adress i stället för nollorna.



Installation av Slavnod:

- Högerklicka på **EtherCAT Master** under **Device** och välj **Add Device**
- Välj Crevis som "Vendor" och markera "NA-9186 EtherCAT Adapter, Crevis".
- Klicka på **Add Device** och **Close**.

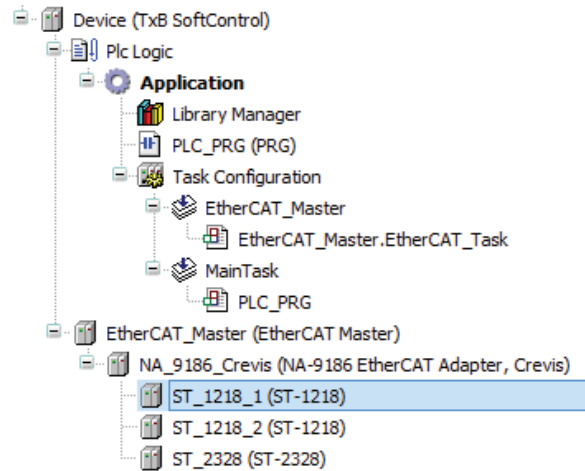
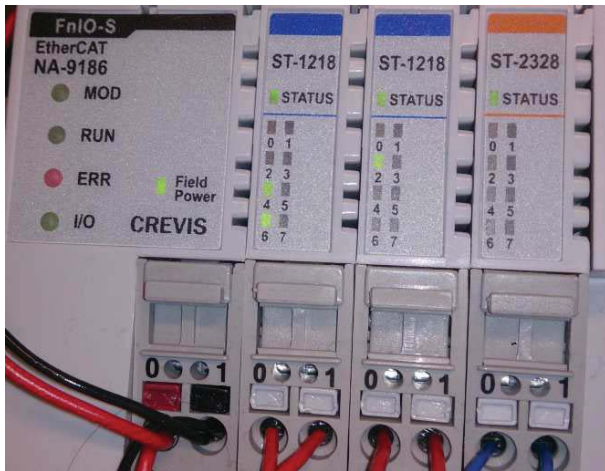


Den nya enheten, **NA_9186_Crevis**, ska nu ligga som en Device under **EtherCAT Master**.

Installation av slavnoder till NA-9186:

- Högerklicka på **NA_9186_Crevis** och välj **Add Device**.
- Välj de moduler som ska användas genom att dubbelklicka, eller klicka på **Add Device**. Namnet på respektive slavnod står på noderna nära det färgade fältet, se bild nedan till vänster.

- När alla önskade slavnoder syns under enheten **NA-9186** enligt bilden nedan till höger, klicka på **Close**.

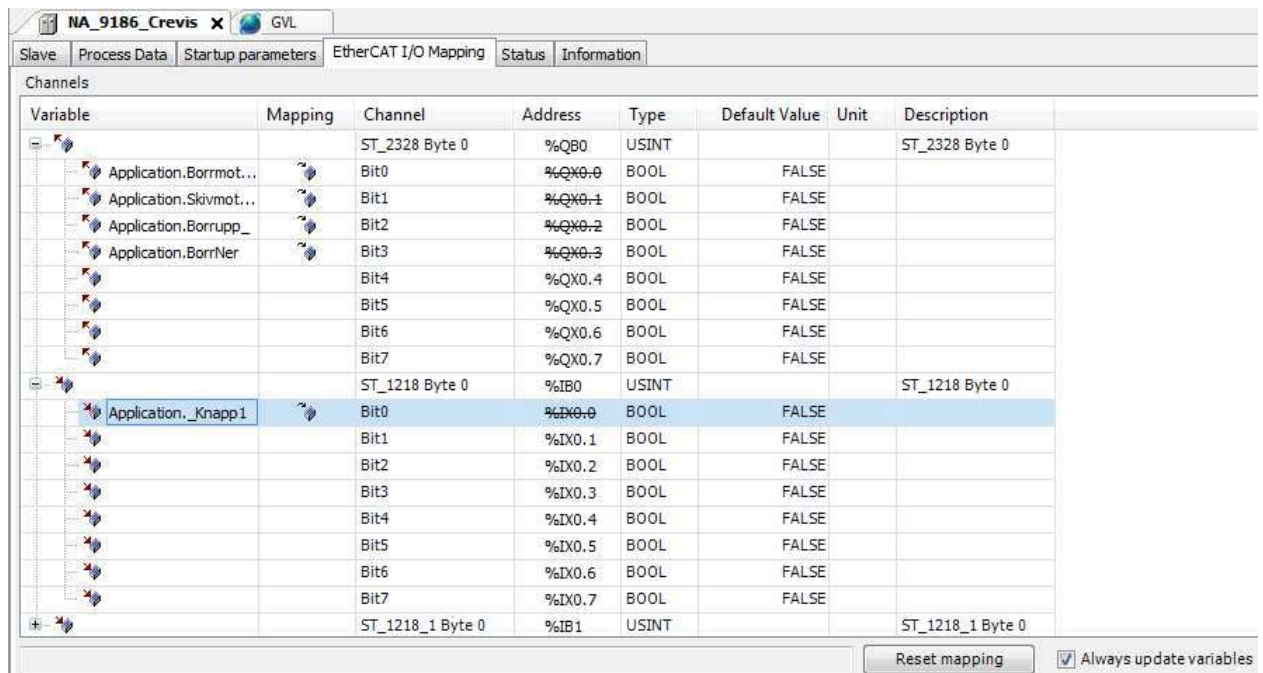


Global variabellista:

- Högerklicka på **Application**
- Välj **Add Object / Global Variabel List**
- Skriv "GVL" och klicka på **Add**.
- Skriv in de globala variabler som ska användas i listan.

EtherCAT-adresser i CODESYS:

- Dubbelklicka på **NA_9186_Crevis** och välj **EtherCAT I/O Mapping**. Se bild nedan.
- Markera **Always update variables** i nedre högra hörnet.
- Expandera en av slavmodulerna, t.ex. en ingångsnod genom att klicka på "+". Dubbelklicka på önskad bit (motsvarar en port längst ner på bilden ovan) och välj under **Application / GVL** namnet på den variabel som ska kopplas till denna bit. Variabelnamnet får i listan i nästa bild ett inledande prefix på "Application." Detta prefix används dock inte vid vanlig CODESYS-programmering.
- Om kopplingen mellan en bit och önskat variabelnamn fungerar visas en böjd pil som pekar på en box i kolumnen Mapping vid respektive bit.
- När alla önskade bitar är kopplade, gör motsvarande för nästa in- eller utgångsnod.



2.5 Konfiguration av iX Developer mot CODESYS

Följande konfiguration görs för att kunna tagga (d.v.s. kunna använda variablerna vid styrning av objekt) i iX Developer.

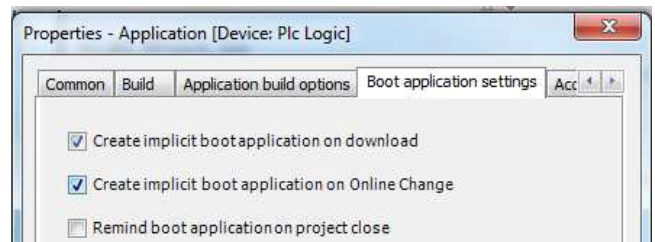
- Ladda ner och kör "TxB_SoftControl_Setup_iX_2.0.msi" [5]. Denna fil gör det möjligt att senare använda den globala variabellistan i CODESYS som taggar då processbilder byggs upp i iX Developer. (Filen behöver inte installeras då en extern PLC används till enbart skärmen, då används GVL som taggar i stället.)
- Högerklicka på programikonen för iX Developer (på skrivbordet, startmenyn etc.) och välj "Kör som administratör".
- Skapa ett nytt projekt för operatörspanel T12B, tryck sedan **Finish**.
- Klicka på det vita "pappret" i övre vänstra hörnet (fortfarande i iX Developer).
- Välj **Update Drivers** i rullgardinsmenyn som öppnas och välj om filen ska hämtas från internet eller annan källa.
- Öppna filen "CoDeSys_SoftControl_Direct_Access_Pre2.mpd" [6] och klicka på **Install**.
- Starta om iX Developer.



Installationen är inte knuten till det tillfälliga projektet ovan och behöver alltså inte göras mer än en gång.

För att ett PLC-projekt ska finnas kvar i SoftControl efter omstart måste dessutom följande inställning göras:

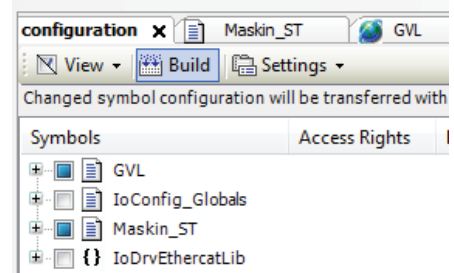
- Öppna CODESYS och önskat projekt.
- Högerklicka på **Application** i trädstrukturen. Välj **Properties... / Boot application settings**.
- De två övre rutorna ska vara förbockade enligt bilden.



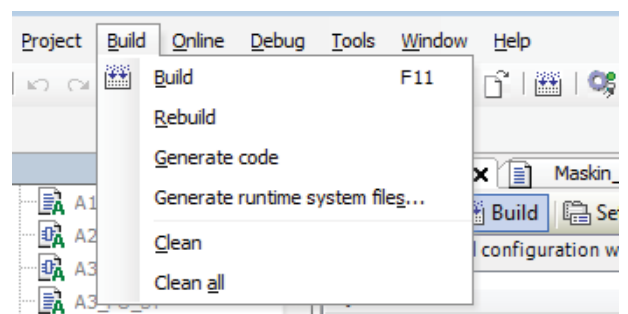
2.6 Skapa tillgång till variabellista i iX Developer

För att skapa tillgång till variabellistan i CODESYS i iX Developer, gör så här:

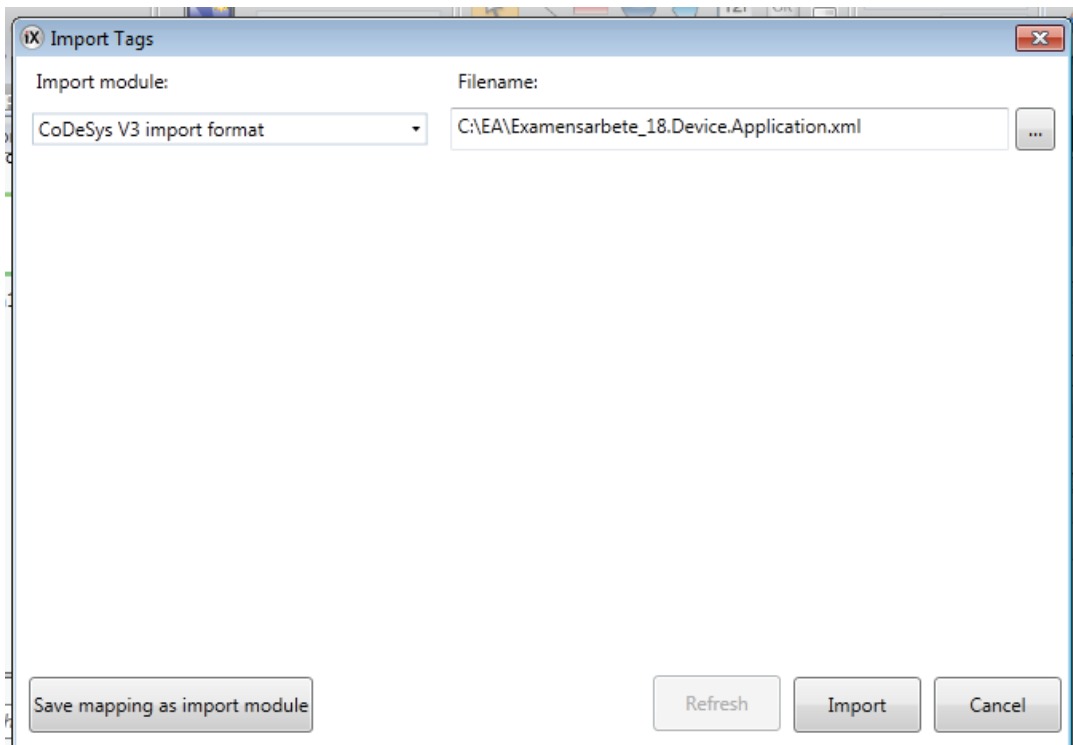
- Högerklicka på **Application** i trädstrukturen i CODESYS.
- Välj **Add Object / Symbol configuration**.
- Välj ett namn och klicka på **Open**.
- Under fliken Symbol Configuration kan val vilka variabler som ska finnas tillgängliga att tagga i iX Developer, både från globala variabellistan (GVL) och de lokala variablerna (Maskin_ST i bilden).



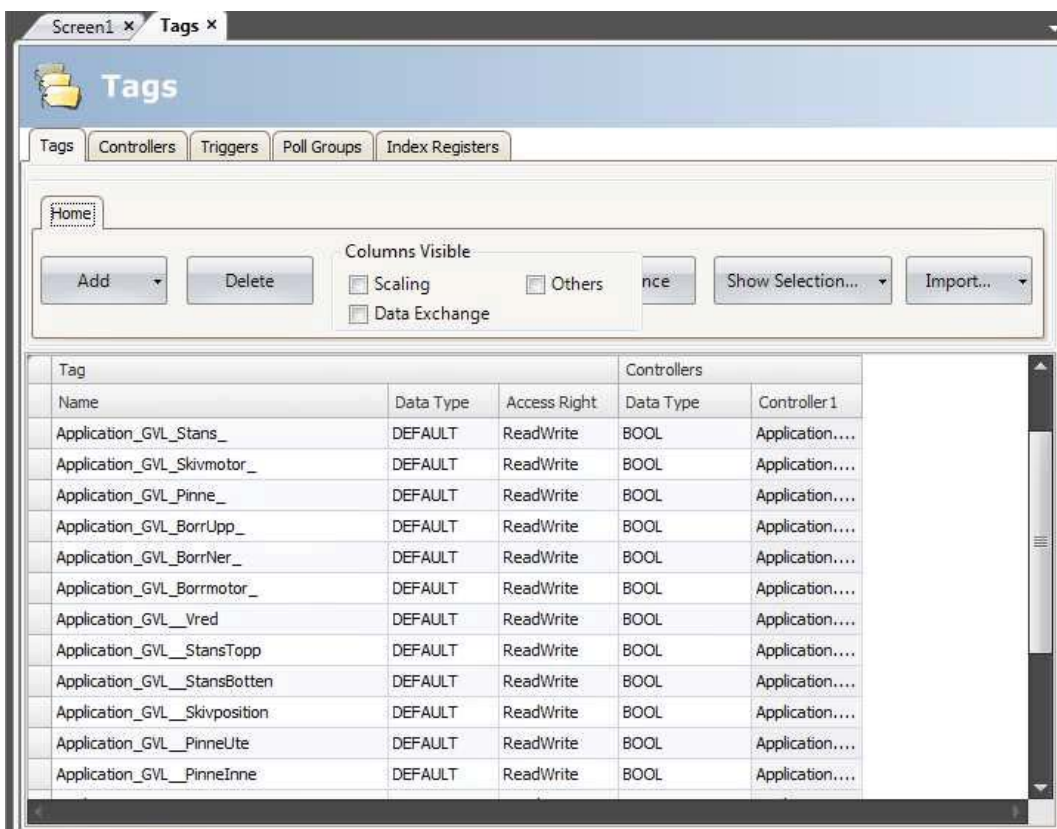
- Välj **Build** i menyn. Om felmeddelanden visas, ändra koden tills byggningen är felfri.
- Välj **Generate Code** enligt bilden till höger.
- Starta iX Developer och det projekt där variablerna ska användas.
- Gå in på tags och tryck på **Import**. Välj **Import tags to [Controller1]**.



- Leta upp filen som genererats i CODESYS. Den ligger i samma mapp som CODESYS-projektet och heter projektnamn.Device.Application.xml



- Resultatet är att variabellistan har importerats till iX Developer och ligger under Tags. Döp inte om variablerna (under name) utan tagga Application.namn i iX-projektet.



3 Översikt av CODESYS-miljön

När konfigurationen är klar och kommunikationen mellan hårdvaruenheterna fungerar kan man börja använda CODESYS.

3.1 Skapa ett projekt

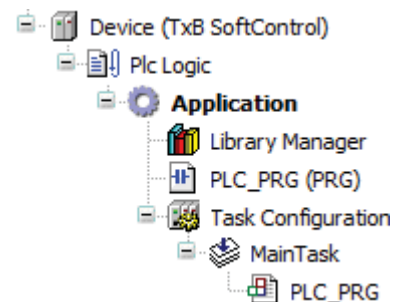
För att skapa ett nytt projekt, välj **File / New Project**.

I rutan Templates finns tre val:

- **Empty project** innehåller enbart "project settings" vilket innebär att bibliotek och applikationer läggs till manuellt.
- **Standard project** innehåller från början en "device" (t.ex. en PLC) som användaren själv väljer samt ett tomt program i valfritt programspråk.
- **Standard project with Application Composer** är ett tredje alternativ, som ej behandlas i denna manual.

Välj nu **Standard Project** samt namn och plats för projektfilen. I nästa dialogruta får man välja Device (CoDeSys Control Win V3) och vilket språk det första PLC-programmet ska skrivas i. En trädstruktur visas nu i fönstret "Device" till vänster på skärmen. Här visas projektets program och hårdvara som projektet kommunicerar med, samt övriga objekt som hör till projektet.

Trädstrukturen (bild till höger) börjar längst upp med projektets namn följt av **CoDeSys Control** på nivån under (Klicka på "+" eller "-" för att expandera eller dölja de objekt som ligger under en enhet i trädet). Under denna "fysiska hårdvara" finns en symbolisk nod med namnet **Plc Logic**, som bara visar att systemet är programmerbart. På nästa nivå ligger **Application** som innehåller den körbara koden i projektet.



Under **Application** finns **Library Manager**, där man kan hämta bibliotek med funktioner och funktionsblock som ingår i IEC61131-3. Under **Application** finns också alla program som görs inom projektet (från början finns här ett tomt program), samt **Task Configuration** som i ett standardprojekt innehåller **Main Task**.

En task är en aktivitet som följer vissa regler, t.ex. vilket tidsintervall det ska köras i. De olika programmen man vill köra kan man koppla till samma task, eller till olika med olika egenskaper. Ett program som inte ligger under en task i trädstrukturen kommer inte att exekveras alls.

3.2 Kör ett program

För att köra ett PLC-program kan följande steg utföras:

1. Bygg programmet (utan fel): **Build / Build**. Även **F11** eller från verktygsfält.
2. Ladda ner programmet till SoftControl: **File / Source download...**
3. Logga in på SoftControl: **Online / Login** eller **Alt-F8** (Motsatsen är **Online / Logout** eller **Ctrl-F8**. Funktionerna finns också i verktygsfältet.)
4. Kör programmet med **Debug / Start** eller **F5**. (Motsatsen är **Debug / Stop** eller **Shift-F8**. Även dessa funktioner finns i verktygsfältet.)

För att underlätta detta räcker det i allmänhet att börja på tredje punkten. Man kan då välja **Login with download** genom att trycka **Nerpil + Enter**. Då utför datorn punkt 1-3 ovan och det enda som återstår är att starta programmet med **Start** eller **F5**.

Att logga in innebär att CODESYS kommunicerar med SoftControl. Man kan alltså redan i detta läge titta på värdet hos ingångar och andra variabler. Däremot börjar inte koden exekveras förrän man väljer **Start**.

Bilden visar inloggat läge, men ej körläge. Tryck på pilen eller **F5** för **Start**. Symbolen till vänster är genvägen för **Build** eller **F8**.



Om man kopplar bort PC:n från SoftControl när man är i körläge fortsätter systemet att köra det nerladdade programmet. (Samma sak händer om man loggar ur eller om CODESYS avslutas utan att man har stoppat exekveringen.) För att stoppa körningen måste man då koppla in PC:n, logga in och välja **Stop**. Andra sätt att stoppa körningen är att ladda ner programmet igen eller att implementera en stopp-funktion på operatörspanelen (och då behöver inte PC:n anslutas). Dessutom stannar givetvis processen om man bryter kontakten mellan denna och SoftControl, eller om spänningsmatningen till processen upphör. Detta innebär dock inte att programexekveringen avbryts. Den kommer att fortsätta i SoftControl även om det inte syns på processen.

När PC:n har kopplats från och SoftControl styr processen självständigt kan matnings-spänningen brytas till både process och SoftControl. Processen stannar, men exekveringen startar om när spänningen slås på igen. PC:n behöver alltså bara kopplas in om program-koden ska redigeras. Eventuellt nödstopp bör i första hand kopplas direkt till processen så ett eventuellt fel i SoftControl inte påverkar säkerheten. Nödstoppet bör också kopplas till SoftControl, så exekveringen vid återställning av stoppet kan styras till rätt ställe i koden.

Man kan också testa sina CODESYS-program lokalt i datorn när inte SoftControl är tillgängligt. Välj **Online / Simulation**. Nu kan man logga in och köra programmet utan inkopplad hårdvara. Givetvis fungerar inte t.ex. ingångar från givare m.m., men dessa kan simuleras genom att dubbelklicka på en variabls värde (som syns direkt i koden i inloggat läge), och ändra "Prepared Value". (Kontakternas "värde" i Ladder visas som blå=TRUE , vit=FALSE) Vid ett tryck på **Ctrl-F7** eller **Debug / Write values** läggs de förberedda värdena in i respektive variabel, och man kan på det sättet simulera precis hur ingångarna ska bete sig. Funktionen fungerar även i normalt läge när man är inloggad mot SoftControl.






4 Menyer och funktionsblock

I detta kapitel gör en djupare genomgång av funktioner tillgängliga från menyn i CODESYS. De funktionsblock som ingår som standard presenteras. Kapitlet visar inga kompletta exempel på programmering. Läsare som direkt vill få en inblick i PLC-programmeringens möjligheter och hur ett komplett program kan se ut i CODESYS hänvisas till kapitel 5.







4.1 Menyer i CODESYS


Här görs en genomgång av rullgardinsmenyerna i CODESYS. De öppnas enklast med musen, men finns också tillgängliga genom att trycka på **Alt**-tangentsen och därefter styra med piltangenterna.

File




	New Project... (Ctrl+N)	Skapa nytt projekt.
	Open Project...(Ctrl+O)	Öppna ett befintligt projekt.
	Close Project	Stäng projekt.
	Save Project (Ctrl+S)	Spara projekt.
	Save Project As...	Spara project med nytt namn eller format.
	Project Archive	Packar ihop/ihop alla filer som hör till ett projekt till en fil.
	Source Upload...	Hämtar program från PLC.
	Source Download...	Laddar ner ett program till PLC.
	Print...	Skriver ut programkod.
	Print Preview...	Förhandsvisar utskrift.
	Page Setup...	Inställningar vid utskrift.
	Recent Projects	Öppna någon av senast använda projekt.
	Exit (Alt+F4)	Stänger Codesys.

Edit

	Undo (Ctrl+Z)	Går tillbaka ett steg och får sista ändringen "ogjord".
	Redo (Ctrl+Y)	Går fram ett steg och lägger till sista ändringen igen.
	Cut (Ctrl+X)	Klipper ut objekt eller text som markerats.
	Copy (Ctrl+Ins)	Kopierar objekt eller text som markerats.
	Paste (Ctrl+V)	Klistrar in objekt eller text som markerats.
	Delete (Del)	Tar bort objekt eller text som markerats.
	Select All (Ctrl+ A)	Markerar allt.

	Find Replace (Ctrl+F, Ctrl+H m.fl.)	Under Find Replace finns möjlighet att söka i koden samt att söka och ersätta ord.
	Browse	Go To Definition letar upp definitionen för den variabel eller det funktionsblock där markören står. Browse Cross References visar övriga placeringar i koden för den variabel som markören pekar på.
	Input Assistant... (F2)	Hjälp för att lägga till funktioner som finns olika bibliotek.
	Auto Declare... (Shift+F2)	Öppnar ett fönster där variabler deklaras, typ, namn och var variabeln ska deklaras.
	Next Message (F4)	Bläddrar till nästa meddelande i meddelandelistan efter att programmets byggts.
	Previous Message (Shift+F4)	Bläddrar till föregående meddelande i meddelandelistan efter att programmets byggts.
	Go To Source Position	För att gå till källan i koden som gett fel, varningar och meddelanden vid byggning.




View
















	Devices (Alt+0)	Visa de enheter (styrsystem) som finns i projektet samt underliggande program och andra objekt.
	POUs (Alt+1)	För avancerade användare.
	Modules (Alt+2)	För avancerade användare.
	Messages (Alt+3)	Visar byggmeddelanden.
	Element Properties	Inställningar vid SFC-programmering.
	ToolBox	Innehåller programmeringsverktyg som FB-block och LD-kontakter.
	Watch	Möjligheter att övervaka variabler vid körning.
	Breakpoints	Visar brytpunkter som finns i programkoden.
	Cross Reference List	Visar övriga placeringar i koden för den variabel som markören pekar på.
	Call Stack	Visar återhopsposition vid körning av program med funktionsanrop.
	Start Page	Visar startsidan i Codesys där project kan öppnas eller nytt projekt kan startas.
	Store	Codesys nerladdningssida på internet.
	Full Screen (Ctrl+Shift+F12)	Visa fullskärmsläge.
	Properties...	Visa egenskaper.

Project

	Add Object	Lägger till objekt.
	Add Folder...	Lägger till en ny folder under aktuell plats i trädstrukturen.
	Add Device...	Lägg till en ny enhet i projektet.
	Scan For Devices...	Söker efter inkopplad hårdvara att lägga i trädstrukturen. Finns även som högerklicks meny från vissa inkopplade hårdvaror.
	Update Device...	Uppdatera information gällande en enhet i projektet.
	Edit Object	Här görs ändringar i markerat objekt som t.ex POU,task konfiguration eller tillagda enheter i projektet.
	Edit Object With...	Här görs ändringar i markerat objekt som t.ex POU,task konfiguration eller tillagda enheter i projektet.
	Set Active Application	Om projektet innehåller fler än en applikation väljs vilken som ska köras här.
	Project Information...	Visar projektinformation som storlek,titel,beskrivning m.m.
	Project Settings...	Visar inställningar gällande projektet.
	Project Environment...	Visar biblioteks-,kompilerings-, och enhetsversioner m.m
	Document...	Visar projektet i utskriftsformat,vad som ska visas kan väljas.
	Compare...	Två project kan jämföras och skillanderna visas efter jämförelsen.
	Export...	Exportera projekt till en vald plats.
	Import...	Exportera projekt från en vald plats.
	Export PLCopenXML...	Exportera tillagd enhet, POU:er och dess inställningar till en vald plats.
	Import PLCopenXML...	Importerera tillagd enhet;POU:er och dess inställningar till en vald plats.
	User Management	Inställningar gällande tillstånd för ändringar i programkoden.


FBD/LD/IL (visas bara vid programmering i dessa språk)

	Insert Network (Ctrl+I)	Lägg till ett nytt nätverk.
	Insert Network Below (Ctrl+T)	Lägg till ett nytt nätverk under det befintligt.
	Insert Label	Syns bara vid FBD-programmering. Lägg till markering att använda vid Go to i koden.


	Toggle Network Comment State (Ctrl+O)	Kommentera bort markerad kod (blir då grönmarkerad).
	Insert Box (Ctrl+B)	Lägg till ny box (FBD) igenom Input Assistant där funktioner finns i olika bibliotek.
	Insert Empty Box (Shift+B)	Lägg till en tom box.
	Insert Box With EN/ENO	Lägg till ny box med Enable Output igenom Input Assistant där funktioner finns i olika bibliotek.
	Insert Empty Box With EN/ENO	Lägg till ny box med Enable Output.
x	Insert Input (Ctrl+Q)	Lägg till input.
	Insert Assignment (Ctrl+A)	Infoga variabelruta.
	Insert Coil (Ctrl+A)	Lägg till slutkontakt (Output).
	Insert Set Coil	Lägg till slutkontakt (Output) som är setdominant.
	Insert Reset Coil	Lägg till slutkontakt (Output) som är resetdominant.
	Insert Jump (Ctrl+L)	Lägg till hopp.
	Insert Return	Lägg till hopp till programstart.
	Insert Contact (Ctrl+K)	Lägg till kontakt (LD).
	Insert Negated Contact	Lägg till negerad kontakt.
	Insert Contact (right) (Ctrl+D)	Lägg till kontakt till höger.
	Insert Contact Parallel (below) (Ctrl+R)	Lägg till en parallel kontakt under markerad kontakt.
	Insert Negated Contact Parallel (below)	Lägg till en negerad parallel kontakt under markerad kontakt.
	Insert Contact Parallel (above) (Ctrl+P)	Lägg till en parallel kontakt över markerad kontakt.
	Paste Contacts (Ctrl+F, Ctrl+G, Ctrl+H)	Klistra in kopierad eller urklippt kontakt.
	Negation (Ctrl+N)	Negera in/utgång eller kontakt.
	Edge Detection (Ctrl+E)	Lägger till en R-Trig eller F-trigfunktion som markeras med en liten pil på en ingång till ett FB eller kontakt.
	Set/Reset (Ctrl+M)	Set/resetdominant slutkontakt.
	Set Output Connection (Ctrl+W)	Finns ej förklarad i hjälpen.
	Insert Branch (Ctrl+Shift+V)	Lägg till förgrening.
	Insert Branch above	Lägg till förgrening nedan.
	Insert Branch below	Lägg till förgrening ovanför.
	Set Branch Start/End Point	Bestäm var förgrening ska börja och sluta.
	Update Parameters (Ctrl+U)	Uppdaterar ut/ingångar på ett FB.
	Remove unused FB call Parameters	Tar bort ut/ingångar på ett FB.

	View	Visa aktuell kod i ett annat språk.
	Go To...	Gå till en label (se ovan) eller ett nätverk (ett hopp i programmet).

Build

	Build (F11)	Kompilerar aktiva program (de som ligger under Task Configuration).
	Rebuild	Bygger de senaste ändringarna.
	Generate code	Används för att exportera variabler till iX Developer (se avsnitt 2.6)
	Generate runtime system files	Skapar externa biblioteksfiler för andra programspråk.
	Clean	Rensa program i PLC:n.
	Clean All	Rensa program och parameterar i PLC:n.






Online

	Login (Alt+F8)	Bygger, laddar ner projektet till styrsystemet och sätter programmet i körläge.
	Logout (Ctrl+F8)	Gå ur körläget.
	Create Boot Application	Skapa bootfil som laddas automatiskt när PLC:en startas.
	Download	Ladda ner projekt till styrsystemet.
	Online Change	Tillåter att göra ändringar i körläge.
	Source download to connected device	Ladda ner program till enheten (styrsystemet) i projektet.
	Multiple Download...	Ger möjlighet att välja vilka Applikationer som ska laddas.
	Reset warm	Gör reset på alla vanliga variabelvärden, d.v.s variablerna får sina initieringsvärden.
	Reset cold	Gör reset på alla vanliga variabelvärden och retainvärden.
	Reset origin	Gör reset på alla vanliga variabelvärden och retainvärden. Applikationen som ligger i PLC raderas.
	Simulation	Används för att sätta CODESYS i simuleringsläge.
	Security	Inställningar för fjärrstyrning, för t.ex. onlinesupport.




Debug

	Start (F5)	Starta program.
	Stop (Shift+F8)	Stoppa program.
	Singel Cycle (Ctrl+F5)	Kör en programcykel.
	New Breakpoint...	Lägg till ny breakpoint.
	Edit Breakpoint...	Editera breakpoint.
	Toggle Breakpoint (F9)	Lägg till / ta bort en breakpoint.
	Disable Breakpoint	Lägg breakpoint i inaktivt läge.
	Enable Breakpoint	Sätt breakpoint i aktivt läge.
	Step Over (F10)	Stega över funktion.
	Step Into (F8)	Stega in i funktion.
	Step Out (Shift+F10)	Stega över funktion.
	Run to Cursor	Kör program fram till markering.
	Set next Statement	Flyttar aktiv position till markören varifrån programmet sedan ska köras.
	Show next Statement	Visar aktiv position till markören varifrån programmet sedan ska köras.
	Write Values (Ctrl+F7)	Lägg in preparerade värden i variabler. Används t.ex i simuleringsvärde.
	Force Values (F7)	Tvingar en variabel att ha ett visst värde under körning.
	Unforce Values (Alt+F7)	Tar bort tvånget.
	Flow Control	Ett felsökningsverktyg som förlänger programcykeltiden så att programmet körs långsammare.
	Display Mode	Väl talsystem för variabelvisning (heltal).




Tools

	Package Manager...	Installations- och drivrutinshantering.
	Library Repository...	Bibliotekshanterare.
	Device Repository...	Välj och installera drivrutiner.
	Visualizations Styles Repository...	Inställningar för visualisering av PLC-program, d.v.s. grafisk visning av variabelvärden m.m.
	License Manager	Visar aktiva licenser för plug-in-program till CODESYS.
	Scripting	Hantering av script från andra program.
	Customize...	Väljer vilka funktioner som finns i menyerna och verktygsfälten.
	Options...	Övriga CODESYS-inställningar.

Window

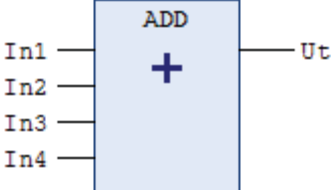
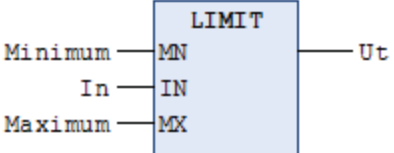
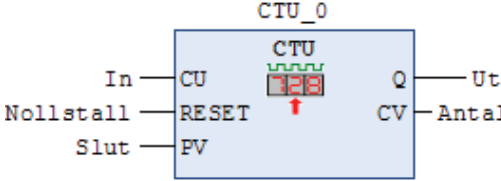
	Next Editor (Ctrl+F6)	Visa nästa öppna flik.
	Previous Editor (Ctrl+Shift+F6)	Visa föregående öppna flik.
	Close all editors	Stäng alla flikar.
	Close all editors of inactive applications	Stänger flikar som hör till andra applikationer. Normalt har man bara en applikation.
	Reset Window Layout	Ändrar fönsterinställningar. Kan användas om meddelandefönstret inte går att hitta.
	New Horizontal Tab Group	Aktiv flik hamnar under de andra i en egen grupp.
	New Vertical Tab Group	Aktiv flik hamnar till höger om de andra i en egen grupp.
	Float	Gör ett "flytande" fönster av t.ex. Devices eller Toolbox.
	Dock	Lås ett fönster (motsatsen till float).
	Auto Hide	Minimerar ett fönster (mot skärmkanten) när det inte används.
	Next Pane (F6)	Gå till nästa panel inom samma flik.
	Previous Pane (Shift+F6)	Gå till föregående panel inom samma flik.
	Toggle First Pane (Alt+F6)	Minimerar första panelen inom samma flik.
	Toggle Second Pane	Minimerar andra panelen inom samma flik.
	(Här visas en lista av öppna fönster)	Välj vilket fönster/flik som ska visas överst.
	Windows...	Fönsterhantering.

Help

	Contents (Ctrl+Shift+F1)	Visa innehåll i hjälpen.
	Index (Ctrl+Shift+F2)	Ämnen i hjälpen ligger i bokstavsorning i en lista över ämnen.
	Search	Sök.
	3S Homepage...	Företaget 3S hemsida.
	About...	Information om Codesys.

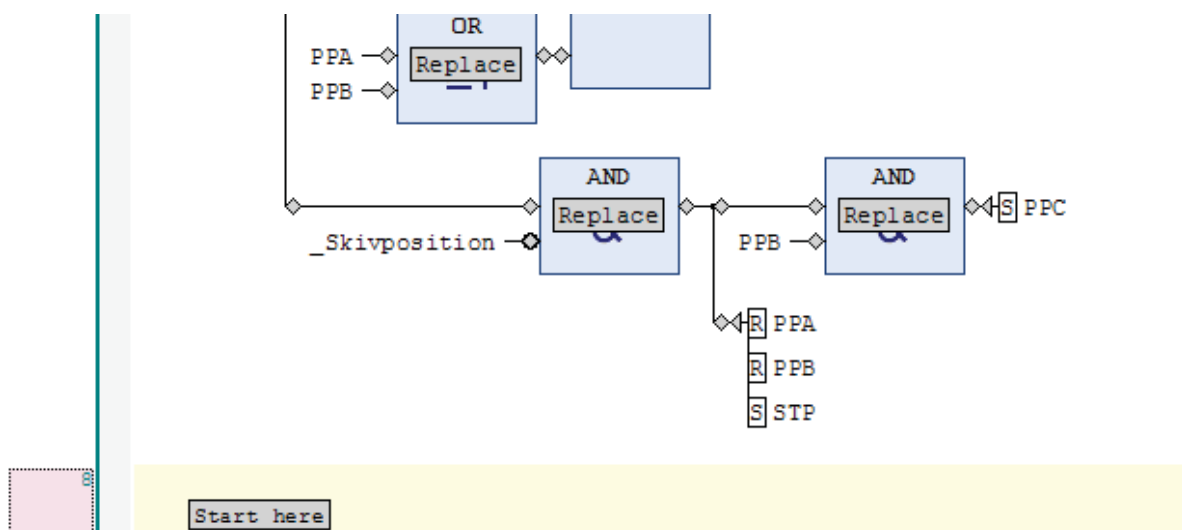
4.2 Funktionsblock

I CODESYS finns ett antal inbyggda logiska, aritmetiska eller tidsberoende funktioner, s.k. *funktionsblock*. Ett program i *Function Block Diagram* (FBD) är som namnet antyder uppbyggt på sådana funktionsblock. Men ett funktionsblock är egentligen dess funktion snarare än dess grafiska form. Alla funktionsblock har också en motsvarighet i textform, för ST-programmering. Figur 4.1 visar exempel på hur funktioner/funktionsblock ser ut i de två språken.

ADD		<p>Enkla aritmetiska eller logiska block skrivs i textform med vanliga operatörer mellan respektive ingång. Motsvarigheten i detta fall är:</p> <pre>Ut:=In1+In2+In3+In4;</pre>
LIMIT		<p>Block som inte bygger på vanliga operatörer skrivs som en funktion med parenteser:</p> <pre>Ut:=LIMIT(Minimum, In, Maximum);</pre>
CTU		<p>Block med minne anropas i ST med sitt instansnamn (CTU_0). Blockets interna beteckningar för in- och utgångar används.</p> <pre>CTU_0(CU:=In, RESET:=Nollställ, PV:=Slut);</pre> <pre>Ut:=CTU_0.Q;</pre> <pre>Antal:=CTU_0.CV;</pre> <div style="border: 1px solid red; padding: 2px; display: inline-block; color: red;">Se figur 5.6 för alternativt skrivsätt.</div>

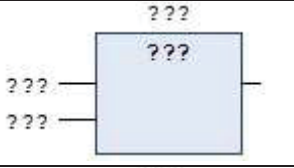
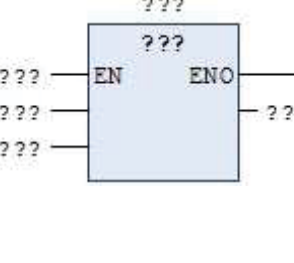
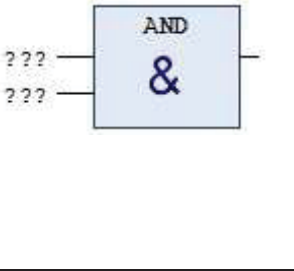
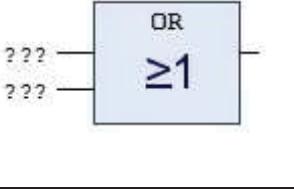
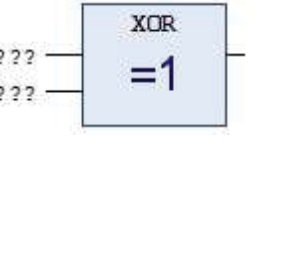
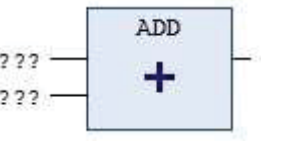
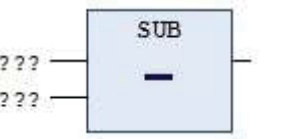
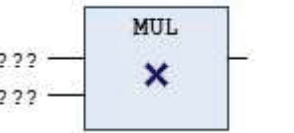
Figur 4.1. Tre grundtyper av funktionsblock i grafisk form (för FBD- och LD-program) och deras motsvarighet i textform (för ST-program).

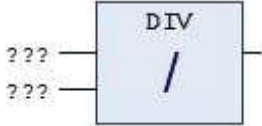
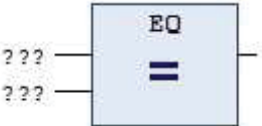
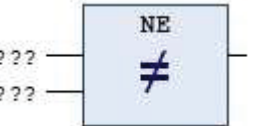
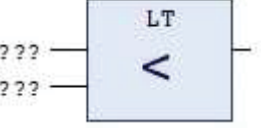
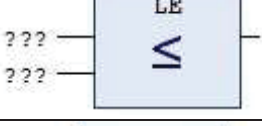

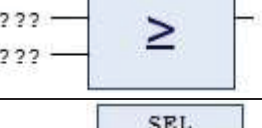
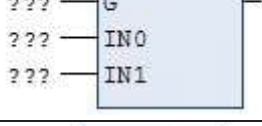
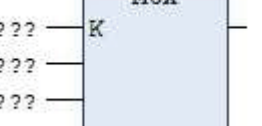
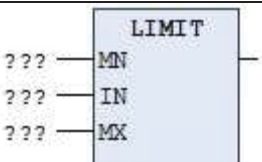
De inbyggda funktionsblocken hämtas genom att platsen där de ska infogas markeras (ett tomt network, eller en in- eller utgång på ett befintligt block). I menyn **FBD/LD/IL** kan man välja **Insert Box** för att välja från en trädstruktur, eller **Insert Empty Box** för att få ett tomt block på önskat ställe på skärmen, och sedan välja vad blocket ska innehålla. De vanligaste blocken kan också dras med musen från verktygsfältet till höger in till önskad plats. När man drar in ett block med musen kan man placera det på alla gråa områden på skärmen, enligt figur 4.2.



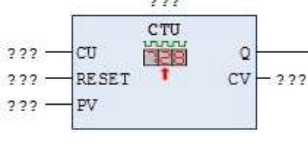
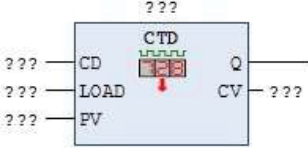
Figur 4.2. Möjliga positioner för placering av funktionsblock. Dra in blocken med musen från verktygsfältet till en av de små kvadraterna eller trianglarna, eller till rektanglarna med "Replace" eller "Start here".

Nedanstående tabell visar de block som finns tillgängliga i verktygsfältet till höger på skärmen.

Empty Box		En "Empty Box " är ett tomt block som kan fyllas med valfri funktion. Klicka på ??? inne i rutan och skriv de första bokstäverna i önskat funktionsblock.
Empty Box with ???		Skillnaden mot föregående block är att detta har en enable-signal. Ingången EN måste ha värdet TRUE för att blocket ska aktiveras. Annars är blocket passivt och inga utgångar får några värden. Utgången ENO är dock en "kortsloten" genomgång från EN. Alla funktionsblock kan väljas med eller utan enable-signal
AND		Den logiska funktionen "och" jämför bit för bit av ingångarna och lägger resultatet på utgången till höger. Om t.ex. 20 och 5 ligger på respektive ingång får utgången värdet 4, eftersom gemensamma bitar i 10100 och 00101 är 00100. Fler ingångar kan läggas till genom att högerklicka på blocket och välja Append Input.
OR		Den logiska funktionen "eller" jämför också bit för bit. Talen 20 och 5 på ingångarna ger 21 på utgången eftersom bitar som finns i 10100 eller 00101 tillsammans ger 10101. Även här kan antalet ingångar väljas fritt.
XOR		Den logiska funktionen "exklusivt eller" jämför för varje bitposition om det finns exakt en 1:a och en 0:a. Talen 20 och 5 på ingångarna ger 17, d.v.s. 10001. (Den mittersta 1:an försvinner eftersom den finns i både 20 och 5). Resultatet kan också räknas ut som differensen mellan ovanstående två resultat (se AND och OR), d.v.s. 21-4=17.
ADD		Blocket ADD adderar ingångarnas värde och lägger resultatet på utgången till höger. Antalet ingångar är valfritt.
SUB		Utgången är här värdet av det den övre ingången minus den nedre.
MUL		Utgången är produkten av ingångarna. Antalet ingångar är valfritt.

DIV		Det övre talet divideras med det nedre och resultatet läggs på utgången. Operationen är en heltalsdivision, d.v.s. eventuella decimaler i resultatet klipps bort. Om man är intresserad av resten som bildas vid en heltalsdivision finns blocket MOD (välj "Empty Box" och skriv MOD).
EQ		Utgången får värdet TRUE om ingångarna har samma värde, annars FALSE.
NE		"Not Equal"-blocket ger inversen av ovanstående, d.v.s. FALSE när ingångarna är lika och TRUE när de är olika.
LT		"Less Than" ger TRUE om den övre ingången är mindre än den nedre.
LE		"Less or Equal than" ger TRUE om den övre ingången är mindre än eller lika med den nedre.
GT		Ger på motsvarande sätt TRUE om den övre ingången är större än den nedre.
GE		Utgången är TRUE om den övre ingången är större än eller lika med den nedre.
SEL		Blocket väljer mellan IN0 och IN1. Om ingången G är FALSE kopplas värdet vid IN0 till utgången. Om G är TRUE kopplas i stället IN1 vidare.
MUX		Blocket väljer på samma sätt som ovanstående, men ingången är här en INT i stället för en BOOL. Om K=0 kopplas värdet på första ingången (under K) till utgången. Om K=1 kopplas nästa ingång o.s.v. Antalet ingångar är valfritt.
LIMIT		Insignalen IN skickas vidare till utgången om den ligger inom intervallet $MN \leq IN \leq MX$. Annars får utgången värdet MN respektive MX i stället.

MOVE		Som namnet antyder flyttas värdet från ingången till utgången på blocket, när "enable" (EN) är TRUE. Precis som med andra block kan man välja med eller utan "enable", men ett MOVE-block utan "enable" saknar mening, eftersom utgången då alltid kommer att ha samma värde som ingången. Ett vanligt "streck" ger alltså samma funktion.
Format-konvertering		Konverteringsblock finns i en mängd varianter, se tabell X.X. Blocket i figuren till vänster saknar frågetecken på ingången, vilket beror på att man normalt utgår från ett block eller en variabel, och därefter lägger till ett konverteringsblock, som då anpassar sin "första del" beroende på data typen på ingången.
R_TRIG		Utgången Q blir TRUE på första klockpulsen när ingången just fått värdet TRUE (positiv flank). Sedan återvänder utgången till FALSE direkt. Alla funktionsblock som har ett minne, d.v.s. utgången beror inte enbart på ingångarnas momentana värde, måste ha ett instansnamn, detta ser man genom de tre frågetecknen ovanför blocket.
F_TRIG		Fungerar motsvarande för negativ flank. Utgången blir TRUE precis när ingången växlat till FALSE, men återvänder nästa klockpuls till FALSE.
RS		RS-vippan sätter TRUE utgången när SET är TRUE, och FALSE på utgången när RESE1 är TRUE. Om båda ingångarna är TRUE samtidigt dominerar RESE1 (1:an betyder dominans).
SR		SR-vippan fungerar som ovanstående, men om båda ingångarna är TRUE samtidigt dominerar SET1.
TON		Blocket "Timer On delay" ger ett fördröjt tillslag på Q när IN slås till. PT anger tidsfördröjningen. Om PT är T#2s kommer Q att bli TRUE två sekunder efter att IN blir TRUE (men bli FALSE samtidigt som IN). På ET visas tiden som har gått från tillslaget. ET måste inte användas.
TOF		Blocket "Timer Off delay" ger i stället ett fördröjt fränslag på Q när IN slås från. På ET visas tiden som har gått från fränslaget. Om fördröjning önskas på både positiv och negativ flank kan TON och TOF seriekopplas.

CTU		CTU räknar positiva flanker på CU och antalet syns på CV. Om man har ett "mål" för antalet läggs detta på PV. När $CV \geq PV$ skickas en TRUE-signal ut på Q. Räknaren nollställs när RESET=TRUE.
CTD		CTD räknar på samma sätt, men i motsatt riktning, från PV ner till noll, i stället för tvärtom. Räknarvärdet ligger på CV och Q blir TRUE när $CV=0$. LOAD används för att återställa CV till startvärdet PV.

Att använda tangentbordet för att programmera FBD är svårare, men inte omöjligt. Välj **Alt-W** och öppna önskat kodfönster. (Eller tryck bara på **Alt** för att välja valfri meny). Styr markören (en rosa rektangel) till önskad position i koden för att lägga till ett funktionsblock. Lägg till en box (**Ctrl-B**) och leta upp önskat block. De vanligaste finns i **Categories / Keywords**. (Växla mellan **Text Search** och **Categories** med **Ctrl-Tab**). LD-programmering kan göras på motsvarande sätt med tangentbordet. (Välj **Alt-I** för att öppna menyn **FBD/LD/IL**.)

Dessutom finns ytterligare CODESYS-funktionsblock som inte finns installerade som standard. För att installera dessa, dubbelklicka på **Library Manager** i trädstrukturen och välj **Add library**. Välj sedan ett bibliotek, t.ex. **Application / Common / Util** och klicka på **OK**.

Konvertering mellan datatyper behövs ibland i PLC-programmering på samma sätt som i andra programspråk. Här är ett urval av möjligheterna. Eftersom funktionsblocken bara har en ingång och en utgång visas här deras namn utan den grafiska utformningen.

BOOL TO INT	Konverterar FALSE till 0 och TRUE till 1.
BOOL TO STRING	FALSE ger strängen 'FALSE', TRUE ger strängen 'TRUE'.
BOOL TO TIME	Tiden räknas i millisekunder. FALSE ger alltså T#0ms, TRUE ger T#1ms.
BOOL TO TOD	FALSE ger TOD#00:00:00.000, TRUE ger TOD#00:00:00.001,
BOOL TO DATE	FALSE ger D#1970-01-01, TRUE ger D#1970-01-02.
BOOL TO DT	FALSE ger DT#1970-01-01-00:00:00 TRUE ger DT#1970-01-01-00:00:01
BYTE TO BOOL, INT TO BOOL etc.	0 ger FALSE, övriga tal ger TRUE.
TIME TO BOOL	Alla tider > T#0ms ger TRUE
STRING TO BOOL	'1' eller 'TRUE' ger TRUE. Allt annat ger FALSE, t.ex. 'FALSE', 'True', '5' och '1.0'.
INT TO SINT	Endast de åtta sista bitarna i talet behålls d.v.s. ett tal i intervallet 0-255. Värdet 256 blir 0, 257 blir 1 o.s.v. Samma resultat som blocket MOD med "256" som nedre ingång.
REAL TO INT	Avrundar till närmaste heltal. 13,4 blir 13, 13,5 blir 14 och -7,5 blir -8
TIME TO DWORD	Gör om tiden i millisekunder till ett stort heltal. T#10m blir 600000.

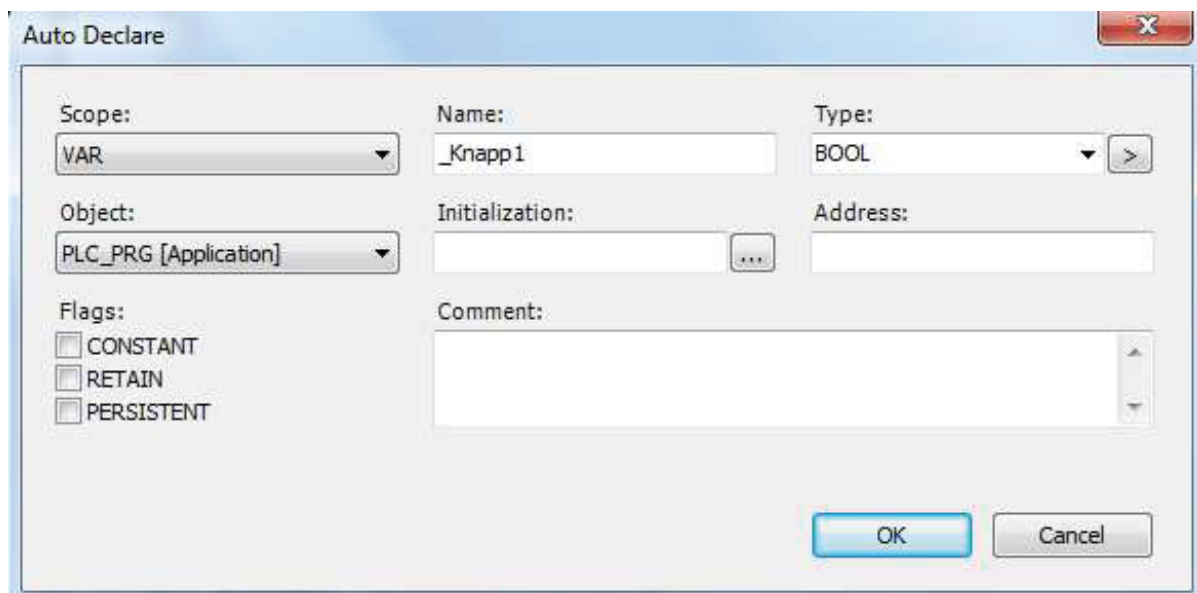
TOD TO SINT	Gör om tiden i millisekunder till ett kort heltal. TOD#00:00:00.050 ger 50.
TIME TO STRING	Gör en sträng av tiden. T#5ms blir 'T#5ms'.
DATE TO BYTE	Räknar sekunder mellan början av 1970 och tiden på ingången, men behåller bara 8 bitar. I praktiken blir detta alltså ett slags slumpstal.
DT TO STRING	Gör en sträng av dag och tid. DT#2014-12-24-15:00:00 blir 'DT#2014-12-24-15:00:00'
STRING TO WORD	Översätter en sträng med ett positivt heltal till detta tal. Decimaltal, negativa tal och övriga strängar ger 0, utom 'TRUE' som ger 1.
STRING TO BYTE	En sträng med ett positivt heltal ger detta tal (eller den del av talet som ryms inom aktuell datatyp). '256' ger 0, '260' ger 4 o.s.v.
STRING TO TIME	En korrekt sträng ger motsvarande tid. 'T#1h30m' ger T#1h30m
TRUNC	Omvandlar från REAL till DINT. Eventuella decimaler klipps därmed bort. 1.9 blir 1, -44.6 blir -44.
TRUNC_INT	Omvandlar från REAL till INT, d.v.s. eventuella decimaler klipps bort och även bitar som inte ryms i en INT.

5 Exempel på programmering i CODESYS

Detta kapitel ger konkreta exempel på utformning av PLC-kod i CODESYS, först i språken ST, FBD och LD, och senare SFC och CFC. Tanken är att läsaren snabbt ska förstå hur de olika språken ser ut, och vad en PLC kan användas till.

5.1 Deklaration av variabler

Välj ett av programmen (under Application i trädstrukturen) genom att dubbelklicka på det. (Det första programmet får automatiskt namnet PLC_PRG.) Ikonen till vänster om detta visar vilket språk som ska användas. Till varje Program/POU hör en variabeldeklaration, som visas genom att dra fram ett fönster som kan vara dolt i överkant av kodfönstret. Här deklarerar de variabler som används, se figur 5.1. Om en variabel ska användas i mer än ett POU läggs den i stället i den globala variabellistan GVL som visas i samma figur. Variabeldeklarationerna kan matas in manuellt, men enklast är ofta att skriva koden som man vill ha den och låta programmets Auto Declare-funktion själv föreslå en deklARATION av de variabler som används.



I Auto Declare-fönstret kan man acceptera förvalda inställningar för variabeln, eller välja egna, t.ex. vilken typ den har och om den ska vara lokal eller global. Oavsett om variabler deklarerar manuellt, eller fylls på automatiskt efterhand som de används i koden, kommer variabellistorna att se ut ungefär som i figur 5.1.

```

PROGRAM Maskin_ST
VAR
  STP: BOOL := TRUE;
  PU: BOOL;
  PI: BOOL;
  BSN: BOOL;
  BSU: BOOL;
  SUP: BOOL;
  PPA: BOOL;
  PPB: BOOL;
  PPC: BOOL;
  Ton1: TON;
END_VAR

VAR_GLOBAL
  _Knapp1: BOOL;
  _Knapp2: BOOL;
  _Vred: BOOL;
  _Knapp3ON: BOOL;
  _BorrTopp: BOOL;
  _BorrBotten: BOOL;
  _StansTopp: BOOL;
  _StansBotten: BOOL;
  _PinneInne: BOOL;
  _PinneUte: BOOL;
  _HogerGivare: BOOL;
  _Skivposition: BOOL;
  Skivmotor_: BOOL;
  Borrmotor_: BOOL;
  BorrUpp_: BOOL;
  BorrNer_: BOOL;
  Pinne_: BOOL;
  Stans_: BOOL;
END_VAR

```

Figur 5.1. En variabellista för ett program (t.v.) och projektets globala variabellista (t.h.).

När en variabel deklaras kan den också ges ett initialvärde, se längst upp till vänster i figur 5.1. Ges inget initialvärde i deklARATIONEN kommer värdet att bli 0 respektive FALSE. Observera dock att denna initiering endast sker när man tömmer PLC-systemet genom Build/Clean i menyn. Det räcker i allmänhet *inte* att stoppa PLC:n, logga ut, bygga om, eller ladda ner programmet. Om inte en PLC:n töms kommer variablerna att behålla sitt senaste värde och initieringsvärdet ignoreras. Om man vill att systemet ska startas med förvalda värden varje gång bör man alltså använda sig av en startsekvens som bara aktiveras i början av körningen, eller när man vill "starta om". I figur 5.1 är variablerna av typen BOOL, eftersom alla givare och styrsignaler som används i processen är binära. Undantaget är Ton1. Längst ner till vänster, som inte är en vanlig variabel utan ett *funktionsblock* av typen TON, *Timer On Delay*. Alla funktionsblock med någon form av minne, d.v.s. utgångarna beror inte enbart på det momentana värden på ingångarna, måste ha ett instansnamn, och detta ska deklaras i variabellistan. TON är alltså blockets typ och Ton1 är namnet för just denna individuella blockinstans.

Eftersom variabeldeklARATIONERNA lätt görs med programmets *Auto Declare*-funktion kommer nu fokus att läggas på programkoden. DeklARATIONERNA för FBD- och LD-koden görs i textform på samma sätt, i fönstret ovanför respektive kodfönster.

5.2 Programkod för borrarprocess.

Detta avsnitt gör först en genomgång av programmet för den borrarprocess som visas i flödesschemat i figur 5.2. I avsnitt 5.3 analyseras motsvarande PLC-program, genom att samma kodsekvens visas i ST, FBD och LD, och skillnaderna belyses. Några allmänna sätt att programmera diskuteras i avsnitt 5.4. Avsnitt 5.5 visar samma program i SFC, som erbjuder andra möjligheter för programmeraren. Därefter lämnas borrarprocessen och avsnitt 5.6 handlar om att göra egna funktionsblock. Slutligen ges i avsnitt 5.7 en kort inblick i CFC, CODESYS:s eget programspråk som bygger på FBD

Observera att variabeldeklarationerna inte visas för borrarprocessen (mer än exemplet i figur 5.1) utan fokus läggs på den "körbara" koden.

När startvredet är frånslaget och processens rörliga delar står stilla, ligger programexekveringen i en loop i väntan på att vredet slås på, se översta loopen i figur 5.2. När vredet är tillslaget fortsätter körningen neråt enligt pilarna. Om skivan är mellan två positioner körs den till nästa position. En pinne med givare kontrollerar om det finns en puck som ska behandlas. Om så är fallet borrar ett hål i denna. En "stans" testar samtidigt borrarat hål i pucken i positionen framför om det finns en sådan, eftersom programmet minns om en puck är på väg dit. (Denna enhet kallas stans i denna manual, men det väsentliga är inte om stansen verkligen behandlar pucken, eller bara kontrollerar det borrarade hålet). När behandlingarna är klara fortsätter skivan att rotera om fler puckar behöver behandlas. Annars stannar den.

Observera att flödesschemat inte är exakt, utan visar en grov tanke på hur systemet ska fungera. Om flödesschemat skulle följas exakt testas bara vredets läge en gång i koden (i den översta romben), och maskinen kan alltså arbeta vidare i många steg, även efter vredet är frånslaget.

Detta problem elimineras i den riktiga programkoden. I ST, FBD och LD görs detta genom att hela slingan, inklusive test av vredet körs i hög hastighet, varv efter varv, men programmet håller ändå reda på vilken ruta och vilket villkor som är aktuellt.

I SFC körs programmet steg för steg, ungefär som man normalt tänker sig ett flödesschema. I varje steg finns dock ett test av vredet som innebär att en lokal resetsekvens körs i stället för det som normalt händer. Därmed kommer inte heller villkoret att uppfyllas för att körningen ska fortsätta. När vredet slås på igen fortsätter processen på samma ställe i koden som den avbröts.



Figur 5.2. Flödesschema för borrarprocess.

5.3 Programkod i ST, FBD och LD

I detta avsnitt visas programkoden för borrprocessen sekvens för sekvens i språken ST, FBD och LD, för att läsaren snabbt ska kunna bekanta sig med språken, men också se skillnader mellan dessa.

5.3.1 Sekvens Reset

Den första sekvensen, Reset, ska köras när startvredet är frånslaget. Den innehåller nollställning ("FALSE") av de flesta av programmets ut signaler och interna variabler. Undantag är BorrUpp_ som höjer bormaskinen till sitt utgångsläge (om den inte redan är där) och STP som är ett av villkoren för att nästa sekvens ska kunna köras. Dessa två variabler ettställs i stället ("TRUE").

Ett streck *i början* av ett variabelnamn t.ex. _Vred betyder i detta program att variabeln är en ingång till systemet. Dessa får alltså sitt värde från en givare eller ett reglage i processen, och ändras inte via programkoden. Ett streck *i slutet* av ett variabelnamn visar i stället att variabeln är en utgång, d.v.s. en styrsignal till processen. På detta sätt ser man tydligt att t.ex. BorrNer_ är en styrsignal och inte en ingång som säger att bormaskinen *är nere*. Variabler som inte är kopplade till processen har i detta exempel inga streck. Man kan välja andra sätt att skilja på variabeltyper, men övriga specialtecken är inte tillåtna i variabelnamn. Siffror är tillåtna, men inte som första tecken.

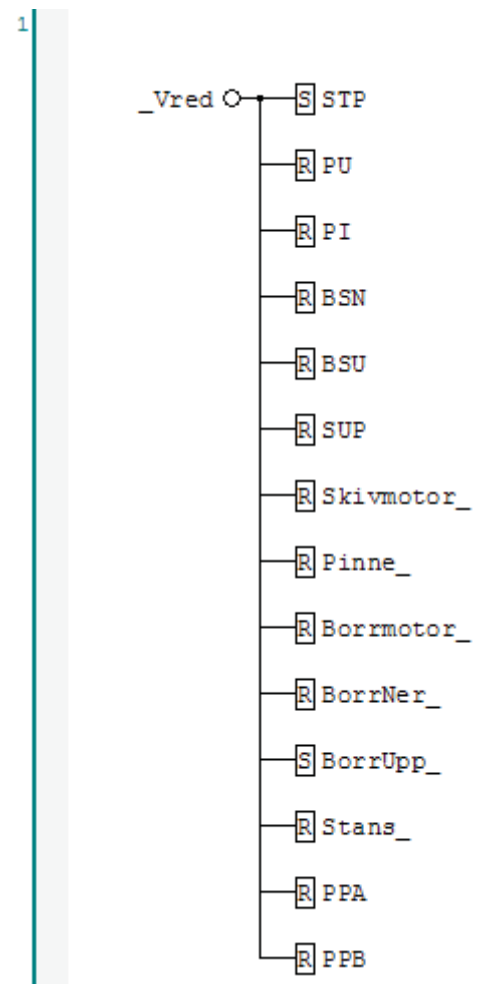
Figur 5.3 visar syntax för ST, *Structured Text*. Koden liknar andra programspråk, som t.ex. C eller Pascal. Tilldelningsoperatoren är := och IF-blocket avslutas med END_IF.

En viktig skillnad mellan PLC-programmering och "vanlig" programmering är att PLC-koden utformas så programmet kan göra flera saker "samtidigt". När IF-blocket i figur 5.3 är avklarat körs programmet vidare, tillsammans med övriga aktiva program (POU). Därför används IF i stället för WHILE i exemplet. Just i detta fall ska inget annat inträffa när maskinoperatören har slagit från _Vred, men det ska ändå vara *möjligt* för andra IF-satser att vara aktiva samtidigt.

Figur 5.4 visar motsvarande kod i FBD, *Function Block Diagram*. Koden är grafisk och numreras i *networks* i stället för rader som i ST. Rad 1 till 16 i ST motsvarar i detta fall

```
1 IF NOT _Vred THEN
2     STP:=TRUE;
3     PU:=FALSE;
4     PI:=FALSE;
5     BSN:=FALSE;
6     BSU:=FALSE;
7     SUP:=FALSE;
8     Skivmotor_:=FALSE;
9     Pinne_:=FALSE;
10    Borrmotor_:=FALSE;
11    BorrNer_:=FALSE;
12    BorrUpp_:=TRUE;
13    Stans_:=FALSE;
14    PPA:=FALSE;
15    PPB:=FALSE;
16 END_IF
```

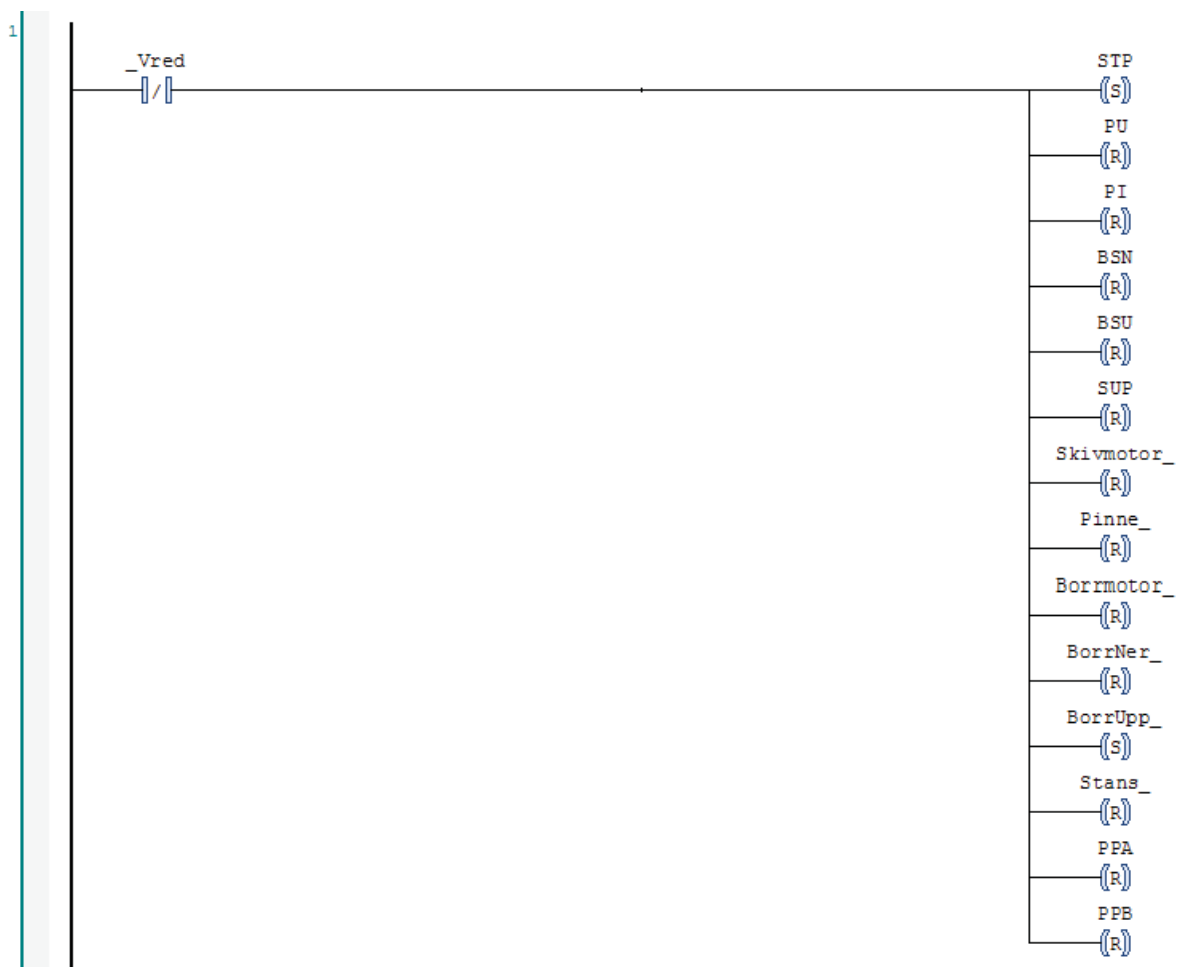
Figur 5.3. ST-kod för sekvens Reset.



Figur 5.4. FBD-kod för sekvens Reset.

network 1 i FBD. Ringen vid `_Vred` innebär negation, och rutorna med S och R betyder *Set* respektive *Reset*, när en etta läggs på i trädstrukturen till vänster om rutorna. När vredet slås på skickas en nolla ut, vilket betyder att variablerna lämnas med oförändrat värde, precis som i ST-koden när IF-villkoret inte är uppfyllt.

Nästan likadant ser det ut i figur 5.5. Koden i LD, *Ladder Logic Diagram*, bygger på en symbolisk spänningskälla, den tjocka svarta linjen till vänster. Insignalerna visas som kontakter som öppnar eller sluter kretsen. Snedstrecket i kontakten `_Vred` betyder negation. När vredet är frånslaget sluts alltså kretsen och "spänningen" når "spolarna" till höger som ettställer (*Set Coil*) eller nollställer variablerna (*Reset Coil*) precis som i FBD. (En spole utan S eller R ger variabeln det momentana värdet som finns på ingången, d.v.s. inversen av `_Vred`. Samma sak gäller vid variabeltilldelningar i FBD när S eller R saknas.)



Figur 5.5. FBD-kod för sekvens Reset.

5.3.2 Sekvens STP (Skiva till position)

STP är en flagga som krävs för att köra IF-blocket i figur 5.6. Vi kan därför kalla denna sekvens STP, även om man egentligen inte behöver sätta namn på olika programdelar i ST, FBD och LD. Sekvensen körs när STP är ettställd, vredet är påslaget och bormaskinen (av säkerhetsskäl) är i sitt övre läge. Alla sekvenser, utom Reset, inleds med att nollställa

föregående sekvens, så inte programmet exekveras på två ställen samtidigt. I andra sammanhang kan det vara önskvärt med parallell exekvering, men för att underlätta felsökning och översättning mellan språken väljs för denna maskin en programstruktur där instruktioner körs i sekvens. Som tidigare nämnts *stannar* inte exekveringen på ett ställe, däremot är koden utformad så den aktiva delen av koden, där utgångar och andra variabler kan *ändras*, i allmänhet är begränsad till ett område inom en sida på skärmen, oavsett vilket språk man väljer.

```

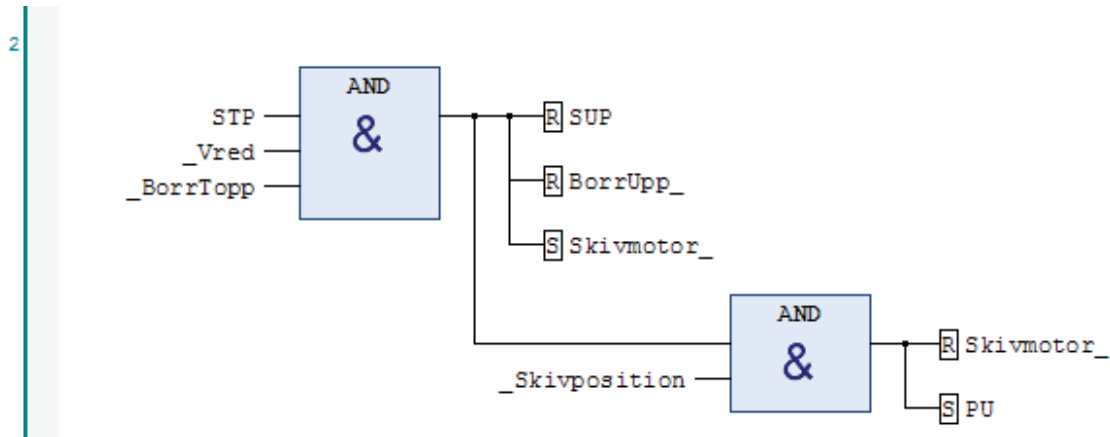
17 IF STP AND _Vred AND _Borrtopp THEN
18     SUP:=FALSE;
19     BorrUpp_:=FALSE;
20     Skivmotor_:=TRUE;
21     IF _Skivposition THEN
22         Skivmotor_:=FALSE;
23         Ton1(IN:=0, PT:=T#0S, Q=>, ET=>);
24         PU:=TRUE;
25     END_IF
26 END_IF

```

Figur 5.6. ST-kod för sekvens STP. Som radnumreringen visar ligger ovanstående kod precis under koden i figur X. Utgångarna Q och ET (rad 23) är fördeklarade i Ton1, och skrivs här endast av pedagogiska skäl. Endast ingångarna är obligatoriska i funktionsanrop i ST.

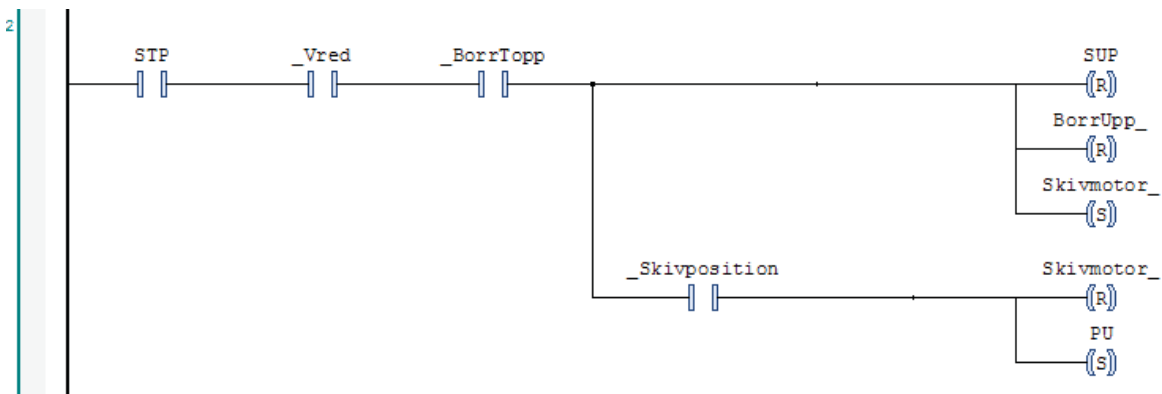
I detta fall nollställs SUP (Skiva ur position), eftersom det är den sekvens som kördes före STP (förutom när STP körs efter Reset-sekvensen, d.v.s. när vredet precis slagits på). Därefter nollställs BorrUpp_, eftersom bormaskinen nu är i toppläge, enligt IF-villkoret. Skivmotorn sätts på och går tills skivan når nästa "position", d.v.s. rätt läge för körning av borr och stans. Skivmotorn slås då av och en timer (Ton1) initieras inför nästa sekvens. Detta innebär i första hand att ingången nollställs (IN:=0 eller IN:=FALSE). Detta krävs för att timern ska starta, eftersom detta sker vid positiv flank på IN. Värdet som ställs på tiden PT spelar ingen roll, eftersom PT kan ges ett nytt värde när timern startas (IN:=1 eller IN:=TRUE). Utgångarna Q och ET visar att vald tid har uppnåtts resp. hur lång tid som har gått. Q är alltså TRUE när ET>PT. Det sista som händer i sekvensen är att PU ettställs för att aktivera sekvens PU. Det första som händer i PU är sedan att STP nollställs, så sekvenserna inte körs samtidigt.

FBD-koden (i figur 5.7) för samma sekvens påminner mycket om ST-koden i logiken. I FBD behövs dock inte initiering av timern eftersom den grafiska FBD-timern i nästa sekvens alltid har en nolla på ingången när den är inaktiv. Därför startar den som den ska när en etta läggs på.



Figur 5.7. FBD-kod för sekvens STP. På samma sätt som i ST fortsätter koden lodrätt genom programmet. Skillnaden är att networks numreras i stället för rader.

LD-koden i figur 5.8 visar samma sak. Det som skiljer i den grafiska utformningen är att AND-funktionen skapas med parallellkopplade kontakter i stället för AND-blocken.



Figur 5.8. LD-kod för sekvens STP.

5.3.3 Sekvens PU (Pinne ut)

Skivan är nu i "position". Efter STP har nollställts går "Pinne" ut för att detektera en puck vid bormaskinen (Se figur 5.9). Nu startas timern som initierades i föregående sekvens. När en sekund har gått (och sekvensen fortfarande är aktiv) anses en puck funnen. (PPB=Puck i Position B). Därefter körs nästa IF-sats som initierar timern på nytt och ettställer villkoret för att gå vidare till nästa sekvens. Om pinnen däremot når sitt yttre läge inom en sekund (`_PinneUte=TRUE`) är facket tomt och PPB ska förbli nollställd, vilket sker eftersom den andra IF-satsen körs och därefter lämnas sekvensen innan villkoret för den första IF-satsen uppfylls.

```

27 IF PU THEN
28     STP:=FALSE;
29     Pinne_:=TRUE;
30     Ton1(IN:=1, PT:=T#1S, Q=>, ET=>);
31     IF Ton1.Q THEN
32         PPB:=TRUE;
33     END_IF
34     IF Ton1.Q OR _PinneUte THEN
35         Ton1(IN:=0, PT:=T#0S, Q=>, ET=>);
36         PI:=TRUE;
37     END_IF
38 END_IF

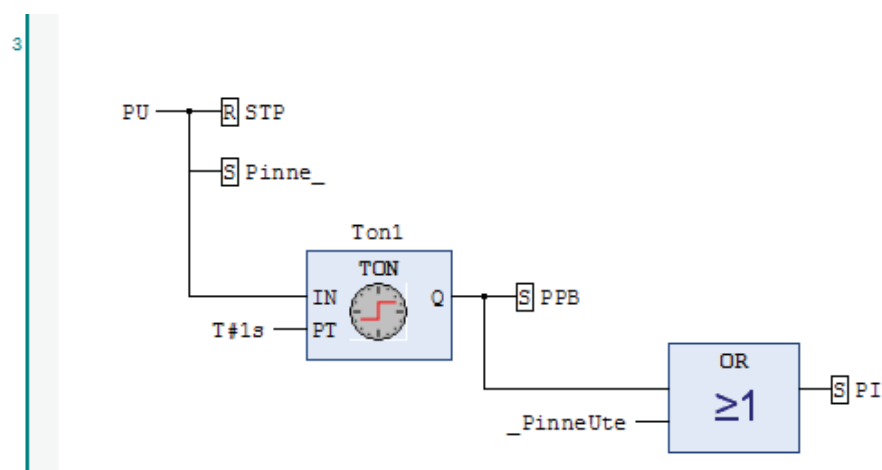
```

Figur 5.9. ST-kod för sekvens PU.

FBD-koden i figur 5.10 fungerar på samma sätt. Det är värt att notera att `_PinneUte` kan ettställa sekvens PI (vid OR-blocket) oavsett värdet på PU. En del av denna sekvens kan alltså i teorin köras utan att sekvensens huvudvillkor PU är aktivt. I praktiken är det dock inget problem eftersom utgången `Pinne_` inte ettställs i någon annan sekvens än denna, vilket därmed gäller även för ingången `_PinneUte`. OR-blocket körs alltså endast i samband med övergången mellan PU och PI.

Om man ändå vill ha exakt samma funktion som i ST kan ett AND-block på den nedre OR-ingången (PU AND `_PinneUte`). Detta sker då på bekostnad av likhet i syntaxen, eftersom inte "AND" används i ST-koden.

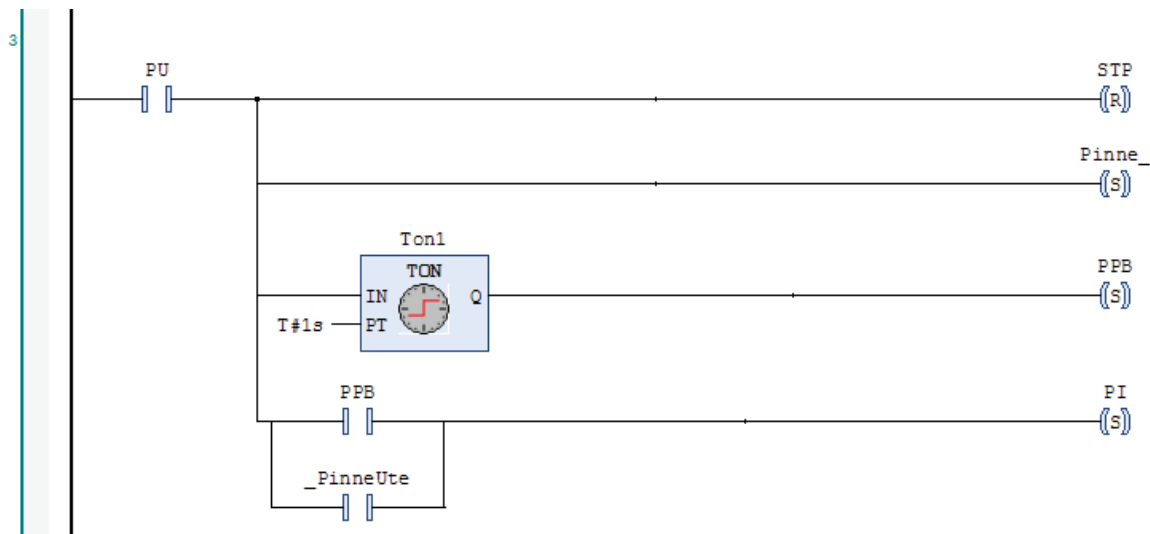
En annan detalj som kan nämnas är timerens utgång ET, som syns i ST-koden, men inte i de grafiska språken. Detta beror helt enkelt på att den inte använts och därmed har plockats bort. Det betyder inte att utgången inte finns, men den syns inte i grafiken. På samma sätt kan ingångar plockas bort om de inte används.



Figur 5.10. FBD-kod för sekvens PU.

LD-koden i figur 5.11 verkar till en början ha stora likheter med FBD. STP nollställs, pinnen går ut, och PPB ettställs om en sekund hinner gå innan sekvensen lämnas. Men den övre insignalen i OR-blocket, som i Ladder-logik byggs med två parallellkopplade kontakter, består inte av `Ton1.Q` utan i stället av `PPB`. Här ser vi alltså en tydlig skillnad från ST-koden,

i detta fall mer i syntax än i funktion, eftersom PPB kommer att ha samma värde som Ton1.Q så länge PU är aktiv (därefter nollställs Ton1.IN och Ton1.Q)



Figur 5.11. LD-kod för sekvens PU.

Anledningen till att man inte kan skapa OR-funktionen som sätter PI genom att parallellkoppla timern med _PinneUte är att kopplingens "utgång" då även påverkar PPB, som alltid kommer att ettställas samtidigt som PI (d.v.s. när sekvensen avslutas). Så fort man gör en OR mellan två signaler i Ladder har man alltså förbrukat möjligheten att använda signalerna var för sig, i samma koppling. Jämför med att två batterier med lite olika spänning parallellkopplas. Det går nu inte att mäta någon annan spänning än den gemensamma. Vid seriekoppling (jämför AND-funktionen) kan man fortfarande mäta över varje enskilt batteri.

I stället för Ton1.Q används alltså en kopia av denna, i detta fall PPB, i parallellkopplingen med _PinneUte. Nu uppnås samma resultat som i ST och FBD. Alternativt kan man koppla in en till Ton1 i stället för PPB i denna parallellkoppling, men det förändrar inte det faktum att syntaxen måste ändras jämfört med FBD. Dessutom är det svårt att göra en sådan koppling i CODESYS. Funktionen "Parallel contact" fungerar bara över andra kontakter, inte över funktionsblock. En kontakt är också enklare än en timer, både sett till funktion och till den grafiska utformningen. Tanken med att ge exempel i LD är att göra så mycket som möjligt av koden med klassiska ladderelement, och så lite som möjligt med funktionsblock. Det är dock tillåtet att använda funktionsblock i Ladder. Att bygga ladder-logik i FBD är däremot omöjligt eftersom ladderkontakterna förutsätter "spänningsmatningen" till vänster. Man kan konvertera ett network genom att kopiera från FBD och klistra in i LD, eller tvärtom, men ofta blir kopian mer komplex, eller helt felaktig, speciellt när timers eller liknande ingår.

5.3.4 Sekvens PI (Pinne in)

Detekteringen är gjord. PU nollställs och pinnen dras tillbaka. Timern startas igen för att ge pinnen tid att gå tillbaka (eftersom givare för pinnens innerläge saknas). En halv sekund senare nollställs timern för sista gången, och villkoret för nästa sekvens ettställs.

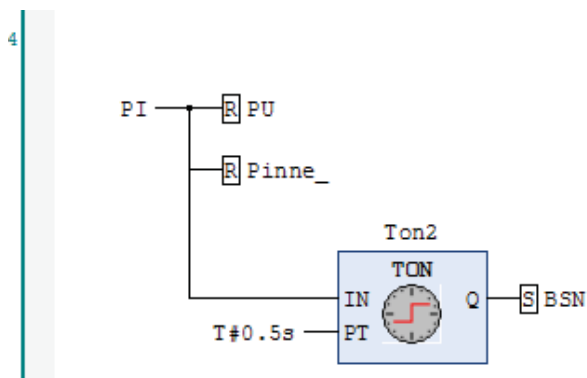
```

39 IF PI THEN
40   PU:=FALSE;
41   Pinne_:=FALSE;
42   Ton1(IN:=1, PT:=T#0.5S, Q=>, ET=>);
43   IF Ton1.Q THEN
44     Ton1(IN:=0, PT:=T#0S, Q=>, ET=>);
45     BSN:=TRUE;
46   END_IF
47 END_IF

```

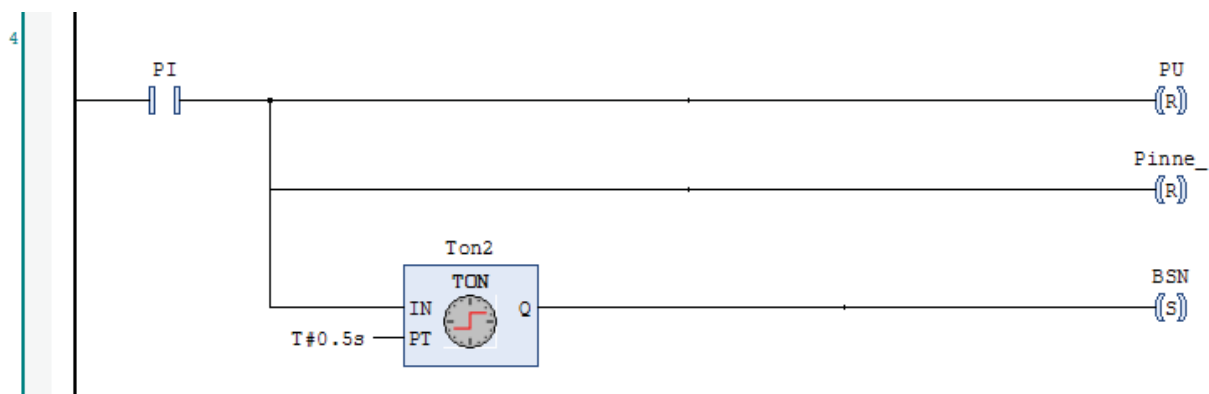
Figur 5.12. ST-kod för sekvens PI.

Jämför ST-koden i figur 5.12 och FBD-koden i figur 5.13. I ST återanvänds Ton1, medan FBD använder en ny timer, Ton2. Problemet med att använda Ton1 på två intilliggande sekvenser är att Ton1.IN aldrig hinner nollställas, eftersom PI ettställs innan PU nollställs. Alternativet är att Ton1.IN nollställs mellan PU och PI, men detta kräver mer kod. Enklast är därför att ha minst två timrar så ingen behöver köras två sekvenser i rad. I detta exempel används för FBD en ny timer för varje situation, totalt tre stycken. I ST behöver en timer nollställas, även om när den har använts på länge. Därför vinner man inget på att ha mer än en timer, förutsatt att inte flera tider ska mätas samtidigt.



Figur 5.13. FBD-kod för sekvens PI.

Figur 5.14 visar samma sekvens PI i LD-kod. Även här används Ton2 för att tidmätningen ska fungera korrekt.



Figur 5.14. LD-kod för sekvens PI.

5.3.5 Sekvens BSN (Borr och stans ner)

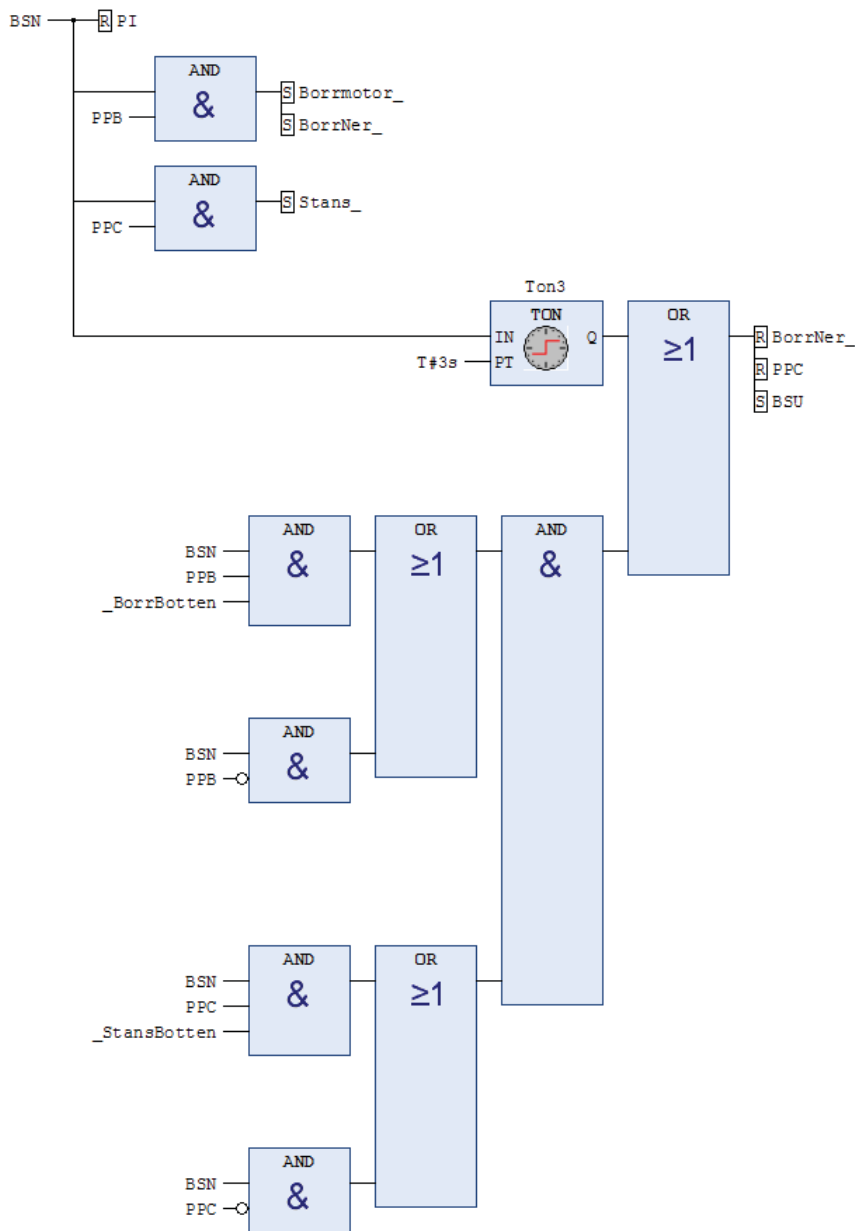
I sekvens BSN nollställs först PI och därefter startar borrmaskinen om det ligger en puck under denna, i position B. Motsvarande gäller stansen och position C. Timern används för att mäta tiden till borret och stansen nått sina bottenlägen, om de är aktiva. I ST-koden (figur 5.15) testas på rad 58 om det finns en puck i position B och att borret har borrarat i denna ända ner, alternativt att det inte finns någon puck. På samma sätt testas stansen. När dessa test är "godkända" eller tre sekunder har gått går programmet vidare. Utan timern skulle programmet låsa sig om borret eller stansen av någon anledning inte når sina bottenlägen. Eftersom IF-satser används i stället för WHILE handlar det inte om en loop som programmet inte kan ta sig ur. Däremot kommer inga fler IF-satser att bli aktiva och borret kommer att fortsätta att borra, vilket kan leda till slitage och överhettning. I sekvensens avslutning nollställs sedan BorrNer_, eftersom den bör vara passiv när borret sedan ska dras uppåt. PPC nollställs (detta måste göras före sista sekvensen körs, men måste inte ske exakt här) och slutligen sätts ingångsflaggan till nästa sekvens, BSU.

```
48 IF BSN THEN
49   PI:=FALSE;
50   IF PPB THEN
51     Borrmotor_:=TRUE;
52     BorrNer_:=TRUE;
53   END_IF
54   IF PPC THEN
55     Stans_:=TRUE;
56   END_IF
57   Ton1(IN:=1, PT:=T#3S, Q=>, ET=>);
58   IF ((PPB AND _BorrBotten) OR NOT PPB) AND ((PPC AND _StansBotten) OR NOT PPC) OR Ton1.Q THEN
59     BorrNer_:=FALSE;
60     PPC:=FALSE;
61     BSU:=TRUE;
62   END_IF
63 END_IF
```

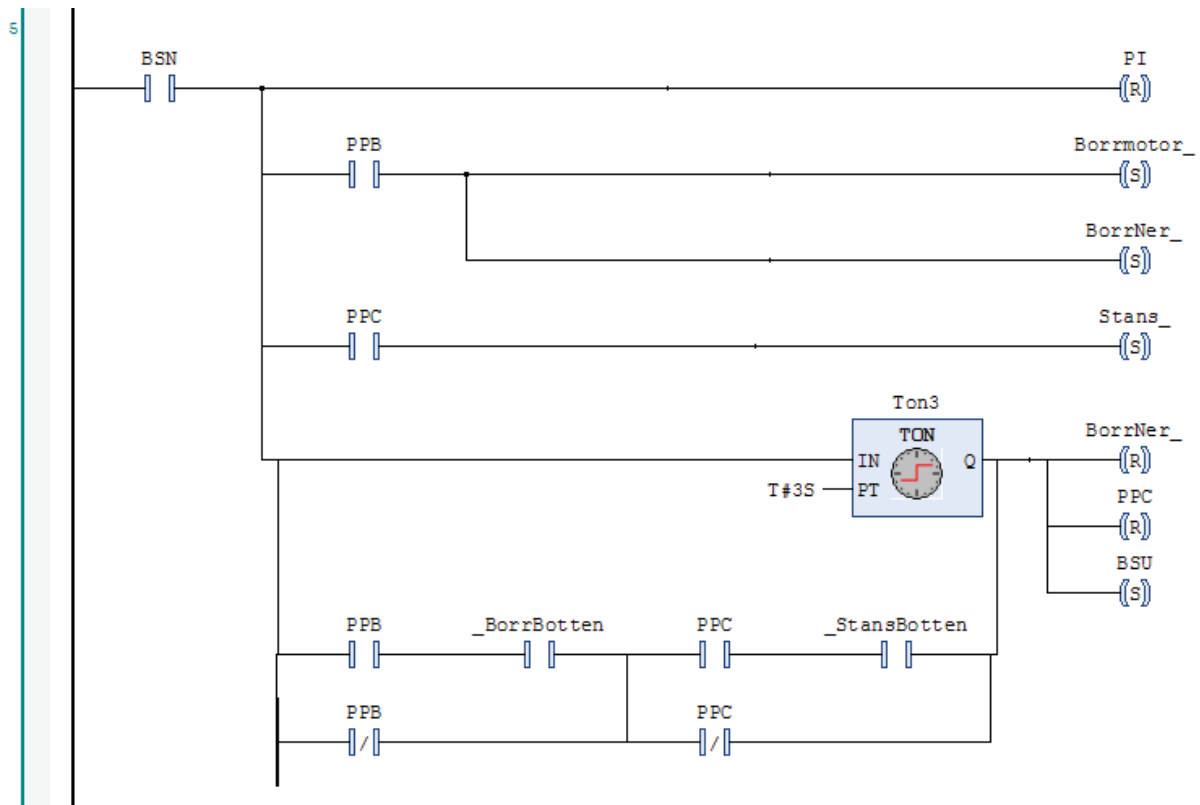
Figur 5.15. ST-kod för sekvens BSN.

FBD-koden är uppbyggd på samma sätt, men den grafiska utformningen av villkoret på rad 58 i ST-koden är mycket stor, se figur 5.16. Logiska block som ligger över och under varandra har en förmåga att ta mer plats än vad som borde behövas. Sekvensflaggan BSN krävs för att detta villkor ska kunna bli aktivt och måste därför läggas som ingång i de fyra AND-blocken i nedre vänstra delen av figur 5.16, alternativt i det "stora" AND-blocket till höger om dessa. En snyggare lösning skulle vara att koppla BSN med linjer från den övre ingången till de fyra AND-blocken. Detta skulle dock innebära extraarbete, om dessa linjer inte var inplanerade redan från början. Varken i FBD eller LD kan linjer dras på "fri hand".

LD-koden i figur 5.17 har helt andra proportioner. Det villkor som breddade ut sig i höjded på skärmen i FBD-koden utgör nu en minoritet av motsvarande network i LD, eftersom AND-funktionen "ligger vågrätt" och därmed inte kräver utrymme på höjden. Längst ner i figuren ses ett "stickspår" på ledningen. Detta är inget tryckfel i rapporten, utan ett fel i programmet som kan uppstå när man kombinerar serie- och parallellkoppling av kontakter. Funktionen påverkas dock inte.



Figur 5.16. FBD-kod för sekvens BSN.



Figur 5.17. LD-kod för sekvens BSN.

5.3.6 Sekvens BSU (Borr och stans upp)

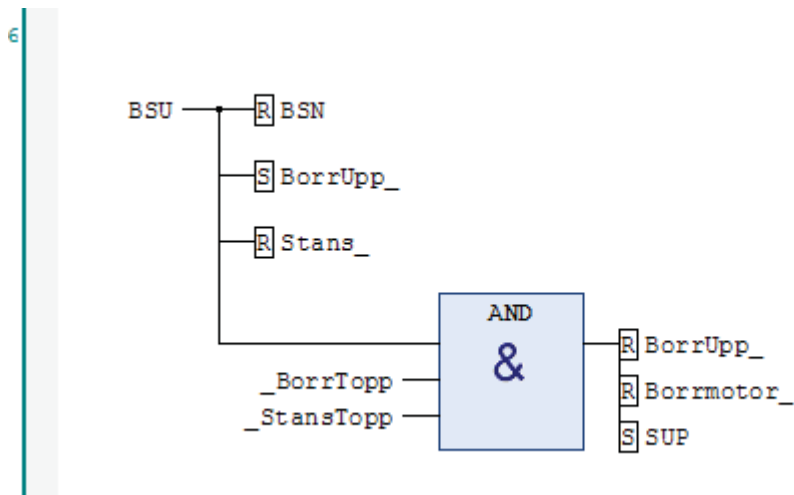
Bortsett från den grafiska utformningen (se figur 5.18, 5.19 och 5.20) är de tre språken identiska i sekvens BSU. Som vanligt inaktiveras först föregående sekvens. Borret går upp och Stans_ sätts till FALSE, vilket också innebär uppgång. Sedan väntar maskinen på att borr och stans nått sitt toppläge och därefter nollställs borrets uppgång och bormotorn, innan nästa sekvens aktiveras.

```

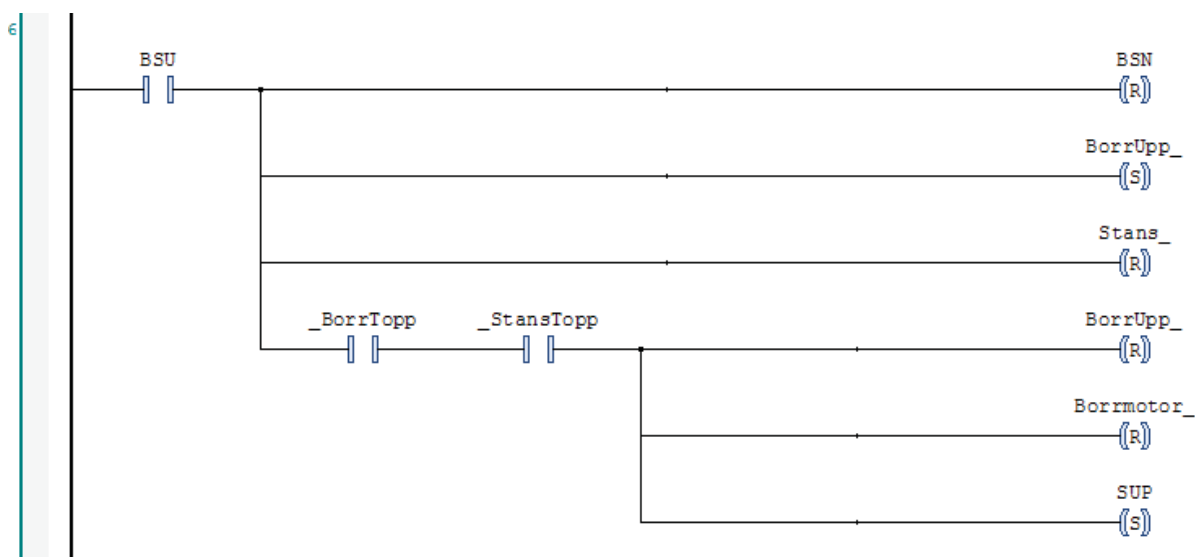
64 IF BSU THEN
65     BSN:=FALSE;
66     BorrUpp_:=TRUE;
67     Stans_:=FALSE;
68     IF _BorrTopp AND _StansTopp THEN
69         BorrUpp_:=FALSE;
70         Borrmotor_:=FALSE;
71         SUP:=TRUE;
72     END_IF
73 END_IF

```

Figur 5.18. ST-kod för sekvens BSU.



Figur 5.19. FBD-kod för sekvens BSU.



Figur 5.20. LD-kod för sekvens BSU.

5.3.7 Sekvens SUP (Stans ur position)

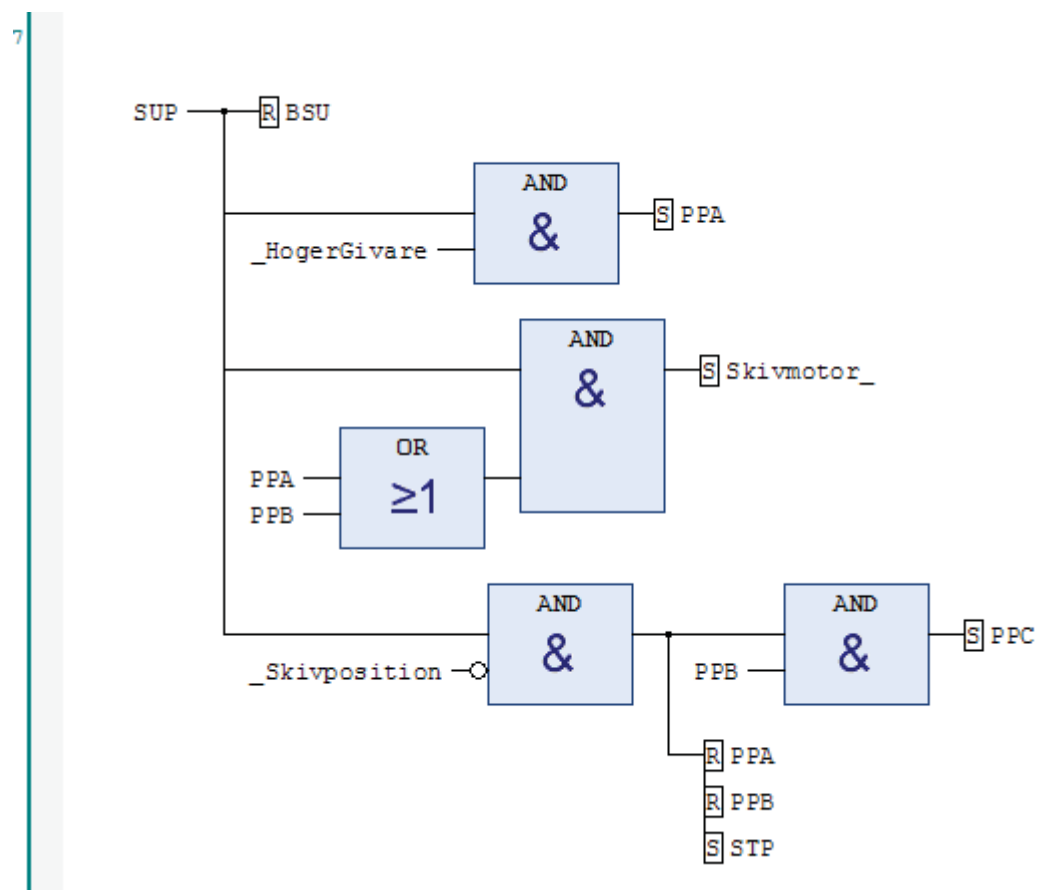
Denna sekvens körs när bearbetningen av puckar är avklarad och skivan ska rotera en kvarts varv till nästa position. Den första sekvensen, STP, kan inte köras direkt nu eftersom skivmotorn då inaktiveras när skivan är i rätt position, vilket den är nu. Därför behövs sekvens SUP som tar skivan *ur* position för att programmet ska kunna köras igen från början. Enligt figur 5.21, 5.22 och 5.23 nollställs BSU-flaggan och sedan testas om det finns en puck i position A (en position före bormaskinen). En puck i A eller B (PPB fick rätt värde i PU) får skivmotorn att gå, men bara så långt att den kommer ur position och STP därmed kan ta vid. Värdet i PPB kopieras till PPC, eftersom det inte finns någon givare i position C. När borr och stans körs är alltså PPC det värde som låg i PPB varvet före. Stansen kommer därmed att fungera korrekt, såvida ingen puck avlägsnas eller läggs till mellan position B och C. Därefter nollställs PPA och PPB inför nästa varv och sekvens STP är redo att starta.

```

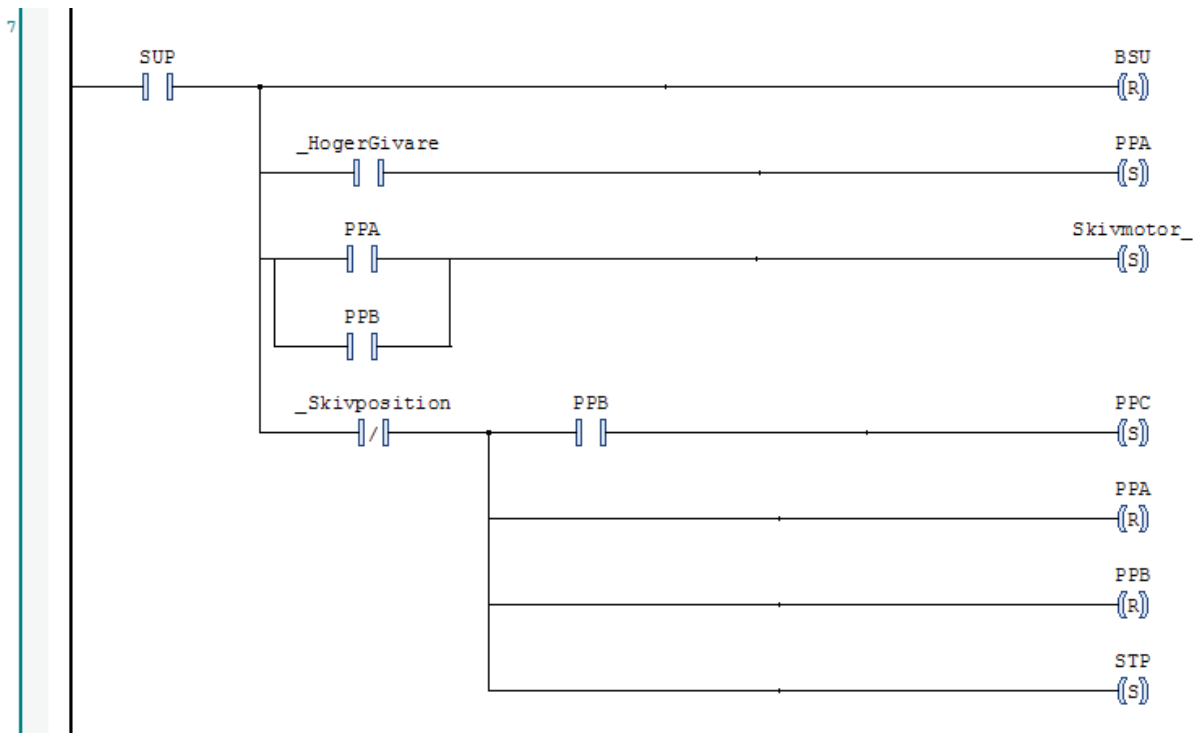
74 IF SUP THEN
75     BSU:=FALSE;
76     IF _HogerGivare THEN
77         PPA:=TRUE;
78     END_IF
79     IF PPA OR PPB THEN
80         Skivmotor_:=TRUE;
81     END_IF
82     IF NOT _Skivposition THEN
83         IF PPB THEN
84             PPC:=TRUE;
85         END_IF
86         PPA:=FALSE;
87         PPB:=FALSE;
88         STP:=TRUE;
89     END_IF
90 END_IF

```

Figur 5.21. ST-kod för sekvens SUP.



Figur 5.22. FBD-kod för sekvens SUP.



Figur 5.23. LD-kod för sekvens SUP.

5.4 Olika sätt att programmera

Precis som i andra utvecklingsmiljöer kan koden byggas upp på ett antal olika sätt. Ovanstående exempel får här namnet *sekvensorienterad programmering* eftersom varje IF-block eller network börjar med en sekvensflagga och körs ett i taget, alltid i samma ordning. Undantaget är Reset som alltid körs när startvredet är frånslaget. Denna typ av kod är ofta att föredra i den här typen processer där saker ska ske i en viss ordning. Både programmerare och andra människor är vana vid tidens gång och att aktiviteter görs enligt ett schema. Eftersom exekveringen alltid befinner sig på ett ställe åt gången (gäller även Reset) blir felsökningen enklare än om flera block eller networks kan jobba parallellt.

Ett annat sätt att göra koden är att utgå från utgångarna, den *utgångsorienterade programmeringen*. Man berättar då för processen hur varje del ska fungera, snarare än vad som ska hända i varje tidpunkt. I ST-kod sätts utgången helt enkelt genom en tilldelning, antingen på en rad, eller uppdelad på flera, om villkoret är komplicerat. I FBD och LD har ett network i detta fall bara *en* utgång, och denna beror på ingångar från processen. Med *utgång* menas i detta fall både styrsignaler till processen och lokala variabler, eftersom dessa, till skillnad från ingångarna, får sitt värde i programmet. Sekvensflaggorna behövs inte eftersom det inte finns några sekvenser.

Följande ST-exempel visar hur PPA sätts med utgångsorienterad programmering. Koden ger inte exakt samma resultat eftersom den körs hela tiden och inte bara i en del av programmet. Men den uppfyller ändå sitt mål, att ta reda på om det finns en puck i position A när skivan är i rätt läge. (När skivan är i fel läge kommer undersidan av denna att uppfattas som en puck. PPA ska alltså inte vara ettställd bara för att _HogerGivare är det.)

```
PPA:=_Skivposition AND _Hogergivare
```

Att hitta ett villkor för att skivmotorn ska köras är betydligt svårare. För det första måste vredet vara påslaget, annars måste alla rörliga delar stå stilla. Skivmotorn ska köras när den är ur position, men också när bearbetningen är avklarad i en position och det finns en inkommande puck i position A eller B.

```
Skivmotor :=_Vred AND (NOT_Skivposition OR ((PPA OR PPB) AND ...))
```

Det som saknas i ovanstående rad är alltså att testa om bearbetningen är avklarad. Ingångarna *_BorrTopp* och *_StansTopp* kan verka användbara, men de säger bara om skivan *kan* köras utan risk, inte att bearbetningen är avklarad. Den utgångsorienterade programmeringen passar alltså bäst när utsignalerna bara ska bero på insignalernas värde, men inte när de dessutom ska variera beroende på tiden. Ibland är det en filosofisk fråga vilket man väljer, men det kan också handla om vilka fel som kan dyka upp och vilken felsökning som är bekvämast. När ett fel uppstår, vet man då alltid i vilken sekvens detta sker? Eller underlättas felsökningen av att man vet vilken utgång som beter sig underligt och därmed kan begränsa felet till denna del av programmet. Dessa frågor kan avgöra hur man bygger koden. Vilket av de tre språken man väljer har mindre betydelse.

Observera att den sekvensorienterade programmeingen inte *kräver* att sekvenserna körs i en speciell ordning. Ordningen kan variera och flera sekvenser kan köras samtidigt. Felsökningen kan dock bli svårare om man inte vet på vilken "skärmsida" man ska leta efter ett fel.

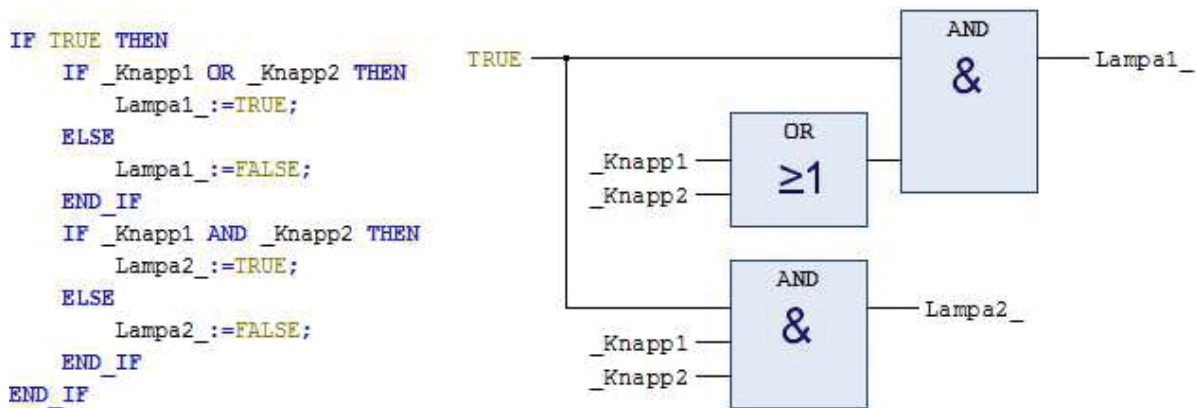
Om man i stället vill låta ingångarna stå i fokus kan man välja den *ingångsorienterade programmeringen*. En anledning kan vara att man vill kunna lägga till givare och tryckknappar till en process, utan att skriva om koden på många ställen, utan bara lägga till ett block eller network.

Som exempel väljs ingången *_HogerGivare*. Även här väljs ST, men koden kan göras identisk i FBD (med AND-block) eller LD (med parallellkoppling).

```
IF _HogerGivare THEN
  IF _Skivposition THEN
    PPA:=TRUE;
  END_IF
END_IF
```

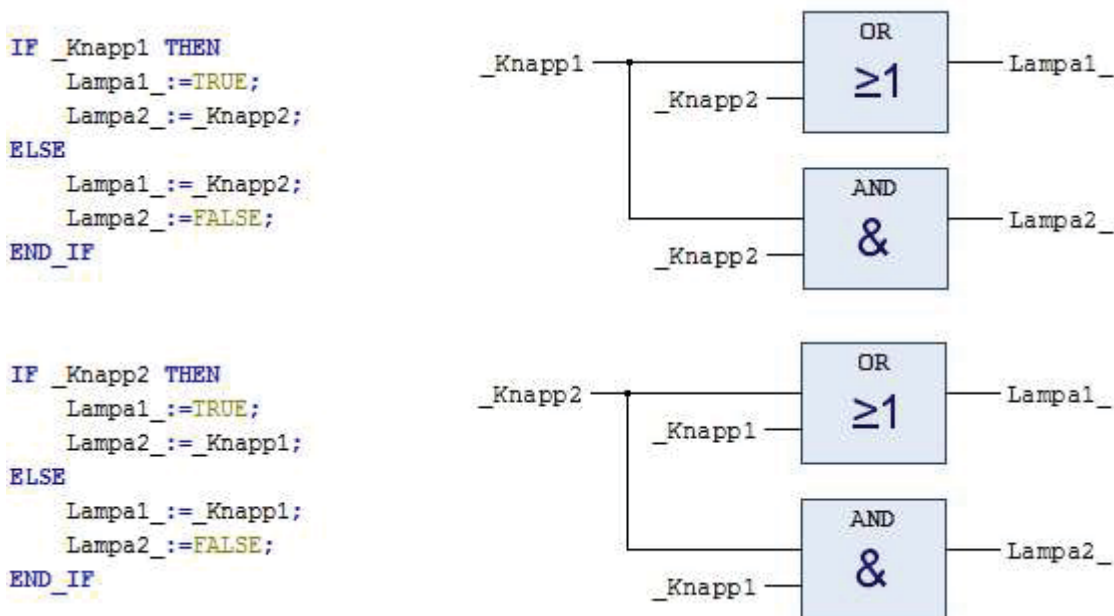
Problemet här är att *mer än en* ingång måste användas, vilket innebär att ingångarna kommer att finnas på mer än ett ställe i koden. Felsökningen blir svårare och ännu värre blir det för ingångar som används i mer än en sekvens.

För att jämföra de tre programmeringssätten tydligare används en enklare process som exempel. Ingångarna är två tryckknappar och utgångarna två lampor, varav den första är tänd när minst en knapp är intryckt och den andra när båda är intryckta. Figur 5.24 visar sekvensorienterad programmering för denna process i ST respektive FBD.



Figur 5.24. Exempel på sekvensorienterad programmering. Varje programdel börjar med en sekvensflagga till vänster.

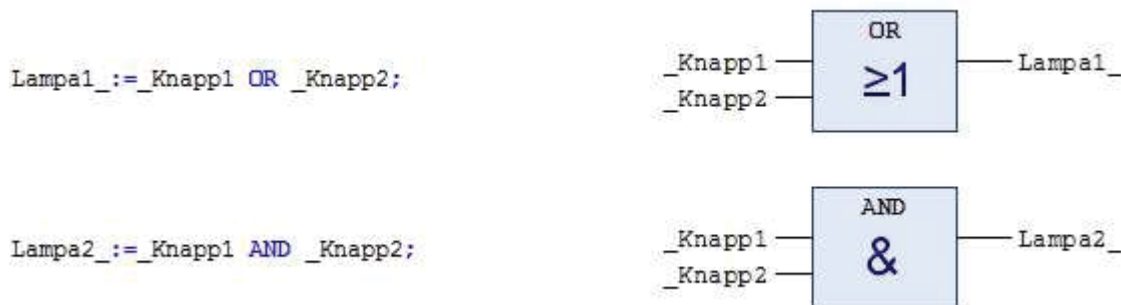
Sekvensvillkoret ligger längst upp till vänster, i båda språken. Detta villkor är satt till TRUE och verkar därmed vara meningslöst, precis som det övre AND-blocket i FBD. Detta gäller dock bara tidsoberoende processer där det bara finns en sekvens, och denna ska köras hela tiden. Figur 5.24 visar alltså det generella tankesättet, som gäller oavsett antal sekvenser.



Figur 5.25. Exempel på ingångsorienterad programmering. Varje programdel börjar med exakt en ingång till vänster.

Ingångsorienterad programmering ger i stället koden som visas i figur 5.25. Resultatet blir att båda lamporna kommer att få sina värden två gånger, vilket i sig inte är en fördel. Någon nackdel är det inte heller så länge båda IF-satserna eller networks ger samma värde till respektive lampa. Fördelen är att koden ger bra överblick om operatören vill veta vad som händer när han trycker på t.ex. knapp 1. Han vet då att bara den övre halvan av koden är aktuell och behöver inte titta på fler ställen. I ett enkelt exempel som detta kan man överväga denna typ av kod, men som nämnts innan blir det svårt att felsöka ett program där samma

ingång används på många ställen. Vid stora program är nästan alla andra sätt att programmera bättre än att slaviskt följa denna form av ingångsorienterad programmering.



Figur 5.26. Exempel på utgångsorienterad programmering. Varje programdel har exakt en utgång, till vänster i ST och till höger i FBD.

Den utgångsorienterade varianten ses i figur 5.26. Programmet är kompakt, men också lättläst. Som tidigare har visats är denna programmeringsmetod ofta svåränvänd när utgångarna inte bara beror på ingångarna, utan även ska följa ett kronologiskt schema.

Däremot kan man ofta lättare uppnå de resultat man vill ha genom att kombinera de tre typer, som t.ex. i figur 5.27. Nackdelen med att blanda för mycket är att felsökningen kan bli betydligt svårare i takt med att koden byggs ut.

```

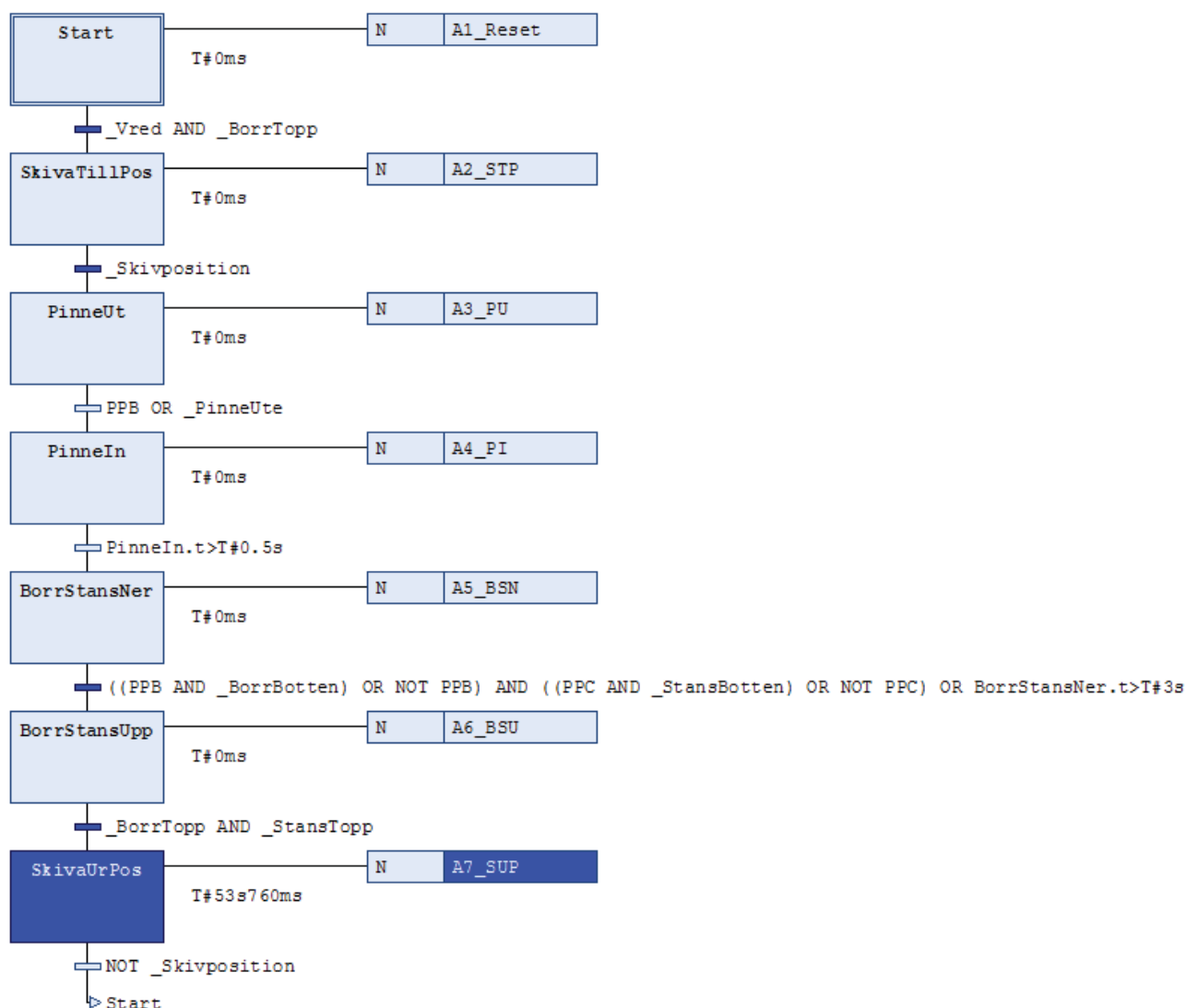
IF Sekvens1 OR Sekvens8 THEN
    Utgang1_ := _Ingang1 OR LokalVariabel3;
    Utgang3_ := Utgang1_ AND Sekvens12;
END_IF
IF _Ingang4 AND Sekvens2 THEN
    LokalVariabel2 := _Ingang1 XOR LokalVariabel9;
    Utgang20_ := LokalVariabel2 AND _Ingang1 AND _Ingang7;
END_IF

```

Figur 5.27. Exempel på hur olika programmeringsmetoder kan kombineras. Koden kan exekveras på många ställen samtidigt vilket i större program ger sämre överblick och svårare felsökning.

5.5 Programkod i SFC

Det fjärde språket som studeras i denna manual är till sin natur ganska annorlunda och får därför ett eget avsnitt. Språket är SFC, **Sequential Function Chart**. Koden bygger i stort sett på två "nivåer", den yttre nivån som ser ut som ett flödesschema, och den inre nivån som använder kod från något av de andra språken. SFC kan alltså inte användas ensamt utan är ett "skal" för övriga språk som kan användas när ett sekventiellt förlopp ska programmeras.



Figur 5.28. SFC-kod för processen. Koden består av ett flödesschema som visar den övergripande programstrukturen. I varje action (rutorna till höger) finns kod i ett annat av språken, som styr händelserna i respektive steg. En action kan skrivas i valfritt språk, men om den skrivs i SFC måste denna kod i sin tur anropa egna actions med kod. Att skriva ett PLC-program med bara SFC går alltså inte.

Exekveringen startar i rutan med dubbla kanter (i figur 5.28 heter denna "Start") och fortsätter lodrätt steg för steg, i takt med att villkoren längs den lodräta linjen uppfylls. Den aktiva rutan är blåmarkerad, och även de *actions* som för tillfället körs, i detta fall A7_SUP. I figur 5.28 är fyra av villkoren uppfyllda (blåmarkerade), men eftersom programmet körs i sekvens kommer inget att hända förrän villkoret under den aktiva rutan uppfylls, d.v.s. "NOT _Skivposition". Tiden som en ruta har varit aktiv visas till vänster om denna, i detta fall nästan 54 sekunder. Vi tittar in i A7_SUP (figur 5.29) för att se varför exekveringen har stannat och vad som krävs för att programmet ska köras vidare.

A7_SUP och även övriga actions är delade i två delar, varav den ena körs när vredet är påslaget och den andra när det är frånslaget. Eftersom SFC-koden körs endast ett varv per processcykel måste exekveringen kunna brytas i varje steg. Detta är det enklaste sättet att lösa detta, om man vill att programmet ska fortsätta på samma ställe när vredet slås på igen. Jämför med de andra språken där ett helt programvarv tar 20 millisekunder, och resetsekvensen därmed bara behöver finnas en gång per varv.

Till stor del är första delen av A7_SUP identisk med Sekvens SUP i figur 5.21. Det som skiljer är att sekvensflaggorna SUP, BSU m.fl. inte behövs, eftersom SFC-koden reglerar övergångarna mellan sekvenser.

Villkoret för att körningen ska fortsätta i figur 5.28 är NOT _Skivposition, vilket innebär att skivan nu är i position, och villkoret därmed inte är uppfyllt. Det som får skivan ur position är skivmotorn och vi ser alltså i figur 5.29 att PPA eller PPB måste ettställas för att motorn ska köras (förutsatt att vredet inte är frånslaget). När en puck läggs i position A kommer alltså exekveringen att fortsätta neråt i figur 5.28 och enligt pilen "hoppa" till Start. Resetsekvensen i Start är som vanligt endast aktiv när vredet är frånslaget. När villkoret _Vred AND _BorrTopp är uppfyllt fortsätter exekveringen till nästa ruta. Övriga actions är uppbyggda motsvarande A7_SUP.

Timerfunktionen TON behövs inte i SFC, eftersom det finns en inbyggd funktion som mäter tiden en ruta har varit aktiv. Villkoret Pinneln.t>T#0.5s är t.ex. uppfyllt när programmet har kört rutan Pinneln i över en halv sekund. Denna timer nollställs när rutan lämnas. Den tid som syns i figur 5.28, T#53s760ms finns på samma sätt i variabeln SkivaUrPos.t.

5.6 Att göra egna funktionsblock

I de flesta programspråk kan man anropa olika former av subrutiner, procedurer eller funktioner, vilket ger både tidsbesparing och bättre struktur i koden. Motsvarande i PLC-programmering kallas funktionsblock. Det är alltså i grunden samma sak som de standardfunktionsblock som visas i avsnitt 4.2, men när man gör egna block är ofta antalet in- och utgångar fler och komplexiteten betydligt större. Ett funktionsblock programmeras i valfritt språk, men när de används (anropas) måste man använda ett program i FBD, LD eller ST.

Syftet är att "slå ihop" en större kod till ett kompakt block av in- och utgångar. Extra värdefullt blir detta om koden ska användas på många ställen, om man enkelt vill kunna distribuera koden till andra användare, eller om man vill hålla koden hemlig, som en svart låda där användaren får disponera in- och utgångar, men inte får veta exakt vad som händer inne i blocket.

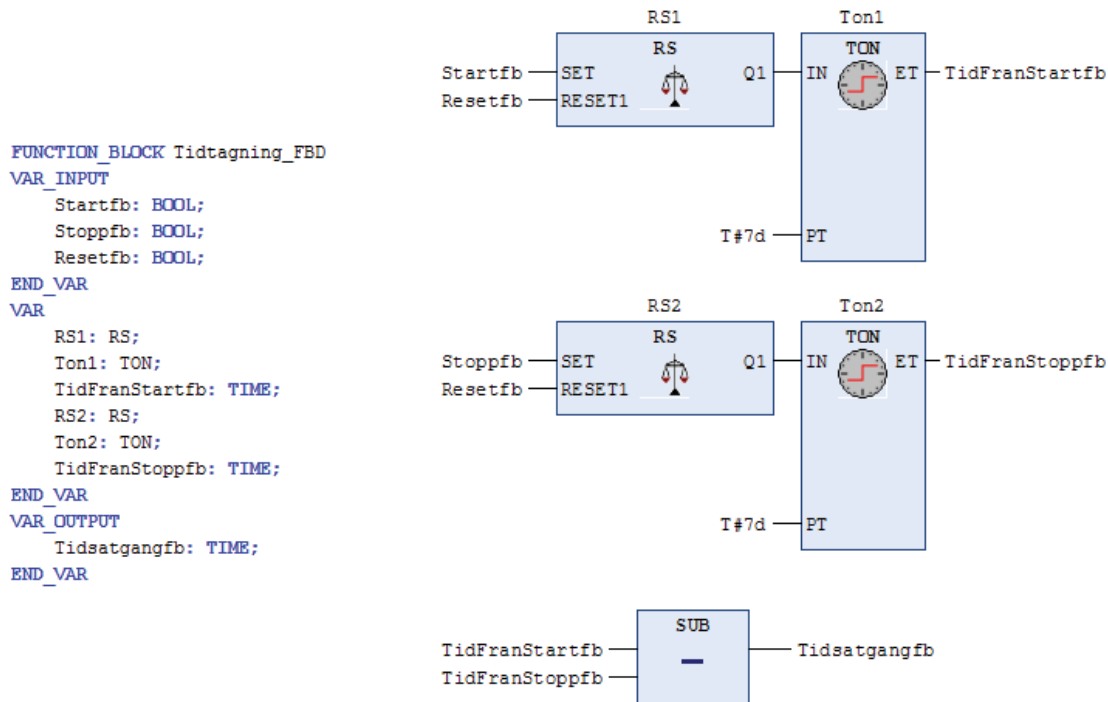
```

IF _vred THEN
  BorrUpp_:=FALSE;
  Borrmotor_:=FALSE;
  IF _HogerGivare THEN
    PPA:=TRUE;
  END_IF
  IF PPA OR PPB THEN
    Skivmotor_:=TRUE;
  END_IF
  IF NOT _Skivposition THEN
    IF PPB THEN
      PPC:=TRUE;
    END_IF
    PPA:=FALSE;
    PPB:=FALSE;
  END_IF
ELSE
  Skivmotor_:=FALSE;
  Pinne_:=FALSE;
  Borrmotor_:=FALSE;
  BorrNer_:=FALSE;
  BorrUpp_:=TRUE;
  Stans_:=FALSE;
END_IF

```

Figur 5.29. Exempel på innehållet i en action, i detta fall A7_SUP. En action kan skrivas i valfritt språk, i detta fall ST.

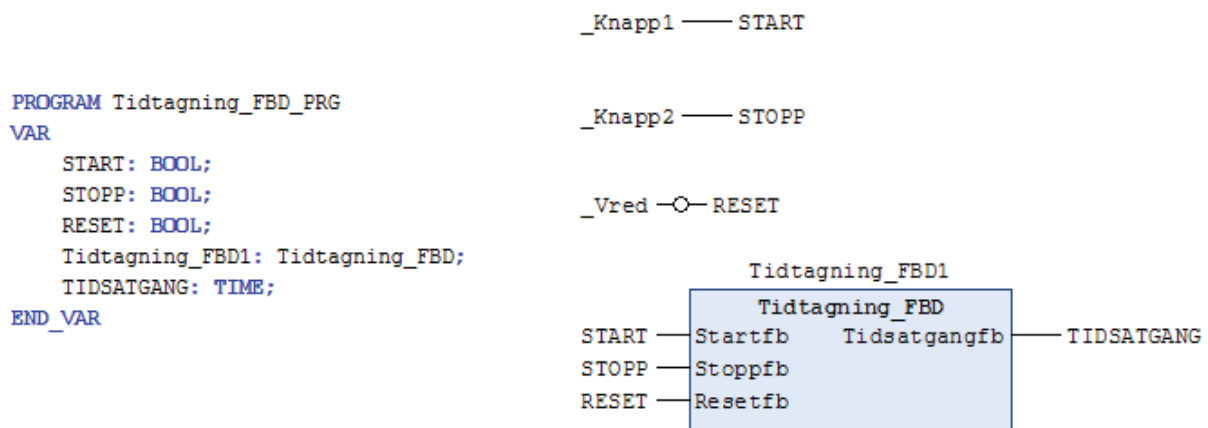
I figur 5.30 visas till höger ett enkelt tidtagarur, som kan användas t.ex. vid olika idrottsevenemang. Först startas Ton1 startas med en etta på Startfb (som sätter en kvarstående etta på RS-vippans utgång Q1) och vid "målgång" startas Ton2 med en etta. Programmet lägger differensen i Tidsatgangfb, som alltså är resultatet av tidsmätningen. Tidtagaruret nollställs med en etta på resetfb. Tiden vid PT anger när Ton1 respektive Ton2 stoppas, och måste därför vara en lång tid, t.ex. sju dagar.



Figur 5.30. Exempel på hur ett funktionsblock definieras. Till vänster visas deklaringen av de lokala variablerna i blocket. Tanken är att de fem blocken till höger ska kunna "anropas" från ett program genom att bara använda ett block.

Till vänster om FBD-koden visas deklaringen av de variabler som används. De är indelade i ingångar, lokala variabler och utgångar. Tillägget fb i variabelnamnen används för att skilja blockets egna variabler från de yttre variabler som används när blocket sedan ska anropas. De vippor och timer-block som används räknas som lokala variabler.

Om funktionsblocket sedan ska anropas från ett FBD-program hämtas en tom "box" som sedan görs om till rätt typ, i detta fall Tidtagning_FBD (se figur 5.31). Nu har de fem blocken i figur 5.30 slagits ihop till ett. De variabler som syns i blocket till vänster och höger, är de som är deklarerade som VAR_INPUT respektive VAR_OUTPUT i figur 5.30. Det är nu dags att fylla på in- och utgångar med variabler. Variablerna skrivs nu utan "fb" och dessutom med versaler för extra tydlighet. I detta exempel kopplas två knappar och vredet från borrprocessen till START, STOPP och RESET, för att tidtagaruret enkelt ska kunna testas. Knapparna och vredet är globala variabler och ska därför inte deklarerars i detta POU.



Figur 5.31. Här används det funktionsblock som deklarerades i figur 5.30. Funktionen hos det nya blocket motsvarar de fem gamla blocken tillsammans. Till vänster visas variabeldeklarationen för programmet till höger. Här deklarerar in- och utsignaler och blockets instans, men inget av det som händer internt i funktionsblocket.

Vid körning av programmet kommer funktionsblocket att arbeta som planerat, och TIDSATGANG är alltså "en funktion av" START, STOPP och RESET. Värdet på variablerna, både de logiska ingångarna och TIDSATGANG syns i körläge till höger om ordet, både när klockan räknar upp och när den är stoppad.

Samma sak som har visats ovan kan göras i ST, enligt figur 5.32. (Variabeldeklarationen görs på motsvarande sätt och visas inte här.)

```

RS1 (SET:=Startfb, RESET1:=Resetfb, Q1=>);
Ton1 (IN:=RS1.Q1, PT:=T#7D, ET=>);
RS2 (SET:=Stoppfb, RESET1:=Resetfb, Q1=>);
Ton2 (IN:=RS2.Q1, PT:=T#7D, ET=>);
Tidsatgangfb:=Ton1.ET-Ton2.ET;

START:=_Knapp1;
STOPP:=_Knapp2;
RESET:=NOT _Vred;
Tidtagning_ST1 (Startfb:=START, Stoppfb:=STOPP, Resetfb:=RESET, Tidsatgangfb=>);
TIDSATGANG:=Tidtagning_ST1.Tidsatgangfb;

```

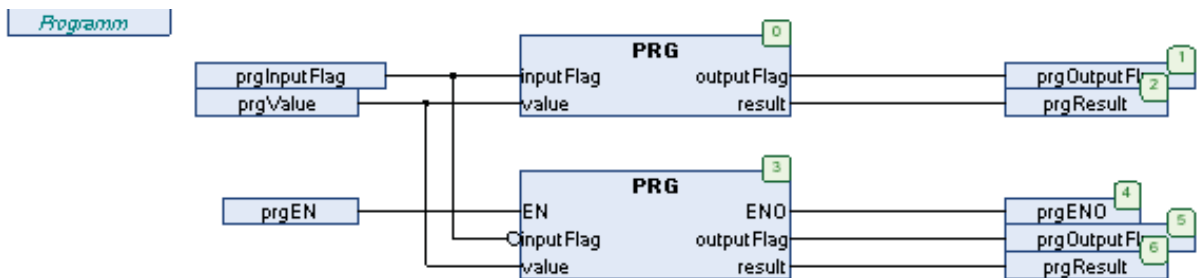
Figur 5.32. Definition (övre halvan) och anrop (nedre halvan) av funktionsblock i ST.

De fem övre raderna är samma funktionsblock som i figur 5.30, men nu i textversion med namnet Tidtagning_ST. De fem nedre raderna motsvarar figur 5.31. TIDSATGANG sätts på sista raden till det värde som funktionsblockets instans Tidtagning_ST1 har på utgång Tidsatgangfb. Sista raden visar alltså hur blockets utgång används, medan raden före visar vilka värden som matas in.

Definitionen av ett funktionsblock och anropet av detta är alltså två separata delar, och kan skrivas i olika språk. Man kan t.ex. bygga ett block i FBD som anropas från ett ST-program, eller bygga ett block i ST som anropas från ett LD-program.

5.7 Programkod i CFC

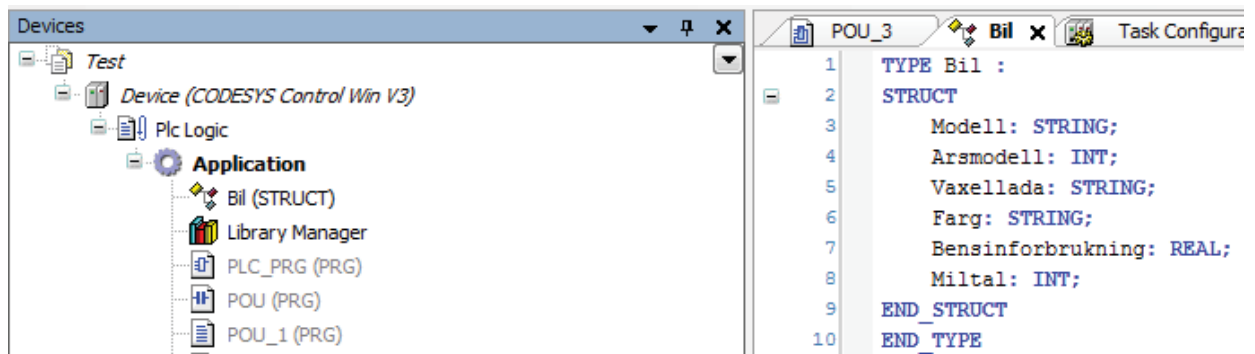
CODESYS har också ett språk som ligger utanför standardspråket i IEC61131-3. Det heter CFC, *Continuous Function Chart*. Till stor del påminner det om FBD och därmed görs ingen komplett genomgång. Skillnader som kan nämnas är att kodytan inte är indelad i networks och att blocken får placeras på valfri plats, inte bara på "Start here". Användaren får själv dra ledningar mellan dem (men även en Auto Connect-funktion finns). När CFC används så finns en meny som heter CFC som ersätter menyn FBD/LD/IL.



Figur 5.33. I CFC placeras blocken valfritt och är inte knutna till networks. Ledningarna dras manuellt och får korsa varandra. Programmeringen är generellt sätt mer flexibel och koden kan göras mer lättläst. De gröna siffrorna är en inbyggd numrering av utgångarna. Exemplet kommer från den inbyggda hjälpfunktionen i CODESYS.

Förutom designskillnader finns också andra skillnader jämfört med FBD, t.ex. blocken Composer och Selector, som visas i det högra verktygsfältet, när ett CFC-program har skapats.

För att demonstrera hur dessa används skapas först en STRUCT. Högerklicka på **Application** i trädstrukturen, Välj **Add Object / DUT...** Nu ska en "data unit type" skapas, i detta fall en Structure. Välj namn och definiera strukturen, t.ex. enligt figur 5.34

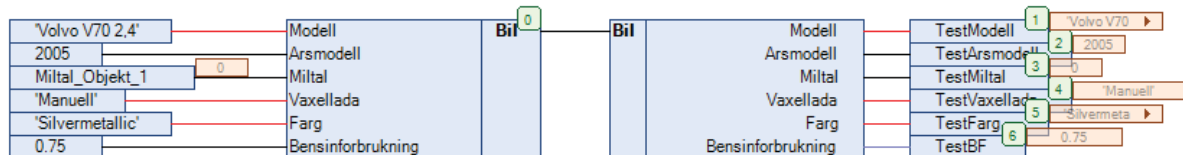


Figur 5.34. En STRUCT har skapats med namnet Bil. Den innehåller sex variabler med lämpliga datatyper. Strukturen visas precis under Application i trädstrukturen.

När strukturen Bil är definierad kan den användas av en Composer i kodytan genom att dess tre frågetecken (???) ersätts med ordet Bil. Syftet med en Composer är att packa ihop informationen till ett enda "paket" som skickas vidare. Linjen i mitten av figur 5.35 "innehåller" alltså all information om bilen, både strängar och tal. Ingångarna på en Composer ska alltså

ha olika datatyper, medan utgången ger en struktur. En Selector är motsatsen, ingången ska ha en struktur och utgångarna får datatyper, enligt strukturens definition.

Expression	Type	Value	Prepared value	Address	Comment
TestModell	STRING	'Volvo V70 2,4'			
TestArsmode	INT	2005			
TestMiltal	INT	0			
TestVaxellada	STRING	'Manuell'			
TestFarg	STRING	'Silvermetallic'			
TestBF	REAL	0.75			
Miltal_Objekt_1	INT	0			



Figur 5.35. Sex parametrar matas in i en Composer, som skickar vidare informationen som ett "paket". Paketet kan sedan sparas i en variabel och användas på andra ställen. I detta fall packas innehållet upp igen av en Selector och fördelas på respektive utgång. Detta exempel gäller en speciell bil. Ingångarna är därmed konstanter, med "Miltal" som undantag, där variabeln Miltal_Objekt_1 är inkopplad. Om man ändrar värde på denna variabel kommer resultatet att synas på utgången TestMiltal.

I CFC kan man också välja att ha olika färger på ledningar beroende på vilken datatyp de transporterar. (**Tools / Options... / CFC editor / View / Edit Line Colors...**). I figur 5.35 har svart färg valts för INT, röd för STRING och blå för REAL. I fallet med REAL fungerar det till höger, men av okänd anledning inte till vänster, vid ingången Bensinforbrukning.

Referenser

- [1] Hämtas från sidan: <http://www.codesys.com/download.html>. CODESYS är gratis och sista utgåvan av programmet finns att hämta utan inloggning. För tidigare versioner och inloggning kontakta 3S-Smart Software Solutions GmbH.
- [2] till [6] Finns i Supportdatabas www.beijer.se, gå in på Support Online och sök i databasen på filnamnet (utan filändelse).
- [7]. Köps från Beijer Electronics AB.

Appendix 2

Programmering i iX Developer

Komma-igång-manual

Innehåll

1 Inledning	I
2 Installation och konfiguration	II
2.1 Systemkrav	II
2.2 Installation av programmet	II
3 Skapa nytt projekt	III
4 iX Developer och Mitsubishi Q02-system	IV
5 Menyerna och grundläggande funktioner	VII
5.1 Home	VII
5.2 Project	IX
5.3 System	IX
5.4 Insert	X
5.5 View	X
5.6 Dynamics	XI
5.7 Actions	XI
6 Programmeringsexempel	XII
6.1 Skapa ett enkelt grafiskt objekt som ändrar färg	XII
6.2 Skapa en hastighetsvisare	XIV
6.3 Larmlista	XV
6.4 Nollställning av larm	XVII
Referenser	XIX

1 Inledning

iX Developer är en utvecklingsmiljö baserad på Windows CE utvecklad av Beijer Electronics. Mjukvaran iX Developer används för att konfigurera PC-styrda tillämpningar och operatörspaneler. Alla grundläggande funktioner som behövs för att konstruera en processbild finns i iX Developer. Programmeringsspråket är grafiskt och av drag- och släpp typ. Länkar till datavärden kallas tags och dessa knyts till objekt som ska visas på skärmen, intervall ställs in då ett objekt ska visas eller röra sig på skärmen. Ett antal fördefinierade funktioner för larmhantering, reglage och grafiska visare finns tillgängliga med det går även att bygga egna funktioner.

2 Installation och konfiguration

Nedanstående avsnitt behandlar installation och systemkrav. Observera att mjukvara, drivrutiner och protokoll kan ha uppdaterats efter att USB-stickan framställdes. Det är därför rekommenderat att den i iX Developer inbyggda uppdateringsfunktionen används före ett projekt skapas.

2.1 Systemkrav

Följande krävs av datorn som iX Developer ska installeras på:

- Microsoft Windows 7, Microsoft Windows Vista eller Microsoft Windows XP SP3.
- Minst 2 GB RAM-minne.
- Processor på minst 2 GHz.
- Grafikkort: Tier 2, DirectX version: 9.0 eller senare, Video RAM: 120 MB eller större,
- Pixel shader: version level 2.0 eller högre, Vertex shader: version level 2.0 eller senare och Multitexture units: 4 eller mer.

2.2 Installation av programmet

iX Developer levereras på en USB sticka [1]. Installationen startas automatiskt när USB-stickan sätts in i porten i datorn. Följ installationsguiden som pågår under installationen. Om installationen inte startas automatiskt kör filen setup.exe file. En ikon skapas under installationen och läggs på skrivbordet, därifrån kan programmets startas eller genom att klicka på Start/All Programs/iX Developer/ iX Developer.

3 Skapa nytt projekt

När programmets startats välj

Create New Project

I nästa fönster väljs vilken skärm och ev. rotation (om skärmen ska användas på högkant) som ska avses och tryck

Next

Välj controller (styrsystem) och protokoll, för Mitsibishis Q02-system välj:

MELSEC och **MC Protocol** därefter **Next**

Välj controller (styrsystem) och protokoll, för SoftControl välj:

CoDeSys och **Direct Access** därefter **Next**

Namnge samt ange plats där projektet sparas och tryck

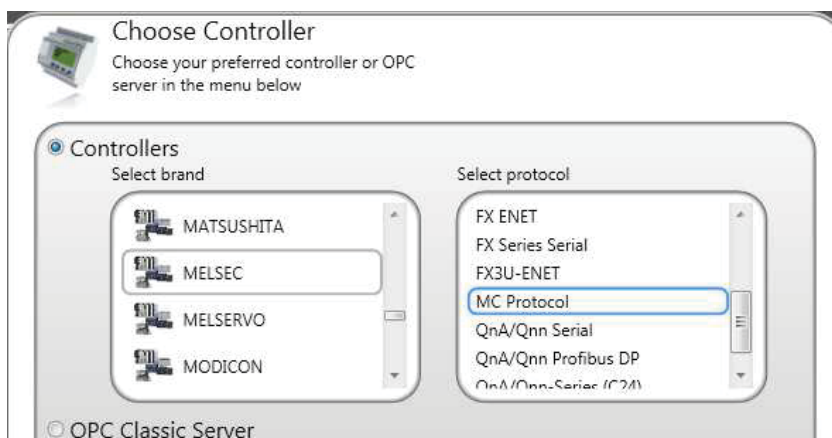
Finish

4 iX Developer och Mitsubishi Q02-system

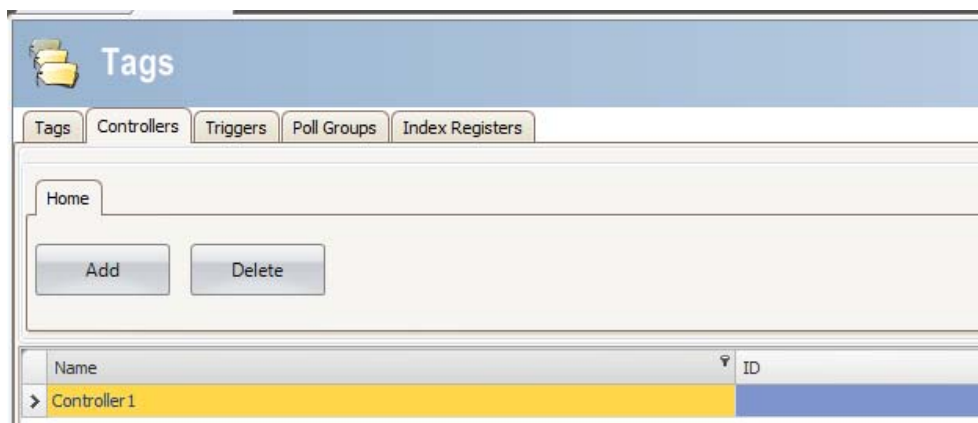
iX Developer går att använda tillsammans med Mitsubishi Q02-system och GX Developer. Observera att PLC:n, PC:n och operatörspanelen måste vara korrekt kopplade till varandra samt att Ethernetkommunikation måste vara upprättad mellan panelen och PLC:n innan nedanstående utförs. Ett test genom att "pinga" PLC:n och panelen från PC:n på deras respektive IP-adresser är att rekommendera.

För att kunna använda variabellistan från GX måste följande göras:

- Starta ett nytt, tomt projekt i GX Developer.
- Skapa ett styrprogram och en globalvariabellista.
- Starta iX Developer, skapa ett nytt projekt och välj MELSEC som controller och MC Protokoll enligt bilden.

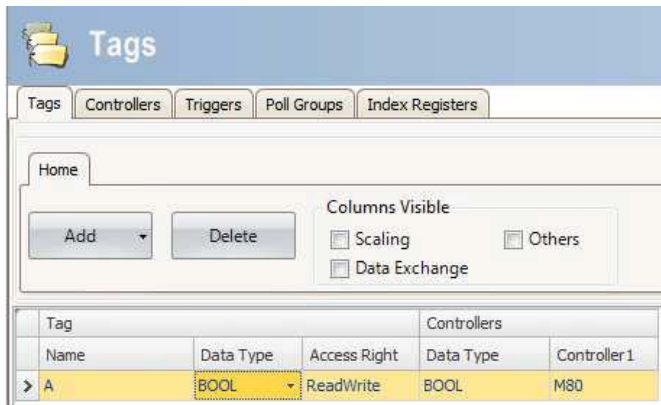


- Gå in i functions, klicka på tags och välj fliken Controllers.

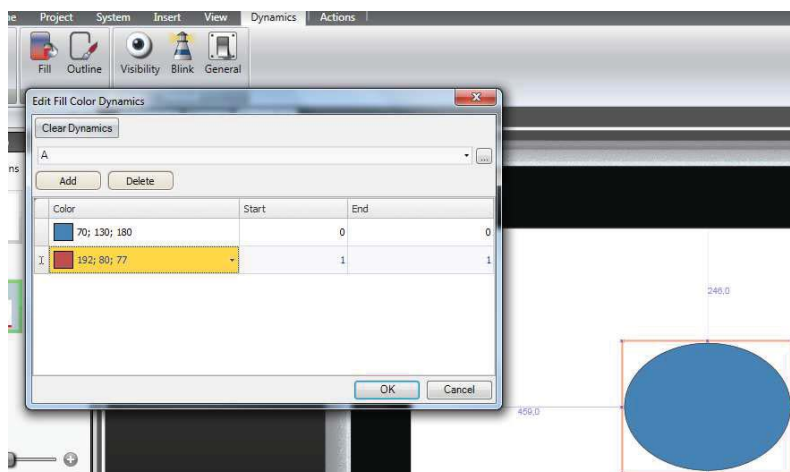


- Klicka på knappen **Settings...** och sedan på fliken Stations. Här ska IP-adressen till PLC:en skrivas in samt ett portnummer. Portnummer från ca 5000 är ganska bra att använda. Skriv in t.ex. 5001, tryck **Apply** och **Ok**.
- Gå in under Tagsfliken igen.

- Namnge en variabel som ska visas på skärmen (namnet behöver inte vara samma som i GX Developer) skriv in adressen som denna ligger på i GX Developers globala variabellista. I detta exempel används M80 för variabeln A.
- Datatypen under Datatype både under Tags och under Controllers måste stämma med datatypen på variabeln i GX Developer. D.v.s är variabeln deklarerad som en BOOL i GX Developer måste den vara en BOOL även i Tags. Under Accessrights ställs in om variabeln på operatörspanelen ska visas enbart (välj då read), både ska visas och kunna påverkas igenom knapptryckning på skärmen (då väljs som i bilden ReadWrite) eller om enbart inläsning till variabeln ska ske (då väljs Write).



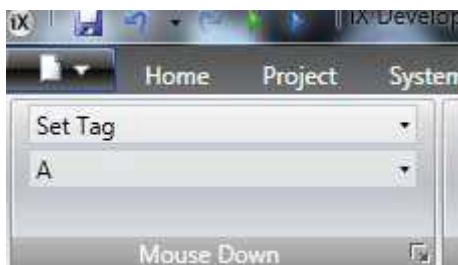
- Nu bör ett test göras för att se om kontakten mellan iX och GX Developer fungerar korrekt. Ett bra test är att bygga ett enkelt program med både inläsning och visning. Här används variabeln A som både kan läsas och skrivas till.
- Skapa en ny skärm (new screen) och lägg till en ellips enligt avsnitt 6.1.
- Tagga denna så att den byter färg mellan två färger, i detta exempel mellan blå och röd.



- Gå in under menyn Action och välj under Mouse Down, Controller klicka på Set tag .



- Under Tags väljs A.



- Gå in under menyn Action och välj under Mouse Up, Controller klicka på Reset tag på motsvarande sätt som för Set Tag.
- Nu kan både inläsning och visning provköras. Bygg både iX och GX Developerprojekten, ladda ner projekten till PLC och panel.
- Sätt GX Developer i Monitoring mode.
- Nu ska ellipsen bli röd då den trycks på via skärmen och bli blå så fort trycket tas bort. Testa alltså att trycka och släppa och se att färgen ändras.

5 Menyer och grundläggande funktioner

En bild av en panelen syns nu på skärmen som till en början är tom med vit bakgrund. Åtta verktygsfält är åtkomliga från menyraden längst upp skärmen. Nedan görs en presentation av några av de viktigaste funktionerna i dessa.

5.1 Home

Verktygsfältet *Home* är indelad i sju grupper av funktioner. De grupper som ej får plats i full storlek på skärmen visas i stället som en knapp med ikon med namn. Klicka på knappen för att se gruppens funktioner.



I *Clipboard*, den första gruppen i verktygsfältet *Home*, finns de vanliga funktionerna klipp, kopiera, klistra in samt *Format Painter* som används för att kopiera formatering (font, textstorlek och färg) från ett objekt till ett annat. Markera t.ex. en blå cirkel (se avsnitt *Objects* längre ner) och tryck på

Format Painter

för muspekaren till t.ex. en röd rektangel (muspekaren visas nu som en pensel) och klicka. Rektangeln blir nu blå.

I gruppen *Screen* kan nya skärmar läggas till genom

Add screen

En tom skärm eller en skärmmall kan läggas till. I rullgardinsmenyn *Background* kan en tidigare skärm väljas som bakgrund. En bakgrund kan väljas då enhetlighet mellan processbilderna önskas.

I *Objects* finns ett stort antal grafiska objekt som kan placeras på skärmen. För att lägga till ett objekt klicka på det och klicka sedan på skärmen. I *Shapes* finns grundläggande geometriska former som kan användas statiskt eller kopplat till en variabel som ändrar tillstånd. *HMI Controls* har fler interaktiva objekt t.ex. knappar, diagram och grafiska visare men även vanlig statisk text. *Media Controls* spelar ljud, video eller visar dokument. Dessutom finns *Special Controls* (fungerar bara på vissa modeller), *Tools For Debug* och *Windows Controls*.

I gruppen *Font* kan typ, storlek och färg ställas in på text som skrivs eller är markerad.

Gruppen *Format* innehåller rullgardinsmenyerna

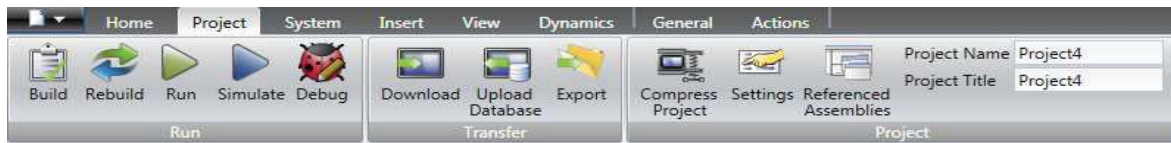
- *Arrange* styr relationen mellan olika objekt t.ex. gruppering och *Bring To Front* (lägger objekt längst fram) samt vänster-och högerjustering av objekt m.fl.

- Quick Styles där bakgrundsfärg och färg på objekt väljs.
- Other Colors där färger för speciella egenskaper hos ett objekt ställs in, ex. varningar för högt och lågt värde.
- Shape Fill används då fyllfärg önskas på objekt.
- Shape Line används då kantlinje önskas runt objekt.
- Shape Effects ger möjlighet att skapa skugga runt objektet eller göra det mer eller mindre transparent.

I gruppen *Tag/Security* ändras säkerhetsinställningar för administratörer respektive operatörer.

I gruppen *Name* kan det förvalda namnet för varje objekt på skärmen ändras.

5.2 Project



Verktögsfältet Project är indelat i följande tre grupper:

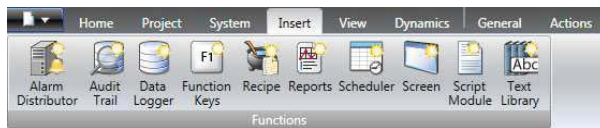
- I gruppen *Run* finns möjlighet att bygga programmet, köra, simulera eller felsöka.
- I gruppen *Transfer* finns funktioner för kompilering och nedladdning av projekt till panel, uppladdning av databas samt export av projekt.
- I gruppen *Project* finns funktioner för komprimering (zippning) av fil, möjlighet att ändra projektets namn (filnamn), titel (som visas vid simuleringen), inställningar och *Referenced Assemblies* där referenser kan göras till Net assemblies (.dll)-filer så att dessa blir tillgängliga vid kodning.

5.3 System



- I gruppen *Time Zone and Region* ställs tid och region (tidformat) in i rullisterna.
- I gruppen *Buzzer* sätts pip ljudet vid tryck på skärmen av/ på.
- I gruppen *Backlight* görs inställningar för bakgrundsbelysningen.
- I gruppen *Serial Ports* ställs anslutningarna in för portarna på baksidan om panelen.
- I gruppen *Servers* görs nätverksinställningar, t.ex. FTP, fjärrstyrning av skärm m.m.
- I grupp *Output devices* görs inställningar för skrivare.
- I gruppen *Service Menu* kan en pinkod matas in som sedan krävs för att komma åt panelens servicemeny som visas då skärmen berörs och inget projekt ligger i panelen.

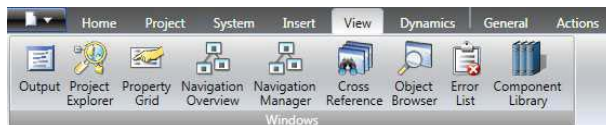
5.4 Insert



I *Insert* ingår bara gruppen *Functions* med följande funktioner:

- *Alarm Distributor* ger möjlighet att skicka alarmnotifikation via skrivare, SMS eller mail.
- *Audit Trail* innebär att kommandon eller variabelvärden loggas. I *Tags* väljs *Others* (i *Column Visible*) vilket ger en kolumn som möjliggör loggning av respektive tag.
- *Med hjälp av Data logger* kan data loggas and sparas i fil.
- *Function Keys* låter användaren koppla ihop en eller flera händelser med funktionstangenterna F1-F24 på tangentbordet.
- *Recipe*
- *Reports*
- *Scheduler* låter användaren schemalägga en eller flera händelser.
- *Screen* lägger till en skärm i project explorer (se kap 4.1).
- *Script Module*
- *Text Library* importerar data från fil.

5.5 View



Under *View* väljs vilka fönster arbetsytan ska bestå av.

- *Output* visar bygginformation och godkännande gällande projektet
- *Project Explorer* visar miniatyrer av skärmarna i projektet. Se kap 4.1
- *Property Grid* visar ett markerat objekts egenskaper och koordinater på skärmen.
- I *Navigation Overview* öppnas en ny flik, *Navigation Manager*, där skärmarna visas liggande bredvid varandra för bättre överblick.
- *Cross Reference* ger en överblick av var en speciell tag används i det aktuella projektet.
- *Object Browser* ger på ett enkelt sätt möjlighet till att arrangera objekt framför eller bakom varandra.
- *Error List* visar fel (errors) och meddelanden om fel som genererats vid byggandet av projektet.

Component Library innehåller ett stort antal återanvändningsbara fördefinierade grafiska objekt för ett projekt, t.ex bild, text och mediafiler. Objekten är indelade i grupper och det går även att spara objekt skapade av användaren i *Component Library*.

5.6 Dynamics



I *Dynamics* finns möjlighet att göra inställningar kopplade till ett objekt med en tag för grafisk visning på skärmen.

- I *Move* anges hur och när (intervall på tagen) ett objekt ska förflytta sig på skärmen.
- I *size* anges hur och när (intervall på tagen) ett objekt ska ändra form.
- I *Fill* hur och när (intervall på tagen) ett objekts färg ska ändras.
- I *Outline* hur och när (intervall på tagen) ett objekts kantlinje ska ändras.
- I *Visibility* anges hur och när (intervall på tagen) ett objekts ska synas på skärmen.
- I *Blink* anges hur, d.v.s intervall och när (intervall på tagen) ett objekt ska blinka.
- *General* ger möjlighet att ändra egenskaperna hos ett objekt i iX runtime (simulering).

5.7 Actions

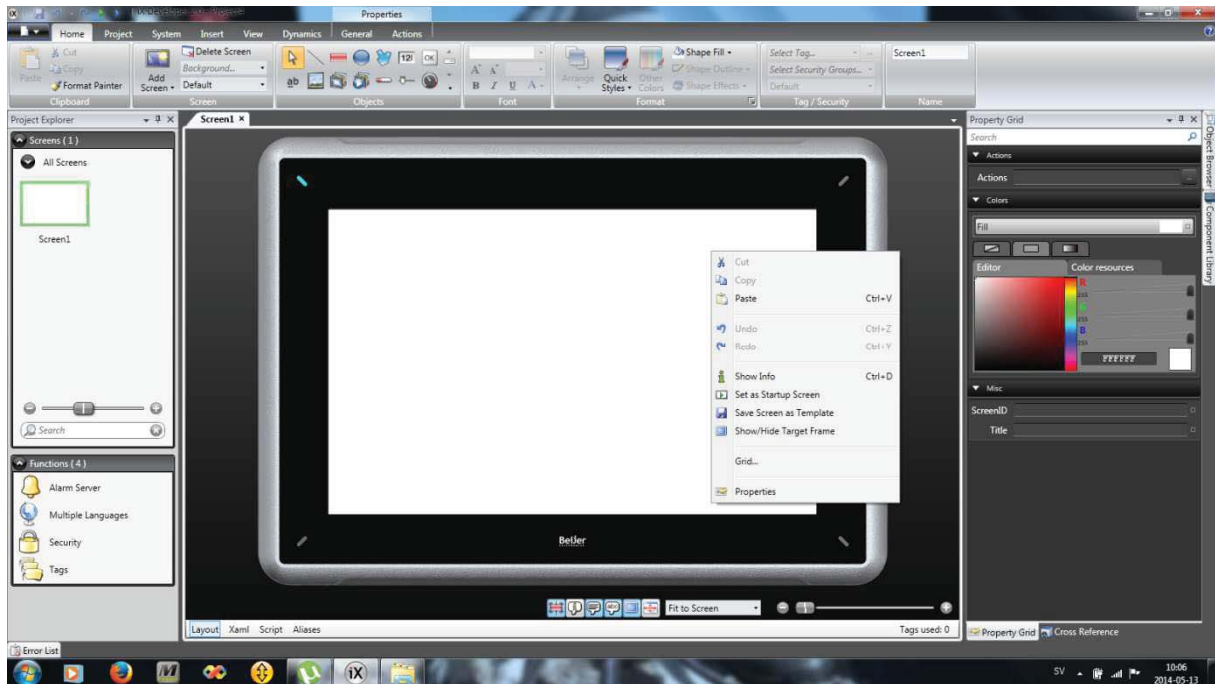


I *Actions* finns möjligheter att använda skärmen för inmatning då önskan finns att via skärmen styra vad som ska visas. Genom att klicka, föra muspekaren över ett objekt, trycka ner musknapp eller släppa musknapp kan händelser kopplas till objekt.

6 Programmeringsexempel

I detta kapitlet ges korta programmeringsexempel på grafiska objekt och larmlista.

6.1 Skapa ett enkelt grafiskt objekt som ändrar färg



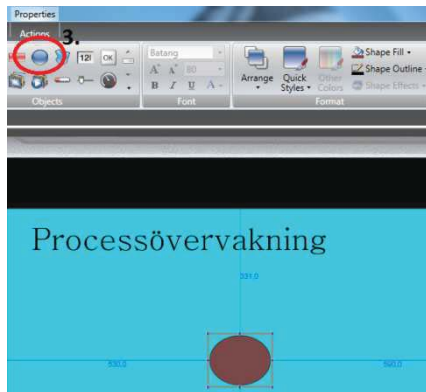
Ett tomt projekt syns nu på med endast en skärm öppen. Bakrunden ska läggas till först, högerklicka och välj properties. Property grid öppnas nu till höger och det ger möjlighet att välja bakgrundsfärg genom att klicka i editor. Den valda bakgrundsfärgen visas på skärmen.

Kanske behövs en överskrift eller rubrik någonstans på skärmen som beskriver vad som visas. Välj flik Home och sedan objects. Där finns en symbol för att lägga till text (1). Texten kan förflyttas m.h.a muspekaren på skärmen och textens koordinaten i förhållande till övre vänstra hörnet visas över, under och till höger och vänster om textrutan. Koordinaterna används för att få objekt helt i mitten av skärmen eller till att inbördes ligga i samma linje. Font (textstil), storlek och färg välj i ritan Font (2).

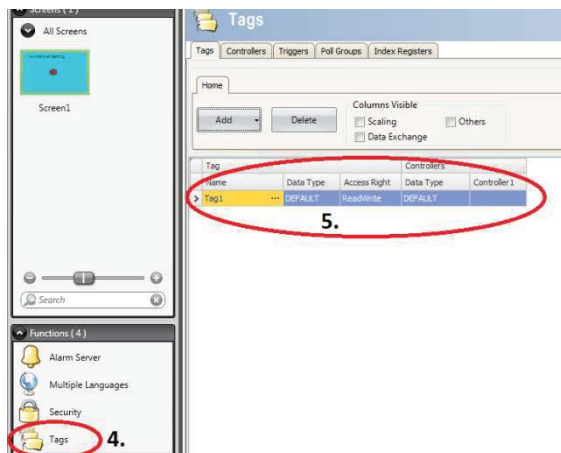


Nu kan objekt läggas till, under fliken Home och objects och klicka på **Ellipse** (3) nu läggs en cirkel till, klicka på cirkelsymbolen och muspekaren visas som ett kryss på skärmen. Vänsterklicka där cirkeln ska placeras.

För att ändra färgen högerklicka på cirkeln och välj properties och då öppnas Property grid på samma sätt som vid val av bakgrundsfärg. Denna färg är nu konstant, önskas att färgen växlar då en viss tag (variabelvärde) ändras värde krävs en taggning.



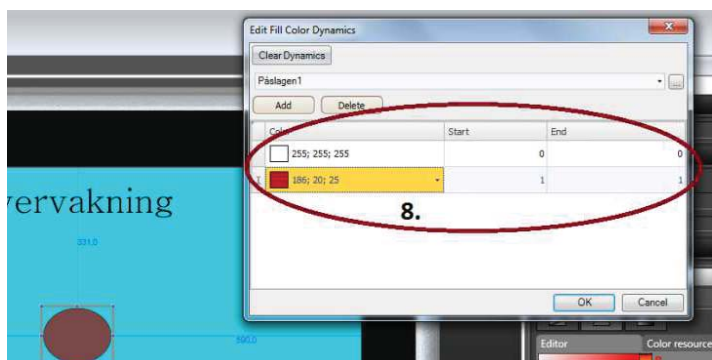
Gå in på Tags (4) under Functions och välj tag, om konfigurationerna gjorts så att den globala variabeln är åtkomlig i tags heter variabelnamnen "Application.varabelnamn". I detta exempel döper vi vår tag till "Påslagen" istället. Ställ in under Datatype (5) vilken typ av data (INT, BOOLSTRING, TIME m.fl. finns att välja mellan) variabeln som ska taggas har samt om läsning och/eller skrivning ska ske. Inläsning är när en variabel ska kunna förändras genom inmatning på skärmen, ska variabeln bara visas väljs read. Fler taggar läggs till igenom knappen Add.



I detta exempel väljs taggen "Påslagen" och sedan ok.



Nu kan de färger väljas som cirkeln ska ha vid olika värden på taggen, dessa väljs i kolumnen colours. I detta exempel ska cirkeln på skärmen växla mellan två färger, den ska vara vit då den boolska taggen har värdet noll (ställs in i kolumnen start) och vara röd då den har värdet ett (ställs in i kolumnen end). Naturligtvis kan flera värden än noll och ett användas om variabeln och taggen är definierade som en sådana. När val av start och slutvärden samt färger är klart, tryck ok.



6.2 Skapa en hastighetsvisare

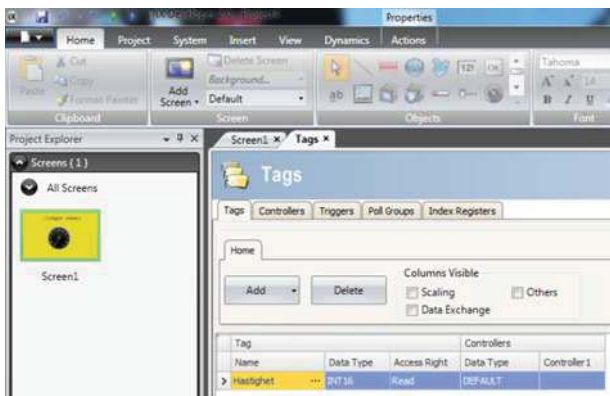
Vid vissa tillämpningar finns behov av att kunna åskådliggöra fler nivåer än noll och ett, som t.ex. vid en hastighet- eller nivåvisare. Ett nytt projekt startades och en ny bakgrundsfärg och överskrift valdes (enligt ex. 1). Tryck på **Circular Meter** i Objects-menyen och lägg till den på skärmen.



Det går att ändra skalan på visaren, färger på de olika fälten kopplade till hastigheterna, bakgrundsfärg m.m under Property Grid.

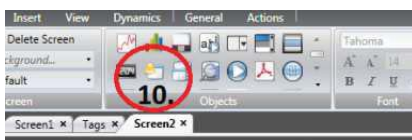


Gå in under tags och välj den tag som ska visas, här heter taggen Hastighet och är en INT16, accessrättigheten ska vara read (läsa) så enbart visning ska ske.

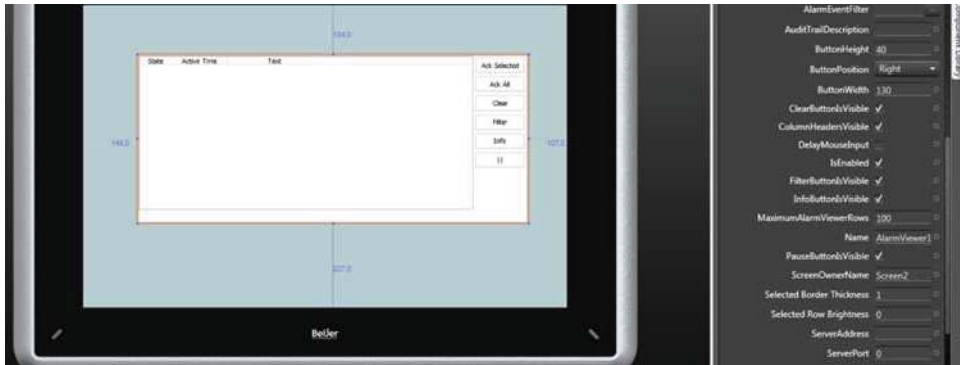


6.3 Larmlista

Tryck på **Alarm Viewer** i objects.



Property Grid syns i vanlig ordning till höger. Här kan knappar tas bort/läggas till, bakgrundsfärg ändras och storlek på knappar m.m.



Knapparna har flyttats till vänster sida, pausknappen har tagits bort, textfonten samt bakgrundsfärgen har ändrats. Vid tryck på **Ack Select**-knappen på skärmen (då projektet laddats ner i panelen) har funktionen att bekräfta ett enskilt larm som visas som en rad och grönmarkerar denna. **Ack-All**-knappen bekräftar samtliga larm i listan och grönmarkerar dem och Clear tar bort larmen från listan. Observera att den tag/de taggar som är knutna till larmlistan inte ändrar värde, utan det är bara visningen av larmen på panelen som styrs av **Ack select**, **Ack All** och **Clear**. Önskas nollställa larmet i styrsystemet måste en ny tag och en knapp knutna till en variabel som nollställer larmet konstrueras.



Gå in i alarmserver, fliken Alarm Items och tryck på Add.

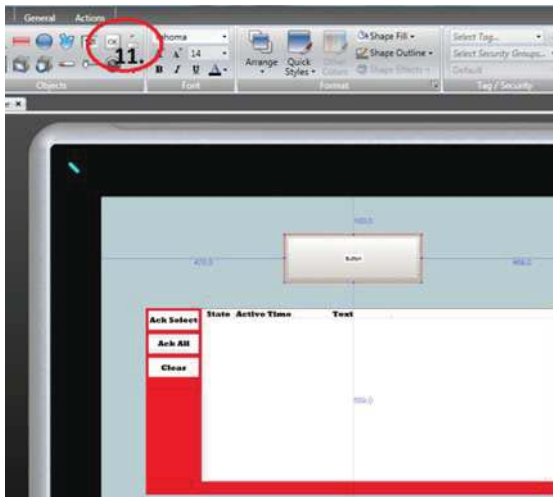


Namnge larmet om det önskas och koppla den tag som ska övervakas i kolumnen tag.



6.4 Nollställning av larm

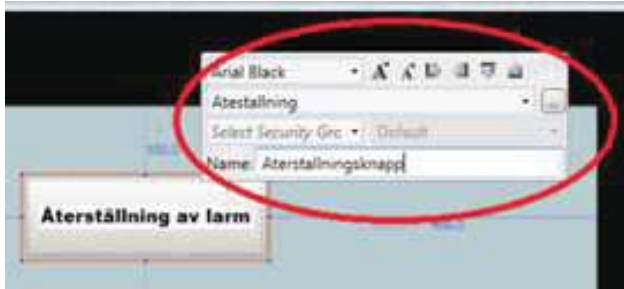
Nu ska en knapp för återställning av larm läggas till ovanför larmlistan. Användaren ska kunna återställa larmet genom att trycka på en knapp på skärmen. Gå in under objects och lägg till en knapp.



Texten ska ändras på knappen, detta genom att högerklicka på texten "Button". Under tags läggs en ny tag, denna ska vara en läsa-tag.



Ett högerklick på knappen och taggning med "Återställning".



I Actions under rullgardinsmenyn Click och vidare under Controller välj **Reset Tag**.



Referenser

[1] iX Developer 2.0 från Beijer Electronics.

